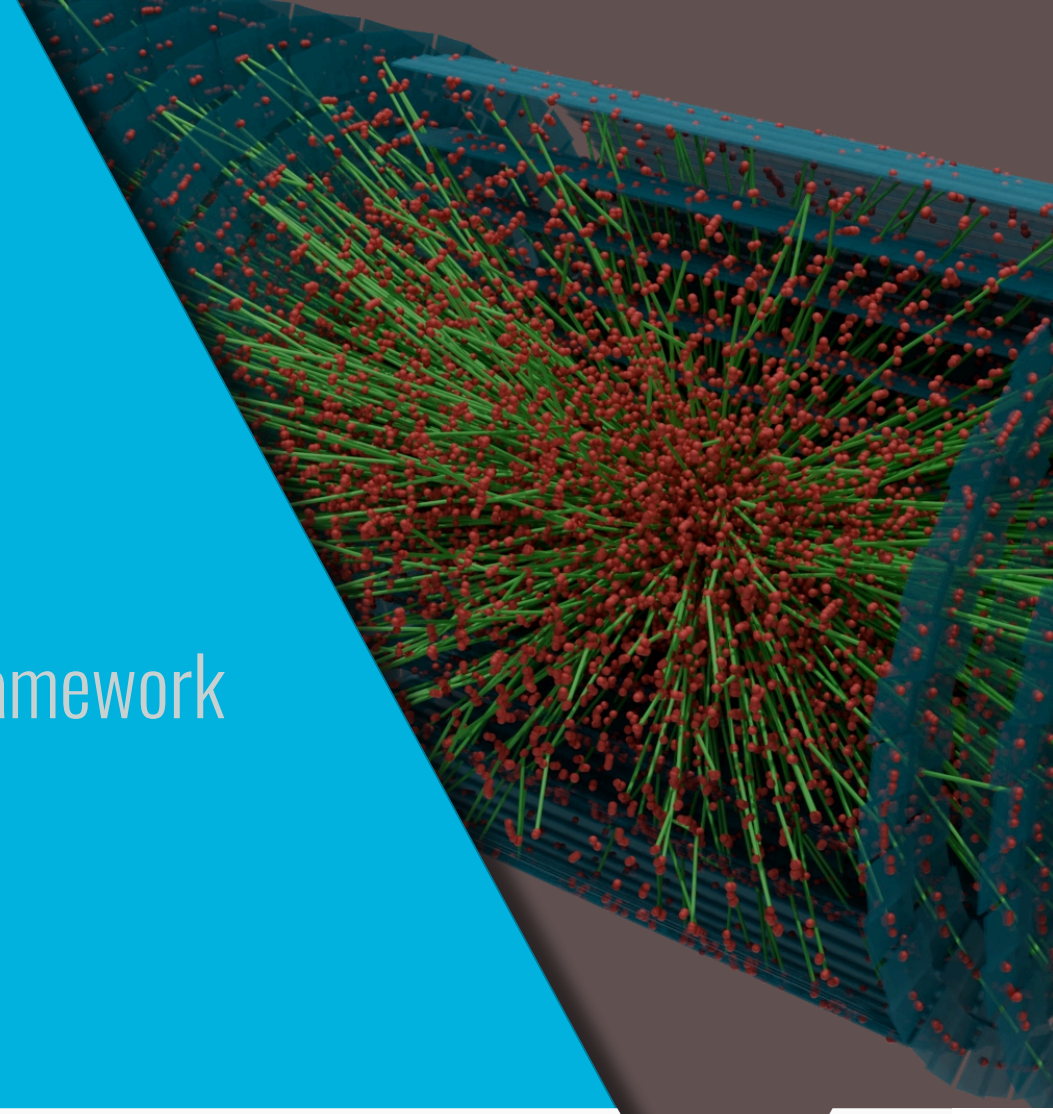
traccc

Integrating the Alpaka framework

Ryan Cross
SWIFT-HEP08 + ExaTEPP
2024/11/11



WARWICK
THE UNIVERSITY OF WARWICK

Overview

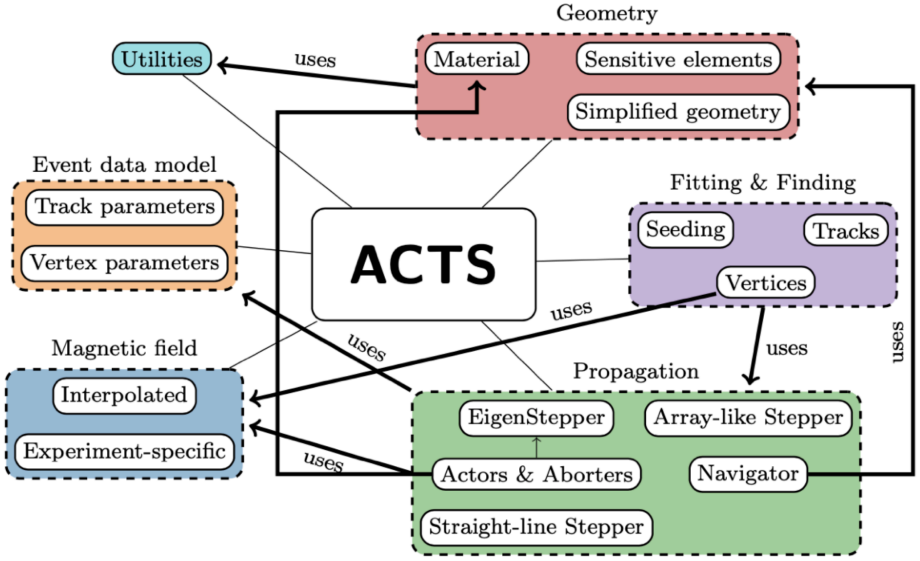
This talk will cover:

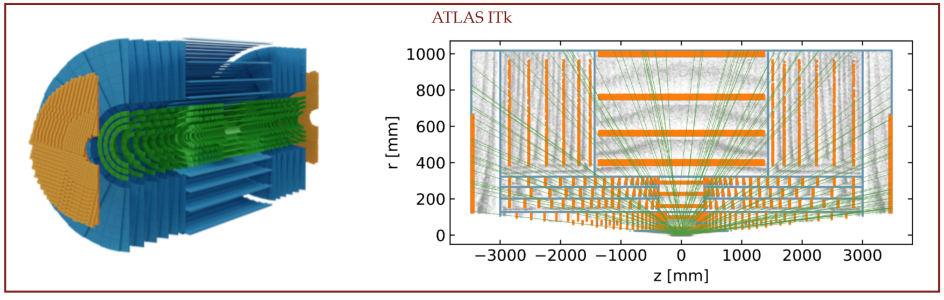
1. **traccc.**
2. **Cross-Platform Abstraction Libraries.**
3. **Where We Are**
4. **Current Work**
5. **What comes next?**

A Common Tracking Software

ACTS is a generic, experiment independent framework/software toolkit, written in C++. Through it, you can get algorithms for track reconstruction that can be used in any experiment, agnostic of any technical details (detector tech, design and event processing framework).

It has been designed in a thread-safe manner, with support for parallel code execution and optimised data structures for speeding up the many linear algebra operations used throughout the code base.



