湖南大學

HUNAN UNIVERSITY

本科生毕业论文（设计）

论文(设计)题目：川菜和湘菜在辣度上的对比研究

学生姓名：十三香

学生学号：201212313231

专业班级：食品工程

学院名称：后勤保障部

指导老师：王守义

2025 年 1 月　日

湖 南 大 学

毕业论文（设计）原创性声明

本人郑重声明：所呈交的论文（设计）是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文（设计）不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

学生签名：　　日期：2025 年　月　日

毕业论文（设计）版权使用授权书

本毕业论文（设计）作者完全了解学校有关保留、使用论文（设计）的规定，同意学校保留并向国家有关部门或机构送交论文（设计）的复印件和电子版，允许论文（设计）被查阅和借阅。本人授权湖南大学可以将本论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本论文（设计）。

本论文（设计）属于

1、保　密□，在_____年解密后适用于本授权书。

2、不保密□。

(请在以上相应方框内打‘‘√’’)

学生签名：　　日期：2025 年　月　日

导师签名：　　日期：2025 年　月　日

川菜和湘菜在辣度上的对比研究

摘　　要

摘要是论文内容的简要陈述，是一篇具有独立性和完整性的短文。摘要应包括本论文的创造性成果及其理论与实际意义。摘要中不宜使用公式、图表，不标注引用文献编号。避免将摘要写成目录式的内容介绍。

关键词： 关键词1；关键词2；关键词3；关键词4

A Research Conducted on the Spiciness between SiChuan Cusine and Hunan Cusine

Abstract

An abstract is a brief summary of a research article, thesis, review, conference proceeding or any in-depth analysis of a particular subject and is often used to help the reader quickly ascertain the paper's purpose. When used, an abstract always appears at the beginning of a manuscript or typescript, acting as the point-of-entry for any given academic paper or patent application. Abstracting and indexing services for various academic disciplines are aimed at compiling a body of literature for that particular subject.

Key Words: Key Word1; Key Word 2; Key Word 3; Key Word 4

目　　录

插图索引

附表索引

1 绪论

eBPF和WebAssembly两者作为沙盒技术，在作用的效果来看

1.1 动机

本文的动机在于。

1.2 贡献

本文在优化机制上所做出的贡献是。

1.3 文章结构

本文编排的结构组织如下：第 2 章介绍 eBPF 和 wasm 等相关工作；第 3 章介绍本文在两者优化机制上所做的贡献；第 4 章介绍本文构建实验的方式和测试结果；第 5 章将给出本文的讨论；第 6 章总结全文。

2 相关工作

2.1 eBPF

eBPF 是一项基于事件驱动的 Linux 内核拓展技术，支持动态地将用户编写的代码通过 eBPF 虚拟机加载到内核态上执行，而无需修改内核代码、插入内核模块或重启系统[1-3]。这一直接运行于内核态而无需通过切换用户态和内核态之间的能力，既减少了数据复制次数又扩展了内核的功能[2,4]； eBPF 的这种灵活性被用于加速特定的任务[5]，如网络包过滤[6-7]、网络流量监控[8]以及文件数据存储[9]和 FPGA 接收网络数据流[10]等。

2.1.1 eBPF 运行流程

eBPF 的运行流程如图所示。用户首先需要通过相应的 eBPF 框架来编写程序，经过如LLVM/clang的编译工具链编译为 eBPF 字节码[3]。 eBPF 字节码再通过 bpf() 加载到内核态[3]，并创建 eBPF 表来维护同一 eBPF 程序的执行状态和 eBPF 程序之间的状态分享[11]2。未经过运行的 eBPF 字节码会由 eBPF 静态验证器来实施检查[5,12]。检查内容包括穷举所有可能的执行情况，eBPF 程序的内存访问情况，程序能否结束，循环次数，无效指令等[3,13]。验证程序不会对内核造成影响后，最后经由 JIT 编译器编译为可执行的机器码[14]，并被加载到内核环境下的虚拟机中等待特定事件触发执行[15]。此外，eBPF 还可以依靠 Linux 提供的 Kprobe 技术将程序插入到几乎任何内核函数所在的位置[3]。

