Homework 3: GCC 与 Clang/LLVM 优化比较

杨浩然 10195501441

操作系统： ubuntu 20.04

2021-9-27

安装 gcc

```
younghojan@younghojan-XPS-15-7590:~$ which gcc g++
/usr/bin/gcc
/usr/bin/g++
younghojan@younghojan-XPS-15-7590:~$ 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.

younghojan@younghojan-XPS-15-7590:~$ 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.
```

已安装 gcc 9.3.0 和 g++ 9.3.0

安装 clang/LLVM

```
younghojan@younghojan-XPS-15-7590:~$ which clang clang++
/usr/bin/clang
/usr/bin/clang++
younghojan@younghojan-XPS-15-7590:~$ clang --version
clang version 10.0.0-4ubuntu1
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
younghojan@younghojan-XPS-15-7590:~$ clang++ --version
clang version 10.0.0-4ubuntu1
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
```

已安装 clang 10.0.0 和 clang++ 10.0.0

安装性能测试集

编译器及优化选项的组合测试

测试数据的分析

测试组合共 6 种：{ gcc/g++, clang/clang++} 与 {-O0, -O1, -O2} 进行搭配。

优化等级

不同的优化等级对应不同的优化效果：

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Os	optimization for code size		--		++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

+increase ++increase more +++increase even more -reduce --reduce more ---reduce even more

- -O0

  > optimize for compile time and debug-ability. Most performance optimizations are disabled. This is the default for GCC.

  对于 -O0 来说，gcc 只优化了编译时间和 debug 能力，大多性能优化的选项是关闭的。

- -O1

  > "pick the low hanging fruit in terms of performance, without impacting compilation time too much."
  >
  > Optimizing compilation takes somewhat more time, and a lot more memory for a large function.
  >
  > With -O, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

  编译时进行优化会多消耗一些时间，对于大的函数会消耗更多的内存。
  对于 -O1 来说，gcc 在不太影响编译时间的前提下，在性能优化选择容易实现的目标。gcc 会尝试减小 codesize 和运行时间。

  -O1(or -O) 打开了这些优化选项

```
-fauto-inc-dec
-fbranch-count-reg
-fcombine-stack-adjustments
-fcompare-elim
-fcprop-registers
-fdce
-fdefer-pop
-fdelayed-branch
-fdse
-fforward-propagate
-fguess-branch-probability
-fif-conversion
-fif-conversion2
-finline-functions-called-once
-fipa-profile
-fipa-pure-const
-fipa-reference
-fipa-reference-addressable
-fmerge-constants
-fmove-loop-invariants
-fomit-frame-pointer
-freorder-blocks
-fshrink-wrap
-fshrink-wrap-separate
-fsplit-wide-types
-fssa-backprop
-fssa-phiopt
-ftree-bit-ccp
-ftree-ccp
-ftree-ch
-ftree-coalesce-vars
-ftree-copy-prop
-ftree-dce
-ftree-dominator-opts
-ftree-dse
-ftree-forwprop
-ftree-fre
-ftree-phiprop
-ftree-pta
-ftree-scev-cprop
-ftree-sink
-ftree-slsr
-ftree-sra
-ftree-ter
-funit-at-a-time
```

- -O2

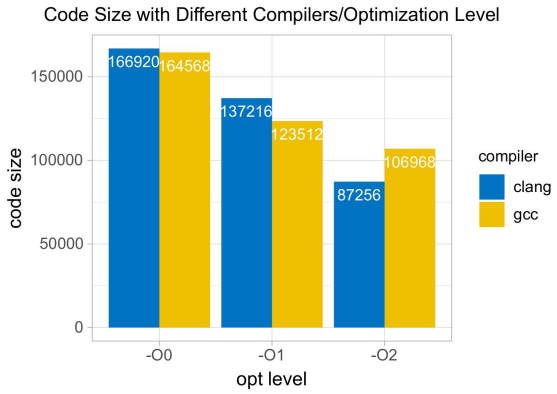
> "optimize for performance more aggressively, but not at the cost of larger code size."
>
> Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to -O, this option increases both compilation time and the performance of the generated code.
>
> -O2 turns on all optimization flags specified by -O and more.

