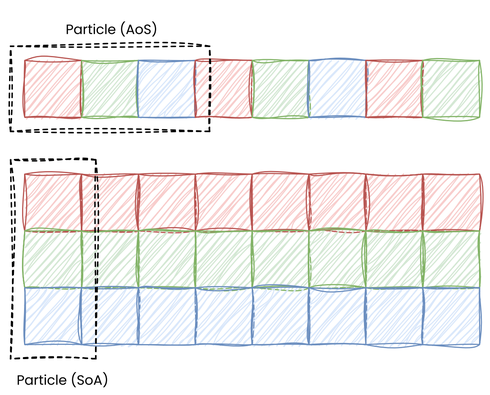
Compiler-supported reduced precision and AoS-SoA transformations for heterogeneous hardware

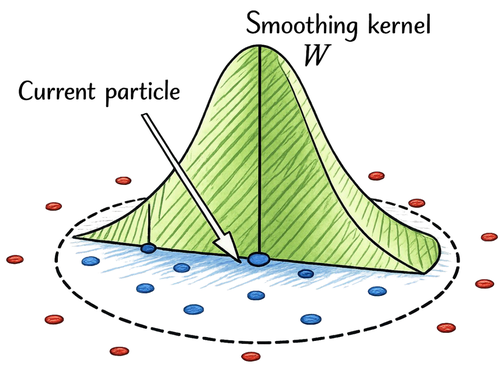
Paweł K. Radtke[1], Tobias Weinzierl[1]

1 INTRODUCTION

Modern heterogeneous systems deliver high arithmetic throughput, yet performance is governed by how data are stored and moved. Particle codes favour array-of-structures (AoS) layouts and compact representations for sorting and communication, whereas accelerators prefer structure-of-arrays (SoA) layouts and conventional formats for efficient memory access.



Smoothed Particle Hydrodynamics (SPH) is a mesh-free Lagrangian method where fluids are represented by particles interacting within a compact support radius, approximating continuous fields through local discrete convolutions.



Runtime is dominated by quadratic neighbourhood kernels (Density, Force), while linear updates (Kick, Drift) contribute comparatively little.

```
1  [[clang::soa_conversion_compute_offload]]
2  [[clang::soa_conversion_handler(device)]]
3  for (int i ...)
4  [[clang::soa_conversion_hoist(1)]]
   for (int j ...) {
     Force(p[i], p[j]);
   }
```

2 OFFLOADING PIPELINE

Baseline pipeline: M → f → M^T
None of the N, U, or C stages are applied. GPU handles the computation. Data is stored in AoS form, and a reduced precision format is used

CPU-based pipeline: N+U+C → M → f → M^T → C^T+U^T+N^T
Here, narrowing (N), unpacking (U), and layout conversion (C) are performed on the CPU before transfer. Host-device communication, as well as offloaded computation processes SoA data

GPU-based pipeline: M → N+U+C → f → C^T+U^T+N^T → M^T
The full AoS buffer is first transferred to the GPU (M). Narrowing (N), unpacking (U), layout transformation (C), and computation (f) are executed on the accelerator

Hybrid pipeline: N → M → U+C → f → C^T+U^T →M^T → N^T
In this mixed variant, narrowing occurs on the CPU to reduce transfer volume, but unpacking and layout conversion are delegated to the GPU. In practice, this increases gather/scatter complexity on the host and was not pursued in the final evaluation.

3 C++ ATTRIBUTES

Developers annotate existing AoS code to request layout conversion, reduced precision, and accelerator offloading.

