RISCV 工具链数据报第四期：

第一/二期数据勘误，以及 Berkeley 缩减静态 codesize TR 解读

在本期数据报中，我们将对 RISCV 指令集上 GCC 和 LLVM-Clang 编译器（以下简称 Clang）的 codesize 对比，以及 RISCV 指令集和 ARM 指令集在 GCC 编译器上的 codesize 对比，使用更加严谨的优化编译选项进行再次报告，并将结合 ARM 官方公开的 ARM Compiler 优化数据来分析 RISCV 的潜在优化空间。本期报告还将解读来自 Berkeley CS 系的技术报告《Reduce Static Code Size and Improve RISC-V Compression》。

关于优化选项使用的说明，请参考第三期数据报第一部分。

关于本期数据报所使用的编译器的构建/下载方式、源代码版本/下载链接，测试方法，请参考第一、二期数据报的相关详细说明。

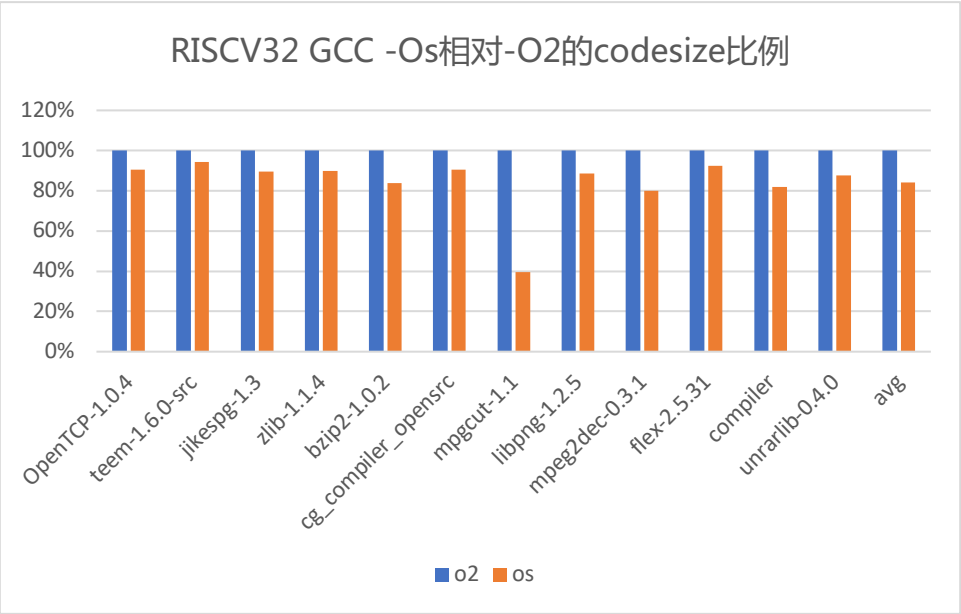
一、RISCV32 和 RISCV64 指令集下，对 GCC 和 Clang，以-O2 为基准，开启 codesize 优化后的 codesize 缩减程度评估。（对应第一期数据报 Part2）

在-O 类型的选项中，GCC 提供了-Os 来优化 codesize，而 Clang 提供了两个选项：-Os 和-Oz，来优化 codesize。-Oz 比-Os 更为激进。选项说明具体见表 1 和相关参考链接。

