第1章. 编译器认知实验

1.1主流编译器介绍

GCC 和 LLVM 是目前工业界使用广泛使用的两个编译器，学习使用这些编译器并观察这些编译器的中间输入和输出，能够帮助了解编译器的功能和工作过程，可以为编译器的构造奠定良好的基础。因此本节首先对这两个编译器进行详细的介绍。

1.1.1. GCC

GCC 是 GNU 的一个开源项目，支持 C、C++、Objective-C、Fortran 和 Ada 等，以及这些语言对应的运行时库。GCC 最初是作为 GNU operating system 编写的编译器[1]。

GCC 刚开始表示 GNU C Compiler 的缩写，第一个版本于 1987 年 3 月 22 日由 Richard M. Stallman 通过 MIT 的一个 FTP 服务器发布，最初只支持 VAX 和 Sun 的机器。第一个版本在 68020 上能够对自己进行编译，并成功完成了 Emacs 的编译。1997 年，部分开发人员觉得 GCC 的开发过于保守，于是创建了一个新的分支，能够以更快的速度对编译器进行更新，这个新的分支就是 EGCS 开源项目。1999 年 4 月份，经过长期的沟通和协商，EGCS 和 GCC 最终实现了合并，EGCS 成为 GNU 官方的 GCC 项目，同时 GCC 从“GNU C Compiler”变成了“GNU Compiler Collection”，所以今天的 GCC 是后者的缩写。两个项目合并之后发布的第一个版本是 GCC 2.95。

GCC 的最新版本可以从 https://gcc.gnu.org/releases.html 下载编译运行。

例如对如下的代码（main.c）：

```
#include<stdio.h>

int main(void)
{
    printf("Hello World!\n");
    return 0;
}
```

最简单的编译命令是：

gcc main.c

默认生成的可执行文件是 a.out，如果需要指定输出文件，则需要使用如下的命令：

gcc main.c –o main

使用-Wall 选项可以输出所有的警告信息：

gcc -Wall main.c -o main

输出 C 源码的预处理结果需要使用-E 选项，默认是输出到控制台，可以将其重定向到文件保存：

gcc -E main.c > main.i

如果只想输出汇编代码而非二进制代码，则需要使用-S 选项：

gcc -S main.c

使用-C 选项只生成 C 程序对应的目标文件，但是不进行链接：

gcc -C main.c

输出所有的中间文件需要使用-save-temps 选项：

gcc -save-temps main.c

与共享库进行链接，应使用-l 选项再加上库程序的名字，例如，程序引用了 math 库中的 pow 函数，则应该使用如下的命令编译和链接：

gcc -Wall main.c -o main –lm

编译生成共享库时，为了保证所生成的库可以被加载到合法的任意地址，需要使用-fPIC 选项，例如如下的命令生成一个名为 test.so 的共享库：

gcc -c -Wall -Werror -fPIC test.c

gcc -shared -o test.so test.o

如果想查看编译器是怎么编译源程序的，以及每一步所使用的命令是什么，则应该使用-V 选项：

gcc -Wall -v main.c -o main

使用-ansi 可以指定编译器编译源码时遵循的语言标准，例如如下代码按照 ISO C89 对源码进行编译：

gcc -Wall -ansi main.c -o main

如果有一些宏需要在编译时代入，则可以使用-D 选项，例如如下的代码：

```
#include<stdio.h>

int main(void)
{
#ifdef MY_MACRO
   printf("\n Macro defined \n");
#endif
   char c = -10;
   // Print the string
     printf("\n The Geek Stuff [%d]\n", c);
     return 0;
}
```

