Accelerating Gaussian beam tracing method with dynamic parallelism on graphics processing units

Zhang Sheng[a,d,1], Lishu Duan[a,b,1], Hanbo Jiang[a,b,c,e,*]

[a]*Ningbo Key Laboratory of Advanced Manufacturing Simulation, Eastern Institute of Technology, Ningbo, China*
[b]*School of Ocean and Civil Engineering, Shanghai Jiao Tong University, Shanghai, China*
[c]*Ningbo Institute of Digital Twin, Eastern Institute of Technology, Ningbo, China*
[d]*School of Mathematics, Hefei University of Technology, Hefei, China*
[e]*Zhejiang Key Laboratory of Industrial Intelligence and Digital Twin, Ningbo, China*

Abstract

This study presents an efficient implementation of the Gaussian beam tracing (GBT) method utilizing graphics processing units (GPUs) to overcome the performance limitations of traditional CPU-based acoustic simulations. The algorithm was implemented and optimized on an NVIDIA RTX A6000 GPU, significantly enhancing the Gaussian beam summation (GBS) performance. We addressed the challenge of irregular control flows inherent to GBT by leveraging CUDA's dynamic parallelism to effectively flatten and dispatch nested loops directly on the GPU. Additionally, a profiling-driven optimization workflow using NVIDIA Nsight Compute enabled targeted improvements, raising SM throughput from 22.27% to 33.32%, L1 cache throughput from 13.15% to 22.15%, and L2 cache throughput from 9.16% to 21.26%. Consequently, the GPU-accelerated GBS algorithm achieved up to an 817× speedup compared to the original single-threaded CPU implementation, while the full computational pipeline reached 112× acceleration in a city-environment scenario involving 16,384 rays. Furthermore, this study introduces innovative strategies for overcoming GPU memory limitations, enabling efficient processing of large-scale ray datasets beyond single-kernel constraints. Finally, we establish systematic performance evaluation methodologies critical for analyzing and tuning GPU-accelerated algorithms, laying

*Corresponding author. Email: hjiang@eitech.edu.cn
[1]These authors contributed equally to this work.

a foundation for future enhancements and scalability improvements.

Keywords: Beam tracing, GPU acceleration, Dynamic parallelism

1. Introduction

The study of sound propagation is crucial for a wide range of applications, including the design of concert halls [1, 2], the enhancement of virtual reality experiences [3], improvements in audio prediction for gaming [4, 5], and the assessment of environmental noise from drones and unmanned aerial vehicles [6, 7, 8]. One of the key challenges in acoustic simulation is modeling the interaction of sound waves with complex environments, which involves reflections, diffractions, and scattering.

Over the past few decades, various simulation approaches have been developed to address these challenges, which can be broadly classified into wave-based and geometrical acoustics methods [9, 10, 11]. Wave-based methods, directly solve the wave equation to capture phenomena like interference and diffraction, while the traditional numerical method such as finite element method [9] and finite difference method [12] are adopted. Despite these methods provide good accuracy, their substantial computational demands make them inappropriate for large-scale applications with complex geometries. Conversely, geometrical acoustics simplifies sound as rays or beams and models reflections in a computationally efficient manner, making it particularly suitable for large environments [13]. In this regard, the image source method and ray tracing (RT) method are widely used in various acoustic applications. The image source method [14] models each reflection from a surface as if it originates from an image source symmetrically positioned on the opposite side of that surface. This method is highly effective for simpler environment such as predicting room impulse responses, analyzing reverberation, and optimizing sound design in spaces like auditoriums and studios. However, as the order of reflections increases, particularly in environments with dense occlusions, the number of image sources grows exponentially, leading to a significant increase in computational complexity [13].

On the other hand, the RT method models sound as a collection of rays that emanate from a source [15]. These rays travel through the environment and interact with surfaces encountered. Various versions of the ray tracing algorithm have been implemented ever since the pioneering work of Krokstadt et al. [16]. However, the standard ray tracing method suffers from limita-

tions such as perfect shadows and the inability to accurately model caustics. [17]. To address these limitations, the Gaussian beam tracing (GBT) method was introduced as an extension to traditional ray tracing method [17]. This approach associates each ray with a beam that has a Gaussian intensity distribution then constructs the sound at any given point by summing the contributions of each beam [18]. Unlike traditional rays, Gaussian beams inherently distribute sound energy over a continuous area, filling in gaps where traditional rays might fail. The improvements are twofold: Physically, Gaussian beams allow for smooth transitions in sound intensity across space, preventing infinite energy accumulation in caustic regions and reducing unwanted spikes or dips in sound pressure [17]; Numerically, Gaussian beams enhance sound field coverage while requiring fewer computational resources. This proves particularly advantageous in complex environments with numerous reflections and refractions [6], as it reduces artifacts that would otherwise demand significant post-processing.

Nevertheless, GBT method still faces challenges, particularly in large environments with complex geometries, where the number of beams and interactions increases significantly, leading to higher computational demands. To address these limitations, parallel computation can be employed. It distributes computational tasks across multiple processing units, thereby reducing the time required to trace beams, compute interactions, and reconstruct the sound field. Among parallel architectures, graphics processing units (GPUs) have proven particularly effective due to their ability to handle many tasks simultaneously. GPUs are equipped with thousands of cores that can process numerous parallel tasks concurrently, making them well-suited for data-intensive applications and independent tasks [19]. This architecture provides significant speedups for highly parallel tasks [20], such as those encountered in geometric acoustics. Spjut et al. investigated a multi-threaded beam tracing algorithm on multicore platforms, achieving significant speedups with an increased number of threads [21]. Cowan and Kapralos discussed GPU ray tracing techniques and demonstrated performance improvements for real-time acoustic prediction [22]. Gkanos et al. implemented the image source method on multiple GPUs and suggested that incorporating beam tracing could further enhance efficiency [23]. Tan et al. proposed a GPU-based tree-accelerated beam-tracing method, achieving a speedup of 66 times compared to conventional techniques [24]. Additionally, Greef et al. employed a ray tracing algorithm for radiotherapy dose calculations on a GPU, achieving a 6 times speedup for the evaluated cases [25].

While significant advancements have been made in accelerating RT method, which is only a part of GBT, the reconstruction of the sound pressure field remains underexplored. This crucial process involves searching for all beams contributing to the sound pressure at a specific space point. Existing approaches do not fully leverage parallel computation capabilities, leading to inefficiencies. This study proposes the use of dynamic parallelism on GPUs, which allows for on-the-fly parallel task generation [26]. By enabling kernels to launch other kernels, this method allows threads to manage their parallelism dynamically without the need for explicit synchronization through the host. This approach not only reduces the overhead of kernel launches but also improves overall resource utilization, which is particularly beneficial for collecting contributions from all Gaussian beams where the computational load varies from point to point [27, 28]. All executable files, detailed documentation, and datasets are available in the following GitHub repository: `https://github.com/dcszhang/accelerate_eitray`.

The remainder of this paper is structured as follows. Section 2 reviews the Gaussian beam tracing method. Section 3 details the proposed numerical algorithm and its CUDA-based implementation, including the CUDA architecture, flat and dynamic parallelism strategies, and considerations on computational precision. Section 4 presents the results and discussion, covering verification against analytical solutions, applications to environmental noise scenarios, GPU acceleration experiments with both flat and dynamic parallelism, GPU resource utilization analysis, and profiling-driven optimizations. Finally, Section 5 concludes the paper and outlines directions for future work.