表 1 GCC 和 LLVM-Clang 的-O 类型 codesize 优化选项

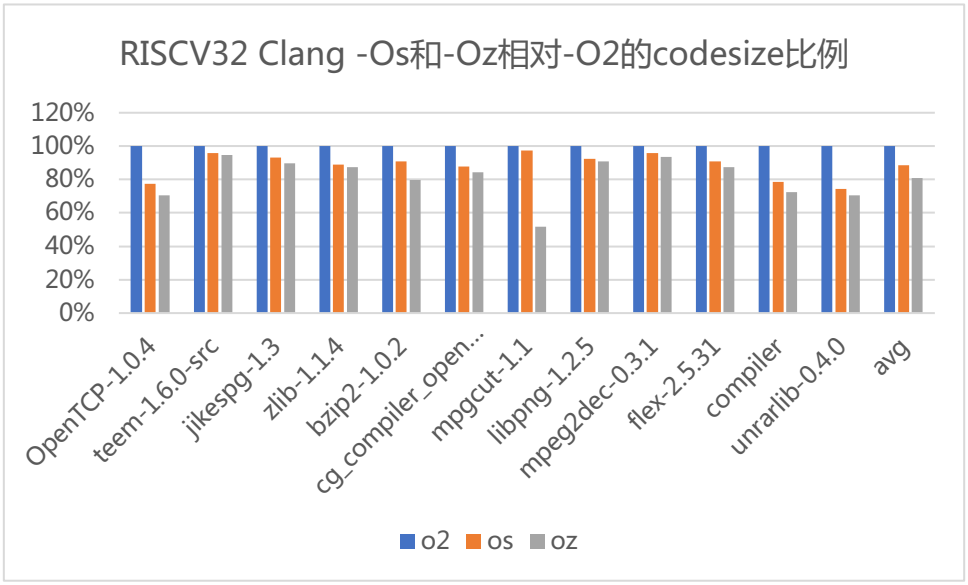
GCC [1]	-Os Optimize for size. -Os enables all -O2 optimizations except those that often increase code size: -falign-functions -falign-jumps -falign-labels -falign-loops -fprefetch-loop-arrays -freorder-blocks-algorithm=stc
Clang [2]	-Os Like -O2 with extra optimizations to reduce code size. -Oz Like -Os (and thus -O2), but reduces code size further.

图一显示了 RISCV32 下，GCC 工具链上的-Os 相对于-O2 的 codesize 比例，平均情况下，开启-Os 后 codesize 是-O2 的 84.07%。



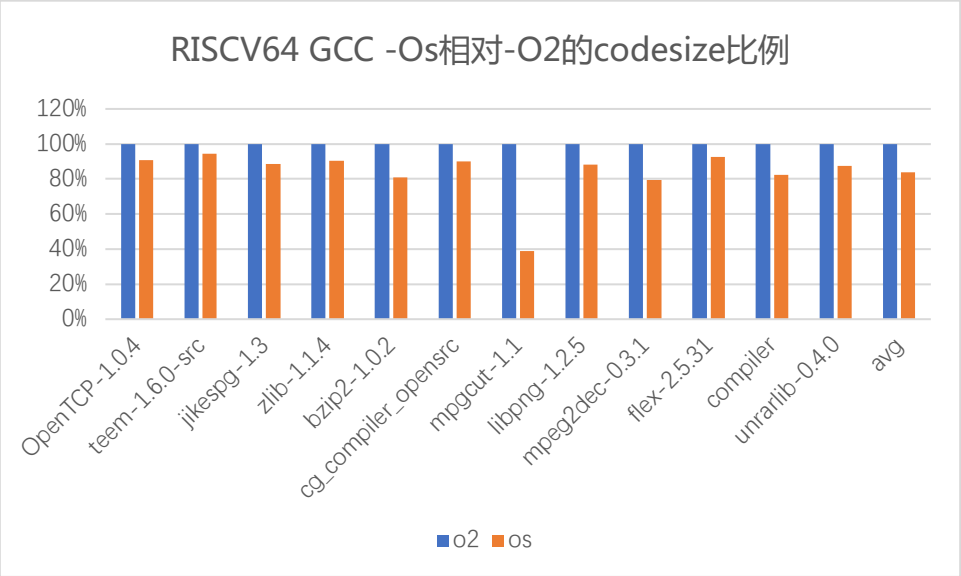
图一 RISCV32 GCC -Os vs -O2

图二显示了 RISCV32 下，Clang 工具链上的-Os 和-Oz 相对于-O2 的 codesize 比例。平均情况下，开启-Os 后 codesize 是-O2 的 88.63%，开启-Oz 后 codesize 是-O2 的 81.06%。开启激进优化选项-Oz 后，相对优化幅度超过了 GCC。



