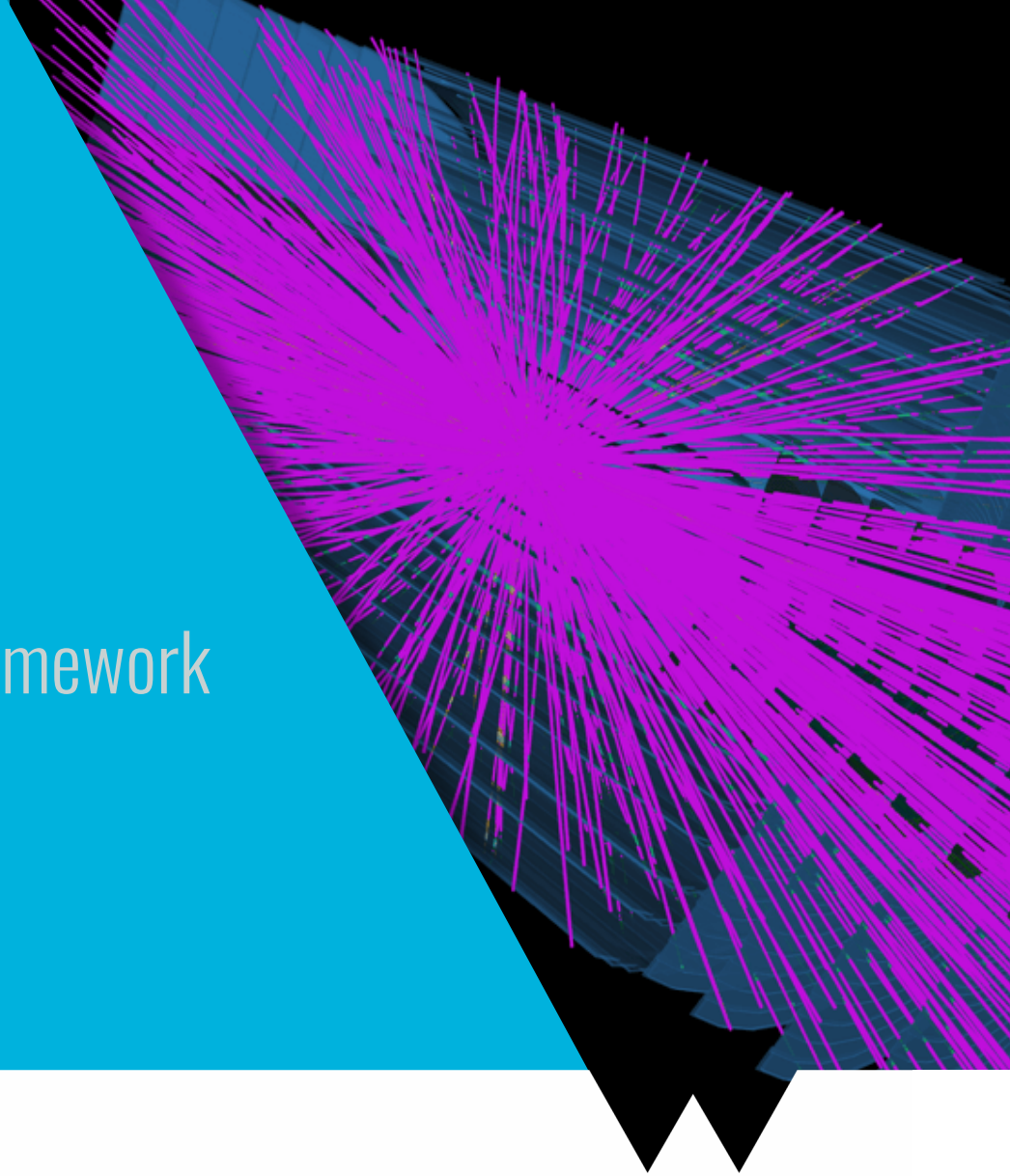
traccc

Integrating the Alpaka framework

Ryan Cross

GridPP51 & SWIFT-HEP07

2024/03/27



WARWICK
THE UNIVERSITY OF WARWICK

Overview

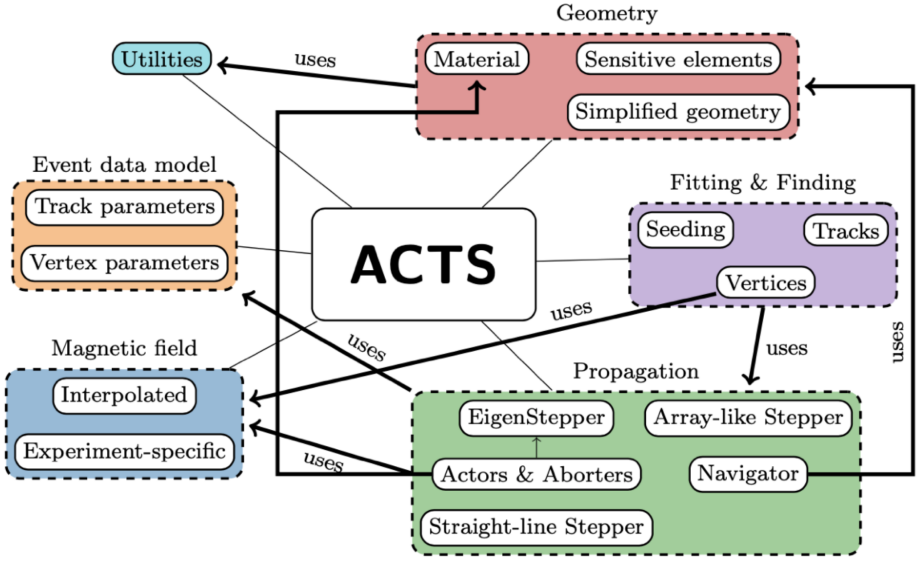
This talk will cover:

