高可用的磁盘阵列 Cache 的设计和实现*

Design and Implementation of a High Availability RAID Cache

熊建刚,冯　丹

XIONG Jian-gang, FENG Dan

(华中科技大学计算机科学与技术学院,湖北 武汉 430074)

(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

摘　要:本文介绍了采用非易失写缓存解决 RAID5 小写问题的设计和实现,该方法通过对 Cache 数据结构的精心组织和采用事务机制对 Cache 进行修改。我们用单片 NVRAM 实现了高可用的磁盘阵列 Cache。

Abstract: This paper introduces the design and implementation of the problem of small write in a RAID-5 system using a non-volatile write cache. This solution implements a high availability RAID Cache, through the elaborate organization of the Cache structure and using a transaction mechanism to modify the Cache. We have implemented a high availability RAID Cache with the single-chip NVRAM.

关键词:Cache;事务机制;NVRAM;磁盘阵列

Key words: cache; transaction; NVRAM; RAID

中图分类号: TP333　　**文献标识码**: A

1　引言

目前对采用 Cache 技术提高磁盘阵列性能的研究有很多。文献[1]利用排队论和 Petri 网模型,从理论上分析了高速缓存对磁盘阵列性能的影响,并提出了提高磁盘阵列性能的若干方法。文献[2]提出一种新型的 Cache 体系结构— RAPID(Redundant, Asymmetrically Parallel, Inexpensive Disk Cache,简称 RAPID),为存储系统提供高性能、高可用的高速缓存系统。Cache 技术提高磁盘阵列性能的一个典型应用是解决 RAID5 的小写问题(RAID5 的小写问题:如果写操作向 RAID5 的少数磁盘写入数据,为了计算新的校验,需要读取其他多数磁盘的数据,由此而引起小写效率低下),通常有两种做法:(1)缓存写数据。文献[3]应用非易失写缓存解决 RAID5 小写问题,采用 Linear Threshold Scheduling 调度算法防止更新磁盘时数据丢失。(2)缓存校验数据。文献[4]利用 DCMT(基于动态缓存标志表)方法缩短响应时间,提高系统整体性能。

本文为解决 RAID5 的小写问题,用单片 NVRAM 实现非易失写缓存。与文献[5]的不同之处在于,我们通过采用事务机制对 Cache 进行修改,来实现高可用的磁盘阵列 Cache。

2　高可用 Cache 的实现原理

为了实现高可用的 Cache,除了采用 NVRAM 作为 Cache,保持数据不丢失,还应该避免对 Cache 的不完整修改。避免对 Cache 的不完整修改,通常有两种方法:(1)对 Cache 的写操作进行记录。若写操作没能成功执行,重复执行记录的写操作,直到成功执行。(2)对 Cache 进行备份。若写操作执行失败,用 Cache 的备份消除对 Cache 的不完整修改。

对 Cache 的写操作进行记录实现很困难,因为 Cache 的写操作随 Cache 存取模式的不同而变化,很难确定用多大的空间来记录 Cache 的写操作。

Dual-copy NVRAM Cache 就是采用对 Cache 备份实

现高可用阵列 Cache。如果写操作每次将小写数据写入到 Cache 的空闲块,可用对 Cache 地址映射表的备份取代对整个 Cache 的备份。假设写操作对 Cache 做了不完全修改,Cache 空闲块被写入部分数据,因为对地址映射表进行了备份,用地址映射表的备份对地址映射表赋值,取消对地址映射表的不完全修改,此时写入部分数据的空闲块在地址映射表中还是记录为空闲块,Cache 回到失败的写操作执行之前的状态。

本文用对 Cache 地址映射表的备份实现高可用阵列 Cache,在单片 NVRAM 实现两片镜像 NVRAM 提供高可用性。设计中采用灵活的调度算法,为当前应用释放缓存空间,以便 Cache 中有足够的空闲块来存放小写数据。

3 Cache 的硬件设计

NVRAM 具有自备电源,能在断电时保持数据不丢失,读写周期为 70～85ns,访问速度较快。目前,单片 NVRAM 最大容量是 2MB,所以采用意法半导体 M48Z2M1 的 2MB 容量的 NVRAM 实现 Cache。根据需求可将多片 M48Z2M1 级联,实现更大容量的 Cache。NVRAM 做在 PCI 适配卡上(简称 Cache 卡,如图 1 所示),阵列控制器通过 PCI 总线访问 NVRAM、读写 Cache。

