基于RAID5的磁盘阵列Cache的研究与实现

王湘娜， 蒋本珊， 徐　渐

（北京理工大学计算机系，北京 100081）

摘　要：通过对RAID5结构的研究，发现磁盘阵列Cache能减少I/O子系统的写响应时间，可将多个小的写转换为大的写，从而减少冗余写的次数。最后给出作者在参与RAID5控制器的研制工作中对磁盘阵列Cache的实现方法。

关键词：磁盘阵列；Cache；冗余；可靠性

Study and Implementation of the RAID5-based Disk Cache

WANG Xiangna,　JIANG Benshan,　XU Jian

(Computer Department of Beijing University, Beijing 100081)

【Abstract】 This paper studies the structure of RAID5 and shows that cache can reduce the write response time，accumulate many small writes to make a full-stripe write and thus reduce the redundant write times. Finally，the paper presents the implementation of a RAID5-based disk cache.

【Key words】RAID; Cache; Redundancy; Reliability

1 概述

近几年，CPU的处理速度以每年40%–100%的比率不断提高，磁盘存储密度也以每年50%的速度增加，平均每3年提高4倍，但磁盘的存取时间却改善甚微，每10年仅减少1/3，目前仍停留在毫秒级，与内存及CPU内部Cache的差距为4～6个数量级。由于计算机系统整体性能取决于系统中性能最差的关键部件，因此以磁盘作为主要外存储器的外存子系统就成为影响整个系统性能提高的瓶颈，因而开发高效快速的外存子系统已是刻不容缓。

独立冗余磁盘阵列(RAID)是改善磁盘存储性能的有效方法之一。它是将要存储的数据按一定规则分块，存放在多个盘上，利用多个盘的并行存取、并行传输来匹配带宽，提高存取速度，对主机来说它相当于一个快速大容量的磁盘。另外，RAID还利用其提供的冗余数据(镜像数据或冗余校验信息)提高了系统的可靠性。目前的RAID分为6级：RAID0～RAID5，其中RAID5采用奇偶校验来提供数据恢复的能力，但它没有独立的校验盘，校验信息分布在各个磁盘驱动器上。RAID5由于存取速度快、数据可靠性高且实现成本较低，因而被广泛地应用。

虽然磁盘阵列利用多个磁盘并行存取并行传输，提高了吞吐率和数据传输率，从而使外存子系统的响应时间得到一定改善。但是对于一些请求，尤其是小块随机请求，它的响应时间仍然主要由磁盘的服务时间(主要是寻道时间、旋转延迟时间和数据传输时间)决定。这种信息的机械定位带来的延时是外存子系统性能差的根本原因之一，并未得到根本性解决，CPU与磁盘I/O子系统之间在速度上的巨大差距仍然存在。而高速缓存Cache作为改善计算机系统性能的一种有效方法，曾被用在CPU和主存之间以解决两者速度不匹配的问题，使得系统的速度接近CPU。与CPU Cache相比，如果在CPU/主存与磁盘阵列之间引入Cache，则Cache面临的数据请求更大，后备存储器延迟更长，因而Cache的作用会更加明显有效。由此可见，磁盘阵列更需要Cache的支持。因此为进一步缩小外存与CPU间数量级的速度差异，满足主机日益增高的I/O请求率，磁盘阵列引入Cache势在必行。

2 RAID5的地址映射

RAID5阵列由N块磁盘组成，数据(data)按块分布存储在N个磁盘上并划分成stripe，在同一物理盘上属于同一个stripe的连续块构成一个strip，strip里连续块的个数称作stripe depth，stripe里包含的总块数称作stripe size。每个磁盘上相同位置的块构成一个校验组(parity group)，每个校验组里包含一个校验块(parity)，它是该组中其它数据块的异或值。应用程序看到的RAID5是一个连续的虚拟磁盘空间，其数据容量是N－1个磁盘的总容量。虚拟磁盘上的逻辑块与其在物理盘上的位置之间具有映射关系，文献[7]中曾给出RAID1结构的地址映射公式，本文将给出RAID5结构下逻辑块到物理块的映射公式：

Stripe number = Virtual block number /(stripe size － stripe depth)

Block number within stripe = MOD(Virtual block number / stripe size)

Disk number = Block number within stripe / stripe depth

Block within strip = MOD(Block number within stripe / stripe depth)

Physical block = Stripe number stripe depth + Block within strip

例如：图1的RAID5由4块磁盘构成，stripe depth为2，stripe size为8。

如果应用程序要访问的逻辑块号是10，计算过程是：

Stripe number = 10 /(8 － 2)= 1

Block number within stripe = MOD(10 / 8)= 2

Disk number = 2 / 2 = 1

Block within strip = MOD(2 / 2)= 0

Physical block = 1 × 2 + 0 = 2

因此，逻辑块10转换为物理块的地址是Disk 1上的第2号块。由于有了这样的映射关系，任何逻辑块都能唯一地对应到一个物理块。不仅如此，根据逻辑块号还能确定相应校

验块所在的磁盘号，可以使用下面的公式(Total disk是组成磁盘阵列的磁盘个数)：

Parity disk number = Total disk － 1 － MOD(Stripe number / Total disk)

