GPU-Enabled Algorithms

Amir Farbin

Zubair Bhatti (Undergrad!)

University of Texas at Arlington

Outline

- Goal: Use FFT as a means of exploring feasibility of GPUs-Enable Algorithms for LBNE.
  - FFT might not be a good choice because it's highly optimizable/parallelizable
- GPU Basics
- Our Setup
- FFT in LBNE
- FFT CPU vs GPU
- HEP's (Unique?) Computing/Software Complexities
- Illustration of HEP GPU computing issues
- Solutions:
  - Concurrency Frameworks
  - Data Parallel Task Manager (DPTM)
- DPTM Design/Prototype
- Workplan
- Final thoughts

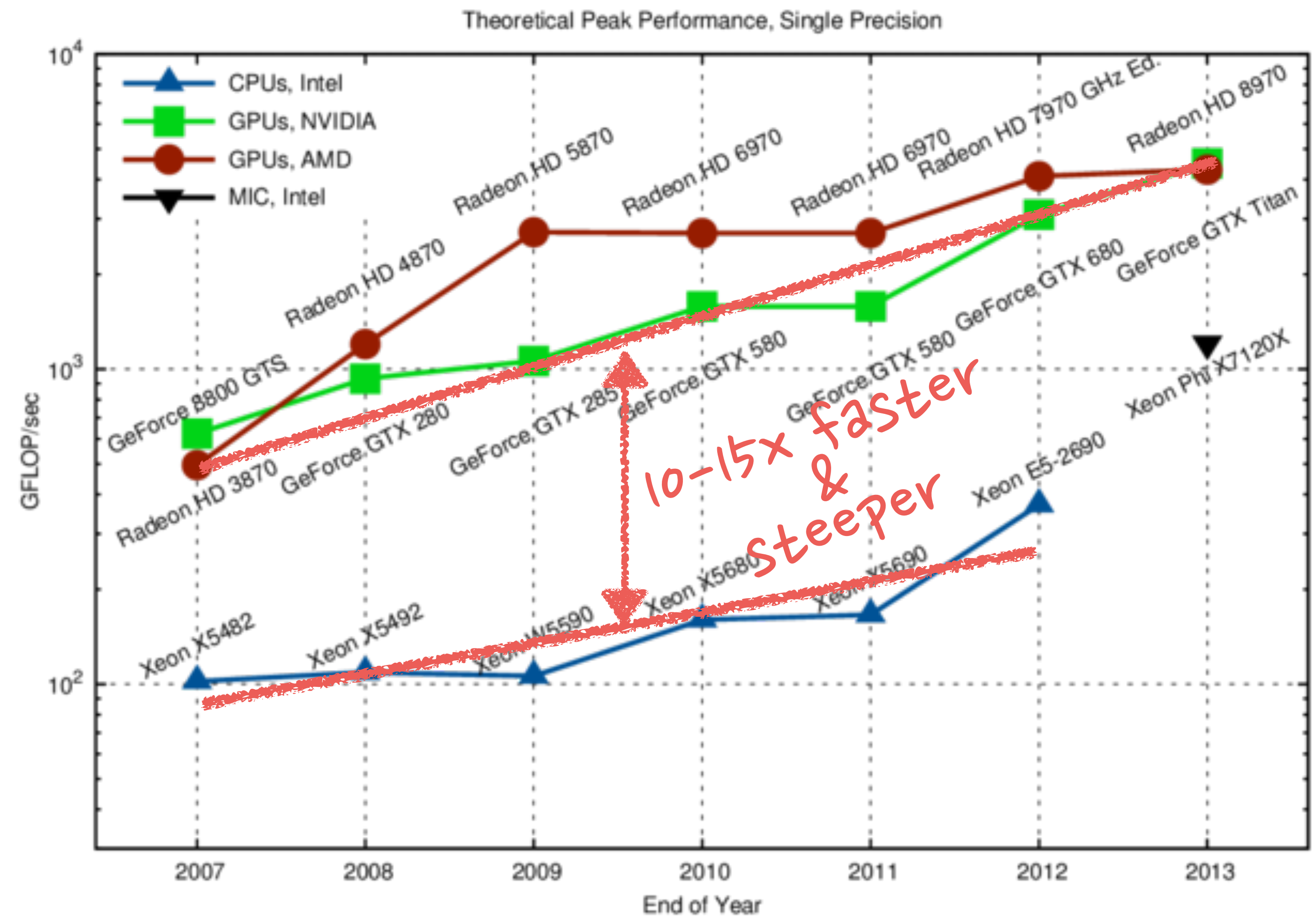
I will skip these if I'm taking too long.

GPU Basics

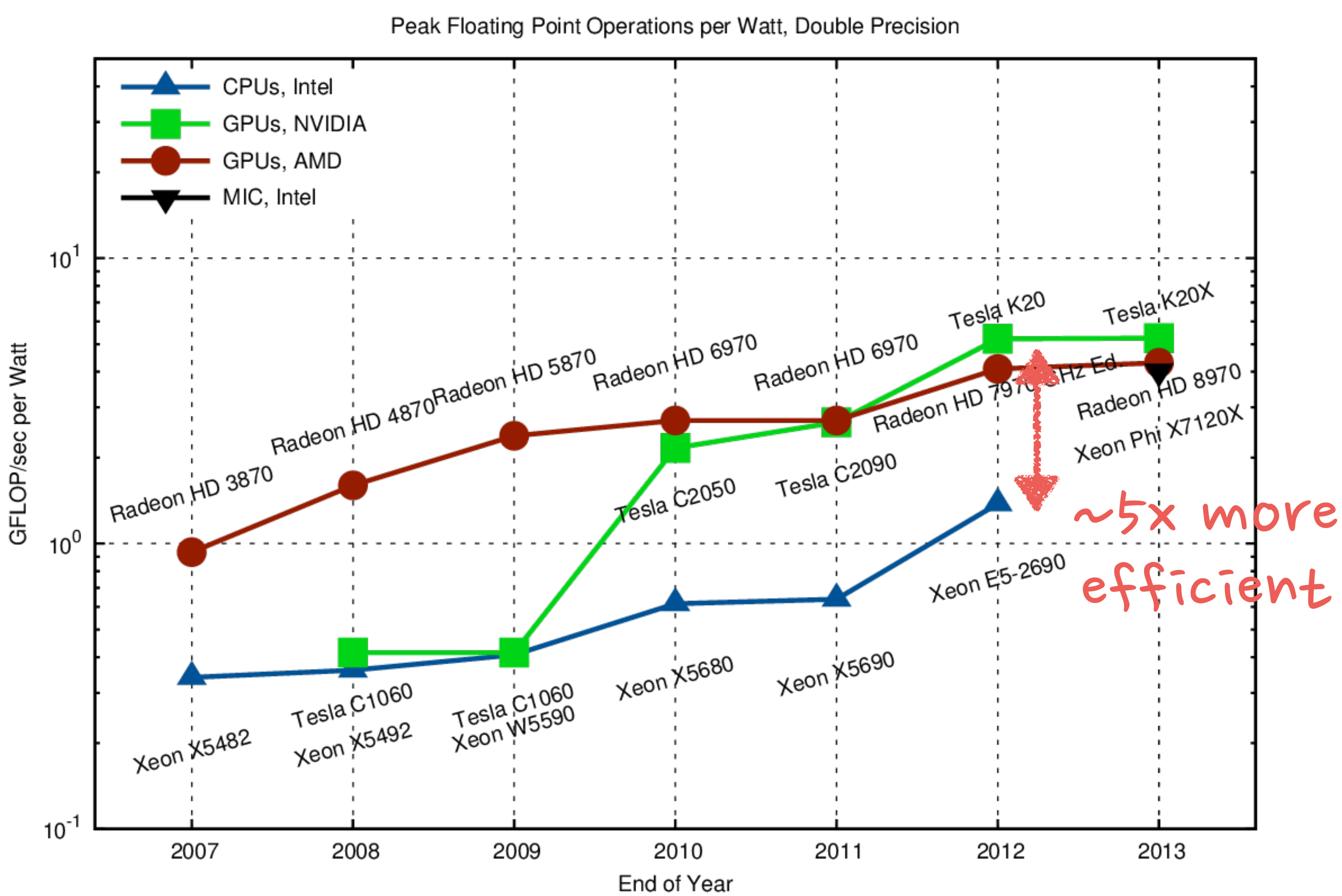
I can elaborate on these over coffee, if you like.

