Reconstruction of Low Energy Neutrino Events with GPUs at IceCube

Maicon Hieronymus[1], Bertil Schmidt[1], and Sebastian Böser[2]

[1] Institute of Computer Science, Johannes Gutenberg University, Mainz, Germany
[2] Institute of Physics, Johannes Gutenberg University, Mainz, Germany

Abstract. IceCube is a cubic kilometer neutrino observatory located at the South Pole that produces massive amounts of data by measuring individual Cherenkov photons from neutrino interaction events in the energy range from few GeV to several PeV. The actual reconstruction of neutrino events in the GeV range is computationally challenging due to the scarcity of data produced by single events. This can lead to run times of several weeks for the state-of-the-art reconstruction method – `Pegleg` – on CPUs for typical workloads of many ten-thousand events. We propose a GPU version of `Pegleg` that probes the likelihood space with several hypotheses in parallel while adapting the amount of parallel sampled hypotheses dynamically in order to reduce computation time significantly. Our results show an average speedup of 14 (with a maximum of over 200) for 5262 reconstructed neutrino events of different flavors on a Titan V GPU compared to the multithreaded CPU version, which enables quicker and broader analysis of IceCube events.

Keywords: Neutrino Oscillation · Neutrino Physics · MultiNest · Reconstruction · GPU.

1 Introduction

IceCube [10] is a cubic kilometer neutrino observatory located at the South Pole. Neutrinos are elementary particles that exist in three different flavors ν_e, ν_μ and ν_τ. Unlike any other particles, neutrinos can change their flavor during propagation [8]. This phenomenon, the so-called neutrino oscillations, implies that neutrino flavours have different masses. The question which of the three neutrino flavors is the heaviest is being investigated under the term neutrino mass ordering. While most neutrino oscillation experiments are insensitive to this question, IceCube can address it through precision measurements of the ubiquitous flux of atmospheric neutrinos and the subtle effects the very dense matter in the earth core has on their flavor oscillations. These matter effects only appear for relatively low energy neutrinos below 15 GeV [1, 13]. With such low energies only very few photon hits per event are detected in IceCube, heavily decreasing the signal-to-noise ratio. Neutrino events are reconstructed by comparing the observed hit-pattern to the expected hit pattern given an 8-dimensional event hypothesis using a maximum likelihood method. The expected light intensity at

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

any position and time for a given hypothesis is obtained from spline approximations to tabulated simulation data (so called *photosplines* [17]) that exploit various approximate symmetries of the light emission and propagation process. Even without noise, the reconstruction of such an event from the signal in IceCube is suffering from causality requirements on the photon arrival times, causing a likelihood function that is neither globally convex nor continuous and has many local minima. In particular gradient descent minimizers and algorithms only using local information struggle with this likelihood function, which increases the complexity of the event reconstruction problem. Efficient and fast evaluation of a suitable event hypothesis has thus become a limiting factor for the analysis of IceCube data in order to study neutrino properties. The algorithm that is currently used, `Pegleg` [16], partitions the exploration of the event likelihood space into two: (i) The vertex of the neutrino interaction and the direction of secondary particles emerging from this are optimized using MultiNest [7, 6] – a multi modal nested sampling algorithm that handles degenerated likelihoods. (ii) Track length of the emerging muon and its energy depositions caused by the event are optimized separately. The reconstruction takes up to 10 mins per event on a typical workstation. This in turn leads to run times of several weeks for a typical analysis with tens of thousands of events [12, 11].

In this paper, we present a GPU version of `Pegleg`, which features a massively parallel MultiNest probing algorithm (explained in Section 3) and massively parallel spline evaluation (explained in Section 4). Previous work on accelerating neutrino oscillation data analyses on GPUs has mainly focused on direct neutrino propagation to calculate oscillation probabilities [3, 15] or direct photon propagation for a given hypothesis. This work is the first to reconstruct neutrinos given measurements from an interaction event by evaluating splines on a GPU.

2 Background

2.1 Millipede Likelihood

Consider the light yield at different positions and times of an interaction event of a neutrino with ice. We search for the parameters of the underlying event, i.e. the source vertex of the event, the energy and the direction of the neutrino (which mostly coincides with the direction of the secondary particles). If a muon neutrino ν_μ interacts in a charged-current interaction, it will also create a muon that can travel several ten meters through the ice, allowing for a) differentiation of ν_μ from other flavors and b) significantly improved directional resolution. The event hypothesis thus has eight free parameters $(\boldsymbol{y}, t, E, L)$ with $\boldsymbol{y} = (y_1, y_2, y_3, y_4 = \theta, y_5 = \phi)^{\mathrm{T}}$ the coordinates and direction of the neutrino event, t the time it occurred, E the energy of the neutrino and L the track length of an outgoing muon if present. The muon track is divided into many segments of fixed length, hence the name Millipede. The light detected by the i-th DOM (Digital Optical Modules) at the position $\boldsymbol{x}_i$ is a superposition of the light emitted at each segment (see Fig. 1) [18]. Eq. 1 describes the amount of

