Workshop on High Throughput Heterogeneous Computing

TRACCC: the ACTS Massively Parallel Tracking Demonstrator

Stephen Nicholas Swatman[1] on behalf of the ACTS developers

Thursday, January 29, 2026

[1]CERN

Introduction – TRACCC and Next Generation Triggers

- TRACCC is an ACTS subproject towards an...
  - **efficient**
  - **massively parallel**
  - **track reconstruction** software package
- TRACCC is supported by the **CERN NGT** project
- Goal: “remarkably increase **efficiency**, **sensitivity** and **modelling** of CERN experiments“
- Through the use of **novel hardware**, including **GPGPUs** (general purpose GPUs)
- **Five-year effort** to radically advance many aspects of **LHC computing**
- *https://nextgentriggers.web.cern.ch/*

Introduction – Motivation

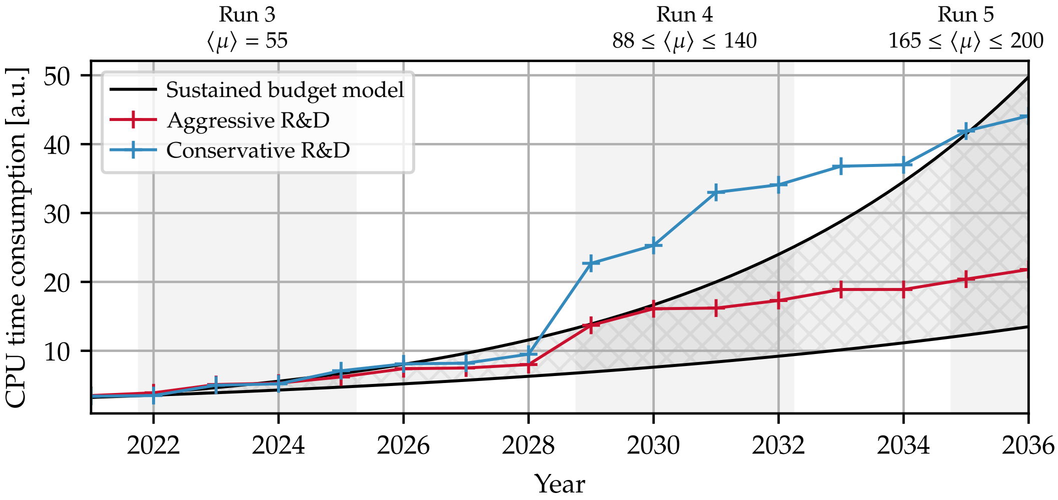


Image adapted from ATLAS

Introduction – Motivation

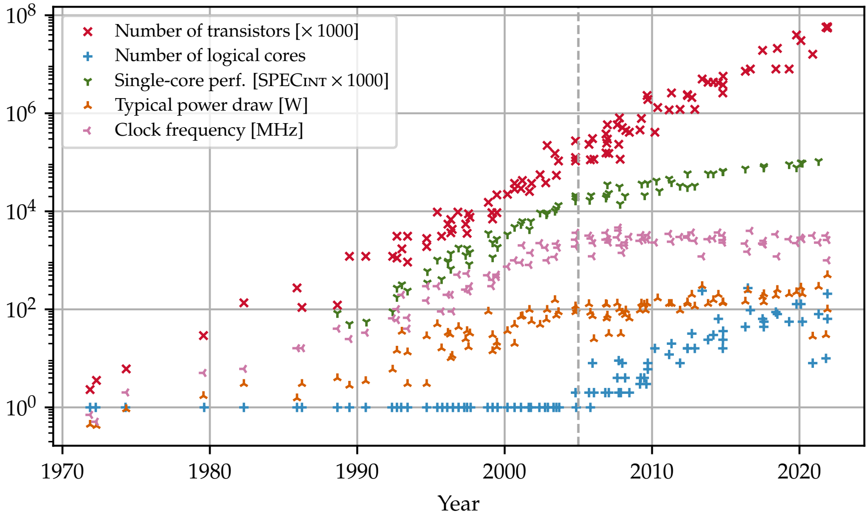
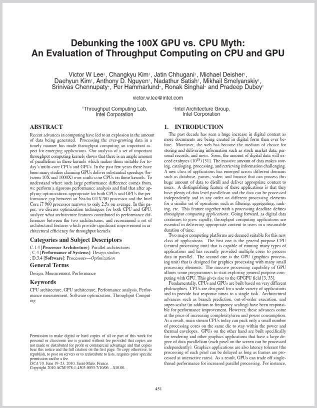