使用如下的命令编译：

gcc -Wall -DMY_MACRO main.c -o main

则会打印输出相应的语句。

有些时候警告中往往预示着一些潜在的错误，因此部分项目在编译时要求将警告当做错误看到，使用-Werror 选项可以完成上述功能：

gcc -Wall -Werror main.c -o main

经过优化的代码调试时可能会出现找不到声明的变量、控制流程发生变化、有些语句被直接跳过等现象。总而言之，不可能对编译之后的代码进行调试。为了让编译器生成额外的信息以利于程序的调试（例如使用 gdb），则需要使用-g 选项。

gcc -Wall -Werror –g main.c -o main

使用-O0 选项可以禁用所有编译器优化，使用-O1、-O2 和-O3 逐步打开更多的优化选项。-Og 则在保证调试工作的同时完成了少量的优化。有关具体的信息

可以查看GCC具体的优化选项,以及其他的优化内容请查阅GCC的相关文档[2]。

1.1.2. LLVM 与 Clang

LLVM项目是2000年在UIUC由Vikram Adve和Chris Lattner教授共同发起的，该项目最初是一个以研究为目标的编译器基础框架。2005 年，Chris Lattner 就职于苹果公司并领导一个专门为苹果开发系统研发编译器的工作组，该工作组基于LLVM支持苹果各类系统研发,并使LLVM成为苹果macOS和iOS开发工具的一部分。从2013年开始，Sony开始使用在PS4的SDK中使用LLVM前端Clang[3]。

LLVM最初是Low Level Virtual Machine的缩写，由于LLVM已经演化为一个包括LLVM IR、LLVM调试器、LLVM C++标准库（完全支持C++11和C++14版本）的项目群和一个广泛参与的编译社区，现在人们已经逐渐淡化其“Low Level Virtual Machine的最初含义。LLVM目前由LLVM基金会管理，由于对LLVM的杰出贡献，Vikram Adve、Chris Lattner和Even Cheng于2012年获得了ACM Software System Award。

LLVM最初打算使用GCC作为其编译器前端，但是GCC却给LLVM的开发人员和苹果公司带来不少困难。GCC 由于历史悠久，代码规模庞大，模块关系复杂，进行前端的嫁接和整合自然不是一件简单的事情,正如一位长期从事GCC开发的人所说,“试图让河马跳舞并不是很有趣”。其中一个问题是苹果的软件系统主要依赖于Objective-C，但是Objective-C的前端在GCC的开发过程中优先级并不是那么高；另外，将GCC整合进苹果的开发IDE也存在开源协议的问题：GCC最终采用了GPL V3开源协议，要求所有的开发者开源其对开源项目的扩展或者修改，然而LLVM则采用类BSD开源协议，没有这样的要求。最终苹果决定研发一个新的编译器前端支持C、Objective-C和C++，这样Clang这个项目就与2007年7月开源了。Clang支持OpenMP和OpenCL等编程框架，后端对接的是LLVM，从LLVM 2.6以后就随着LLVM一起发布和更新。

Clang与GCC高度兼容，GCC的大多数命令行接口和选项在Clang中都得到了很好的支持，Clang也实现了许多GNU对语言的扩展并且默认支持，实现了许多编译器intrinsics纯粹是为了保持与GCC的兼容性，例如Clang实现了C11

中的原子原语，同时也实现了 GCC 的“__sync_*”以兼容 GCC 和 libstdc++。很多时候，我们可以使用 Clang 直接替换 GCC 编译项目。

在 Ubuntu 虚拟机中编译 LLVM 占用空间约 40 GB，此外还需要预留 swap 空间，建议在构建虚拟机时预留 60GB 以上的空余空间。下载 LLVM 源码后，在父目录下创建 build 目录,用于存放构建的中间产物和最终的可执行文件。进入 build 目录，执行：

cmake ../llvm -DLLVM_TARGETS_TO_BUILD=X86 -DCMAKE_BUILD_TYPE=Debug

然后执行 make 即可开始编译，对于 4 核 CPU，可以执行 make –j4 加速编译过程。

1.2实验目的

本实验的目的是了解工业界常用的编译器 GCC 和 LLVM，熟悉编译器的安装和使用过程，观察编译器工作过程中生成的中间文件的格式和内容，了解编译器的优化效果，为编译器的学习和构造奠定基础。