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

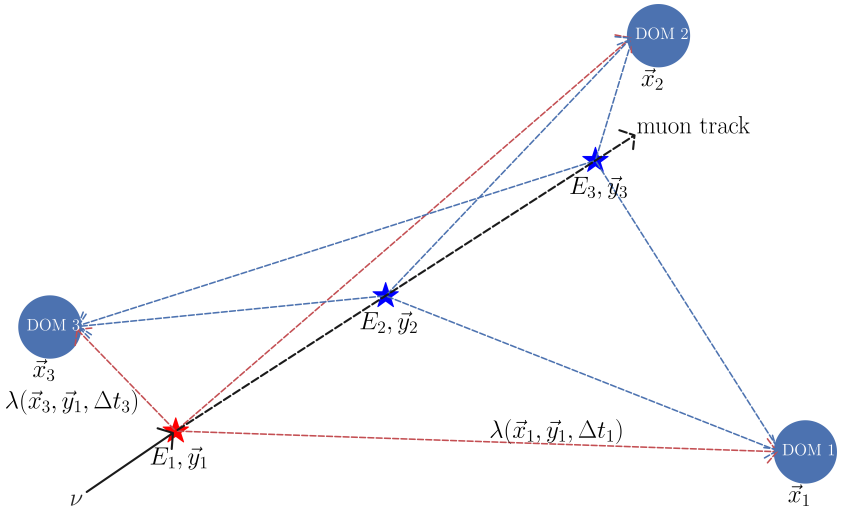

Fig. 1. A sketch of the millipede hypothesis with an incoming neutrino from the bottom left. The red star marks the origin of the neutrino event, with the blue stars marking track segments of an outgoing track. Each star is handled as a distinct photon source. The colored dotted lines depict the photons that arrive at each DOM given a photon source.

expected photons μ_l for a time bin k of a DOM i, where l is the index over all time bins of all DOMs.

$$\mu_l = \rho_l + \sum_{j=1}^{J} \lambda(\boldsymbol{x}_i, \boldsymbol{y}_j, \Delta t_{i,k}) \cdot E_j. \tag{1}$$

In Eq. 1, λ denotes how the expected light intensity at a position $\boldsymbol{x_i}$ scales with the energy loss E_j at a position $\boldsymbol{y}_j$. $\Delta t_{i,k}$ is the time delay between the photon production and detection at sensor i during time bin k. ρ_l is the noise which is specific for every DOM, such that $\rho_l = \rho_{l'}$ for all $l \neq l'$ that belong to the same DOM. λ is typically evaluated by using spline tables that describe a B-spline surface over a rectangular d-dimensional knot grid [17]. Each DOM i can have a different amount n_i of valid time bins k that are used during reconstruction such that we have a total of M DOMs, J energy losses and $M' \geq M$ total time bins. We can summarize Eq. 1 for all DOMs by Eq. 2.

$$\begin{pmatrix} \mu_1 - \rho_1 \\ \mu_2 - \rho_2 \\ \vdots \\ \mu_{M'} - \rho_{M'} \end{pmatrix} = \begin{pmatrix} \lambda_{1,1} & \lambda_{1,2} & \dots & \lambda_{1,J} \\ \lambda_{2,1} & \lambda_{2,2} & \dots & \lambda_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{M',1} & \lambda_{M',2} & \dots & \lambda_{M',J} \end{pmatrix} \cdot \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_J \end{pmatrix}. \tag{2}$$

Given a loss pattern we calculate the likelihood of photon counts, such that the predicted photon counts μ can be compared to the actual signal. The amount

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

of detected photons N_l at one time bin of one DOM follows a Poisson distribution [9] with mean μ_l, which expands for N_l photons to [4]:

$$\mathcal{L}_l = \frac{\mu_l}{N_l!} e^{-\mu_l}. \tag{3}$$

The overall likelihood is the product of all contributions:

$$\mathcal{L} = \prod_{l=1}^{M'} \mathcal{L}_l = \prod_{l=1}^{M'} \frac{\mu_l}{N_l!} e^{-\mu_l}. \tag{4}$$

In order to reduce numerical instabilities, it is common to use the logarithm and work with the log-likelihood (llh) or the negative log-likelihood, which in turn needs to be minimized. Using the continuous charge variable instead of discrete photon counts [16] we can approximate the factorial with the gamma function, resulting in Eq. 5.

$$\ln \mathcal{L} = \sum_{l=1}^{M'} (N_l \cdot \ln \mu_l - \ln(\Gamma(N_l + 1)) - \mu_l). \tag{5}$$

The evaluation of the non-algebraic function λ for each μ_l makes the calculation of an analytical maximum of Eq. 5 infeasible. Thus, a numerical maximum likelihood approach is used instead.

