The FtsTrackFinder and Online Processing

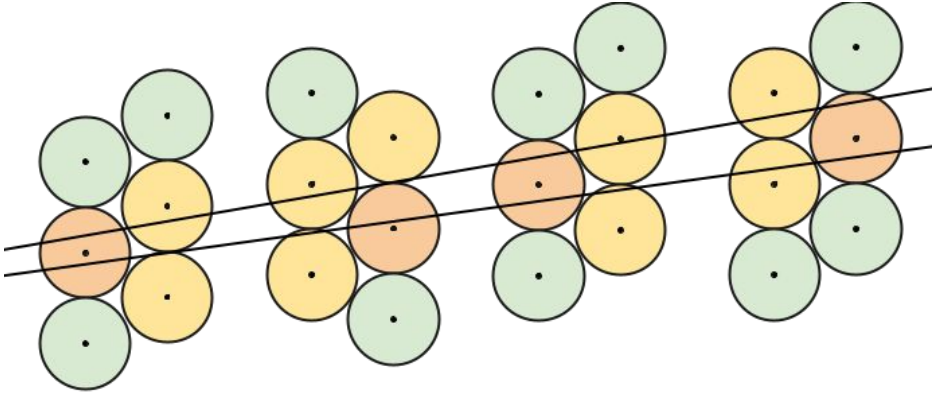
FtsTrackFinder improvements and accelerated implementation, PandaRoot for online processing concept, SYCL programming model

PANDA Collaboration Meeting 23/2 | 13.06.2023

Bartosz Soból

Updated FtsTrackFinder

- Algorithm details: talk at CM 22/2
- Modifications in ZOX hit selection
  - Additional constraint - single hit per layer
- Bug fixes



Accuracy results - full tracks

- Sim tracks passing through at least 40 straws
- Reco tracks with all parts reconstructed
- Sim and reco tracks considered as matched when at least 80% of straws match

Accuracy results - full tracks

10000 events, muons, **theta ∈ (0.0, 15.0)**

muons	beam mom	mom	ratio new	ratio prev
1	1,0	0,55	**97%**	95%
3	1,0	0,55	**95%**	93%
5	1,0	0,55	**94%**	90%
8	1,0	0,55	**92%**	85%
1	5,0	2,55	**97%**	95%
3	5,0	2,55	**96%**	94%
5	5,0	2,55	**96%**	92%
8	5,0	2,55	**94%**	88%
1	15,0	5,55	**96%**	95%
3	15,0	5,55	**97%**	94%
5	15,0	5,55	**96%**	92%
8	15,0	5,55	**95%**	88%

10000 events, muons, **theta ∈ (2.5, 5.0)**

muons	beam mom	mom	ratio new	ratio prev
1	1,0	0,55	**96%**	95%
3	1,0	0,55	**91%**	86%
5	1,0	0,55	**87%**	76%
8	1,0	0,55	**80%**	61%
1	5,0	2,55	**97%**	95%
3	5,0	2,55	**94%**	88%
5	5,0	2,55	**91%**	77%
8	5,0	2,55	**86%**	62%
1	15,0	5,55	**97%**	95%
3	15,0	5,55	**94%**	88%
5	15,0	5,55	**91%**	77%
8	15,0	5,55	**87%**	62%

Accuracy results - FT1234 tracks (low energy)

- Simulation tracks passing through
  - at least 26 straws in FT1234
  - less than 14 straws in FT56 (can't be reconstructed)
- Reco tracks reconstructed only in FT1234
- Sim and reco tracks considered as matched when at least 80% of straws in FT1234 match

