CITS3402 High Performance Computing Project 2

Parallel implementation of search based on MPI and OpenMP

Sijing Aloysius Loh 24028283

Thai Hoang Long Nguyen 23147438

Introduction

In this project, we integrated Message Passing Interface (MPI) with our existing Open Multi-Processing (OpenMP) parallel implementation of the Fish School Behavior (FSB) program. This allowed us to explore the intriguing realm of parallel and distributed computing, making substantial strides towards optimising execution times and achieving more efficient resource utilisation. Our primary objective was to investigate the integration of MPI into our parallel implementation of FSB and subsequently assess its impact on execution times.

This project has 2 deliverables. The first is a basic MPI communication program for FSB. The program generates the fish as it would for FSB but no collective action is performed. Instead, the x and y coordinates of each fish and its weight are written onto a file. Then, the master node sends an even number of fish to each node and immediately receives it back from them. This is done using MPI_Scatter and MPI_Gather which make use of collective operations. These collective operations are highly optimised in MPI and result in better performance than individual send and receive constructs. Data of each fish is sent by creating a MPI_Datatype using create_mpi_struct(). Once the master receives data back from each worker, it writes the data onto another file. To ensure that the communication was successful, the 2 files are compared. If they are identical, then communication was successful.

The second deliverable is the actual implementation of FSB with both MPI and OpenMP. It is a combination of the parallel program done in project 1, and the first deliverable of this project. Similar to the first project, we executed this program with different variables and recorded their execution times.

We conducted 2 experiments to comprehensively evaluate the performance of the new MPI-OpenMP program and to provide insights into how these two parallel computing paradigms could work in synergy. All experiments and measurements are performed

and tested using the Setonix Supercomputer to achieve consistent measurements. For both experiments, we ran the program with 1,000,000 fish and 3,000 steps.

Experiments and Analysis

Experiment 1:

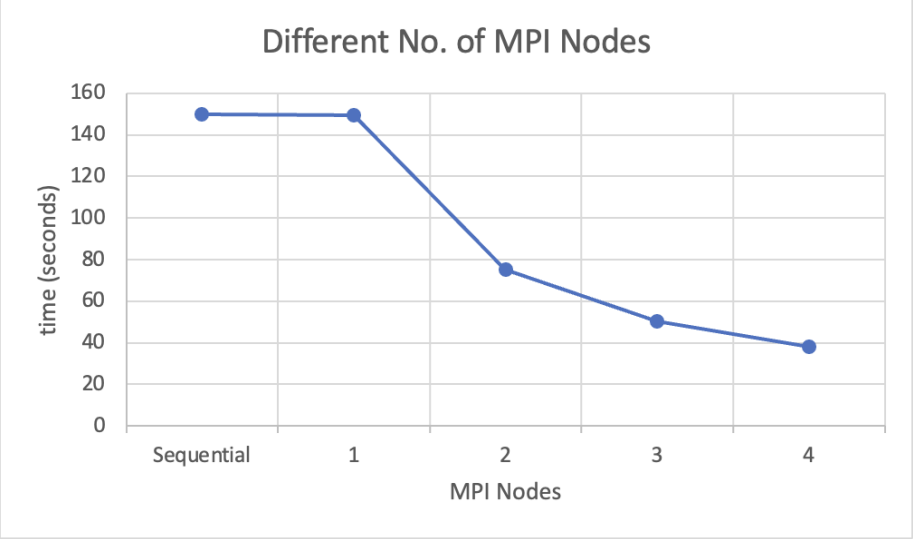
For the first experiment, we executed the program with a varying number of MPI nodes while allocating 1 thread to each node for all executions. The aim is to see the effect that MPI has on the execution time of the FSB program independently of OpenMP.

The table and graph below shows the execution time of the program with 1, 2, 3, and 4 MPI nodes. As a control, the execution time for the sequential implementation of FSB from project 1 was included. Firstly, it is observed that the execution times for the sequential program is similar to that of the program with 1 MPI node. This is not surprising as there is no other node for the main node to split the work up with.