2.1.2 eBPF 字节码指令集

eBPF 程序在运行时只能操控 11 个 64 位寄存器，为 R0 至 R10，其功能分别是：

(1) R0：保存函数返回值和eBPF程序退出值；
(2) R1~R5：保存函数调用参数。如参数少于五个不会使用所有寄存器，超出五个则会以压栈的形式传参；

(3) R6~R9：保存调用函数的上下文；

(4) R10：只读的栈帧寄存器，用于访问栈。

同时，eBPF 程序能在一个固定大小的栈上执行四种基本运算：访存、算术、分支判断和调用[16]。

2.1.3 eBPF 存在的问题

Mohamed Mohamed Husain 等人撰写的文献[17] 汇总整理了截至 2023 年 8 月为止的 eBPF 的漏洞报告列表。在比重上，eBPF-Helper(eBPF内核帮助函数) 贡献了 16.7%的CVE； eBPF即时编译器贡献了 5.6%；eBPF核心贡献了 11.1%； eBPF验证器则占比最大，为 44.4%，其它则占 22.2%。

因内核稳定性需要，eBPF 规范不支持浮点运算[18]，也不允许死循环的存在[3]；还会使用 eBPF 验证器检查 eBPF 程序中是否包含潜在的不安全操作和不可接受的性能开销[19]。而 eBPF 的验证器的处理逻辑，截至 2024 年，就产生了 eBPF 本身超过半数的 CVE[20]。而且，当前的 eBPF 在验证 eBPF 程序上有所局限。eBPF 的调用栈最大深度只允许为 512[3]30，且经 JIT 编译后的指令数对于非特权程序最多不能超过 4096 条，特权程序不能超过 100 万条[21]； eBPF 的验证器存在假阳性误报[16,20]以及李浩和古金宇等人在文献 [19][2,5,6]指出的假阴性（即验证器自身的漏洞被利用于恶意程序的绕过）问题。

另外，对于字节码解释编译阶段，申文博等人的研究[22] 表明 eBPF 字节码缺少注入与劫持防护，其解释器在执行时不会检测所执行的程序，其解释流可以劫持并用于执行任意的字节码，可被利用绕过现有的内核代码完整性保护机制，实现内核任意代码注入攻击。

这些事实，使得开发者不得不从验证器的角度开发[19]或重构[5]代码，加重了开发者的负担。

2.2 Web Assembly 概况

wasm 是一种为增强Web浏览器性能和拓展性而面向 C，C++，rust 和 Go 等高级编程语言设计的紧凑的[23-25] 中间编译表示规范[26-30]（或称字节码）。 wasm 字节码通过如 Emscripten 和 rustc 等 wasm 前端编译器来生成[31]， wasm 字节码格式紧凑，空间占用小，也因此用于加速处理计算密集型任务，在边缘计算和物联网领域得到了应用[32]。因这种字节码并不能直接交由处理器译码执行，还需经过解释器或 JIT 或 AoT（Ahead of Time）等后端编译器转译为与当前运行环境下CPU指令集一致的机器码[32]6，wasm 有跨 OS 和处理器架构的可移植性和可扩展性[27-28,30,32-35]。

2.2.1 Web Assembly 内存模型

wasm 的内存模型定义了 wasm 程序的线性内存布局和内存对象。分配给 wasm 的内存页大小总是 64K 的倍数[25]。为了防止 wasm 程序越界访存，runtime需要对访存行为做边界检查。除此通过沙盒环境来隔离程序的安全机制以外，wasm 还通过自身的内存布局来约束程序的访存操作。这一线性内存并不存储全局变量或者局部变量，全局变量保存在一张固定大小的、名为全局索引空间(Global index space)的表上[25,35-36]；局部变

量存储于一个受到保护的调用栈上，调用栈保存有函数的返回地址。而在wasm中，线性内存既不可执行，也不可跳转，然而却不允许内存被标记为只读，所有线性内存上的数据均可写[28]。操作系统在执行程序时还会运用 ASLR 来随机编排栈、堆区、代码段的地址空间，但 wasm 只有线性内存并且内存编排方式一定[28]。

但由于栈上不存在类似于金丝雀的栈溢出检查标记，wasm 有栈溢出风险。

2.2.2 Web Assembly 数据类型

wasm 借鉴了安全语言设计，采用静态强类型系统，只提供四种基本数据类型[26,28,37]，分别为 32 位整数、32 位浮点数、64 位整数和 64 位浮点数。高级语言中的复杂类型，都在编译阶段被编译为这四类基本数据类型。