Cache 卡由 NVRAM 和 FPGA 组成。NVRAM 通过 FPGA 连接 PCI 总线。FPGA 实现两部分功能:PCI Core 实现 PCI 协议 Target 功能和转换逻辑;完成 PCI Core 后端接口到 NVRAM 接口的转换。PCI Core 从 Altera 公司购买,只需简单配置便可下载到 FPGA。转换逻辑由 SingleReg 模块和 NVRAMCtrl 模块组成(如图 1 上部分所示),用 Verilog 对 FPGA 编程实现。

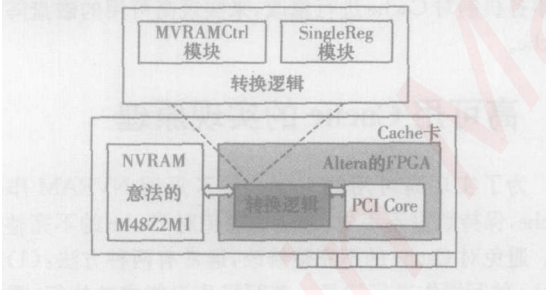

图 1 Cache 卡方框图

SingleReg 模块实现 32 位寄存器的读写。采用一个状态机来响应 PCI Core 后端接口的读写命令,完成对 32 位寄存器的读写。支持单周期的读写命令,对突发读写命令完成一个周期读写后便中止交易。由于 NVRAM 读写周期较长——M48Z2M1 的读写周期是 70～85ns,且一个周期只能读写一个字节,所以此模块不支持突发读写。

NVRAMCtrl 模块实现 NVRAM 的读写。将 32 位寄存器的数据写入 NVRAM,或从 NVRAM 读出数据到此 32 位寄存器。由于需要三个 PCI 周期(90ns)读写一个字节,所以此模块主要完成同步和串并转换(8～32 位或 32～8 位)。若要级联多个 NVRAM,NVRAMCtrl 模块需要增加一个编码器对 NVRAM 使能信号编址;同时,PCI Core 需要重新配置,为 NVRAM 分配更大的地址空间。

4 Cache 的软件设计

Cache 驱动程序用 C 语言编写,通过对 Cache 的管理和调度来实现高可用的 Cache。

4.1 Cache 的组织

Cache 容量为 2MB,将 Cache 按块进行组织,块容量为 8kB,共 256 块,如图 2 所示。

Cache 的前 255 个数据块是数据区,存放小写数据。最后一块是地址表区,存放两份地址映射表:Index0 和 Index1,Index1 对 Index0 进行备份。

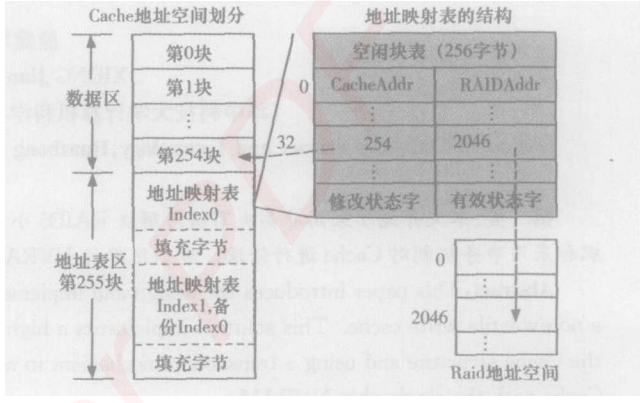

图 2 Cache 的组织结构

地址映射表如图 2 浅灰色部分所示,包含四个部分:空闲块表、查找表和两个状态字。空闲块表是 256 元素的字节数组,前 255 个元素记录 Cache 数据区 255 个数据块的空闲状态,最后一个元素是填充字节,保持地址边界为 4 的倍数。

查找表是 255 元素的结构数组,记录着 Cache 数据区数据块的地址映射关系,每一个结构体{ RAIDAddr:块在 RAID 地址空间的地址;CacheAddr:取值为 0～254,对应数据区 0～254 块}对应一个地址映射关系。如图 2 中虚线部分所示,查找表中第 32 个元素记录着 RAIDAddr 为 2 046 的数据块存放在 Cache 的第 254 块中。Cache 采用全相连映射,为加速对查找表的访问,查找表以 RAIDAddr 为索引升序排列。

有效状态字记录地址映射表的数据是否有效,如果有效状态字为假,说明写操作对地址映射表做了不完整修改。修改状态字记录 Index0 的内容(空闲块表和查找表)与 Index1 的内容是否相同,如果修改状态字为真,说明 Index0 的内容与 Index1 的内容不相同;Index1 不使用修改状态字,Index1 修改状态字始终设为假,便于 Index0 与 Index1 之间相互赋值。

