COL729: Lab 0 - Benchmarking

Prashant Agrawal (2018CSZ8011)

Hardware Configuration

CPU: Intel Core i7-7700 @ 3.60 GHz
CPU cores: 4 (siblings: 8)
Architecture: x86_64
Memory: 15.551 GB
Storage: 901 GB

Software Configuration

OS: Ubuntu 18.04 LTS
Compilers: Clang (v3.9.0), LLVM (v6.0), Flang (v6.0.1), GCC (v7.3.0), ICC (v19.0.1)
Benchmark: SPEC CPU2017 Integer rate
SPEC tuning: Base
Copies: 4

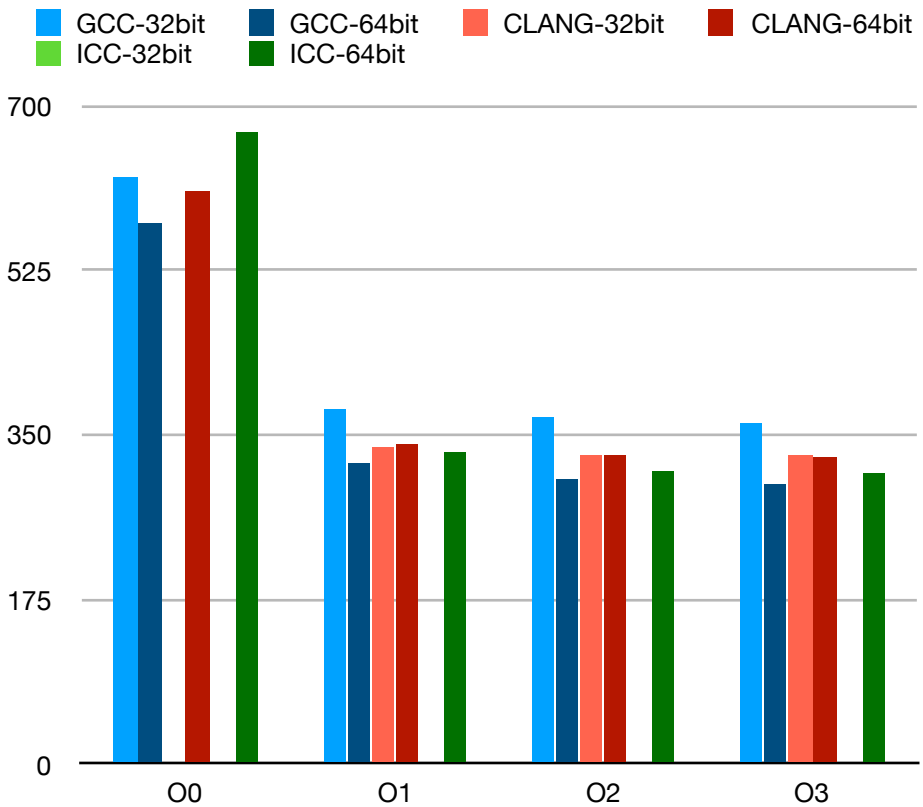
Additional Notes

No -march flags were specified.

