CSC469 Assignment 2

Abhinav Gupta, 998585963 Zeeshan Qureshi, 997108954

18 Mar 2013

Design

We implemented a simplified version of the Hoard allocator (Berger et all) with the following features:

- One heap for each processor and a shared global heap.
- Super-block size is equal to the system page size.
- Minimum slot size of 8.
- Super-block bins with slot sizes $2^3, 2^4, \ldots$.
- Bit-vector to mark empty blocks instead of a free list.
- A super-block is associated with each thread but threads that do not own that super-block can free it as well.
- No notion of an emptiness fraction. A super-block is moved to the global heap if emptied out.

Every time a new allocation request comes in the thread id is hashed to a processor heap and the request size is rounded up to the nearest power of two (slot). Then the heap is checked to see if the thread id already has a super-block allocated for the given slot size. If yes, then mark the slot used in that super-block's bit vector; otherwise allocate new super-block from global heap/extend heap (`sbreak`).

When a `free` request comes in the pointer is rounded down to the nearest super-block address, the heap is locked and the bit vector is updated, marking the slot unused. If the super-block becomes completely empty at that point, it is moved to the global heap.

Optimizations

The following optimizations were made:

- A bit vector in the super-block header to track free slots. This is more memory efficient than a list of empty super-blocks.
- Pre-computed 8-bit lookup tables to determine the first empty slot in a bit vector (the first zero-bit).

Possible further optimizations considered but not implemented due to lack of time:

- $2P$ heaps to reduce lock contention from multiple threads.
- Better hashing function in lieu of modulo that takes the processor id into account and assigns a heap accordingly.

Performance

All benchmarks were executed on Amazon EC2 High-CPU Extra Large Instances. These instances have 7 GB of RAM and 8 cores.

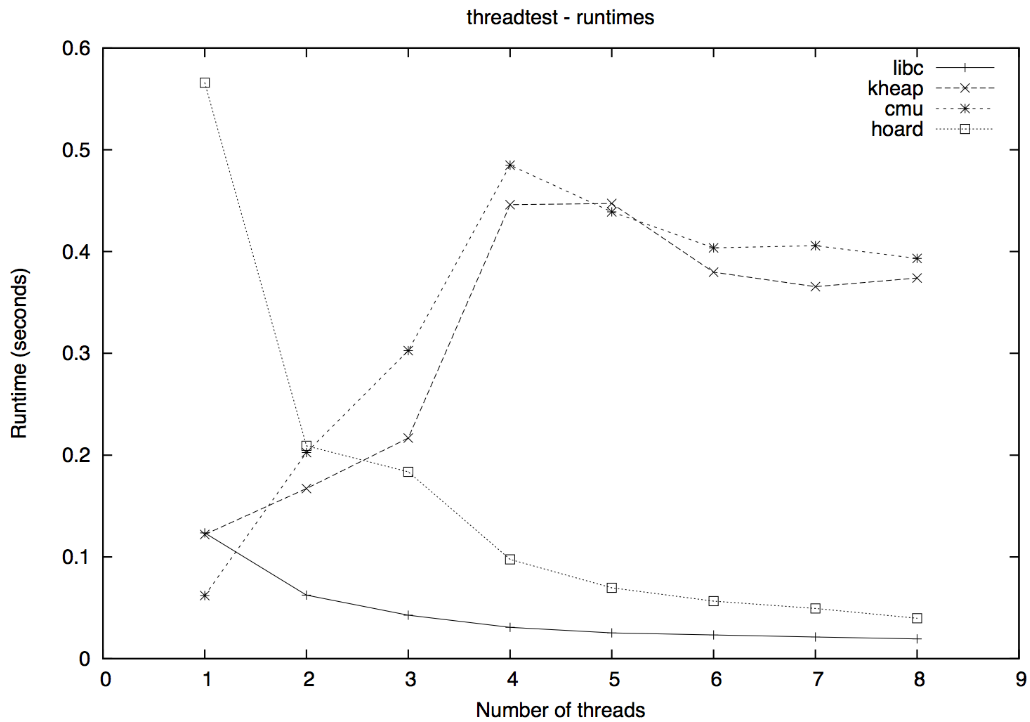
Memory Overhead

Each super-block has the following memory overhead:

- 36 byte fixed header including: signature, process id, thread id, slot size, free slots, total slots, data offset, and previous and next pointers.
- $\left\lceil \frac{\text{\# of slots}}{8} \right\rceil$ bytes for the bit vector aligned to an 8-byte boundary.

