基于BOLT的静态指令分布统计

华东师范大学数据学院
2025级代健坤

Benchmark选取

在对CppPerformanceBenchmarks所有benchmark进行处理后，选择了以下较为典型、容易理解、方便分析和解释的5个程序。

1. 计算密集型：matrix_multiply、convolution、shift
2. 内存操作型：memcmp
3. 控制流密集型：binary_search

X86-64指令分类

参考Intel® 64 and IA-32 Architectures Software Developer Manuals -> CHAPTER 5 INSTRUCTION SET SUMMARY

Data Transfer Instructions：负责在内存与通用寄存器以及段寄存器之 间进行数据的转移

Arithmetic Instructions：对存储在内存中或通用寄存器中的数据 进行整数运算

Logical Instructions：对数据执行基本的与、或、异或和非 逻辑运算

Shift and Rotate Instructions：对操作数中的位 进行移位和旋转操作

Control Transfer Instructions：控制转移指令包含跳转、条件跳转、循环以及调用和返回等操作，用于控制程序流程。

Comparison Instructions：对比两个操作数的大小或相等性，并通过设置状态标志位为后续的条件分支提供 依据

Miscellaneous Instructions：其他

反汇编出现的X87 FPU、MMX、INTEL® SSE、INTEL® SSE2指令被分类到以上各基本指令中。

工具运行逻辑

1. 使用bolt对可执行文件反汇编 llvm-bolt --print-disasm ./bin/memcpy > ./disasm/memcpy_disasm_output.txt 2>&1 -o ./bolt_output/memcpy.bolt

   或使用脚本batch_bolt.sh对二进制文件批量反汇编

2. 使用preprocess.py得到反汇编文件中出现的指令 python preprocess.py disasm/*（可选）
3. 手动/AI辅助，对指令进行分类（可选）
4. 对反汇编文件指令进行统计和绘制堆叠柱状图 python3 main.py ./disasm/divide_disasm_output.txt
5. 对反汇编文件按照函数进行统计和绘制柱状图 python3 function_stat.py /memcmp/*（可选）

   https://github.com/JiankunDai/inst-static-analysis

指令分布计算

- 指令分类统计:

  按指令类型(算术、逻辑、存储、控制等)分类

  统计每类指令的出现次数

- 计算比例:

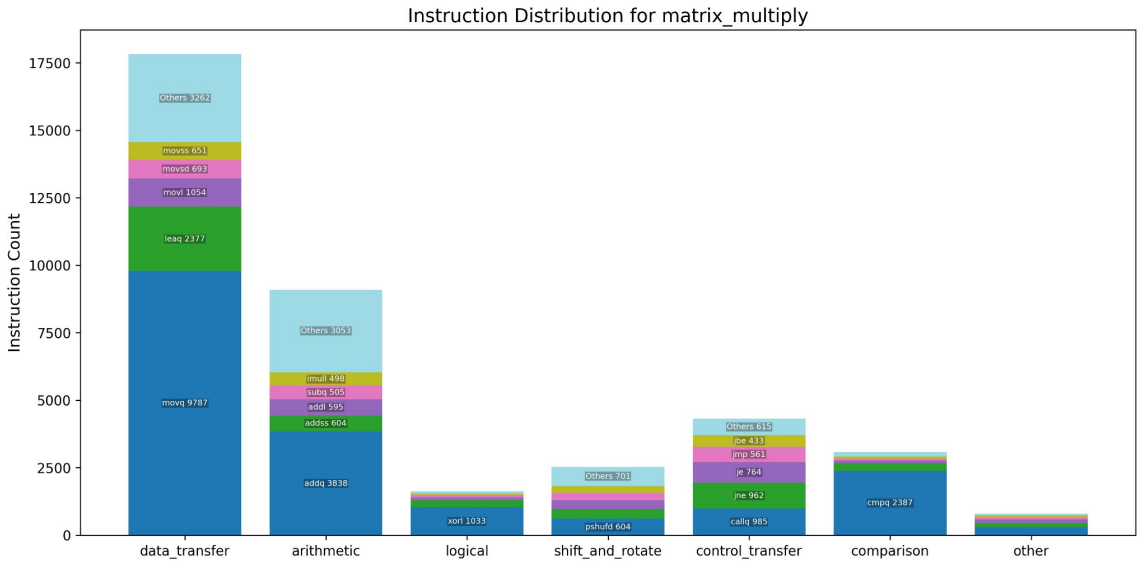
  总指令数 = Σ(各类指令数)

