Parallel Programing: HW4

Andres Imperial

February 14, 2020

Implementation

I implemented my parallelized mandelbrot by having each process compute the number of rows that they were going to be responsible for. They know how many other threads there are and where to index into the image. For example if the there were 4 threads they would take the rows in the following order: 1 2 3 4 1 2 3 4 Once every thread has done their work they will then send all of it back to thread 0 and thread 0 will write everyones work in order to an image file. Running my program serially I get a time of about 550 ms and when I run multi-threaded with 8 threads I get a better time around 140 ms.

Code:

Listing 1: main.cpp

```
#include <chrono>
#include <cmath>
#include <complex>
#include <cstdio>
#include <fstream>
#include <mpi.h>
#include <vector>

using namespace std;
#define MCW MPI_COMM_WORLD

int WIDTH = 512;
int HEIGHT = 512;
float re0;
float re1;
float c0;
float c1;

int value(int x, int y) {
  complex<float> point((static_cast<float>(x) * (re0 - re1) / WIDTH) + re1,
```

```
                        (static_cast<float>(y) * (c0 - c1) / HEIGHT) + c1);

  complex<float> z(point);
  unsigned int numIters = 0;
  for (; abs(z) < 2 && numIters <= 34; ++numIters) {
    z = z * z + point;
  }

  if (numIters < 34) {
    return 255 * numIters / 33;
  }

  return 0;
}

int main(int argc, char **argv) {
  if (argc != 4) {
    re0 = 0.5;
    re1 = 0.010;
    c0 = 0.00019;
    c1 = c0 + (re1 - re0);
  } else {
    re0 = atoi(argv[1]);
    c0 = atoi(argv[2]);
    re1 = atoi(argv[3]);
    c1 = c0 + (re1 - re0);
  }

  int rank, size;
  int data;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MCW, &rank);
  MPI_Comm_size(MCW, &size);

  auto start = std::chrono::high_resolution_clock::now();

  int numRows = HEIGHT / size;
  if (rank < static_cast<int>(HEIGHT) % size) {
    ++numRows;
  }

  std::vector<int> subRows(numRows * WIDTH);

  for (int row = 0; row < numRows; ++row) {
    for (int col = 0; col < WIDTH; ++col) {
      int val = value(col, row * size + rank);
```

```
    subRows[row * WIDTH + col] = val;
  }
}

if (rank == 0) {
  ofstream myBrot("./myBrot.ppm");

  // Image header
  myBrot << "P3\n" << WIDTH << " " << HEIGHT << " 255\n";

  if (myBrot.good()) {
    // Recieve data
    vector<vector<int>> finalRows(size, vector<int>{});
    finalRows[0] = subRows;

    for (int i = 1; i < size; ++i) {
      int tempRows = HEIGHT / size;
      if (i < static_cast<int>(HEIGHT) % size) {
        ++tempRows;
      }

      std::vector<int> temp(tempRows * WIDTH);
      MPI_Recv(&temp[0], tempRows * WIDTH, MPI_INT, i, 0, MCW,
               MPI_STATUS_IGNORE);
      finalRows[i] = temp;
    }

    for (int row = 0; row < HEIGHT; ++row) {
      for (int col = 0; col < WIDTH; ++col) {
        auto val = finalRows[row % size][row / size * WIDTH + col];
        switch (row % 6) {
        case 0:
          myBrot << val << ' ' << 0 << ' ' << 0 << "\n";
          break;
        case 1:
          myBrot << val << ' ' << val << ' ' << 0 << "\n";
          break;
        case 2:
          myBrot << 0 << ' ' << val << ' ' << 0 << "\n";
          break;
        case 3:
          myBrot << 0 << ' ' << val << ' ' << val << "\n";
          break;
        case 4:
          myBrot << 0 << ' ' << 0 << ' ' << val << "\n";
          break;
```

```
            case 5:
              myBrot << val << ' ' << 0 << ' ' << val << "\n";
              break;
            };
          }
        }

        myBrot.close();
      } else {
        throw std::runtime_error("was not able to open file");
      }
    } else {
      // Send data to thread 0
      MPI_Send(&subRows[0], numRows * WIDTH, MPI_INT, 0, 0, MCW);
    }

    MPI_Finalize();

    if (rank == 0) {
      auto stop = std::chrono::high_resolution_clock::now();
      auto duration =
          std::chrono::duration_cast<std::chrono::milliseconds>(stop - start);
      printf("%li\n", duration.count());
    }

    return 0;
}
```

How to run:

```
mpic++ main.cpp -o main
mpirun -np <n> -oversubscribe ./main
OR
mpirun -np <n> -oversubscribe ./main <Re0> <Im0> <Re1>
```

Output

Timing subject to change dependent on machine used for execution.