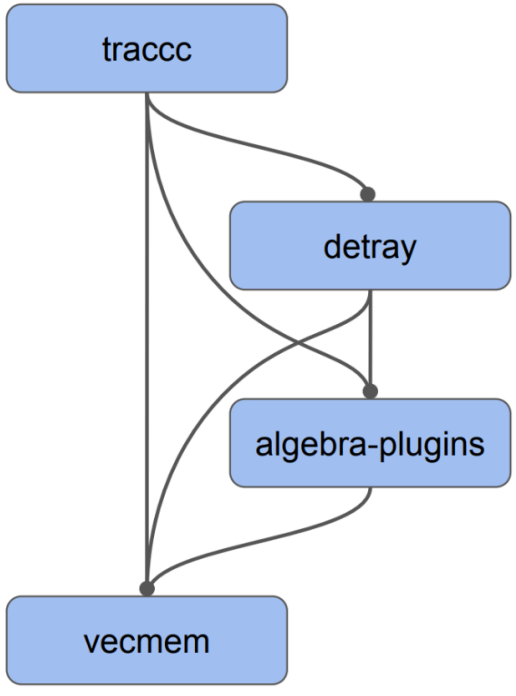


ACTS R&D Projects

Many of the core algorithms in ACTS have been ported to CUDA and SYCL, but there is a limit as to how far this can go. Full offloading is difficult, with some of the event data model and geometry not being the most GPU-friendly.

To tackle this, ACTS has launched several R&D projects:

- traccc - Tracking Algorithms on the GPU.
- detray - A GPU based Geometry Builder.
- algebra-plugin - Provides varying algebra plugins for the other projects.
- vecmem - A GPU Memory Management Tool for the other projects.

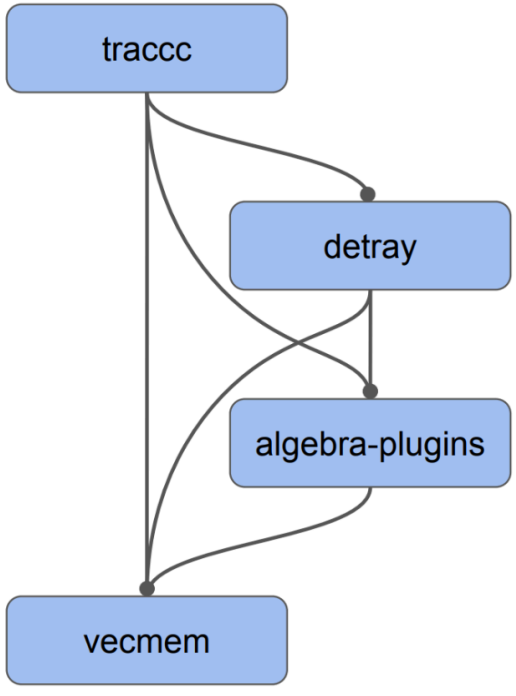


ACTS R&D Projects

Many of the core algorithms in ACTS have been ported to CUDA and SYCL, but there is a limit as to how far this can go. Full offloading is difficult, with some of the event data model and geometry not being the most GPU-friendly.

To tackle this, ACTS has launched several R&D projects:

- traccc - Tracking Algorithms on the GPU.
- detray - A GPU based Geometry Builder.
- algebra-plugin - Provides varying algebra plugins for the other projects.
- vecmem - A GPU Memory Management Tool for the other projects.



traccc specifically, is aiming to establish a sensible event data model and algorithms that are able to exploit parallelisation architecture, whilst relying heavily on the other projects.

Cross-Platform Abstraction - What?

There are a few abstraction approaches worth talking about in the context of traccc. Whilst the broad goal of allowing a single code base to target many different accelerator backends is the same, the approach and technical details differ.

Cross-Platform Abstraction - What?

There are a few abstraction approaches worth talking about in the context of traccc. Whilst the broad goal of allowing a single code base to target many different accelerator backends is the same, the approach and technical details differ.

- **SYCL** is a higher level programming model, developed by the Khronos group (OpenCL/OpenGL/Vulkan and more). It defines an abstraction layer that enables code for heterogeneous processors via a 'single-source' style in standard C++. Supports many backends: CUDA, AMD GPUs, Intel GPUs, OpenMP, MPI, Vulkan, `std::thread`, OpenCL and more.

Cross-Platform Abstraction - What?

There are a few abstraction approaches worth talking about in the context of traccc. Whilst the broad goal of allowing a single code base to target many different accelerator backends is the same, the approach and technical details differ.

- **SYCL** is a higher level programming model, developed by the Khronos group (OpenCL/OpenGL/Vulkan and more). It defines an abstraction layer that enables code for heterogeneous processors via a 'single-source' style in standard C++. Supports many backends: CUDA, AMD GPUs, Intel GPUs, OpenMP, MPI, Vulkan, `std::thread`, OpenCL and more.

- **Kokkos** is C++ based programming model, which provides methods that abstract away details of parallel execution and memory management, such that code can be written for many shared-memory programming models in a unifed way. Supports CUDA, HIP, SYCL, HPX, OpenMP and `std::thread`.

Cross-Platform Abstraction - What?

There are a few abstraction approaches worth talking about in the context of traccc. Whilst the broad goal of allowing a single code base to target many different accelerator backends is the same, the approach and technical details differ.

- **SYCL** is a higher level programming model, developed by the Khronos group (OpenCL/OpenGL/Vulkan and more). It defines an abstraction layer that enables code for heterogeneous processors via a 'single-source' style in standard C++. Supports many backends: CUDA, AMD GPUs, Intel GPUs, OpenMP, MPI, Vulkan, `std::thread`, OpenCL and more.

- **Kokkos** is C++ based programming model, which provides methods that abstract away details of parallel execution and memory management, such that code can be written for many shared-memory programming models in a unifed way. Supports CUDA, HIP, SYCL, HPX, OpenMP and `std::thread`.

- **alpaka** is a header-only C++ 20 abstraction library for accelerator development. It aims to provide performance portability across a range of accelerators through the abstraction of the underlying levels of parallelism. Support CUDA, OpenMP, `std::thread`, TBB, HIP and OpenAcc.

Cross-Platform Abstraction - How?

Despite having differing ways of interacting with them, advertising themselves differently and more...they all have the same objective: **Write your code once**, and through the libraries abstraction methods, end up with a code base that supports a variety of accelerator backends.

The specific interface to achieve this differs between each of the options, but some broad steps are the same.

Cross-Platform Abstraction - How?

Despite having differing ways of interacting with them, advertising themselves differently and more...they all have the same objective: **Write your code once**, and through the libraries abstraction methods, end up with a code base that supports a variety of accelerator backends.

The specific interface to achieve this differs between each of the options, but some broad steps are the same.

Get an accelerator device:

```
accelerator = getAcceleratorDevice();
queue = getDeviceQueue(accelerator);
```

Define an operation for the device to perform:

```
job = [](auto accelerator, auto config, auto items) {
    auto item = items[getThreadIndex()];
    ...
};
```

Run the jobs in parallel:

```
queue.submit(job, configuration, items);
queue.wait();
```

Why alpaka?

I've just outlined three projects that support the "write once, support many" paradigm, and both SYCL and Kokkos are already implemented in traccc, with differing levels of functionality. So why a third?

alpaka was chosen as a possible candidate for a few reasons:

- **Simplicity**: alpaka is a lightweight, header-only library, which makes integration into traccc very easy, as well as it being written in the same modern C++20 as traccc/acts.
- **Familiarity**: The alpaka abstraction model is very similar to the CUDA grid-blocks-thread model, making writing code for alpaka simple, and familiar for those with CUDA experience, whilst also providing a CPU and non-CUDA based implementation.
- **Community Support**: alpaka has been used extensively at CMS, including in `cms-sw` and their HLT achieving performance close to that of the native CUDA codebase, from a single source code that can be utilised on many devices.