Next, we noticed that the execution time for the program decreases as the number of MPI nodes increases. Notably, there is almost a 50% decrease in the average execution time when going from 1 node to 2 nodes. This is expected since the work performed on fishes is shared among other nodes where each node is assigned equal numbers of fish. Since each node now has less work to do, the total execution time decreases. We can also see Amdahl's Law in effect where subsequent increments in MPI nodes lead to a smaller decrease in execution times (33% and 24% for 2 to 3 and 3 to 4 nodes respectively). This suggests that we might be approaching a point of diminishing returns if we continue to increase the number of nodes. When there are more nodes, the part of the program that cannot be parallelised becomes a bigger fraction of the total execution time, leading to diminishing returns.

Table 1 and graph 1

MPI Nodes	t1	t2	t3	tavg
Sequential	149.98868	149.11089	150.38073	149.82677
1	149.13331	149.56225	149.85662	149.5174
2	74.882034	75.001434	75.164763	75.016077
3	50.051971	50.390138	50.62603	50.356046
4	38.062993	37.895301	38.801091	38.253128



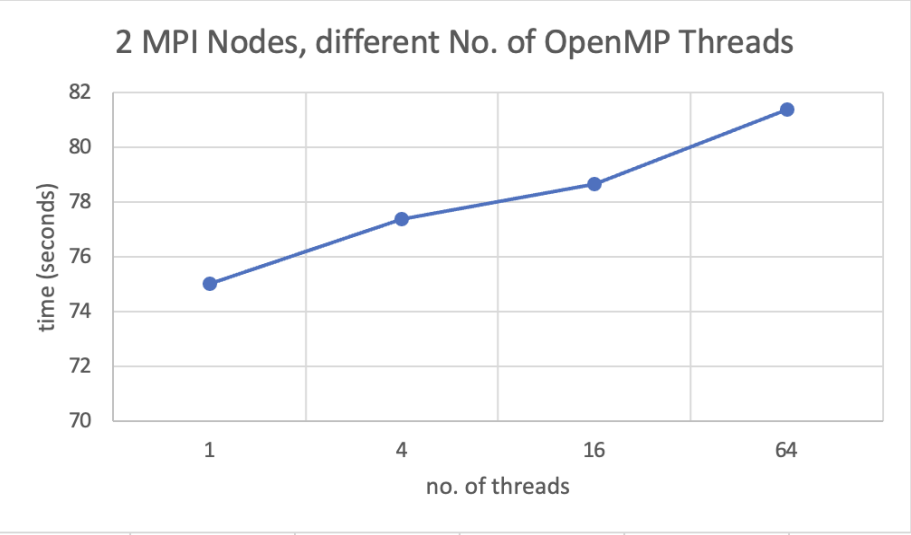
Experiment 2:
In the second experiment, we introduced OpenMP into the program to see the effects of having different numbers of threads under each MPI node. For each number of nodes (2, 3, 4), we ran the program with 1, 4, 16, and 64 threads under each node and measured the execution times.

The results can be seen in the following tables and graphs. Firstly, it is observed that increasing the number of MPI nodes leads to a faster execution time. This observation aligns with the results of the first experiment. Additionally, for each number of MPI nodes, we found that increasing the number of threads under each node increases the execution time. This is similar to the results observed in project 1 where we saw that the execution time increased when the number of threads increased. We speculated that it was due to overheads in parallelisation from thread creation, synchronisation and management. This is also a likely explanation for the slower execution times here.

It is also noted that the increased execution times is greater when there are more MPI nodes. In the case of 2 nodes, the difference in execution times between 1 thread and 64 threads is an 8% increase. This goes up to 12% and 13% for 3 and 4 nodes respectively.