2. Gaussian beam tracing

Figure 1 illustrates the Gaussian beam model, where the ray represents the path of sound propagation and is typically computed using a ray tracing algorithm. The Gaussian-shaped energy distribution around the ray is given by the following expression:

$$p(s, q_1, q_2, t) = \phi \left(\frac{C(s)}{\det[Q(s)]} \right)^{1/2} \times \exp\left[-\mathrm{i}\omega \left(t - \int_s^{s_0} \frac{\mathrm{d}s}{C(s)} \right) + \frac{\mathrm{i}\omega}{2} \left(\mathrm{q}^T P Q^{-1} \mathrm{q} \right) \right], \quad (1)$$

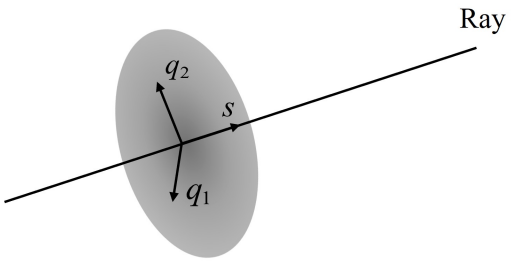


Figure 1: Schematic of a Gaussian beam and the associated coordinates.

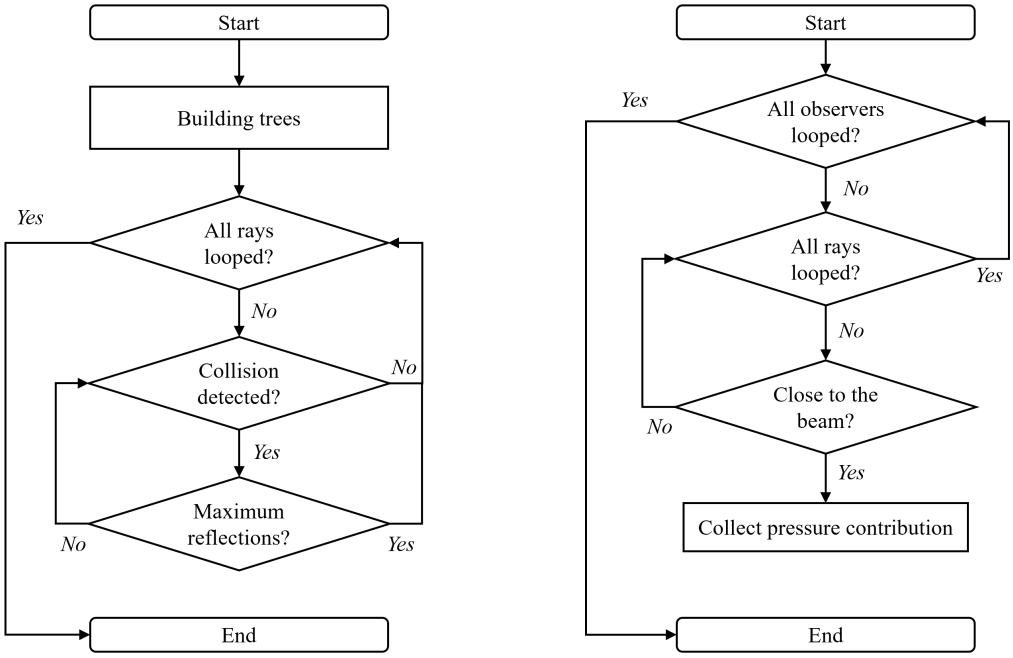
where ϕ is a real constant, and (s, q_1, q_2) represent the ray-centered coordinates. Here, $\det[\cdot]$ represents the determinant operator, and P and Q are matrices that satisfy the following relations:

$$\frac{\partial \mathrm{Q}}{\partial s} = c\mathrm{P}, \quad \frac{\partial \mathrm{P}}{\partial s} = 0, \tag{2}$$

where c is the sound speed. The method for calculating ray paths has been extensively studied and will therefore not be repeated here. Interested readers are encouraged to refer to the relevant literature for more detailed implementations [29]. Subsequently, the contributions of all nearby Gaussian beams along each ray are gathered to estimate the sound pressure p at any observation point R

$$p(R, \omega) = \int \int \Phi(\gamma_1, \gamma_2) P(R_\gamma, \omega) \exp[\mathrm{i}\omega T(R, R_\gamma)] \mathrm{d}\gamma_1 \mathrm{d}\gamma_2, \tag{3}$$

where $\Phi(\gamma_1, \gamma_2)$ is the weighting function, $P(R_\gamma, \omega)$ represents the complex amplitude along the ray R_γ, and $T(R, R_\gamma)$ is the propagation time from the point R_γ to R. In this formula, another important coordinates called ray coordinates (s, γ_1, γ_2) is presented, which are connected to the whole ray field, and γ_1 and γ_2 are the parameters of the ray at the source. For more details on the GBT method, please refer to the literature [6]. It is worth noting that numerically evaluating the above integration is time-consuming, but this process can be significantly accelerated using GPU calculations.



(a) RT process (b) GBS process

Figure 2: The programming flowcharts of the RT and GBS processes. The proposed dynamic parallelism targets to accelerate looping all rays in each process.

3. Algorithm and implementation

3.1. Overview

This section presents the numerical algorithm for Gaussian beam tracing and its parallel implementation. The overall workflow consists of two primary processes: ray tracing (RT) and Gaussian beam summation (GBS), as illustrated in Figure 2. In the RT process, sound propagation paths are determined by tracing rays and modeling their interactions with obstacles. The complexity of this stage is driven by the number of rays and boundary elements, where the latter are typically triangular facets representing reflective surfaces. Subsequently, the GBS process aggregates contributions from all Gaussian beams to calculate sound pressure at observation points. The corresponding computational demand is primarily influenced by the number of rays and observation points.

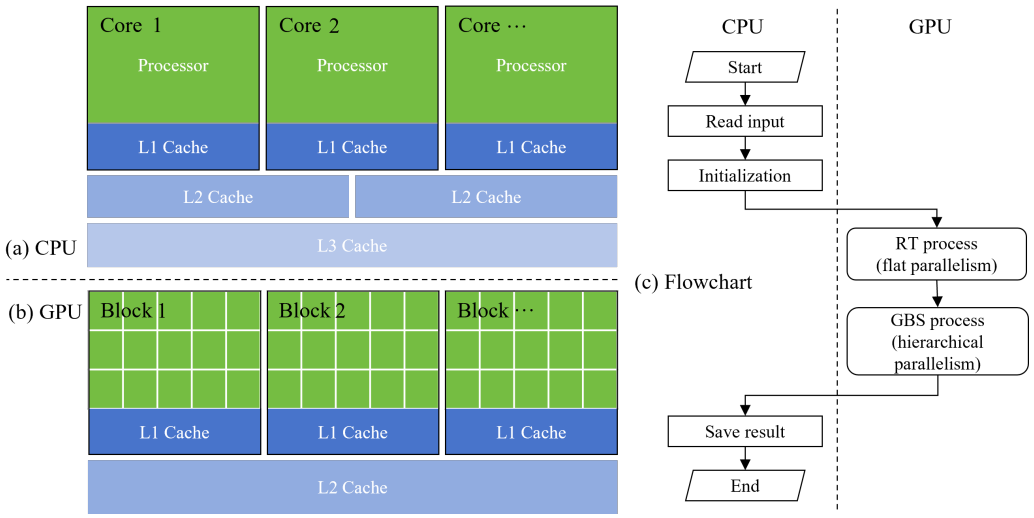


Figure 3: (a) Typical CPU architecture; (b) Typical GPU architecture; Green indicates the processor while blue represents the memory; (c) Flowchart of the GPU acceleration implementation.

