Track reconstruction on heterogeneous architectures with SYCL

PANDA Forward Tracker

Hardware Acceleration
Lab

Bartosz Soból, MSc

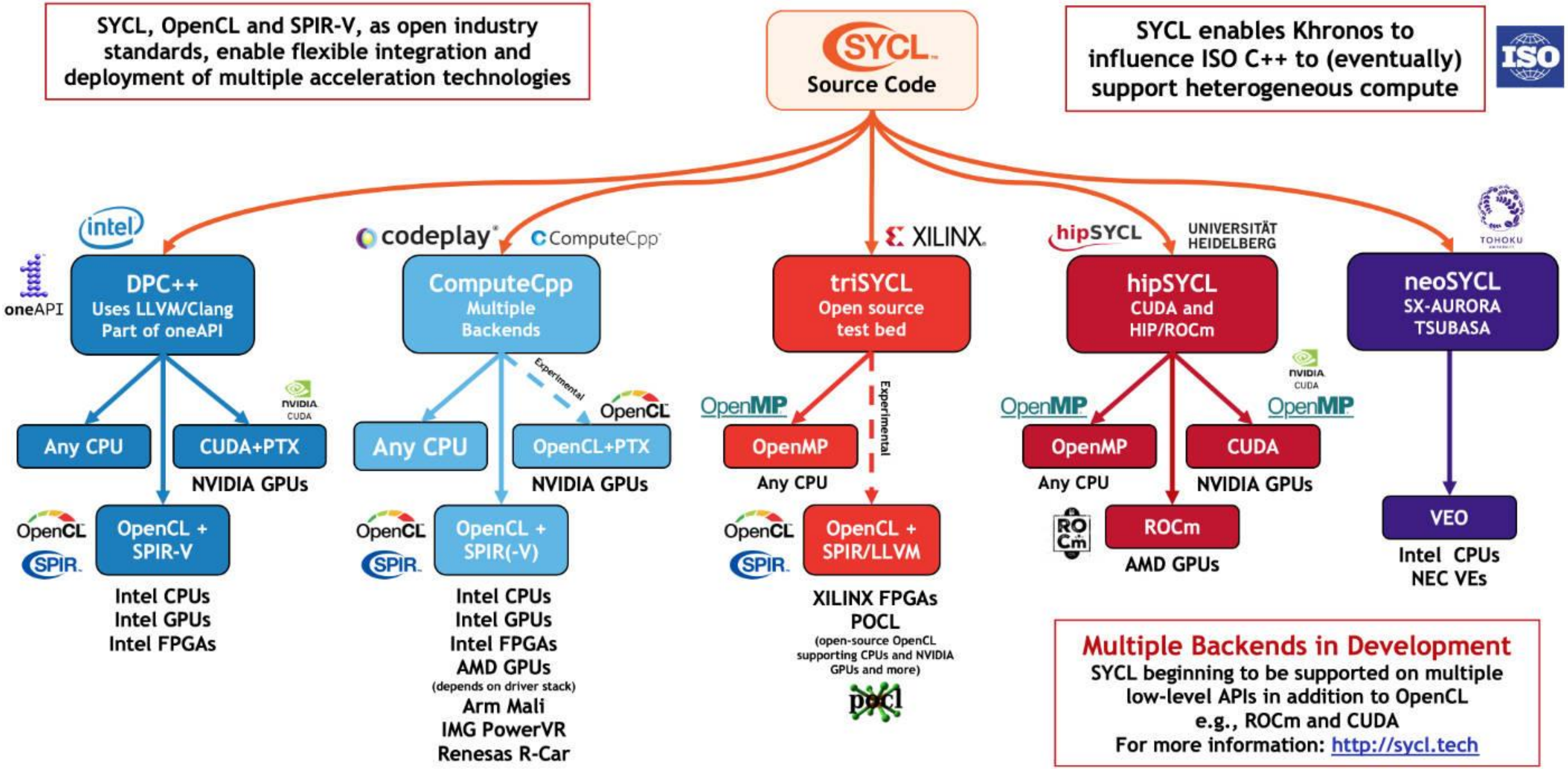
Presentation plan

1. SYCL technology overview
2. Research overview & status
3. Tracking algorithm implementation details
4. Preliminary performance results
5. Follow-up
6. Summary

What is SYCL?

