

A Unified Interface for Different Memory Layouts

Jolly Chen

CERN & University of Twente

jolly.chen@cern.ch

Oliver Gregor Rietmann

CERN

oliver.rietmann@cern.ch

Leonardo Beltrame

CERN & Politecnico di Milano

leonardo.beltrame@cern.ch

23rd International Workshop on Advanced Computing and Analysis Techniques in Physics Research

Track 1: Computing Technology for Physics Research

September 11, 2025



UNIVERSITY
OF TWENTE.

The Memory Layout Problem

- ▶ The memory layout of data structures affects the **Memory Latency** of read/write operations
 - The optimal layout depends on the algorithm, dataset, architecture, etc.
- ▶ Two common ways of organizing data in C++:

Array of Structures (AoS)

```
1 struct S { double x1, x2, x3, x4; };  
2 template <int N> using AoS = S[N];
```

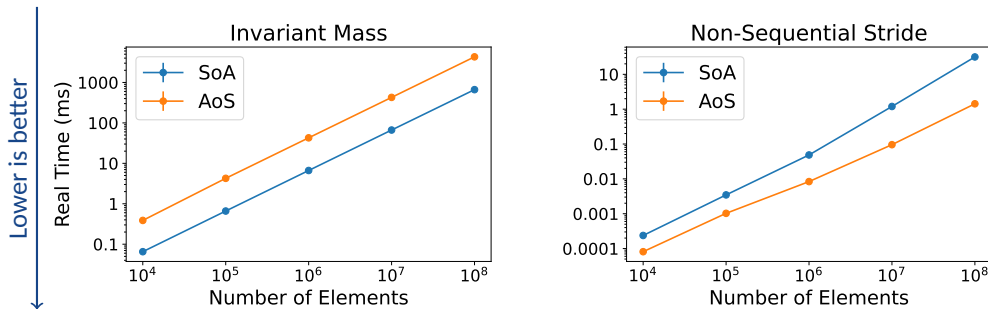


Structure of Arrays (SoA)

```
1 template <int N>  
2 struct SoA { double x1[N], x2[N], x3[N], x4[N]; };
```



AoS versus SoA



- ▶ Need to carefully select the data layout for optimal performance¹ – GPUs often favour SoAs.
- ▶ **Problem:** changing the data layout requires *a lot* of code rewriting
 - Also affects code readability e.g., `particles.momentum[i]` instead of `particles[i].momentum`
- ▶ **Solution:** **abstract** data definition from memory layout, with a **unified interface**

¹Examples compiled with GCC 14 -O3 -ftree-vectorize -ffast-math on AMD EPYC 9654

Approaches ³

In production

- ▶ **CMSSW**: C++ Preprocessor Macros and Boost.PP (Eric Cano, Andrea Bocci, Leonardo Beltrame, Markus Holzer)
 - + Production experience on CPU and GPU
 - Macros: no type safety, difficult to maintain and debug², compilation time

Under development

- ▶ **SoA Wrapper**: Advanced Template Metaprogramming (Oliver Rietmann)
 - + Lightweight implementation (2 header files)
 - Templates: difficult to understand, maintain and debug
- ▶ **SoA Reflection**: C++ Reflection (Jolly Chen)
 - + Potentially maintainability, debuggability, readability, compilation speed
 - Reflection features become available from C++26

Maintainability: Reflection > Template > Macros

Ready for use: Macros > Template > Reflection

²See CppCon talk "Are We Macro-free Yet?": <https://www.youtube.com/watch?v=c6NkeF1eChs>

³See also <https://indico.cern.ch/event/1347968/#1-data-layout-our-reality> for more past approaches

A Unified Interface

Let's say we have this data structure:

```
struct PxPyPzM { double x, y, z, M; };
```

To define the SoA version, for each solution we write...

CMSSW

```
1 GENERATE_SOA_LAYOUT(  
2   PxPyPzM,  
3   SOA_COLUMN(double, x),  
4   SOA_COLUMN(double, y),  
5   SOA_COLUMN(double, z),  
6   SOA_COLUMN(double, M))  
7  
8  
9 using SoA = PxPyPzM<>;  
10 SoA vectors(buf, N);
```

SoA Wrapper

```
1 template <template <class>  
2         class F>  
3 struct PxPyPzM {  
4     F<double> x, y, z, M;  
5 };  
6  
7 using SoA =  
8     wrapper::wrapper<  
9         PxPyPzM, std::vector>;  
10 SoA vectors(buf);
```

SoA Reflection

```
1 struct PxPyPzM {  
2     double &x, &y, &z, &M;  
3 };  
4  
5  
6  
7 using SoA =  
8     rmpp::SoA<PxPyPzM>;  
9 SoA vectors(buf,  
10             byte_size, N);
```

...and to access the data:

CMSSW

```
1 auto vw = SoA::View(vectors);  
2 auto x = vw[i].x()
```

SoA Wrapper

```
1 auto x = vectors[i].x  
2
```

SoA Reflection

```
1 auto x = vectors[i].x  
2
```

C++ Reflection

- ▶ Upcoming new operators in C++26 (P2996):

`^^x` “Lift” operand `x` to a **reflection** value of type `std::meta::info`
`[: refl :]` “Splice” a reflection to **produce grammatical elements**

- ▶ Experimental Token Sequences *after* C++26 (P3294):

`^^{ x }` A token sequence
`queue_injection(...)` Inject sequence at the end of `constexpr` block