2.2.3 Web Assembly 运行环境

wasm 字节码以模块为单位，被限制于沙盒环境中加载和运行[38-40]。 wasm 的运行时可分为 4 个组件[32,41]：

(1) wasm 解释器；
(2) wasm 后端编译器；
(3) wasm 运行时环境；
(4) wasi。

而也有部分 wasm 解释器，如 wasmer① 和 wasmtime②，实现了后端编译器对wasm字节码的转译[32]。

wasm 运行时环境这确保了其运行时安全，并让其成功地于浏览器之外为非网页程序提供沙盒环境[34,42]。

2.2.4 Web Assembly 存在的问题

截至 2022 年，贺宁宇等人在文献[43]指出某些 Photoshop、工业 CAD 和 3D 游戏借助 wasm 而运行在浏览器上。

3 eBPF和Web Assembly的优化机制

针对第 2 节提出的问题，本文从前端编译器、沙盒运行环境以及解释器出发，结合了 wasm 和 ebpf 两者独有或共有的特点，对两者的安全机制做出了如下的优化与缓解措施。在保证原有性能不降低 5% 的前提下，成功地提升了两者的性能。

3.1 ebpf的优化策略

本文对于 ebpf 采用了页表标记的手段，是从……。

3.2 WebAssembly的优化策略

本文对于 wasm 的改进手段是……

①https://github.com/wasmerio/wasmer.

②https://github.com/bytecodealliance/wasmtime.

4 实验

4.1 实验环境

本文用于搭建、采集的计算机配置为Intel 11th Gen Intel(R) Core(TM) i5-1135G7@2.40 GHz，内存 16GB，操作系统为 Linux-6.5.1。

4.2 实验构建

管理学和人文社会学科的论文主体应包括对研究问题的论述及系统分析，比较研究，模型或方案设计，案例论证或实证分析，模型运行的结果分析或建议、改进措施等。

5 讨论

本文的提出的设计虽然能较好的完成预期目的，但日后仍应从...出发。

6 结语

本文就当前eBPF和WebAssembly目前存在的问题做了汇总研究，并探讨、搭建了相关实验以优化其中的安全问题。

参考文献

[1] H. Sun, et al. Finding Correctness Bugs in eBPF Verifier with Structured and Sanitized Program[C]. in: Proceedings of the Nineteenth European Conference on Computer Systems. Athens Greece: ACM, 2024: 689-703.

[2] Yi He, et al. Cross Container Attacks: The Bewildered eBPF on Clouds[C]. in: 32nd USENIX Security Symposium (USENIX Security 23). Anaheim, CA: USENIX Association, 2023: 5971-5988.

[3] L. Rice. Learning eBPF: Programming the Linux Kernel for Enhanced Observability, Networking, and Security[M]. First edition. Sebastopol, CA: O'Reilly Media, 2023.

[4] 张子君. Linux系统eBPF攻击建模及防护技术研究[D]. 浙江杭州: 浙江大学, 2024.

[5] Hao Sun, et al. Validating the eBPF Verifier via State Embedding[C]. in: 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). Santa Clara, CA: USENIX Association, 2024: 615-628.

[6] M. A. M. Vieira, et al. Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications[J]. ACM Comput. Surv., 2020, 53(1).

[7] The Tcpdump Group. Tcpdump, a powerful command-line packet analyzer[EB/OL]. 2024. https://www.tcpdump.org/.

[8] C. Cassagnes, et al. The rise of eBPF for non-intrusive performance monitoring[C]. in: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. 2020: 1-7.

[9] Yuhong Zhong, et al. XRP: In-Kernel Storage Functions with eBPF[C]. in: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). Carlsbad, CA: USENIX Association, 2022: 375-393.

[10] M. S. Brunella, et al. hXDP: Efficient Software Packet Processing on FPGA NICs[C]. in: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, 2020: 973-990.

[11] T. A. Benson, et al. NetEdit: An Orchestration Platform for eBPF Network Functions at Scale[C]. in: Proceedings of the ACM SIGCOMM 2024 Conference. Sydney NSW Australia: ACM, 2024: 721-734.

[12] Yusheng Zheng, et al. Bpftime: Userspace eBPF Runtime for Uprobe, Syscall and Kernel-User Interactions[Z]. 2023. arXiv: 2311.07923.

[13] Y. Zhou, et al. Electrode: Accelerating Distributed Protocols with eBPF[C]. in: 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). Boston, MA: USENIX Association, 2023: 1391-1407.

[14] 李有霖. 基于模糊测试的eBPF漏洞挖掘技术研究[D]. 四川成都: 电子科技大学, 2023.

[15] Mao Jinsong, et al. Merlin: Multi-tier Optimization of eBPF Code for Performance and Compactness[C]. in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. La Jolla CA USA: ACM, 2024: 639-653.

[16] E. Gershuni, et al. Simple and Precise Static Analysis of Untrusted Linux Kernel Extensions [C]. in: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. Phoenix AZ USA: ACM, 2019: 1069-1084.