- Open standard, higher-level heterogeneous programming model - CPU, GPU, FPGA, ...
- Based on standard C++11 and newer - without language extensions
- Single source for host and kernel/device code
  - Tools for C++ work with SYCL (IDEs, static analysis, linters, formatters, …)
  - 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
- OpenCL concepts reused (context, device, queue, memory layers, …)

Implementations



SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL Source Code

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

intel
oneAPI

DPC++
Uses LLVM/Clang
Part of oneAPI

Any CPU

CUDA+PTX
NVIDIA GPUs

OpenCL + SPIR-V
Intel CPUs
Intel GPUs
Intel FPGAs

codeplay ComputeCpp

ComputeCpp
Multiple Backends

Any CPU

Experimental

OpenCL+PTX
NVIDIA GPUs

OpenCL + SPIR(-V)
Intel CPUs
Intel GPUs
Intel FPGAs
AMD GPUs
(depends on driver stack)
Arm Mali
IMG PowerVR
Renesas R-Car

XILINX

triSYCL
Open source test bed

OpenMP
Any CPU

Experimental

OpenCL + SPIR/LLVM
XILINX FPGAs
POCL
(open-source OpenCL supporting CPUs and NVIDIA GPUs and more)

hipSYCL
UNIVERSITÄT HEIDELBERG

hipSYCL
CUDA and HIP/ROCm

OpenMP
Any CPU

CUDA
NVIDIA GPUs

ROCm
AMD GPUs

TOHOKU

neoSYCL
SX-AURORA TSUBASA

VEO
Intel CPUs
NEC VEs

Multiple Backends in Development

SYCL beginning to be supported on multiple low-level APIs in addition to OpenCL e.g., ROCm and CUDA

For more information: http://sycl.tech

Implementations

- Current status:
  - Intel is investing a lot in SYCL (oneAPI, DPC++)
  - Xilinx is also working on SYCL (triSYCL, fork of DPC++)
  - Ongoing ROCm/CUDA backend research project on Heidelberg University (hipSYCL)
  - Another CUDA effort by Intel
- Supported devices:
  - CPUs
  - All Intel hardware
  - AMD and NVIDIA GPUs with limitations
  - Experimentally on Xilinx FPGAs

How to try SYCL?

- hipSYCL supports the widest range of devices
  - *https://github.com/illuhad/hipSYCL/blob/develop/doc/installing.md*
- Can be installed from repository
  - CUDA must be installed separately (if needed)
- Or built from sources
  - Requirements:
    cmake, boost, llvm and clang >=8, python3, CUDA/ROCm,
    C++17 compiler

Example: vector addition

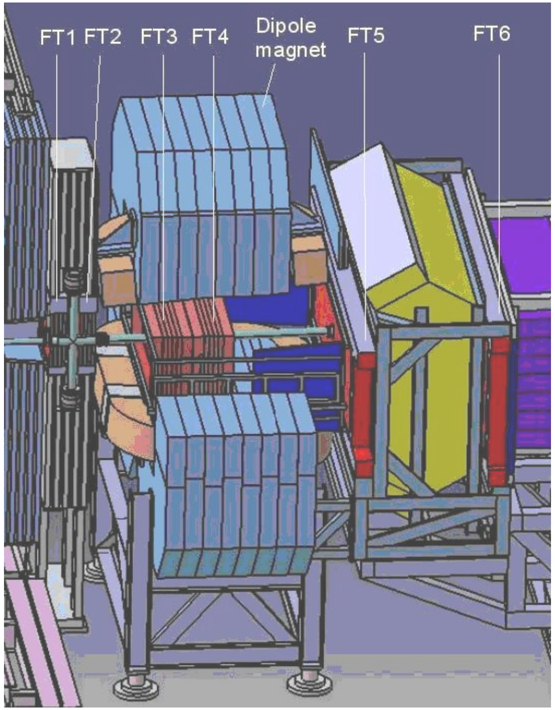
```cpp
// vadd.cpp
#include <SYCL/sycl.hpp>
#include <array>
int main() {
   constexpr int SIZE = 4;
   std::array<int, SIZE> vec_a{1, 2, 3, 4}, vec_b{5, 6, 7, 8}, vec_c{};

   sycl::queue queue{sycl::gpu_selector()};
   sycl::range<1> rng{SIZE};
   {
       sycl::buffer<int, 1> a_buff(vec_a.data(), rng);
       sycl::buffer<int, 1> b_buff(vec_b.data(), rng);
       sycl::buffer<int, 1> c_buff(vec_c.data(), rng);
       queue.submit([&](sycl::handler &cgh) {
           auto a_acc = a_buff.get_access<sycl::access::mode::read>(cgh);
           auto b_acc = b_buff.get_access<sycl::access::mode::read>(cgh);
           auto c_acc = c_buff.get_access<sycl::access::mode::write>(cgh);
           auto kernel = [=](sycl::id<1> id) {
               c_acc[id] = a_acc[id] + b_acc[id];
           };
           cgh.parallel_for<class VectorAdd>(rng, kernel);
       });
   } // vec_c == {6, 8, 10, 12}
}
```

