Homework 1, Part 3

Cooper Faber

April 2021

1 Write-Up

In this part of the assignment, we investigated the performance improvements and overhead of multi-threading. Three loops were written and compared - all were incrementing every element of a size 1024 integer array from 0 to 524288 (originally twice that, I scaled it down to compensate for my poor hardware). Our first loop is basic, and serves as a reference. It sequentially takes each element, increments it hundreds of thousands of times, and moves to the next. Our second loop is very similar to the first, but with the addition of threading. Now, we are able to increment multiple elements of the array at the same time. However, performance is still stalled by the amount of time incrementing the element takes.

1.1 Array C

Our third loop, on the other hand, had some independent thought put into it. My strategy for increasing performance involved a technique from part one: unrolling. As gone over in lecture, the cache line size for x86 systems is 64 bits, or 16 integers - so of course, I chose to unroll the incrementation to have each thread increment 16 elements at the same time. This takes advantage of ILP, as each incrementation is independent from the previous and following ones. Now, if we were doing any operation that required modification of other elements of the array, this wouldn't work; but that's what part 2 was about. The improvement in performance can be seen clearly in Figure 1 (next page), which illustrates the relative speedup of round robin(blue) and unrolling(green), relative to the sequential, single-threaded computation. Values on the graph are averages of 3 runs of the program at each thread count.

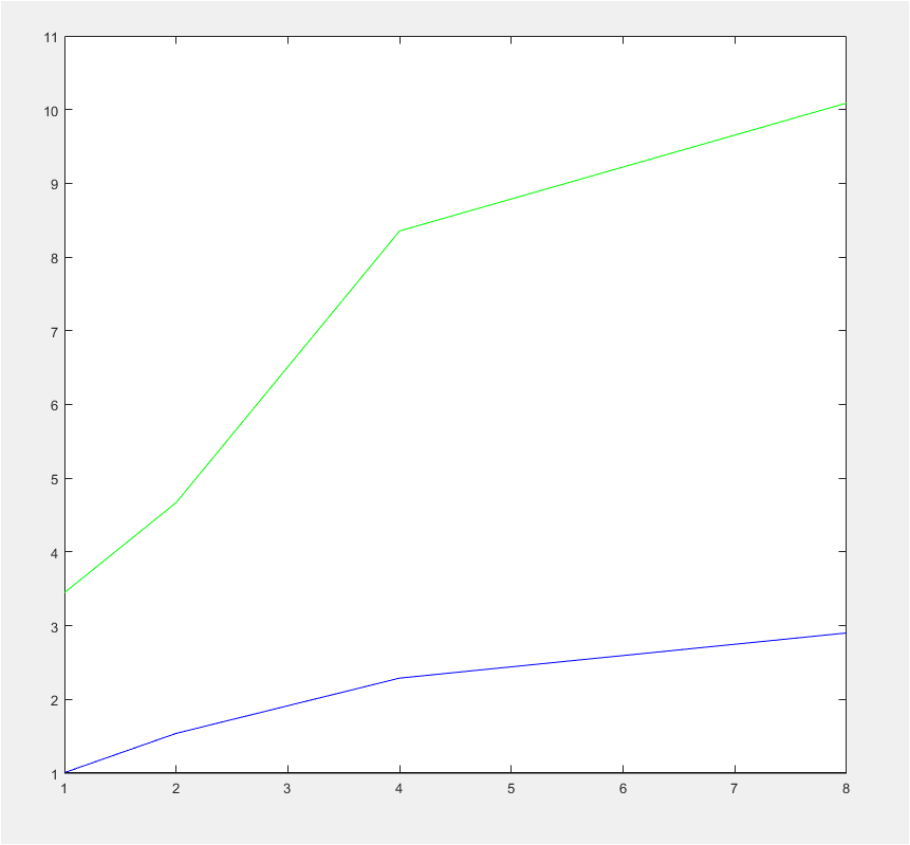

Figure 1: Figure 1

2 Analysis

As evidenced, the speedup from unrolling and multi-threading is significantly higher than just multi-threading. The round-robin speedups increase slower as the amount of threads increase, seemingly capping at slightly under 3. This is likely because the computations themselves are taking the majority of the time (i.e bottlenecking the speedups). This helps with explaining why the unrolling method does not see this slow-down - the improvements are related less to multi threading, and more to actually improving the speed of the computation by doing multiple at the same time. Following this, we can see that we pass speedups of 10(!)x as we hit 8 threads. One interesting thing to note is that the unrolling method has a significant amount of variance as compared to round robin. Averages taken for 4 threads ranged from (approx) 5.5x-13x. This can be correlated with the additional computations done simultaneously - the variation can be attributed to variations in the way the compiler executes the code, as well as local machine variables.