Revealing Internals of GNU Compiler Collection (GCC)

Zhiqiang Lin

Department of Computer Science
University of Texas at Dallas

March 21st, 2012
Outline

1. Overview
2. Internals, Architecture
3. USAGE
4. DEMO
5. Summary
Why Study GCC

Understand how machine code gets generated

Source code-based program analysis

- Compiler parses the source code
- Compiler knows exactly the behavior of the program
- Compiler has to know the machine details (when generating the code)
- Compiler-based security solutions
- Source code auditing (Integer overflow, buffer overflow, format string)

Understanding the tool-chain

- Compiler, linker, loader
Understanding the tool-chain

2009 June 20
14:39:38 GMT
Understanding the tool-chain
Understanding the tool-chain

2009 June 20
14:39:38 GMT

2009 June 20
16:39:38 Local (GMT+2)
GNU Compiler Collection – GCC

- **Developer(s)** GNU Project
- **Initial release** May 23, 1987
- **Stable release** 4.6.3 / March 1, 2012; 19 days ago
- **Written** in C, C++
- **Operating system** Cross-platform
- **Platform** GNU
- **Type** Compiler
- **License** GNU General Public License (version 3 or later)
- **Website** gcc.gnu.org
History of GCC

- GCC 1.xx
- GCC 2.xx
- EGCS
  - Enhanced (Experimental) GNU Compiler System
  - Fortran, Chill front-end.
  - C and C++ standard libraries.
    - libc, stdlibc++
  - Java, Ada, more optimization.
- GCC 4.0, Apr. 2005
- GCC 4.6.3 March 2012
History of GCC (cont’d)

- The newest version is not the best.
  - Compiler needs correct, stable and fast code.

- more stable and popular version:
  - gcc 2.6.3, 2.7.2.3
  - egcs-1.1.2
  - gcc 2.95.3
  - gcc 3.3
GCC Steering Committee

- To prevent any particular individual, group or organization from getting control over the project.
- 13 members
  1. Joe Buck (Synopsys)
  2. David Edelsohn (IBM)
  3. Kaveh R. Ghazi
  4. Jeffrey A. Law (Red Hat)
  5. Marc Lehmann (nethype GmbH)
  6. Jason Merrill (Red Hat)
  7. David Miller (Red Hat)
  8. Mark Mitchell (CodeSourcery / Mentor Graphics)
  9. Toon Moene (Koninklijk Nederlands Meteorologisch Instituut)
 10. Gerald Pfeifer (SUSE)
 11. Joel Sherrill (OAR Corporation)
 12. Jim Wilson (Cisco)
 13. Richard Stallman (Free Software Foundation)
The Structure of Compiler

Source program

- Lexical analyzer
- Syntax analyzer
- Semantic analyzer
- Intermediate code generator
- Code optimizer
- Code generator

Target program

Symbol-table manager

Error handler

Front-end

Back-end
Language Supported by GCC

- C, C++, Objective C
- Ada 95 (GNAT)
- Fortran 77, Fortran 95
- Pascal
- Modula-2, Modula-3
- Java (supported from gcc 3.0)
- Cobol
- Chill (Cygnus)
- ...
Machines Supported by GCC

GCC target processor families as of version 4.3 include:

- Alpha
- ARM
- Atmel AVR
- Blackfin
- H8/300
- HC12
- IA-32 (x86)
- IA-64
- MIPS
- Motorola 68000
- PA-RISC
- PDP-11
- PowerPC
- R8C/M16C/M32C
- SPARC
- SPU
- SuperH
- System/390/zSeries
- VAX
- x86-64

Lesser-known target processors supported in the standard release have included:

- 68HC11
- A29K
- ARC
- AVR32
- D30V
- DSP16xx
- ETRAX CRIS
- FR-30
- FR-V
- Intel i960
- IP2000
- M32R
- MCORE
- MIL-STD-1750A
- MMIX
- MN10200
- MN10300
- Motorola 88000
- NS32K
- ROMP
- Stormy16
- V850
- Xtensa

Additional processors have been supported by GCC versions maintained separately from the FSF version:

- Cortus APS3
- D10V
- EISC
- eSi-RISC
- Hexagon
- LatticeMico32
- LatticeMico8
- MeP
- MicroBlaze
- Motorola 6809
- MSP430
- NEC SX architecture
- Nios II and Nios
- OpenRISC 1200
- PDP-10
- PIC24/dsPIC
- System/370
- TIGCC (m68k variant)
- Z8000
Code Size of GCC

For GCC 2.7.2:

- Distributed front ends: 109,380 lines.
  - C: 22,415
  - C++: 72,112
  - Objective-C: 7,973 + 6,880 (library)
- Base compiler: 226,057 lines.
  - Optimizer: 56,581
- Total size: 566,556 lines.

