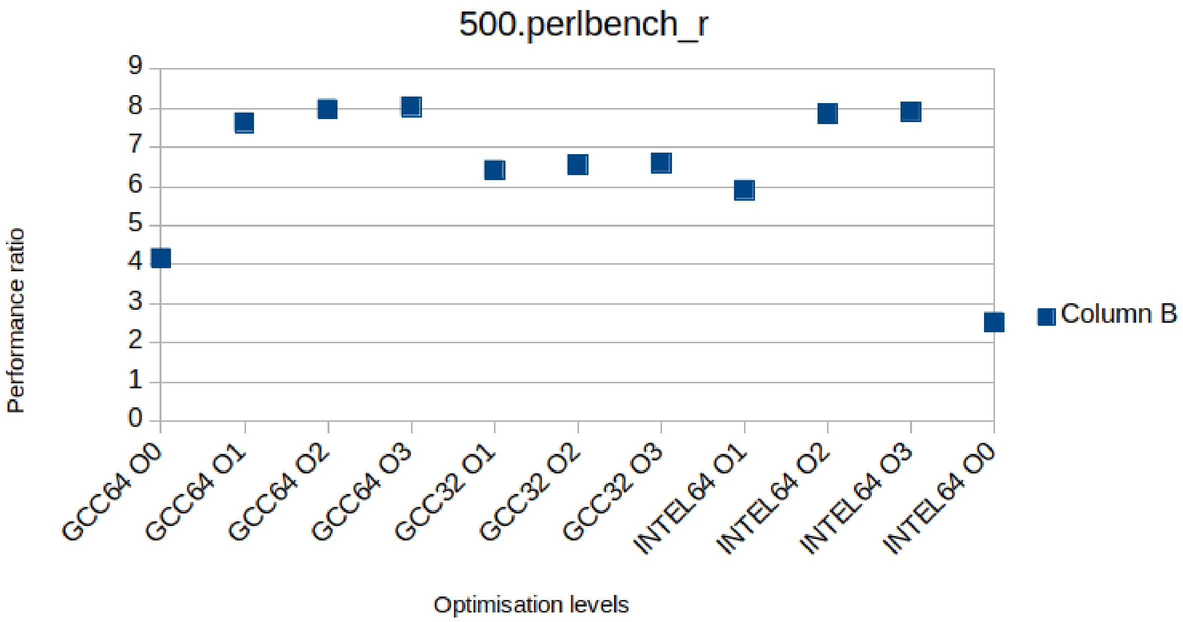
# COL 729 COMPILER OPTIMISATIONS

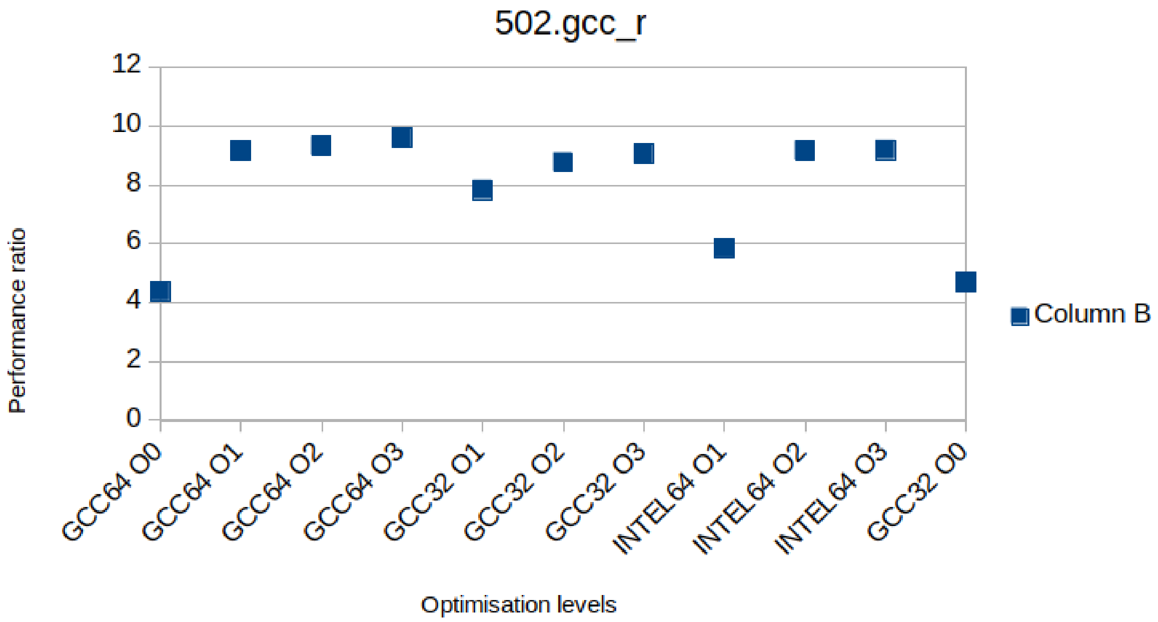
## LAB 0 BENCHMARKING

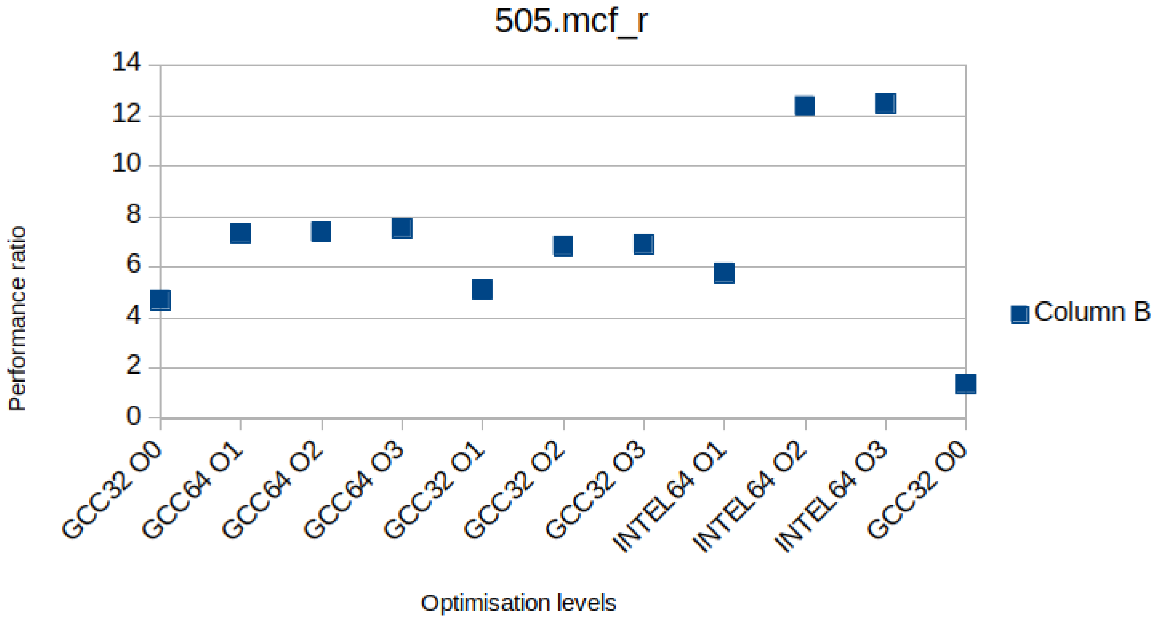
Namrata Jain
2018MCS2840

