STRATEGIC DYNAMIC MEMORY MANAGEMENT USING HEAP LAYERS

Panagiotis Katsalifis
Alexandra Chelidoni

ALLOCATOR DESIGN: FROM SPECIALIZED TO GENERAL

- Leveraging Patterns: Using application-specific allocation patterns allows for specialized memory managers that significantly boost performance.
- Library Strategy: Effective specialized allocators should be generalized and organized into a reusable library for broader use.
- The "Full Circle" Design: A robust general-purpose allocator is best defined as a smart combination of these highly optimized, special-purpose components.

HEAP LAYERS LIBRARY

- Modular Architecture: It provides a flexible C++ framework that allows you to build high-performance allocators by stacking different functional "layers".

- Simplified Development: It streamlines the process of writing complex memory managers, making it easier to create both specialized (custom) and standard (general-purpose) allocators.

USING HEAP LAYERS

- It uses C++ templates to build and parameterize several building blocks.

- Combines building blocks to construct different allocation policies.

EXAMPLE: KINGSLEY ALLOCATOR

```
template <class PerClassHeap, class BigHeap>
  class KingsleyHeap :
   public StrictSegHeap<Kingsley::NUMBINS,
                        Kingsley::size2Class,
                        Kingsley::class2Size,
                        PerClassHeap,
                        BigHeap> {};

}
```

```
class TopHeap : public SizeHeap<UniqueHeap<ZoneHeap<MmapHeap, 65536> > > {};

class TheCustomHeapType :
  public ANSIWrapper<KingsleyHeap<AdaptHeap<DLList, TopHeap>, TopHeap> > {};
```

LAYER CATEGORIES

memory flow

- Top Heap
- Building Block Layer
- Composition Layer

Adapter Layer-Miscellaneous

TOP HEAP

"Source" heap that has no superheap. It simply fetches large chunks of raw memory.

- Mmap (classic syscall as a heap class)
- StaticBufferHeap

```
template <int BufferSize>
class StaticBufferHeap {
```

BUILDING BLOCK LAYER

It defines how memory is stored and tracked. It takes memory from a top heap and organizes it.

- ZoneHeap (Arena allocator)

```
template <class SuperHeap, size_t ChunkSize>
class ZoneHeap : public SuperHeap {
```

- FreelistHeap

```
template <class SuperHeap>
class FreelistHeap : public SuperHeap {
```

COMPOSITION LAYER

It doesn't store memory itself; it decides which child heap should handle a request. Its role is to combine multiple heaps into a smarter system.

- Hybrid Heap

```
template <int BigSize, class SmallHeap, class BigHeap>
class HybridHeap : public SmallHeap {
```

- Segregated Heap

```
template <int NumBins,
          int (*getSizeClass) (const size_t),
          size_t (*getClassMaxSize) (const int),
          class LittleHeap,
          class BigHeap>
class SegHeap : public LittleHeap {
```

ADAPTER LAYER-MISCELLANEOUS

They can actually fit anywhere between the basic layer categories. Each one serves a different purpose.

