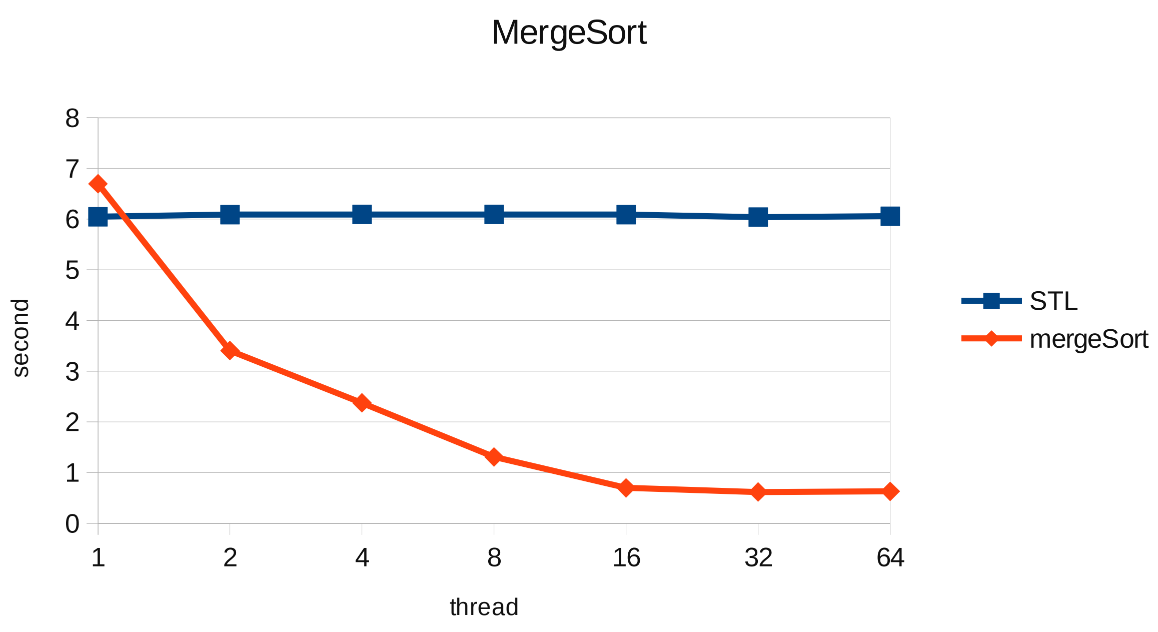
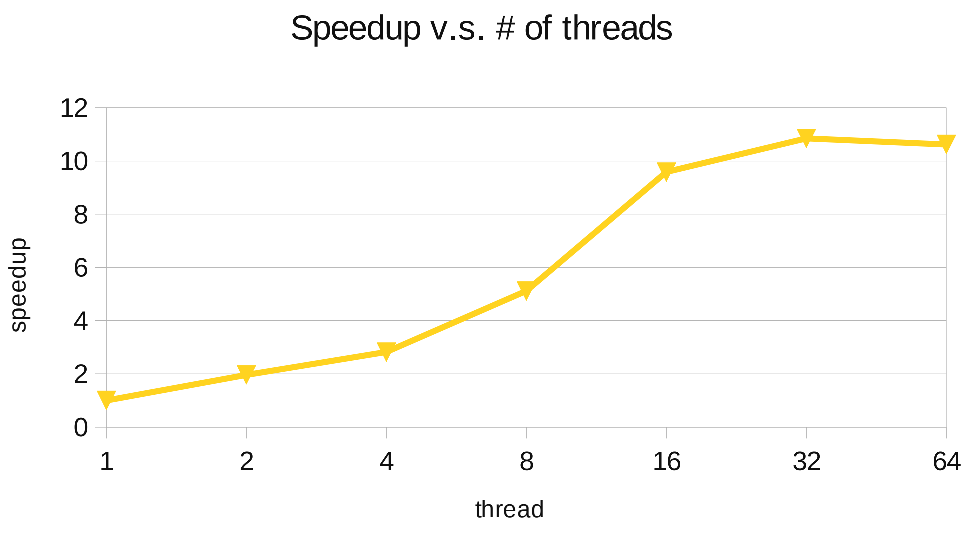
CME213 PA4

