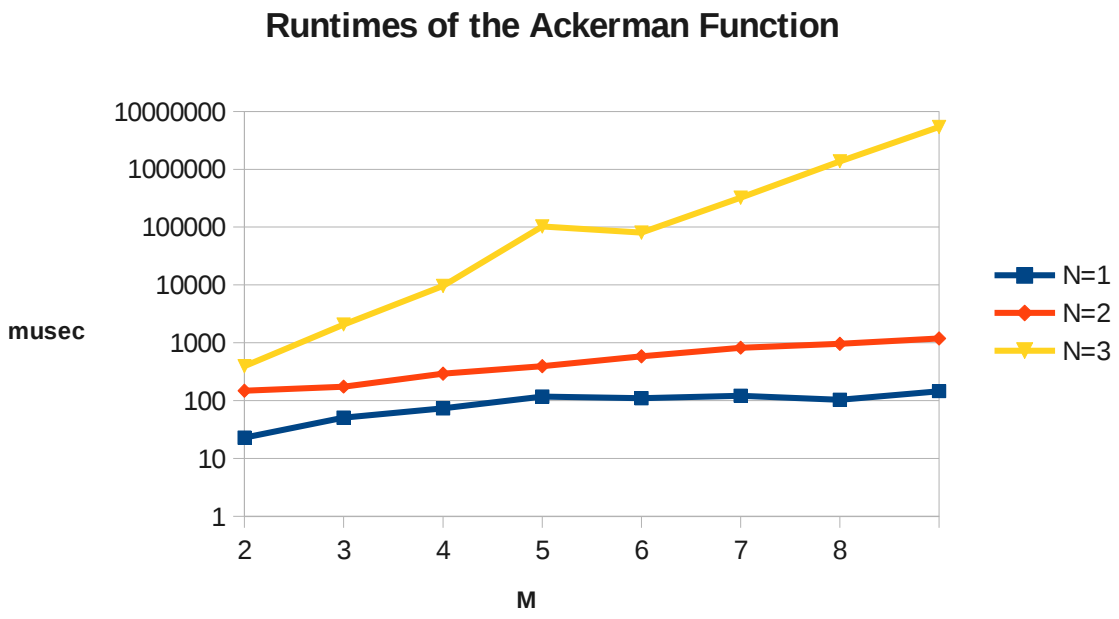
Matthew Harper

Analysis of Memory Allocator

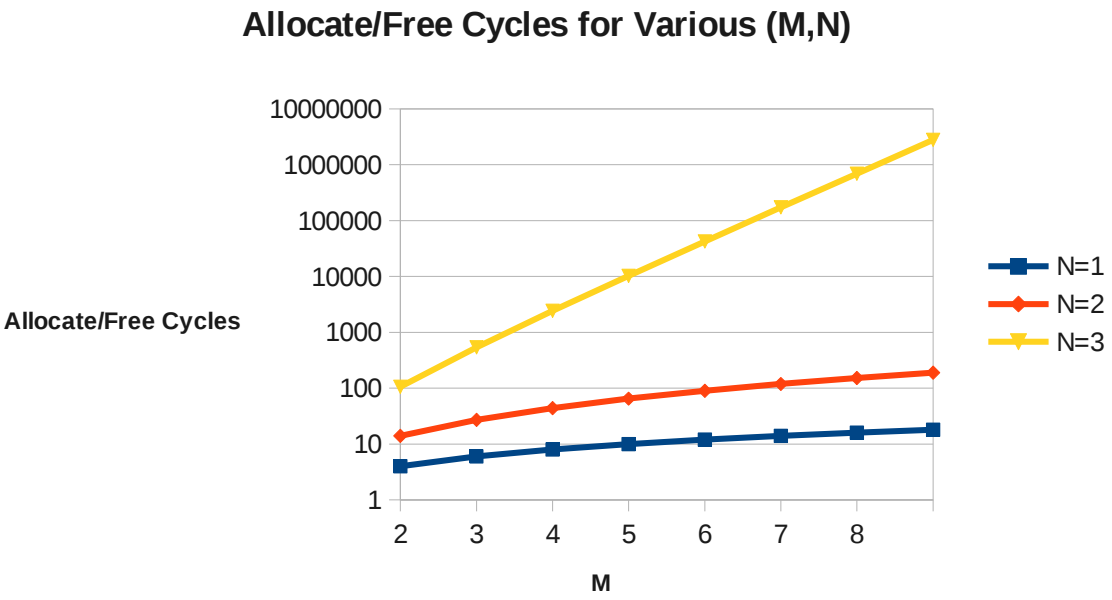
As demonstrated in Graph 1 , for an input N into the Ackerman function, the time required to compute the function increases with increasing values of M. The runtimes for N=1 and N=2 are relatively similar. However, the runtimes for N=3 are orders of magnitude greater than N=1 or N=2. From (3,2) onward, the Ackerman function takes at least 10 times as long to complete compared to any other N value. The number of total allocation and deallocations follows a given (N,M) consistently increases with increasing M values, as is evidenced in Graph 2. As with the runtimes, the number of allocations and frees required by N=3 is massively greater than either N=1 or N=2. The total number of cycles required seems to increase at an exponential rate.

One of the largest bottlenecks slowing the performance of the allocator is the coalesce function which merges two free “buddy” blocks into a larger free block. Ideally, merging could be done in constant time at the time “free” is called. However, the function to merge blocks in this implementation requires looping over the entire free list. This loop is required because of the way large blocks are broken down into smaller blocks. When a block is split, the new total amount of free space between the headers is not the same as the amount of free space before the split. One of the blocks must sacrifice space to create new header for the new block. In this implementation, the right block of a split is rounded down to the previous power of two. Because the two new blocks are of different sizes, (but are still buddies), the coalesce function must check every free list of one size bigger to find it’s buddy. Looping through the entire free list also guarantees that a merge will have the new block ready to find a buddy in the same loop.

To improve the performance of this implementation, fragmentation could be minimized during block splitting. For example, the current implementation splits a block into two uneven sized blocks. The larger of the two split blocks has exactly half the amount of free space as the original block. The smaller of the split blocks has free space equal to a quarter of the original space. This means that with every split, up to a quarter of the free space before the split is completely unavailable to be allocated to the user. An improved, more efficient version of an allocator could form blocks out of the unavailable space and add them to the free list to be used.



Graph 1


Allocate/Free Cycles for Various (M,N)
Allocate/Free Cycles
10000000
1000000
100000
10000
1000
100
10
1
2
3
4
5
6
7
8
M
N=1
N=2
N=3

Graph 2