第 33 卷 第 1 期 计 算 机 工 程 2007 年 1 月
Vol.33 No.1 Computer Engineering January 2007
·工程应用技术与实现· 文章编号：1000—3428(2007)01—0259—03 文献标识码：A 中图分类号：TP338.7

# 基于 Intel 80321 的 RAID5 系统性能优化

赵 昕，戚文芽，廖 军
(解放军信息工程大学信息工程学院，郑州 450002)

**摘 要：**结合磁盘存储阵列的应用，实现了基于 Intel 80321 I/O 处理器的 RAID 系统。该文利用了 I/O 处理器应用加速单元的硬件特性，在软件方面用高效读、写高速缓存的管理策略，实现了 RAID5 系统性能的优化，实际运行结果表明，该性能优化能达到预期效果。
**关键词：**80321 I/O 处理器应用加速单元；高速缓存管理；RAID5 优化

# Optimization of RAID5 System Based on Intel 80321

ZHAO Xin, QI Wenya, LIAO Jun
(College of Information Engineering, Information Engineering University, Zhengzhou 450002)

**【Abstract】**Combined with the application of the storage array, the RAID system is achieved based on the Intel 80321 I/O processor. And the optimization of RAID5 system is put forward as follow: the implication of the application accelerator unit and the management strategy of the cache. It is proved that the performance of the system meets the expectations.
**【Key words】**80321 IOP AAU; Cache management; Optimization of RAID5

Intel 80321 I/O处理器为 544 脚PBGA封装，集成了Intel XScale内核高性能I/O芯片。该处理器内核可以运行在600MHz环境中，最高带宽可达 1 064MBps的 133MHz PCI-X总线可提高系统I/O的性能。核心单元通过 64bits、200MHz内部总线实现的传输速度高达 1.6GBps。集成的内存控制器支持 1GB、64B及ECC的DDR SDRAM全速运行于 200MHz以及flash的控制接口。DMA控制器提供了低时延，高吞吐量的PCI设备与内存的数据传输，并通过两个独立的DMA通道与内部总线相连。除此之外集成了如性能监视单元、同步串行端口单元、外部总线接口、$I^2C$总线接口、应用加速单元(AAU)等智能单元。尤其AAU是专门设计用来提高应用中数据计算与传输性能的模块。数据传输的高效性是系统设计中所要考虑的首要问题，而如何充分利用I/O处理器的硬件特性尤其是AAU是系统优化的基础。

## 1 磁盘阵列 RAID 介绍

近年来，硬盘在容量、存取速度及可靠性方面都得到了很大提高，但这仍然跟不上处理器等其它硬件的发展要求，使得硬盘成为计算机系统中的一个瓶颈。为了解决系统对磁盘高速存取的要求，人们采取了多种措施。1988 年，美国加州大学伯克利分校的 D.A.Patterson 教授提出的廉价冗余磁盘阵列(Redundant Array of Inexpensive Disks, RAID)就是其中一种。RAID 将普通硬盘组成一个磁盘阵列，应用层写入的数据通过 RAID 控制器分成多个数据块，然后并行写入磁盘阵列。在读取数据时，RAID 控制器并行读取分散在磁盘阵列中各个硬盘上的数据，把它们重新组合后提交应用层。由于采用并行读写操作，因此提高了存储系统的存取效率。此外，RAID 还可以采用镜像、奇偶校验等措施，来提高系统的容错能力，保证数据的可靠性。其中 RAID5 就采用这种方式，即通过采用数据分块并行传送的方法，在数据分块之后计算它们的校验和，然后把数据块和校验信息交叉写到阵列中的每个硬盘上。实际应用中，RAID5 由于其具有数据并行高速传输，当一个硬盘损坏时可通过校验数据恢复，以及较低的成本等特点，在获得较高的可靠性的同时，又保持了令人满意的数据传输率和读写性能，得到广泛应用，亦为本文讨论的对象。

由于 RAID5 自身特点，导致其写入性能受限的后果。首先是奇偶校验的计算，因每次磁盘写入操作必须计算的校验信息，占用很大的系统资源。其次对于小写请求，需执行“ 读、修改、写 ”的一系列的操作，需要 4 次磁盘 I/O 操作。大量的小块数据写入将极大地降低了 RAID5 的磁盘性能。本文针对应用中以上 2 个难点分别对处理器的 AAU，缓存策略进行改进，通过降低系统资源占用及读写速率，提高磁盘性能。

