GCC Switches

- Prepared By:

  Ankit Desai

  MT01

  Dharamsinh Desai Institute of Technology, M.Tech. Sem 1
- Submitted To:

  J T Lalchandani

  Associate Professor

  DDIT.

Index

1. Debugging Options: Symbol tables, measurements, and debugging dumps.

==========

-g

Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF 2). GDB can work with this debugging information.

On most systems that use stabs format, -g enables use of extra debugging information that only GDB can use; this extra information makes debugging work better in GDB but will probably make other debuggers crash or refuse to read the program. If you want to control for certain whether to generate the extra information, use -gstabs+, -gstabs, -gxcoff+, -gxcoff, or -gvms

GCC allows you to use -g with -O. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

The following options are useful when GCC is generated with the capability for more than one debugging format.

-ggdb
    Produce debugging information for use by GDB. This means to use the most expressive format available (DWARF 2, stabs, or the native format if neither of those are supported), including GDB extensions if at all possible.

-gstabs
    Produce debugging information in stabs format (if that is supported), without GDB extensions.

-feliminate-unused-debug-symbols
    Produce debugging information in stabs format (if that is supported), for only symbols that are actually used.

-gcoff
    Produce debugging information in COFF format (if that is supported).

-gxcoff
    Produce debugging information in XCOFF format (if that is

```
    supported).

There are many other similar options available which can be found
at:

http://gcc.gnu.org/onlinedocs/gcc/Debugging-
Options.html#Debugging-Options

================================================================
-p
    Generate extra code to write profile information suitable for
    the analysis program prof. You must use this option when
    compiling the source files you want data about, and you must
    also use it when linking.
================================================================
-Q
    Makes the compiler print out each function name as it is
    compiled, and print some statistics about each pass when it
    finishes.

Example:

jazz@linuxmint ~/Desktop/MTech/ACT/gcc $ gcc -Q -o test test.c
 printdata main
Analyzing compilation unit
Performing interprocedural optimizations
    <visibility>    <early_local_cleanups>    <summary    generate>
<inline>Assembling functions:
 printdata main
Execution times (seconds)
 preprocessing          :   0.01 (20%) usr   0.03 (75%) sys   0.04
(29%) wall     129 kB (12%) ggc
 lexical analysis       :   0.01 (20%) usr   0.01 (25%) sys   0.03
(21%) wall       0 kB ( 0%) ggc
 parser                 :   0.01 (20%) usr   0.00 ( 0%) sys   0.01
( 7%) wall     380 kB (35%) ggc
 tree PHI insertion     :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01
( 7%) wall       0 kB ( 0%) ggc
 tree SSA to normal     :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.03
(21%) wall       0 kB ( 0%) ggc
 expand                 :   0.01 (20%) usr   0.00 ( 0%) sys   0.00
( 0%) wall       9 kB ( 1%) ggc
 TOTAL                  :   0.05             0.04             0.14
1090 kB
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $
================================================================
-ftime-report
    Makes the compiler print some statistics about the time
    consumed by each pass when it finishes.
```

Example:

```
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $ gcc -ftime-report -o test
test.c

Execution times (seconds)
 preprocessing          :   0.02 (50%) usr   0.00 ( 0%) sys   0.02
(25%) wall     129 kB (12%) ggc
 lexical analysis       :   0.00 ( 0%) usr   0.03 (75%) sys   0.02
(25%) wall       0 kB ( 0%) ggc
 parser                 :   0.01 (25%) usr   0.00 ( 0%) sys   0.03
(37%) wall     380 kB (35%) ggc
 tree operand scan      :   0.01 (25%) usr   0.00 ( 0%) sys   0.00
( 0%) wall       2 kB ( 0%) ggc
 TOTAL                  :   0.04             0.04             0.08
1090 kB
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $
```

==========

-fmem-report

Makes the compiler print some statistics about permanent memory allocation when it finishes.