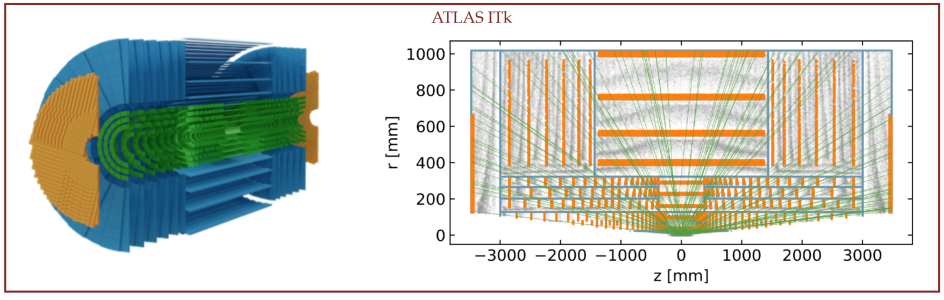
1. **traccc.**
2. **Cross-Platform Abstraction Libraries.**
3. **Where We Are**
4. **Current Work**
5. **What comes next?**

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.



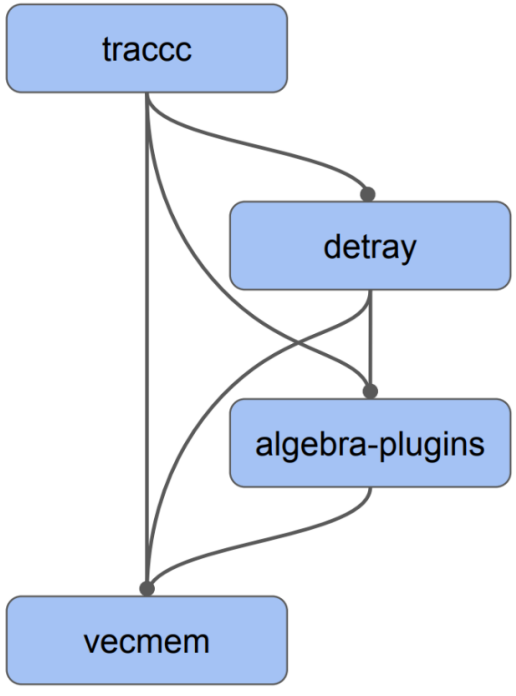


ACTS R&D Projects

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.

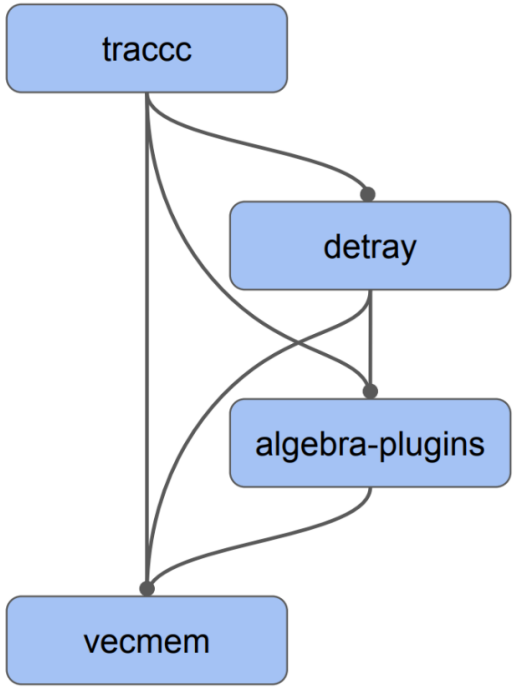


ACTS R&D Projects

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 is 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 is 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 is a few 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 is a few 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++ 17 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.

kokkos

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++17 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 is 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 is a reasonable starting point, fairly self-contained and easy to implement. This was added in PR #431.

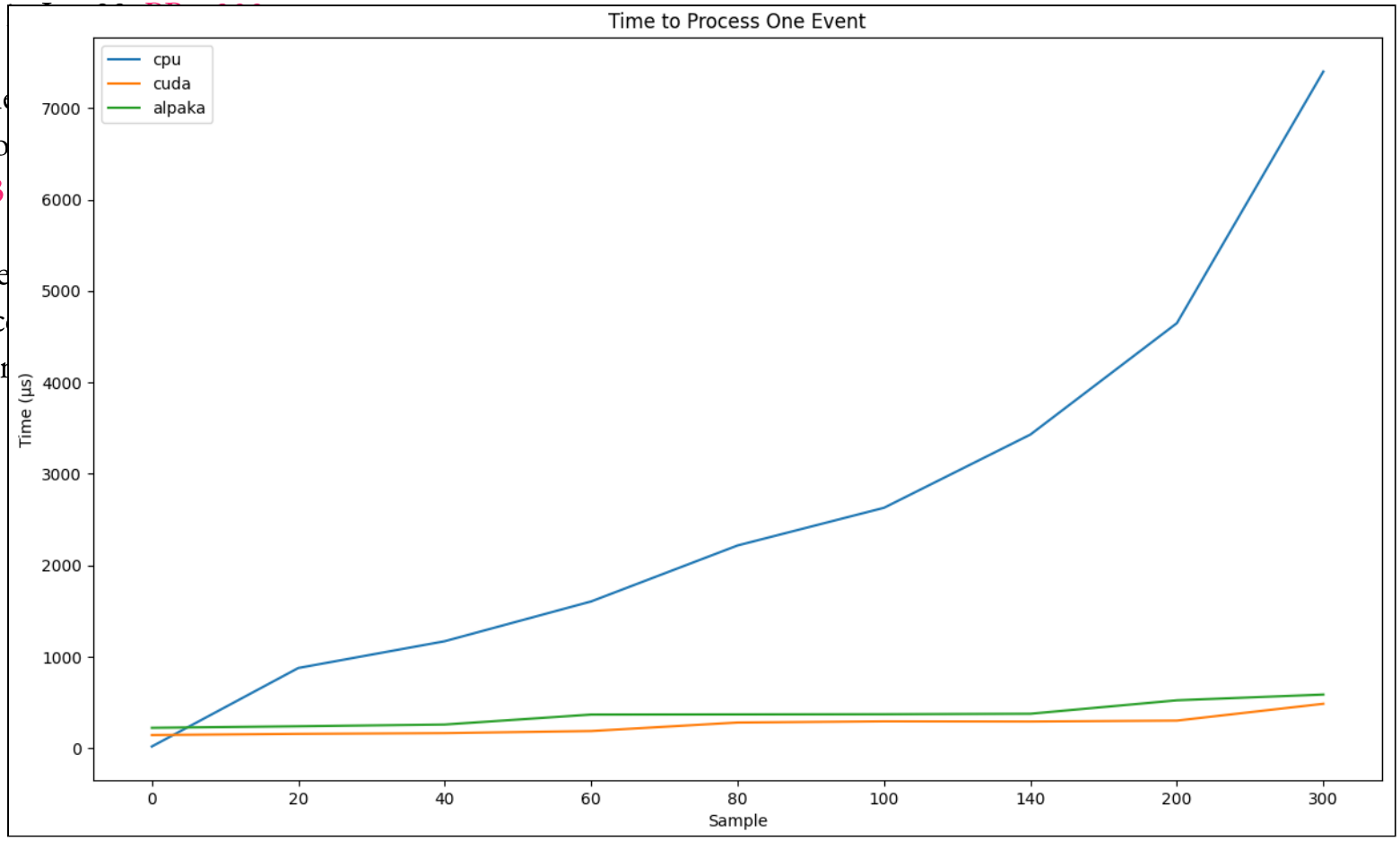
The spacepoint binning gave me a first look at development with Alpaka, as well as developing inside of traccc/ACTS. My previous slides, given at a UK SWIFT-HEP / GRIDPP meeting, give a bit of a better overview of that work, as well as some more basic comparisons of Alpaka vs CUDA.

