南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

玩具级编译器~~LLVM~~的设计、实现与维护

Yingwei Zheng

2024/6/15

关于本人

- GitHub: dtcxzyw
- LLVM中端/RISC-V后端 contributor/reviewer
- 近半年在LLVM提交总数前20: git shortlog -sn --since="2023-12-14"
- 2023年编译系统设计赛RISC-V赛道 特等奖
  - ~~把第二名的性能分压到了50分以下~~

File display

#	用户名	队伍	提交次数(ASC)	最后提交时间(ASC)	正确分	性能分	总分
1	202314325201374	CMMC/ 南方科技大学	8	2023-08-21 16:23:25	98	92.6925	93.8708
2	202310007201692	bit.newnewcc/ 北京理工大学	23	2023-08-21 14:18:13	100	49.3855	60.6219
3	202310558201558	Yat-CC/ 中山大学	17	2023-08-21 14:53:12	99	47.8161	59.1789
4	202310006201898	喵喵队引体向上/ 北京航空航天大学	38	2023-08-21 16:50:16	100	45.5433	57.6327
5	202310055201721	生成式智能人工队/ 南开大学	18	2023-08-21 17:18:56	100	45.4676	57.5738

```
  527  Simon Pilgrim
  445  Craig Topper
  382  Fangrui Song
  381  Timm Bäder
  362  Nikita Popov
  350  Kazu Hirata
  286  Florian Hahn
  282  Joseph Huber
  217  Jay Foad
  207  Peter Klausler
  195  Alexey Bataev
  194  Matt Arsenault
  177  Vitaly Buka
  173  David Green
  171  LLVM GN Syncbot
  162  Mehdi Amini
  154  Vlad Serebrennikov
  149  Yingwei Zheng
  147  Mark de Wever
  141  Luke Lau
```

编译器设计 —— 信息损失

- 编译是一个信息逐级损失的过程
  - 在两种表示形式中，某些表示往往在另一个表示形式中没有完全等价的表示
  - C代码→机器码： 指针类型信息丢失
  - 将sdiv X, Y sink到某个后继基本块：丢失了在原基本块上Y != 0的信息
- 好的优化编译器能够尽可能迟地丢弃信息
  - Canonicalization → Opt → Lower | Canonicalization → Opt → Lower | …
- 传统优化模式：AST → IR → SDAG/GISel → MIR → ASM
- 分层越多，损失信息越晚，优化越好做
  - 重新发明 MLIR（Multi-Level Intermediate Representation）
    - MLIR: Scaling Compiler Infrastructure for Domain Specific Computation (CGO’21)
  - 思考：能不能用一套统一表示来描述AST→ ASM的过程?
    - LLVM/CMMC MIR：Flags表示当前IR满足的条件，比如是否为SSA形式，是否完成了寄存器分配等

编译器设计 —— 规范化 (Canonicalization)

- 考虑优化(X + C1) + C2 → X + (C1 + C2)，我们有：
  - (X + C1) + C2
  - (C1 + X) + C2
  - C1 + (X + C2)
  - C2 + (X + C1)
- 将常量C放到加号右侧，我们只需考虑一种模式！
- 规范形式（Canonical form）
  - 无损变换（tgt→src仍然是正确的）
  - 几乎所有模式匹配代码仅需考虑规范形式（代码--/bug--）

- 思考：如果我们为每个阶段的表示都定义一个“规范程度”（Canonical rank），并保证规范程度非增的优化只运行有限次，就能避免优化死循环 :)

编译器设计 —— 未定义行为

- 信息损失的原因之一：未定义行为 (Undefined Behavior, UB)
  - RISC-V中DIV除0不会引发异常，所以翻译为RV汇编后不能假设除数非0
- 语言规范给予编译器的利剑
  - 法无禁止皆可为：只需保证对于**在源代码中良定义**的输入，源代码和目标代码的**外部可观测行为**一致。
  - SysY语言中可利用的信息：
    - 除数非0
    - 访问不越界
- 思考：能不能在IR中为指令提供描述更多未定义行为的方式?
  - Poison-generating flags：add nuw %x, %y → %x + %y 不发生无符号溢出
  - Taming Undefined Behavior in LLVM (PLDI’17)：Immediate UB/Deferred UB

