Bernhard Manfred Gruber
CERN, CASUS, HZDR, TU Dresden

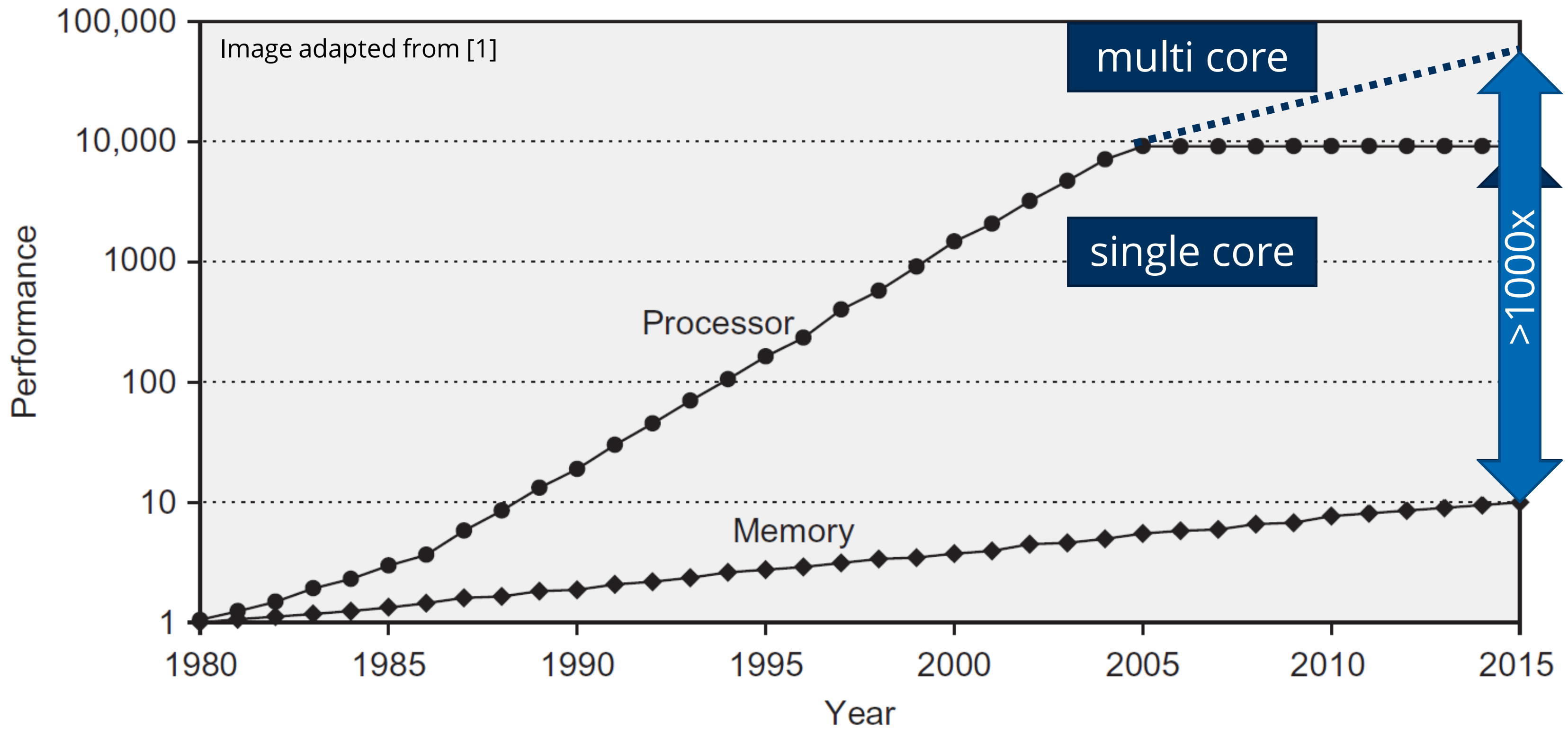
LLAMA: A Low-Level Abstraction of Memory Access

Memory layout optimization and efficient interconversion of data structures for heterogeneous architectures

23rd Gentner Day @ CERN // April 26th, 2023

SPONSORED BY THE Federal Ministry of Education and Research

Performance, latency and power gap



Google: 62.7% energy spent on data movement [3]

Access	Latency [ns]
Register	0.2
L1	1
L2	3-10
L3	10-20
Memory	50-100

~500x

Data from [1]

Operation	Cost [pJ]
32b FP Add	0.9
64b Cache	10-100
64b DRAM	1300-2600

~1000x

Data from [2]

Growth by hardware diversity

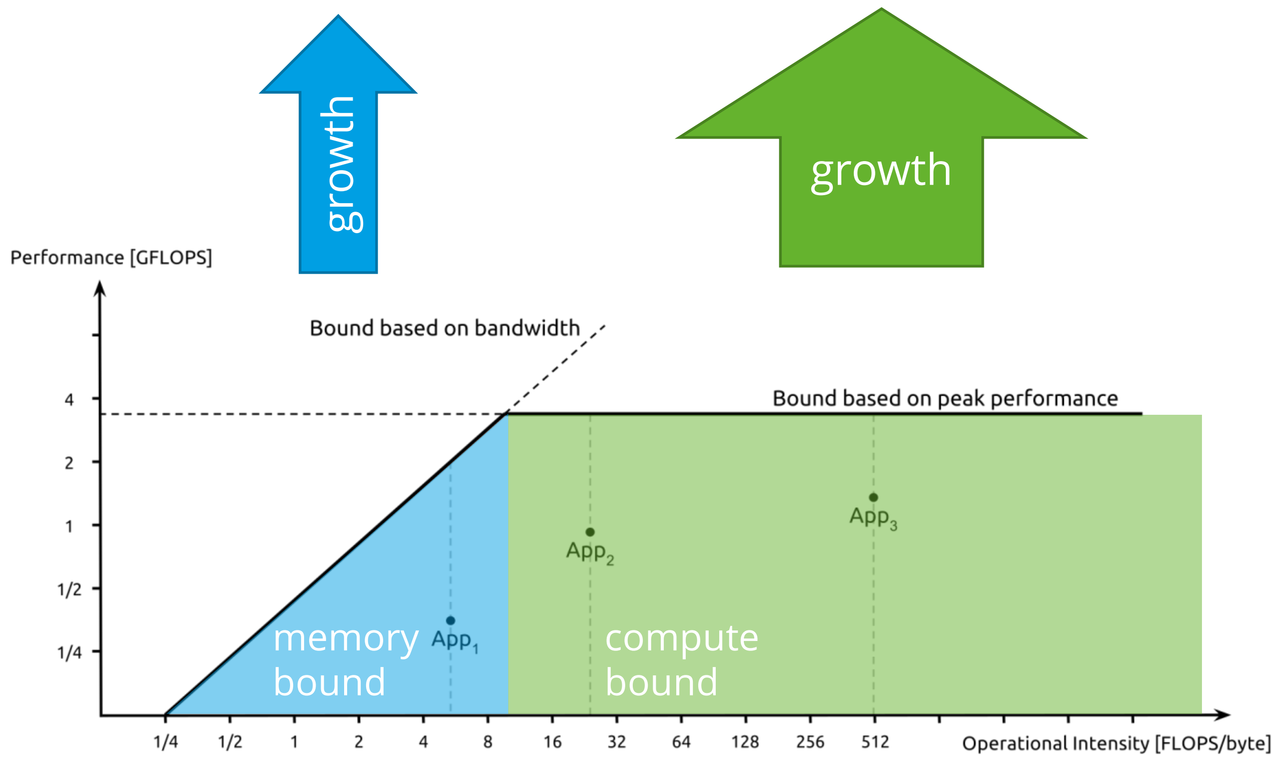


Image from [4]

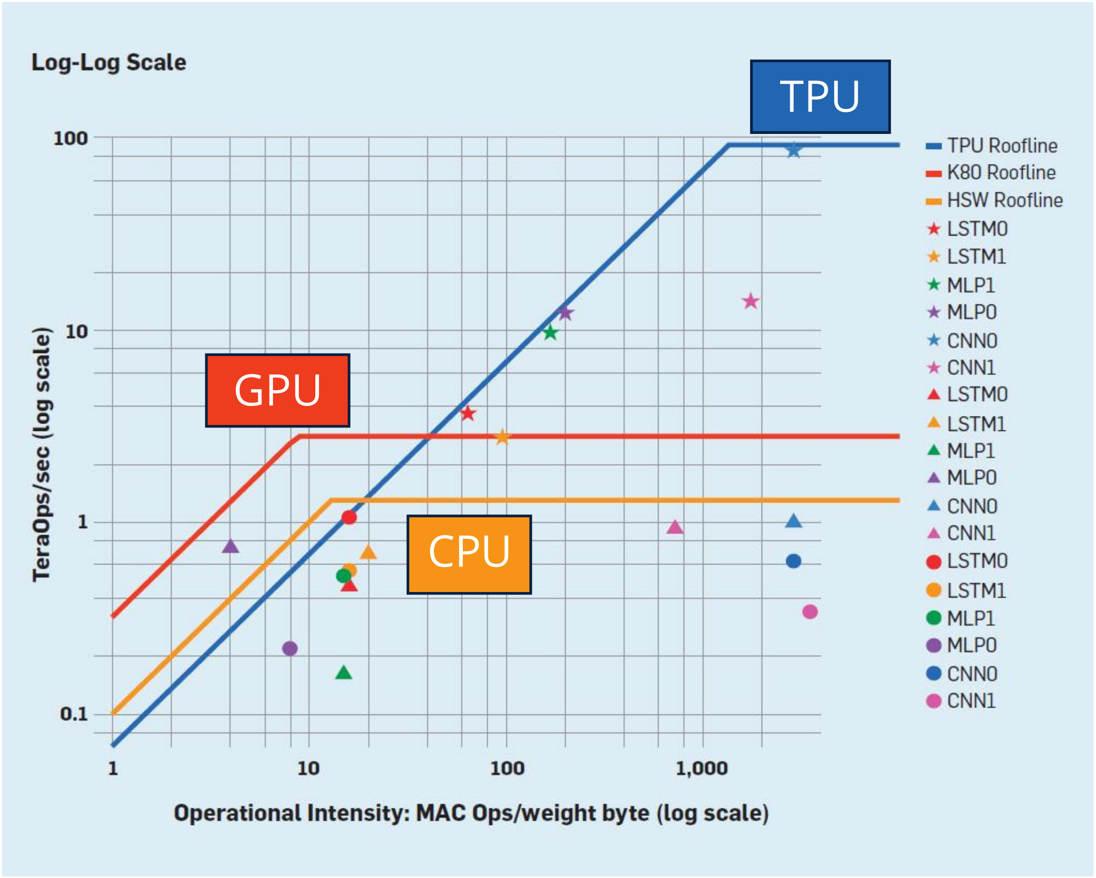


Image from [5]

