Compilers and their Options

Ananya Mantravadi
CS19B1004

Options in GCC

- `-Wall:` Enables all the warnings in GCC
- `-E`: Stop after preprocessing, output preprocessor code
- `-S`: Stop after compilation, output assembly code
- `-c`: Produce only object files without linking
- `--help:` For displaying a list of all available options
- `-fsigned-char:` Treat char variables as signed
- `--version:` To display the version information
- `-g:` Generate complete debug information
- `-fexceptions:` To enable exception handling
- `-Q:` Prints out the function names as they are compiled and statistics
- `-time` (file_name): Prints CPU time taken by each subprocess during compilation
- `-march=`(cpu_name): Generates code that may not run on processors other than cpu_name
- `-ansi`: Equivalent to -std=c90 (C mode); -std=c++98 (C++ mode)
- `-std=`: To determine the standard language for compilation

Options in LLVM

- `-time:` To time individual commands
- `-###:` print but do not run the commands to run for compilation.
- `-fsyntax-only:` Stop after preprocessing, parsing and type checking stages.
- `-ftrapv:` Generates code to catch integer overflow errors.
- `--help`: For displaying a list of all available options
- `-print-file-name=`(file_name): To print the full library path of the file
- `-save-stats:` To save stats of code generation in the current directory
- `-Weverything:` Enables all the warnings
- `--version:` To display version information
- `--no-warnings:` To suppress all warnings
- `-funroll-loops:` To turn on loop unroller
- `-nocpp:` To disable all predefined and command line preprocessor macros

Frontends of GCC and LLVM

The frontend is a component of a compiler that is specific to a particular language. It is responsible for transforming high-level language to local intermediate form. Currently, the main GCC distribution contains frontends for C(gcc), C++(g++), Objective-C, Java(gcj), Fortran (gfortran), Ada (GNAT), Go (gccgo), and D(GDC). In addition to these, there are other frontends supporting Pascal, Mercury, and COBOL which are maintained separately. LLVM contains frontends for C, C++, Objective-C, Objective-C++ (Clang), C#, Ada, Fortran(flang), Haskell, Delphi, Julia, Java bytecode, Rust, Swift, Common Lisp, etc.

Code for different architectures (different backends) using GCC, through cross-compilation

1) ARM aarch64:

```
        .arch armv8-a
        .file   "main1.c"
        .text
        .align  2
        .global fib
        .type   fib, %function
fib:
.LFB13:
        .cfi_startproc
        stp     x29, x30, [sp, -32]!
        .cfi_def_cfa_offset 32
        .cfi_offset 29, -32
        .cfi_offset 30, -24
        mov     x29, sp
        stp     x19, x20, [sp, 16]
        .cfi_offset 19, -16
        .cfi_offset 20, -8
        mov     w19, w0
        mov     w20, 0
.L3:
        cmp     w19, 1
        bgt     .L2
        add     w0, w19, w20
        ldp     x19, x20, [sp, 16]
        ldp     x29, x30, [sp], 32
        .cfi_remember_state
        .cfi_restore 30
        .cfi_restore 29
        .cfi_restore 19
        .cfi_restore 20
        .cfi_def_cfa_offset 0
        ret
.L2:
        .cfi_restore_state
        sub     w0, w19, #1
        sub     w19, w19, #2
```

```
        bl      fib
        add     w20, w20, w0
        b       .L3
        .cfi_endproc
.LFE13:
        .size   fib, .-fib
        .section        .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string "Exec time = %g\n"
        .section        .text.startup,"ax",@progbits
        .align  2
        .global main
        .type   main, %function
main:
.LFB14:
        .cfi_startproc
        stp     x29, x30, [sp, -32]!
        .cfi_def_cfa_offset 32
        .cfi_offset 29, -32
        .cfi_offset 30, -24
        mov     x29, sp
        str     x19, [sp, 16]
        .cfi_offset 19, -16
        bl      clock
        mov     x19, x0
        mov     w0, 45
        bl      fib
        bl      clock
        sub     x0, x0, x19
        adrp    x1, .LC0
        add     x1, x1, :lo12:.LC0
        scvtf   d0, x0
        mov     w0, 1
        bl      __printf_chk
        mov     w0, 0
        ldr     x19, [sp, 16]
        ldp     x29, x30, [sp], 32
        .cfi_restore 30
        .cfi_restore 29
        .cfi_restore 19
        .cfi_def_cfa_offset 0
        ret
        .cfi_endproc
.LFE14:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
        .section        .note.GNU-stack,"",@progbits
```

