Jake Christensen

CS 322 Wtr 2014

Assignment 6 Open MP

For our final task in our tour of concurrent libraries in C, this week we were asked to take a look at OpenMP. The problem the book asked was about solving a triangular matrix. The idea is that if we have a triangular matrix A, and a given vector of values for b, we want to calculate the vector x such that Ax = b. This task requires creating a matrix, and then running it through a pair of loops that perform calculations that ultimately provide us with our vector x. Two possible approaches to solving this problem were presented, one that operated based on row orientation, and one based on column. Our job was to take these nested loops and parallelize them using OpenMP, a library that works very well with loops like these.

In order to apply methods from OpenMP on our two approaches we needed to analyze the two algorithms to understand where we could add parallelization without affecting the calculation of our x values. When looking at our algorithms it is important to figure what areas of the calculation are dependant portions before them. Looking first at the column oriented method of solving this problem, I could see an area that would cause some problems if the whole problem were multithreaded. The outer loop performs an operation and saves it in the x vector. In the inner loop another calculation is made and saved into the x vector, at a *different* index than that in the outer loop. This means that if another thread makes changes on the outer loop, it is going to affect the calculation on the inner loop. Therefore we need every thread to finish the outer loop and meet back up before executing the inner loop, and cannot add an OpenMP pragma to it. Looking at the row oriented calculation, we see that the value row is the index for every x vector value we are operating on in both loops. This tells us that the calculations for each value in the x vector are done on one single variable in the vector, and we should be able to fully parallelize the both loops with OpenMP without affecting the values.

When putting these theories to the test I ran into some trouble. Building and getting the calculations to work was relatively painless, but when I began collecting data about the different programs, I was confounded at what it meant. In testing both methods in serialized vs. parallel I found that the performance of the serialized program greatly exceeded that of the OpenMP parallelized solutions. Regardless of the size of the matrix the parallel programs took about twice as long as the serial counter parts. Adjusting the scheduler helped with performance in some cases but it still never came anywhere close to the same speeds as the serialized program. I have a couple of theories for what might be causing this. First, I've observed with experiments in other parallel libraries that performance can degrade as the number of threads exceeds the number of cores. If OpenMP spins up a thousand threads for a 1000 x 1000 matrix, this may weigh down the program with the overhead of managing threads. Another possibility, is the always present possibility that I have used the tool wrong, and am incorrectly parallelizing the loops. This seems much more likely, or else OpenMP would be a useless tool when dealing with large for loops, as the majority of developers don't have access to massive clusters of thousands of cores to run their software on.

I've read and heard plenty of different analogies for what parallel programming is like. After spending the last two months getting my hands dirty with the tools that make parallel programs work I'd like to offer my own. Parallel programming is a lot like juggling, the biggest difference is that you don't have any guarantee which ball is going to come down next. In some projects I was successful in utilizing the power of multiple processors to speed up a program, but in others I was left clueless as to why a particular program was behaving the way it did. Going forward with my studies I plan to experiment with parallel libraries in some higher level languages such as Java, to see if they offer a method that is easier to control and understand.