2.2 Pegleg

`Pegleg` [16] employs a modified Millipede likelihood where a possible muon track is divided into segments of fixed energy losses – a good approximation for low-energy events with minimum ionizing muons. The event likelihood is minimized by using three different layers: Layer 1 optimizes the six parameters $(\boldsymbol{y}, t) = (y_1, y_2, y_3, y_4 = \theta, y_5 = \phi, t)$ using the MultiNest algorithm. Layer 2 receives fixed parameters $(\boldsymbol{y}, t)$ from Layer 1 and optimizes the track length $L = N \cdot l$ by fitting the number of track segments N with a given spacing l. Finally Layer 3 internally optimizes the energy loss $E_c = E_1$ of Eq. 2 of the first segment, i.e. the initial cascade.

Note that the response matrix Λ (see Eq. 2) contains the intensity changes for every DOM time bin and photon source, where every column corresponds to a different photon source as shown in Fig. 2. The first column is the initial cascade source and every other column is another track segment. We start with a single column and calculate the energy loss E_c and likelihood in Layer 3. All energy losses of the track segments are approximated by a minimum ionizing muon. We add muon segments successively and evaluate the likelihood of the new response matrix for every added segment until a predefined amount of subsequent columns does not improve the likelihood [16]. Evaluating a column of Λ is computationally expensive due to the spline evaluations via `BSPLVB` [2]. Thus, there is a negligible overhead when calculating J segments iteratively compared to calculating J segments at once.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

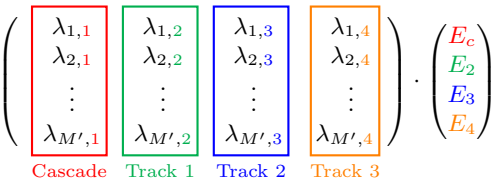

Fig. 2. The first column of the response matrix Λ corresponds to the initial cascade of photons. All other columns correspond to a track segment of an outgoing muon. Increasing the track length translates directly to adding columns and energy losses.

For Layer 3 the parameters $(\boldsymbol{y}, t, L)$ are fixed and the last remaining free parameter is the energy loss E_c of the cascade segment. Using Eq. 2 we can describe the amount of expected photons for every muon track by Eq. 6.

$$\boldsymbol{\mu}_{\text{eff}} = \boldsymbol{\rho} + \Lambda \cdot (0, E_2, E_3, \ldots, E_J)^{\mathrm{T}}. \tag{6}$$

With every energy loss $E_2, E_3, \ldots, E_J$ fixed to minimum ionizing muon, we can solve Eq. 6 directly for the effective noise term $\boldsymbol{\mu}_{\text{eff}} = (\mu_{\text{eff},1}, \mu_{\text{eff},2}, \ldots, \mu_{\text{eff},M'})^{\mathrm{T}}$. To calculate the energy loss E_c, we take the first column of Λ and apply the Newton method [19], i.e. we calculate the gradient of the likelihood from Eq. 4 with respect to E_c and follow it's second gradient until we find the maximum likelihood with the iterative scheme shown in Eq. 7

$$\frac{\mathrm{d}\tilde{\mathcal{L}}^{(k+1)}}{\mathrm{d}E_c} = \frac{\mathrm{d}\tilde{\mathcal{L}}^{(k)}}{\mathrm{d}E_c} - \frac{\mathrm{d}\tilde{\mathcal{L}}^{(k)}}{\mathrm{d}E_c} \frac{\mathrm{d}^2\tilde{\mathcal{L}}^{(k)}}{\mathrm{d}^2 E_c} \quad \text{with } k = 1, 2, \ldots \,, \tag{7}$$

In Eq. 7 $\frac{\mathrm{d}\tilde{\mathcal{L}}^{(1)}}{\mathrm{d}E_c}$ is an initial estimate, using either 20 GeV which leads to $\mathcal{O}(10)$ steps until convergence or the result of a previous iteration which reduces the number of steps to $\mathcal{O}(3)$ with a tolerance of 10^{-6} GeV. The gradient of $\mathcal{L}$ can be calculated by Eq. 8.

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}E_c} = \sum_{l=1}^{M'} \frac{N_l}{\lambda_{l,1} E_c + \mu_{\text{eff},l}} - \lambda_{l,1}. \tag{8}$$

Considering that the second term $\lambda_{l,1}$ in Eq. 8 is constant and the other part is strictly monotonically decreasing with E_c, there is at most one positive value of E_c that maximizes the likelihood.

3 Parallelizing MultiNest on a GPU

In this section we explain our GPU parallelization of Multinest. The spline approximations of the photon expectation tables do not vary throughout the minimization process. Thus, we need to copy them only once to the GPU device. The corresponding data transfer becomes negligible for the $\mathcal{O}(10000)$ calls for the likelihood in MultiNest. DOMs that are too far away or didn't see any charge

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

are excluded in each evaluation of the splines, i.e. their entries in the response matrix Λ (Eq. 2) are 0, leaving us with $\mathcal{O}(100)$ evaluations for a column of the response matrix for low energy events.

