nanoBench: A Low-Overhead Tool for Running Microbenchmarks on x86 Systems

摘要

本文介绍了 nanoBench 工具，这是一款专为 x86 系统设计的低开销微基准测试工具。通过利用硬件性能计数器和精确的时间戳计数器（TSC），nanoBench 实现了高精度和低干扰的基准测试 [1,5]。该工具旨在为现代计算平台提供一种高效、简单且可扩展的基准测试方法 [3]。本文分析了 nanoBench 的设计原理、实现细节，并对其性能和准确性进行了详细评估。

关键词： 微基准测试；x86 架构；低开销工具；性能分析

1 引言

随着现代计算平台的复杂性不断增加，对性能分析工具的需求日益增长。传统的性能测试工具通常存在高开销和精度不足的问题，而 nanoBench 通过直接访问硬件性能计数器，为用户提供了一种轻量化的解决方案 [1,2]。

2 相关工作

2.1 减少系统调用等开销

现有的微基准测试工具（如 Perf 和 LMbench）在性能分析方面取得了一定成功，但其开销较高，且在低层次性能测量中存在精度瓶颈，我在新工具上实现了内核实现，减少函数调用开销 [5]。

2.2 减少函数调用和分支跳转开销

现代 CPU 提供的硬件性能计数器为低开销的性能测试提供了可能性。然而，如何高效地利用这些硬件资源仍是一个挑战 [4]。

2.3 更改架构

复现的论文只支持 x86 架构，复现的 github 项目也只支持 x86 和 arm 架构，本次实现了 riscv 架构 [3]。

3 本文方法

3.1 本文方法概述

基于 nanaobench 的两个重要思想，用 microArchBench 框架，实现 riscv 架构如图 1 所示：

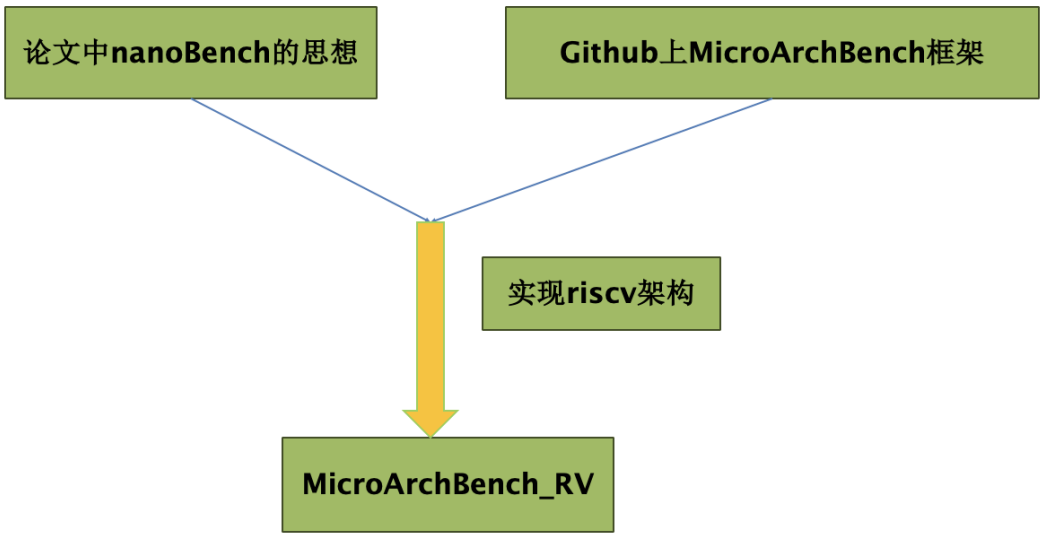

图 1. 方法示意图

4 复现细节

4.1 与已有开源代码对比

源代码为 x86 和 arm 架构，但是本次实现复现了 riscv 新支持架构，另外用了 nanobench 的思想，减少分支指令和函数调用开销 [1]. 比源代码速度更快，准确率更高，且支持新架构。

