Benchmark Comparison for Different Compilers

Comparison of Different Compilers

Anurag Aggarwal (2018SIY7504)

Run Details

Following configuration was used to run the benchmarks:

- Architecture: x86_64
- Processor:
  - Model: Intel(R) Core(TM) i7-4790S CPU @ 3.20GHz
  - L1d cache: 32K
  - L1i cache: 32K
  - L2 cache: 256K
  - L3 cache: 8192K
  - RAM: 8 GB
- Compilers Used:
  - Clang: 3.9.0
  - GCC: 7.2.1
  - ICC: 17.0.1.132
- Runs Not Done:
  - ICC 32: Getting A mismatch in the Benchmarks, due to which the benchmarks did not run.

Result

Below are the individual results with their analysis.

500.perlbench_r (Figure 1):

In the benchmark, the best result was achieved by ICC, 64 Bit, with O2 optimization level. Although with O3 the performance is worse than GCC 64 Bit O2 & O3. GCC consistently outperformance Clang on all optimization levels.

32 Bit Performance is on a bit lower side with respect to corresponding 64-bit compilers & optimization levels. This can be attributed to being able to utilize the RAM properly.

The contrast between O0 & Other optimization levels is very visible. So, optimizations can help get a lot of performance improvements.

One stark behavior is the comparison between ICC 64 Bit O3 and O2. With O3 we see a approx. 10% degradation in performance w.r.t. O2. Considering the following except from Intel Optimization Guide, the performance degradation is expected as this benchmark does not heavily rely on Floating point calculations.

Using the O3 optimizations may not cause higher performance unless loop and memory access transformations take place. The optimizations may slow down code in some cases compared to O2 optimizations. The O3 option is recommended for applications that have loops that heavily use floating-point calculations and process large data sets