To achieve high GPU occupancy, we evaluate n_p points in parallel, where each point represents an event hypothesis. Since we sample multiple points in parallel, we do not expect a sampling efficiency of 1, but allow undersampling of likelihood regions. This is reasonable as long as the sampling space includes the global minimum or as long as the ellipsoids that enclose the current sampling space are allowed to grow large enough.

3.1 Generating Initial Live Points

MultiNest starts with a number of live points n_{live}. We use the GPU to generate n_p many points in parallel, where $n_{\text{live}} \leq n_p$. We initialize the live points by generating n_p many points and taking the best n_{live} ones as start. The points are generated inside a hypercube in parallel on the CPU using OpenMP, whereby the likelihood is calculated on the GPU. In case the seed for the parameter space boundaries may not be good enough to generate points with different likelihoods at first try, we generate a new set if the difference of the highest and lowest found likelihood is lower than 10^{-4} which occurs when all points are far away from the minimum.

3.2 Sampling the Complete Space

In this stage MultiNest samples points within the hard constraint $\mathcal{L}_{points_i} > \mathcal{L}_{\text{lowest}}$, i.e. a new point has to have a higher likelihood than the lowest likelihood of the live points. We switch to ellipsoidal sampling after a certain sampling efficiency cannot be reached within five iterations in a row. For parallel sampling we keep that condition but we do not necessarily calculate new likelihoods in every iteration. Instead, we sample n_p many points in iteration i and iterate over that list of points until one satisfies the hard constraint. The inverse of the number of samples to find such a point is the sampling efficiency in iteration i. Algorithm 1 shows the pseudocode where we sample n_p points in parallel and process those as they were sampled individually. As long as we do not find a new point inside the hard constraint $\mathcal{L}_{points_i} > \mathcal{L}_{\text{lowest}}$ we do the following: We check if any sampled points are left from previous iterations. If this is not the case, we sample new points from the hypercube in parallel using OpenMP and calculate the likelihoods using the GPU. Subsequently, we iterate over the remaining points until one of them satisfies the hard constraint. If it is the last remaining point, we note that we need more points and exit the loop. If no point satisfies the hard constraint, we repeat the loop. Note, the approach presented in [6] applies a similar sampling scheme on a compute cluster using MPI with the difference that each process evaluates a single point. In contrast, our approach takes advantage of the compute power of a GPU by dynamically changing the number of parallel sampled points as outlined in the next subsection.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

Algorithm 1 Sampling the complete space

```
do
    if not remaining_points then
        for j=1,n_p do in parallel                          ▷ Using OpenMP
            points_j ← get_random(thread_id)
        L_points ← get_llh(points)                          ▷ GPU
    for i=remain_idx, n_p do
        if L_points_i >lowlike then
            accept points_i
            if i == n_p then
                remaining_points ← False
                remain_idx ← 1
            else
                remaining_points ← True
                remain_idx getsi + 1
            exit
        if i == n_p then
            remaining_points ← False
            remain_idx ← 1
    if point accepted then exit
enddo
```

3.3 Sampling in Ellipsoids

Once the sampling efficiency consecutively falls below a threshold (e.g. five times as default), we switch to ellipsoidal sampling. The clustering and the overall algorithm remain the same, except for the sampling itself. We sample $n_{p,i}$ points at iteration i in parallel and check if any of them satisfies the hard constraint and then continue with possibly remaining points for the next iteration. Now we might have one or several distinct clusters, where each cluster is enclosed by one or more (overlapping) ellipsoids [5]. Hence we use parallel sampling within an isolated cluster. For each point to be sampled we randomly choose one ellipsoid and generate a point within this ellipsoid in parallel with OpenMP. After $n_{p,i}$ points have been generated, the GPU evaluates their likelihood. We choose a point as in Section 3.2 but if a point lies in k ellipsoids, we accept it with probability $1/k$ as does the CPU version (see Algorithm 2). During ellipsoidal sampling, the space to be explored varies and therefore we scale the number of points $n_{p,i}$ to sample in parallel by means of function $\delta(V;\alpha) \in [0,1]$ that depends on the volume fraction V of the isolated cluster we want to sample from and a user-defined value α that controls the behaviour of that function. The overhead of sampling many points in parallel is negligible, whereas it dominates the calculation time if few points are sampled in parallel. Therefore we decrease the number of points sampled in parallel faster when the ellipsoids cover only a small fraction of the parameter space and decrease slower for bigger ellipsoids. To find an appropriate number of points for a volume fraction $V \in [0,1]$ that is covered by the ellipsoids, we use a simple linear interpolation scheme with 5

