CS3210 Assignment 2
MPI Basketball

Chong Yun Long
A0072292H

1 Basketball Training Session

1.1 Pseudocode

Code Listing 1 Pseudocode for Training

```
DO
    Field broacast coordinates to all players
    Players run towards ball
    Field gathers players coordinates
    Decides winner and informs all players
    Winner informs field of ball using a scatter operation.
    Field prints round info
FOR 900 times
```

1.2 MPI Communication

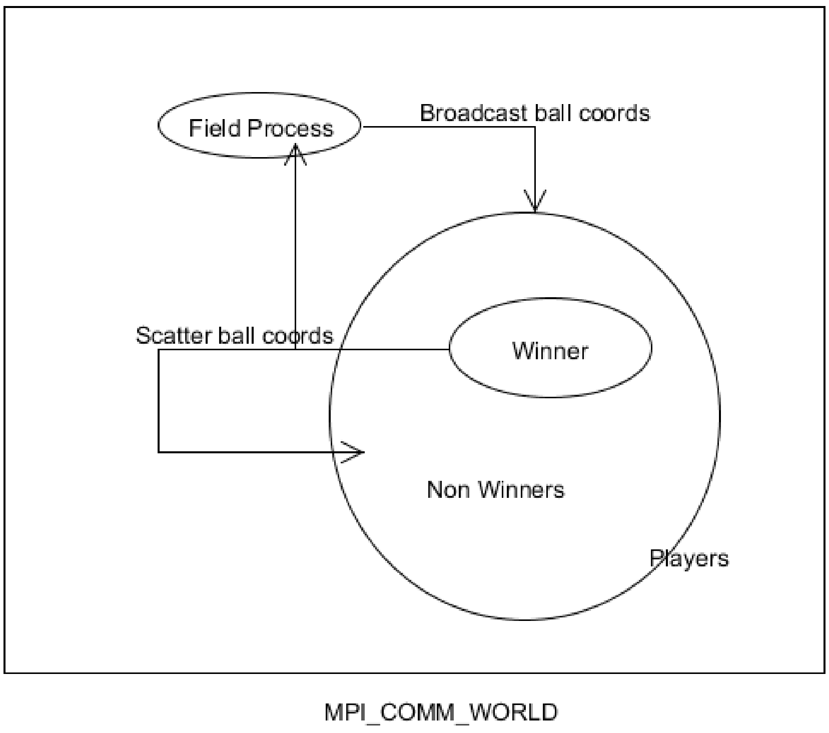

Figure 1: Communication diagram for Training

Figure 1 summarises the communication involved in training. There are only 2 types of communication:

1. Field process $\overrightarrow{broadcast}$ All players
2. Winner $\overrightarrow{scatter}$ Non-winners, Field process

1.3 Important Design Decisions

1.3.1 Communication

For training, since efficiency is not of consideration, I have placed all the parameters required of players into struct called player. This struc will be used all communications throughout the match. To achieve this, we make use of *MPI_Type_struct* function in the MPI specification which allows us to define a custom struct to be used for message passing.

There is the additional requirement which states: **The player is not allowed to guess or to know beforehand the position of the ball or of another player.** This poses a problem when winner has to inform the field process of the new ball location. To do this, the winner performs a scatter operation. Coordinates (-1,-1) is scattered to other players while the actual coordinates is scattered to the field and the winner process.

1.3.2 Output

The ball coordinates printed is after a ball throw. The total distance ran by the player, total number of times he reached the ball, total number of times the ball he won the ball are all inclusive of the current round.

1.4 Program compilation and execution

To compile the program please make use of the make file provided.
Alternatively, compile with the following command:

```
mpicc Training.c -o Training
```

To run:

```
mpirun -np 6 ./Training
```

Machinefile and Rankfile are not needed for program execution.

2 Basketball Match

2.1 Issues with program execution

This program does not execute correctly on the lab machines running on OpenMPI 1.4 with 12 processes enabled. However, the program is able to execute smoothly without any errors or deadlock under the following configurations/MPI distributions:

1. Number of processes was reduced to 8 (2 field, 4 players on each team) on Intel(R) Core(TM) i7-2600 lab machine
2. Tembusu cluster (path: /opt/openmpi/bin/mpicc)
3. OpenMPI 1.5 (Tested with Intel Core 2 Duo CPU E8400 @ 3.00GHz)
4. MPICH2 (Tested with Intel Core 2 Duo CPU E8400 @ 3.00GHz)
5. Visual Studios HPC Pack 2008 R2 SDK with SP4 (Tested with Intel Core 2 Duo CPU E8400 @ 3.00GHz)

A possible explanation for the deadlock is that OpenMPI 1.4 does not guarantee thread safety when there is over-subscription. Hence it works smoothly on Tembusu cluster since there are 16 cores and when number of process is reduced to 8 since the Intel(R) Core(TM) i7-2600 has 8 cores.