GCC 3.2: 133Mbytes
Source: http://vmakarov.fedorapeople.org/spec/comparison.html. Here core is compiler itself (gcc directory) for GCC and LLVM itself. All is all gcc sources for GCC and LLVM plus gcc frontend sources for LLVM.
Compilation System

- Compilation system includes the phases
  - Preprocessor
  - Compiler
  - Optimizer
  - Assembler
  - Linker

- Compiler Driver coordinates these phases.
GCC Execution

 GCC
 -> Input file
     |    output file
     |    |
     |    |
 cpp       cc1        gas (assembler)  Id (linker)
 ->       ->         ->             ->
    g++       (assembler)

Overview
Internals, Architecture
USAGE
DEMO
Summary
The Structure of GCC

- Parsing
- TREE
- RTL
- Global Optimizations
  - Jump Optimization
  - Common Subexpr. Elimination
  - Loop Optimization
  - Data Flow Analysis
- Instruction Combining
- Instruction Scheduling
- Register Class Preferencing
- Register Allocation
- Peephole Optimizations
- Assembly

Programming Languages:
- C
- C++
- ObjC
- Fortran
NAME

gcc - GNU project C and C++ compiler

SYNOPSIS

gcc  [-c|-S|-E]  [-std=standard]
     [-g]  [-pg]  [-Olevel]
     [-Wwarn...]  [-pedantic]
     [-I...dir]  [-L...dir]
     [-D...macro[=defn]]  [-U...macro]
     [-f...option]  [-m...machine-option]
     [-o  outfile]  infile...
Options

--help   Display this information
     (Use '-v --help' to display command line options of sub-processes)
-dumpspecs  Display all of the built in spec strings
-dumpversion  Display the version of the compiler
-dumpmachine  Display the compiler’s target processor
-print-search-dirs  Display the directories in the compiler’s search path
-print-libgcc-file-name  Display the name of the compiler’s companion library
-print-file-name=<lib>  Display the full path to library <lib>
-print-prog-name=<prog>  Display the full path to compiler component <prog>
-print-multi-directory  Display the root directory for versions of libgcc
Options (cont'd)

- `print-multi-lib`: Display the mapping between command line options and multiple library search directories.
- `Wa,<options>`: Pass comma-separated `<options>` on to the assembler.
- `Wp,<options>`: Pass comma-separated `<options>` on to the preprocessor.
- `Wl,<options>`: Pass comma-separated `<options>` on to the linker.
- `Xlinker <arg>`: Pass `<arg>` on to the linker.
- `save-temps`: Do not delete intermediate files.
- `pipe`: Use pipes rather than intermediate files.
- `specs=<file>`: Override builtin specs with the contents of `<file>`.
- `std=<standard>`: Assume that the input sources are for `<standard>`.
- `B <directory>`: Add `<directory>` to the compiler’s search paths.
Options (cont’d)

- `b <machine>` Run gcc for target `<machine>`, if installed
- `V <version>` Run gcc version number `<version>`, if installed
- `v` Display the programs invoked by the compiler
- `E` Preprocess only; do not compile, assemble or link
- `S` Compile only; do not assemble or link
- `c` Compile and assemble, but do not link
- `o <file>` Place the output into `<file>
- `x <language>` Specify the language of the following input files

Permissible languages include: c c++ assembler none

‘none’ means revert to the default behaviour of guessing the language based on the file’s extension
Examples

- **-E**: preprocessor output.

- **-g**: debug information.

- **-O0**~**-O1**: optimization level.
  - default is **-O1**.