  各类指令比例 = (该类指令数)/总指令数 × 100%

matrix_multiply

矩阵乘法程序包含较多的算数指令(约25%)，如add和imul

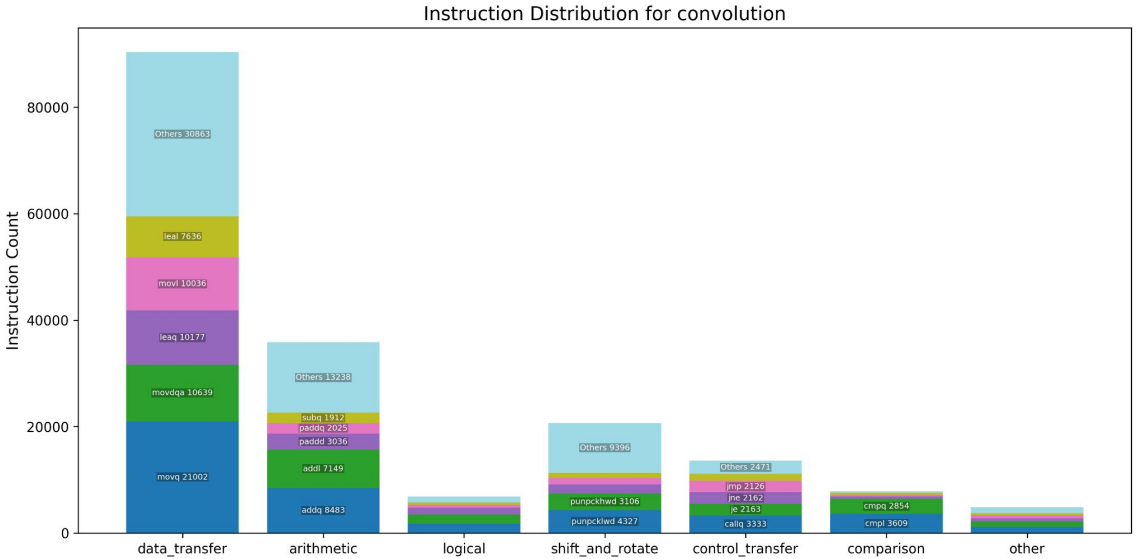
多重嵌套循环每个都需要独立的终止条件检查，每次循环迭代都会生成一个cmpq指令来比较循环变量



convolution

卷积程序包含较多的算术运算（约25%），如addq、addl指令。

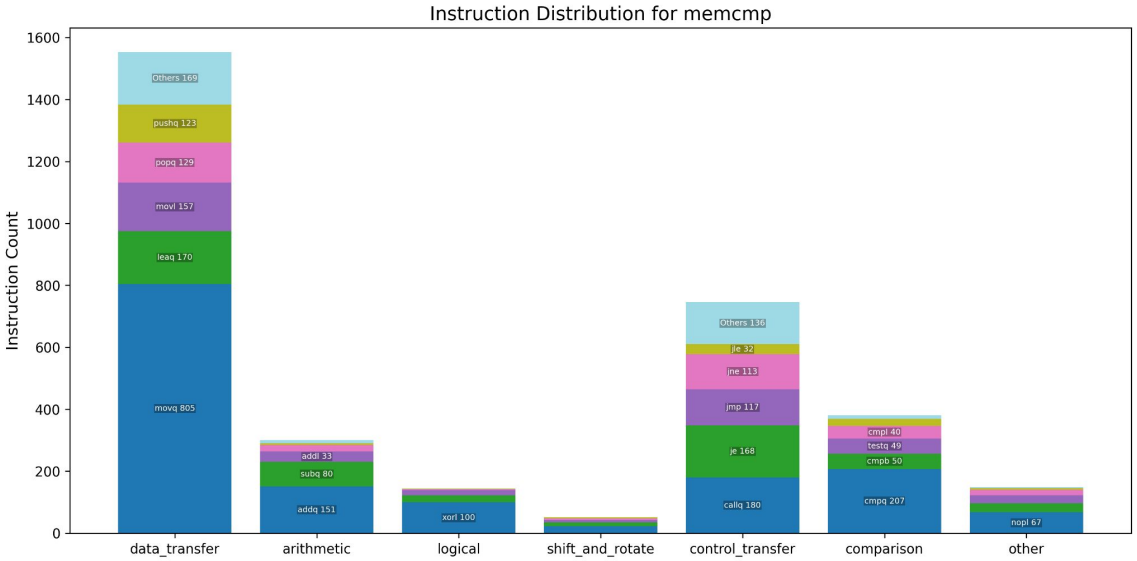
与矩阵乘法程序不同的是，由于编译器优化卷积运算，有较多SIMD指令paddd等生成。



