BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2025
http://excel.fit.vutbr.cz

GPU acceleration of acoustic field propagator

Martin Ludvík*

Abstract
The aim of the project is to accelerate the acoustic field propagator using graphic card. Acoustic field propagator is one step solver, that simulates how acoustic waves (sound) travels through medium. Original propagator is implemented using Matlab, that supports GPU acceleration of matrix math using gpuArray. To use the full potential of GPU computational power we can rewrite it in C++ and CUDA. With the use of precomputed values stored in constant memory of the GPU, and reduction of needed memory transfer, the speed up in some cases reach 90 times of the original matlab version. Even with the gpu accelerated Matlab version, CUDA is still 5 times faster and can handle 8 times bigger acoustic field grid. Thanks to speed up, researchers can simulate the behaviour of transmitter quicker, with the number of simulation in the thousands.

*xludvi11@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Ultrasound is an important tool in medical science. Today, the usage of ultrasound is spreading to the space of therapy. Transcranial ultrasound therapy is one method, where ultrasound is focused to either stimulate parts of the brain to treat conditions like depression or Parkinson's, or destroy tiny areas of damaged or malfunctioning brain tissue (like treating tremors or tumors). To prevent unintentional damage to healthy parts of the brain, we can simulate the wave propagation.

Acoustic field propagator (AFP) is a one-step wave propagation solver [1]. Compared to classic simulation, where we simulate the propagation with small time steps to the desired time, with AFP the solution is computed in one step using Green's function.

Originally the propagator is written using Matlab. The focus of this project was to accelerate the solver using GPUs. The first step was to rewrite the solution using C++. This was done to ensure correctness and for easier debugging. Simultaneously, in this step the parts of the propagator that can be precomputed were separated. The tested version written in C++ was then converted to a CUDA kernel.

Before launching the kernel, the constant parts of the propagator are precomputed on the CPU. These values are moved to GPU constant memory along with other variables. To allow starting bigger simulations, the propagator can be started in three versions: quick, balanced, and low memory, each with decreasing memory requirements. For speed up of execution of a large amount of small kernels, I used CUDA Graph. This way the startup of kernels is quicker and some operations can be executed in parallel.

Resulting in 800% acceleration compared to gpuArray Matlab version and up to 6000%. Additionally, by converting from garbage collection memory management in Matlab to manual, the peak memory usage is two times smaller. Exact speed-up of my solution compared to the original Matlab version and version using gpuArray is shown in Chart 1.

2. Acoustic field propagator

As mentioned before, AFP is a one-step wave propagation solver. That means we can solve the acoustic grid state at any time instantaneously. The equation Figure 2 describes the propagator function.[2] The $\mathcal{F}$ and $\mathcal{F}^{-1}$ represent forward and inverse Fourier transforms. $A(x)e^{i\phi(x)}$ represents a single-frequency ultrasound source.And finally, the $I(k,t)$ represents the core equation of the acoustic field propagator broken down in Figure 3.

3. Converting AFP to CUDA

The implementation consisted of three similar versions of the AFP. One using wrapping cancellation, second that supports nonlinear acoustics, and the final one with support for heterogeneous sound speed and absorption medium.

First step was to analyze the original implementations in MATLAB. I profiled the code to identify where the algorithm spends the most time. Based on this analysis, I focused on the core of the propagator, the equation shown in Figure 3.

Before converting code to CUDA, I decided to write the AFP for CPU in C++ first. This way, the process of rewriting the AFP is easier to verify and test for correctness. Start the rewrite process with conversion from matrix notation to loop notation. With this completed and compared to the original, next step was to identify the constant and repeating parts of the AFP equation. Separated parts that are the same for every point of the acoustic grid will be precomputed, and dependent parts can be computed only once per point.

To make the CUDA version quicker on GPUs that have slower double precision computing performance, the kernel and parameter classes are templated with a precision parameter, so the user can run all operations in single or double precision.

Process of converting this C++ CPU version to CUDA code was simple. Precomputed parts of AFP and needed parameters are copied to device constant memory, for quicker access and reduction of register usage. For fast Fourier transform I used CUDA library cuFFT that is highly optimized for execution on GPU. To reduce memory transfer, wave numbers are computed in the same kernel as AFP, whereas in the original version they are precomputed to matrix. When the memory transfer is necessary the host memory is allocated with cudaHostMalloc function so the memory is pinned and is not pageable.

I decided to allow users to select from three versions, quick, balanced, and low memory, that each differ in device memory requirements. For this AFP version, the exact memory requirements are shown in Chart 2. The theoretical requirement for the quick version can be defined by this equation:

$$M = numberOfPoints * poitSize * 3 \tag{1}$$

Balanced version only requires 2/3 and low memory 1/3 of the quick version.

3.1 Non-linear version

With this version, we can simulate the non-linearity of the source of acoustic waves. The AFP core of this version is very similar to the first, so the class for the kernel is inherited. But the number of kernels and memory transfers drastically increases. For this reason, I decided to use CUDA Graph to combat this reality. CUDA Graph helps with the reduction of kernel execution speed and parallel execution of memory and computational tasks. Compared to the wrap cancelling version, the device memory requirement is dependent on the number of harmonic frequencies we are solving for. The theoretical requirement for the quick version:

$$\begin{aligned} M = numberOfPoints * poitSize * \\ (numberOfHarmonics + 3) \end{aligned} \tag{2}$$

3.2 Heterogenous version

Last version is capable of simulating acoustic media with heterogeneous sound speed and absorption. Similarly to the previous nonlinear version, there is a need for a high number of iterations, so CUDA Graph is used. Quick mode solves the propagator equation only once and saves the result for multiplication in each iteration. On the other side is low memory mode that computes the propagator in each iteration to save memory usage. Device memory requirement looks as follows:

$$M = numberOfPoints * poitSize * 5 \tag{3}$$
$$M = numberOfPoints * poitSize * 3 \tag{4}$$
$$M = numberOfPoints * poitSize * 2 \tag{5}$$

3 Quick, 4 Balanced, 5 Low memory version.

4. Result of implementation

Program was tested on my machine with the following specs:

- CPU - Intel Core i7 13700k
- RAM - DDR5 64GB 5600MHZ
- GPU - Nvidia RTX 3080 Ti 12GB

Detailed results are shown in Chart 1 and Chart 2. Wave warping cancellation version speed-up compared to the original version is approximately 60x to 90x. Comparing my quick mode with MATLAB gpuArray version, my implementation is approximately 8 times quicker, while using 1/6 of the device memory.

My nonlinear AFP implementation is 90 times quicker, and heterogeneous AFP speed-up is 11x.

5. Conclusions

The state of the solution is satisfying, with only the downside of the memory limitation of a single GPU. The potential to use multiple GPUs to increase both speed and memory budget is what can be improved in my implementation.

Acknowledgements

I would like to thank my supervisor Jiří Jaroš for his help.

References

[1] Bradley Treeby, Jakub Budiský, Elliott Wise, Jiří Jaroš, and Ben Cox. Rapid calculation of acoustic fields from arbitrary continuous-wave sources. *Journal of the Acoustical Society of America*, 143(1):529–537, 2018. http://asa.scitation.org/doi/10.1121/1.5021245.

[2] Bradley E. Treeby and B. T. Cox. A k-space green's function solution for acoustic initial value problems in homogeneous media with power law absorption. *The Journal of the Acoustical Society of America*, 129(6):3652–3660, 06 2011. https://doi.org/10.1121/1.3583537.