- **-x language**: input file’s language.
  - Ex: gcc -x java test.java

- **-S**: output assembly.
Examples (cont’d)

- **-m???**: special for target’s option.
  - Ex1: gcc -m68000 test.c
  - Ex2: gcc -mcpu=i686 test.c

- **-Idir**: header files will be searched in dir.
  - Ex: gcc -I/home/pschen/include test.c

- **-Ldir**: library will be searched in dir.
  - Ex: gcc -l/home/pschen/include test.c

- **-da**: output all RTL files.
At the prompt, type

gcc -ansi -Wall pgm.c

where pgm.c is the C program source file.

- **ansi** is a **compiler option** that tells the compiler to adhere to the **ANSI C standard**.

- **Wall** is an option to turn on all compiler **warnings** (best for new programmers).
The Result: a.out

- If there are no errors in pgm.c, this command produces an **executable file**, which is one that can be executed (run).
- The gcc compiler names the executable file **a.out**.
- To execute the program, at the prompt, type **./a.out**
- Although we call this process “compiling a program”, what actually happens is more complicated.
Outline

1. Overview
2. Internals, Architecture
3. USAGE
4. DEMO
5. Summary
Using GCC

Source File  *pgm.c*

Modified Source Code in RAM

Program Object Code File  *pgm.o*

Other Object Code Files (if any)

Executable File  *a.out*
Preprocessing

- Performed by a program called the **preprocessor**
- Modifies the source code (in RAM) according to **preprocessor directives (preprocessor commands)** embedded in the source code
- Strips comments and white space from the code
- The source code as stored on disk is **not** modified.
gcc -E pgm.c >pgm.i

pgm.i

```c
1 # 1 "pgm.c"
2 # 1 "<built-in>"
3 # 1 "<command-line>"
4 # 1 "pgm.c"
5 # 1 "/usr/include/stdio.h" 1 3 4
6 # 28 "/usr/include/stdio.h" 3 4
7 # 1 "/usr/include/features.h" 1 3 4
8 # 323 "/usr/include/features.h" 3 4
9 # 1 "/usr/include/bits/predefs.h" 1 3 4
10 # 324 "/usr/include/features.h" 2 3 4
11 # 356 "/usr/include/features.h" 3 4
12 # 1 "/usr/include/sys/cdefs.h" 1 3 4
13 # 353 "/usr/include/sys/cdefs.h" 3 4
14 # 1 "/usr/include/bits/wordsize.h" 1 3 4
15 # 354 "/usr/include/sys/cdefs.h" 2 3 4
16 # 357 "/usr/include/features.h" 2 3 4
17 # 388 "/usr/include/features.h" 3 4
18 # 1 "/usr/include/gnu/stubs.h" 1 3 4
...
846 extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__));
847 # 936 "/usr/include/stdio.h" 3 4
848
849 # 2 "pgm.c" 2
850 int main()
851 {
852 printf("pid=%d\n", getpid());
853 return 0;
854 }
```
Compilation

- Performed by a program called the **compiler**
- Translates the preprocessor-modified source code into **object code (machine code)**
- Checks for **syntax errors** and **warnings**
- Saves the object code to a disk file, if instructed to do so (we will not do this).
- If any compiler errors are received, no object code file will be generated.
- An object code file will be generated if only warnings, not errors, are received.
Linking

- Combines the program object code with other object code to produce the executable file.
- The other object code can come from the Run-Time Library, other libraries, or object files that you have created.
- Saves the executable code to a disk file. On the Linux system, that file is called `a.out`.
  - If any linker errors are received, no executable file will be generated.
gcc -v -o test0 test0.c
Reading specs from /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/specs
Configured with: ../configure –prefix=/usr –mandir=/usr/share/man
  –infodir=/usr/share/info –enable-shared –enable-threads=posix
  –disable-checking –with-system-zlib –enable-__cxa_atexit –host=i386-redhat-linux
Thread model: posix
gcc version 3.2.3 20030502 (Red Hat Linux 3.2.3-20)
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/cc1 -lang-c -v -D__GNUC__=3
-D__GNUC_MINOR__=2 -D__GNUC_PATCHLEVEL__=3
   -D__GXX_ABI_VERSION=102 -D__ELF__ -Dunix -D-gnu_linux__ -Dlinux
   -D__ELF__ -D_unix__ -D-gnu_linux__ -D_linux__ -D_unix -D_linux
   -Asystem=posix -D__NO_INLINE__ -D__STDC_HOSTED__=1 -Acpu=i386
   -Amachine=i386 -Di386 -D__i386 -D__386 -D__tune_i386__ test0.c -quiet
   -dumpbase test0.c -version -o /tmp/ccBPEyC.r.s
GNU CPP version 3.2.3 20030502 (Red Hat Linux 3.2.3-20) (cpplib) (i386 Linux/ELF)
GNU C version 3.2.3 20030502 (Red Hat Linux 3.2.3-20) (i386-redhat-linux)

