Accelerated Neutrino Oscillation Probability Calculations and Reweighting on GPUs

Richard Calland

University of Liverpool

GPU Computing in High Energy Physics

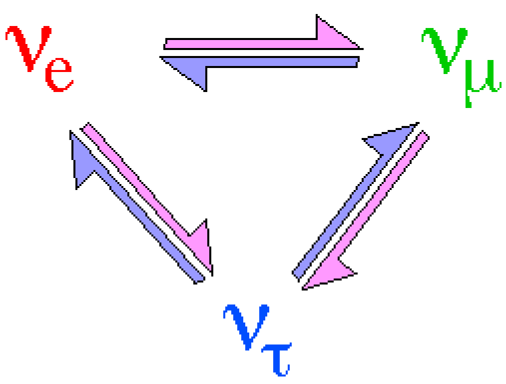
University of Pisa, 11th September 2014

Introduction

- Neutrino Oscillations
- The T2K Experiment
- Oscillation Analysis Strategy
  - Benefits from GPUs
- Conclusions

Neutrinos

- Neutrinos are the lightest of all known particles
- Thought to be massless, until neutrino oscillations were discovered as a solution to the solar neutrino problem
  - Neutrinos are a mix of mass (ν_1,ν_2,ν_3) and flavour (ν_e,ν_μ,ν_τ) eigenstates
  - Neutrino created as one flavour has a non-zero probability of being observed later as a different flavour
  - Mass and flavour states related via the PMNS mixing matrix (analogous to CKM matrix in quarks)



Oscillation Model

$$P(\nu_\alpha \to \nu_\beta) = \delta_{\alpha\beta} - 4\sum_{i>j} \mathrm{Re}(U^*_{\alpha i} U_{\beta i} U_{\alpha j} U^*_{\beta j}) \sin^2\left(\tfrac{1}{4}\Delta m^2_{ij} \frac{L}{E}\right)$$

$$+ 2\sum_{i>j} \mathrm{Im}(U^*_{\alpha i} U_{\beta i} U_{\alpha j} U^*_{\beta j}) \sin\left(\tfrac{1}{2}\Delta m^2_{ij} \frac{L}{E}\right)$$

Probability of neutrino ν_α oscillating to type ν_β

L = distance travelled by neutrino

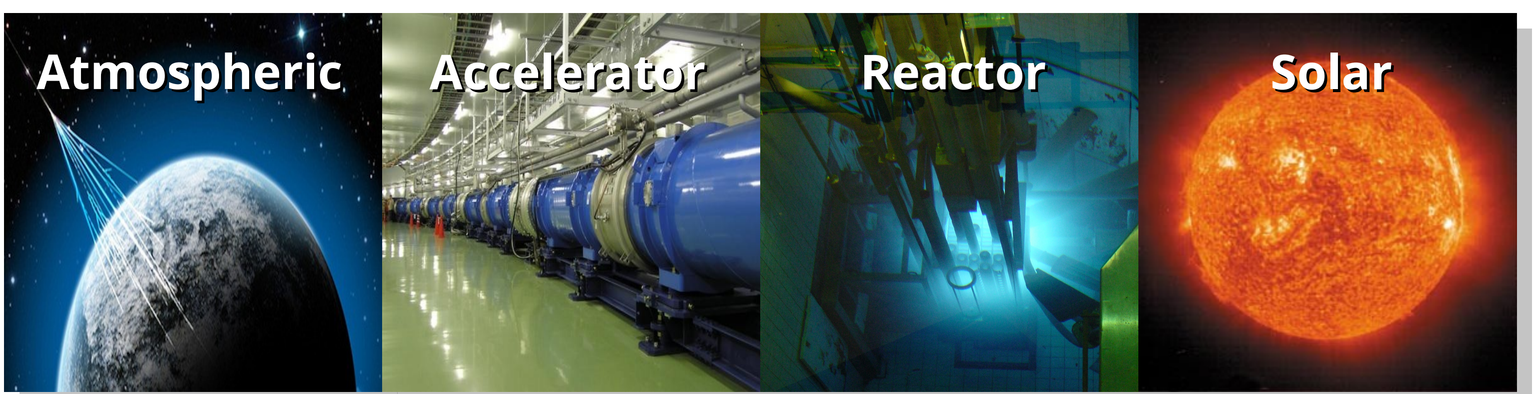
E = energy of neutrino

We want to measure these!

$$U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{pmatrix} \begin{pmatrix} c_{13} & 0 & s_{13}e^{-i\delta} \\ 0 & 1 & 0 \\ -s_{13}e^{i\delta} & 0 & c_{13} \end{pmatrix} \begin{pmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$c_{ij} = \cos\theta_{ij}$

$s_{ij} = \sin\theta_{ij}$

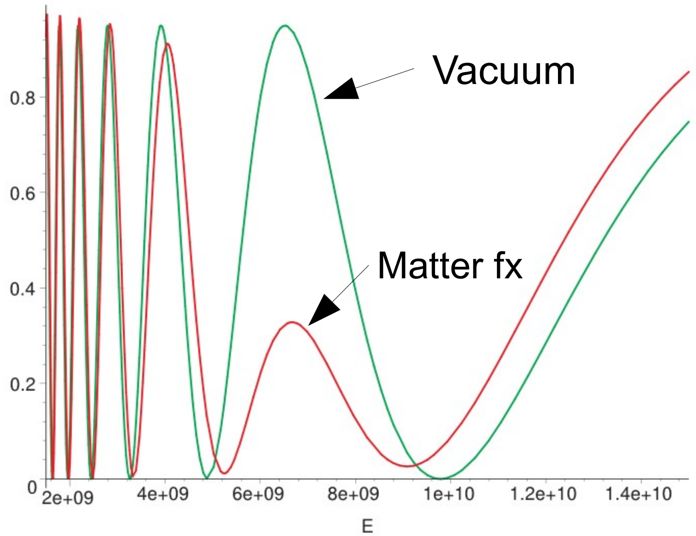


Neutrino Oscillations in Matter

- The interactions with ambient electrons in matter cause the neutrinos to feel an extra potential
- These so-called “matter effects” must be modelled
- Addition of an extra potential term to calculations creates a different oscillation probability compared to vacuum

$$i\frac{d}{dt}\begin{pmatrix}\nu_e\\ \nu_\mu\end{pmatrix} = \begin{pmatrix}-\frac{\Delta m^2}{4E}\cos 2\theta + V & \frac{\Delta m^2}{4E}\sin 2\theta\\ \frac{\Delta m^2}{4E}\sin 2\theta & \frac{\Delta m^2}{4E}\cos 2\theta\end{pmatrix}\begin{pmatrix}\nu_e\\ \nu_\mu\end{pmatrix}$$

2 neutrino approximation



- Need to re-diagonalize the matrix to find the neutrino mass eigenstates in matter

Analytical Solution

Arbitrary state vector in flavour space

$$|\psi(t)\rangle = \sum_i \psi_i(t)\,|\nu_i\rangle$$

Initial state transition amplitude

$$A(\nu_\alpha \to \nu_\beta) = \sum_i U^\dagger_{i\beta}\psi_i(t)$$

Time evolution of the state

$$i\,d\psi_j(t)/dt = m_j^2/(2E)\psi_j(t) - \sum_k \sqrt{2}\,G N_e U_{ej} U^\dagger_{ke}\psi_k(t)$$

$$\equiv H_{kj}\psi_k(t)\,,$$

Has 3 independent solutions for row vector ψ_j, assemble into matrix X

$$\alpha = 2\sqrt{2}EGN_e + \delta m^2{}_{12} + \delta m^2{}_{13}\,,$$

$$\beta = \delta m^2{}_{12}\delta m^2{}_{13} + 2\sqrt{2}EGN_e[\delta m^2{}_{12}(1-|U_{e2}|^2) + \delta m^2{}_{13}(1-|U_{e3}|^2)]$$

$$\gamma = 2\sqrt{2}EGN_e\delta m^2{}_{12}\delta m^2{}_{13}|U_{e1}|^2.$$

Solutions are the roots of the arc-cos

$$M_i^2 = -\tfrac{2}{3}(\alpha^2 - 3\beta)^{1/2}\cos\left[\tfrac{1}{3}\arccos\left(\frac{2\alpha^3 - 9\alpha\beta + 27\gamma}{2(\alpha^2-3\beta)^{3/2}}\right)\right]$$

$$X = \sum_k \left[\prod_{j\neq k}\frac{(2EH - M_j^2 1)}{\delta M_{kj}^2}\right]\exp\left(-i\frac{M_k^2 L}{2E}\right)$$

$$A(\nu_\alpha \to \nu_\beta) = \sum_{ij} U_{\alpha i} X_{ij} U^\dagger_{j\beta}$$