Example: compilation (with CMake)

```cmake
# CMakeLists.txt

project(sycl_vadd LANGUAGES CXX)

set(HIPSYCL_PLATFORM cuda CACHE STRING "hipSYCL platform: cpu|rocm|cuda" FORCE)
set(HIPSYCL_GPU_ARCH sm_30 CACHE STRING "hipSYCL GPU arch. eg. gfx803 or sm_30" FORCE)

find_package(hipSYCL CONFIG REQUIRED)

add_executable(sycl_vadd vadd.cpp)
add_sycl_to_target(TARGET sycl_vadd SOURCES vadd.cpp)
```

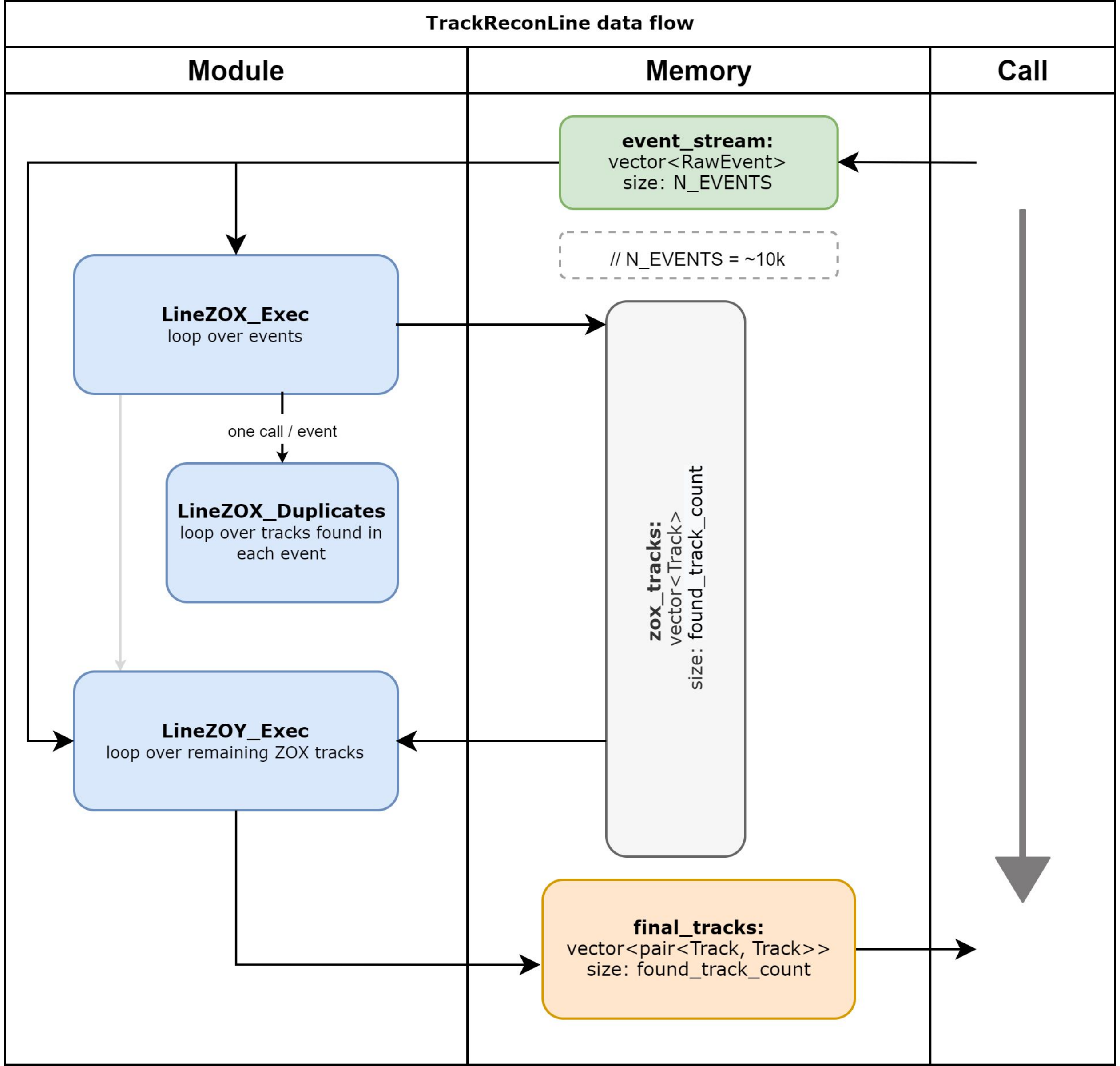
Research overview

