Science & Technology
Facilities Council

Topics in Software Engineering

Steven Lamerton
Software Engineering Group, STFC

Overview

- Compiler flags
- Static analysis
- Debugging
- Profiling
- Visualisation

Science & Technology
Facilities Council

Compiler Flags

Compiler Flags

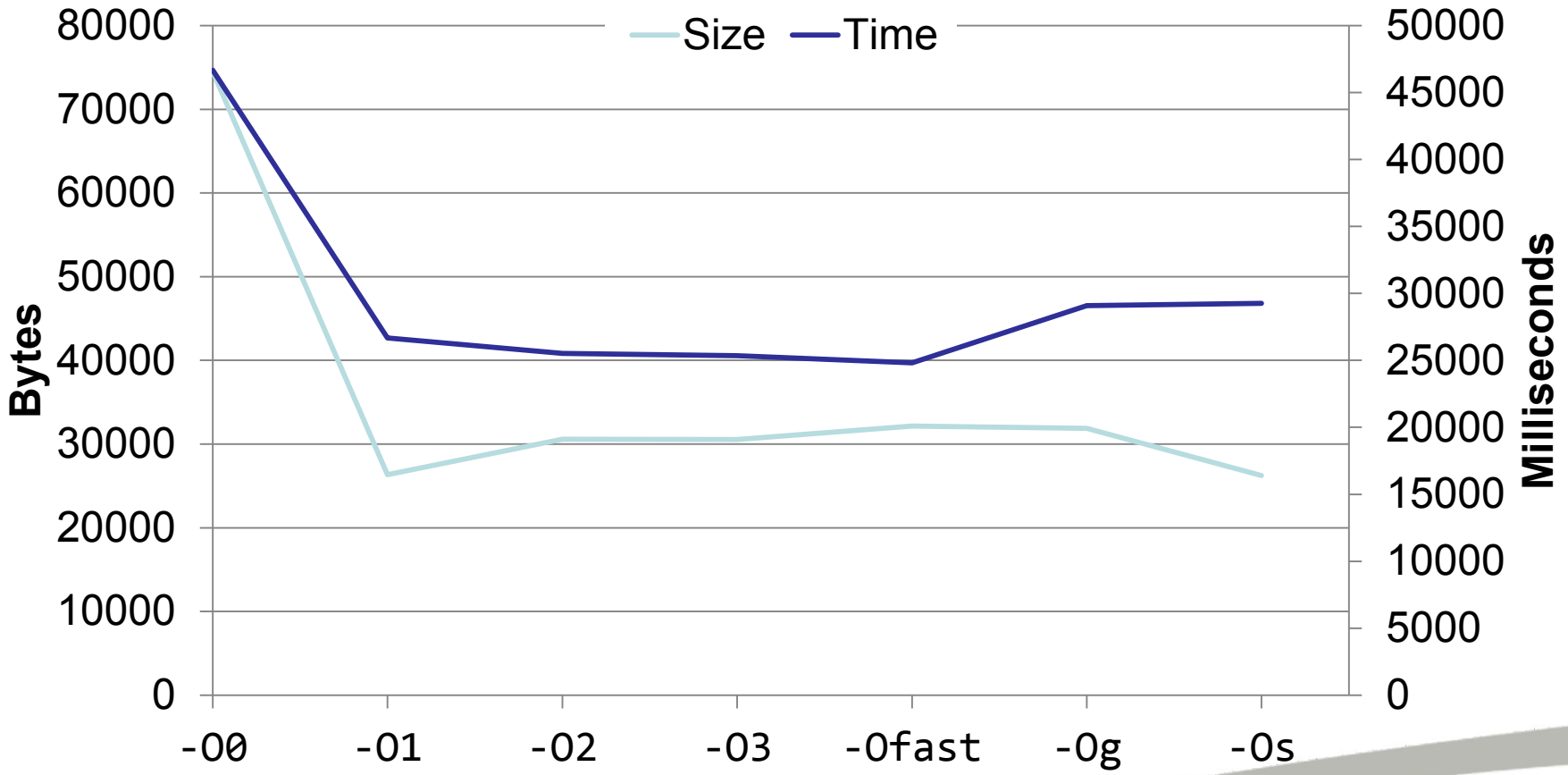
- Compilers have lots (and lots and lots) of flags to tune their operation, but a basic set can make a big difference
- The flags I talk about here are primarily for GCC and will work mostly work for C, C++ and Fortran
- Clang has many similar flags, but not entirely the same, check their documentation for more details
- GCC documentation available at: https://gcc.gnu.org/onlinedocs/gcc/Invoking-GCC.html

Optimisation Flags

- `-O[0|1|2|3]`
  Increasing levels of optimisation, none is the default
- `-Os`
  Optimise for size
- `-Ofast`
  Disregard standards compliance for potential speed
- `-Og`
  Optimise for debugging experience

Use `-O3` in production, `-Og` when developing

Results



Science & Technology Facilities Council

Warning Flags

- `-Wall`
  Enables most warnings, all are easy to work around
- `-Wextra`
  Additional warnings not included in `–Wall`
- `-Wpedantic`
  Issues warnings required by ISO standards
- `-Werror`
  Turns warnings into errors

Use these all the time

Debugging and Profiling Flags

- `-g`
  Produce debugging information whilst compiling, should be used with `-Og` for optimal experience
- `-pg`
  Generate profiling information for gprof
- `-fsanitize=[address|thread|leak|undefined]`
  Enable runtime checks for various error types

Other Useful Flags

- `-std=[c99|c11|c++11|c++14|f2003|f2008]`
  Set the language standard to be used
- `-ffpe-trap=invalid`
  Break on NaNs (Fortran only)

OpenFOAM Options

- Only two options in OpenFOAM:
  - `WM_COMPILE_OPTION=Opt`
  - `WM_COMPILE_OPTION=Debug`
- Opt implies `-O3`
- Debug implies `–O0 -g`
- Both enable a range of warnings including

  `-Wall`

  `-Wextra`

Exercise

- Two options, either use your own code or take a copy of the tarball from https://www.ccp-wsi.ac.uk/?q=software_engineering_workshop
  - The tarball includes codes to try both warning and optimisation flags
- Try using different levels of optimisation and warnings
- To test performance call your executable using the time command:

```
time <your executable>
```

Science & Technology
Facilities Council

Static Analysis

Introduction

- Static analysis is the analysis of a code without running it, usually by inspecting the source code
- Used to find classes of error not picked up by compilers
- Many options available both free and commercial, we will focus on cppcheck as it is good and widely available

cppcheck

- Simple command line interface
- Finds a wide range of issues:
  - Bounds checking
  - Memory leaks
  - Resource leaks
  - Condition checks
  - Unused functions / variables
  - Unreachable code

cppcheck

- `cppcheck [OPTIONS] [files or paths]`
- `--enable=all`
  Enable all checks, subset are also available
- `--std=[c89|c99|c11|c++03|c++11]`
  Set the language standard
- `--inconclusive`
  Show inconclusive results
- `-q`
  Only print warnings / errors

Examples

- Found suspicious equality comparison. Did you intend to assign a value instead?
- Expression 'exp(x) - 1' can be replaced by 'expm1(x)' to avoid loss of precision.
- Technically the member function 'Foam::convexPolyhedral::edgeCutLabel' can be const.
- Variable 'phi' is assigned a value that is never used.

```
062         if (!phicp.coupled())
063         {
064             phicp == 0;
065         }
```

Exercise

- Similarly to before, either use your own code or some from the tarball
- Note that the classes of bugs are quite different to those shown by compiler warning flags
- As with compiler warnings it is best to use static analysis fro the start