

OpenMP
CSI 702- Fall 2010 - Assignment 6
Due: Monday April 26

Overview

For this assignment, you will experiment with several of the code tuning techniques we covered in class. You may write your codes in C, C++, or Fortran. The timing results should be done using the profiling tools for the appropriate compiler. Details for submission will be posted in the final version of this document on-line.

Objectives At the end of this assignment, students will

- Learn how to use OpenMP to solve practical problems
- Use basic OpenMP constructs
- Compare OpenMP implementations to serial implementation, and understand the challenges of moving to OpenMP, including work patterns
- Learn to use the OpenPBS to submit jobs on the Gmice machine

Requirements:

For this assignment-

- You may use C, C++, or Fortran for the serial code.
- You **MUST** use OpenMP to parallelize the code.
- You need to create both a serial and parallel version of each code.
- Include a makefile that compiles the code into separate executables and performances a timing test.
- **You must use the gmice cluster (gmice.gmu.edu) for running the profiles on this project.** The sample script from class can be modified to run the jobs using the queues discussed in our lecture. There are 8 cores on a single node, so you should try to scale your jobs to AT LEAST 8 cores using OpenMP. You may wish to scale this higher to examine how adding additional threads might increase performance. You may use other computers to profile the program as well, but using gmice is required for this project.

- *You need to have a write up for each problem that documents*
 - The parallel approach taken
 - The problems you had during the implementation
 - A quick summary of the OpenMP commands used
 - A performance analysis of the code, including changing the problem size and the number of nodes that the code runs on
 - Comparisons of the output of the serial and parallel codes. This might include visualization - if appropriate - of the results. Are the results identical?

Turning the assignment in For the programs you are turning in, please follow these instructions carefully.

- Create a directory named “youreemail-openmphpw” where “youreemail” is your email account name at gmu.edu. (Just “jwallin”, not “jwallin@gmu.edu”).
- Create a clean copy of the source file, the documentation, your openpbs job file, and Makefile, and any other files into separate directories.
- Create a file using tar and gzip named “youreemail-hw5.tgz” that contains the main “youreemail-openmphpw” directory and its subdirectories.
- Email this file to the instructor.

Grading For this problem, the grades will be based on:

- (35%) The quality of the documentation. How well did you document the speedup, the algorithm, and any problems you used? Did the sample runs show the results clearly?
- (35%) The code - does it compile and run? Does it accomplish the objectives of the assignment? Is it readable, and is there enough internal documentation to make it understandable?
- (20%) The scaling - does it scale well up to the eight cores? Was gmice and qsub used effectively to test this.
- (10%) Is the assignment turned in following the correct format?

Mandelbrot Set

The Mandelbrot Set is an iterative map in the complex plane that has chaotic properties. Given an initial location in the complex plane $C = x + iy$ where $i = \sqrt{-1}$, we can go through the following iterations.

$$r_0 = c$$

$$r_i = r_{i-1}^2 + c$$

remember, for complex multiplication

$$(x + iy) \times (x + iy) = x^2 - y^2 + 2ixy$$

After a given number of iterations, many points begin to diverge. Any iteration where $|r_i| > 2$ will leave the complex plane.

To make a Mandelbrot plot, you need to loop through the complex plane from $x = (-2, 2)$ and $y = (-2i, 2i)$. The pixel color at location C is associated with the number of iterations it takes to reach $C > 2$.

It is important to note that some starting points will never reach $C > 2$, so a maximum iteration number is needed in the problem (500 iterations, for example.)

For this problem, you are to make a parallel version of the Mandelbrot problem that uses the manager/worker programming pattern. You cannot use the fine grain scatter decomposition we discussed in the notes for this problem.

1. Using your previous serial code for the Mandelbrot set, add the adaptations needed to make it run effectively using multicores with OpenMP.
2. Experiment with the the scheduling types and sizes. Try using both static, dynamic and guided with different chunk sizes.
3. Examine the performance as a function of iteration size and grid size. Plot the results to show the speedup you get for the code under different conditions. Be sure to summarize the results in your final report.
4. All the code, the report, and the documentation should be turned in following the instructions above.