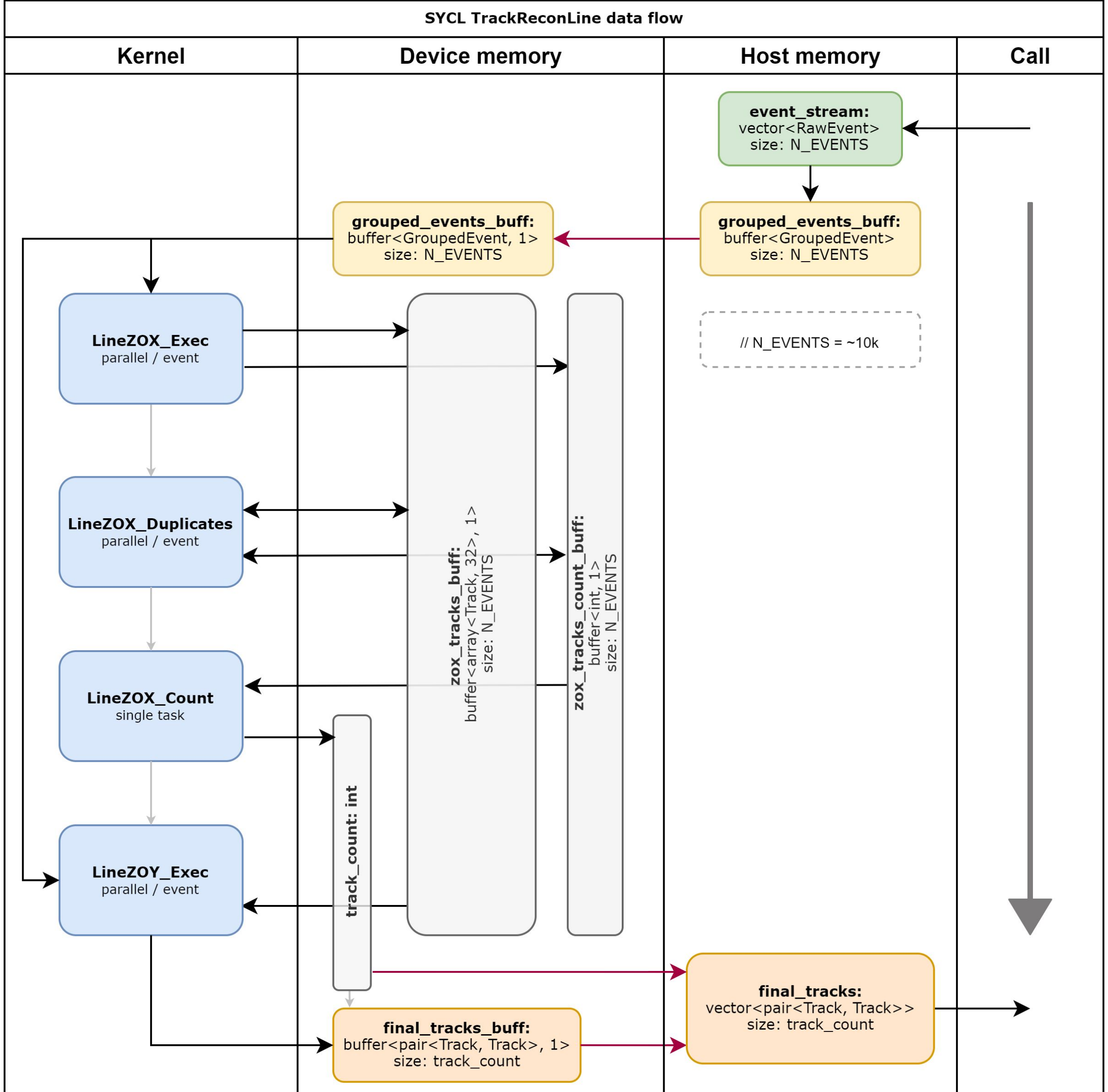
- Track reconstruction algorithm for PANDA Forward Tracker
  - 48 layers in 6 sections, total about 11k of straws
  - Two different algorithms for free particle and in EM field
- Investigating possibilities for online processing
- Using heterogeneous computing platforms
  - Multicore CPU, GPGPU, Xilinx Alveo FPGA
  - What platform and type of accelerator performs best?
- We've chosen SYCL for our software



