Difference in Execution Time for Sequential and Single-Threaded Parallel Vector Summations

Jack Rowe (jbr2826, jackrowe)

April 2024

1 Overview

This algorithm is programmed in C++ using OpenMP. The goal is to determine the execution time for various numbers of vector sizes. This was done for a sequential loop and a single-threaded parallel loop, and times were compared. The project features one executable (`vectorsum`), however multiple different compilers were used in this exercise (`intel/19`, `gcc`, `intel/24`). Additionally, a shell script was used to collect and output the data, and a Python script using Matplotlib was used to create organized visualizations. Output text files used for the visualizations have been saved in this repository.

2 intel/19

This is the default compiler on LS6. Here, there is a consistent positive difference between the sequential and STP (single-threaded parallel) execution times. Looking into the compiler report, it is clear that the following 2D loop was reversed in order for both sections.

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

This provides speedup for both sections, as it allows the vector to be loaded once into a higher cache instead of loading pieces of it many times from main memory. The STP section benefits from an additional speedup by having all of this data allocated in a cache that is specific and close to the running thread.

$$SignedDiff. = Seq.Time - Para.Time$$

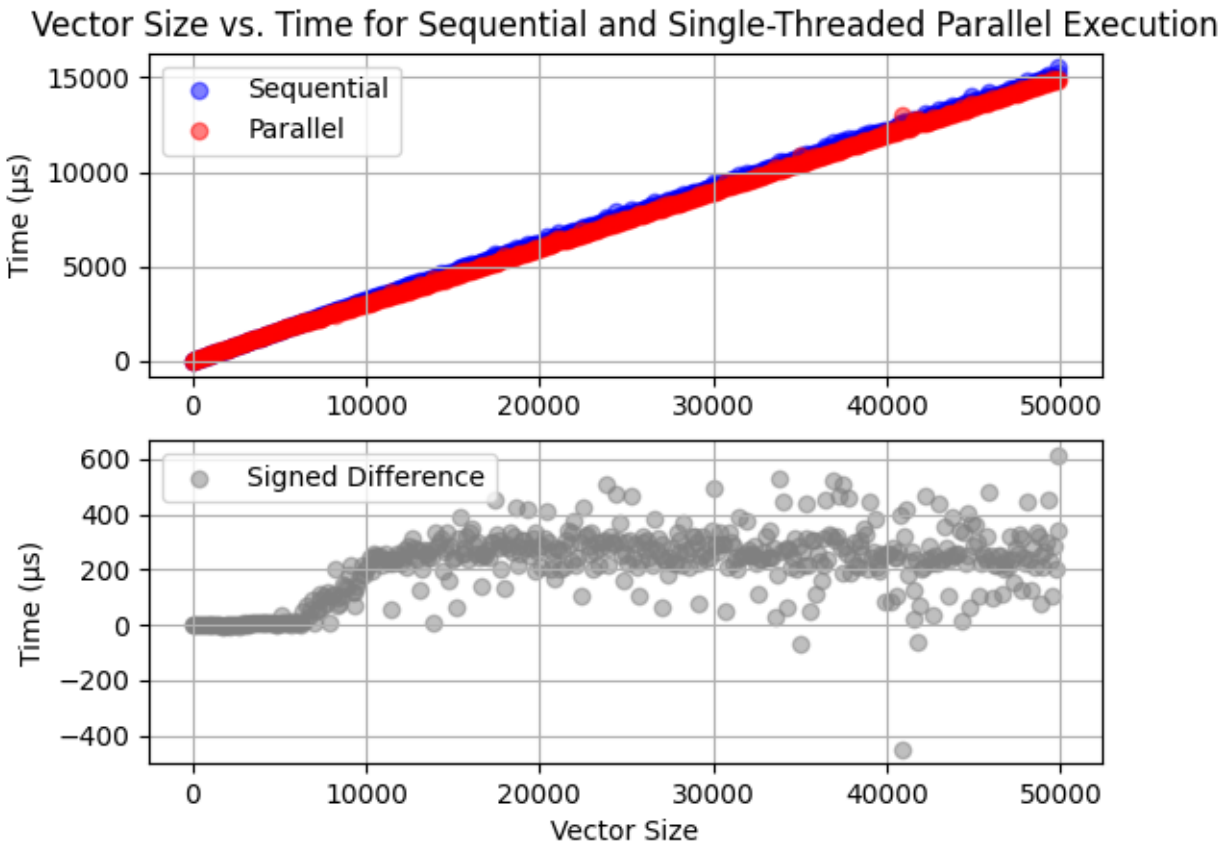

Figure 1: intel/19 Compiled Times

3 gcc

This compiler provides similar time data to `intel/19`, however it is noticeably more compact but with a few unpredictable sections. These are visible as squiggles in the top graph (red and blue), and as highly scattered areas in the second graph.