[[clang::soa_conversion_compute_offload]]
Applied to a loop, this compound attribute requests both AoS–SoA transformation and accelerator execution

[[clang::soa_conversion_handler(host|device)]]
Specifies where the data transformation stages are executed. In the host mode, narrowing, unpacking, and layout conversion are performed on the CPU before data are transferred to the accelerator

[[clang::soa_conversion_hoist(N)]] [1]
Used in nested loop contexts, this attribute hoists transformation stages out of inner loops and associates them with an outer loop level

[[clang::truncate_mantissa(N)]] [2]
Attached to struct members, this attribute reduces the number of mantissa bits used for storage

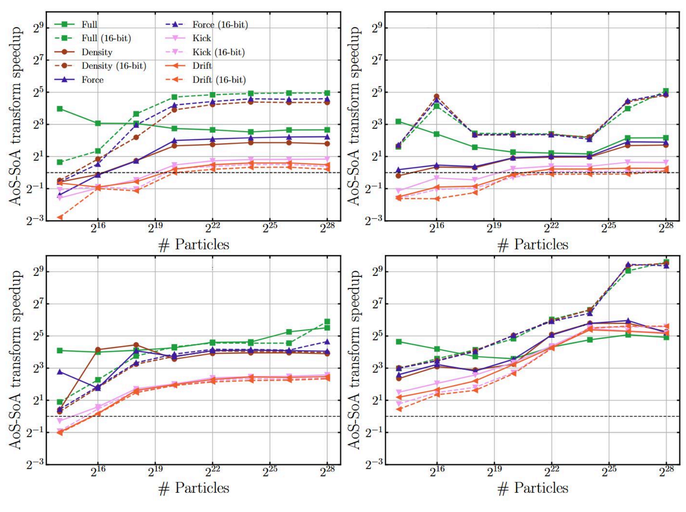
4 COMPILER SUPPORT

The proposed mechanisms are implemented entirely within the Clang/LLVM front end. In the first pass, the compiler analyses annotated loops. In the second pass, it performs an in-memory source-to-source rewrite, injecting prologue and epilogue code for narrowing, precision conversion, AoS–SoA transformation, and host–device synchronisation, **while rewiring the loop body to use the transformed buffers**.

```
TranslationUnitDecl
|-CXXRecordDecl <line:1:1, line:5:1> line:1:8 referenced struct Particle definition
| |-DefinitionData pass_in_registers aggregate standard_layout trivially_copyable pod trivial literal
| | |-DefaultConstructor exists trivial needs_implicit
| | |-CopyConstructor simple trivial has_const_param needs_implicit implicit_has_const_param
| | |-MoveConstructor exists simple trivial needs_implicit
| | |-CopyAssignment simple trivial has_const_param needs_implicit implicit_has_const_param
| | |-MoveAssignment exists simple trivial needs_implicit
| | `-Destructor simple irrelevant trivial needs_implicit
| |-CXXRecordDecl <col:1, col:8> col:8 implicit struct Particle
```

5 PERFORMANCE

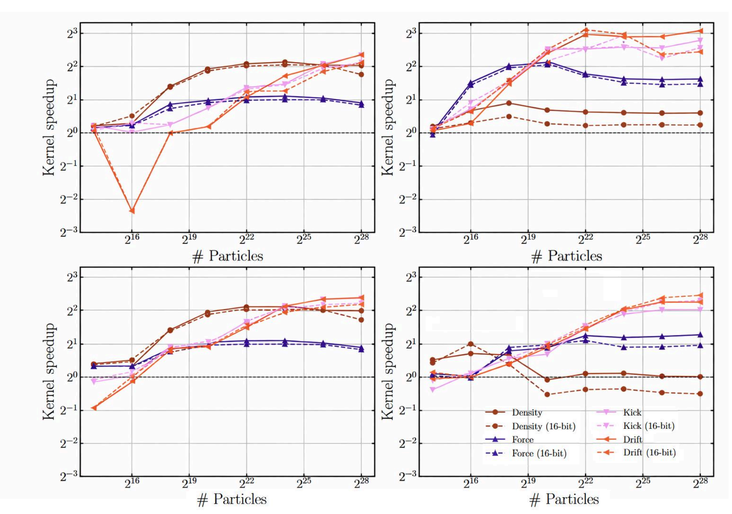
AoS-SoA transformation placement
We compare the cost of performing the AoS–SoA conversion on the CPU versus on the GPU. The effect is strongest on high-bandwidth systems such as GH200 and MI300A, where GPU-based conversion can outperform CPU conversion by orders of magnitude for reduced-precision variants.



