Homework 2

Tom Kelliher, CS 325

30 points, due Feb. 17, 2012

The Assignment

The objective of this assignment is to use Open MPI to parallelize the serial Mandelbrot set image program accompanying this assignment, using the given problem parameters (boundaries of the complex plane, image resolution, disk escape value (`dSq`), maximum number of iterations per point, and point coloring algorithm and colors).

Your solution should use dynamic task assignment and work for any reasonable number of processes (between 2 and, say, 24). The root process should dynamically assign one row to a worker requesting work. Solutions using static task assignment, sending a row-at-a-time from workers to the root will be penalized three points. Solutions using static task assignment, sending a single point-at-a-time from workers to the root will be penalized six points. Solutions which assume a particular number of processes (see item 3 under *Structuring Your Program*, below) will be penalized three points.

Documentation Requirements

Minimally, your program must contain your name and an overview at its top similar to this:

```
/**************************************************************************
 * Tom Kelliher, Goucher College.
 *
 * mandelbrot.c --- Serial program to generate a PPM image
 *                  representation of the Mandelbrot set for some
 *                  rectangular portion of the complex plane.
 *
 * Due to the use of sqrtf(), the math library needs to be compiled-in:
 *
 *    gcc -lm -o mandelbrot mandelbrot.c
 *
 * The PPM image is written to stdout.  It will be HUGE, so it is
 * advisable to pipe the output to pnmtojpeg (on a Linux system)
 * and redirect the filter's output to a file:
 *
 *    ./mandelbrot | pnmtojpeg > image.jpg
 *
 * The pack()/unpack() routines, unnecessary in a serial environment,
 * _might_ be useful in a message passing environment, but that is a
 * claim that ought to be tested.
 **************************************************************************/
```

If the program you submit for a grade does not work, I will need further documentation within the program itself in order to assign partial credit. (You risk receiving **no** partial credit if you do not provide sufficient documentation.) One component of this documentation should be an analysis, to the best of your ability, of why the program is not working.

Designing Your Program

1. `pnmtojpeg` (see *Compiling and Running Your Program*) will probably not work if your program prints anything other than the PPM image to `stdout` via `printf()`. Therefore, any debugging messages should be printed to `stderr` using `fprintf()`:

   ```
   fprintf(stderr, "Rank %d received row %d.\n", rank, row);
   ```

   This will allow debugging messages to still display on the terminal, even as the PPM image is being piped to `pnmtojpeg`.

2. Your program will be complex enough that the code for the root and the workers should be in their own functions:

   ```
   int main(int argc, char *argv[])
   {
      ...

      if (rank == 0)
         root();
      else
         worker(rank);

      ...
   }
   ```

3. In your program design, you should assume that the root process performs no point computations. Hence, at least two processes are needed. The root process will only assign tasks to workers (dynamic task assignment), collect results into a two-dimensional array, and output the PPM image.

4. Note that all of the problem parameters are known in advance by all the processes. Hence, under dynamic task assignment every worker process knows its initial work assignment — the rank i worker process starts off working on row $i - 1$.

   Similarly, under static task assignment every worker process knows that it will handle $k = 2304/(size - 1)$ rows, starting with row $k \times (rank - 1)$ with the rank $size - 1$ worker process handling the remainder rows if $size - 1$ doesn't evenly divide 2304. (It turns out that 12 evenly divides 2304, so there are no remainder rows if the number of running processes is 13.)

5. As described in the textbook, you will find it handy to use `DATA`, `TERMINATE`, and `RESULT` message tags. Both the root and workers should execute `MPI_Recv()` using the wildcard tag. The actual message tag may be discovered by referencing `status.MPI_TAG`, assuming that `status` is the `MPI_Status` parameter to the `MPI_Recv()` call. Similarly, `status.MPI_SOURCE` may be consulted to determine the source of a message received when using the source wildcard.

Walk, Then Run

You may find the following approach to solving this problem helpful. I started by first solving a slightly easier problem — I wrote a program that performed the dynamic task assignment work. The worker processes sent "results" to the root, which printed which worker "finished" which row. The root then assigned a new row to the worker. At this stage, I didn't concern myself with the Mandelbrot sequence calculation, I just wanted to be sure that every row got assigned to a worker and that all the processes terminated at the correct time. Then, knowing that I had the message passing and work assignments parts of the problem under control, I was easily able to extend this simpler program to solve the actual problem.

If you do not follow this sort of an approach, you run the risk of having to debug your full parallel program in one fell swoop, a notoriously difficult task.

Compiling and Running Your Program

Due to the use of `sqrtf()`, the math library will need to be linked into your program:

```
mpicc -lm -o parMandelbrot parMandelbrot.c
```

You will also need to include the math library's header file, `math.h`.

Remember, PPM files are **GINORMOUS**, so pipe your program's output to `pnmtojpeg` and redirect the latter's output to a file:

```
mpirun parMandelbrot | pnmtojpeg > image.jpg
```

Seeking Assistance

I strongly encourage you to begin this assignment as soon as it is issued and to come to my office if you need assistance. If you email me to describe a problem, describe the problem carefully and attach your source code. If extensive debugging will be required to determine the cause of the problem, I will let you know that you will need to visit me in person to resolve the problem. (In other words, I will not debug your program for you.)

Submitting Your Assignment

By the start of class on the 17th, send me your source code and a JPG image showing the result of running your program. Debugging messages in your source code should be commented-out. Late work will be penalized 15% per day, with the weekend counting as one day. (An unforeseen circumstance preventing you from finishing an assignment on time, while rare, may warrant a request for a deadline extension. Any such request must be made in writing, state the length of the extension requested, explain the nature of the unforeseen circumstance, include a strong justification for the requested extension, and be made at least 24 hours in advance of the assignment deadline. The strength of your justification and the nature of your unforeseen circumstance will determine whether or not the request is granted.)