Tanner Pflager
CS475
PA3 Report

In practice, the memory efficient algorithm for KnapSack Problem performed well in comparison to the full-table code, at least for my test cases.

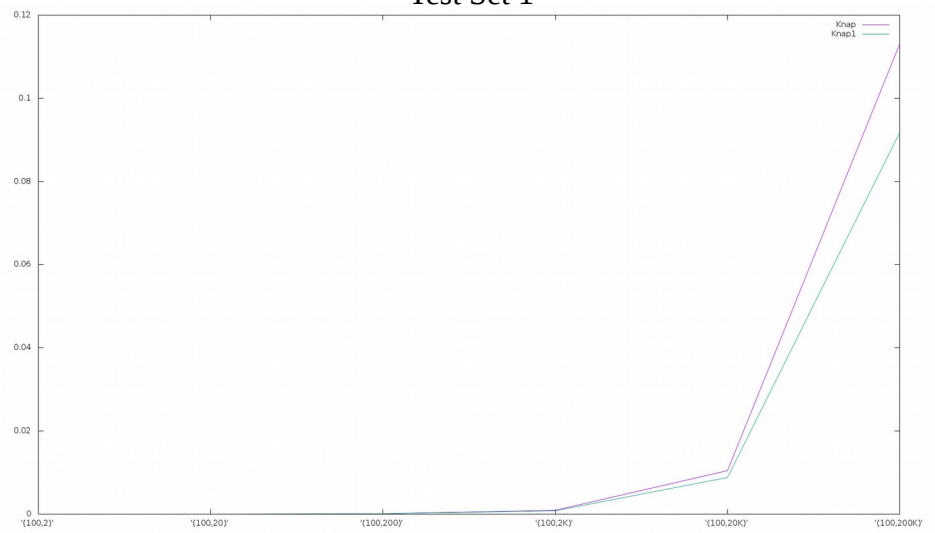
I ran a set of 3 different tests to analyze the performance of each algorithm. The first set consists of a fixed value of N, the next set consists of a fixed value of C, and the last set consists of a fixed value N*C.

I used Test Sets 1 and 2 to see if either N or C had more effect than the other. The results led me to the conclusion that C has a greater effect on the performance of both algorithms than N.

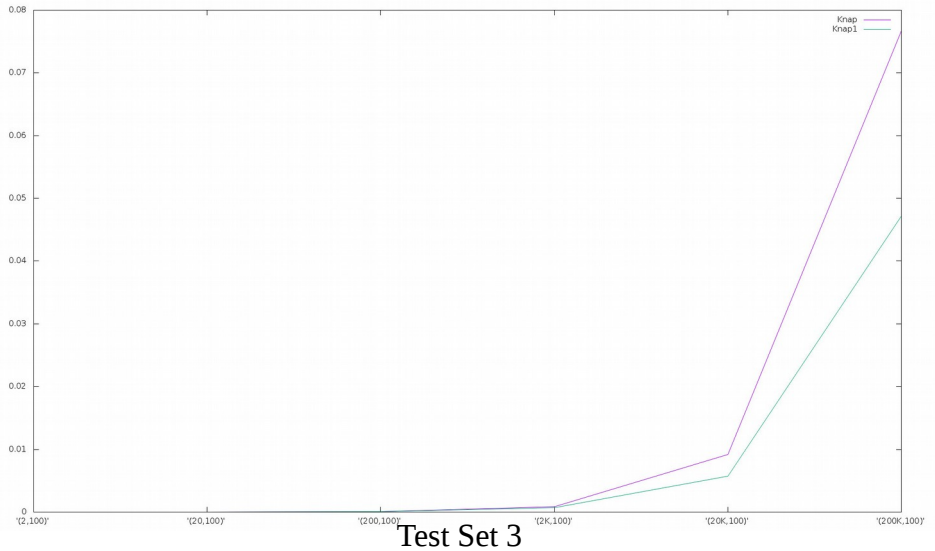
In Test Set 3, (N*C) was always equal to 4Billion. As you can see in the graph that the full-table code (the purple line) drops to 0. Note that this is because the full-table code failed on all 25 attempts. The case it failed N = 20Million and the C = 200. Test Set 3 shows that with large values of N and C, the Memory Efficient KnapSack algorithm out performs its full-table counter part. For the test cases ran in Test Set 3, we can see that the KnapSack problem is memory bound not compute bound.

The full-table algorithm is limited to
(C*N) < 6, 000, 000, 000
Where as the the memory efficient algorithm is limited to
(N+C) < 6, 000, 000, 000

