MCS 572, Spring 2008
Project 1: The Mandelbrot set

1. Compile and run the serial program `mandelbrot.c` from my webpage. Observe how the computational time may very depending on the coordinates of the area which you specify in the command line parameters of the program (recall that many more iterations are performed for the points which belong to the Mandelbrot set than for those which are farther away from it).

2. Implement the parallel version of the program using the static workload allocation. Namely, at runtime you have to divide the number of testing points \( N \) by the number of assigned processors \( p \), and each slave process has to receive coordinates of \( N/p \) points from the master process before the beginning of computation via a single MPI communication, then, after computation, send the number of iterations for each point back also via a single MPI communication. The master process has to write the postscript file after receiving the data. As for partitioning of the Mandelbrot set domain, I suggest horizontal strips (just as I have used for image transformations), but it is ultimately up to you, partition the domain however you want, just make sure each slave receives equal (or at least roughly equal) number of testing points.

3. Perform the following computations with the static workload allocation:

   a) Set the resolution of the Mandelbrot domain to 1000 \( \times \) 1000, 1400 \( \times \) 1400, 2000 \( \times \) 2000 and 2800 \( \times \) 2800 points.

   b) For each resolution, compute the problem on 2, 4, 8 and 16 processors and record the computation time using `C time()` function on the master node. Include both the communication and computation routines between your `time()` statements. In total, you should have 16 recorded times.

   c) From the data you have at this point, compute the speedup for each problem total size, scaled speedup for each fixed chunk size (observe that domain sizes scale approximately by \( \sqrt{2} \), which means that the number of points scales roughly by 2 like number of processors), and efficiency.

4. Implement the parallel version of the program using the dynamic workload allocation. For that, do just what you did for the static workload allocation, however partition the domain into many more parts than the number of processors \( p \) (in this work you will use 10\( p \) and 100\( p \)). Then, after the computation begins, send first \( p \) parts to the slave processes, and whenever any slave process sends the data back, send the new part to this process to continuously load it with work.

5. Perform the same computations as for the static workload distribution with the same parameters, i.e. resolution and the number of processors, and partition the Mandelbrot domain into 10\( p \) and 100\( p \) equal parts (so that each processor on average executes 10 and 100 parts, respectively) for the dynamical workload distribution. In total you should have 32 recorded times, due to two different partitions.

6. Like for the static workload distribution, compute the speedup for each problem total size, scaled speedup for each fixed chunk size, and efficiency. Compare the results with the results from the static workload distribution and observe the trends.

7. For this project, you should return the C code of your program (both static and dynamic load distribution), the speedup/efficiency analysis, and the plots of the Mandelbrot set you produced. In fact, you are free to compute an arbitrary subset of the Mandelbrot set instead (however it should be the same for all computations), just make sure that it has ample amount of points which belong and do not belong to the Mandelbrot set, to ensure larger uncertainty in computational time between different points.