Photon tracking with GPUs in IceCube

Dmitry Chirkin[a], for the IceCube Collaboration[1]

[a] *University of Wisconsin, Madison, WI 53703, USA*

Abstract

GPUs (graphics processing units) have become increasingly popular in the recent years for scientific calculations involving large numbers of similar steps. Photon propagation is a necessary part of simulating detector response to passing charged particles in IceCube that is an ideal application for use with GPUs. We discuss the principle ideas and practical issues of running such an application within the simulation chain used within our collaboration.

Keywords: photon propagation, IceCube

1. Photon tracking: introduction

Traditionally, tracking of the photons in IceCube is performed with photonics [1]. This involves tracking a sufficiently large number of photons within the detector so that probability distribution functions can be built for all interesting initial and final photon coordinates and directions, and for all possible arrival times. These are tabulated or optionally parametrized into interpolation tables and saved on disk. The tables are then used during the simulation or reconstruction to estimate the mean numbers of photons arriving at given times, and the actual numbers of photons are sampled from Poisson distributions with those means.

This process is complicated by the fact that the tables typically need to be created in 7 or more dimensions and each dimension needs to be tabulated with sufficiently small granularity. This normally leads to tables that are so large that they cannot be fit into the memory of typical computing nodes on which the simulation is run. The table generation is slow, the resulting simulation suffers from a wide range of binning artifacts, and the simulation is slow with much time spent loading the tables into memory. In the recently developed approach some of these problems are reduced by parametrization and interpolation techniques, which not only reduce the binning artifacts and improve precision of the tables, but also speed up the simulation.

Despite the recent improvements the process remains a 2-step process, in which the photon tables are first created and then used during the simulation. The reason for this has traditionally been the fact that running the simulation this way was still significantly faster than a direct approach, in which photons are created and propagated as needed, during the course of the simulation itself.

Email address: dima@icecube.wisc.edu (Dmitry Chirkin)

[1] http://icecube.wisc.edu

2. Photon Propagation Code

The photon propagation code (PPC) [2] was initially written to study the feasibility of direct photon propagation for simulation of events in IceCube. The simple nature of photon propagation physics allowed us to focus on the code optimization, to make sure the simulation ran as fast as possible. The simulation was written in C++, then re-written entirely in Assembly for the 32-bit i686 architecture with SSE vector optimizations. The Assembly version of the program used the SSE instructions for photon rotation and locating the optical sensor closest to the photon segment, while the calculation of the scattering angle was performed in one go using only the registers of the FPU stack.

A project called i3mcml [3] demonstrated that significant acceleration of the photon propagation is possible by using the graphics processing units (GPUs). We confirmed this with a version of PPC that employs the NVIDIA GPUs (graphics processing units) via the CUDA programming interface [4]. Recently a new version was written that additionally uses OpenCL [4], supporting both NVIDIA and AMD GPUs, and also multi-CPU environments. The relative performance of these different implementations (for simulating both in-situ light sources, or flashers, and Cerenkov light from muons) is compared in Table 1. We have studied the simulation with these versions of PPC and i3mcml and were able to demonstrate excellent agreement between them.

	C++	Assembly	GTX 295 GPU
flasher	1.00	1.25	147
muon	1.00	1.37	157

Table 1: Speedup factor of different implementations of PPC compared to the C++ version. The GPU used in this comparison was either of the two in the NVIDIA GTX 295 video card.

The reason for the substantial acceleration of PPC on GPUs is the highly parallel nature of the simulation of the photon propagation. All of the simulated photons go through the same simulation steps (see Figure 1): photon propagation between the scattering points, calculation of the scattering angle and new direction, and evaluation of whether the current photon segment intersects any of the optical sensors of the detector array. The GPUs are designed to perform the same computational operation in parallel across multiple threads. Each thread works on its own photon for as long as the photon exists. When the photon is absorbed or hits the detector the thread receives the new photon from a pool of photons for as long as that pool is not empty. Although a single thread runs slower than a typical modern computer CPU core, running thousands of them in parallel results in the much faster processing of photons from the same pool on the GPU.

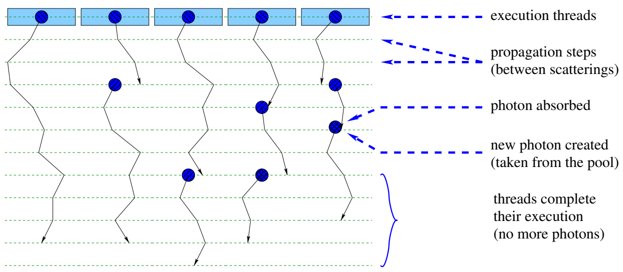

Figure 1: Parallel nature of the photon propagation simulation: tracking of photons entails the computationally identical steps: propagation to the next scatter, calculation of the new direction after scatter, and evaluation of intersection points of the photon track segment with the detector array. These same steps are computed simultaneously for thousands of photons.

