William Garner

Programming Homework 1

My goal for this homework assignment was to let the compiler do as much of the optimization for me as possible. My thought was that the compiler is really smart and better than me at optimization, so I just tried to help it do the optimization that was needed. I did start off with modifying the code by changing the loop order from i-j-k to j-k-i and getting rid of the running sum that the original code kept track of. Changing this loop order allowed me to calculate partial answers for elements of the C matrix and eliminate unnecessarily declared local variables. Doing these optimizations increased my average peak performance from about 4% to about 7%. Next, I focused on helping the compiler optimize automatically. I switched from using the gcc compiler to using the Intel C compiler. Then, I looked up an alphabetical list of all the flags I could use with the Intel compiler to try to make my code more efficient. I started with using -qopt-report5 to create a report of how the compiler was trying to optimize my code. This allowed me to see what I still needed it to do, and I looked for flags that would accomplish those things. I used -funroll-loops to tell the compiler that the loops needed to be unrolled. I also tried using -unroll-aggressive, but I got better performance with -funroll-loops. Through the report I also discovered that unrolling the loops helped the compiler to vectorize the math inside the loop as well as use fused multiply-add instructions. This brought my average peak performance up to about 15%. I also added the flag -march=haswell to let the compiler know that it was working on a Haswell machine. This increased my performance a little more to about 17%.

After I had used compiler flags and directives to improve my efficiency, I went back to trying to improve the code. I transposed the A matrix which caused me to have to switch the

loop order again to j-i-k. This increased my performance to about 23%. I added blocking to the transpose, and it brought my performance up to about 26%. When I tried adding a level of blocking to the actual matrix multiplication, my performance dropped down to about 4%, so I took that blocking back out of the code. Next, I added the restrict keyword to the incoming argument for the C matrix. This gave my code a large performance boost to about 32%. Finally, I added the flag -ipo to my list of compiler flags. This gave my code its last large performance boost to about 37% average peak performance. After some last minute tweaks I was able to get a peak performance around 42.7%

I did not get to test my code on a different machine to see the change in performance because I was not able to get access to another computer that had the Intel C compiler.