Analysis: Memory Allocator

Suqian Wang

From the result of repeated experiments for increasing parameters a and b in the Ackerman function, we can see the numbers of allocate/free operations are nearly linear and small when a <= 2. When a = 3 the numbers of allocate/free operation grows rapidly and extremely large compared to before. The compilation time has a similar tendency, but its variation when a=3 is larger. See appendix for result presented in charts.

The bottlenecks in the system have two aspects. One situation occurs when the user requests small blocks but there are only large blocks exists. First, there will be a loop from the free list of basic block size until it finds the list of the current smallest block which also enough for the user. In addition, if the user keeps allocating small blocks and not freeing the memory, eventually, the memory segment will be small and scattered which will be hard to use. Then, the compiler takes a long time to recursively breaking the large block down to small blocks until there is a block whose size is just suitable for the user. The other situation occurs when user trying to free its allocated memory recursively. If the buddy does not exist or at the end of the free list, the user has to traverse the whole free list. Since this is a recursive procedure, the user has to traverse every list if every merged block has their buddy and have to traverse the whole list to find them.

The allocate bottleneck is kind of a behavior of the buddy system. However, we can have a better implementation for the freeing memory part. We could have a Boolean variable serve as flag in each header, set its value to true if its buddy is in the list and vice versa, therefore, we don’t have to traverse the list and efficiency can be improved.

Appendix

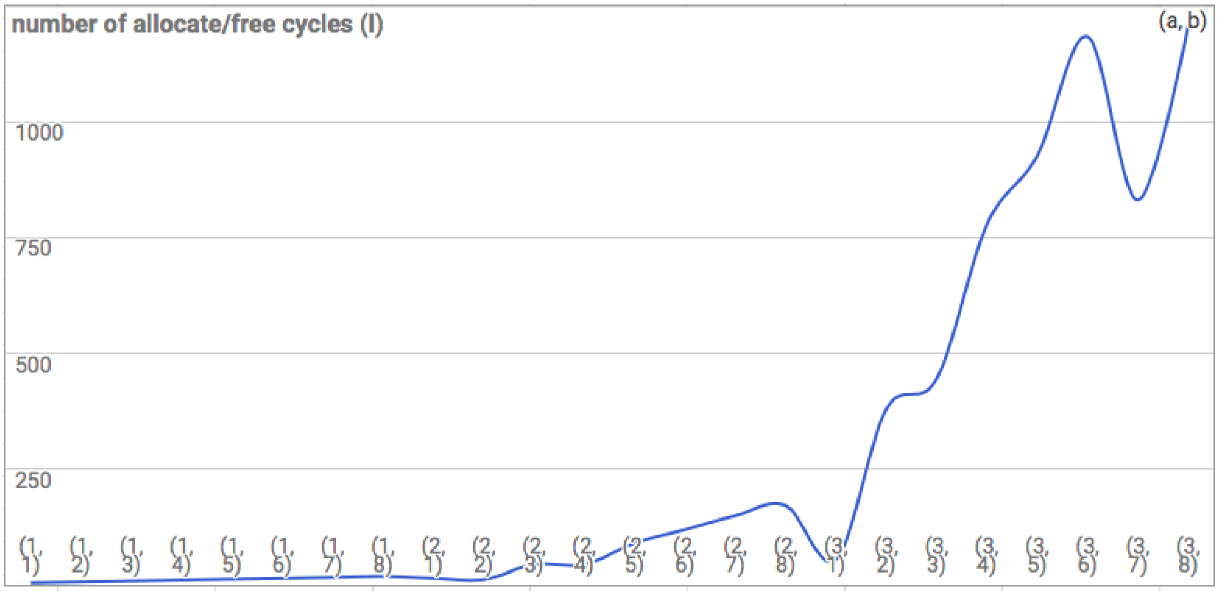

Figure 1 number of allocate/free operations

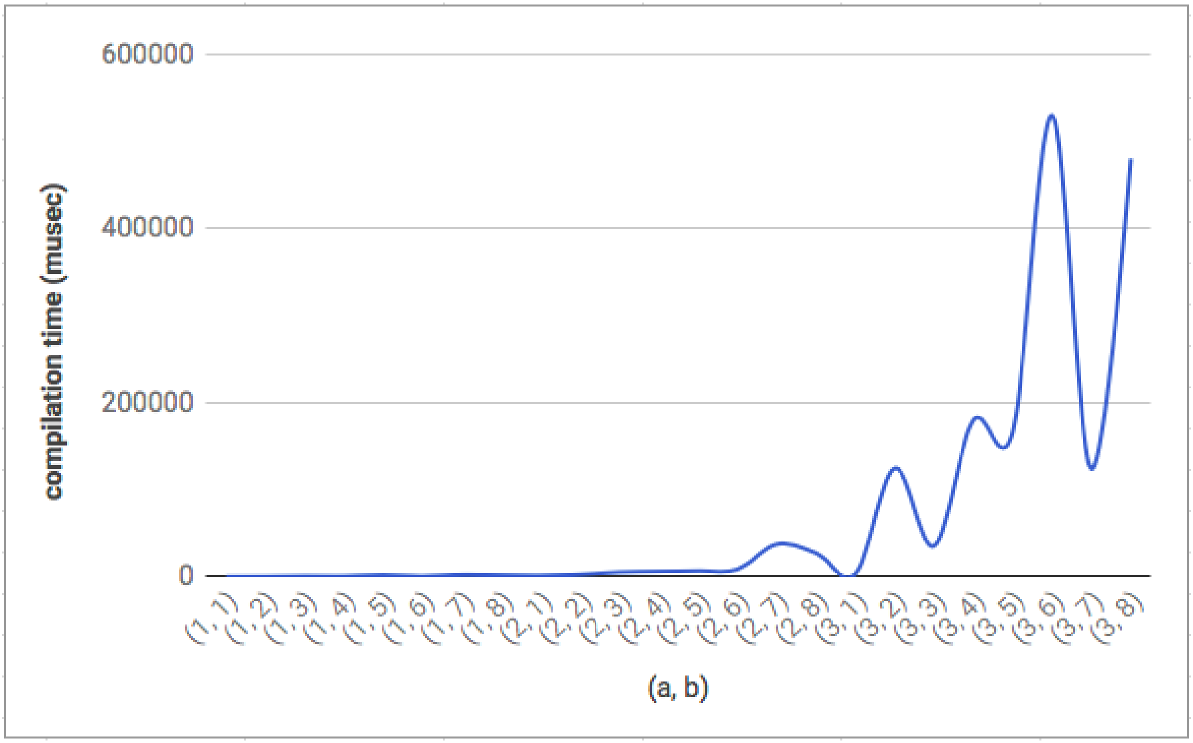

Figure 2 compilation time