Exercise 04

VU Performance-oriented Computing, Summer Semester 2024

Calvin Hoy

2024-04-13

(A) Basic Optimisation Levels

I created a the Bash script `bench_level.sh`, which utilises `benchmark.sh` from exercise sheet 2.

The following plots were created using `gnuplot` (see `plot.sh`). The left axis shows execution time in seconds for `wall`, `user` and `system`, while the right axis shows memory use in kilobytes.

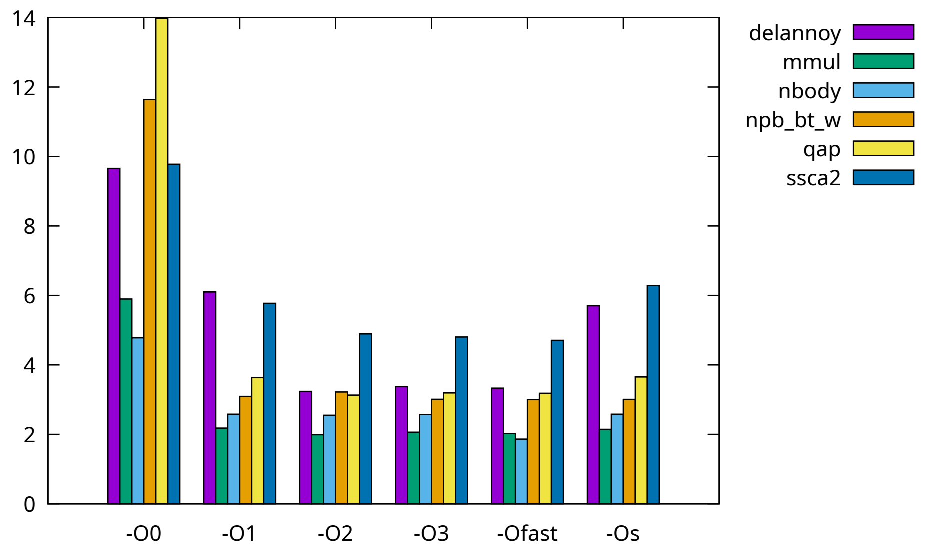

Figure 1: Wall time with different optimisation levels

We can see that for all test cases, `-O3` is faster than `-O2`, `-O2` faster than `-O1`, and `-O1` is much faster than `-O0`. `delannoy 13` profits more from `-O2` than most test cases, in particular compared with `ssca2 15`. `-Ofast` only produces minor gains, though it has a measurable impact on `nbody`. The `-Os` build meanwhile produces code roughly on par with `-O1` in performance, although it is somewhat slower for `ssca2 15`.

System time is close or equal to zero for all tested programs, and user time is basically the same as wall time, thus they are not shown here.

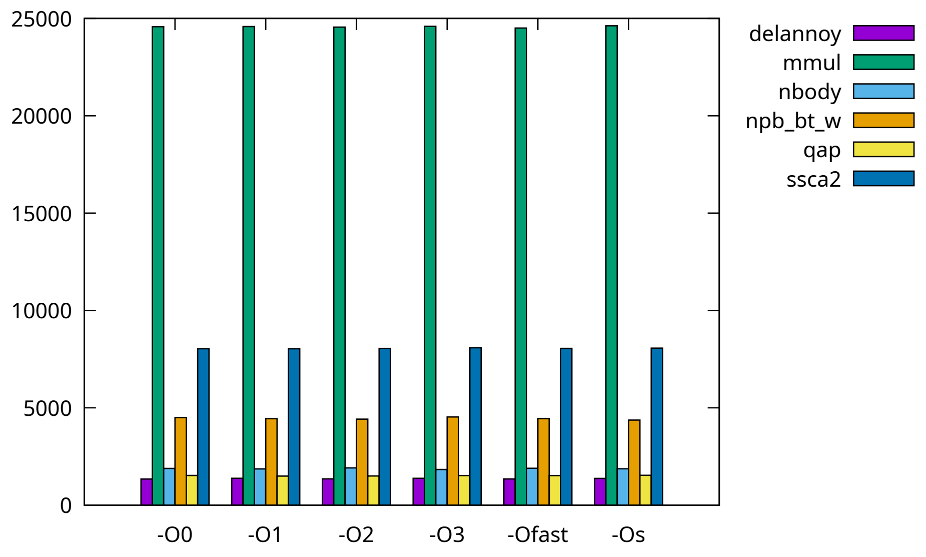

Figure 2: Memory use with different optimisation levels

Memory use is, for all intents and purposes, identical between all optimisation levels.

(B) Individual Compiler Optimisations

The flags that are enabled/changed with `-O3` over `-O2` are the following:

```
-fgcse-after-reload
-fipa-cp-clone
-floop-interchange
-floop-unroll-and-jam
-fpeel-loops
-fpredictive-commoning
-fsplit-loops
-fsplit-paths
-ftree-loop-distribution
-ftree-partial-pre
-funroll-completely-grow-size
-funswitch-loops
-fvect-cost-model=dynamic       # O2: =very-cheap
-fversion-loops-for-strides
```