AoS–SoA layout transformation speedup on GPU relative to the CPU baseline for the HI00 node (top left), the MI200 node (top right), the GH200 node (bottom left), and the MI300A node (bottom right). The 16-bit variants use the reduced-precision SoA buffers. Values above 1 indicate that the GPU-based conversion is faster, whereas values below 1 mean the CPU-based conversion is more performant.

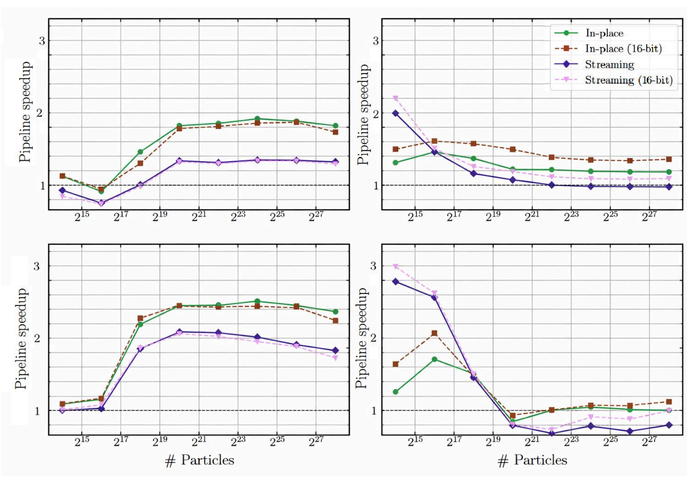
5 PERFORMANCE CONT.

AoS-SoA transformation placement
Here, we isolate kernel execution and measure the impact of consuming SoA rather than AoS data. Memory-bound kernels benefit the most: Kick and Drift achieve the largest speedups. The compute intensive Force kernel shows more moderate gains, and the Density kernel exhibits vendor-dependent behaviour.



SPH kernel compute speedup using SoA vs AoS layout for the HI00 node (top left), the MI200 node (top right), the GH200 node (bottom left), and the MI300A node (bottom right). The 16-bit variants use the reduced-precision SoA buffers. Values above 1 indicate that the SoA layout provides a speedup, whereas values below 1 mean the vanilla AoS layout is more performant.

End-to-end speedups
Lastly, we report full pipeline speedups, combining transformation, transfer, and kernel execution. The improvements are more modest than kernel-level measurements suggest, because Force and Density dominate runtime and the transformation stages themselves incur non-negligible cost. On Nvidia systems, the In-place strategy consistently outperforms Streaming, reaching up to 2.6× on GH200 and 1.9× on H100. AMD platforms exhibit gains below 1.4×.



SPH kernel compute speedup using SoA vs AoS layout for the HI00 node (top left), the MI200 node (top right), the GH200 node (bottom left), and the MI300A node (bottom right). The 16-bit variants use the reduced-precision SoA buffers. Values above 1 indicate that the SoA layout provides a speedup, whereas values below 1 mean the vanilla AoS layout is more performant.

6 CONCLUSIONS

- Layout, precision, and conversion placement form a coupled transformation pipeline
- Performance depends on orchestration, not isolated optimisations
- GPU-side transforms favour high-bandwidth systems; streaming and reduced precision are vendor-dependent
- Reduced precision shows architecture-sensitive speedups
- Language-level control enables systematic rather than ad-hoc optimisation

7 REFERENCES

[1] doi.org/10.1007/978-3-031-85697-6_20

[2] doi.org/10.48550/arXiv.2406.06095

[1]Department of Computer Science, Durham University, UK