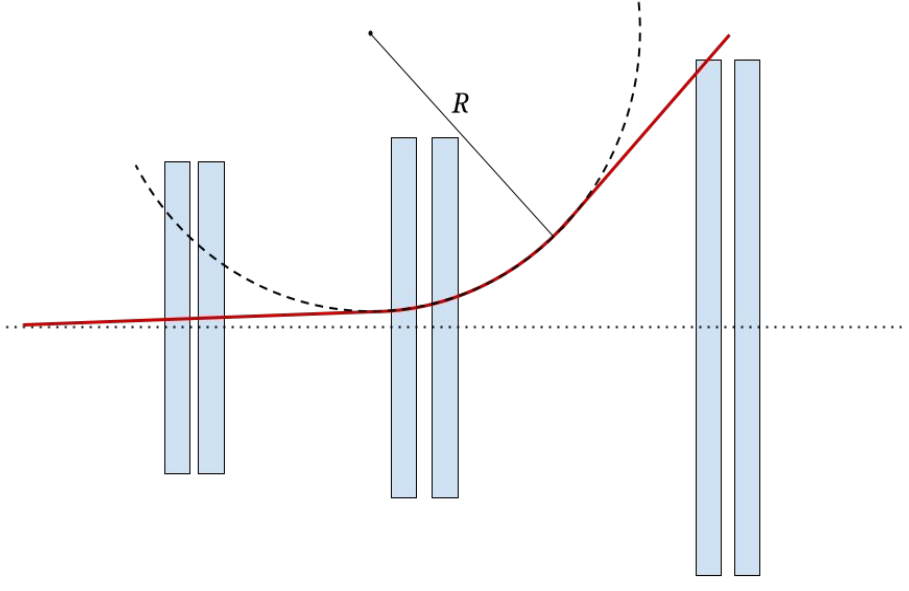
10000 events, muons, **theta ∈ (0.0, 15.0)**

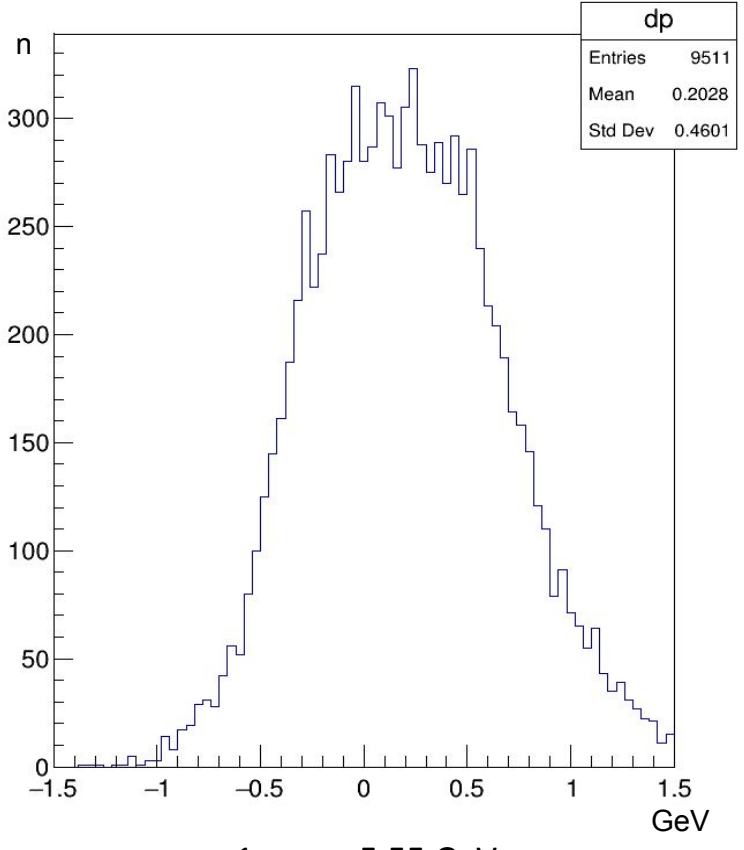
muons	beam mom	mom	ratio new	ratio prev
1	1,0	0,55	**85%**	82%
3	1,0	0,55	**80%**	75%
5	1,0	0,55	**80%**	74%
8	1,0	0,55	**80%**	71%

Momentum estimation - 2022

Method 1

- $P = 0.3BR$
- B from the middle of the magnet (Z = 464cm)
- Radius from straws in FT34



