金枪鱼之夜

OSPP 2025 项目成果分享

分享人：贺泽邦；项目导师：陈泱宇

Tunight @ TUNA | OSPP 2025

项目简介

GCC-Fortran 的 Function Multi-Versioning 功能实现

项目简介

基本信息

- 导师：陈泱宇
- 项目基本要求：
  1. 基于 GCC master 分支，为 Fortran 语言的 `subroutine` 与 `function` 声明添加 `target` / `target_clones` 的 ATTRIBUTES 支持。
  2. 将该实现提交给开源上游

项目简介

什么是 Function Multi-Versioning?

- 函数多版本（Function Multi-Versioning / FMV）是一种编译/链接技术
- 可以在同一个二进制中包含某个函数的多个实现版本
- 可以是针对某个特定 CPU 特性（例如 `AVX512`、`SSE4.2` 等）
- 甚至也可以针对某个 CPU 架构甚至处理器型号（例如 `arch=atom` Intel Atom 等）
- *Function-* 是函数粒度的多版本，除此之外还有诸如 `-march=<arch>` 的整二进制或模块 / 编译单元粒度的多版本

项目简介

FMV 有什么应用前景?

- “一次编译、多处运行”
  - 二进制执行文件/通用库可以在不同的 CPU 型号/特性之间兼顾兼容性和性能
  - ~~某 12-th Gen CPU 由于 E-Core 不支持 AVX512 而完全不支持 AVX512~~
- 有效减小体积/初始化开销
  - 函数粒度的多版本可以针对热点代码/函数进行优化，而无需其他冗余或者初始化开销
- 支持碎片化的生态（战未来）
  - 例如 RISC-V ，即使 RVA 配置相同，支持的扩展能力也不同
  - 我项目导师的 rv64.zip 就是为了解决 RISCV 生态碎片化提出的

项目简介

FMV 有什么应用前景？(Cont'd)

- 以 RISCV 的向量扩展为例子
  - RVV 0.7.1, 0.8, 1.0
  - `XTheadVector` 扩展（玄铁），based on RVV `0.7.1`
- 右图来自 `rv64.zip`

Microarchitecture	Real chip come to market	RVA Profile	Additional Important Extensions
Sifive U74	2022 (StarFive JH7110)	RVA20U64	Zba, Zbb
Xuantie C910	2023 (TH1520)	RVA20U64	xthead*
Xuantie C920	2023 (SG2042)	RVA20U64	xthead*
SiFive P550	2024 (ESWIN EIC7700X)	RVA20U64	Zba, Zbb
XiangShan Nanhu V2	2024	RVA20U64	Zba, Zbb, Zbs
Spacemit X60	2024 (Spacemit K1)	RVA22U64	Vector, Zicond
Xuantie C920v2		RVA22U64	Vector, Zicond, Zfa, Zawrs

项目简介

FMV 的实现原理

- IFUNC
  - GCC FMV 的核心分发机制依赖于 GNU 间接函数（Indirect Function, IFUNC）。这是一种在**运行时**选择函数实现的机制。IFUNC 是一种特殊的符号类型（`STT_GNU_IFUNC`），它并不直接指向函数代码，而是指向一个解析器（resolver）函数
- Multi-versioning + Dispatcher / Resolver 函数
  - IFUNC 可自定义 resolver 函数逻辑，而 `target_clones` 等是编译器自动生成 Resolver
  - Resolver 负责在运行时检测当前 CPU 的特性，并返回一个指向最合适的函数实现版本的指针
- `main` → `_Z4callv.resolver` → `_Z4callv.avx2_avx512f_mmx` / `_Z4callv`

项目成果

项目成果

关于 FMV 的功能实现

支持的语法（C 语言风格参数传递）：

```
!GCC$ ATTRIBUTES TARGET_CLONES("default", "avx", "avx512f") :: MySub
!GCC$ ATTRIBUTES TARGET("avx2") :: MyFunc
```

- 前端
  - 增加 `target`, `target_clones` 的 attribute，和对应的语法解析
  - 为符号扩展了存储字段用于检查和持久化
- 中端
  - 实现附加所需的 attribute，复用了现有的（C-like）`target` 和 `target_clones` 实现
- 测试
  - 书写了 `dejagnu` 格式的测试（由于使用 AVX 作为测试，存放在 `i386` testsuite 中）

项目成果

从 RFC 到收官被咕

- 被上游咕咕了
  - 发了多封邮件，CC 了众多 Fortran Maintainer，但是没～有～回～答～
  - 唯一收到的回复是提醒我邮件被送进垃圾箱了没看到，重新发了一封也没下文了（
  - P.S. 可能涉及到新的语法/特性大家都比较谨慎，没有 patch 好过
- 没能实现 C++ 风格的 `target`
  - 也是具体实践到这个项目才知道，C 和 C++ 中 `target` 的用法有区别
  - C++ 中支持“重载”一个函数，也就是可以在相同函数名的不同版本写完全不一样的代码
  - C 只支持为特定函数生成不同版本，多次定义会报 redefined
  - 由于 Fortran 和 C++ 的模型存在比较大的差异（见后文），和导师讨论之后决定优先级降低

项目经历

分享收获/踩坑

C vs C++

多版本生成机制的不同

- 对于 `.c`
  - GCC 中如果直接对两个相同函数名的函数进行 FMV 会报重复定义的错误
  - Clang 会要求先定义一个无参数的原型，也就是例如 `void call(void);`
- 对于 `.cpp`
  - G++/Clang 会智能的判断并且自动生成一个 dispatcher 分发到不同的逻辑
- 还有一些小的例如 suffix 生成的不同

