

ReL4：高性能异步微内核设计与实现

廖东海

**** 年 * 月

ReL4:

高性能异步微内核设计与实现

廖东海

北京理工大学

中图分类号： TQ028.1

UDC分类号： 540

- ★ 特别类型
- ☐ 交叉研究方向
- ☐ 政府项目留学生

ReL4： 高性能异步微内核设计与实现

作者姓名廖东海

学院名称计算机学院

指导教师陆慧梅

答辩委员会主席***

申请学位工学硕士

学科 / 类别计算机科学与技术

学位授予单位北京理工大学

论文答辩日期**** 年 * 月

ReL4: Design and Implementation of High-performance Asynchronous Microkernel

Candidate Name:	<u>Liao Donghai</u>
School or Department:	<u>****</u>
Faculty Mentor:	<u>Lu Huimei</u>
Chair, Thesis Committee:	<u>***</u>
Degree Applied:	<u>****</u>
Major:	<u>****</u>
Degree by:	<u>Beijing Institute of Technology</u>
The Date of Defence:	<u>*, ****</u>

研究成果声明

本人郑重声明：所提交的学位论文是我本人在指导教师的指导下独立完成的研究成果。文中所撰写内容符合以下学术规范（请勾选）：

☐ 论文综述遵循“适当引用”的规范，全部引用的内容不超过50%。

☐ 论文中的研究数据及结果不存在篡改、剽窃、抄袭、伪造等学术不端行为，并愿意承担因学术不端行为所带来的一切后果和法律责任。

☐ 文中依法引用他人的成果，均已做出明确标注或得到许可。

☐ 论文内容未包含法律意义上已属于他人的任何形式的研究成果，也不包含本人已用于其他学位申请的论文或成果。

☐ 与本人一同工作的合作者对此研究工作所做的任何贡献均已在学位论文中作了明确的说明并表示了谢意。

特此声明。

签 名：

日期：

关于学位论文使用权的说明

本人完全了解北京理工大学有关保管、使用学位论文的规定，其中包括：

① 学校有权保管、并向有关部门送交学位论文的原件与复印件；

② 学校可以采用影印、缩印或其它复制手段复制并保存学位论文；

③ 学校可允许学位论文被查阅或借阅；

④ 学校可以学术交流为目的，复制赠送和交换学位论文；

⑤ 学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

签 名：

日期：

导师签名：

日期：

摘要

微内核将大部分系统服务运行在用户空间，相比于宏内核有更好的稳定性、扩展性和内核安全性，在微内核系统中，应用程序通过进程间通信 (IPC) 而非系统调用来请求服务，频繁的 IPC 造成的特权级切换将产生巨大开销，成为了系统的性能瓶颈。以 seL4 为代表的现代微内核将同步 IPC 作为主要通信手段，并以异步通知机制作为辅助手段来提升系统并发度，这些机制在一定程度上减少了 IPC 的开销，提升了系统性能，然而它们在设计上仍有三点不足：1) 在支持同步 IPC 的情况下冗余地支持了异步通知机制，这违反了内核最小化原则；2) 通知机制依赖内核的转发，会造成大量的特权级切换；3) 系统调用和同步 IPC 会导致无关请求顺序执行，无法充分利用硬件资源。

针对以上三点缺陷，本文设计并实现了 ReL4——一套异步化的高性能微内核架构，其主要设计理念是将异步通知机制作为内核支持的唯一通信手段，在用户态通过共享缓冲区实现 IPC 的数据传递，并通过通知机制进行同步，保证了内核最小化原则；同时 ReL4 基于用户态中断，在兼容 seL4 接口的基础上，设计了无需内核转发的 U-notification 机制，避免了通知机造成的大量特权级切换；最后，ReL4 通过异步运行时来实现异步 IPC 和异步系统调用，避免了无关 IPC 和系统调用请求的顺序执行，在充分利用硬件资源，提升系统并发度的同时，提升系统易用性。