- GPU=Massively Parallel co-processor (Lots of little cores)
  - Evolved from Fix Function to General Purpose Computing.
  - Originally driven by gaming. General Computing on GPUs pushed by NVidia. Now everyone does it.
- GPUs can't replace CPUs… they are co-processors.
  - Developing software for GPUs is more complex than for CPUs.
- Exponential Increase in computing power (eg FLOPS) wrt to CPUs.
- One 2013 Mac Pro (7 TFLOPS from GPUs) would be world's 8th fastest supercomputer 10 years ago.
- Today: Oak Ridge's Titan (and most other supercomputers) rely on GPUs.

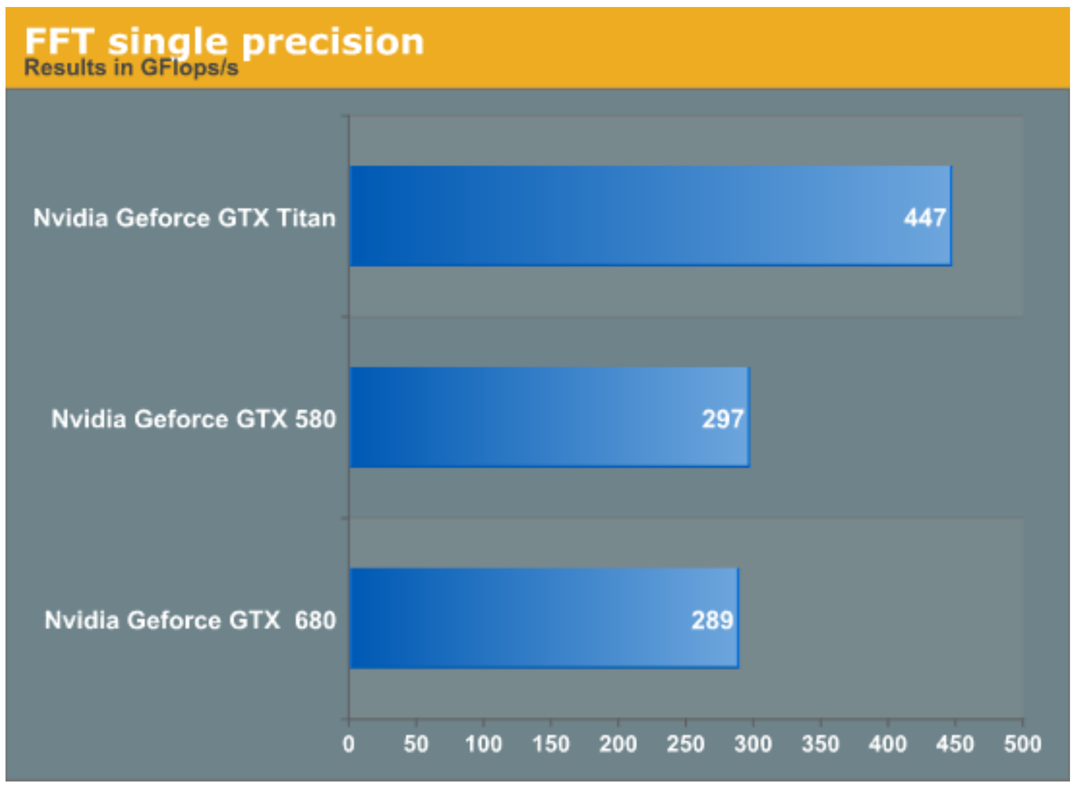
CPU vs. GPU



Theoretical Peak Performance, Single Precision
GFLOP/sec
End of Year
2007
2008
2009
2010
2011
2012
2013
CPUs, Intel
GPUs, NVIDIA
GPUs, AMD
MIC, Intel
Radeon HD 3870
Radeon HD 4870
Radeon HD 5870
Radeon HD 6970
Radeon HD 6970
Radeon HD 7970 GHz Ed.
Radeon HD 8970
GeForce 8800 GTS
GeForce GTX 280
GeForce GTX 285
GeForce GTX 580
GeForce GTX 580
GeForce GTX 680
GeForce GTX Titan
Xeon X5482
Xeon X5492
Xeon W5590
Xeon X5680
Xeon X5690
Xeon E5-2690
Xeon Phi X7120X
10-15x faster
&
steeper



Peak Floating Point Operations per Watt, Double Precision
GFLOP/sec per Watt
End of Year
2007
2008
2009
2010
2011
2012
2013
CPUs, Intel
GPUs, NVIDIA
GPUs, AMD
MIC, Intel
Radeon HD 3870
Radeon HD 4870
Radeon HD 5870
Radeon HD 6970
Radeon HD 6970
Radeon HD 8970
Tesla K20
Tesla K20X
Tesla C2050
Tesla C2090
Tesla C1060
Tesla C1060
Xeon X5482
Xeon X5492
Xeon W5590
Xeon X5680
Xeon X5690
Xeon E5-2690
Xeon Phi X7120X
~5x more
efficient



FFT single precision
Results in GFlops/s
Nvidia Geforce GTX Titan
447
Nvidia Geforce GTX 580
297
Nvidia Geforce GTX 680
289
0
50
100
150
200
250
300
350
400
450
500

Setup

3.4 GHz * 2 FPUs * 2 FLOPs per cycle * 12 cores * 90% (efficiency) = 146 gigaflops.

	Cores/ Threads	Peak Single Precision GFLOP/s	Peak Double Precision GFLOP/s	Memory	Cost
Mac Pro (2.66 GHz Xeon X5650)	2 x 6 / 24	Estimate: ~ 150	Estimate: ~ 85	26 GB	~$5000 (full system)
NVidia GTX650	384 / 2 x 2048	812	X	1 GB	$120
NVidia GTX780	2304 / 12 x 2048	3977	166	3 GB	$550
NVidia Titan	2688 / 14 x 2048	4500	1500	6 GB	$1250

could be 25-30x faster than CPU!

Reason for Titan

Hard to find... should be at UTA Monday!

FFT in LBNE

I'm sure I've made stupid mistakes here... please correct me (later?).

	35t	10kt	34kt
Wires	2,048	307,200	~1,044,480
Samples	6.55 Million	983 Million	3.3 Billion
Data Size (MB)	25 MB	3.75 GB (2 GB from Tom)	12.750 GB (6.8 GB)
Cosmics/Readout (Surface)	~0.25	~70	~2400
Estimated deconvolution Time/ readout (no 0 suppression)	~ 7 sec	30 mins	~1.5 hours
Potential deconvolution Time/ readout on GPU (no 0 suppression)	O(milliseconds)	~1 min	~3 mins
Surface Zero-suppressed samples/readout	~8000	2.24 Million	7.6 Million
Estimated deconvolution Time / readout (with 0 suppression)	tiny	~4 sec	~14 sec
Potential deconvolution / readout time on GPU (with 0 suppression)	very tiny	O(milliseconds)	~0.5 sec

← naive scaling (Cosmics/Readout, 34kt)

← Assuming optimized FFT build (Estimated deconvolution time, 34kt)

Assuming Today's GPU technology (Potential deconvolution time on GPU)