2) x86-64:

```
        .file   "main1.c"
        .text
        .globl  fib
```

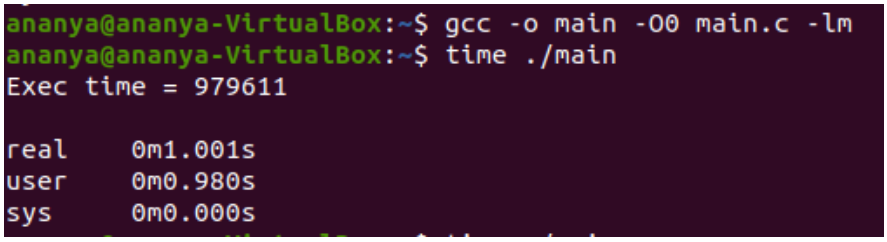
```
        .type   fib, @function
fib:
.LFB13:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        xorl    %ebp, %ebp
        pushq   %rbx
        .cfi_def_cfa_offset 24
        .cfi_offset 3, -24
        movl    %edi, %ebx
        pushq   %rcx
        .cfi_def_cfa_offset 32
.L3:
        cmpl    $1, %ebx
        jg      .L2
        leal    (%rbx,%rbp), %eax
        popq    %rdx
        .cfi_remember_state
        .cfi_def_cfa_offset 24
        popq    %rbx
        .cfi_def_cfa_offset 16
        popq    %rbp
        .cfi_def_cfa_offset 8
        ret
.L2:
        .cfi_restore_state
        leal    -1(%rbx), %edi
        subl    $2, %ebx
        call    fib
        addl    %eax, %ebp
        jmp     .L3
        .cfi_endproc
.LFE13:
        .size   fib, .-fib
        .section    .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string "Exec time = %g\n"
        .section    .text.startup,"ax",@progbits
        .globl  main
        .type   main, @function
main:
.LFB14:
        .cfi_startproc
        endbr64
        pushq   %rbx
        .cfi_def_cfa_offset 16
```

```
        .cfi_offset 3, -16
        call    clock@PLT
        movl    $45, %edi
        movq    %rax, %rbx
        call    fib
        call    clock@PLT
        leaq    .LC0(%rip), %rsi
        movl    $1, %edi
        subq    %rbx, %rax
        cvtsi2sdq %rax, %xmm0
        movb    $1, %al
        call    __printf_chk@PLT
        xorl    %eax, %eax
        popq    %rbx
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE14:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
        .section .note.GNU-stack,"",@progbits
        .section .note.gnu.property,"a"
        .align 8
        .long    1f - 0f
        .long    4f - 1f
        .long    5
0:
        .string  "GNU"
1:
        .align 8
        .long    0xc0000002
        .long    3f - 2f
2:
        .long    0x3
3:
        .align 8
```

Optimization levels

Program 1:

Default (no optimization) using O0:



No. of lines of assembly code in main.s after running `gcc -S -O0 main.c` = 132

Using O1:

```
ananya@ananya-VirtualBox:~$ gcc -o main -O1 main.c -lm
ananya@ananya-VirtualBox:~$ time ./main
Exec time = 316519

real    0m0.336s
user    0m0.313s
sys     0m0.004s
```