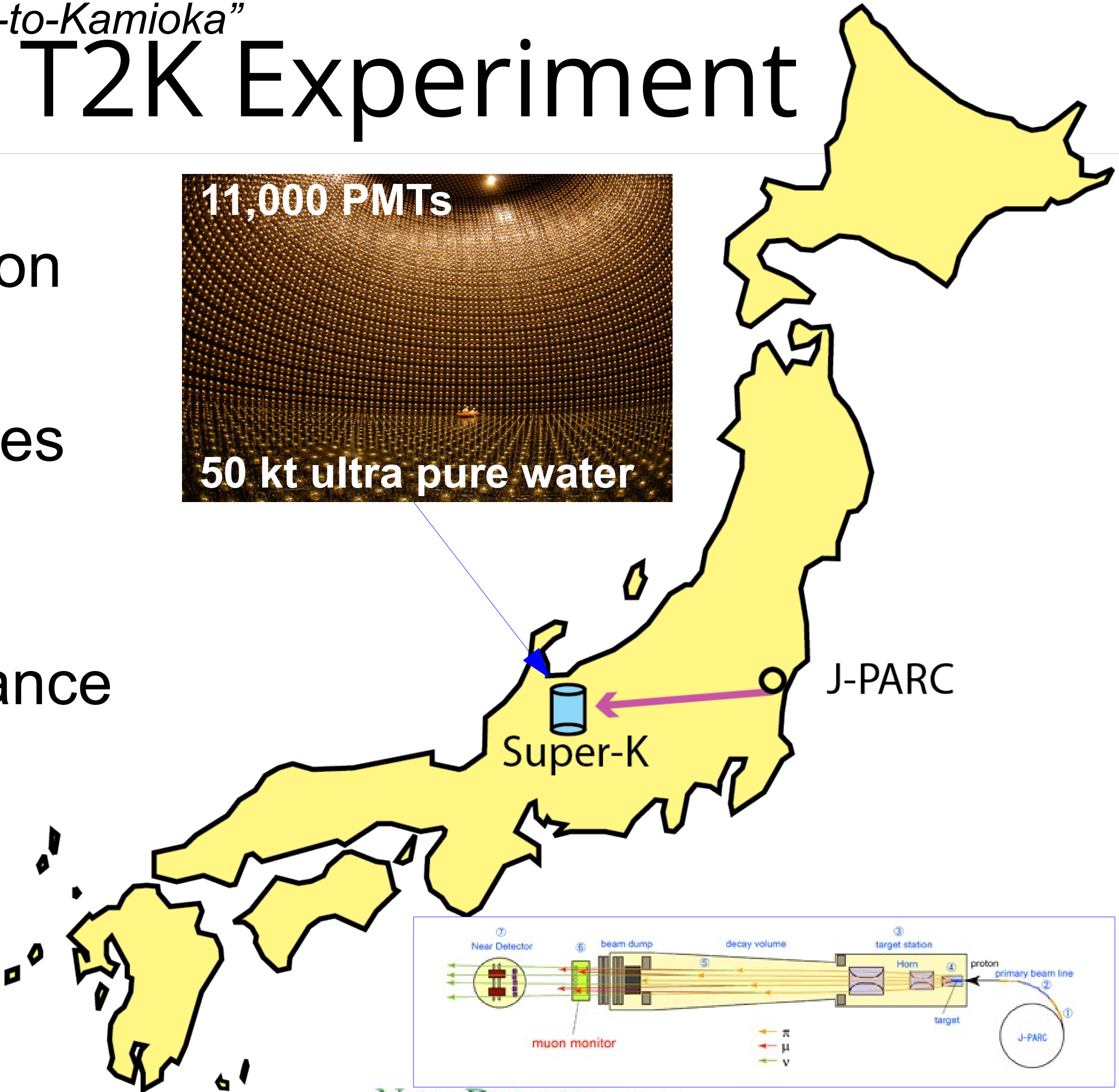
Probability = |A|²

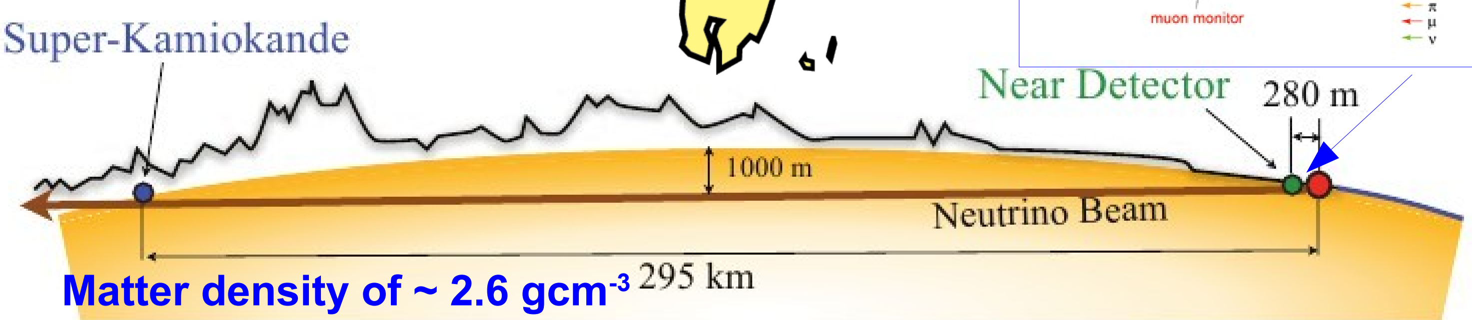
Barger et al. Phys. Rev. D22(1980) 2718

"Tokai-to-Kamioka"

The T2K Experiment

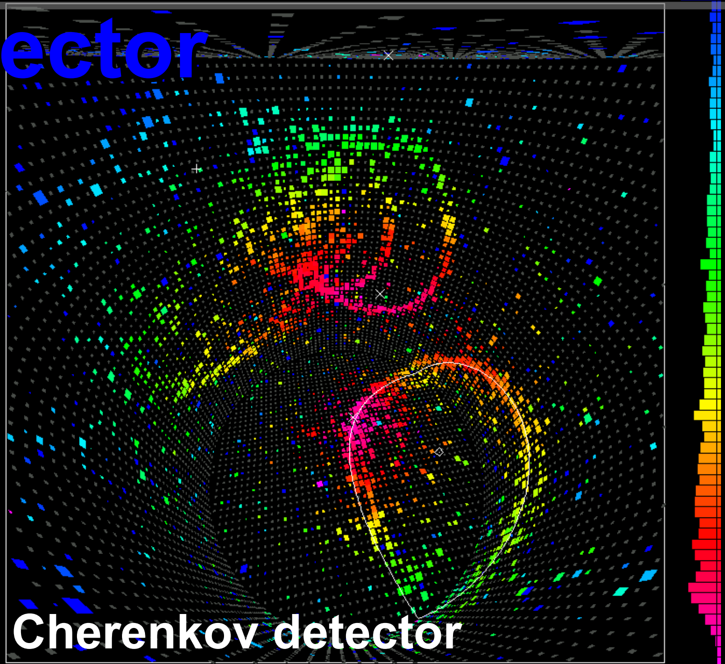
- Study intense beam of accelerator-produced muon neutrinos
- Near detector characterises beam
- Far detector observes oscillation effect at a distance of 295 km from source