memcmp

内存比较的主要流程是通过循环或迭代器对单字节或同时对多字节进行比较。

指令组成主要包括内存加载，比较，差值计算和分支跳转。

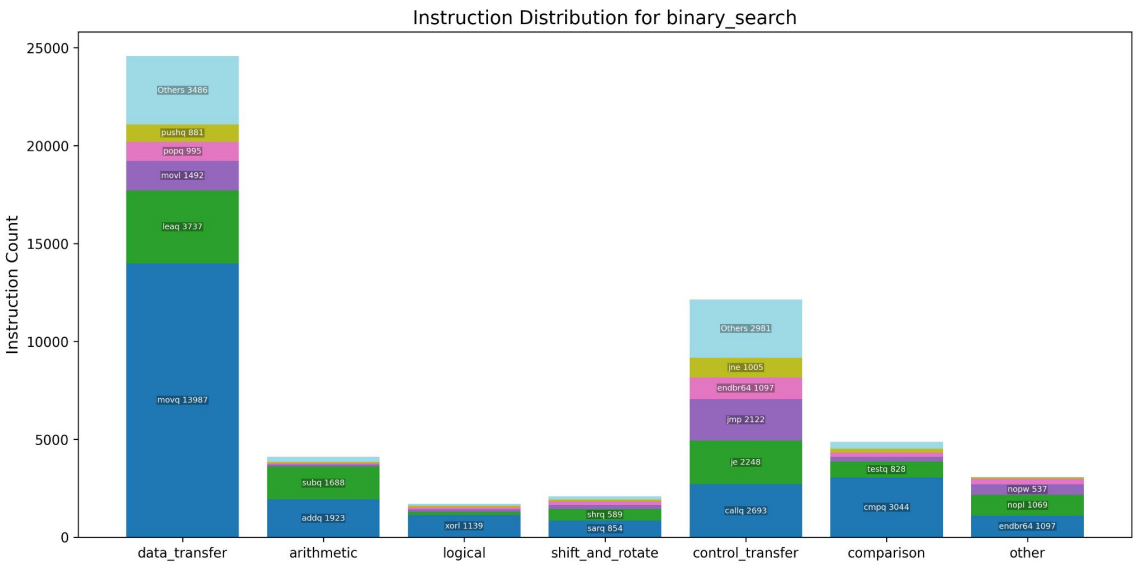


binary_search

二分搜索由于循环和条件分支较多，所以控制转移指令占比较多（约23%），例如je、jmp。

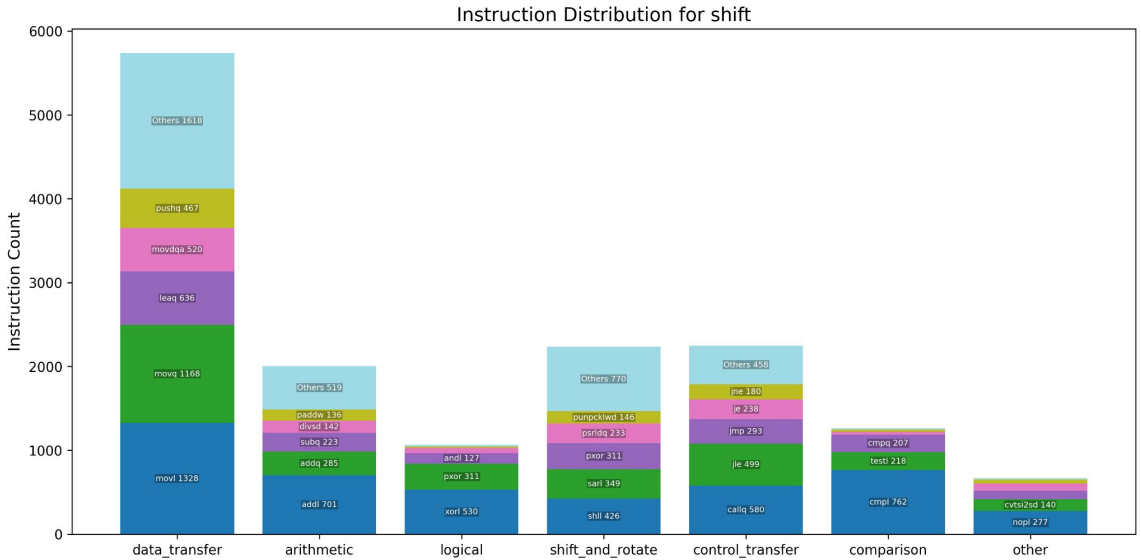
由于存在较多的迭代器比较和元素值比较，比较指令也出现较多（约9%），例如cmpq和testq。

