Parallel Baseball

Christian Winwood
christianwinwood@gmail.com
Shawn Smith
shawnsmith88@protonmail.com

March 27, 2021

1 Project description

The purpose of this program is to simulate a baseball game. The game consists of 9 innings with two teams. Each team will get half an inning to bat and attempt to increase their score while the opposing team will try to prevent them from scoring. At the end of the game, the team with the most points will be known as the winner. The application has multiple processes, each representing a different player on the field. The pitcher process is the root process that keeps track of controlling the game and keeping track of the different scores. The messages being passed represent the ball. The pitcher sends a message to the batter - this represents the pitch. The batter then randomly decides whether to hit it and passes a message to a fielder. The fielder then messages the pitcher of whether they caught the ball or made an error. The pitcher then determines team scoring, out counting, team switching, and inning increment and continues with a new pitch.

2 Compile and Run

The compile and run commands are the same they have always been. First you use

mpic++ baseball.cpp

in order to compile the program. This creates an a.out files that you need to run. To run it you use the command

mpirun –oversubscribe -np 9 a.out

and the program will run.
Note: You must use 9 processes.

3 Overview of Effort

We decided to put in a lot of effort in a short amount of time to complete this project. We sat down and dedicated an entire weekend to solely working on this project. After doing some very basic overview design and setting up some helper functions, we dove in. We were able to complete our goal and finished the project within our Hackathon time period. It was a team effort that we are proud of.

4 Description of Tests

This program has a lot of variables, such as the difficulty of getting a hit, the difficulty of making a play, and the probability of getting multiple bases. By changing those variables, the scores of the games tend to be impacted, but neither team gains an advantage from these changes. lower hit difficulty, higher play making difficulty and higher multi-base probability tend to contribute towards higher scoring games, whereas higher hit difficulty, lower play making difficulty and lower multi-base probabilities tend to lower the scores in games.

5 Concluding Remarks

All in all, this was a fun and challenging project. We had to utilize many things we have learned in this class so far and the results are an interesting baseball game to read through.