Completed Work

The first steps around integration of alpaka in traccc were performed by Stewart Martin-Haugh, as part of a PR in Jan 23: PR #300.

Completed Work

The first steps around integration of alpaka in traccc were performed by Stewart Martin-Haugh, as part of a PR in Jan 23: PR #300.

I then built upon this base to add the first tracking code, to add a spacepoint binning algorithm. This algorithm was a reasonable starting point, fairly self-contained and easy to implement. This was added in PR #431.

Completed Work

The first steps around integration of alpaka in traccc were performed by Stewart Martin-Haugh, as part of a PR in Jan 23: PR #300.

I then built upon this base to add the first tracking code, to add a spacepoint binning algorithm. This algorithm was a reasonable starting point, fairly self-contained and easy to implement. This was added in PR #431.

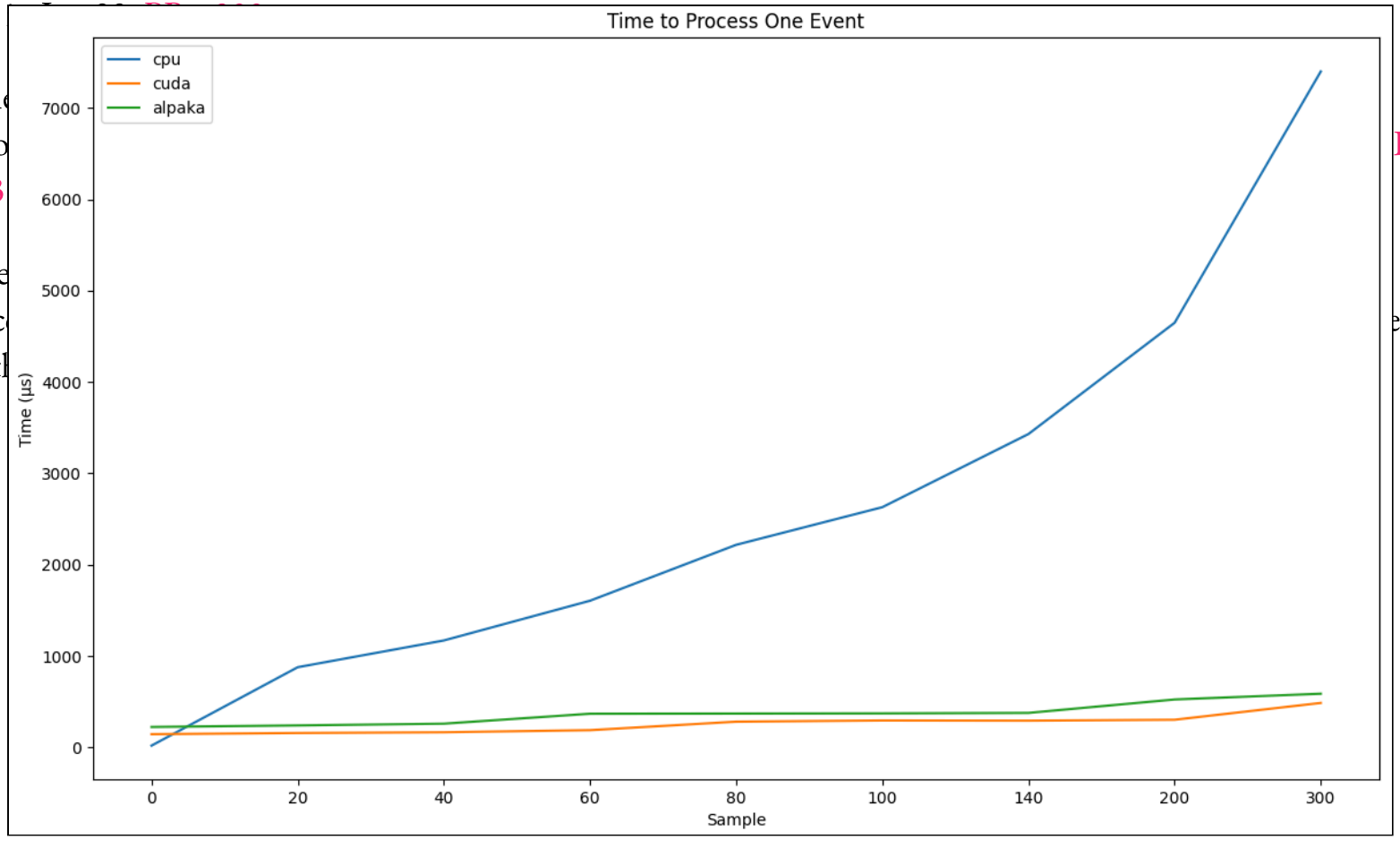
The spacepoint binning gave me a first look at development with Alpaka, as well as developing inside of traccc/ACTS. My older slides, given at a UK SWIFT-HEP / GRIDPP meeting, give a bit of a better overview of that work, as well as some more basic comparisons of Alpaka vs CUDA.

Completed Work

The first steps around integration of alpaka in traccc were performed by Stewart Martin-Haugh, as part of a PR

I th
algo
#43

The
trac
of t



Completed Work

Following the spacepoint binning, the next portion of completed work was around seeding, which build from the spacepoint binning.

This comprised of a lot more algorithms, compared to the relatively self-contained and small binning work, but the end result is something closer to Physics, and as such, means we could compare results against the native CUDA version more easily.

Completed Work

Following the spacepoint binning, the next portion of completed work was around seeding, which build from the spacepoint binning.

This comprised of a lot more algorithms, compared to the relatively self-contained and small binning work, but the end result is something closer to Physics, and as such, means we could compare results against the native CUDA version more easily.