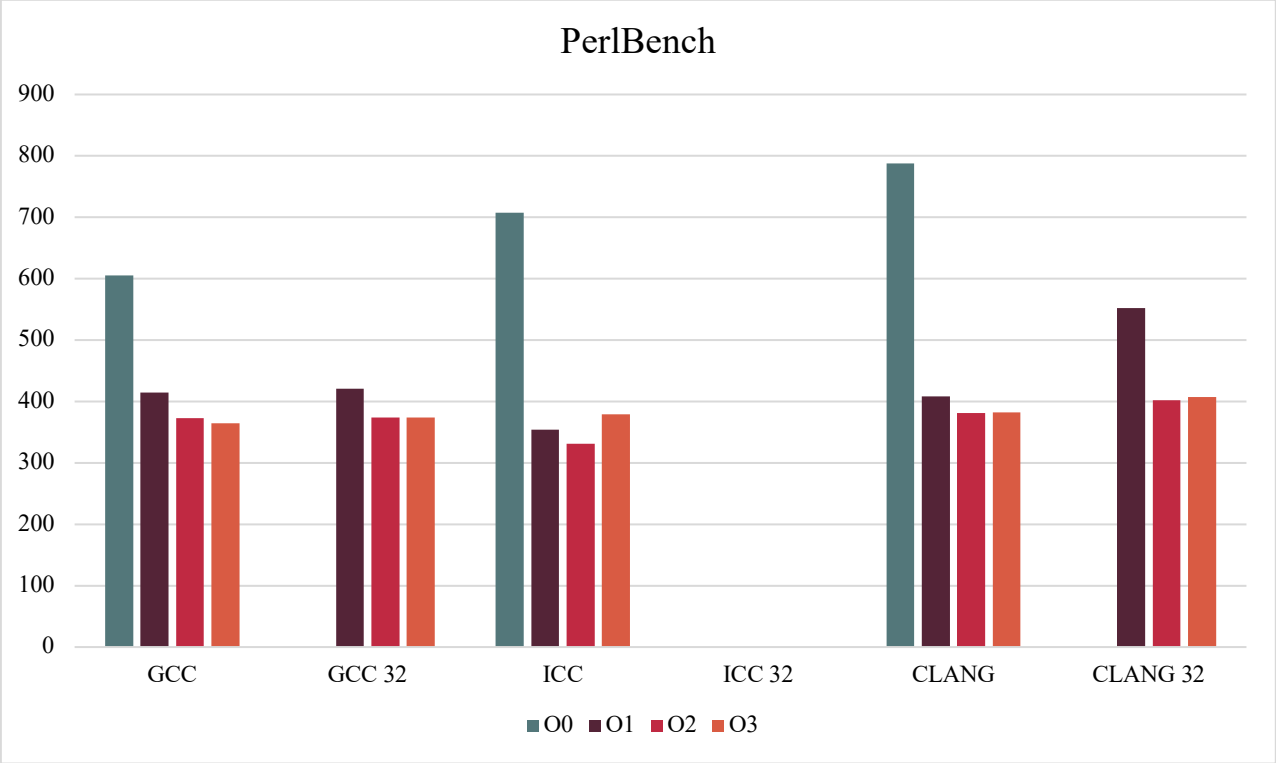

Figure 1

500.gcc_r (Figure 2):

The performance of different optimizations level in 500.gcc_r and 500_perlbench_r is the same. The only differences are:

- GCC 64 Bit O3 is marginally better than ICC 64 Bit O2. The difference is less than 3%.

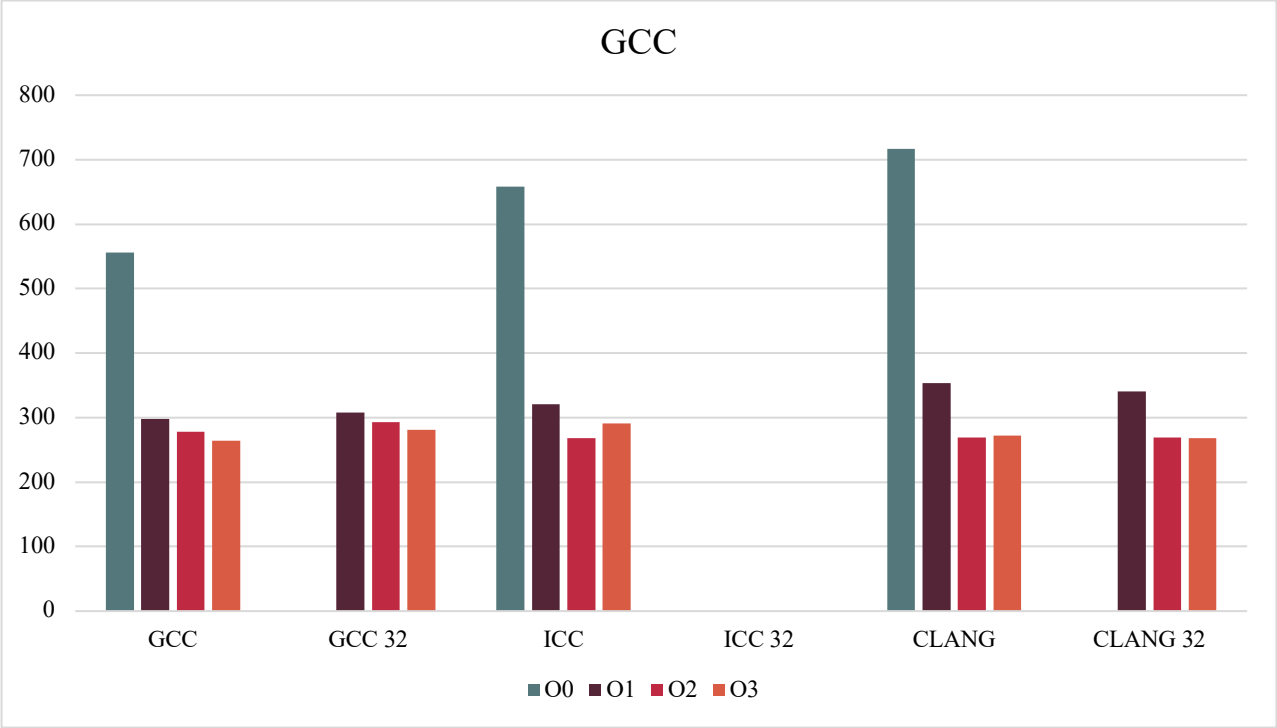

Figure 2

500.mcf_r (Figure 3):

The results are almost similar to Perlbench benchmark, the only difference between that ICC O2 & O3 have better runtime than corresponding GCC levels.