本文在 FPGA 上实现了 ReL4 的原型系统，将 U-notification、异步 IPC 和异步系统调用分别与 seL4 进行了对比测试，并评估了真实的 TCP Server 在 ReL4 的性能表现。测试结果表明，相比于 seL4，ReL4 能够大幅减少系统中特权级切换的次数，在低并发场景下有着接近的性能，在高并发场景下拥有更卓越的表现，从而证明了 ReL4 架构的可行性和优越性。

关键词：微内核；异步；进程间通信；用户态中断

Abstract

The microkernel runs most system services in the user space. Compared with the monolithic kernel, it has better stability, extensibility, and kernel security. In a microkernel system, applications request services through inter-process communication (IPC) instead of system calls. The frequent privilege level switches caused by IPC generate huge overheads, which become a performance bottleneck of the system. Modern microkernels represented by seL4 use synchronous IPC as the main communication means and adopt an asynchronous notification mechanism as an auxiliary means to improve the system concurrency. These mechanisms reduce the IPC overhead to a certain extent and enhance the system performance. However, there are still three design deficiencies: (1) The asynchronous notification mechanism is redundantly supported while synchronous IPC is supported, which violates the principle of kernel minimization; (2) The notification mechanism relies on the forwarding of the kernel, resulting in a large number of privilege level switches; (3) System calls and synchronous IPC will cause irrelevant requests to be executed sequentially, failing to fully utilize hardware resources.

Aiming at the above three defects, this paper designs and implements ReL4, a set of asynchronous high-performance microkernel architecture. Its main design concept is to use the asynchronous notification mechanism as the only communication means supported by the kernel. In the user space, the data transfer of IPC is realized through a shared buffer, and synchronization is carried out through the notification mechanism, ensuring the principle of kernel minimization. At the same time, based on user-space interrupts and on the basis of being compatible with the seL4 interface, ReL4 designs the U-notification mechanism that does not require kernel forwarding, avoiding a large number of privilege level switches caused by the notification mechanism. Finally, ReL4 implements asynchronous IPC and asynchronous system calls through an asynchronous runtime, avoiding the sequential execution of irrelevant IPC and system call requests. While fully utilizing hardware resources and improving system concurrency, it also enhances the usability of the system.

This paper implements a prototype system of ReL4 on an FPGA. The U-notification, asynchronous IPC, and asynchronous system calls are respectively compared and tested with

seL4, and the performance of a real TCP Server on ReL4 is evaluated. The test results show that compared with seL4, ReL4 can significantly reduce the number of privilege level switches in the system, has a similar performance in low-concurrency scenarios, and has more excellent performance in high-concurrency scenarios, thus proving the feasibility and superiority of the ReL4 architecture.

Key Words: microkernel; asynchronous; inter-process communication; user-mode interrupt

目录

第 1 章 绪论	1
1.1 课题研究的背景和意义	1
1.2 国内外研究现状及发展趋势	2
1.2.1 微内核 IPC 的发展现状	2
1.2.2 特权级切换	3
1.3 主要研究内容	5
参考文献	6
攻读学位期间发表论文与研究成果清单	8
致谢	9
作者简介	10

插图

表格

表 1.1	seL4 一次 IPC 中各操作的开销占比	3
表 1.2	国内外研究现状汇总	4

主要符号对照表

BIT	北京理工大学的英文缩写
\LaTeX	一个很棒的排版系统
$\LaTeX 2_{\epsilon}$	一个很棒的排版系统的最新稳定版
$X_{\text{Y}}\TeX$	\LaTeX 的好兄弟，事实上他有很多个兄弟，但是这个兄弟对各种语言的支持能力都很强
ctex	成套的中文 \LaTeX 解决方案，由一帮天才们开发
H_2SO_4	硫酸
$e^{\pi i} + 1 = 0$	一个集自然界五大常数一体的炫酷方程
$2\text{H}_2 + \text{O}_2 \longrightarrow 2\text{H}_2\text{O}$	一个昂贵的生成生命之源的方程式