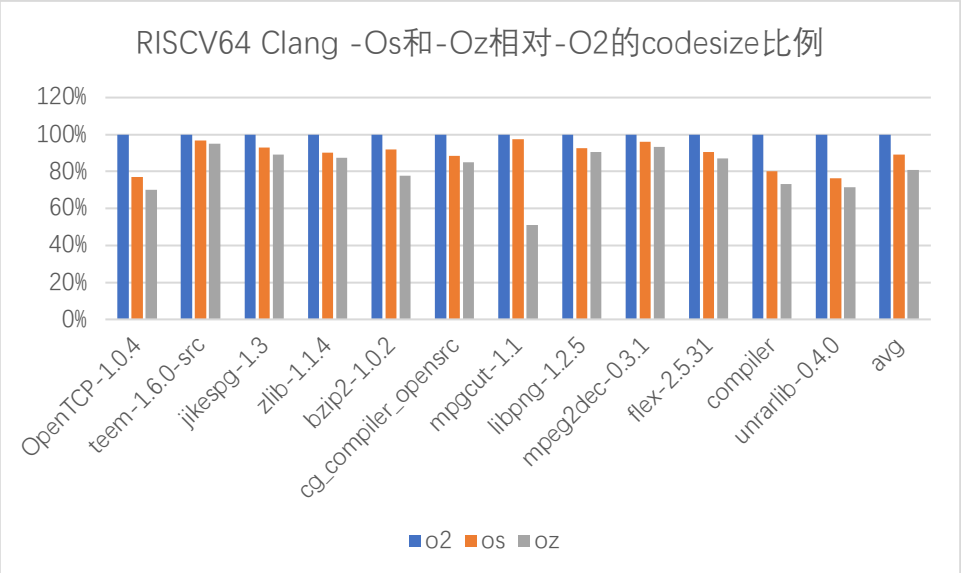
图二 RISCV32 Clang -Os & -Oz vs -O2

下图三显示了 RISCV64 下，GCC 工具链上的-Os 相对于-O2 的 codesize 比例，平均情况下，开启-Os 后 codesize 是-O2 的 83.67%，与 RISCV32 一致。



图三 RISCV64 GCC -Os vs -O2

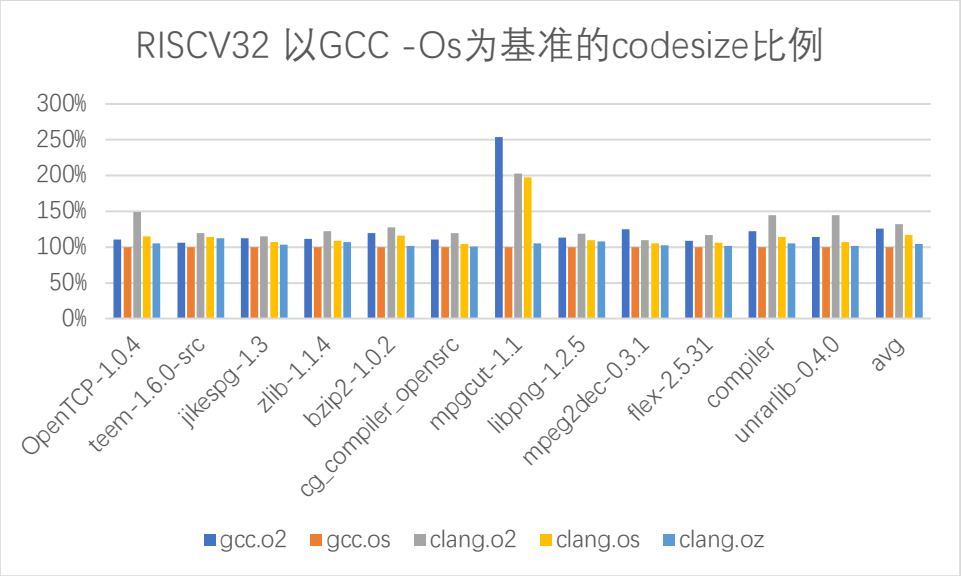
图四显示了 RISCV64 下，Clang 工具链上的-Os 和-Oz 相对于-O2 的 codesize 比例，平均情况下，开启-Os 后 codesize 是-O2 的 89.22%，开启-Oz 后 codesize 是-O2 的 80.91%，与 RISCV32 一致。开启激进优化选项-Oz 后，相对优化幅度也超过了 GCC。