Here we see that Clang performance does not improve with O2 and O3, but slightly decreases. Also, the difference between O0 and O1 is more than 300%. While GCC and ICC only have at max 50% increase in runtime between O0 and O1.

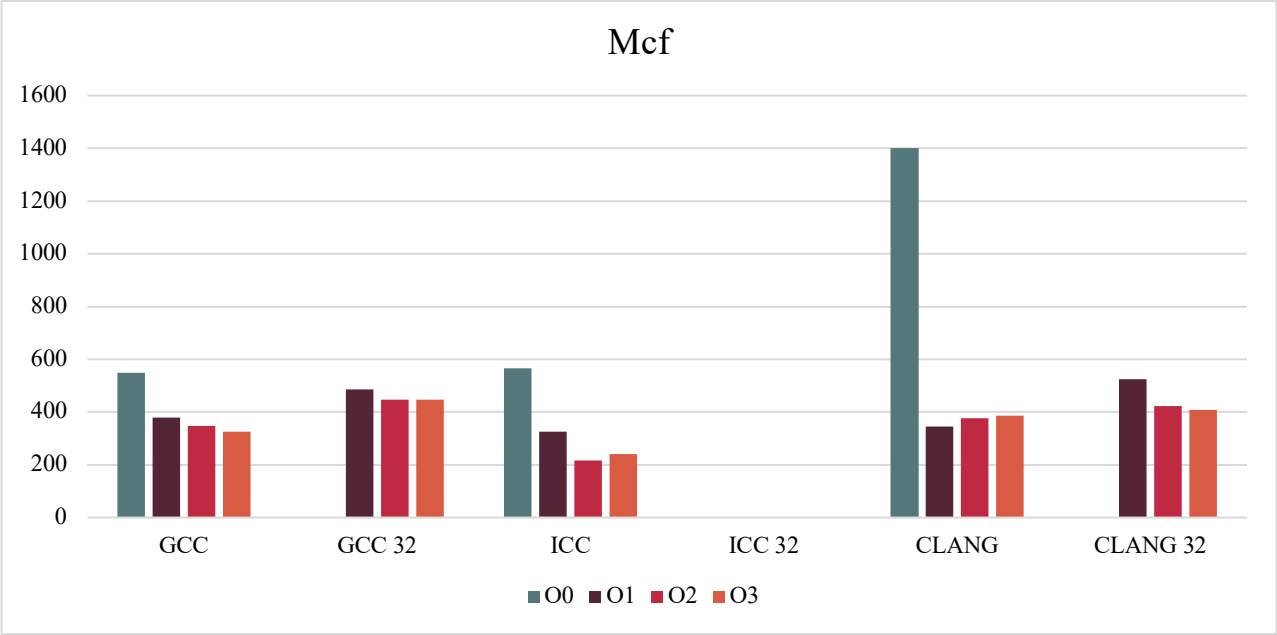

Figure 3

520.onmetpp_r (Figure 4):

This benchmark shows performance improvement between ICC O2 & O3 levels, this could be because this benchmark simulates a 10 Gb Ethernet network, so it has to handle large data set, in which O3 performance should improve as per Intel Optimization guide.

