Towards GPU-based tracking in ACTS and ATLAS

Stephen Nicholas Swatman

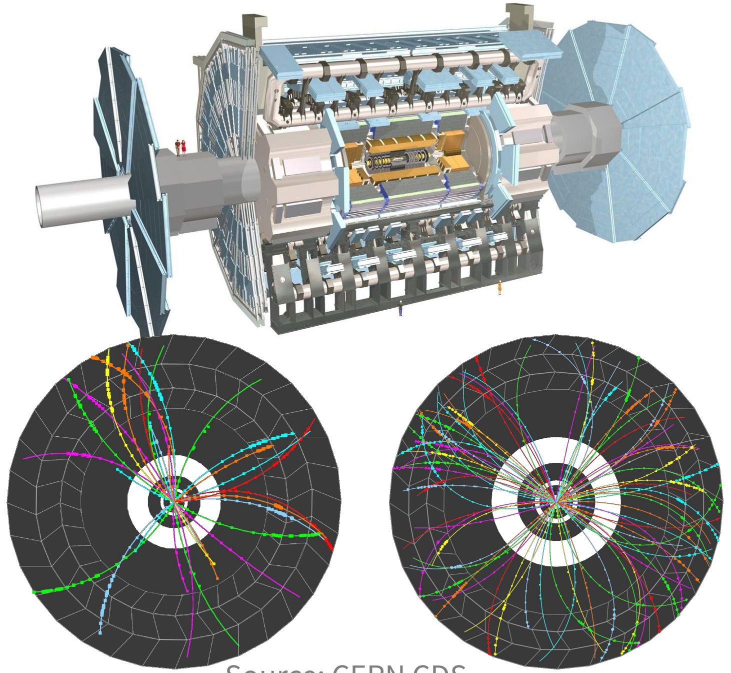
May 3, 2024

Introduction – About Me & My PhD

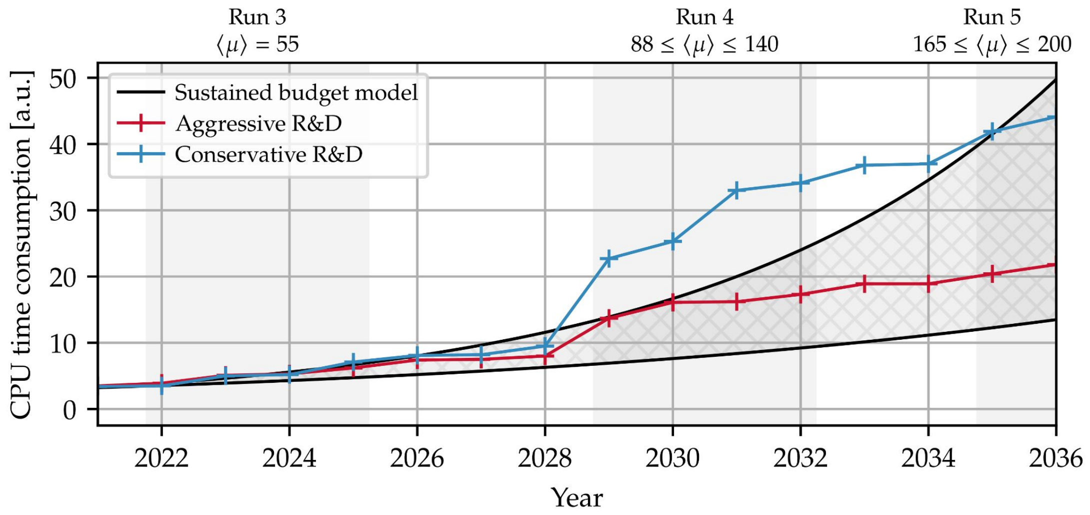
- Finished a MSc at **UvA** + **VU** in 2019, graduation project at **Nikhef**
- Started a PhD in **CS** at **UvA (PCS)** + **CERN** in 2020
- Now writing up – aim to defend in **autumn**
- *Golden staple*-like PhD thesis
- In this talk: very brief overview of my work, and some insights into **GPU tracking**

Problem Statement – Track Reconstruction

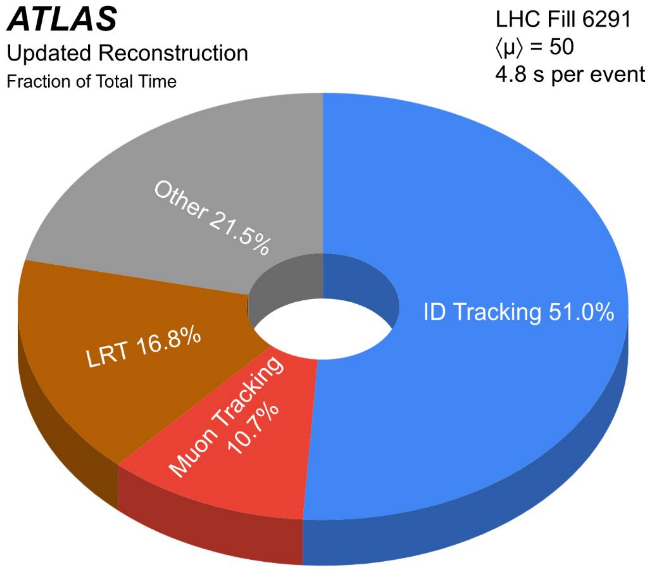
- Increased pile-up threatens to make track reconstruction infeasible for HL-HLC, FCC
- More **efficient compute** → more **available resources** → more interesting **physics**
- Compute complexity scales ~$O(\mu^2)$
- This will require research into novel **algorithms**, novel **hardware**, etc.
- In my case: how can this run on **GPUs**?