由于部分测试函数使用递归版本实现，有较多call指令的生成。（猜想）



shift

相对于其他程序，shift程序特点是有着较多的移位运算和逻辑运算，如shl和xor。



分析方法的缺陷

试图通过字母频率分析一本书的主题，而忽略了单词和段落的结构。

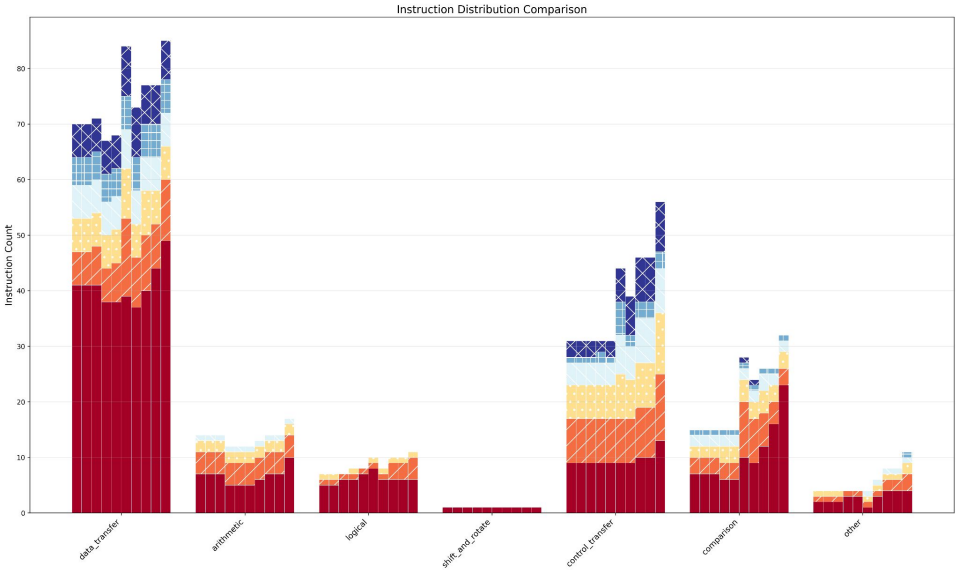
所以尝试了深入分析单个程序的函数级静态指令分布。

对memcmp的进行函数级别的指令分布统计

对memcmp的测试函数分别进行了指令分布的统计。

同一个指令类型自左向右分别为为stl库、for循环、C++迭代器、for循环展开、数据宽度优化、缓存行优化的实现所产生的指令。

虽然更高级别的实现方式需要的指令数量略微增多，但通过运行benchmark观察到其带来的数据处理效率提升是显著的。



可执行文件静态指令分析的意义

- 热点识别:统计高频指令可指导定位性能瓶颈。
- 编译器优化验证:对比不同编译优化级别的指令分布，验证编译器的优化效果。
- 处理器微架构设计:统计指令比例可指导CPU流水线、缓存大小、分支预测器的设计和指导专用硬件加速单元设计。

静态指令分布对指导程序优化的缺陷

静态指令分布分析在指导程序优化时虽然提供了有价值的信息，但仍存在多方面的局限性和缺陷，可能误导优化方向或掩盖真实性能问题。

- **忽略动态执行行为**

  静态分析无法反映运行时分支的实际走向，导致CPU分支预测的失误，影响流水线效率。

  未执行的指令会被静态统计计入，导致优化资源浪费。

- **缺乏数据依赖关系**

  静态统计无法识别由于数据依赖造成的流水线阻塞

- **部分指令开销低估**

  占比低的指令也可能成为瓶颈

总结

静态指令分布分析是程序优化的起点而非终点。

其核心缺陷在于脱离实际执行的时空上下文，仅依赖静态数据可能导致优化南辕北辙。

有效的优化需结合静态与动态分析、硬件特性及**具体应用场景**，形成闭环反馈。

AI创作声明

本作业的思路、指令分类和数据呈现较多的借鉴和参考了腾讯元宝所提供内容。