Snowmass2021 - Letter of Interest

IceCube and IceCube-Gen2 Experimental Algorithm Parallelization

Thematic Areas: (check all that apply □/■)
■ (CompF1) Experimental Algorithm Parallelization
■ (CompF2) Theoretical Calculations and Simulation
■ (CompF3) Machine Learning
□ (CompF4) Storage and processing resource access (Facility and Infrastructure R&D)
□ (CompF5) End user analysis
□ (CompF6) Quantum computing
□ (CompF7) Reinterpretation and long-term preservation of data and code

Contact Information:
Alex Olivas (University of Maryland) [aolivas@umd.edu],

Authors (alphabetical):
Kevin Meagher (University of Wisconsin–Madison) [kmeagher@icecube.wisc.edu],
Alex Olivas (University of Maryland) [aolivas@umd.edu],
David Schultz (University of Wisconsin–Madison) [dschultz@icecube.wisc.edu],
on behalf of the IceCube[1] and IceCube-Gen2[2] Collaboration [analysis@icecube.wisc.edu]

Abstract:
The IceCube Neutrino Observatory is a km^3 neutrino detector deployed at the South Pole. IceCube measures neutrinos by detecting the optical Cherenkov photons produced in neutrino-nucleon interactions. The IceCube Neutrino Observatory is currently planning for a significant upgrade, which includes various new types of calibration modules, further increasing the computational needs of simulation and data processing. The increase in complexity and competition for precious resources demands their efficient use once acquired. It is well known in many computational communities, both within and outside the scientific community, that future gains in hardware performance are going to come from additional cores and specialized hardware designed with a concurrent programming model in mind (e.g. GPU, TPU, etc...). The core software group in IceCube is planning to adapt its production code to take advantage of both specialized and standard multi-core, multi-GPU systems.

[1] Full author list available at https://icecube.wisc.edu/collaboration/authors/snowmass21_icecube
[2] Full author list available at https://icecube.wisc.edu/collaboration/authors/snowmass21_icecube-gen2

Introduction

The IceCube Neutrino Observatory [1], located at the South Pole, instruments a cubic kilometer of Antarctic ice. IceCube uses 5160 digital optical modules (DOMs) arranged on 86 strings in a hexagonal array to detect Cherenkov radiation from relativistic charged particles emitted during neutrino interactions. This configuration can detect neutrinos as high as EeV energies while the center, more concentrated region of DOMs, called DeepCore, is optimized to extend the detection energy down to a GeV. The IceCube Upgrade plans to improve on the resolution of GeV neutrinos by adding an additional seven strings concentrated around DeepCore with varying DOM designs and additional calibration devices [2]. To improve resolution of TeV to EeV neutrino detection and increase the detection rate, Gen2 will add 120 strings to instrument a total volume of 7.9 km^3 [3].

With the ever increasing demands on computational resources and increasing complexity of modern analyses it is becoming critical to utilize all accessible resources efficiently. This also requires the ability to run in diverse, heterogeneous environments, since cloud, grid, and cluster configurations are rarely coordinated with global homogeneity as a goal. The modern scientific software stack needs to be able to run efficiently on a wide range of operating systems and hardware. Concurrency, on all levels, is the key to achieving optimal efficiency. Machine Learning is driving the development of specialized hardware, and code developed for a single core, single GPU system is rarely trivially portable. IceCube has tentative plans to invest over the next several years in porting code to Kokkos [4] (or another similar framework/library) in order to broaden the distributed hardware available to the collaboration. This includes IceCube's internal C++/Python-based framework. While the hardware access gains are obvious, this comes at a cost where a continued, and possibly increased, investment in modern C++ training, emphasizing best practices, will be needed.

Existing Parallel Code

The the highly parallel nature of the simulation of photon propagation benefits with substantial acceleration of our simulation on GPGPUs [5]. All of the simulated photons go through the same steps before getting absorbed or hitting a sensor: photon propagation between the scattering points, calculation of the scattering angle and new direction, and evaluation of whether the current photon segment intersects with any of the optical sensors of the detector array. Each GPU performs the same computational operation on individual photons in parallel across multiple threads. Although a single thread runs slower than a typical modern CPU core, running thousands of them in parallel results in much faster processing of photons on the GPU. Through the use of GPUs, we have achieved significant acceleration of the photon propagation, by factors of 150 or more compared to running on a single CPU core. The same code that propagates photons in simulation can be used for reconstruction as well, allowing for on-the-fly event hypothesis generation.