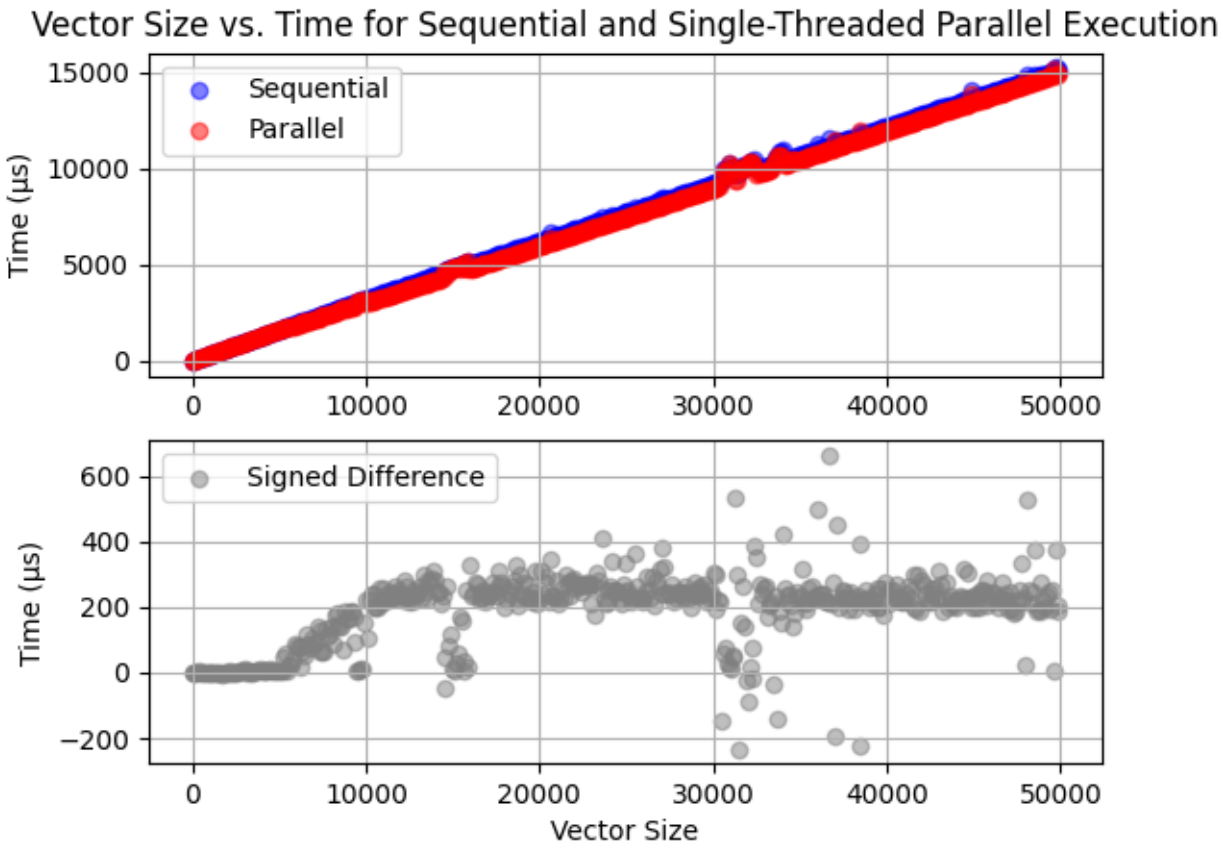

Figure 2: gcc Compiled Times

4 intel/24

Notably, this compiler differs from `intel/19` in that it does not reverse the order of the loop. However, there is still a larger speedup gained from this compiler. It is extremely noticeable on the first graph as a gap between the red and blue scatter plots. It also has a much more compact distribution in the signed difference graph.