Problems and challenges

Many programs are memory bound, or will soon become

Performance relies on maximizing data throughput, data locality, efficient parallel access

Optimizations depend on full control over data layout and memory access

Efficiently using modern hardware requires respecting its internal structure

Compute and memory hardware is increasingly heterogeneous

Porting to new architectures is expensive

Performance portability is challenging

Different hardware requires different data layouts and access patterns

Different data layouts require different indexing syntax

Changing data layout requires rewriting code

```
data[i].x
data.x[i]
data[i/8].x[i%8]
```

Indexing different data layouts.
Here: AoS, SoA and AoSoA8

Related work – taxonomy

Problem:
Memory perf/latency/power gaps vs. portability

Memory Hardware

Software

Domain specific solutions

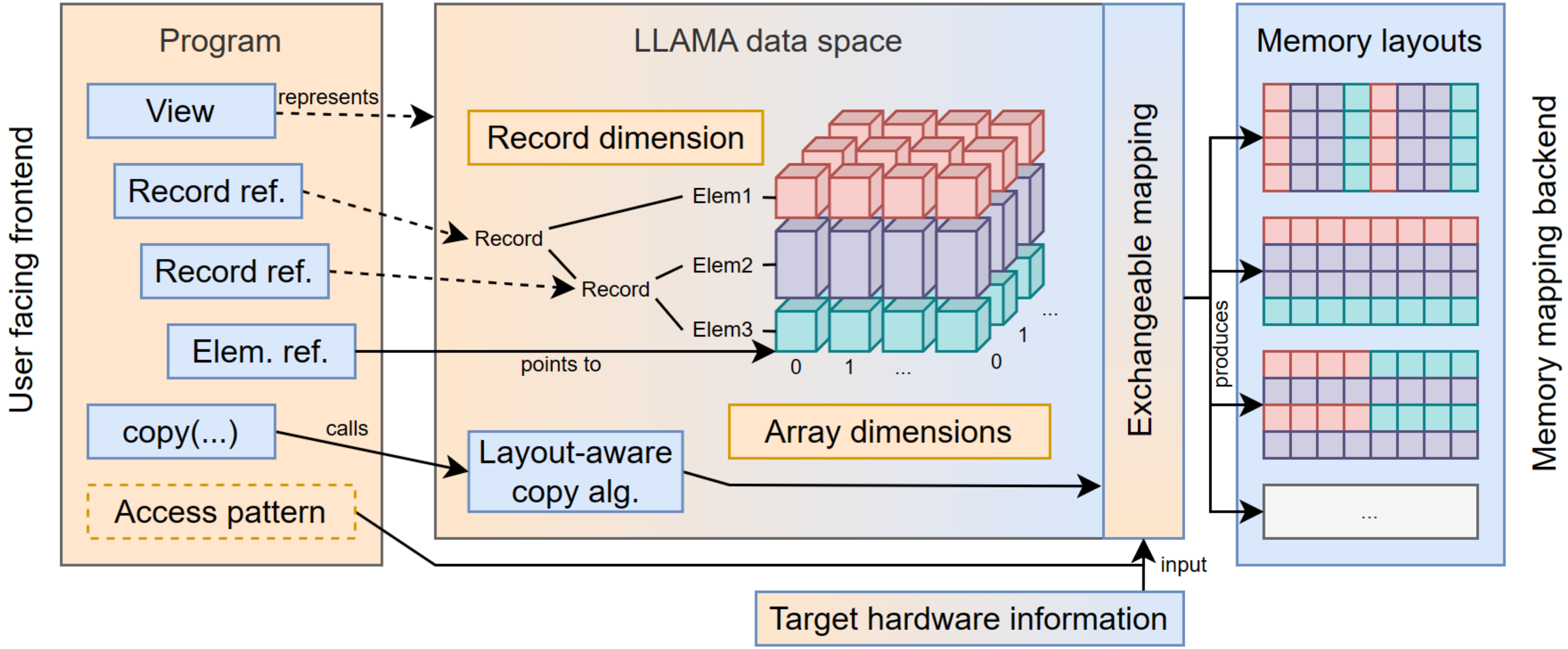
Generic techniques

Related work: Generic techniques

- Generic techniques
  - Data layout
    - Multi. dim. arrays
    - SoA
    - AoSoA and SIMD
    - Struct layout
    - Reflection
  - Memory access
  - Data reduction
  - Memory allocation
  - Memory profiling

No wholistic solution, combining all these optimizations techniques

Concept


User facing frontend
Program
View
represents
Record ref.
Record ref.
Elem. ref.
copy(...)
calls
Access pattern
LLAMA data space
Record dimension
Record
Record
Elem1
Elem2
Elem3
points to
0
1
...
0
1
...
Array dimensions
Layout-aware
copy alg.
Exchangeable mapping
produces
input
Target hardware information
Memory layouts
...
Memory mapping backend

Case study: HEP analysis

High-Energy-Physics (HEP) analyzes energetic particle collisions

Stored as events, 1 event = data of one collision inside a detector

Estimate: ~15k high-energy/nuclear physicists working in this field

B2HHH decay with LHCb Open Data [6,7] and ROOT [8]

Simple, but representative analysis

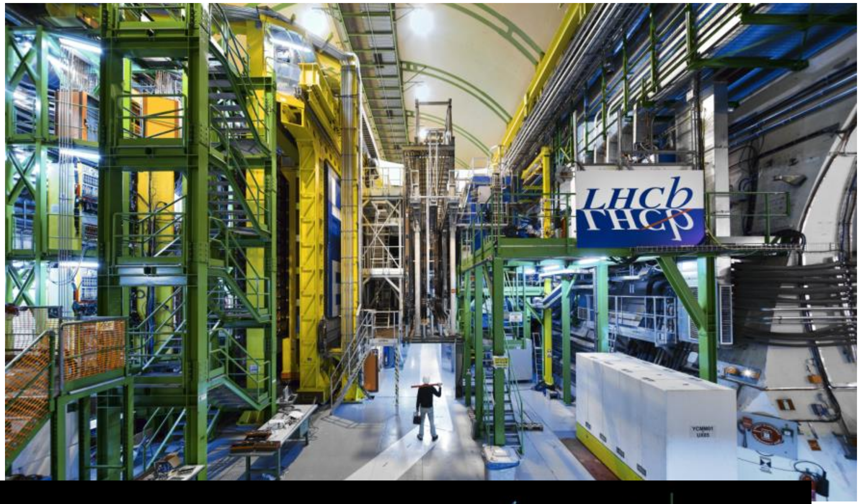
— "Real" analyses: more data/observables

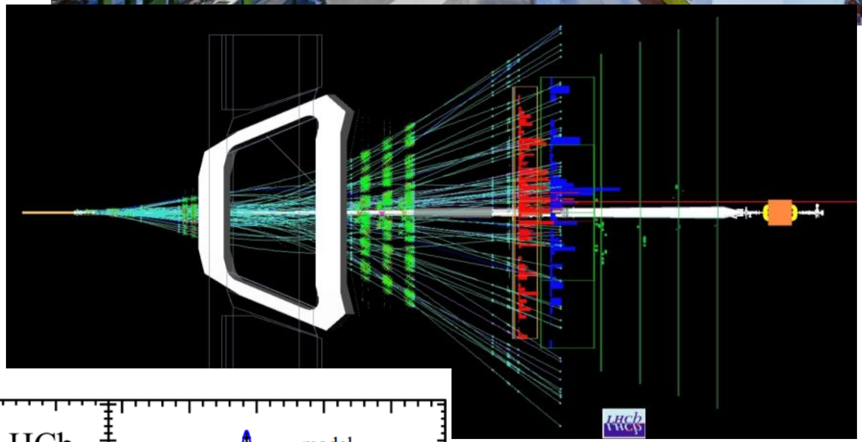
— Standard workflow: read events, filters/cuts, compute, histograms

— IO (network + disk) and memory bound

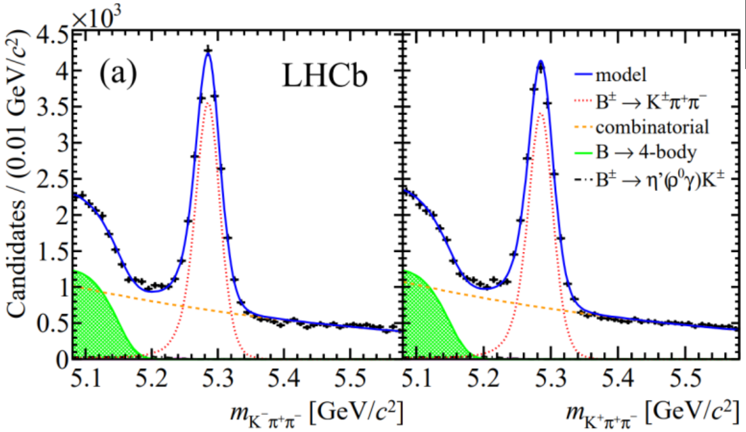
Only study in-memory data layouts, no IO

Baseline: SoA multi blob layout





Images © CERN



Histogram from [89]

Generate insight

Event loop
Filters and cuts
Comp. observ-able
Fill hist.

```
#pragma omp parallel for
for(size_t i = 0; i < n; i++) {
    auto&& event = view[i];

    if(event(H1isMuon{})) continue;
    if(event(H2isMuon{})) continue;
    if(event(H3isMuon{})) continue;

    if(event(H1ProbK{}) < probKCut) continue;
    if(event(H2ProbK{}) < probKCut) continue;
    if(event(H3ProbK{}) < probKCut) continue;

    if(event(H1ProbPi{}) > probPiCut) continue;
    if(event(H2ProbPi{}) > probPiCut) continue;
    if(event(H3ProbPi{}) > probPiCut) continue;

    // compute bmass ...

    hists[omp_get_thread_num()].Fill(bmass);
}
```

How often is which data touched?