图四 RISCV64 Clang -Os & -Oz vs -O2

二、在相同指令集下，GCC 和 Clang 的 codesize 对比。（对应第一期数据报 Part3）

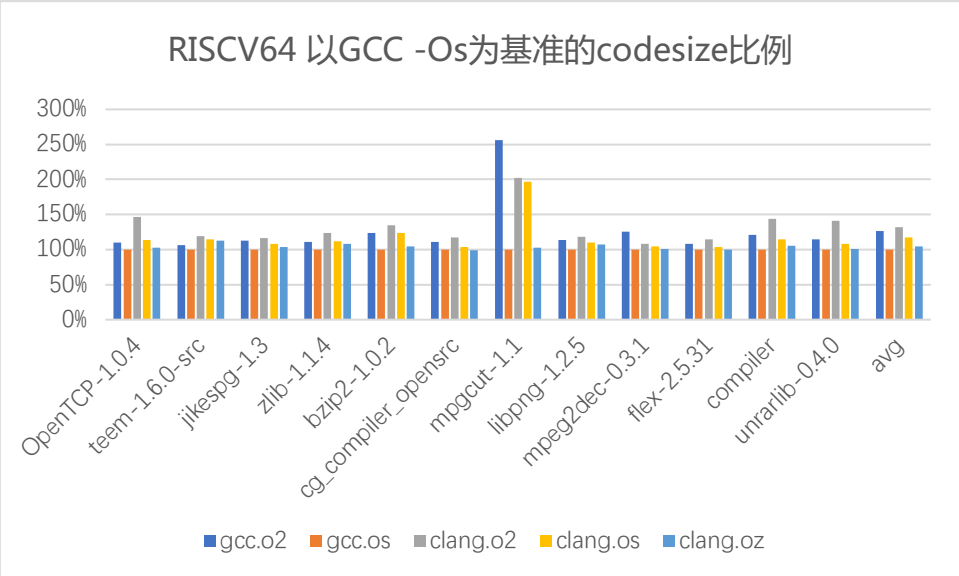
下图五显示了 RISCV32 下，以 GCC -Os 为基准，GCC -O2， Clang -O2，Clang -Os， Clang -Oz 的相对 codesize 比例。平均情况下，GCC -O2 的 codesize 是 GCC -Os codesize 的 125.39%，Clang -O2 的 codesize 是 GCC -Os codesize 的 132.32%，Clang -Os 的 codesize 是 GCC -Os codesize 的 117.04%，Clang -Oz 的 codesize 是 GCC -Os codesize 的 104.4%。这提示我们，使用 Clang 激进的-Oz 选项能够得到接近但略差于 GCC -Os 的 codesize 优化效果（4%）；如果仅使用 Clang -Os，那么 codesize 将比 GCC -Os 要高出 17%；如果在 Clang 上使用-O2，那么 codesize 将比 GCC -O2 高出 8.34%；以上几点说明 Clang 还有待改进之处。



图五 RISCV32 GCC vs Clang

下图六显示了 RISCV64 下，以 GCC -Os 为基准，GCC -O2， Clang -O2，Clang -Os， Clang -Oz 的相对 codesize 比例。平均情况下，GCC -O2 的 codesize 是 GCC -Os codesize 的 126.11%，Clang -O2 的 codesize 是 GCC -Os codesize 的 132.16%，Clang -Os 的 codesize 是 GCC -Os

codesize 的 117.74%，Clang -Oz 的 codesize 是 GCC -Os codesize 的 104.11%。数字与分析结论与 RISCV32 一致。