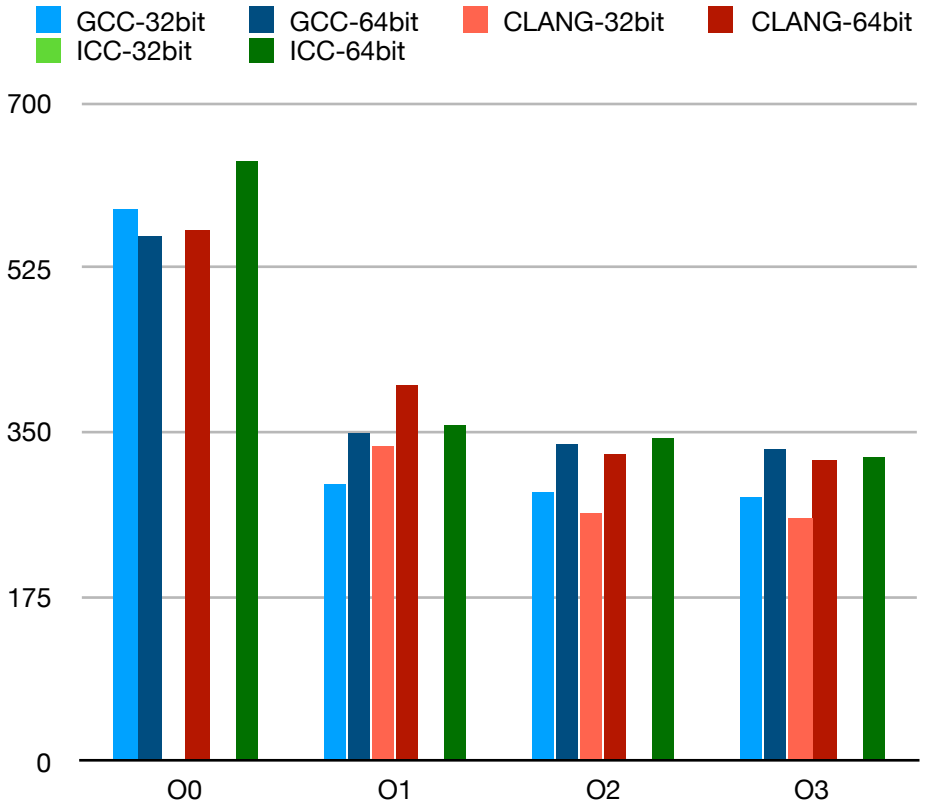
Unable to run the following configurations:

- **Benchmarks on ICC 32-bit:** I was initially facing compilation and runtime errors with running various benchmarks on ICC 32-bit, but later was able to run it. However, due to the limited time remaining, I couldn't run most of the configurations on ICC 32-bit.
- **548.exchange2_r on Clang 32-bit:** This benchmark is written in Fortran, and the only supported Fortran compiler on LLVM-6.0, Flang, is not supported to work on 32-bit.
- **Unoptimised benchmarks on Clang 32-bit:** These benchmarks couldn't be finished in time.