Depends on filters …

Instrumentation to the rescue!

Count accesses to fields

2 LOCS in LLAMA

Field	Size	Reads
H1isMuon	4	8556118
H2isMuon	4	7368489
H3isMuon	4	6951588
H1PX	8	23895
H1PY	8	23895
H1PZ	8	23895
H1ProbK	8	6311517
H1ProbPi	8	26959
H2PX	8	23895
H2PY	8	23895
H2PZ	8	23895
H2ProbK	8	623038
H2ProbPi	8	26012
H3PX	8	23895
H3PY	8	23895
H3PZ	8	23895
H3ProbK	8	95742
H3ProbPi	8	25359
Total		**150MB**

Dataset size: 1574MB

Heatmap – AoS

H[1-3]isMuon hot

100% - 81.25%

H1PropK warm

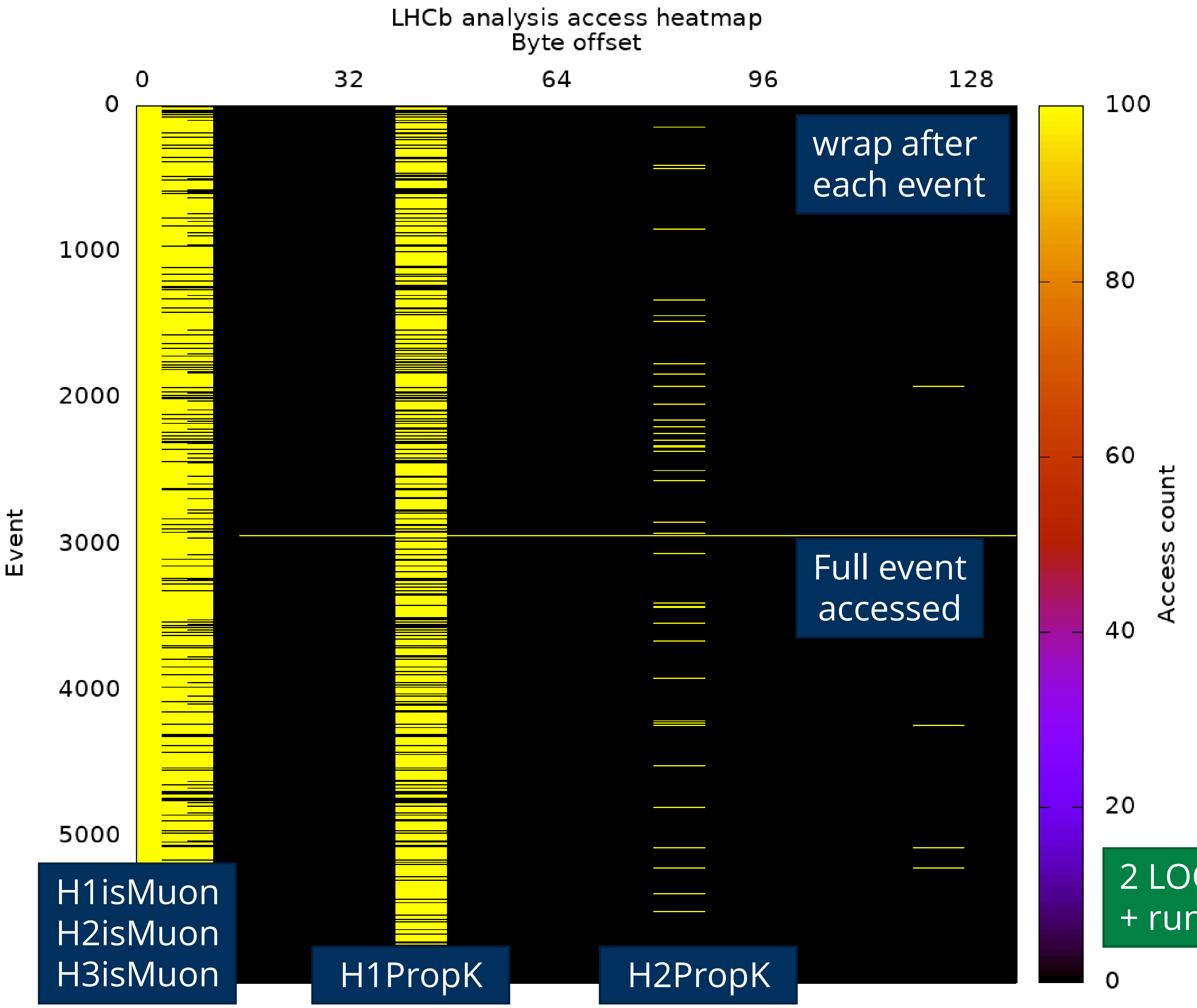
73.77%

H2PropK sparse

7.28%

Rest is cold

1.12% – 0.28%



2 LOCS in LLAMA + run plot script

Data reduction

H[1-3]isMuon are int32, but only store 0 or 1

Pack into 1 bit each

Reduce the FP precision

Exponent: look at min/max value