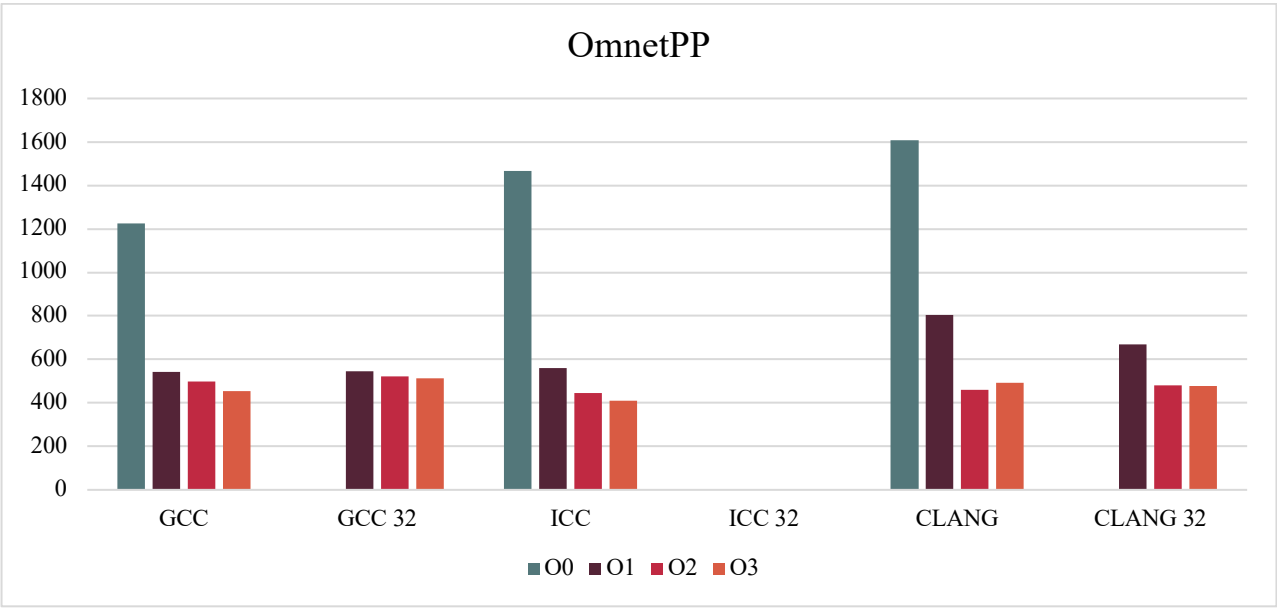

Figure 4

523.xalancbmk_r (Figure 5):

The benchmark has biggest difference between O0 & O1 for all the compilers. The difference ranges between 300% - 400%. Here again ICC O2, outperforms all the benchmarks.

The benchmark converts XML & XSL to HTML.