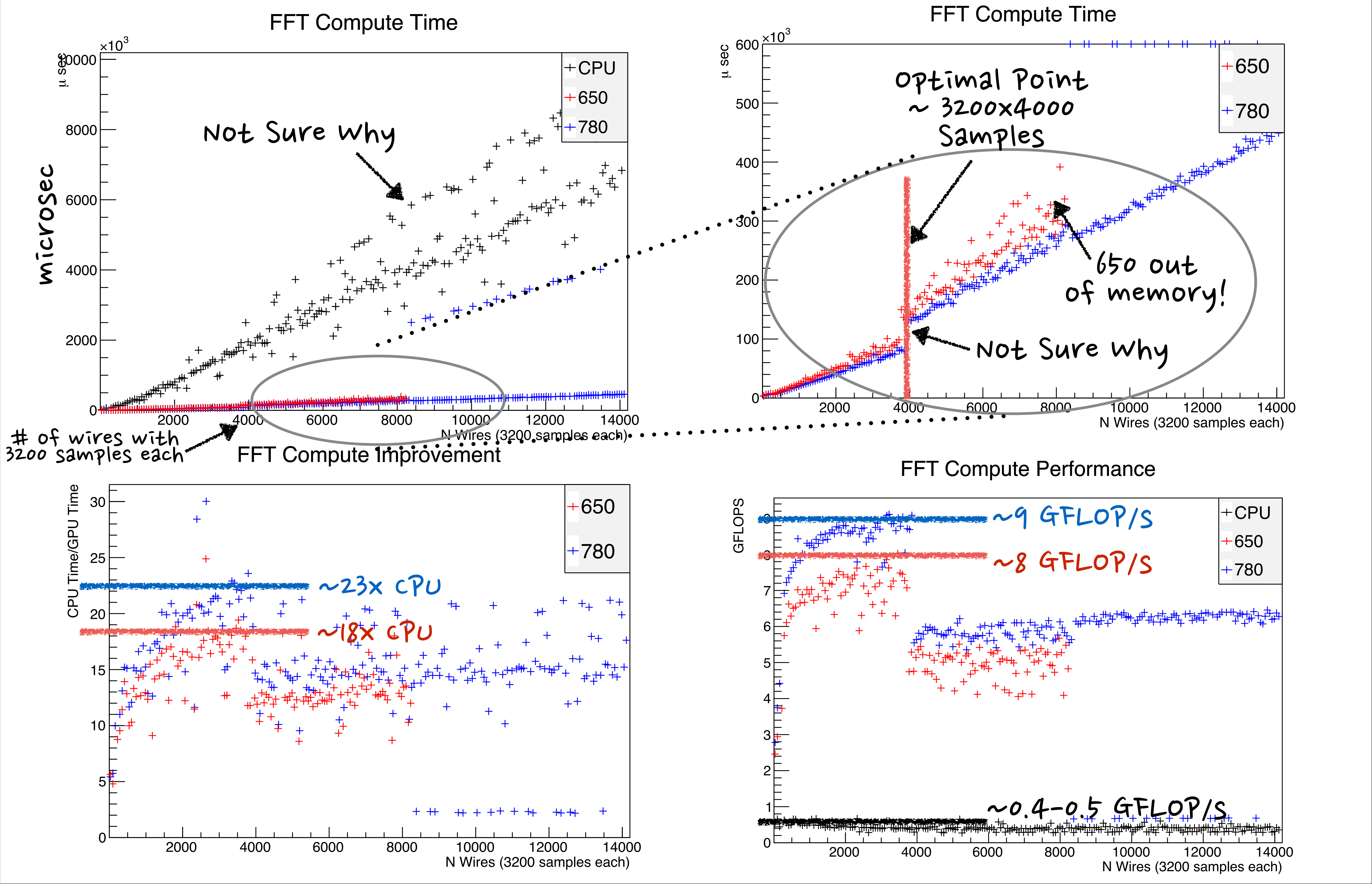
← very naive and probably wrong extrapolation assumes: ~4000 hits/track ~8 samples/hit (Surface zero-suppressed samples, 34kt)

→ conclusion zero suppression is the real solution

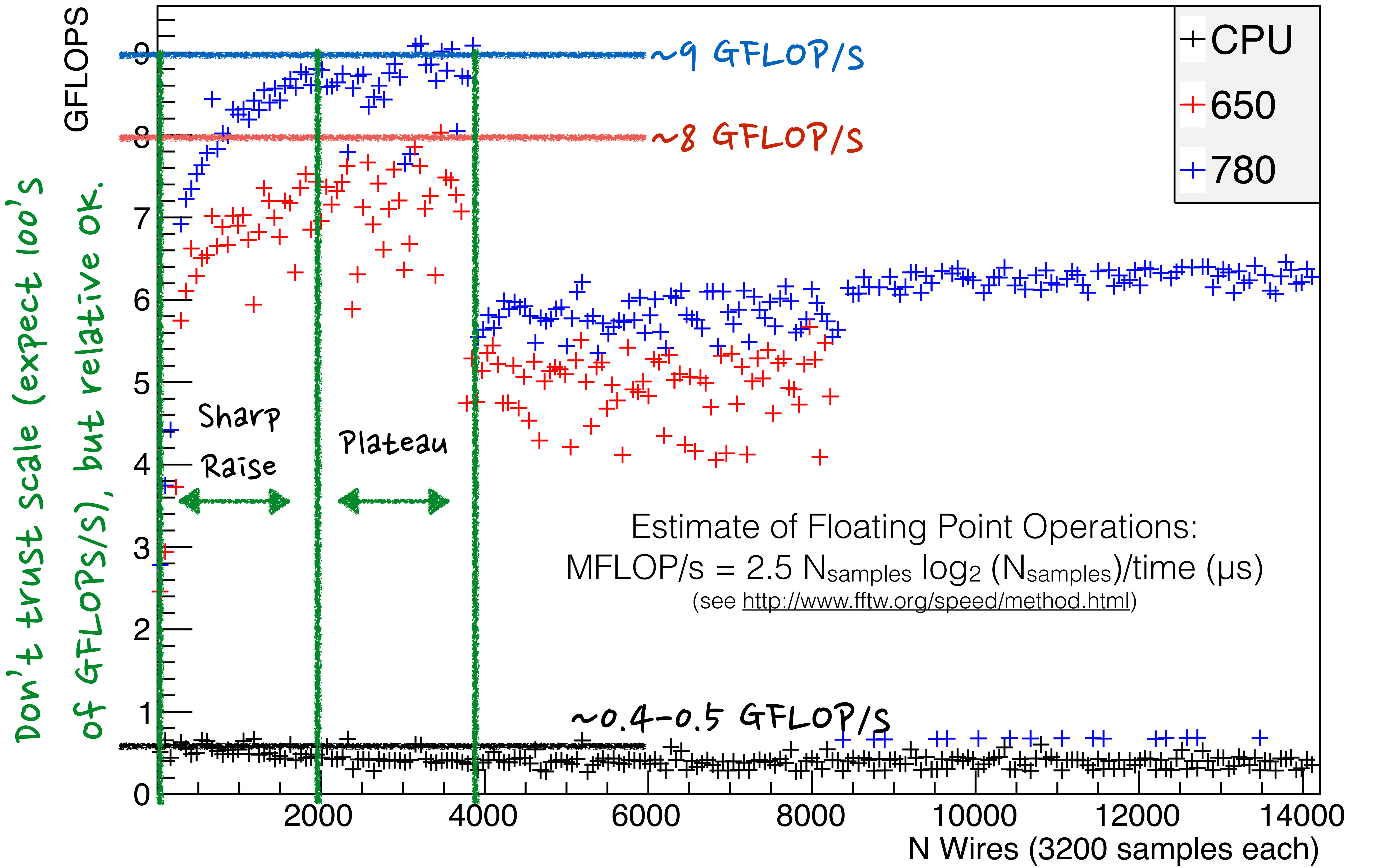
FFT on CPU and GPU

- Let's compare performance:
  - Fastest FFT in the West (FFTW) really is the fastest.
  - We used FFTW3 on CPU… same as ROOT (which is used by LBNE).
  - We used FFTCU on GPU… based on FFTW3.
    - We checked and the results are nearly identical.
  - We generated Random Vectors to FFT (use same vector for CPU and GPU).
  - Data has to be transferred between CPU and GPU:
    - Can be done asynchronously (ie transfers simultaneous with computations)…
    - AMD's APU (and other future GPU's) share memory/pointers between CPU/GPU and require no transfer
      - ➡ We will just time the compute performance, not the transfer.
  - Benchmark as function of number of wires… 3200 samples per wire.

FFT CPU vs GPU



FFT Compute Performance



conclusion: Feed enough data to GPU and you'll get ~23x faster FFTS.

30 mins hit finding ⟶ 1-2 mins for 10kt!

Great… how do we get it?

- FFT in LArSoft:
  - *CalWire* uses SignalShapingService to deconvolve Raw ADC into "Wires".
  - *SignalShapingService* uses *SignalShaping*, which uses the *LArFFT util*, which uses *Root's FFTW3 wrappers*.
  - *GausHitFinder* then finds hits on the deconvolved wires.
- Just need to wrap CUFFT in a class inheriting from TVirtualFFT… and we are done!
- Be a bit more clever and we can do the whole hit finding on the GPU!
  - Best to merge CalWire and GaussFitFinder to minimize GPU/CPU data transfers.

