# Report- Parallelizing the CFS search using OpenMP

## Approach

I have evaluated the runtimes of all 3 functions ie. Count sort, serial radix sort and parallel radix sort. I have been able to observe that count-sort has the best performance speed wise usually outperforming both parallel radix and serial radix. I implemented the serial radix sort and parallelized two loops in the radix sort to obtain the parallel radix sort. The loops which were parallelized were the ones which were used to assign the sorted values and the loop used to count the occurrence of a particular digit.

## Observation

The program was run on an Intel Core i5-8250(8th gen) processor which has 4 cores and 8 logical processors. Used gcc version 7.4.0 to compile the code.

Most significantly it was observed that the runtimes varied between different runs with the same configuration. Therefore, the tabulated values are just the typical values.

The observations are in line with expectations for a 4 core CPU. The runtime decreases significantly till around 4 threads. Increasing the number of threads further doesn’t significantly improve performance. Below is a comparison of the observed runtimes for the RMAT22 dataset with the values which we would have expected with Amdahl’s Law. I have calculated the value of ‘f’ based on the timings of the first and second runs ie. with thread count 1 and 2.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of threads** | **Speedup according to Amdahls Law** | **Expected time acc AL** | **Actual values(Parallel radix schedule static,1)** |
| 1 | 1 | 55.6 | 55.6 |
| 2 | 1.666666667 | 33.36 | 33.02 |
| 3 | 2.142857143 | 25.94666667 | 22.258 |
| 4 | 2.5 | 22.24 | 21.166 |
| 5 | 2.777777778 | 20.016 | 16.898 |
| 6 | 3 | 18.53333333 | 17.4564 |
| 7 | 3.181818182 | 17.47428571 | 19.351 |
| 8 | 3.333333333 | 16.68 | 18.711 |
| 16 | 4 | 13.9 | 16.821 |

Runtimes with different schedule configurations-

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of threads** | **Schedule Guided** | **Schedule Static** | **Schedule dynamic** |
| 1 | 50.5266 | 55.6 | 153.534 |
| 2 | 26.02 | 33.02 | 83.07 |
| 3 | 16.509 | 22.258 | 64.994 |
| 4 | 17.436061 | 21.166 | 61.64 |
| 5 | 18.1 | 16.898 | 49.768 |
| 6 | 14.59 | 17.4564 | 49.904 |
| 7 |  | 19.351 | 43.787 |
| 8 | 13.59 | 18.711 | 47.943 |
| 16 | 13.532579 | 12.7241 | 41.44 |

Runtimes for different sorting functions.

|  |  |  |
| --- | --- | --- |
| **Configurations** | **RMAT22(Threads=1)** | **wiki-vote(Threads=1)** |
| Count sort | 10.18308 | 0.000797 |
| Serial radix sort | 11.346 | 0.005116 |
| Parallel radix sort(guided,1) | 50.5266 | 0.016244 |
| Parallel radix sort(static) | 55.6 | 0.015594 |
| Parallel radix sort(dynamic) | 153.534 | 0.033749 |

All values in the table are in seconds.

## Conclusion

In general, we are able to observe that the performance increases proportional to the number of threads till thread number 4. Further performance improvement is insignificant.

Further it was also observed that the overhead which was added for the parallel run for RMAT22 was ~40s while that added for ‘wiki-vote.txt’ was ~0.01s which suggests that the overhead might be due to cache coherency issues or something similar.

In conclusion for this scenario and this parallelization technique the modification has not helped in improving performance as the parallelization adds too much overhead which is not compensated for by running the workload on multiple cores.