Parallel and SIMD Programming – Assignment #2 (Multi-threaded Mandelbrot viewer)

Due on 08/20/2008

Using the base code provided, you will implement a parallel Mandelbrot fractal generator / viewer. The sample code implements a simple, single-threaded Mandelbrot viewer. The viewer code should not need significant modification. Instead, create a new subclass of the `mandelbrot` base class using the `serial_mandelbrot` as a model. Call your new class `parallel_mandelbrot`. The only modification necessary to the viewer code should be to use `parallel_mandelbrot` instead of `serial_mandelbrot`.

`serial_mandelbrot` operates in the following manner. The `solve` method invalidates the `solved_iterations` data. It then calls `solve_area` for the entire area. `solve_area` calls `solve_point` for each point along the perimeter of the area. If every point along the perimeter has the same iteration count, the entire area is filled with that iteration count. Otherwise the region is subdivided into quadrants, and `solve_area` is called on each quadrant.

`parallel_mandlebrot` should implement this same algorithm. There are numerous opportunities for concurrency in this algorithm. Select and exploit at least one. Along with your program please submit a document that explains the concurrency you have chosen and how you have exploited it. This explanation is worth 10% of the grade for this assignment. The remainder of the grade is based on the implementation of that explanation using the usual programming assignment rubric.

Criteria	Excellent	Good	Satisfactory	Unacceptable
Completion	Program correctly implements all required elements in a manner that is readily apparent when the program is executed. User interface is complete and responsive to input. Program documents user interface functionality.	Program implements all required elements, but some elements may not function correctly. User interface is complete and responsive to input.	Program implements most required elements. Some of the implemented elements may not function correctly. User interface is complete and responsive to input.	Many required elements are missing. User interface is incomplete or is not responsive to input.
Correctness	Program executes without errors. Program handles all special cases. Program contains error checking code.	Program executes without errors. Program handles most special cases.	Program executes without errors. Program handles some special cases.	Program does not execute due to errors. Little or no error checking code included.
Efficiency	Program uses solution that is easy to understand and maintain. Programmer has analysed many alternate solutions and has chosen the most efficient. Programmer has included the reasons for the solution chosen.	Program uses an efficient and easy to follow solution (i.e., no confusing tricks). Programmer has considered alternate solution and has chosen the most efficient.	Program uses a logical solution that is easy to follow, but it is not the most efficient. Programmer has considered alternate solutions.	Program uses a difficult and inefficient solution. Programmer has not considered alternate solutions.
Presentation & Organization	Program code is formatted in a consistent manner. Variables, functions, and data structures are named in a logical, consistent manner. Use of white space improves code readability.	Program code is formatted in mostly consistent with occasional inconsistencies. Variables, functions, and data structures are named in a logical, mostly consistent manner. Use of white space neither helps or hurts code reability.	Program code is formatted with multiple styles. Variables, functions, and data structures are named in a logical but inconsistent manner. Use of white space neither helps or hurts code reability.	Program code is formatted in an inconsistent manner. Variables, functions, and data structures are poorly named. Use of white space hurts code reability.
Documentation	Code clearly and effectively documented including descriptions of all global variables and all non-obvious local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted, as are the input requirements and output results.	Code documented including descriptions of most global variables and most non-obvious local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted, as are the input requirements and output results.	Code documented including descriptions of the most important global variables and the most important local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted.	No useful documentation exists.

This rubric is based loosely on the "Rubric for the Assessment of Computer Programming" used by Queens University (http://educ.queensu.ca/ compsci/assessment/Bauman.html).