Mantissa: domain knowledge, theory, …

Our use case: 6 exp. and 16 man. bits

But: bit-packing introduces overhead

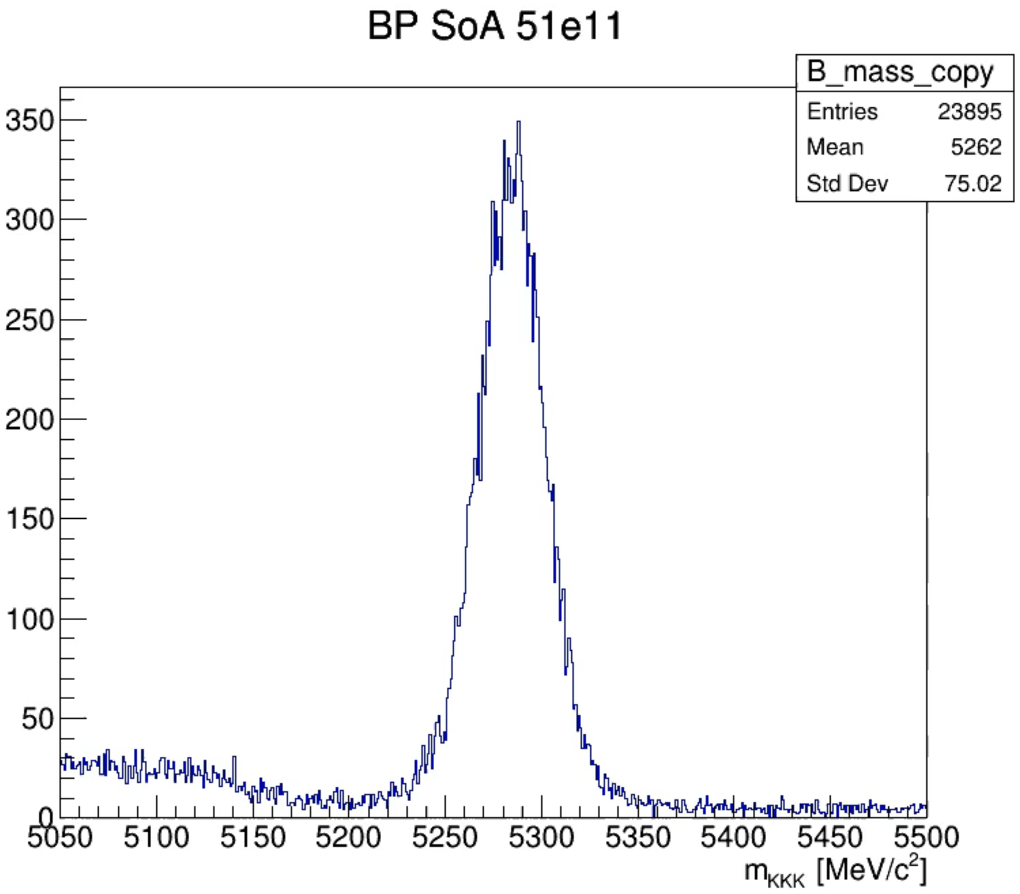
Changing data types usually faster

Uses dedicated hardware instructions

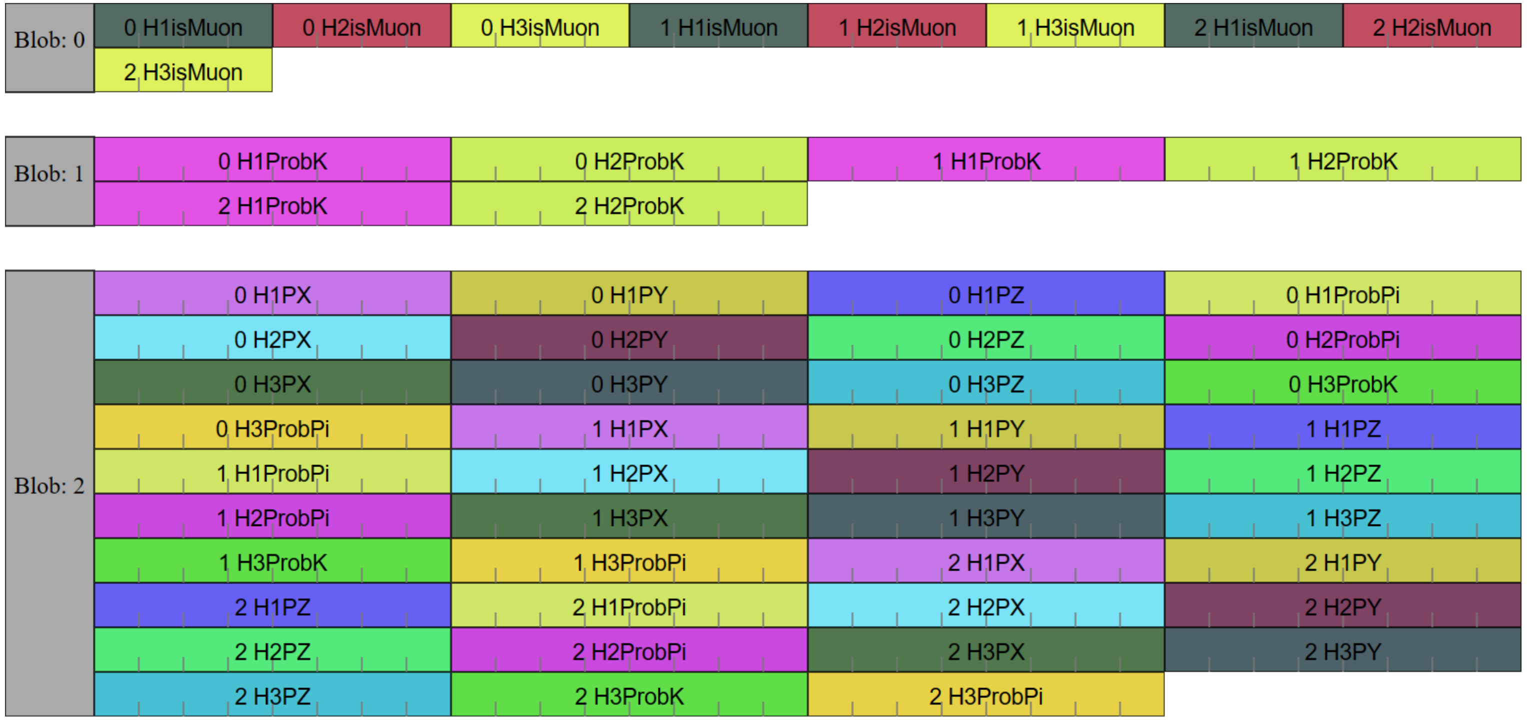
Also: Runtime not the only concern

Data size matters especially for storage

Analysis result with different mantissa bits



The optimal layout



Using pure data layout

Separate H[1-3]isMuon into AoS

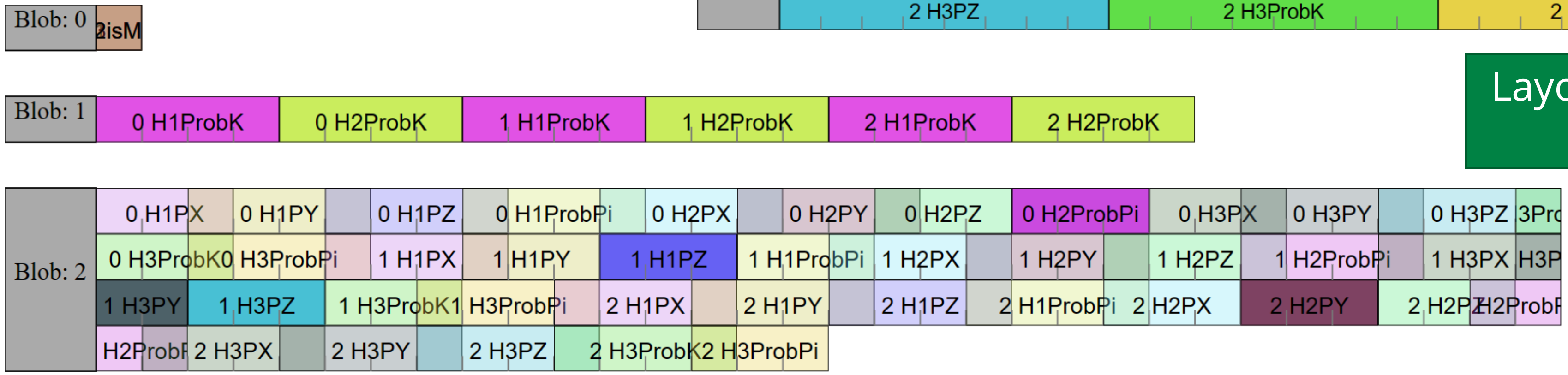
Separate H[1-2]PropK into AoS

With data reduction

Pack H[1-3]isMuon into 1 bit each

Change type of H[1-2]PropK to float

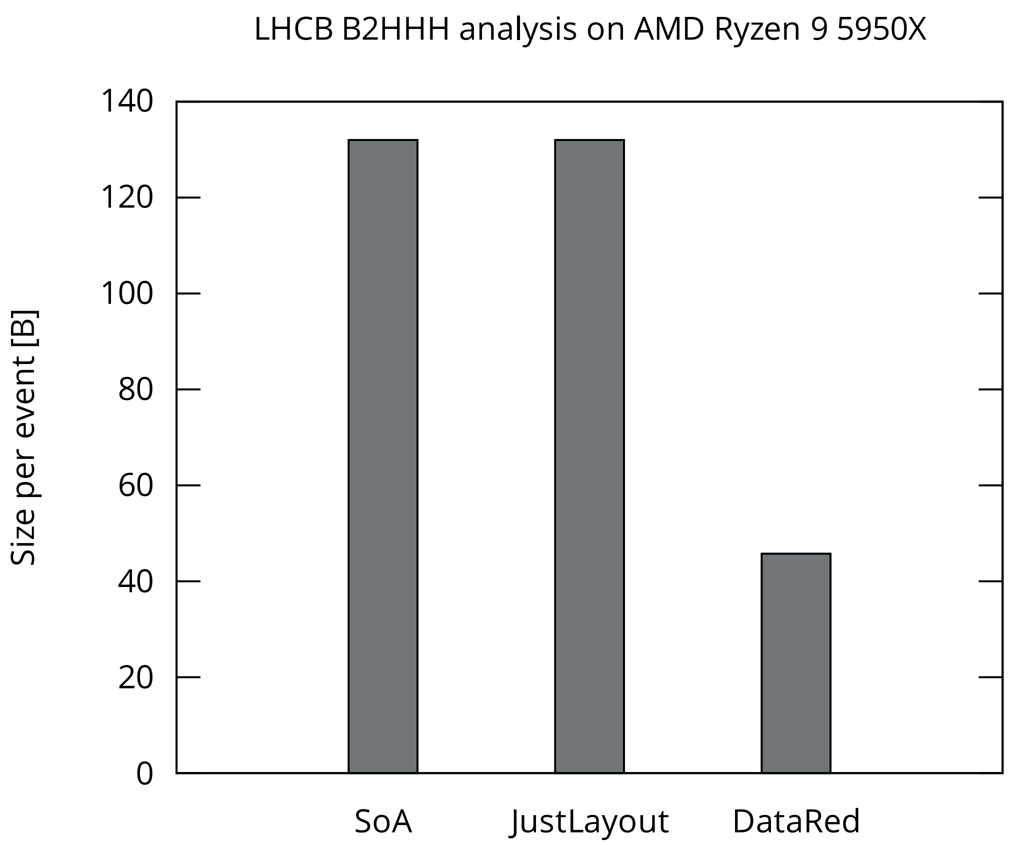
Bitpack the rest



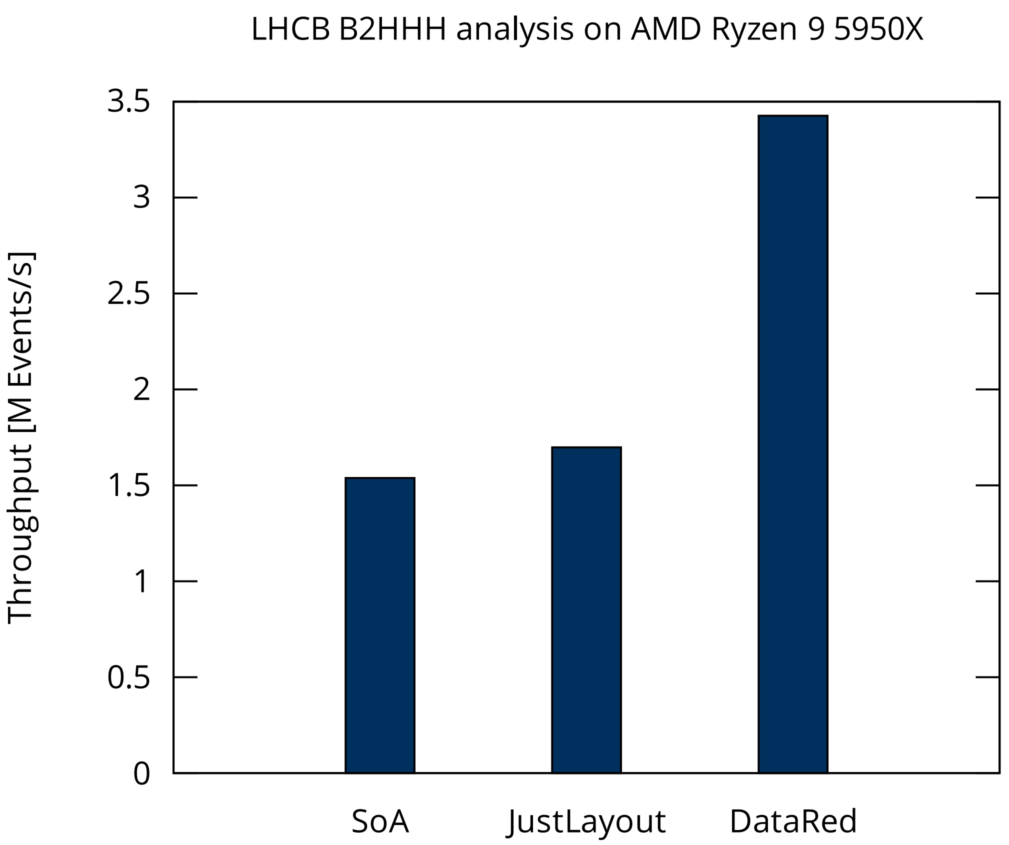
Layout visualization of 3 events
1 LOC with LLAMA

