HPC Assignment 2 - Flat World Climate

Peter Ukkonen[1]

[1]Affiliation not available

March 12, 2019

1. Profiling

Before I set out to parallelize the code, I did a basic profiling by measuring the time elapsed on the three computational parts of the code: radiation, energy emission, and diffusion. Diffusion was by far the most expensive part, and radiation second. This means the diffusion function should be the prime focus.

2. Parallelization

Using OpenMP, making this code run in parallel (to at least some degree) is trivial. The longitude-latitude points (which are actually atmospheric columns in real climate models) involve independent calculations (with respect to other points) with the exception being the diffuse equation, which depends on neighbouring grid cells. However, even this part can be made run in parallel using openMP, since the calculations themselves are independent (do not require computations from neighbouring grid cells to finish).

I did parallelized the diffuse part simply by putting an *#omp parallel for* before the middle loop, and an *#omp parallel simd* before the inner loop. The simd command tells the compiler that it's safe to vectorize the inner loop (again, due to no computational dependency).

I spent a lot of time trying different things to make the diffuse code run faster, mainly: getting rid of the (i-1) and (i+i) iterations **since these are not contiguous in memor**y. I did this by copying the "up" and "down" as *vectors* before the innermost loop. However, this did not improve performance, instead it decreased slightly, probably since copying from memory is expensive.

In the radiation and energy emission part of the code, I used a simple parallel for above the outermost loop, since these were simple totally independent loop iterations.

3. Results

Figures 1 and 2 show respectively the absolute and relative performance in running time. While the performance of the parallel code is "OK" relative to the sequential (2 - 2.6 factor of speedup depending on the problem size), the small differences between using NUM_THREADS=4 and NUM_THREADS=10 shows that the scaling is really bad. Probably there are some false sharing issues with the memory, which I did not know to fix. A brief test with the "copy up and down as vectors" strategy I tried in the diffuse code suggested that the scaling was better even though the code was overall slightly slower due to memory being

copied. This again makes false sharing the likely culprit. False sharing means that the different parallel workers are sloshing back and forth on the cache line.

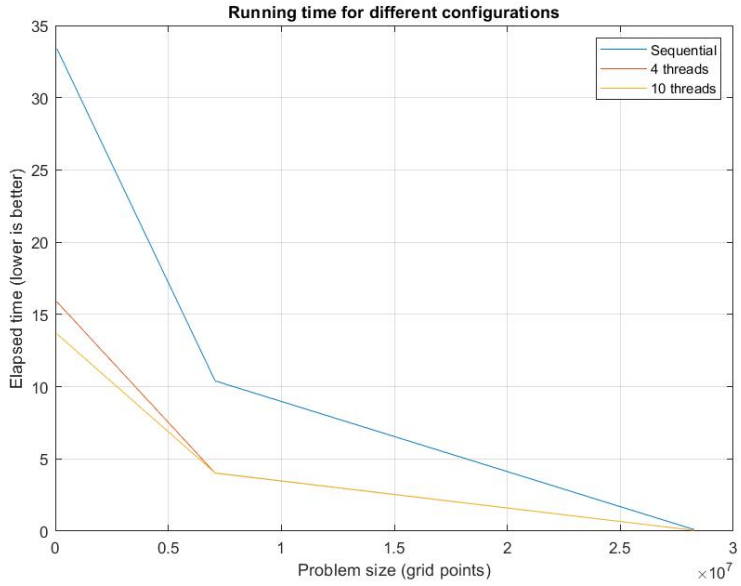

Figure 1: This is a caption

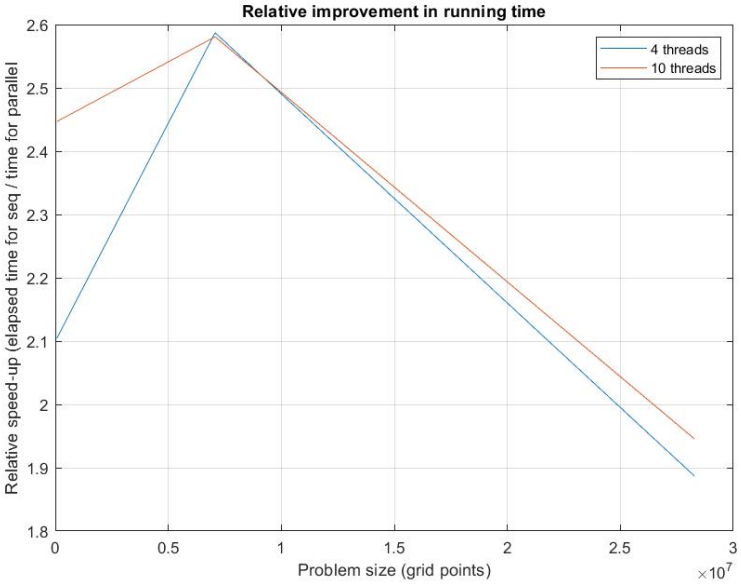

Figure 2: This is a caption