SoaAlloc: Accelerating Single-Method Multiple-Objects Applications on GPUs

Matthias Springer (Tokyo Institute of Technology)

東京工業大学
Tokyo Institute of Technology

https://github.com/prg-titech/soa-alloc

Existing dyn. memory allocators for GPUs [4] are fast in allocation but slow in memory access (do-all). Our allocator (SoaAlloc) is good at both. It outperforms other allocators by 2x or more through SOA data layout for better memory bandwidth utilization, block states for lower fragmentation and hierarchical lock-free bitmaps for faster allocations.

Context

- Frequent pattern in HPC code: Run Same Instruction on Multiple Data (SIMD).
- In Object-oriented Programming Terms: Run Same Method on Multiple Objects (SMMO).
- Examples: Agent-based Simulations (traffic flow, Fish-and-Shark, ...), Barnes-Hut, ...
- Corner Stone of OOP: Dynamic Memory Management (new/delete)

Data Layout Strategies: AOS and SOA

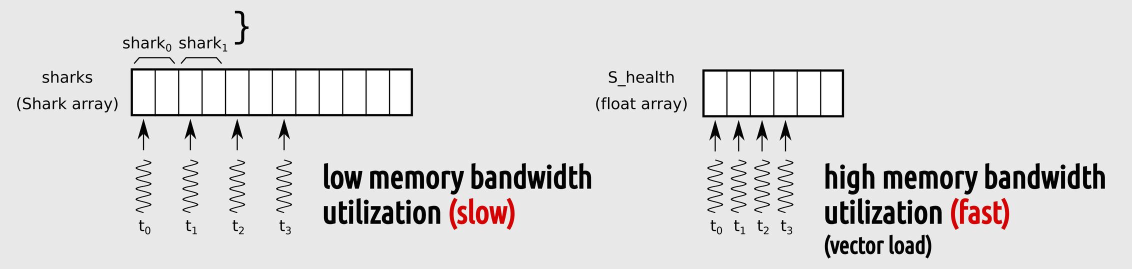
Array of Structures (AOS)

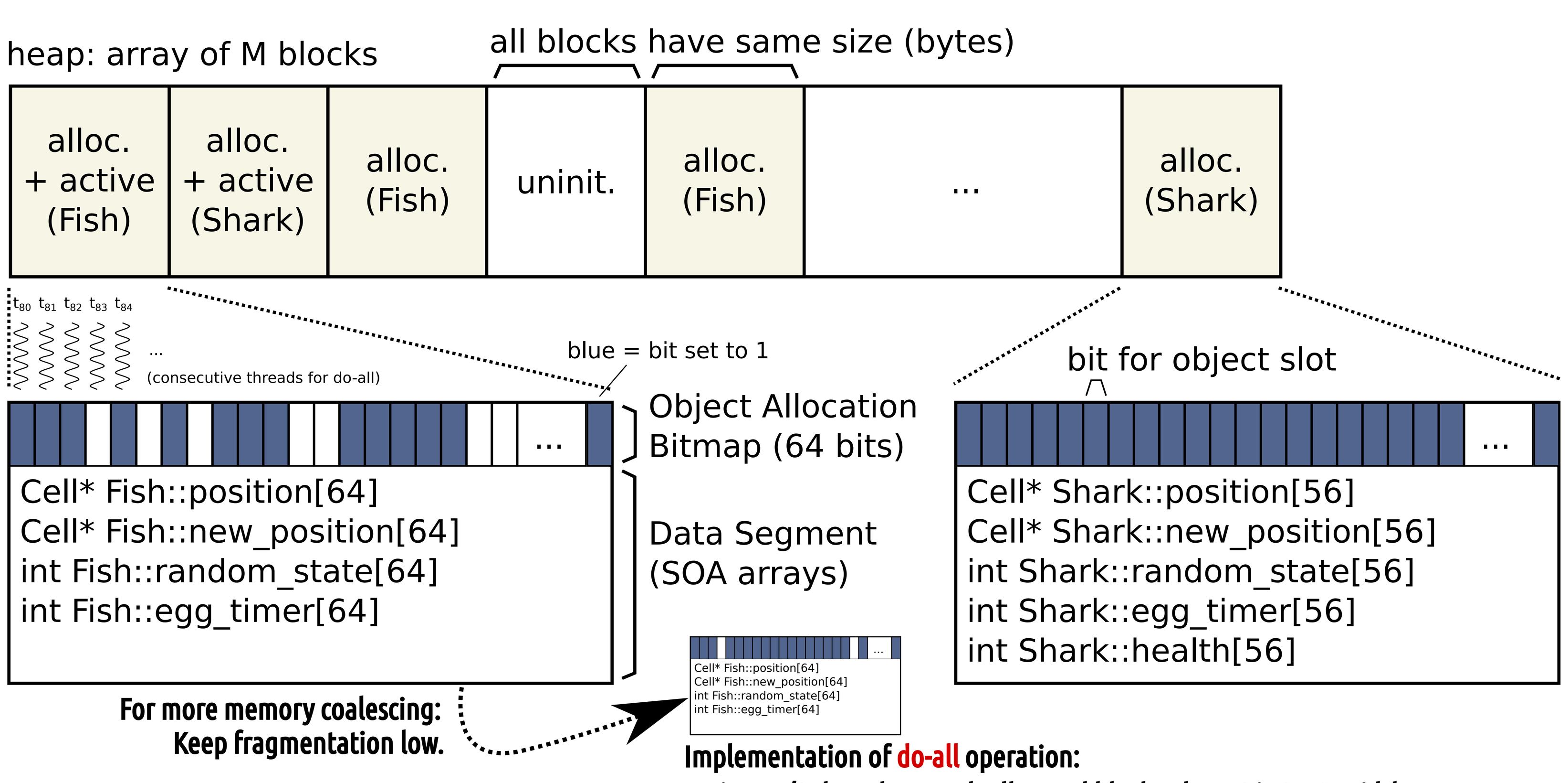
```
struct Shark {
  Cell* pos;  float health;

  void move() {
    health--;
    pos = pos->rand();
  }
};
Shark sharks[100000];
```

