

GCC : The GNU Compiler Collection

Austin LUG
June, 2008

Jason Schonberg

Overview

I. What is a Compiler?

II. Why use a Compiler?

III. Did the Compiler do a good job?

IV. What else can a Compiler do for me?

I. What is a Compiler?

A Compiler is a computer program that translates a high level language into machine code.

A short example using C

```
#include <stdio.h>
#include <math.h>

int main()
{
printf("The square root of 2 is %f\n",sqrt(2));
return 0;
}
```

```
$ gcc -Wall -o run sqrt.c
$ ./run
```

```
The square root of 2 is 1.414214
```

The GNU Compiler Collection Supports Many Programming Languages

C

C++

Ada

Java

Pascal

Fortran

The GNU Compiler Collection Supports Multiple Architectures

Mips, Alpha, Sparc, X86, X86-64, Itanium,
PowerPC, Motorola 68K, Motorola 68HC11,
Vax

And many more . . .

A little computer history

For those who may not remember what a vax looks like . . .



System introduced in 1977

VAX – the “Virtual Address eXtension” of the PDP-11's 16-bit architecture to a 32 bit architecture

II. *Why Use a Compiler?*

The alternative is to write in a low level language such as assembly or machine code. That's tedious, non-portable, error prone and not practical for applications of any significant size.

Compiler Flow

- 1) Parse human provided code
- 2) Translate to RTL
- 3) Optimize
- 4) Link to resolve references to external library calls
- 5) Generate machine code



Let's Consider Another Example

```
#include <stdio.h>

int main()
{
int x = 2;
printf("%d\n",5*x);
return 0;
}
```

```
$ gcc -Wall -o mul mul.c
$ ./mul
```

```
10
```

Let's see what the compiler did

```
$ gcc -c -O0 mul.c  
$ objdump -S mul.o
```

mul.o: file format elf32-i386

Disassembly of section .text:

00000000 <main>:

```
0: 8d 4c 24 04      lea  0x4(%esp),%ecx
4: 83 e4 f0         and  $0xffffffff0,%esp
7: ff 71 fc        pushl -0x4(%ecx)
a: 55             push  %ebp
b: 89 e5          mov  %esp,%ebp
d: 51            push  %ecx
e: 83 ec 24       sub  $0x24,%esp
11:c7 45 f8 02 00 00 00 movl $0x2,-0x8(%ebp)
18:8b 55 f8       mov  -0x8(%ebp),%edx
1b:89 d0         mov  %edx,%eax
1d:c1 e0 02     shl  $0x2,%eax
20:01 d0         add  %edx,%eax
22:89 44 24 04   mov  %eax,0x4(%esp)
26:c7 04 24 00 00 00 00 movl $0x0,(%esp)
2d:e8 fc ff ff ff call 2e <main+0x2e>
32:b8 00 00 00 00 mov  $0x0,%eax
37:83 c4 24     add  $0x24,%esp
3a:59         pop  %ecx
3b:5d         pop  %ebp
3c:8d 61 fc     lea  -0x4(%ecx),%esp
3f: c3         ret
```

Try again with higher optimization

```
$ gcc -c -O2 mul.c  
$ objdump -S mul.o
```

mul.o: file format elf32-i386

Disassembly of section .text:

00000000 <main>:

0:	8d 4c 24 04	lea 0x4(%esp),%ecx
4:	83 e4 f0	and \$0xffffffff,%esp
7:	ff 71 fc	pushl -0x4(%ecx)
a:	55	push %ebp
b:	89 e5	mov %esp,%ebp
d:	51	push %ecx
e:	83 ec 14	sub \$0x14,%esp
11:	c7 44 24 04 0a 00 00	movl \$0xa,0x4(%esp)
18:	00	
19:	c7 04 24 00 00 00 00	movl \$0x0,(%esp)
20:	e8 fc ff ff ff	call 21 <main+0x21>
25:	83 c4 14	add \$0x14,%esp
28:	31 c0	xor %eax,%eax
2a:	59	pop %ecx
2b:	5d	pop %ebp
2c:	8d 61 fc	lea -0x4(%ecx),%esp
2f:	c3	ret