Future Algorithms

The IceCube detector simulation includes individually calibrated PMT waveforms, optimized event resampling for low-energy background simulation, and detailed models of the optical properties of the ice. The detector simulation is currently the second largest consumer of resources in the simulation chain. The current detector, consisting of roughly 5160 identically operating Digital Optical Modules (DOM), naturally lends itself to parallelization. Given the large number of modules that can potentially process waveforms in a given event, there are potential gains that can be achieved through the use of GPUs, as opposed to multi-core systems, for both the detector response simulation and the waveform deconvolution. The potential gains from a GPGPU implementation will soon be an active area of study within the IceCube software group.

A potential area of overlap with the Machine Learning group is the use of Generative Adversarial Networks (GANs) for fast simulation of traditionally resource-intensive showers. Simulation of sufficient background statistics continues to limit several analysis channels in IceCube, especially analyses sensitive to muon bundles from cosmic-ray air showers (common at moderate to high energies). Background-reduction techniques developed internally are currently limited in utility to single-muon backgrounds. Simulating backgrounds that include muon bundles (multiple muons originating from a single hadronic shower, with relatively small lateral spacings and small opening angles) still requires a full shower simulation using CORSIKA where only the information from energetic muons at IceCube depths is saved. We forsee the use of GANs to quickly simulate muon bundles at IceCube depths becoming an active area of investigation over the next several years.

IceCube's core production chain was written serially, with only select additions being able to take advantage of multiple cores or accelerators. With the resurgence of heterogeneous computing and hardware performance expected to improve mainly via increased counts of GPU and CPU cores, we will need to adapt our core software to allow for parallel execution. Such changes allow us to consider other languages and libraries that may not have existed two decades ago when the IceCube software was first developed. While `Go` is somewhat popular in the system space, the garbage collector is an unpalatable comparison to Python's memory issues. `Rust` is another alternative, with special interest in its memory correctness by design. Modern versions of C++, such as C++20, offer significant improvements over C++98 while still being somewhat familiar to programmers. Parallel libraries are also of interest for use on accelerators, with Kokkos currently as the most attractive option. Such libraries, if fully integrated throughout the codebase, would allow significant portability, running a single codebase on multiple architectures.

Best Practices

The ability to predict resource usage, to allow for efficient scheduling in a distributed environment, is of high priority. This need will become even more so as the number, type, complexity, and density of optical modules increases during the Upgrade. Software engineering best practices dictate knowing the time and space complexity of each critical component of a software system. IceCube invests in code reviews as an attempt to manage and mitigate the accumulation of technical debt. Code reviews are required for all new projects before inclusion into the production stack, though historically there has not been a requirement on reporting, let alone optimizing, components' resource usage. It is the goal of IceCube to fully understand resource scaling behaviors in the core production chain and make reporting and optimization a criteria in future code reviews.

The IceCube software group is currently working on tools to expand use of standard CI/CD practices and extend them to include Continuous Benchmarking and Validation (CB/CV), to catch software commits that impact resource usage and high-level physics systematics, respectively, in a timely manner (e.g. nightly). This will become even more critical as IceCube adopts, develops, and ports more concurrent software, which can be notoriously difficult to debug compared to serial implementations.

References

[1] M. G. Aartsen et al. The IceCube Neutrino Observatory: Instrumentation and Online Systems. *JINST*, 12(03):P03012, 2017.

[2] Aya Ishihara. The IceCube Upgrade – Design and Science Goals. *PoS*, ICRC2019:1031, 2020.

[3] M.G. Aartsen et al. IceCube-Gen2: The Window to the Extreme Universe. 8 2020.

[4] Kokkos: Core libraries. https://github.com/kokkos/kokkos.

[5] Martin Merck, Dmitry Chirkin, Juan Carlos Diaz Velez, and Heath Skarlupka. IceCubes GPGPU's cluster for extensive MC production. *J. Phys. Conf. Ser.*, 396:022046, 2012.