图六 RISCV64 GCC vs Clang

三、本节将对 ARM 和 RISCV 指令集的 codesize 在 GCC 工具链上，分别用-O2 和-Os 进行对比。

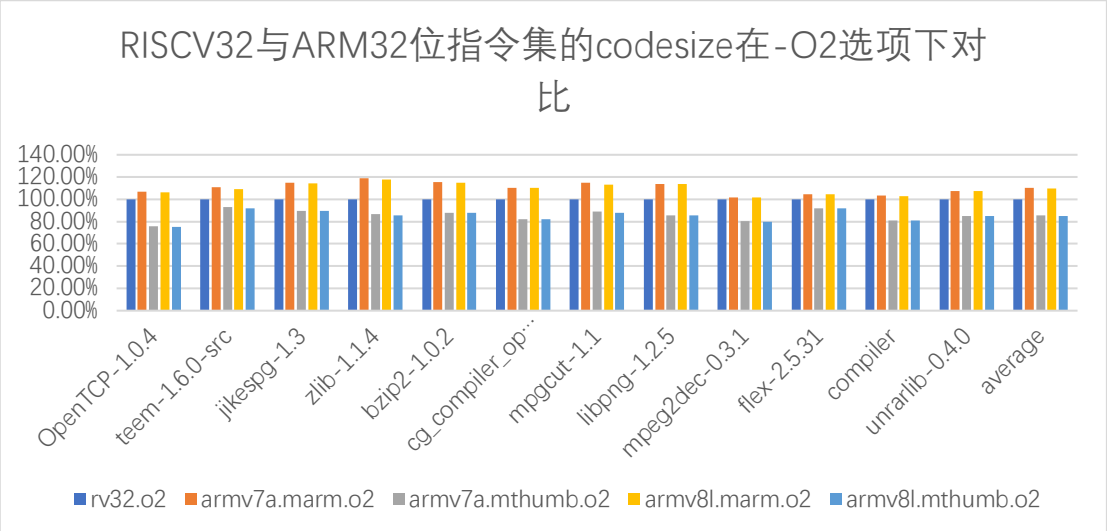
测试对照组和编译选项如下表，图例名字见红色字体部分：

表 2 ARM 和 RISCV codesize 对比测试对照组和编译选项

		工具链名称	O2	Os
32 位指令集	RISCV32	riscv32-unknown-elf-gcc/g++	-O2 -march=RISCV32imafdc 【rv32.o2】	-Os -march=RISCV32imafdc 【rv32.os】
	armv7a-marm	arm-linux-gnueabihf-gcc/g++	-O2 -mfpu=vfpv3-d16 --machine=arm -mtune=cortex-a9 -march=armv7-a 【armv7a.marm.o2】	-Os -mfpu=vfpv3-d16 --machine=arm -mtune=cortex-a9 -march=armv7-a 【armv7a.marm.os】
	armv7a-mthumb	arm-linux-gnueabihf-gcc/g++	-O2 -mfpu=vfpv3-d16 --machine=thumb -mtune=cortex-a9 -march=armv7-a 【armv7a.mthumb.o2】	-Os -mfpu=vfpv3-d16 --machine=thumb -mtune=cortex-a9 -march=armv7-a 【armv7a.mthumb.os】
	armv8l-marm	armv8l-linux-gnueabihf-gcc/g++	-O2 -mfpu=neon-fp-armv8 --machine=arm -march=armv8-a 【armv8l.marm.o2】	-Os -mfpu=neon-fp-armv8 --machine=arm -march=armv8-a 【armv8l.marm.os】

	armv8l-mthumb	armv8l-linux-gnueabihf-gcc/g++	-O2 -mfpu=neon-fp-armv8 --machine=thumb -march=armv8-a【armv8l.mthumb.o2】	-Os -mfpu=neon-fp-armv8 --machine=arm -march=armv8-a【armv8l.mthumb.os】
64位指令集	RISCV64	riscv64-unknown-elf-gcc/g++	-O2 -march=RISCV64imafdc【rv64.o2】	-Os -march=RISCV64imafdc【rv64.os】
	aarch64	aarch64-linux-gnu	-O2 -mcpu=cortex-a53【a64.o2】	-Os -mcpu=cortex-a53【a64.os】

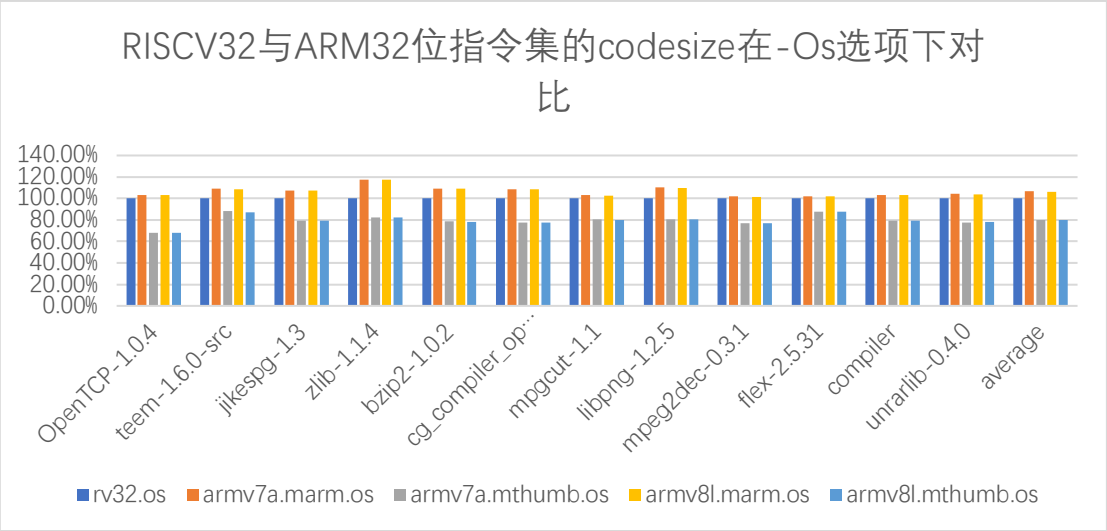
下图七显示了 armv7a/armv8l 指令集下分别采用 arm 和 thumb 指令集选项时，在 GCC -O2 选项下的 codesize 相对于 RISCV32 的比例。具体地，armv7a.marm.o2 的 codesize 是 rv32.o2 codesize 的 110.27%，armv7a.mthumb.o2 的 codesize 是 rv32.o2 codesize 的 85.68%，armv8l.marm.o2 的 codesize 是 rv32.o2 codesize 的 109.71%，armv8l.mthumb.o2 的 codesize 是 rv32.o2 codesize 的 85.26%。可以看到 armv8 的 ISA 与 armv7a 的 ISA 数据基本一致，且 ARM 的 thumb 指令集能带来相对 RISCVC 扩展 15%的 codesize 缩减。