Image adapted from Karl Rupp

Introduction – GPGPU computing

- GPUs are incredible **compute accelerators**
    - Over 10,000 cores!
- But GPUs are **not magic**
- They will never give **asymptotic advantage**
- And they can be **difficult** to program
    - As are CPU SIMD lanes!
- **Constant** factors are very important
    - But non-GEMM performance is **sometimes exaggerated**



Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU

Victor W Lee[†], Changkyu Kim[†], Jatin Chhugani[†], Michael Deisher[†], Daehyun Kim[†], Anthony D. Nguyen[†], Nadathur Satish[†], Mikhail Smelyanskiy[†], Srinivas Chennupaty[*], Per Hammarlund[*], Ronak Singhal[*] and Pradeep Dubey[†]

victor.w.lee@intel.com

[†]Throughput Computing Lab, Intel Corporation

[*]Intel Architecture Group, Intel Corporation

ABSTRACT

Recent advances in computing have led to an explosion in the amount of data being generated. Processing the ever-growing data in a timely manner has made throughput computing an important aspect for emerging applications. Our analysis of a set of important throughput computing kernels shows that there is an ample amount of parallelism in these kernels which makes them suitable for today's multi-core CPUs and GPUs. In the past few years there have been many studies claiming GPUs deliver substantial speedups (between 10X and 1000X) over multi-core CPUs on these kernels. To understand where such large performance difference comes from, we perform a rigorous performance analysis and find that after applying optimizations appropriate for both CPUs and GPUs the performance gap between an Nvidia GTX280 processor and the Intel Core i7 960 processor narrows to only 2.5x on average. In this paper, we discuss optimization techniques for both CPU and GPU, analyze what architecture features contributed to performance differences between the two architectures, and recommend a set of architectural features which provide significant improvement in architectural efficiency for throughput kernels.

Categories and Subject Descriptors

C.1.4 [**Processor Architecture**]: Parallel architectures ; C.4 [**Performance of Systems**]: Design studies ; D.3.4 [**Software**]: Processors—*Optimization*

General Terms

Design, Measurement, Performance

Keywords

CPU architecture, GPU architecture, Performance analysis, Performance measurement, Software optimization, Throughput Computing

1. INTRODUCTION

The past decade has seen a huge increase in digital content as more documents are being created in digital form than ever before. Moreover, the web has become the medium of choice for storing and delivering information such as stock market data, personal records, and news. Soon, the amount of digital data will exceed exabytes (10^{18}) [31]. The massive amount of data makes storing, cataloging, processing, and retrieving information challenging. A new class of applications has emerged across different domains such as database, games, video, and finance that can process this huge amount of data to distill and deliver appropriate content to users. A distinguishing feature of these applications is that they have plenty of data level parallelism and the data can be processed independently and in any order on different processing elements for a similar set of operations such as filtering, aggregating, ranking, etc. This feature together with a processing deadline defines *throughput computing applications*. Going forward, as digital data continues to grow rapidly, throughput computing applications are essential in delivering appropriate content to users in a reasonable duration of time.

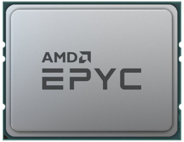
Two major computing platforms are deemed suitable for this new class of applications. The first one is the general-purpose CPU (central processing unit) that is capable of running many types of applications and has recently provided multiple cores to process data in parallel. The second one is the GPU (graphics processing unit) that is designed for graphics processing with many small processing elements. The massive processing capability of GPU allures some programmers to start exploring general purpose computing with GPU. This gives rise to the GPGPU field [3, 33].