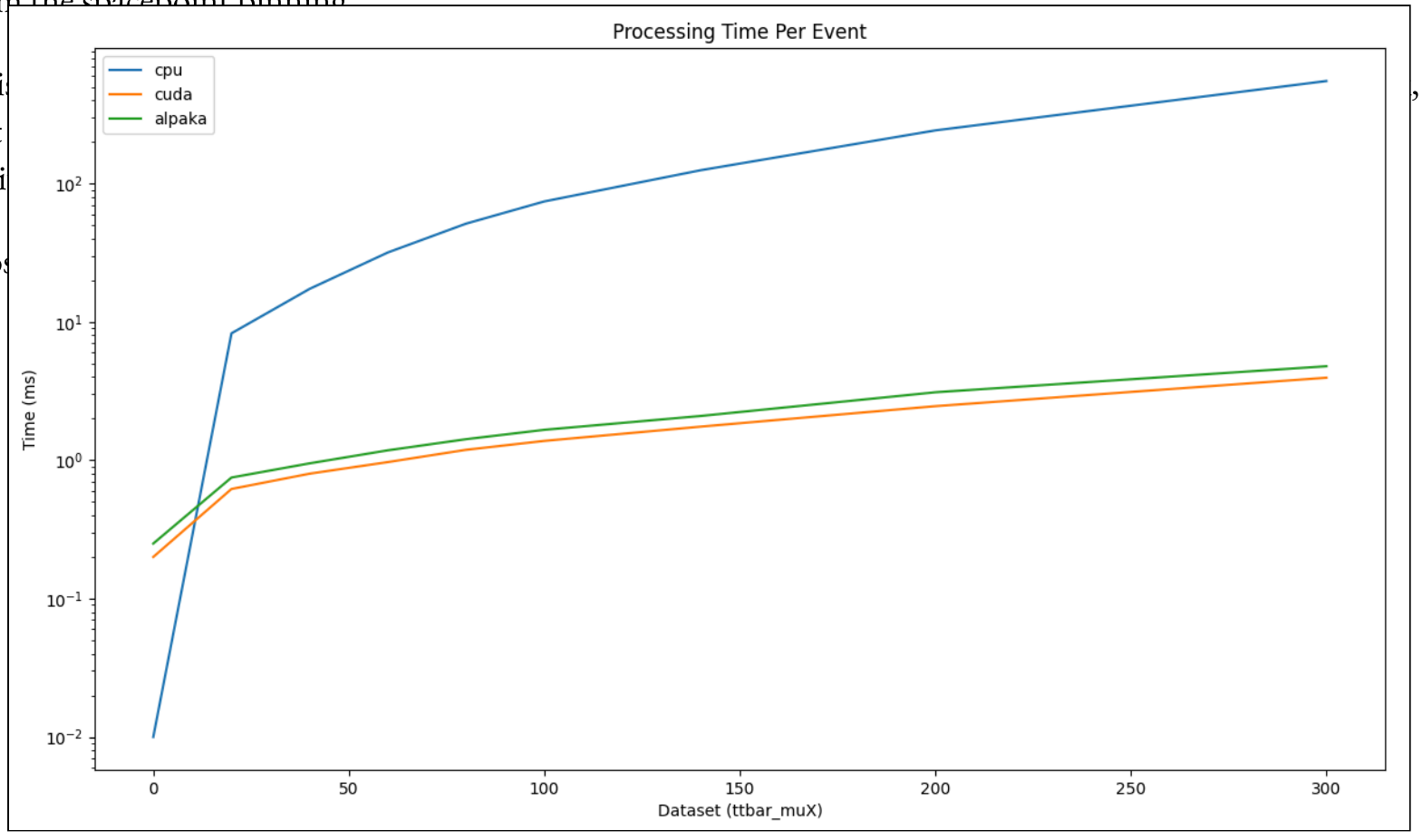
Most of that work was merged as part of PR (#451) last year.

Completed Work

Following the spacepoint binning, the next portion of completed work was around seeding, which build from the spacepoint binning

This
but
nati

Mos

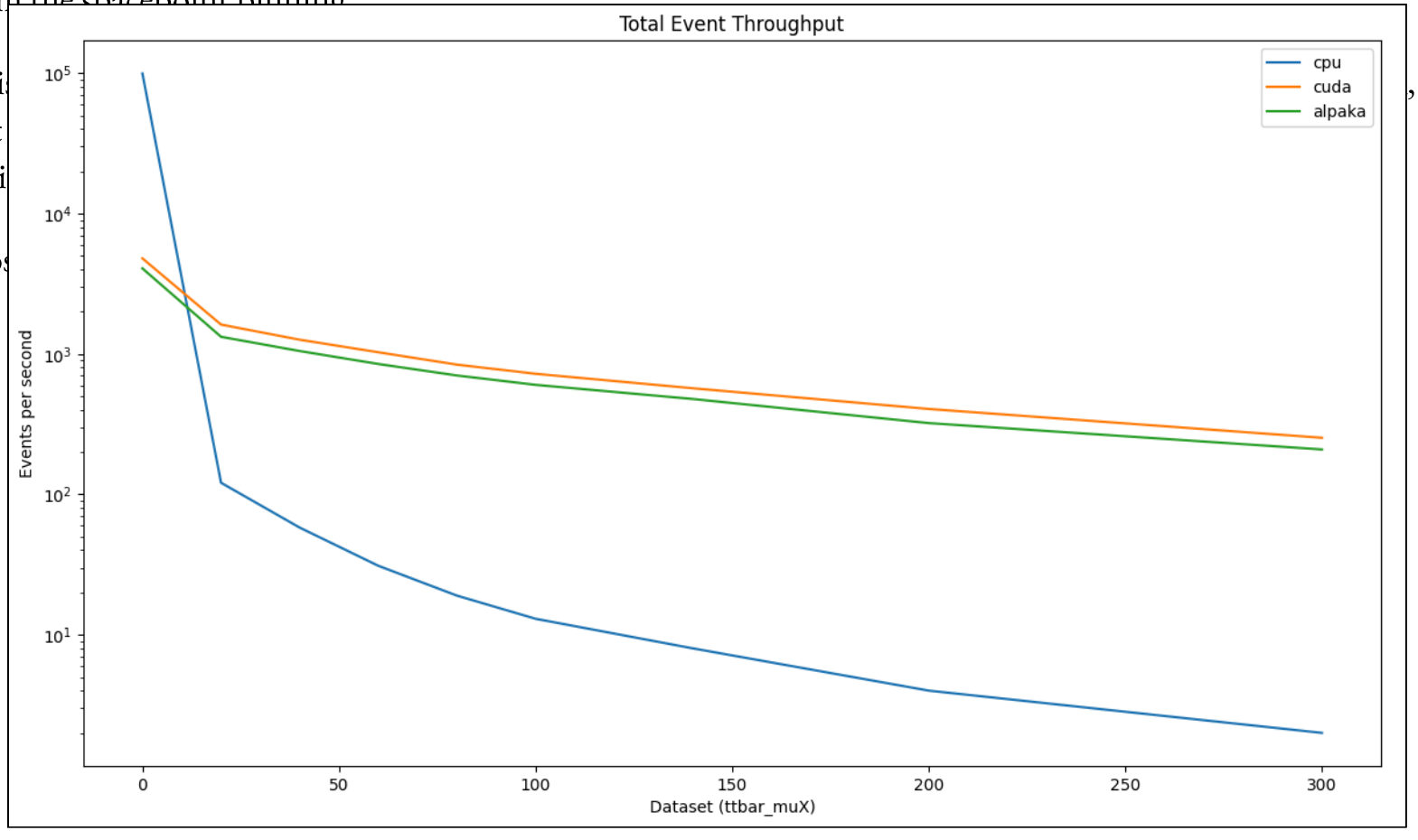


Completed Work

Following the spacepoint binning, the next portion of completed work was around seeding, which build from the spacepoint binning

Thi , but nati

Mos



Current Work

So, since the last update there has been a few main pieces of work from Stewart and myself, which I'll go into a bit more detail on.

They are:

Current Work

So, since the last update there has been a few main pieces of work from Stewart and myself, which I'll go into a bit more detail on.

They are:

- Validate the full tracking chain. The last slides I showed took us up to the point where we had throughput performance for almost the full chain, but not the final track finding and fitting steps.

Current Work

So, since the last update there has been a few main pieces of work from Stewart and myself, which I'll go into a bit more detail on.

They are:

- Validate the full tracking chain. The last slides I showed took us up to the point where we had throughput performance for almost the full chain, but not the final track finding and fitting steps.
- Improving the robustness of the current code. Whilst the current code works well enough and compares favourably to the CUDA code, we still are not able to get a complete understanding of the multi-threaded performance of Alpaka, due to various issues.

