6550 Parallel Programming HW5

Riley Densley

October 2019

1 Mandelbrot on Single Process

I tried a couple variations on this approach. The first method I tried was to have one master sending tasks to the other processes. The master would send a row number to a process and the process would generate the results for that row and send the results back to the master. When the master receives a result they send that process another task. This continues until all the rows are completed. On my machine this took 3.168 seconds. Faster than 5.63 seconds with one process.

One issue is that the master process has to take care of the received input and then send more tasks. This takes time. I tried a different solution where there was a master and a sink process. The master would send tasks and and the sink would receive the result. This in theory would speed up the task distribution. Surprisingly the time taken was 3.43 seconds. Almost the same as one master.

2 Code

```
#include <iostream>
#include <mpi.h>
#include <random>
#include <unistd.h>
#include <cmath>
#include <fstream>
#include <chrono>
#include <stdlib.h>
#include <queue>
#define MCW MPI_COMM_WORLD

using namespace std;
//using namespace std::chrono;

struct Complex
{
  double r;
  double i;
};

Complex operator+(Complex s, Complex t)
{
  Complex v;
  v.r = s.r + t.r;
  v.i = s.i + t.i;
  return v;
}

Complex operator*(Complex s, Complex t)
{
  Complex v;
  v.r = s.r * t.r - s.i * t.i;
  v.i = s.r * t.i + s.i * t.r;
  return v;
}

int mandelbrot(Complex c, int maxIters)
{
  int i = 0;
  Complex z;
  z = c;
  while (i < maxIters && z.r * z.r + z.i * z.i < 4)
  {
    z = z * z + c;
    i++;
  }
  return i;
}

string COLORS[] = {
    "66 30 15 ",    // brown 3
    "25 7 26 ",     // dark violett
    "9 1 47 ",      // darkest blue
    "4 4 73 ",      // blue 5
    "0 7 100 ",     // blue 4
    "12 44 138 ",   // blue 3
    "24 82 177 ",   // blue 2
    "57 125 209 ",  // blue 1
    "134 181 229 ", // blue 0
    "211 236 248 ", // lightest blue
    "241 233 191 ", // lightest yellow
    "248 201 95 ",  // light yellow
    "255 170 0 ",   // dirty yellow
    "204 128 0 ",   // brown 0
    "153 87 0 ",    // brown 1
    "106 52 3 "     // brown 2
};

int main(int argc, char **argv)
{
  int rank, size;
  int data;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MCW, &rank);
  MPI_Comm_size(MCW, &size);
  MPI_Status mystatus;
  srand(time(NULL));
  MPI_Request request;

  int maxIters = 500;
  const int rows = 516;
  const int cols = 516;
  // double re[2] = {stod(argv[1]), stod(argv[3])};
  // double im[2] = {stod(argv[2]), stod(argv[2]) + (stod(argv[3]) - stod(argv[1]))};

  //diagonal -.722 .2003 -.718
  // twist -.72043 .2024 -.72019
  double re[2] = {-.72043, -.72019};
  double im[2] = {.2024, .2024 + (-.72019 + .72043)};

  double stepC = double((re[1] - re[0]) / cols);
  double stepR = double((im[1] - im[0]) / rows);

  if (rank == 0)
  {
    auto begin = chrono::high_resolution_clock::now();

    int **mand = new int *[rows];
    for (int r = 0; r < rows; r++)
      mand[r] = new int[cols];

    bool done = false;
    int currentRow = 0;
    int activeTasks = 0;
    queue<int> availableProcesses;
    int *mandPart = new int[cols];

    for (int i = 1; i < size; i++)
      availableProcesses.push(i);
```

```
while (!done)
{
  if (activeTasks < size - 1 && currentRow < rows)
  {
    // Send a task
    int sendTo = availableProcesses.front();

    availableProcesses.pop();
    MPI_Send(&currentRow, 1, MPI_INT, sendTo, 0, MCW);

    currentRow++;
    activeTasks++;
  }
  else if (activeTasks > 0)
  {
    //recieve a task
    int flag = 0;

    int *mandPart = new int[cols];
    MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MCW, &mystatus);

    MPI_Recv(&*mandPart, cols, MPI_INT, mystatus.MPI_SOURCE, mystatus.MPI_TAG, MCW, MPI_

    mand[mystatus.MPI_TAG] = mandPart;

    // queue waiting processor
    availableProcesses.push(mystatus.MPI_SOURCE);
    activeTasks--;
  }
  else
  {
    done = true;
  }
}

for (int i = 1; i < size; i++)
{
  int killer = -1;
  MPI_Send(&killer, 1, MPI_INT, i, 0, MCW);
}

// Write to file
ofstream fout;
fout.open("mandelout6.ppm");
fout << "P3\n"
```

```
        << rows << " " << cols << endl
        << "255\n";

  for (int row = rows - 1; row >= 0; row--)
  {
    for (int col = 0; col < cols; col++)
    {
      if (mand[row][col] >= maxIters)
      {
        fout << "0 0 0 ";
      }
      else
      {
        int colorStep = maxIters / 16;
        int j = 0;
        bool set = false;
        int current = mand[row][col];
        for (int i = 0; i < 16; i++)
        {
          if ((i + 1) * colorStep > current)
          {
            fout << COLORS[i];
            set = true;
            break;
          }
        }
        if (!(set))
          fout << COLORS[15];
      }
    }
    fout << endl;
  }
  fout.close();

  auto end = chrono::high_resolution_clock::now();
  auto dur = end - begin;
  auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(dur).count();
  cout << "Total time taken is " << ms / 1000.0 << " seconds\n";

  delete[] mandPart;
  for (int i = 0; i < rows; i++)
    delete[] mand[i];
  delete[] mand;
}

else
```

```
  {
    //Slave
    bool done = false;
    int row;
    Complex current;

    while (!done){
      MPI_Recv(&row, 1, MPI_INT, 0, 0, MCW, MPI_STATUS_IGNORE);

      //create mandelbrot of this row
      if (row == -1) {
        cout << rank << " is done working\n";
        break;
      }
      int *mandWork = new int[cols];

      for (int col = 0; col < cols; col++){
        current.i = double(row) * stepR - im[1];
        current.r = double(col) * stepC + re[0];
        mandWork[col] = mandelbrot(current, maxIters);
      }

      // Send completed madelbrot
      MPI_Send(&*mandWork, cols, MPI_INT, 0, row, MCW);
      delete[] mandWork;
    }
  }
  MPI_Barrier;

  MPI_Finalize();

  return 0;
}
```

3 Output

3.1 Commands

```
  mpic++ assign5.cpp
  mpirun -np 16 ./a.out -.72043 .20240 -.72019
```

3.2 Output

```
Total time taken is 3.168 seconds
```