Benchmark

In-memory size



Throughput



HEP analysis case study: Summary

LLAMA can visualize layouts, trace access counts and heatmaps

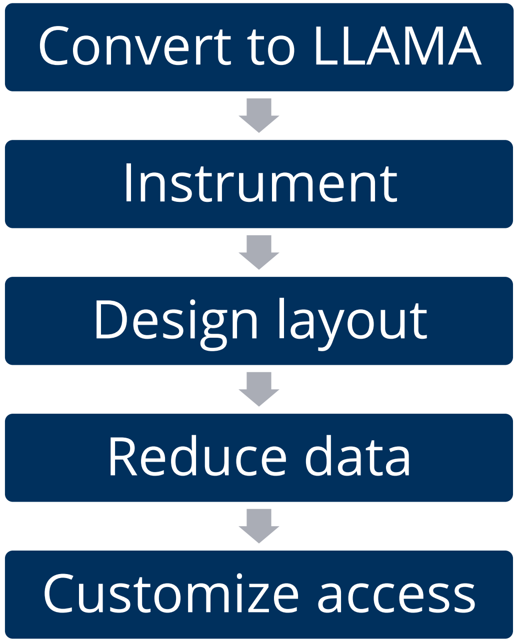
Helps to understand your data, data layouts and access pattern

LLAMA's mappings are versatile

Combine them into custom layouts, including user-defined mappings

Reducing precision/bits can vastly accelerate programs and reduce storage

LLAMA can greatly help with precision studies



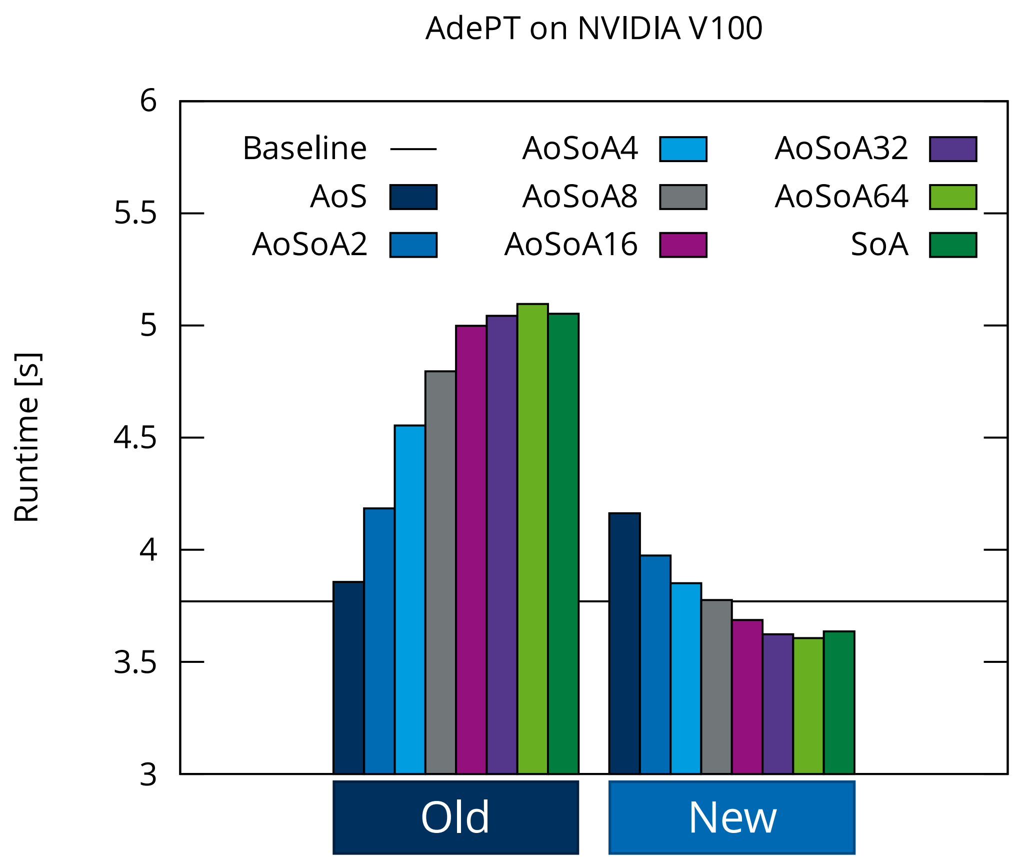
Future work

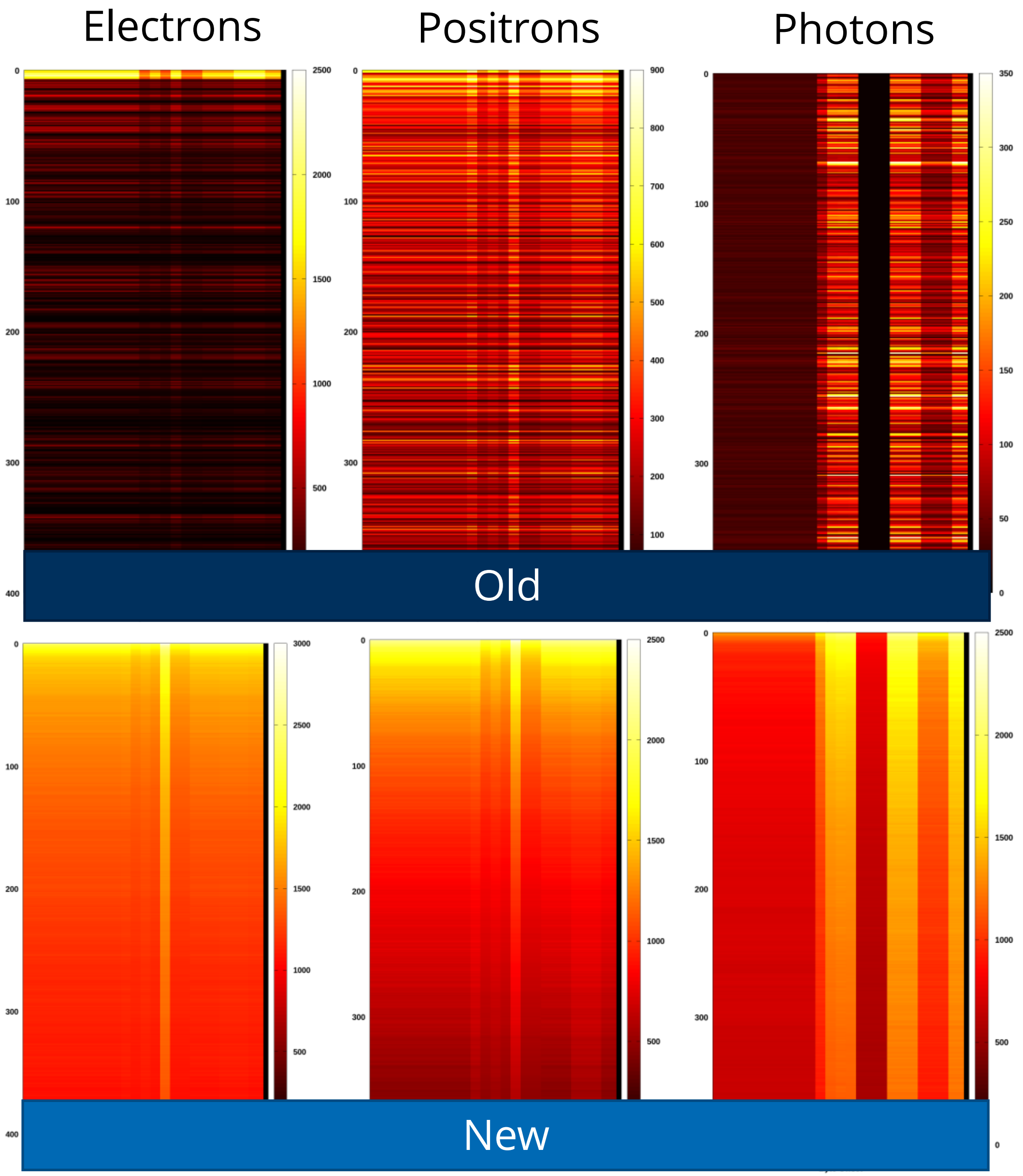
Data only read once: customize access using streaming instructions

Outlook

Within ROOT, for long-running analyses: JIT-compile with a better layout after a while

AdePT case study highlights





Changing layout: edit 1 LOC, recompile, rerun

Integration cost:
From 1336 LOCs, 178 ins./226 del. (git)
Compile time increase: 27%

Summary and conclusions

Programs are increasingly memory bound, regarding throughput, latency and power

Compute and memory hardware becomes increasingly diverse

There is no wholistic, generic and portable solution for memory-related optimizations

LLAMA is a novel abstraction to fill this gap, it …

… addresses: data layout, mem. access, data reduction, SIMD, layout-aware copy, instrum., and layout vis.
- Largely decoupled, algorithmically transparent and fully user-extensible
- Coherently integrated into a concise and well-designed API

… supports fast data layout exploration and rapid porting to new architectures with minimal code changes

… facilitates systematic data layout engineering with instrumentation and performance metrics

… allows for hardware specific tuning using memory accessors and SIMD

Thank you, questions?

