RISCV 工具链数据报第五期：

LLVM RISCV codesize 优化一例和业界要闻

在第四期数据报中，我们对在 GNU GCC 和 LLVM-Clang 工具链上 RISCV32、RISCV64 的 codesize 数据进行了对比，以 GCC -Os 选项为 100%，codesize 的相对比例如下表：

表 1 RISCV32 和 RISCV64 下 GCC 和 Clang 的 codesize 对比

	GCC -Os	GCC -O2	Clang -O2	Clang -Os	Clang -Oz
RISCV32	100%	125.39%	132.32%	117.04%	104.4%
RISCV64	100%	126.11%	132.16%	117.74%	104.11%

（注：表 1 的评测标准、方法和数据来源可详细参考“RISCV 工具链数据报第四期”）

本期数据报中，我们将详述一个 PLCT 近日所完成的 LLVM 上 RISCV codesize 的优化案例，并报告优化结果。

一、优化点分析和评估

在对 GCC 和 Clang 的 codesize 差距来源进行分析的过程中，我们发现，Clang 在-O2、-O3、-Os、-Oz，以及--ffp-contrast=fast 选项下，都无法生成单/双精度的 fused multiply-add/sub（以下简称 FMA，fused multiply-accumulate），但 GCC 可以在-O2、-O3 和-Os 选项下生成 RISCV 指令集 F 标准扩展中所规定的 fmadd.s/d, fmsub.s/d, fnmadd.s/d 和 fnmsub.s/d 指令。这个差别，将会给浮点乘加计算密集的应用，比如矩阵乘法、卷积和 FFT 等带来 codesize 的显著损失，比如，对 CSiBE 的 teem-1.6.0-src 用例，GCC 产生的目标文件中有 69 个文件包含了共 5009 条 FMA 指令，而 LLVM 将对应地产生 2 倍数目的指令，生成的库文件将多出 20036 个字节的 code。

同时，实验表明，同版本的 Clang 是可以在 Mips32/64 和 AARCH64 指令集下生成 FMA 指令的。这说明，LLVM 的 RISCV 的后端可能还有一些缺陷，很有必要对其进行分析，找出原因并进行改进。

二、查找和改进

首先检查 RISCV 后端代码是否支持 FMA 的指令描述，发现 https://reviews.llvm.org/D54205 已经提交了 RISCV 所有 8 条 FMA 指令的后端支持，intrinsics 的测试也已经添加，但无法从高级语言直接生成 RISCV FMA 指令。

按照经验，FMA 的产生属于编译器中“代码生成模块”的“指令选择”部分的功能，对 LLVM 开启 debug 选项“--debug-only=isel”后观察对比 RISCV32 和 MIPS32 的 log dump，发现 isel pass 产生了不同结果，下表中高亮部分显示了不同的指令选择结果，这是导致 RISCV 后端无法生成 FMA 指令的直接原因:

riscv32 isel log	mips32 isel log
ISEL: Starting selection on root node: t58: f32 = fadd t52, t57 ISEL: Starting pattern match Initial Opcode index to 25012 TypeSwitch[f32] from 25014 to 25017 Morphed node: t58: f32 = FADD_S t52, t57, TargetConstant:i32<7> ISEL: Match complete!	ISEL: Starting selection on root node: t64: f32 = fadd t58, t63 ISEL: Starting pattern match Initial Opcode index to 51317 Match failed at index 51320 Continuing at 51370 TypeSwitch[f32] from 51382 to 51385 Morphed node: t64: f32 = MADD_S t58, t61, t62 ISEL: Match complete!

在LLVM中，指令选择依赖于后端指令描述的SelectionDAG匹配模版Pattern，其定义在 llvm/include/llvm/Target/Taget.td文件的 class Instruction中。Pattern描述SelectionDAG的匹配模版，将机器无关的IR映射

到机器相关的IR上。明确这些背景后，我们可以大致猜测目前RISCV的后端可能没有恰当的描述FMA指令的Pattern。继续查看LLVM中RISCV后端对F和D类型指令的指令定义文件 RISCVInstrInfoF.td 和 RISCVInstrInfoD.td， 发现目前的FMA指令pattern描述如下（以F类型为例，D类型相同）。

```
// fmadd: rs1 * rs2 + rs3
def : Pat<(fma FPR32:$rs1, FPR32:$rs2, FPR32:$rs3),
          (FMADD_S $rs1, $rs2, $rs3, 0b111)>;
// fmsub: rs1 * rs2 - rs3
def : Pat<(fma FPR32:$rs1, FPR32:$rs2, (fneg FPR32:$rs3)),
          (FMSUB_S FPR32:$rs1, FPR32:$rs2, FPR32:$rs3, 0b111)>;
// fnmsub: -rs1 * rs2 + rs3
def : Pat<(fma (fneg FPR32:$rs1), FPR32:$rs2, FPR32:$rs3),
          (FNMSUB_S FPR32:$rs1, FPR32:$rs2, FPR32:$rs3, 0b111)>;
// fnmadd: -rs1 * rs2 - rs3
def : Pat<(fma (fneg FPR32:$rs1), FPR32:$rs2, (fneg FPR32:$rs3)),
          (FNMADD_S FPR32:$rs1, FPR32:$rs2, FPR32:$rs3, 0b111)>;
```