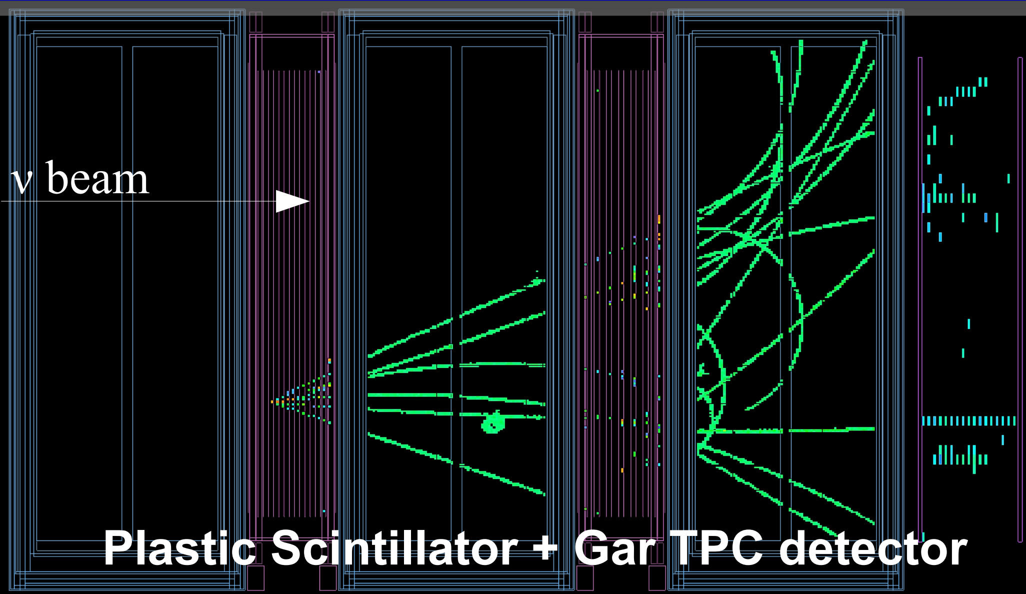


Detector Events

Super-Kamiokande Far Detector

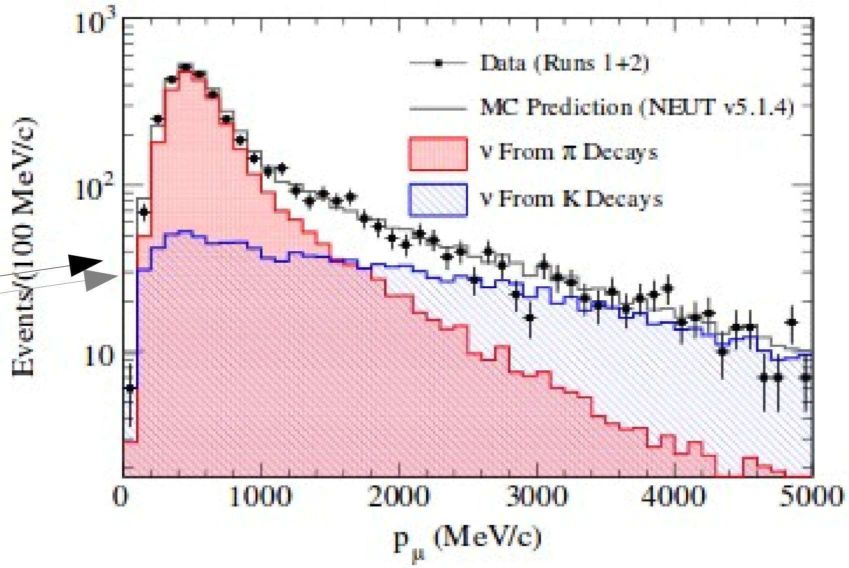


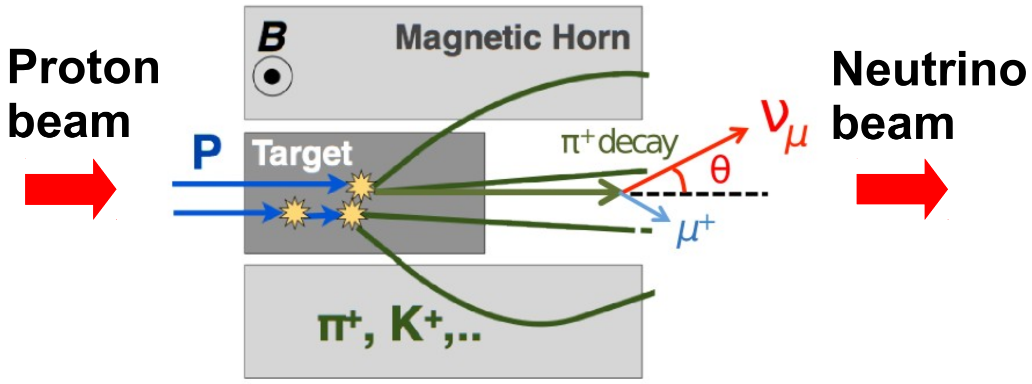
ND280 Near Detector

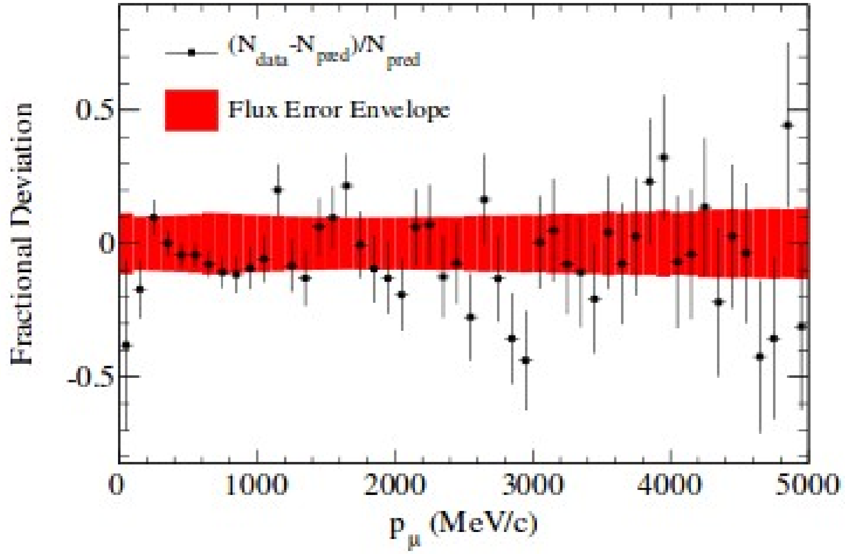


Understanding the beam

- Beam is simulated from accelerator to the neutrino target, and finally the interactions inside the detectors
- Events measured in the near detector help tune the flux prediction
- We know what event rates to expect at the far detector for a null oscillation hypothesis







Analysis Strategy

- T2K looks for a deficit of ν_μ and an appearance of ν_e neutrinos at the far detector
- Constrain beam flux and cross section systematics using the near detector
- Construct a binned likelihood using detector PDFs made from Monte Carlo
- Use a **Markov Chain Monte Carlo** to sample the high-dimensional posterior probability

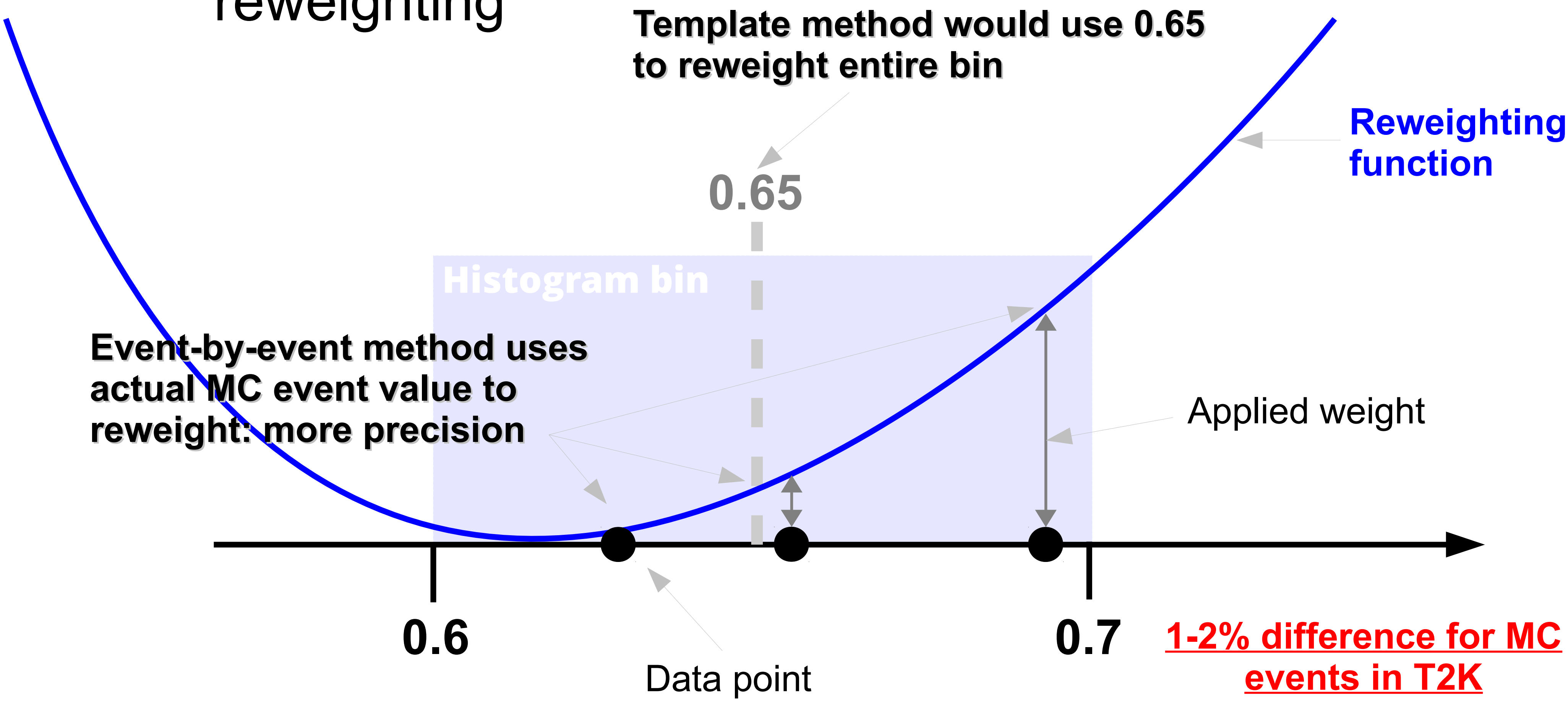
This is where we can benefit from GPUs; the focus of this talk!

Constructing a PDF

- To calculate a binned likelihood from Monte Carlo, there are generally two methods:
  - Fill a histogram with MC, reweight each bin according to your model — ***Template method***
  - Reweight each MC event according to your model, fill a histogram — ***Event-by-event method***
- Reweighting is a common method to model the response of the PDF to changes in your model
- The point is that you can either create histograms ("templates"), and throw away your MC, or
- Keep all your MC in memory, and make a histogram at every iteration of your fitting algorithm
- Obviously, the second method is far more computationally demanding...

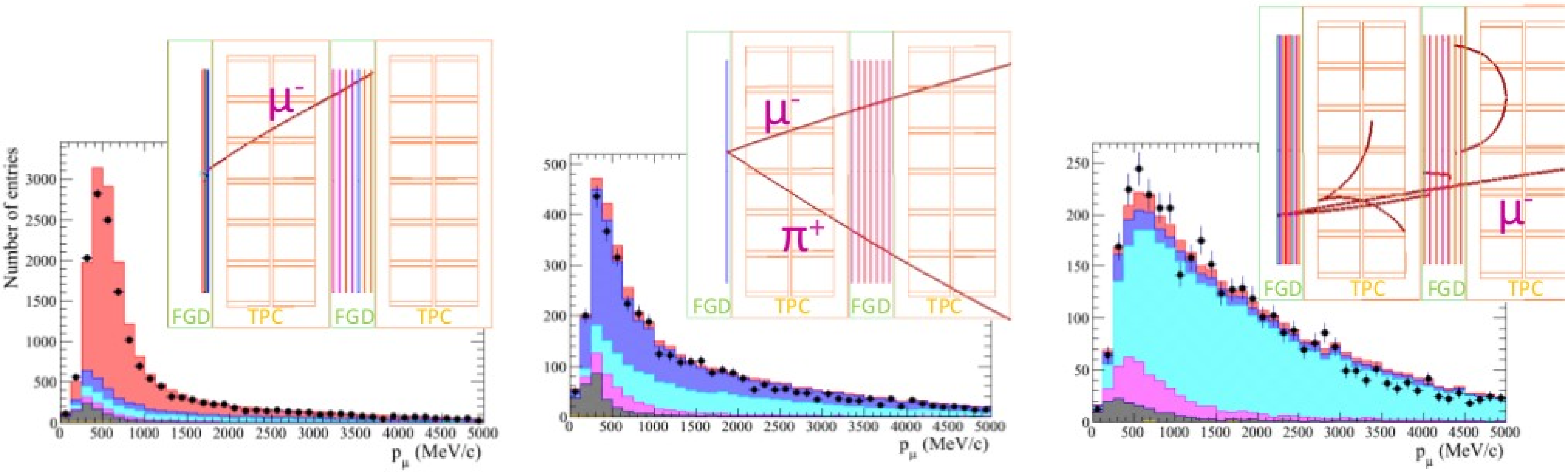
Why Event-by-event?