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
over



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.

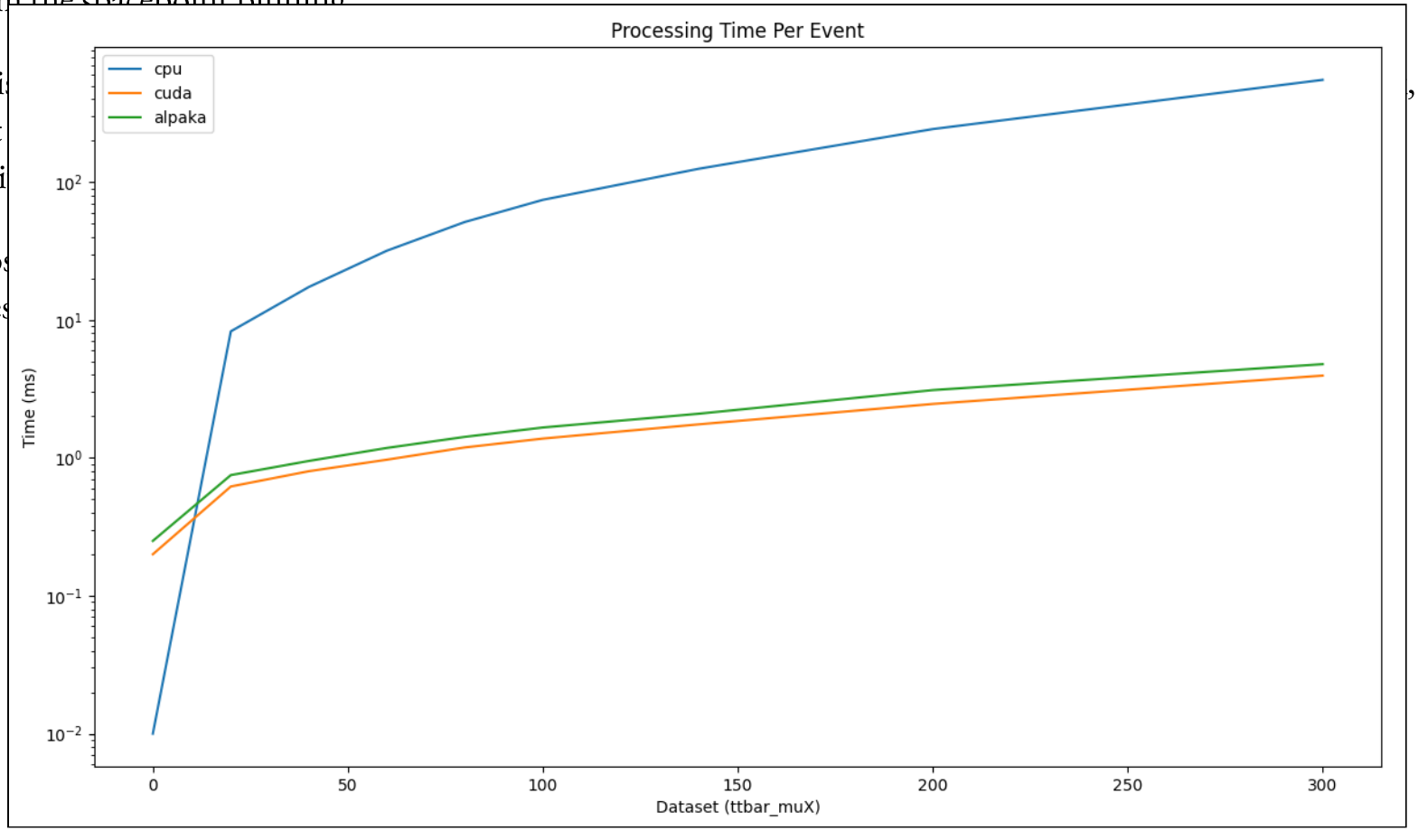
Most of that work was merged recently as part of PR (#451), with a secondary PR incoming to include the latest throughput examples and general tidying up.