6 Code

```
#include <iostream>
#include <mpi.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>

#define MCW MPI_COMM_WORLD

// Reset bases after inning
// limit to 9 innings

enum Result {
    MISS,
    SINGLE,
    DOUBLE,
    OUT
};

enum Tag {
    PITCH,
    HIT,
    RESULT,
    GAMEOVER
};

enum Position{
    PITCHER,
    BATTER,
    FIELDER
};

enum Pitch {
    FASTBALL,
```

```
34      CURVEBALL,
        CHANGEUP
36  };

38  enum Bases {
        FIRST,
40      SECOND,
        THIRD,
42      HOME
    };
44
    std::string pitchToString(Pitch p) {
46      switch (p) {
            case Pitch::FASTBALL: return "Fastball";
48          case Pitch::CHANGEUP: return "Changeup";
            case Pitch::CURVEBALL: return "Curveball";
50          default: return "AHHH IDK!!!!";
        }
52  }

54  int getPitchDifficulty(Pitch pitch) {
        switch (pitch) {
56          case Pitch::FASTBALL: return 1;
            case Pitch::CHANGEUP: return 2;
58          case Pitch::CURVEBALL: return 3;
            default: return 1;
60      }
    }
62
    std::string rankToPosition(int rank) {
64      switch (rank) {
            case 2: return "1B";
66          case 3: return "2B";
            case 4: return "SS";
68          case 5: return "3B";
            case 6: return "LF";
70          case 7: return "CF";
            case 8: return "RF";
72          default: return "Coach";
        }
74  }

76  std::string resultToString(int result) {
        switch (result) {
78          case Result::DOUBLE: return "DOUBLE";
            case Result::SINGLE: return "SINGLE";
80          default: return "doesn't matter";
        }
82  }

84  void runPitcher() {
        int data;
86      MPI_Status status;
        int awayScore = 0;
88      int homeScore = 0;
        int inning = 1;
90      bool topOfInning = true;
        int strikes = 0;
92      int outs = 0;
        bool running = true;
94      std::array<int, 3> runners;

96      for(int i = 0; i < runners.size(); ++i) {
            runners[i] = 0;
98      }

100     while(running) {
            // Throw pitch
102         int pitch = rand() % 3;
            std::cout << "PITCHER: pitched a " << pitchToString(Pitch(pitch)) << std::
        endl;
104         MPI_Send(&pitch, 1, MPI_INT, 1, Tag::PITCH, MCW);

106         MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, Tag::RESULT, MCW, &status);

108         if(data != Result::MISS) {
                std::cout << "Ball hit to " << rankToPosition(status.MPI_SOURCE) << std
        ::endl;
110         }

112         if(data == Result::OUT) {
                std::cout << "OUT!!" << std::endl;
114             if(++outs == 3) {
                    std::cout << "End of inning" << std::endl;
116                 for(int i = 0; i < runners.size(); ++i) {
                        runners[i] = 0;
118                 }
                    outs = 0;
120                 if(!topOfInning) {
                        topOfInning = true;
122                     if(++inning == 10) {
```

```
                        running = false;
                    }
                }
                else {
                    topOfInning = false;
                }
            }
        }
        else if(data == Result::MISS) {
            std::cout << "BATTER: missed, strike " << strikes << std::endl;
            if(++strikes == 3) {
                strikes = 0;
                std::cout << "Batter struck out" << std::endl;
                if(++outs == 3) {
                    std::cout << "End of inning" << std::endl;
                    for(int i = 0; i < runners.size(); ++i) {
                        runners[i] = 0;
                    }
                    outs = 0;
                    if(!topOfInning) {
                        topOfInning = true;
                        if(++inning == 10) {
                            running = false;
                        }
                    }
                    else {
                        topOfInning = false;
                    }
                }
            }
        }
        else if(data == Result::SINGLE || data == Result::DOUBLE) {
            std::cout << resultToString(data) << std::endl;
            bool addedHitter = false;
            for(int i = 0; i < runners.size(); ++i) {
                if(runners[i] != 0) {
                    runners[i] += data;
                    if(runners[i] > 3) {
                        std::cout << "SCORE!!!" << std::endl;
                        runners[i] = 0;
                        if(topOfInning) {
                            awayScore++;
                        }
                        else{
                            homeScore++;
                        }
                    }
                }
                else if(!addedHitter) {
                    addedHitter = true;
                    runners[i] += data;
                }
            }

        }
    }
    std::cout << "Game Over!\nFinal Score:\nAway: " << awayScore << "\nHome: " <<
     homeScore << std::endl;
}

void runBatter() {
    int data;
    MPI_Status status;

    while(true) {
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MCW, &status);

        if(status.MPI_TAG == Tag::GAMEOVER) {
            break;
        }

        if(status.MPI_TAG == Tag::PITCH) {
            Pitch pitch = Pitch(data);
            float swing = (rand() % 100) + 1;
            float attempt = swing / getPitchDifficulty(pitch);
            if(attempt > 25) { // Hit
                int hitTo = (rand() % 7) + 2;
                int hitDifficutly = rand() % 100;
                MPI_Send(&hitDifficutly, 1, MPI_INT, hitTo, Tag::HIT, MCW);
            }
            else {
                int miss = Result::MISS;
                MPI_Send(&miss, 1, MPI_INT, Position::PITCHER, Tag::RESULT, MCW);
            }

        }
    }


}
```

```
void runFielder() {
    int data;
    MPI_Status status;
    while(true) {
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MCW, &status);

        if(status.MPI_TAG == Tag::GAMEOVER) {
            break;
        }

        int attempt = rand() % 100;

        int diff = attempt - data;
        int message;

        if(diff > 0) {
            message = Result::OUT;
        }
        else if(diff > -30) {
            message = Result::SINGLE;
        }
        else {
            message = Result::DOUBLE;
        }
        MPI_Send(&message, 1, MPI_INT, Position::PITCHER, Tag::RESULT, MCW);
    }

}

void runGame(Position pos) {

    if(pos == PITCHER) {
        runPitcher();
        int pill = 0xDEADBEEF;
        for(int i = 1; i < 9; ++i) {
            MPI_Send(&pill, 1, MPI_INT, i, Tag::GAMEOVER, MCW);
        }
    }
    else if(pos == BATTER) {
        runBatter();
    }
    else {
        runFielder();
    }

}
Position setPosition(int rank) {
    switch(rank) {
        case 0: return PITCHER;
        case 1: return BATTER;
        default: return FIELDER;
    }
}

int main(int argc, char **argv) {

    int rank, size;
    Position position;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);
    srand(rank + time(NULL));

    position = setPosition(rank);

    runGame(position);

    MPI_Finalize();
    return 0;
}
```

./baseball.cpp