Current Work

So, since the last update there has been a few main pieces of work from Stewart and myself, which I'll go into a bit more detail on.

They are:

- Validate the full tracking chain. The last slides I showed took us up to the point where we had throughput performance for almost the full chain, but not the final track finding and fitting steps.
- Improving the robustness of the current code. Whilst the current code works well enough and compares favourably to the CUDA code, we still are not able to get a complete understanding of the multi-threaded performance of Alpaka, due to various issues.
- Verifying Alpaka with HIP. When the main draw of using an abstraction library is multi-vendor support, that support needs testing and any HIP-specific changes implementing.

Each of these pieces of work are at different stages of completion, and I'll go into a touch more detail on them each now.

HIP

Stewart has been championing the work of testing Alpaka with HIP, to verify the code running on an AMD GPU.

We've hit a lot of issues on this path, mostly based around Thrust, which is a C++ parallel algorithm library...but written by Nvidia for CPUs or CUDA.

HIP

Stewart has been championing the work of testing Alpaka with HIP, to verify the code running on an AMD GPU.

We've hit a lot of issues on this path, mostly based around Thrust, which is a C++ parallel algorithm library...but written by Nvidia for CPUs or CUDA.

Thrust is used quite heavily in a few places, usually for sorting and filtering of arrays of data on the GPU. There is a potential solution though, which is the rocThrust library, which should work on AMD GPUs. However, we have hit a lot of issues in getting the library to be picked up and used throughout traccc.

One of the complications is that Thrust is not just used in traccc, but also in part in detray, vecmem and more. That means any solution we come up with either needs to be duplicated across a few repos, or a fix needs to be done in such a way that it is picked up and used automatically by the downstream projects.

HIP

Stewart has been championing the work of testing Alpaka with HIP, to verify the code running on an AMD GPU.

We've hit a lot of issues on this path, mostly based around Thrust, which is a C++ parallel algorithm library...but written by Nvidia for CPUs or CUDA.

Thrust is used quite heavily in a few places, usually for sorting and filtering of arrays of data on the GPU. There is a potential solution though, which is the rocThrust library, which should work on AMD GPUs. However, we have hit a lot of issues in getting the library to be picked up and used throughout traccc.

One of the complications is that Thrust is not just used in traccc, but also in part in detray, vecmem and more. That means any solution we come up with either needs to be duplicated across a few repos, or a fix needs to be done in such a way that it is picked up and used automatically by the downstream projects.

Recently, Fabrice Le Goff has started working on EFTracking, specifically on AMD GPUs performance with traccc. Their work (from my understanding) is not Alpaka or similar specific, but the hope is that any solution around Thrust should work for multiple approaches, so we can hopefully work together to reach a sensible result.

Alpaka Robustness Testing

That leads nicely into to improving the robustness of the Alpaka code.

Right now, all the code shown works and runs for many 100s of events, but there is intermittent memory access issues, at least in the Alpaka-CUDA version.

Alpaka Robustness Testing

That leads nicely into to improving the robustness of the Alpaka code.

Right now, all the code shown works and runs for many 100s of events, but there is intermittent memory access issues, at least in the Alpaka-CUDA version.

Debugging of these errors has been on-and-off over the past few months, but I've managed to narrow it down to a single file, and the error only occurs in cases where multiple events are run in sequence, which certainly points towards a certain class of memory errors.

I likely just need a few days to do a deep-dive on the debugging and try and check the state of all the variables going into the seeding code.

Track Finding + Fitting

At the previous SWIFT-HEP meeting, I had implemented algorithms up to the point of having `Prototrack` objects.

I have since ported over the remaining bits of the CUDA code, and have a complete tracking chain written fully in Alpaka.

This track finding and fitting code did come with its own set of challenges:

Track Finding + Fitting

At the previous SWIFT-HEP meeting, I had implemented algorithms up to the point of having `Prototrack` objects.

I have since ported over the remaining bits of the CUDA code, and have a complete tracking chain written fully in Alpaka.

This track finding and fitting code did come with its own set of challenges:

- The track finding and fitting code is the place (at least in traccc) that most heavily uses Thrust as part of its algorithms. That doesn't mean much for the CUDA-Alpaka version, but may be a pain-point going forward.

Track Finding + Fitting

At the previous SWIFT-HEP meeting, I had implemented algorithms up to the point of having `Prototrack` objects.

I have since ported over the remaining bits of the CUDA code, and have a complete tracking chain written fully in Alpaka.

This track finding and fitting code did come with its own set of challenges:

- The track finding and fitting code is the place (at least in traccc) that most heavily uses Thrust as part of its algorithms. That doesn't mean much for the CUDA-Alpaka version, but may be a pain-point going forward.
- Some of the more sophisticated objects / data-structures in these algorithms need a bit of massaging to convince Alpaka that they are `is_trivially_copyable`. This isn't difficult, just a trait, but a bit of a pain when developing to spot.

Track Finding + Fitting

At the previous SWIFT-HEP meeting, I had implemented algorithms up to the point of having `Prototrack` objects.

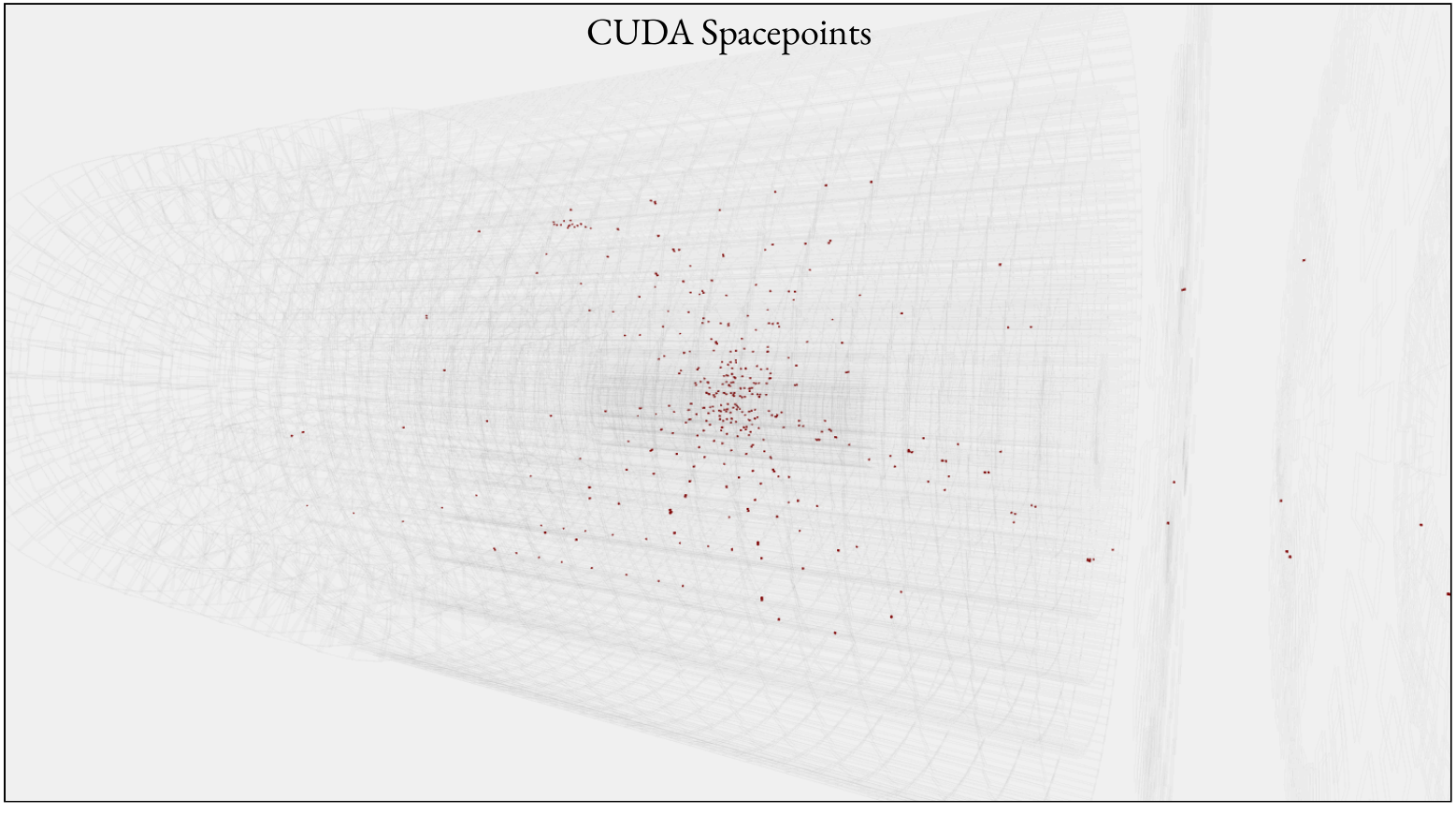
I have since ported over the remaining bits of the CUDA code, and have a complete tracking chain written fully in Alpaka.