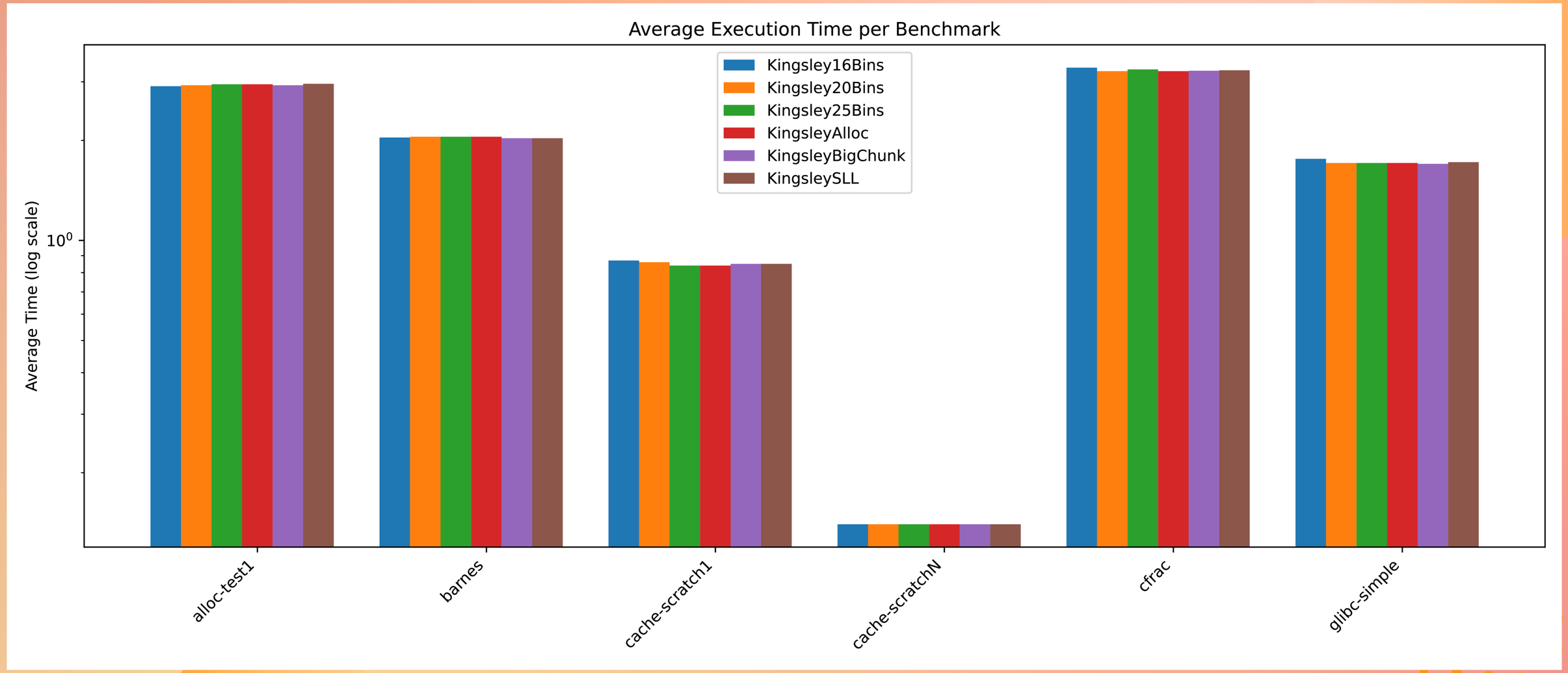
- SizeHeap (metadata to keep track of the heap size)
- LockedHeap & ThreadSpecificHeap (adds thread-safety)
- ANSI Wrapper (generates the standard C interface for the custom heap)