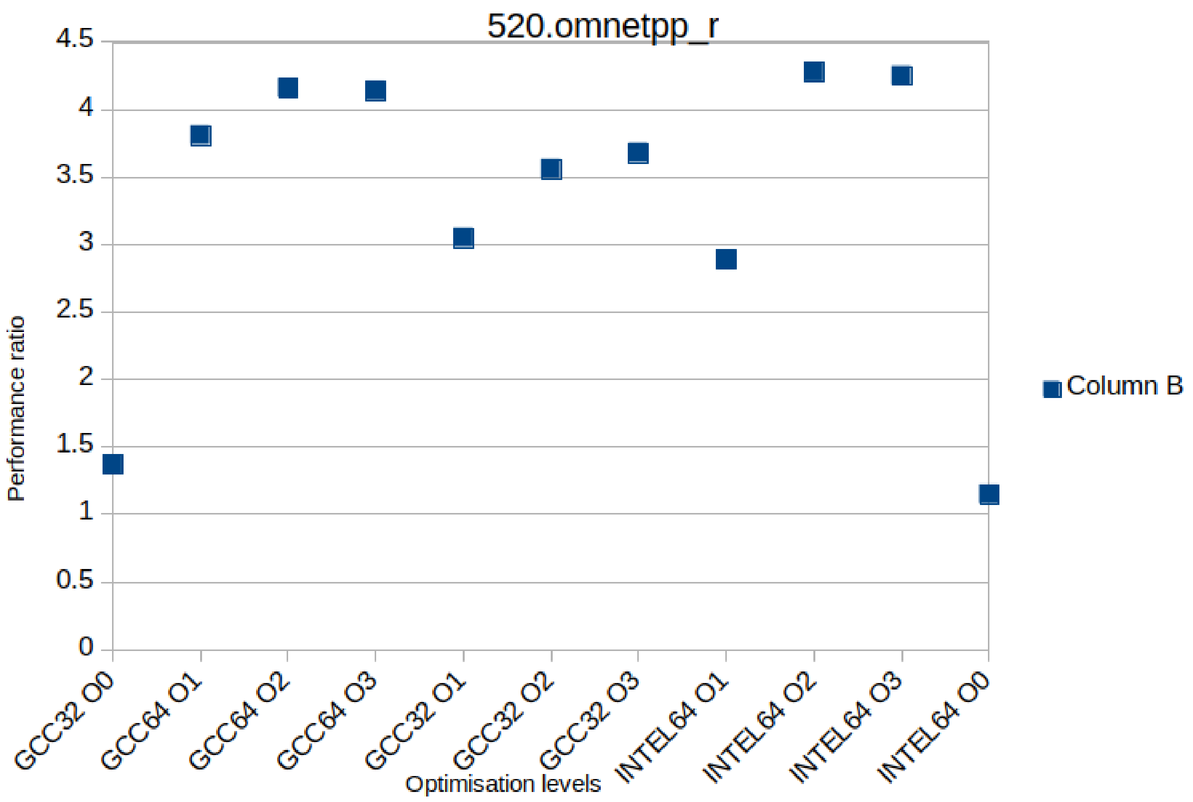
## RUNTIME RESULTS

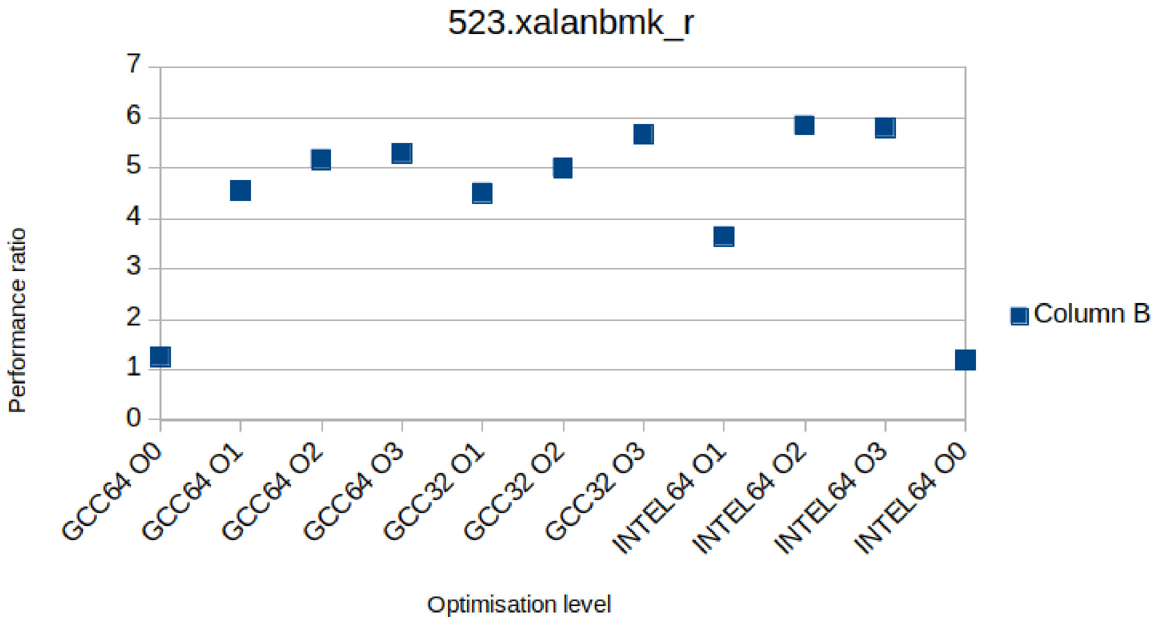
1. Performance ratio with optimisation level -O3 is highest followed by -O2 and -O1 .
2. Performance ratio with 64-bit executables is higher than that with 32-bit executable.
3. Without using optimisation the benchmarks took much time to run thus giving low performance ratio.
4. For some benchmarks, GCC has the better ratio.
5. Plots for each benchmark:

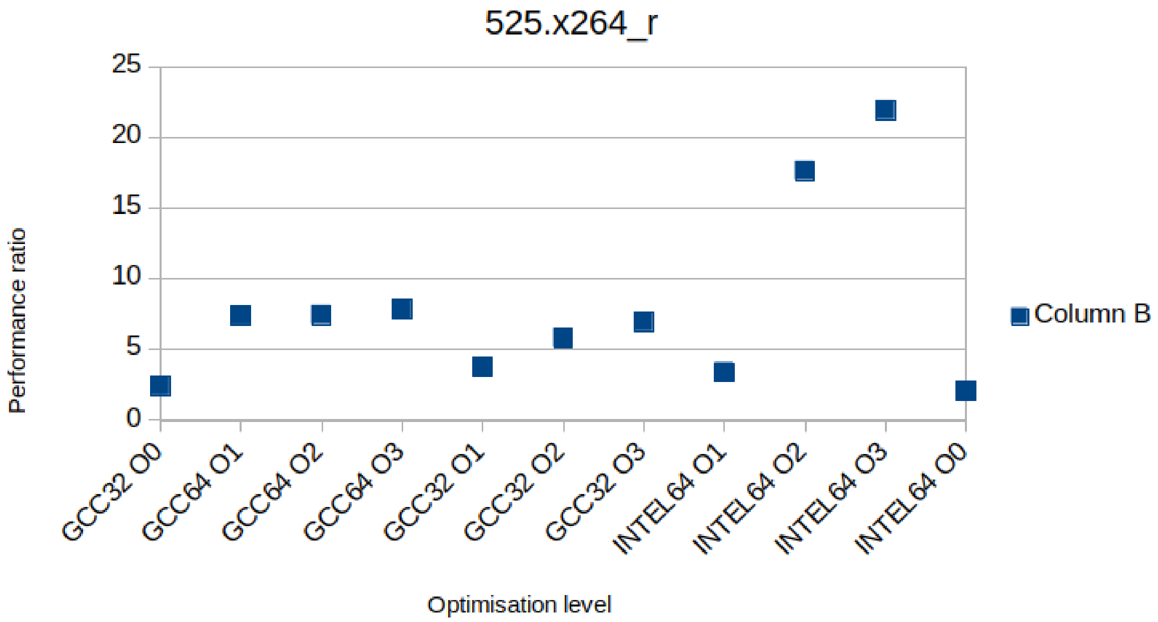


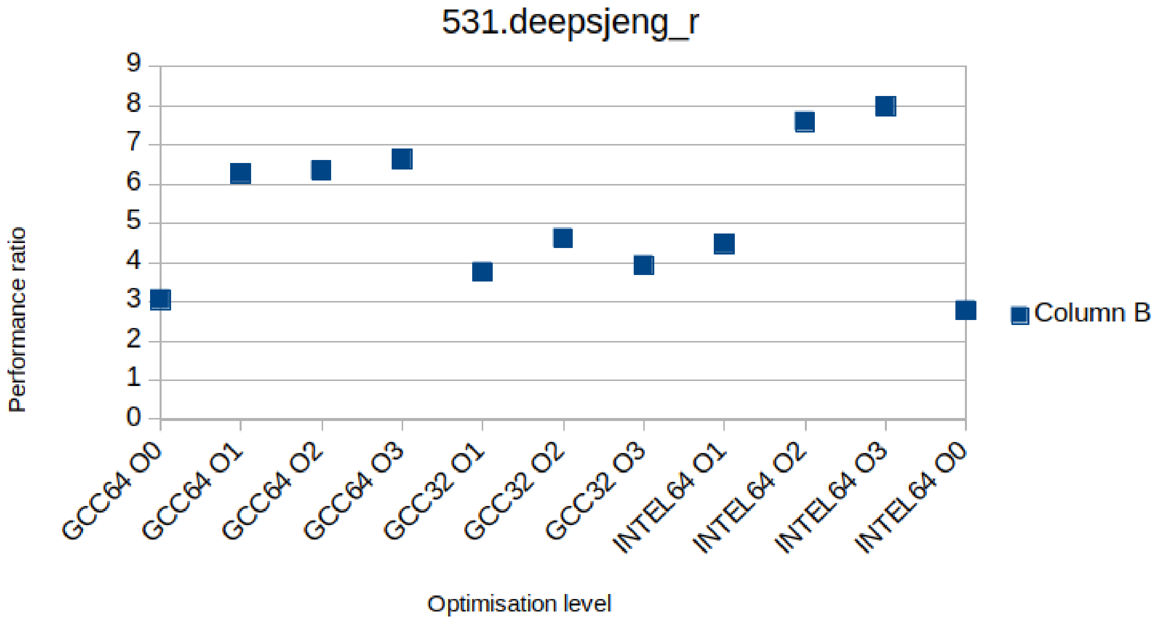

502.gcc_r
Performance ratio
0
2
4
6
8
10
12
GCC64 O0
GCC64 O1
GCC64 O2
GCC64 O3
GCC32 O1
GCC32 O2
GCC32 O3
INTEL64 O1
INTEL64 O2
INTEL64 O3
GCC32 O0
Optimisation levels
■ Column B