2.2 Pseudocode

Code Listing 2 Pseudocode for Match

```
DO
    DO
        Field 0 send ball coordinates to all players
        Players run towards ball
        Players send their own information to their own respective fields
        Field decides winner and informs the players that they are in charge
        Winners field send information of players it is in-charge of to winner
        Winner will either pass to his own teammate or try to score
        Winner inform his field whether the ball has missed the target
        IF ball has missed
            Field randomly chooses a location within 8 feet of target location
        Field 1 sends information about its players to Field 0
        Field 0 prints round information
        FOR 2700 times
    Switch sides
FOR 2 times
```

2.3 MPI Communication

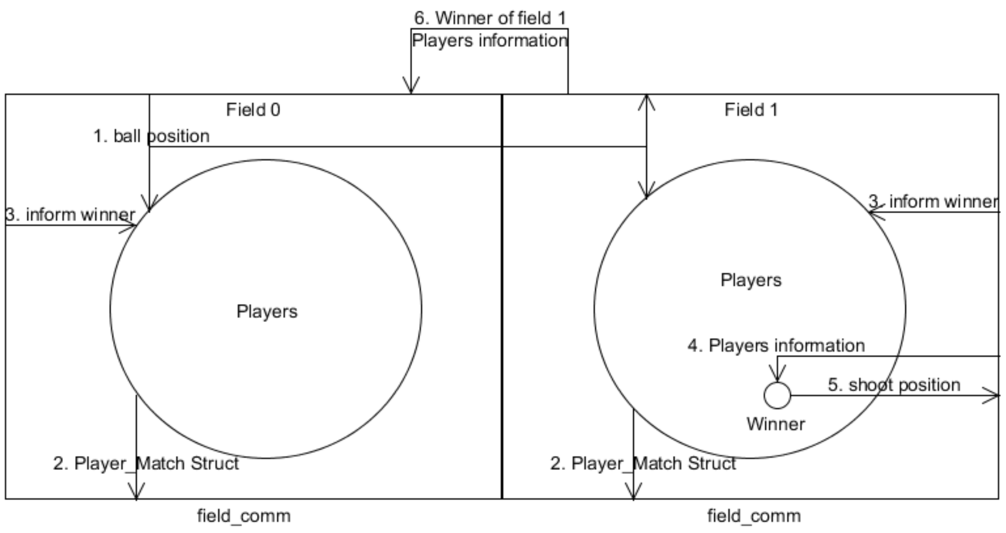

Figure 2: Communication diagram for Match

There are in total 6 types of communication in Match:

1. Field 0 → Field 1, Players
2. Players → respective Field
3. Field → respective Players
4. Field with Winner → Winner
5. Winner → respective Field
6. Field 0 → Field 1

The types if communication are numbered in chronological sequence.

2.3.1 Informing players of winner

For this step: *Field decides winner and informs the players that they are in charge.*
The field only informs their own players. Hence in this situation, all players of Field 0 will receive a winner of -1 since the ball is not in their field. Only players of Field 1 will be informed of the winner.

2.4 Important Design Decisions

For match, there are two data structures each process will have 2 struct: *Player_Match* and *Player_Stats*. *Player_Match* will be used for all the data that is used for communication while *Player_Stats* contains all the data which are hardly changes throughout the match.

Code Listing 3 Struct used in Match

```
typedef struct {
    int global_rank;
    int player_id;
    int initial_coords[2] ;
    int final_coords[2] ;
    int reached ;
    int challenge ;
    int ball_coords[2] ;
    int team;

} Player_Match;

typedef struct {
    int global_rank;
    int court_rank;
    int speed;
    int dribble;
    int shooting;
    int post[2];
    MPI_Comm team_comm;
    MPI_Comm court_comm;
    int court_group_size;


} Player_Stats;
```

2.4.1 Passing Strategy

The passing strategy used in the program is simple. It follows the algorithm described below:

Code Listing 4 Struct used in Match

```
max_chance = chance of shooting to post
target_coords = post
IF my teammate is closer to the post
    IF max_chance < chance of passing to teammate AND
    chance of scoring < PASS_FACTOR * chance of passing to teammate
        target_coords = teamate's position
```

If passing to teammate on the same field is easier than score, the player will pass to teammate. The PASS_FACTOR determines the strategy of the team. If PASS_FACTOR is set to a large value, players are more defensive and will attempt to pass the ball if a teammate is nearby. If PASS_FACTOR is set to a small value, players are more aggressive and will try to score even if a teammate is nearby. For assignment 2, PASS_FACTOR of 1 is used. If the chance of scoring is lesser than chance of passing, a player will pass the ball to teammate.

