David Vondran

02/16/2022

Homework 2 Write up

Part 1:

For part 1 of this assignment, I was able to effectively get the effective time of execution down in incremental steps from the initial execution time of around 20 seconds, to the final run time of around .36 seconds. The first improvement in run time was gained by changing the compiler flags, including the terms “-O3 -mavx -std=c++11” in the compiler to help optimize the run time. The next large improvement that helped me gain a significant amount of time was to remove the calculations for val outside of the nested loops and give them their own loop to compute all values. As it is not dependent on any of the internal values in the loops, and only on the size of the given input, we can precompute all the values to be referenced later so that we do not need to compute them one by one in the nested loops, which would have been in O(n^2) time as opposed to the O(n) time from my solution. I also decided to move some of the internal calculations in the nested loops to a new function which I called “getSquaredVal”, which was purely done to make the code easier to understand while I was attempting to make it work. This could be undone I believe with little to no change in effective run time. Finally, the last improvement in run time came from loop unrolling the outer j loop, as this meant that I could easily save iterations on the j loop while still hitting every “val” value of I, which it is dependent on. I decided to unroll by a factor of 5 which I felt effectively reduced the time well and fit into the given size of the input. I also had to modify the comparison method located in the test file to accept values within a precision of 5 decimal places, as the floating-point arithmetic was causing rounding errors and causing outputs of the same value to not be recognized. This was accomplished by multiplying the value by 100000, then taking the floor function of the value and rounding down, and then dividing again by 100000. This resulted in values truncated at 5 decimals and able to be compared correctly.

Part 2:

For this part, I had a large amount of difficulty coming up with ways in which to optimize this code. I unfortunately was very short on time, however, I felt that I was close to being able to make significant breakthroughs in this problem. One thing which I was attempting to do was to compute the value of A[i*N+k] at least partially outside of the loops and then access the value of this computation inside, meaning that it would not have to be computed every single time, and could just be looked up from the array with what would be constant time. However, the code and computation for this became very messy, with me not being able to ultimately reach an answer that was effective and fully efficient within the time of the deadline. One other idea that I had was to combine certain values together similar to how I did in part 1, such as the i*N in the accessing of the A matrix, however, this interfered with the effectiveness of my loop unrolling and I found that it was more efficient to have the unrolling occur on its own rather than try to get them to work together with the limited amount of time I had left. I also wanted to try to unroll the j loop as well, which I believe would have also had a very significant effect on the execution time of the algorithm, however with the further unrolling of the I loop it quickly became a mess of code and was difficult to work around, so I wanted to find more modular solutions before I broke into further unrolling. Ultimately, I got the time down from the initial ~76 seconds to around 46 seconds, a time decrease of 30 seconds, which is very significant. While not as far as I believe I can take this algorithm, I believe the time savings to be worthwhile given the smallish nature of the changes to the code. I should like to attempt to further improve the algorithm when more time is available for a potentially increased grade should you agree.