Fundamentally, CPUs and GPUs are built based on very different philosophies. CPUs are designed for a wide variety of applications and to provide fast response times to a single task. Architectural advances such as branch prediction, out-of-order execution, and super-scalar (in addition to frequency scaling) have been responsible for performance improvement. However, these advances come at the price of increasing complexity/area and power consumption. As a result, main stream CPUs today can pack only a small number of processing cores on the same die to stay within the power and thermal envelopes. GPUs on the other hand are built specifically for rendering and other graphics applications that have a large degree of data parallelism (each pixel on the screen can be processed independently). Graphics applications are also latency tolerant (the processing of each pixel can be delayed as long as frames are processed at interactive rates). As a result, GPUs can trade off single-thread performance for increased parallel processing. For instance,

451

Introduction – GPGPU computing

For around 8,000 EUR in 2026:



AMD EPYC 9555P
64 cores
360W TDP

NVIDIA RTX PRO 6000 Blackwell
24,064 cores
300W TDP

Device	Cores	× Cycles/s	× FLOP/cycle	= FLOP/s
AMD EPYC 9555P	64	4.40B	64	18.0T
NVIDIA RTX 6000 BW.	24,064	2.29B	2	110.2T

Introduction – Parallelism

Embarrassingly	Invitingly	Humblingly
parallel	parallel	parallel

Terminology due to Raph Levien

Introduction – Parallelism

Embarrassingly parallel	Invitingly parallel	Humblingly parallel
Axpy		
Bitcoin mining		
Shaders		

Terminology due to Raph Levien

Introduction – Parallelism

Embarrassingly parallel	**Invitingly** parallel	**Humblingly** parallel
Axpy	*gemm*	
Bitcoin mining	Histogramming	
Shaders	FFT	

Terminology due to Raph Levien

Introduction – Parallelism

Embarrassingly parallel	Invitingly parallel	Humblingly parallel
Axpy	*gemm*	*SpMV*
Bitcoin mining	Histogramming	(De-)compression
Shaders	FFT	Sorting

Terminology due to Raph Levien

Introduction – Parallelism

Embarrassingly parallel	Invitingly parallel	Humblingly parallel
Axpy	*gemm*	*SpMV*
Bitcoin mining	Histogramming	(De-)compression
Shaders	FFT	Sorting
Track reconstruction		

Terminology due to Raph Levien

Introduction – Parallelism

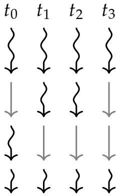
Embarrassingly parallel	**Invitingly** parallel	**Humblingly** parallel
Axpy	*gemm*	*SpMV*
Bitcoin mining	Histogramming	(De-)compression
Shaders	FFT	Sorting

~~Track reconstruction~~ $\xrightarrow[\text{On GPGPUs}]{}$ Track reconstruction

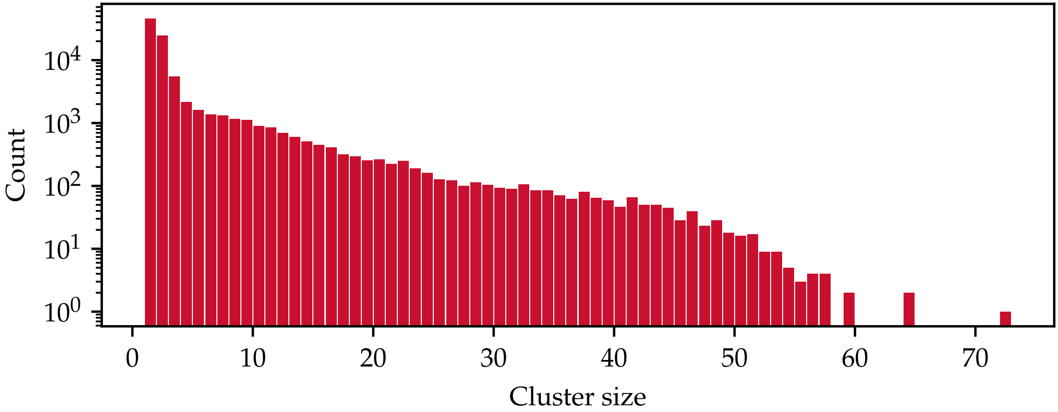
Terminology due to Raph Levien

- GPU threads run in **lockstep**
- One **instruction stream** is broadcast to a **group of threads** (32–64)
- **Branch divergence** causes idle time
- As do **unequal loop structures**
- Behaviour much like **SIMD lanes**

	m_0	m_1	m_2	m_3
`int n = thread_id();`	✓	✓	✓	✓
`prologue();`	✓	✓	✓	✓
`if (0 < n < 3) {`				
`  branch1();`	✗	✓	✓	✗
`} else if (n == 0) {`				
`  branch2();`	✓	✗	✗	✗
`}`				
`epilogue();`	✓	✓	✓	✓