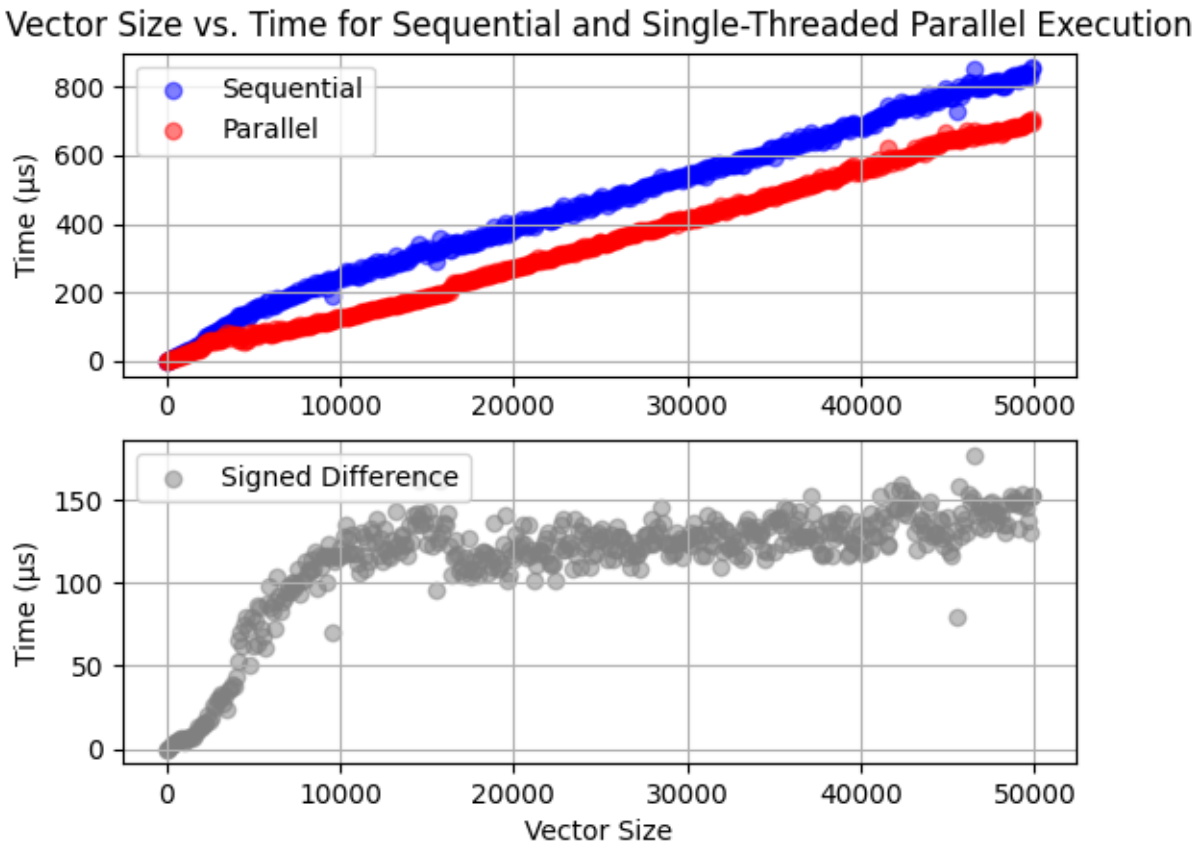

Figure 3: intel/24 Compiled Times

5 Optimization Level

Using `intel/24`, the following three figures were produced with optimization level 0, 2, and 3 respectively (default is level 2, which is the same figure seen in Figure 3, but has been copied here for convenience).

5.1 Level 0

Level 0 is mostly defined by an increasingly random distribution of signed difference between sequential and parallel execution times. As the compiler suggests, it appears that no optimizations were made to the code, and so any time difference should be completely random as the code and cache/memory structure should be identical.