- There are several advantages to using the event-by-event method:
  - Retain more shape information within the bin when reweighting



Why Event-by-event?

- Better treatment of systematic uncertainties
  - Model event migration between samples (e.g. PID / reconstruction efficiency parameters)
  - Model event migration between bins (e.g. energy scale)
- Cannot easily treat migrations using templates



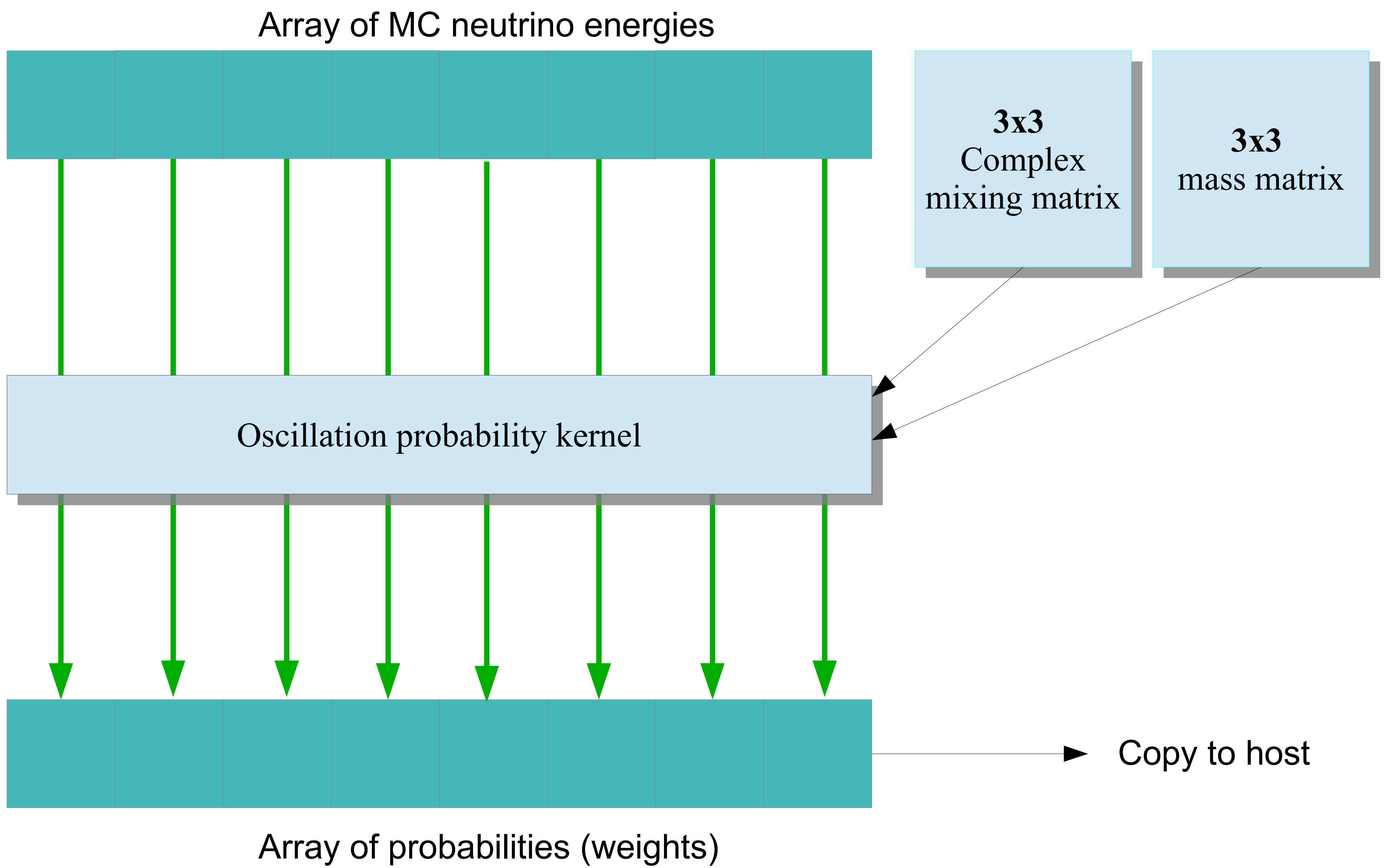
Feasibility on CPU

- In T2K, have **~1M** MC events for far detector, and **~500,000** for near detector
- This means moving from ~100 calculations per fit iteration (i.e.100 bins) to **~ 1 million calculations per iteration** of the sampler
  - Each fit itself requires many millions interations
- Computationally prohibitive: Can GPUs help?
- Offload two most CPU intensive reweighting tasks:
  - Calculation of oscillation probability
  - Calculation of response functions for cross section modelling

Oscillation Probability

- Need to model neutrino oscillation in our PDF
  - We saw earlier there is a cumbersome analytical solution
- Luckily, someone already wrote a library to do this
  - http://www.phy.duke.edu/~raw22/public/Prob3++/
  - Produces a weight for a given neutrino energy
- Unfortunately, rather slow ~ 2-3 seconds to reweight all 1,000,000 MC entries
- Need to do this **~20 million** times for one analysis
  - 2,000,000,000,000 calculations needed! (2×10^{12})
  - **~2 years on a single machine with no GPU!**
- Ported some functions to GPU using CUDA 5
  - Propagation through constant matter density
  - Acceptable for long baseline experiments like T2K