This track finding and fitting code did come with its own set of challenges:

- The track finding and fitting code is the place (at least in traccc) that most heavily uses Thrust as part of its algorithms. That doesn't mean much for the CUDA-Alpaka version, but may be a pain-point going forward.
- Some of the more sophisticated objects / data-structures in these algorithms need a bit of massaging to convince Alpaka that they are `is_trivially_copyable`. This isn't difficult, just a trait, but a bit of a pain when developing to spot.
- On the better side though, this is the end of the chain, so comparing between CUDA, CPU and Alpaka is now much more intuitive. An issue early on was that comparing the spacepoints or similar is very abstract, and also difficult due to the numbers of spacepoints in an event. Whereas comparing tens of tracks is easily doable in an event display.

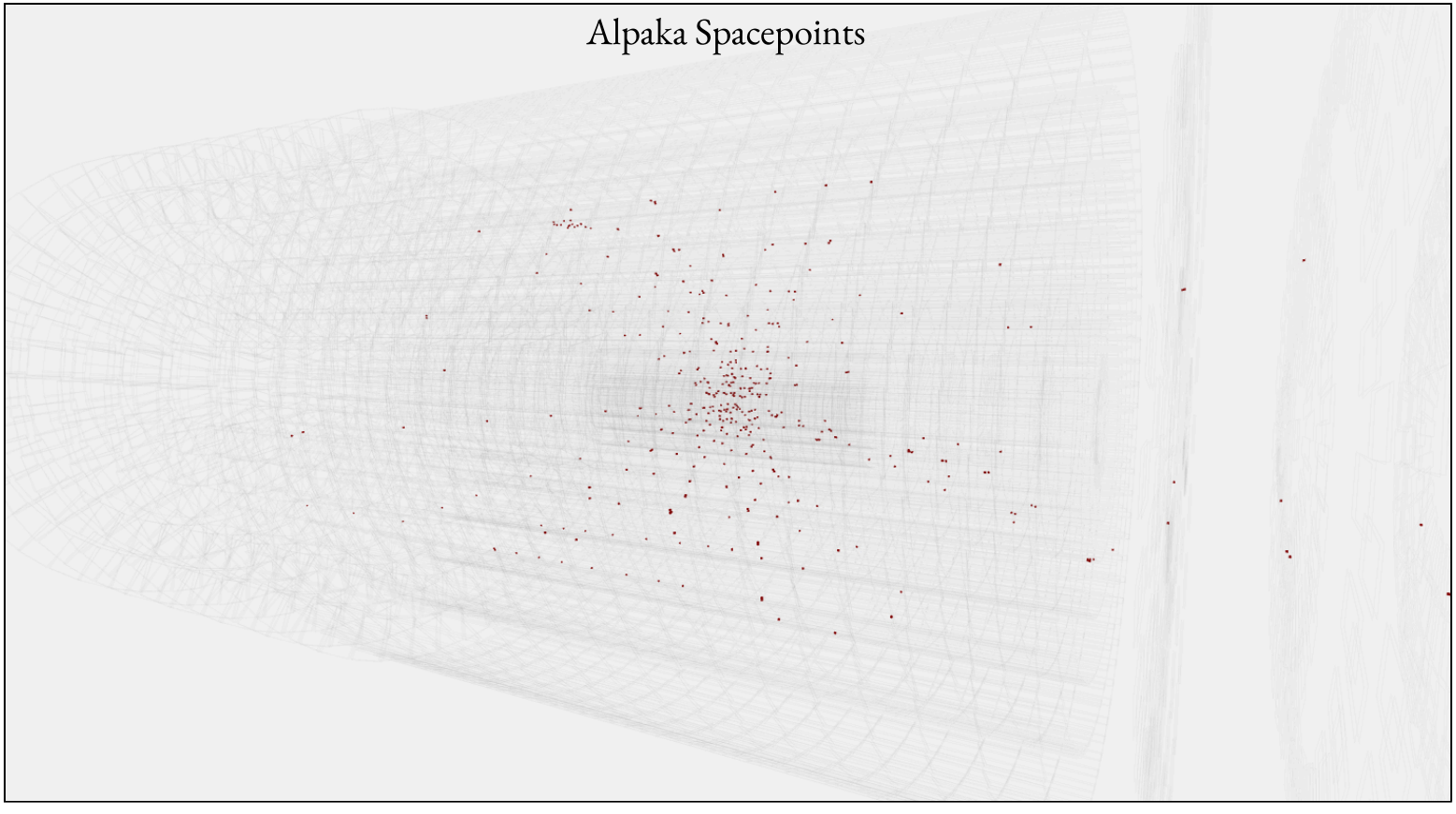
Some Quick Comparisons

A little while back, I adapted my LArTPC event display to work with traccc / detray, such that I could compare the reconstruction performance in real-time. Here are a few comparisons between the CUDA and Alpaka produced states:



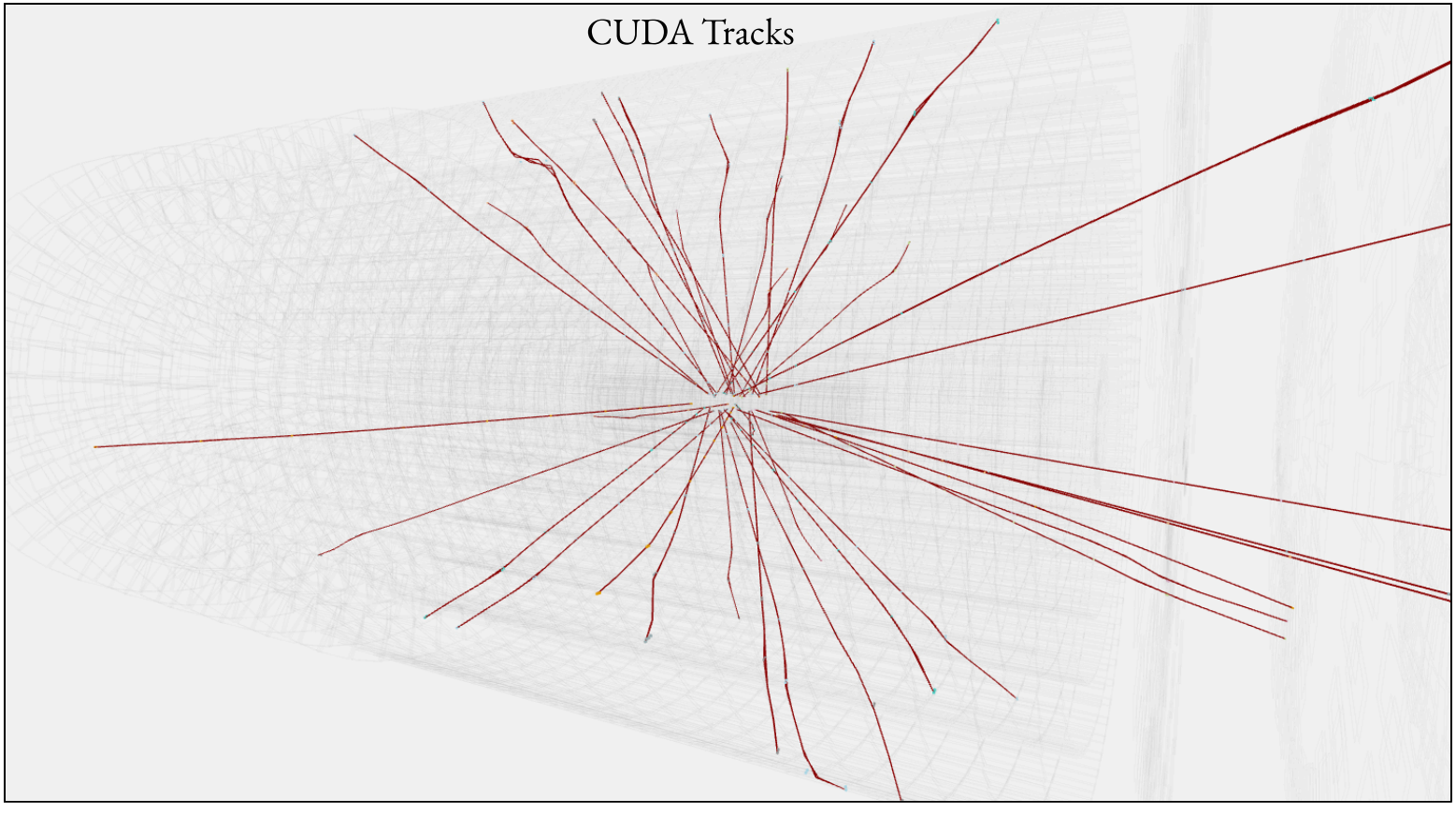
Some Quick Comparisons

A little while back, I adapted my LArTPC event display to work with traccc / detray, such that I could compare the reconstruction performance in real-time. Here are a few comparisons between the CUDA and Alpaka produced states:



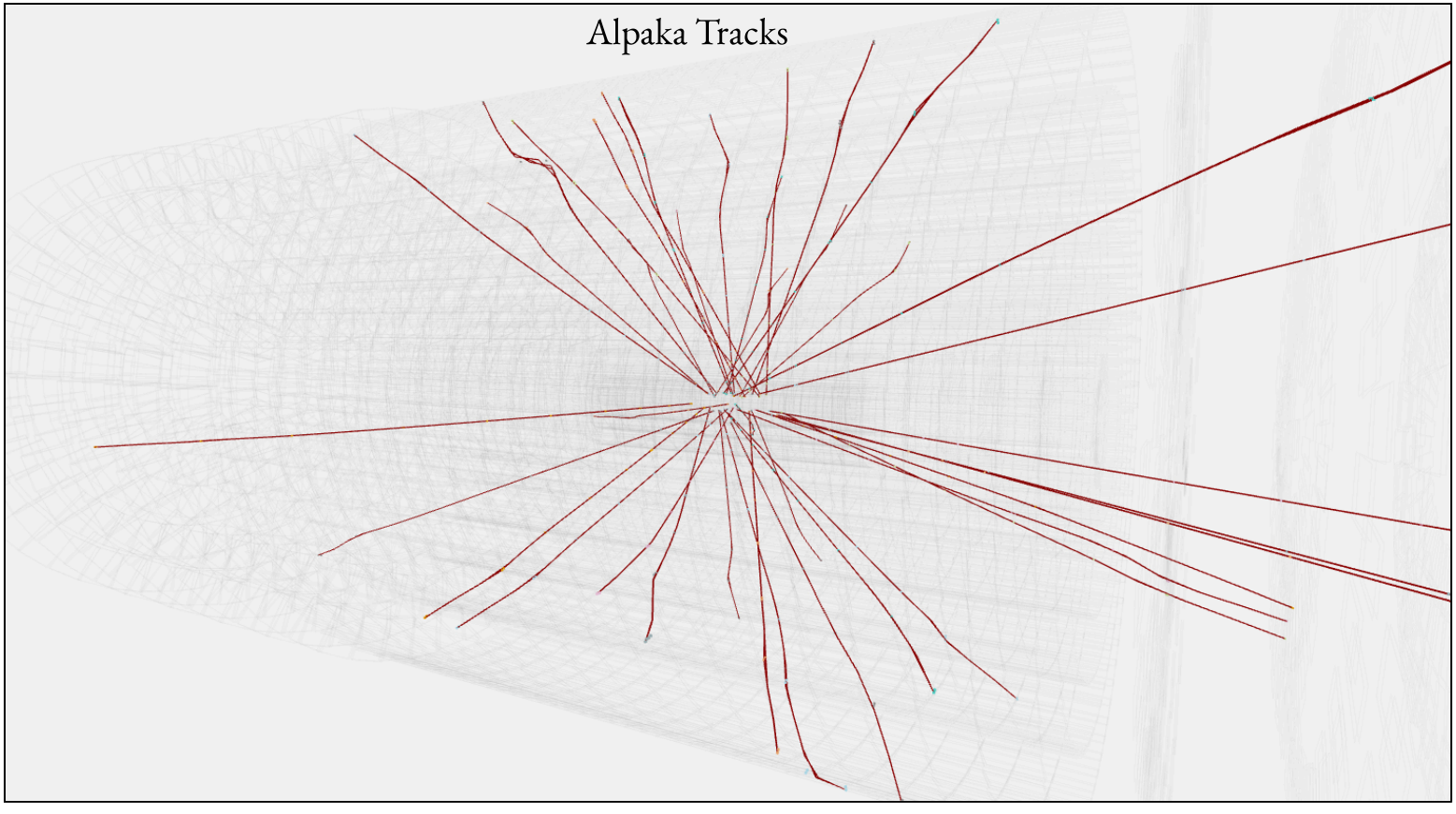
Some Quick Comparisons

A little while back, I adapted my LArTPC event display to work with traccc / detray, such that I could compare the reconstruction performance in real-time. Here are a few comparisons between the CUDA and Alpaka produced states:



Some Quick Comparisons

A little while back, I adapted my LArTPC event display to work with traccc / detray, such that I could compare the reconstruction performance in real-time. Here are a few comparisons between the CUDA and Alpaka produced states:



What is next?

