Project 2

1. Table of Contents

2. Team Members

1. Rohan Tahiliani
2. Abhishek Srivastava

3. Project Description

We have implemented barriers that allow programs to synchronize multiple nodes or threads at a program point. These barriers are helpful when a program requires all concurrent threads/processes in execution to reach a certain point before any individual thread/process can progress. This can be required for programs that share datasets that need to be updated by every thread/process before any thread/process can read from them.

The MPI barrier allows a program to synchronize multiple nodes running in a distributed environment by using message passing.

The OpenMP barrier allows a program to synchronize threads running on a single machine to synchronize by providing commands that allow critical section manipulation by only a single thread and sharing global data atomically.

4. Project Division

Rohan Tahiliani: Implemented MPI barriers using Tournament and MCS algorithm, wrote the test suite, combined the barriers, and measured the results for the OpenMP barriers, MPI barriers and OpenMP-MPI combined barriers, write-up.

Abhishek Srivastava: Implemented OpenMP barriers using Dissemination and MCS algorithm, combined the barriers, measured the results for the OpenMP barriers.

5. Algorithms

1. MPI Barriers

a. Tournament Barrier: The tournament barrier initializes each node by assigning it the appropriate opponent and role. The role decides the fate of the node in a specific round while the opponent specifies who the node will face.

   The barrier has been modified in a way to allow messages to be exchanged instead of spinning on variables that need to be modified in the opponent's memory. The Rsend and Receive methods allow nodes to exchange messages synchronously, and a node cannot proceed after sending till the opponent receives the message. This takes care of race conditions. Each round, depending on its role, a node will WIN, LOSE, or be the CHAMPION. If the rounds get over, or the node has completed its role for the barrier, then the nodes start dropping out of using either BYE or DROPOUT.

b. MCS Barrier: The MCS barrier initializes each node by assigning child and parent pointers. As in the MCS algorithm there are 4 children in the arrival tree and 2 children in the wakeup tree.

The algorithm begins with each node waiting for its children (if it has any) to reach the barrier, after which the node notifies its parent (if it has one) that it has reached the barrier. Once the root node gets notified by its children it begins the wakeup process. It notifies (two of) its children (if it has any) that all nodes have arrived and the node can proceed to wakeup children or continue with program execution.

2. OpenMP Barriers

(Abhishek)

a. Dissemination Barrier: The threads wait for their specific partner that is calculated each round. The thread updates its own sense and starts waiting for its partner's sense. This partner updates its sense after which the thread can proceed to the next round. Once all the rounds are achieved the nodes have achieved the barrier. The pub variable keeps a count of how many threads have achieved the barrier. After all threads have been achieved the pub variable has the same value as displayed by each node. This also shows that no thread completes the barrier before other threads have reached.

b. MCS Barrier: The threads all have a global variable initialized to figure the role of each node in every round. The threads start with 4 children in the arrival process and 2 children in the wakeup process. The threads wait for their children to complete and then notify their own parent if they have one. The root node begins the wakeup process by waking up its 2 children who proceed to wake their children and so on. Once all threads have completed the barrier the pub variable that is shared by all has the same count which indicates that no thread crosses the barrier before other threads have reached.

3. MPI-OpenMP Combined Barrier:

This barrier was developed using the MCS algorithm implemented for MPI and OpenMP. The MCS algorithm has been implemented exactly the same way as shown in part D.1.b and D.2.b.

