OpenEuler | Compiler SIG

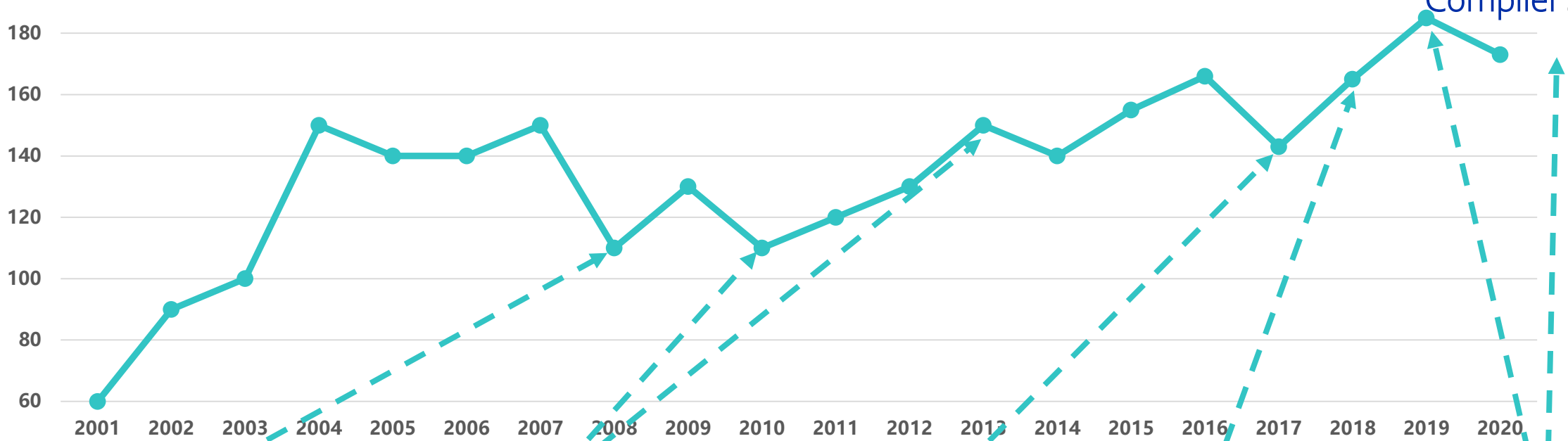
自动反馈优化特性介绍

单位：华为技术有限公司

汇报人：李彦成 黄晓权

反馈优化研究趋势

OpenEuler Compiler SIG



- 软硬结合的反馈优化提出[1, 2]
- 轻量化全局优化LIPO提出[3]
- 软硬结合反馈优化引入LBR [4, 5]，并与LIPO结合[5]
- 数据中心profiling框架GWP提出[6]
- AutoFDO框架提出[7]
- 函数布局优化提出[8]，结合硬件profiling和hfsort
- 二进制反馈优化工具BOLT提出[9, 10]

Google团队

Facebook团队

[1] Feedback-Directed Optimizations with Estimated Edge Profiles from Hardware Event Sampling, Open64 Workshop at CGO’08.
[2] Feedback-Directed Optimizations in GCC with Estimated Edge Profiles from Hardware Event Sampling, Proceedings of the GCC Developers’ Summit, 2008.
[3] Taming hardware event samples for FDO compilation, In CGO’10.
[4] Lightweight Feedback-Directed Cross-Module Optimization, In CGO’10.
[5] Taming hardware event samples for precise and versatile feedback directed optimizations, TOC, 2013.
[6] Google-wide profiling: A continuous profiling infrastructure for data centers. MICRO, 2010.
[7] AutoFDO: Automatic Feedback-Directed Optimization for Warehouse-Scale Applications, CGO’16.
[8] Optimizing Function Placement for Large-Scale Data-Center Applications, CGO’17.
[9] BOLT: A Practical Binary Optimizer for Data Centers and Beyond, CGO’19.
[10] Lightning BOLT: Powerful, Fast, and Scalable Binary Optimization, CC’21.