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
late

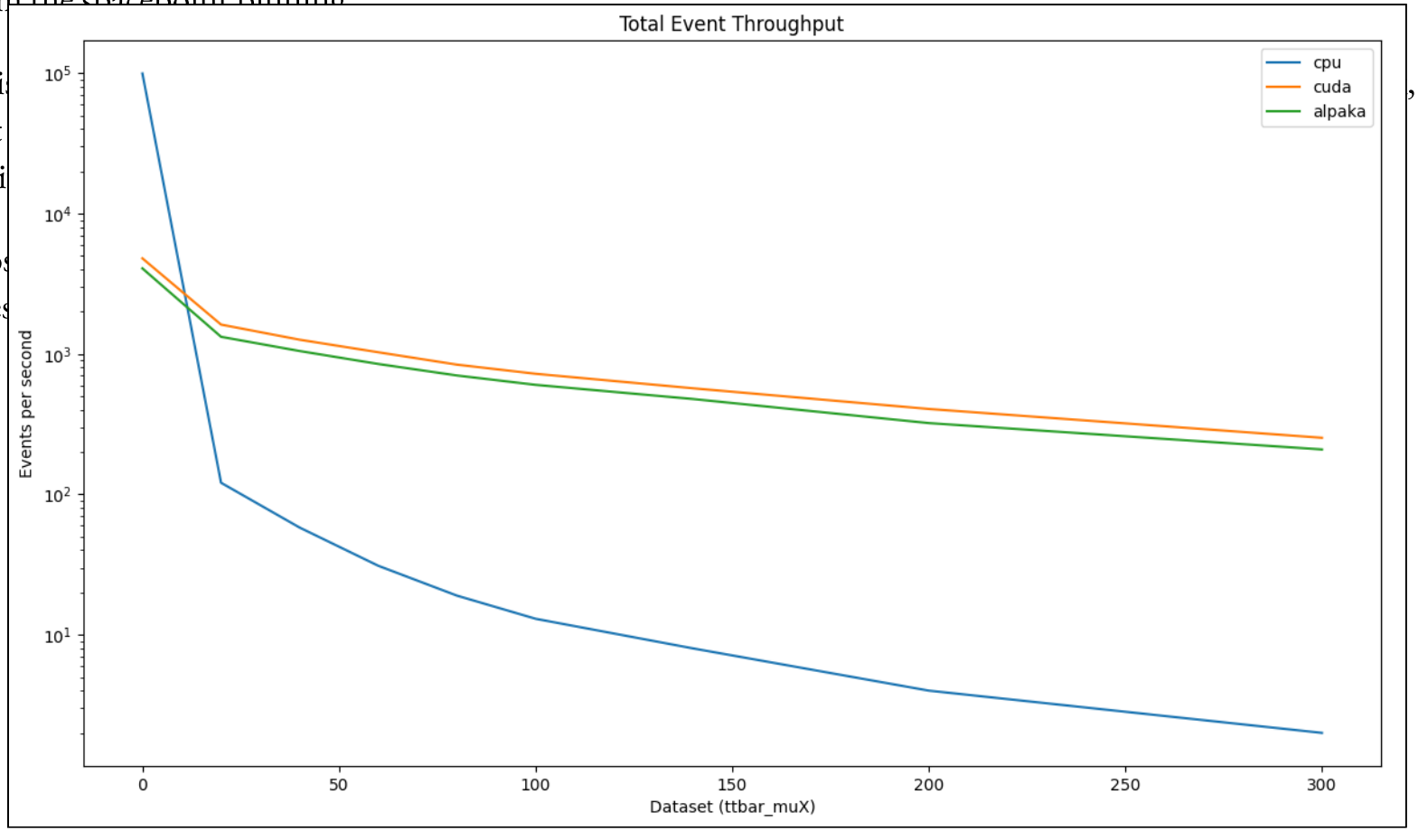


Completed Work

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

Thi
but
nati

Mos
late



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:

- 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.

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:

- 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.
- 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:

- 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.
- 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.
- Continue the porting process of further algorithms, bringing the Alpaka implementation closer to completely matching the CUDA one.

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.

There have been a giant number of PRs around this, covering all the main projects, traccc, vecmem and detray (PR #504, PR #511, PR #519, PR #272, PR #631, PR #652, PR #654).

HIP

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

There have been a giant number of PRs around this, covering all the main projects, traccc, vecmem and detray (PR #504, PR #511, PR #519, PR #272, PR #631, PR #652, PR #654).

This is fairly indicative of the complications around supporting these sorts of abstraction libraries though. Despite the main draw being "write once, compile N times", that does get more complicated when you consider that the newly supported accelerator back-ends have their own compiler, with their own flags, macros, dealing with of code, so your code base needs to support every possible option at once.

HIP

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

There have been a giant number of PRs around this, covering all the main projects, traccc, vecmem and detray (PR #504, PR #511, PR #519, PR #272, PR #631, PR #652, PR #654).

This is fairly indicative of the complications around supporting these sorts of abstraction libraries though. Despite the main draw being "write once, compile N times", that does get more complicated when you consider that the newly supported accelerator back-ends have their own compiler, with their own flags, macros, dealing with of code, so your code base needs to support every possible option at once.

This is fairly advanced now, with it actually compiling and being able to partially run the code. Remaining issues are potentially less to do with architecture level changes still being needed, and more to do with brittleness in the existing Alpaka code.

HIP

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

There have been a giant number of PRs around this, covering all the main projects, traccc, vecmem and detray (PR #504, PR #511, PR #519, PR #272, PR #631, PR #652, PR #654).

This is fairly indicative of the complications around supporting these sorts of abstraction libraries though. Despite the main draw being "write once, compile N times", that does get more complicated when you consider that the newly supported accelerator back-ends have their own compiler, with their own flags, macros, dealing with of code, so your code base needs to support every possible option at once.

This is fairly advanced now, with it actually compiling and being able to partially run the code. Remaining issues are potentially less to do with architecture level changes still being needed, and more to do with brittleness in the existing Alpaka code.

The end goal here is that we can have the full examples running on both Nvidia and AMD, with only a single flag change at compile time to target the relevant architectures. All the changes made for this work also help improve the code for potential later test with other accelerators.

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 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 CUDA version.

I've slowly being debugging this over a number of months, but haven't found any obvious smoking gun yet. I have gotten much more experience with the CUDA debugging experience in alpaka which has been very useful, and the bug has been mostly narrowed to a single file, but further work is needed to check what the exact bug is.