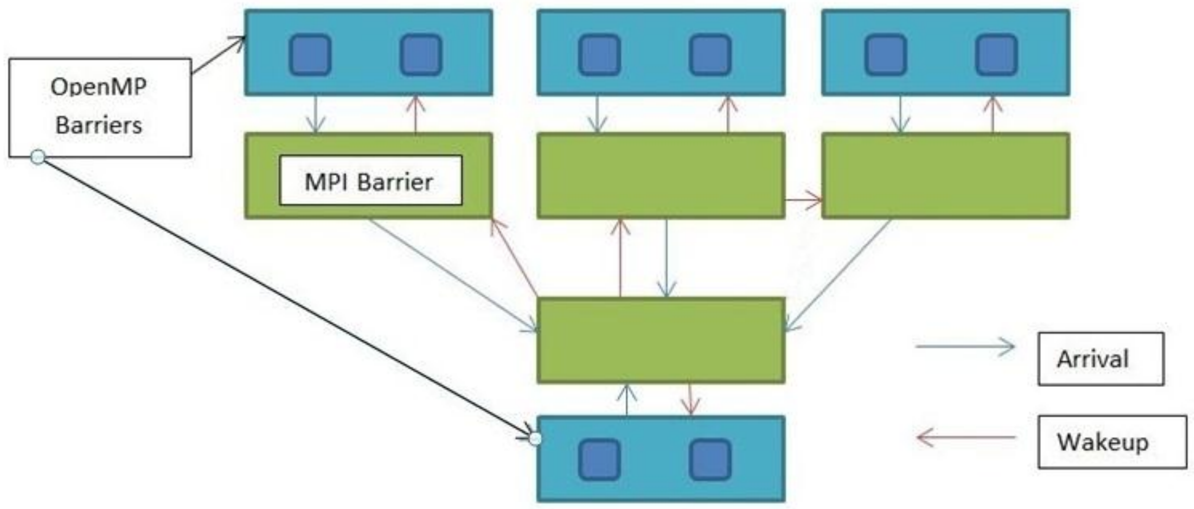

Figure 5-1

The combined barrier has been implemented in a way that the nodes participating in the MPI barrier have got threads running on them that implement the OpenMP barrier and once that has been achieved on a node, the node proceeds to complete its role in the MPI barrier. The sequential flow of the combined barrier can be seen in the diagram shown in Figure 1.

The Blue Nodes are the nodes that are in the OpenMP barrier. Once they complete they will notify that node that the barrier has been achieved and the Green nodes come into play. The reason for the single node to be below three nodes is because of the behavior of the MCS algorithm, it depicts the parent child relationship during the arrival and wakeup.

6. Experiments

To measure the performance of the barriers, the test suite measures the time required for each process to achieve a barrier as well as the total time required for all the processes.

To get an accurate measurement of time required for the processes in MPI, we implemented 8 barriers so that we get a fair estimate of how much time is required for a single barrier, which also accounts for variances which might occur during execution.

In the OpenMP barriers, we took the measurement for single threads to achieve a barrier and also ran the entire program 20 times for each set of results so that we get accurate results, and averaged out the results.

We use the 'gettimeofday' Linux system call for measurements, which measures time in microseconds. The calls are made before the initialization and after the barriers are achieved, to measure the total time. To measure individual barriers, we make a call before the barrier is entered and after the thread/process achieves the barrier.

7. Results and Analysis

Figure 7-1 shows the comparison of the MPI algorithms for 2, 4 16 and 32 processes. The green line represents the MCS barrier, while the red line represents the Tournament barrier. We can see that initially the tournament and MCS barriers perform relatively equally, but, as the number of processes increases the time required for the MCS barrier starts performing poorly. The reason is that, the MCS algorithm relies heavily on modifying memory on other processors whereas the algorithm for MPI has been implemented using message passing. Since the nodes in the tournament barrier have to wait for lesser messages in the arrival process and so they tend to perform better than the MCS algorithm. Also as the number of nodes waiting for 4 children to complete increases in the MCS barrier, the performance starts degrading more.