Source: CERN CDS

Problem Statement – Track Reconstruction



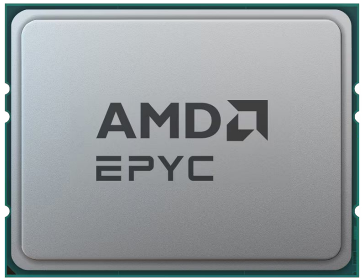
Numbers by ATLAS, plot by me



10.1007/s41781-023-00111-y

Massively Parallel Computing

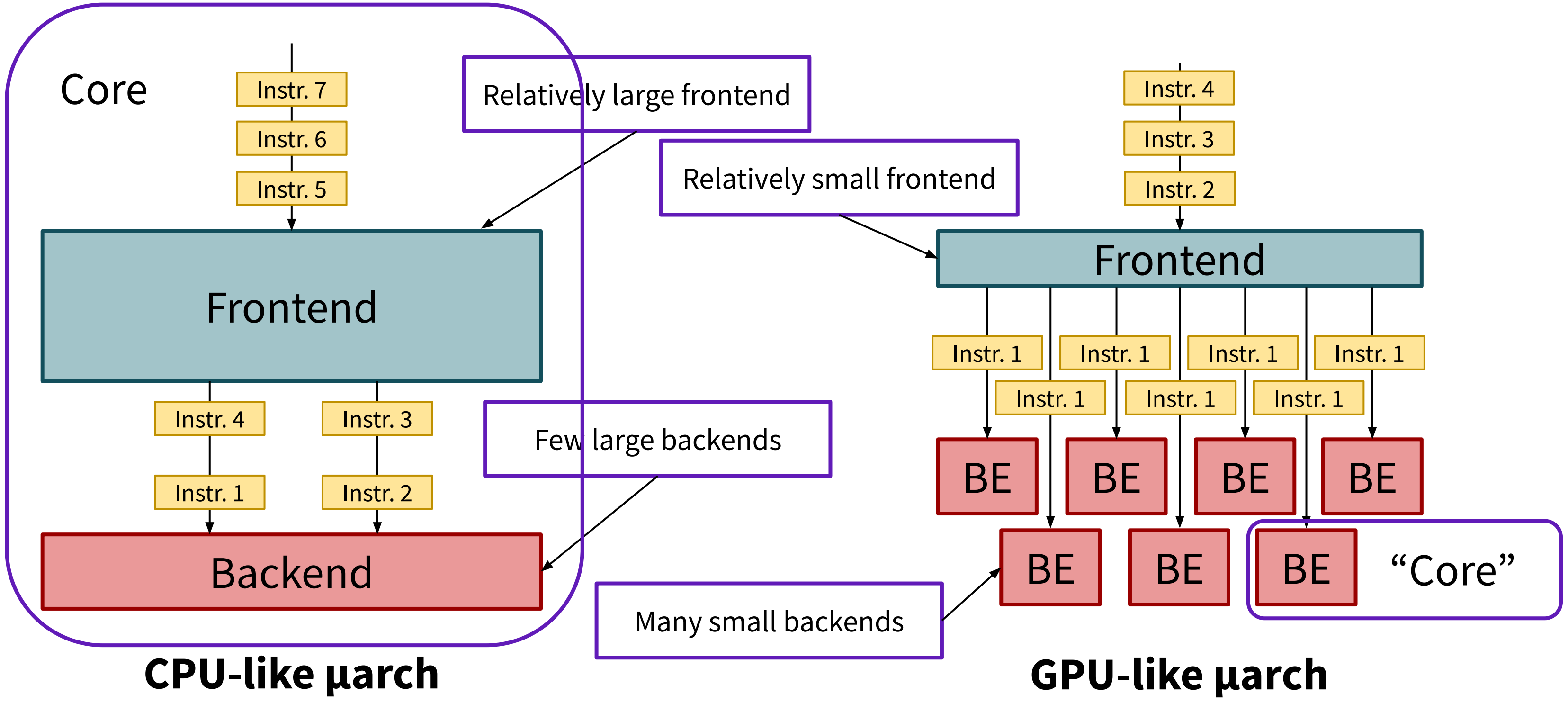
€10,000



AMD EPYC 9554
64 cores
360W TDP

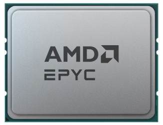
NVIDIA RTX 6000
18,176 cores
300W TDP

Massively Parallel Computing



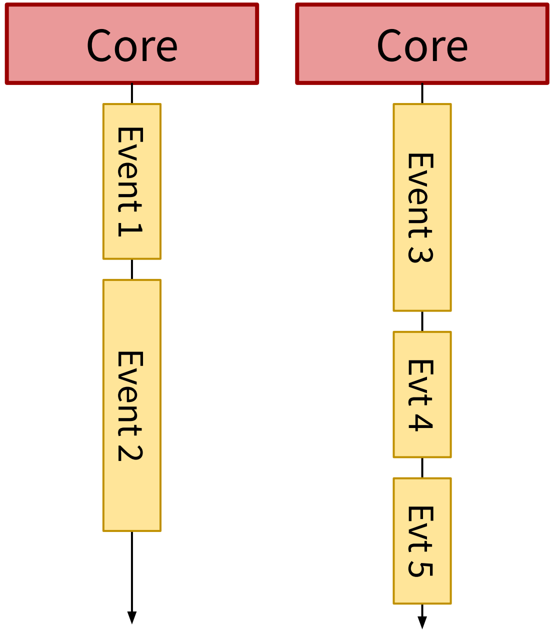
Massively Parallel Computing

Can be programmed independently

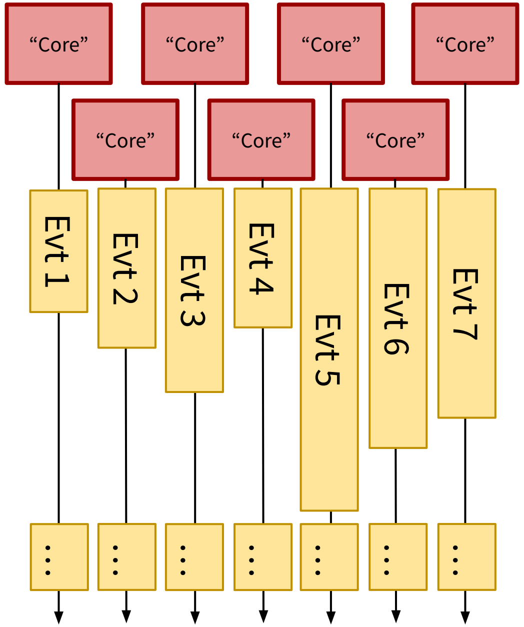
	Cores	×	Cycles/s	×	FLOP/cycle	=	FLOP/s
	64		3.75B		48		11.5 TFLOP/s
							↓ **8×**
	18,176		2.51B		2		91.1 TFLOP/s

Must be carefully programmed in lockstep

Massively Parallel Computing



CPU-like µarch



GPU-like µarch

Can track reconstruction be implemented efficiently on massively parallel systems?

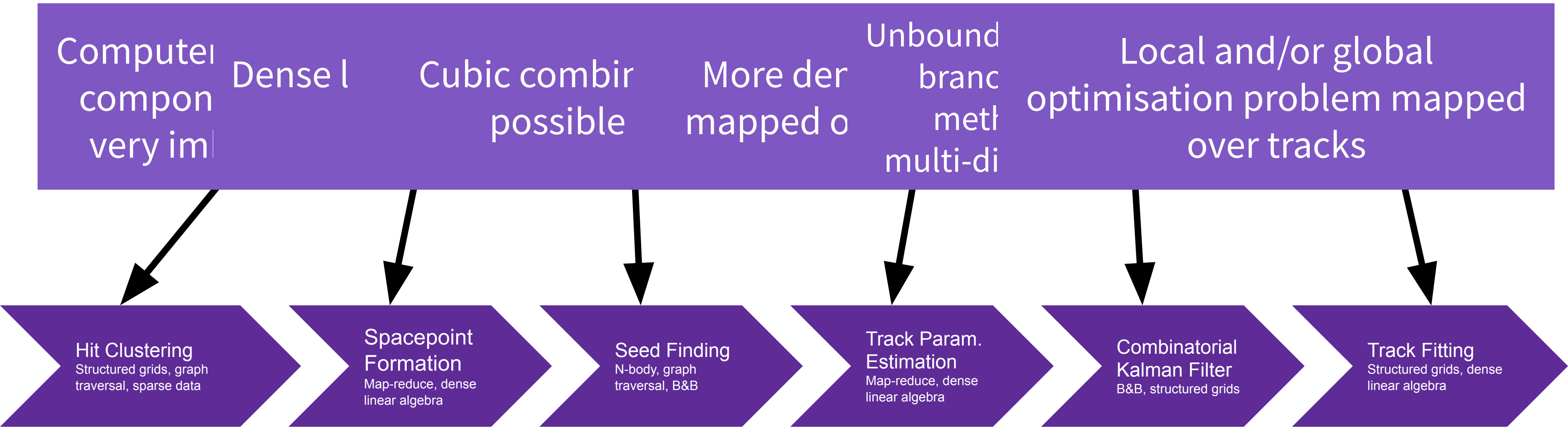
ACTS – A Common Tracking Software

- **ACTS** is **A Common Tracking Software**
- Goals are to be…
  - **Feature complete**
  - **Well-written**
  - **Extensible**
  - **Experiment-agnostic** (sPHENIX, ePIC, etc.)
  - an **R&D platform**
- Originally based on **ATLAS tracking**
- Now being migrated **back into Athena**

What are the challenges in developing track reconstruction algorithms to massively parallel architectures?

Problem Statement – State-of-the-Art

- Track reconstruction consists of a **task graph** of structurally different algorithms
- Here denoted according to their “13 dwarves” classification
- Varying degrees of **complexity** and **challenge**…

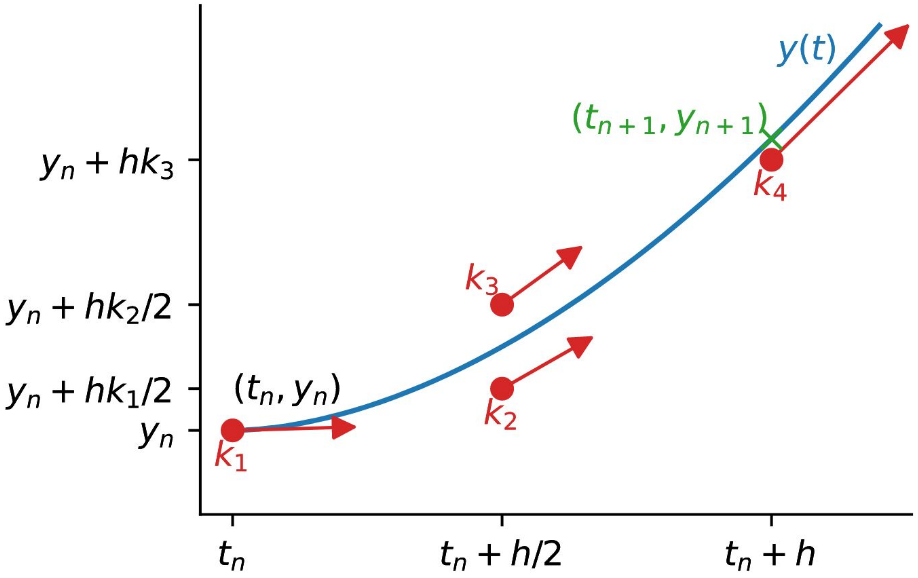


Computer compon very iml
Dense l
Cubic combir possible
More der mapped o
Unbound branc meth multi-di
Local and/or global optimisation problem mapped over tracks
Hit Clustering
Structured grids, graph traversal, sparse data
Spacepoint Formation
Map-reduce, dense linear algebra
Seed Finding
N-body, graph traversal, B&B
Track Param. Estimation
Map-reduce, dense linear algebra
Combinatorial Kalman Filter
B&B, structured grids
Track Fitting
Structured grids, dense linear algebra

How can structured grid data be represented in order to maximize the efficiency of arbitrary computations?