[17] M. H. N. Mohamed, et al. Understanding the Security of Linux eBPF Subsystem[C]. in: Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems. Seoul Republic of Korea: ACM, 2023: 87-92.

[18] A. Osaki, et al. Dynamic Fixed-Point Values in eBPF: A Case for Fully in-Kernel Anomaly Detection[C]. in: Aintec '24: Proceedings of the Asian Internet Engineering Conference 2024. New York, NY, USA: Association for Computing Machinery, 2024: 46-54.

[19] 李浩, 等. 基于PKS硬件特性的eBPF内存隔离机制[J]. 软件学报, 2023, 34(12): 5921-5939.

[20] Peihua Zhang, et al. HIVE: A Hardware-assisted Isolated Execution Environment for eBPF on AArch64[C]. in: 33rd USENIX Security Symposium (USENIX Security 24). Philadelphia, PA: USENIX Association, 2024: 163-180.

[21] S. Y. Lim, et al. SafeBPF: Hardware-assisted Defense-in-depth for eBPF Kernel Extensions [Z]. 2024. arXiv: 2409.07508.

[22] Q. Liu, et al. Interp-flow Hijacking: Launching Non-control Data Attack via Hijacking eBPF Interpretation Flow[C]. in: Computer Security – ESORICS 2024: 29th European Symposium on Research in Computer Security, Bydgoszcz, Poland, September 16–20, 2024, Proceedings, Part III. Bydgoszcz, Poland: Springer-Verlag, 2024: 194-214.

[23] Yutian Yan, et al. Understanding the Performance of Webassembly Applications[C]. in: Proceedings of the 21st ACM Internet Measurement Conference. Virtual Event: ACM, 2021: 533-549.

[24] B. L. Titzer. A Fast In-Place Interpreter for WebAssembly[Z]. 2022. arXiv: 2205.01183.

[25] P. Daniel, et al. Discovering Vulnerabilities in WebAssembly with Code Property Graphs [C]. in: 2019.

[26] WebAssembly Community Group. Introduction of WebAssembly[Z]. https://webassembly.github.io/spec/core/intro/introduction.html. 2024.

[27] D. Lehmann, et al. Wasabi: A Framework for Dynamically Analyzing WebAssembly[C]. in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. Providence RI USA: ACM, 2019: 1045-1058.

[28] D. Lehmann, et al. Everything Old is New Again: Binary Security of WebAssembly[C]. in: 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, 2020: 217-234.

[29] S. Bhansali, et al. A First Look at Code Obfuscation for WebAssembly[C]. in: Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks. San Antonio TX USA: ACM, 2022: 140-145.

[30] M. Waseem, et al. Issues and Their Causes in WebAssembly Applications: An Empirical Study[Z]. 2024. arXiv: 2311.00646.

[31] A. Romano, et al. An Empirical Study of Bugs in WebAssembly Compilers[C]. in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). Melbourne, Australia: IEEE, 2021: 42-54.

[32] Yixuan Zhang, et al. Research on WebAssembly Runtimes: A Survey[Z]. 2024. arXiv: 2404.12621.

[33] J. Bosamiya, et al. Provably-Safe Multilingual Software Sandboxing using WebAssembly [C]. in: 31st USENIX Security Symposium (USENIX Security 22). Boston, MA: USENIX Association, 2022: 1975-1992.

[34] 庄骏杰, 等. WebAssembly安全研究综述[J]. 计算机研究与发展, 2024, 61(8): 1-27.

[35] P. P. Ray. An Overview of WebAssembly for IoT: Background, Tools, State-of-the-Art, Challenges, and Future Directions[J]. Future Internet, 2023, 15(8): 275.

[36] WebAssembly Community Group, et al. WebAssembly Specification[Z]. Specification. 2024.

[37] WebAssembly Community, et al. WebAssembly Specification[Z]. Specification. 2024.

[38] E. Johnson, et al. WaVe: A Verifiably Secure WebAssembly Sandboxing Runtime[C]. in: 2023 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE, 2023: 2940-2955.

[39] Yusheng Zheng, et al. Wasm-Bpf: Streamlining eBPF Deployment in Cloud Environments with WebAssembly[Z]. 2024. arXiv: 2408.04856.

[40] A. Haas, et al. Bringing the Web up to Speed with WebAssembly[C]. in: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. Barcelona Spain: ACM, 2017: 185-200.

[41] Yixuan Zhang, et al. Characterizing and Detecting WebAssembly Runtime Bugs[J]. ACM Transactions on Software Engineering and Methodology, 2024, 33(2): 1-29.