Scientific publications

— **B. M. Gruber**, G. Amadio, J. Blomer, A. Matthes, R. Widera, M. Bussmann, "LLAMA: The low-level abstraction for memory access", Software: Practice and Experience 53 (1), 115-141, 2022.

— **B. M. Gruber**, G. Amadio, S. Hageböck, "Challenges and opportunities integrating LLAMA into AdePT", Proceedings to the 21st International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT22), 2023.

— **B. M. Gruber**, "Updates on the Low-Level Abstraction of Memory Access", Proceedings to the 21st International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT22), 2023.

— J. Stephan, ..., **B. M. Gruber** et al., "Performance portability with alpaka," unpublished, 2023.

GitHub: https://github.com/alpaka-group/llama

Financial disclosure

This work has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research (grant no. 13E18CHA)

This work was partially funded by the Center of Advanced Systems Understanding (CASUS) which is financed by Germany's Federal Ministry of Education and Research (BMBF) and by the Saxon Ministry for Science, Culture and Tourism (SMWK) with tax funds on the basis of the budget approved by the Saxon State Parliament.

SPONSORED BY THE

Federal Ministry of Education and Research

References

[1] John L Hennessy and David A Patterson. Computer architecture: a quantitative approach. Sixth. Elsevier, 2017.

[2] Mark Horowitz. “1.1 Computing’s energy problem (and what we can do about it)”. In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC). 2014, pp. 10–14. doi: 10.1109/ISSCC.2014.6757323.

[3] Amirali Boroumand et al. “Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks”. In: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS ’18. Williamsburg, VA, USA: Association for Computing Machinery, 2018, 316–331. isbn: 9781450349116. doi: 10.1145/3173162.3173177. url: https://doi.org/10.1145/3173162.3173177.

[4] Giu.natale. Example of a Roofline model. 2016. url: https://commons.wikimedia.org/wiki/File:Example_of_a_Roofline_model.svg (visited on 2023-04-18).

[5] Norman P. Jouppi et al. “A Domain-Specific Architecture for Deep Neural Networks”. In: Commun. ACM 61.9 (2018), 50–59. issn: 0001-0782. doi: 10.1145/3154484. url: https://doi.org/10.1145/3154484.

[6] R. Aaij et al. “Measurement of CP Violation in the Phase Space of $B^\pm \to K^\pm \pi^+ \pi^-$ and $B^\pm \to K^\pm K^+ K^-$ Decays”. In: Physical Review Letters 111.10 (2013). doi: 10.1103/physrevlett.111.101801. url: https://doi.org/10.1103%2Fphysrevlett.111.101801.

[7] LHCb collaboration. Matter Antimatter Differences (B meson decays to three hadrons) - Project Notebook. 2017. doi: 10.7483/OPENDATA.LHCB.K6BL.RF22. url: http://opendata.cern.ch/record/4902 (visited on 2023-04-19).

[8] Rene Brun and Fons Rademakers. “ROOT—an object oriented data analysis framework”. In: Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 389.1-2 (1997), pp. 81–86.

Backup slides

Example – N-body simulation 1/3

```
using FP = float;
constexpr FP timestep = 0.0001, eps2 = 0.01;
constexpr int steps = 5, problemSize = 64 * 1024;

namespace tag {
  struct Pos{}; struct Vel{}; struct X{}; struct Y{};
  struct Z{}; struct Mass{};
}
using V3 = llama::Record<
  llama::Field<tag::X, FP>,
  llama::Field<tag::Y, FP>,
  llama::Field<tag::Z, FP>>;
using Particle = llama::Record<
  llama::Field<tag::Pos, V3>,
  llama::Field<tag::Vel, V3>,
  llama::Field<tag::Mass, FP>>;
```

Example – N-body simulation 2/3

```
void pPInteraction(auto&& pi, auto&& pj) {
  auto dist = pi(tag::Pos{}) - pj(tag::Pos{});
  dist *= dist;
  const auto distSqr = eps2 +
                       dist(tag::X{}) + dist(tag::Y{}) + dist(tag::Z{});
  const auto distSixth = distSqr * distSqr * distSqr;
  const auto invDistCube = FP{1} / sqrt(distSixth);
  const auto sts = (pj(tag::Mass{}) * timestep) * invDistCube;
  pi(tag::Vel{}) += dist * sts;
}

void update(auto& particles) {
  LLAMA_INDEPENDENT_DATA
  for(std::size_t i = 0; i < problemSize; i++) {
    llama::One<Particle> pi = particles(i);
    for(std::size_t j = 0; j < problemSize; ++j)
      pPInteraction(pi, particles(j));
    particles(i)(tag::Vel{}) = pi(tag::Vel{});
  }
}
```

Example – N-body simulation 3/3

```
void move(auto& particles) {
  LLAMA_INDEPENDENT_DATA
  for(std::size_t i = 0; i < problemSize; i++)
    particles(i)(tag::Pos{}) += particles(i)(tag::Vel{}) * timestep;
}
int main() {
  using ArrayExtents = llama::ArrayExtentsDynamic<std::size_t, 1>;
  using Mapping = llama::mapping::AoS<ArrayExtents, Particle>; // !!!
  auto mapping = Mapping{ArrayExtents{problemSize}};
  auto view = llama::allocViewUninitialized(mapping); // !!!
  for(auto&& p : view) {
    p(tag::Pos{}, tag::X{}) = random();
    // ...
    p(tag::Mass{}) = random();
  }
  for(std::size_t s = 0; s < steps; ++s) {
    update(view);
    move(view);
  }
}
```

Change mapping with this line

Set custom blob alloc. or accessor on this line

Mapping example: AoS implementation

```
template<typename TArrayExtents, typename TRecordDim, bool AlignAndPad = true,
  typename TLinearizeArrayDimsFunctor = LinearizeArrayDimsCpp,
  template<typename> typename FlattenRecordDim = FlattenRecordDimInOrder>
struct AoS : MappingBase<TArrayExtents, TRecordDim> {
  private:
    using Base = MappingBase<TArrayExtents, TRecordDim>;
    using size_type = typename Base::size_type;
  public:
    inline static constexpr bool alignAndPad = AlignAndPad;
    using LinearizeArrayDimsFunctor = TLinearizeArrayDimsFunctor;
    using Flattener = FlattenRecordDim<TRecordDim>;
    inline static constexpr std::size_t blobCount = 1;
    using Base::Base;
    LLAMA_FN_HOST_ACC_INLINE constexpr auto blobSize(size_type) const -> size_type {
      return LinearizeArrayDimsFunctor{}.size(Base::extents())
        * flatSizeOf<typename Flattener::FlatRecordDim, AlignAndPad>;
    }
    template<std::size_t... RecordCoords>
    LLAMA_FN_HOST_ACC_INLINE constexpr auto blobNrAndOffset(
      typename Base::ArrayIndex ai,
      RecordCoord<RecordCoords...> = {}) const -> NrAndOffset<size_type> {
      constexpr std::size_t flatFieldIndex = Flattener::template flatIndex<RecordCoords...>;
      const auto offset = LinearizeArrayDimsFunctor{}(ai, Base::extents())
        * static_cast<size_type>(flatSizeOf<typename Flattener::FlatRecordDim, AlignAndPad>)
        + static_cast<size_type>(flatOffsetOf<typename Flattener::FlatRecordDim, flatFieldIndex,
                                                                                   AlignAndPad>);
      return {size_type{0}, offset};
    }
};
```

LHCB B2HHH analysis layout 4 definition

```
using Mapping = llama::mapping::Split<
    llama::ArrayExtentsDynamic<RE::NTupleSize_t, 1>,
    RecordDim,
    mp_list<mp_list<H1isMuon>, mp_list<H2isMuon>, mp_list<H3isMuon>>,
    llama::mapping::AlignedAoS,
    llama::mapping::BindSplit<
        mp_list<mp_list<H1ProbK>, mp_list<H2ProbK>>,
        llama::mapping::AlignedAoS,
        llama::mapping::AlignedAoS,
        true>::fn,
    true>;
```

LHCB B2HHH analysis layout 8 definition

```
using Mapping = llama::mapping::Split<
    llama::ArrayExtentsDynamic<RE::NTupleSize_t, 1>,
    RecordDim,
    mp_list<mp_list<H1isMuon>, mp_list<H2isMuon>, mp_list<H3isMuon>>,
    llama::mapping::BindBitPackedIntAoS<llama::Constant<1>, llama::mapping::SignBit::Discard>::fn,
    llama::mapping::BindSplit<
        mp_list<mp_list<H1ProbK>, mp_list<H2ProbK>>,
        llama::mapping::BindChangeType<llama::mapping::BindAoS<>::fn, mp_list<mp_list<double, float>>>::fn,
        llama::mapping::BindBitPackedFloatAoS<llama::Constant<6>, llama::Constant<16>>::template fn,
        true>::fn,
    true>;
```

Case study: AdePT

Significant compute workload in High Energy Physics (HEP): particle transport simulation

Traditional codes (Geant4) and CPU based

New: Accelerated demonstrator of electromagnetic Particle Transport (AdePT)

C++/CUDA prototype for offloading electromagnetic (EM) physics to GPUs using VecGeom and G4HepEm

GitHub: https://github.com/apt-sim/AdePT

Profiling showed: bound by memory access

Ideal testbed for LLAMA!