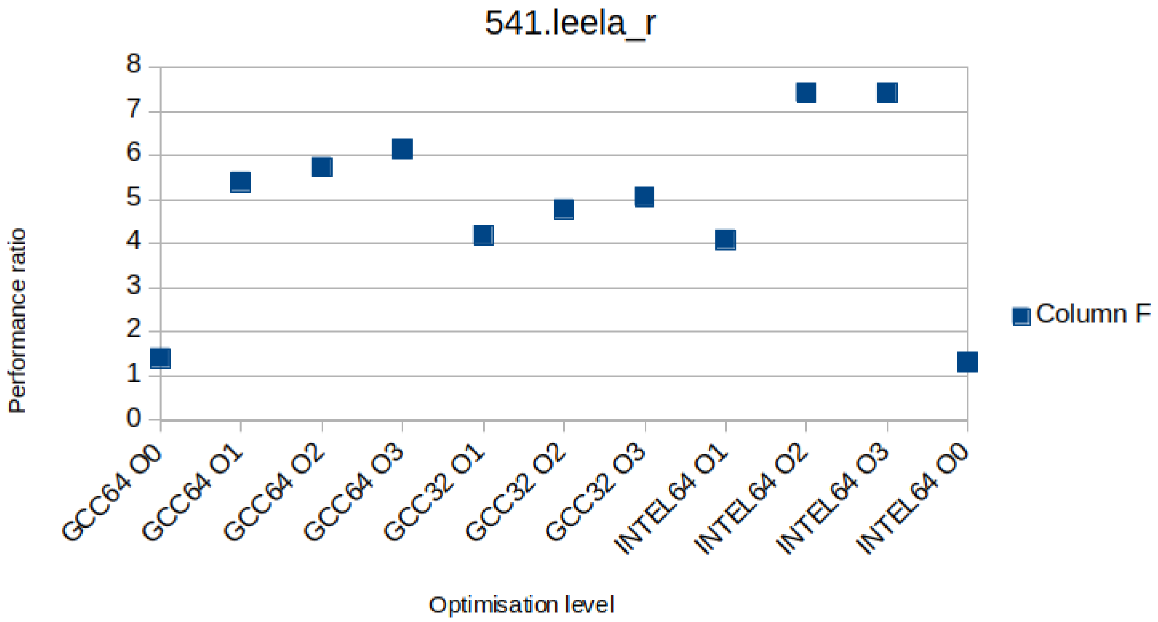

505.mcf_r
Performance ratio
0
2
4
6
8
10
12
14
GCC32 O0
GCC64 O1
GCC64 O2
GCC64 O3
GCC32 O1
GCC32 O2
GCC32 O3
INTEL64 O1
INTEL64 O2
INTEL64 O3
GCC32 O0
Optimisation levels
■ Column B

