Numerical Enhancements and Parallel GPU Implementation of a 3D Gaussian Beam Model[⋆]

Rogério M. Calazan$^{1[0000-0001-6025-2777]}$, Orlando C. Rodríguez$^{2[0000-0002-0375-1892]}$, and Sérgio M. Jesus$^{2[0000-0002-6021-1761]}$

[1] Institute of Sea Studies Admiral Paulo Moreira,
Arraial do Cabo, Brazil
moraes.calazan@marinha.mil.br
https://www.marinha.mil.br/ieapm/

[2] Laboratory of Robotics and Systems in Engineering and Science (LARSyS),
Campus de Gambelas, Universidade do Algarve, Faro, Portugal
http://www.siplab.fct.ualg.pt

Abstract. Despite the increasing performance of modern processors it is well known that the majority of models that account for 3D underwater acoustic predictions still require a high computational cost. In this context, this work presents strategies to enhance the computational performance of a ray-based 3D model. First, it is presented an optimized method for acoustic field calculations, that accounts for a large number of sensors. Second, the inherent parallelism of ray tracing and the high workload of 3D propagation are carefully considered, leading to the development of parallel algorithms for field predictions using a GPU architecture. The strategies were validated through performance analyses and comparisons with experimental data from a tank scale experiment, and the results show that model predictions are computationally efficient and accurate. The combination of numerical enhancements and parallel computing allowed to speedup model calculations for a large number of receivers.

Keywords: Underwater acoustics · numerical modeling · 3D propagation · GPU parallel computing.

1 Introduction

Since its early development, underwater predictions of acoustic 3D propagation are well known to be highly time consuming; initial research in this area relied in fact on dedicated computer architectures to carry on model execution [1, 2]. Even today, despite the increasing performance of modern processors, models that take into account 3D propagation still have a high computational cost [3]. The corresponding (high) runtime of a single prediction can easily explain why 3D models are generally put aside when dealing with problems of acoustic inversion

[⋆] Supported by National Council for Scientific and Technological Development (CNPq)/Science without Borders Program (Contract No. 401407/2014-4).

[5]. Additionally, predictions of sonar performance can take advantage of full 3D modeling to improve accuracy, with a ray model playing a central role in such task due to its capability to handle high frequencies, say, above 500 Hz [6]. Furthermore, monitoring of shipping noise represents also an important field of research since shipping noise propagates at long distances, with 3D effects becoming more relevant as distances increase. On the other side, ship density is high in the vicinity of harbors, making bottom interactions and 3D effects important in shallow waters and littoral environments [8, 9].

Computational model performance can be significantly improved through the combination of numerical enhancements with parallel computing. In this sense, the generation of optimized algorithms requires task-specific analysis and development of new methods, that parallel computing cannot be able to overcome alone. Additionally, the performance of *graphic processing units* (hereafter GPU) motivated several implementations of scientific applications, including underwater acoustic models. For instance, a split-step Fourier parabolic equation model implemented in a GPU is discussed in [11] and the discussion presented in [12] describes a GPU-based version of a Beam-Displacement Ray-Mode code; both works considered only highly idealized 2D waveguides and the results showed a significant improvement regarding the computational performance. A parallel version of the C-based version of TRACEO (called cTRACEO), based on a GPU architecture, was discussed in detail in [13]; the discussion showed that parallelization drastically reduces the computational burden when a large number of rays needs to be traced; performance results for such 2D model outlined the computational advantages of considering the GPU architecture for the 3D case. Preliminary research into parallelization in a coarse-grained fashion was also explored using OpenMPI [14, 15]. Performance analysis showed that the parallel implementation followed a linear speedup when each process was addressed to a single physical CPU core. However, such performance was achieved at the cost of using high-end CPUs, designed for computer servers without network communication (which probably would decrease the overall performance). Thus, the *best* parallel implementation was 12 times faster than the sequential one, meaning that the execution took place in a CPU with 12 physical cores.

The work presented here describes the development of numerical enhancements and optimization of the sequential version of the *TRACEO3D* Gaussian beam model [16–18], leading to improvements of its performance; the description is followed by further analysis of the GPU hardware multithread, and the coding elements of the time consuming model structure, into a parallel algorithm that takes advantage of the GPU architecture. This development looks carefully to the inherent parallelism of ray tracing and to the high workload of computations for 3D predictions. Furthermore, validation and performance assessments are presented considering experimental data from a tank scale experiment. The results show that a remarkable performance was achieved without compromising accuracy. This paper is organized as follows: the TRACEO3D model is described in 2; numerical enhancement are presented in Section 3; the detailed structure of parallel GPU implementation is presented in Section 4; Section 5 presents the validation results, in which experimental data were considered. Conclusions and future work are presented in Section 6.