[42] S. Narayan, et al. Swivel: Hardening WebAssembly against Spectre[Z]. 2021.

[43] S. Cao, et al. WASMixer: Binary Obfuscation for WebAssembly[Z]. 2023. arXiv: 2308.03123.

[44] Yoann Ghigoff, et al. BMC: Accelerating Memcached Using Safe in-Kernel Caching and Pre-Stack Processing[C]. in: 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). USENIX Association, 2021: 487-501.

[45] J. Dejaeghere, et al. Comparing Security in eBPF and WebAssembly[C]. in: eBPF '23: Proceedings of the 1st Workshop on EBPF and Kernel Extensions. New York, NY, USA: Association for Computing Machinery, 2023: 35-41.

[46] Leaning Technologies. An Enterprise-Grade C++ Compiler For The Web[Z]. https://leaningtech.com/cheerp/. 2024.

[47] Emscripten Contributors. emscripten[Z]. https://emscripten.org/. 2015.

[48] B. Gbadamosi, et al. The eBPF Runtime in the Linux Kernel[Z]. 2024. arXiv: 2410.00026.

[49] I. Bastys, et al. SecWasm: Information Flow Control for WebAssembly[G]. in: G. Singh, et al. Static Analysis: vol. 13790. Cham: Springer Nature Switzerland, 2022: 74-103.

[50] isovalent. eBPF Docs[EB/OL]. 2024 [2024-12-25]. https://github.com/isovalent/ebpf-docs.

[51] A. Jangda, et al. Not so Fast: Analyzing the Performance of WebAssembly vs. Native Code [C]. in: 2019 USENIX Annual Technical Conference (USENIX ATC 19). Renton, WA: USENIX Association, 2019: 107-120.

[52] J. Jia, et al. Kernel Extension Verification Is Untenable[C]. in: Proceedings of the 19th Workshop on Hot Topics in Operating Systems. Providence RI USA: ACM, 2023: 150-157.

[53] H.-C. Kuo, et al. Verified Programs Can Party: Optimizing Kernel Extensions via Post-Verification Merging[C]. in: Proceedings of the Seventeenth European Conference on Computer Systems. Rennes France: ACM, 2022: 283-299.

[54] S. Y. Lim, et al. Unleashing Unprivileged eBPF Potential with Dynamic Sandboxing[C]. in: Proceedings of the 1st Workshop on eBPF and Kernel Extensions. New York NY USA: ACM, 2023: 42-48.

[55] S. Yuan, et al. End-to-End Mechanized Proof of a JIT-accelerated eBPF Virtual Machine for IoT[C]. in: A. Gurfinkel, et al. Computer Aided Verification. Cham: Springer Nature Switzerland, 2024: 325-347.

[56] R. Sahu, et al. Enabling BPF Runtime Policies for Better BPF Management[C]. in: Proceedings of the 1st Workshop on eBPF and Kernel Extensions. New York NY USA: ACM, 2023: 49-55.

[57] skanehira. Writting a Wasm Runtime in Rust[EB/OL]. 2024. https://github.com/skanehira/writing-a-wasm-runtime-in-rust/.

[58] Marcos A. M. Vieira, et al. Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications[J]. ACM Computing Surveys, 2021, 53(1): 1-36.

[59] 张秀宏. WebAssembly原理与核心技术[M]. 北京: 机械工业出版社, 2020.

[60] Wenlong Zheng, et al. WASMDYPA: Effectively Detecting WebAssembly Bugs via Dynamic Program Analysis[C]. in: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Rovaniemi, Finland: IEEE, 2024: 296-307.

[61] J. Ménétrey, et al. A Comprehensive Trusted Runtime for WebAssembly With Intel SGX [J]. IEEE Transactions on Dependable and Secure Computing, 2024, 21(4): 3562-3579.

致 谢

时光荏苒岁月如梭，转眼自己在HNU的本科生涯就此画上了句号。回想过往的憧憬，愈发发觉有涯的人生实在无法穷尽渴望研习讨论的知识，愈发感受到愿景落空后的久萦心间的遗憾。我深深地感谢我的导师xxx，他最初向我提出了这个问题，他凭着非凡的直觉，坚持认为代数方法将是富有成果的。 I owe a deep dept to my advisor Professor xxx, who originally suggested the topic to me and who, with remarkable intuition, insisted that the algebraic approach would be fruitful.