“Not so fast kemosabe…”

- Some issues:
  - We saw that we need to simultaneously FFT about 2000 wires to get maximal performance.
    - That’s OK… we can refactor CalWire… not so hard.
  - As GPUs get faster, we’ll need to simultaneously process more and more wires to be optimal.
    - Maybe 1 event doesn’t have enough data.
      - Probably OK for FFTs in 34t LBNE… but this may be true of other algorithms (e.g. tracking).
      - Example: Tracking (eg ATLAS trigger).
    - Our embarrassingly parallel HEP code serially processes one event at time…

“concurrency problem”

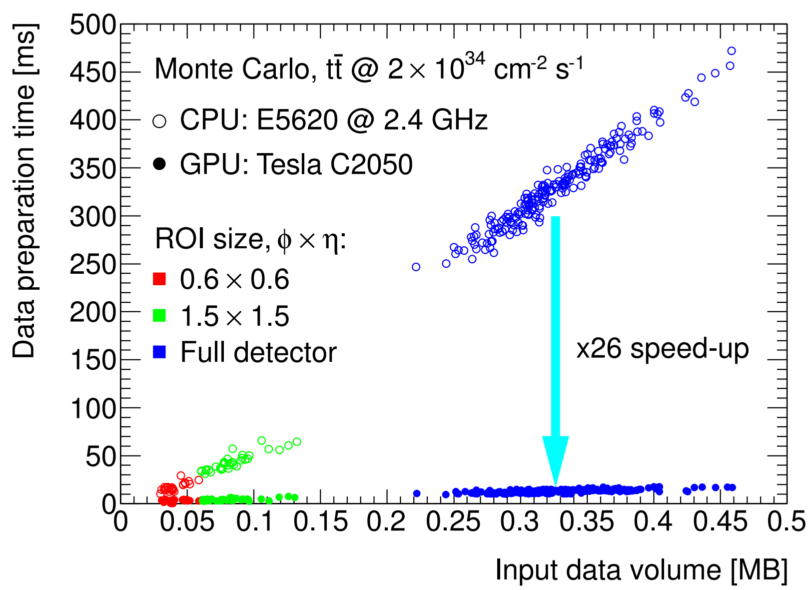


Figure 7. Performance improvement for data preparation steps.

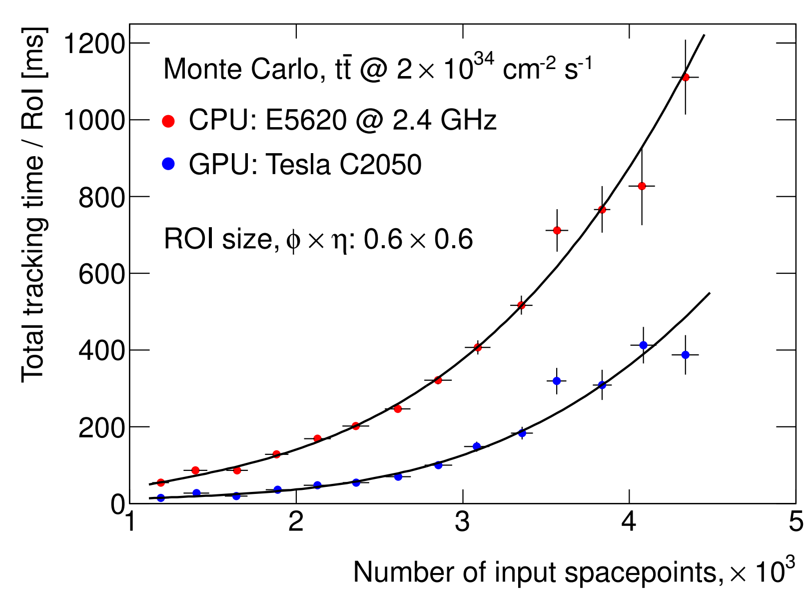
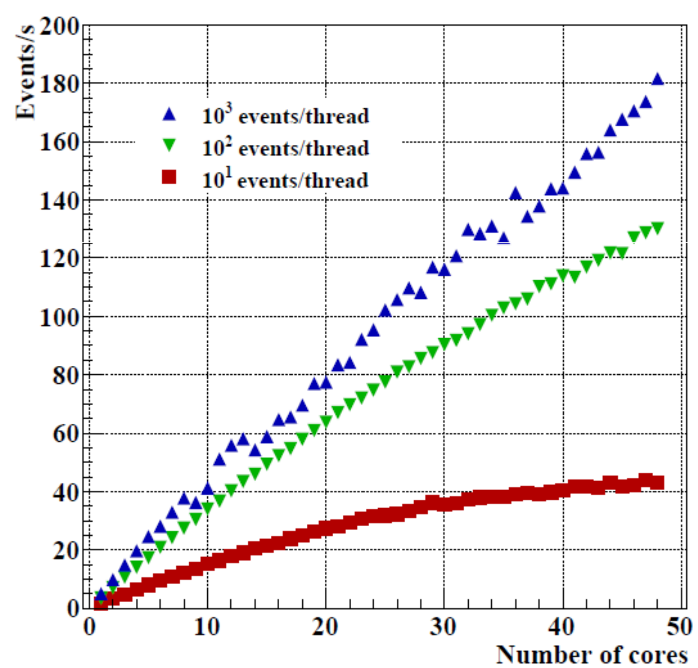


Figure 8. Performance improvement for tracking steps.



From: http://iopscience.iop.org/1742-6596/396/1/012018/pdf/1742-6596_396_1_012018.pdf and http://www.stfc.ac.uk/PPD/resources/pdf/ppd_seminar_111005_talk_Abdeslem_Djaoui.pdf

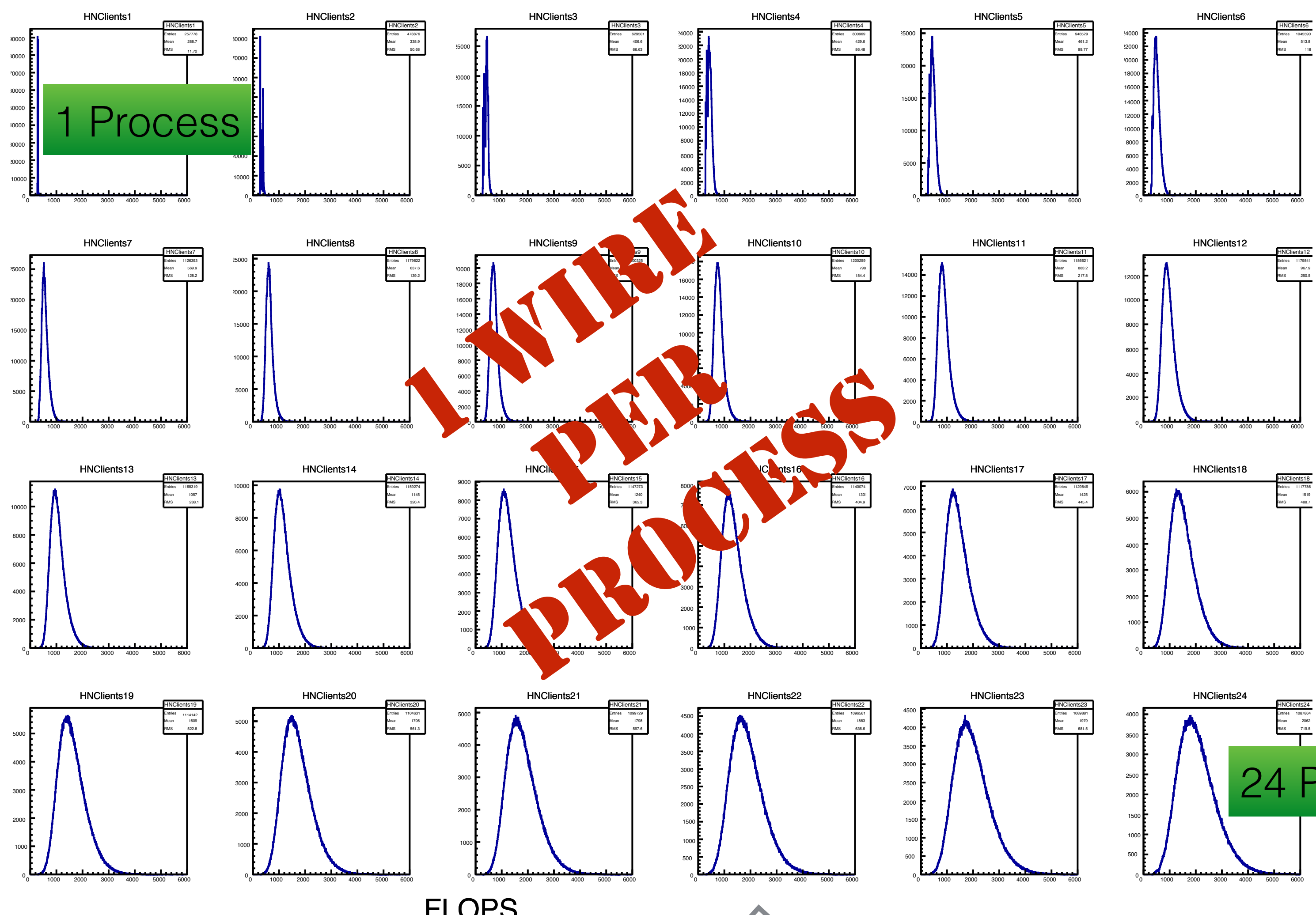
- More issues:
  - For most applications (eg image processing or even HEP trigger), a machine can just run one GPU algorithm at a time… but for HEP reconstruction, many algorithms have to smartly share the GPU.
    - HEP applications are very complex, running 100's of algorithms, dozens of which may benefit from GPU.
  - # CPU processes (~ # cores) ≠ # of GPUs
    - CPU processes will compete for GPU resources…
      - Ultimately CPU and GPU will wait for each other!
    - Must be clever to fully utilize both GPU and CPU.
- Let's demonstrate…