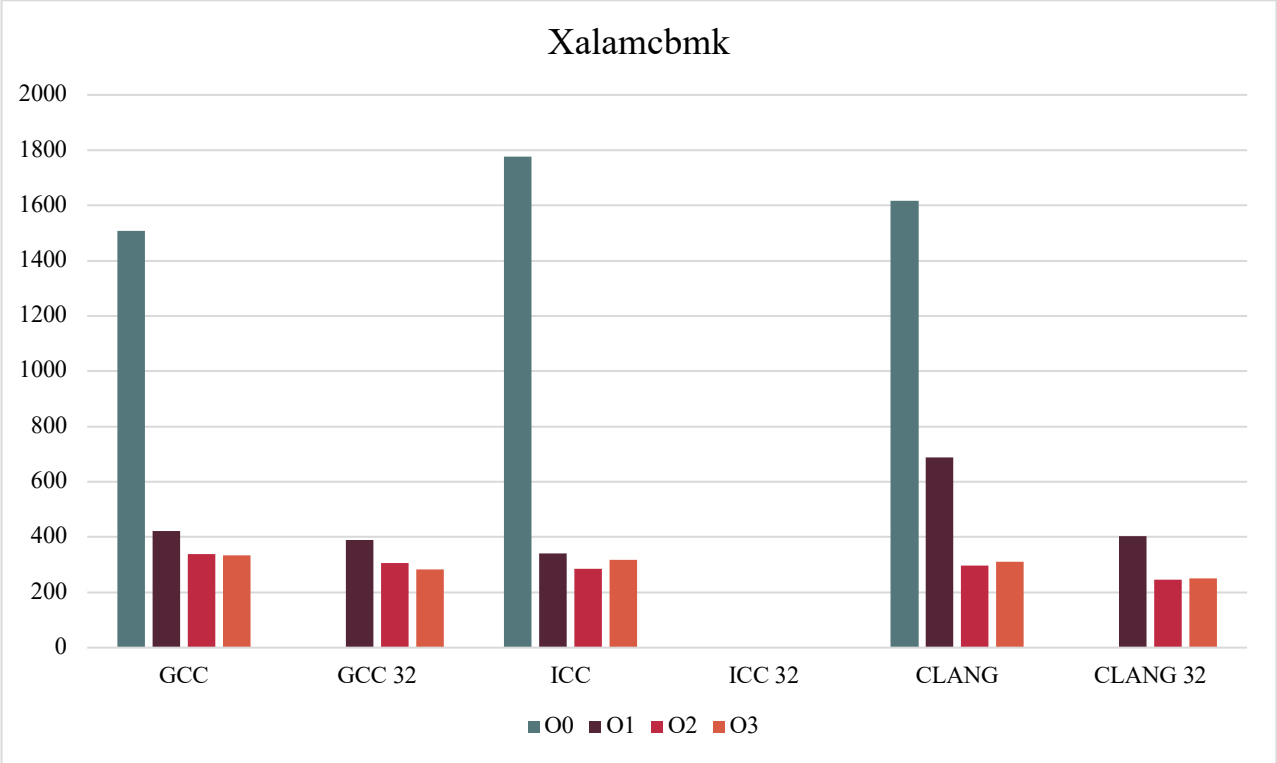

Figure 5

525.x264_r (Figure 6):

The most striking feature about this benchmark is the stark difference ICC O2 & O3 vs Rest of the benchmarks. There is more than 100% increase in runtime with corresponding GCC optimization levels. This is a video conversion utility and handles large amount of data. Due to which we see ICC O3 shows slight improvement over O2 flag.

Clang also outperforms GCC by about 33%, so It seems that GCC is missing some optimizations which are in part captured by Clang and fully by ICC.

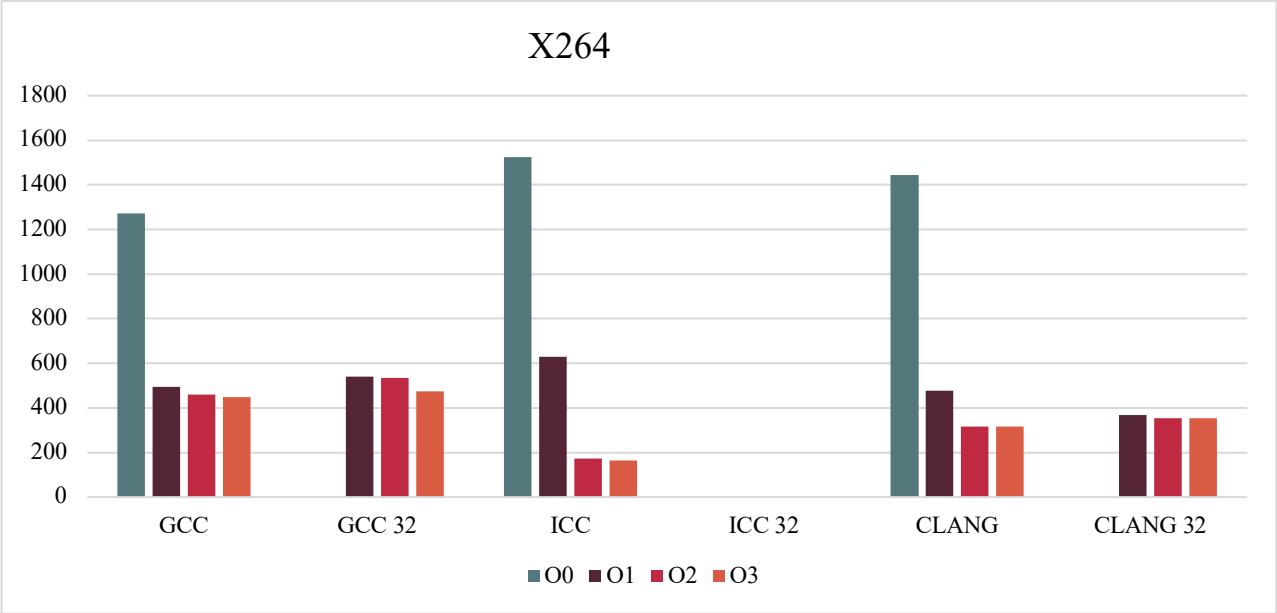

Figure 6.