## 2 系统实现

系统模块如图 1 所示。

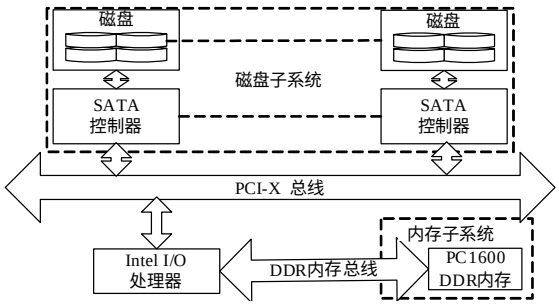


**图 1 系统模块**

存储系统分为三大模块：(1)核心处理器模块，主要包括

**基金项目：**国家“ 863 ”计划基金资助重大项目(2005AA121210)
**作者简介：**赵 昕(1981－)，男，硕士，主研方向：信息处理，嵌入式系统开发；戚文芽，副教授、博士；廖 军，工程师
**收稿日期：**2006-01-28 **E-mail：**zhaoxin@plaieu.cn

I/O 处理器；(2)磁盘子系统模块，包括 SATA 磁盘控制器及高速磁盘；(3)内存子系统模块为系统提供缓存资源，其中包括 RAID 子系统高速缓存。内存子系统可以直接通过单独的总线实现与处理器之间的数据交互，无须占用 PCI-X 带宽，同时处理器利用高速总线实现了对磁盘子系统的管理。

## 3 RAID5 系统优化

RAID5 系统的优化基于上述三大模块结构，首先实现基于 I/O 处理器特性的优化，其次由于高速缓存在系统三大结构中至关重要的桥梁作用，最后提出了对管理高速缓存进行优化的策略。

### 3.1 基于 AAU 的实现与优化

AAU 通过内部总线在不占用 PCI-X 资源的前提下，利用直接寻址的方式实现数据于处理器及内存子系统之间的交互。其通过 I/O 处理器内部单元协同处理的特性，尤其是 AAU 提供的布尔运算加速单元在硬件上实现了 RAID5 所需的 XOR 运算。可以快速实现多数据块在系统内存中读出、计算、写回的过程。支持 1KB (8B*128)的存储序列，最高支持 32 个数据块并行 XOR 运算及传输，每个数据块最大支持 16MB。

基于以上特性，本文通过以下方法充分利用 AAU，AAU 实现状态转移图如图 2 所示。

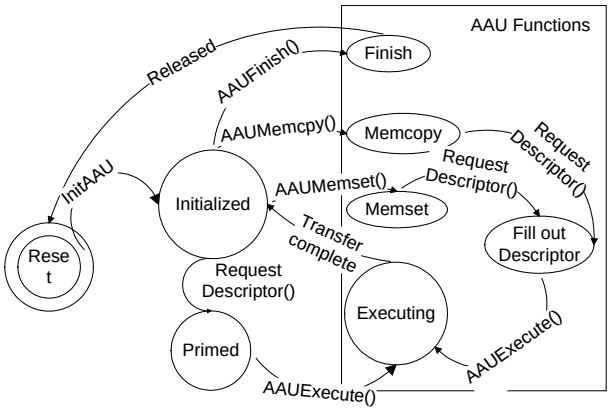


**图 2 AAU 状态转移**

具体来说包含以下 5 种基本状态：

(1)Reset：在该状态下设置中断处理，分配并对齐 AAU 描述符，为了便于后续 AAU 描述符的实现对 AAU 的数据传输进行初始化。随后进入 Initialized 状态。

(2)Initialized：初始化完成后请求描述符。

(3)Primed :初始化后如果成功申请描述符后进入此状态，将当前描述符加入到链表，启动 AAU 后进入 Executing 状态。

(4)Executing：进行中断处理，执行 AAU 运算。如果运算完成返回 Initialized 状态。

(5)Finish :在 Initialized 状态下执行 AAUFinish 操作进入此状态。释放由 AAU 执行所分配的内存，分离中断处理，进入 Reset 状态。

