N-body memory layout exploration

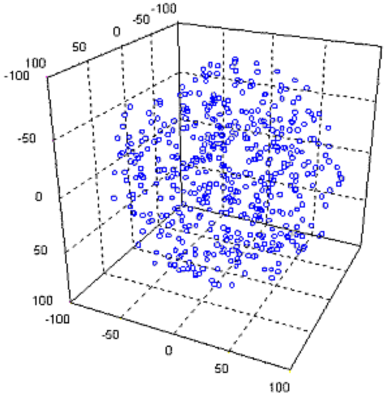
Oliver Geisel & Lisa Hentschke

January 30, 2021

Structure

The n-body simulation

- simulate the interaction of *n* particles
- each particle has
  - position x
  - position y
  - position z
  - velocity x
  - velocity y
  - velocity z
  - mass
- per particle: 7 float values → 28 Byte



http://astro.dur.ac.uk/~nm/pubhtml/nbody/nbody.html

Recap: AoS vs SoA

Array of Structures (AoS)

```
struct Pixel {
  float r;
  float g;
  float b;
  float a;
} pixels[N];
```

Structure of Arrays (SoA)

```
struct Pixel {
  float r[N];
  float g[N];
  float b[N];
  float a[N];
} pixels;
```

from the slides 04-GPU-Memory

Solution Strategies

- rewrite CPP-code in CUDA: implement AoS, SoA and AoSoA memory layouts
- implemented shared memory variants
- for SoA: implemented two sub-variants: B and T
  - B: compute one particle per block
  - T: compute one particle per thread
- We tested on several devices

Example on K80

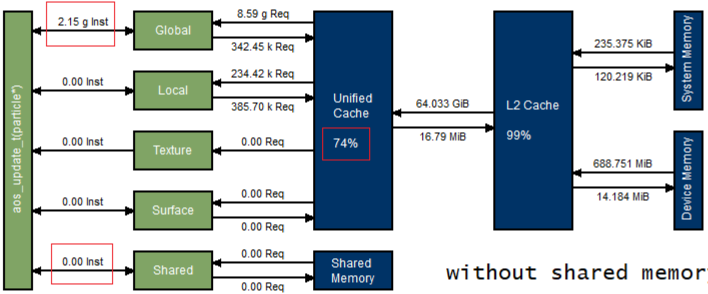
```
16 k particles (448.000 kiB)
Benchmarks:                             Thread,         Thread_shared,  Move
AoS                                     56.5121ms       17.3868ms       0.036544ms
AoS                                     56.4797ms       17.4133ms       0.033280ms
AoS                                     54.9357ms       15.6404ms       0.033632ms
AoS                                     50.7257ms       15.6060ms       0.032544ms
AoS                                     50.7504ms       15.5858ms       0.032768ms
AVG:                                    53.8807ms       16.3264ms       0.033754ms

Benchmarks: Block,      Block_shared,   Thread,         Thread_shared,  Move
SoA       55.0906ms     21.4492ms       22.8277ms       14.2324ms       0.0110ms
SoA       54.3634ms     21.4436ms       22.8197ms       13.0962ms       0.0083ms
SoA       49.3135ms     19.3027ms       20.4029ms       12.7189ms       0.0083ms
SoA       48.4277ms     18.4641ms       18.8272ms       11.7521ms       0.0092ms
SoA       44.3154ms     17.7516ms       18.8140ms       11.7349ms       0.0092ms
AVG:      50.3021ms     19.6823ms       20.7383ms       12.7069ms       0.009184ms

Benchmarks:                             Thread,         Thread_shared,  Move
AoSoA                                   28.2286ms       43.9762ms       0.222112ms
AoSoA                                   28.2296ms       43.6562ms       0.220672ms
AoSoA                                   28.2461ms       43.9120ms       0.221504ms
AoSoA                                   28.2501ms       43.8764ms       0.222688ms
AoSoA                                   28.2302ms       44.0284ms       0.218080ms
AVG:                                    28.2369ms       43.8898ms       0.221011ms
```