反馈优化编译器社区支持情况

OpenEuler Compiler SIG

2004年：GCC3.4支持反馈优化

2014年：Clang 3.5支持反馈优化及AutoFDO

2014年：Google的AutoFDO采样数据格式转换工具开源

2015年：GCC 5.0支持自动反馈优化

2016年：Facebook的hfsort工具github开源

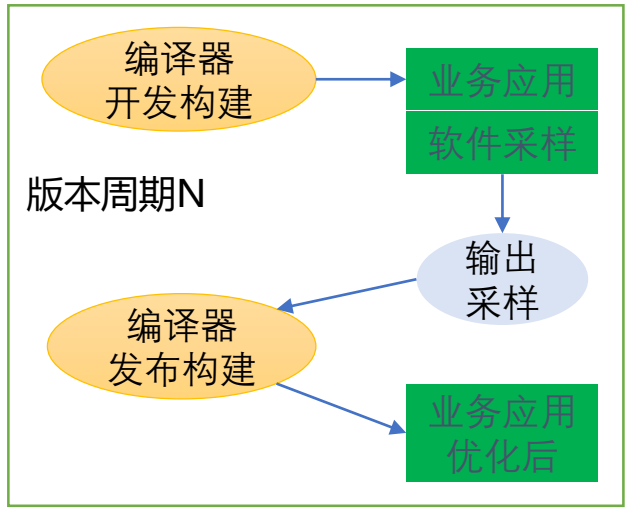
2018年：Facebook的BOLT工具github开源

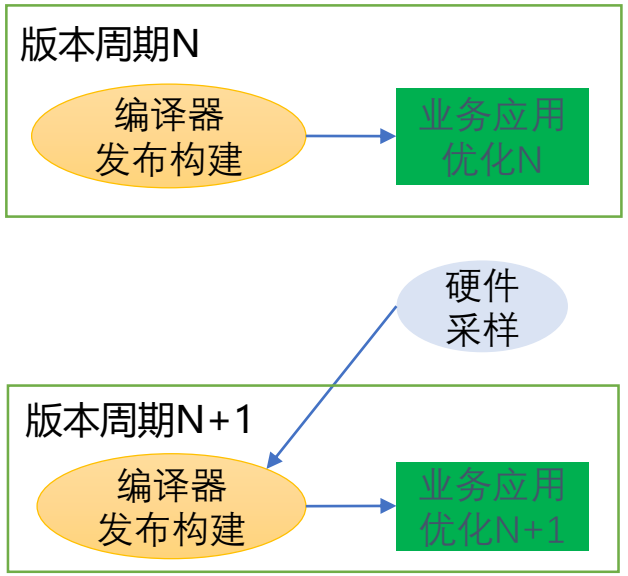
2020年：Clang支持CSSPGO，与Google的采样数据格式转换工具解耦

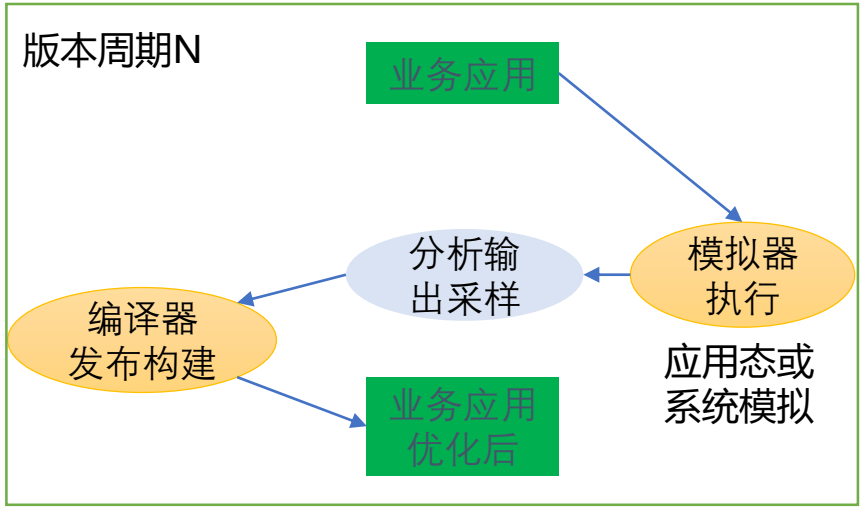
2021年：GCC社区讨论AutoFDO维护及与采样格式转换工具解耦

不同形式反馈优化的区别

OpenEuler Compiler SIG







插桩反馈编译：
1. 版本周期内编译器两次构建
2. 软件插桩代码输出采样，时间长，无硬件事件（cache miss）

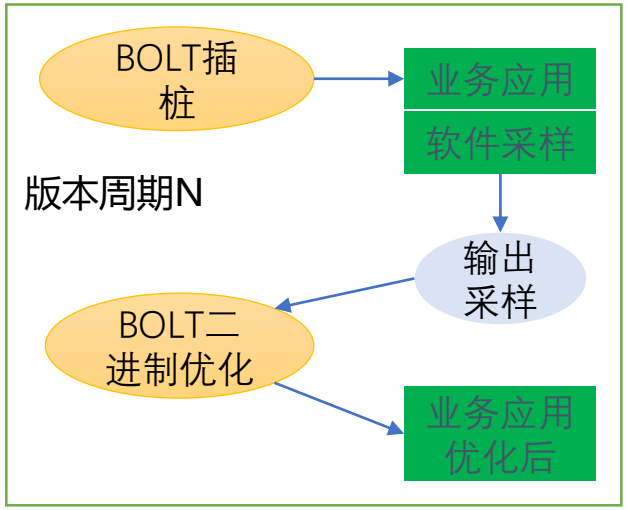
自动反馈编译AutoFDO：
1. 版本周期内编译器一次发布构建
2. 硬件输出采样，时间短有精度损耗，有硬件事件（cache miss）

