Akash Sinha
NU ID: 001344425

Assignment 5

Parallel Sort algorithm

The program aims to sort an array leveraging the multithreading capability of the processor to improve performance. Merge Sort is a good candidate for this, as it consists of distinct segments that can be processed in parallel then merged together. As such, bottom up merge sort with multithreading using a fixed thread pool has been used. Assume size of the array is N and cutoff size is C.

Algorithm:

1. Create a fixed thread pool of desired size
2. Create a callable task queue
3. Create N/C System sort tasks each operating on C elements and push them to the queue
4. Call InvokeAll on the queue to complete the tasks in parallel till they are all completed
5. Create merge tasks that operate on twice as many elements as the previous cutoff and push them to the queue
6. Call InvokeAll on the queue to complete the tasks in parallel till they are all completed
7. Repeat steps 5-6 until the cutoff size includes the whole array

Benchmark Data –

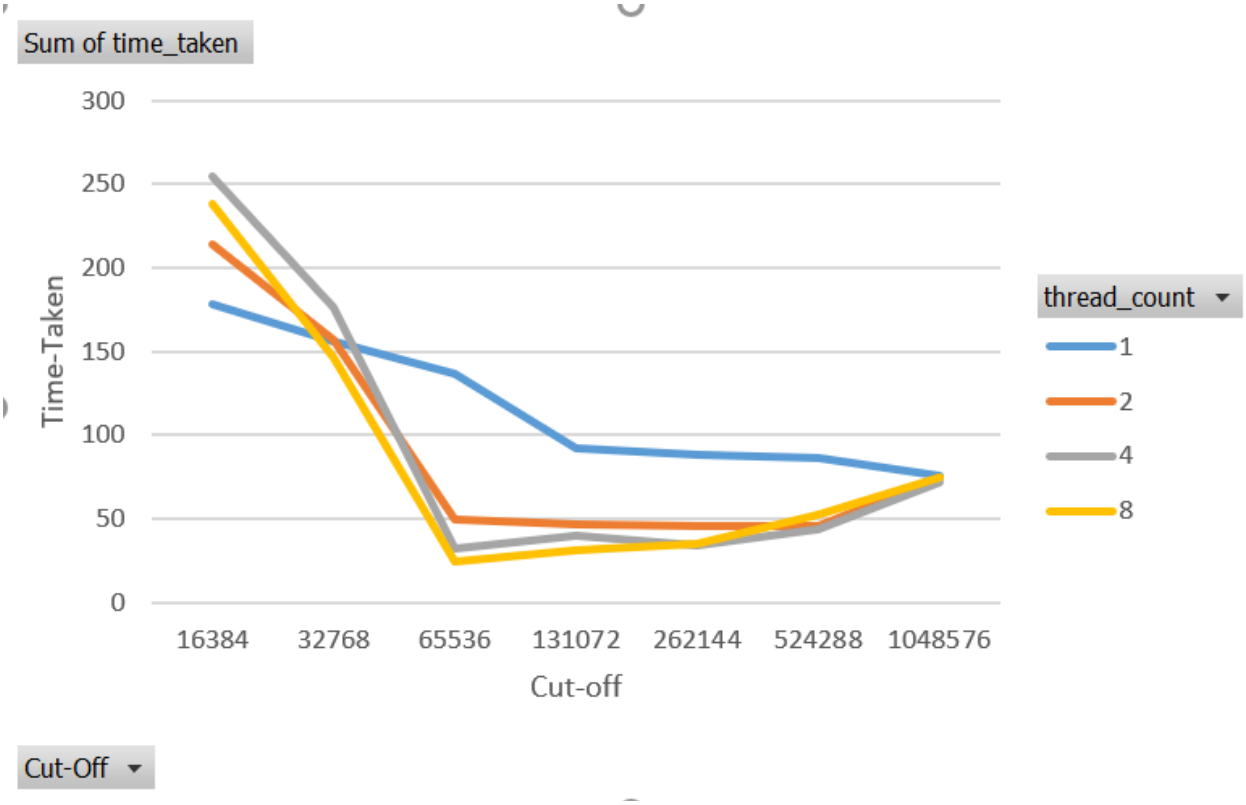
The benchmarking was done on an Intel i7-8th Gen with 2 cores and 4 threads.
Thread Pool sizes tested were 1,2,4,8.
Array Sizes tested were from 220 to 224.
Cutoffs tested were from 214 to 224.

The goal is to measure time taken to sort with respect to:

1. Size of the array
2. Cutoff size
3. Thread count in the pool

Data - dataAll.csv

Graph -



Observations/Conclusions –

1. The time taken to sort on a single thread decreases as cutoff size increases. This makes sense as more of the array is sorted using the highly efficient system sort rather than the custom merge.
2. However, for sorting on multiple threads, the smaller cutoff means more segments that can be processed in parallel improving the time. For 4 threads, the trend almost reverses where the time taken increases with cutoff size as it becomes less able to leverage multithreading.
3. However, since the processor used has only 4 threads, using a pool of 8 threads doesn't improve the performance and, in some cases, even degrades it as the thread overhead is increased without increase in multiprocessing.