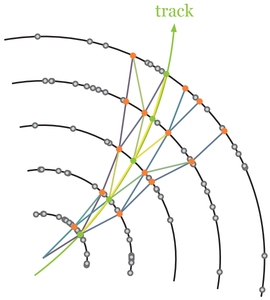


Track Reconstruction – Clustering



Track Reconstruction – Combinatorial Kálmán Filter

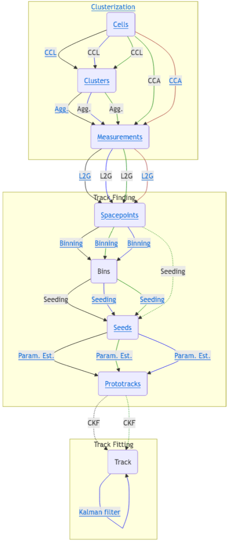
- The **Combinatorial Kálmán Filter** extends seeds
- **Branches** frequently, contains **nested, unbound loops**
- One of the biggest **bottlenecks** and most **complicated** algorithms
- Presents *many* challenges:
  - How do we manage the **combinatorics**?
  - How do we describe our **detector** in **GPU memory**?
  - How do we keep **magnetic field accesses** fast?



Source: Paul Gessinger

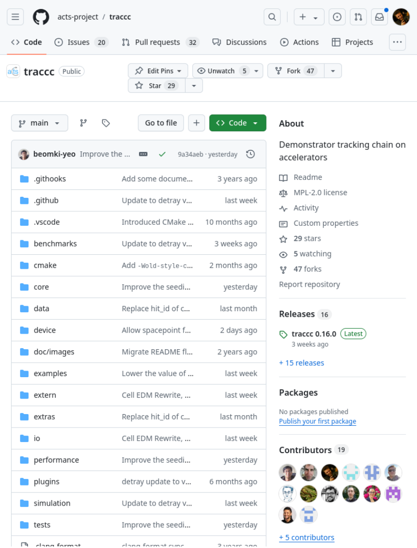
Track Reconstruction – Summary

- Around **8 subproblems** with wildly different characteristics
- Map **non-trivially** to massively parallel hardware
  - Imbalance, divergence, irregular access patterns, etc.
- Requires much more than a **naive** porting exercise!



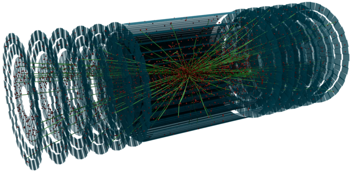
Implementation – TRACCC

- TRACCC is our **open-source massively parallel track reconstruction pipeline**
- **Designed** from the ground up **for GPGPUs**
- Algorithms often **completely rethought**
- Aim: match **physics performance** of homogeneous solutions
- See e.g. 10.5281/ZENODO.8119504 for more info