compiled by GNU C version 3.2.3 20030502 (Red Hat Linux 3.2.3-20).
gcc -v

ignoring nonexistent directory "/usr/i386-redhat-linux/include"
#include "...
#include <...> search starts here:
/usr/local/include
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/include
/usr/include
End of search list.

as -V -Qy -o /tmp/ccaVmECA.o /tmp/ccBPEYcR.s
GNU assembler version 2.14.90.0.4 (i386-redhat-linux) using BFD version 2.14.90.0.4 20030523

/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/collect2 –eh-frame-hdr -m elf_i386
-dynamic-linker /lib/ld-linux.so.2 -o test0

/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/..../crt1.o
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/..../crti.o
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/crtbegin.o
-L/usr/lib/gcc-lib/i386-redhat-linux/3.2.3 -L/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/..../
/tmp/ccaVmECA.o -lgcc -lgcc_eh -lc -lgcc -lgcc_eh
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/crtend.o
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/..../crtnc.o
```assembly
    .file  "pgm.c"
    .section  .rodata
    .LC0:
        .string "pid=%d\n"
    .text
    .globl main
    .type  main, @function
main:
    .LFB0:
      .cfi_startproc
      pushq  %rbp
      .cfi_def_cfa_offset 16
      movq  %rsp, %rbp
      .cfi_offset 6, -16
      .cfi_def_cfa_register 6
      movl $0, %eax
      call getpid
      movl %eax, %edx
      movl $.LC0, %eax
      movl %edx, %esi
      movq %rax, %rdi
      movl $0, %eax
      call printf
      movl $0, %eax
      leave
      .cfi_def_cfa 7, 8
      ret
    .LFE0:
      .size  main, .-main
      .ident  "GCC: (Ubuntu/Linaro 4.5.2-8ubuntu4) 4.5.2"
```
zlin@zlin-desktop:/tmp$ objdump -d pgm.o

pgm.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:  55 push %rbp
  1:  48 89 e5 mov %rsp,%rbp
  4: b8 00 00 00 00 mov $0x0,%eax
  9: e8 00 00 00 00 callq e <main+0xe>
  e:  89 c2 mov %edx,%esi
 10: b8 00 00 00 00 mov $0x0,%eax
 15:  89 d6 mov %edx,%esi
 17:  48 89 c7 mov %rax,%rdi
 1a: b8 00 00 00 00 mov $0x0,%eax
 1f: e8 00 00 00 00 callq 24 <main+0x24>
 24: b8 00 00 00 00 mov $0x0,%eax
 29: c9 leaveq
 2a: c3 retq
zlin@zlin-desktop:/tmp$ objdump -d a.out
...
0000000000400544 <main>:
  400544:      55  push %rbp
  400545:      48 89 e5  mov %rsp,%rbp
  400548:      b8 00 00 00 00  mov $0x0,%eax
  40054d:      e8 f6 fe ff ff  callq 400448 <getpid@plt>
  400552:      89 c2  mov %eax,%edx
  400554:      b8 5c 06 40 00  mov $0x40065c,%eax
  400559:      89 d6  mov %edx,%esi
  40055b:      48 89 c7  mov %rax,%rdi
  40055e:      b8 00 00 00 00  mov $0x0,%eax
  400563:      e8 c0 fe ff ff  callq 400428 <printf@plt>
  400568:      b8 00 00 00 00  mov $0x0,%eax
  40056d:      c9  leaveq
  40056e:      c3  retq
  40056f:      90  nop
...
Outline

1 Overview
2 Internals, Architecture
3 USAGE
4 DEMO
5 Summary
Compiler is a key system components

GCC

Overview
Architecture
USAGE
References

- http://vmakarov.fedorapeople.org/spec/comparison.html
- GCC Internals http://gcc.gnu.org/onlinedocs/gccint/