3. Simulation with photon propagation code

The direct photon simulation with PPC is typically used in the two scenarios shown in Figure 2. In the first the in-situ light sources of the detector are simulated for calibrating the detector and the properties of the surrounding ice. It is possible to very quickly re-simulate the detector response to a variety of ice scattering and absorption coefficients finely tabulated in depth bins. This allows for these coefficients to be fit directly, by finding the combination that is a best simultaneous fit to all of the in-situ light source calibration data [5]. For the 10 meter depth bins, 200 coefficients are fitted (with scattering and absorption defined in 100 layers spanning 1 km of depth of the detector), with nearly a million possible ice parameter configurations tested in less than a week on a single GPU-enabled computer. This method is intractable with the photonics-based simulation, as each new parameter set would require generation of the new set of photonics tables, each generation taking on the order of a week of computing time of a $\sim$ 100-CPU cluster.

In the second scenario the Cerenkov photons created by the passing muons and cascades are simulated as part of the larger simulation of the detector response to atmospheric and other fluxes of muons and neutrinos. The simulation is able to account for some effects that are difficult to implement with the photonics-based simulation, because their simulation would lead to additional degrees of freedom, thus increasing the size of the parametrization effort and tables many-fold. One of these

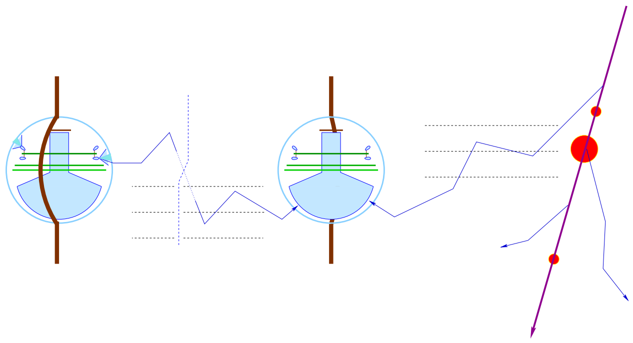

Figure 2: Typical simulation scenarios: photons emitted by the detector are tracked as part of the calibration procedure (left). Cerenkov photons emitted by a passing muon and cascades along its track are tracked to simulate the typical IceCube events (right).

is the tilt of the ice layers, i.e., dependence of the ice parameters not only on the depth, but also on the xy surface coordinates (shown in Figure 3).

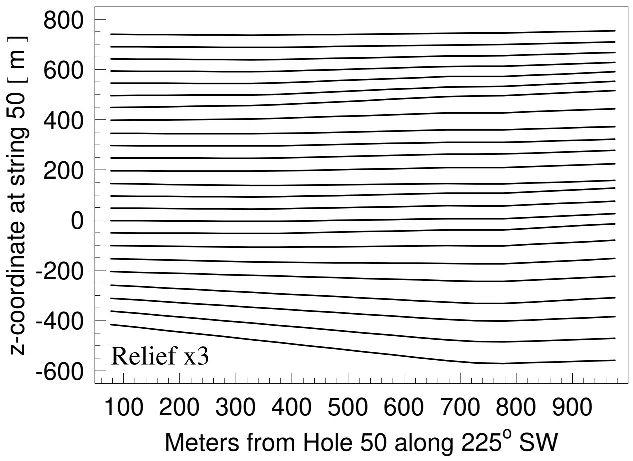

Figure 3: Extension of ice layers along the average gradient direction (taken from [6]). The y-axis shows the layer shift (relief) from its position at the location of a reference string at the distance shown on the x-axis from this string along the average gradient direction (225 degrees SW). The relief shown is amplified by a factor of 3 for visual clarity of the ice layer tilt.

Effects that are treated precisely with PPC (and only approximately with photonics) include the simulation of the longitudinal profile of light generation by cascades and the angular distribution of the Cerenkov photons around the emitting muons or cascades.

Other effects implemented recently only into PPC include the direct simulation of the somewhat different ice properties in the column of ice refrozen around the detector strings after they have been deployed; and the slight azimuthal dependence of the scattering function.

4. Concurrent execution and runtime optimization

PPC keeps track of several execution time counters that help judge the performance of the GPU code. One counter operates

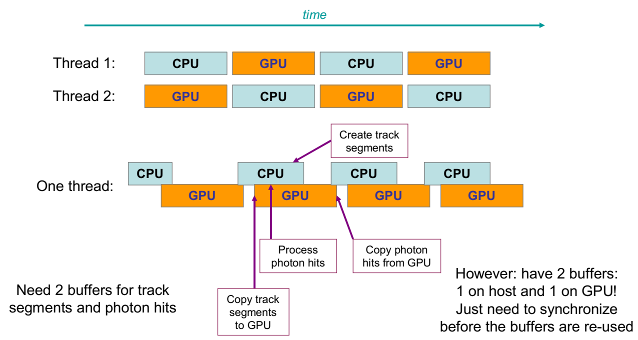

Figure 4: Concurrent execution on CPU and GPU sides. Two possible solutions are shown: the first (top) requires running at least two ppc threads was eventually replaced with the solution (bottom) that is enabled in a single thread.