Example:

```
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $ gcc -fmem-report -o test
test.c
Memory still allocated at the end of the compilation process
Size   Allocated        Used    Overhead
8              16k         15k         384
32           4096        2912          48
64           4096         960          40
128           220k         217k        1980
256            12k        8960          96
512          8192        4096          64
1024           16k          14k         128
2048           36k          34k         288
4096           20k          16k         160
8192           24k          24k          96
44           4096        1232          40
104           232k         227k        2088
92           8192        7176          72
80             68k          65k         612
88           4096        1584          36
56            164k         159k        1640
84           4096         168          36
60           4096        2700          40
28            268k         266k        3216
16             20k          16k         320
36             12k        8712          132
```

```
12              16k          13k         288
40            4096          880          44
Total         1168k        1108k          11k

String pool
entries         2883
identifiers     2883 (100.00%)
slots           16384
deleted         0
bytes      34k (4095M overhead)
table size      64k
coll/search     0.0821
ins/search      0.2172
avg. entry      12.16 bytes (+/- 6.93)
longest entry   45

??? tree nodes created

(No per-node statistics)
Type hash: size 2039, 884 elements, 0.632107 collisions
DECL_DEBUG_EXPR  hash: size 1021, 0 elements, 0.000000 collisions
DECL_VALUE_EXPR  hash: size 1021, 0 elements, 0.000000 collisions
No gimple statistics
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $
```

-time[=*file*]
Report the CPU time taken by each subprocess in the compilation sequence. For C source files, this is the compiler proper and assembler (plus the linker if linking is done).

Without the specification of an output file, the output looks like this:

```
# cc1 0.12 0.01
# as 0.00 0.01
```

The first number on each line is the "user time", that is time spent executing the program itself. The second number is "system time", time spent executing operating system routines on behalf of the program. Both numbers are in seconds.

With the specification of an output file, the output is appended to the named file, and it looks like this:

```
0.12 0.01 cc1 options
0.00 0.01 as options
```

The "user time" and the "system time" are moved before the program name, and the options passed to the program are displayed, so that one can later tell what file was being compiled, and with which

options.

Example:

```
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $ gcc -time -o test test.c

# cc1 0.04 0.03
# as 0.00 0.01
# collect2 0.05 0.02
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $
```

2. Options That Control Optimization

These options control various sorts of optimizations.

Without any optimization option, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

The compiler performs optimization based on the knowledge it has of the program. Compiling multiple files at once to a single output file mode allows the compiler to use information gained from all of the files when compiling each of them.

Not all optimizations are controlled directly by a flag. Only optimizations that have a flag are listed in this section.

Most optimizations are only enabled if an -O level is set on the command line. Otherwise they are disabled, even if individual optimization flags are specified.

Depending on the target and how GCC was configured, a slightly different set of optimizations may be enabled at each -O level than those listed here. You can invoke GCC with `-Q --help=optimizers' to find out the exact set of optimizations that are enabled at each level.

==========

-O1
Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. With -O, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

==========

-O2
Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to -O, this option increases both compilation time and the performance of the generated code.

==========

-O3
Optimize yet more. -O3 turns on all optimizations specified by -O2 & even more.

==========

-Os
Optimize for size. -Os enables all -O2 optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size.

==========

-Ofast
Disregard strict standards compliance. -Ofast enables all -O3 optimizations. It also enables optimizations that are not valid for all standard compliant programs.

==========

-floop-interchange
Perform loop interchange transformations on loops. Interchanging two nested loops switches the inner and outer loops. For example, given a loop like:

```
DO J = 1, M
  DO I = 1, N
    A(J, I) = A(J, I) * C
  ENDDO
ENDDO
```

loop interchange will transform the loop as if the user had written:

```
DO I = 1, N
  DO J = 1, M
    A(J, I) = A(J, I) * C
  ENDDO
ENDDO
```

which can be beneficial when N is larger than the caches, because in Fortran, the elements of an array are stored in memory contiguously by column, and the original loop iterates over rows, potentially creating at each access a cache miss. This optimization applies to all the languages supported by GCC and is not limited to Fortran. To use this code transformation, GCC has to be configured with --with-ppl and --with-cloog to enable the Graphite loop transformation infrastructure.

==========

-floop-strip-mine

Perform loop strip mining transformations on loops. Strip mining splits a loop into two nested loops. The outer loop has strides equal to the strip size and the inner loop has strides of the original loop within a strip. The strip length can be changed using the loop-block-tile-size parameter. For example, given a loop like:

```
DO I = 1, N
  A(I) = A(I) + C
ENDDO
```

loop strip mining will transform the loop as if the user had written:

```
DO II = 1, N, 51
  DO I = II, min (II + 50, N)
    A(I) = A(I) + C
  ENDDO
ENDDO
```

This optimization applies to all the languages supported by GCC and is not limited to Fortran. To use this code transformation, GCC has to be configured with --with-ppl and --with-cloog to enable the Graphite loop transformation infrastructure.

