CSCI 3366 (Introduction to Parallel and Distributed Processing) Spring 2001

Homework 2

Assigned: February 2, 2001.

Due: February 13, 2001, 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

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 revised it to match the textbook's description more closely; this will be your starting point.

2 Details

2.1 The sequential program

First obtain code for the revised sequential program:

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

[1] `http://www.cs.trinity.edu/~bmassing/CS3366_2001spring/Homeworks/HW02/Problems/mandelbrot-seq.c`

[2] `http://www.cs.trinity.edu/~bmassing/CS3366_2001spring/Homeworks/HW02/Problems/Makefile.mandelbrot-seq`

(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 this command:

```
make -f Makefile.mandelbrot-seq 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 three optional command-line arguments:

- The maximum number of iterations to be done at each point (default 100). By adjusting this number you can vary the total amount of computing being done. This allows you to see how your parallel code compares to this original sequential code for various "problem sizes". (It is often the case that parallel programs don't seem much faster than their sequential counterparts for small problems — as in Homework 1 — but do as the size of the problem increases.)
- The width and height of the display, in pixels.

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-window programs.

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.c[3]
- Makefile.mandelbrot-mpi[4]

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

[3] `http://www.cs.trinity.edu/~bmassing/CS3366_2001spring/Homeworks/HW02/Problems/mandelbrot-mpi.c`

[4] `http://www.cs.trinity.edu/~bmassing/CS3366_2001spring/Homeworks/HW02/Problems/Makefile.mandelbrot-mpi`

```
make -f Makefile.mandelbrot-mpi mandelbrot-mpi
```

Run it as you would run any other MPI program; e.g., to run it with one slave process (hence two processes total), use this command:

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

It 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 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 has several pieces of information that may be needed by the slave processes — the width and height of the display area, the variables used to scale points and colors, etc. 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 not need to modify any of the X-related code in the master process; focus on the part of the code between the two calls to `MPI_Wtime()`. (You also should not change these calls.)

Note that the original sequential program was written in C and that I did not attempt a conversion to C++. Students are welcome to do so, but it is probably easier to learn to work with C.

3 What to turn in

Submit your revised 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_2001spring/Notes/pgmguidelines/index.html