2 The TRACEO3D Gaussian beam model

The *TRACEO3D* Gaussian beam model [17] corresponds to a three-dimensional extension of the *TRACEO* 2D model [19]. TRACEO3D produces a prediction of the acoustic field in two steps: first, the set of Eikonal equations is solved in order to provide ray trajectories; second, ray trajectories are considered as the central axes of Gaussian beams, and the acoustic field at the position of a given hydrophone is computed as a coherent superposition of beam influences. The beam influence is calculated along a given normal based on the expression [20, 3, 21]

$$P(s,n_1,n_2) = \frac{1}{4\pi}\sqrt{\frac{c(s)}{c(0)}\frac{\cos\theta(0)}{\det\mathbf{Q}}}\,\varPhi\exp\left[-i\omega\tau(s)\right] \ , \tag{1}$$

where $\varPhi = \prod_{\substack{i=1,2\\j=1,2}} \varPhi_{ij}$ and the coefficients are given by

$$\varPhi_{ij} = \exp\left[-(\ \frac{\sqrt{\pi|n_i n_j|}}{\varDelta\theta}Q^{-1}_{ij}\ )^2\right] \ , \tag{2}$$

with $\varDelta\theta$ standing for the elevation step between successive rays, and Q^{-1}_{ij} representing the elements of $\mathbf{Q}^{-1}$; n_1 and n_2 are calculated through the projection of $\mathbf{n}$ onto the polarized vectors along the ray; s corresponds to the ray arc length, and $c(s)$ and $\tau(s)$ stand for the sound speed and travel time along the ray, respectively; the complex matrix $\mathbf{Q}(s)$ describes the beam spreading, while $\mathbf{P}(s)$ describes the beam slowness.

3 Numerical enhancements

3.1 Calculation of normals

In the original version of TRACEO3D ray influence at a receiver located at the position $\mathbf{r}_h$ was calculated using the following procedure:

- Divide the ray trajectory into segments between successive transitions (surface/bottom reflection, or bottom/surface reflection, etc.);
- Proceed along *all* segments to find *all* ray normals to the receiver; to this end:
  - Consider the ith segment; let $\mathbf{r}_A$ and $\mathbf{r}_B$ be the coordinates of the beginning and end of the segment, respectively, and let $\mathbf{e}_A$ and $\mathbf{e}_B$ be the vectors corresponding to $\mathbf{e}_s$ at A and B; where $\mathbf{e}_s$ is defined as a unitary vector, which is tangent to the ray.
  - Calculate the vectors $\varDelta\mathbf{r}_A = \mathbf{r}_h - \mathbf{r}_A$ and $\varDelta\mathbf{r}_B = \mathbf{r}_h - \mathbf{r}_B$.
  - Calculate the inner products $P_A = \mathbf{e}_A \cdot \varDelta\mathbf{r}_A$ and $P_B = \mathbf{e}_B \cdot \varDelta\mathbf{r}_B$.

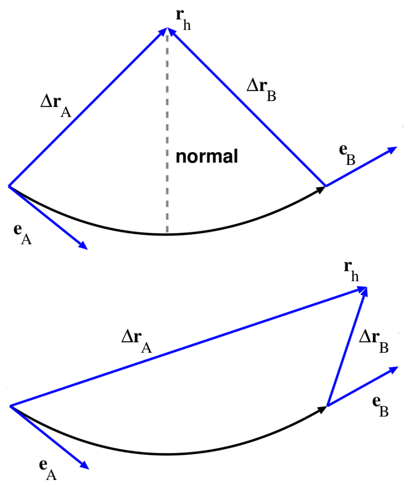


Fig. 1. Normal search along a ray segment, with $P_A = \mathbf{e}_A \cdot \Delta\mathbf{r}_A$ and $P_B = \mathbf{e}_B \cdot \Delta\mathbf{r}_B$. *Top:* the hydrophone is at a position for which $P_A \times P_B < 0$, thus a normal exists, and it can be found by bisection somewhere along the segment. *Bottom:* the hydrophone is at a position for which $P_A \times P_B > 0$; thus, there is no normal and the ray segment has no influence at the hydrophone position.

  - If $P_A \times P_B < 0$ a normal exists and it can be found through bisection along the segment; once the normal is found the corresponding influence at the receiver can be calculated.
  - If $P_A \times P_B > 0$ there is no normal (and no influence at the receiver); therefore, one can move to segment $i + 1$.
- The ray influence at the receiver is the sum of influences from all segments.

The search for a normal along a ray segment is illustrated in Fig. 1. The influence of a Gaussian beam decays rapidly along a normal, but it never reaches zero; therefore, the procedure is to be repeated for *all* rays and *all* receivers.

As shown in [17] field predictions using this method exhibit a good agreement with experimental data, but the runtime is often high and increases drastically as range, number of rays and number of sensors increase. The numerical enhancement of field calculations is described in the next Section.