第 1 章 绪论

1.1 课题研究的背景和意义

随着科技的发展，微内核广泛应用于工控系统、嵌入式系统等领域。相比于宏内核，微内核将内存管理、设备驱动、文件系统等与核心功能分离，运行在用户空间，这种隔离机制使得单个服务的故障不会直接影响到内核和其他服务，从而提升了系统的整体稳定性^[1]；微内核通过精简核心功能，减少攻击面，从而提升内核安全性；此外，模块化的设计使得系统更易于维护和升级。在微内核系统中，应用程序通过进程间通信 (IPC) 而非系统调用来请求服务，这或许能够满足早期性能不敏感的软件需求，然而在对软件性能有着更高要求的今天，频繁 IPC 造成的特权级切换将产生巨大开销，成为系统的性能瓶颈^[2]。

30 年前 Liedtke 提出的 L4^[3] 重新设计了微内核系统，通过组合系统调用、快速路径、消息寄存器等优化手段，从硬件层到软件层对 IPC 进行了系统性优化，证明了微内核的 IPC 也可以很快，之后以 seL4^[4] 为代表的现代微内核的 IPC 框架也基本延续了 L4 的设计理念，以同步 IPC 作为主要的通信方式。然而同步 IPC 迫使单线程中上下文无关的请求以顺序的形式执行，系统只能通过多线程实现并发，而为了更好地利用硬件资源，现代微内核大多引入异步的通知机制来简化并发程序设计，提升多核的利用率，这违反了微内核的最小化设计原则，增加了内核的复杂性。

而随着软件复杂性的提升，用户希望系统级软件如数据库管理系统、网络服务器等能够快速处理大量系统调用和 IPC^[5]，而微内核将操作系统的大部分服务（如网络协议栈、文件系统等）移到用户态，从而使得 IPC 数量和频率激增，特权级切换成为性能瓶颈。此外，新出现的硬件漏洞如 Meltdown^[6] 和 Spectre^[7] 漏洞促使内核使用 KPTI 补丁^[8] 来分离用户程序和内核的页表，进一步增加了陷入内核的开销。最后，现代微内核的外设驱动往往存在于用户态，外设中断被转化为异步通知，需要用户态驱动主动陷入内核来进行接收，这在一定程度上成为了外设驱动的性能瓶颈^[9]。综上所述，由 IPC 和通知机制引起的特权级切换已经成为制约系统性能的主要因素。

本文提出 ReL4，一个用 Rust 编写的高性能异步微内核，它将同步 IPC 从内核中移除，基于用户态中断技术设计了无需陷入内核的 U-notification 机制，在兼容 capability 机制的基础上改造微内核的通知机制，并利用改造后的 U-notification 和异步化编程

设计和实现了一套绕过内核的异步 IPC 和异步系统调用框架。ReL4 在设计理念上将内核最小化原则贯彻得更加彻底，并通过软硬件协同的方式进一步提升微内核的 IPC 性能，为下一代微内核的发展指出一个可能的方向。

1.2 国内外研究现状及发展趋势

1.2.1 微内核 IPC 的发展现状

现代微内核的大部分 IPC 优化始于 Liedtke 提出的 L4，由于之前的微内核 IPC 存在性能瓶颈，L4 从硬件优化、系统架构、软件接口的各个方面对 IPC 进行了重新设计。其中的优化角度可以简单划分为内核路径优化和上下文切换优化。

