湖南大學

HUNAN UNIVERSITY

本科生毕业论文（设计）

论文(设计)题目: 川菜与湘菜在辣度上的对比研究

学生姓名: _____

学生学号: _____

专业班级: _____

学院名称: _____

指导老师: _____

2024 年 12 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：2024 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：2024 年　月　日

导师签名：　　日期：2024 年　月　日

川菜与湘菜在辣度上的对比研究

摘　　要

摘要是摘要。摘要应包括成果、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： eBPF; WebAssembly; 安全机制; 优化研究

The research on 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: eBPF; WebAssembly; security mechanism; optimizations

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

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

2 相关工作

2.1 eBPF

扩展的伯克利包过滤器（extended Berkeley Packet Filter，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或者插入内核模块[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提高了内核执行性能，又充分扩展了内核的功能[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]以及数据存储[9]和嵌入式开发[10]领域等。

2.1.1 eBPF运行流程

用户首先需要通过相应的eBPF框架来编写程序，通过[3] eBPF程序运行时，只能操控11个寄存器（为R0至R10），其功能分别为：

(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；
(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：加载、存储、算术运算和分支判断[5]。运行BPF程序前，会经过编译工具链（如clang）编译为BPF字节码，而后由BPF系统调用加载到内核态等待通过eBPF验证器的静态验证[11]。通过验证后，会经由BPF Just-In-Time(JIT)编译器编译为可执行的机器码并被加载到虚拟机中执行。

2.1.2 eBPF的限制

由于eBPF程序运行在内核态，加载到内核中的代码在编写不当时可能会导致内核崩溃。为确保从用户态加载进内核的代码不会引起这种情况，eBPF会使用自身的验证器检查 eBPF程序中的是否包含潜在的不安全操作和不可接受的性能开销。程序通过验证后，即时编译器 (Just-In-Time Compiler, JIT Compiler)被用来将eBPF指令转化为主机指令以加快执行速度[12]。

而仅eBPF的验证器的处理逻辑，就贡献了eBPF本身超过半数的CVE[13]。而且，当前的eBPF在性能上有所局限，eBPF的调用栈最大深度只允许为512，且经JIT 编译后的指令数最多不能超过100万条；eBPF的验证器存在假阳性[13]和假阴性问题[14]。假阳性即实现逻辑复杂但是不会影响其它程序正常运行的程序可能不会被验证通过，假阴性是指有的程序可能会

2.2 Web Assembly概况

Web Assembly（缩略为wasm）是一种为增强Web浏览器性能和拓展性而面向C，C++，rust 和Golang等高级编程语言设计的中间编译表示（或称字节码）[15-19]。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)编译器转译为对应机器可执行的机器码[20]6。因而，wasm有着跨OS和处理器架构的可扩

展性[16-17,19-23]。

作为弥补JavaScript性能瓶颈的补充技术之一[23]， wasm生成的字节码经转译后有接近原生机器码（即直接通过编译器编译得到的机器码）的执行性能[24-25]。即使如此，有研究[20, 26]指出，由于这种转译增加了总体汇编指令的个数、访存加载数、分支数以及额外的安全检查（如栈溢出和间接执行检查），造成L1缓存不命中率的上升，wasm的性能相比原生编译的可执行程序明显下降。此外，文献[27]也指出Spectre推测执行攻击能绕过wasm的内存隔离机制。

综合来看，wasm的安全性在软件和硬件上均有需要加固之处。

2.2.1 Web Assembly运行环境

在设计上 wasm 代码被限制在沙盒环境下运行[25,28]，受内存边界检查[20]，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[22,27,29]。

2.2.2 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。wasm在内存上的布局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；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。

线性内存在设计上不可执行，也不可跳转。然而wasm不允许内存被标记为只读，相反，所有线性内存上的数据均可写[17]。

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。

2.2.3 Web Assembly数据类型

Wasm充分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[15,17]，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类基本类型。

2.2.4 Web Assembly控制流完整性

wasm控制流完整性检查，包括直接函数调用检查、函数返回检查以及间接函数调用检查三个方面[22,30]。 wasm的函数并不能执行任意地址跳转。函数地址被记录在一张表上，调用函数需要通过取出表上的索引才能完成跳转。

间接函数跳转则是在函数需要在执行时动态确定（如C++的多态或函数指针赋值后调用的形式）才会被转译出来[30]。

3 eBPF和Web Assembly的优化机制

针对第2节提出的问题，本文从编译器方面、硬件特性方面出发，结合了wasm和ebpf各自的优缺点，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……

3.3 安全性和性能的折衷

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] Sun H, Xu Y, Liu J, et al. Finding Correctness Bugs in eBPF Verifier with Structured and Sanitized Program[C/OL]. in: Proceedings of the Nineteenth European Conference on Computer Systems. Athens Greece: ACM, 2024: 689-703 [2024-11-26]. DOI: 10.1145/3627703.3629562.