==========

-floop-block

Perform loop blocking transformations on loops. Blocking strip mines each loop in the loop nest such that the memory accesses of the element loops fit inside caches. The strip length can be changed using the loop-block-tile-size parameter. For example, given a loop like:

```
DO I = 1, N
  DO J = 1, M
    A(J, I) = B(I) + C(J)
  ENDDO
ENDDO
```

loop blocking will transform the loop as if the user had written:

```
DO II = 1, N, 51
  DO JJ = 1, M, 51
    DO I = II, min (II + 50, N)
```

```
              DO J = JJ, min (JJ + 50, M)
                A(J, I) = B(I) + C(J)
              ENDDO
            ENDDO
          ENDDO
        ENDDO
```

which can be beneficial when M is larger than the caches, because the innermost loop will iterate over a smaller amount of data that can be kept in the caches. This optimization applies to all the languages supported by GCC and is not limited to Fortran. To use this code transformation, GCC has to be configured with --with-ppl and --with-cloog to enable the Graphite loop transformation infrastructure.

==========

-floop-flatten

Removes the loop nesting structure: transforms the loop nest into a single loop. This transformation can be useful to vectorize all the levels of the loop nest.

==========

Click here for more optimization related options:
http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Optimize-Options

==========

3. Options Controlling the Preprocessor

These options control the C preprocessor, which is run on each C source file before actual compilation.

If you use the -E option, nothing is done except preprocessing. Some of these options make sense only together with -E because they cause the preprocessor output to be unsuitable for actual compilation.

==========

-o *file*

Write output to *file*. This is the same as specifying *file* as the second non-option argument to cpp. gcc has a different interpretation of a second non-option argument, so you must use -o to specify the output file.

==========

-w

Suppress all warnings, including those which GNU CPP issues by default.

==========

-nostdinc
    Do not search the standard system directories for header files. Only the directories you have specified with -I options (and the directory of the current file, if appropriate) are searched.

==========

-nostdinc++
    Do not search for header files in the C++-specific standard directories, but do still search the other standard directories. (This option is used when building the C++ library.)

==========

-include *file*
    Process *file* as if #include "file" appeared as the first line of the primary source file. However, the first directory searched for *file* is the preprocessor's working directory *instead of* the directory containing the main source file. If not found there, it is searched for in the remainder of the #include "..." search chain as normal.

If multiple -include options are given, the files are included in the order they appear on the command line.

==========

-imacros *file*
    Exactly like -include, except that any output produced by scanning *file* is thrown away. Macros it defines remain defined. This allows you to acquire all the macros from a header without also processing its declarations.

All files specified by -imacros are processed before all files specified by -include.

==========

-C
    Do not discard comments. All comments are passed through to the output file, except for comments in processed directives, which are deleted along with the directive.

You should be prepared for side effects when using -C; it causes the preprocessor to treat comments as tokens in their own right. For example, comments appearing at the start of what would be a directive line have the effect of turning that line into an ordinary source line, since the first token on the line is no longer a `#'.