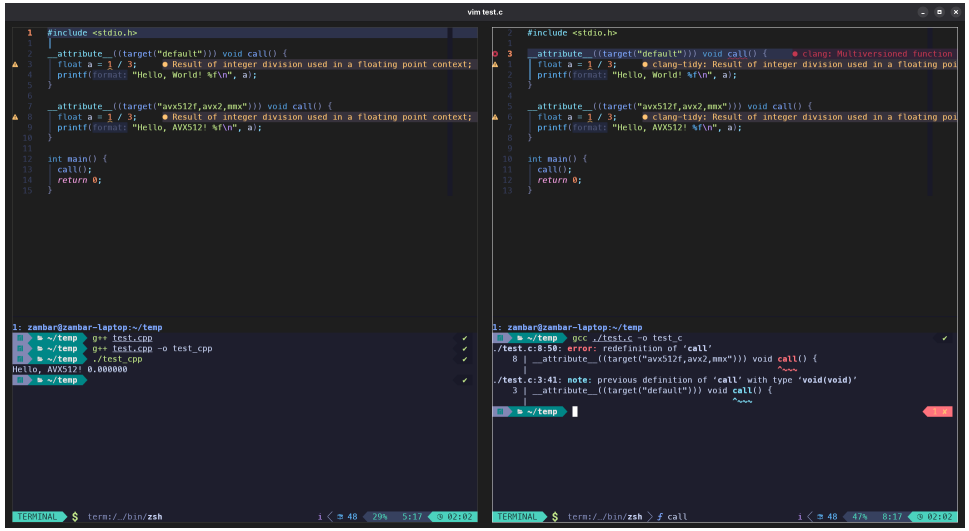

Figure 1: C 和 C++ 的多版本并不相同

符号去重计数器

关于 somefunc.1 的 suffix

- 找到 langhooks.cc 中添加计数器的情况：

```
if (TREE_PUBLIC (decl) || DECL_FILE_SCOPE_P (decl))
```

- 也就是，函数名会附加计数器当且仅当：
  - !TREE_PUBLIC (decl)：函数不是公共（public）的
  - !DECL_FILE_SCOPE_P (decl)：函数不是文件作用域的
- 即：**非公共且非文件作用域的函数会获得计数器。**
- Fortran 的函数可见性规则更复杂(可以通过 readelf 查看)
- 内部过程和模块内的私有过程通常不是公共的
- 生成出来就会带 .1 等类似的 suffix

C++ vs Fortran

实现 C++ 风格的多版本的阻碍

- 重复定义
  - C++ 的属性解析在前，之后再进入 `decls_match` 时已经有了属性（被标记为 `DECL_FUNCTION_VERSIONED`）
  - C++ 前端主要只进行一致性检查，留给中后端继续检查（`decl`），最晚可以延迟到 linking 阶段
  - Fortran 的重复检测主要在前端符号表构建时（`gfc_symbol`），重复定义会直接报错

如何给 GCC 写测试?

DejaGnu 初探

- GCC 中的测试大部分是基于 DejaGnu 框架，放在 `gcc/testsuite/` 目录下
- 可以在 “写 C/CPP” 的基础上，通过特定的注释来实现指定编译选项、检查编译产物等丰富的功能
- 例如我的测试需要模拟特定架构、检查汇编产物、甚至特定报错
  - `! { dg-require-effective-target avx512f }`
  - `! { dg-final { scan-assembler "\\.*\\.avx" } }`
  - `! { dg-final { scan-assembler "vmovss" } }`
  - `! { dg-error "TARGET_CLONES arguments must be character constants" }`

如何给 GCC 上游发邮件/Patch

- Commit 记得带上签名
- Commit 规范和传统的 `fix` / `feat` 等不一样
- 使用 `git email` 模块
  - 如果手动发送需要保证是纯文本消息，HTML 会被邮件列表拒收
  - `--subject-prefix="RFC"`：Request for Comments
  - `--cover-letter` 写 summary
- 关于礼仪（?）
  - TO/CC 的人可以从 Maintainers 里面找，也可以直接找修改的部分周围的 Author
  - 神秘的 GCC Coding Convention: `Lines shall be at most 80 columns.`，似乎对邮件也适用

收获和总结

这次是我第一次参加 OSPP，也是第一次对超大型项目进行功能增添和 patchwork，收获颇丰。首先非常感谢我的导师**陈泱宇老师**，在这几个月里面积极沟通，并且将自己曾经的 GCC Patch 经验不厌其烦地传授给我，把握我向上游的邮件风格，提出了许多的指导性建议，并且向我展示了这个工作在 RISC-V 生态下的巨大潜力。其次还要感谢清华 TUNA，**党老师和陈老师**也在每周的例会中旁听和跟进我们项目的进展，对项目的方向有很大的推动和把握作用。以及需要感谢 **PLCT 实验室、和一众个人博客**，让我能够从 scratch 学习到 GCC 的贡献方法，以及众多的灵感和思路。最后还要感谢和我一起参加这次 OSPP 的**犬戎同学**，虽然未曾谋面，但是私下我们也一起交流了很多项目上的心得，共同参加数次周会，是不可多得的朋友。

Thanks for listening

Any questions?