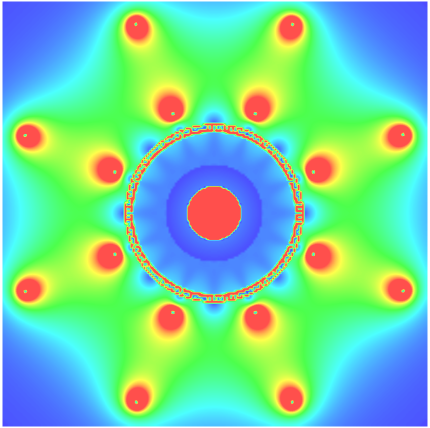
Spin-Off Projects – DETRAY

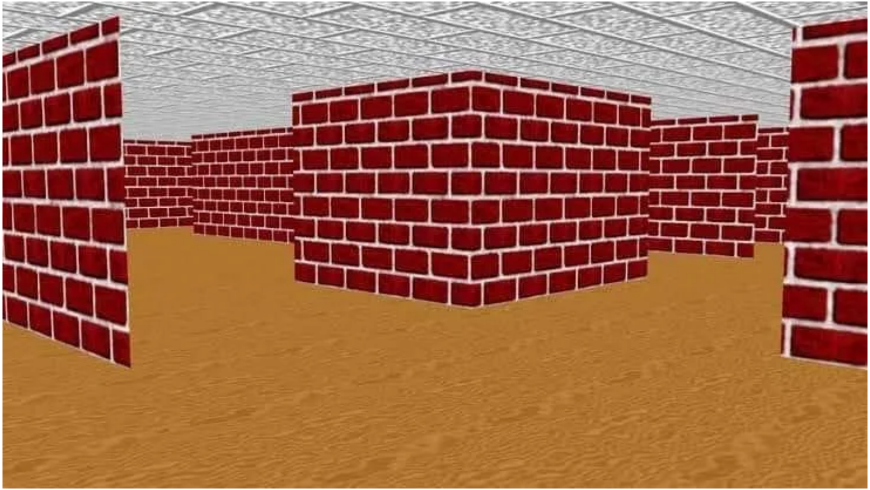
- **Detector descriptions** are classically **polymorphic**, which doesn't fly in **GPGPUs**
- **DETRAY** is our heterogeneous **detector geometry**
  - *Crucial* component of any non-trivial reconstruction
- *Tremendous* amount of work by the DETRAY devs
- See 10.1088/1742-6596/2438/1/012026 for more info



Spin-Off Projects – COVFIE

- Reconstruction features **highly frequent, highly irregular structured grid access**
- **COVFIE** is our library for handling arbitrary vector fields incl. magnetic fields
- **Cross-platform** performance through **compile-time composition**
- Allows e.g. use of **texture memory**
- See 10.1145/3578244.3583723 for more info





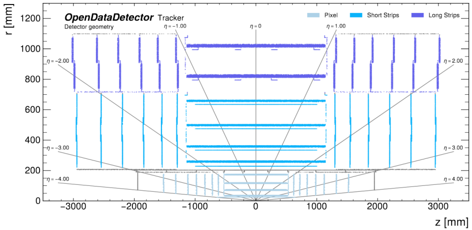
Source: Microsoft

Spin-Off Projects – Further Contributions

- The TRACCC effort also (indirectly) produced **models** and **methods**
- Novel **derivations of Jacobian matrices**: 10.1016/j.nima.2024.169734
- **Models** for **thread divergence**: 10.1109/MASCOTS56607.2022.00026
- **Genetic algorithms** for **structured grid layouts**: 10.1145/3629526.3645034
- Novel method for **transparent** SoA and AoS layouts
- **Throughput models** for heterogeneous task graphs

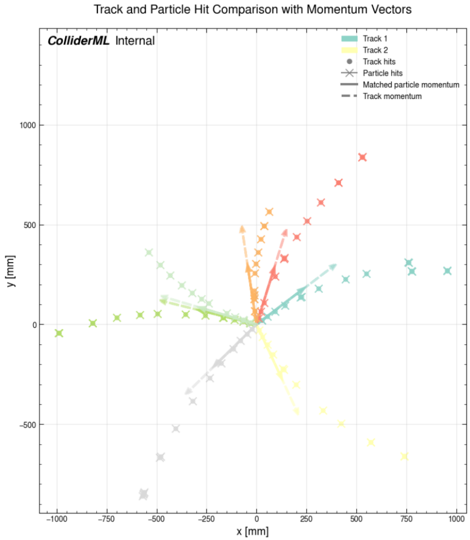
The Detector – The ODD

- ATLAS is great, but an **open-source detector** gives us some great benefits:
  - No plot **approvals**
  - Free code and data **sharing**
  - Ease of use for **non-ATLAS users**
  - Freedom from the **grimy real world**
- This is why we "built" the **OpenDataDetector**



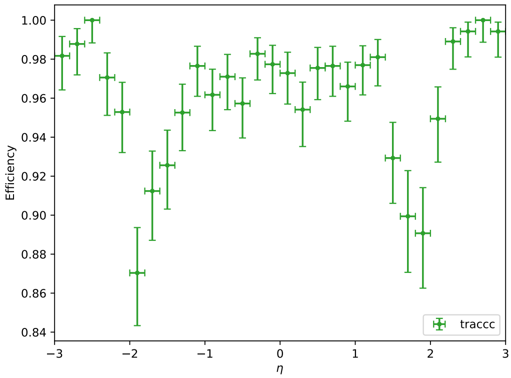
The Detector – The ODD

- The ODD served as the base for the wildly successful **TrackML** Kaggle challenge
- Also serves as the main **evaluation** tool for TRACCC
- Recently released **ColliderML**: the biggest freely available **high-luminosity dataset** for e.g. ML training
- See *https://colliderml.com/*



The Good Parts – Success in Physics!