对于内核路径优化，L4 通过物理消息寄存器来传递短消息，从而避免了内存拷贝，然而随着访存速度的加快，消息寄存器的零拷贝优化带来的收益逐渐减弱，使用物理寄存器导致的平台依赖和编译器优化失效反而限制了系统的性能^[10]，因此物理的消息寄存器逐渐被现代微内核以虚拟消息寄存器代替；此外，L4 使用临时映射的形式来进行长消息的传递，避免多余的内存拷贝，但却在内核中引入了缺页异常的可能性，增加了内核行为的复杂性^[10]，现代微内核一般放弃了这个优化；针对常用且普遍的 IPC 场景，L4 设计了专门的快速路径，避免了复杂繁琐的参数解析和任务调度，然而该优化手段对消息长度、任务优先级有着严格的限制，也无法对多 CPU 核心进行支持。

对于特权级的切换优化，L4 使用的物理消息寄存器在一定程度上减少了上下文切换的开销，但其副作用超过了优化收益导致其被虚拟消息寄存器代替^[10]；同时 L4 敏锐地观察到大部分 IPC 通信遵循 C/S 模型，因此通过组合系统调用的形式，将 Send + Reply 组合为 Call，将 Reply + Recv 组合成 ReplyRecv，从而减少了特权级切换的频率，该优化至今作为现代微内核的重要优化手段，但仍然无法避免特权级的切换；此外，L4 通过通过 ASID 机制，在快表项中维护地址空间标识符，减少快表冲刷的频率，缓解了快表污染的问题，然而依然无法避免特权级切换带来的快表污染和缓存失效。

除此之外，L4 仅支持同步 IPC，对于多核架构，同步 IPC 会导致服务调用被顺序执行，导致资源浪费。其次，同步 IPC 强制用户态以多线程的形式处理并发请求，导致了线程同步的复杂性。现代微内核在内核中引入异步通知机制，简化并发编程模型，却使得内核功能冗余，违反了内核最小化原则。

操作	占比
保存和恢复上下文	14.5%
地址空间切换	59.4%
fast - path 检查	20.1%
消息拷贝	1.5%
其他	4.5%

表 1.1 seL4 一次 IPC 中各操作的开销占比

总而言之，现代微内核在单核环境下的 IPC 内核路径上的优化已经较为完善，在最理想的情况下仅需要两次特权级切换，然而对多核环境下，由于需要核间中断，IPC 无法进入快速路径，导致多核下的 IPC 内核路径依旧冗长。而现代微内核在特权级切换的优化方面仍然停留缓解的层面上，导致特权级切换会成为 IPC 的性能瓶颈。

1.2.2 特权级切换

特权级的开销主要分为两部分，一部分是直接开销，包括了保存上下文带来的额外指令开销，另一部分是间接开销，地址空间切换所引起的缓存污染会导致 CPU 执行效率降低。表1.1展示了在 FPGA 上部署的 seL4 一次 Call IPC 操作所带来的开销占比，大部分的开销都花费在地址空间切换上，其次是上下文的切换和 fast-path 检查，如果 fast-path 检查失败，将会进入 slow-path 进行更加冗长的解码和调度流程，进一步降低 IPC 性能。

本文聚焦现代微内核架构设计中的特权级切换开销，旨在设计一种新型的 IPC 架构，减少甚至消除 IPC 和系统调用中的特权级切换，先前已经有大量的工作从硬件的角度致力于减少特权级切换开销。

从硬件出发的角度出发，大多数工作通过设计特殊的硬件或者特殊的指令来绕过内核实现 IPC。如 SkyBridge^[11] 允许进程在 IPC 中直接切换到目标进程的虚拟地址空间并调用目标函数，它通过精心设计一个 Root Kernel 提供虚拟化的功能，通过 VMFUNC 地址空间的直接切换，并通过其他一系列软件手段来保证安全性，但这种方案仅适用于虚拟化环境中。XPC^[12] 则直接使用硬件来提供一个无需经过内核的同步功能调用，并提供一种新的空间映射机制用于调用者与被调用者之间的零拷贝消息传递，然而该方案没有相应的硬件标准，也没有一款通用的处理器对其进行支持。这些方法都基于特殊的环境或者没有标准化的硬件来实现，适用范围有限。