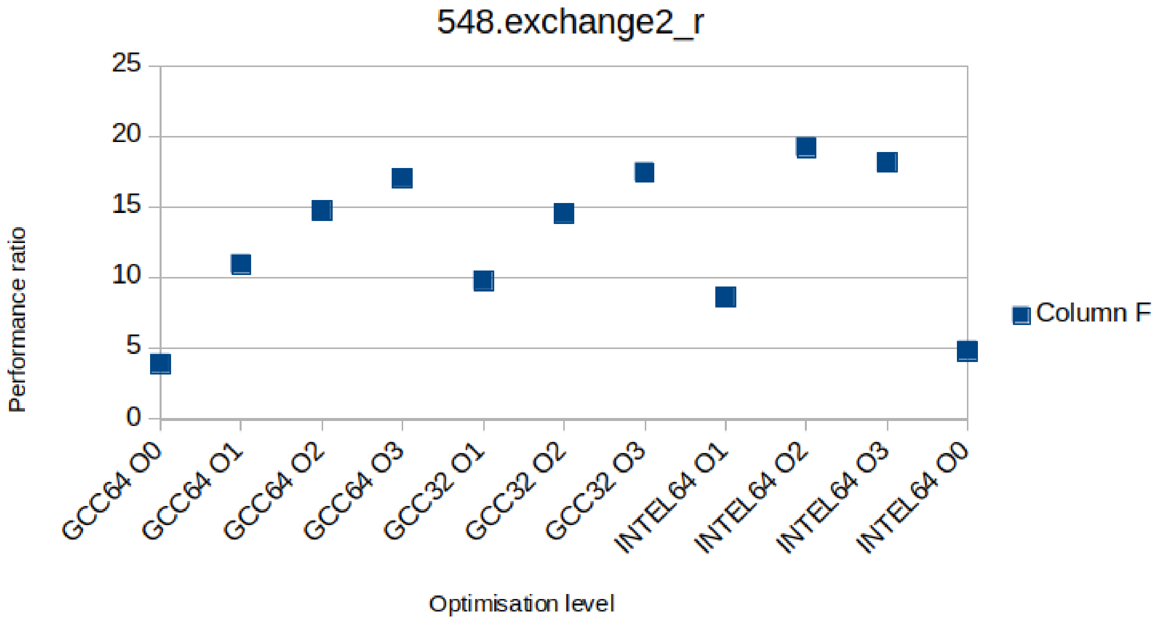

520.omnetpp_r
Performance ratio
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
GCC32 O0
GCC64 O1
GCC64 O2
GCC64 O3
GCC32 O1
GCC32 O2
GCC32 O3
INTEL64 O1
INTEL64 O2
INTEL64 O3
INTEL64 O0
Optimisation levels
■ Column B

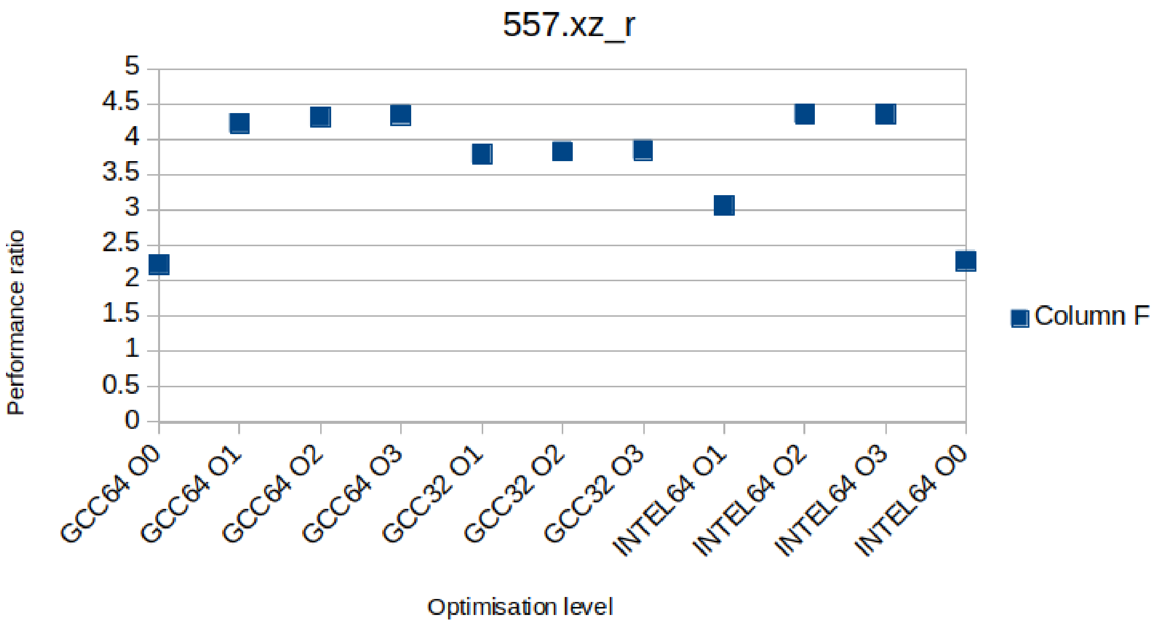

523.xalanbmk_r
Performance ratio
7
6
5
4
3
2
1
0
GCC64 O0
GCC64 O1
GCC64 O2
GCC64 O3
GCC32 O1
GCC32 O2
GCC32 O3
INTEL64 O1
INTEL64 O2
INTEL64 O3
INTEL64 O0
Optimisation level
■ Column B


525.x264_r
Performance ratio
25
20
15
10
5
0
GCC32 O0
GCC64 O1
GCC64 O2
GCC64 O3
GCC32 O1
GCC32 O2
GCC32 O3
INTEL64 O1
INTEL64 O2
INTEL64 O3
INTEL64 O0
Optimisation level
■ Column B


531.deepsjeng_r
Performance ratio
9
8
7
6
5
4
3
2
1
0
GCC64 O0
GCC64 O1
GCC64 O2
GCC64 O3
GCC32 O1
GCC32 O2
GCC32 O3
INTEL64 O1
INTEL64 O2
INTEL64 O3
INTEL64 O0
Optimisation level
■ Column B