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 CUDA version.

I've slowly being debugging this over a number of months, but haven't found any obvious smoking gun yet. I have gotten much more experience with the CUDA debugging experience in alpaka which has been very useful, and the bug has been mostly narrowed to a single file, but further work is needed to check what the exact bug is.

It isn't seen at all in the examples that only run a single event, but only in those that run many events in sequence, at least when using the CUDA back-end. I'm also doing some testing with a CPU, single-threaded back-end to see if I can reproduce the error there, which would make debugging a lot easier (at least compared to hundreds / thousands of CUDA kernels running).

Track Finding + Fitting

The next obvious step, once you have the "Prototracks" from the current code, is to perform a more sophisticated track finding and fitting procedure, by porting that code from CUDA to Alpaka.

That does, however, come with its own new complications compared to the ported code so far:

Track Finding + Fitting

The next obvious step, once you have the "Prototracks" from the current code, is to perform a more sophisticated track finding and fitting procedure, by porting that code from CUDA to Alpaka.

That does, however, come with its own new complications compared to the ported code so far:

- The largest difference is the very heavy usage of the Thrust library, a Nvidia-maintained CUDA parallelisation library. This obviously makes my job potentially more difficult, as the behaviours of the code now need to either be changed, or re-implemented in an Alpaka-friendly way.

Track Finding + Fitting

The next obvious step, once you have the "Prototracks" from the current code, is to perform a more sophisticated track finding and fitting procedure, by porting that code from CUDA to Alpaka.

That does, however, come with its own new complications compared to the ported code so far:

- The largest difference is the very heavy usage of the Thrust library, a Nvidia-maintained CUDA parallelisation library. This obviously makes my job potentially more difficult, as the behaviours of the code now need to either be changed, or re-implemented in an Alpaka-friendly way.
- The objects and data-structures in use are more complicated, making more work when porting the code to ensure Alpaka can easily use the objects and they are in a form Alpaka expects.
  - Alpaka expects all objects to be `std::is_trivially_copyable` which wasn't an issue so far, but looks like it could be slightly more awkward here. However, newer Alpaka versions allow you to add traits to help with convincing Alpaka of that.

Track Finding + Fitting

The next obvious step, once you have the "Prototracks" from the current code, is to perform a more sophisticated track finding and fitting procedure, by porting that code from CUDA to Alpaka.

That does, however, come with its own new complications compared to the ported code so far:

- The largest difference is the very heavy usage of the Thrust library, a Nvidia-maintained CUDA parallelisation library. This obviously makes my job potentially more difficult, as the behaviours of the code now need to either be changed, or re-implemented in an Alpaka-friendly way.
- The objects and data-structures in use are more complicated, making more work when porting the code to ensure Alpaka can easily use the objects and they are in a form Alpaka expects.
  - Alpaka expects all objects to be `std::is_trivially_copyable` which wasn't an issue so far, but looks like it could be slightly more awkward here. However, newer Alpaka versions allow you to add traits to help with convincing Alpaka of that.

Work is slowly moving here, first testing within thrust with a CUDA target only, to slightly delay working out the best / most appropriate approach to replace the Thrust code.

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.

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.
- More work in ongoing to verify alpaka with non-CUDA targets, improve the robustness of the alpaka implementation, and further complete porting of algorithms to alpaka.