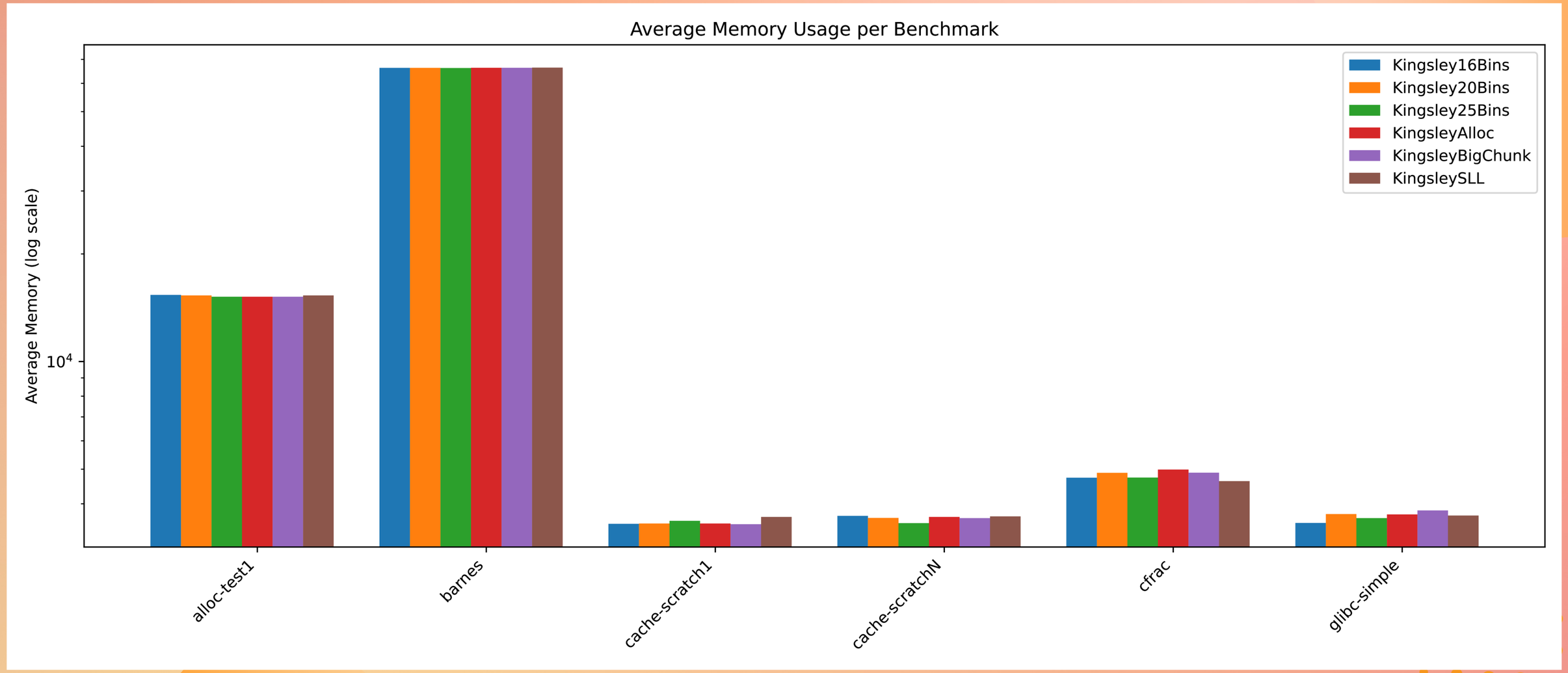
EXPERIMENTATION

- Design Space Exploration based on the Kingsley Allocator
- Creation of some simplified custom allocators
- Run benchmarks and compare with sysmalloc and mimalloc
- Evaluate results based on memory footprint and execution time

DESIGN SPACE EXPLORATION

- We tried tweaking some parameters of the Kingsley allocator
  - ↳ Negligible changes between different versions
- Made the Kingsley allocator thread safe
  - ↳ we used a global lock, which slowed down execution time by a lot