Structure of Arrays (SOA)

```
Cell* S_pos[100000];
float S_health[100000];

void S_move(int id) {
  S_health[id]--;
  S_pos[id] = S_pos[id]->rand();
}
```

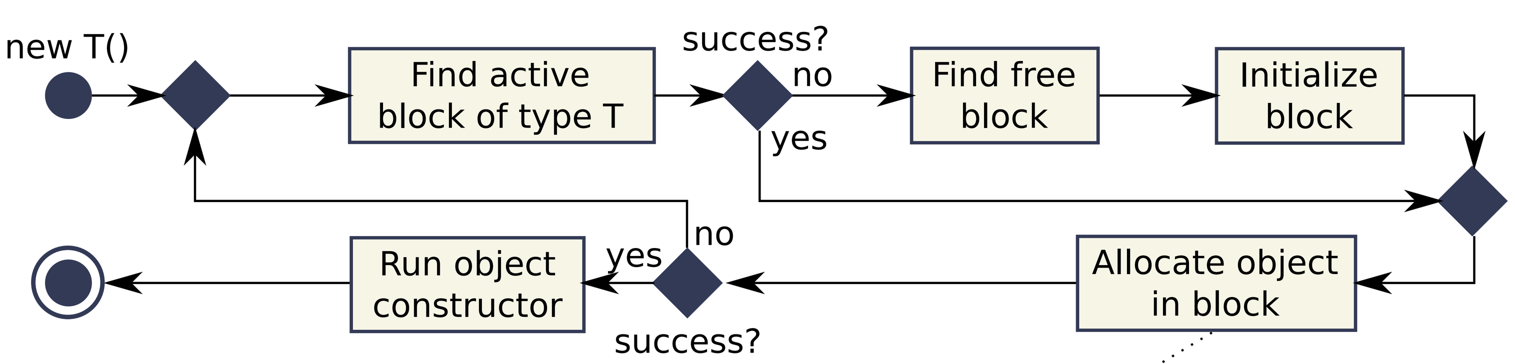
Low fragmentation = compact array





Implementation of do-all operation:
Assign 64/S threads to each allocated block, where S is SIMD width.
In CUDA terminology: Assign 64 consecutive CUDA threads to a block.

Object Allocation (simplified)