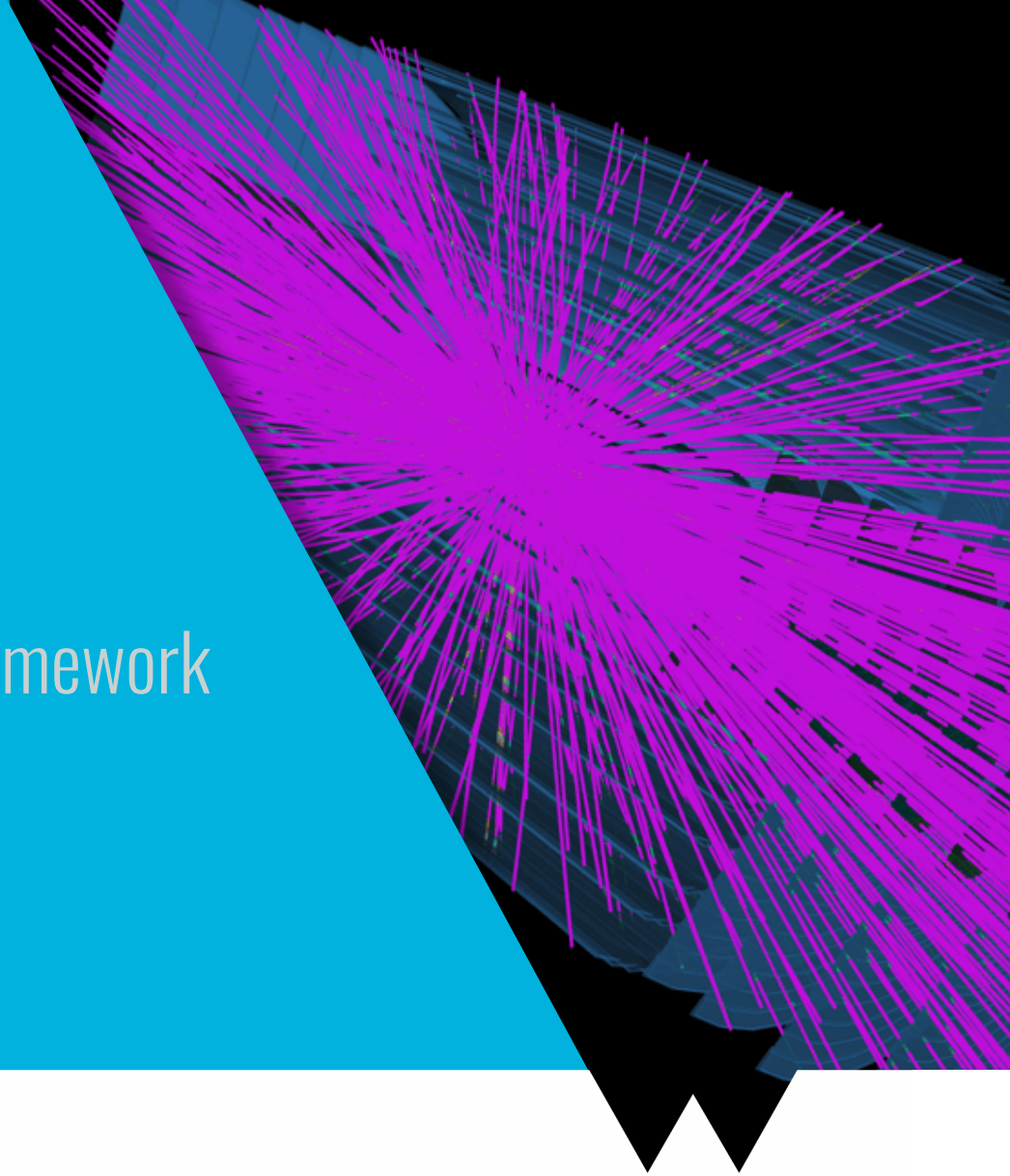
traccc

Integrating the Alpaka framework

Ryan Cross

GridPP51 & SWIFT-HEP07

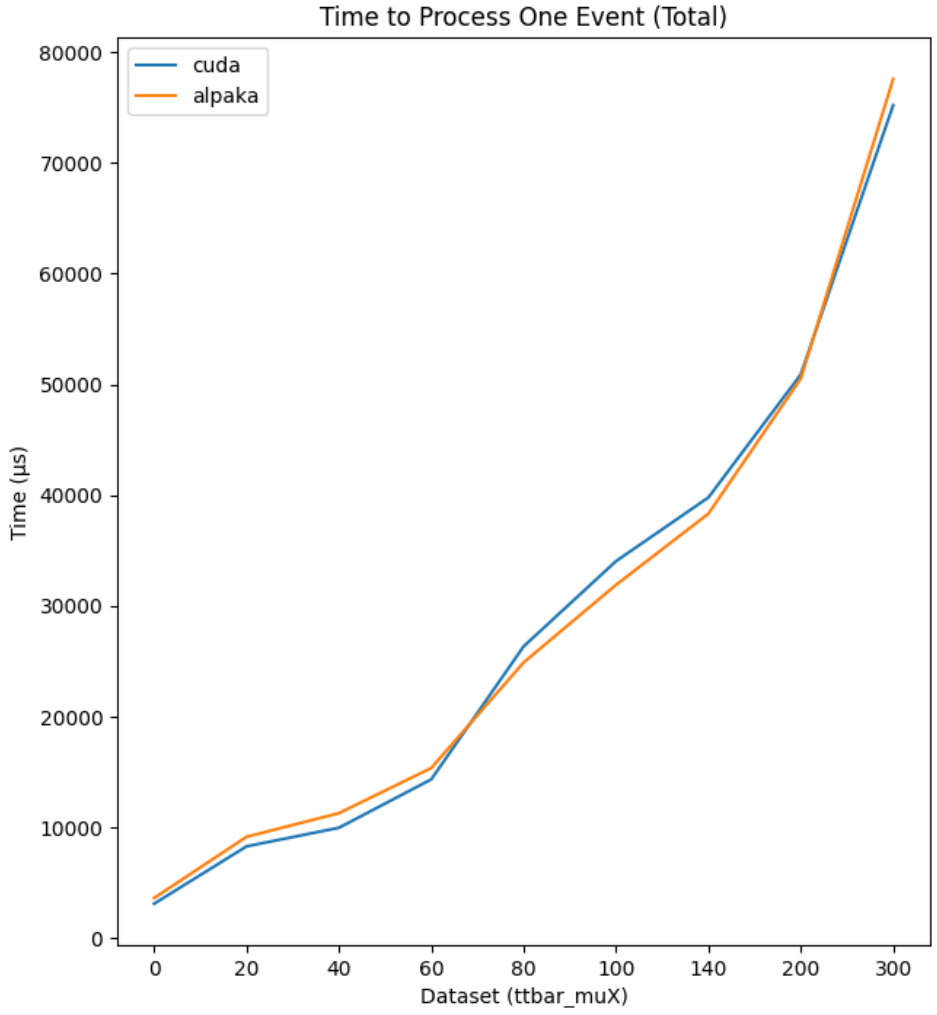
2024/03/27