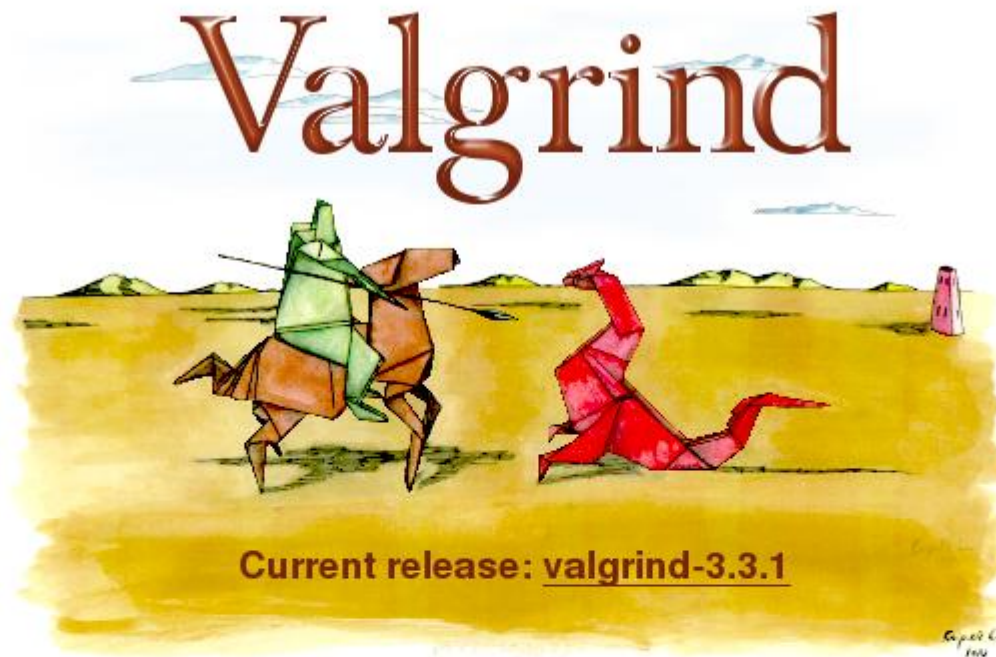
What else can the compiler do?

Use the `-g` flag and the compiler will introduce additional debugging information.

This is useful if you are trying to determine what's happening in misbehaving code.

Also useful with tools such as Valgrind.

www.valgrind.org



Valgrind is an award-winning instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.

The Valgrind distribution currently includes five production-quality tools: a memory error detector, a thread error detector, a cache and branch-prediction profiler, a call-graph generating cache profiler, and a heap profiler. It also includes two experimental tools: a data race detector, and an instant memory leak detector. It runs on the following platforms: X86/Linux, AMD64/Linux, PPC32/Linux, PPC64/Linux.

Other useful flags

-finline-functions

create local copies of simple functions instead of making a function call.

-funroll-loops

unroll those loops whose iteration count can be determined at compile time

-fstack-protector

emit extra code to check for buffer overflows such as stack smashing attacks.

-ftest-coverage

Create a side file for use with gcov.

-pg

Compile with code profiling to have the system determine where most of your run time is being spent.

-ansi

Compile with the ISO C90 standard

-static

On those systems that support it, build with static libraries

There are 100s of additional flags which can be used to control code generation, compiler optimizations as well as hardware and architecture specific features.

Consult the gcc man page for additional information.

Consider Again The Compiler Flow

- 1) Parse human provided code
- 2) Translate to RTL
- 3) Optimize
- 4) Link to resolve references to external library calls
- 5) Generate machine code

**What happens if step 5
generates machine code for
a different machine?**



Compilers

- ◆ Native Compiler

 - Runs on Machine A, builds code that runs on Machine A

- ◆ Cross Compiler

 - Runs on Machine A, builds code that runs on Machine B

Why use a cross compiler?

- ◆ The target machine is so new that a native compiler does not exist yet.
- ◆ The target machine has too little memory to run a native compiler.
- ◆ The target machine is too slow to do useful development.
- ◆ The target system uses a different architecture than the development system

Linux on the Sega Dreamcast

<http://linuxdc.sourceforge.net/>



The LinuxDC (Linux on Dreamcast) Project aims to fully support the Linux kernel and operating system on the Sega Dreamcast game console. We hope to provide tools, binaries, documentation, and application software that enable users of LinuxDC to utilize their Dreamcast in ways previously thought not possible.

CPU: 128-Bit Hitachi SuperH4 RISC
(360 Mips, 800MB/sec Data Throughput)
CPU speed: 200MHz
RAM: 26 Megabytes
(16MB main/8MB video/2 MB sound)

More Dreamcast info

<http://www.consoledatabase.com/consoleinfo/segadreamcast/index.html>

Announced in September 1997

Release in Japan November 1998

Release in America September 1999

(\$199.99 retail)

Release in Europe October 1999

Release in New Zealand November 1999

What's needed to build a compiler?

- ◆ The source code of GCC
And an older version to build the new version with.
- ◆ The source code of Glibc
Building C requires the C library.
- ◆ Binutils
This is the assembler, the linker, BFD, objdump, nm and others.
- ◆ Linux kernel source (for system headers)

Typical setup – specify a host and a target machine

After getting all the pieces together, building a cross compiler will look something like this:

```
$ configure --prefix=/usr/local/ --target=arm-  
linux --with-gnu-as --with-gnu-ld  
--host=i686-linux-elf  
$ make language=c
```

<http://www.kegel.com/crosstool/>

Other examples of Linux based embedded devices

- ◆ Phone systems (cell phones and PBXs)
- ◆ Robots
- ◆ Routers
- ◆ PDAs
- ◆ Digital music players