3.2 The receiver grid strategy

To reduce drastically the runtime without compromising accuracy one can follow the approach described in [3], which suggests that for each ray segment one considers *not all* receivers, but only those “insonified” (i.e. bracketed) between the endpoints of a ray segment. For a given subset of receivers one can proceed sequentially within the subset (for instance, from the ocean surface to the ocean

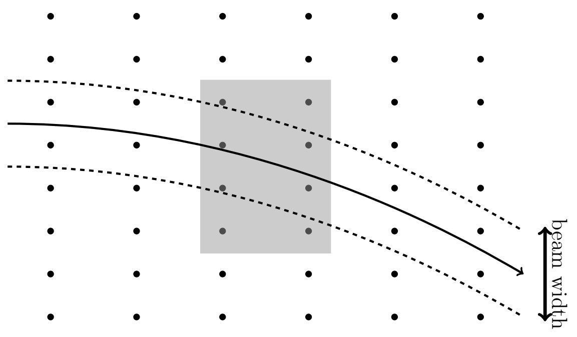


Fig. 2. The receiver grid (vertical view): the black dots represent *all* the receivers of a rectangular array, while the solid line represents the ray trajectory; the ray influence is only relevant within the limits of the beam width, represented by the dashed lines, and the gray rectangle represents the grid of receivers considered for the calculation of ray influence.

bottom), and rely on simple algebra to determine the parameters of ray influence; the procedure is then repeated for all ray segments. An examination of the BELLHOP3D ray tracing code [22] reveals that the determination of the subset of receivers is achieved by testing *all* receiver positions within the array, looking to find the ones within the endpoints of the ray segment. The approach implemented in TRACEO3D goes further, and looks to optimize the selection of a subset of receivers (called the *receiver grid*, see Fig. 2) based on the following considerations:

- A "finite" beam width W is defined along the ray, given by the expression
$$W = \left| \frac{Q_{11}(s)\Delta\theta}{\cos\theta(s)} \right| \ . \tag{3}$$
- There is no need to consider all receivers from the ocean surface to the ocean bottom, but only those within the neighborhood defined by W

The main idea on the basis of this strategy is that beyond the distance defined by W the influence is too small to be of any importance. Therefore, as one moves along each ray segment the receiver grid is determined by the receivers bracketed by both *the ray segment and* W. In this way one can avoid not only the query in the entire set of receivers forming the array, but also the query of all receivers bracketed by the ray segment. The method can take advantage of Cartesian coordinates to determine efficiently the indexes of the receivers lying inside the receiver grid. The specific details of this enhancement are described in the next Section.

3.3 Ray influence calculation algorithm

The specific details of optimization are shown in the pseudo-code of Algorithm 1, which summarizes the sequential steps regarding field calculations. Let n and r stand for the number of rays and receivers, respectively. The optimization starts by tracing the ray for a given pair of launching angles. Then, the algorithm marches through the ray segments, and solves the dynamic equations to calculate the ray amplitude and the beam spreading. As shown in lines 13 and 14 a subset of receivers is computed from r for each segment k of the ray. The ray influence is computed only if a normal to the receiver is found at a given segment (see lines 15 and 16). Line 23 presents the final step, in which coherent acoustic pressure for each receiver is calculated. As will be shown in Section 4 the set of nested loops constitutes a fundamental stage of the algorithm, allowing a substantial improvement of the model's performance. Details regarding the computation of the receiver grid are shown in the pseudo-code of Algorithm 2, where the integers l_{low} and l_{high} control the array indexes that form the receiver grid according to W at each coordinate axis. The receiver indexes increase or decrease their values, considering only the neighborhood, according to the position of the ray segment inside the receiving array; the entire procedure is designed to be flexible enough to account for different ray directions.

Algorithm 1 Sequential ray influence calculation

```
 1: load environmental data
 2: let $\boldsymbol{\phi}$ = set of azimuth angles
 3: let $\boldsymbol{\theta}$ = set of elevation angles
 4: let $\boldsymbol{r}$ = set of receivers
 5: consider $n = length(\boldsymbol{\phi}) \times length(\boldsymbol{\theta})$
 6: for $j := 1 \rightarrow length(\boldsymbol{\phi})$ do
 7:     for $i := 1 \rightarrow length(\boldsymbol{\theta})$ do
 8:         while ray $(\theta_i, \phi_j)$ exists do
 9:             solve the Eikonal equations for segment $k$
10:         end while
11:         for $k := 1 \rightarrow raylength$ do
12:             solve the dynamic equations of segment $k$
13:             calculate $W$ at segment $k$
14:             compute receiver grid $\boldsymbol{g}$ from $\boldsymbol{r}$ according to $W$
15:             for $l := 1 \rightarrow length(\boldsymbol{g})$ do
16:                 if $ray_k$ and $g_l$ are $\perp$ then
17:                     compute $ray_k$ influence at $g_l$
18:                 end if
19:             end for
20:         end for
21:     end for
22: end for
23: return the coherent acoustic pressure for each receiver
```