Research – Vector Fields – Amuse Bouche

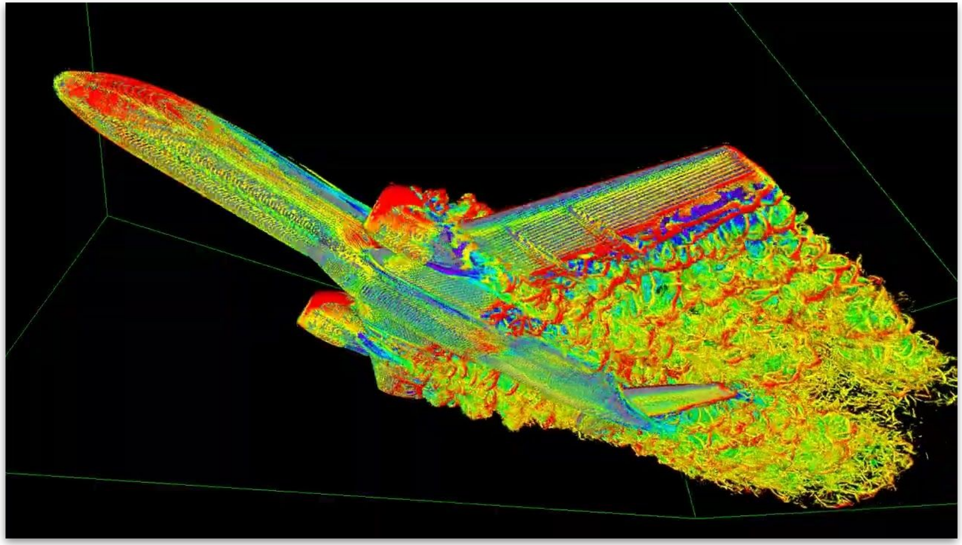
- Most experiment **magnetic fields** are fairly homogeneous, but not quite
- Those irregularities matter when e.g. **propagating particle motion**
- Storing ATLAS B-field: **~200 MB**
- **Accessed** millions of times per second!
- How do we do that **quickly**?
- We don't even know how to do than on CPUs, and if we did it would not translate!



Source: ACTS Project

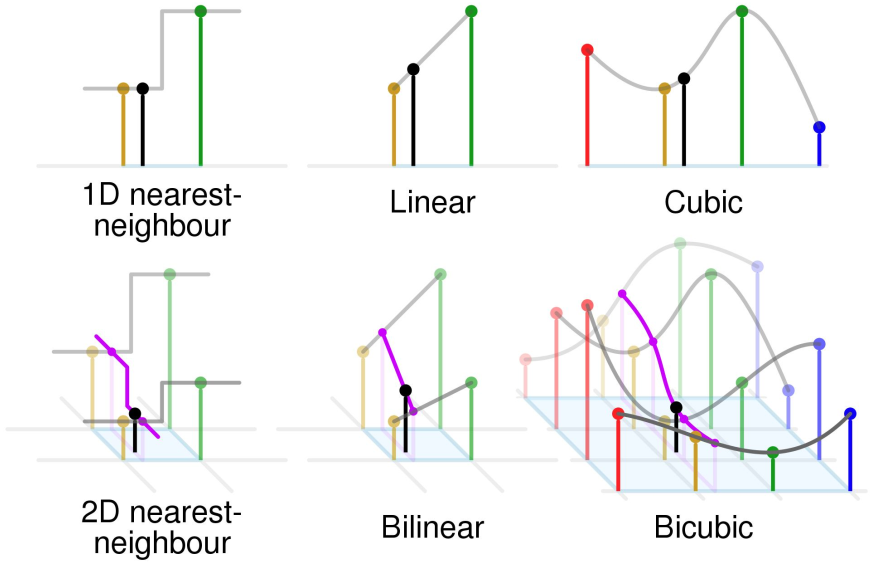
Research – Vector Fields

- **Multi-dimensional data** is everywhere HPC
  - Magnetic fields
  - But also CFD, lattice QCD, etc.
- Must be accessed very frequently in **hard-to-predict patterns** for iterative numerical methods
- Increasing **cache efficiency** can improve **performance** for these kernels
- **Design space is large**
  - Many **functional** and **non-functional** properties come into play


Source: Moritz Lehmann

Research – Vector Fields

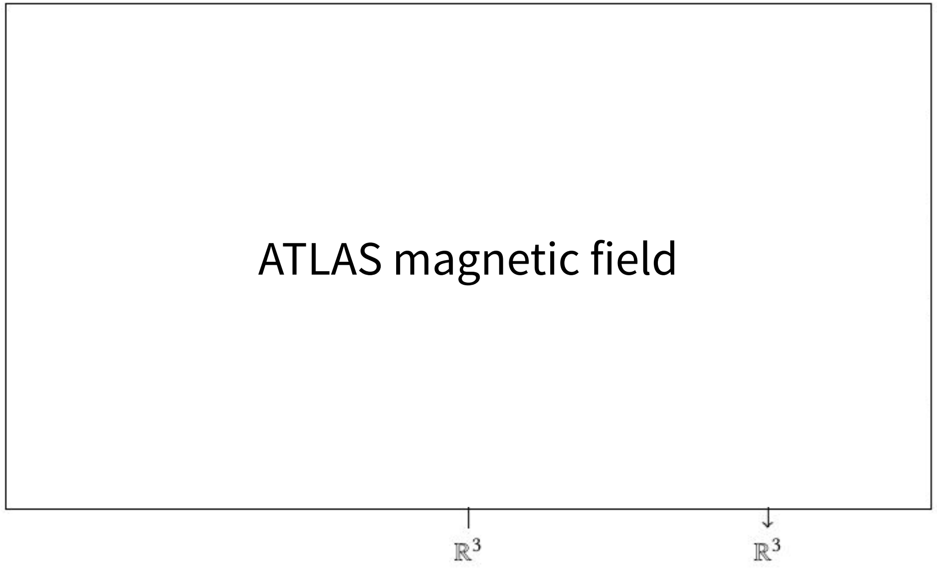
- Interpolation methods (F & EF)
  - NN, linear, cubic, etc.
- Boundary checking (F & EF)
  - Wrap, mirror, zero, etc.
- Array layout (EF)
  - Row-major, column-major, *Morton*, etc.
- Coordinate transformation (F & EF)
  - Affine, polar, etc.
- Storage location (EF)
  - Main memory, CUDA memory, texture memory, etc.
- Potentially many more!
- Across many **devices** and with many access patterns