从软件出发的角度出发，相关工作主要分为两类：第一类方法通过将用户态和内

优化方法	详细分类	实例	缺点
减少内核路径	临时地址映射	[3]	上下文切换开销已经成为性能瓶颈
	快速路径	[3, 4, 24, 25, 26]	
	消息寄存器		
减少上下文切换开销	消息寄存器	[3, 4, 24, 25, 26]	无法从根本上消除切换开销
	组合系统调用		
	ASID 机制	[4]	与微内核设计理念相悖，无法有效地实施到微内核中
	统一地址空间	[13, 14, 15, 16, 17, 18]	
	批量系统调用	[19]	
硬件优化	虚拟化指令	[11]	仅适用于虚拟化环境
	直接硬件辅助	[12]	没有硬件标准，没有通用硬件的支持
		用户态中断	—

表 1.2 国内外研究现状汇总

核态的功能扁平化来减少内核与用户态的切换开销，如 **unikernel**^[13-15] 将所有用户态代码都映射到内核态执行，**Userspace Bypass**^[16] 通过动态二进制分析将两个系统调用之间的用户态代码移入内核态执行，从而减少陷入内核的次数，**kernel bypass**^[17,18] 则通过将硬件驱动（传统内核的功能）移入用户态，从而减少上下文的切换。这些方法要么需要特殊的硬件支持，要么难以与微内核的设计理念兼容。第二类方法则是允许用户空间对多个系统调用请求排队，并通过一次提交将他们注册给内核，如 **FlexSC**^[19] 通过在用户态设计一个用户态线程的运行，将用户态线程发起的系统调用自动收集，然后陷入内核态进行批量执行。该方法虽然可以有效的减少陷入内核的次数，但如何设置提交的时机难以把握，过短的提交间隔将导致切换次数增加，过长的提交间隔则会导致 CPU 空转。

虽然现有工作难以广泛且有效地应用到微内核中，但他们的思路值得借鉴，他们的缺陷驱使研究者去寻求更好的方案。在硬件方面，一种新型的硬件技术方案——用户态中断^[20,21] 逐渐被各个硬件平台（x86，RISC-V）采纳，它通过在 CPU 中新增中断代理机制和用户态中断的状态寄存器，当中断代理机制检测到状态寄存器发生变化时，会将中断以硬件转发的形式传递给用户态程序，从而绕过内核。该硬件方案已经在 **Sapphire Rapids x86** 处理器上和 **RISCV** 的 **N** 扩展中有了一定的支持，适用范围更加广泛。而在软件方面，异步被广泛用于请求合并和开销均摊，传统类 **Unix** 系统提供的类似 **select IO** 多路复用接口相对简陋，迫使用户态代码采用事件分发的编程范式来处理异步事件，代码相对复杂，可读性较弱。而新兴的 **Rust**^[22,23] 语言对异步有着良好的支持，其零成本抽象的设计也让它作为系统编程语言有着强大的竞争力。使用 **Rust** 进行内核和用户态基础库的开发，可以更好地对异步接口进行抽象，改善接口的易用性和代码的可读性。

1.3 主要研究内容

本文以免费开源的 seL4 的 IPC 系统为主要研究对象，介绍了 seL4 中 IPC 的设计细节以及相关设计缺陷，针对 seL4 的相关缺陷，基于用户态中断，给出了高性能异步微内核的整体设计方案，包含了无需内核转发的通知机制 U-notification、异步 IPC 与异步系统调用，并基于 Rust 语言在 RISC-V 平台实现了 ReL4 原型系统。在 FPGA 上将 U-notification、异步 IPC 和异步系统调用分别与 seL4 进行了对比测试，并评估了 ReL4 在真实的 TCP Server Benchmark 上的性能表现。论文内容结构安排如下：