编译器实现 —— 善用工具避免常见错误

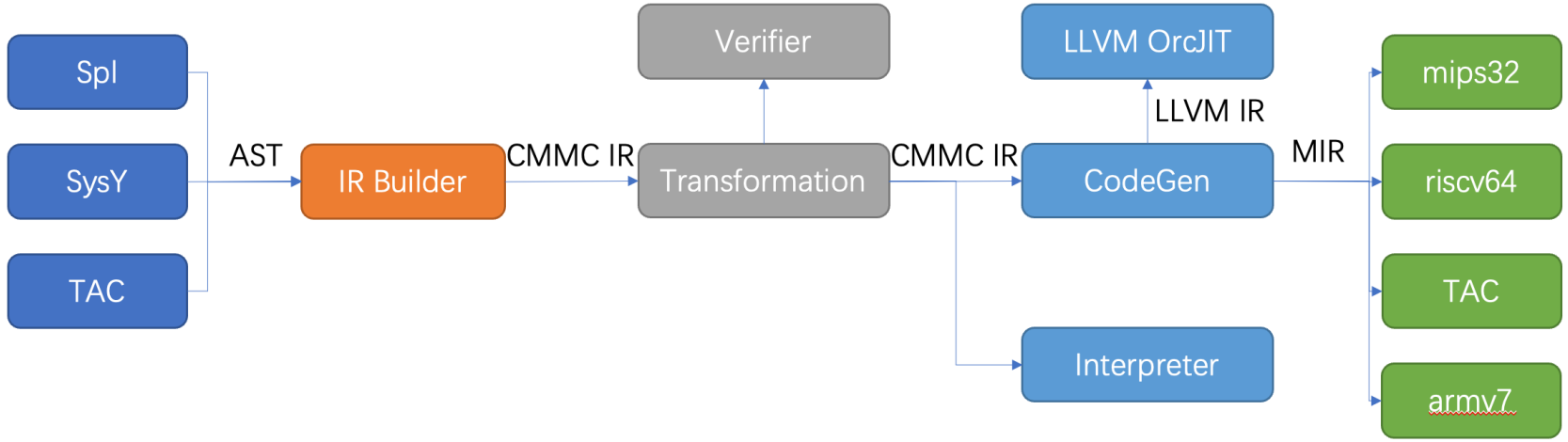
- 经常断言：在错误被传播前将其捕获
  - ~~队友代码和你的代码的假设很可能不一致~~
- 静态分析工具：clang-tidy/clippy
- Sanitizers（~~Rust党请忽略~~）：asan/msan/ubsan
- 使用DSL（Domain Specific Language）生成模板代码/面条代码：
  - LLVM: .td + TableGen
  - CMMC: YAML + Jinja2
  - 指令验证/打印/抽象一条龙服务
  - 2294行DSL = 4238行指令选择 + 5781行指令描述 + 233行指令调度模型

```
## 2.6 Load and Store Instructions
Load:
  Format: "$Mnemonic:Template $Rd:GPR[Def], $Imm:Imm12[Metadata]($Rs1:BaseLike[Use]) # $Alignment:Align[Metadata]"
  Flag: [Load]
```

  - 思考：如何为DSL实现模板功能（基于Load定义RV中的LB(U)/LH(U)/LW(U)/LD）？

编译器实现 —— 模块化设计

- 多前端/多后端设计：快速甩锅
- 多pass设计：基于二分快速定位缺陷模块
- 后端写不完想先验证中端：解释器/LLVM JIT
- ~~决赛前多写代码是为了决赛时不写代码~~



- 思考：如何只运行一次编译器就能定位出缺陷模块?