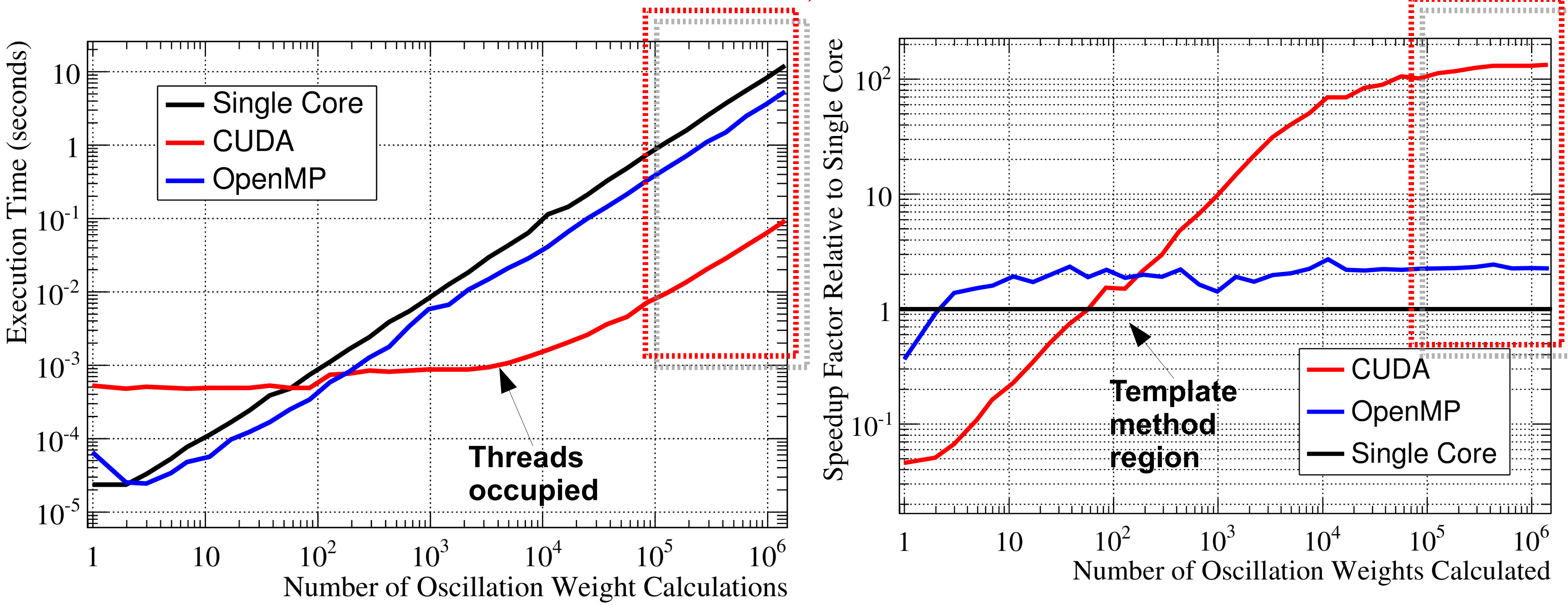
CUDA Implementation



Benchmarks

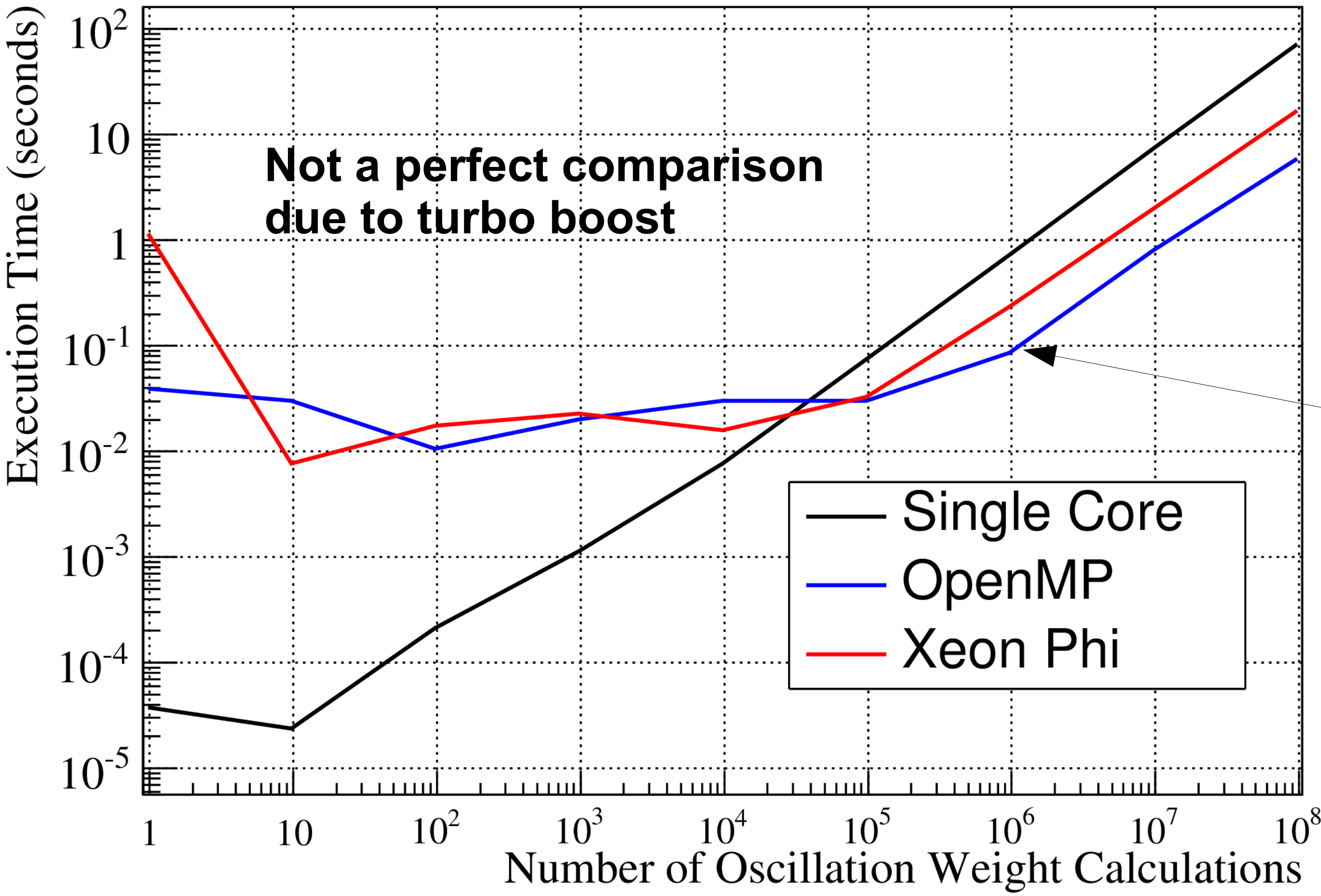
Intel Xeon "Westmere" CPU E5640 @ 2.67GHz
NVIDIA "Fermi" M2070 GPU - 448 CUDA cores

Region for event-by-event method



- Standalone benchmark: measure execution time as a function of number of oscillation calculations in **double precision**
- For ~100,000-1M concurrent calculations, CUDA approaches **2 orders of magnitude speed-up**
- Multi-core implementation using OpenMP (limited to 4 physical cores) is also compared

Xeon Phi Comparison

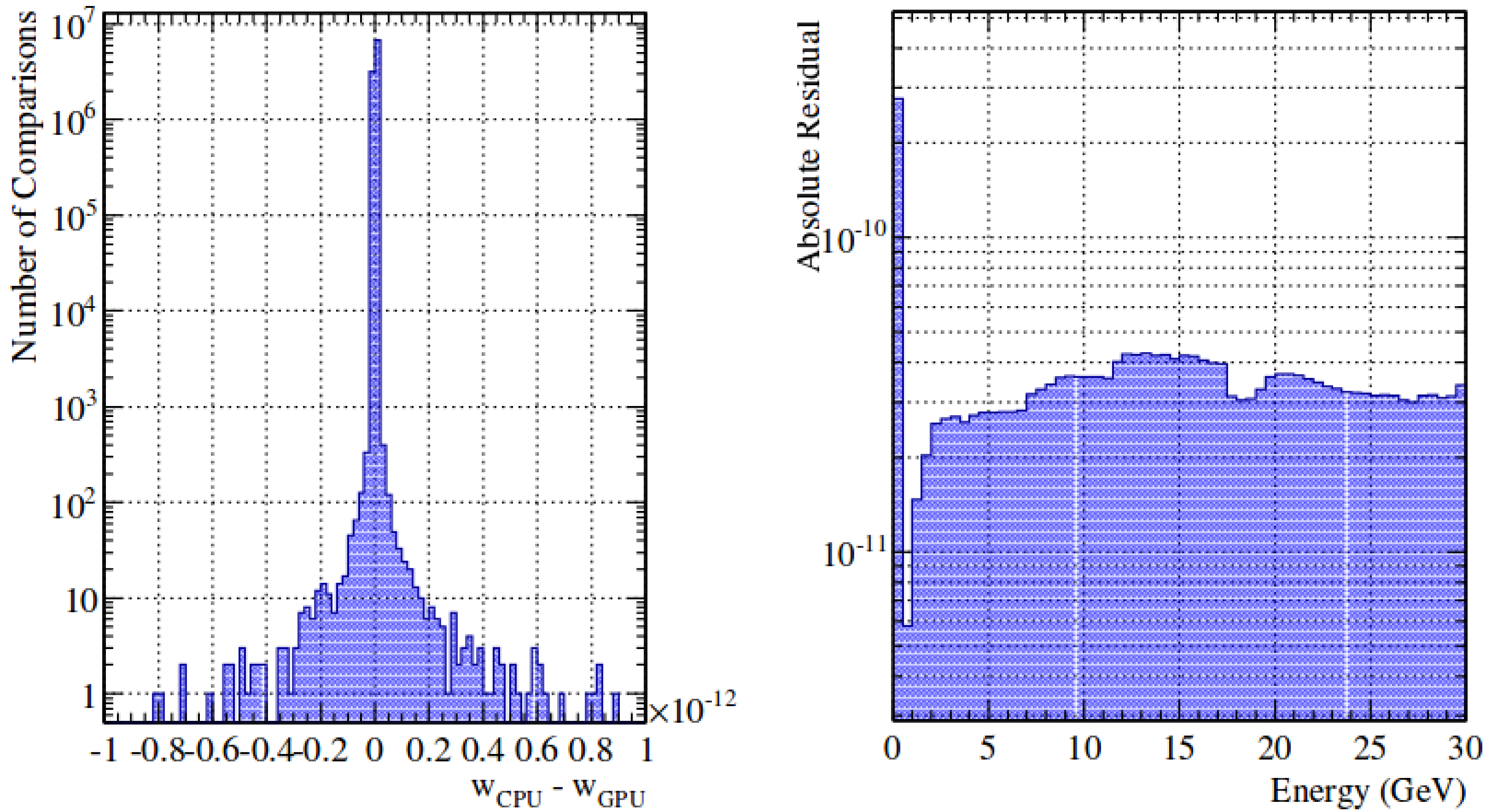


Competitive with M2070 GPU

Intel Xeon "Ivybridge" CPU E5-2680v2 @ 2.80GHz
Xeon Phi 7120P co-processor

Dual socket machine, 40 logical cores with hyperthreading.