3.2. CUDA architecture

We now turn to the numerical implementations utilizing the compute unified device architecture (CUDA), a parallel computing platform and application programming interface developed by NVIDIA®. The CUDA programming model enables us to leverage computational resources from both CPU and GPU platforms. Figure 3(a) illustrates the memory hierarchy in CPUs, consisting of three levels of cache memory: L1, L2, and L3. Here, L1 performs the smallest and fastest, directly integrated into each core; L2 denotes larger and slower, dedicated to each core or shared among a few cores; and L3 represents the largest and slowest, shared across all cores, acting as a buffer between the cores and the main memory (random access memory, RAM). In contrast, the GPU functions as a computation grid, consisting of hundreds of blocks, each containing thousands of processors, as shown in Figure 3(b). Threads within these blocks run concurrently, processing data in parallel using shared L1 cache memory. Each block can execute cooperatively via barrier synchronization. However, blocks typically do not share data directly with one another, except through global memory (L2) or other memory structures.

The proposed GBT implementation consists of both sequential and par-

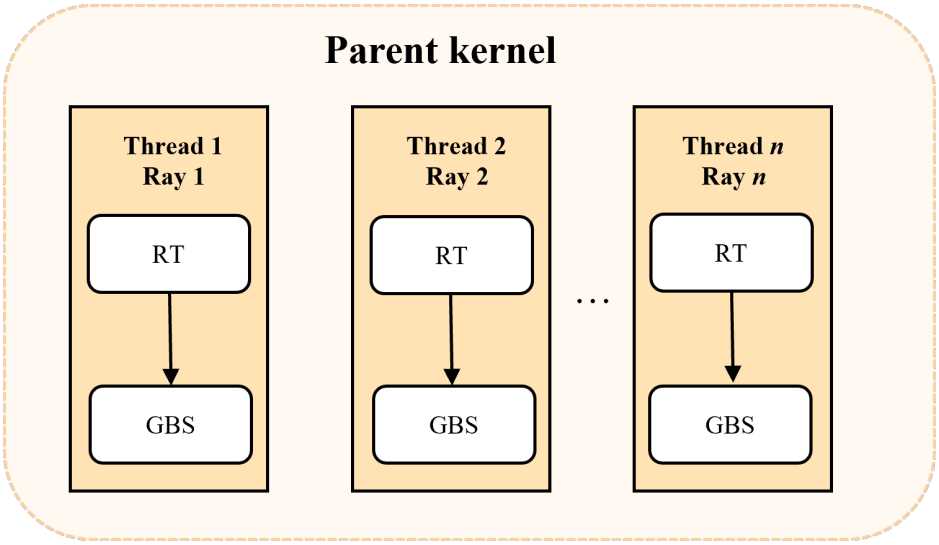


Figure 4: The multi-threaded programming model of flat parallelism on GPU. These threads run simultaneously.

allel components. As shown in Figure 3(c), tasks with low degrees of parallelism, such as file input or output operations, are assigned to the CPU to leverage its strengths and efficiency in sequential processing. Conversely, tasks like the RT and GBT processes, which involve high degrees of parallelism and heavy computational loads, are offloaded to the GPU which excels at handling numerous parallel operations simultaneously. Once these computations are completed, the results are transferred back to the CPU, where the remaining output and prediction processes continue in the same manner as the traditional CPU algorithm. This division of workflow ensures optimal utilization of each processor based on the nature of the tasks.

3.3. Flat parallelism

Figure 4 illustrates the flowchart of flat parallelism, detailing the entire algorithmic process of executing the RT and GBS steps. The algorithm handles an array of rays, with the initial direction of each ray defined by specified combinations of elevation and azimuth angles. Each ray's computation, encompassing both RT and GBS processes, is assigned to a dedicated thread. By evenly distributing these tasks across processors, flat parallelism exploits the computational capabilities of GPUs. After all rays are looped, the GPU computation finished and the whole workflow enters CPU phase.

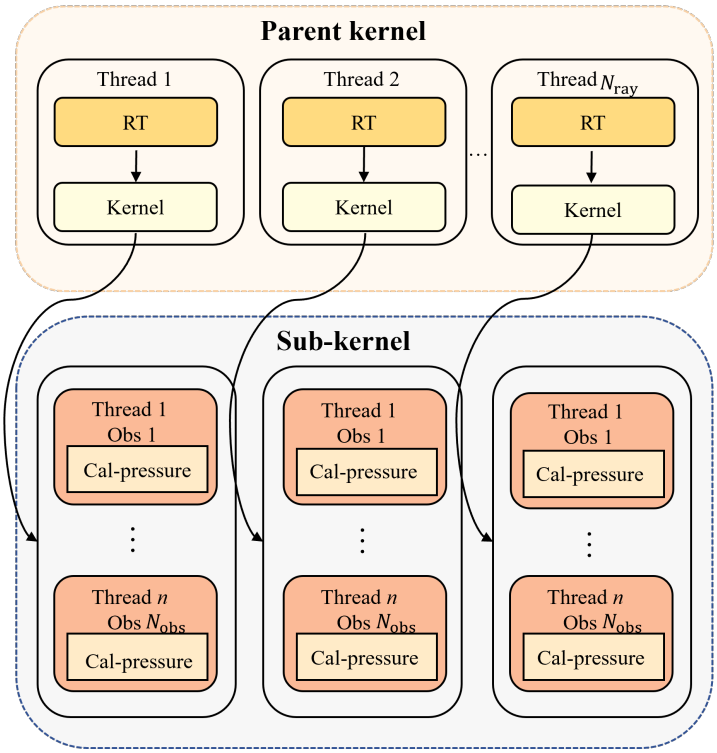


Figure 5: The multi-threaded programming model of dynamic parallelism on GPU.

3.4. Dynamic parallelism

In GBT process, each beam appears to be independent, making it well-suited for flat parallelism. However, the computational load varies across different observers and beams. This variation arises from the need to loop through all beams that contribute to the sound pressure in the GBS process, as described by Equation (3). The length of each beam can vary significantly, especially in complex environments where sound propagation is more intricate. Using flat parallelism resulted in uneven execution times: lightly loaded threads completed quickly, leaving heavy tasks to dominate the overall runtime.

To address these inefficiencies, the present study adopts dynamic parallelism, a programming model introduced by NVIDIA [26], which allows kernels to launch additional kernels during execution, as illustrated in Figure 5. This hierarchical execution model enables fine-grained control over task allocation and workload balancing. Specifically, when a thread encoun-

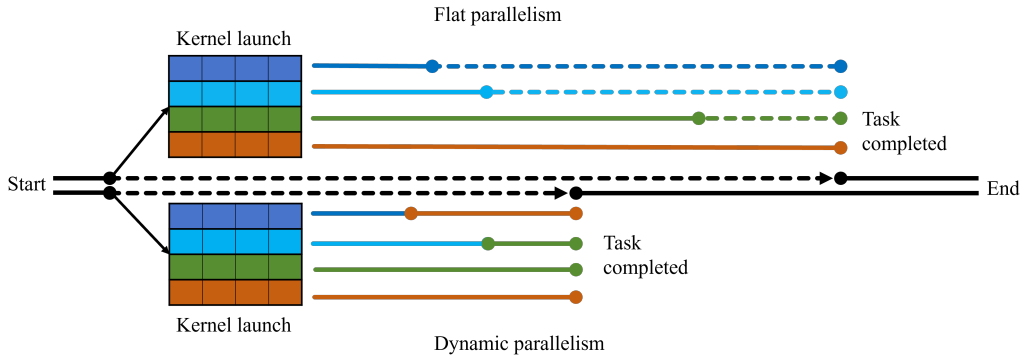


Figure 6: Comparison between flat and dynamic parallelisms.

ters a computationally intensive task, it can dynamically spawn child kernels to divide the workload further. These child kernels are executed independently, allowing idle GPU cores to be reallocated in real time to assist with heavy tasks. This mechanism ensures that computational resources are utilized efficiently, reducing idle time and mitigating bottlenecks.