智能预取：
1. 版本周期内模拟器执行、编译器发布构建、算法分析采样数据
2. 软件模拟输出采样，时间非常长，有硬件事件（cache miss）

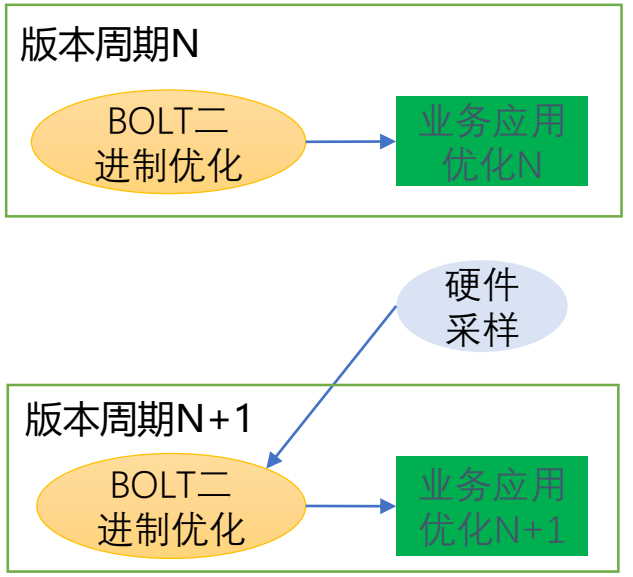
根本差异：采样数据收集方式

BOLT二进制反馈优化

OpenEuler Compiler SIG



插桩反馈优化



硬件采样反馈优化

优势：

- 二进制层面直接优化，不需要对应回源码，精确度高
- 二进制层面的全局优化，优化效果好，与编译器的LTO优化相比应用场景更广泛
- 可以与AutoFDO叠加使用

劣势：

- 二进制级别优化难以感知代码高级语义，丧失了做部分高级优化的机会
- 与AutoFDO合用时需要进行两次采样

当前AutoFDO及BOLT的短板

OpenEuler Compiler SIG

多样性算力相关问题：

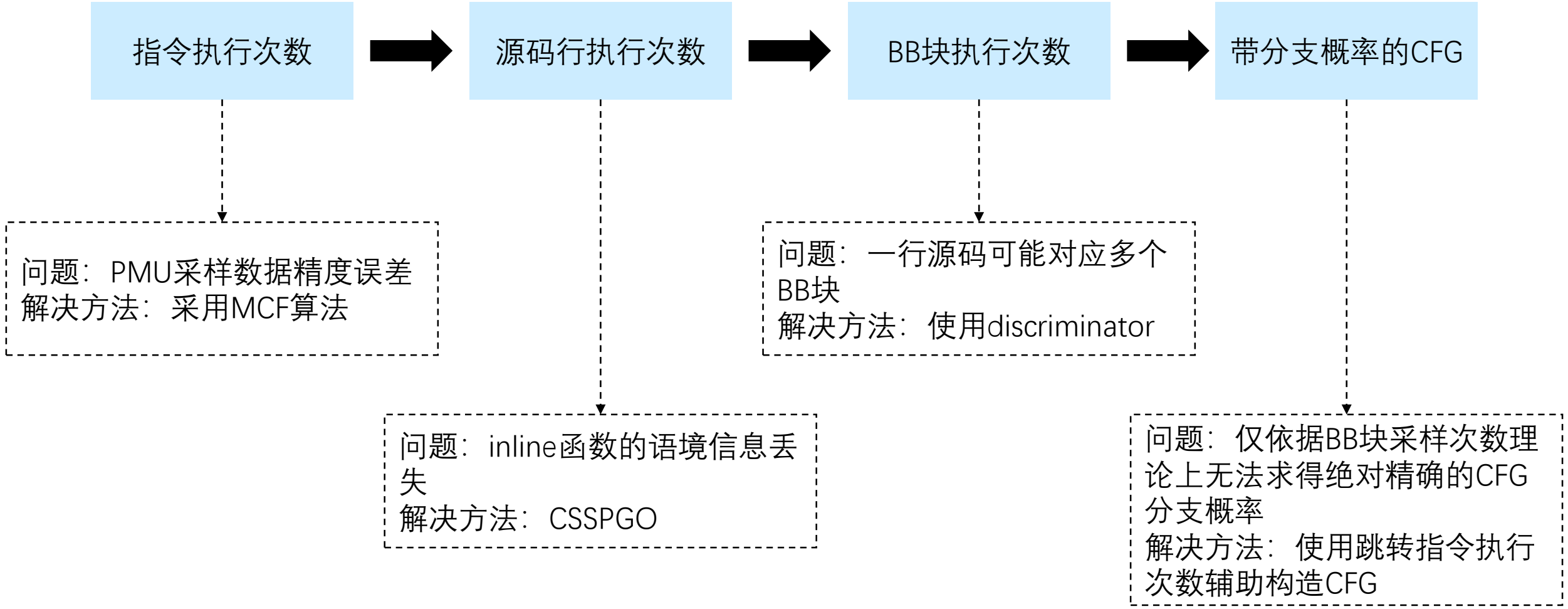
- 强依赖x86 PMU的LBR采样能力，对于其他架构的支持并不友好
- 缺少LBR而仅依据BB块采样次数很难准确地推断代码分支的执行概率