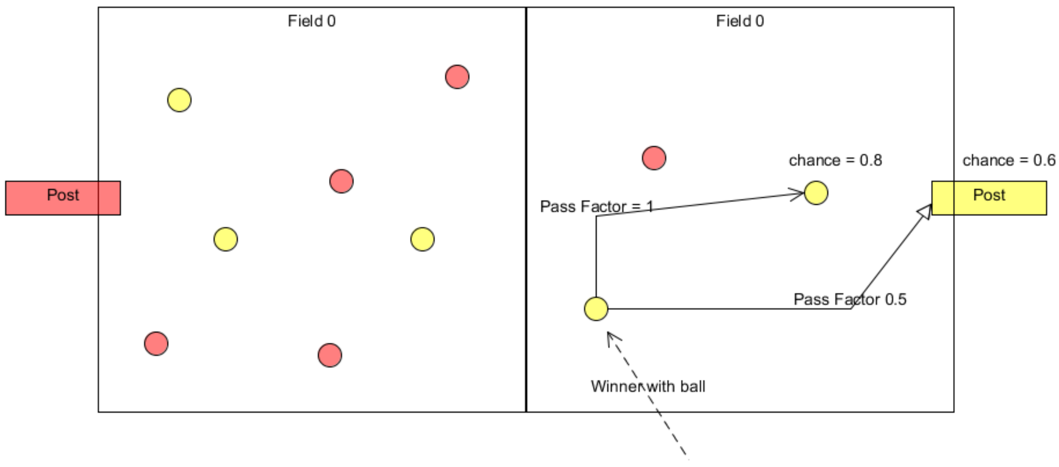

Figure 3: Ball passing strategy

Consider Figure 3, if the PASS_FACTOR is 1, the ball will be pass to team mate since $0.6 < 1 * 0.8$. However, if the PASS_FACTOR is 0.5, the winner will attempt to score with the ball since $0.6 > 0.5 * 0.8$.

Note that players will only pass to teammate belonging to the same field.

2.4.2 Player's Stats

$$\begin{aligned} dribble &= rand()\%5 + 1 \\ speed &= 5 + rand()\%5 \\ shooting &= 15 - dribble - speed \end{aligned}$$

A greater emphasis was given to the player's speed, followed by dribbling and shooting. Speed is given high priority because of the size of the court. Dribbling is given a low priority because there are only a few occasions where players compete for the same ball .

2.4.3 Output

The program prints output in the following format:
round no. + newline
score + newline
ball coordinates after throw + newline
player id, initial coordinates, final coordinates, whether they reached the ball, whether they kicked the ball , ball challenge, shoot target location

The ball coordinates printed is after a ball throw. Consider the example below where a player successfully scored.

Code Listing 5 Output after scoring

```
31
2 2
128 39
0 64 39 71 39 0 0 -1 -1 -1
1 72 39 81 39 0 0 -1 -1 -1
2 64 39 71 39 0 0 -1 -1 -1
3 106 39 114 39 0 0 -1 -1 -1
4 67 39 75 39 0 0 -1 -1 -1
0 105 39 111 39 0 0 -1 -1 -1
1 88 39 93 39 0 0 -1 -1 -1
2 72 39 81 39 0 0 -1 -1 -1
3 88 39 93 39 0 0 -1 -1 -1
4 114 39 121 39 0 0 -1 -1 -1
32
2 4
128 32
0 71 39 78 39 0 0 -1 -1 -1
1 81 39 90 39 0 0 -1 -1 -1
2 71 39 78 39 0 0 -1 -1 -1
3 114 39 122 39 0 0 -1 -1 -1
4 75 39 83 39 0 0 -1 -1 -1
0 111 39 117 39 0 0 -1 -1 -1
1 93 39 98 39 0 0 -1 -1 -1
2 81 39 90 39 0 0 -1 -1 -1
3 93 39 98 39 0 0 -1 -1 -1
4 121 39 128 39 1 1 3 128 32
33
2 4
64 32
0 78 39 74 36 0 0 -1 -1 -1
1 90 39 83 37 0 0 -1 -1 -1
2 78 39 74 36 0 0 -1 -1 -1
3 122 39 115 38 0 0 -1 -1 -1
4 83 39 78 36 0 0 -1 -1 -1
0 117 39 112 38 0 0 -1 -1 -1
1 98 39 94 38 0 0 -1 -1 -1
2 90 39 83 37 0 0 -1 -1 -1
3 98 39 94 38 0 0 -1 -1 -1
4 128 39 122 38 0 0 -1 -1 -1
```

A player scored on round 32. The ball coordinates shown in round 32 is (128,32), at the basketball post before the game restarts. At round 33 the coordinates of the ball resets to the middle of the court.

The round number does not reset itself after the first half. The game ends when the round number reaches 5400.

2.5 Average Time per Round

On Intel(R) Core(TM) i7-2600

It took 3.34201 seconds to complete 5400 rounds (linux time).
It took 0.61889 milliseconds to complete 1 rounds (linux time).

On Tembusu

It took 0.58675 seconds to complete 5400 rounds (linux time).
It took 0.10866 milliseconds to complete 1 rounds (linux time).

2.6 Program compilation and execution

To compile the program please make use of the make file provided.
Alternatively, compile with the following command:

```
mpicc Match.c -o Match -lrt -lm
```

To run:

```
mpirun -np 12 ./Match
```

Machinefile and Rankfile are not needed for program execution.