如与逻辑块10同属一个校验组的校验块所在磁盘号为：

Parity disk number = 4 － 1 － MOD(1 / 4)= 2

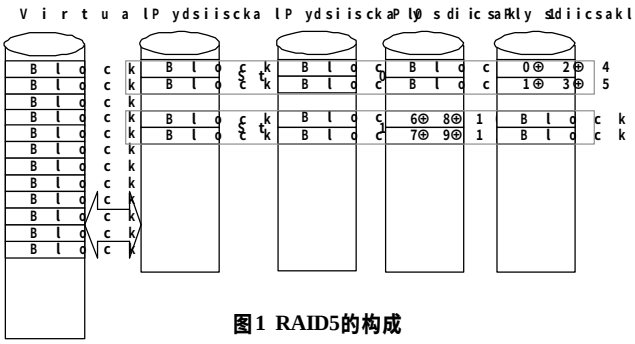

图1 RAID5的构成

3 RAID5写操作

RAID5中当有一个磁盘失效时，磁盘阵列仍能继续工作，失效数据可由其它盘上的数据和校验通过异或而重新得到。例如，图1中如果disk 0发生故障，当要读取Block 8里的数据时，可以这样计算：Block 8 = (6⊕8⊕10)⊕Block 6 ⊕ Block 10，从而能够恢复数据。而这种可靠性的代价是：为了保持数据的一致性，每次写数据要同时更新数据块和校验块。校验块的计算可使用公式：

$Parity_{new} = Parity_{old} \oplus Data_{old} \oplus Data_{new}$

例如，当图1中Block 8要被改写为new data时，校验块的更新是这样计算的：

$Parity_{new}$ = (6⊕8⊕10) ⊕ Block 8 ⊕ new data

如果一次写要修改stripe里所有的数据，这种情况称作full-stripe write，这时新校验的计算可与写数据同时进行，而不必到磁盘上去读取数据。但在实际应用中这种full-stripe write是不多见的，更经常的情况是对部分stripe数据的小块写请求(partial-stripe write)。这时，同属一个校验组的其它盘上的数据块和校验块都需要从磁盘上读取。于是，对RAID5的写操作分解为以下多步完成：(1)读要改写的旧数；(2)读旧校验信息；(3)旧数据和旧校验信息做异或计算；(4)将异或计算的结果与要写入的新数据进行异或得到新的校验信息；(5)将新的校验信息写回到校验块；(6)最后写入新的数据。其中步骤(1)和(2)的数据存储在不同的盘上，可以并行执行，步骤(5)和(6)也可以并行执行。这种写操作称为"读–修改–写"(read-modify -write)，因此一次写请求的时间相当于一次并行读，一次并行写，再加上进行异或计算的时间。由此可见，RAID5由于引入冗余数据，完成一次写操作的时间比单个盘所花的时间要多，从而大大降低了RAID5的速度。而引入了Cache后，要写到磁盘上的数据和校验可先暂存在Cache里，并由控制器报告主机操作已经完成，主机不必等待本次写请求的完成，而实际对磁盘的写操作则可在I/O请求结束后的空闲时间里将暂存数据一次写回到磁盘上。因此从主机的角度来看，写响应时间变为一次 I/O读时间加上异或计算的时间，比不使用Cache的响应时间少了一次I/O写的时间。由于异或计算通常由硬件完成，速度很快，可忽略不计，因此使用Cache后，RAID5写响应时间几乎等于单个盘的写响应时间。另外，来自应用程序的多个小块写请求暂存在Cache里，使得有足够的块可以构成一个full-stripe write，从而避免了更多的读—修改—写顺序操作。而且，根据访问局部性原理，主机可能在短时间内对某个块多次进行修改，而把块放在Cache里，就可以减少往磁盘上的冗余写操作。

4 关键技术

Cache地址变换方式决定着数据块存放在Cache的什么地方的问题，它直接影响命中率和性能。一般常用的地址变换有3种：全相联(Fully associative)、直接映射(Direct-mapped)和组相联(Set associative)，常用的变换方式是组相联映像。组相联映射方式中，组数和组的大小也影响着命中率。一般来说，每组的块容量越大，块的冲突概率和Cache的失效率越低，但组内块数太多时，进行组内相联比较所花时间较多，造成查表速度降低，实现成本增加。