Figure 6 compares flat and dynamic parallelism. In flat parallelism, tasks are statically distributed across GPU threads during kernel launches. This approach often leads to inefficiencies, as lightly loaded threads finish early and remain idle, while heavily loaded threads create bottlenecks. In contrast, dynamic parallelism redistributes the workload dynamically, enabling idle cores to assist heavily loaded threads. This adaptive rebalancing accelerates task execution significantly reduces overall computation time. The transition from flat to dynamic parallelism is particularly advantageous in environments with highly non-uniform task distributions, ensuring better load balancing and more efficient resource utilization.

4. Results and discussion

In this study, GBT algorithm relies on high computational precision due to the extensive use of complex numbers in the calculations. To ensure the accuracy and numerical stability of the results, we employed double precision (FP64) throughout the algorithm. This choice aligns with the specific needs of large-scale acoustic simulations, where small numerical errors can accumulate and significantly impact the accuracy of the final results.

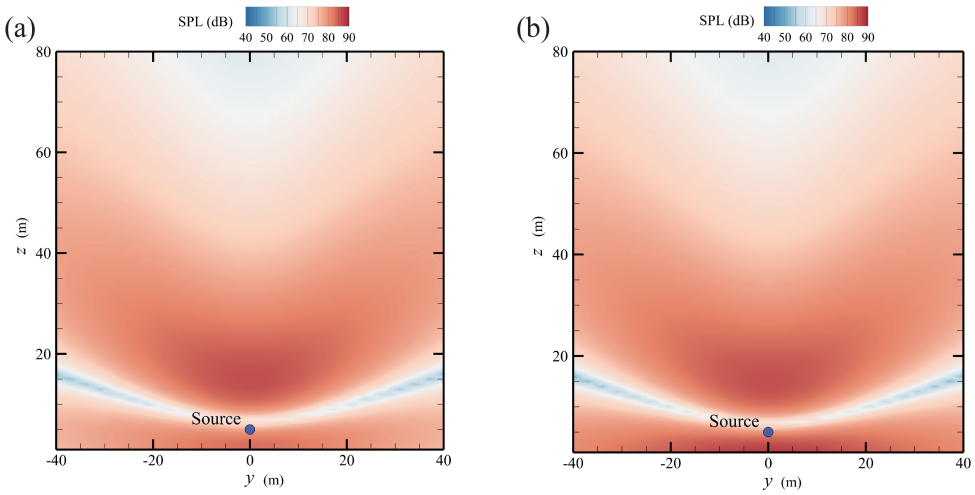


Figure 7: Comparison of SPL calculated by (a) GBT method and (b) analytical solution at $f = 50$Hz.

4.1. Verification

Here, the sound pressure level (SPL) is defined as:

$$\mathrm{SPL} = 20 \log_{10}\left(\frac{p}{p_{\mathrm{ref}}}\right) \ \mathrm{dB}, \tag{4}$$

where p is the magnitude acoustic pressure and $p_{\mathrm{ref}} = 20\ \mu$Pa is the reference sound pressure.

To verify the correctness of the method described in this paper, the reflection of a monopole sound source above an infinitely long plate was computed. The results obtained by the GBT method are compared with the analytical solution. The sound source is positioned at $z = 5$ with a frequency of 500Hz.

Figure 7 compares the sound field of a monopole sound source positioned above a rigid ground at $z = 5$m with a frequency of 50Hz. The results demonstrate that the GBT solver reliably reproduces the sound field structure, as validated against the analytical solution. Figure 8 also illustrates the case for a sound source frequency of 500Hz, where the GBT solver's results continue to closely match the analytical solution. The comparison further confirms the solver's accuracy across different frequencies. Figure 9 compares the sound pressure level distribution at $z = 10$, showing an excellent match between the

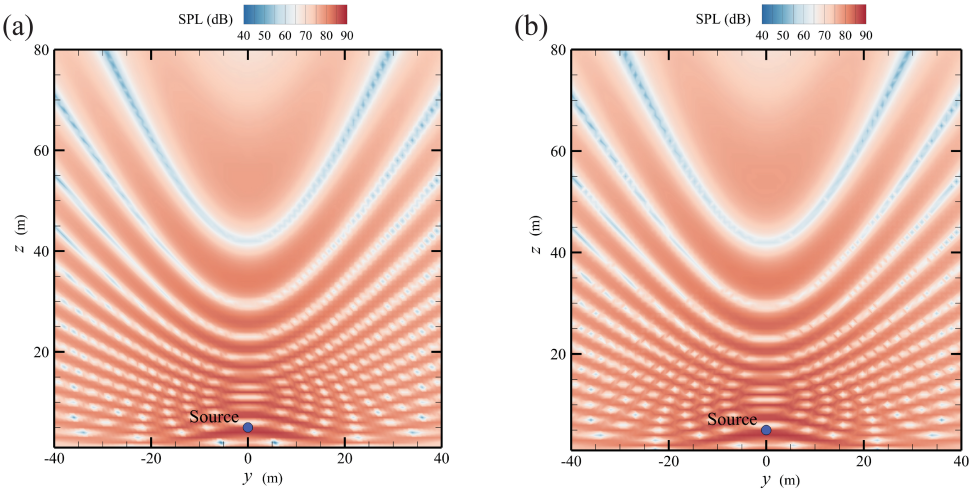


Figure 8: Comparison of SPL calculated by (a) GBT method and (b) analytical solution at $f = 500$Hz.

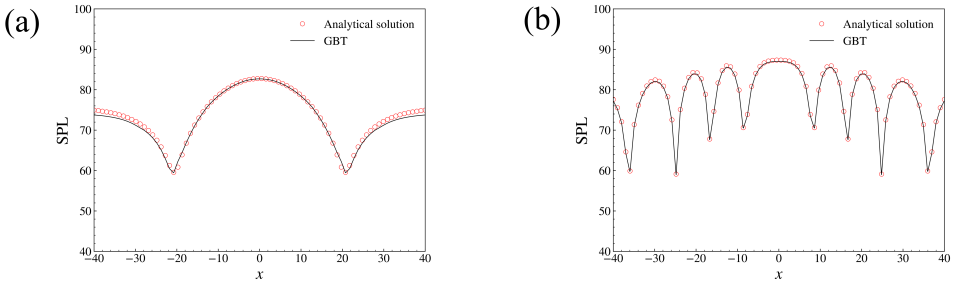


Figure 9: Comparison of SPL at $z = 10$ for various frequencies: (a) 50Hz and (b) 500Hz.

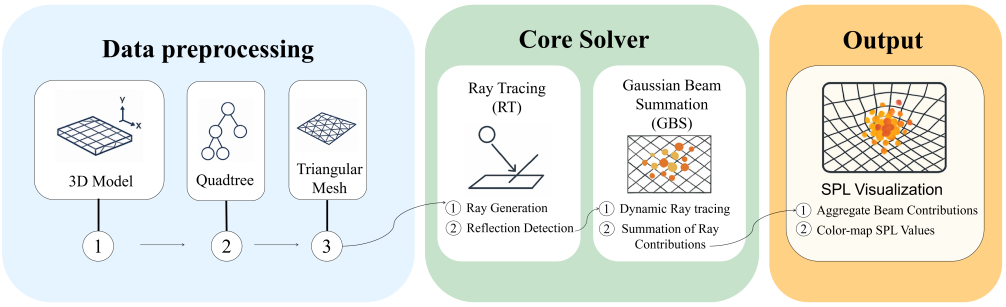


Figure 10: Single-threaded sound field prediction algorithm framework.

two results.At $f = 50$Hz, the GBT results deviate by up to 1.8dB, whereas at $f = 500$Hz the maximum error falls to about 0.5dB. These observations confirm that the GBT method is better suited to higherfrequency problems.