531.deepsjeng_r (Figure 7):

In this benchmark Clang outperformance GCC by approx. 10%, but again ICC O2 comes out as winner. In this benchmark we have a stark difference between 32 Bit & 64 Bit runs. It could be because of the fact that this benchmark uses Alpha Beta tree searching, which is a Memory intensive algorithm, so presence of more RAM in 64 Bit should be helpful in this benchmark.

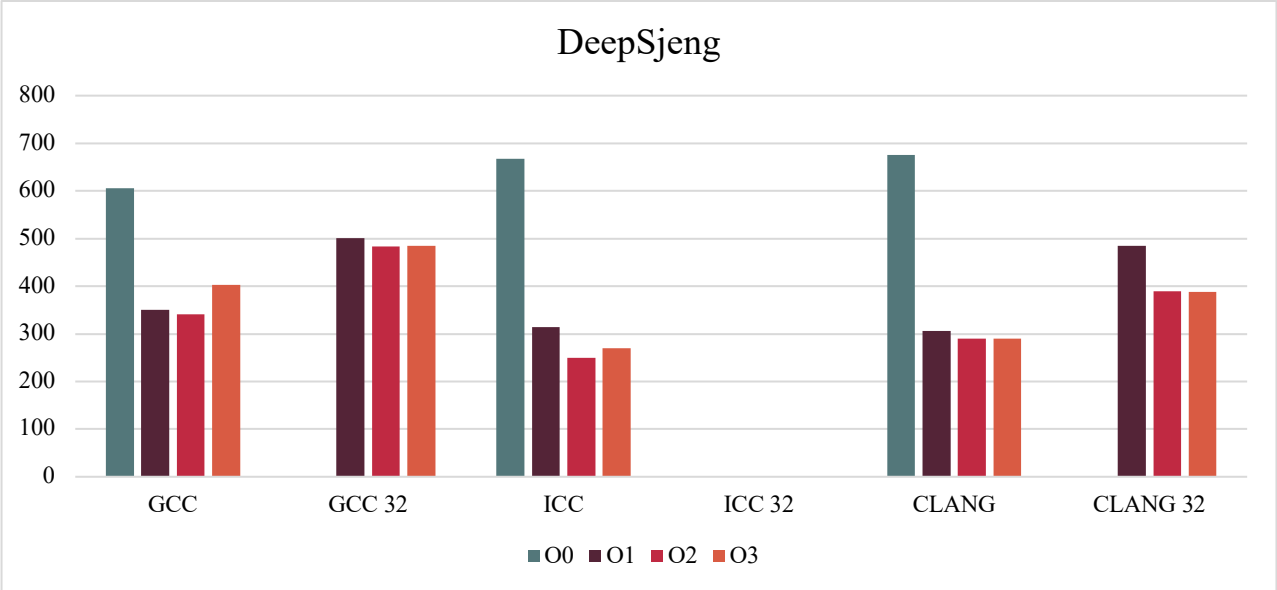

Figure 7

541.leela_r (Figure 8):

The results in this benchmark are similar to the trend that is visible in all the visible benchmarks, except some outliers, ICC O2 is the best performer.