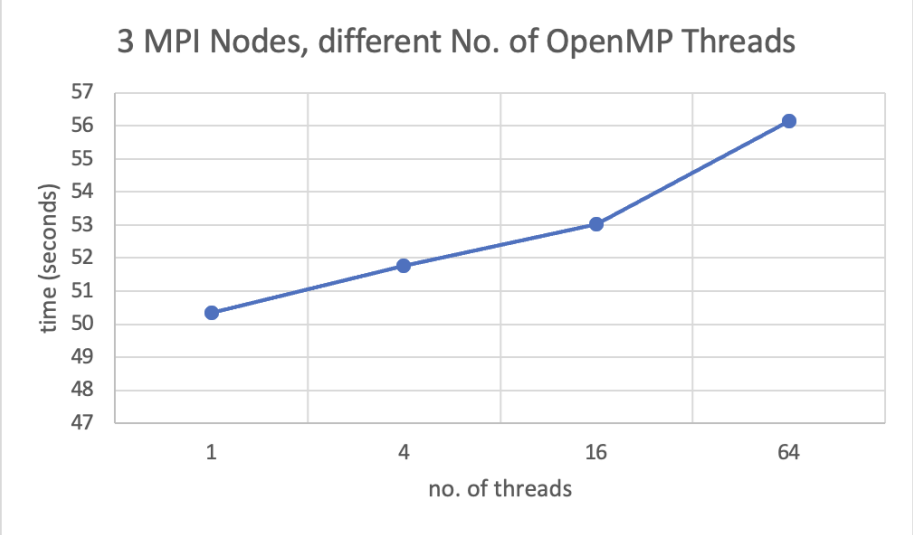
2 nodes:

Threads	t1	t2	t3	tavg
1	74.882034	75.001434	75.164763	75.016077
4	77.838011	76.993845	77.347762	77.393206
16	78.759711	78.219836	79.001837	78.660461
64	81.879134	80.927187	81.339272	81.381864



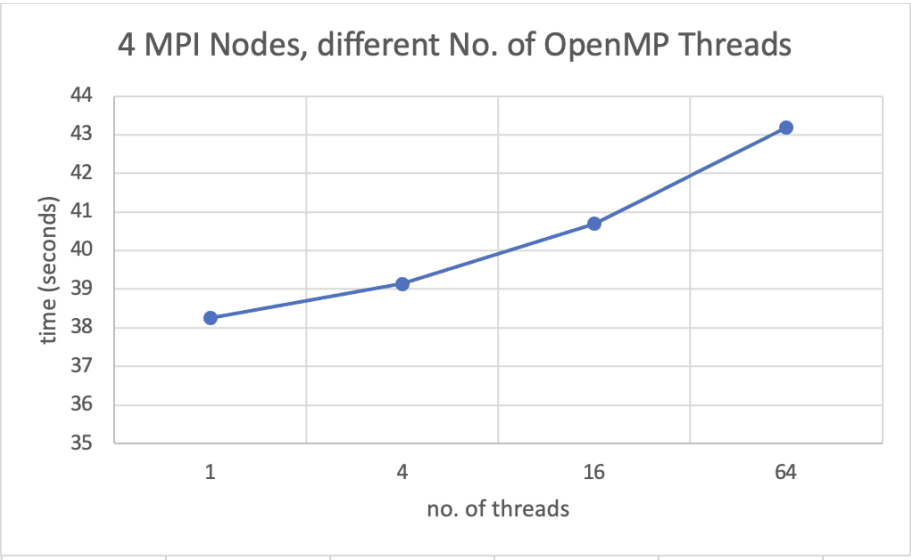
3 nodes:

Threads	t1	t2	t3	tavg
1	50.051971	50.390138	50.62603	50.356046
4	52.218782	51.69282	51.402873	51.771492
16	52.972527	52.937262	53.187462	53.032417
64	56.158761	56.402826	55.872648	56.144745



4 nodes:

Threads	t1	t2	t3	tavg
1	38.062993	37.895301	38.801091	38.253128
4	39.22327	38.918371	39.28239	39.141344
16	40.483864	41.293739	40.283048	40.686884
64	43.262592	43.283738	42.983749	43.176693



Conclusion:

In this project, we successfully integrated MPI with the existing OpenMP parallel implementation of the FSB program. Our experiments demonstrated that increasing the number of MPI nodes led to significant reductions in execution time, validating the benefits of parallelisation. It is worth noting that the speedup will eventually flatten out at some point. Introducing OpenMP threads added an additional layer of parallelisation, but it came with overheads that increased the execution times. Achieving optimal performance requires a careful balance between MPI nodes and OpenMP threads, taking into account the workload and system characteristics. These findings provide valuable insights for further optimising the MPI-OpenMP implementation on the Setonix Supercomputer and highlight the complexity of parallel and distributed computing.