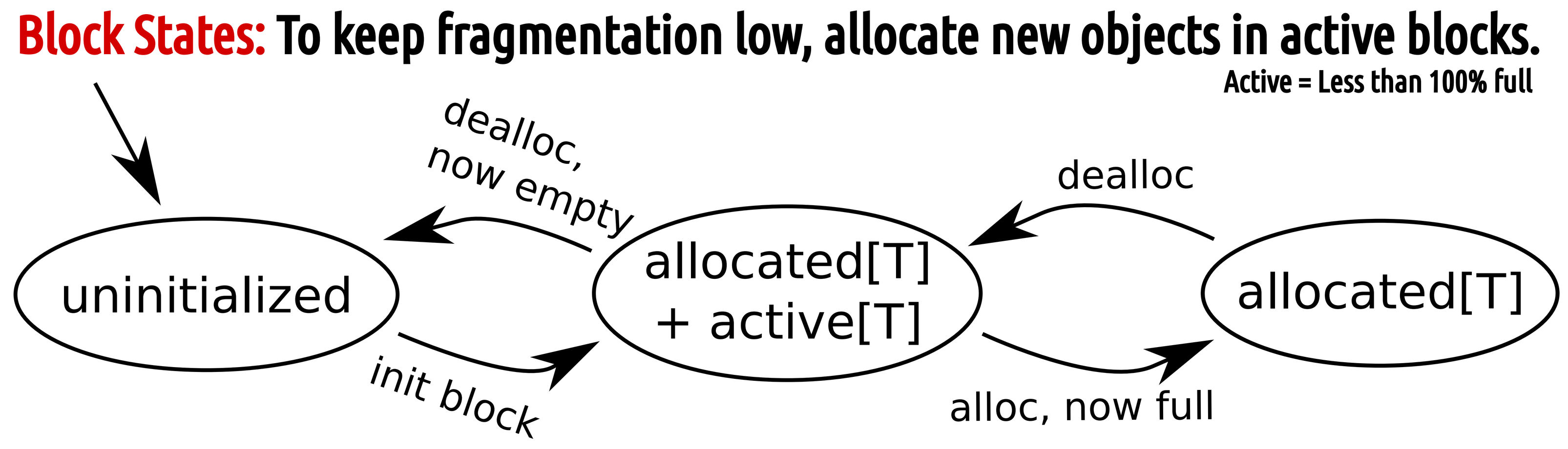


Lock-Free Allocation: Select a bit, try to set it (atomically), check if the operation was successful.

```
while (true) {
  pos = ffs(~obj_alloc_bitmap);  // find-first-set
  if (pos == NONE) return FAIL;
  before = atomicOr(&obj_alloc_bitmap, 1 << pos);
  if ((before & (1 << pos)) == 0) return pos;
}
```

Block States: To keep fragmentation low, allocate new objects in active blocks.

Active = Less than 100% full



Built on Ideas from Previous Work

[1] R. Strzoka: Abstraction for AoS and SoA Layout in C++. GPU Computing Gems Jade Edition, 2012. C++/CUDA embedded DSL for switching between AoS and SoA layout.

[2] M. Springer, H. Masuhara: Ikra-Cpp: A C++/CUDA DSL for Object-Oriented Progr. with Structure-of-Arrays Layout. WPMVP 2018. SoA with OOP abstractions in C++/CUDA.

[3] J. Franco, M. Hagelin, T. Wrigstad, S. Drossopoulou, S. Eisenbach: You Can Have It All: Abstraction and Good Cache Performance. Onward! 2017. Pointers → Global References.

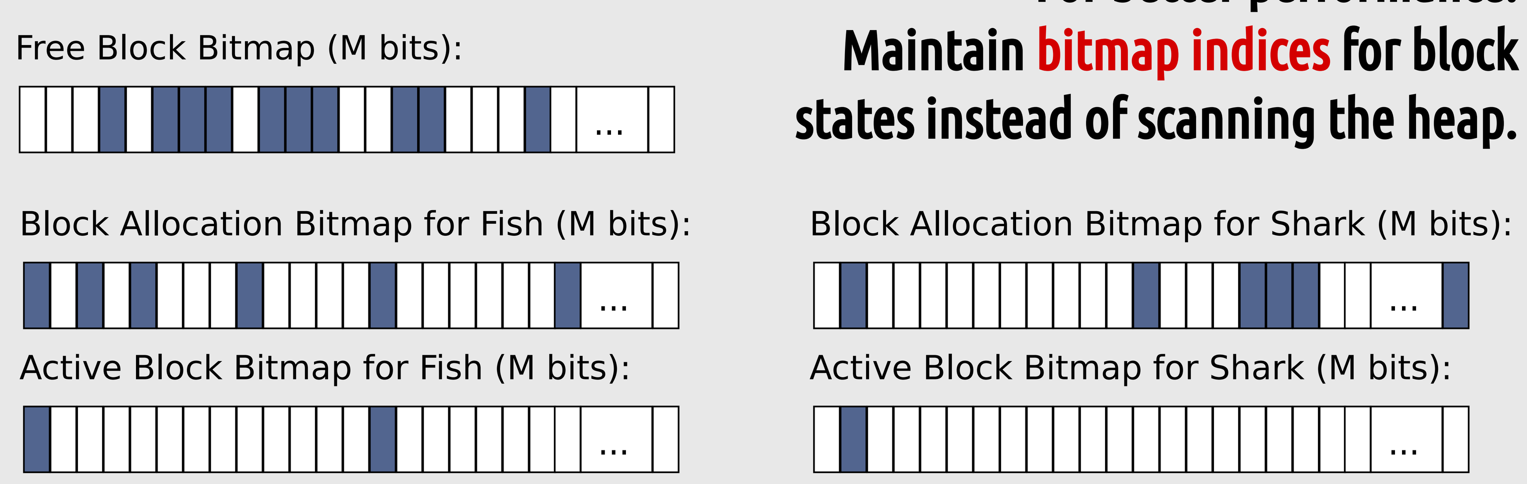
[4] M. Steinberger, M. Kenzel, B. Kainz, D. Schmalstieg: ScatterAlloc: Massively Parallel Dynamic Memory Allocation for the GPU, InPar 2012. Fast (de)allocation, but not optimized for access of structured data. (Neither is any other GPU memory allocator.)