编译器实现 —— 了解目标平台CPU

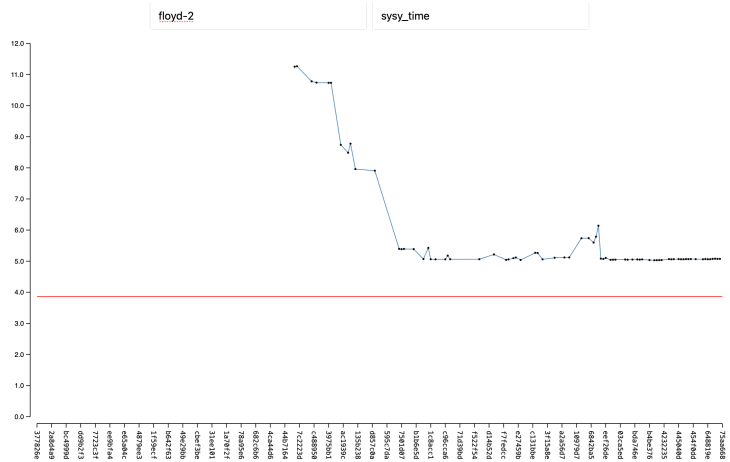
- 时刻记住程序跑在板子上而不是qemu上
- 指令集支持：ARM —— IDIV/NEON RISC-V —— Zba/Zbb
- 性能模型：
  - 前端：OOO/Issue width/分支预测
  - 后端：各指令的延迟与吞吐（该指令会在哪些pipeline/port上被执行？）
  - 指令组合：Macro fusion/Cascade FMA/Short Forward Branch on SiFive-7
  - 是否支持寄存器重命名？如果不支持的话需要调整寄存器分配以避免伪依赖（~~喜提顺序单发射~~）
- 内存子系统：
  - Instruction：ITLB/ICache
    - 指令对齐（特别是跳转目标）
    - Tail Duplication
  - Data：DTLB/DCache
    - 矩阵访问模式
    - Software/Hardware prefetcher
    - 自动并行调度（N个任务，M个进程，K=N/M，第I个线程执行任务I*K…(I+1)*K-1还是I,M+I,2M+I,…,(K-1)M+I）

编译器维护 —— LLVM社区是个草台班子

- 人总是会犯错的，哪怕过了多人代码审核也有错误发生
  - ~~当然没经过审核的commit更容易翻车~~
- 缺陷模块重灾区
  - 前端：C++ Lambda + Templates/C++20 Modules/Concepts
  - 中端：InstCombine/ValueTracking/SCEV/Vectorizer
  - 后端：X86/AArch64/RISC-V
- 缺陷类型
  - Crash: 4536
  - Miscompilation: 1159
  - Hang: 77
- 软件工程拯救草台班子堆出的屎山代码

编译器维护 —— 持续集成

- 基于GitHub Actions，每次提交后都执行：
  - 构建：–Wall –Werror
  - 正确性验证
  - 性能监测：
    - 早期：使用qemu + tcg plugin统计指令数
    - 后期：下发到板子上执行并收集PMU数据，并上传至github pages