技术自身的短板：

- 目前AutoFDO只聚焦于插桩反馈优化的能力，未充分的利用PMU能够采集的信息
- 目前BOLT与AutoFDO结合使用需要进行一次额外的采样和额外的链接

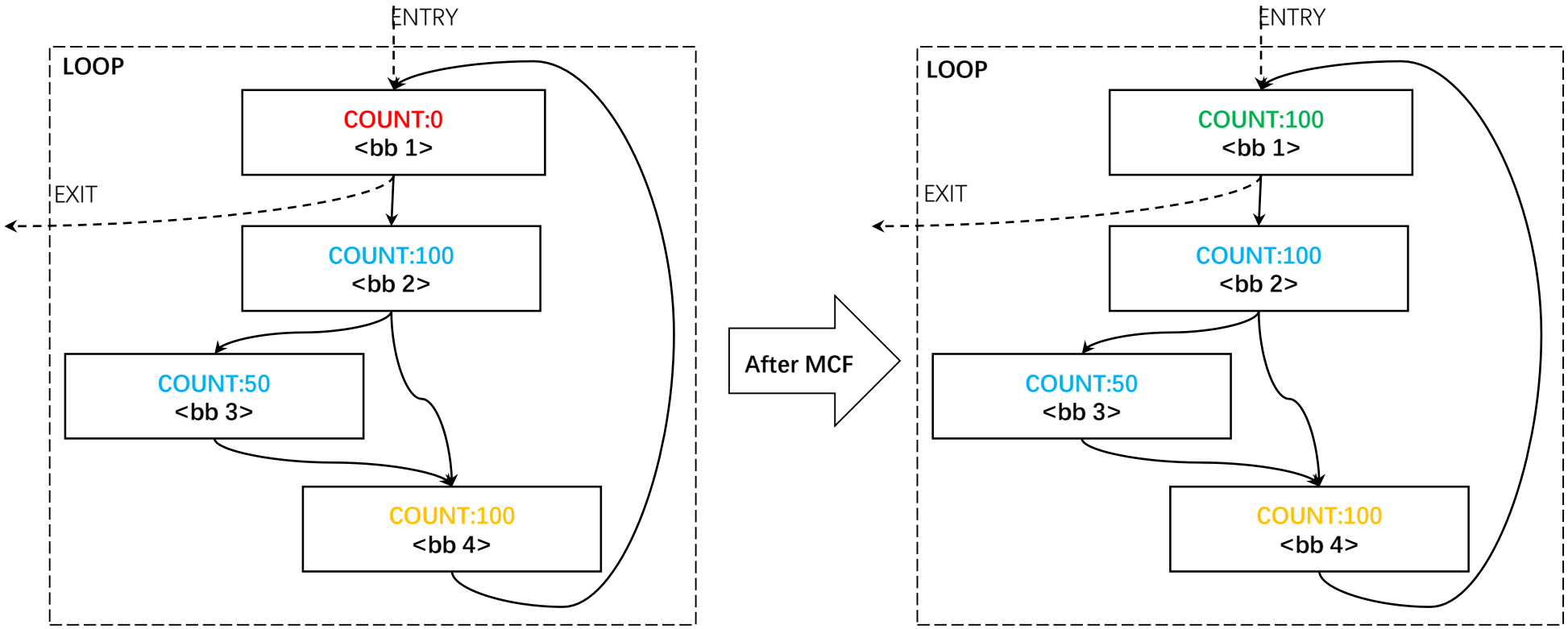
CFG构造精度问题

OpenEuler Compiler SIG



PMU采样数据准确性提升

OpenEuler Compiler SIG



通过MCF算法提升采样数据准确性

Discriminator

Source:

0: FUNCTION()