第 1 章，绪论部分，阐述了本课题的研究背景及意义，国内外研究现状和发展趋势，最后概述了本课题的研究内容和论文的组织结构。

第 2 章，介绍了 seL4 系统的发展历史和主要特征，并对 seL4 中的基本概念以及 IPC 设计进行了系统性阐述。

第 3 章，系统设计部分。首先介绍了 ReL4 与 seL4 的关系，ReL4 的特点以及设计目标，然后介绍了 ReL4 如何对系统中各个部分的通知机制进行支持和优化，最后介绍了用于保证兼容性和提升易用性的异步运行时的组成部分和设计思路。

第 4 章，系统实现部分。主要介绍了新增的系统调用，以及基于 U-notification 的异步运行时如何实现异步 IPC 和异步系统调用。

第 5 章，兼容性讨论。主要讨论了 ReL4 在通知机制与 IPC 机制中对 seL4 的兼容程度。

第 6 章，系统评估部分。首先介绍了 ReL4 的测试方案，然后针对 ReL4 的各个机制进行了性能评估和分析，最后测试了 ReL4 在真实的 TCP Server Benchmark 上的性能表现。

第 7 章，总结与展望，对本次设计和论文工作进行了总结，并提出了进一步的研究方向与对未来工作的展望。

参考文献

- [1] Abhilash T. Microkernel vs monolithic kernel design trade - offs[J]. International Journal For Multidisciplinary Research, 2024, 6(2): 1 – 5.
- [2] Liedtke J. Toward real microkernels[J]. Communications of the ACM, 1996, 39(9): 70–77.
- [3] Liedtke J. Improving ipc by kernel design[C]. Proceedings of the fourteenth ACM symposium on Operating systems principles. [S.l.: s.n.], 1993: 175–188.
- [4] Klein G, Elphinstone K, Heiser G, et al. sel4: Formal verification of an os kernel[C]. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. [S.l.: s.n.], 2009: 207–220.
- [5] Caruso J. 1 million iops demonstrated[M]. [S.l.: s.n.], 2021.
- [6] Lipp M, Schwarz M, Gruss D, et al. Meltdown: Reading kernel memory from user space[J]. Communications of the ACM, 2020, 63(6): 46–56.
- [7] Kocher P, Horn J, Fogh A, et al. Spectre attacks: Exploiting speculative execution[J]. Communications of the ACM, 2020, 63(7): 93–101.
- [8] kernel development community T. Page table isolation(pti)[M]. [S.l.: s.n.], 2021.
- [9] Blackham B, Shi Y, Heiser G. Improving interrupt response time in a verifiable protected microkernel [C]. Proceedings of the 7th ACM european conference on Computer Systems. [S.l.: s.n.], 2012: 323–336.
- [10] Heiser G, Elphinstone K. L4 microkernels: The lessons from 20 years of research and deployment [J]. ACM Transactions on Computer Systems (TOCS), 2016, 34(1): 1–29.
- [11] Mi Z, Li D, Yang Z, et al. Skybridge: Fast and secure inter-process communication for microkernels [C]. Proceedings of the Fourteenth EuroSys Conference 2019. [S.l.: s.n.], 2019: 1–15.
- [12] Du D, Hua Z, Xia Y, et al. Xpc: architectural support for secure and efficient cross process call [C]. Proceedings of the 46th International Symposium on Computer Architecture. [S.l.: s.n.], 2019: 671–684.
- [13] Kuo H C, Williams D, Koller R, et al. A linux in unikernel clothing[C]. Proceedings of the Fifteenth European Conference on Computer Systems. [S.l.: s.n.], 2020: 1–15.
- [14] Olivier P, Chiba D, Lankes S, et al. A binary-compatible unikernel[C]. Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. [S.l.: s.n.], 2019: 59–73.
- [15] Yu K, Zhang C, Zhao Y. Web service appliance based on unikernel[C]. 2017 IEEE 37th International

