Efficient simulation of multiple Mie-scattering events using NVIDIA CUDA

Simon Streicher, Oliver Kalthoff
Department of Medical Informatics, Heilbronn University, Germany
Hochschule Heilbronn Heilbronn University
ENGINEERING BUSINESS INFORMATICS

Motivation

Mie theory[1] describes the scattering of electromagnetic waves on spherical particles whose diameter is close to the wavelength of the incident radiation. A variety of computer algorithms[2] exist to calculate scattering parameters. These are sufficiently exact but computationally slow, so the simulation of radiation transport in a medium can be a time consuming task. To the best of our knowledge only few attempts have been made to adapt algorithms to parallel processors such as programmable graphics adapters.

During development, Matlab was chosen as a rapid-prototyping platform. Verifying our simulation (using a fixed parameter set) using Matlab Simulation code took over 1.5 hours to finish. This time-consuming simulation in Matlab was the reason to develop a GPU-accelerated version of our simulation software.

During development, problems occured - of course. These are described in the section „Problems“. Some solutions might be helpful for other people porting their Matlab code to the NVIDIA CUDA platform.

Problems

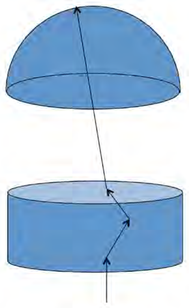
- maxing out the number of available cores/threads on GPU
- generating Pseudo-Random-Numbers
  - thread based
  - at runtime: it‘s unknown, how many pseudo-random numbers are neededduring simulation
  - seed-offset: thread based offset
- using a fast method of MonteCarlo simulation in Matlab
  - requires custom code in CUDA
  - massive complexity reduction

Methods & Implementation

We have implemented a parallel algorithm to calculate perpendicular and parallel polarization of scattered waves, from which other scattering parameters can be derived. This facilitates simultaneous tracking of particular waves in scattering media. Our code can be invoked from scripting languages like Matlab. It can be invoked from high-level languages such as C/C++ and is executable on conventional processors as well. We tested our approach using Monte Carlo simulations. The results correspond to reference implementations. Additionally we used a high precision measuring device with angular distribution of 0.1° to verify the simulation results.[3]

With Matlab and a CPU, one photon per processor is propagated. Using an Intel Core i5 CPU, 4 photons can be propagated at the same time. The simulation steps are described as follows:

1. inject a photon into cylindrical sample
2. scatter photon:
   1. calculate angles Θ,Φ
   2. calculate one free path
   3. repeat until stop criterium is reached
3. calculate position on detector
4. continue with next photon



The goal is to use all available threads of the GPU. During simulation process a photon is independent from each other photon. As stated before, when executing the simulation, we don‘t know how many times a photon will be scattered on it‘s path. This is the main reason, why we cannot precompute pseudo-random-numbers in order to generate them at thread-level. We decided to use the NVIDIA Thrust Library for implementation.

Using the transform-operation, we were able to adapt our algorithm to the GPU. Within the trasform-operation, a main functor, which performs the propagation process, is called.

Figure 1 gives an overview of the execution process involving the CPU for precalculations and the GPU to propagate the photons.

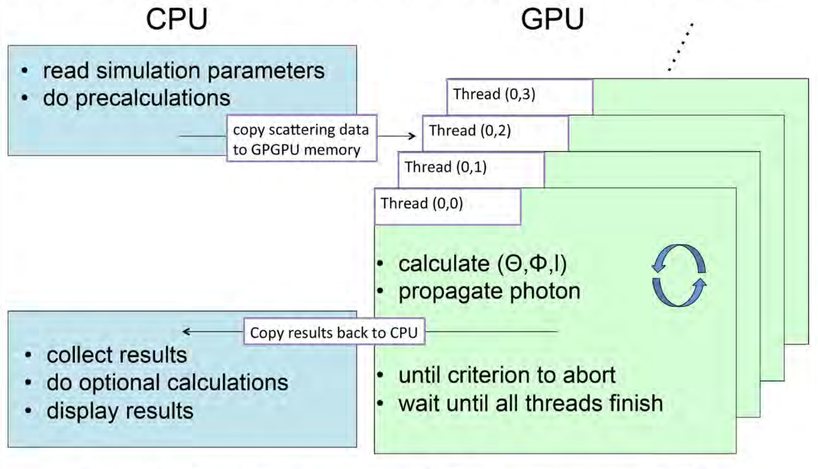


Figure1: Execution process of the scattering algorithm

Results

Table 1 shows the comparison of time consumption using a CPU and a GPU. For our CPU benchmarks we used an Intel Core i5 quad-core 2.66 Ghz with 16GB Ram, for the GPU benchmarks a GeForce TITAN with 6GB Memory was used. Matlab was used in version R2014b, CUDA version was 6.5 as well as THRUST version 1.7.1. The Matlab code was modified to use all four CPU cores. To measure execution time, 1E6 photons have been simulated.

Cylinder height	1000mm	500mm	250mm
CPU [sec]	1444,48	704,23	346,55
GPU [sec]	84,62	37,87	17,69
Speedup	17,07	18,60	19,59

Table1: Comparison of execution time

Table 1 compares execution times on CPU and GPU. Figure2 shows the “angle resolved scattering”, comparing the result from simulation with the result from the high precision measuring device.

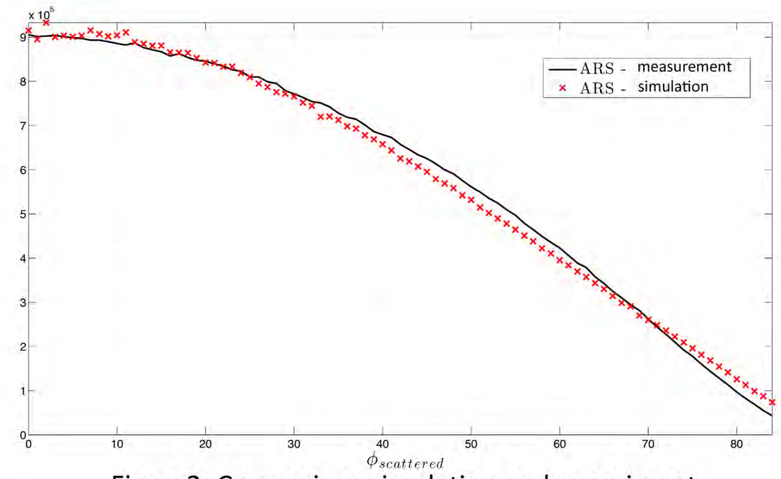


Figure2: Comparison simulation and experiment

Conclusion

We have shown that execution time can be reduced significantly compared to sequential approaches. Additionally we were able to do a laboratory experiment to verify our software simulation. By using a GPU, more complex scenarios can be simulated consuming less time than a CPU implementation.

All this was achieved using a stock NVIDIA GeForce TITAN.

Future work

- modify Mie theory to simulate deformed particles
  - requires modification of theoretical background
- optimize current algorithm
- use multiple GPUs and automatically scale amongst them
- using a hybrid-cluster: use CPUs to coordinate GPUs work

References

[1] Mie, G. (1908). Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen. A*nnalen der Physik*, *330*(3), 377-445.

[2] Craig F. Bohren and Donald R. Huffman. Absorption and Scattering of Light by Small Particles Wiley- VCH, April 1998. p. 477N.

[3] Streicher, S., Kampmann, R., Sinzinger, S., & Kalt- hoff, O. (2013, March). Efficient and precise simulation of multiple Mie scattering events using GPGPUs. In *SPIE OPTO* (pp. 86190K). International Society for Optics and Photonics.