附录 A

公式推导如 (A.1) 式所示。

$$\begin{aligned}\int \frac{x+\sin x}{1+\cos x}\mathrm{d}x &= \int \frac{x}{2\cos^2\frac{x}{2}}+\tan\frac{x}{2}\mathrm{d}x \\ &= \int x\mathrm{d}\tan\frac{x}{2}+\tan\frac{x}{2}\mathrm{d}x \\ &= \frac{x}{2}\tan\frac{x}{2}-\int\tan\frac{x}{2}\mathrm{d}x+\int\tan\frac{x}{2}\mathrm{d}x \\ &= x\tan\frac{x}{2}+C\end{aligned} \tag{A.1}$$

所形成的交换图如图 A.1，图 A.2，图 A.3 和图 A.4 所示。

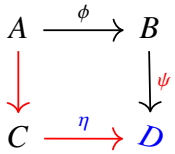
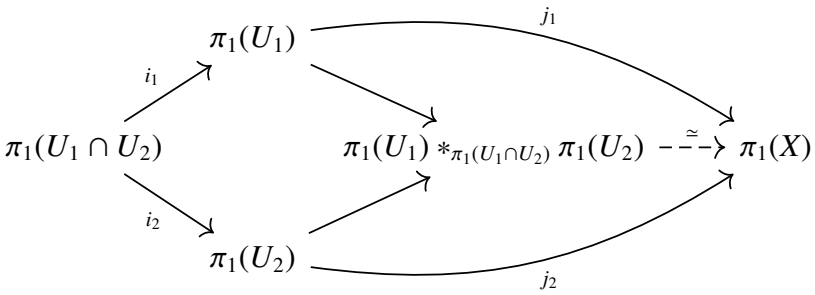
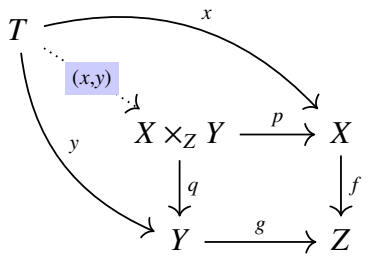


图 A.1　交换图例一



注：关系与图A.1类似，但不多.

图 A.2　交换图例二



注：关系与图A.1，图A.2类似，但不多.