Algorithm 2 Compute the receiver grid

```
1: let $l_{low}$ = lower array index inside grid
2: let $l_{high}$ = high array index inside grid
3: consider $W$ as beam width at $ray_k$
4: while $l_{high}$ or $l_{low}$ are inside grid do
5:    if $W < \boldsymbol{r}\,(l_{low} - 1)$ then
6:       decrement $l_{low}$
7:    else if $W > \boldsymbol{r}\,(l_{low})$ then
8:       increment $l_{low}$
9:    else
10:      exit
11:   end if
12:   if $W < \boldsymbol{r}\,(l_{high})$ then
13:      decrement $l_{high}$
14:   else if $W > \boldsymbol{r}\,(l_{high} + 1)$ then
15:      increment $l_{high}$
16:   else
17:      exit
18:   end if
19: end while
```

4 Parallel GPU implementation

The proposed parallel implementation is addressed for NVIDIA [23] GPUs, using the Compute Unified Device Architecture (CUDA). This programming model implements a data-parallel function, denominated *kernel*, which is executed by all threads during a parallel step. Generally speaking, a CUDA program starts in the *host*, as a CPU sequential program, and when a kernel function is launched, it is executed in a grid of parallel threads into the GPU or *device*.

4.1 Memory organization

Acoustic predictions in a three-dimensional scenario demand the tracing of a high number of rays. In the sequential algorithm the ray trajectory information (such as, for instance, Cartesian coordinates, travel time, complex decay, polarized vectors, caustics, matrices $\mathbf{P}$ and $\mathbf{Q}$, etc.) are stored in memory to be used at later steps. Such storage makes sense considering that the sequential algorithm keeps one ray at a time in memory. However, handling thousands of rays in parallel rapidly exceeds the available memory in a given device. To circumvent this issue calculation of ray paths and amplitudes are performed in a single step, for each ray segment at a time, storing in memory only the values required to execute such calculation. A sketch of this strategy is presented in Fig 3, where the horizontal arrow represents the direction in which memories are updated, and t corresponds to the current time step in which calculations are taking place; $t - 1$ and $t - 2$ represent previous steps, that are required to be held in memory. Small arrows connecting memory positions represent the values accessed by the

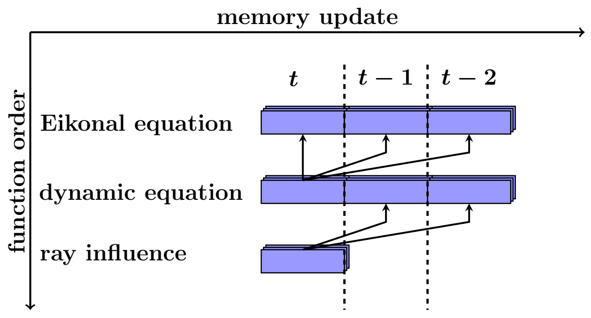


Fig. 3. Schematic representing the memory update sequence (horizontal arrow), where t stands for current computation time and $t-1$ and $t-2$ for previous times when values are held in memory. Small arrows connecting memory positions represent the values accessed for the corresponding function to perform computations in time t. The vertical arrow represents the order in which the functions are executed for a single ray segment.

corresponding function to perform computations in time t. The vertical arrow indicates the order in which functions are computed in the current time. After calling the functions for a given ray segment the values stored in memory at time $t-1$ are copied to the position $t-2$, and the values regarding t are copied to position $t-1$, meaning that the values at $t-2$ are discharged. A new iteration then starts to solve the next ray segment, following the same rules. In this way, the storage requires only three segments to be held in memory, reducing drastically the amount of data stored. This organization allows further updates of data into registers to be kept, reducing the global memory access and overcoming the problems of divergences in the pattern of memory access, a drawback of parallel ray tracing algorithm. The performance of memory access is also increased by loading part of the environmental information into the shared memory at the kernel initialization.

An overview of how data from the parallel implementation is organized into device memories is shown in Table 1. The memory type was chosen considering the respective data size and the frequency in which the data is accessed. For instance, data regarding environmental boundaries (surface and bottom) was initially put into the shared memory. However, when representing 3D waveguides, the number of coordinates became too large to fit in this type of memory and the data was thus moved to the global memory. On the other hand, the sound speed data was kept in shared memory since it was frequently accessed during ray trajectory calculations and the access takes place in an unpredictable order.

4.2 Parallel field calculation

A general view of the parallel version of field calculation is presented in Fig. 4 and in Algorithm 3, regarding the parallel flowchart and procedures, respectively.

Table 1. TRACEO3D memory organization into a parallel implementation: n_{ssp} is the number of points in the sound speed profile, n_{sur} and n_{bot} is the number of grid points defining the surface and bottom, respectively; n stands for the number of rays, h represents the number of receivers and m is the number of candidate regions.

