ACRC How-To: Getting Started with Compiler Flags

Compiler flags can help us to:

- **optimize our programs for execution speed**,
- **analyse program performance**, and
- **find bugs in our code**.

Below is a sample from the wide array of possible flags to help you get started on these tasks using three popular compilers—GNU, Intel and PGI.

Compiling for Speed

GNU (gcc, g++, gfortran) (version 4.4.6 and above)

For later compiler versions, you can load the modules:

- **gcc/4.4.6** on bluecrystal phase 1.
- **languages/gcc-4.6.2** or **languages/gcc-4.7** on bluecrystal phase 2.

	Optimisation flags of the form **-O<level>** are the first port of call when compiling for speed.
-O2	This level of optimization is safe for most programs. You can see which individual flags are triggered when using this ***composite*** level by typing: **gcc -c -Q -O2 –help=optimizers | less**.
-O3	Will enable more optimization flags, but may not suit some programs.
-Ofast	**NB** This level enables the **-ffast-math** setting which should be used with caution. It will cause incorrect output from programs which require strict adherence to the IEEE floating-point arithmetic standard.
-funroll-loops	Unrolls loops. May or may not make your program run faster.
	The following two flags are important for making best use of the particular hardware you will be running your program on. For example, they can be used to ensure code ***vectorization*** to make best use of wide registers.
-march=native	Will enable all instruction subsets supported by the local machine.
-mtune=native	Will produce code optimized for the local machine under the constraints of the selected instruction set.

Intel (version 10.1 and above)

For Bluecrystal phases 2 & 3 (which contain Intel processors), the Intel compiler is likely to give you the fastest running executable.

-O2	The default optimization level. Suits most programs.
-O3	Enables more aggressive loop and memory access optimizations—such as scalar replacement—loop unrolling, code replication to eliminate branches, loop blocking to allow more efficient use of cache and additional data prefetching. **NB This level of optimization also includes a setting akin to GNU's -ffast-math and so comes with all the attendant warnings about IEEE floating-point compliance**. **-fp-model precise can be used in conjunction to ensure only value-safe optimizations**.

Gethin Williams, 2013.

University of BRISTOL

-xHOST	Generates specialized code to run exclusively on the host processor type.

PGI (version 7.2 and above)

-fast	Enables a generally optimal set of flags. See pgcc -fast -help, for the list of inividual flags.

Compiling to Assess Performance

All three compilers will instrument executables for use with the GNU profiler, **gprof**. When instrumented in this way, a running executable will record, for example, the frequency and duration of each function call within the progam. The compilers can additionally provide information regarding any automatic vectorization of code.

GNU

-pg	Instrument for gprof.
-ftree-vectorizer-verbose=2	Report vectorization diagnostics including loops which were and loops which were not vectorized.

Intel

-p	Instrument for gprof.
-vec-report3	Report vectorization diagnostics including loops which were and loops which were not vectorized.

PGI

-pg	Instrument for gprof.
-Minfo=vect	Report vectorization diagnostics.

Compiling to Debug

In a similar vein to profiling, all the compilers can instrument executables such that they can be interrogated by a debugger such as the GNU debugger, gdb. Note that instrumenting for debugging usually implies that all optimizations for speed are removed. Compilers can also provide other useful facilities, such as calling stack backtraces and array-bounds checks.

GNU

-g	Instrument code for GNU debugger.
-Wall	Enable all warnings.
-ftraceback	**Fortran only**: Produce a traceback of the calling stack if a runtime error is encountered. (Best used with -g, for more intelligible results).
-fbounds-check	**Fortran only**: generate additional code to check that indices used to access arrays are within the declared range.

Intel

-g	Instrument code for GNU debugger.
-Wall	Enable all warnings.
-traceback	**Fortran only**: Produce a traceback of the calling stack if a runtime error is encountered. (Best used with **-g**, for more intelligible results).
-CB	**Fortran only**: Enable runtime array-bounds checking.
-CU	**Fortran only**: Enable runtime checks for use of uninitialized variables.
-CA	**Fortran only**: Enable runtime checks for pointers.

PGI

-g	Instrument code for GNU debugger.
-Mbounds	Add array bounds checking.
-traceback	Produce a traceback of the calling stack if a runtime error is encountered. (Best used with **-g**, for more intelligible results).

Further Information

GNU	**http://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html**
Intel	**http://software.intel.com/sites/products/collateral/hpc/compilers/compiler_qrg12.pdf**
PGI	**http://www.pgroup.com/doc/pgiug.pdf**