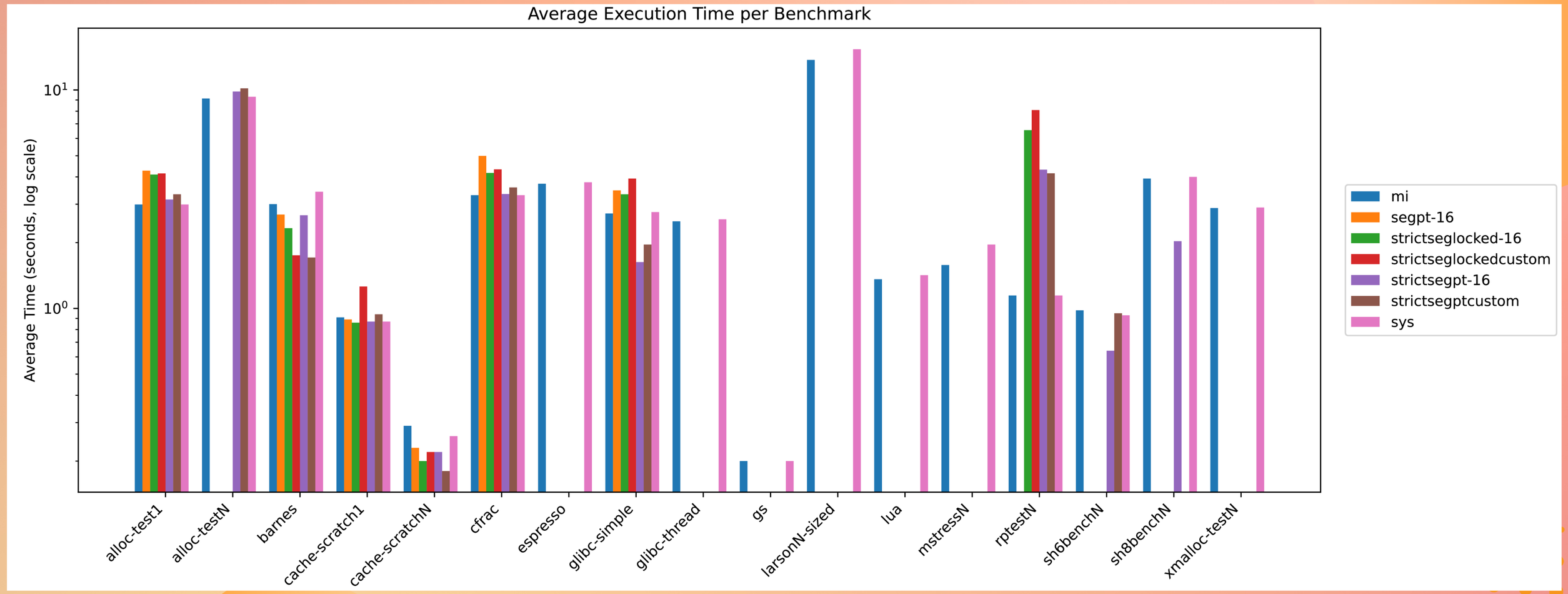

Average Execution Time per Benchmark
Kingsley16Bins
Kingsley20Bins
Kingsley25Bins
KingsleyAlloc
KingsleyBigChunk
KingsleySLL
Average Time (log scale)
10^0
alloc-test1
barnes
cache-scratch1
cache-scratchN
cfrac
glibc-simple