==========

-CC
    Do not discard comments, including during macro expansion. This is like -C, except that comments contained within macros are also passed through to the output file where the macro is

```
    expanded.

In addition to the side-effects of the -C option, the -CC option
causes all C++-style comments inside a macro to be converted to C-
style comments. This is to prevent later use of that macro from
inadvertently commenting out the remainder of the source line.

The -CC option is generally used to support lint comments.

================================================================
-trigraphs
    Process trigraph sequences. These are three-character
    sequences, all starting with `??', that are defined by ISO C
    to stand for single characters. For example, `??/' stands for
    `\', so `'??/n'' is a character constant for a newline. By
    default, GCC ignores trigraphs, but in standard-conforming
    modes it converts them. See the -std and -ansi options.

The nine trigraphs and their replacements are
         Trigraph:       ??(  ??)  ??<  ??>  ??=  ??/  ??'  ??!  ??-
         Replacement:      [    ]    {    }    #    \    ^    |    ~


================================================================


-v
    Verbose mode. Print out GNU CPP's version number at the
    beginning of execution, and report the final form of the
    include path.

Example:

jazz@linuxmint ~/Desktop/MTech/ACT/gcc $ gcc -v -o test test.c
Using built-in specs.
Target: i486-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu
4.4.3-4ubuntu5' --with-bugurl=file:///usr/share/doc/gcc-
4.4/README.Bugs --enable-languages=c,c++,fortran,objc,obj-c++
--prefix=/usr --enable-shared --enable-multiarch --enable-linker-
build-id --with-system-zlib --libexecdir=/usr/lib --without-
included-gettext --enable-threads=posix --with-gxx-include-
dir=/usr/include/c++/4.4 --program-suffix=-4.4 --enable-nls
--enable-clocale=gnu --enable-libstdcxx-debug --enable-plugin
--enable-objc-gc --enable-targets=all --disable-werror --with-
arch-32=i486 --with-tune=generic --enable-checking=release
--build=i486-linux-gnu --host=i486-linux-gnu --target=i486-linux-
gnu
Thread model: posix
gcc version 4.4.3 (Ubuntu 4.4.3-4ubuntu5)
COLLECT_GCC_OPTIONS='-v' '-o' 'test' '-mtune=generic' '-
march=i486'
```

```
 /usr/lib/gcc/i486-linux-gnu/4.4.3/cc1 -quiet -v test.c
-D_FORTIFY_SOURCE=2 -quiet -dumpbase test.c -mtune=generic
-march=i486 -auxbase test -version -fstack-protector -o
/tmp/ccJnS0wf.s
GNU C (Ubuntu 4.4.3-4ubuntu5) version 4.4.3 (i486-linux-gnu)
     compiled by GNU C version 4.4.3, GMP version 4.3.2, MPFR
version 2.4.2-p1.
GGC heuristics: --param ggc-min-expand=63 --param ggc-min-
heapsize=62644
ignoring nonexistent directory "/usr/local/include/i486-linux-gnu"
ignoring nonexistent directory "/usr/lib/gcc/i486-linux-
gnu/4.4.3/../../../../i486-linux-gnu/include"
ignoring nonexistent directory "/usr/include/i486-linux-gnu"
#include "..." search starts here:
#include <...> search starts here:
 /usr/local/include
 /usr/lib/gcc/i486-linux-gnu/4.4.3/include
 /usr/lib/gcc/i486-linux-gnu/4.4.3/include-fixed
 /usr/include
End of search list.
GNU C (Ubuntu 4.4.3-4ubuntu5) version 4.4.3 (i486-linux-gnu)
     compiled by GNU C version 4.4.3, GMP version 4.3.2, MPFR
version 2.4.2-p1.
GGC heuristics: --param ggc-min-expand=63 --param ggc-min-
heapsize=62644
Compiler executable checksum: 5998ce5f1765e99eea5269f4c1e38d44
COLLECT_GCC_OPTIONS='-v' '-o' 'test' '-mtune=generic' '-
march=i486'
 as -V -Qy -o /tmp/ccR1asmd.o /tmp/ccJnS0wf.s
GNU assembler version 2.20.1 (i486-linux-gnu) using BFD version
(GNU Binutils for Ubuntu) 2.20.1-system.20100303
COMPILER_PATH=/usr/lib/gcc/i486-linux-
gnu/4.4.3/:/usr/lib/gcc/i486-linux-gnu/4.4.3/:/usr/lib/gcc/i486-
linux-gnu/:/usr/lib/gcc/i486-linux-gnu/4.4.3/:/usr/lib/gcc/i486-
linux-gnu/:/usr/lib/gcc/i486-linux-gnu/4.4.3/:/usr/lib/gcc/i486-
linux-gnu/
LIBRARY_PATH=/usr/lib/gcc/i486-linux-gnu/4.4.3/:/usr/lib/gcc/i486-
linux-gnu/4.4.3/:/usr/lib/gcc/i486-linux-
gnu/4.4.3/../../../../lib/:/lib/../lib/:/usr/lib/../lib/:/usr/lib/
gcc/i486-linux-gnu/4.4.3/../../../:/lib/:/usr/lib/:/usr/lib/i486-