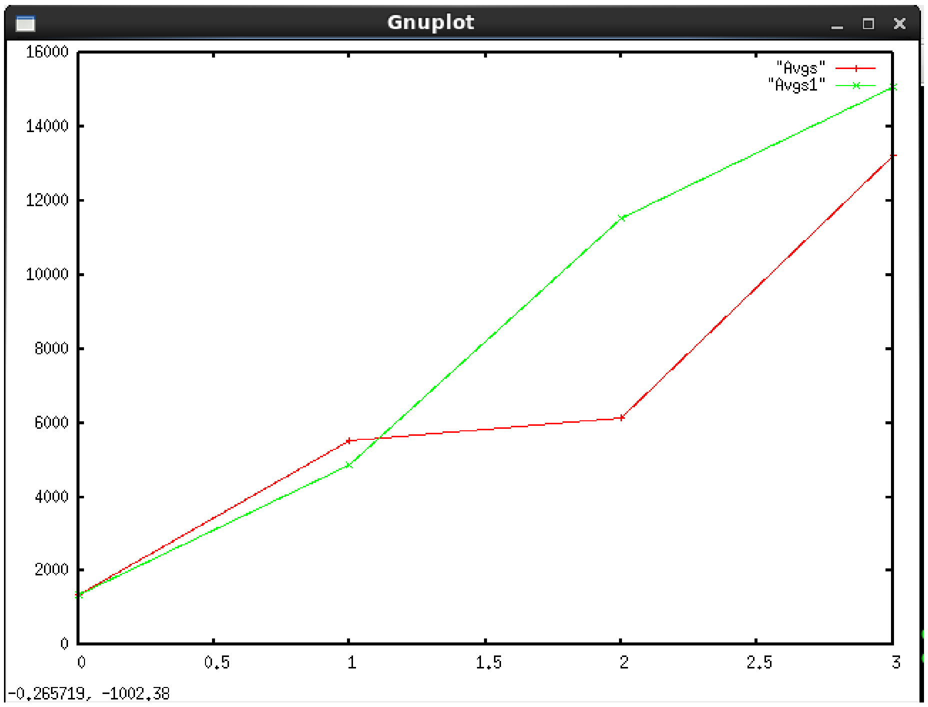

Figure 7-1

(Abhishek) Figure 7-2 shows the comparison of the Dissemination barrier and MCS barrier for OpenMP for 2, 4 and 8 threads. The red line shows the performance for the Dissemination barrier and the green line shows the performance of the MCS barrier. The Dissemination barrier shows considerably better performance. Both barriers experience degraded performance as the number of threads increases.

