CSCI 3366 (Introduction to Parallel and Distributed Processing), Spring 2002

Homework 2

Assigned: February 7, 2002.

Due: February 19, 2002, at 5pm.

Credit: 40 points.

> *Note:* The HTML version of this document may contain hyperlinks. In this version, hyperlinks are represented by showing both the link text, formatted like this, and the full URL as a footnote.

Contents

1 Overview **1**

2 Details **1**

2.1 The sequential program 1

2.2 The starter MPI program 2

2.3 What you need to do 3

3 What to turn in **4**

1 Overview

The textbook presents two parallel algorithms for computing the Mandelbrot set. For this assignment, your mission will be to implement one of these algorithms.

The text also mentions that sequential code that solves this problem and displays the results graphically can be downloaded from the Web. I obtained this code and made several revisions (color display rather than black-and-white, command-line arguments to allow a more interesting range of calculations, C++ rather than C, etc.); this will be your starting point. The original program was written in C; I converted it to C++ but used the C bindings for MPI function calls.

2 Details

2.1 The sequential program

First obtain code for the revised sequential program:

- mandelbrot-seq.cpp[1]
- Makefile[2]

[1] http://www.cs.trinity.edu/~bmassing/CS3366_2002spring/Homeworks/HW02/Problems/mandelbrot-seq.cpp

[2] http://www.cs.trinity.edu/~bmassing/CS3366_2002spring/Homeworks/HW02/Problems/Makefile

(I recommend that you download these two files either by using your browser's "save as" function or with the Linux command `wget` (type `wget` followed by the URL of the file you want to obtain). Cutting and pasting text appears to be error-prone.)

Try compiling the program and running it. To compile the program, use the command

```
make mandelbrot-seq
```

(The makefile provides some additional libraries without which the program will not compile/link.) Run it as you would run any other sequential program, i.e., by typing its name (`mandelbrot-seq`). It has several optional command-line arguments:

- The center and size of the region to examine. By default, the program performs calculations for the region described in the textbook, a square of side 4 centered at (0, 0). To specify instead that you want the upper right quadrant of this square, for example, you would type

  ```
  mandelbrot-seq 0.5 0.5 2
  ```

- The maximum number of iterations to be done at each point (default 100). As the size of the region decreases, it may be necessary to increase the number of iterations to see more detail. So to repeat the above example, doubling the number of iterations, you would type

  ```
  mandelbrot-seq 0.5 0.5 2 200
  ```

By adjusting these parameters you can zoom in on a particular subregion and examine it in more detail. As a side benefit, increasing the number of iterations means there is more total calculation to be done and makes a parallel solution more attractive.

Usage notes:

- Note that since the program produces a graphical display, you will need to either be sitting in front of the computer you're running it on or connected to it in a way that supports running X-based programs. An example of the latter situation: You are sitting in front of `Janus01` but running your program from an `ssh` session on `Dwarf1`. You can run the Mandelbrot program from the `ssh` session, but to do so you must first tell it where to display the output, by issuing this command (from the `ssh` session):

  ```
  export DISPLAY=Janus01.cs.trinity.edu:0.0
  ```

  (Of course, if you are sitting in front of a different machine, replace `Janus01` with its name.)

- Note also that the program reports how much time was spent doing calculations (as opposed to setting up the display, etc.); you can compare this number to the number produced by your parallel program to see if parallelization really helped.

2.2 The starter MPI program

Your mission will be to take the above sequential program and turn it into an MPI-based program with a "master" process that sets things up and displays the results, and one or more "slave" processes that perform the calculations. To help you get started, I am providing a "starter" MPI program that sets up a master process and one or more slaves, but continues to do all the calculations in the master process. It also times the actual calculations (using `MPI_Wtime()`) and prints the result. Your job will be to move the calculational part of the program into the slave processes, as described in the textbook. You can obtain code for the starter MPI program here:

- `mandelbrot-mpi.cpp`[3]
- `Makefile`[4] (for compiling with the LAM implementation — if you are using MPICH, you will have to make appropriate modifications)

Try compiling and running it as is. To compile the program, use the command

```
make mandelbrot-mpi
```

(The makefile provides some additional libraries without which the program will not compile/link.) Run it as you would run any other MPI program; e.g., to run it with one slave process (hence two processes total), use the command

```
mpirun -np 2 mandelbrot-mpi
```

This program has the same three optional command-line arguments as the sequential program. Note that to provide command-line arguments to an MPI program, you put them *after* the program name, e.g.:

```
mpirun -np 2 mandelbrot-mpi 0.5 0.5 1 1000
```

Like the sequential program, this starter parallel program reports how much time was spent doing calculations (as opposed to setting up the display, etc.); you can compare this number to the number produced by your parallel program to see if parallelization really helped.

2.3 What you need to do

Once you have the starter program compiled and running, start thinking about how to modify it. You can use either of the algorithms described in the text (static or dynamic task assignment), and you can have the slave processes send back their results one point at a time or you can figure out how to combine many points into a single message (e.g., by sending back results for a row at a time). One thing to notice is that the master process may have information that the slaves need but cannot cleanly compute for themselves (examples are the variables `min_color` and `max_color`, which are computed by the master process as it initializes the X display). You should decide which of these variables will be needed in the slave processes and then include code to send them from the master to all the slaves.

You should only need to modify code in the two functions `master_pgm()` and `slave_pgm()`. Look for the following comments:

```
// START OF SECTION TO CHANGE
// END OF SECTION TO CHANGE
```

You should not need to modify anything in `main()` or the X-related functions. (An exception is the parameter to function `usleep()`, which may need to be increased to allow the X window to finish initializing before the program starts writing to it. If your program's display has a blank (white) space at the top, try increasing this parameter.)

[3] `http://www.cs.trinity.edu/~bmassing/CS3366_2002spring/Homeworks/HW02/Problems/mandelbrot-mpi.cpp`

[4] `http://www.cs.trinity.edu/~bmassing/CS3366_2002spring/Homeworks/HW02/Problems/Makefile`

3 What to turn in

Submit your source code as described in the Guidelines for Programming Assignments[5], using a subject header of "cs3366 hw 2".

[5] `http://www.cs.trinity.edu/~bmassing/CS3366_2002spring/Notes/pgmguidelines/index.html`