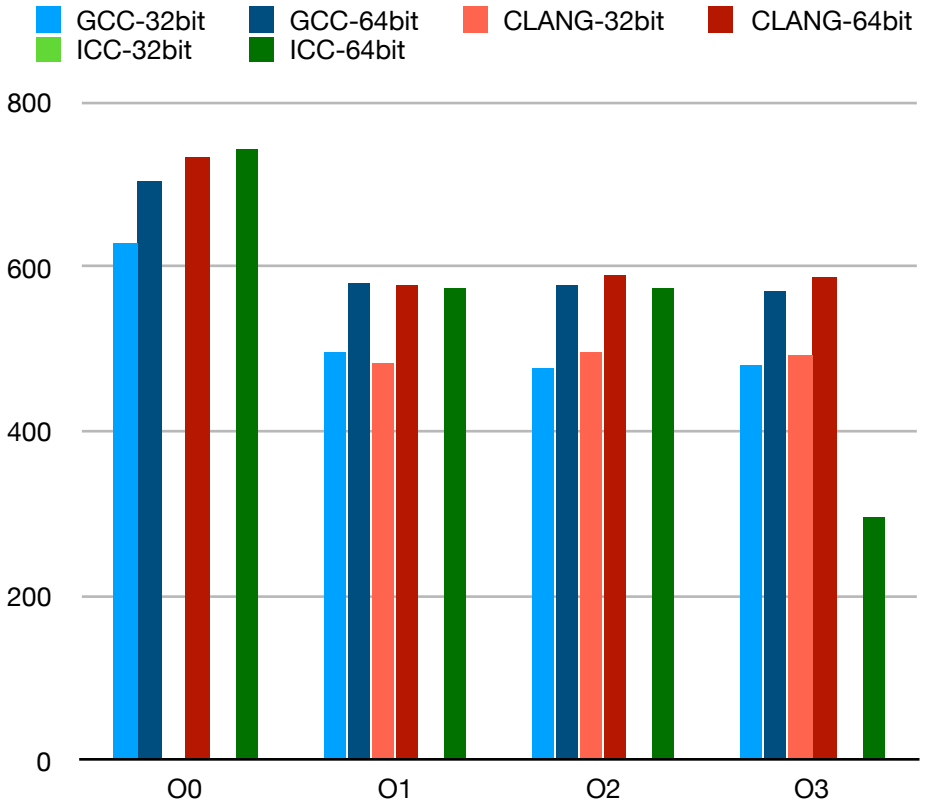
Benchmark: 500.perlbench_r



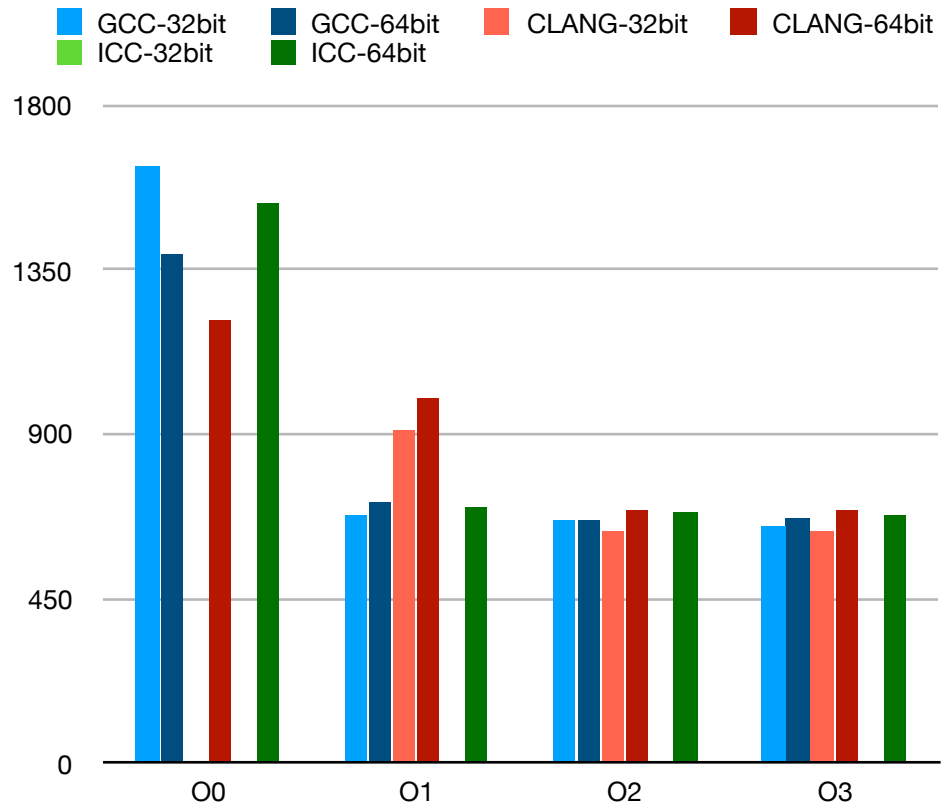
Benchmark: 502.gcc_r



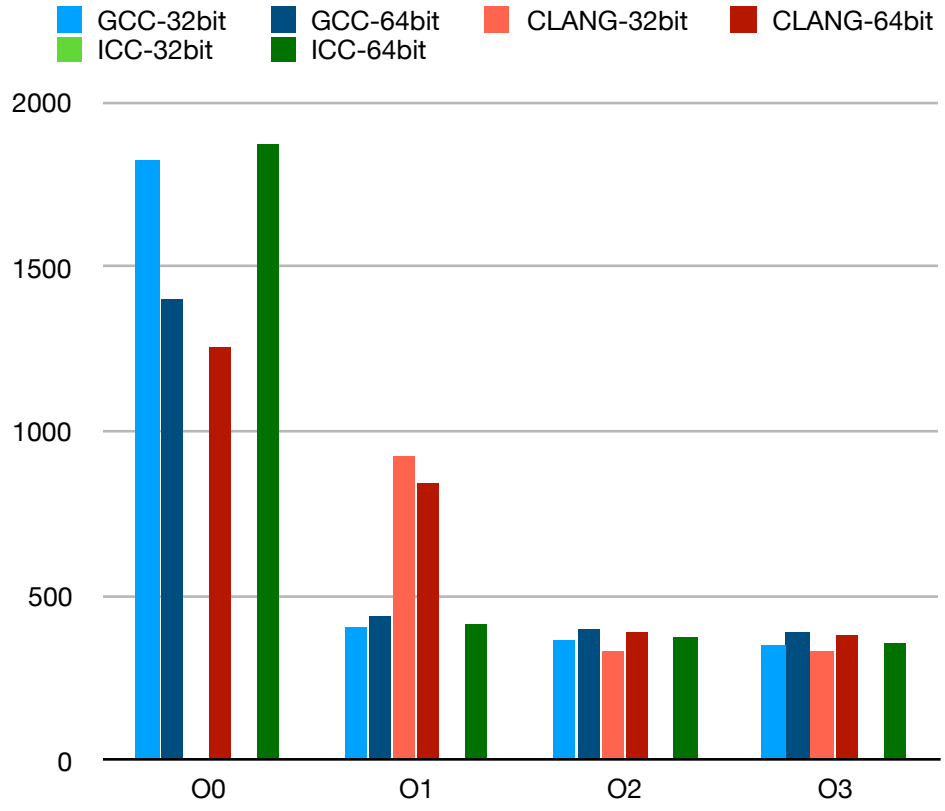
Benchmark: 505.mcf_r



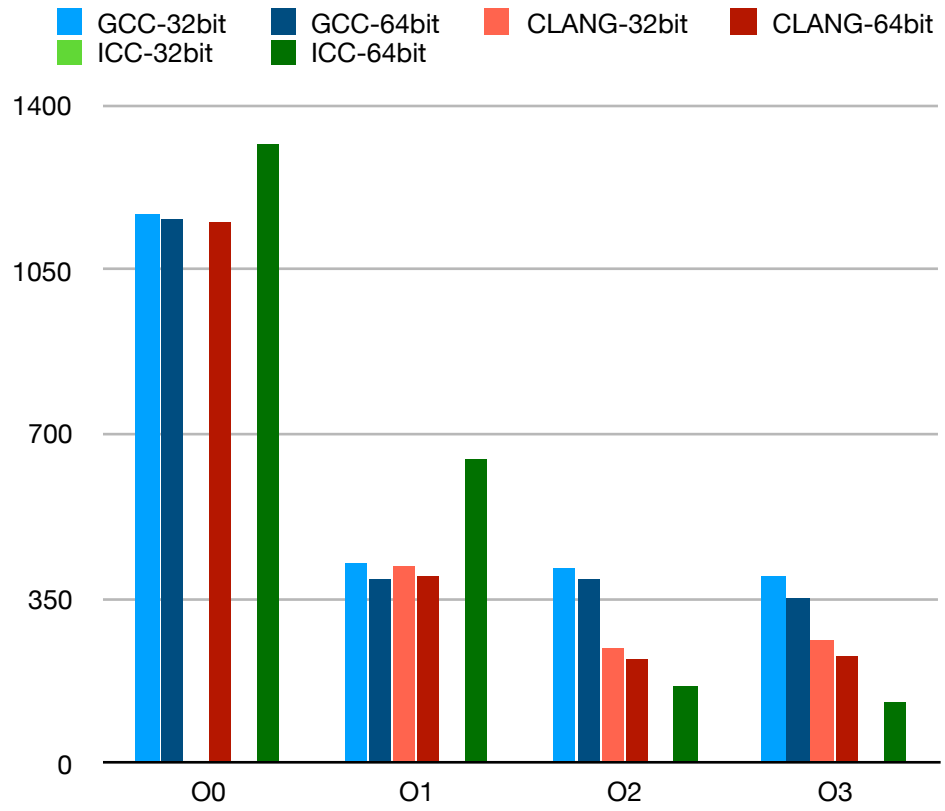
Benchmark: 520.omnetpp_r



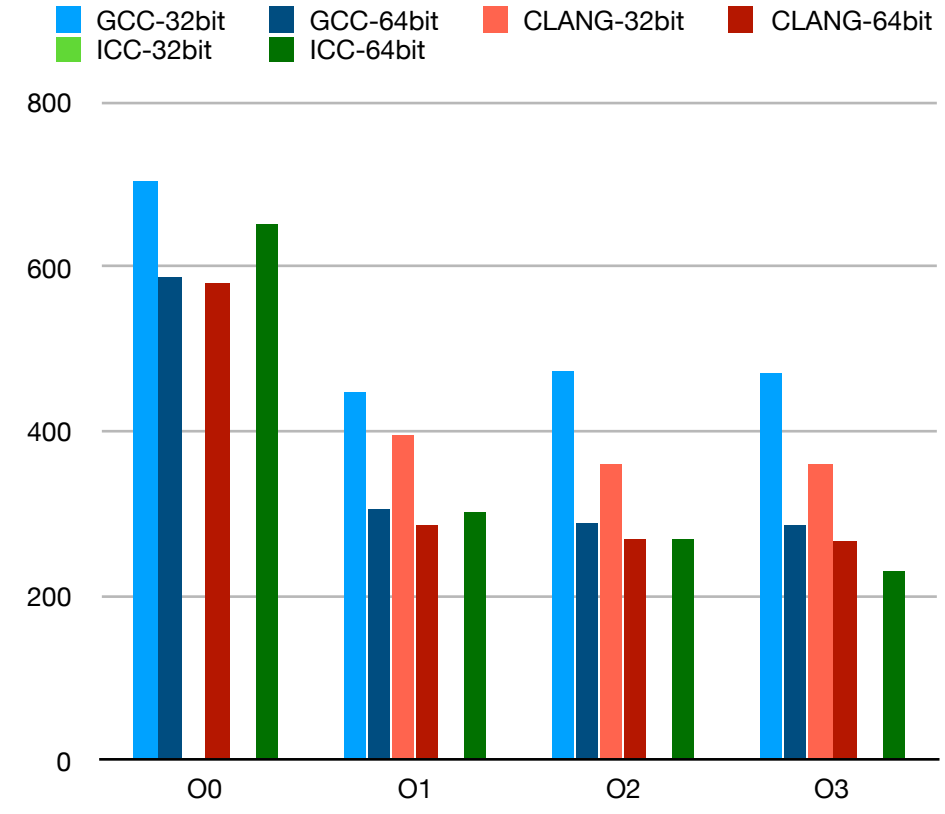
Benchmark: 523.xalancbmk_r



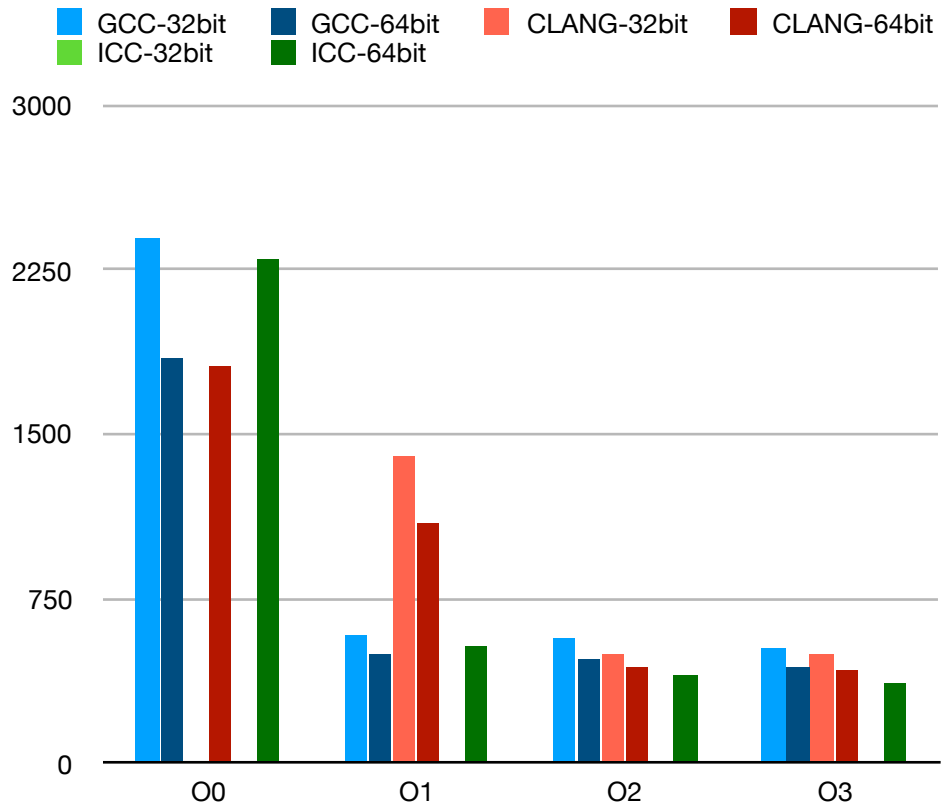
Benchmark: 525.x264_r



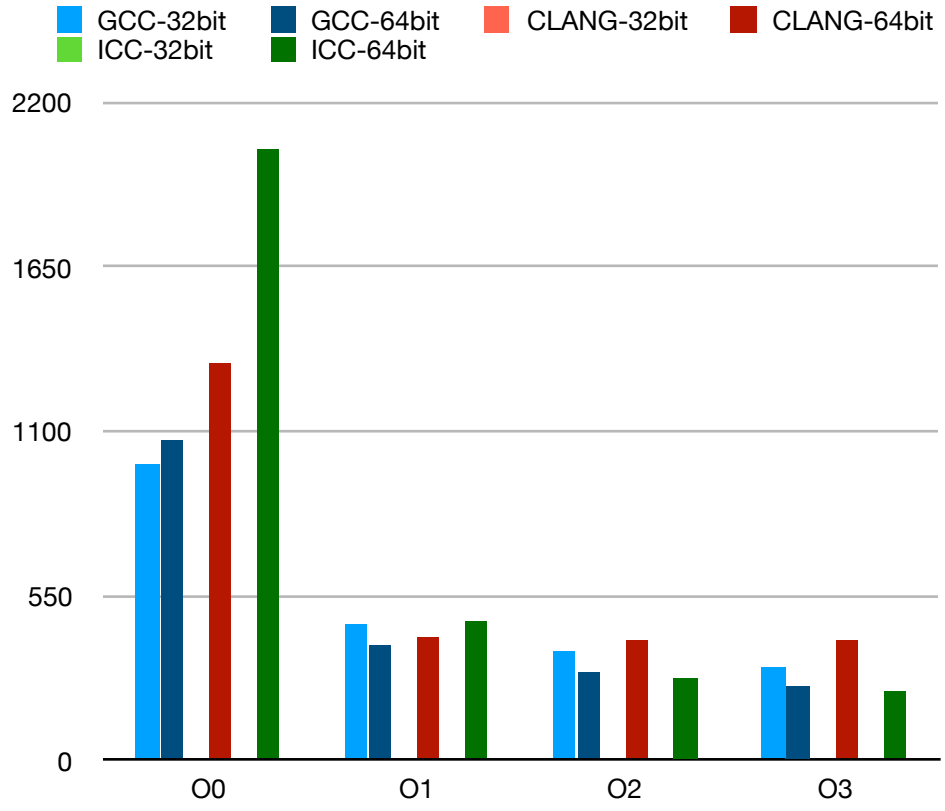
Benchmark: 531.deepsjeng_r



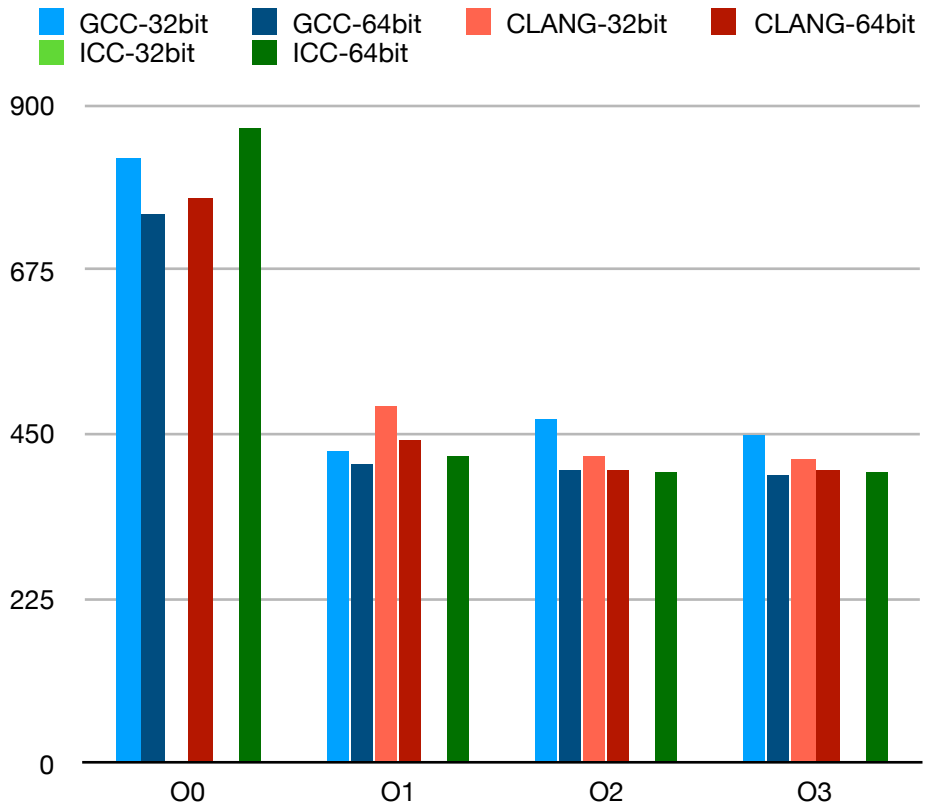
Benchmark: 541.leela_r



Benchmark: 548.exchange2_r



Benchmark: 557.xz_r



Which optimisations are most/least consequential?