1: for(init_stmt ; test_stmt ; update_stmt) {

2: LOOP_BODY

3: }

FUNCTION() total:X

head:X

1: LOOP_COUNTS

2: LOOP_COUNTS

FUNCTION() total:X

head:X

1.0: 1

1.1: LOOP_COUNTS + 1

1.2: LOOP_COUNTS

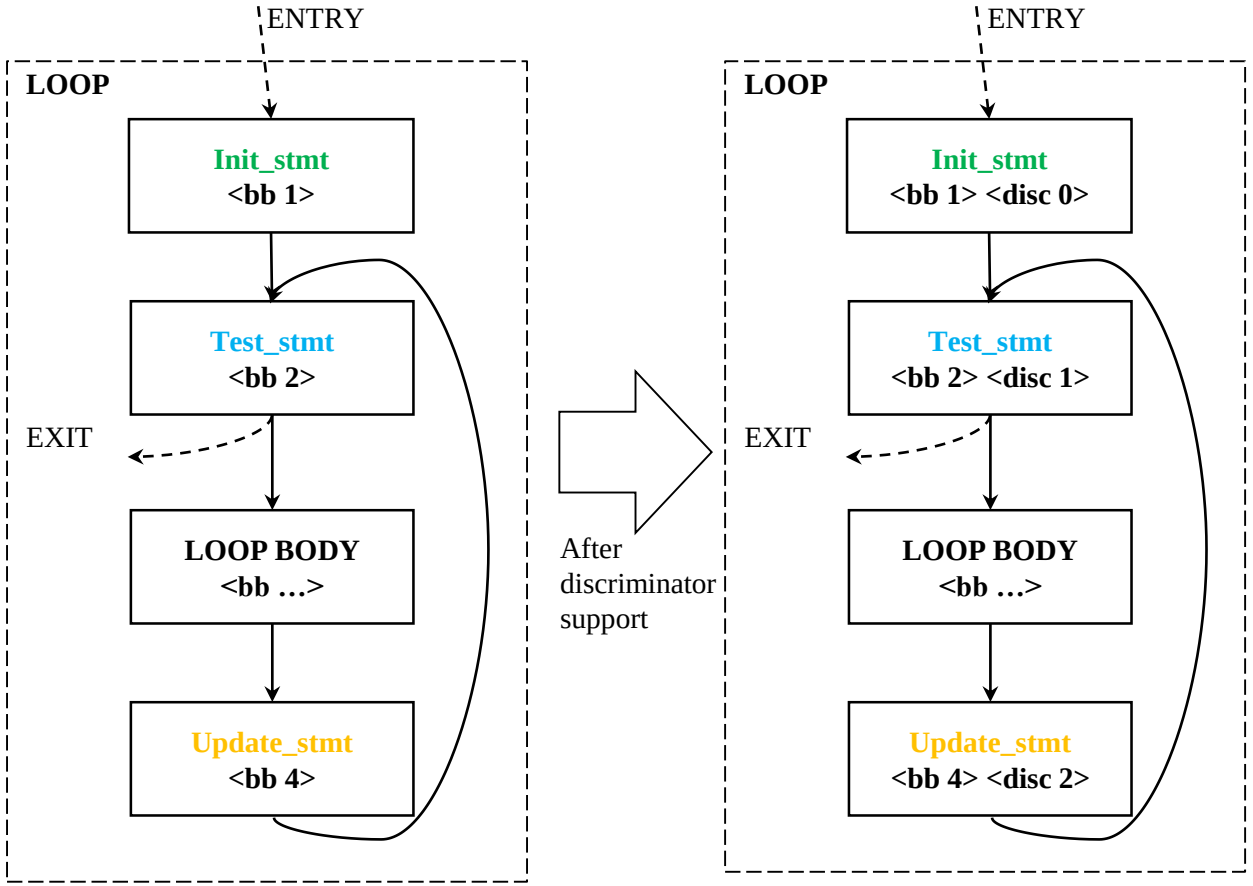
2: LOOP_COUNTS

Discriminator支持前：

- 同一行源码对应多个BB块的情况下，编译器无法区分多个不同分支的BB块计数；

Discriminator支持后：

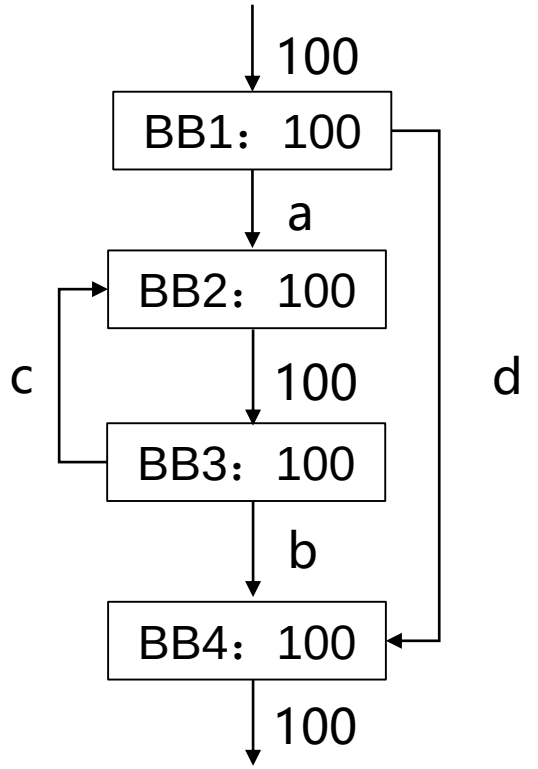
- 同一行源码的对应的每个BB块计数都能够被正确处理；



BB块采样求CFG分支概率的局限

OpenEuler Compiler SIG

问题：



$$\begin{cases} a+c=100 \\ b+c=100 \\ b+d=100 \end{cases}$$

无穷多解

解决方案

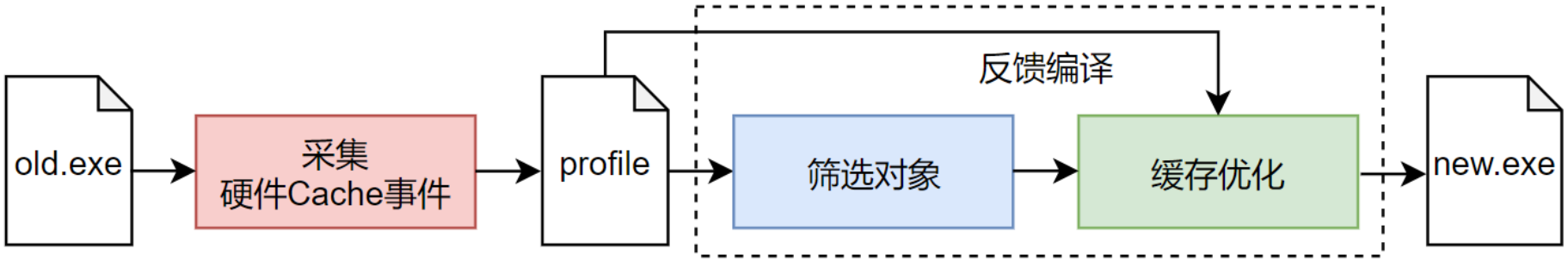
指令执行次数 → 源码行执行次数 → BB块执行次数

分支指令跳转次数 → 源码行跳转次数 → BB块跳转次数

BB块执行次数、BB块跳转次数 → 带分支概率的CFG

预取反馈优化

OpenEuler Compiler SIG



- 使用硬件Cache事件指导预取对象的筛选
- 提出一种分支加权的预取距离算法，基于反馈信息计算分支的执行概率，精准的计算预取距离，提升优化精度

分支加权的预取距离计算方法

OpenEuler Compiler SIG

利用反馈信息，计算控制流图的主要执行分支，提高预取准确性

1、筛选计算集中于循环体内的循环：

$$\mathrm{loop}_{\mathrm{rate}} = 1 - \frac{\mathrm{sum}\left(\mathrm{exit}_{\mathrm{bb}} \times \mathrm{exit}_{\mathrm{edge}}\right)}{\mathrm{header}_{\mathrm{bb}}}$$

$$= 1 - \frac{100 \times 1\%}{100} = 99\%$$

2、计算循环体内BB块执行概率：

$$\mathrm{BB1} = 100\%,$$

$$\mathrm{BB2} = \mathrm{pre}_{\mathrm{bb}} \times \mathrm{pre}_{\mathrm{edge}} = 100\% \times 1\% = 1\%$$

