CompactGpu: Massively Parallel Memory Defragmentation on GPUs

Matthias Springer (Tokyo Institute of Technology)  https://github.com/prg-titech/dynasoor

Why Defragment GPU Memory?
- Space efficiency: Reduce memory usage
- Improve runtime performance:
  - Accessing compact data requires fewer vector accesses
  - Better memory coalescing

Background: GPU Architecture and Dynamic Memory Allocation
- Pattern: Many small allocations, mostly same size
- For good mem. access performance: Structure of Arrays (SOA) data layout
- Recent NVIDIA GPUs have 128-byte vector registers
- Memory access in aligned, 128-byte transactions

(a) Compact SOA Layout: 3 memory transactions required

(b) Fragmented SOA Layout: 6 memory transactions required

(c) Clustered SOA Layout: 3 memory transactions required

For illustration purposes: Vector length 32 byte (4 scalars) instead of 128 byte (32 scalars). N-body sim.

Design Requirements
- Extension to the DynaSOAr dynamic GPU memory allocator
- Parallel, in-place, stop-the-world defragmentation approach
- To reduce defragmentation overhead: Uniform control flow, little synchronization, efficient memory access

Related Work
- R. Veldema, M. Philippsen. Parallel Memory Defragmentation on GPUs. MSPC '12
  Assumes many different allocation sizes, not in-place, large runtime overhead
- H. Boehm. Space Efficient Conservative Garbage Collection. PLDI '93
  Similar problem: How to find all pointers to moved objects that must be rewritten?

Defragmentation by Block Merging:
parallel defrag<Fish>()

Running example: Fish-and-Sharks simulation

Definition of Defragmentation Candidates:
- Depends on defrag. factor n (problem-specific, compile-time parameter)
  - n = 1: ≤ 50% full
  - n = 2: ≤ 66.6% full
  - Arbitrary n: ≤ n/(n+1) full
  - Guaranteed target frag.: 1/(n+1)

Ex. fish@ agent* (n = 2)
- No free slots for object slots
- Cell* Agent* (n = 2)
- All blocks have same size (bytes)

Example for t1: Choose Src./Target Blocks

- Stream compaction result
- Consecutive threads per block

Step 1: Choose Src./Target Blocks

- b = &object_SLOT[0]; b = b + 1; for (j = b; j < n slots; j++) { ... }
- s = &source; t = &target
- for (i = 0; i < t1; i++) { ... }
- for (i = 0; i < t2; i++) { ... }

Steps 4-5: Rewrite Pointers to Relocated Objects with Bitmap

- How to find all Fish* Agent* values on the heap?
  - Option 1: Scan heap, look for anything that could be a pointer.
  - Option 2: Utilize DynaSOAr’s data layout DSL. Scan only mem. locations of SOA arrays with base type Fish* Agent*.
- Memory transactions: 2 memory reads + 1 write for relocated objects, 1 memory read for all others

Generalization: Other Allocators?
- Many other GPU allocators (Halloc, ScatterAlloc) use hashing (very high frag.) and do not utilize SOA.

Generalization: Other Allocators?
- Many other GPU allocators (Halloc, ScatterAlloc) use hashing (very high frag.) and do not utilize SOA.

- Many other GPU allocators (Halloc, ScatterAlloc) use hashing (very high frag.) and do not utilize SOA.

- Many other GPU allocators (Halloc, ScatterAlloc) use hashing (very high frag.) and do not utilize SOA.