[2] He Y, Guo R, Xing Y, et al. Cross Container Attacks: The Bewildered eBPF on Clouds [C/OL]. in: 32nd USENIX Security Symposium (USENIX Security 23). Anaheim, CA: USENIX Association, 2023: 5971-5988. https://www.usenix.org/conference/usenixsecurity23/presentation/he.

[3] Rice L. 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] Sun H, Su Z. Validating the eBPF Verifier via State Embedding[C/OL]. in: 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). Santa Clara, CA: USENIX Association, 2024: 615-628. https://www.usenix.org/conference/osdi24/presentation/sun-hao.

[6] Vieira M A M, Castanho M S, Pacífico R D G, et al. Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications[J/OL]. ACM Comput. Surv., 2020, 53(1). https://doi.org/10.1145/3371038. DOI: 10.1145/3371038.

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

[8] Cassagnes C, Trestioreanu L, Joly C, et al. The rise of eBPF for non-intrusive performance monitoring[C/OL]. in: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. 2020: 1-7. DOI: 10.1109/NOMS47738.2020.9110434.

[9] Zhong Y, Li H, Wu Y J, et al. XRP: In-Kernel Storage Functions with eBPF[C/OL]. in: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). Carlsbad, CA: USENIX Association, 2022: 375-393. https://www.usenix.org/conference/osdi22/presentation/zhong.

[10] Brunella M S, Belocchi G, Bonola M, et al. hXDP: Efficient Software Packet Processing on FPGA NICs[C/OL]. in: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, 2020: 973-990. https://www.usenix.org/conference/osdi20/presentation/brunella.

[11] Zheng Y, Yu T, Yang Y, et al. Bpftime: Userspace eBPF Runtime for Uprobe, Syscall and Kernel-User Interactions[Z/OL]. 2023. arXiv: 2311.07923 [cs] [2024-12-01]. DOI: 10

.48550/arXiv.2311.07923.

[12] 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 [2024-11-27].

[13] Zhang P, Wu C, Meng X, et al. HIVE: A Hardware-assisted Isolated Execution Environment for eBPF on AArch64[C/OL]. in: 33rd USENIX Security Symposium (USENIX Security 24). Philadelphia, PA: USENIX Association, 2024: 163-180. https://www.usenix.org/conference/usenixsecurity24/presentation/zhang-peihua.

[14] 李浩, 古金宇, 夏虞斌, 等. 基于PKS硬件特性的eBPF内存隔离机制[J/OL]. 软件学报, 2023, 34(12): 5921-5939. DOI: 10.13328/j.cnki.jos.006762.

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

[16] Lehmann D, Pradel M. Wasabi: A Framework for Dynamically Analyzing WebAssembly [C/OL]. in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. Providence RI USA: ACM, 2019: 1045-1058 [2024-11-28]. DOI: 10.1145/3297858.3304068.

[17] Lehmann D, Kinder J, Pradel M. Everything Old is New Again: Binary Security of WebAssembly[C/OL]. in: 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, 2020: 217-234. https://www.usenix.org/conference/usenixsecurity20/presentation/lehmann.

[18] Bhansali S, Aris A, Acar A, et al. A First Look at Code Obfuscation for WebAssembly [C/OL]. in: Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks. San Antonio TX USA: ACM, 2022: 140-145 [2024-11-28]. DOI: 10.1145/3507657.3528560.

[19] Waseem M, Das T, Ahmad A, et al. Issues and Their Causes in WebAssembly Applications: An Empirical Study[Z/OL]. 2024. arXiv: 2311.00646 [cs] [2024-11-28]. DOI: 10.48550/arXiv.2311.00646.

[20] Zhang Y, Liu M, Wang H, et al. Research on WebAssembly Runtimes: A Survey[Z/OL]. 2024. arXiv: 2404.12621 [cs] [2024-12-11]. DOI: 10.48550/arXiv.2404.12621.

[21] Bosamiya J, Lim W S, Parno B. Provably-Safe Multilingual Software Sandboxing using WebAssembly[C/OL]. in: 31st USENIX Security Symposium (USENIX Security 22). Boston, MA: USENIX Association, 2022: 1975-1992. https://www.usenix.org/conference/usenixsecurity22/presentation/bosamiya.