Source: Wikipedia

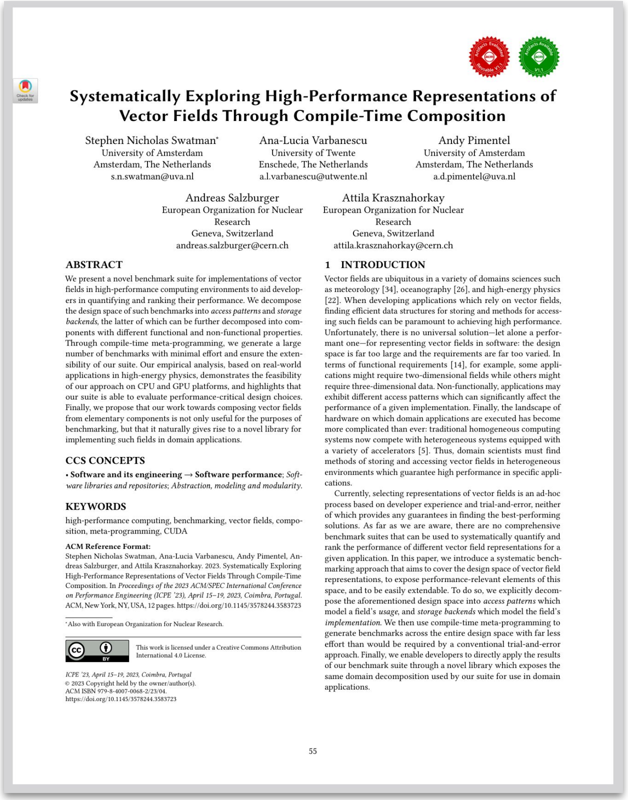
Research – Vector Fields

- We developed a **category-theoretical** method for **decomposing** this design space (like on the previous slide)
- Can be re-composed at **compile time** with zero run-time overhead
- Allows our method to serve as both a **benchmarking suite** and **design space exploration tool**…
- …as well as a **library** for implementing heterogeneous multi-dimensional arrays



Research – Vector Fields

- This work was published in **ICPE'23**
- And it was nominated as candidate to the **best paper award**!
- https://doi.org/10.1145/3578244.3583723



Systematically Exploring High-Performance Representations of Vector Fields Through Compile-Time Composition

Stephen Nicholas Swatman*
University of Amsterdam
Amsterdam, The Netherlands
s.n.swatman@uva.nl

Ana-Lucia Varbanescu
University of Twente
Enschede, The Netherlands
a.l.varbanescu@utwente.nl

Andy Pimentel
University of Amsterdam
Amsterdam, The Netherlands
a.d.pimentel@uva.nl

Andreas Salzburger
European Organization for Nuclear Research
Geneva, Switzerland
andreas.salzburger@cern.ch

Attila Krasznahorkay
European Organization for Nuclear Research
Geneva, Switzerland
attila.krasznahorkay@cern.ch

ABSTRACT

We present a novel benchmark suite for implementations of vector fields in high-performance computing environments to aid developers in quantifying and ranking their performance. We decompose the design space of such benchmarks into *access patterns* and *storage backends*, the latter of which can be further decomposed into components with different functional and non-functional properties. Through compile-time meta-programming, we generate a large number of benchmarks with minimal effort and ensure the extensibility of our suite. Our empirical analysis, based on real-world applications in high-energy physics, demonstrates the feasibility of our approach on CPU and GPU platforms, and highlights that our suite is able to evaluate performance-critical design choices. Finally, we propose that our work towards composing vector fields from elementary components is not only useful for the purposes of benchmarking, but that it naturally gives rise to a novel library for implementing such fields in domain applications.

CCS CONCEPTS

• **Software and its engineering → Software performance**; *Software libraries and repositories*; *Abstraction, modeling and modularity*.

KEYWORDS

high-performance computing, benchmarking, vector fields, composition, meta-programming, CUDA

ACM Reference Format:
Stephen Nicholas Swatman, Ana-Lucia Varbanescu, Andy Pimentel, Andreas Salzburger, and Attila Krasznahorkay. 2023. Systematically Exploring High-Performance Representations of Vector Fields Through Compile-Time Composition. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3578244.3583723

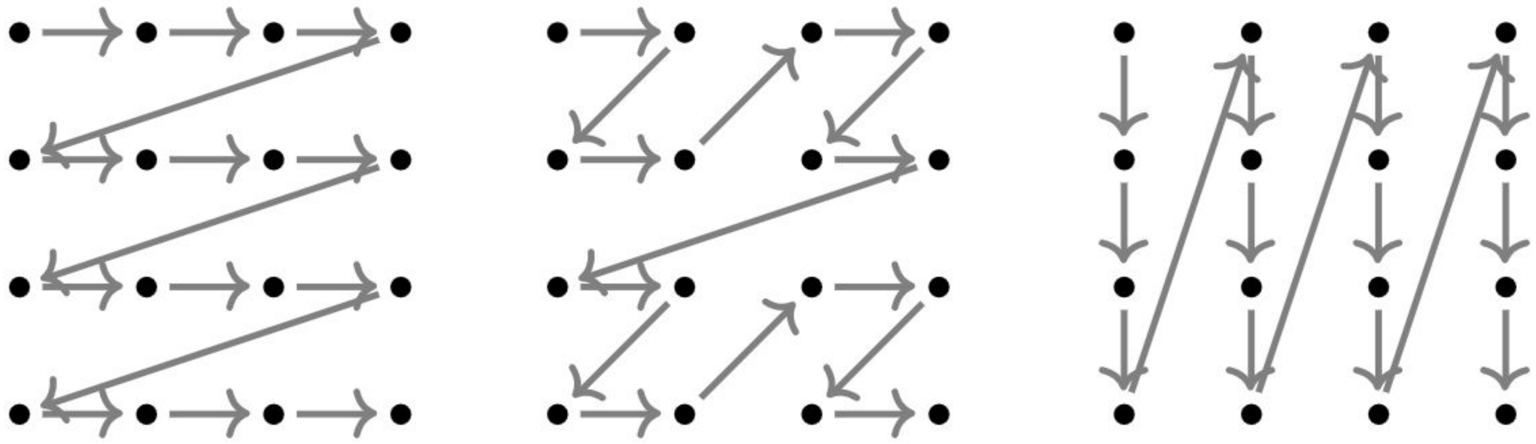
*Also with European Organization for Nuclear Research.

1 INTRODUCTION

Vector fields are ubiquitous in a variety of domains sciences such as meteorology [34], oceanography [26], and high-energy physics [22]. When developing applications which rely on vector fields, finding efficient data structures for storing and methods for accessing such fields can be paramount to achieving high performance. Unfortunately, there is no universal solution—let alone a performant one—for representing vector fields in software: the design space is far too large and the requirements are far too varied. In terms of functional requirements [14], for example, some applications might require two-dimensional fields while others might require three-dimensional data. Non-functionally, applications may exhibit different access patterns which can significantly affect the performance of a given implementation. Finally, the landscape of hardware on which domain applications are executed has become more complicated than ever: traditional homogeneous computing systems now compete with heterogeneous systems equipped with a variety of accelerators [5]. Thus, domain scientists must find methods of storing and accessing vector fields in heterogeneous environments which guarantee high performance in specific applications.

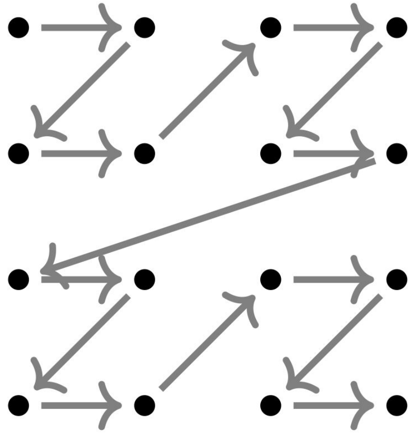
Currently, selecting representations of vector fields is an ad-hoc process based on developer experience and trial-and-error, neither of which provides any guarantees in finding the best-performing solutions. As far as we are aware, there are no comprehensive benchmark suites that can be used to systematically quantify and rank the performance of different vector field representations for a given application. In this paper, we introduce a systematic benchmarking approach that aims to cover the design space of vector field representations, to expose performance-relevant elements of this space, and to be easily extendable. To do so, we explicitly decompose the aforementioned design space into *access patterns* which model a field's *usage*, and *storage backends* which model the field's *implementation*. We then use compile-time meta-programming to generate benchmarks across the entire design space with far less effort than would be required by a conventional trial-and-error approach. Finally, we enable developers to directly apply the results of our benchmark suite through a novel library which exposes the same domain decomposition used by our suite for use in domain applications.