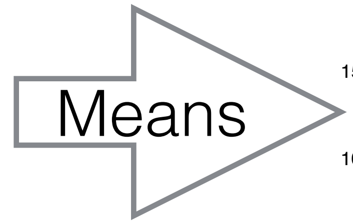
"complexity problem"

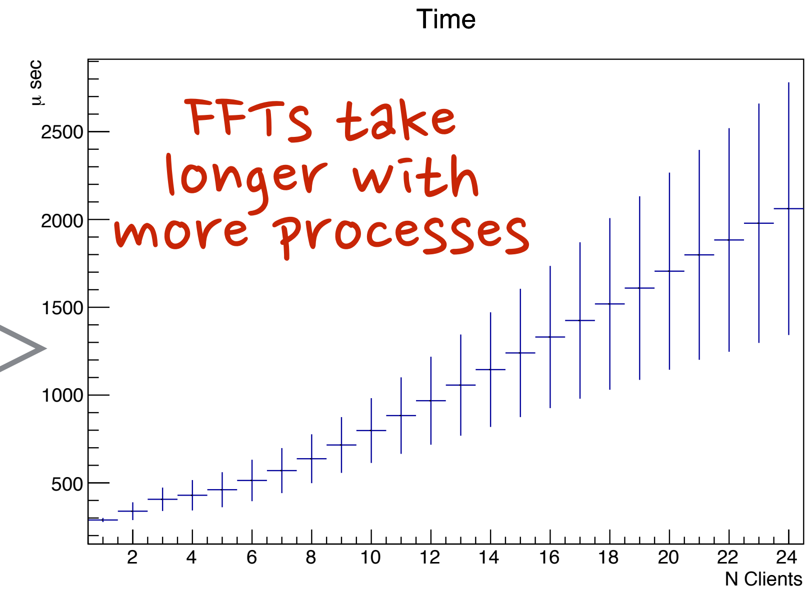
"Full utilization problem"