Data	*Symbol or name*	*Type*	*Size*
source information		shared	12
environment parameters		shared	14
sound speed profile		shared	$3 + n_{ssp}$
array coordinates		global	$3 + 3 \times h$
surface coordinates		global	$7 + n_{sur}$
bottom coordinates		global	$7 + n_{bot}$
coherent acoustic pressure	cpr	global	h
cpr all rays	$ncpr$	global	$n \times h$
ray coordinates		register/local	3×3
travel time	τ	register/local	3
complex amplitude	A	register/local	3
polarized vectors		register/local	3×3
P, **Q**	**P**, **Q**	register/local	3×4

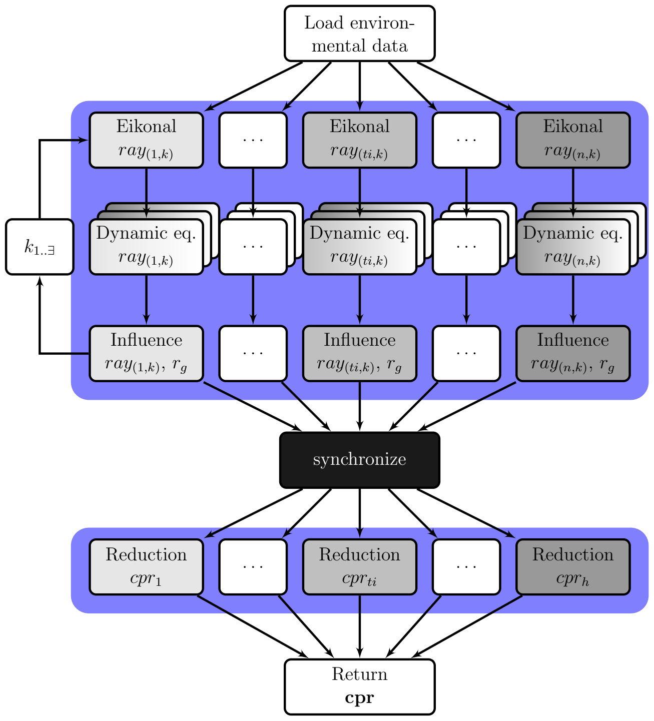


Fig. 4. Parallel flowchart of field calculation: blue regions correspond to kernel functions executed in parallel into device; outside blue regions the code is executed sequentially into the host.

Algorithm 3 Parallel field calculation

```
1: load environmental data
2: let $\boldsymbol{\phi}$ = set of azimuth angles
3: let $\boldsymbol{\theta}$ = set of elevation angles
4: let $\boldsymbol{r}$ = set of receivers
5: consider $n = length\,(\boldsymbol{\phi}) \times length\,(\boldsymbol{\theta})$
6: let $p$ = number of threads per block
7: let $b$ = n/p (number of blocks)
8: kernel $\ll b, p \gg$ ray influence calculation
9: synchronize
10: let $p$ = number of threads per block
11: let $b$ = h/p (number of blocks)
12: kernel $\ll b, p \gg$ pressure by sensor reduction
13: return the coherent acoustic pressure
```

Two main stages can be noted which corresponds to the blue regions at the flowchart and as parallel kernels in the algorithm. The strategy adopted the inherent ray tracing parallelism, addressing each pair of launching angles (θ, ϕ) as a single parallel thread, even though it could lead to the concentration of additional work per thread. However, since several instructions at the dynamic equations step are independent, they need to be organized sequentially to take advantage of instruction level parallelism (represented by the parallel blocks in depth) in the corresponding step. The proposed parallel algorithm is logically organized in a grid of b blocks, where each block has p threads. The first kernel (line 8) calculates the ray influence, where the number of threads launched into the device corresponds to n.

An overview of the kernel *ray influence calculation* is shown in Algorithm 4. Each thread computes the propagation of a single ray and its contributions to the entire field; the contributions are stored separately for each ray. It should be noted that, as shown in Table 1, the size of *ncpr* corresponds to $n \times h$ and the index to access global memory is calculated using a relative value of the grid index l' (see line 10 of Algorithm 4). After the kernel execution a device synchronization is performed to ensure that the acoustic field calculation for all rays was concluded. Then, a second kernel is launched to perform a parallel reduction over the values in *ncpr*. Each thread is addressed to a given receiver, and it adds sequentially the contribution of each ray to the corresponding receiver.

5 Validation

5.1 Implementation

The TRACEO3D model was written using the FORTRAN programing language in double precision. Thus, the interface was kept in FORTRAN, using its functions to read the inputs and write the outputs, while the parallel portion was encoded using the CUDA C platform. The CUDA C and FORTRAN environments

Algorithm 4 Kernel ray influence calculation

```
1: let $ti$ = block index $\times$ grid index + thread index
2: let $\theta_i = ti \bmod length(\boldsymbol{\theta})$
3: let $\phi_j = ti/length(\boldsymbol{\theta})$
4: while ray $(\theta_i, \phi_j)$ exists do
5:    solve the Eikonal equations for segment $k$
6:    compute the dynamic equations for segment $k$
7:    compute the receiver grid $\boldsymbol{g}$ from $\boldsymbol{r}$
8:    for $l := 1 \rightarrow length(\boldsymbol{g})$ do
9:       if $ray_k$ and $g_l$ are $\perp$ then
10:         $ncpr[ti + n \times l'] =$ acoustic pressure regarding $ray_k$ at $g_l$
11:      end if
12:   end for
13: end while
```

were connected using the ISO C Binding library [24], which is a standardized way to generate procedures, derived-type declarations and global variables, which are inter-operable with C. The parallel implementation was compiled in a single precision version (numerical stability was already addressed in [13]); comparisons between the parallel and the sequential version will be shown to properly clarify this issue. The single precision version allows the use of low-end devices or mobile equipments to provide predictions with high performance. Additionally, the FORTRAN sequential implementation was compiled with the optimization flag $-O3$, which was found to decrease the total runtime in 50%. The hardware and software features that were addressed when comparing the sequential and parallel model version of TRACEO3D are shown in Table 2. Fifteen runs were performed for the validation case. The maximum and minimum values were then discarded, and the average runtime was computed from the remaining thirteen runs.