4.2 Cache 的驱动程序

Cache 驱动采用三层体系结构,即接口层、业务层、数据层,如图 3 所示,箭头表示模块间的依赖关系。Cache 驱动采用高度模块化的层次结构,易于扩展。只需要修改 Cache 驱动的数据层,便可级联多个 NVRAM 来扩展 Cache 容量;只需要修改 Cache 驱动的接口层,便可移植到不同的磁盘阵列;只需要修改 Cache 驱动的业务层调度模块,便可灵活配置 Cache 调度算法。

元数据模块主要有三个函数:UpdataIndex0、Upda-

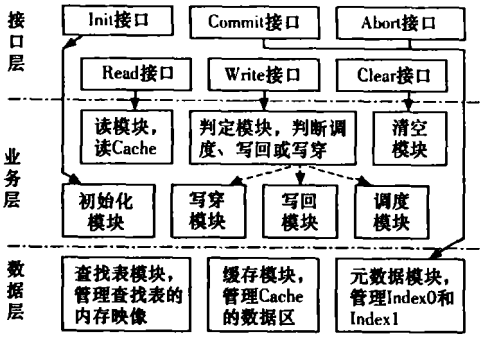

图 3　Cache 驱动的体系结构

[illegible]

(1) UpdataIndex0 :根据需求修改 Index0 ,首先置 Index0 有效状态字为假 ,修改状态字为真 ,然后修改其内容 ;

(2) UpdataIndex1 :提交对 Cache 的修改 ,首先置 Index0 有效状态字为真 ,然后置 Index1 有效状态字为假 ,置 Index0 修改状态字为假 ,接着用 Index0 更新 Index1 ,最后置 Index1 有效状态字为真 ;

(3) ResetIndex :修复对 Cache 的不完整修改 ,使 Index0 和 Index1 的内容相同 ,置 Index0 和 Index1 的有效状态字为真 ,Index0 的修改状态字为假。

判定模块调用查找表模块做出判断 :若缓存小写 ,调用写回模块 ;若写入 RAID ,调用写穿模块 ;若 Cache 被填满 ,调用调度模块释放部分 Cache 空间。

写回模块缓存小写 ,首先调用元数据模块和查找表模块计算小写的地址映射关系 ;然后调用元数据模块 UpdataIndex0 函数记录小写的地址映射关系 ;接着调用缓存模块将小写数据写入 Cache 的空闲块 ;最后调用元数据模块 UpdataIndex1 函数提交写操作。

写穿模块写穿 Cache ,首先调用查找表模块计算与命令相邻的数据块和被命令覆盖的数据块 ;然后调用元数据模块 UpdataIndex0 函数删除与命令相邻和被命令覆盖的数据块的地址映射关系 ,同时调用读模块读出与命令相邻的数据块一起构成新的写命令。

调度模块释放 Cache ,首先调用查找表模块计算最不需要缓存的数据块 ;然后调用读模块读出要释放的数据块 ,将这些要释放的数据块与写命令一起构成新的写命令 ,同时调用元数据模块 UpdataIndex0 函数删除要释放的数据块的地址映射关系。

清空模块调用读模块读出 Cache 中缓存的所有数据块 ,等待写入 RAID 中 ,同时调用元数据模块 UpdataIndex0 函数清空 Index0 ,调用查找表模块清空内存查找表。

读模块首先调用查找表模块计算命中的数据块 ,然后调用缓存模块读出这些数据块。

Init 接口处理初始化任务 ,调用初始化模块 ,初始化各数据结构。Clear 接口负责理清空 Cache 任务 ,调用清空模块清空 Cache 中缓存的数据。Read 接口处理读操作 ,调用读模块读出命中的数据块。Write 接口处理写操作 ,调用判定模块 ,写回或写穿 Cache。Commit 接口提交写事务 ,调用元数据模块 UpdataIndex1 函数提交写操作。Abort 接口撤消写事务 ,调用元数据模块 ResetIndex 函数 ,来恢复失败的写操作对 Cache 的不完整修改。