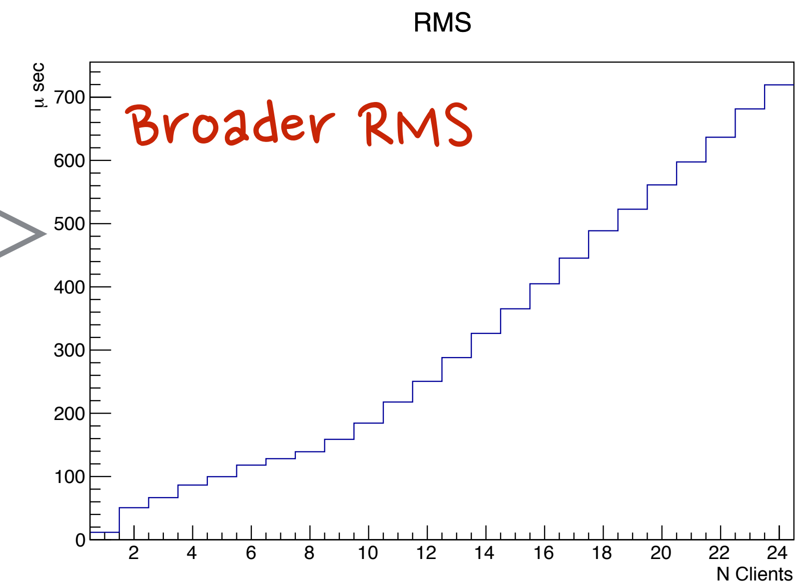
Multiple Processes

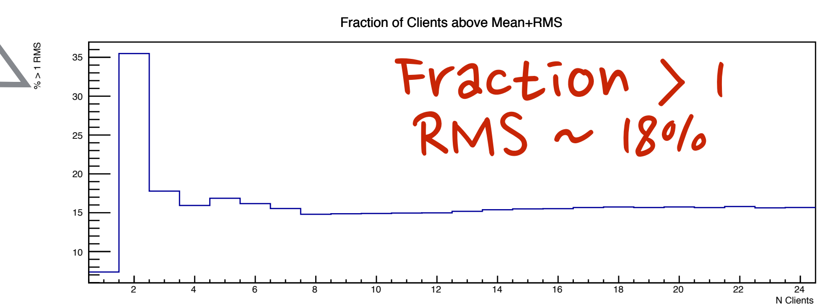
Time per FFT on GPU

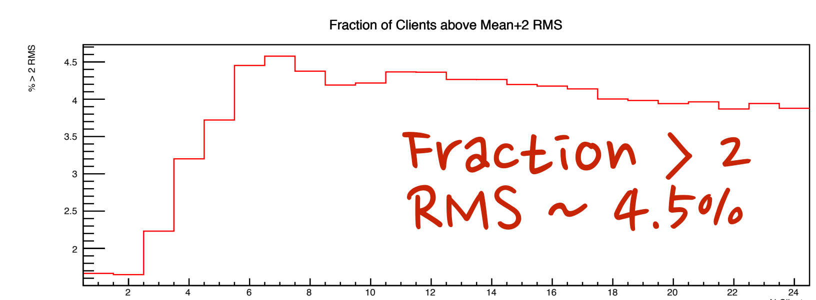


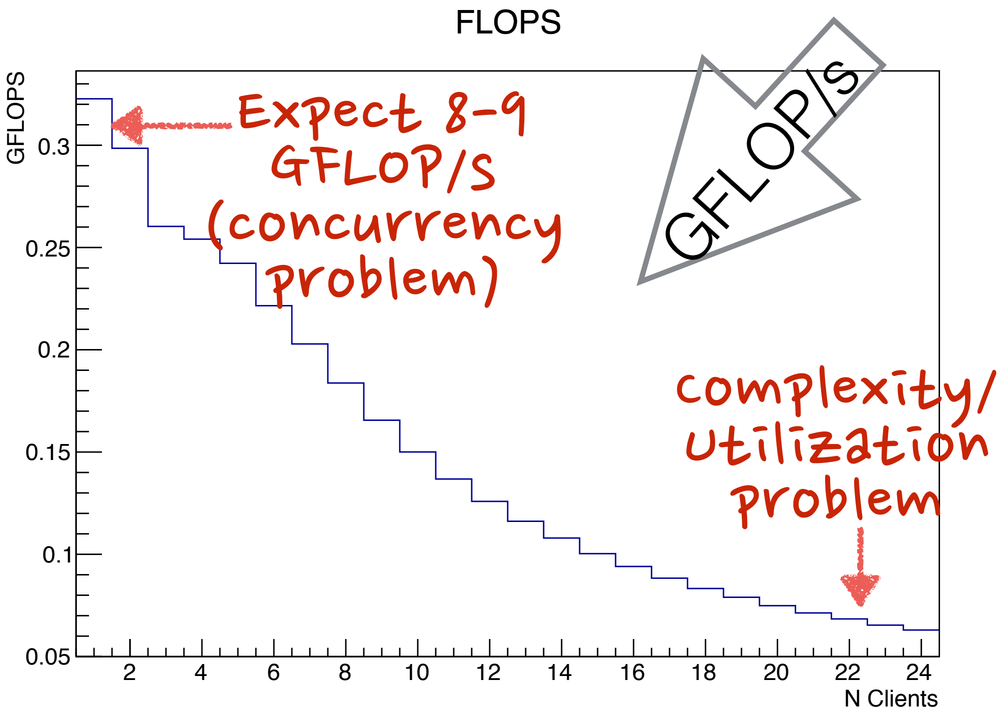








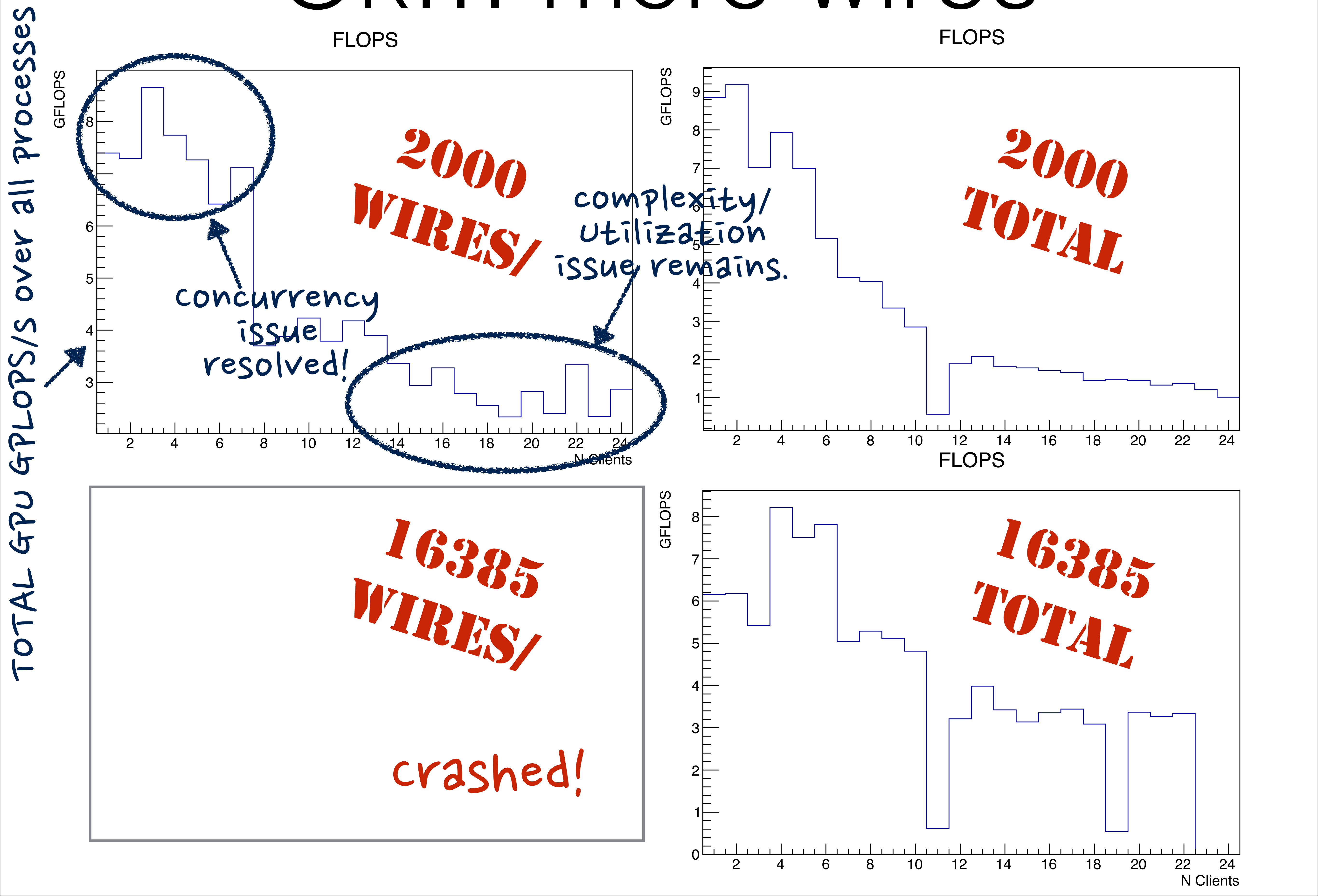




If we imagine each FFT is a different algorithm:

utilization problem ➜ complexity problem

Ok… more wires



Solutions?

- Large concurrency effort in HEP…
    - See: http://concurrency.web.cern.ch
    - Designing a concurrent framework. Example:
        - Each algorithm becomes a thread
        - Every algorithm can simultaneously process multiple events worth of data.
        - Not just a solution for GPU efficiency… also maximizes CPU efficiency.
    - But this probably requires complete rewrite of HEP frameworks and/or algorithms.
        - Not practical for legacy software such as current experiment reconstructions and GEANT4
- Our (AF + Box Leangsuksun) idea: **Data Parallel Task Manager (DPTM)**
    - Addresses all problems as well as …
        - It's hard for typical physicist to parallelize their code, then optimize, etc…
        - Machines on the GRID could have different GPUs… GPU code typically has to be optimized for every

Are you ready to give up LArSoft for a new framework?

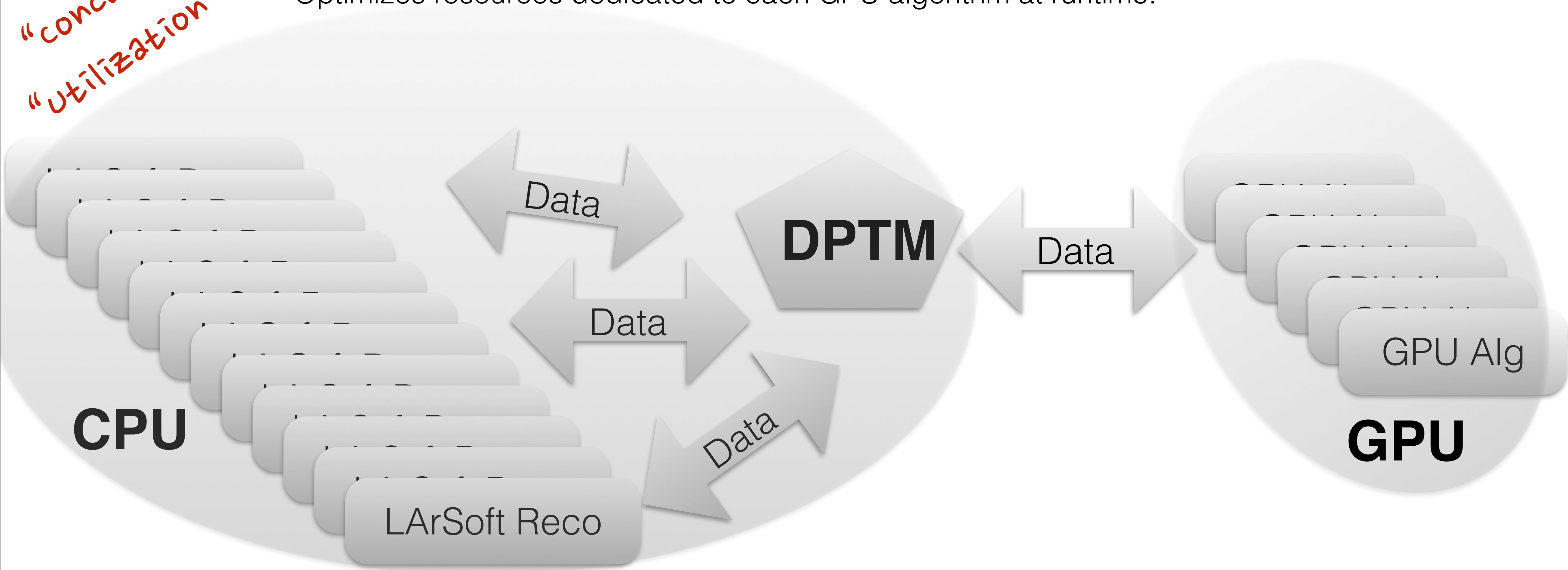
"Expertise Problem"