No. of lines of assembly code in main.s after running `gcc -S -O1 main.c` = 104

Using O2:

```
ananya@ananya-VirtualBox:~$ gcc -o main -O2 main.c -lm
ananya@ananya-VirtualBox:~$ time ./main
Exec time = 1

real    0m0.001s
user    0m0.001s
sys     0m0.000s
```

No. of lines of assembly code in main.s after running `gcc -S -O2 main.c` = 84

Using O3:

```
ananya@ananya-VirtualBox:~$ gcc -o main -O3 main.c -lm
ananya@ananya-VirtualBox:~$ time ./main
Exec time = 1

real    0m0.001s
user    0m0.001s
sys     0m0.000s
```

No. of lines of assembly code in main.s after running `gcc -S -O3 main.c` = 84

Using Os:

```
ananya@ananya-VirtualBox:~$ gcc -o main -Os main.c -lm
ananya@ananya-VirtualBox:~$ time ./main
Exec time = 1

real    0m0.001s
user    0m0.001s
sys     0m0.000s
```

No. of lines of assembly code in main.s after running `gcc -S -Os main.c` = 74

Program 2:

Default (no optimization) using O0:

```
ananya@ananya-VirtualBox:~$ gcc -o main1 -O0 main1.c -lm
ananya@ananya-VirtualBox:~$ time ./main1
Exec time = 9.26242e+06

real    0m10.827s
user    0m9.243s
sys     0m0.020s
```

No. of lines of assembly code in main1.s after running `gcc -S -O0 main1.c` = 99

Using O1:

```
ananya@ananya-VirtualBox:~$ gcc -o main1 -O1 main1.c -lm
ananya@ananya-VirtualBox:~$ time ./main1
Exec time = 5.94508e+06

real    0m6.046s
user    0m5.934s
sys     0m0.012s
```

No. of lines of assembly code in main1.s after running `gcc -S -O1 main1.c` = 90

Using O2:

```
ananya@ananya-VirtualBox:~$ gcc -o main1 -O2 main1.c -lm
ananya@ananya-VirtualBox:~$ time ./main1
Exec time = 4.68804e+06

real    0m5.361s
user    0m4.689s
sys     0m0.000s
```

No. of lines of assembly code in main1.s after running `gcc -S -O2 main1.c` = 99

Using O3:

```
ananya@ananya-VirtualBox:~$ gcc -o main1 -O3 main1.c -lm
ananya@ananya-VirtualBox:~$ time ./main1
Exec time = 4.56211e+06

real    0m4.624s
user    0m4.563s
sys     0m0.000s
```

No. of lines of assembly code in main1.s after running `gcc -S -O3 main1.c` = 123

Using Os:

```
ananya@ananya-VirtualBox:~$ gcc -o main1 -Os main1.c -lm
ananya@ananya-VirtualBox:~$ time ./main1
Exec time = 5.39775e+06

real    0m5.555s
user    0m5.390s
sys     0m0.008s
```

No. of lines of assembly code in main1.s after running `gcc -S -O3 main1.c` = 89

Using Ofast:

```
ananya@ananya-VirtualBox:~$ gcc -o main1 -Ofast main1.c -lm
ananya@ananya-VirtualBox:~$ time ./main1
Exec time = 4.50516e+06

real    0m4.589s
user    0m4.506s
sys     0m0.000s
```

No. of lines of assembly code in main1.s after running `gcc -S -Ofast main1.c` = 123

Here, user time is the time the CPU spent in running the executable. From the above findings, we observe that -O1,-O2,-O3 give increased speed-ups as compared to -O0 which has no optimization. We notice an optimization-size trade-off when Program 2 is compiled with -O3. It turns on more expensive optimizations and gives the best speed up but produces the largest relative size of the executable(123). We also notice that -Os selects optimizations that reduce the size of an executable, and gives the smallest possible size of the code generated (74 and 89 respectively). GCC does not provide -Oz optimization level and hence we see an error, it is available in LLVM.-Og is used for optimizing debugging experience rather than speed or size. -Ofast optimizes for speed disregarding exact standards compliance, and we observe that it is faster but has a larger code size.