Research status

- Reconstruction of linear parts of tracks implemented
    - In plain C++ (single threaded) and with SYCL
- SYCL implementation was tested and benchmarked
    - With two SYCL implementations - ComputeCpp and hipSYCL
    - On different Intel CPUs and CUDA GPUs - Quadro K2000 and Tesla K40
    - All with one source code


TrackReconLine data flow
Module
Memory
Call
event_stream:
vector<RawEvent>
size: N_EVENTS
// N_EVENTS = ~10k
LineZOX_Exec
loop over events
one call / event
LineZOX_Duplicates
loop over tracks found in each event
zox_tracks:
vector<Track>
size: found_track_count
LineZOY_Exec
loop over remaining ZOX tracks
final_tracks:
vector<pair<Track, Track>>
size: found_track_count

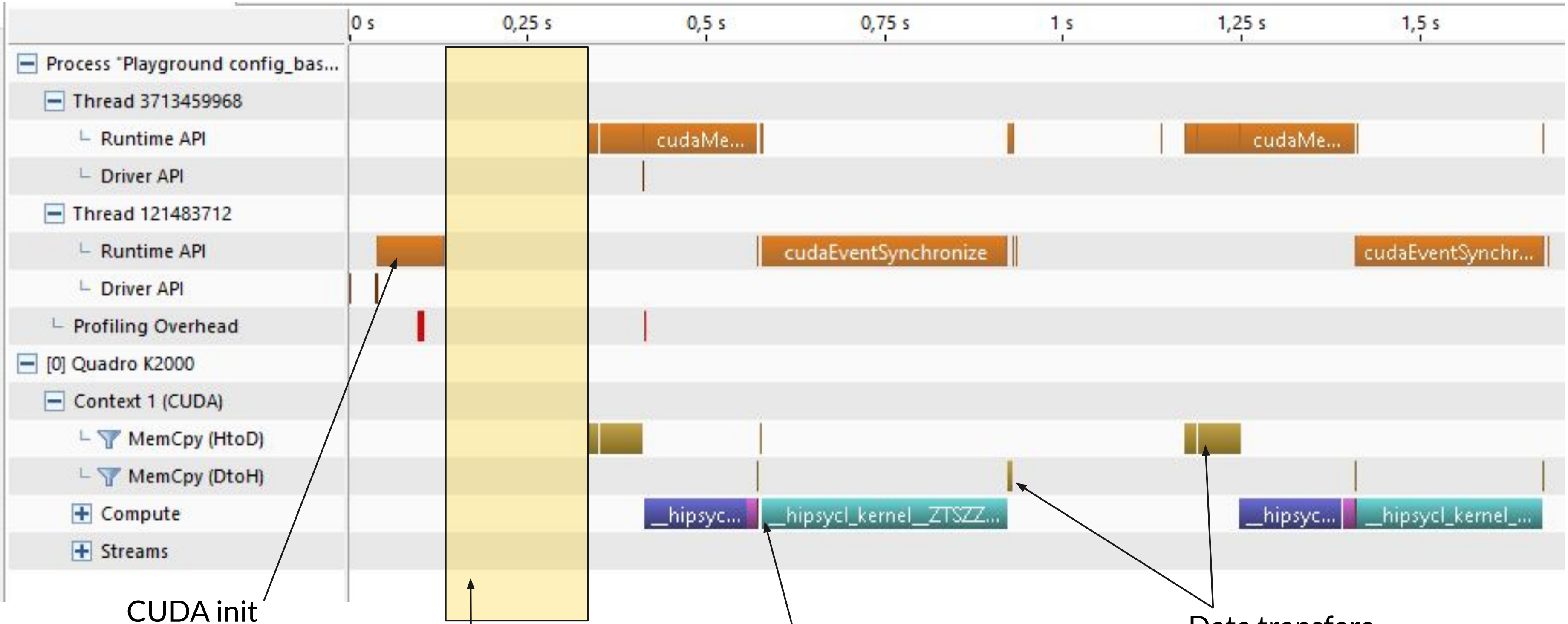

SYCL TrackReconLine data flow
Kernel
Device memory
Host memory
Call
event_stream:
vector<RawEvent>
size: N_EVENTS
grouped_events_buff:
buffer<GroupedEvent, 1>
size: N_EVENTS
grouped_events_buff:
buffer<GroupedEvent>
size: N_EVENTS
// N_EVENTS = ~10k
LineZOX_Exec
parallel / event
LineZOX_Duplicates
parallel / event
LineZOX_Count
single task
LineZOY_Exec
parallel / event
zox_tracks_buff:
buffer<array<Track, 32>, 1>
size: N_EVENTS
zox_tracks_count_buff:
buffer<int, 1>
size: N_EVENTS
track_count: int
final_tracks_buff:
buffer<pair<Track, Track>, 1>
size: track_count
final_tracks:
vector<pair<Track, Track>>
size: track_count

Performance results

data: 10 simulation files (1,3,5 muons; 0.55, 2.55, 5.55 GeV; lambda), ~100k events total

SYCL impl / backend / device type	**device**	**data x1 [ms]**	**data x2 [ms]**	**data x4 [ms]**
none / C++ single thread / CPU	i7-7700k	2230	4430	8880
hipSYCL / OpenMP / CPU	i7-7700k	4650	8800	17500
ComputeCpp / OpenCL / CPU	i7-7700k	7320	14400	28700
ComputeCpp / OpenCL / iGPU	HD 630	12000	-	-
hipSYCL / CUDA / GPU	Quadro K2000	21380	38500	**-**
hipSYCL / CUDA / GPU	Tesla K40	12900	20400	35800

Quadro K2000 profiling



CUDA init

CPU code memory alloc

Kernels processing