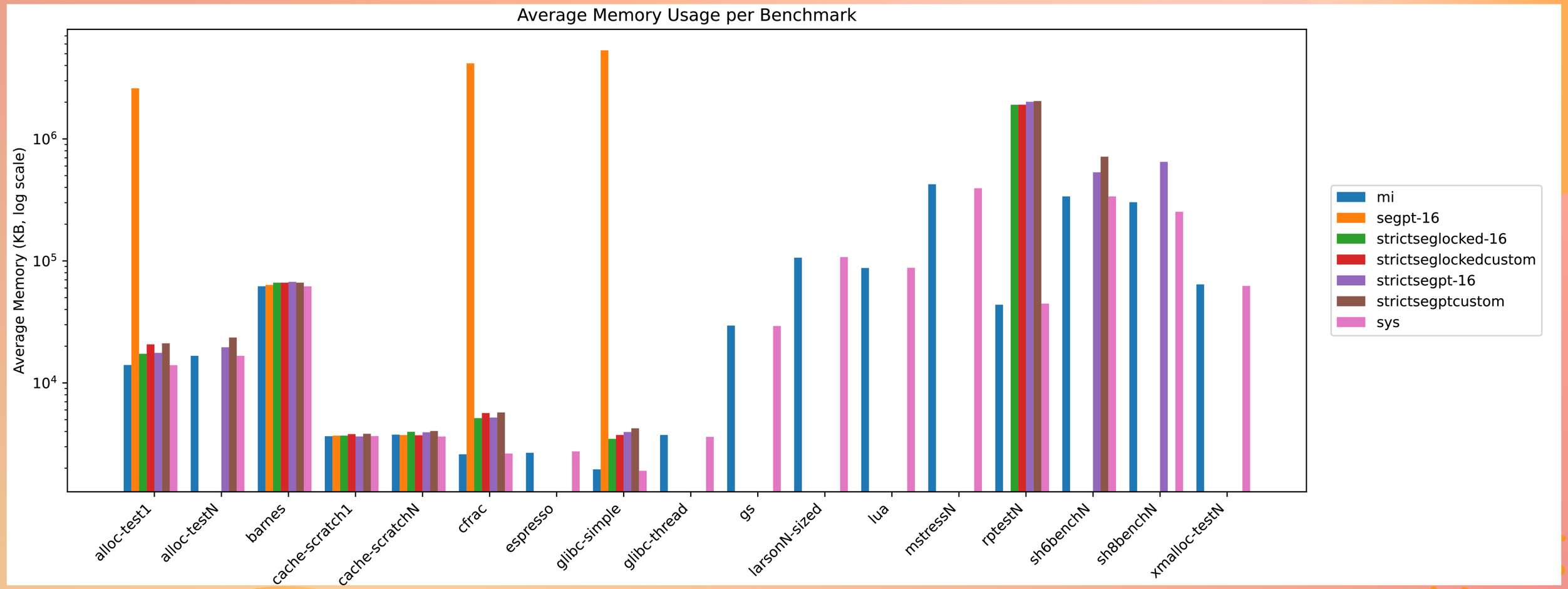

Average Memory Usage per Benchmark
Average Memory (log scale)
10^4
Kingsley16Bins
Kingsley20Bins
Kingsley25Bins
KingsleyAlloc
KingsleyBigChunk
KingsleySLL
alloc-test1
barnes
cache-scratch1
cache-scratchN
cfrac
glibc-simple

CUSTOM ALLOCATORS

- Created a simple MmapArena

- Based on Segregated Policy created
  1) SegPT-16bins
  2) StrictSegPT-16bins
  3) StrictSegLocked-16bins
  4) StrictSegPTCustom (using a custom policy for segregation)
  5) StrictSegLockedCustom


Average Execution Time per Benchmark
Average Time (seconds, log scale)
10^1
10^0
alloc-test1
alloc-testN
barnes
cache-scratch1
cache-scratchN
cfrac
espresso
glibc-simple
glibc-thread
gs
larsonN-sized
lua
mstressN
rptestN
sh6benchN
sh8benchN
xmalloc-testN
mi
segpt-16
strictseglocked-16
strictseglockedcustom
strictsegpt-16
strictsegptcustom
sys


Average Memory Usage per Benchmark
Average Memory (KB, log scale)
10^6
10^5
10^4
alloc-test1
alloc-testN
barnes
cache-scratch1
cache-scratchN
cfrac
espresso
glibc-simple
glibc-thread
gs
larsonN-sized
lua
mstressN
rptestN
sh6benchN
sh8benchN
xmalloc-testN
mi
segpt-16
strictseglocked-16
strictseglockedcustom
strictsegpt-16
strictsegptcustom
sys

REFERENCES

- Heap Layers Repository: https://github.com/emeryberger/Heap-Layers
- Mimalloc-Bench Repository: https://github.com/daanx/mimalloc-bench
- [1] A. Alexandrescu and E. D. Berger, "Policy-Based Memory Allocation," C++ Report, vol. 12, no. 9, Oct. 2000.
- [2] E. D. Berger, B. G. Zorn, and K. S. McKinley, "Composing High-Performance Memory Allocators," in Proc. ACM SIGPLAN 2001 Conf. Program. Lang. Des. Impl. (PLDI), Snowbird, UT, USA, 2001, pp. 114–124.

THANK YOU !