Dong-Bang Tsai

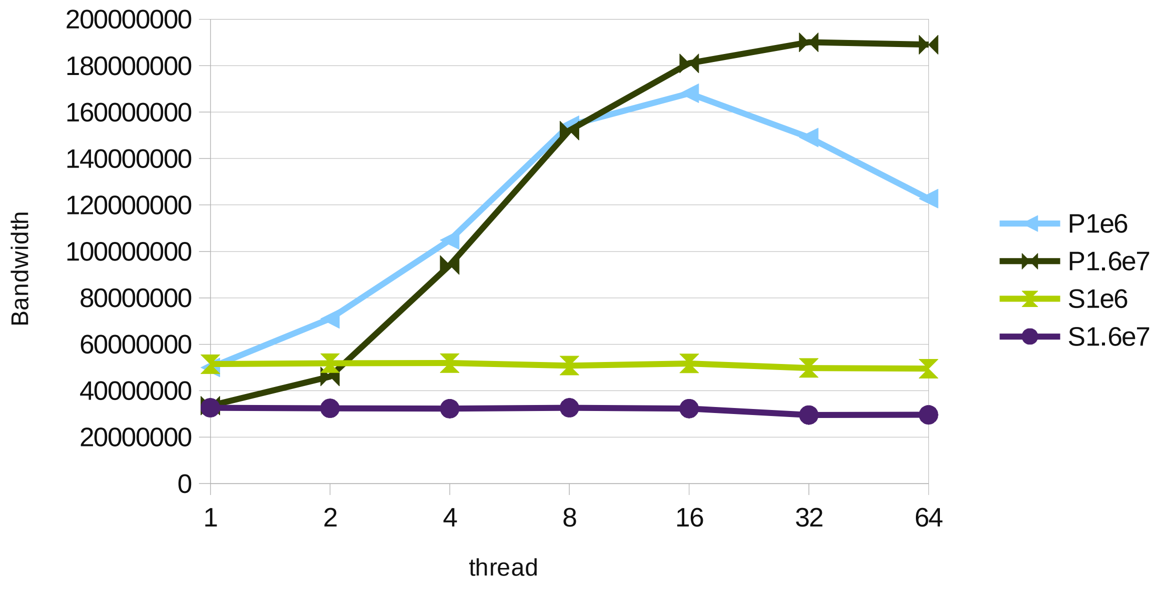
(1) Since most of the operations we have done are simply moving and copying data, those implementations are memory bound. However, there are signification speedup when those sort algorithms run in multithreads; the reason is each physical core has it own memory controller, which will provide dedicated memory bandwidth.

(2)





(3) Here, the blockSize I used in radixsort is 8192, and numBits is 8. I found that it's the near optimal parameter for this implementation.



a) For serial implementation, sorting with 1,000,000 has slightly better performance. However, for parallel implementation, the performance of sorting 1,000,000 will start to decrease when # of threads is higher than 16, and the performance of sorting 16,000,000 at least doesn't start to decease until 64 threads.

The reason is very simple. The physical processors in this machine is 12, once the # of threads is higher than 12, the parallelization overhead will be dramatically increased. Since sorting less data also trends to have more overhead, and combine those two factors, the performance of sorting 1,000,000 elements start to decrease when # of threads is 16.

(4)

a) For c < 1, in both algorithms, the speedup will be dramatically increased when we increased c. Although, there is still small penalty of overhead such that it's not linear speedup as shown in the figure.

b) When 1 < c < 4, there is small increased in performance when we increase c. Although the numbers of threads running in the kernel are larger than physical processors, however, some of them are not running all the time in the kernel, so the extra threads can take advantage of that which increase the performance.

c) When c>>10, the overhead in parallelization will be too large, so the performance will be decreased.

According to my experiment, the number of elements will not change the behavior too much. Although when the number of elements are larger, the overhead due to the parallelization may be able to ignore when c is not too large. However, when c is actually too large, the overhead will cause the performance decreased.