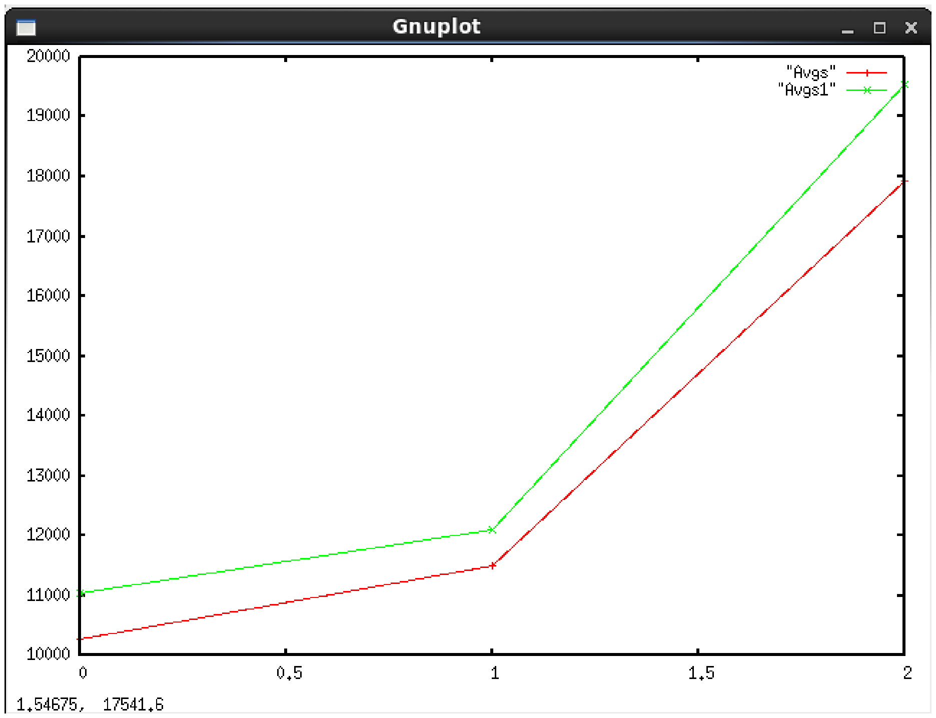

Figure 7-2

Figure 7-3 shows the performance of the combined OpenMP-MPI barrier which implements the MCS algorithm for both parts. The performance drops exponentially as the number of processes increases. Seeing the results from the OpenMP and MPI barriers we can see that the effect worsens for the combined barrier when it goes from 16 to 32 nodes as compared to the MPI barrier. This could be a result of the OpenMP barrier slowing down nodes which gets amplified as the number of nodes increases, as the nodes need to wait for the OpenMP barrier to complete and more parents need to wait for their children to complete, and also more nodes have to wait for 4 children to complete as compared to the 2 and 8 process runs.

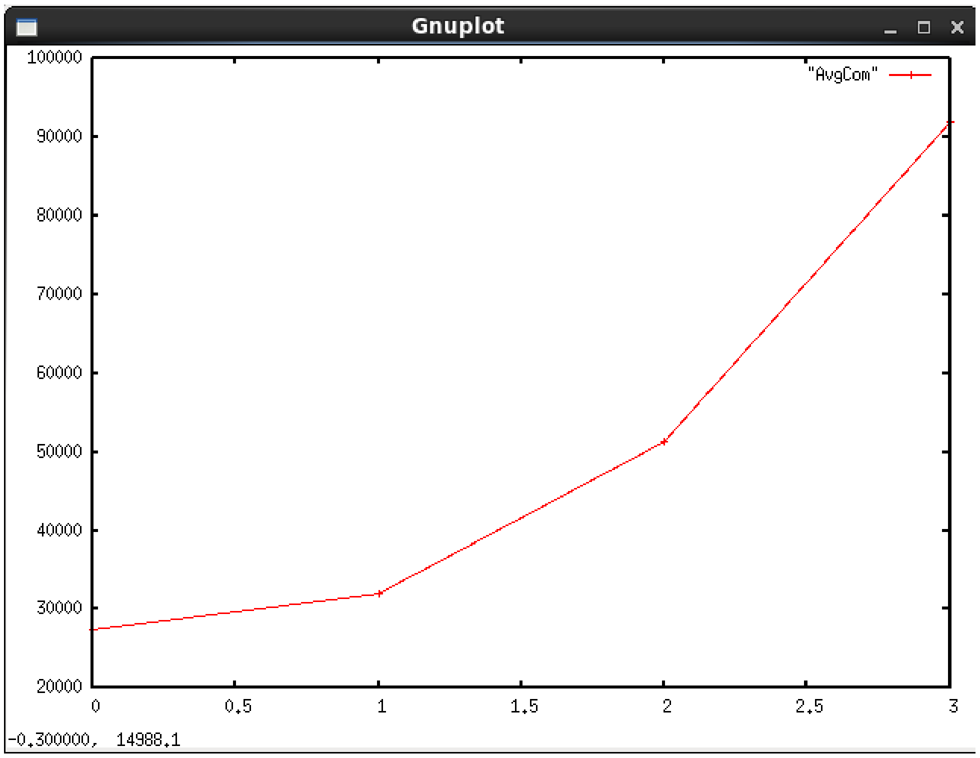

Figure 7-3

8. Conclusion

The MCS barriers seem to have slower performance than the tournament and dissemination barriers because of each node having to wait on 4 children whereas the tournament and dissemination barriers only have waiting on 2 nodes. This seems to be the root cause of the performance degradation in the MCS algorithm. When message passing is used for barriers, it is better to use the tournament barrier.