PDC - Lab FAT

Aadhitya Swarnesh I

[redacted]

05 - November - 2020

Question 1

Consider in a Virtual Chess fest, 4 teams are participating. There are four processor nodes in a cluster out of which one is the master. Each node has a specialized chess algorithm implemented. Each player will play in all the processor nodes against their respective algorithm. Their points at the end of each play is sent to the master. The master finds the final sum of points of each player and adjudges the final winner. Let the points scored by each player in a particular processor node is generated locally by the processor node and has to be communicated to the master node. Implement the above scenario using MPI application with 4 worker nodes in the MPI cluster and a master node.

Code :

```
#include<mpi.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>


int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int rank;
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int powerOfPlayers[world_size-1];
    for (int i = 0; i < world_size-1; i++) {
        powerOfPlayers[i] = (int)(rand);
    }

    if (rank == 0) {
        // Master thread recieves the score.

        // We print the scores of the players.
        printf("The powers of the players are : \t");
        for (int i = 0; i < world_size-1; i++) {
            printf("%d\t", powerOfPlayers[i]);
        }
        printf("\n");

        // Initialize the score board;
        int score_board[world_size-1];
        for (int i=0; i<world_size-1; i++) {
            score_board[i] = 0;
        }

        int total_matches = (world_size-1) * (world_size
- 2) / 2;
        int match_winner;
        for (int i=0; i < total_matches; i ++) {
            // Recieving the winner of match 'i'
            MPI_Recv(&match_winner, 1, MPI_INT,
MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

            // Updating the score board.
            score_board[match_winner-1] ++;
        }
```

```
        // To find the winner of the tournament.
        int winner = -1;
        for (int i=0; i < world_size-1; i ++) {
            if (winner == -1) {
                winner = 0;
            }
            else {
                if (score_board[winner] < score_board[i])
{
                    winner = i;
                }
            }
        }

        // To print the winner of the tournament.
        printf("The Master thread reporting -- The winner
of the tournament is : %d with %d wins.\n", (winner + 1),
score_board[winner]);
    }
    else {
        // Each processor will play with the processors
after it.
        for (int i = rank + 1; i < world_size; i ++) {
            // Now the match is between 'rank' and
'rank+i' processors.

            int winner;
            if (powerOfPlayers[rank-1] >
powerOfPlayers[rank + i - 1]) {
                printf("%d Process Reporting : In match
between %d and %d, %d has won the match.\n", rank, rank,
(rank+i), rank);
                winner = rank;
```

```
                // Sending the winner's details to the
master.
                MPI_Send(&winner, 1, MPI_INT, 0, 0,
MPI_COMM_WORLD);
            }
            else {
                printf("%d Process Reporting : In match
between %d and %d, %d has won the match.\n", rank, rank,
(rank+i), (rank+i));
                winner = rank + i;
                // Sending the winner's details to the
master.
                MPI_Send(&winner, 1, MPI_INT, 0, 0,
MPI_COMM_WORLD);
            }
        }
    }

    MPI_Finalize();
}
```

Output :

```
(base) Aadhityas-MacBook-Air:fat aadhitya$ mpicc fat.c
(base) Aadhityas-MacBook-Air:fat aadhitya$ mpirun -np 5 --oversubscribe a.out
The powers of the players are :         1830978003      1830978003      1830978003      1830978003
1 Process Reporting : In match between 1 and 3, 3 has won the match.
2 Process Reporting : In match between 2 and 5, 2 has won the match.
1 Process Reporting : In match between 1 and 4, 4 has won the match.
1 Process Reporting : In match between 1 and 5, 1 has won the match.
2 Process Reporting : In match between 2 and 6, 2 has won the match.
3 Process Reporting : In match between 3 and 7, 3 has won the match.
The Master thread reporting -- The winner of the tournament is : 2 with 2 wins.
```

Algorithm :

The game of chess has been implemented using MPI programming model. We will explain in detail how the entire process of the code goes on.

Starting on, we run the program with a total of 5 processes, and so there are 4 worker processes and a master process. The master processes is the one who is organising this competition, and the worker processes are the competitors in the game. Initially, we randomly generate the power for each of the players in the game. The master process then displays the power of each player in the game.

In each of the game the player **'i'** would play against the players **'I+1 … n',** where **'n'** is the total number of the players. This is to ensure that each of the players only play a single match. In the match between say player 'a' and player 'b', then the winner of the match is relayed back to the master process. The process sends the rank of the winning process back to the master process. The winner of each match is decided by the power of the players which was initialised in the earlier stages.

The master process receives a total of **(world_size-1) * (world_size - 2) / 2** wins or the messages from the worker processes. Once this is done the master process proceeds to calculate the winner. Every time the master process receives a rank from the worker nodes, it increments that node's score in the score board that it maintains. After all the matches, the master finds the player with maximum score and prints that value in the console.

Once all this is done, all the processors reach the end and thus we end the program after finalising or ending the MPI constructs.