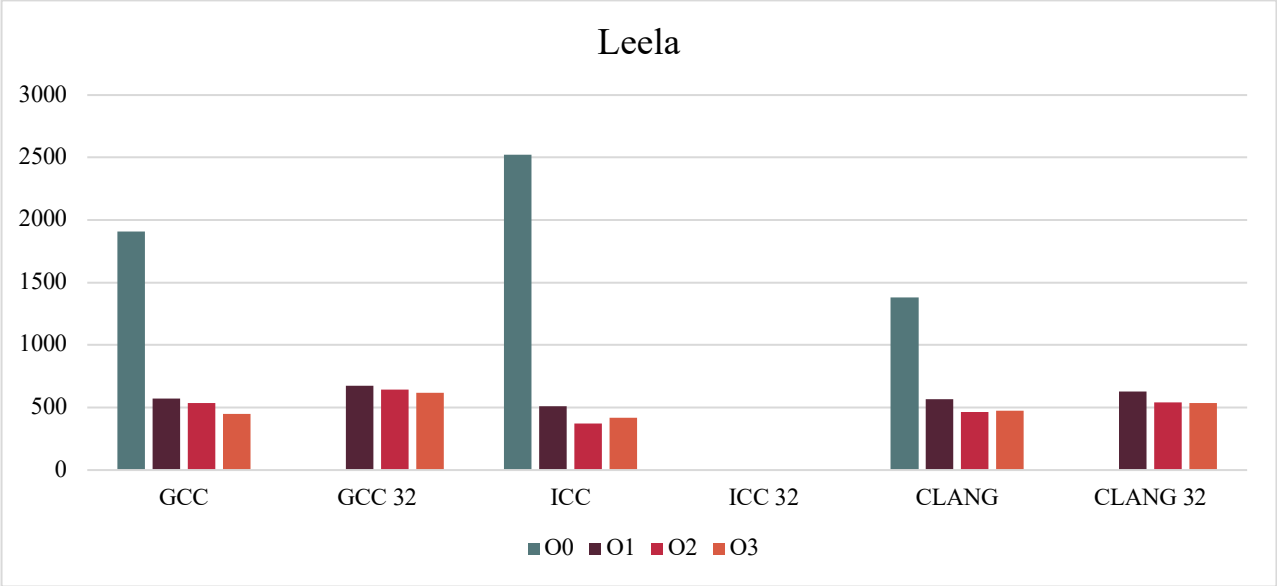

Figure 8

548.exchange2_r (Figure 9):

The results in this benchmark are again following the general trend with ICC O2 being the best performer. The only outlier is that Clang O1 performs better than O2 and O3, Also GCC O2 performs poorly compare to both O1 & O3. It looks like GCC & Clang optimizations heuristics are not helping the runtime but only slowing the process.

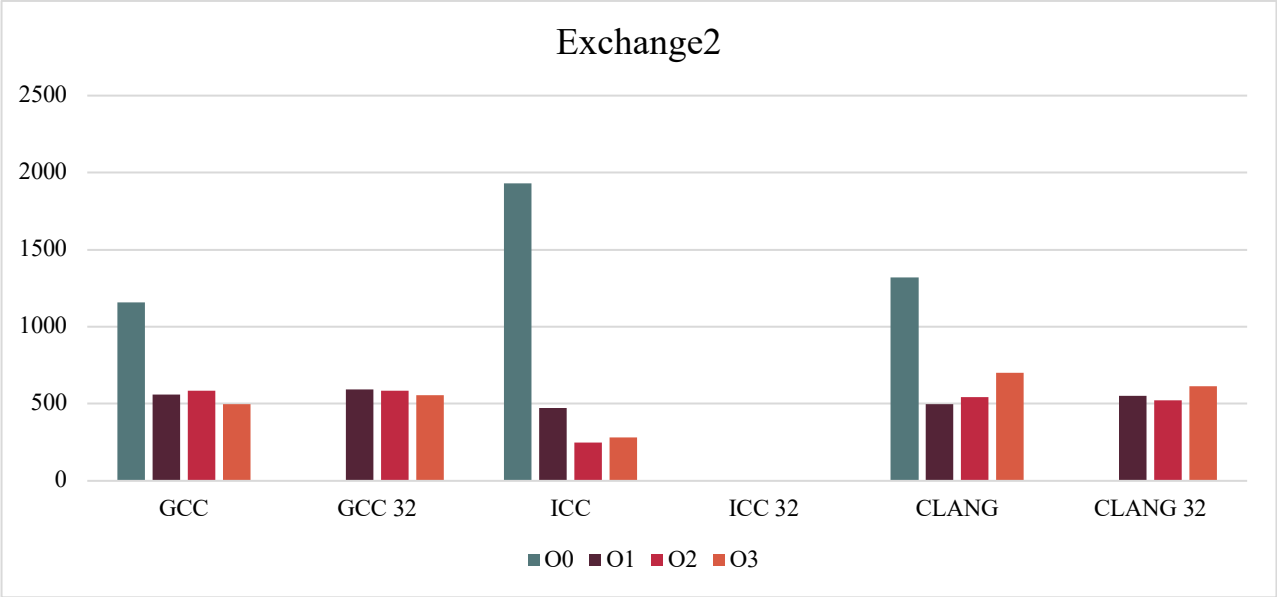

Figure 9

557.xz_r (Figure 9):

The results in this benchmark again follow the general trend, with ICC O2 being the clear winner.