Data transfers

GPU performance

- Execution time on Tesla K40 is about 40% better then on Quadro K2000
  - But it’s still the slower than on tested CPUs
- On Quadro K2000
  - Data transfers take almost half of the single core CPU execution time
  - Single event processing time is ~5-6 times worse then on CPUs
- Kernel code definitely needs optimizations for GPU

CPU performance

- Processing of 1 event takes on average 0.025ms (on intel 7700k CPU)
- When it’s forced to take 0.1ms on average:

```
auto start = clock::now();
while(duration_cast<microseconds>(clock::now() - start).count() < 75){ }
```

- hipSYCL / OpenMP / CPU - **5900 ms** / 1x data
  - 4600 ms without *artificial* complexity
- C++ single thread - **15000 ms** / 1x data
  - 2500 ms (at most) without *artificial* complexity

Performance results

- Implemented part of the algorithm (linear, FT12,56) turned out no to be suitable for acceleration on tested GPUs (in current form at least)
    - Single event processing is cheap, but contains short loops, and branches
        - Hard to decompose into smaller/nested kernels
        - Not easily parallelizable
    - GPU execution may not be faster than CPU even after optimisations
    - But it may change drastically when other parts of the algorithm are implemented
    - Also, more modern GPGPUs may behave better in this kind of task

Next steps

- **Implementation of the rest of the tracking algorithm**
- **Optimizations of SYCL kernels GPUs**
  - More benchmarking on different devices
- **Explore FPGA possibilities with SYCL**
- Test also using Intel’s OneAPI/DPC++

Summary

- We try to implement online track reconstruction on heterogeneous platform
- Using modern and developing technology - SYCL
- As a result we have first pieces of working **portable** code
- With some performance issues
- But we now know our tools and have the ability to test it on different hardware and software in order to find the best solution
- A lot of work ahead