Table 2. *Host*/*Device* hardware and software features.

Feature	Value	Unit
Host - CPU Intel i7-3930k		
Clock frequency	3500	MHz
Compiler	gfortran 5.4.0	–
Optimization flag	$-O3$	–
Device - GPU GeForce GTX 1070		
CUDA capability	6.1	–
CUDA driver	9.1	–
Compiler	nvcc 9.1.85	–
Optimization flag	none	–
Clock frequency	1683	MHz
Number of SM	15	–
Max threads per SM	2048	–
Warp size	32	–

5.2 The tank experiment

The laboratory-scale experiment took place at the indoor tank of the *Laboratoire de Mécanique des Fluides et d'Acoustique – Centre National de la Recherche Scientifique* (LMA-CNRS) laboratory in Marseille. The experiment was carried out in 2007 in order to collect 3D acoustic propagation data using a tilted bottom in a controlled environment. A brief description of the experiment (which is described in great detail in [5, 25]) is presented here. The inner tank dimensions were 10 m long, 3 m wide and 1 m deep. The bottom was filled with sand and a rake was used to produce a mild slope angle $\alpha \approx 4.5°$. For simulation purposes a scale factor of 1000 : 1 is required to properly account for the frequencies and lengths of the experimental configuration in the model. Thus, experimental frequencies in kHz become model frequencies in Hz, and experimental lengths in mm become model lengths in m. For instance, an experimental frequency of 180.05 kHz becomes a model frequency of 180.05 Hz, and an experimental distance of 10 mm becomes a model distance of 10 m. Sound speed remains unchanged, as well as compressional and shear attenuations. The *ASP-H* data set (for horizontal measurements of across-slope propagation) is composed of time signals, recorded at a fixed receiver depth denominated z_r, and source/receiver distances starting from $Y = 0.1$ m until $Y = 5$ m in increments of 5 mm, providing a sufficiently fine representation of the acoustic field in terms of range. Three different source depths were considered, namely $z_s = 10$ mm, 19 mm and 26.9 mm, corresponding to data subsets referenced as ASP-H1, ASP-H2 and ASP-H3, respectively. Acoustic transmissions were performed for a wide range of frequencies. However, comparisons are presented only for data from the ASP-H1 subset with the highest frequency of 180.05 kHz; this is due to the fact that the higher the frequency the better the ray prediction. Bottom parameters corresponded to $c_p = 1700$ m/s, $\rho = 1.99$ g/cm^3 and $\alpha_p = 0.5$ dB/λ. Sound speed in the water was considered constant, and corresponded to 1488.2 m/s. Bottom depth at the source position was $D(0) = 48$ mm.

5.3 Comparisons with experimental data

The set of waveguide parameters provided by the tank scale experiment was used to calculate predictions in the frequency domain. Transmission loss (TL) results are presented in Fig. 5, where *Bisection* means the original algorithm that the sequential version of TRACEO3D uses to calculate ray influence, *Grid* stands for the sequential method presented in Section 3.2, and *GPU Grid* corresponds to the parallel implementation. In general, model predictions were able to follow accurately the experimental curve over the full across-slope range. Nevertheless, a slight shift in phase can be observed at 2 km and 2.4 km in all simulation predictions. Besides, minor discrepancies can be noted between the predictions at the far field.

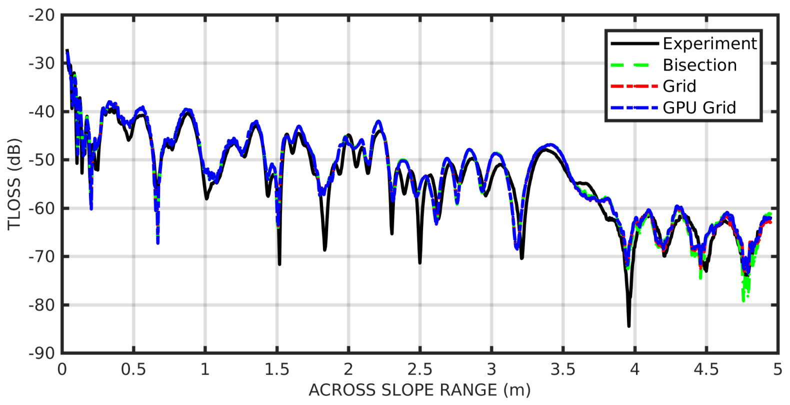


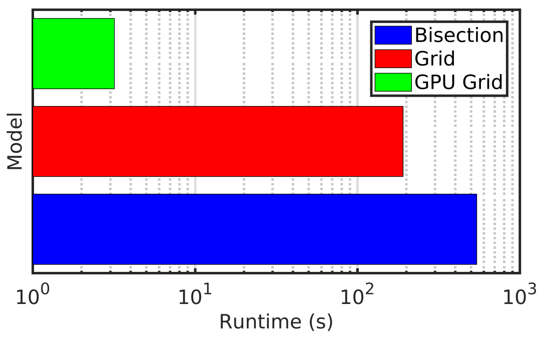
Fig. 5. Comparisons with the experimental data for LMA CNRS H1 @ 180.05 kHz.