Cache替换算法的确定主要是看是否有利于提高Cache命中率，其次要看算法是否便于实现。常用的Cache替换算法有先进先出(FIFO)算法和近期最少使用法(LRU)，这两种替换算法在传统Cache实现中可获得很好的性能，并且LRU替换算法是一种堆栈型算法，堆栈型算法不会出现增大Cache容量反而降低命中率的现象，所以在存储层次结构中普遍采用LRU算法。

磁盘阵列系统中，主机以扇区为单位进行I/O请求，最小寻址单位是扇区，但是盘阵列以块为单位进行数据存储，因此Cache块的大小对命中率影响非常大。当块的大小由很小变得较大时，命中率首先会增加。这是因为根据局部性原理：在一个较短的时间内，程序频繁访问的是那些物理地址相邻的块，因此增大Cache块的大小可使得更多有用的数据被装入Cache。但是，当块的大小变得相当大时，大量无用数据被送入Cache，占用Cache大量有效空间，反而命中率下降。实际上，块大小与命中率的关系比较复杂，它取决于特定持续的局部性特征。以往的测试表明块大小为4~8个扇区时，应用请求的平均访问时间最小。

5 基于RAID5的磁盘阵列Cache的实现

我们使用了一个带UPS的SDRAM作为磁盘阵列的Cache，其容量最大可达256MB。系统在主机侧要为来自应用程序的I/O请求建立队列，队列遵循先来先服务(FIFS)的原则，同时可处理最多8个I/O请求。每个I/O请求的格式为： <op，LBA，nblk>，其中op代表请求类型(读或写)，LBA表示逻辑空间的起始块地址，nblk表示I/O请求的块数(512B/块)。一个逻辑I/O请求转换为物理请求后可能对应多个物理盘请求，对第i块盘的请求格式为：<i，LBA[i]，nblk[i]>，其中是磁盘号，LBA[i]是第i块盘上的起始块地址，nblk[i]是第i块盘上的请求块数，由和LBA[i]生成物理地址Disk_LBA。

我们将Cache块大小取为4kB(8个扇区)，一块称为一页，并将Cache按页划分，由于Cache与磁盘之间的数据传输是以页为单位，因此I/O请求也要按页划分。由于RAID5中并没有独立的校验盘，校验块和数据块一样是分布在各个磁盘上的，因此应能判断该页是数据页还是校验页。为此定义了几个参数：NSEC_IN_STRIP表示一个strip里包含的扇区个数，NPAGE_IN_STRIP表示一个strip里包含的页数，NSEC_IN_PAGE表示一个页里包含的扇区个数；PAGE[i][j]则表示位于第号盘上的第页。根据和就能够计算出该页所在的stripe号并能知道该页是否为校验页。计算公式为：

stripe_num = first_stripe + (j * NSEC_IN_PAGE) / NSEC_IN_STRIP

is_parity = MOD(stripe_num / total_disk)= = total_disk －1 － i)

其中，first_stripe是本次I/O映射到磁盘阵列上的第一个stripe，total_disk是构成磁盘阵列的磁盘个数。

对于Cache写操作，如果命中，则Cache里的数据被修改(MOD位为1，称为脏块)，但此时磁盘上的数据还未来得及更新，仍然是旧的数据。为保持磁盘与Cache的一致性，需要在适当的时候将Cache里修改过的数据写回到磁盘上。我们选择write back方式进行回写，即只向Cache写入，并用标记加以标明(WB置为1)，直到Cache中被修改的页要被进入的页取代时，才将其写回磁盘。当有一个脏块(块A)要写回到磁盘上时，还要同时计算并改写相应的校验块，以保持数据的一致性。假设这时有与该块同属一个校验组的另一块(块B)也要进行回写，如果在块A进行异或计算过程中块B读旧校验，则校验信息改写后，造成数据不一致。为防止这种情况发生，在回写某块前，将同一校验组的所有其它块锁住(LOCK位置1)，直到该块与校验块都已写回到磁盘后才能解锁。

Cache控制器执行的命令有3种类型：request、search dirty和release clean，分别对应Cache控制器的3个任务，即：访问Cache，根据访问Cache的结果决定如何为读写请求分配页面并置相应Flag标志位；检查Cache中有无未被加锁且被修改过的页，准备以后将其回写到磁盘；释放Cache中未被加锁且未被修改的页，将其加入到空闲页链表。其中后两种任务是在系统负载较轻时进行。