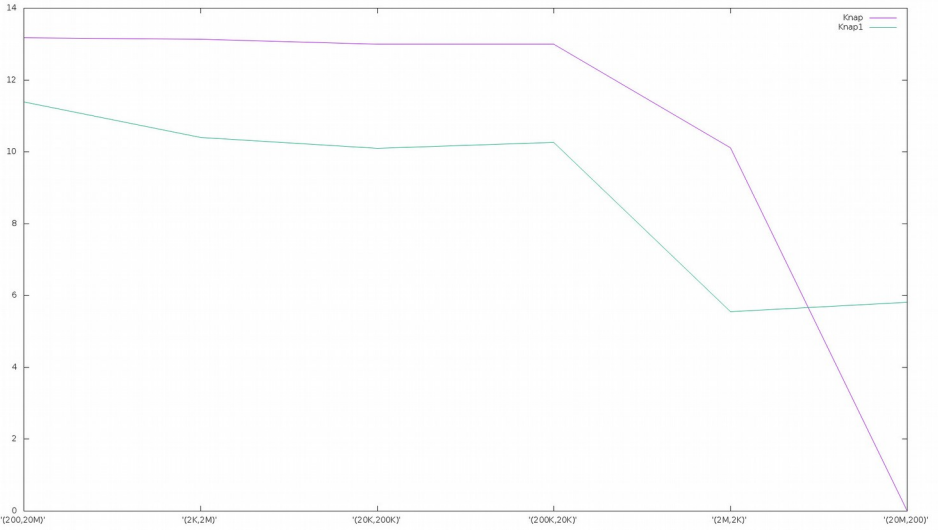
Test Set 1



Test Set 2



Test Set 3

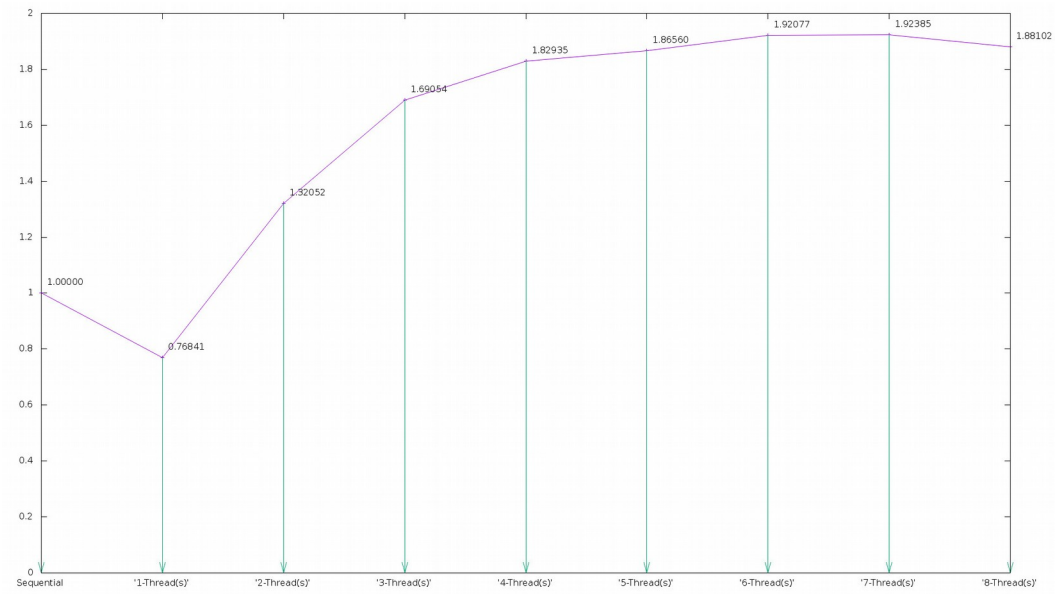


Knap2

Version 1 Depth = 4

Knap3

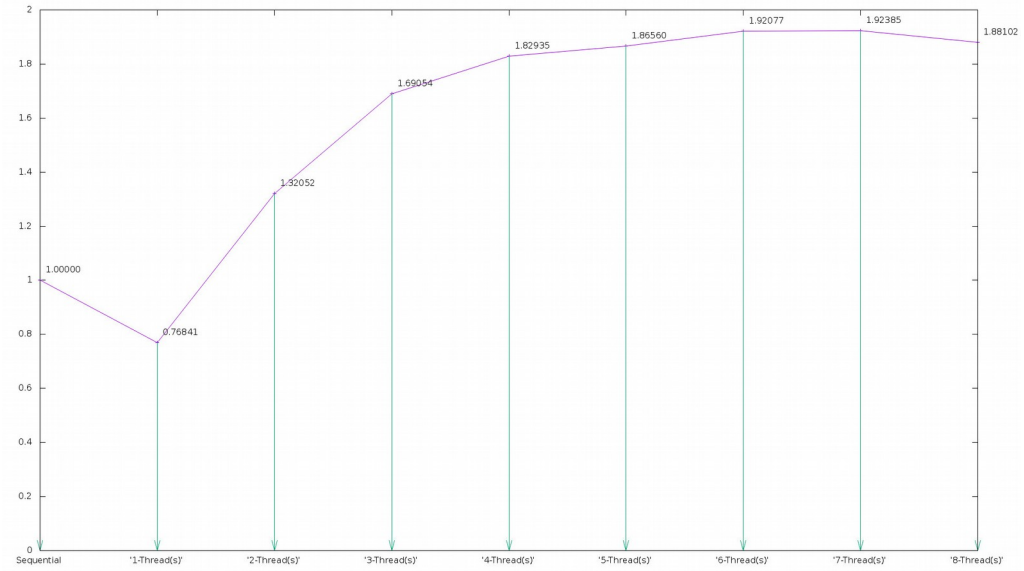
In Knap3, I didn't not achieve Ideal speedup. The reason, I



believe is ideal speedup is because of the fork join overhead that was introduced when parallelizing the for loops for each leaf on the tree.

Knap4

In my attempts in using coarse grain parallelization and fine grain parallelization, I did not achieve good speedup. However, I believe I know the



reason for this. I to set the depth = 8 for my testing. However, this causes a problem when trying to parallelize the for loops as well as the recursive calls. As soon as the tree reached the depth of 8, the threads would be forced to context switch during the iterations of the for loops (if the omp run time system decided so). This caused additional overhead and hurt my over all speedup. If I were to chose a new depth it would be 4 so that every branch that is being executed has a “team” of two threads that could be used for the for loops in the get_last_row calls. The Speed up was similar to that of Knap3