# 上机作业 A3：GCC 与 Clang/LLVM 优化比较

**布置周：第 4 周　9 月 27 日**
**提交周：第 6 周　10 月 11 日**

**目标：**通过本次练习，希望同学们掌握 gcc 与 Clang/LLVM 的安装方法，了解 gcc 及 Clang/LLVM 的常用编译优化选项，并掌握编译后代码的性能比较方法。

## 1. 安装 gcc（建议安装 9.3.0 版本）

在 Ubuntu 系统上安装 gcc/g++开发工具包，确保 gcc 和 g++在可执行文件的搜索路径上，并且可以正确运行，e.g.

```
bhuang@LAPTOP-BHUANG:~$ which gcc g++
/usr/bin/gcc
/usr/bin/g++
```

如何验证 gcc 和 g++版本? E.g.

$ gcc -- version

```
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 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.
```

$ g++ -- version

```
g++ (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 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.
```

## 2. 安装 Clang/LLVM（建议安装 10.0.0 版本）

在 Ubuntu 系统上安装 clang/clang++开发工具包，确保 clang 和 clang++在可执行文件的搜索路径上，并且可以正确运行，e.g.

```
bhuang@LAPTOP-BHUANG:~$ which clang clang++
/usr/bin/clang
/usr/bin/clang++
```

如何验证 clang 和 clang++的版本

$ clang -- version

```
clang version 10.0.0-4ubuntu1
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
```

$ clang++ -- version

```
clang version 10.0.0-4ubuntu1
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
```

# 3. 安装性能测试集

我们用的测试集从 https://gitlab.com/chriscox/CppPerformanceBenchmarks 上下载，并经过一定的修改。为了缩短测试时间，对一些测试例我们有意识地减少了测试的迭代次数。

把 CppPerformanceBenchmarks.tar 解包到自己的测试目录下即可。查看 makefile，可以发现里面包含了好些不同的测试。

# 4. 编译器及优化选项的组合测试

- 选取某一个具体的测试（e.g. lookup_table, loop_fusion etc.），修改 makefile，使得 makefile **仅编译及运行你所关注的测试例 （把其它测试例删掉）**
  主要修改
  ```
  BINARIES = machine \
             ….
  ```
  及
  ```
  report:   $(BINARIES)
            echo "##STARTING Version 1.0" > $(REPORT_FILE)
            date >> $(REPORT_FILE)
            echo "##Compiler: $(CC) $(CXX)" >> $(REPORT_FILE)
            echo "##CFlags: $(CFLAGS)" >> $(REPORT_FILE)
            echo "##CPPFlags: $(CPPFLAGS)" >> $(REPORT_FILE)
            echo "System Information collected by program: " >> $(REPORT_FILE)
            ./machine >> $(REPORT_FILE)
            …
  ```
  两部分的相关行。
- 如果你选取的测试例的运行时间太短(导致不同编译优化选项间的数据无法进行合理比较)，可以修改此测试对应 cpp 文件中 iterations 变量的值，一旦确定了 iterations 变量的值，在后续的测试中不能再修改此值。相应地，如果你选取的测试例的运行时间太长，则可以减少此测试对应 cpp 文件中 iterations 变量的值。

  ```
  // this constant may need to be adjusted to give reasonable minimum times
  // For best results, times should be about 1.0 seconds for the minimum test run
  // on 3Ghz desktop CPUs, 4000k iterations is about 1.0 seconds
  int iterations = 4000;
  ```

- 对于一个特定的测试例，按照下面的测试组合进行测试并进行数据收集，由于每一次测试的结果都存在 report.txt 文件内，需要在每次测试运行完把 report.txt 进行重命名以保留此次测试的数据，比如我们可以用以下命令保留用 gcc 在-O0 编译优化选项下收集的测试数据：

```
$mv report.txt report.txt.gcc-O0
```

**测试组合**:

**编译器**: { gcc/g++, clang/clang++}

**编译器优化选项**: {-O0, -O1, -O2}

共有 2*3=6 种组合

- 编译器用 CC 及 CXX 两个宏来控制，编译优化选项用 OPTLEVEL 来控制。CC 和 CXX 需要保持一致，比如说 CC 选 clang，则 CXX 必须为 clang++。运行一个具体测试的命令如下（以 clang 编译器，-O1 编译优化选项为例）:

```
$make clean
$make report CC=clang CXX=clang++ OPTLEVEL=-O1
$mv report.txt report.txt.clang-O1
```

同时记下此组合下编译出来测试例的 code size。比如选择了 loop_interchange 测试例，则记下在“clang”+“-O1”的组合下 loop_interchange 测试例的 code size 为 109368 字节。

```
-rwxr-xr-x  1 bhuang bhuang  109368 Aug 26 09:50 loop_interchange
```

# 5. 测试数据的分析

请针对选定的测试例及收集到的 6 个测试数据结果文件（以及相应测试例编译后的 code size），进行数据分析，总结分析洞见，撰写分析报告。要求:

- 需要对不同组合下测试例的 code size 及运行性能进行分析
- 尽量用可视化的形式来呈现分析结果
- 分析洞见的总结尽量简明扼要，如有可能请加上对所总结洞见的合理解释