图七 32 位 ISA 在 GCC -O2 下的对比

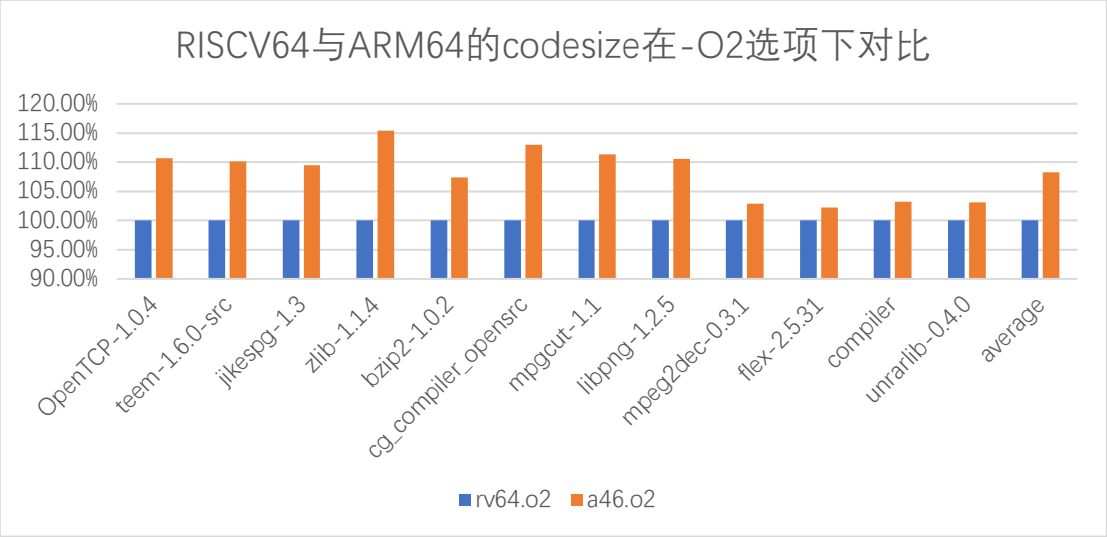
下图八显示了 armv7a/armv8l 指令集下分别采用 arm 和 thumb 指令集选项时，在 GCC -Os 选项下的 codesize 相对于 RISCV32 的比例。具体地，

armv7a.marm.os 的 codesize 是 rv32.os codesize 的 106.56%，armv7a.mthumb.os 的 codesize 是 rv32.os codesize 的 79.78%，armv8l.marm.os 的 codesize 是 rv32.os codesize 的 106.35%，armv8l.mthumb.os 的 codesize 是 rv32.os codesize 的 79.64%。可以看到 armv8 的 ISA 与 armv7a 的 ISA 数据基本一致，且 ARM 的 thumb 指令集能带来相对 RISCVC 扩展 20%的 codesize 缩减。