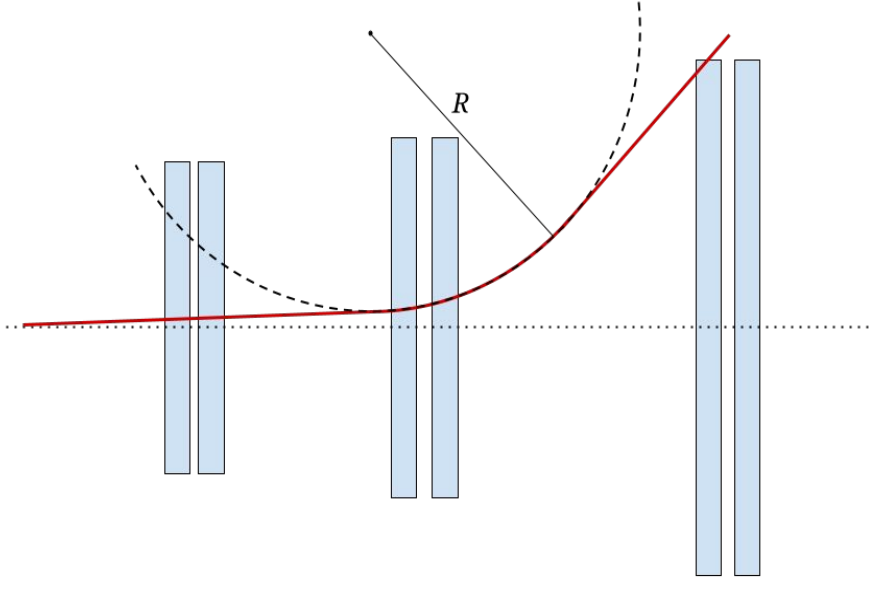


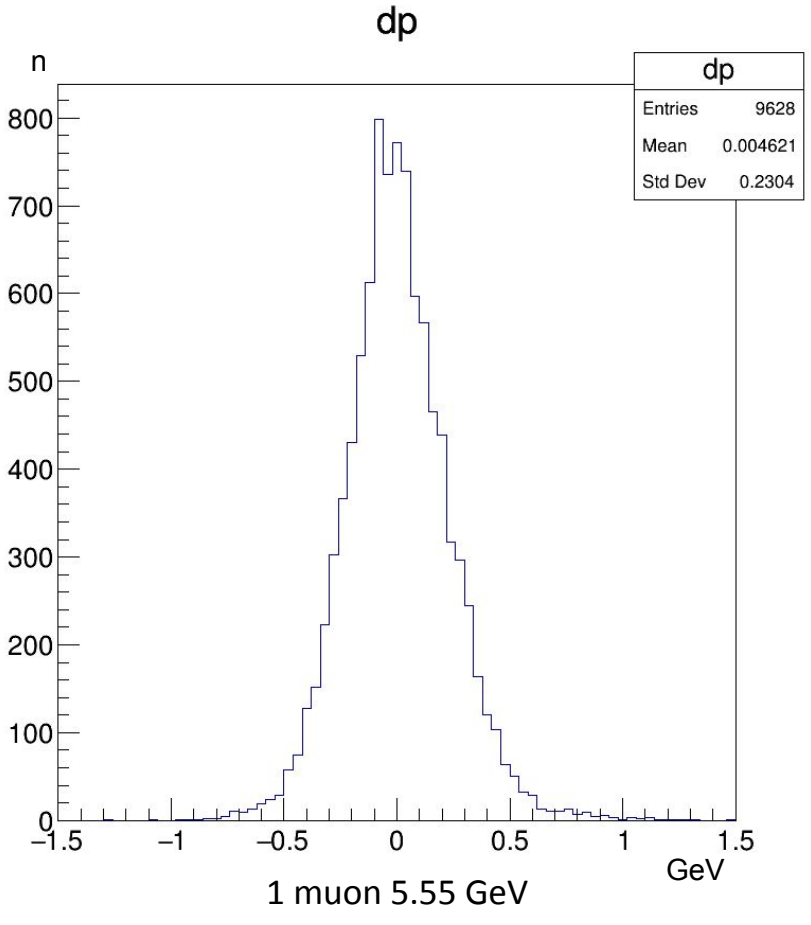
1 muon 5.55 GeV

Momentum estimation

Method 1

- $P = 0.3BR$
- B from the middle of the magnet (Z = 464cm)
- Radius from straws in FT34