[22] 庄骏杰, 胡霜, 华保健, 等. WebAssembly安全研究综述[J/OL]. 计算机研究与发展, 2024, 61(8): 1-27. https://crad.ict.ac.cn/cn/article/doi/10.7544/issn1000-1239.202330049. DOI: https://doi.org/10.7544/issn1000-1239.202330049.

[23] Ray P P. An Overview of WebAssembly for IoT: Background, Tools, State-of-the-Art, Challenges, and Future Directions[J/OL]. Future Internet, 2023, 15(8): 275 [2024-12-05]. DOI: 10.3390/fi15080275.

[24] Haas A, Rossberg A, Schuff D L, et al. Bringing the Web up to Speed with WebAssembly [C/OL]. in: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. Barcelona Spain: ACM, 2017: 185-200 [2024-11-28]. DOI: 10.1145/3062341.3062363.

[25] Johnson E, Laufer E, Zhao Z, et al. WaVe: A Verifiably Secure WebAssembly Sandboxing Runtime[C/OL]. in: 2023 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE, 2023: 2940-2955 [2024-11-26]. DOI: 10.1109/SP46215.2023.10179357.

[26] Jangda A, Powers B, Berger E D, 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.

[27] Narayan S, Disselkoen C, Moghimi D, et al. Swivel: Hardening WebAssembly against Spectre[Z/OL]. 2021 [2024-12-02]. DOI: 10.48550/ARXIV.2102.12730.

[28] Zheng Y, Yu T, Yang Y, et al. Wasm-Bpf: Streamlining eBPF Deployment in Cloud Environments with WebAssembly[Z/OL]. 2024. arXiv: 2408.04856 [cs] [2024-11-28]. DOI: 10.48550/arXiv.2408.04856.

[29] Wen E, Weber G. Wasmachine: Bring IoT up to Speed with A WebAssembly OS[C/OL]. in: 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). 2020: 1-4. DOI: 10.1109/PerComWorkshops48775.2020.9156135.

[30] Daniel P, Lopes R, Santos N, et al. Discovering Vulnerabilities in WebAssembly with Code Property Graphs[C]. in: 2019.

[31] Haas A, Rossberg A, Schuff D L, et al. Bringing the web up to speed with WebAssembly [C/OL]. in: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. Barcelona Spain: ACM, 2017: 185-200 [2024-11-28]. https://dl.acm.org/doi/10.1145/3062341.3062363. DOI: 10.1145/3062341.3062363.

[32] Skanehira. Writting a Wasm Runtime in Rust[Z]. https://github.com/skanehira/writing-a-wasm-runtime-in-rust/. 2023.

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。

附录 A

公式推导如 (1) 式所示。

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

所形成的交换图如图 1，图 2 和图 3 所示。

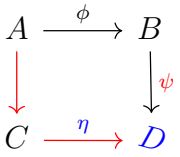
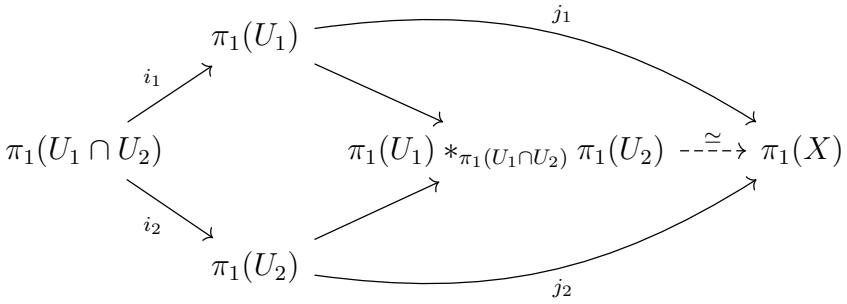
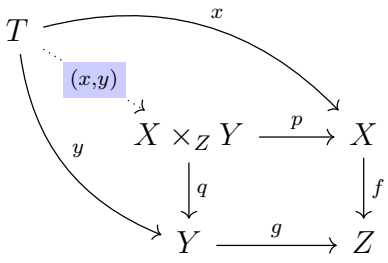


图 1　交换图例一



注：关系与图1类似，但不多.

图 2　交换图例二



注：关系与图1，图2类似，但不多.

图 3　交换图例三

附录 B

所用图片为图 4。

```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 4　所用代码集

所用表格为表 1 和表 2。

表 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	$\mathcal{L}l$	$\mathcal{Y}y$	$\mathbb{L}$	$\mathbb{Y}$	$\mathfrak{L}\mathfrak{l}$	$\mathfrak{Y}\mathfrak{y}$	$\mathscr{L}l$	$\mathscr{Y}y$
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 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.