Cache里包含tag表和页表。tag表由tagdata项组成，tagdata的数据结构如图2。

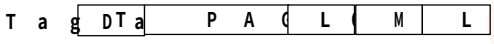

图2 tagdata的数据结构

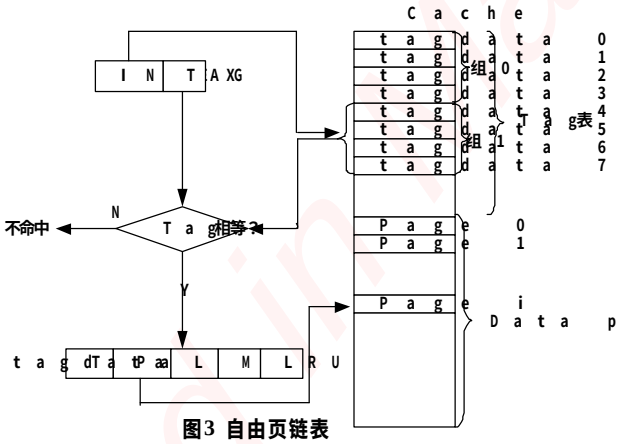

图3 自由页链表

LRU、MOD和LOCK分别表示页面最近使用情况、页是否被修改以及是否被加锁。无论I/O请求是读还是写，都要先进行Alloc_page，看要访问的页是否在Cache里。对页的检索方法使用了组相联的思想，当要访问某一页时，由物理地址Disk_LBA形成TAG_INDEX，根据INDEX检索tag表得到一组(4个)tagdata项，再将TAG与4个tagdata中的TAG值进行比较，如果有相等的TAG值，则表明对该页的访问为命中，否则为访问失效。页表记录了页面的使用情况，并将所有空闲页(free page)组织成一个自由页链表(free-page-list)，见图3。

当访问Cache失效时，有3种情况：(1)tag表里有空的tagdata且页表里有free page存在。这时可直接从自由页链表中取出一页，并在tag表中分配一个tagdata项记录该页。(2)有空的tagdata项但不存在可用的页面，这时要调用任务release clean检查Cache页表，将所有未被修改过的页(clean page)加入到free-page-list中以供下次分配时使用。如果没有clean page，则调用任务search dirty查找修改过的页(MOD位为1)，将其标记为需要写回到磁盘，之后才能作为可用页面进行分配。为了减少这种情况的发生，可以在系统负载较轻时进行search dirty和release clean，将大量被占用的页重新变为可用，从而在有tagdata项可分配时，保证有可用的页进行分配。(3)tag表里已不存在空的tagdata，则必须从tag表里淘汰一个tagdata，同时页表里与之对应的页也将被替换出Cache。替换策略要用到tagdata里的3个标志位LOCK，MOD，LRU。首先替换未被加锁且未被修改过的tagdata(LOCK和MOD位均为0)，其对应的页可直接被新的数据所覆盖。如果没有这样的页则替换掉未加锁但修改过页，在替换前先要将该页的内容写回到磁盘，然后才能装入新的数据。在选择可替换的tagdata项时，如果有多个tagdata满足条件则选择LRU值最小的进行替换。

6 结束语

本文阐述了将Cache引入磁盘阵列(RAID5)的必要性，表明Cache技术是改善RAID5小块写问题的有效途径，并分析了磁盘阵列Cache的地址映射、替换算法及Cache块大小等影响Cache性能的技术因素。在对磁盘阵列Cache的实现过程中，我们使用了组相联映射、LRU替换算法等比较成熟的技术，在Cache回写策略上采用write back方式，从而提高了写磁盘速度，减少冗余写盘操作。另外通过对校验组加锁，有效地防止了同一校验组里多个块同时降级而导致的数据不一致现象。由于磁盘阵列技术在国内外均处于起步阶段，还有许多关键问题需要进一步研究。我们实现的磁盘阵列Cache也还有需要改进的地方，例如：如何提高Cache命中率(如采用预取策略)，加锁的粒度多大为合适(目前为一个校验组)，何时将Cache里需要回写的数据和校验清空(写回磁盘)等。这些问题都还有待进一步探讨，有效解决这些问题将能够更充分地发挥磁盘阵列Cache的作用。

参考文献

1 刘桂兰，祝天龙，周学仁等. 廉价冗余磁盘阵列(RAID)Cache 浅析. 电子计算机外部设备，1995，16(6)

2 祝天龙，裴先登，周学仁等. 冗余交叉磁盘阵列性能研究. 计算机 工程与设计，1994，16(2)

3 Massiglia P. The RAIDbook: A Source Book for Disk Array Technology (Fifth Edition). The RAID Advisory Board , St. Peter, MN，1996-02

4 Menon J, Cortney J. The Architecture of a Fault-tolerant Cached RAID Controller. Proceedings of the 20th Annual International Symposium on Computer Architecture，San Diego，Calif，1993: 76

5 Massiglia P. RAID for Enterprise Computing. Copyright @ VERITAS Software Corporation，1999

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
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计

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
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现

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
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现

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
18. 基于 Windows CE 的可视电话的研究与实现

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
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植

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
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计

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
15. 基于 UEFI 固件的攻击验证技术研究

Programming:

1. 计算机软件基础数据结构 - 算法