This validation case demonstrates the accuracy of the proposed method and its implementation even for high-frequency sound reflection problems.Based on the high accuracy and stability demonstrated in the foregoing validation, this section extends the method to more realistic environmental noise scenarios in order to evaluate its performance in a single-threaded pipeline and to visualize the resulting sound fields.

4.2. Application to environmental noise

In our single-threaded GBT pipeline (shown in Figure 10), we first convert the noise source parameters, geometric environment model, and observer position into an acceleration-friendly data structure by building a Quadtree [30] over the 3D model and extracting a triangular mesh optimized for fast intersection tests. The RT module then generates Gaussian beams from the source and performs Quadtree-accelerated collision detection against the mesh to compute each beam's precise propagation path and intersection parameters. Finally, the GBS module aggregates the pressure contributions of all beams whose trajectories fall within a prescribed influence radius around each observation point, applies beam weighting functions, sums the individual contributions to obtain the total sound pressure, and color-maps the resulting SPL values to produce the final sound field visualization.

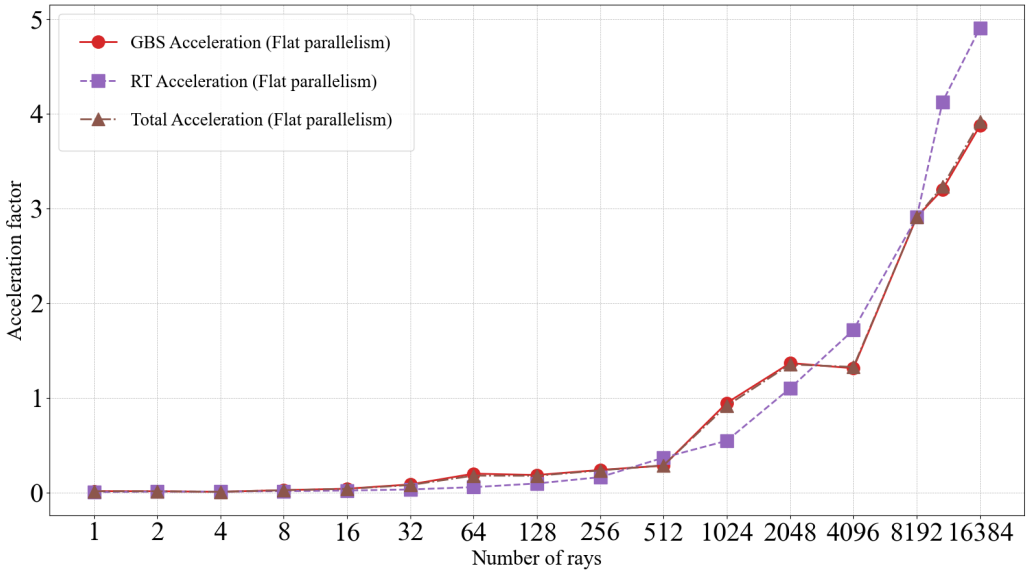


Figure 11: Acceleration factors of the GBS and RT stages under flat parallelism as a function of ray count.

4.3. GPU acceleration with flat parallelism

In this preliminary phase, we port the single-threaded GBT pipeline to the GPU using a flat parallelism approach, where each ray is processed by an individual CUDA thread. Figure 11 plots the speedup of the GBS and RT stages against the number of rays, ranging from 1 to 16384. The GBS speedup increases steadily from 0.013× to 3.47×, while the RT speedup grows from 0.04× to 4.90×. Figure 12 shows that, at 16384 rays, the GBS module still dominates over 95% of the total execution time in the CPU baseline, causing the overall speedup to closely follow the GBS curve and reach a maximum of 3.47×. At low ray counts, the GPU acceleration factor is below 1 due to GPU underutilization and kernel launch overheads. In practice, GPUs become advantageous only when the computational workload exceeds a certain threshold. To further identify performance bottlenecks under flat parallelism, we collected hardware metrics using NVIDIA Nsight Compute. Table 1 summarizes the key throughput metrics measured with 16384 rays.

These metrics reveal that, although the GBS stage achieves up to 3.47× speedup, SM utilization remains very low (3.23%) and global memory bandwidth is barely utilized (1.68%). The high L1/TEX cache throughput (24.64%)

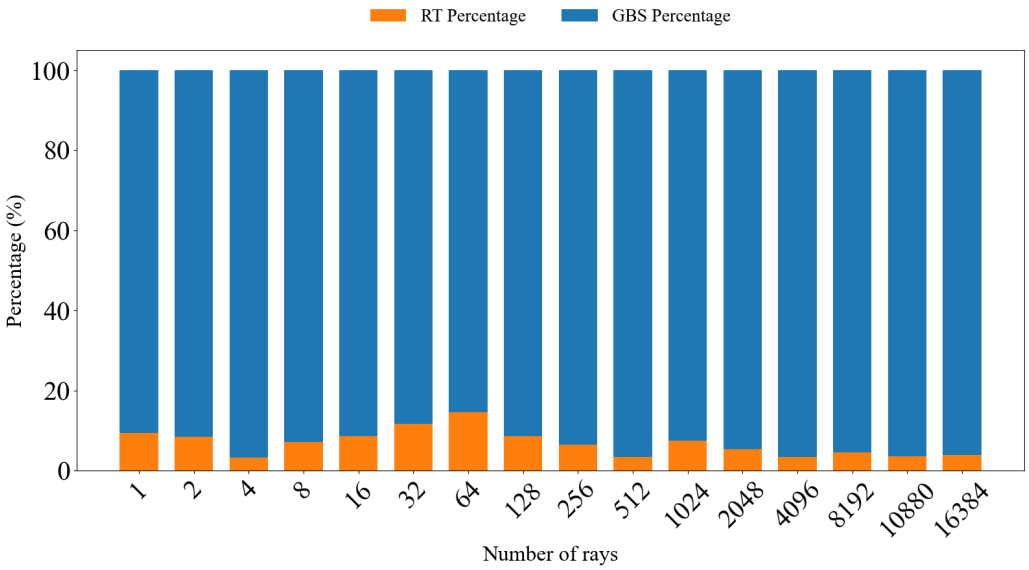


Figure 12: Runtime breakdown between RT and GBS stages under flat parallelism at 16384 rays.

suggests that data locality is sufficient, but the low DRAM throughput indicates a severe underutilization of off-chip memory bandwidth. Together with the dominant GBS workload, these findings point to kernel launch overhead and workload imbalance as the main bottlenecks in the flat parallel approach.

4.4. GPU acceleration with dynamic parallelization

To address the workload imbalance observed under flat parallelism, we implemented CUDA dynamic parallelism, allowing threads in the GBS stage to spawn child kernels for processing variable workloads. It is worth noting that CUDA dynamic parallelism was first introduced in NVIDIA's Kepler architecture (2012) and is generally applicable to all NVIDIA GPU architectures released since then, rather than being specific to the NVIDIA RTX A6000 GPU used in this study.

Hardware and Software Configuration. All experiments were carried out on a workstation equipped with an NVIDIA RTX A6000 GPU and CUDA Toolkit 12.3. The CPU baseline is measured on an AMD 9754 @ 2.25 GHz using single-threaded execution.

Table 1: Nsight Compute metrics under flat parallelism (16384 rays).

Metric	**Throughput [%]**
Compute (SM) Throughput	3.23
Memory Throughput	1.68
L1/TEX Cache Throughput	24.64
L2 Cache Throughput	1.68
DRAM Throughput	0.50

Simulation Scenarios. We consider two representative acoustic environments:

- **Free Field**: an open-space scenario without reflections.
- **City Environment**: a complex urban scenario including buildings, terrain, and roads.

