CS 5500 Homework 7

Sally Devitry (A01980316)

Approach/Implementation

In this assignment, we were tasked with implementing Sender-Initiated Distributed Random Dynamic Load Balancing with Dual-Pass Ring Termination.

The basic unit of work, a task, is represented by an integer i in the range [1-1024], incrementing by random numbers i times as the following code shows:

```
for (int i=0; i<myWork.front(); i++) {
    tempWork += rand()%size;
}
```

Each process has a queue in which it places tasks to perform and each process goes through the following steps:

-check to see if any new work has arrived
-if the number of tasks in the queue exceeds the -threshold, then 2 tasks are sent to random destinations
-perform some work
-for half the processes (even processes), if the number of tasks generated is less than a random number [1024-2048], generate 2 new tasks, and place them at the end of the process task queue. for the other half of the processes, generate no new tasks.

To terminate the program, the system implements a dual-pass ring termination algorithm. The dual-pass ring termination algorithm allows for graceful termination in a program where it's difficult to distinguish between a process idling or being completely done.

Some overall utilization examples are as follows:

8 processes: 3.39 seconds on average
12 processes: 3.82 seconds on average
16 processes: 3.64 seconds on average

My program seems to distribute tasks more evenly when the condition for passing tasks isn't so high. When I change the code to pass tasks to other processes after it's own queue only has 10 tasks. This does distribute the tasks more evenly, but the average execution time didn't go up.

Run Commands

```
mpirun -np 8 --oversubscribe a.out
mpirun -np 12 --oversubscribe a.out
```

Some example output (8 processes):

```
I am rank 4 with task queue size: 2
process: 4 doing work: 306
rank: 2 recieved 621
I am rank 4 with task queue size: 3
process: 4 doing work: 507
I am rank 6 with task queue size: 2
process: 6 doing work: 306
I am rank 4 with task queue size: 4
process: 4 doing work: 54
I am rank 0 with task queue size: 2
process: 0 doing work: 507
I am rank 6 with task queue size: 3
I am rank 4 with task queue size: 5
process: 4 doing work: 416
I am rank 4 with task queue size: 6
process: 4 doing work: 973
process: 6 doing work: 507
I am rank 6 with task queue size: 4
I am rank 4 with task queue size: 10
process: 4 doing work: 46
I am rank 6 with task queue size: 7
I am rank 4 with task queue size: 11
process: 6 doing work: 381
process: 4 doing work: 1015
I am rank 4 with task queue size: 10
process: 6 doing work: 235
I am rank 4 with task queue size: 11
process: 4 doing work: 318
rank 5recieved 626
I am rank 5 with task queue size: 1
process: 5 doing work: 626
I am rank 4 with task queue size: 10
process: 4 doing work: 255
rank 6recieved 845
I am rank 6 with task queue size: 10
process: 6 doing work: 895
process: 6 doing work: 626
rank 5recieved 845
I am rank 5 with task queue size: 1
process: 5 doing work: 845
rank 1recieved 46
I am rank 1 with task queue size: 1
process: 1 doing work: 46
```

```
rank 1recieved 155
I am rank 1 with task queue size: 1
process: 1 doing work: 155
I am rank 7 with task queue size: 11
process: 7 doing work: 570
...
```

Concluding Remarks

Overall, this assignment was a good exercise in practicing my MPI skills. The most difficult part was certainly implementing the dual-pass-ring termination.
I originally thought the token passing should happen outside of the while loop, but realized that it had to occur along with the task balancing. I thought the algorithm for the termination was a very cool algorithm.

Appendix (Code)

```
#include <iostream>
#include <queue>
#include <mpi.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

#define MCW MPI_COMM_WORLD

using namespace std;
using namespace std::chrono;

int main(int argc, char **argv){
    queue<int> myWork;
    int work;
    int rank, size;
    int data;
    int tempWork;
    int randRange = 1024;
    bool white = true;
    int token = -2; //-2 for black, -1 for white
    int dest1;
    int dest2;
    int tokenDest;
    int totalGenerated = 0;
    MPI_Status mystatus;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);
    srand(time(NULL));
    MPI_Request request;
```

```
if (rank == 0) {
    work = rand()%randRange;
    cout << "process 0 sending out initial work of: " << work << endl;
    MPI_Send(&work,1,MPI_INT,2,0,MCW);
}
if (rank == 2) {
    MPI_Recv(&work,1,MPI_INT,MPI_ANY_SOURCE,0,MCW,MPI_STATUS_IGNORE);
    cout << "rank: " << rank << " recieved " << work << endl;
    myWork.push(work);
}

while (true) {
    int msgRecv;
    MPI_Iprobe(MPI_ANY_SOURCE,MPI_ANY_TAG,MCW,&msgRecv,&mystatus);

    if (msgRecv) {
        MPI_Recv(&work,1,MPI_INT,MPI_ANY_SOURCE,0,MCW,MPI_STATUS_IGNORE);
        cout << "rank " << rank << "recieved " << work << endl;
        if (work == -3) {
            //recieved poison
            break;
        }
        myWork.push(work);
    }
    //recieve token if there
    MPI_Irecv(&token,1,MPI_INT,MPI_ANY_SOURCE,1,MCW,&request);
    token = token;
    if (rank == 0 && token == -1) {
        //send poison
        token = -3;
        for (int proc=1; proc<size; proc++) {
            MPI_Isend(&token,1,MPI_INT,proc,MPI_ANY_TAG,MCW,&request);
        }

        break;
    }

    if (myWork.size() != 0) {
        cout << "I am rank " << rank << " with task queue size: " << myWork.size() << endl;
        if (myWork.size() > 10) {
            work = myWork.front();
            myWork.pop();
            dest1 = rand()%size;
            if (dest1 < rank) {
                white = false;
            }
            MPI_Isend(&work,1,MPI_INT,dest1,0,MCW,&request);
            work = myWork.front();
            myWork.pop();
            dest2 = rand()%size;
            if (dest2 < rank) {
                white = false;
```

```
                }
                MPI_Isend(&work,1,MPI_INT,dest2,0,MCW,&request);
            }
            //perform some work
            cout << "process: " << rank << " doing work: " << myWork.front() << endl;
            for (int i=0; i<myWork.front(); i++) {
                tempWork += rand()%size;
            }
            myWork.pop();
        }

        if (rank % 2 == 0 && totalGenerated < rand() % randRange+100) {
            myWork.push(rand() % randRange);
            myWork.push(rand() % randRange);
            totalGenerated+=1;
        }
        if (rank == 0) {
            token = -1;
        }
        if (myWork.size() == 0 && token < 0) {
            if (token == -3) {
                break;
            }
            if (rank == size-1) {
                tokenDest = 0;
            }
            else {
                tokenDest = rank+1;
            }
            MPI_Isend(&token,1,MPI_INT,tokenDest,1,MCW,&request);
        }
    }

    MPI_Finalize();

    return 0;
}
```