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

Algorithm 2 Sampling in ellipsoids

```
do
  if not remaining_points then
    for j=1,n_p do in parallel                          ▷ Using OpenMP
      ell_j ← get_random_ellipsoid(thread_id)
      points_j ← get_random(thread_id, ell_j)
    L_points ← get_llh(points)                          ▷ GPU
  for i=remain_idx, n_p do
    if L_points_i >lowlike then
      k ← 0
      for j=1,n_ellipsoids do
        if point_in_ell(points_i, j then) k ← k + 1
      accept points_i with prob 1/k
      if point accepted then
        if i == n_p then
          remaining_points ← False
          remain_idx ← 1
        else
          remaining_points ← True
          remain_idx ← i + 1
        exit
    if i == n_p then
      remaining_points ← False
      remain_idx ← 1
  if point accepted then exit
enddo
```

knots, which depend on the user-defined value α:

$$
\begin{aligned}
f_1(V;\alpha) &= \frac{4}{3}\cdot\frac{\alpha}{1-\alpha}\cdot V \\
f_2(V;\alpha) &= \frac{\alpha}{3}+\frac{2}{3}\cdot\frac{\alpha}{1-\alpha}\cdot V \\
f_3(V;\alpha) &= \alpha\left(1-\frac{4}{3}\right)-\frac{4}{3\alpha}+\frac{8}{3}+V\left(\frac{4}{3\cdot\alpha}-\frac{4}{3}\right) \\
f_4(V;\alpha) &= \frac{5}{3}-\frac{2}{3\alpha}+V\left(\frac{2}{3\cdot\alpha}-\frac{2}{3}\right)
\end{aligned}
\tag{9}
$$

The function δ is shown in Fig. 3 for different values of α.

4 Evaluating the Response Matrix on a GPU

We want to evaluate several hypotheses in parallel on the GPU device, where the evaluation itself is not different to the CPU version, which uses SuiteSparse to

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

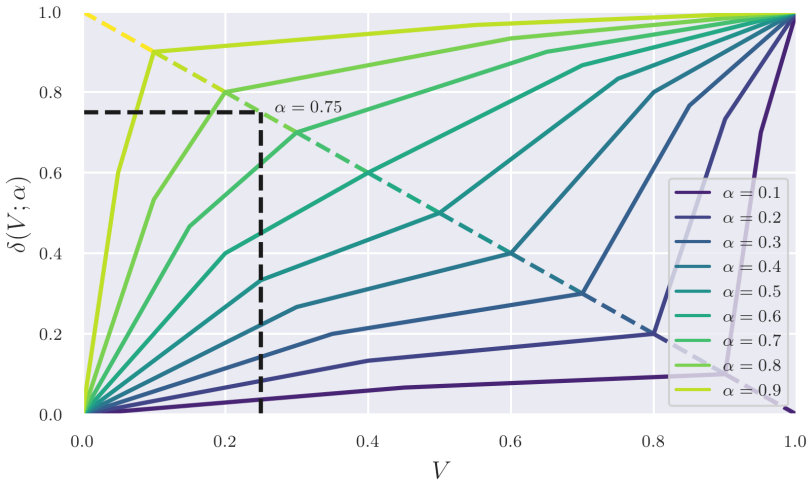

Fig. 3. A visualization of the scaling function $\delta(V;\alpha)$ to scale the number of parallel sampled points given a user-defined variable α and the volume fraction V that the isolated cluster covers from which we want to sample from. By setting α the user can define the intersection of the function with the diagonal.

handle the response matrix. However, different hypothesis might have its optimum at various amounts of track segments and hence converge at different time points. To balance the workload we thus overlap computation and communication using CUDA streams.

For n_p hypotheses and n_{streams} streams, we divide the work into chunks of size $n_{\text{chunk}} = \lfloor n_p / n_{\text{streams}} \rfloor$ where the last stream contains $n_p \mod n_{\text{streams}}$ many hypotheses. On the CPU, we create n_p many threads to process each stream in parallel. Each hypothesis and therefore each kernel call is assigned to one CUDA thread block. The CPU checks every k column and corresponding likelihood evaluations if a stream has finished its work and sets it to inactive. Hence each stream uses n_{chunk} many CUDA thread blocks from which some might end directly if their hypothesis has already converged.

The workflow for one stream is as follows: (i) Evaluate k columns of an hypothesis. (ii) Calculate the cascade energy loss for each of the additional columns and the likelihood. (iii) Synchronize the stream and copy an array of convergence checks for all hypotheses to CPU. (iv) If all converged, set the stream to inactive and start an asynchronous copy of the found likelihoods from device to host. If at least one hypothesis has not converged, launch the kernels for evaluating columns and calculating the energy loss and likelihoods for the next k columns.