Research – Morton Layouts – Amuse Bouche



Research – Morton Layouts

- The Morton curve provides **balanced locality** compared to row-major and column-major layouts
- Calculated by **interleaving** the bits of the **binary expansions** of the input coordinates!
  - Can be done efficiently on modern commodity hardware
- But what if you were to interleave the bits in **arbitrary patterns**?



$$f(5,3,4) = f(101_2, 011_2, 100_2) = 101010110_2 = 342_{10}$$

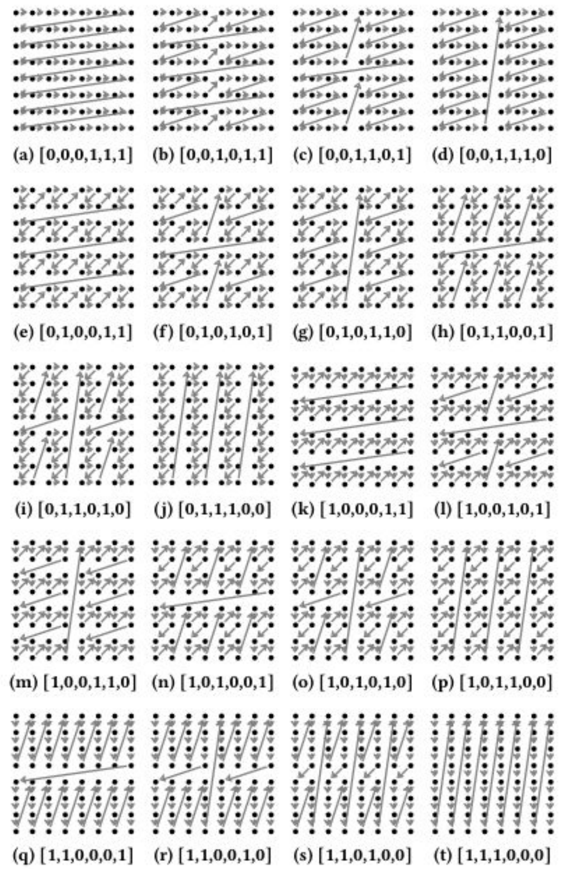
Research – Morton Layouts

$$f(1011_2, 0101_2) = \begin{array}{r} 01000101_2 \\ \underline{\vee 00100010_2} \\ 01100111_2 \end{array} = 103_{10}$$

$$f(1011_2, 0101_2) = \begin{array}{r} 01000110_2 \\ \underline{\vee 00010001_2} \\ 01010111_2 \end{array} = 87_{10}$$

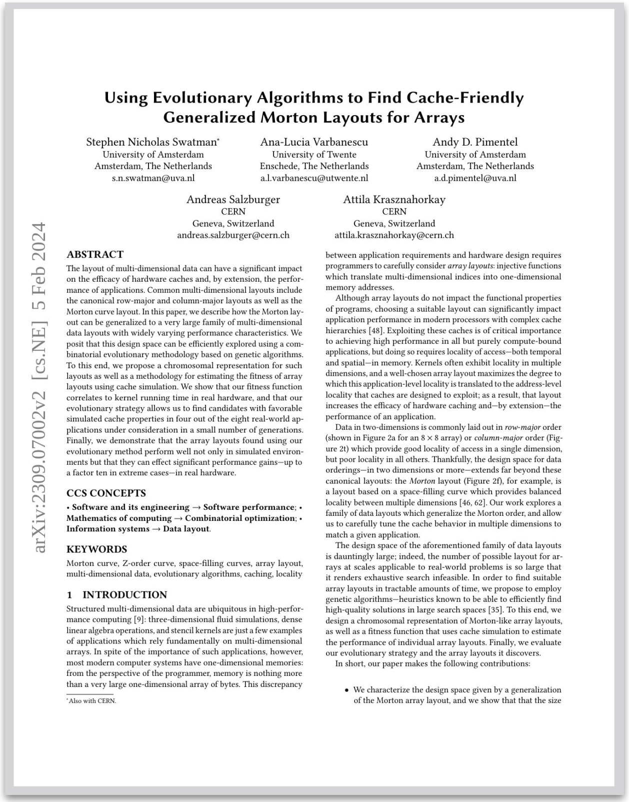
Research – Morton Layouts

- Turns out this gives you *very* large families of array layouts, all of which have **different cache properties**!
- We propose that **evolutionary algorithms** can be used to efficiently explore this design space
- We show that we can significantly **improve performance** – up to 10 times in extreme cases
- Automated, **problem-agnostic** optimisation method!



Research – Morton Layouts

- This was accepted to **ICPE’24**
- To be presented in **London** next week!
- https://doi.org/10.1145/3629526.3645034 (not yet active)



arXiv:2309.07002v2 [cs.NE] 5 Feb 2024

Using Evolutionary Algorithms to Find Cache-Friendly Generalized Morton Layouts for Arrays

Stephen Nicholas Swatman[*]
University of Amsterdam
Amsterdam, The Netherlands
s.n.swatman@uva.nl

Ana-Lucia Varbanescu
University of Twente
Enschede, The Netherlands
a.l.varbanescu@utwente.nl

Andy D. Pimentel
University of Amsterdam
Amsterdam, The Netherlands
a.d.pimentel@uva.nl

Andreas Salzburger
CERN
Geneva, Switzerland
andreas.salzburger@cern.ch

Attila Krasznahorkay
CERN
Geneva, Switzerland
attila.krasznahorkay@cern.ch

ABSTRACT

The layout of multi-dimensional data can have a significant impact on the efficacy of hardware caches and, by extension, the performance of applications. Common multi-dimensional layouts include the canonical row-major and column-major layouts as well as the Morton curve layout. In this paper, we describe how the Morton layout can be generalized to a very large family of multi-dimensional data layouts with widely varying performance characteristics. We posit that this design space can be efficiently explored using a combinatorial evolutionary methodology based on genetic algorithms. To this end, we propose a chromosomal representation for such layouts as well as a methodology for estimating the fitness of array layouts using cache simulation. We show that our fitness function correlates to kernel running time in real hardware, and that our evolutionary strategy allows us to find candidates with favorable simulated cache properties in four out of the eight real-world applications under consideration in a small number of generations. Finally, we demonstrate that the array layouts found using our evolutionary method perform well not only in simulated environments but that they can effect significant performance gains—up to a factor ten in extreme cases—in real hardware.

CCS CONCEPTS

• **Software and its engineering → Software performance**; • **Mathematics of computing → Combinatorial optimization**; • **Information systems → Data layout**.

KEYWORDS

Morton curve, Z-order curve, space-filling curves, array layout, multi-dimensional data, evolutionary algorithms, caching, locality

1 INTRODUCTION

Structured multi-dimensional data are ubiquitous in high-performance computing [9]: three-dimensional fluid simulations, dense linear algebra operations, and stencil kernels are just a few examples of applications which rely fundamentally on multi-dimensional arrays. In spite of the importance of such applications, however, most modern computer systems have one-dimensional memories: from the perspective of the programmer, memory is nothing more than a very large one-dimensional array of bytes. This discrepancy between application requirements and hardware design requires programmers to carefully consider *array layouts*: injective functions which translate multi-dimensional indices into one-dimensional memory addresses.

Although array layouts do not impact the functional properties of programs, choosing a suitable layout can significantly impact application performance in modern processors with complex cache hierarchies [48]. Exploiting these caches is of critical importance to achieving high performance in all but purely compute-bound applications, but doing so requires locality of access—both temporal and spatial—in memory. Kernels often exhibit locality in multiple dimensions, and a well-chosen array layout maximizes the degree to which this application-level locality is translated to the address-level locality that caches are designed to exploit; as a result, that layout increases the efficacy of hardware caching and—by extension—the performance of an application.

Data in two-dimensions is commonly laid out in *row-major* order (shown in Figure 2a for an 8×8 array) or *column-major* order (Figure 2t) which provide good locality of access in a single dimension, but poor locality in all others. Thankfully, the design space for data orderings—in two dimensions or more—extends far beyond these canonical layouts: the *Morton* layout (Figure 2f), for example, is a layout based on a space-filling curve which provides balanced locality between multiple dimensions [46, 62]. Our work explores a family of data layouts which generalize the Morton order, and allow us to carefully tune the cache behavior in multiple dimensions to match a given application.

The design space of the aforementioned family of data layouts is dauntingly large; indeed, the number of possible layout for arrays at scales applicable to real-world problems is so large that it renders exhaustive search infeasible. In order to find suitable array layouts in tractable amounts of time, we propose to employ genetic algorithms—heuristics known to be able to efficiently find high-quality solutions in large search spaces [35]. To this end, we design a chromosomal representation of Morton-like array layouts, as well as a fitness function that uses cache simulation to estimate the performance of individual array layouts. Finally, we evaluate our evolutionary strategy and the array layouts it discovers.