5.4 Performance analysis

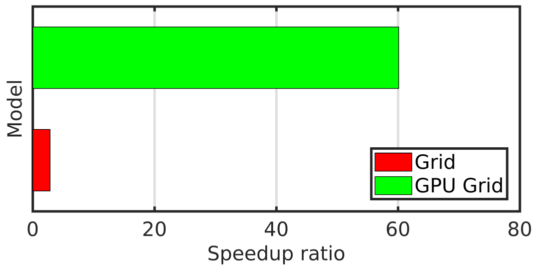
The best result found during the execution configuration optimization is presented on both Table 3 and Fig. 6. Speedup rates are presented separately, comparing the improvement regarding the numerical enhancement and the improvement achieved with the parallel GPU implementation. Thus, the speedup ratio of CPU (Grid) is calculated dividing the Bisection runtime by the Grid runtime, and for the CPU + GPU (Grid) dividing the Grid runtime by the GPU runtime. It is important to remark that the CPU (Grid) was able to decrease the runtime in 2.83 times, while the parallel GPU implementation achieved 60 times of performance, which indeed represents a outstanding improvement. Combining both speedups the total improvement was about 170 times (2.83×60.11), reducing the runtime **from 542.3 s to 3.18 s**. The mean square error (MSE) between the prediction generated by each model implementations and the experimental data is shown in Fig. 7. One can see that the difference among the implementations are of the same order of magnitude regarding the whole array of receivers. However, due to the far field discrepancies, the MSE differences among implementations are 0.084 dB from Grid to Bisection and 0.011 dB from GPU grid to Bisection. The parallel implementation only achieves such accuracy by using IEEE 754 compatible mathematical functions [26], and compiling without the flag *–fast-math*; because this flag enables performance optimization at the cost of introducing some numerical inaccuracies.

Table 3. Results of runtime and speedup ratio for TL predictions.

Model	CPU (Bisection)	CPU (Grid)	CPU + GPU (Grid)
Runtime (s)	542.3	191.16	3.18
Speedup ratio	1	2.83	60.11



(a)



(b)

Fig. 6. (a) Runtime and (b) speedup for TL predictions of the tank scale experiment. Speedup rates are presented separately, comparing the improvement regarding the numerical enhancement and the improvement achieved with the parallel GPU implementation.

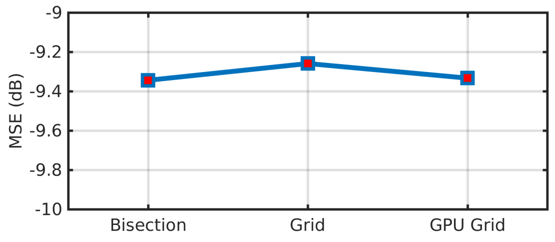


Fig. 7. MSE of TRACEO3D predictions against experimental data (LMA CNRS H1 @ 180.05 kHz) using three different approaches: Bisection, Grid and GPU Grid.

6 Conclusion

The discussion presented in this paper proposed numerical enhancements and a parallel GPU implementation of the TRACEO3D model. The calculation of ray influence was addressed using a receiver grid, i.e. a subset of adjacent receivers within the array, with the goal of decreasing runtime while keeping accuracy. After the enhancement of numerical issues parallel algorithms were developed considering a GPU architecture, that could take advantage of the inherent ray tracing parallelism and the high workload of 3D propagation. The validation results were performed using experimental data collected from a tank scale experiment. The method was found to be computationally efficient and accurate dealing with arrays containing a large number of sensors, although some optimization was required in order to define the proper borders of ray influence given by the finite beam width.

Performance results show that the implementation achieved a speedup around 170 times faster than the sequential one, combining the improvements of numerical enhancement and parallel implementation without compromising accuracy. Despite the significant improvements in speedup it can be not guaranteed that the adopted parallel algorithms exhausted all solutions of parallelization. It is believed that additional combinations of thread granularities and memory organization can have the potential to achieve a greater performance. The speedup issue is certainly of immense interest for intensive applications of a 3D model, a topic which is currently under intense discussion. Future work will be oriented to further validation in typical ocean environments with complex bathymetries and tests with different thread granularities, requiring new memory organization and execution configuration parameters.