1.3实验内容与方法

本实验主要的内容为在 Linux 平台上安装和运行工业界常用的编译器 GCC 和 LLVM，如果系统中没有安装，则需要首先安装编译器，安装完成后编写简单的测试程序，使用编译器编译，并观察中间输出结果。

对于 GCC 编译器，完成编译器安装和测试程序编写后，按如下步骤完成：

- 查看编译器的版本
- 使用编译器编译单个文件
- 使用编译器编译链接多个文件
- 查看预处理结果：gcc -E hello.c -o hello.i
- 查看语法分析树：gcc -fdump-tree-all hello.c
- 查看中间代码生成结果：gcc -fdump-rtl-all hello.c
- 查看生成的目标代码（汇编代码）：gcc –S hello.c –o hello.s

对于 LLVM 编译器，完成编译器安装和测试程序编写后，按如下步骤完成：

- 查看编译器的版本
- 使用编译器编译单个文件
- 使用编译器编译链接多个文件
- 查看编译流程和阶段：clang -ccc-print-phases test.c -c
- 查看词法分析结果：clang test.c -Xclang -dump-tokens
- 查看词法分析结果 2：clang test.c -Xclang -dump-raw-tokens
- 查看语法分析结果：clang test.c -Xclang -ast-dump
- 查看语法分析结果 2：clang test.c -Xclang -ast-view
- 查看编译优化的结果：clang test.c -S -mllvm -print-after-all
- 查看生成的目标代码结果：clang test.c –S

具体按照如下步骤完成：

（1）编译器安装

默认 Linux 系统中已经安装了 GCC，登录系统直接使用即可，相关使用文档可以见前文或者参考 GCC 官方文档。

Clang 和 LLVM 需要手动安装，Ubuntu 系统中使用下列命令安装：

sudo apt-get install llvm

sudo apt-get install clang

也可以下载 llvm 和 clang 的源码，手动编译安装。对应的下载地址为：http://releases.llvm.org/download.html。

（2）编写测试程序

可以使用语言认知部分的 C 语言程序，或者其他的 C 语言程序作为编译器的输入进行观测。需要注意的是，不但要求对单个 C 文件进行编译，还要求对多个 C 文件一起编译和链接。

（3）运行编译器进行观测

分别运行 GCC 和 LLVM 编译器，首先要学习编译器的基本使用方法，其次要通过编译选项查看分析编译器的中间结果，及其与输入源码之间的对应关系，最后使用-O0、-O1、-O2 和-O3 分别使用两个编译器对输入程序进行优化编译，并对比 GCC 和 LLVM 优化后程序的运行效率。

（4）编写实验报告

根据观测结果编写实验报告。

注意事项：

- GCC 编译器的优化选项是大写字母 O，小写字母 o 用于指定输出文件名字，注意不要混淆；
- “clang test.c -Xclang -ast-view”需要从源码编译 clang 和 LLVM 才能正确运行，因为 clang 在调试版本中才有这个选项的支持。LLVM 和 clang 的编译需要较大的内存。
- “gcc -fdump-tree-all hello.c”生产的文件比较多；
- “gcc -fdump-rtl-all”生成的文件比较多；
- 并不是所有的源程序，经过优化之后性能都有提升，甚至会出现-O3 优化之后的性能低于-O2 性能的情况；
- 一般情况下，LLVM 优化后的程序运行效率较高，但不是对所有的程序都成立。

1.4实验提交内容

本实验仅提交实验报告，具体包括如下内容：

- 实验目的和内容
- 实现的具体过程和步骤
- GCC 运行结果分析
- LLVM 运行结果分析
- GCC 与 LLVM 对比分析
- 实验心得体会

1.5参考文献

[1]. https://gcc.gnu.org/wiki/History

[2]. https://gcc.gnu.org/onlinedocs/gcc-9.2.0/gcc/Optimize-Options.html#Optimize-Options

[3]. https://en.wikipedia.org/wiki/LLVM

[4]. https://cppinsights.io/

[5]. https://godbolt.org/

[6]. https://caiorss.github.io/C-Cpp-Notes/Tooling.html