Used Profilers

- nvprof on Taurus
- Nvidia Visual Profiler version 11.2

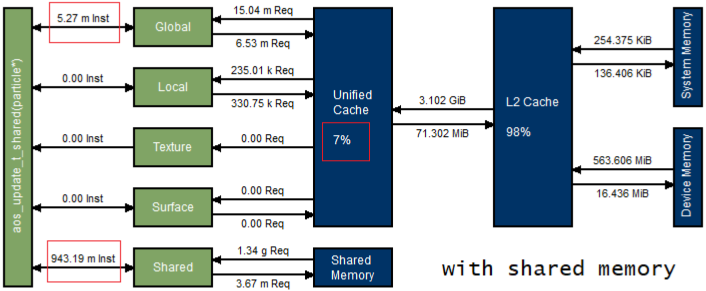


without shared memory

tested on 1070; version 128k paricles on mem layout AoS; subversion T

Used Profilers

- nvprof on Taurus
- Nvidia Visual Profiler version 11.2



with shared memory

tested on 1070; version 128k paricles on mem layout AoS; subversion T

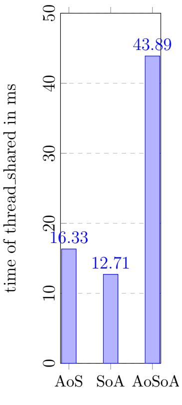
Compare the memory structures

We tested on K80 (Taurus), v100 (Taurus), 1070 (private, driver version 461.09), and RTX 2080 (private, driver version 461.40) respectively

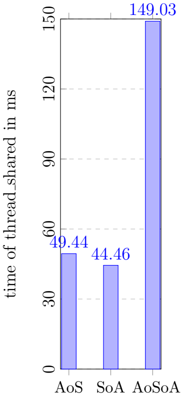
Memory Layout:

- K80 - GDDR 5 with SDRAM
- v100 - HBM 2
- 1070 - GDDR 5
- RTX 2080 - GDDR 6

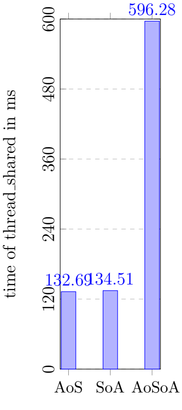
Compare the memory structures - on K80



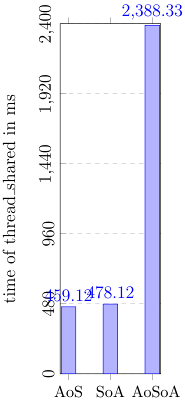
16K



32K

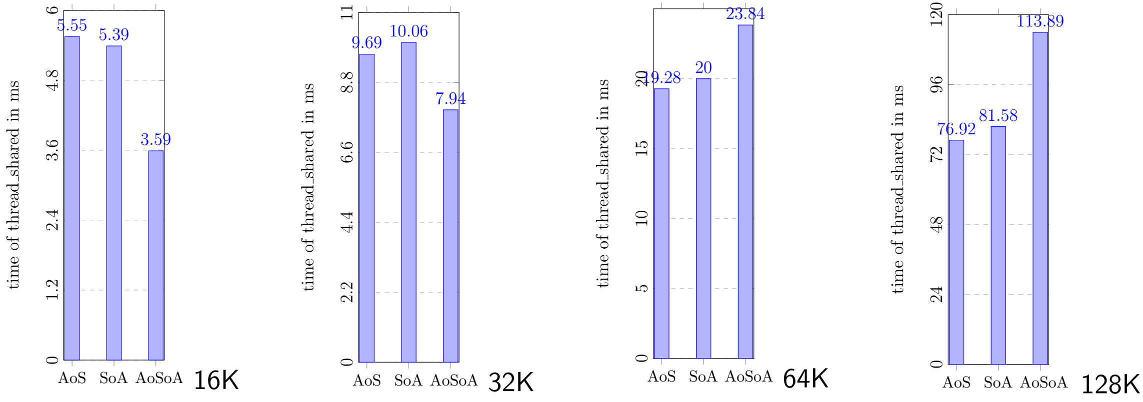


64K

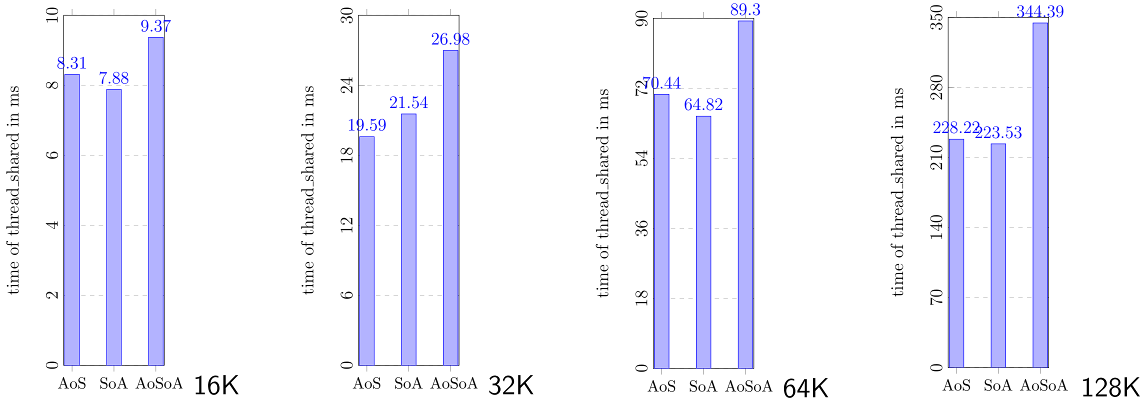


128K

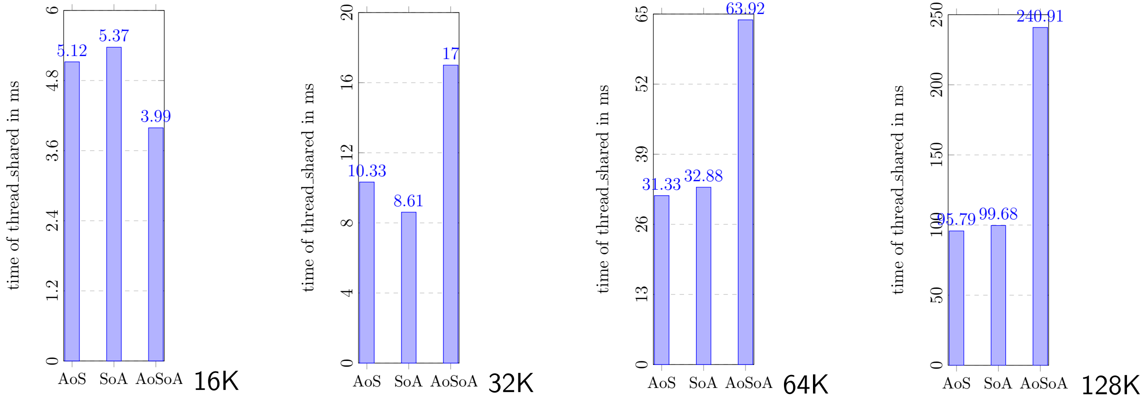
Compare the memory structures - on v100



Compare the memory structures - on 1070



Compare the memory structures - on RTX 2080



Conclusion

- AoSoA can be optimized, but ...
- AoS or SoA heavily depends on architecture $\rightarrow$ just take the easier implementation

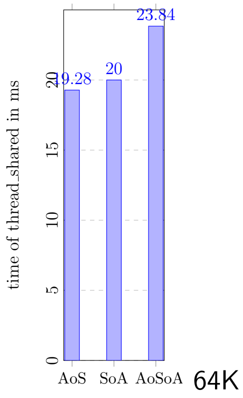
Performance of AoSoA on 16K

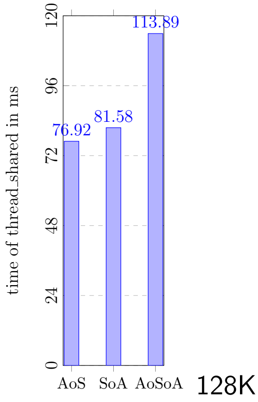
- has nothing to do with the mem layout
- SMs aren't fully occupied
- we configured 32 threads per block

AoS vs SoA

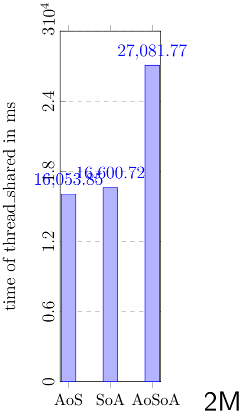
- performance AoS vs SoA are somewhat equally
- contrary to facts from the lecture → WHY?
- K80, v100, and RTX 2080 have configurable L1 chaches (we chose 32KiB)
- also read-only data cache for K80 (48KiB), for 1070 (no RODC) → AoS behaves differently
- that could be the reason why AoS can be better than SoA (bigger chache)
- guess: at bigger n AoS can't catch up? but ...

On v100 with 2 million particles





. . .



The Explanation?

- we don't really know
- cache hierarchy is dynamic and complex, e.g.

K80	48 RODC	dyn. 16-48 L1	(it's 32 in our case)
V100	128 UDC	rest L1	(it's 96 in our case)
RTX 2080	96 UDC	rest L1	(it's 64 in our case)
1070	– RODC	48 L1	

*values are given in KiB

RODC ... read-only data cache
UDC ... unified memory cache

What else could we try?

- use texture memory (indirectly we already do that)
  - $\rightarrow$ mass never changes
- use surface memory? tensor cores?
- change computation (not part of the task)
- optimize AoSoA (not sure if it will be better compared to SoA)

Validation

Two ways:

- **visualize** a 3-body problem with our code, see if it's right
- **compare** with reference computation (use reference values)