References

1. O. G. Johnson, "Three-dimensional wave equation computations on vector computers," *Proceedings of the IEEE*, vol. 72, pp. 90–95, Jan 1984.
2. A. Tolstoy, "3-D propagation issues and models," *Journal of Computational Acoustics*, vol. 4, no. 03, pp. 243–271, 1996.
3. F. B. Jensen, W. A. Kuperman, M. B. Porter, and H. Schmidt, *Computational Ocean Acoustics.* New York: Springer Science & Business Media, 2th ed., 2011.
4. T. Jenserud and S. Ivansson, "Measurements and modeling of effects of out-of-plane reverberation on the power delay profile for underwater acoustic channels," *IEEE Journal of Oceanic Engineering*, vol. 40, no. 4, pp. 807–821, 2015.
5. F. Sturm and A. Korakas, "Comparisons of laboratory scale measurements of three-dimensional acoustic propagation with solutions by a parabolic equation model," *The Journal of the Acoustical Society of America*, vol. 133, no. 1, pp. 108–118, 2013.
6. P. C. Etter, *Underwater Acoustic Modeling and Simulation.* CRC Press, 4th ed., 2013.
7. S. M. Reilly, G. R. Potty, and M. Goodrich, "Computing acoustic transmission loss using 3D Gaussian ray bundles in geodetic coordinates," *Journal of Computational Acoustics*, vol. 24, no. 01, pp. 1650007/1–24, 2016.

8. C. Soares, F. Zabel, and S. M. Jesus, "A shipping noise prediction tool," in *OCEANS 2015-Genova*, pp. 1–7, IEEE, 2015.
9. R. Calazan and O. C. Rodríguez, "TRACEO3D ray tracing model for underwater noise predictions," in *Doctoral Conference on Computing, Electrical and Industrial Systems*, pp. 183–190, Springer, 2017.
10. D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach.* Morgan kaufmann, 2013.
11. P. Hursky and M. B. Porter, "Accelerating underwater acoustic propagation modeling using general purpose graphic processing units," in *OCEANS 2011*, pp. 1–6, IEEE, 2011.
12. X. Sun, L. Da, and Y. Li, "Study of BDRM asynchronous parallel computing model based on multiple cuda streams," in *Computational Intelligence and Design (ISCID), 2014 Seventh International Symposium on*, vol. 1, pp. 181–184, IEEE, 2014.
13. E. Ey, "Adaptation of an acoustic propagation model to the parallel architecture of a graphics processor," Master's thesis, University of Algarve, 2013.
14. R. Calazan, O. C. Rodríguez, and N. Nedjah, "Parallel ray tracing for underwater acoustic predictions," in *Proceedings of the 17th ICCSA2017*, vol. 10404, (3–6 July, Trieste, Italy), pp. 43–55, 2017.
15. "Open source high performance computing." https://www.open-mpi.org/. Accessed 2018-06-13.
16. "Ocean acoustics library." http://oalib.hlsresearch.com/. Accessed 2018-07-03.
17. O. C. Rodriguez, F. Sturm, P. Petrov, and M. Porter, "Three-dimensional model benchmarking for cross-slope wedge propagation," in *Proceedings of Meetings on Acoustics*, vol. 30, (25–29 June, Boston, MA), p. 070004, ASA, 2017.
18. R. Calazan and O. C. Rodríguez, "Simplex based three-dimensional eigenray search for underwater predictions," *The Journal of the Acoustical Society of America*, vol. 143, no. 4, pp. 2059–2065, 2018.
19. O. C. Rodriguez, J. M. Collis, H. J. Simpson, E. Ey, J. Schneiderwind, and P. Felisberto, "Seismo-acoustic ray model benchmarking against experimental tank data," *The Journal of the Acoustical Society of America*, vol. 132, no. 2, pp. 709–717, 2012.
20. V. Červenỳ and I. Pšenčík, "Ray amplitudes of seismic body waves in laterally inhomogeneous media," *Geophysical Journal International*, vol. 57, no. 1, pp. 91–106, 1979.
21. M. M. Popov, *Ray theory and Gaussian beam method for geophysicists.* Salvador, Bahia: EDUFBA, 2002.
22. M. B. Porter, "BELLHOP3D user guide," tech. rep., Heat, Light, and Sound Research, Inc., 2016.
23. "CUDA C programming guide," tech. rep., Nvidia Corporation, 2018. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Accessed 2018-05-16.
24. "The GNU FORTRAN compiler." https://gcc.gnu.org/onlinedocs/gfortran/Interoperability-with-C.html. Accessed 2018-06-05.
25. A. Korakas, F. Sturm, J.-P. Sessarego, and D. Ferrand, "Scaled model experiment of long-range across-slope pulse propagation in a penetrable wedge," *The Journal of the Acoustical Society of America*, vol. 126, no. 1, pp. EL22–EL27, 2009.
26. "Floating point and IEEE 754 compliance for NVIDIA GPUs," tech. rep., Nvidia Corporation, 2018. https://docs.nvidia.com/cuda/floating-point/index.html. Accessed 2018-05-31.