Investigating the Performance Benefits of Struct-of-Arrays Storage in Monte Carlo Transport

Braxton Cuneo[a]

Center for Exascale Monte-Carlo Neutron Transport (CEMeNT), [a]Oregon State University, Lawrence Livermore National Laboratory

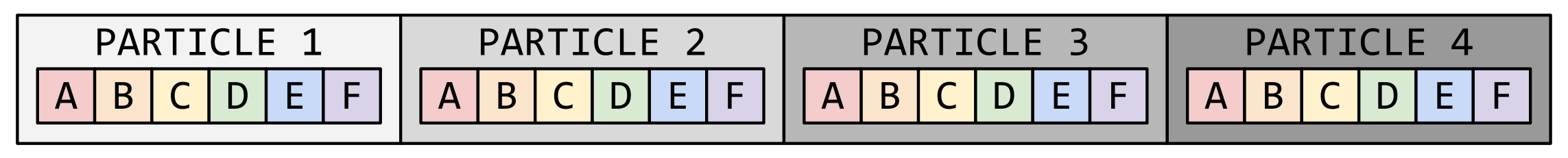
Objective

Refactor the code base of Mercury and Imp to implement a struct-of-arrays data layout for particles.

Motivations

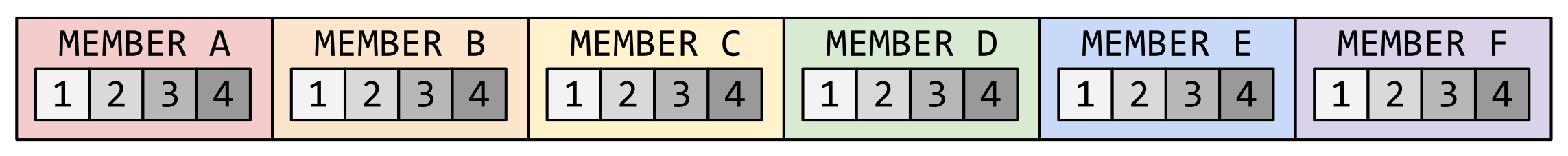
GPUs offer a higher processing throughput relative to CPUs when the processing performed follows certain patterns. Notably, when GPU programs access memory, operations occurring in the same cache line are coalesced, giving the combined result of many accesses at the cost of one. [1]

In the code base shared by Mercury and Imp, particles have been represented as structs, contiguous blocks of memory where each block contains all of the members belonging to a specific particle instance. Furthermore, when dealing with groups of neutrons, these structs have been organized into arrays.



An example of an array-of-structs layout.

While this data layout is useful in contexts where many members of a small group of particles are accessed in sequence, by grouping data by member rather than by particle, GPU programs may access a single member across many particles with low memory usage thanks to coalesced access.



An example of a struct-of-arrays layout.

Refactoring Particle Member Access

The code base Mercury and Imp are built upon is upwards of 300,000 lines, with upwards of 4,000 references to particle members. The scale of the changes required, as well as the potential debugging times that could be incurred by a typo in a manual refactor, necessitated some measure of automation.

To aid in the refactoring of the code base a multi-stage refactoring script was developed to:

1. Cross-reference Imp and Mercury source files
2. Locate declarations of particle variables
3. Find member accesses into matching variables
4. Detect whether accesses involve reading or writing
5. Replace accesses with their corresponding “get” and “set” access functions based off of context

This automation not only reduced the potential for errors in refactoring, but also reduced the labor required by code reviewers; By eliminating trivial refactors, those that required manual intervention could be focused upon.

Struct-of-Arrays Implementation

To make the struct-of-arrays datatype extensible, operations such as array allocations, resizes, and deallocations were implemented as recursive variadic template functions. This not only reduces code duplication, but allows memory management and particle members to be modified quickly and reliably.

```
template <class T> void Buffer_Alloc
(size_t num, T*& pointer) { allocation(T,num); }

template <class T, class... Tail> void Buffer_Alloc
(size_t num, T*& pointer, Tail&...tail)
{ Buffer_Alloc(num,pointer); Buffer_Alloc(num,tail...); }

// As used in constructor:
Buffer_Alloc( num, member_1, member_2, /*... etc ...*/);
```

Backwards compatibility from the implemented SoA type to the pre-existing AoS type was achieved by implementing indexing, nullability and null_ptr comparison, allowing it to be treated much like a pointer. Validity of data is further protected by the deletion of non-moving assignment operations.

Results

Input	Particles	Platform	AoS Runtime (s)	SoA Runtime(s)	Speedup
Godiva in Water (History)	2.0e+5	CPU-genie	236.1	229.2	1.030
Godiva in Water (Event)	2.0e+5	CPU-genie	218.4	233.7	0.934
Godiva in Water (History)	2.0e+5	CPU-ansel	292.3	291.5	1.003
Godiva in Water (Event)	2.0e+5	CPU-ansel	284.9	283.9	0.996
Godiva in Water (History)	2.0e+5	GPU-ansel	55.5	55.2	1.005
Godiva in Water (Event)	2.0e+5	GPU-ansel	38.2	37.3	1.022
Godiva in Water (History)	2.0e+7	GPU-ansel	605.5	606.5	0.998
Godiva in Water (Event)	2.0e+7	GPU-ansel	195.3	175.6	1.112
Crooked Pipe (History)	2.0e+5	GPU-ansel	308.2	298.2	1.033
Crooked Pipe (Event)	2.0e+5	GPU-ansel	2060.8	1660.7	1.241

Conclusions

The implemented struct-of-arrays layout for particles did improve performance in both Mercury and Imp in some cases cases.

Future Work

Potential additions and improvements to the SoA particle representation include:

- A runtime switch between AoS and SoA representations
- The ability to add custom particle members to the SoA representation at runtime

References

[1] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk, and W.-m. Hwu, “Optimization principles and application performance evaluation of a multithreaded gpu using cuda,” in *Proceedings of the 13th ACM SIGPLAN Symposium on principles and practice of parallel programming*, PPoPP ’08, pp. 73–82, ACM, 2008.

Acknowledgements

This work was supported by the Center for Exascale Monte-Carlo Neutron Transport (CEMeNT) a PSAAP-III project funded by the Department of Energy, grant number DE-NA003967.