"Heterogeneous GPU Problem"

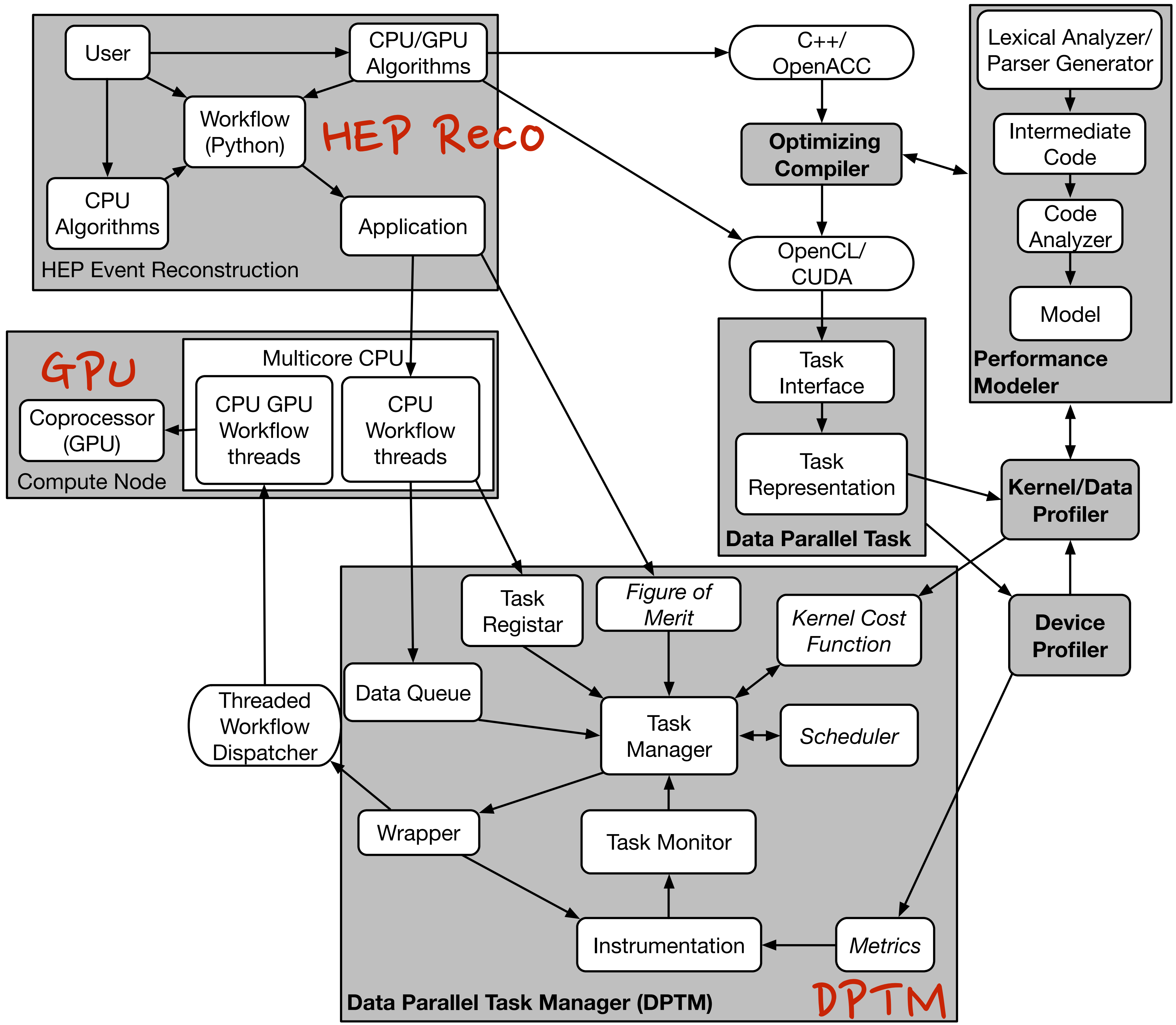
DPTM Idea

- Run large instances of existing application (eg serial reconstruction)… memory permitting.
- Application processes requests processing of various data using various algorithms.
- DPTM collects requests
  - Determines optimal configuration of GPU algorithm at runtime.
  - Collects (aggregates) data from CPU multiple processes and simultaneously processes on GPU.
  - Optimizes resources dedicated to each GPU algorithm at runtime.

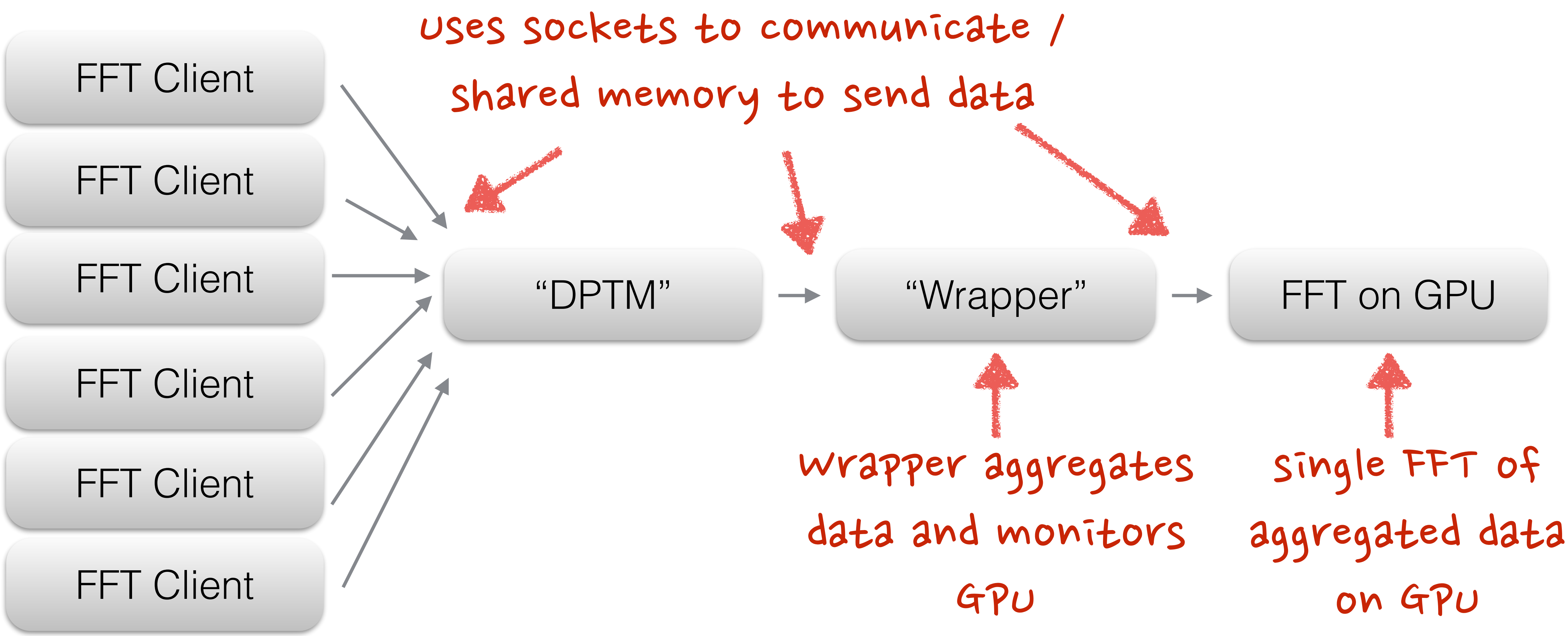
"Heterogeneous"
"concurrency"
"Utilization"



Full DPTM Design



DPTM Prototype



Proof that DPTM Resolves Issues

HOPEFULLY BY END OF THE WEEK!

