cs6550

Brett Bonar

November 6, 2018

1 Implementation

My implementation begins with each process generating a random number of tasks to place on its queue. The queue in this instance is actually represented as the number of tasks to complete as each task is just a 1 second sleep. The main loop for each process includes checking if its number of tasks is higher than a threshold (16) and passing off 2 tasks to 2 random processes if it is. The process then performs one of the tasks off its queue then checks to see if it has received any new jobs. When a process's task queue is empty, it starts an MPI Recv to wait for the white/black token to be passed to it. If it is process 0 then it immediately sends the first white token to the next process.

2 Results

The timing and process load results were fairly even for each process. One exception is if a process has too few tasks and manages to complete before it was able to receive any new work. Otherwise the continual cycle of sending additional tasks to random processes and requesting new work with each iteration of the main loop helped ensure that each process was given a fairly even load.

3 Code

```
#include <iostream>
#include <unistd.h>
#include <cmath>
#include <mpi.h>
#include <string>
#include <vector>
#include <ctime>
#include <iostream>
#include <fstream>
#include <chrono>
#include <thread>
#define MCW MPI_COMM_WORLD

const int TASK_THRESHOLD = 16;
const int JOB = 0;
const int TOKEN = 1;
const int DONE = 1;

bool handleToken(int rank, int size, int& token, bool& isWhite)
{
  MPI_Recv(&token, 1, MPI_INT, MPI_ANY_SOURCE, TOKEN, MCW,
      MPI_STATUS_IGNORE);
  std::cerr << rank << " received token " << token << std::endl;
  if (rank == 0)
  {
    if (token == 1)
    {
      return true;
    }
    else
    {
      token = 1;
      MPI_Send(&token, 1, MPI_INT, (rank + 1) % size, TOKEN, MCW);
      std::cerr << rank << " sent token " << token << " to " << (
          rank + 1) % size << std::endl;
    }
  }
  else
  {
    if (!isWhite)
    {
      token = 0;
    }
    MPI_Send(&token, 1, MPI_INT, (rank + 1) % size, TOKEN, MCW);
    std::cerr << rank << " sent token " << token << " to " << (rank
         + 1) % size << std::endl;
    isWhite = true;
  }

  return false;
}

int main(int argc, char **argv){
  int rank, size;
  int data;
  int sendData = 1;
  bool isWhite = true;

  MPI_Request jobRequest;
  MPI_Request tokenRequest;
  MPI_Request sendRequest;
  MPI_Request doneRequest;
-
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MCW, &rank);
  MPI_Comm_size(MCW, &size);

  srand(time(nullptr) * rank);
  int numTasks = rand() % 32 + 2;

```

```
std::cerr << "Rank: " << rank << ", Start Tasks: " << numTasks << std::endl;

MPI_Irecv(&sendData, 1, MPI_INT, MPI_ANY_SOURCE, JOB, MCW, &jobRequest);

while (numTasks)
{
  if (numTasks)
  {
    if (numTasks > TASK_THRESHOLD)
    {
      int count = 2;
      for (int i = 0; i < count; i++)
      {
        int target = rand() % size;
        if (target < rank)
        {
          isWhite = false;
          //std::cerr << rank << " is black " << std::endl;
        }
        //std::cerr << rank << " sending task to " << target << std::endl;
        MPI_Isend(&sendData, 1, MPI_INT, target, JOB, MCW, &sendRequest);
      }
      numTasks -= count;
      //std::cerr << "Rank: " << rank << ", # Tasks: " << numTasks << std::endl;
    }

    int jobFlag = 0;

    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    numTasks--;
    std::cerr << rank << " completed a task, now has " << numTasks << std::endl;

    MPI_Test(&jobRequest, &jobFlag, MPI_STATUS_IGNORE);
    while (jobFlag)
    {
      numTasks++;
      std::cerr << rank << " received a task, now has " << numTasks << std::endl;
      MPI_Irecv(&sendData, 1, MPI_INT, MPI_ANY_SOURCE, JOB, MCW, &jobRequest);
      MPI_Test(&jobRequest, &jobFlag, MPI_STATUS_IGNORE);
    }
  }
}


//MPI_Test(&doneRequest, &doneFlag, MPI_STATUS_IGNORE);

std::cerr << rank << " is finished with all tasks" << std::endl;

int token = 1;
```

```
  if (rank == 0)
  {
    MPI_Send(&token, 1, MPI_INT, (rank + 1) % size, TOKEN, MCW);
    std::cerr << rank << " sent token " << token << " to " << (rank + 1) % size << std::endl;
  }

  int pass = 0;
  while (!handleToken(rank, size, token, isWhite))
  {
    if (pass > 0)
    {
      break;
    }
    pass++;
  }

  std::cerr << rank << " is done" << std::endl;

  MPI_Finalize();

  return 0;
}
```

4 Compile and Run Commands

```
mpic++ Assignment9/Assignment9.cpp -o Assignment9/run.out
time mpirun -np 8 Assignment9/run.out
```