Evaluating the response matrix of n_p hypotheses is done with a CUDA kernel consisting of 512 threads per CUDA thread block, one CUDA thread block per hypothesis, n_{streams} streams and 40944 Bytes of shared memory and 128 registers per thread. Given the sparse nature of a low energy neutrino event, only few

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

DOMs, typically 10 to 60, and few time bins of a DOM, about 1 to 6 bins, deliver a non-zero entry of the response matrix which is why we settle with 512 threads in favour of more register memory per thread. According to [14], at least 128 threads have to be invoked to fully utilize processing units on Volta. Adding a response column involves a check for the number of valid bins in this DOM and a check if the DOM is too far away for the given hypothesis. Once a thread processes a DOM with valid bins, it calculates mean photon counts for the DOM given the hypothesis and evaluates an amplitude splinetable. In case there are more time bins, we also need the probability quantiles for every bin using time splinetables. Each thread evaluates several splines for an entry with `BSPLVB`, where we do not have a memory layout for our splines that ensures coalesced memory access, marking the evaluation process as the most time consuming part of the overall reconstruction. Layer 3 has least impact on run time. Here we start a CUDA kernel consisting of 256 threads per CUDA thread block, one CUDA thread block per hypothesis, n_{streams} streams and 96 registers per thread and 40872 Bytes of shared memory to calculate the gradient from Eq. 12 in parallel. The size, here the amount of doubles, of shared memory is determined via

$$\text{size} = \max\left(\frac{n_{\text{threads}}}{2 \cdot w} + n_{\text{valid\_bins}}, m_s + n_{\text{valid\_bins}}\right), \tag{10}$$

where $w = 32$ is the typical size of a warp and the number of valid bins depends on the seen photons by all DOMs. m_s is the maximum amount of segments that can be used. The number of valid bins is the number of all bins in worst case, which can be a multiple of 5160 in case of the IceCube detector. Therefore the number of threads is restricted to a small number. We calculate the gradient of $\mathcal{L}$ with respect to the cascade energy E_c from Eq. 8. Since $\sum_{l=1}^{M'} \lambda_{l,1} = \lambda_{\text{sum}}$ remains constant, we only need to update the first part of the term. To calculate the amount of seen photons for the cascade, we use

$$N_1 = \sum_{l=1}^{M'} \mu_{\text{data},l} \cdot \lambda_{l,1}, \tag{11}$$

where $\mu_{\text{data},l}$ is the data vector of seen charges. The second derivative is

$$\frac{\mathrm{d}^2\mathcal{L}}{\mathrm{d}E_c^2} = \left(\sum_{l=1}^{M'} \frac{N_l \cdot \lambda_{l,1}}{(\lambda_{l,1} E_c + \mu_{\text{eff},l})^2}\right) - \lambda_{\text{sum}}. \tag{12}$$

We leave $\mu_{\text{eff},l}$ in shared memory and every row is being processed by another thread, such that all memory reads are done in a coalesced manner. The partial results of each thread is distributed with `sum_and_broadcast` in every iteration that utilizes warp shuffles and shared memory. Calculating the likelihood after the Newton method converged is done by evaluating Eq. 5.

5 Performance Evaluation

We compare the run time and the accuracy of our GPU version with the CPU version of `Pegleg`. We have measured run times depending on the energy of the

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

neutrino events for both versions and discuss the achieved accuracy, where we use a fixed seed for generating the initial live points. The used parameters for the GPU version are shown in Table 1. The CPU version (compiled using GCC V6.3.0) runs on an Intel Xeon E5-2680@2.70 GHz with 16 threads and with 8 GB DDR3 RAM. The GPU version (compiled using CUDA V9.2.148) runs on an Intel Core i5-3450@3.10 GHz with 8 GB DDR3 RAM and a NVidia Titan V with 12 GB HBM2 VRAM. We use the GRECO (GeV Reconstructed Events with Containment for Oscillations) sample [16] which is a simulated set from different Monte Carlo generators. It provides a low energy sample with high statistics at $E_\nu \sim 5$ GeV. These low energy events deposit photons in $\mathcal{O}(10)$ DOMs, where most DOMs feature a single time bin due to the few number of photons (~ 1) that are detected per DOM. This leads to a low signal to noise ratio and hence a poor resolution for the zenith angle, the neutrino energy and track against cascade separation.

Table 1. Settings of the GPU MultiNest algorithm that delivers the best run time while reconstructing all tested events with a low error.