on the CPU side, by calculating the time elapsed waiting for the GPU code to return. While waiting for the GPU code the CPU can run other parts of the simulation (e.g., trigger simulation), but to maximize the use of the GPUs these parts should finish quicker than the GPU part. The typical utilization of the GPUs achieved in our tests is $\gtrsim 90\%$. The implementation of the concurrent execution of the program on both CPU and GPU sides is facilitated by the fact that the GPU side works on its own memory buffer, which is exchanged with the main computer memory buffer only at the beginning or at the end of the execution on the GPU side. So, while the CPU processes photon detector hits created by the previous run on the GPU, the GPU is running through the photons previously prepared by the CPU (see Figure 4).

The other two execution time counters calculate the minimum and maximum time spent by different threads running on the GPU. If these are close to each other, all execution units of the GPU have been equally loaded. The difference we observe is typically on the order of $\sim 0.5\%$.

5. Hardware considerations: our GPU cluster

The simulation of a single day of experimental background data of the full IceCube detector completes in about 10 days of calculation on a small 3-computer GPU cluster equipped with 18 GPUs (3 NVIDIA GTX 295 cards per computer, each card contains 2 GPUs). This 3-computer cluster was built from consumer-available parts for approximately $10k. This cluster is assisted by a larger CPU-only computing cluster that runs the simulation of cosmic rays and several other tasks.

Unfortunately we experienced problems with 3 out of the 24 GPUs that we have experimented with. The problems included jobs exiting with errors, GPUs locking up, or sometimes even the entire computer locking up and requiring rebooting. Further investigation revealed that the jobs that fail also occasionally produce NANs or INFs in their GPU threads. With a small bit of in-line assembly we found that only threads running on specific hardware multiprocessors (MPs) on each of the faulty GPUs (each of the GTX 295 GPUs has 30 MPs) are affected. By checking within threads during execution whether they are running on the faulty MP and immediately stopping those threads it was possible to use the faulty GPUs for calculation at 29/30 of its capacity. The jobs running on these GPUs are 3% slower, but run without further problems. We do continuously monitor and record the unexpected NAN or INF conditions in the GPU threads or problems reported by the CUDA interface; however these became extremely rare after disabling the 3 faulty MPs as described above.

To optimize the use of the GPU-enabled computers further we are experimenting the Directed Acyclical Graph (DAG) tools [7]. This involves separating simulation segments into tasks, and assigning these tasks to DAG nodes. DAG assigns separate tasks to different computer nodes; execution of photon propagation simulation is performed on dedicated GPU nodes.

For many simulations the GPU segment of the simulation chain is much faster than the rest of the simulation. For these, a small number of GPU-enabled machines can consume the data from a large pool of CPU cores. However, the optimal DAG configuration differs depending on the specific simulation.

We are currently running the GPU simulation routinely on our cluster at UW-Madison, which is being upgraded to include 48 more Tesla M2070 GPUs (built around the 3 x DELL PowerEdge C410x and 6 x DELL PowerEdge C6145 for $\sim$ $200k). We are experimenting with running the PPC-based simulation on other IceCube sites in U.S., Canada, and Germany.

6. Concluding remarks

We have developed a photon propagation tool that can replace the older two-step photon tracking paradigm in certain situations, while achieving more precision, better description of the physics of the process and shorter run time. The program is capable of running on both CPU cores and GPU hardware, achieving very significant speed up (factors in excess of ~ 100) on the latter.

Although we have encountered some hardware problems while running on 3 out of 24 of our consumer-grade GPU cards, it was possible to identify and disable the faulty parts of the GPUs and continue to use the problem cards at 97 % capacity. We are in a process of building a professional-grade cluster of high-end GPU nodes, which will contribute to further production of the simulated data.

References

[1] J. Lundberg, et al., Light tracking through ice and water - scattering and absorption in heterogeneous media with photonics, Nucl. Instrum. Methods A 581 (2007) 619. arXiv:astro-ph/0702108v2.
[2] D. Chirkin, photon propagation code: http://icecube.wisc.edu/~dima/work/WISC/ppc.
[3] T. Abuzayyad, i3mcml: http://wiki.icecube.wisc.edu/index.php/i3mcml.
[4] NVIDIA CUDA: http://www.nvidia.com/cuda.
[5] D. Chirkin, et al., Study of south pole ice transparency with icecube flashers, Contribution to the 32st International Cosmic Ray Conference, Beijing, China, 11-18 August 2011, session HI.2.3, contribution 0333.
[6] R. C. Bay, et al., South pole paleowind from automated synthesis of ice code records, J. Geophys. Res. 115 (2010) D14126.
[7] DAG: http://en.wikipedia.org/wiki/directed_acyclic_graph.