4.3　以写事务对 Cache 进行修改

如图 4 所示 ,分七个步骤对 Cache 进行修改 ,向右的箭头指示该步骤因故障而中止执行 ,系统重启后进行故障处理步骤 ;向下的箭头指示该步骤正常执行完毕进入下一个步骤。因为使用一条指令对状态字进行修改 ,所以步骤 (1) 、步骤 (3) 、步骤 (4) 和步骤 (6) 都是原子操作 ,要么执行 ,要么不执行 ;而步骤 (2) 和步骤 (5) 包含很多任务 ,是非原子操作 ,在执行的过程中可以被中断 ;相邻的步骤之间也是可以被中断的。

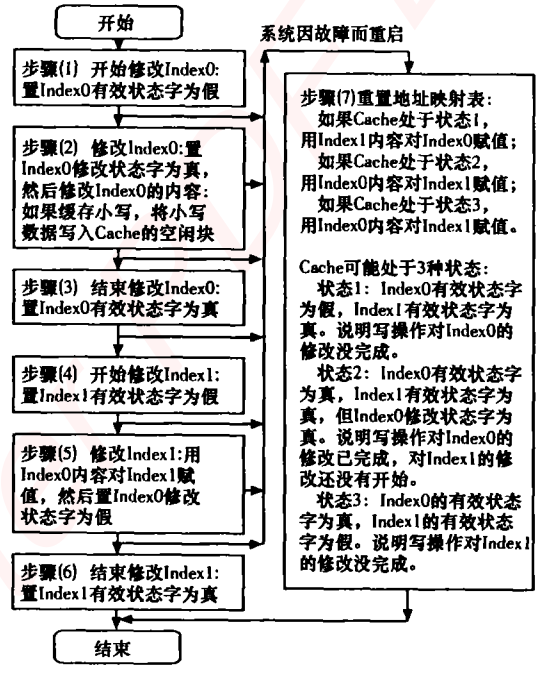

图 4　Cache 的写操作流程

若写操作正常执行 (步骤 (1) - 步骤 (2) - 步骤 (3) - 步骤 (4) - 步骤 (5) - 步骤 (6)) ,则写操作提交 ,Cache 得到完整的修改 ;若写操作由于出现故障而执行失败 ,系统将重新启动 ,执行骤 (7) ,使 Cache 还原到这次失败的写操作执行之前的状态 ,则写操作撤消。总之 ,写操作要么提交 ,要么全部撤销。写操作是一个正确的状态变换 ,而且写操作提交的结果保存在 NVRAM 中 ,整个 Cache 系统始终处于确定状态。所以 ,对 Cache 写操作构成写事务 ,Cache 的数据始终保持一致有效。

如果 Write 接口接收到需要缓存的小写命令 ,写操作从步骤 (1) 执行到步骤 (6) ,写事务提交。

如果 Write 接口接收到需要写入 RAID 的写命令 ,写操作从步骤 (1) 执行到步骤 (2) ,对 Cache 做部分修改 ,返回写命令。如果用户调用 Write 接口导致调度模块被调用 ,调度模块执行写操作的步骤 (1) 到步骤 (2) ,对 Cache 做部分修改 ,返回写命令。如果用户调用 Clear 接口来清空 Cache ,执行写操作的步骤 (1) 到步骤 (2) ,对 Cache 做部分修改 ,返回写命令。当写命令写入 RAID 后 ,用户调用 Commit 接口 ,执行写操作的步骤 (3) 到步骤 (6) ,完成对 Cache 的修改 ,写事务提交。

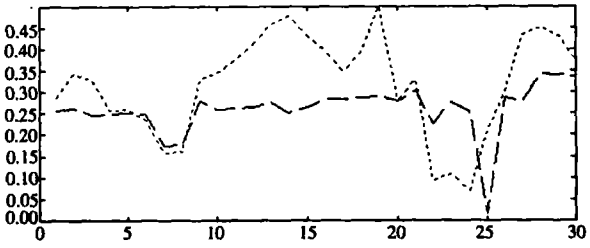

图 1　BP 神经网络预测结果

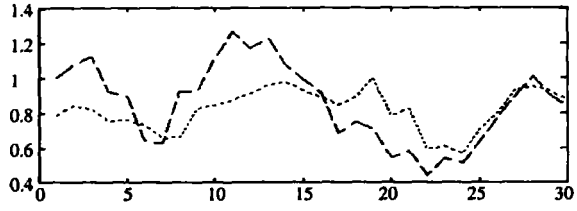

图 2　基于主成分分析的 BP 神经网络预测结果

本文虽然获得了初步的结论,但还存在很多问题:影响股票指数变动的因素极为复杂,除了本文所选的技术指标外,还包括总体经济指标,如国民总值、总就业率、汇率以及税率等。本研究所采用的输入变量只限于技术指标,没有包括影响股市的各个因素。神经网络隐层节点数目的确定,则是通过多次尝试获得的。采用何种神经网络结构以获得满意的结果将是后续研究的内容。

参考文献:

[1] 张秀艳,徐立本. 基于神经网络集成系统的股市预测模型[J]. 系统工程理论与实践,2003,23(9):67-70.