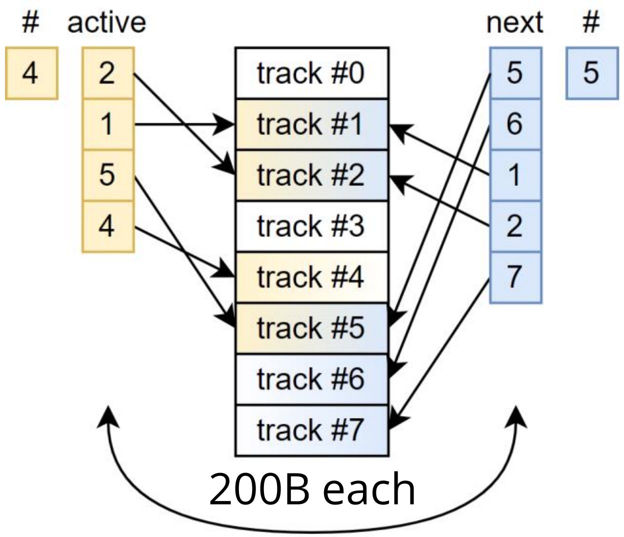
More on AdePT:

Talk and proceedings at ACAT21: "Offloading electromagnetic shower transport to GPUs: the AdePT project"

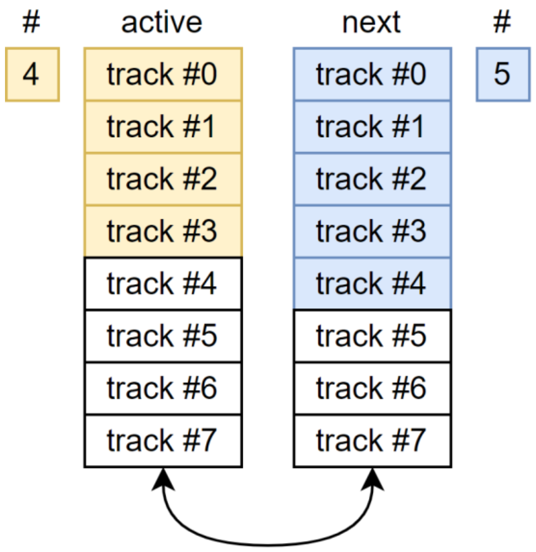
Talk at 27th Geant4 Collaboration meeting: "AdePT status report and discussion"