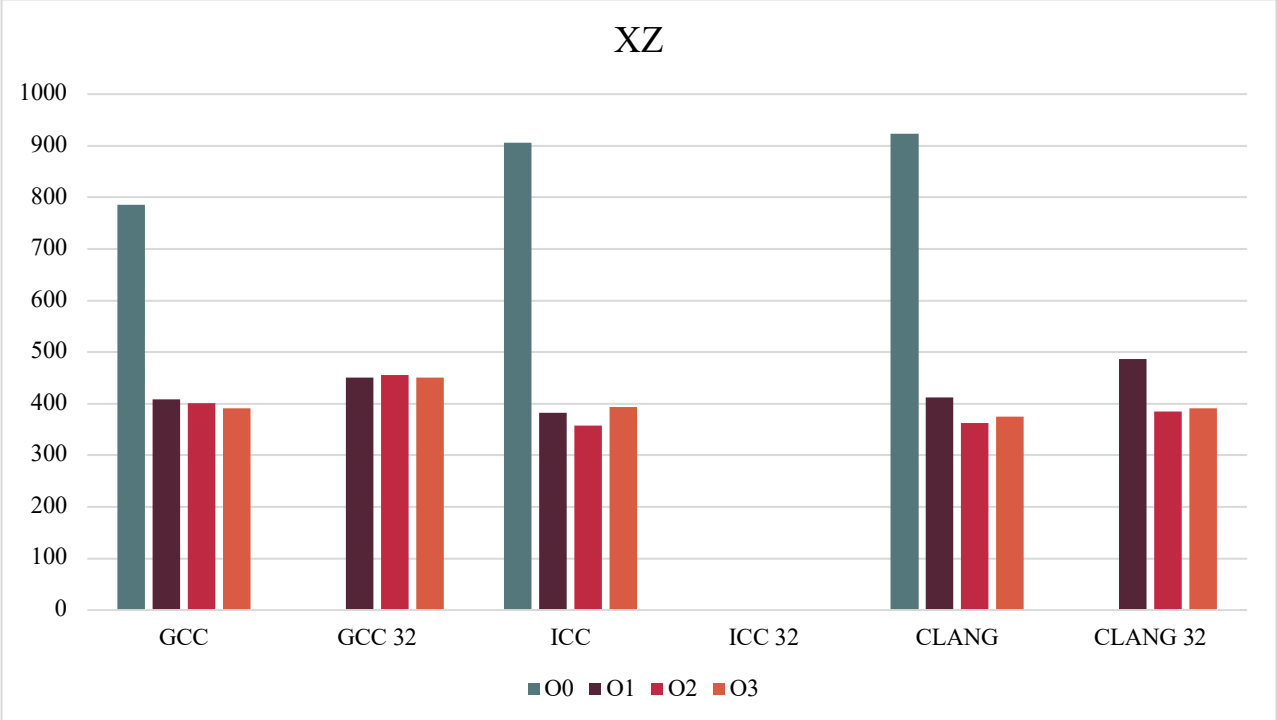

Figure 10

Conclusions

Looking at the runs of all the 10 benchmarks, one thing that is clear is that ICC O2 provides best runtime optimizations, ICC O3 is useful when we have Floating point operations and Large Memory Handling.

GCC consistently outperforms Clang in most of the runs, except for benchmarks like 525.x264_r, where GCC seems to miss a lot of optimization some of which are caught by Clang, but most of them are caught of ICC.

The difference between O0 & other optimizations levels is very considerable, this is true for all the compilers.

Lastly, as expected all the benchmarks performs better in 64 Bit runs than 32 Bit runs, this could be easily be attributed to ability to use more RAM.