In short, our paper makes the following contributions:

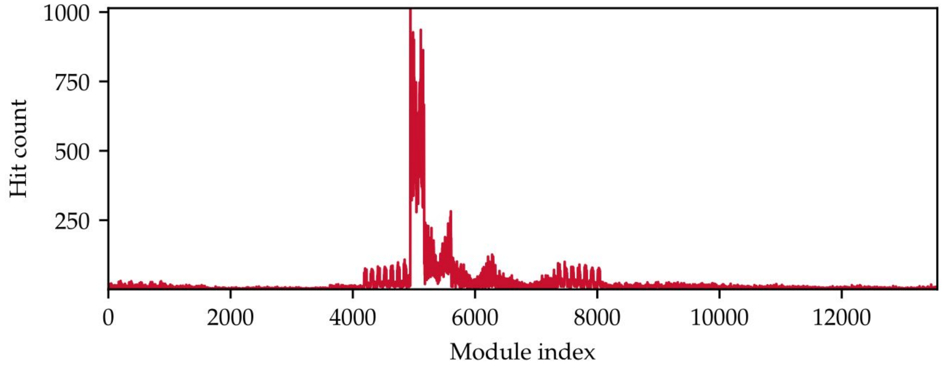
- We characterize the design space given by a generalization of the Morton array layout, and we show that that the size

[*] Also with CERN.

How can the effects of thread imbalance in SIMT workloads be modelled and how can they be mitigated?

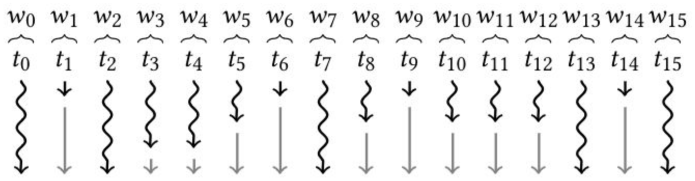
Research – Thread Imbalance – Amuse Bouche

- We know we cannot assign one event to one thread: **lockstep execution**
- Turns out we also cannot assign one module to one thread: **too imbalanced**
- We will need **fundamentally** different algorithms for **clustering**
- But can we **predict** what will work?
  - And can we use the graph on the right?

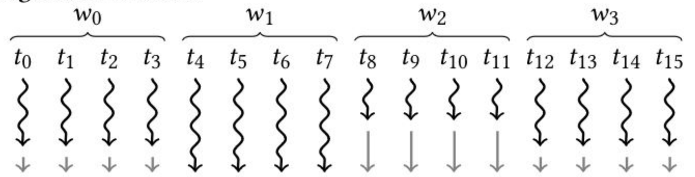


Research – Thread Imbalance

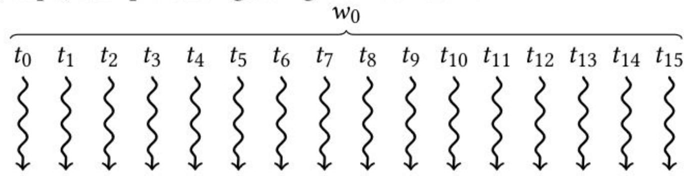
- A lot of our kernels are **imbalanced**: different threads execute **different amounts of work**
  - Different cluster sizes
  - Different branching factors
- This is not a problem on CPUs, but **strongly impacts GPU performance**
- Can be mitigated using **thread coarsening** or **thread refinement**
- But what is the **performance impact** of these techniques?


w_0 w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9 w_{10} w_{11} w_{12} w_{13} w_{14} w_{15}
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15}

(a) Per-thread granularity: each thread in a thread group processes a single unit of work.


w_0 w_1 w_2 w_3
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15}

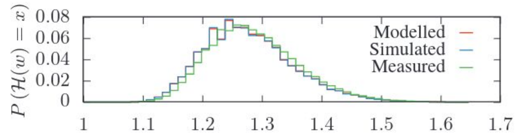
(b) Intermediate granularity: threads in a thread group form sub-groups, each processing a single unit of work.


w_0
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15}

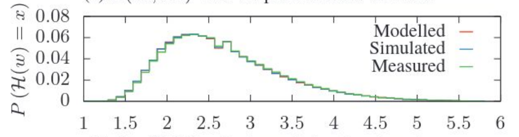
(c) Per-group granularity: all threads in a thread group process a single unit of work.

Research – Thread Imbalance

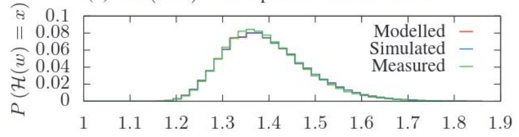
- We propose a **statistical model** that can infer the overhead of SIMT execution
- Metric for "*how suitable for GPU execution is this workload*"
- Uses **only** distribution of **thread load**: no hardware details, etc.
- Allows **early evaluation** of feasibility of different **parallelism strategies**!



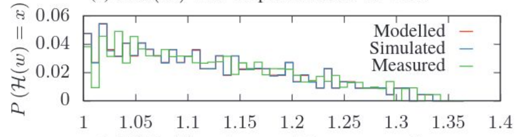
(a) B(40, 0.5) with 16 parallel units of work.



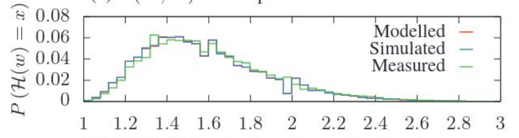
(b) Geo(0.05) with 8 parallel units of work.



(c) Pois(30) with 32 parallel units of work.



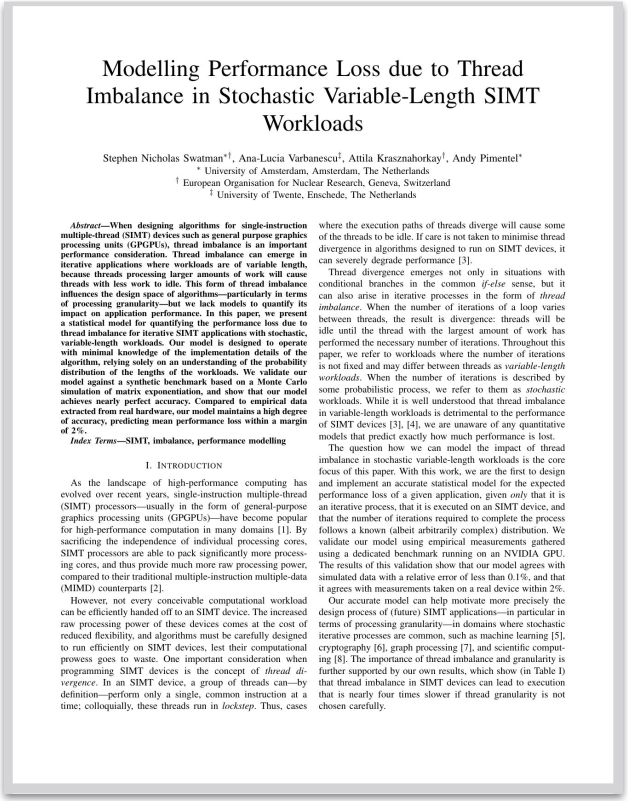
(d) U(20, 40) with 2 parallel units of work.



(e) NB(5, 0.3) with 4 parallel units of work.

Research – Thread Imbalance

- This work was published in **MASCOTS'22**
- Which was nice and also in Nice
- https://doi.org/10.1109/MASCOTS56607.2022.00026



Modelling Performance Loss due to Thread Imbalance in Stochastic Variable-Length SIMT Workloads

Stephen Nicholas Swatman[*][†], Ana-Lucia Varbanescu[‡], Attila Krasznahorkay[†], Andy Pimentel[*]
[*] University of Amsterdam, Amsterdam, The Netherlands
[†] European Organisation for Nuclear Research, Geneva, Switzerland
[‡] University of Twente, Enschede, The Netherlands

***Abstract*—When designing algorithms for single-instruction multiple-thread (SIMT) devices such as general purpose graphics processing units (GPGPUs), thread imbalance is an important performance consideration. Thread imbalance can emerge in iterative applications where workloads are of variable length, because threads processing larger amounts of work will cause threads with less work to idle. This form of thread imbalance influences the design space of algorithms—particularly in terms of processing granularity—but we lack models to quantify its impact on application performance. In this paper, we present a statistical model for quantifying the performance loss due to thread imbalance for iterative SIMT applications with stochastic, variable-length workloads. Our model is designed to operate with minimal knowledge of the implementation details of the algorithm, relying solely on an understanding of the probability distribution of the lengths of the workloads. We validate our model against a synthetic benchmark based on a Monte Carlo simulation of matrix exponentiation, and show that our model achieves nearly perfect accuracy. Compared to empirical data extracted from real hardware, our model maintains a high degree of accuracy, predicting mean performance loss within a margin of 2%.**

***Index Terms*—SIMT, imbalance, performance modelling**

I. INTRODUCTION

As the landscape of high-performance computing has evolved over recent years, single-instruction multiple-thread (SIMT) processors—usually in the form of general-purpose graphics processing units (GPGPUs)—have become popular for high-performance computation in many domains [1]. By sacrificing the independence of individual processing cores, SIMT processors are able to pack significantly more processing cores, and thus provide much more raw processing power, compared to their traditional multiple-instruction multiple-data (MIMD) counterparts [2].