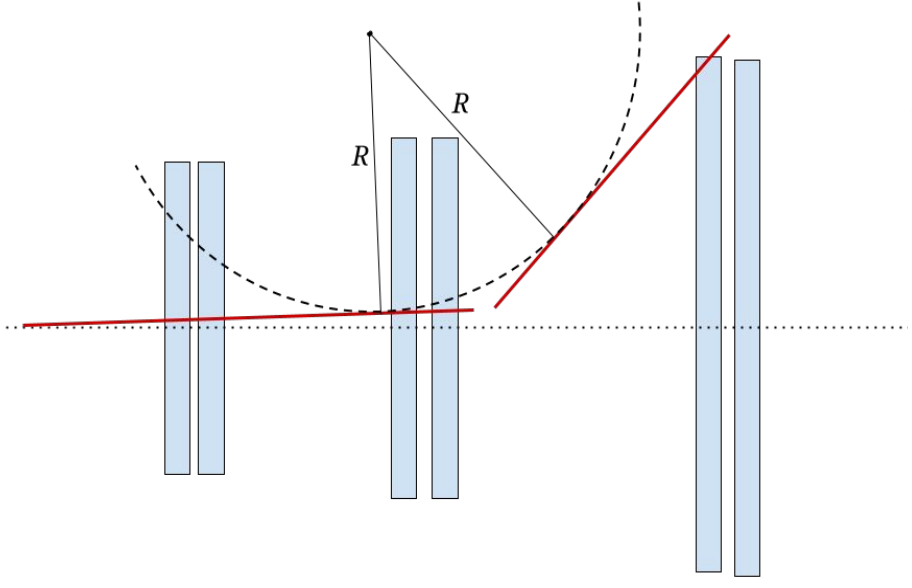


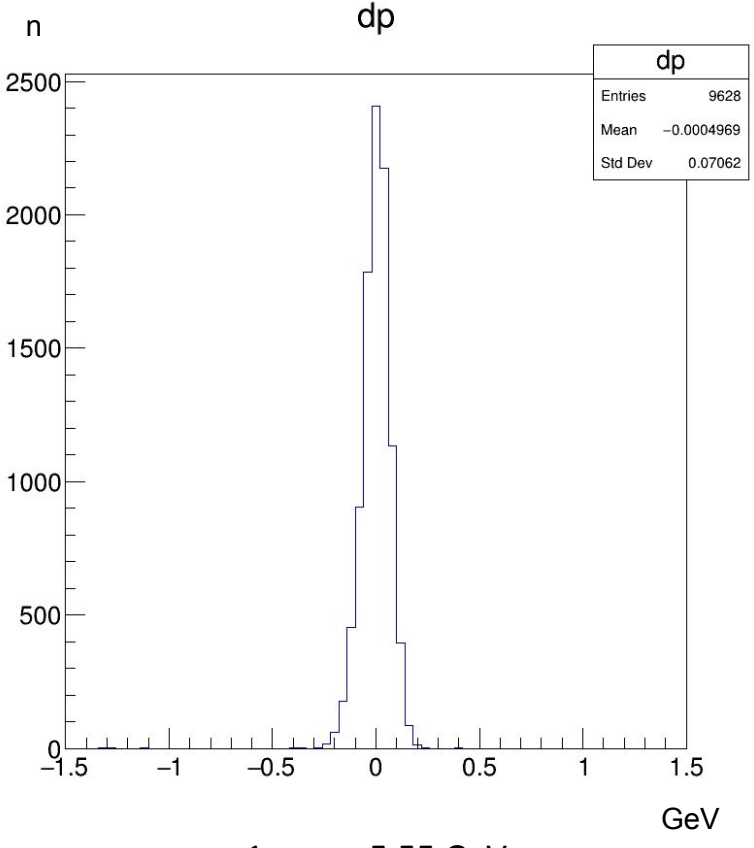
1 muon 5.55 GeV

Momentum estimation

Method 2 (only higher energy particles)

- $P = 0.3BR$
- B from the middle of the magnet (Z = 464cm)
- Radius from tracks in FT12/56