Both scenarios used identical simulation parameters, summarized in Table 2, to ensure a fair comparison.

Table 2: Simulation parameters for Free Field and City Environment scenarios

Parameter	**Free Field**	**City Environment**
Air temperature, T_a (°C)	20	20
Relative humidity, H_r (%)	70	70
Atmospheric pressure, P_a (atm)	1	1
Number of source frequencies, f_s	5	1
Dimensionality, D	3D	3D
Elevation angle range, $[\Theta_{\min}, \Theta_{\max}]$ (°)	[0,180]	[0,180]
Azimuth angle range, $[\Phi_{\min}, \Phi_{\max}]$ (°)	[0,360]	[0,360]
Time steps, N_{steps}	8000	5000
Reflection limit, $R_{\max}$	10	20
Time step size, Δt (s)	1e–4	1e–4
Observation points, N_o	13586	10201

Chunking for Memory Constraints. The RTX A6000 can process at most 11364 rays per kernel launch due to GPU memory limits. For larger ray counts (e.g., 16384), we partition the workload into chunks of 11364 and 5020 rays. This chunking incurs under 2% overhead for kernel relaunch and buffer management, without affecting the overall acceleration trends.

Evaluation Protocol. We compare GPU execution against the single-threaded CPU baseline, measuring:

1. Speedup factors for the GBS and RT stages.
2. Runtime breakdown between GBS and RT.
3. Hardware utilization metrics collected via NVIDIA Nsight Compute.

The following section presents the measured acceleration performance and runtime analyses for both scenarios.

4.4.1. Numerical analysis of City Enviroment case

Our GPU multi-threaded algorithm demonstrated significant acceleration compared to the traditional CPU single-threaded algorithm in the "City Enviroment" experiment. This is particularly evident when the number of rays is increased to 16384, with the acceleration of the GBT reaching an impressive 798 times and the overall acceleration reaching approximately 88 times. These remarkable results are clearly illustrated in Figure 13. The increased thread-parallel computing power provided by GPU dynamic parallelism is the main reason for the significant performance improvement. Furthermore, the timeshare analysis of the GBS and RT components clearly identifies a performance bottleneck in the sound field prediction algorithm. In multi-threaded GPU execution, the time taken by the GBS component is significantly reduced to the extent that RT occupies more than 90% of the time, as shown in Figure 14. This highlights the significant advantage of GPU dynamic parallelism in processing complex acoustic computations. The overall acceleration ratio is notably smaller than the acceleration ratio of the GBS component, due to the lower acceleration efficiency in the RT module.The limitation of parallel acceleration potential arises primarily from the inherent serial dependencies within the RT computations, which result in less effective GPU parallelization compared to the highly parallelizable GBS stage. Although the RT component also executes on the GPU, certain computational steps within RT remain inherently sequential, limiting the achievable acceleration. Nevertheless, the GPU multi-threaded sound field prediction algorithm demonstrates significant acceleration, up to a hundred times in complex enviroments.

The RTX A6000 can process at most 11364 rays per kernel launch due to GPU memory limits. For experiments with 16384 rays, we partition the

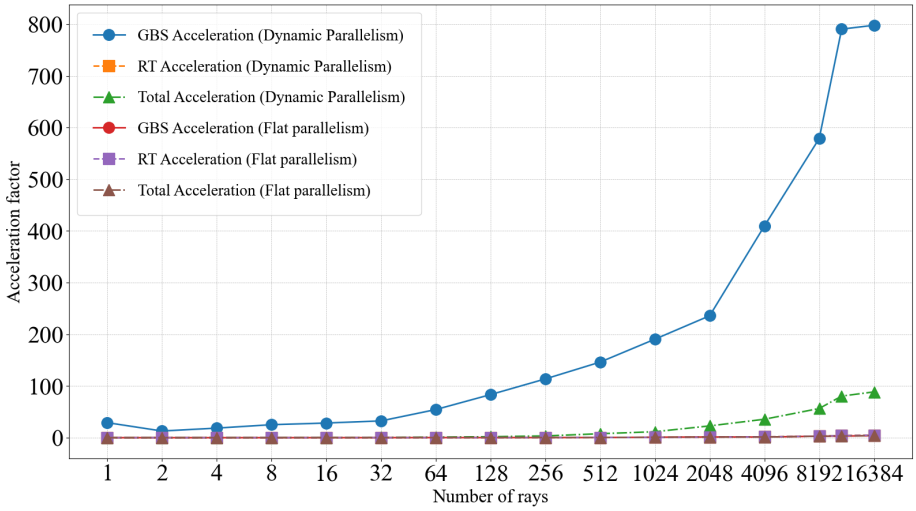


Figure 13: Performance of multi-threaded GPU sound field prediction in City Environment using dynamic parallelism: Acceleration factors for different ray counts.

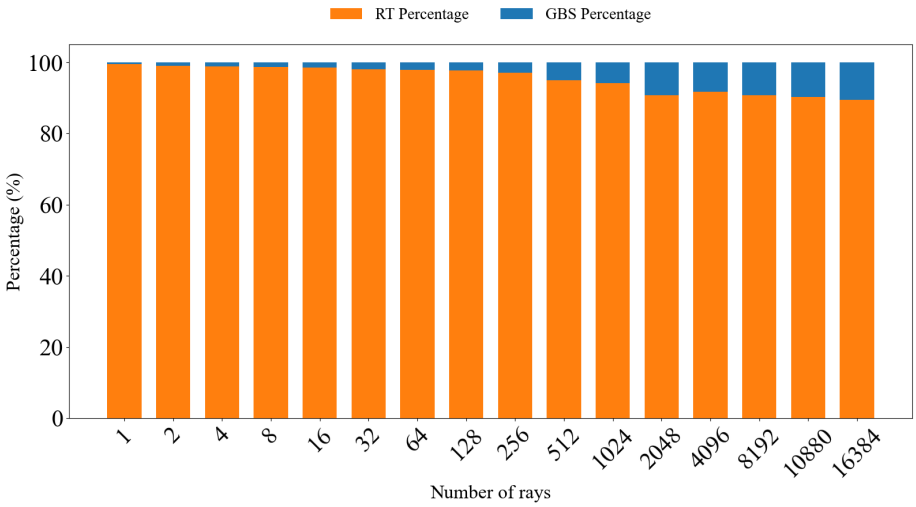


Figure 14: Percentage of two main processes in a multi-threaded sound field prediction algorithm in City Environment using dynamic parallelism GPUs.

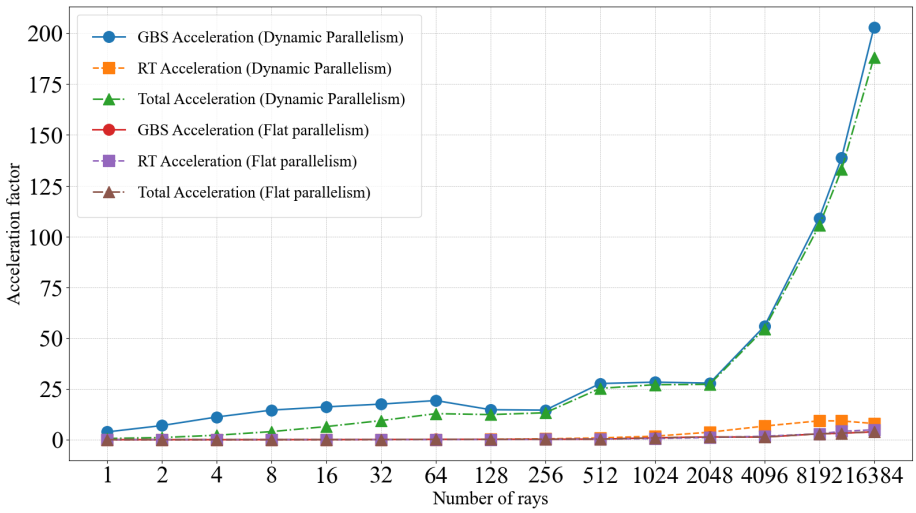


Figure 15: Performance of multi-threaded GPU sound field prediction in Free Field using dynamic parallelism: Acceleration factors for different ray counts.

workload into two chunks (11364 and 5020 rays). This chunking incurs under 2% overhead for kernel relaunch and buffer management, yet the GBS speedup of 798× closely matches the 790× speedup observed with 10800 rays. This demonstrates that chunking does not degrade performance and highlights the algorithm's stability and scalability.