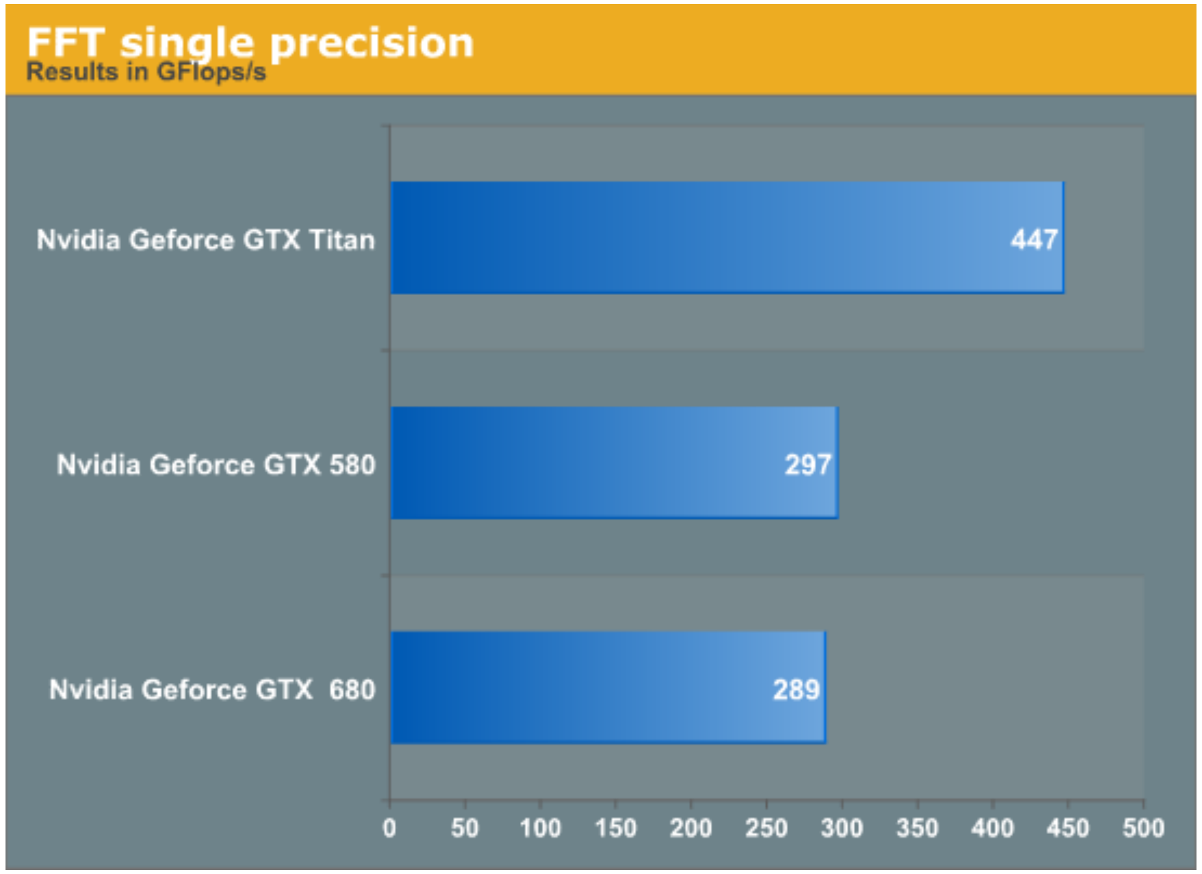
Workplan

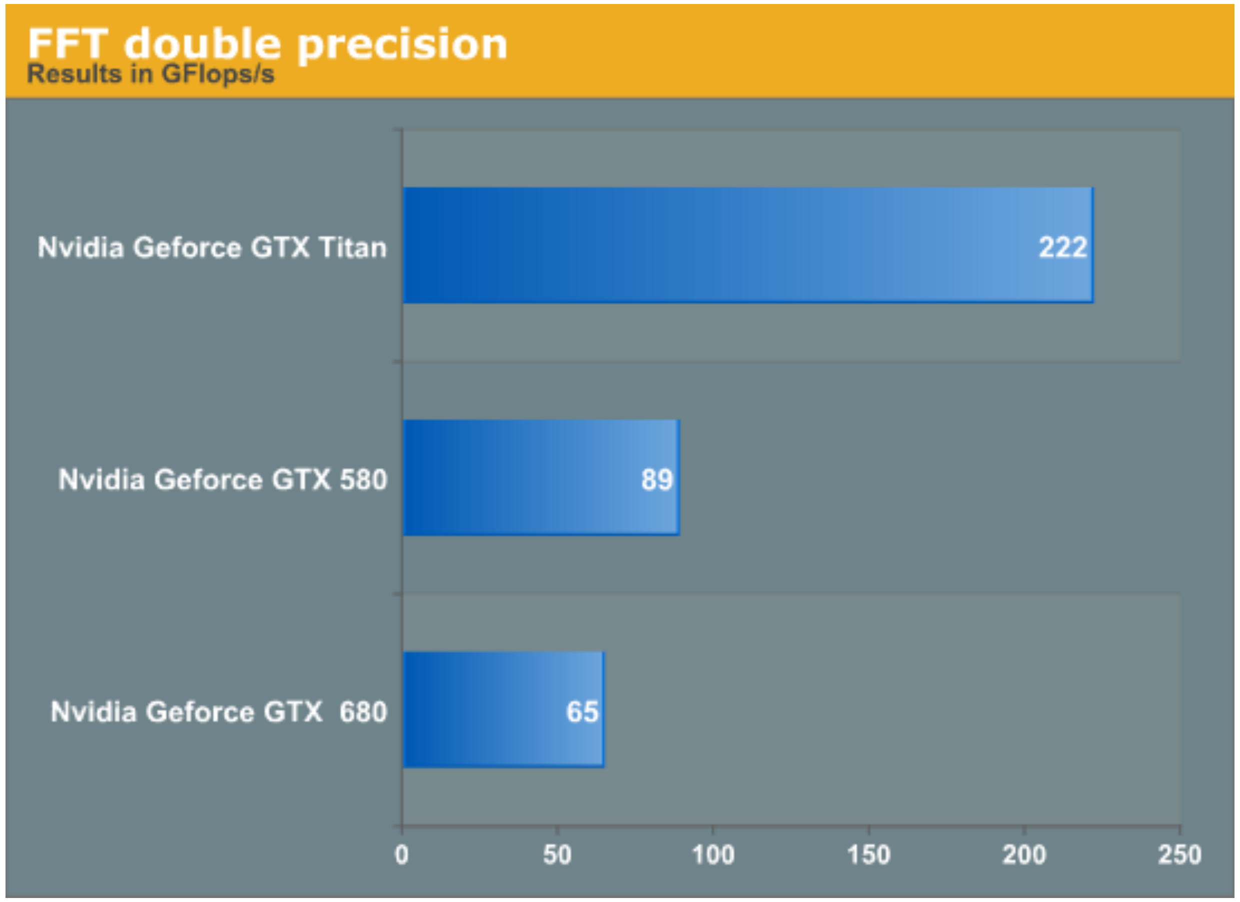
- Finish DPTM Prototype demonstration.
- Build LBNE LArgSoft on Mac (thanks Brett et al).
  - Or use the FNAL GPU nodes…
- Wrap CUFFT in class inheriting from TVirtualFFT
- Study impact on LBNE deconvolution…
  - Scaling studies (N wires, M processes, …)
  - Integrate with DPTM prototype…
- Rewrite CalWire to batch deconvolve.
- Rewrite GPU Kernel (in CUDA) for GaussHitFinder
- …

Final Thoughts

- Imagine today's 8th most powerful supercomputer on your desk in 10 years.
- The potential for using GPUs in LBNE can't be ignored.
    - LArg TPC is potentially very well suited for GPU reconstruction.
        - Massively parallel/uniform data.
        - Reconstruction algorithms similar to image processing.
    - We can quickly get some benefit in reconstruction… but not optimal.
    - Lots of **R&D required** here… determine if and when does it make sense for us to rely on GPUs.
        - We should understand the issues, time-scale, …
        - GPUs are a moving target… performance and available technologies evolving very rapidly..
- GPUs for back-end electronics / Trigger?
    - Imagine **streaming reconstruction**… **trigger on full reconstructed events**. Is there a case for this in LBNE?
    - Using OpenCL you can use the **exact same algorithm on CPU, GPU, DSP, FPGAs, and ASICs**.
        - Greatly simplifies everything.

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
8	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301
9	DOE/NNSA/LLNL United States	**Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216	4,293.3	5,033.2	1,972
10	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM	147,456	2,897.0	3,185.1	3,423





http://www.anandtech.com/show/6774/nvidias-geforce-gtx-titan-part-2-titans-performance-unveiled/3