- Conference on Distributed Computing Systems Workshops (ICDCSW). [S.l.]: IEEE, 2017: 280–282.
- [16] Zhou Z, Bi Y, Wan J, et al. Userspace bypass: Accelerating syscall-intensive applications[C]. 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). [S.l.: s.n.], 2023: 33–49.
- [17] Jeong E, Wood S, Jamshed M, et al. {mTCP}: a highly scalable user-level {TCP} stack for multicore systems[C]. 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). [S.l.: s.n.], 2014: 489–502.
- [18] Yang Z, Harris J R, Walker B, et al. Spdk: A development kit to build high performance storage applications[C]. 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). [S.l.]: IEEE, 2017: 154–161.
- [19] Soares L, Stumm M. {FlexSC}: Flexible system call scheduling with {Exception-Less} system calls [C]. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10). [S.l.: s.n.], 2010.
- [20] Nassif N, Munch A O, Molnar C L, et al. Sapphire rapids: The next-generation intel xeon scalable processor[C]. 2022 IEEE International Solid-State Circuits Conference (ISSCC): volume 65. [S.l.]: IEEE, 2022: 44–46.
- [21] The risc-v instruction set manual. Volume ii: Privileged architecture, document version 1.12[M]. [S.l.: s.n.], 2020.
- [22] Levy A, Andersen M P, Campbell B, et al. Ownership is theft: Experiences building an embedded os in rust[C]. Proceedings of the 8th Workshop on Programming Languages and Operating Systems. [S.l.: s.n.], 2015: 21–26.
- [23] Balasubramanian A, Baranowski M S, Burtsev A, et al. System programming in rust: Beyond safety [C]. Proceedings of the 16th workshop on hot topics in operating systems. [S.l.: s.n.], 2017: 156–161.

攻读学位期间发表论文与研究成果清单

(二) 发表的学术论文

- [1] XXX, XXX. Static Oxidation Model of Al-Mg/C Dissipation Thermal Protection Materials[J]. Rare Metal Materials and Engineering, 2010, 39(Suppl. 1): 520-524. (SCI收录, IDS 号为 669JS, IF=0.16)
- [2] XXX, XXX. 精密超声振动切削单晶铜的计算机仿真研究 [J]. 系统仿真学报, 2007, 19 (4) : 738-741, 753. (EI 收录号: 20071310514841)
- [3] XXX, XXX. 局部多孔质气体静压轴向轴承静态特性的数值求解 [J]. 摩擦学学报, 2007 (1) : 68-72. (EI 收录号: 20071510544816)
- [4] XXX, XXX. 硬脆光学晶体材料超精密切削理论研究综述 [J]. 机械工程学报, 2003, 39 (8) : 15-22. (EI 收录号: 2004088028875)
- [5] XXX, XXX. 基于遗传算法的超精密切削加工表面粗糙度预测模型的参数辨识以及切削参数优化 [J]. 机械工程学报, 2005, 41 (11): 158-162. (EI 收录号: 2006039650087)
- [6] XXX, XXX. Discrete Sliding Mode Control with Fuzzy Adaptive Reaching Law on 6-PEES Parallel Robot[C]. Intelligent System Design and Applications, Jinan, 2006: 649-652. (EI 收录号: 20073210746529)

(二) 申请及已获得的专利 (无专利时此项不必列出)

- [1] XXX, XXX. 一种温热外敷药制备方案: 中国, 88105607.3[P]. 1989-07-26.

(三) 参与的科研项目及获奖情况

- [1] XXX, XXX. XX 气体静压轴承技术研究, XX 省自然科学基金项目. 课题编号: XXXX.
- [2] XXX, XXX. XX 静载下预应力混凝土房屋结构设计统一理论. 黑龙江省科学技术二等奖, 2007.

致谢

本论文的工作是在导师.....。

作者简介

本人…。