可以发现，这些Pattern使用了“fma opcode”（高亮部分显示）来统一描述了fmadd.s, fmsub.s, fnmsub.s 和fnmadd.s 指令中 a*b+c 的运算模式，但并没有说明“乘”和“加”运算的组合关系。虽然这种描述是正确的，但LLVM的前端流程在SelectionDAG中却不会自动产生fma类型的节点（除非人为在后端DAG2DAG pass中添加额外代码或者书写intrinsics），这就导致了编译器在指令选择阶段遇到了fadd/fsub指令之后，不会去探测其在

SelectionDAG图中相邻的节点是否是fmul运算，也就不可能匹配到fmadd.s, fmsub.s, fnmsub.s 和fnmadd.s指令。

基于上述原理，我们将Pattern做了如下修改，并成功编译了修改后的代码。（RISCVInstrInfoD.td的修改是相同的，此处略）

```
// fmadd: rs1 * rs2 + rs3
def : Pat<(fadd (fmul FPR32:$rs1, FPR32:$rs2), FPR32:$rs3),
          (FMADD_S $rs1, $rs2, $rs3, 0b111)>;
// fmsub: rs1 * rs2 - rs3
def : Pat<(fadd (fmul FPR32:$rs1, FPR32:$rs2), (fneg FPR32:$rs3)),
          (FMSUB_S FPR32:$rs1, FPR32:$rs2, FPR32:$rs3, 0b111)>;
// fnmsub: -rs1 * rs2 + rs3
def : Pat<(fadd (fmul (fneg FPR32:$rs1), FPR32:$rs2), FPR32:$rs3),
          (FNMSUB_S FPR32:$rs1, FPR32:$rs2, FPR32:$rs3, 0b111)>;
// fnmadd: -rs1 * rs2 - rs3
def : Pat<(fadd (fmul (fneg FPR32:$rs1), FPR32:$rs2), (fneg FPR32:$rs3)),
          (FNMADD_S FPR32:$rs1, FPR32:$rs2, FPR32:$rs3, 0b111)>;
```

三、效果评测

首先对优化前后的LLVM-Clang的codesize进行对比，受影响的CSiBE case见下表，codesize的delta最大达到了3.02%。这里只用RISCV32为例来说明，对RISCV64指令集，结果是相同的。

表2 Clang编译器上RISCV32指令集FMA优化的效果对比

	clang rv32优化后 -Oz	clang rv32优化前 -Oz	delta
teem-1.6.0-src	801229	825405	3.02%

bzip2-1.0.2	54868	54872	0.01%
cg_compiler_opensrc	113538	113550	0.01%
mpgcut-1.1	8914	8934	0.22%
libpng-1.2.5	98026	98202	0.18%

然后我们对GCC -Os选项下生成的FMA指令，和优化后Clang在-Oz选项下生成的FMA指令数目进行了对比。可以看到，除了bzip2，Clang生成的FMA指令数目是等于或者大于GCC所生成指令数的。数目差异的原因，可能跟两个工具链中不同的前端流程相关。

表3 GCC和Clang在RISCV32上产生的FMA指令条数对比

	GCC -Os	Clang -Oz
teem-1.6.0-src	5009	5552
bzip2-1.0.2	2	1
cg_compiler_opensrc	3	3
mpgcut-1.1	3	5
libpng-1.2.5	43	43

最后，我们对优化后Clang编译器在激进codesize优化选项-Oz选项下的codesize跟GCC -Os选项下的codesize比例进行再次对比，以获得本案例优化对LLVM的改进效果。可以看到，这个优化，确实获得了Clang -Oz相对GCC -Os的codesize的改进。虽然总体数值是很微小的，但也说明了LLVM RISCV后端是有潜能的，需要逐步挖掘。处理器后端调优往往是一个漫长而需要持久投入的过程，PLCT实验室将在专注于RISCV体系结构，进行更多的优化工作并开源。

表4 优化前后Clang -Oz 相对GCC -Os的codesize对比

	GCC -Os	Clang -Oz 优化前	Clang -Oz 优化后
RISCV32	100%	104.4%	104.09%
RISCV64	100%	104.11%	103.80%

本次优化的代码下载方式：repo 地址：https://github.com/isrc-cas/riscv-llvm， 分支名：plct-rvopt-1。

RISCV 编译器业界要闻：

1. 12 月 14 日 RISCV-V-SPEC 更新了版本，从 0.8 演进到了 0.9-draft（https://github.com/riscv/riscv-v-spec/commit/3aa56af7957b07b7528415125f80274781b27e08）。 PLCT 实验室将继续跟进 LLVM 上 RVV 指令的实现并开源。
2. 对 RISCV 还在拟定阶段的 bitmanip 指令（https://github.com/riscv/riscv-bitmanip），资深编译器公司 Embecosm 已经在 github 开源了其 GCC 和 LLVM 的支持并在持续跟进，参考链接：

   https://www.embecosm.com/2019/10/22/llvm-risc-v-bit-manipulation-extension/
   https://www.embecosm.com/2019/10/22/gcc-risc-v-bit-manipulation-extension/

Reference：

[1] https://llvm.org/docs/CodeGenerator.html#selectiondag-select-phase