Rohit Agarwal
B17100 17 Aug' 20

Intro to Heterogeneous Computing (CS508), IIT Mandi
Assignment 3 (OpenMP)

Number of processors in MPI: 8
Number of threads in OpenMp: 8

Note: Time taken by all the programs only includes the time for main execution (not for I/O). So, they can easily be compared.

Problem 3

Number of clusters = 5,
Sequential : 0.00106 s **MPI** : 0.000443 s **OpenMP** : 0.00226 s

Time taken by the MPI program (p = 8) is the least among all of them. This could be because it effectively parallelizes the for loop (which runs for iterations=150). The Sequential and OpenMP program takes almost a similar time**.** This might have occurred because the time reduced by parallel execution in OpenMp is almost the same to the overhead of fork & join. This will be more effective when number of iterations is more.
As we increase the number of processors in MPI, time taken by it also increases and surpasses that of sequential and OpenMP programs.
As we keep on decreasing the number of threads in OpenMp, time also reduces. **=> This shows that overhead associated with a fork & join is more on comparing with the time reduced due to parallel execution.**

Note: (Loop unrolling) Loops that run for a very small number of iterations (less than 5) are executed sequentially in openMP to avoid unnecessary fork-join overhead time.

Problem 2

Sequential : 0.000032 s **MPI** : 0.000358 s **OpenMP** : 0.0038 s

Time taken by the Sequential program is the least among all of them. This could be because the overhead time exceeds the time, reduced by parallel execution, in MPI and OpenMP programs. OpenMP program takes the largest time due to the significant overhead in the **fork-join**

process. Critical pragma also increases the time as no other thread could run in the critical section.

As we increase the number of processors in MPI after 8, the time taken by it also increases and surpasses that of sequential and OpenMP programs.
As we keep on decreasing the number of threads in OpenMp, time also reduces. **=> This shows that overhead associated with a fork & join is more on comparing with the time reduced due to parallel execution.**

Note1: Loops that run for a very small number of iterations (less than 5) are executed sequentially in openMP to avoid unnecessary fork-join overhead time.
Note2: In getStdDev function, Method 1(atomic pragma) is used for execution of standard deviation instead of using two for loops because it avoids the overhead associated with the extra fork-join process.

These time values may also be affected by OS scheduling and interrupts. So they are comparable when there is a significant difference between them.

Problem 1

For n = 10,000
Sequential: 0.001910 s **MPI:** 0.013081 s **OpenMP:** 0.255676 s

Sequential programs are the most efficient ones. The significant difference in time for sequential and OpenMp programs is due to the overhead associated with the fork-join process. Functional level parallelism is used in it to sort the two subarrays simultaneously. Note: We ignored doing the nested fork & join as it is not recommended.

MPI program is efficient than the OpenMp program, it may be because of the optimized in built functions provided by the MPI library to broadcast and gather the elements. It would have taken lesser time than sequential but an extra step is required in quick sort for MPI program to merge the sorted sub arrays(Merge the sorted subarrays received from different processors that take O(n*k) time). In sequential, this step is not needed as it is an in-place sorting algorithm that takes place in the same array.

As we keep on decreasing the number of threads in OpenMp, time also reduces. **=> This shows that overhead associated with a fork & join is more on comparing with the time reduced due to parallel execution.**