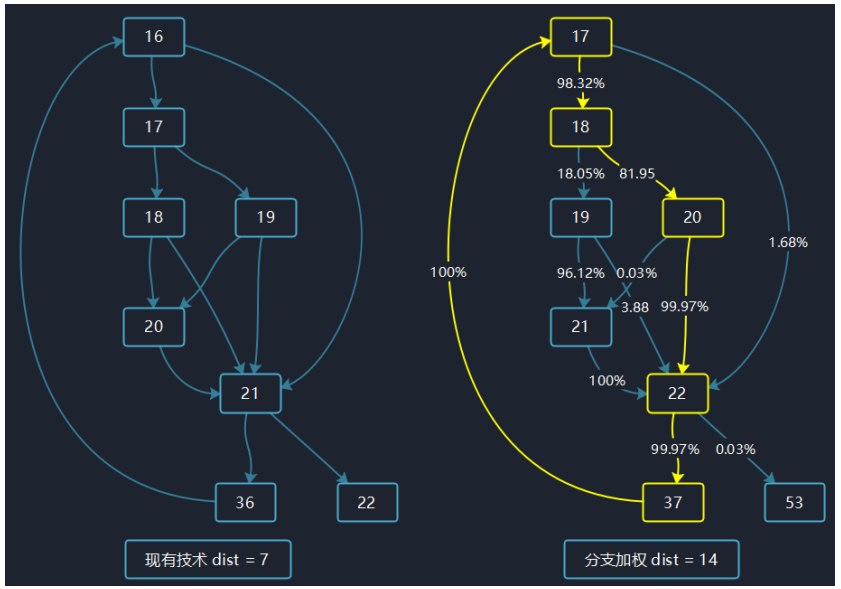
3、得到循环体的期望运行时间：

$$\mathrm{loop}_{\mathrm{time}} = \mathrm{sum}(\mathrm{BB}_{\mathrm{prob}} \times \mathrm{sum}(\mathrm{inst}_{\mathrm{time}}))$$

4、根据循环体的期望运行时间求取预取距离：

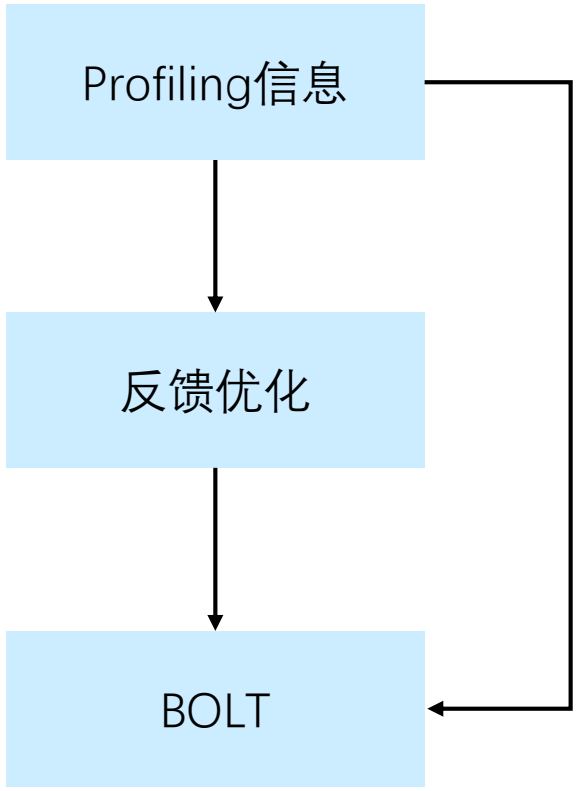
$$\mathrm{prefetch}_{\mathrm{dis}} = \frac{\mathrm{latency}}{\mathrm{loop}_{\mathrm{time}}}$$

```
for ( ; arc < *end_arc; arc += num_threads) {
  if( arc->ident > BASIC)
  {
    /* red_cost = bea_compute_red_cost( arc ); */
    static long long cnt = 0;
    cnt++;
    red_cost = arc->cost - arc->tail->potential + arc->head->potential;
    if( bea_is_dual_infeasible( arc, red_cost ) )
    {
      basket_sizes[thread]++;
      perm[basket_sizes[thread]]->a = arc;
      perm[basket_sizes[thread]]->cost = red_cost;
      perm[basket_sizes[thread]]->abs_cost = ABS(red_cost);
      perm[basket_sizes[thread]]->number = 0;
    }
  }
}
```

