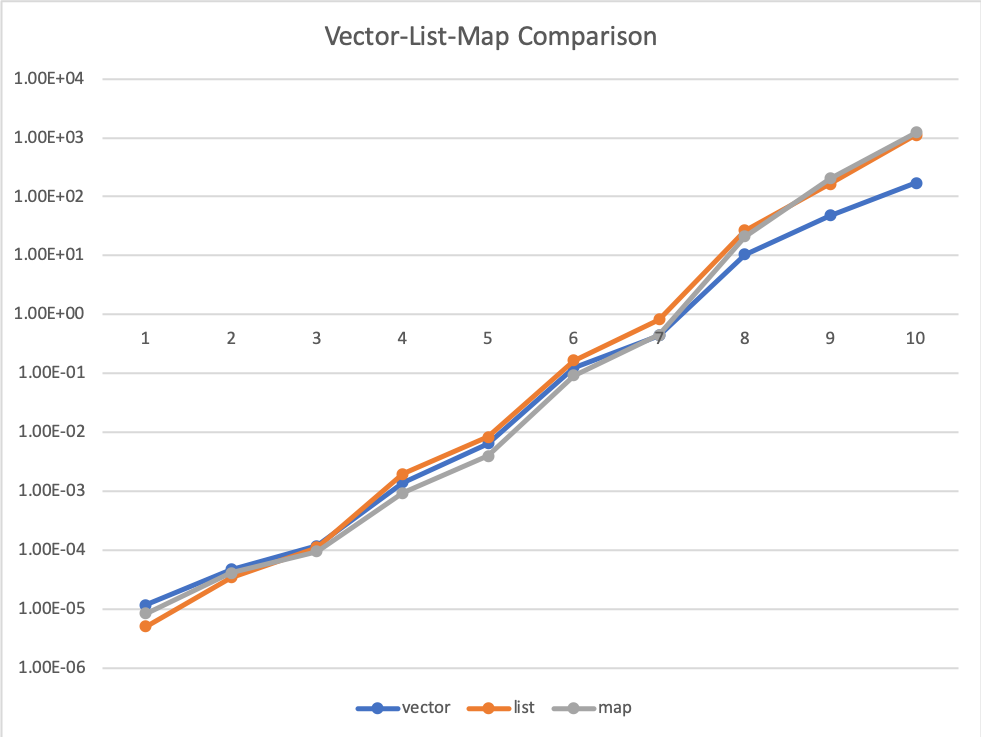
James Yang

UNI: jy2816

February 25, 2020

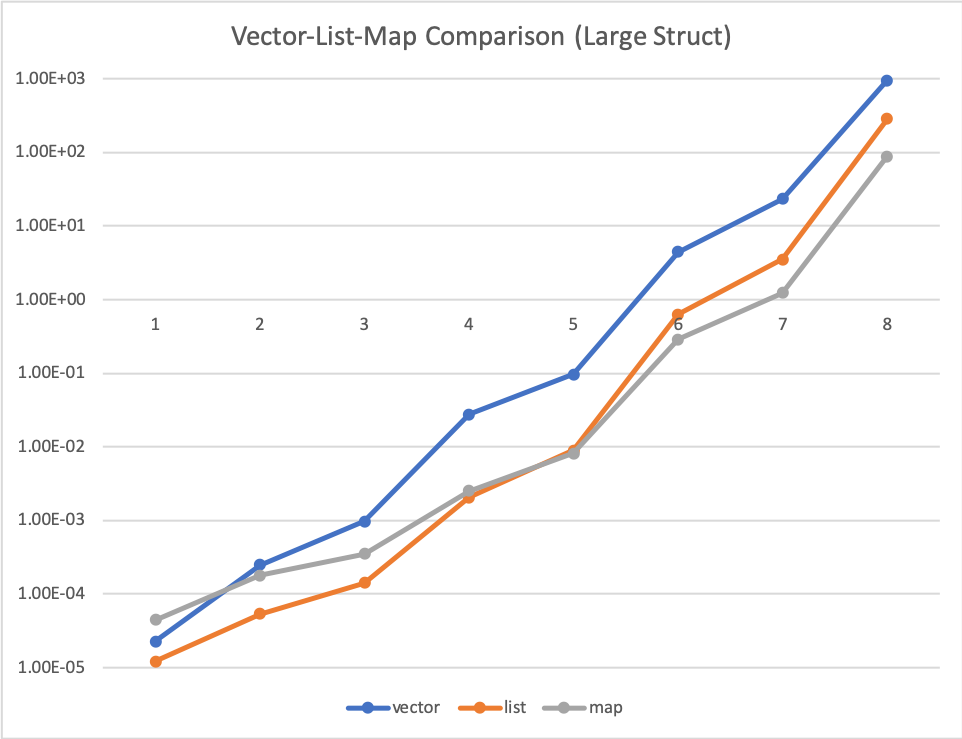
Vector List Map Benchmark Writeup



The above figure shows the first benchmark results for vector, list, and map. The graph is in log-scale (base 10). The number of elements ranged from [10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000, 200000]. The result shows that vector actually performs the worst when the number of elements is relatively low (10, 50, 100). This is because vector resizes more frequently and every resize allocates a new chunk of memory that will lead to cache misses. Furthermore, inserting into vector causes shifting of the elements that requires copying. List is relatively cheap in terms of allocating memory and inserting (O(1)). The pointer indirection is too few to cause any overhead in this case. Map takes a little more time because of the log(N) reordering that happens with lots of cache misses.

Between 3 and 7 (100 – 10000), map seems to perform best. This is because log(N) has better asymptotic performance and there is enough cache misses that list’s O(N) algorithm is clearly slower. Vector starts to perform better because the cache hits that come from its contiguous structure starts to shine when there are a lot more cache misses for list and map. It still can’t beat map because of all the copying.

The largest sized trials show that vector greatly outperforms list and map. This is because cache hits constantly help with vector’s performance while cache misses constantly take a toll on list and map with pointer indirection.



The above shows the same benchmark but with a large struct. We created a struct to contain an array of 1000 integers and also contains an integer representing the randomly generated integer. This exercise essentially mimics the structure of list in vector by separating out the actual randomly generated integers in each element by 4000 bytes. Now we lose the cache benefits for the vector and it just performs incredibly bad. This is kind of embarrassing.