  - 保险起见，将代码提交到gitlab上的过程也是自动化的

sysy_time | gcc

Testcase Name	Baseline	Current	Delta	Delta (%)
floyd-0	0.000039	0.000125	0.00	220.51%
vector_mul1	4.336868	7.647694	3.31	76.34%
vector_mul2	4.336868	7.645925	3.31	76.30%
vector_mul3	4.331131	7.359341	3.03	69.92%
crypto-3	1.104617	1.556963	0.45	40.95%
crypto-2	0.835838	1.176488	0.34	40.76%
crypto-1	1.019811	1.435239	0.42	40.74%
floyd-1	0.007901	0.010627	0.00	34.50%
floyd-2	3.82814	5.080674	1.25	32.72%
gameoflife-oscillator	8.646897	10.627355	1.98	22.90%
gameoflife-gosper	11.024635	12.949303	1.92	17.46%
03_sort2	6.411623	7.169772	0.76	11.82%
transpose1	5.203974	5.606196	0.40	7.73%
03_sort3	1.063625	1.125402	0.06	5.81%
03_sort1	0.350654	0.363721	0.01	3.73%
transpose2	13.136108	13.304167	0.17	1.28%
transpose0	5.328811	5.363732	0.03	0.66%
layernorm1	3.173718	3.176931	0.00	0.10%
layernorm3	3.179776	3.177977	−0.00	−0.06%
layernorm2	3.187427	3.176119	−0.01	−0.35%
large_loop_array_3	4.290017	4.091858	−0.20	−4.62%
large_loop_array_2	7.83833	7.376859	−0.46	−5.89%
large_loop_array_1	3.887821	3.643789	−0.24	−6.28%
median2	17.790588	16.402693	−1.39	−7.80%
median1	0.002375	0.00178	−0.00	−25.05%
shuffle1	17.953774	12.786357	−5.17	−28.78%

编译器维护 —— 模糊测试（Fuzzing）

- 如何确保隐藏样例能够被正确优化?
  - 随机生成符合SysY语法的代码进行测试!
- 测试先知（Oracles）：
  - Crash (with sanitizers)
  - Infinite loop
  - Miscompilation：differential testing (ground truth: gcc/clang/O0/O1/O2)
- 如果前端支持更多的语法，可以直接使用现成c compiler fuzzer：
  - Finding and Understanding Bugs in C Compilers (PLDI’11)
  - 1M tests/day on a 40C server
- 思考：fuzzer发现不同bug所需的期望测试次数呈指数级上升，如何解决?
  - Coverage-guided/Mutation-based
  - ~~引入新的bug把次数拉下来~~

编译器维护 —— 优化机会发现

- ~~瞪眼法~~
- 超参数搜索（Unroll size等）
- 离线优化机会发现→编译器简单实现
  - 死代码检测：Finding Missed Optimizations through the Lens of Dead Code Elimination (ASPLOS’22)
  - 分块法指导指令选择：挑选测试代码中的短序列，观察gcc/llvm的输出，并使用llvm-mca估计性能
  - SMT Solver判断逻辑表达式恒真/恒假：提取约束 → 转为SMT表达式 → 丢给z3
    - Exp == F unsat → Exp 恒真
  - 超优化器：Souper: A Synthesizing Superoptimizer (Arxiv, 2018)
- 验证优化正确性：
  - 中端验证：Alive2: Bounded Translation Validation for LLVM (PLDI’21)
  - 后端指令选择验证：提升到LLVM IR再丢给Alive2 https://github.com/dtcxzyw/rvtv

通用优化编译器的未来

- ~~没有未来~~
  - 快去转行DSL/DSA compiler，性能秒杀GCC/LLVM
- Future Directions for Optimizing Compilers (Arxiv, 2018)
  - 基于SMT Solver自动合成编译器!
- 但从软件工程的角度来看，我们还有很多机会（感谢GCC/LLVM）：
  - 编译器测试（Fuzz4all，ICSE’24）
    - ~~屎山通病~~
  - 错误定位与最小复现样例生成
    - ~~楼上搞fuzzing的怎么交完issue就跑了~~
  - 代码评审
    - ~~有没有天才LLM来解放我的脑子~~

小广告

- 中国科学院软件研究所程序语言与编译技术实验室（PLCT Lab）招收实习生：https://github.com/plctlab/weloveinterns/blob/master/open-internships.md

- 欢迎大家参与LLVM及其它基础系统软件的开发：
  - LLVM急缺有经验的contributor/reviewer/maintainer/code owner

Q&A