541.leela_r
Performance ratio
0
1
2
3
4
5
6
7
8
GCC64 O0
GCC64 O1
GCC64 O2
GCC64 O3
GCC32 O1
GCC32 O2
GCC32 O3
INTEL64 O1
INTEL64 O2
INTEL64 O3
INTEL64 O0
Optimisation level
■ Column F


548.exchange2_r
Performance ratio
0
5
10
15
20
25
GCC64 O0
GCC64 O1
GCC64 O2
GCC64 O3
GCC32 O1
GCC32 O2
GCC32 O3
INTEL64 O1
INTEL64 O2
INTEL64 O3
INTEL64 O0
Optimisation level
■ Column F


557.xz_r
Performance ratio
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
GCC64 O0
GCC64 O1
GCC64 O2
GCC64 O3
GCC32 O1
GCC32 O2
GCC32 O3
INTEL64 O1
INTEL64 O2
INTEL64 O3
INTEL64 O0
Optimisation level
Column F

1. Which compilers are better in what aspects?

The choice of the compiler can have a significant impact on the overall performance of the compiled benchmark. There are several aspects to compiler performance: compiler speed, code quality, error diagnostics, maintainabilty, portability and run time environment.

I used the GCC and Intel compilers for running benchmarks. The performance ratio of almost half the benchmarks were better with GCC compilers, and rest were with ICC. The result deviated in some cases with the optimisation levels.

The performance of ICC may be attributed to the Intel-i5 8th gen processor in the machine. ICC was found better in some optimisation level of the benchmarks: Video compression, AI recursive solution generator, route planning.

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

64-bit executables are generally faster.
x64 architecture has a few more registers which allows easier optimizations, but the pointers are now larger in size and using pointers results in a higher memory access time. So some application with low probabilty may perform better with 32-bit executables.

It is better to compile the program for the system's default word size (32-bit or 64-bit), since if we compile a library as a 32-bit binary and provide it on a 64-bit system, we will force anyone who wants to link with the library to provide their own library (and any other library dependencies) as a 32-bit binary, when the 64-bit version is the default available which can slow down the process.

3. How are the various optimisation levels different? How do these differ accross compilers?

Optimisation levels:
a. GCC:

O0: No optimisation.

O1: This level provides the most common forms of optimization which do not require any speed-space tradeoffs. Comparatively speaking, the executable files produced by this kind of
optimization should be smaller and faster than with the first level - -O0, because of the reduced
amounts of data that need to be processed after simple optimization.

O2: This option provides further optimization than the first two ones, since it could support certain levels of instruction scheduling. Based on this fact, the compiler takes longer to compile programs and needs further requirements for memory consumption than with -O1. And the
executables should not increase in size.

O3: This option turns on more expensive optimizations, such as function inlining, in addition to all the optimizations of the lower levels -O2 and -O1. The -O3 optimization level may increase the speed of the resulting executable, but can also increase its size. Under some circumstances where these optimizations are not favorable, this option might actually make a program slower.

b. ICC

O0: No optimisation

O1: This option enables optimizations for speed and disables some optimizations that increase code size and affect speed. To limit code size, this option enables global optimization which includes data-flow analysis, code motion, strength reduction and test replacement, split-lifetime analysis, and instruction scheduling. This option also disables inlining of some intrinsics.

O2: This option enables optimizations for code speed.

O3: Performs O2 optimizations and enables more aggressive loop transformations. 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.

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

Yes. Some benchmarks run faster on a compiler and slower on the rest.
ICC was found better in some optimisation level of the benchmarks: Video compression, AI recursive solution generator, route planning.
Code complexity, modularity, source language, library links can influence the result between compilers.

References:

1. https://software.intel.com/en-us/articles/step-by-step-optimizing-with-intel-c-compiler

2. http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html