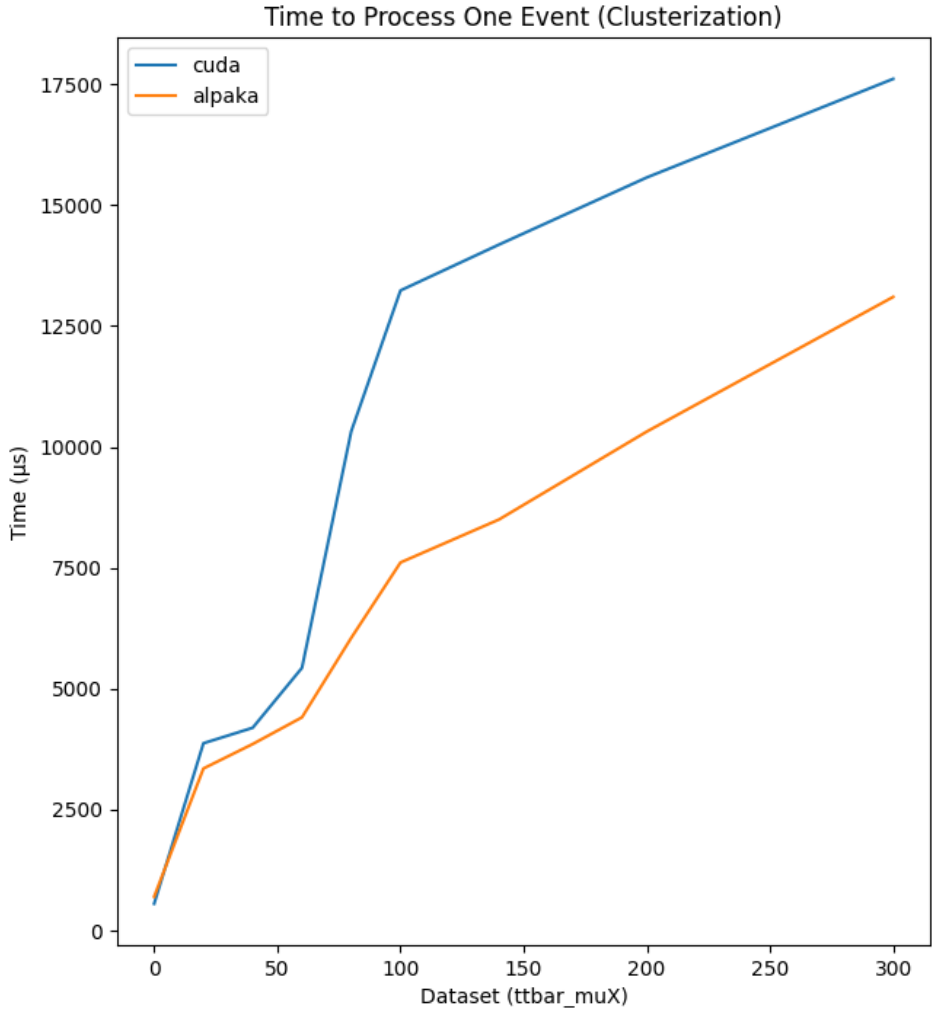
WARWICK
THE UNIVERSITY OF WARWICK

Backup Slides

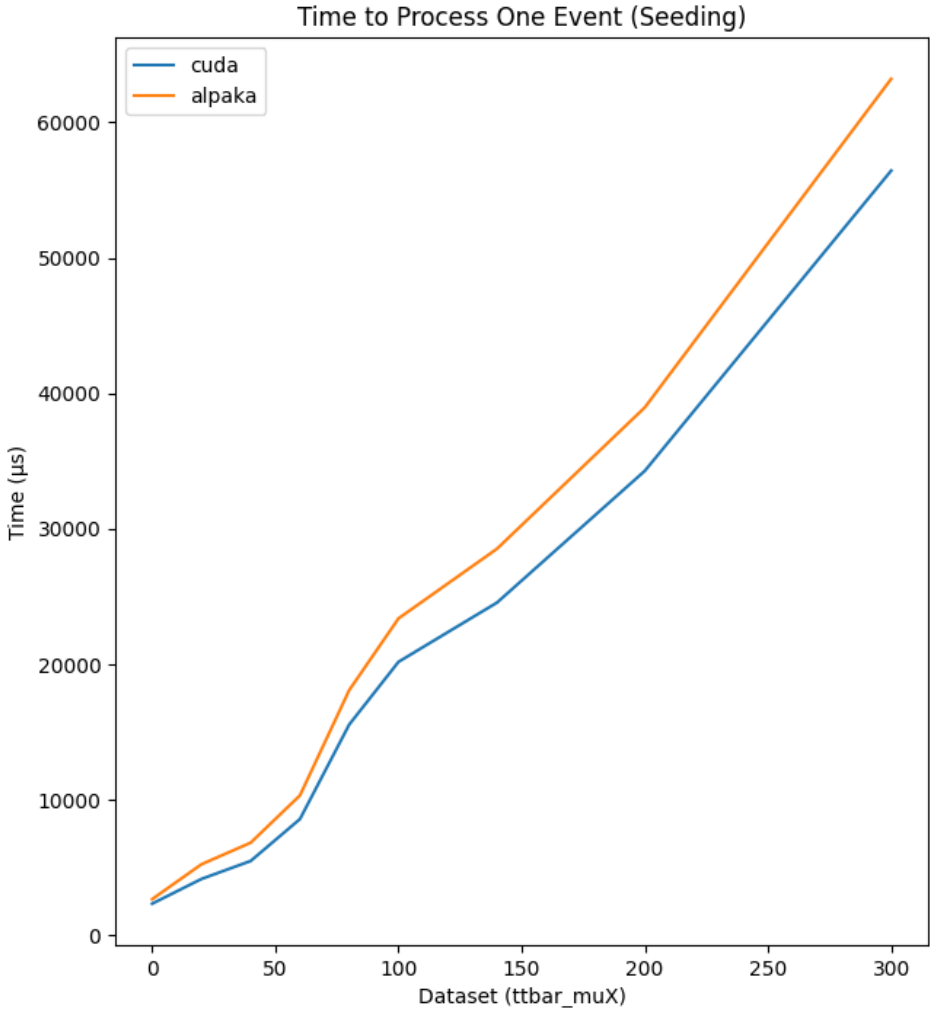
CUDA vs alpaka only