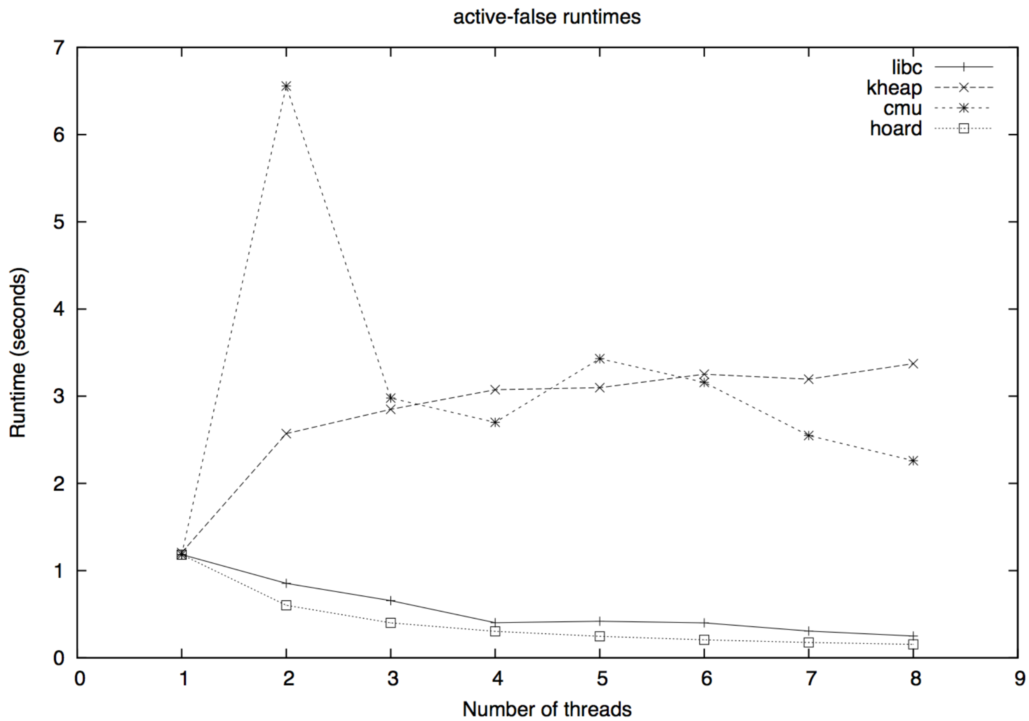
Thus, a 4K super-block has 498 8-byte slots, 250 16-byte slots, 125 32-byte slots, ...

Scalability



Because of the overhead of maintaining separate heaps and super-block bins our implementation is slower than kheap, CMU and libc on a single-threaded test, but as the number of processes increases we beat both, kheap and CMU and come very close to libc.

Active False Sharing

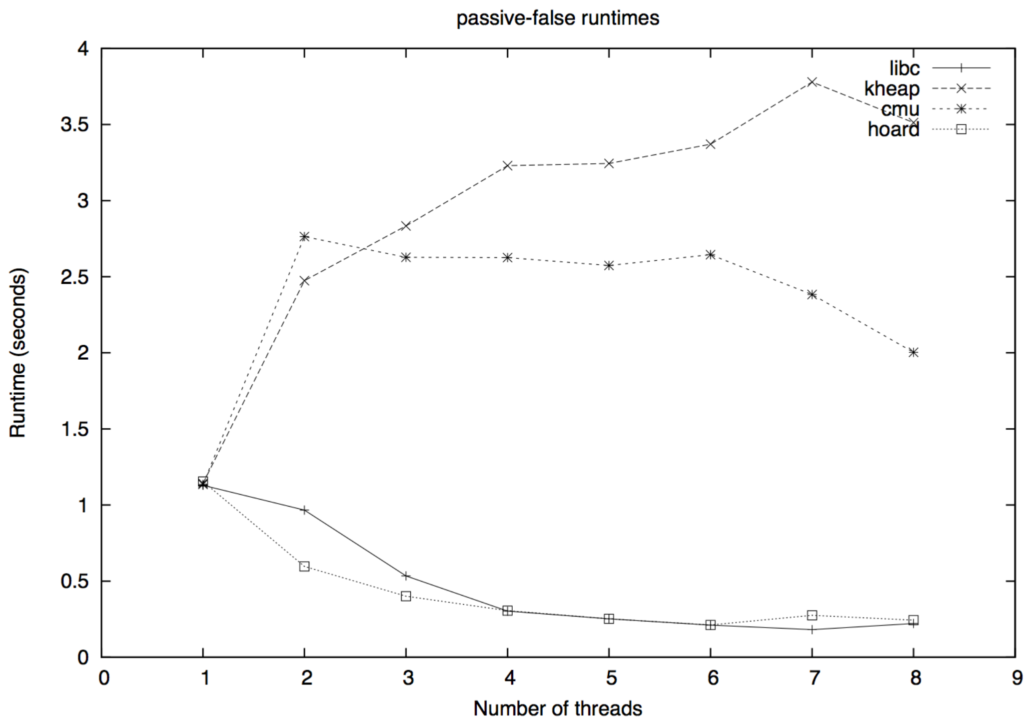


We beat all three, libc, kheap and CMU, on this benchmark.

Having thread owned super-blocks made sure that the probability of active false sharing is very low.

Since a typical page size is 4KB, and memory requests are generally of the same size, and the number of threads is bounded, even if every thread allocates only a single object, the maximum allocator overhead per thread is 4KB. Thus, with an SMP system with 100 threads and 8 processes the memory fragmentation will be roughly 1MB. Since super-blocks are moved through the global heap and re-used for successive allocations, therefore for most use cases, the fragmentation will be really low.

Passive False Sharing



We beat kheap and CMU and were very close (sometimes ahead) of libc.

Hashing process ids to heaps reduces the probability of passive false sharing.

Larson

The benchmark was taking far too long to execute on all implementations but libc, so we were unable to compare our performance to others due to lack of time.

Bibliography

- D. Porter. *The Art and Science of Memory Allocation.* http://www.cs.stonybrook.edu/~porter/courses/cse506/f11/slides/malloc.pdf
- E.D. Berger, K.S. McKinley, R.D. Blumofe, P.R. Wilson. *Hoard: A Scalable Memory Allocator for Multithreaded Applications.*