[2] 林杰,郭耀煌. 用神经网络方法预测股票短期走势[J]. 西南交通大学学报,1998,33(3):299-304.

[3] 张健,陈勇. 人工神经网络之股票预测[J]. 计算机工程,1997,23(2):52-55.

[4] 孟祥泽,刘新勇,车海平,等. 基于遗传算法的模糊神经网络股市建模与预测[J]. 信息与控制,1997,26(5):388-392.

[5] 吴文锋,吴冲锋. 股票价格波动模型探讨[J]. 系统工程理论与实践,2000,20(4):63-69.

[6] 阎平凡,张长水. 人工神经网络与模拟进化计算[M]. 北京:清华大学出版社,2000.

[7] 陈小前,罗世彬,王振国,等. BP 神经网络应用中的前后处理过程研究[J]. 系统工程理论与实践,2002,22(1):65-70.

(上接第 118 页)

[3] J McHugh, S Abiteboul, R Goldman, et al. Lore: A Data-Base Management System for Semistructured Data[J]. ACM SIGMOD Record, 1997, 26(3): 54-66.

[4] T Bray, J Paoli, C M Sperberg-McQueen. Extensible Mark-up Language(XML) 1.0. 2nd Ed[EB/OL]. http://www.w3.org/TR/2000/REC-xml-20001006,200-01.

[5] P V Biron, A Malhotra. XML Schema Part 2: Datatypes[EB/OL]. http://www.w3.org/TR/xmlschema-2,2001-05.

[6] S Abiteboul, S Cluet, T Milo. Querying and Updating the File[A]. Proc of the Conf on Very Large Data Bases (VLDB)[C]. 1993. 73-84.

(上接第 121 页)

如果写操作在执行中因故障被中止,系统重启,调用 Abort 接口,执行步骤(7),重置地址映射表,对地址映射表进行一致性检查,取消失败的写操作对地址映射表所作的不完整修改,使地址映射表还原到这次失败的写操作执行之前的状态,写事务撤消。

5　实验结果及分析

采用 FC 磁盘阵列作为实验平台,将阵列配置成 RAID5,分块大小设为 8kB。持续向阵列读写数据,Cache 都正常工作。将阵列作为服务器,能正确运行 7×24 小时。在阵列工作过程中,任意进行重启,Cache 始终是一致有效。但是,阵列中的数据偶尔遭到破坏,因为大写采用写穿策略直接写到阵列中,突然重启会导致大写部分写入阵列,破坏阵列数据的完整性。这个问题有待进一步研究,以设计出高可用的磁盘阵列。

6　结束语

本文采用单片 NVRAM 实现磁盘阵列的 Cache,为解决 RAID5 的小写问题提供一个高可用、经济的解决方案。相对于 Dual-copy NVRAM Cache,此方案以一半的费用提供一样的可用性和性能。相对于 RAPID-Cache[2],此方案以简单的设计、低廉的造价提供一样的可用性和性能。此外,此方案的实施是高度模块化和层次化的,具有良好的可扩展性、可移植性。

参考文献:

[1] 周可,张江陵,冯丹. Cache 对磁盘阵列性能的影响[J]. 电子学报,2003,31(9):1337-1340.

[2] Yiming Hu, Qing Yang, Tycho Nightingale. RAPID-Cache - A Reliable and Inexpensive Write Cache for High Performance Storage Systems[J]. IEEE Trans on Parallel and Distributed Systems, 2002, 13(3): 290-307.

[3] A Varma, Q Jacobson. Destage Algorithms for Disk Arrays with Non-Volatile Caches[A]. Proc of 22nd Annual Int'l Symp on Computer Architecture[C]. 1995. 83-95.

[4] 赵亮,刘光明. 利用 DCMT 来解决 RAID 5 的小写问题[J]. 计算机工程与科学,2002,24(5):104-107.

[5] 缪军海,朱兰娟,吴智铭. RAID 中 Cache 的设计与实现[J]. 微型电脑应用,2001,17(4):29-31.

[6] John L Hennessy, David A Patterson. Computer Architecture: A Quantitative Approach. Third Edition[M]. 北京:机械工业出版社,2002.

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通

12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计

5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究