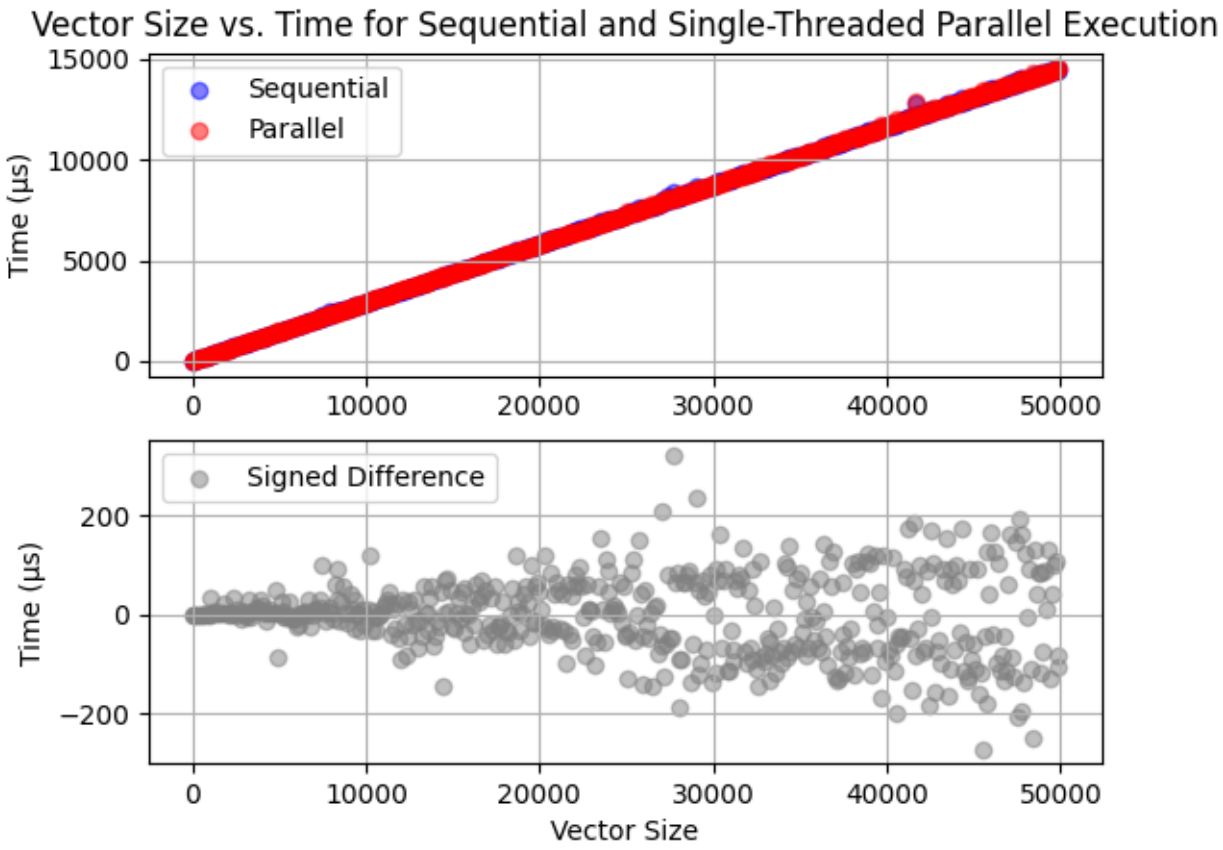

Figure 4: intel/24 Compiled Times (Level 0)

5.2 Level 2

See the `intel/24` section for details.