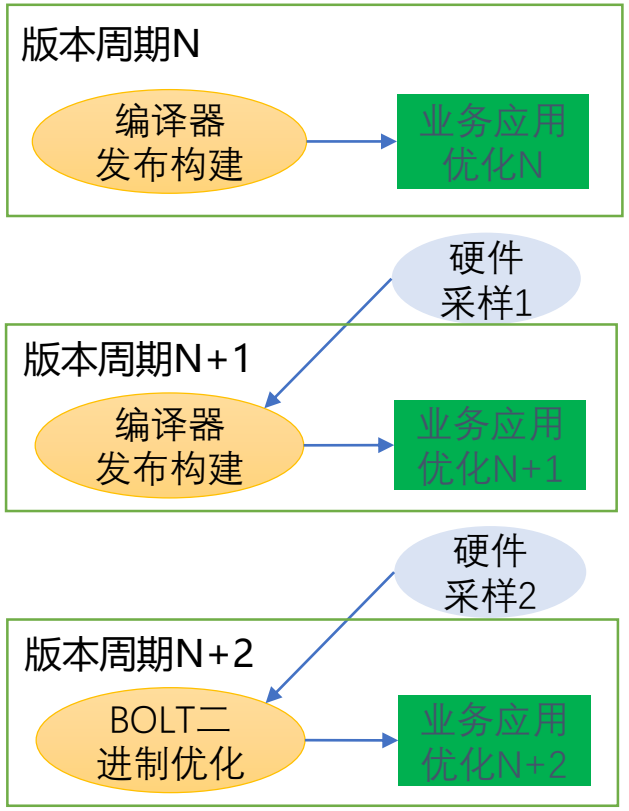


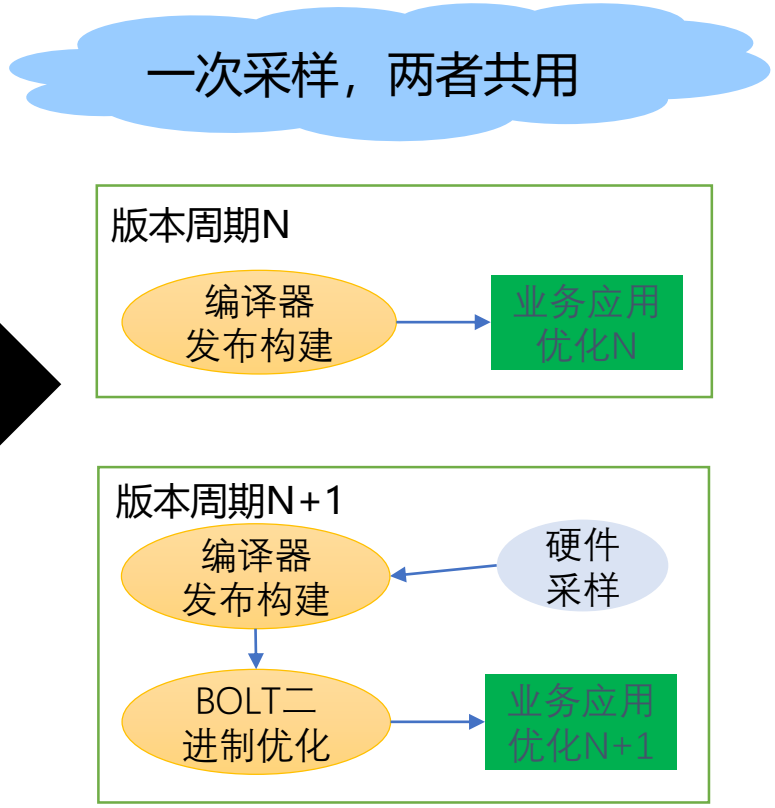
反馈优化与BOLT的集成

OpenEuler Compiler SIG



集成后的反馈优化流程





A-FOT

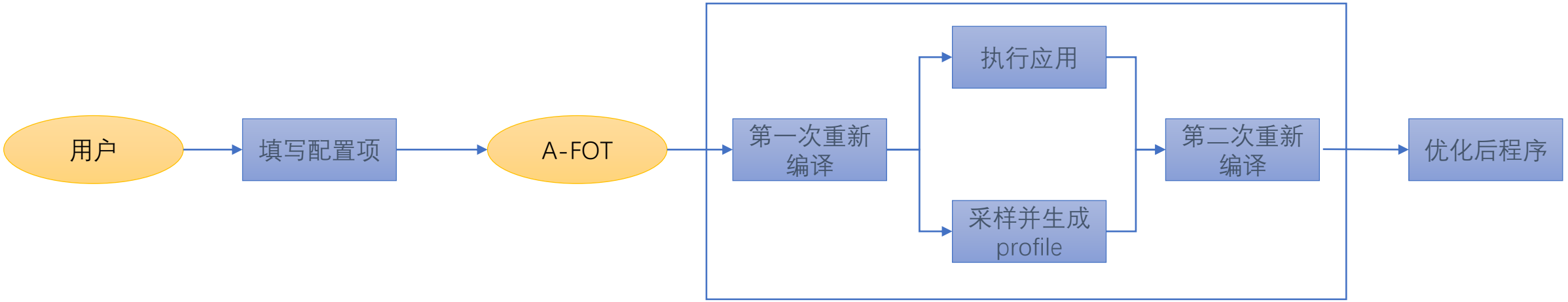
OpenEuler Compiler SIG

A-FOT(automatic feedback-directed optimization tool)：

- 是一款用于提升编译器openEuler GCC自动反馈优化特性易用性的工具
- 用户通过较少的配置即可自动完成反馈优化的相关步骤（包括编译、采样、分析、优化等）

技术细节：

- 支持AutoFDO、AutoPrefetch、AutoBOLT模式，应用模板设计模式，易于增改优化模块
- 使用wrapper自动添加所需编译选项
- 支持自动检测优化结果
- 目前仅支持单实例应用优化
- 未来计划集成分析能力，自动确定采用何种优化模式




OpenEuler | Compiler SIG