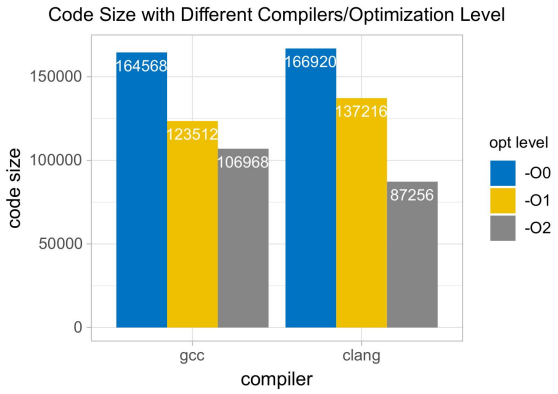
对于 -O2 来说，gcc 更积极地优化性能，但不会以更大的代码量为代价。

除了 -O1 打开的所有优化选项外， -O2 还打开了以下优化选项：

```
-falign-functions  -falign-jumps
-falign-labels  -falign-loops
-fcaller-saves
-fcode-hoisting
-fcrossjumping
-fcse-follow-jumps  -fcse-skip-blocks
-fdelete-null-pointer-checks
-fdevirtualize  -fdevirtualize-speculatively
-fexpensive-optimizations
-ffinite-loops
-fgcse  -fgcse-lm
-fhoist-adjacent-loads
-finline-functions
-finline-small-functions
-findirect-inlining
-fipa-bit-cp  -fipa-cp  -fipa-icf
-fipa-ra  -fipa-sra  -fipa-vrp
-fisolate-erroneous-paths-dereference
-flra-remat
-foptimize-sibling-calls
-foptimize-strlen
-fpartial-inlining
-fpeephole2
-freorder-blocks-algorithm=stc
-freorder-blocks-and-partition  -freorder-functions
-frerun-cse-after-loop
-fschedule-insns  -fschedule-insns2
-fsched-interblock  -fsched-spec
-fstore-merging
-fstrict-aliasing
-fthread-jumps
-ftree-builtin-call-dce
-ftree-pre
-ftree-switch-conversion  -ftree-tail-merge
-ftree-vrp
```