图 A.3　交换图例三

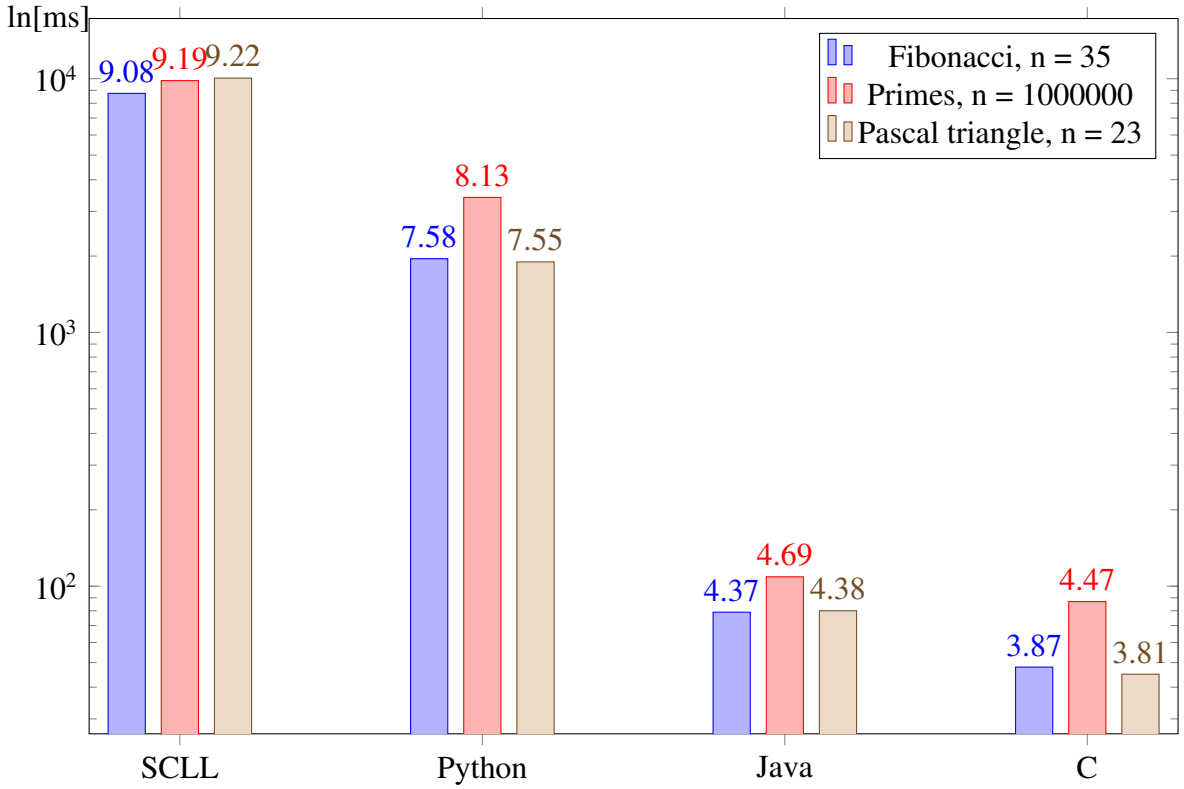


图 A.4　性能统计图表

附录 B

所用图片为图 B.1。

```python
#!/usr/bin/python
from bcc import BPF

program = r"""
int hello(void *ctx) {
    bpf_trace_printk("Hello World!");
    return 0;
}
"""

b = BPF(text=program)
syscall = b.get_syscall_fnname("execve")
b.attach_kprobe(event=syscall, fn_name="hello")

b.trace_print()
```

(a) 测试代码一

```
(module $a.out.wasm
  (type (;0;) (func))
  (type (;1;) (func (param i32)))
  (type (;2;) (func (param i32) (result i32)))
  (type (;3;) (func (result i32)))
  (func $__wasm_call_ctors (type 0)
    nop)
  (func $_emscripten_stack_restore (type 1) (param i32)
    local.get 0
    global.set $__stack_pointer)
  (func $_emscripten_stack_alloc (type 2) (param i32) (result i32)
    (local i32)
    global.get $__stack_pointer
    local.get 0
    i32.sub
    i32.const -16
    i32.and
    local.tee 1
    global.set $__stack_pointer
    local.get 1)
  (func $emscripten_stack_get_current (type 3) (result i32)
    global.get $__stack_pointer)
  (table (;0;) 1 1 funcref)
  (memory (;0;) 258 258)
  (global $__stack_pointer (mut i32) (i32.const 66560))
  (export "memory" (memory 0))
  (export "__wasm_call_ctors" (func $__wasm_call_ctors))
  (export "__indirect_function_table" (table 0))
  (export "_emscripten_stack_restore" (func $_emscripten_stack_restore))
  (export "_emscripten_stack_alloc" (func $_emscripten_stack_alloc))
  (export "emscripten_stack_get_current" (func $emscripten_stack_get_current)))
```