O1 is the most consequential optimisation, and O3 is the least consequential optimisation. In fact, for most benchmarks, there is very little benefit after O1 (except for Clang in some benchmarks).

Which compilers are better in what aspects?

ICC seems to be overall best, performing especially well in higher optimisation levels. This is expected as the Intel compiler takes advantage of the micro-architecture specific instructions, whereas other compilers were used without any -march flags.

When comparing GCC with Clang, both are roughly comparable in most benchmarks by higher optimisation flags. Clang does aim to support most of the same optimisation flags as GCC, and thus this observation is as expected.

Which are faster: 32-bit or 64-bit executables? Why?

In most cases, 64-bit executables are somewhat faster than 32-bit executables, but not by a large amount (some benchmarks also show that 64-bit executables are slower than 32-bit executables). Interestingly, optimisation passes don't have any significant effect on the relative performance of 32-bit vs 64-bit executables.

64-bit executables can be faster if the generated code contains 64-bit instructions and have 64-bit data paths, which enables them to process more data than 32-bit systems. 64-bit code can also utilise more general purpose registers, resulting in faster access time.

How are various optimisation levels different? How do they differ across compilers?

O1 optimizes with a basic set of optimisation flags to create a balance between the runtime performance and the compilation time. O2 optimizes more - nearly everything that doesn't involve a code size vs. performance tradeoff. O3 introduces some additional optimisations on top of that.

For ICC/Clang, the vectorisation optimisations get enabled at O2, resulting in much better performance from O2 onwards, whereas in GCC/Clang, these optimisations get enabled only in O3.

Does the kind of benchmark influence the results between compilers? Why?

For most benchmarks, the performance of all compilers is roughly similar by the time O3 flag is switched on. Notable exception to these are the video encoding benchmark 525.x264_r, and the combinatorial optimisation benchmark 505.mcf_r where ICC performs way better than GCC/Clang. The reason could be attributed to extensive micro-architecture specific optimisation by ICC which considerably improves the predominantly arithmetic operations required for those benchmarks.