This Work: Goals and Challenges

- **Goal: Object-oriented programming with good cache/memory performance on SIMD architectures (via SOA data layout)**
- **Focus: Dynamic memory management (C++ `new` / `delete`)**
- **API:** `Allocator::new<T>(...)`, `Allocator::delete<T>(ptr)`, `Allocator::do_all<T>(void (T::*method)())`
- **Challenge: When to allocate SOA ararys? How large?**
- **Challenge: Memory fragmentation reduces the benefit of SOA.**

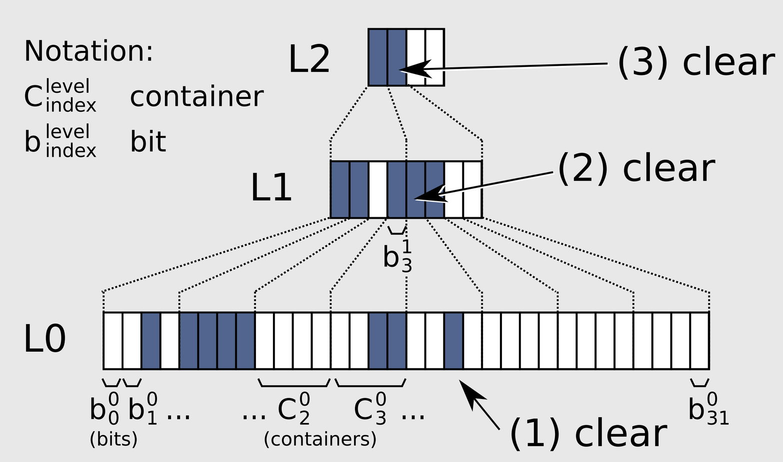
e.g.: `do_all<Shark>(&Shark::prepare)`

For better performance: Maintain bitmap indices for block states instead of scanning the heap.

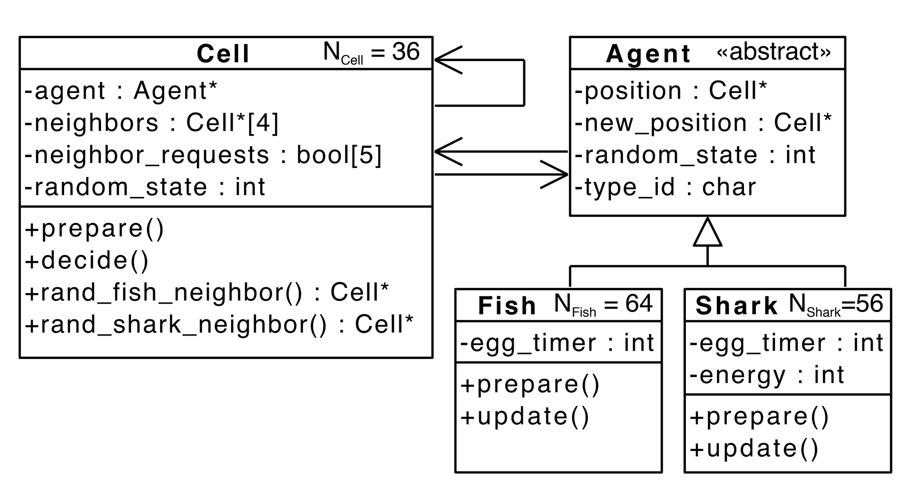


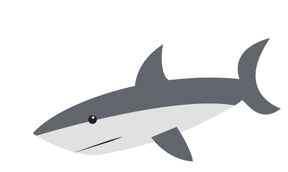
For even better performance: Hierarchical bitmaps.

Avoid scanning empty bitmap parts.




```
void Bitmap::try_find_set() {
  if (has_higher_level_bitmap) {
    cid = next_level.try_find_set();
    if (cid == FAIL) return FAIL;
  } else { cid = 0; }
  offset = ffs(containers[cid]);
  if (offset == NONE) return FAIL;
  return 64*cid + offset;
}
```





Benchmark: Fish and Shark