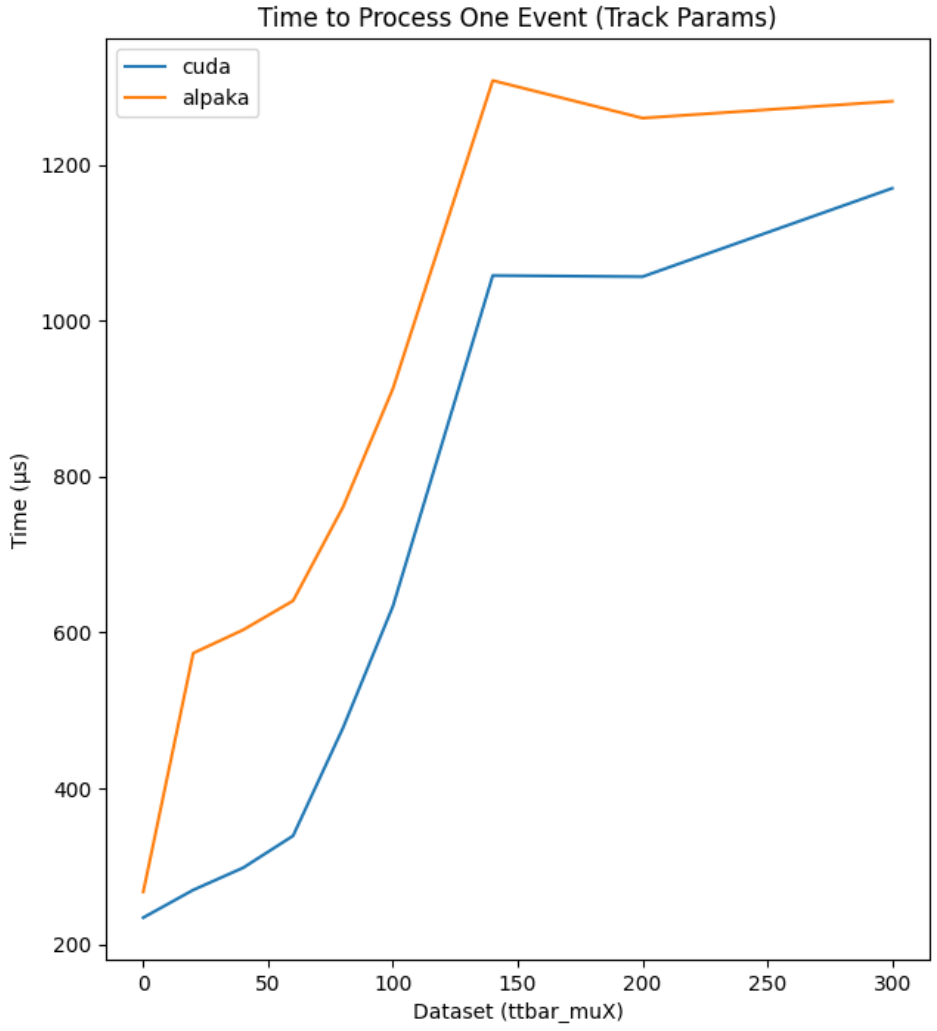
CUDA vs alpaka only



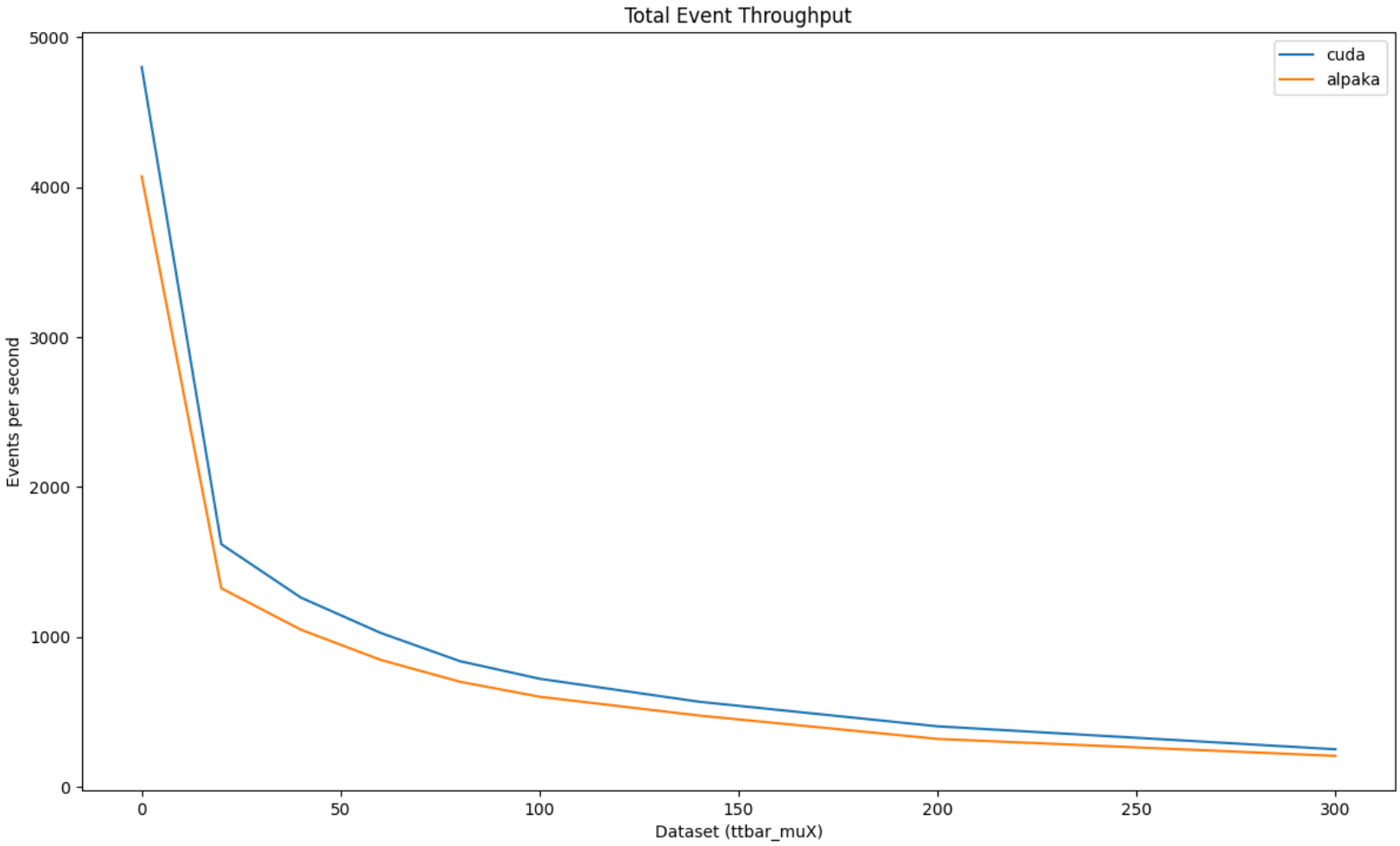
CUDA vs alpaka only



CUDA vs alpaka only



CUDA vs alpaka only



CUDA vs alpaka only