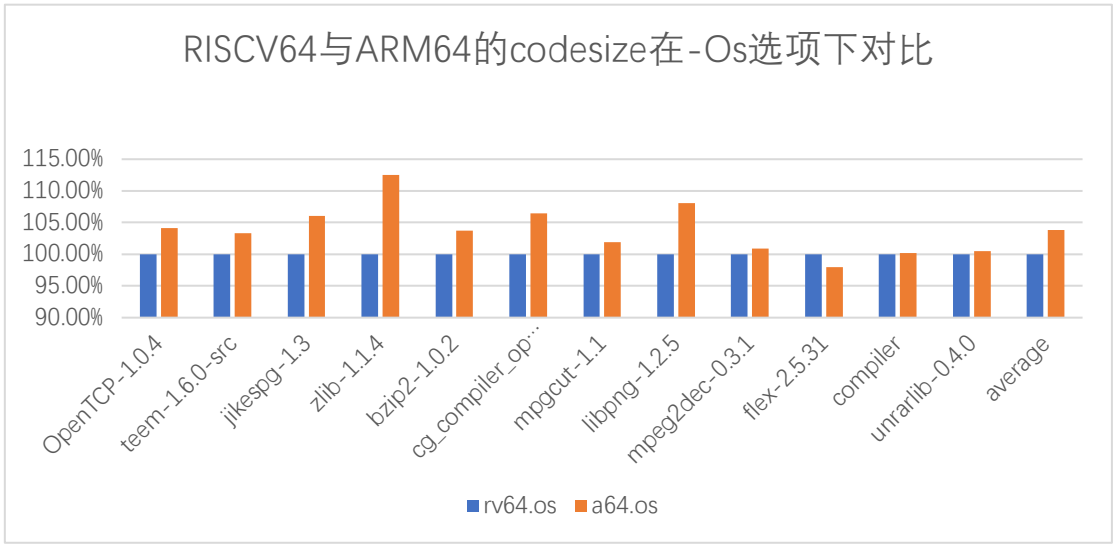
图八 32 位 ISA 在 GCC -Os 下的对比

下图九显示了 AARCH64 指令集在 GCC -O2 选项下的 codesize 相对于 RISCV64 的比例。平均情况下，a64.o2 的 codesize 是 RISCV64.o2 的 108.3%。



图九 64 位 ISA 在 GCC -O2 下的对比

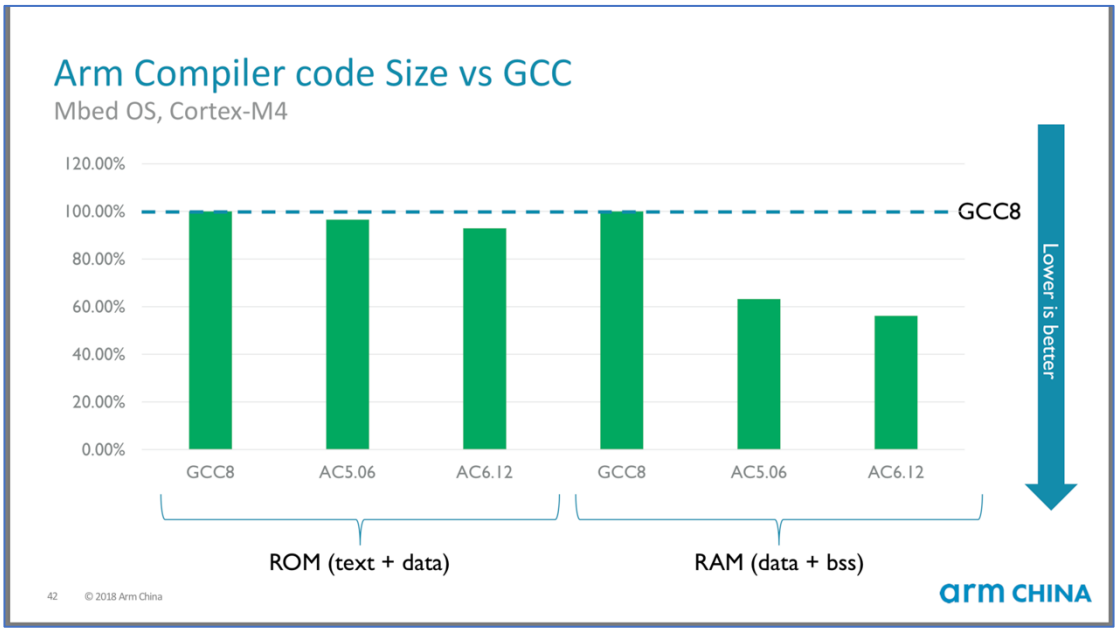
下图十显示了 AARCH64 指令集在 GCC -Os 选项下的 codesize 相对于 RISCV64 的比例。平均情况下，a64.os 的 codesize 是 RISCV64.os 的 103.8%。



图十 64 位 ISA 在 GCC -Os 下的对比

关于 ARM codesize，我们从链接[3]下载得到了来自 ARM China 官方的 ARM 最新集成开发环境宣传资料，其中介绍了 ARM 的商用闭源编译器 ARM Comipler5 和 ARM Compiler6 相对于 GCC8 公版编译器的 codesize 优化程度。我们通过查阅资料了解到，ARM Compiler5 （AC5.*）是 ARM 自研 20 多年的 armcc，ARM Compiler6 （AC6.*）是 ARM 基于 LLVM 架构开