However, not every conceivable computational workload can be efficiently handed off to an SIMT device. The increased raw processing power of these devices comes at the cost of reduced flexibility, and algorithms must be carefully designed to run efficiently on SIMT devices, lest their computational prowess goes to waste. One important consideration when programming SIMT devices is the concept of *thread divergence*. In an SIMT device, a group of threads can—by definition—perform only a single, common instruction at a time; colloquially, these threads run in *lockstep*. Thus, cases where the execution paths of threads diverge will cause some of the threads to be idle. If care is not taken to minimise thread divergence in algorithms designed to run on SIMT devices, it can severely degrade performance [3].

Thread divergence emerges not only in situations with conditional branches in the common *if-else* sense, but it can also arise in iterative processes in the form of *thread imbalance*. When the number of iterations of a loop varies between threads, the result is divergence: threads will be idle until the thread with the largest amount of work has performed the necessary number of iterations. Throughout this paper, we refer to workloads where the number of iterations is not fixed and may differ between threads as *variable-length workloads*. When the number of iterations is described by some probabilistic process, we refer to them as *stochastic* workloads. While it is well understood that thread imbalance in variable-length workloads is detrimental to the performance of SIMT devices [3], [4], we are unaware of any quantitative models that predict exactly how much performance is lost.

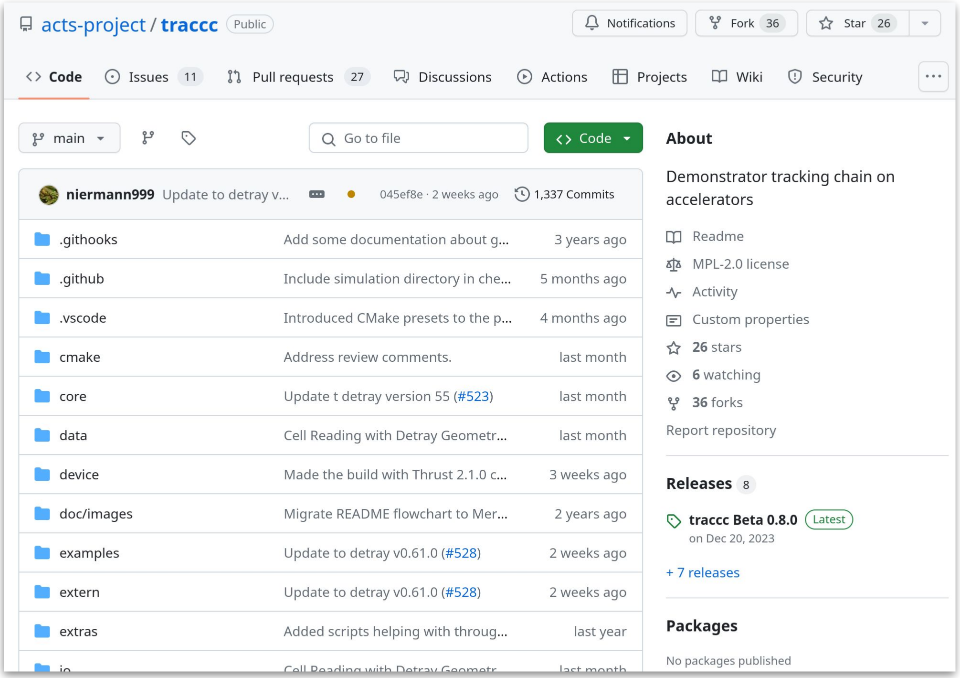
The question how we can model the impact of thread imbalance in stochastic variable-length workloads is the core focus of this paper. With this work, we are the first to design and implement an accurate statistical model for the expected performance loss of a given application, given *only* that it is an iterative process, that it is executed on an SIMT device, and that the number of iterations required to complete the process follows a known (albeit arbitrarily complex) distribution. We validate our model using empirical measurements gathered using a dedicated benchmark running on an NVIDIA GPU. The results of this validation show that our model agrees with simulated data with a relative error of less than 0.1%, and that it agrees with measurements taken on a real device within 2%.

Our accurate model can help motivate more precisely the design process of (future) SIMT applications—in particular in terms of processing granularity—in domains where stochastic iterative processes are common, such as machine learning [5], cryptography [6], graph processing [7], and scientific computing [8]. The importance of thread imbalance and granularity is further supported by our own results, which show (in Table I) that thread imbalance in SIMT devices can lead to execution that is nearly four times slower if thread granularity is not chosen carefully.

How do the extra-functional properties of novel track reconstruction algorithms compare to state-of-the-art solutions?

traccc – GPU Tracking Demonstrator

- Now: back to the application under study
- We have developed a **track reconstruction chain** for massively parallel devices
- Integrates **novel algorithms**
- Tracking on **TrackML** and **ODD**-like detectors


acts-project / traccc Public

niermann999 Update to detray v... 045ef8e · 2 weeks ago 1,337 Commits

.githooks	Add some documentation about g...	3 years ago
.github	Include simulation directory in che...	5 months ago
.vscode	Introduced CMake presets to the p...	4 months ago
cmake	Address review comments.	last month
core	Update t detray version 55 (#523)	last month
data	Cell Reading with Detray Geometr...	last month
device	Made the build with Thrust 2.1.0 c...	3 weeks ago
doc/images	Migrate README flowchart to Mer...	2 years ago
examples	Update to detray v0.61.0 (#528)	2 weeks ago
extern	Update to detray v0.61.0 (#528)	2 weeks ago
extras	Added scripts helping with throug...	last year

About

Demonstrator tracking chain on accelerators

Readme
MPL-2.0 license
Activity
Custom properties
26 stars
6 watching
36 forks
Report repository

Releases 8

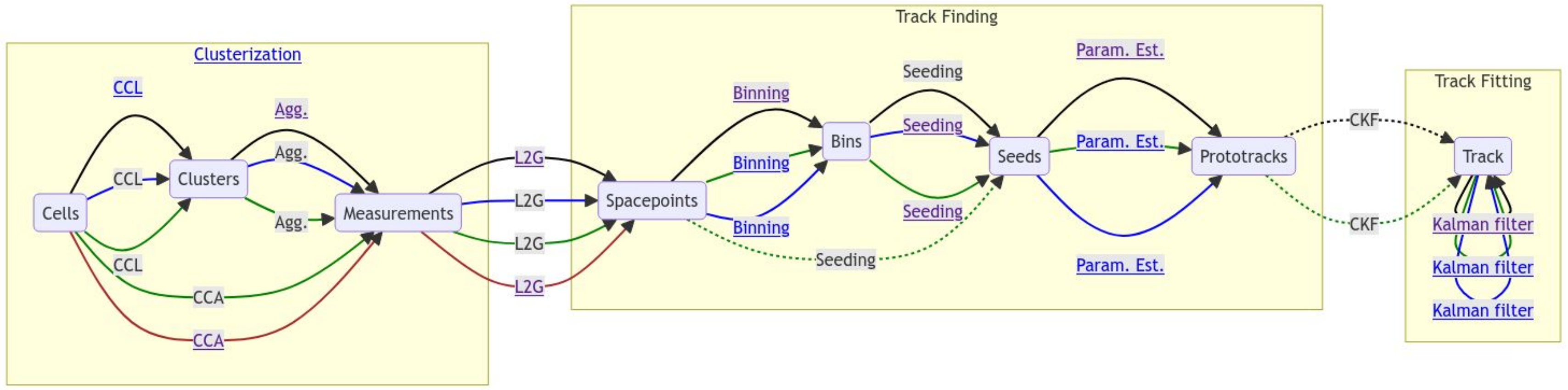
traccc Beta 0.8.0 Latest
on Dec 20, 2023