linux-gnu/
COLLECT_GCC_OPTIONS='-v' '-o' 'test' '-mtune=generic' '-
march=i486'
 /usr/lib/gcc/i486-linux-gnu/4.4.3/collect2 --build-id --eh-frame-
hdr -m elf_i386 --hash-style=both -dynamic-linker /lib/ld-
linux.so.2 -o test -z relro /usr/lib/gcc/i486-linux-
gnu/4.4.3/../../../../lib/crt1.o /usr/lib/gcc/i486-linux-
gnu/4.4.3/../../../../lib/crti.o /usr/lib/gcc/i486-linux-
gnu/4.4.3/crtbegin.o -L/usr/lib/gcc/i486-linux-gnu/4.4.3
-L/usr/lib/gcc/i486-linux-gnu/4.4.3 -L/usr/lib/gcc/i486-linux-
```

```
gnu/4.4.3/../../../../lib -L/lib/../lib -L/usr/lib/../lib
-L/usr/lib/gcc/i486-linux-gnu/4.4.3/../../.. -L/usr/lib/i486-
linux-gnu /tmp/ccR1asmd.o -lgcc --as-needed -lgcc_s --no-as-needed
-lc -lgcc --as-needed -lgcc_s --no-as-needed /usr/lib/gcc/i486-
linux-gnu/4.4.3/crtend.o /usr/lib/gcc/i486-linux-
gnu/4.4.3/../../../../lib/crtn.o
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $
================================================================
```

```
-H
    Print the name of each header file used, in addition to other
    normal activities. Each name is indented to show how deep in
    the `#include' stack it is. Precompiled header files are also
    printed, even if they are found to be invalid; an invalid
    precompiled header file is printed with `...x' and a valid one
    with `...!' .
```

```
Example:
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $ gcc -H -o test test.c
. /usr/include/stdio.h
.. /usr/include/features.h
... /usr/include/bits/predefs.h
... /usr/include/sys/cdefs.h
.... /usr/include/bits/wordsize.h
... /usr/include/gnu/stubs.h
.... /usr/include/bits/wordsize.h
.... /usr/include/gnu/stubs-32.h
.. /usr/lib/gcc/i486-linux-gnu/4.4.3/include/stddef.h
.. /usr/include/bits/types.h
... /usr/include/bits/wordsize.h
... /usr/include/bits/typesizes.h
.. /usr/include/libio.h
... /usr/include/_G_config.h
.... /usr/lib/gcc/i486-linux-gnu/4.4.3/include/stddef.h
.... /usr/include/wchar.h
... /usr/lib/gcc/i486-linux-gnu/4.4.3/include/stdarg.h
.. /usr/include/bits/stdio_lim.h
.. /usr/include/bits/sys_errlist.h
. /usr/include/math.h
.. /usr/include/bits/huge_val.h
.. /usr/include/bits/huge_valf.h
.. /usr/include/bits/huge_vall.h
.. /usr/include/bits/inf.h
.. /usr/include/bits/nan.h
.. /usr/include/bits/mathdef.h
... /usr/include/bits/wordsize.h
.. /usr/include/bits/mathcalls.h
.. /usr/include/bits/mathcalls.h
.. /usr/include/bits/mathcalls.h
Multiple include guards may be useful for:
```

```
/usr/include/bits/huge_val.h
/usr/include/bits/huge_valf.h
/usr/include/bits/huge_vall.h
/usr/include/bits/inf.h
/usr/include/bits/mathdef.h
/usr/include/bits/nan.h
/usr/include/bits/predefs.h
/usr/include/bits/stdio_lim.h
/usr/include/bits/sys_errlist.h
/usr/include/bits/typesizes.h
/usr/include/gnu/stubs-32.h
/usr/include/gnu/stubs.h
/usr/include/wchar.h
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $
```

==========

```
--version
    Print out GNU CPP's version number. With one dash, proceed to
    preprocess as normal. With two dashes, exit immediately.

Example:
jazz@linuxmint ~/Desktop/MTech/ACT/gcc $ gcc --version -o test
test.c
gcc (Ubuntu 4.4.3-4ubuntu5) 4.4.3
Copyright (C) 2009 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.

jazz@linuxmint ~/Desktop/MTech/ACT/gcc $
```

==========

Click here for more optimization related options:
http://gcc.gnu.org/onlinedocs/gcc/Preprocessor-Options.html#Preprocessor-Options

==========

4. Passing Options to the Assembler

You can pass options to the assembler.

`-Wa,`*option*
Pass *option* as an option to the assembler. If *option* contains commas, it is split into multiple options at the commas.

---

5. Options for Linking

These options come into play when the compiler links object files into an executable output file. They are meaningless if the compiler is not doing a link step.

`-l` *library*
Search the library named *library* when linking.

Example:

```
gcc thread.c -l pthread
```

`-s`
Remove all symbol table and relocation information from the executable.

Acknowledgement:

more options and information can be found at:
http://gcc.gnu.org/onlinedocs/gcc/Invoking-GCC.html