CSCE 313.506-F18:
Programming Assignment 1

Tristan Seifert

Due: Friday, Sep. 14, 2018

---

1 Overview

Compile the program using the provided `makefile`, which produces an executable called `memtest`: this program accepts the parameters `-b` to specify the smallest block size, and `-s` for specifying the overall memory size, in bytes.

Internally, the buddy allocator uses the `BlockHeader` structs at the start of each block for housekeeping and meta-data. These blocks contain the pointers used by the `LinkedList` class to traverse the free list, as well as allocated/free and validity flags, as well as the block's size.

To test the optimizations, set the `USE_OPTIMIZED` define to 1 at the top of `BuddyAllocator.cpp`.

2 Improvements

The initial implementation of the allocator was rather naive in how it handled memory: blocks are split at allocation time, and merged when they are freed, and performed a lot of sanity checks to prevent mis-use of the API it exposes. This approach works, but it makes freeing memory quite slow.

An improvement that I applied to my allocator was to first remove those sanity checks – testing whether a freelist operation was valid, finding corrupted blocks, etc. – that ran for every call to `BuddyAllocator::alloc(size_t)` and `BuddyAllocator::free(void *)`. This reduced the runtime for allocations by approximately 5-15%, depending on how many memory allocation cycles took place.

Additionally, deallocation performance can be improved significantly by deferring merging of blocks until absolutely necessary. I tested this by not merging blocks in `BuddyAllocator::free(void *)`: instead, they are simply returned to the freelist for later allocations, and are merged when there are no larger blocks to split available. While this can increase fragmentation considerably, for the essentially random allocation pattern used by the Ackermann function, this reduced the number of merges by almost 95%, improving performance of deallocation by approximately 20%.

A possible optimization, if the sizes of allocations are known beforehand, a zone allocator can be a better option: blocks of a fixed size are pre-allocated, and a free list contains those blocks that haven't been allocated. This is similar to how the buddy allocator behaves, if there were only one size of block.

3 Performance Data

Data on Page 2

n	m	cycles	Time (sec)	Time (with improvements, sec)
3	2	541	0.028	0.021
3	4	10307	0.579	0.43
3	6	172233	11.3	7.9
3	8	2785999	176.91	116.22
2	2	27	0.0061	0.0033
2	4	65	0.0065	0.0039
2	6	119	0.00502	0.0039
2	8	189	0.012	0.0084
1	2	6	0.000259	0.000259
1	4	10	0.002231	0.002231
1	6	14	0.000961	0.000961
1	8	18	0.000271	0.000271

Table 1: Performance Data for the Buddy Allocator

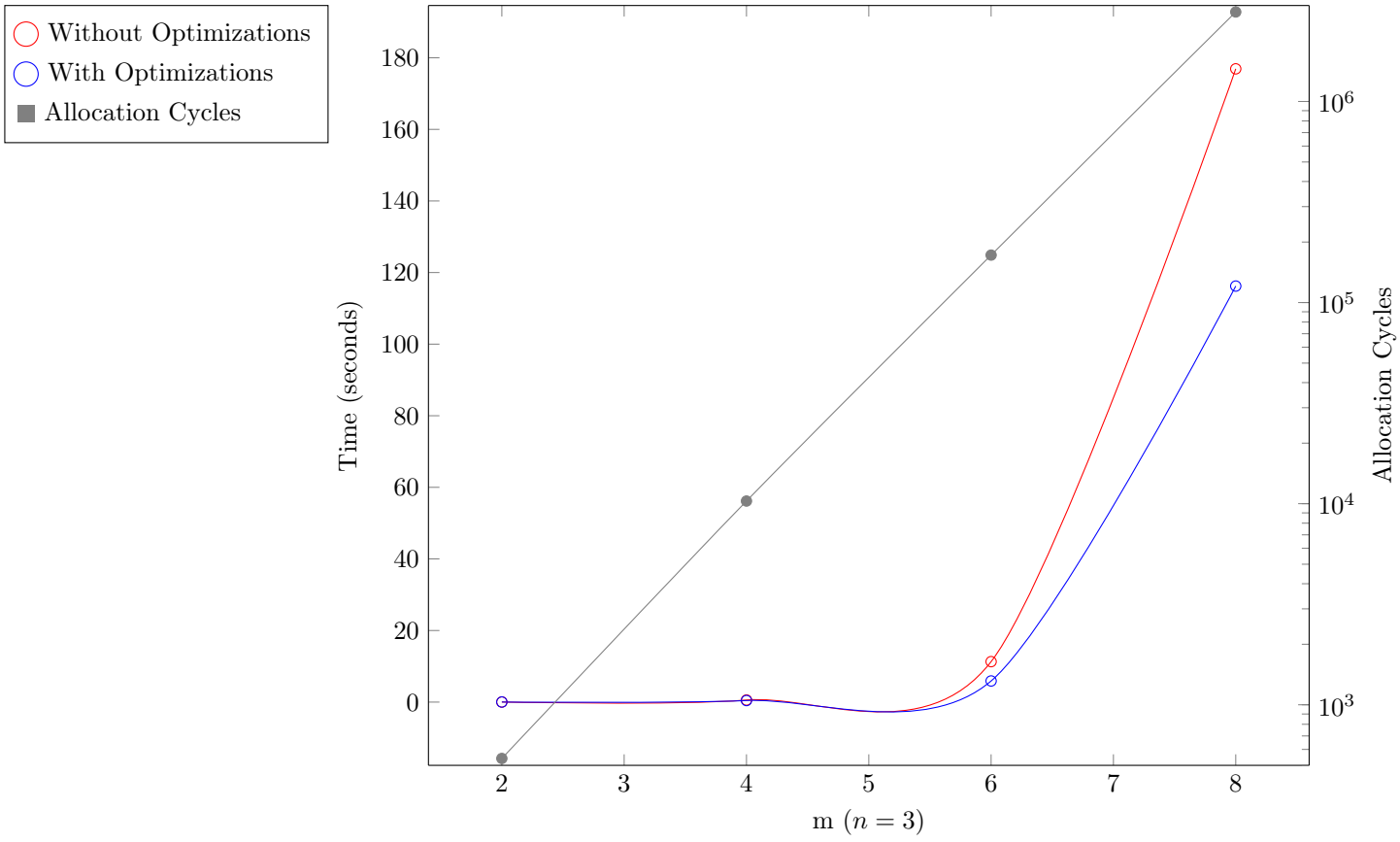

Figure 1: Comparison of Performance