发的商业编译器。从图十一的 slides 截图中可以看到，相对官方 GCC8， AC6 工具链能够带来代码段和 RO 数据段的约 8%～9%的优化幅度，在 RW 数据段和 bss 段上有约 44%左右的缩减。在 PLCT 的评测中，我们只对代码段和只读数据段的 codesize 指标进行了报告，PLCT 的评测数据显示，目前在 Clang 中，RISCV32/RISCV64 在-Oz 下的 codesize 均比 GCC 公版的-Os 选项高 4%。如果我们以 ARM 商业编译器目前所获得的优化幅度来作为最优预期，那么粗略估计 RISCV32/RISCV64 在 Clang 上还会有 12%～13%的优化程度。



图十一 来自 ARM 官方宣传资料的 codesize 数据[3]

三、来自 Berkeley CS， Peijie Li 技术报告 《Reduce Static Code Size and Improve RISC-V Compression》[3]解读

这篇 TR 的是 2019 年 5 月 16 日发布的，内容简介如下：RVC 是 RISCV 指令集的扩展指令集子类之一，它将标准指令集中的某些特殊形式的指令进行压缩，从 32bit 压缩成 16bit，以达到缩减 codesize 的目的。目前的标准 RISCVC 已经在 MiBench 上达到了 35%+的 codesize 缩减（ -O3 -Os –msave-restore -flto ）；该 TR 在 ZephyrRTOS、Apache Mynewt、

MiBench、 MediaBench 上，用 GCC8.1.0 （-O3 -Os -msave-restore）的上生成的汇编代码的统计数据，对那些“尚未被压缩的”且“出现概率较大”的指令编码进行进行了进一步的 ISA 编码优化，这些优化暂时使用了浮点压缩指令的编码，更进一步地获得了 5%的 codesize 缩减。

通过这篇 TR，我们了解到当使用 GCC 对 RISCV code 进行编译时，除了使用-Os，还可以使用 -msave-restore 来对 codesize 进行进一步的缩减，这项优化将函数的 prologue 中的入栈指令段替换为 jal 指令，将 epilogue 中的出栈指令段替换为 j 指令，同时在 text 段插入相应的 save 和 restore 例程，以达到“同样类型的 prologue 和 epilogue”在一份 executable 中只有一段代码，从而达到缩减 codesize 的目的。该选项在 GCC 中的介绍如下[4]:

-msave-restore -mno-save-restore Do or don’t use smaller but slower prologue and epilogue code that uses library function calls. The default is to use fast inline prologues and epilogues.

我们在 CSiBE 上测试了-Os -msave-restore 相对-Os 的优化效果，codesize in byte 和缩减幅度如下表 3 所示 ，平均情况下，codesize 减小了 2.4%。但需要关注的是，该选项会增加程序中跳转指令的数量，每个 prologue 的改写会增加一条 jal 和一条 j，每个 epilogue 的改写会增加一条 j，且 ret 指令的 PC 和 return address 将出现很多“一对多”的情况，这将会给处理器架构中的跳转指令设计和返回地址预测带来额外的挑战。

表 3 CiSiBE 上-msave-retsore 的 codesize 缩减幅度

	RISCV32 -os	RISCV32 -os -msave-restore	reduce
OpenTCP-1.0.4	1442	1407	2.51%
teem-1.6.0-src	2556	2491	2.63%
jikespg-1.3	12264	11987	2.31%
zlib-1.1.4	2068	2016	2.58%
bzip2-1.0.2	6025	5882	2.43%
cg_compiler_opensrc	5132	4986	2.93%

mpgcut-1.1	8696	8350	4.14%
libpng-1.2.5	5092	4949	2.88%
mpeg2dec-0.3.1	1549	1522	1.77%
flex-2.5.31	9528	9445	0.89%
compiler	3925	3885	1.01%
unrarlib-0.4.0	4547	4427	2.70%
average			**2.40%**

Reference：

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

[2] https://clang.llvm.org/docs/CommandGuide/clang.html

[3] https://aijishu.com/a/1060000000079263?aff=eeeknow

[4] https://gcc.gnu.org/onlinedocs/gcc-9.2.0/gcc/RISC-V-Options.html#RISC-V-Options

[5] https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-107.pdf