Few things to do:

- Continue with the three mentioned bits of work.

What is next?

Few things to do:

- Continue with the three mentioned bits of work.
- Consider more intelligent ways of dealing with the multiple accelerator backends.
  - That is, intelligent ways of dealing with the few bits of compiler specific code: `vecmem/cuda/X.h` vs `vecmem/hip/X.h`, or building sensible `workDivs` to deal with the hardware differences (Fixed vs variable warp size, CPU core count differences etc.)

What is next?

Few things to do:

- Continue with the three mentioned bits of work.
- Consider more intelligent ways of dealing with the multiple accelerator backends.
  - That is, intelligent ways of dealing with the few bits of compiler specific code: `vecmem/cuda/X.h` vs `vecmem/hip/X.h`, or building sensible `workDivs` to deal with the hardware differences (Fixed vs variable warp size, CPU core count differences etc.)
- Extend the testing and verifying work once HIP is working, to ensure that CUDA and HIP continue to work.

What is next?

Few things to do:

- Continue with the three mentioned bits of work.
- Consider more intelligent ways of dealing with the multiple accelerator backends.
  - That is, intelligent ways of dealing with the few bits of compiler specific code: `vecmem/cuda/X.h` vs `vecmem/hip/X.h`, or building sensible `workDivs` to deal with the hardware differences (Fixed vs variable warp size, CPU core count differences etc.)
- Extend the testing and verifying work once HIP is working, to ensure that CUDA and HIP continue to work.
- Finally, more in-depth benchmarking of the Alpaka implementation, to help understand if / where bottlenecks are, and if there is anything in our Alpaka code that needs improving.

What is next?

Few things to do:

- Continue with the three mentioned bits of work.
- Consider more intelligent ways of dealing with the multiple accelerator backends.
  - That is, intelligent ways of dealing with the few bits of compiler specific code: `vecmem/cuda/X.h` vs `vecmem/hip/X.h`, or building sensible `workDivs` to deal with the hardware differences (Fixed vs variable warp size, CPU core count differences etc.)
- Extend the testing and verifying work once HIP is working, to ensure that CUDA and HIP continue to work.
- Finally, more in-depth benchmarking of the Alpaka implementation, to help understand if / where bottlenecks are, and if there is anything in our Alpaka code that needs improving.
- Further exploitation of Alpaka. We've still not really done any testing with Intel GPUs (which may not really be an issue with the current GPU landscape...), but we should try to use every back-end we get from Alpaka.

Conclusion

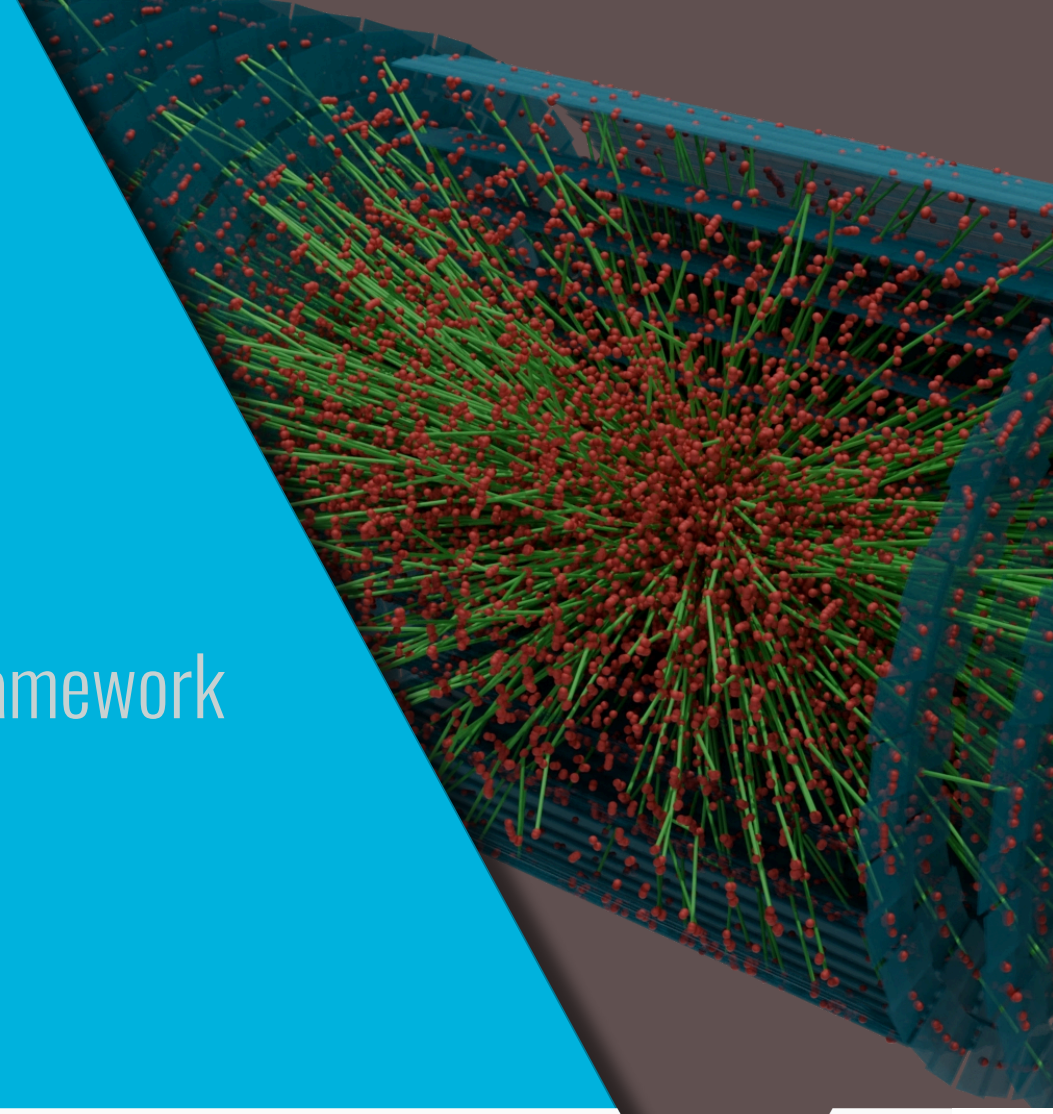
In Conclusion:

- traccc is a R&D effort as part of the ACTS project, working on exploiting GPUs and other accelerators to speed up tracking across a range of experiments.
- As part of that, many different acceleration abstraction libraries have been implemented, with alpaka being the newest.
- alpaka has good support already in HEP, and its parallelisation model make it a strong candidate for being the general purpose abstraction library.
- This talk gives a brief overview of the already completed work porting algorithms to utilise Alpaka in traccc.
- We are now at the point of having basically the full tracking chain of algorithms implemented in Alpaka, allowing direct comparison to the CPU and CUDA versions.
- More work in ongoing to verify alpaka with non-CUDA targets and improve the robustness of the alpaka implementation.

traccc

Integrating the Alpaka framework

Ryan Cross
SWIFT-HEP08 + ExaTEPP
2024/11/11



WARWICK
THE UNIVERSITY OF WARWICK

Backup Slides