Overall, these results confirm that dynamic parallelism effectively balances workloads across GPU cores, shifting the remaining bottleneck to the RT stage and paving the way for further optimizations in ray tracing.

4.4.2. Numerical analysis of Free Field Enviroment

The "Free Field" scenario exhibit particularly noteworthy performance when the number of rays increased to 16,384. Although the acceleration factor for GBS only reached 200 times, the overall acceleration factor significantly rose to 188 times, as illustrated in Figure 15. A shift in the time proportion between the RT module and the GBS module occurred, as shown in Figure 16, where the roles reversed in terms of their contribution to the total computation time.

The unusual bar chart highlights the performance dynamics of the sound field prediction algorithm across different enviroment. Initially, the RT phase occupies a higher proportion of the computational load, while the GBS phase

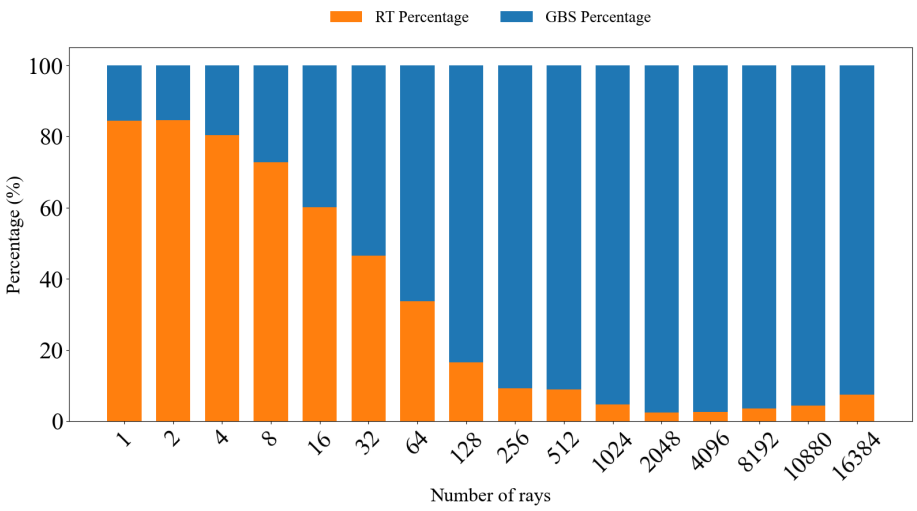


Figure 16: Percentage of two main processes in a multi-threaded sound field prediction algorithm in Free Field using dynamic parallelism GPUs.

remains relatively low. This shift occurs because, in an obstacle-free environment, the RT computation remains relatively lightweight regardless of ray count, whereas the number of Gaussian beams to sum in the GBS stage grows linearly with the ray count. Consequently, GBS becomes the primary workload at high ray counts. These results confirm that dynamic parallelism effectively adapts to changing workloads.

4.4.3. GPU resource usage

The analysis of GPU resource utilization is conducted using Nsight compute during the city environment experiment. The results are presented in Tables 3, 4, and 5.

Table 3: Optimized Nsight compute software performance analysis table

Metric	**Throughput** [%]
Compute (SM) Throughput	22.27
Memory Throughput	9.16
L1/TEX Cache Throughput	13.15
L2 Cache Throughput	9.16
DRAM Throughput	2.04

A significant change observed in Table 3 is the substantial increase in the

utilization of Streaming Multiprocessors (SM) resources from a mere 3.23% to 22.27% by implementing dynamic parallel processing for the GBS section. This significant improvement indicates that the dynamic parallelism strategy plays a key role in enhancing the efficiency of GPU computational resource utilization.

Table 4: Performance analysis of the Nsight compute software for RT process graph

Metric	**Throughput** [%]
Compute (SM) Throughput	1.67
Memory Throughput	3.05
L1/TEX Cache Throughput	4.38
L2 Cache Throughput	3.05
DRAM Throughput	0.32

As seen in Table 4, the complexity of the computation leads to a higher demand for registers per GPU thread, with an average of 199 registers needed. This high register demand poses a limitation on the granularity of dynamic parallelism. Registers are a highly valuable resource in GPUs, and the excessive consumption of registers by each thread means that fewer threads can be executed simultaneously. Therefore, our dynamic parallelism optimization strategy is approaching its performance limit under the current conditions.

Table 5 shows that the GPU utilization rate for executing the RT module alone is only 1.67%, a low utilization rate that significantly reduces the overall GPU performance. This indicates significant room for performance improvement during the RT process. Thus, a focus of future work could be to further optimize the RT algorithm to improve its resource utilization efficiency on the GPU, thereby achieving a higher acceleration ratio and better performance for the overall sound field prediction algorithm.

Table 5: Analysis of the resources used by the optimised

Launch Statistics	**Value**
Grid Size	6
Registers Per Thread	199
Block Size	256
Thread	1536
Waves Per SM	0.13

4.5. Profiling-Driven Optimization

To close the remaining performance gap under dynamic parallelism, we conducted a two-stage profiling campaign using NVIDIA Nsight Compute:

1. **Baseline measurement:** Collect hardware counters (SM throughput, register usage, cache and memory throughput) for the unoptimized kernels.
2. **Bottleneck analysis:** Identify hotspots and inefficiencies in compute, memory, and register utilization.

Targeted Optimizations. Based on the profiling insights, we apply the following refinements:

- **Reduce register pressure:**
  - Applied `__launch_bounds__(256,2)` and split long kernels to cap register usage per thread.
  - Refactored three large data structures into on-demand local variables, lowering registers per thread from 199 to 128.
- **Improve instruction throughput:**
  - Replaced divergent warp reductions with `__shfl_sync` to eliminate branch divergence.
  - Switched global loads to `__ldg` for read-only data, boosting memory read efficiency.
- **Optimize texture cache:**
  - Bound static terrain meshes to `cudaTextureObject_t`, enhancing data locality and reducing memory access latency in the RT stage.

These changes raised block occupancy from 22.27% to 34.18% and significantly improved SM and cache utilization.

Quantitative Outcomes. Table 6 compares key microarchitectural metrics before and after optimization, and Table 7 reports application-level speed-ups in the environmental noise case.

These results confirm that a profiling-driven approach—targeting register pressure, instruction throughput, and cache utilization—is essential for fully exploiting modern GPUs in large-scale acoustic simulations.