- TRACCC provides **good physics performance** on the ODD
- And we are very nearly within limits for the **ATLAS ITk**
- Given the **from-scratch** nature of TRACCC, this is an impressive result!



The Good Parts – Success in Compute!

April 2025

Kernel	1edca0f
`fit`	280.68 ms
`propagate_to_next_surface`	118.21 ms
`find_tracks`	26.36 ms
`count_triplets`	14.16 ms
`find_triplets`	5.98 ms
`build_tracks`	1.07 ms
Total	**450.89 ms**

January 2026

Kernel	9bcb542
`propagate_to_next_surface`	7.80 ms
`find_tracks`	1.73 ms
`ccl_kernel`	825.79 μs
`count_doublets`	815.01 μs
Total	**13.16 ms**

The Good Parts – Success in Compute!

- We managed to increase our performance $30\times$ **in 9 months**
- Current performance makes us **competitive with CPU solutions**
- **Realistic cost savings** with current solution
- But these are **percentage** savings (not orders of magnitude)
- Perhaps the benefit will increase more?

Kernel	`9bcb542`
`propagate_to_next_surface`	7.80 ms
`find_tracks`	1.73 ms
`ccl_kernel`	825.79 μs
`count_doublets`	815.01 μs
Total	**13.16 ms**

The Lessons Learned – Portability and Code Sharing

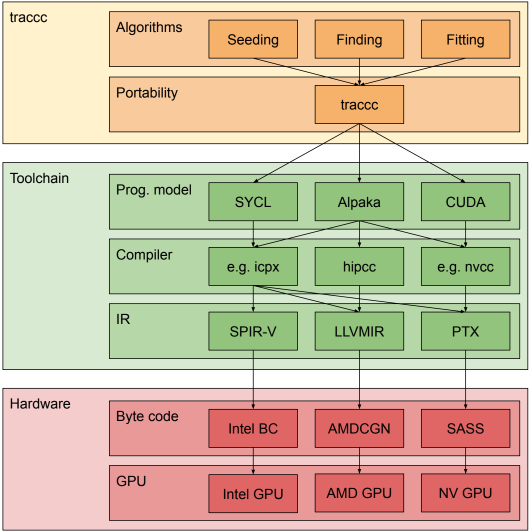
- TRACCC set out with ambitious ideas
- **Share** as much code **between CPU and GPU** as possible
  - In order to reduce **maintenance**
- Support as many **programming models** as possible
  - In order to support many devices
  - NVIDIA CUDA, AMD HIP, SYCL, etc.
- Unfortunately, neither of these approaches really worked out
  - That's R&D for you!

The Lessons Learned – Code Sharing

- **Sharing code** between **CPUs and GPUs** is tricky
- Shareable code is generally limited, watch out for:
  - Code with any **dynamic memory allocation** (incl. `std::vector`)
  - Code with **large amounts of stack usage**
  - Code with **unbound loops** (or large bound loops)
  - Early returns, **complex control flow**
- Setting out to share too much leads to issues: **start small and unify later**

The Lessons Learned – Portability

- Our approach to portability has resulted in **high maintenance and little benefit**
- “like wearing two raincoats on top of each other”
- Cross-platform support forces meeting at the **smallest common denominator**
- Recommendation, either:
  - Focus on **performance** in *one* programming model and port later; or
  - Focus on a *single* **portability** solution from the start



The Lessons Learned – Performance Monitoring

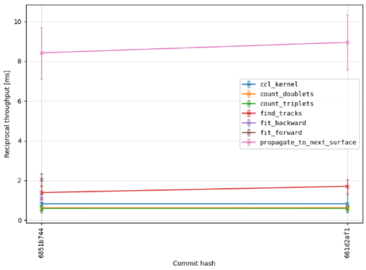
- HEP has a strong culture of monitoring **physics performance**
- For a project like TRACCC, compute performance is also a **first-class monitorable** – at **kernel level**
- Only last year did we get **continuous compute monitoring**
- Critical for informing **accept–reject decisions**
- Also track performance **changes over time** to find **regressions**

stephenswat commented on Dec 5, 2025 · Member · Author

Performance summary

Here is a summary of the performance effects of this PR:

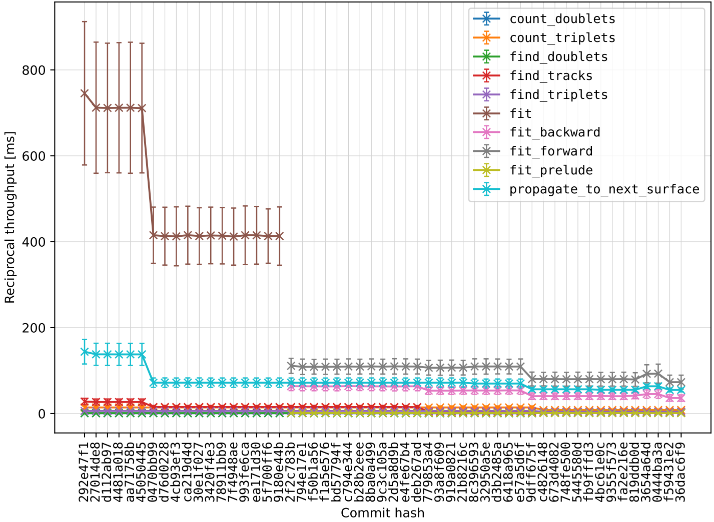
Graphical



Tabular

Kernel	Reciprocal Throughput			Parallelism	
	6851b74	661d2af	Delta	6851b74	661d2af
`ccl_kernel`	823.86 μs	823.66 μs	-0.0%	1.37	1.37
`fill_fitting_sort_keys`	152.79 ns	—	—	22.08	nan
`find_tracks`	1.39 ms	1.70 ms	22.7%	1.80	1.85
`count_doublets`	615.67 μs	623.37 μs	1.3%	1.60	1.61
`count_triplets`	588.61 μs	590.85 μs	0.4%	1.02	1.02
`find_doublets`	453.93 μs	446.97 μs	-1.5%	3.08	3.08

The Lessons Learned – Performance Monitoring



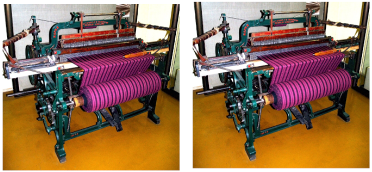
The Lessons Learned – The Latency Myth

"My CPU solution runs in 10 ms *and my GPU solution runs in* 4 ms, *so my GPU solution is 2.5 times faster"*

The Lessons Learned – The Latency Myth

"Factory A makes a carpet in 10 hours and factory B takes 4 hours, so factory B produces 2.5 times more carpets"

The Lessons Learned – The Latency Myth



8 looms, 10 h. / carpet = 0.8 carpets / h

2 looms, 4 h. / carpet = 0.5 carpets / h

“Factory A makes a carpet in 10 hours and factory B takes 4 hours, so factory B produces 2.5 times more carpets”

The Lessons Learned – The Latency Myth

- For throughput-critical applications, **latency is not enough**!
- Compute **throughput** using latency: $T = \frac{N}{L}$
    - Computation of N differs for CPUs and GPUs
- If you want a latency-like metric, use **reciprocal throughput**
    - "How long does it take to produce a carpet on average?"
    - "What is the average amount of time between carpets being finished?"
- Both **measured in time**, but **semantically** different!

Open Challenges – Scheduling and Placement

- **Scheduling** and **placement** remain difficult questions for us
- **Dynamic** scheduling between CPU and GPU risks hard-to-debug **runtime issues**
- Static scheduling risks **imbalance** between CPU and GPU
    - Can be alleviated with MPI/SaaS – but needs **networking**
- Requires integration of **asynchronous** execution in Gaudi

Conclusion

- Thanks to the hard work of **many**, TRACCC is a **functional**, **performant** track reconstruction pipeline in ACTS
- Track reconstruction is **difficult to implement** for GPGPUs due to **irregularity**
- **Solutions** to many hurdles **researched** and **developed**
- TRACCC currently provides **competitive performance** for **ATLAS EF tracking**
- To get **involved**: CERN Mattermost, ACTS (#traccc and friends), bi–weekly meeting

Backup slides

Backup – GPGPU computing