COE 379L: Homework 10

Ashton Cole

April 10, 2024

1 Introduction

In this assignment, the impact of compiler choice and optimization level is measured for runtime of a simple nested for loop conducting summation and multiplication operations on vectors. Analysis is conducted on both serial and parallelized version of the code.

2 Methodology

The following loop is timed within `HW10.c`. The program then prints the serial and parallel runtimes for the user.

```
for (int iloop = 0; iloop < nloops; iloop++) {
        for (int i = 0; i < vectorsize; i++) {
                outvec[i] += invec[i] * loopcoeff[iloop];
        }
}
```

Essentially, a new vector of size `vectorsize` is created, where the ith entry is the sum of the ith entry of another vector, each time multiplied by coefficients from a vector of size `nloops`. It would be more efficient to use the associative rule and assign the product of the ith entry and the sum of the loop coefficients vector, but this program is intended primarily to test performance.

The loops can be parallelized with OpenMP, but only the inner loop should be made parallel. This is because otherwise there is a race condition, where iterations with a different `iloop` but the same `i` attempt to read and then replace the same output vector entry at the same time.

The script is compiled and run on a single development node on the Lonestar 6 supercomputer. Three C compilers are used: Intel 19.1.1, Intel 24.1, and GNU 11.2.0. Each is tested for the optimization levels `-O0`, `-O1`, `-O2`, and `-O3`, with higher numbers representing more optimizatoins, and ideally, a faster runtime. It is worth noting that between the Intel 19 and 24 compilers, the C compilation command changed from `icc` to `icx`. The executables are run using 16 threads and a `vectorsize` of 1,000,000.

3 Results

Table 1: Runtimes (s) for various compiler optimizations in serial and parallel on a Lonestar 6 node.

Optimization	Intel 19.1.1		Intel 24.1		GNU 11.2.0	
	Serial	Parallel	Serial	Parallel	Serial	Parallel
`-O0`	1.27575	0.12185	0.98183	0.10101	1.00503	0.06986
`-O1`	0.19651	0.02686	0.19668	0.02678	0.29993	0.02349
`-O2`	0.05377	0.01913	0.13139	0.01868	0.29457	0.02156
`-O3`	0.05372	0.01516	0.13204	0.01958	0.07263	0.01541

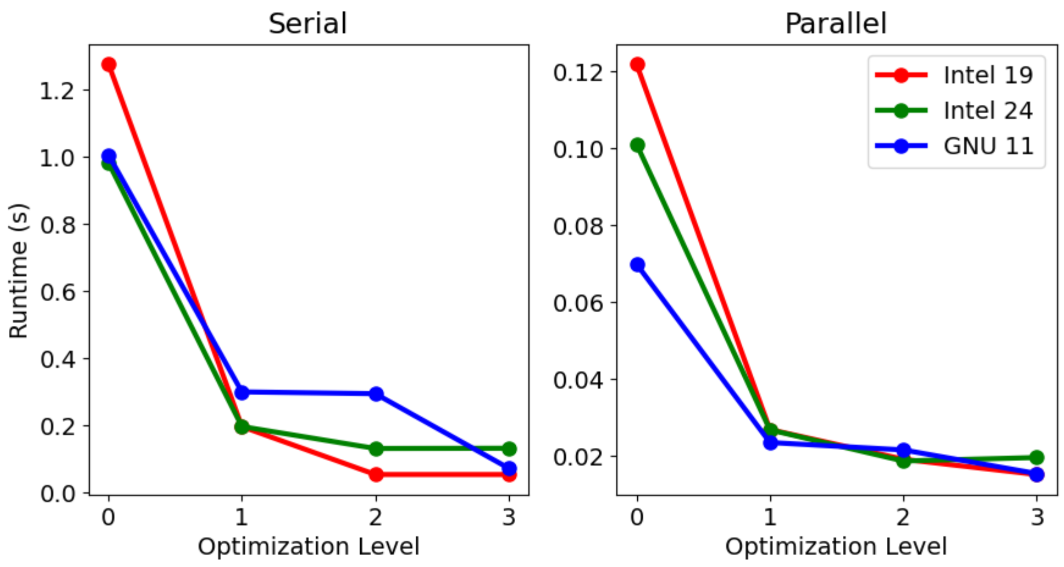

Figure 1: Runtimes for various compiler optimizations in serial and parallel on a Lonestar 6 node.

4 Conclusions

A couple of conclusions may be drawn. Firstly, parallelization into 16 threads decreases execution time tenfold, i.e. the speedup is not perfectly efficient. Each optimization level offers approximately the same relative level of improvement. None of the compilers stand out as significantly advantageous in this case. Finally, the first optimization level `O1` offers a significant advantage, but performance improvements past that are relatively limited. Higher optimization levels tend to take longer to compile, so this is a reminder that the highest optimization may not be necessary or appropriate.