(b) 测试代码二

图 B.1　所用代码集

所用表格为表 B.1 和表 B.2。

表 B.1 数学字体对照表

mathrm		mathbf		mathit		mathsf		mathtt		mathcal		mathbb		mathfrak		mathscr	
Aa	Nn	**Aa**	**Nn**	*Aa*	*Nn*	Aa	Nn	Aa	Nn	$\mathcal{A}a$	$\mathcal{N}n$	$\mathbb{A}$	$\mathbb{N}$	$\mathfrak{A}\mathfrak{a}$	$\mathfrak{N}\mathfrak{n}$	$\mathscr{A}a$	$\mathscr{N}n$
Bb	Oo	**Bb**	**Oo**	*Bb*	*Oo*	Bb	Oo	Bb	Oo	$\mathcal{B}b$	$\mathcal{O}o$	$\mathbb{B}$	$\mathbb{O}$	$\mathfrak{B}\mathfrak{b}$	$\mathfrak{O}\mathfrak{o}$	$\mathscr{B}b$	$\mathscr{O}o$
Cc	Pp	**Cc**	**Pp**	*Cc*	*Pp*	Cc	Pp	Cc	Pp	$\mathcal{C}c$	$\mathcal{P}p$	$\mathbb{C}$	$\mathbb{P}$	$\mathfrak{C}\mathfrak{c}$	$\mathfrak{P}\mathfrak{p}$	$\mathscr{C}c$	$\mathscr{P}p$
Dd	Qq	**Dd**	**Qq**	*Dd*	*Qq*	Dd	Qq	Dd	Qq	$\mathcal{D}d$	$\mathcal{Q}q$	$\mathbb{D}$	$\mathbb{Q}$	$\mathfrak{D}\mathfrak{d}$	$\mathfrak{Q}\mathfrak{q}$	$\mathscr{D}d$	$\mathscr{Q}q$
Ee	Rr	**Ee**	**Rr**	*Ee*	*Rr*	Ee	Rr	Ee	Rr	$\mathcal{E}e$	$\mathcal{R}r$	$\mathbb{E}$	$\mathbb{R}$	$\mathfrak{E}\mathfrak{e}$	$\mathfrak{R}\mathfrak{r}$	$\mathscr{E}e$	$\mathscr{R}r$
Ff	Ss	**Ff**	**Ss**	*Ff*	*Ss*	Ff	Ss	Ff	Ss	$\mathcal{F}f$	$\mathcal{S}s$	$\mathbb{F}$	$\mathbb{S}$	$\mathfrak{F}\mathfrak{f}$	$\mathfrak{S}\mathfrak{s}$	$\mathscr{F}f$	$\mathscr{S}s$
Gg	Tt	**Gg**	**Tt**	*Gg*	*Tt*	Gg	Tt	Gg	Tt	$\mathcal{G}g$	$\mathcal{T}t$	$\mathbb{G}$	$\mathbb{T}$	$\mathfrak{G}\mathfrak{g}$	$\mathfrak{T}\mathfrak{t}$	$\mathscr{G}g$	$\mathscr{T}t$
Hh	Uu	**Hh**	**Uu**	*Hh*	*Uu*	Hh	Uu	Hh	Uu	$\mathcal{H}h$	$\mathcal{U}u$	$\mathbb{H}$	$\mathbb{U}$	$\mathfrak{H}\mathfrak{h}$	$\mathfrak{U}\mathfrak{u}$	$\mathscr{H}h$	$\mathscr{U}u$
Ii	Vv	**Ii**	**Vv**	*Ii*	*Vv*	Ii	Vv	Ii	Vv	$\mathcal{I}i$	$\mathcal{V}v$	$\mathbb{I}$	$\mathbb{V}$	$\mathfrak{I}\mathfrak{i}$	$\mathfrak{V}\mathfrak{v}$	$\mathscr{I}i$	$\mathscr{V}v$
Jj	Ww	**Jj**	**Ww**	*Jj*	*Ww*	Jj	Ww	Jj	Ww	$\mathcal{J}j$	$\mathcal{W}w$	$\mathbb{J}$	$\mathbb{W}$	$\mathfrak{J}\mathfrak{j}$	$\mathfrak{W}\mathfrak{w}$	$\mathscr{J}j$	$\mathscr{W}w$
Kk	Xx	**Kk**	**Xx**	*Kk*	*Xx*	Kk	Xx	Kk	Xx	$\mathcal{K}k$	$\mathcal{X}x$	$\mathbb{K}$	$\mathbb{X}$	$\mathfrak{K}\mathfrak{k}$	$\mathfrak{X}\mathfrak{x}$	$\mathscr{K}k$	$\mathscr{X}x$
Ll	Yy	**Ll**	**Yy**	*Ll*	*Yy*	Ll	Yy	Ll	Yy	$\mathcal{L}l$	$\mathcal{Y}y$	$\mathbb{L}$	$\mathbb{Y}$	$\mathfrak{L}\mathfrak{l}$	$\mathfrak{Y}\mathfrak{y}$	$\mathscr{L}\ell$	$\mathscr{Y}y$
Mm	Zz	**Mm**	**Zz**	*Mm*	*Zz*	Mm	Zz	Mm	Zz	$\mathcal{M}m$	$\mathcal{Z}z$	$\mathbb{M}$	$\mathbb{Z}$	$\mathfrak{M}\mathfrak{m}$	$\mathfrak{Z}\mathfrak{z}$	$\mathscr{M}m$	$\mathscr{Z}z$

注：此数据仅为示例，实际数据可能有所不同.

表 B.2 各组分 $\lg(B_i)$ 值

序号	T=1500K		T=2000K	
	组分	$\lg B_i$	组分	$\lg B_i$
1	O_2^+	5.26	HO_2	6.43
2	HO_2	5.26	O_2^+	6.42
3	H_2O^+	4.76	H_2O^+	6.18
4	N_2^+	3.97	H	6.12
5	H	3.54	H_2^+	6.04
6	OH	3.29	OH	5.91
7	CO^+	3.26	O	5.59
8	H_2^+	2.54	N_2^+	4.87
9	O	2.30	CO^+	3.98
10	H_2O_2	1.62	CO_2^+	3.76
11	CO_2^+	1.40	H_2O_2	3.09
12	HCO^*	-0.47	HCO^*	0.24
13	N^+	-4.85	N^+	-2.81
14	CH_2O^+	-6.91	CH_2O^*	-6.13
15	NO^+	-16.60	NO^+	-11.76

注：“+”表示重要成分，“*”表示冗余组分.