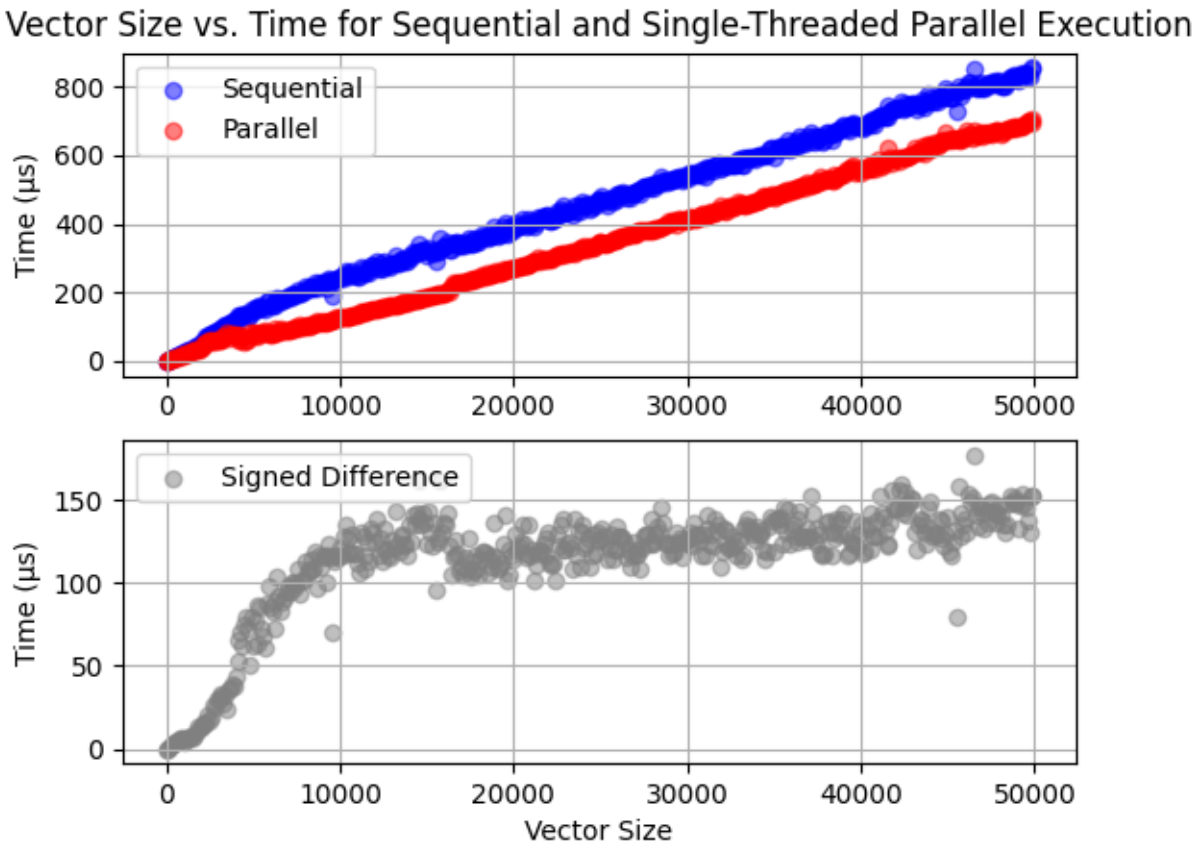

Figure 5: intel/24 Compiled Times (Level 2)

5.3 Level 3

This is the maximum optimization level that was able to be obtained with this compiler. The graph shows some interesting data, with the signed difference graph crossing the x-axis twice over the range of vector sizes. The optimizations here do not provide an incredible overall speedup compared to level 2, and there is not much difference in magnitude between the sequential and parallel versions.

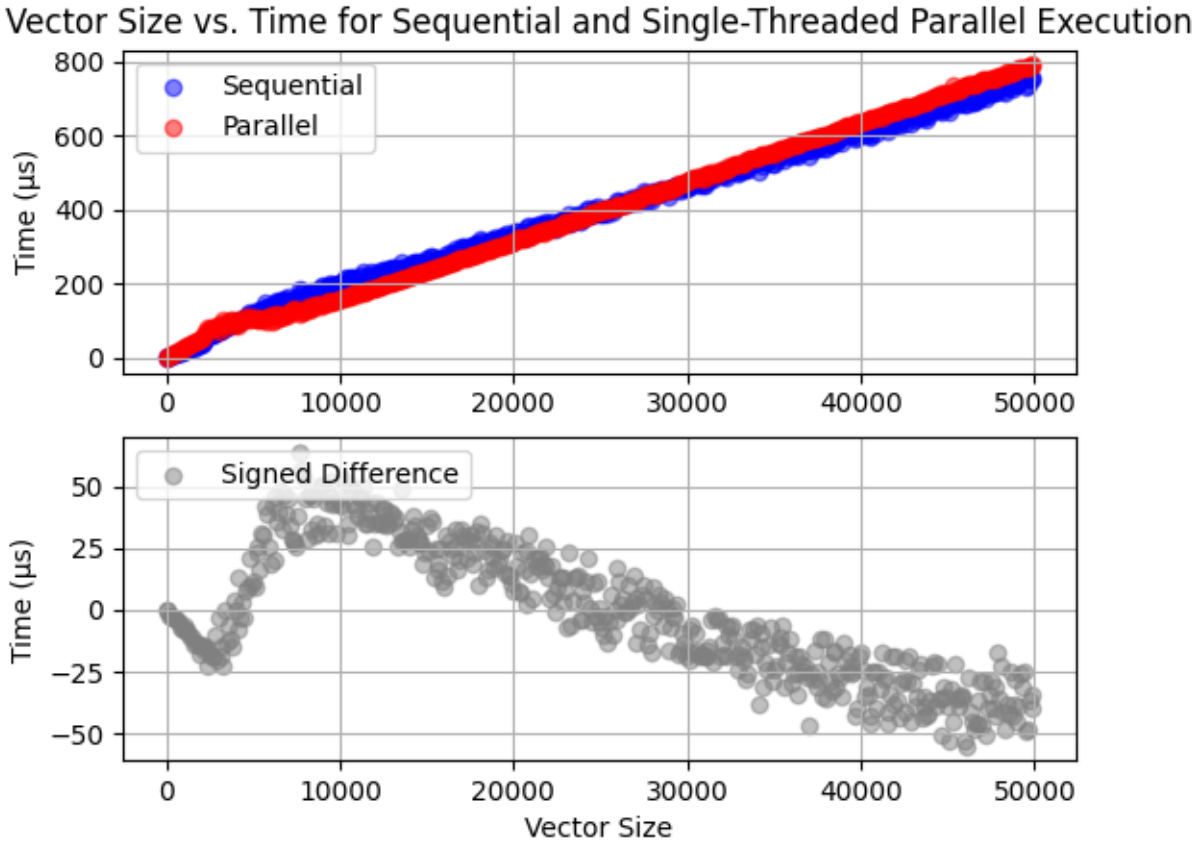

Figure 6: intel/24 Compiled Times (Level 3)