+ 7 releases

Packages

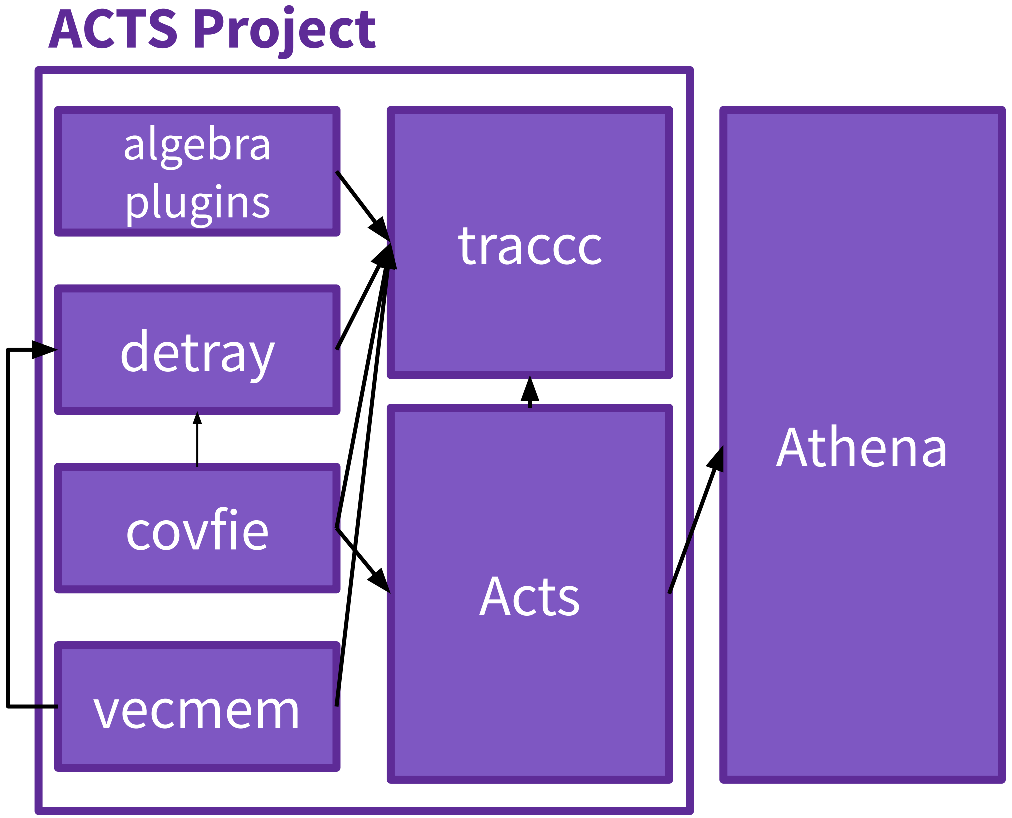
No packages published

traccc – GPU Tracking Demonstrator



ACTS Project – Subprojects

- R&D consists of many **subprojects** for **HEP** and **HPC** in general
- **traccc**: tracking demonstrator
- **algebra-plugins**: linear algebra
- **detray**: detector description
- **vecmem**: memory management
- **covfie**: vector field storage
    - Plasma physics at HZDR



Key Point – Reproducibility

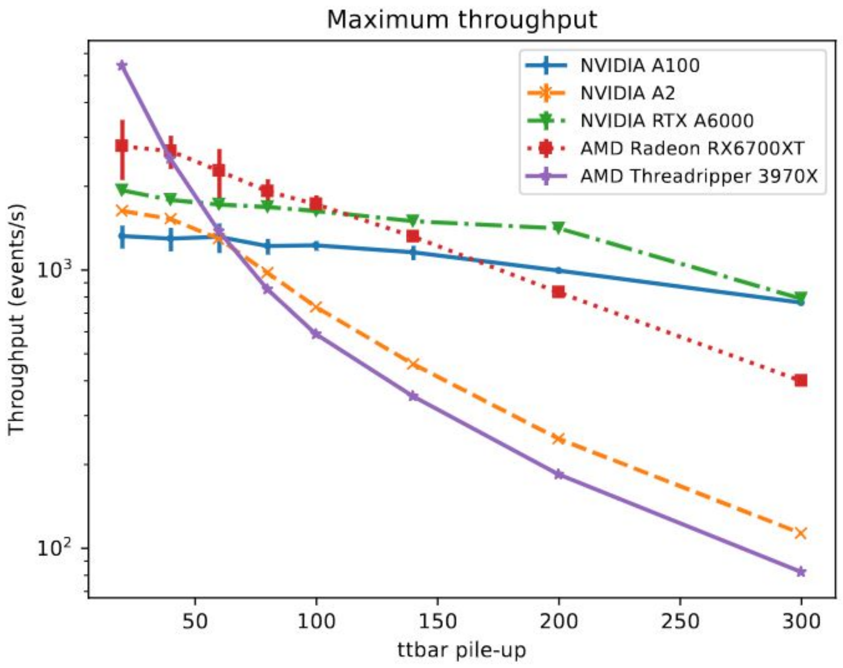
- Aim: to write software and develop methods that can be **used and improved on**
  - LHC lifetime: 17 more years
- Try to avoid "PhDware": software that becomes unusable after the end of the PhD
- **Artifact** evaluation tracks take extra effort but reward handsomely
  - Nice **stickers**
  - In some cases taken into account for reviews, rebuttals, etc.
- Also served on AE committees for SC, CGO, ICPE, and ICPP

Artifacts Available
acm
V1.1
Artifacts Evaluated
acm
Reusable V1.1
Results Reproduced
acm
V1.1

Source: ACM

Results

- *Preliminary* results on **TrackML** data show that our GPU-based solutions work well
- **Outperform** similarly priced CPUs at **higher pile-up values**
- **Factor ~10** gain in throughput



Source: Guilherme Metelo Rita de Almeida

Outstanding Challenges

- So far tested only on **simple geometries**: how do we integrate e.g. **ITk**
- **Combinatorial Kálmán Filter**: important step with high combinatorics
    - How do we **distribute branches** over threads?
- Integration into **Athena** is ongoing work
- How to **schedule** and **place** algorithms?
    - GPUs have **separate memories**: transfers are not free

Research – Throughput Models

- Our scheduling is an open problem, but can we somehow estimate the **throughput** of our **task graph on heterogeneous systems**…
- …using only the throughputs of the **individual kernels**?
- We propose that we can create an **optimistic upper bound** for this based on work in the data flow community
- Using **linear programming** we can solve a resource-constrained **maximum flow problem**!

$$\begin{tikzcd}
\boxed{d_2(A)} \arrow[r, "d_2(k_1)"] \arrow[d, shift right, "(d_2 \triangleright d_1)(A)"'] & \boxed{d_2(B)} \arrow[r, "d_2(k_2)"] \arrow[d, shift right, "(d_2 \triangleright d_1)(B)"'] & \boxed{d_2(C)} \arrow[d, shift right, "(d_2 \triangleright d_1)(C)"'] \\
\boxed{d_1(A)} \arrow[r, "d_1(k_1)"] \arrow[u, shift right, "(d_1 \triangleright d_2)(A)"'] & \boxed{d_1(B)} \arrow[r, "d_1(k_2)"] \arrow[u, shift right, "(d_1 \triangleright d_2)(B)"'] & \boxed{d_1(C)} \arrow[u, shift right, "(d_1 \triangleright d_2)(C)"']
\end{tikzcd}$$

$$\begin{array}{ll}
\text{maximise} & \sum_{e \in E^-(t)} \boldsymbol{x}_e f(e) \\
\text{subject to} & \forall v \in T' \setminus \{s, t\} : \sum_{e \in E^+(v)} \boldsymbol{x}_e f(e) = \sum_{e \in E^-(v)} \boldsymbol{x}_e f(e) \\
& \forall e \in E : 0 \leq \boldsymbol{x}_e \leq 1 \\
& \forall r \in D \cup I : 0 \leq \sum_{e \in Q(r)} \boldsymbol{x}_e \leq 1
\end{array}$$

Conclusions

- ATLAS needs **aggressive R&D** to tackle **high-μ compute challenges**
- **Massively parallel** track reconstruction under development in **ACTS**
- Despite **irregular workloads**, we can exploit **GPUs** well in **TrackML** and **ODD**
- **Performance** is very **competitive** at μ ≥ 100
- Complex **geometries**, **scheduling**, and **placement** remain open questions

The ACTS project
https://github.com/acts-project/

Bi-weekly R&D meeting
https://indico.cern.ch/category/16958/