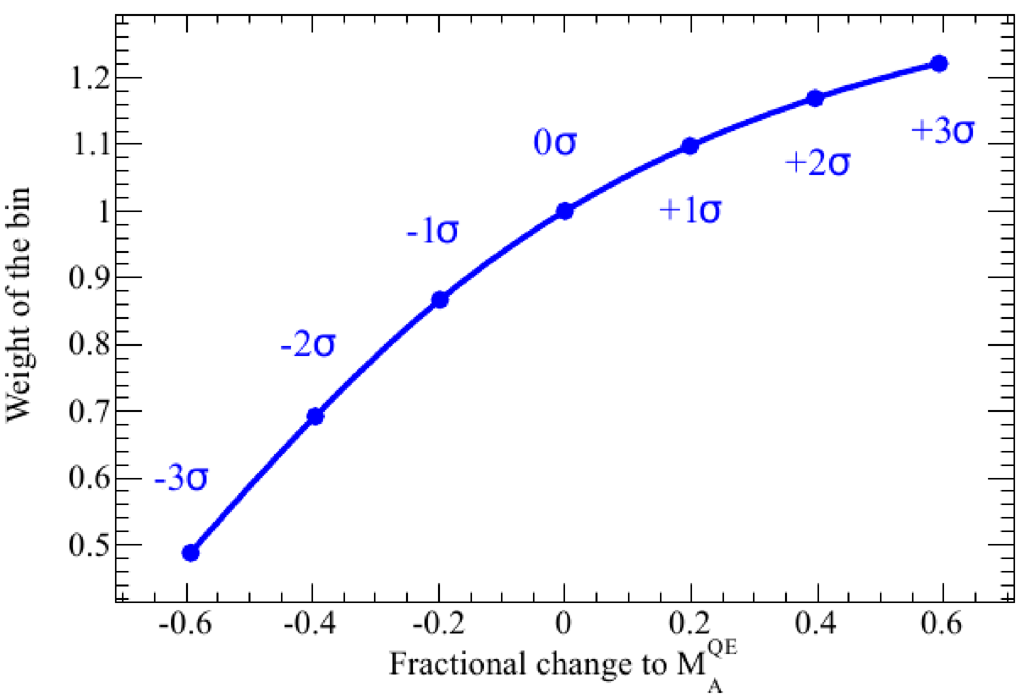
Validation

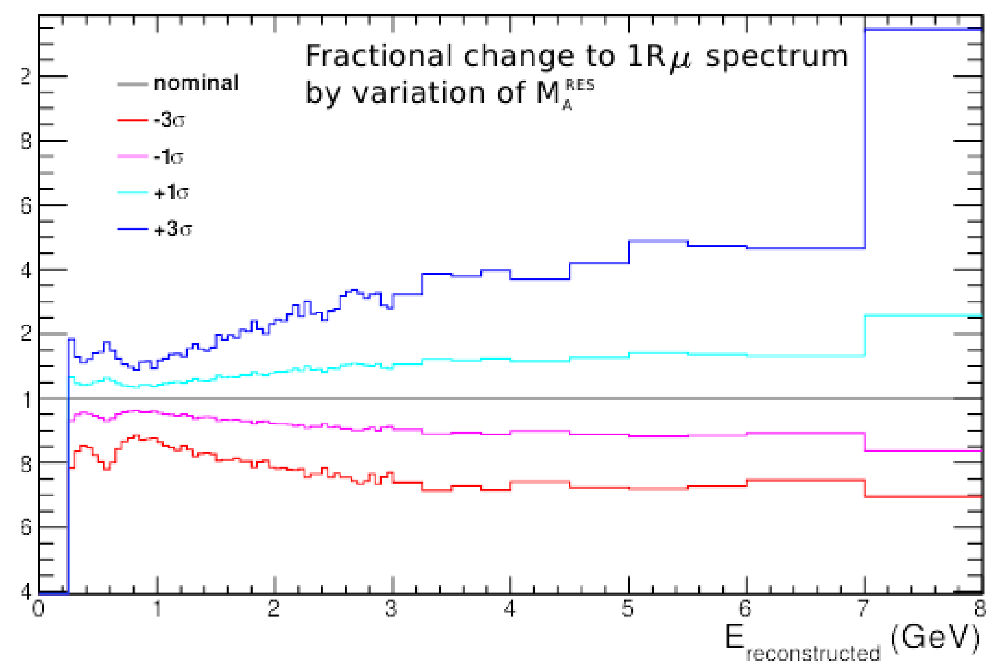


- 10 million random comparisons between CPU and GPU calculations
- Agreement to 10^{-12} precision, **more than good enough for this application**
- Difference attributed to extended ALU of CPU and different hardware implementations of non-associative calculations

Repsonse Functions

- Next biggest bottleneck is the modelling of cross section response
  - Cross section model parameters have non-linear response
- We use cubic splines to encode how the PDF responds to changes in cross section parameters
  - Rerunning the MC generator for each sample is no possible





Reformatting

- Our splines are formatted as `TSpline3` cubic spline objects
  - http://root.cern.ch
- Lots of bloat: ~4M instantiations of a C++ class
- Try to reformat to perform better on GPU
  - A more sensible data access pattern
- Instead of a class, format as an array and use a kernel function to evaluate:
  - Locate polynominal inside spline
  - Evaluate the polynominal at `x`
  - Save the response of spline as a weight

T2K Convert TSpline3 Into an array

- This is the TSpline3 as an array:

Array index

[0] [1][2][3][4][5][6][7] [8][9][10][11] ... [32][33][34][35]

Array content

[N] [x1 -- x7][{Y1,B1,C1,D1} -- {Y7,B7,C7,D7}]

Number of knots in Spline (current N=7)

x position of each knot

Four poly* parameters for poly #1

Four poly* parameters for poly #7

Minimal Eval(x) function:

1) sort input x in range of [x1 -- x7], return poly #n
2) get params for poly #n
3) evaluate polynomial $f(x) = ax^3 + bx^2 + cx + y$

- Convert TSpline3 objects into a monolithic array

Spline#1	Spline#2		Spline#999,999	Spline#1,000,000
[N][X][P][P][P][P][P][P][P]	[N][X][P][P][P][P][P][P][P]	....	[N][X][P][P][P][P][P][P][P]	[N][X][P][P][P][P][P][P][P]

N= # of knots in particular spline	[X] = set of x-params for spline	[P]...[P] = Spline i's Polynomials

GPU Implementation

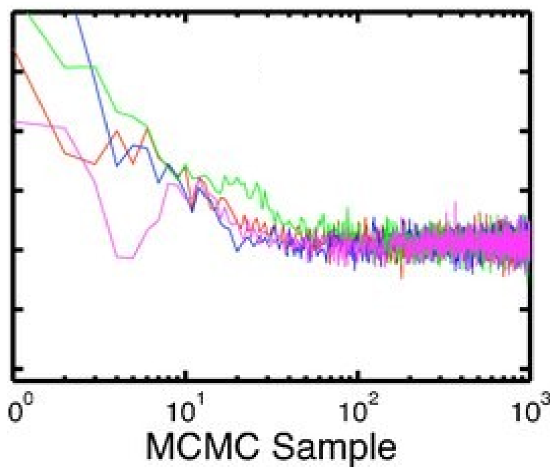
- This monolithic array is now smaller in memory and slightly faster on CPU
- Copy large (~1.2 Gb) array onto GPU RAM at initialization, keep it there (read-only resource)
- Every iteration, evaluate all splines with a CUDA kernel and push the weight from each spline back onto CPU RAM
- GPU implementation yields **~20 speed-up** over `TSpline3` version for the evaluation of 4,000,000 splines
  - Monolith array is **~3-5x** faster on GPU
- Many ways to improve this basic implementation
  - Use of shared memory
  - Asynchronous data transfer

Analysis Deployment

- As previously mentioned, analysis uses a **Markov Chain Monte Carlo** to sample the high dimensional space of the model with respect to the data
- MCMC is very scalable to high numbers of parameters
  - 5 Detector samples demand ~200 parameter fit
- This equates to needing ~ 50 million MCMC samples
- With each step taking ~5 seconds if executed on CPU, this means **2800 CPU days**
- **In GPU mode, this is ~140 GPU days**

Cluster Deployment

- Multiple MCMC runs can be executed and combined
  - Each chain produces independent samples
- This lends itself perfectly to distributing the analysis load across a GPU HPC cluster
- Run multiple chains using the same model and data, but different starting configuration

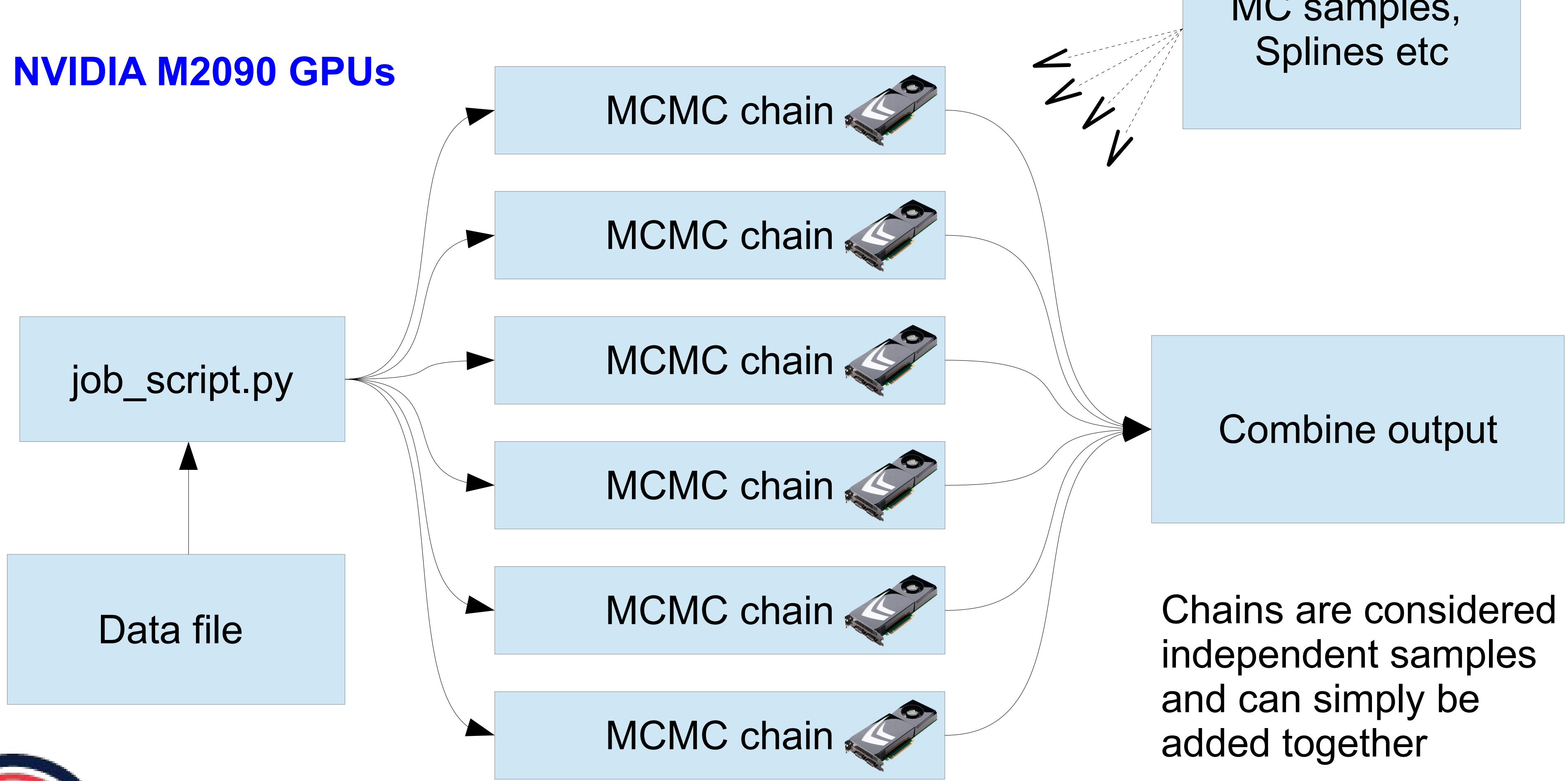


All chains converge on the same stationary distribution

GPU HPC Cluster

We would like to acknoledge the SES (Science Engineering South) Centre for Innovation service (CFI) Emerald HPC cluster for their support

