Project 1: Desert Road Map

Story

Wile E. Coyote need to catch that Roadrunner, the Roadrunner is the only eatable thing in the dessert, it's that simple! We all know that a large variety of ACME equipment has previously failed to help Wile so now he need a new approach. The first thing Wile must have to stand a chance to catch the Roadrunner is information on the desert, precise information, e.g. a map that may zoom into any desired detail. Fortunately this particular dessert may be described by a very simple mathematical function, also known as the Mandelbrot set. Unfortunately calculating the map is a large computational problem, and since we cannot apply any mechanical zoom techniques, we must generate a new image for each zoom factor of the required map. Thus to get his information fast enough Wile need a fast mechanism for generating a map. To this end we need a map generator that can utilize more than one CPU.

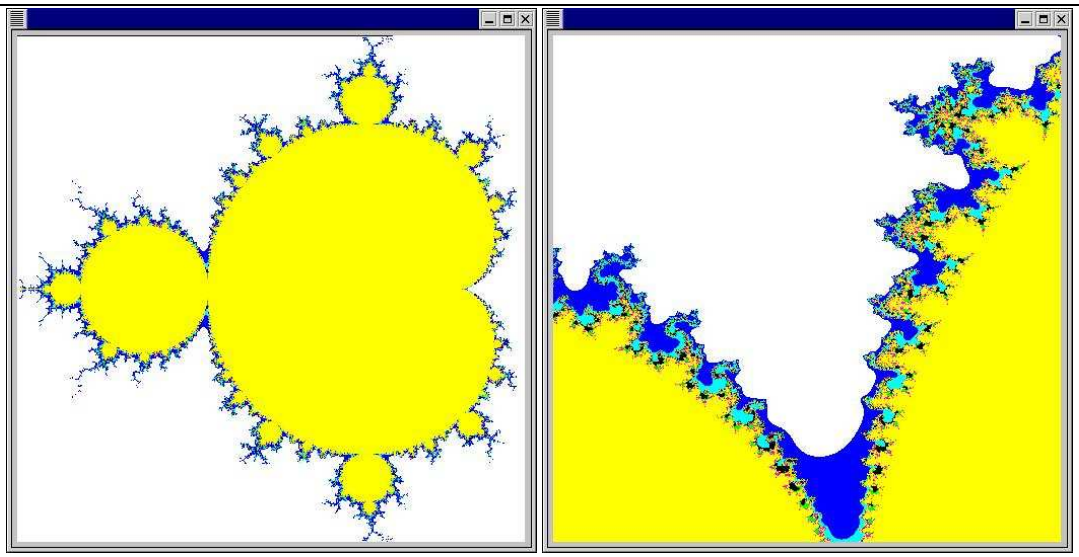

Figure 1 The map of the dessert, first the initial map and then the final zoom

Considerations on the parallel version

The parallel version of the Mandelbrot set belongs to the class of embarrassingly parallel applications, just as the Monte Carlo Pi program that we saw in chapter 3. In the case of the Mandelbrot set we can calculate each point independently of the others, which gives us a large degree of freedom in the choice of defining the grain-size for the parallel version. The picture is rendered by testing if the equation

$$f(x, y, r, s) = f(x, y, r^2 - s^2 + x, 2rs + y)$$

diverges, for each point (x,y) with initial r and s values of 0. If a diversion is not found within one hundred iterations the attempt is preempted. Each point in the rendered picture is a color representation of the number of iterations that was needed to test for divergence.

The essential problem in designing a parallel version of the Mandelbrot set is to orchestrate the work in a well-balanced way. Since each point (x,y) can diverge in a few iterations or may iterate one hundred times before the test is terminated the task of calculating the individual points is highly unbalanced and we need to try an find a way of achieving a work distribution which creates an overall balanced execution. Fortunately we are not likely to have one processor per point, so we can expect of average the unbalance over many points. If we have *n* processors we can distribute the work as:

- Split the map into *n* blocks horizontally
- Split the map into *n* blocks vertically
- Split the map into *n* grid-blocks
- Take every *n*'th row
- Take every *n*'th column
- Take every *n*'th point

Choosing the correct one of the above orchestration methods is an essential part of a successful implementation.

A minor issue is the actual graphic rendering of each point, this may introduce wrong results if done in parallel without extra synchronization and adding the synchronization will destroy any performance gain, this too must be handled too but this is not all that hard to handle.

Programming Task

Write two parallel versions of the map generation code, you may use the sequential version as provided as a core. Both parallel versions should be based on a multithreaded shared memory model and execute on a multiprocessor with physical shared memory. One version should use a static orchestration approach, where orchestration can be defined either at compile-time or initially at run-time. The other solution should be based on a producer-consumer approach, which should define the number of workers either at compile-time or run-time.

Your report should identify the various choices that have been made as well as individual techniques that have been applied to improve performance, and the impact of each should be documented. The provided code creates an initial map, the overall desert map as shown in Figure 1 and then zooms to a portion of the rim known as the horse head valley and returns the time taken to create all the involved fractals. Your parallel version should do the same and our report should include speedup numbers for both versions and you should try to explain the results.

Real World Relevance

Embarrassingly parallel applications:

- QCD
- Monte Carlo
- Many image-processing algorithms