Parameter	Used Value
number of streams	4
minimum parallel evaluations per calls	96
parallel evaluations per calls	480
α	0.3
n_{nlive}	30
efficiency	2.0
maximum modes	10
tolerance	1.1
importance sampling	False

5.1 Speedup

Fig. 4 shows a heatmap of run times over energy ranges from 1.00 GeV to 980.71 GeV for 5262 events of different flavors reconstructed by the GPU and CPU. The average speedup is 14.09 with a minimum at 1.29 and a maximum at 248.40 and an interquantile range of 10.47. The median run times for the GPU and CPU versions slightly increase with the energy of the event. Calculating Pearson's correlation coefficient, a value of 0.76 (0.64) for the GPU (CPU) run time and the number of activated DOMs and 0.46 (0.37) for the run times and the number of time bins for each DOM. The GPU version scales best with the track length (correlation value of 0.07 compared to 0.33 of the CPU version) thanks to the usage of CUDA streams. The speedup correlates with the number of DOMs with a value of 0.39, with the number of time bins with a value of 0.06 and with the track length with a value of 0.73. The wide variety of speedups can be explained as follows:

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

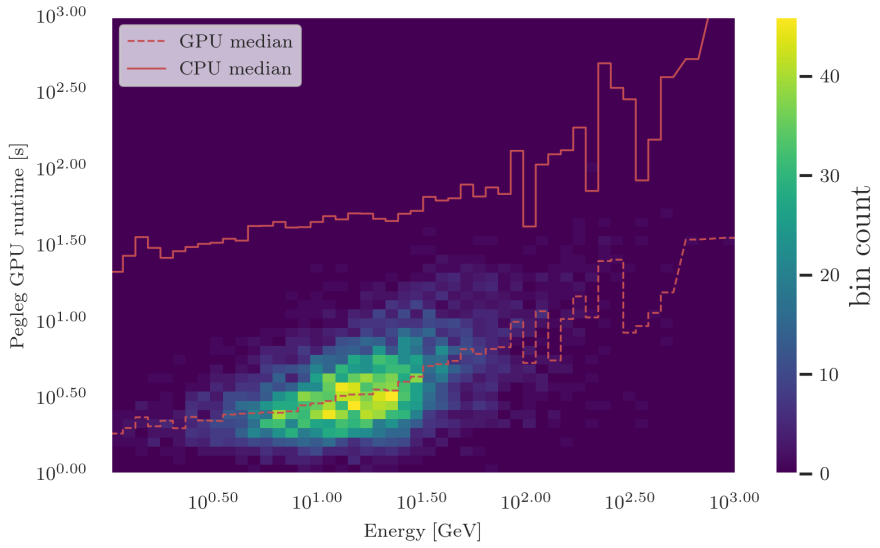

Fig. 4. 5262 events reconstructed with the GPU and CPU. The energy of an event correlates directly to the signal-to-noise ratio and the number of activated DOMs.

- Either one of the versions stops prematurely in a local minimum.
- The initial live points are favourable for one of the versions, such that it converges after fewer iterations.
- The event can be reconstructed within few iterations for both versions, such that the overhead of the GPU version of `Pegleg` dominates the runtime.

5.2 Accuracy

High speedups of the GPU version could mean a reconstruction prematurely terminated and did not reach the same accuracy as the CPU version. In order to investigate this in more detail, we compare between the calculated reconstruction and the provided ground truth on an event-by-event basis in Table 2 for 5262 events. With the exception of the time parameter, there is no systematic difference visible, i.e. all differences are centered around or close to zero. Overall, the CPU version yields slightly better reconstructions and has smaller interquartile ranges than our GPU version with the chosen setup that yields speedups of up to 250. This indicates for events that take especially long on CPU, that the GPU version stops early. For those events the CPU version usually yields a hypothesis with at least one parameter outside the interquantile range, rendering the result barely useful for further processing. Additional parameter tuning of the GPU version could enhance the accuracy especially for events in the tails which could lead to speedups closer to the mean.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

Parameter	CPU median	GPU median	CPU IQR	GPU IQR
Δt	7.87	−14.57	50.40	54.35
Δy_1	−0.31	0.29	14.81	16.33
Δy_2	0.34	0.64	16.28	19.58
Δy_3	−0.13	0.92	7.88	8.47
$\Delta\theta$	−0.03	−0.11	0.64	0.76
$\Delta\phi$	0.01	−0.04	1.65	2.17

Table 2. A detailed comparison of the achieved accuracy by comparing the reconstructed parameters to the truth. The CPU version is most of the time closer to the truth and has shorter tails with the GPU version not far behind. The interquantile range is similar, such that one can expect similar behavior of the GPU implementation.

6 Conclusion

The computational analysis of data produced by low energy neutrino events is a major computational challenge for neutrino physicists at IceCube. In this paper we have shown how GPUs can be efficiently used to accelerate this task by an order-of-magnitude while achieving comparable accuracy. Our GPU version scales best with larger track lengths and achieves further speedup with more activated DOMs, whereas the number of time bins has no effect on the speedup. An interesting direction for further acceleration could be achieved by replacing the B-spline surface with neural networks that describe the light yield λ for a given light source and detector position. This can potentially achieve an even higher efficiency on GPUs compared to the B-spline approach.