AdePT track data structure

Single sparse buffer

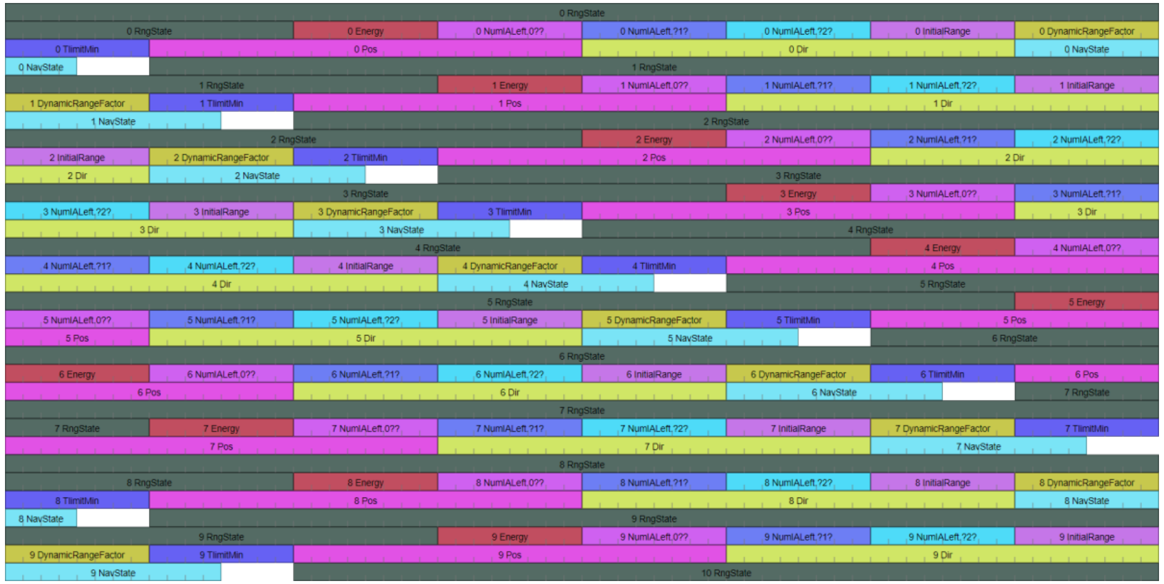


New: Double dense buffers



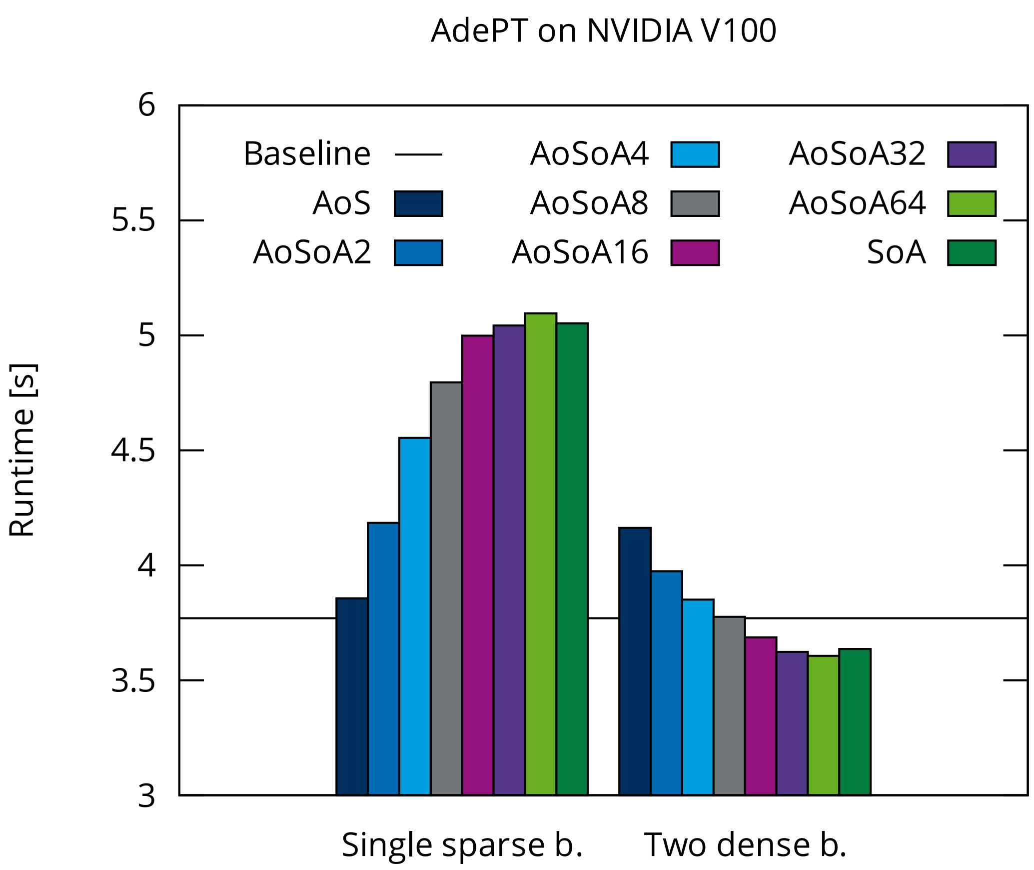
Drawback: copies more memory

wrapped after 64B

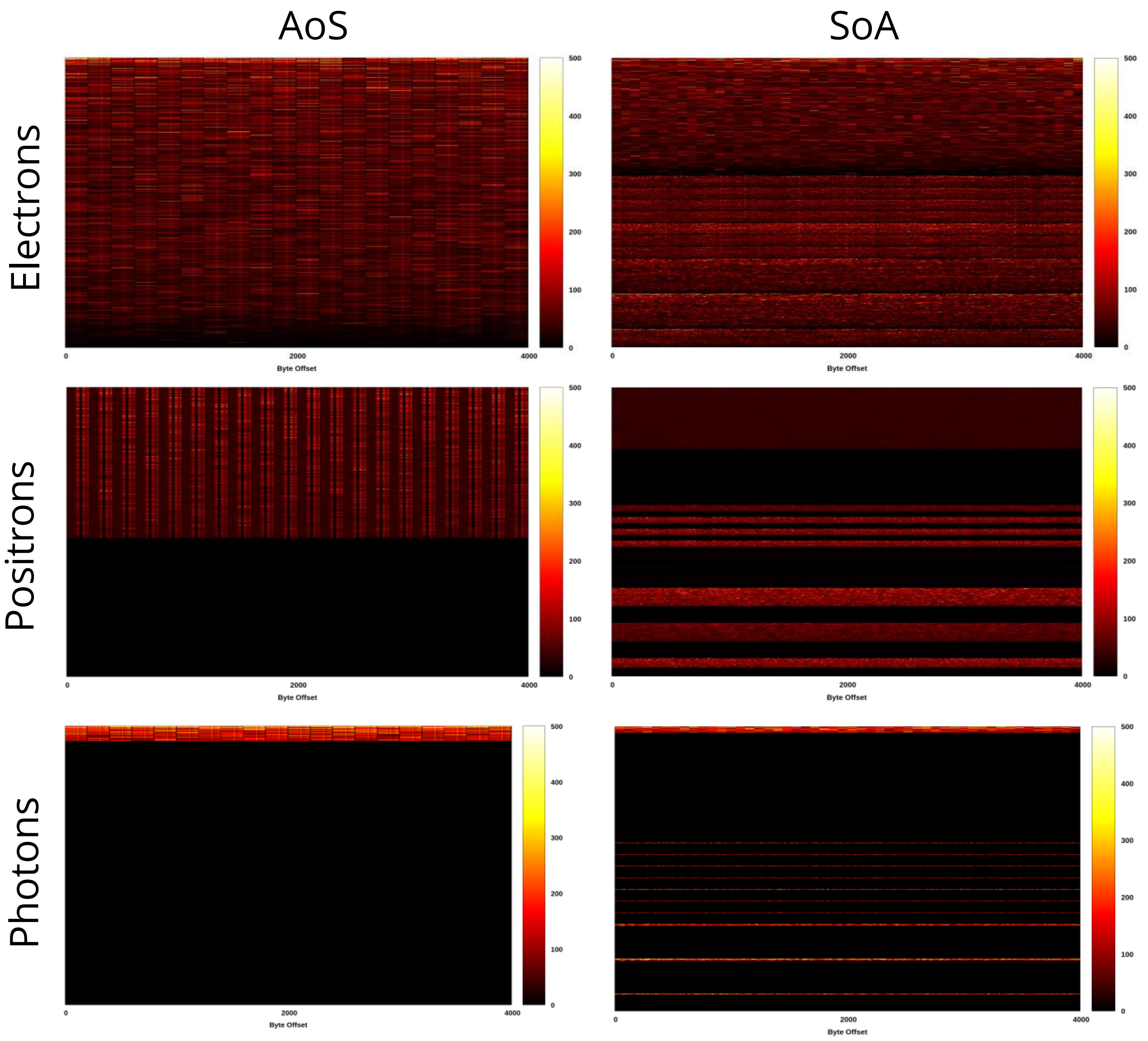


Layout visualization: 1 LOC with LLAMA

Benchmark & Instrumentation



Changing layout is changing 1 LOC, recompile, rerun

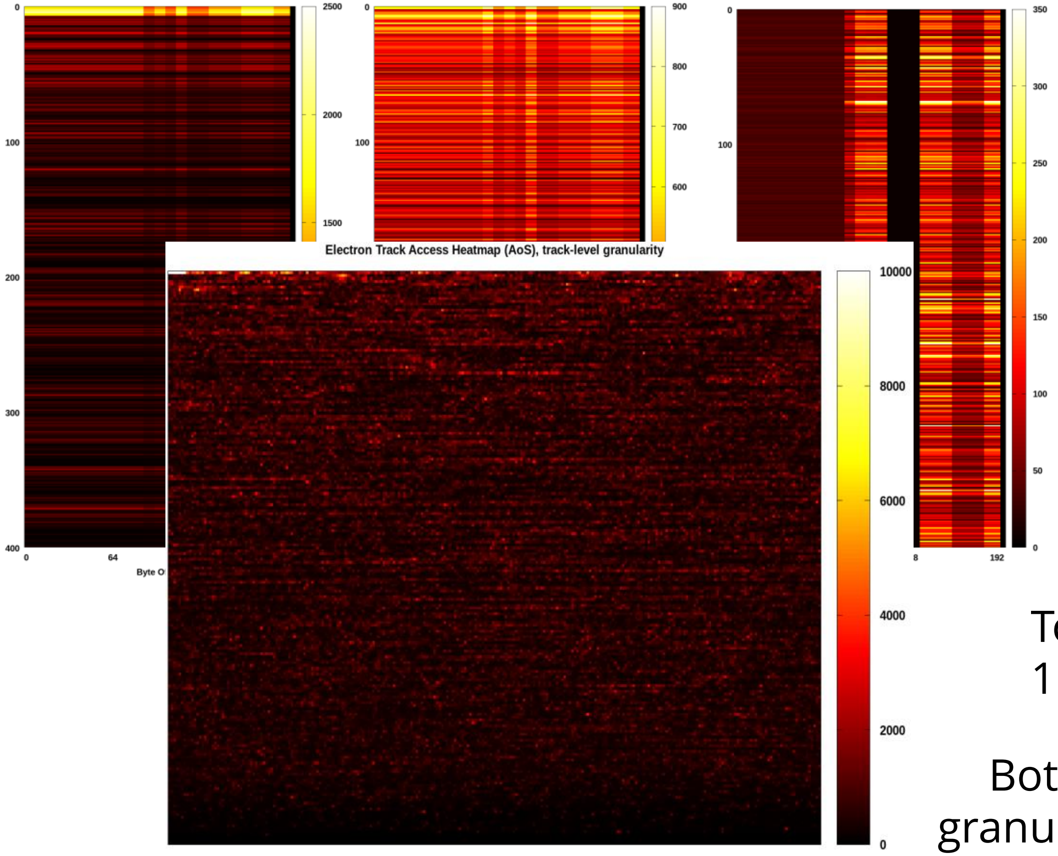


Heatmaps only: Single sparse buffer

Instrumentation

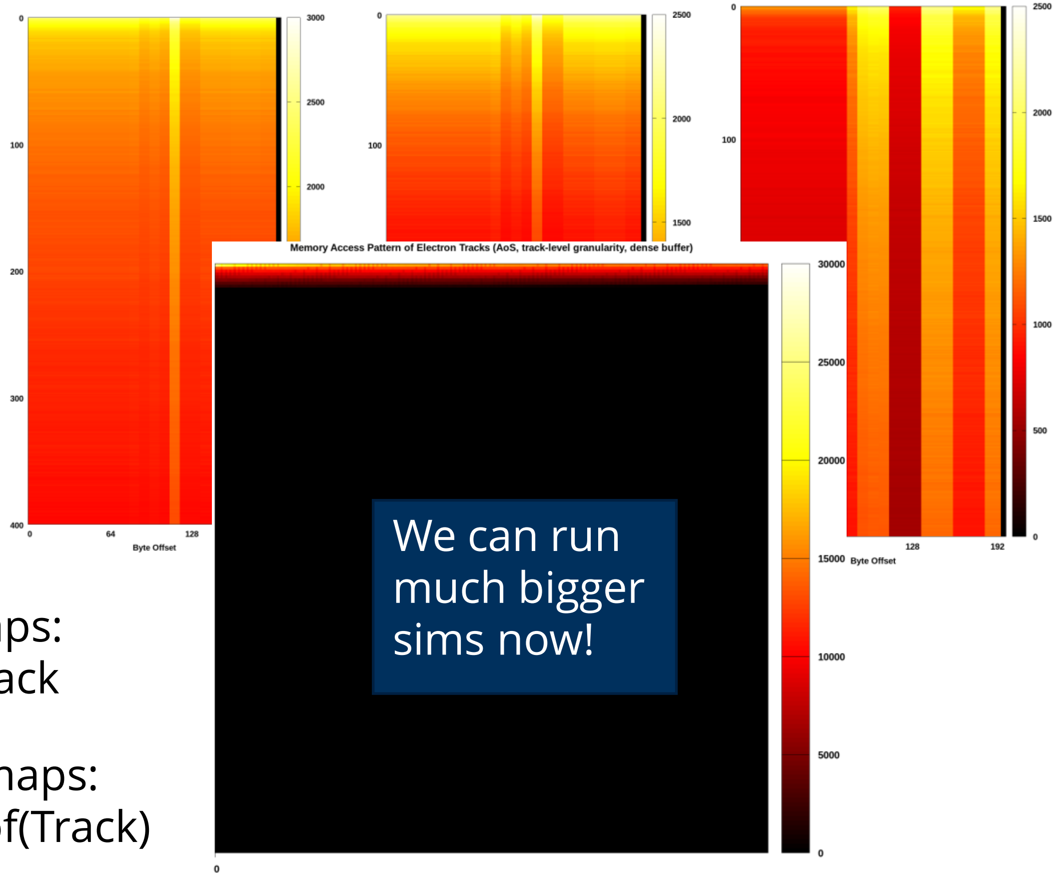
Single sparse buffer

Electrons Positrons Photons



Double dense buffer

Electrons Positrons Photons



We can run much bigger sims now!

Top heatmaps:
1 row = 1 track

Bottom heatmaps:
granularity sizeof(Track)

Adept: Summary and conclusions

Memory layout and access pattern must fit together

SoA is not a silver bullet, requires dense access pattern

AoS works substantially better with sparse and random access

Memory access visualization gives crucial insights for further optimization

Split and regroup data structs based on access pattern

Based on instrumentation results, the AdePT team can now engineer better layouts

LLAMA caused slight abstraction overhead

Different inlining decisions and register allocation

Compile time increased for incremental build by 27% (1 .cpp, 3 .cu files)

Invasive code changes necessary around your data structure

Benchmark has 1336 LoCs (cloc), LLAMA integration: 178 ins. 226 del. (git)

But: porting to the next architecture may now just be a single line switch!

AdePT: Track before/after LLAMA integration

```cpp
struct Track {
  RanluxppDouble rngState;
  double energy;
  double numIALeft[3];
  double initialRange;
  double dynamicRangeFactor;
  double tlimitMin;
  vecgeom::Vector3D<Precision> pos;
  vecgeom::Vector3D<Precision> dir;
  vecgeom::NavStateIndex navState;


  __device__ void InitAsSecondary(
      const Track &parent) {

    // ...
    this->pos      = parent.pos;
    this->navState = parent.navState;
  }
};
```

```cpp
struct RngState {}; struct Energy {}; // ...
using Track = llama::Record<
  llama::Field<RngState, RanluxppDouble>,
  llama::Field<Energy, double>,
  llama::Field<NumIALeft, double[3]>,
  llama::Field<InitialRange, double>,
  llama::Field<DynamicRangeFactor, double>,
  llama::Field<TlimitMin, double>,
  llama::Field<Pos, vecgeom::Vector3D<vecgeom::Precision>>,
  llama::Field<Dir, vecgeom::Vector3D<vecgeom::Precision>>,
  llama::Field<NavState, vecgeom::NavStateIndex>>;

template <typename SecondaryTrack>
__device__ void InitAsSecondary(SecondaryTrack &&track,
    const vecgeom::Vector3D<Precision> &parentPos,
    const vecgeom::NavStateIndex &parentNavState) {
  // ...
  track(Pos{})      = parentPos;
  track(NavState{}) = parentNavState;
}
```