codesize

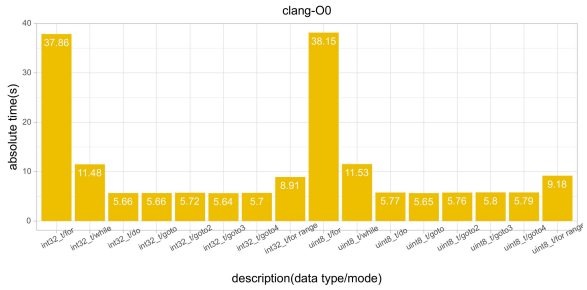


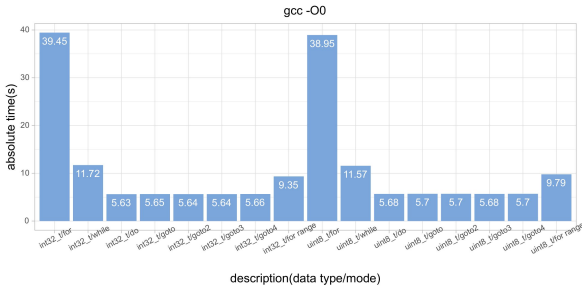


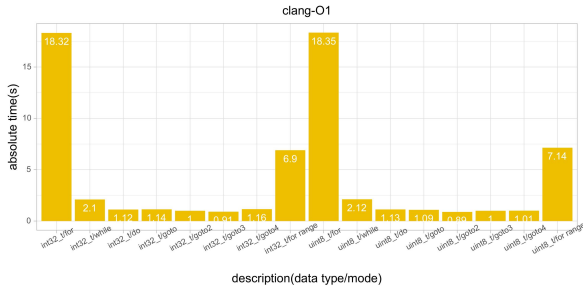
观察到：

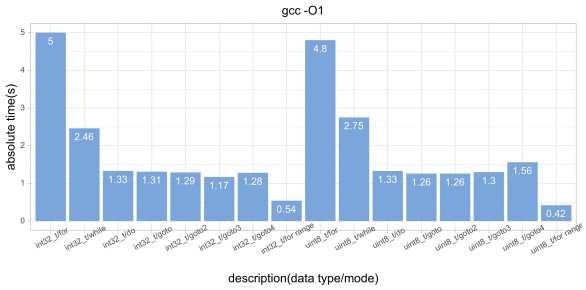
1. 考虑编译器的种类，在 -O0 和 -O1 的优化等级下，使用 gcc/g++ 生成代码的尺寸小于使用 clang/clang++ 生成代码的尺寸；
   而在 -O2 的优化等级下，使用 clang/clang++ 生成代码的尺寸小于使用 gcc/g++ 生成代码的尺寸。
2. 考虑优化等级的高低，无论使用何种编译器，优化等级越高，生成代码的尺寸越小。

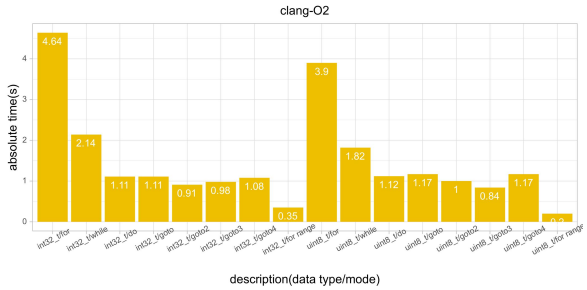
run time

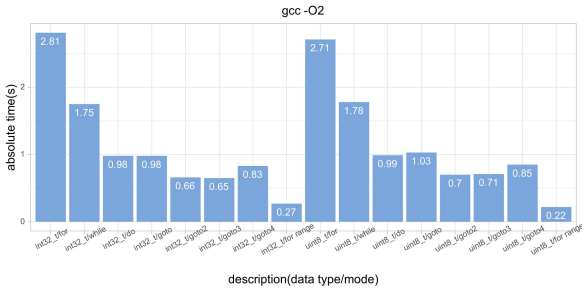


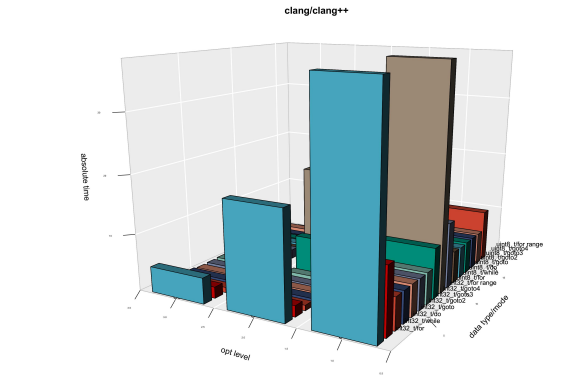


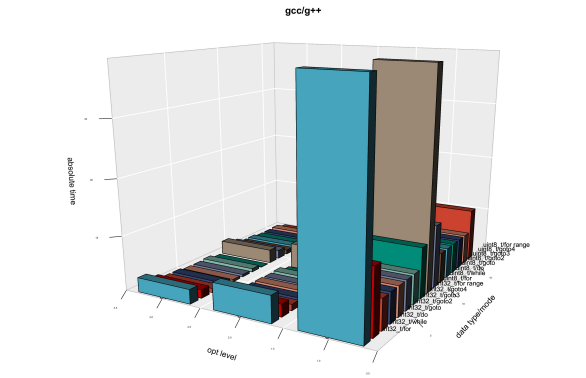












从图像上来看：

- 无论数据类型是 `int32_t` 或 `uint8_t`，无论优化级别是 `-O0` 或 `-O1` 或 `-O2`，运行时间最长的循环方式总是 `for`。
- 在 `-O1` 优化级别下，使用 clang/clang++ 编译的程序，运行时间最短的循环方式是 `for range`。
- 在 `-O2` 优化级别下，使用 clang/clang++ 和 gcc/g++ 编译的程序，运行时间最短的循环方式是 `for range`。