Acknowledgements: Parts of this research were conducted using the supercomputer MOGON and auxiliary services offered by Johannes Gutenberg University Mainz (hpc.uni-mainz.de) which is a member of the AHRP and the Gauss Alliance e.V. This paper is supported by the NVidia GPU Grant Program, which donated a Titan V for this research.

References

1. Aartsen, M.G., Abbasi, R., Abdou, Y., Ackermann, M., Adams, J., Aguilar, J.A., Ahlers, M., Altmann, D., Auffenberg, J., Bai, X., et al.: Measurement of atmospheric neutrino oscillations with icecube. Physical Review Letters **111**(8) (Aug 2013). https://doi.org/10.1103/physrevlett.111.081801
2. de Boor, C.: On "best" interpolation. Journal of Approximation Theory **16**, 28–42 (1976). https://doi.org/10.1016/0021-9045(76)90093-9
3. Calland, R.G., Kaboth, A.C., Payne, D.: Accelerated event-by-event neutrino oscillation reweighting with matter effects on a GPU. Journal of Instrumentation **9**(04), P04016–P04016 (apr 2014). https://doi.org/10.1088/1748-0221/9/04/p04016
4. Cook, R.J., Lawless, J.: The Statistical Analysis of Recurrent Events. Statistics for Biology and Health, Springer-Verlag (2007). https://doi.org/10.1007/978-0-387-69810-6

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9

5. Feroz, F., Hobson, M.P.: Multimodal nested sampling: an efficient and robust alternative to markov chain monte carlo methods for astronomical data analyses. Monthly Notices of the Royal Astronomical Society **384**(2), 449–463 (2008). https://doi.org/10.1111/j.1365-2966.2007.12353.x
6. Feroz, F., Hobson, M.P., Bridges, M.: MultiNest: an efficient and robust bayesian inference tool for cosmology and particle physics. Monthly Notices of the Royal Astronomical Society **398**(4), 1601–1614 (2009). https://doi.org/10.1111/j.1365-2966.2009.14548.x
7. Feroz, F., Hobson, M.P., Cameron, E., Pettitt, A.N.: Importance nested sampling and the MultiNest algorithm. arXiv:1306.2144 [astro-ph, physics:physics, stat] (2013-06-10). https://doi.org/10.21105/astro.1306.2144
8. Giunti, C., Kim, C.W.: Fundamentals of Neutrino Physics and Astrophysics. Oxford University Press (2007). https://doi.org/10.1093/acprof:oso/9780198508717.001.0001
9. IceCube Collaboration: Energy reconstruction methods in the IceCube neutrino telescope. Journal of Instrumentation **9**(03), P03009P03009 (Mar 2014). https://doi.org/10.1088/1748-0221/9/03/p03009
10. IceCube Collaboration: The IceCube neutrino observatory: instrumentation and online systems. Journal of Instrumentation **12**(3), P03012–P03012 (2017-03). https://doi.org/10.1088/1748-0221/12/03/P03012
11. IceCube Collaboration: Computational techniques for the analysis of small signals in high-statistics neutrino oscillation experiments (2018)
12. IceCube Collaboration: Measurement of atmospheric tau neutrino appearance with icecube deepcore. Phys. Rev. D **99**, 032007 (Feb 2019). https://doi.org/10.1103/PhysRevD.99.032007
13. IceCube Collaboration: Development of an analysis to probe the neutrino mass ordering with atmospheric neutrinos using three years of IceCube DeepCore data. The European Physical Journal C **80**(1), 9 (2020). https://doi.org/10.1140/epjc/s10052-019-7555-0
14. Jia, Z., Maggioni, M., Staiger, B., Scarpazza, D.P.: Dissecting the NVIDIA volta GPU architecture via microbenchmarking (2018), http://arxiv.org/abs/1804.06826
15. Kallenborn, F., Hundt, C., Böser, S., Schmidt, B.: Massively parallel computation of atmospheric neutrino oscillations on cuda-enabled accelerators. Computer Physics Communications **234**, 235–244 (2019). https://doi.org/10.1016/j.cpc.2018.07.022
16. Leuermann, M.: Testing the Neutrino Mass Ordering with IceCube DeepCore. Phd thesis, RWTH Aachen University (2018). https://doi.org/10.18154/RWTH-2018-231554
17. van Santen, J., Whitehorn, N.: Photospline: smooth, semi-analytic interpolation for photonics tables. Tech. rep., University of Wisconsin-Madison (2011-05-19)
18. Verpoest, S.: Search for particles with fractional charges in IceCube based on anomalous energy loss. Phd thesis, Ghent University (2018-06), https://lib.ugent.be/fulltxt/RUG01/002/479/620/RUG01-002479620_2018_0001_AC.pdf
19. Ypma, T.: Historical development of the newtonraphson method. SIAM Review **37**(4), 531–551 (1995-12-01). https://doi.org/10.1137/1037125

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50371-0_9