370 NVIDIA M2090 GPUs

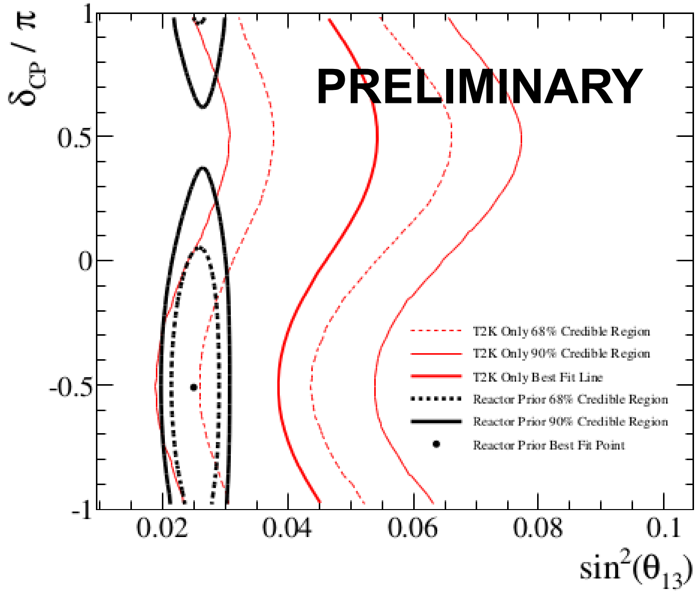


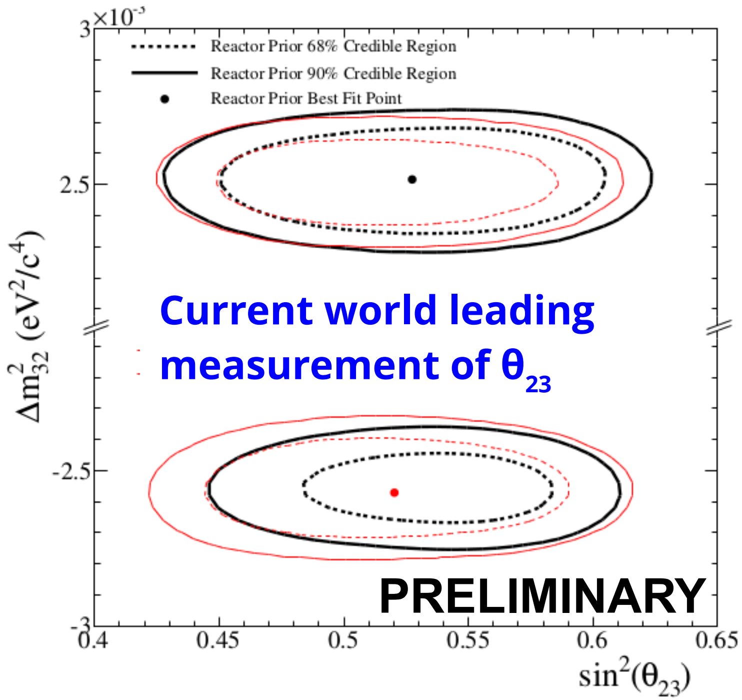
Chains are considered independent samples and can simply be added together

CENTRE FOR INNOVATION

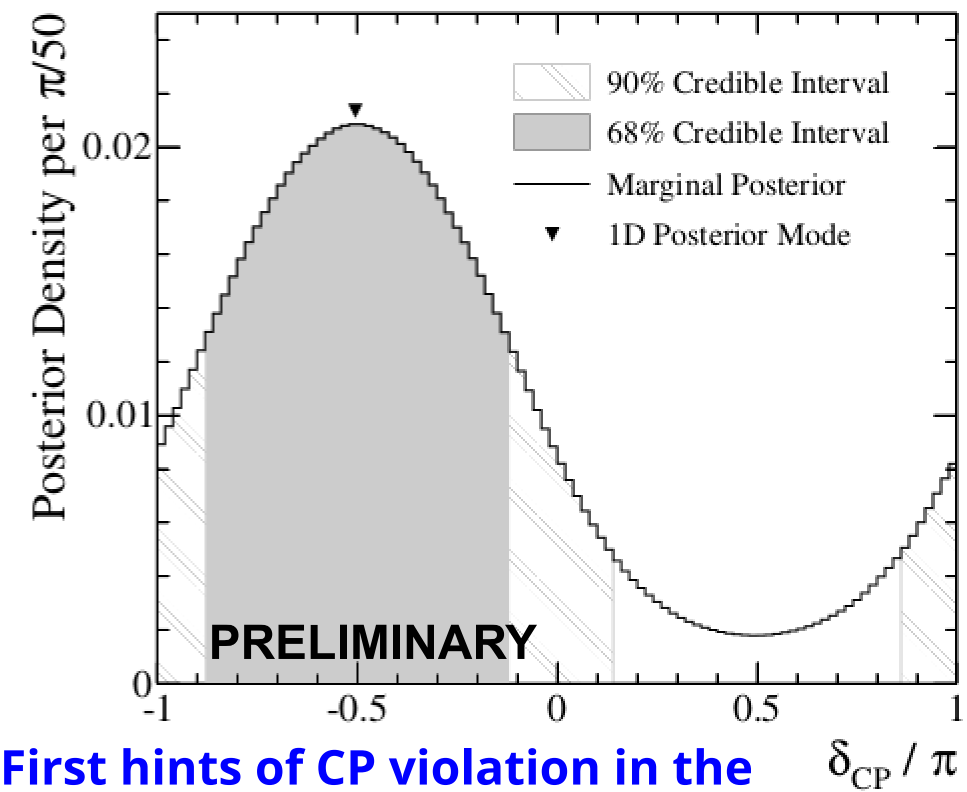
....

T2K 2014 Oscillation Results





Current world leading measurement of θ_{23}



First hints of CP violation in the lepton sector

Results from the Bayesian analysis presented in this talk. When combined with reactor measurements (Daya Bay etc), constraint on δ_{cp} emerges.

Currently writing paper for submission to **Phys. Rev. D**.

Conclusions

- What was once an unfeasible reweighting method has been made possible with the use of GPUs
- Calculation of oscillation probability with matter effects saw **2 orders of magnitude** speed-up
- Response function calculations saw **~20** speed-up
- In general, the analysis saw a **~20** speed-up
  - Using Emerald cluster, 1 fit takes **0.5** days (compared to **~10** days)
  - Move more reweighting (all) functionality to GPU to improve
- Utilized the Emerald HPC facility to run thousands of validation fits and finally the official result
- *"Accelerated Event-by-Event Neutrino Oscillation Reweighting with Matter Effects on a GPU" - JINST 9 2014*
  - http://arxiv.org/abs/1311.7579
- `http://hep.ph.liv.ac.uk/~rcalland/probGPU/`

Thankyou for your attention!

Backup Slides

Benchmark Code Snippet

CPU

```
clock.Start();

 for (int i = 0; i < N; ++i)
   {
     bNu->SetMNS( nominal[0], nominal[2], nominal[1], nominal[3], nominal[4], nominal[5], 100.0, true );
     bNu->propagateLinear( 2, 295, 2.6 );
     sample_weights[i] = bNu->GetProb(2, 2);
   }

 clock.Stop();
```

OpenMP

```
  clock.Start();

#pragma omp parallel for num_threads(4)
  for (int i = 0; i < N; ++i)
    {
      bNu->SetMNS( nominal[0], nominal[2], nominal[1], nominal[3], nominal[4], nominal[5], 100.0, true );
      bNu->propagateLinear( 2, 295, 2.6 );
      sample_weights[i] = bNu->GetProb(2, 2);
    }

  clock.Stop();
```

CUDA

```
  clock.Start();

  setMNS(nominal[0], nominal[2], nominal[1], nominal[3], nominal[4], nominal[5], true);
  GetProb(2, 2, 295, 2.6, energy, N, sample_weights);

  clock.Stop();
```

```
extern "C" __host__ void GetProb(int Alpha, int Beta, double Path, double Density, double *Energy, int n,
double *oscw)
{
  size_t dmsize = 3*3*sizeof(double);
  typedef double dmArray[3];
  dmArray *d = (dmArray*)malloc(dmsize);
  memcpy(d, &dm, dmsize);
  dmArray *dm_device;
  cudaMalloc((void **) &dm_device, dmsize);
  cudaMemcpy(dm_device, dm, dmsize, cudaMemcpyHostToDevice);

  size_t mixsize = 3*3*2*sizeof(double);
  typedef double mixArray[3][2];
  mixArray *m = (mixArray*)malloc(mixsize);
  memcpy(m, &mix, mixsize);
  mixArray *mix_device;
  cudaMalloc((void **) &mix_device,mixsize);
  cudaMemcpy(mix_device, mix, mixsize, cudaMemcpyHostToDevice);

  size_t size = n * sizeof(double);
  double *energy_device = NULL;

  cudaMalloc((void **) &energy_device, size);
  cudaMemcpy(energy_device, Energy, size, cudaMemcpyHostToDevice);

  double *osc_weights;
  cudaMalloc((void **) &osc_weights, size);

  dim3 block_size;
  block_size.x = 512;

  dim3 grid_size;
  grid_size.x = (n / block_size.x) + 1;

  propagateLinear<<<grid_size, block_size>>>(Alpha, Beta, Path, Density, mix_device, dm_device, energy_device,
osc_weights,n);

  cudaMemcpy(oscw, osc_weights, size, cudaMemcpyDeviceToHost);
  clean_up(); // cudaFree everything
}
```