1 muon 5.55 GeV

Updated FtsTrackFinder

- Any feedback on algorithm performance will be much appreciated
  - Evaluation on more complex/realistics physics cases required
- Usage example macro can be found in PandaRoot repository:

  `macro/tracking/trackingTasks/ftsTrackFinder.C`

PandaRoot for online processing

- PandaRoot *Core* module
  - Simple, trivially copyable data structs for Hits, Digis, Tracks, etc.
  - Algorithms impl. operating only on simple data types ^
  - No *TObject* inheritance
    - Can be made available in ROOT context by dictionaries (linkdef)
  - Ability to be compiled standalone
    - Parameters/geom will be difficult here
- ROOT wrappers for *Core* data types
  - Classes owning simple objects
  - *TObject* or FairRoot base classes inheritance
    - Additional functionality for offline analysis and deep ROOT interop

PandaRoot for online processing

- Simplest example

```
struct Hit {
    uint16_t fStrawId;
    float fIsochrone;
};

class HitWrapper : public TObject{
public:
    inline uint16_t GetStrawId() {
        return fHit.fStrawId;
    }
    inline float GetIsochrone() {
        return fHit.fIsochrone;
    }

private:
    Hit fHit;
    ClassDef(HitWrapper, 1);
};
```

How to implement online/offline, CPU/GPU

- ***Option 1*** - Plain C++
- ***Option 2*** - Accelerated/parallelized - native: CUDA, HIP, OpenMP
  - Hard to maintain, multiple implementations for different hardware
- ***Option 3*** - Accelerated/parallelized - vendor and platform agnostic: SYCL
  - Hardware in the GreenCube will change before PANDA operation
  - *FtsTrackFinder* implemented
  - Potential CBM cooperation
    - They are interested in multi-backend vendor-agnostic solutions
    - I'm implementing one CBM algorithm in SYCL to compare with native version
    - They also have an in-house solution
  - Other options: Kokkos, Alpaka

Potential for SYCL in PandaRoot

- Supports wide range of platforms
  - Vendor agnostic GPU support is crucial
  - Always possible to fall-back to CPU when necessary
  - Easy to test CPU vs GPU performance
- Living ecosystem
  - Evolving standard, growing community, open-source and stable compilers (Intel DPC++, hipSYCL)
- Easy basics and easy to deploy on local machines (hipSYCL, OpenMP CPU backend)
  - Can be even bundled using CMake
- SYCL code will most probably work on Virgo successor