Table 6: Nsight Compute metrics before and after profiling-guided refinement

Metric	**Baseline**	**Optimized**	Δ
Registers per thread	199	128	–36%
Compute (SM) Throughput	22.27%	33.32%	+11.05%
L1/TEX Cache Throughput	13.15%	22.15%	+9.00%
L2 Cache Throughput	9.16%	21.26%	+12.10%

Table 7: Application-level speed-ups (City Environment, 16384 rays)

Stage	**Baseline**	**Optimized**
GBS only	7.98×10^2	$\mathbf{8.17 \times 10^2}$
Full pipeline	88×	**112**×

5. Conclusion

In this work, we have presented a GPU-accelerated implementation of the Gaussian Beam Tracing (GBT) method using CUDA, combining both flat and dynamic parallelism to tackle the computational demands of large-scale acoustic simulations. Our key findings and contributions are:

- **High-performance GPU implementation:** We port the GBT algorithm to CUDA, achieving up to **817**× speedup in the Gaussian Beam Summation (GBS) stage and **112**× overall pipeline acceleration in complex city environments.
- **Dynamic parallelism for workload balancing:** By enabling kernels to spawn child kernels at runtime, we effectively addressed the irregular and highly non-uniform workloads of GBS, reducing idle GPU resources and overcoming the limitations of flat parallelism.
- **Memory-aware chunking mechanism:** To accommodate GPU memory constraints, we designed a lightweight chunking strategy that partitions large ray sets into manageable batches, incurring less than 2% overhead while preserving high speedups.
- **Comprehensive validation and evaluation:** We verify algorithmic accuracy against analytical solutions and conducted extensive experiments in both free-field and urban scenarios, demonstrating robustness and scalability across diverse acoustic environments.

These results underscore the feasibility of real-time or near–real-time sound field prediction for applications in architectural acoustics, environmental noise assessment, and virtual reality. In future work, we will focus on further optimizing the ray tracing stage—reducing register pressure, enhancing memory bandwidth utilization, and exploring hybrid parallelism strategies—to push performance even higher and broaden the applicability of GPU-based acoustic simulation techniques.

Additionally, given that the primary synchronization requirement is summing contributions from different beams, extending our GPU-accelerated implementation to multiple GPUs represents a promising avenue for further scalability and performance improvements. Investigating the performance and scalability of multi-GPU deployments will thus form an integral part of our future research.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No. 12302346) and Zhejiang Provincial Natural Science Foundation of China (No. LQ24A040014).

References

[1] John C Allred and Albert Newhouse. Applications of the Monte Carlo method to architectural acoustics. *The Journal of the Acoustical Society of America*, 30(1):1–3, 1958.

[2] Michael Barron. *Auditorium Acoustics and Architectural Design*. Spon Press, 2009.

[3] Michael Vorländer. Virtual acoustics. *Archives of Acoustics*, 39(3):307–318, 2014.

[4] Nikunj Raghuvanshi, Rahul Narain, and Ming C Lin. Efficient and accurate sound propagation using adaptive rectangular decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):789–801, 2009.

[5] Qichen Tan, Haoyu Bian, Jingwen Guo, Peng Zhou, Hong Kam Lo, Siyang Zhong, and Xin Zhang. Virtual flight simulation of delivery drone

noise in the urban residential community. *Transportation Research Part D: Transport and Environment*, 118:103686, 2023.

[6] Haoyu Bian, Qichen Tan, Siyang Zhong, and Xin Zhang. Efficient computation of broadband noise propagation using Gaussian beam tracing method. *The Journal of the Acoustical Society of America*, 151(5):3387–3397, 2022.

[7] Qichen Tan, Yuhong Li, Han Wu, Peng Zhou, Hong Kam Lo, Siyang Zhong, and Xin Zhang. Enhancing sustainable urban air transportation: Low-noise UAS flight planning using noise assessment simulator. *Aerospace Science and Technology*, 147:109071, 2024.

[8] Qichen Tan, Siyang Zhong, Renhao Qu, Yuhong Li, Peng Zhou, Hong Kam Lo, and Xin Zhang. Low-noise flight path planning of drones based on a virtual flight noise simulator: A vehicle routing problem. *IEEE Intelligent Transportation Systems Magazine*, 2024.

[9] JA Eaton and BA Regan. Application of the finite element method to acoustic scattering problems. *AIAA Journal*, 34(1):29–34, 1996.

[10] Carl R Hart and Siu-Kit Lau. Prediction of urban sound propagation via adaptive beam tracing. *The Journal of the Acoustical Society of America*, 129(4):2481–2481, 2011.

[11] Eduard Deines, Martin Bertram, Jan Mohring, Jevgenij Jegorovs, Frank Michel, Hans Hagen, and Gregory M Nielson. Comparative visualization for wave-based and geometric acoustics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1173–1180, 2006.

[12] Hanbo Jiang, Xin Zhang, and Xun Huang. Reduced-basis boundary element method for efficient broadband acoustic simulation. *Journal of Sound and Vibration*, 456:374–385, 2019.

[13] Samuli Laine, Samuel Siltanen, Tapio Lokki, and Lauri Savioja. Accelerated beam tracing algorithm. *Applied Acoustics*, 70(1):172–181, 2009.

[14] Jont B Allen and David A Berkley. Image method for efficiently simulating small-room acoustics. *The Journal of the Acoustical Society of America*, 65(4):943–950, 1979.

[15] Hilmar Lehnert. Systematic errors of the ray-tracing algorithm. *Applied Acoustics*, 38(2-4):207–221, 1993.

[16] Asbjørn Krokstad, Staffan Strom, and Svein Sørsdal. Calculating the acoustical room response by the use of a ray tracing technique. *Journal of Sound and Vibration*, 8(1):118–125, 1968.

[17] Michael B Porter and Homer P Bucker. Gaussian beam tracing for computing ocean acoustic fields. *The Journal of the Acoustical Society of America*, 82(4):1349–1359, 1987.

[18] Michael B Porter and Yong-Chun Liu. Finite-element ray tracing. *Theoretical and computational acoustics*, 2:947–956, 1994.

[19] Bingsheng He, Mian Lu, Ke Yang, Rui Fang, Naga K Govindaraju, Qiong Luo, and Pedro V Sander. Relational query coprocessing on graphics processors. *ACM Transactions on Database Systems (TODS)*, 34(4):1–39, 2009.

[20] Cristobal A Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. *Communications in Computational Physics*, 15(2):285–329, 2014.

[21] Josef Spjut, Andrew Kensler, Daniel Kopta, and Erik Brunvand. TRaX: A multicore hardware architecture for real-time ray tracing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(12):1802–1815, 2009.

[22] Brent Cowan and Bill Kapralos. GPU-based real-time acoustical occlusion modeling. *Virtual Reality*, 14:183–196, 2010.

[23] Konstantinos Gkanos, Finnur Pind, Hans Henrik Brandenborg Sørensen, and Cheol-Ho Jeong. Comparison of parallel implementation strategies for the image source method for real-time virtual acoustics. *Applied Acoustics*, 178:108000, 2021.

[24] Jundong Tan, Zhuo Su, and Yunliang Long. A full 3-D GPU-based beam-tracing method for complex indoor environments propagation modeling. *IEEE Transactions on Antennas and Propagation*, 63(6):2705–2718, 2015.

[25] M De Greef, J Crezee, JC Van Eijk, R Pool, and Arjan Bel. Accelerated ray tracing for radiotherapy dose calculations on a GPU. *Medical Physics*, 36(9Part1):4095–4102, 2009.

[26] Design Guide. CUDA C programming guide. *NVIDIA, July*, 29:31, 2013.

[27] Ben Cope, Peter YK Cheung, Wayne Luk, and Lee Howes. Performance comparison of graphics processors to reconfigurable logic: A case study. *IEEE Transactions on Computers*, 59(4):433–448, 2010.

[28] Pawel Czarnul. Programming, tuning and automatic parallelization of irregular divide-and-conquer applications in DAMPVM/DAC. *The International Journal of High Performance Computing Applications*, 17(1):77–93, 2003.

[29] L. Speer and Xiaohang Yue. An updated cross-indexed guide to the ray-tracing literature. *Computers & Graphics*, 1992.

[30] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.