CS 5500 HW3 : Too Many Chefs

Marshal Taylor

February 1, 2022

Abstract

A concise report describing my implementation of Too Many Chefs

1 Introduction

To keep the number of parameters passed between each of the processors as small as possible, I opted to merge the layer that the ball was on, and the iteration of the simulation into a single variable. I multiply the number of iterations by the number of layers to get the total layers in the whole simulation. Essentially, I have created a really really really long pachiko board. Every time the ball hits a layer that should be the end of an iteration, we record where the ball landed in a text file. Then, based on command line parameters, we either pick the ball up and set it into a specific column, or out it into a random column. This happens continuously until the ball reaches layer 0.

2 Commands

1) Compile: VS code Build

```
2)mpiexec -np {number of processsors} .\Too Many Chefs.exe
```

3) The number of cooks is always 1. The number of chefs is the number of processors - 1.

3 Implementation

3.1 Cook Code

1) If the rank is 0, then the processor will be assigned as a cook. We initialize the order queue at 0. We will probe all unrecieved messages, and increment the order queue for each of the unreceived message, then receive it. Once there are no unrecieved messages, we can the start to process each of the order requests. If there are no orders, we can 420 for 1 second. If the order queue surpasses 20, we then need to send a message to each of the other processors, ie the chefs, and then quit. Once all of the chefs are made aware of the cook quitting,it terminates the program.

```
int rank, size;
 int data;
 int order_queue = 0;
 MPI_Request request;
 MPI_Status status;
 int flag = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MCW, &rank);
 MPI_Comm_size(MCW, &size);

 if (!rank) {
     // I am cook
```

```
        while (order_queue < 20) {
            MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, MCW, &flag, &status);
            while (flag) {
                MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 1, MCW, MPI_STATUS_IGNORE);
                MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, MCW, &flag, &status);
                order_queue++;
            }
            cout << "Order Queue " << order_queue << endl;
            if (order_queue) {
                 // cooking!
                order_queue--;
                cout << "Cook: I gotta cook! " << endl;
                Sleep(1000);
            }
            else {
                cout << rank << ": " << "420." << endl;
                Sleep(1000);
            }
        }
        // recieve all outstanding orders
        while (flag) {
            MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 1, MCW, MPI_STATUS_IGNORE);
            MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, MCW, &flag, &status);
        }
        for (int i = 1; i < size; i++) {
            cout << i << " You Friking Frick! When will you learn that you actions have
            MPI_Send(&rank, 1, MPI_INT, i, 1, MCW);
        }

    }
```

3.2 Chef Code Code

2) This is the chef's code. We initialize a random seed, then proceed to the while loop. The chef first checks to see if a cook walked out. If not, it takes a random time to think of a of a message (1 to 5 seconds), and then sends it to the chef. If a cook does walk out, it gets mad and leaves.

```
    srand(time(NULL) + rank);
    while (1) {
        MPI_Iprobe(0, MPI_ANY_TAG, MCW, &flag, &status);
        if (flag) {
            cout << rank << " You youngins just don't to work anymore." << endl;
            MPI_Recv(&data, 1, MPI_INT, 0, 1, MCW, MPI_STATUS_IGNORE);
            break;
        }
        else {
            Sleep((rand() % 5 + 1) * 1000);
            MPI_Send(&rank, 1, MPI_INT, 0, 1, MCW);
            cout << "Chef: I Hungy." << endl;
        }
    }
```

4 Expected Output

If the number of chefs and cooks is the same, the program will continue indefinitely. Ie, 2 processors. If the processors are 3 or greater, then it is expected for the program to terminate, as one cook cannot

keep up with 2 or more chefs. There are print statements for each action that a processor can take. 420, cooking, submitting orders, quitting, and getting flustered.

```
mpiexec -np 8 .\Too Many Chefs.exe
```