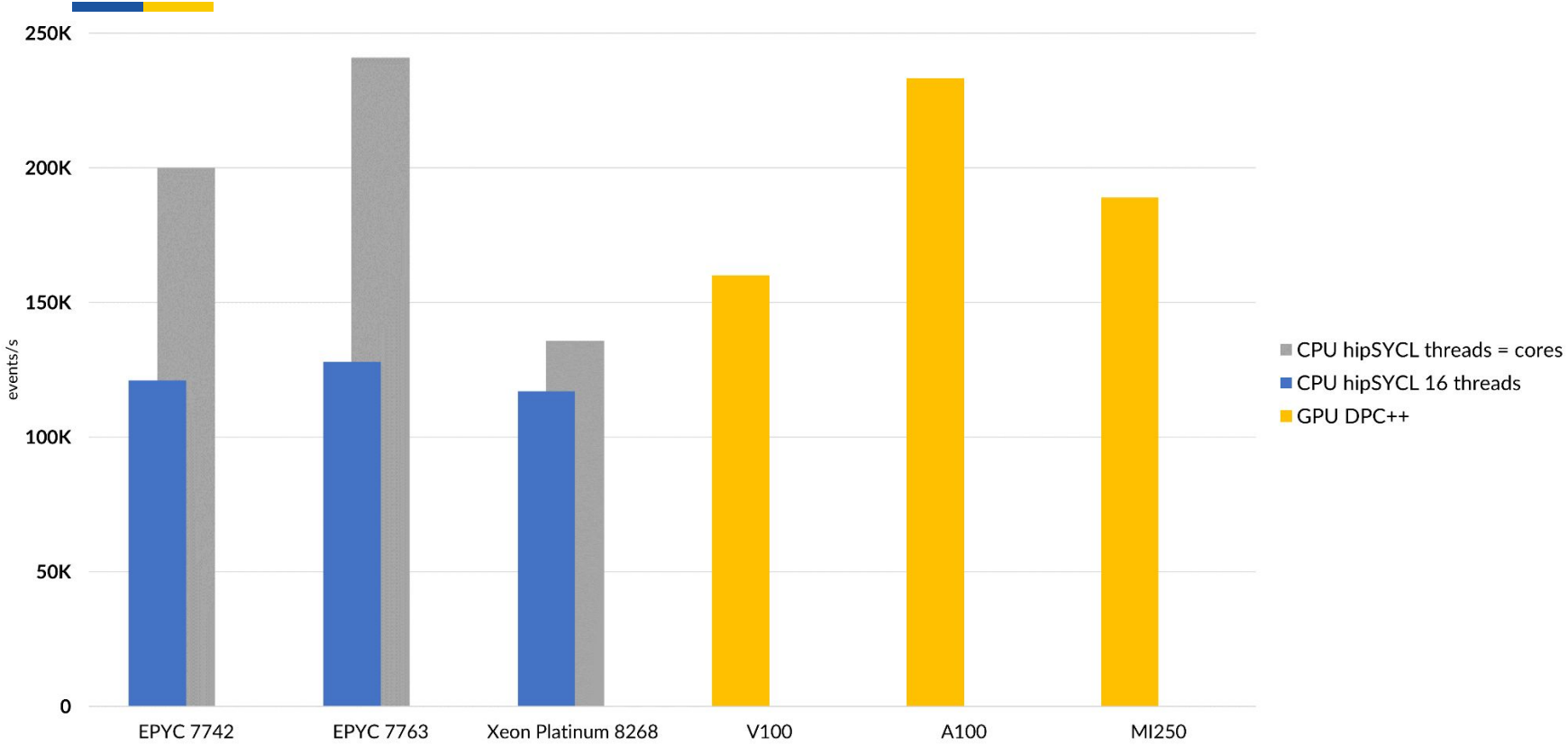
FtsTrackFinder SYCL implementation strategy

- Start with plain C++ single-threaded code
- Port to SYCL with minimal possible effort
- Introduce optimisations
  - Mainly data-flow and memory layout
  - Try to keep kernel code similar to initial version
  - Try to stay within simpler SYCL interfaces (buffers, ranges)
- Result: 7 Kernels + helper functions, ~1.5k lines of accelerated code
- Single code for different platforms

Performance evaluation

- Modern hardware from all leading vendors
  - Rome, Milan, Cascade Lake - CPU
  - V100, A100, MI250 - GPU
  - Alveo U280 - FPGA
- Two major implementations:
  - hipSYCL (0.9.4)
  - DPC++ (2023.0, 2023.1 - MI250)
    - triSYCL/sycl - U280
- Compared with native CUDA implementation on NVIDIA GPUs

Performance summary



Performance summary

- Algorithm itself isn't ideal for GPU
  - Lot's of branches and not parallelizable short loops
  - More data-bound than compute-heavy
- CPU parallelization is quite good
  - Up to ~16 threads
- GPU performance is mediocre compared to CPU
  - Increases with larger *batch* size
  - Probably wouldn't improve significantly even with further fine-tuning
  - GPU optimisations also positively affect CPU performance
  - With SYCL, we have an efficient CPU version for free

Performance summary

- Alveo U280 performance ~2 orders of magnitude worse
  - Didn’t introduce any FPGA-specific optimisations
  - Adventure making the code compile and run
    - Still has some issues - potentially compiler bugs
  - Great to have SYCL available on such platforms
    - Certain algorithms can highly benefit
- Intel FPGAs - tools probably more mature, but not tested (yet)