- ▶ Two experimental compilers available:

- Fork of Clang: <https://github.com/bloomberg/clang-p2996> (also on Compiler Explorer)
- EDG: <https://godbolt.org/z/qdMhcj8nf>

Example of C++ Reflection Usage – Inspection

```
1 struct S1 { int x; };
2
3 // compile-time variable
4 constexpr std::meta::info member =
5     nonstatic_data_members_of(^^S1, std::meta::access_context::current())[0];
6
7 // Check the reflection of the type.
8 static_assert(type_of(member) == ^^int);
9
10 // Check the identifier of the member variable.
11 static_assert(identifier_of(member) == "x");
```

Try it on Compiler Explorer: <https://godbolt.org/z/3ejYTYoY6> (Clang)

Example of C++ Reflection Usage – Helpers

```
1 struct S1 { int x; double y; };
2
3 int main() {
4     S1 s{0, 1.};
5
6     constexpr auto ctx = std::meta::access_context::current();
7     template for (constexpr auto m :
8         define_static_array(nonstatic_data_members_of(^S1, ctx))) {
9         if constexpr(type_of(m) == ^^int) {
10             printf("%d\n", s.[: m :]);
11         } else if constexpr(type_of(m) == ^^double) {
12             printf("%f\n", s.[: m :]);
13         }
14     }
```



```
1 printf("%d\n", s.x);
2 printf("%f\n", s.y);
```

Output — Try it on Compiler Explorer: <https://godbolt.org/z/K9sEP4Gca> (Clang)

```
0
1.000000
```


Example of C++ Reflection Usage – Injection

```
1 struct S1 { int x; };
2
3 struct S2 {
4     constexpr {
5         constexpr auto x_refl = nonstatic_data_members_of(~~S1)[0];
6         queue_injection(~~{
7             std::vector<typename[: \((type_of(x_refl)) :> \id(identifier_of(x_refl)));
8         });
9     };
```



```
1 struct S1 { int x; };
2
3 struct S2 {
4     std::vector<int> x;
5 };
```

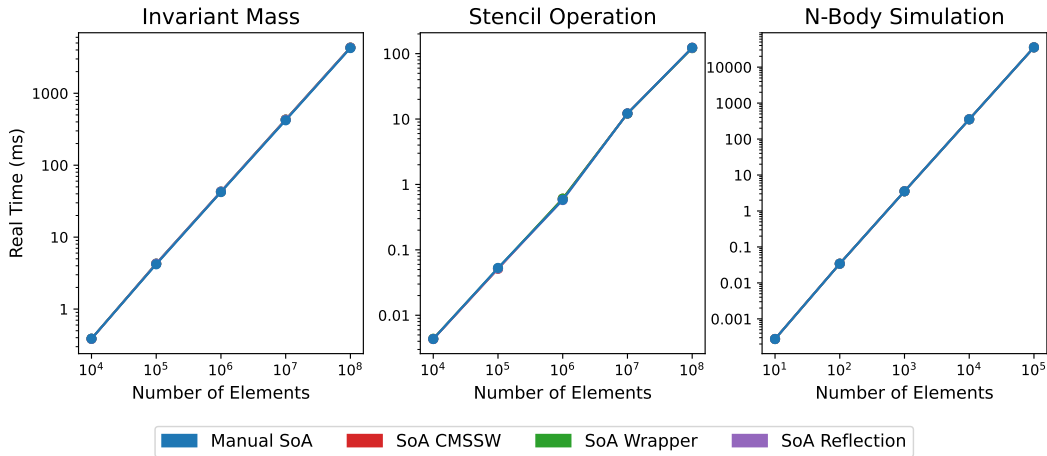
Try it on Compiler Explorer: <https://godbolt.org/z/feWjdnMMf> (EDG)

Performance Comparison

- ▶ We developed an automated benchmarking repository using GitHub CI
 - <https://github.com/cern-nextgen/wp1.7-soa-benchmark>
- ▶ AMD EPYC 9654 96-Core Processor, 1 core
- ▶ GCC 14.2.1 with experimental EDG reflection frontend
 - `-O3 -ftree-vectorize -march=native -funroll-loops -ffast-math -lto`
- ▶ Three benchmarks:
 - Invariant mass computation with PxPyPzM vectors
 - Stencil operation (1D poisson equation solver)⁴
 - N-body simulation
- ▶ Performance of all SoA versions (CMSSW, Wrapper, Reflection) should have the **same performance** as manual SoA

⁴Thanks to Markus Holzer (CMS) for adding the Stencil benchmark to the repo

Performance Comparison



Take Home Messages

- ▶ Data Layouts can significantly impact the performance of your application
- ▶ We are investigating solutions to change data layouts with **minimal code changes** and **zero runtime overhead**
 - CMSSW: <https://github.com/cms-sw/cmssw/blob/master/DataFormats/SoATemplate>
 - SoA Wrapper: <https://github.com/cern-nextgen/wp1.7-soa-wrapper>
 - SoA Reflection: <https://github.com/cern-nextgen/reflmempp>
- ▶ Future Work: GPU support, apply to large real-world code, convergence of solutions
- ▶ Contact: jolly.chen@cern.ch, oliver.riettmann@cern.ch,
leonardo.beltrame@cern.ch

CMSSW ACAT25 Poster



This work has been funded by the Eric & Wendy Schmidt Fund for Strategic Innovation through the CERN Next Generation Triggers project under grant agreement number SIF-2023-004.