Copy mixing matrix and mass matrix (matter effects) to device

Could copy to constant / texture memory

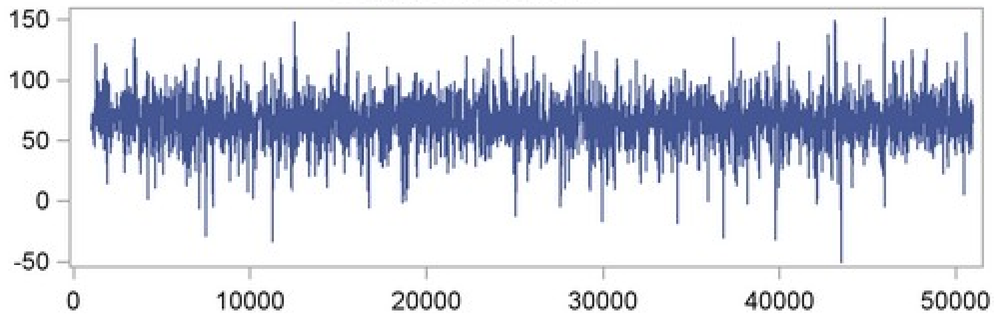
Copy Monte Carlo event energies to device

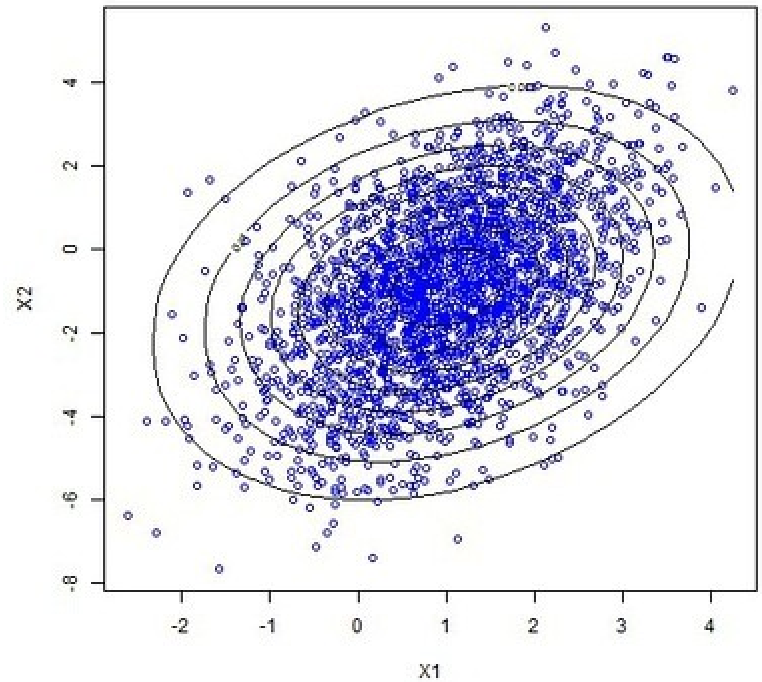
Execute kernel

Copy oscillation weights back to host

Markov Chain Monte Carlo

$$p(H_i|D,I) = \frac{p(H_i|I)p(D|H_i,I)}{p(D|I)}$$





- To evaluate the posterior distribution, need to integrate over high-dimensions
- MCMC provides an efficient way to perform the ~200-dimensional integral
- MCMC performs a semi-random walk through parameter space, following the path of the likelihood function
- Can run multiple chains on a cluster and combine output

Method

1 Throw 200 proposed new values of parameters

2 Evaluate likelihood function

$$
\begin{aligned}
-\ln(P) = & \sum_{i}^{ND280bins} N_i^p(\vec{b},\vec{x},\vec{f},\vec{d}) - N_i^d + N_i^d ln[N_i^d/N_i^p(\vec{b},\vec{x},\vec{f},\vec{d})] \\
& + \sum_{i}^{SK1R_\mu bins} N_i^p(\vec{b},\vec{x},s\vec{k}d) - N_i^d + N_i^d ln[N_i^d/N_i^p(\vec{b},\vec{x},s\vec{k}d)] \\
& + \sum_{i}^{SK1R_e bins} N_i^p(\vec{b},\vec{x},s\vec{k}d) - N_i^d + N_i^d ln[N_i^d/N_i^p(\vec{b},\vec{x},s\vec{k}d)] \\
& + \frac{1}{2}\sum_{i}^{E_\nu bins}\sum_{j}^{E_\nu bins} \Delta b_i (V_b^{-1})_{i,j} \Delta b_j \\
& + \frac{1}{2}\sum_{i}^{xsecpars}\sum_{j}^{xsecpars} \Delta x_i (V_x^{-1})_{i,j} \Delta x_j \\
& + \frac{1}{2}\sum_{i}^{fsipars}\sum_{j}^{fsipars} \Delta f_i (V_f^{-1})_{i,j} \Delta f_j \\
& + \frac{1}{2}\sum_{i}^{nd280det}\sum_{j}^{nd280det} \Delta d_i (V_d^{-1})_{i,j} \Delta d_j \\
& + \frac{1}{2}\sum_{i}^{skdet}\sum_{j}^{skdet} \Delta skd_i (V_{skd}^{-1})_{i,j} \Delta skd_j
\end{aligned}
$$

performed on GPU

Complete cycle ~20 million times

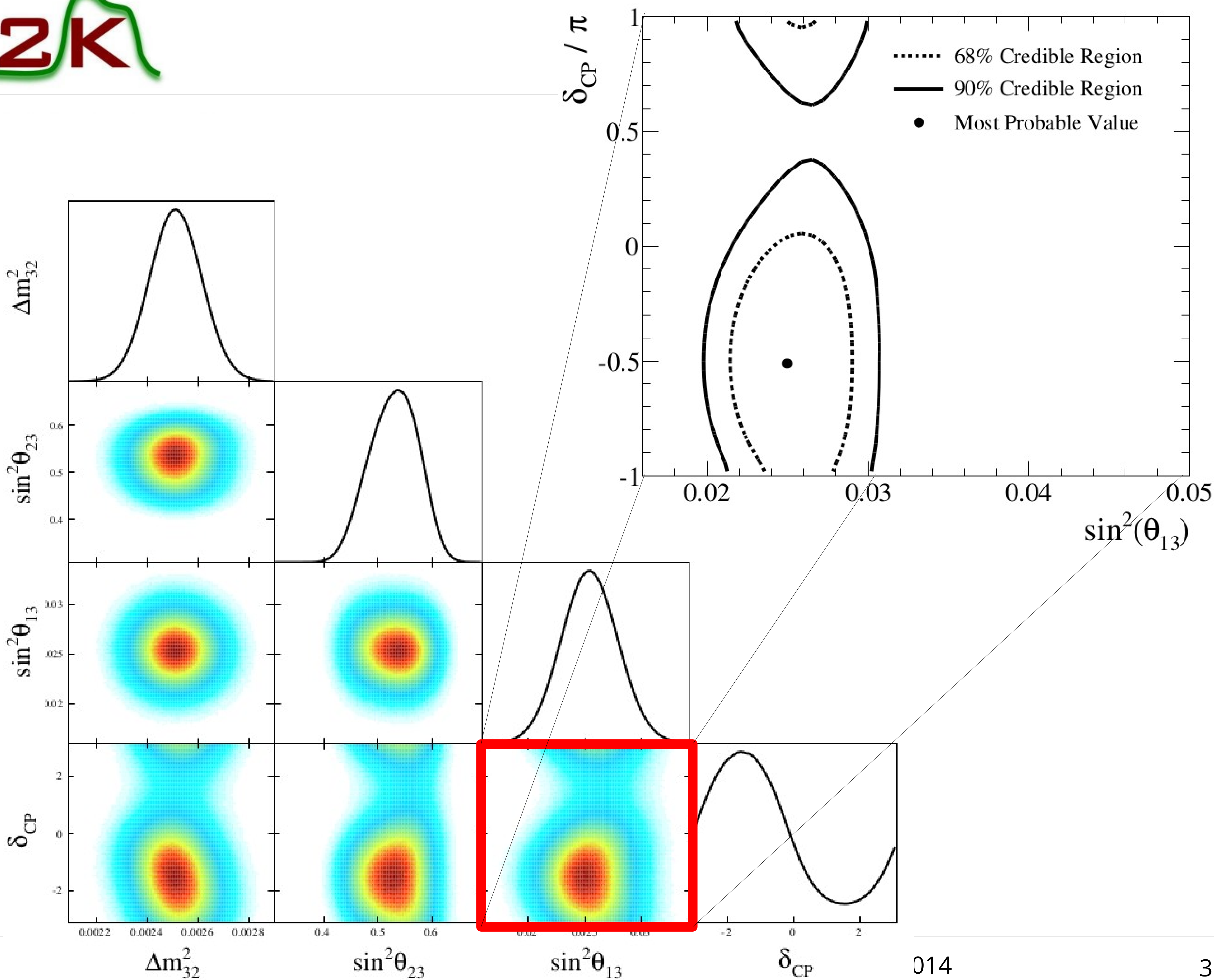
3 Is new state better than current?

Accept or reject

Write to disk

Time per step ~0.3 seconds

~20x speed-up


δ_CP / π
68% Credible Region
90% Credible Region
Most Probable Value
sin²(θ₁₃)
Δm²₃₂
sin²θ₂₃
sin²θ₁₃
δ_CP

Creating detector PDFs

- Use detector Monte Carlo to construct empirical PDFs of expected neutrino data distributions
- Apply neutrino oscillation model (and systematic model) by reweighting MC