Summary

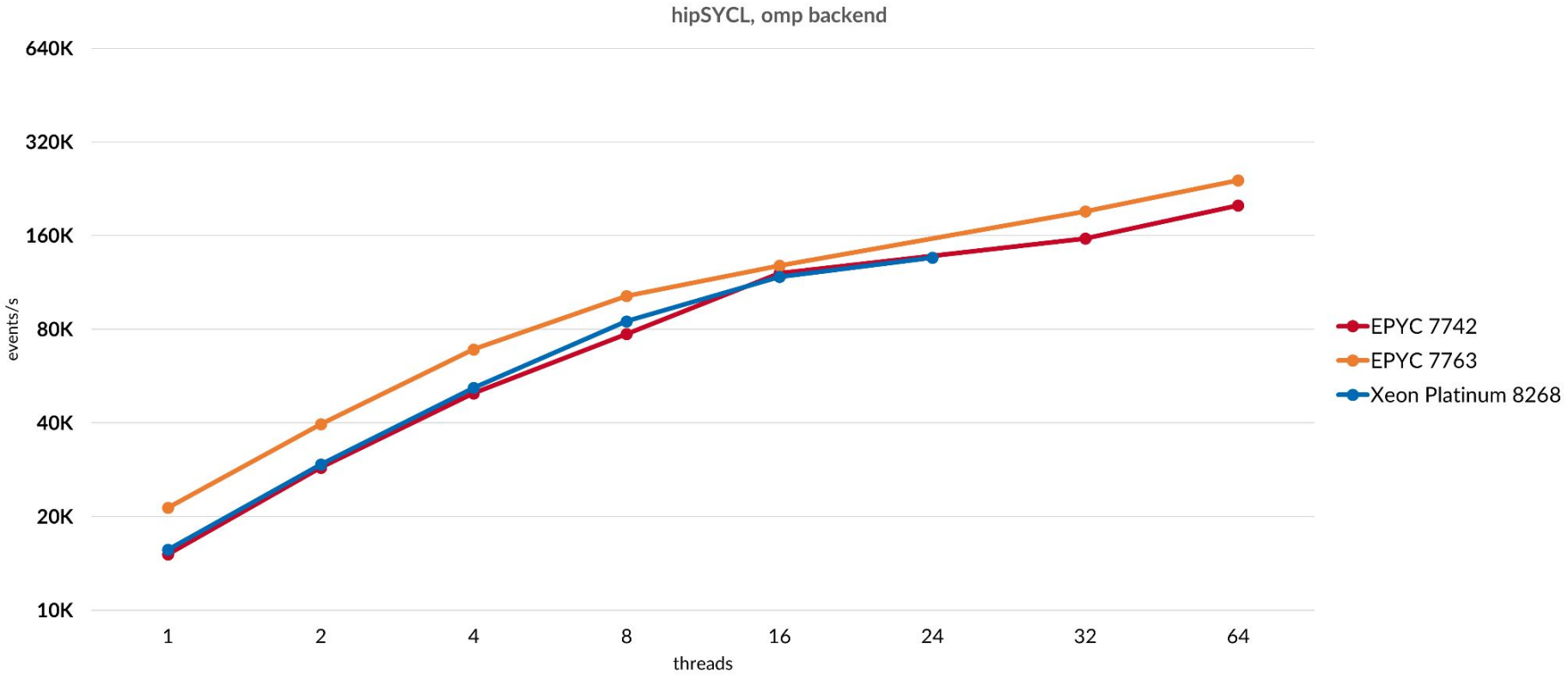
- SYCL can be used to build software for heterogeneous platform
  - With single programming model / API
- If GPU fails to deliver performance, parallel CPU version is *free*
  - Resources saved
- *FtsTrackFinder* already uses small simple data types internally
- SYCL implementation is being adapted to PandaRoot Core
  - Almost finished
  - It’s not efficient on GPU, but can work on CPU and as a demo
- Other algorithms and subsystems can follow
  - I can help with implementation

Backup

What is SYCL?

- Open standard, higher-level heterogeneous programming model - CPU, GPU, FPGA, ...
- Based on standard C++17 - without language extensions
  - Tools for C++ work with SYCL (IDEs, static analysis, linters, formatters, ...)
- Single source for host and kernel/device code
  - Kernel == any callable (function, lambda, function object)*
  - C++ functions called by kernel are also compiled as a part of device code
  - Implicit device-host separation
- Implicit memory management and task scheduling

CPU performance



GPU performance