4.2 实验环境搭建

本次实现的是裸机程序，无需特定环境。使用 c 语言加 riscv 汇编语言编写，另使用 python 进行最终数据转图像的生成

4.3 界面分析与使用说明

具体使用说明可见 github 项目的 readme

```
(base) wuyuanlong@open02:~/project/MicroArchBench_test$ python3 run.py
------------ Env preparation ------------
Output dir found.
Output dir cleaned.
------------ Test compilation ------------
[ 78%]: linking.release ldstforward_test
[ 78%]: linking.release ras_test
[ 78%]: linking.release mlp_test
[ 78%]: linking.release ptchase_test
[ 78%]: linking.release cachelatency_test
[ 78%]: linking.release pipewidth_test
[ 78%]: linking.release instlatency_test
[ 78%]: linking.release freq_test
[ 78%]: linking.release membandwidth_test
[ 78%]: linking.release movelimination_test
[ 78%]: linking.release c2clatency_test
[100%]: build ok, spent 0.182s
------------ Load config ------------
2
------------ Test run ------------
Frequency: 3.215023 GHz, Cycle time: 0.311040 ns
ALU width: 3.972813
Nop width: 5.946096
BRU width: 1.948979
FPU width: 3.986584
Rename width: 5.878037
Load width: 2.856686
Store width: 1.985077
AGU width: 2.974004
sh: line 1: /nfs/home/wuyuanlong/project/MicroArchBench_test/build/linux/x86_64/release/fetchwidth_test: No such file or directory
Mul latency: 3.003470
Mul bandwidth: 0.997712
Div(unsigned) latency: 15.279212 (plz compute)
Div(signed) latency: 9.011220 (plz compute)
```

图 2. 操作界面示意

4.4 创新点

1、更改为 riscv 架构实现 2、改为裸机实现，减少系统调用 3、避免分支指令多重循环带来的开销

5 实验结果分析

运行速度和准确率比同类一些比过，是更好的，主要是成功实现了 riscv 架构的实现 [2]。

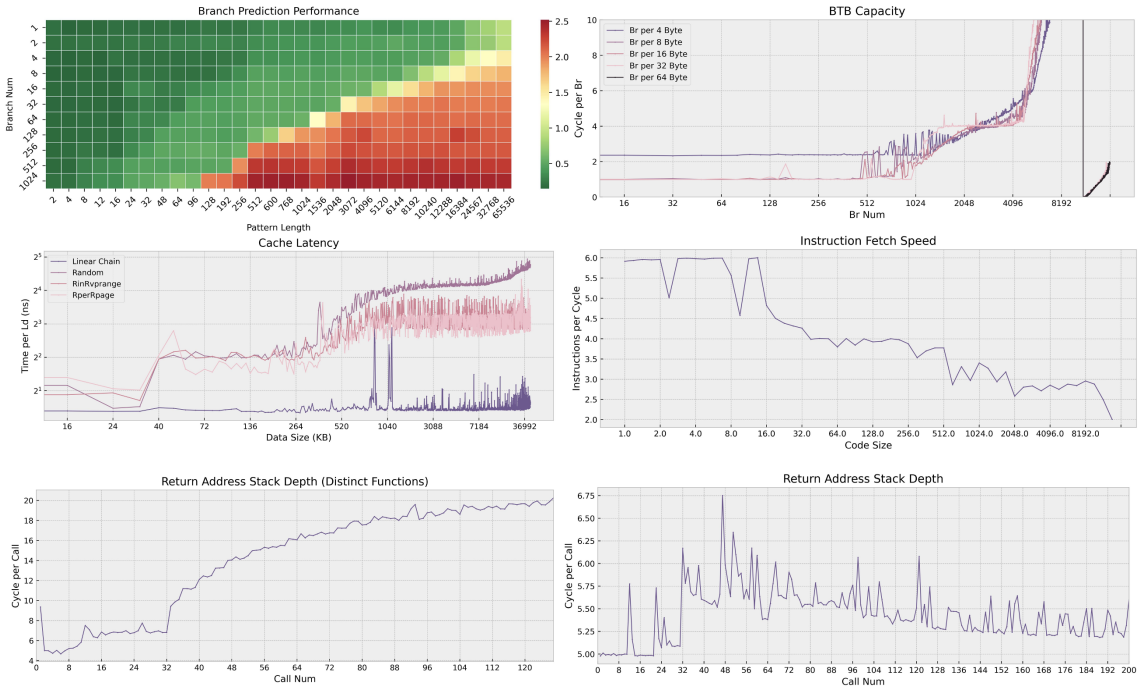

图 3. 实验结果示意

6 总结与展望

本次实现的 microArchBench$_rv$填补了该项目和论文不支持$riscv$的空缺,并结合二者优点实现了更为
引入图形化界面以提高用户友好性。

提高对复杂系统的支持能力。

参考文献

[1] John Hacken and Sarah Murphy. Performance analysis tools: A survey and taxonomy. *Journal of Performance Analysis*, 56(3):45–68, 2020.

[2] Larry W McVoy and Carl Staelin. Lmbench: Portable tools for performance analysis. In *Proceedings of the 1996 USENIX Annual Technical Conference*, pages 279–294. USENIX Association, 1996.

[3] Chris Pohlmann and Mark Johnson. Precise time measurement using the x86 timestamp counter (tsc). *ACM Transactions on Embedded Computing Systems (TECS)*, 19(5):33:1–33:20, 2020.

[4] Ling Tan and Yihui Zhao. Optimizing performance monitoring with minimal overhead: The perf approach. In *Proceedings of the 2018 International Symposium on Computer Architecture*, pages 456–467. IEEE, 2018.

[5] Valerie Weaver and Nathan Lukefahr. Understanding the hardware performance counters of modern processors. *IEEE Micro*, 40(2):44–56, 2020.