以上基本状态之间的跳转主要依靠以下 7 个主要动作来实现 :RequestDescriptor，memcopy，memset，FilloutDescriptor，AAUFinish，InitAAU，AAUExecute。AAU 所需的描述符数据结构如下：

```
struct AAUDescriptor
{ void * nda;   //下一描述符地址
  void * sar[4];      //源地址
  void * dar;  //目的地址
  unsigned int bc;    //计数
  unsigned int dc;    //描述符控制字
void * esar0[4];      //扩展源地址 0
unsigned int edc0;    //扩展描述符控制字 0
void * esar1[8];      //扩展源地址 1
unsigned int edc1;    //扩展描述符控制字 1
void * esar2[8];      //扩展源地址 2
unsigned int edc2;    //扩展描述符控制字 2
void * esar3[8];      //扩展源地址 3
unsigned int pad[21];// 64words  对齐};
```

此外通过对所有分配给 AAU 的描述符的跟踪记录，实现了描述符的链表。

在实现 AAU 的基础上进一步对以下部分进行优化：(1)错误控制能力，及时处理突发错误，使系统在保证性能的同时稳定运行。(2)实现 AAU 编程的可重入性，通过同步与互斥的机制，针对多任务的环境优化。(3)通过提高每次运算描述符的分配率，增强 AAU 的执行性能。(4)程序代码上通过对基于 Intel XScale 内核寄存器的重新配置，使硬件资源利用率最优。

### 3.2 高速缓存读写性能优化

虽然磁盘阵列利用多个硬盘并行存取而提高了数据传输效率。但响应时间仍然主要由磁盘寻道、旋转延迟和数据传输等时间决定。尤其难以克服的机械定位延时是磁盘系统性能差的主要原因之一。其中处理器、高速缓存与磁盘 I/O 子系统之间的数据传输在速度上仍存在巨大差距。因此作为传输桥梁的高速缓存，对其优化显得十分重要。本文采用从系统中划分出独立管理的磁盘 I/O 高速缓存的方式，在读、写两方面分别进行了优化。

3.2.1 Cache 读性能优化策略

读 Cache 的基本方式是，如果命中则将其中数据拷贝到该任务的数据区；如果没有命中，需要从磁盘中读取该数据块至应用层数据区，同时放入 Cache 进行管理。根据应用的不同，存在替换和预取两种管理方式。

首先是替换策略，其出发点是读出不久的数据可能再次被使用。常用算法有以下 2 种：(1)先进先出(FIFO)算法，选择最早装入的数据块作为被替换的块 ;(2)近期最少使用(LRU)算法，选择近期最少访问的数据块作为被替换的块。其中 FIFO 算法最容易实现，但 LRU 算法能比较正确地反映数据的局部性。因为当前最少使用的数据，在将来可能最少被访问。LRU 算法与 FIFO 算法类似，根据数据的使用情况来预测未来数据使用情况，但其比 FIFO 更能正确地反映数据局部性。然而 LRU 算法可能导致 Cache 中存放了大量只访问过一次的数据，访问可能性高的数据反而被替换出去。为了改进该算法的不足本文提出了区域 LRU 替换策略。将 Cache 划分为“非活动区”和“活动区”两部分，每个部分的数据块按最近被访问的次数排列。如果一次访问失败，数据将放入 Cache 非活动区中访问次数最多的一端；如果命中，数据将被移动到 Cache 活动区中访问次数最多的一端，而活动区中访问最少的数据将移到非活动区访问次数最多的一端。实现了活动区中数据至少可被访问两次，同时活动区中移出的数据还有被访问的机会。经过反复试验表明，在本系统中该算法性能在活动区大小占整个读 Cache 的 65%时达到最优。其次是预取策略，其出发点是已经读入的数据将不再需要，而是即将读入的数据。该策略利用的是数据局部性，多用于

多媒体音视频服务器等需要顺序访问大文件的系统。直接影响性能的是预取块的大小。因此获得最优预取性能的有效方法是获取文件信息，以便适应预取数据块的大小。

根据不同存取模式信息选择不同缓存策略。本文采用由应用层将存取数据信息提供给存储系统的策略，由此来优化I/O性能。

3.2.2 Cache写性能优化策略

RAID5读数据时可以并行操作，但是写数据需要将数据条中的原有数据读出并更新校验后写入。最简单的写入是一次修改数据条里所有数据，这时新校验的计算可与写数据同时进行，而不必读取磁盘数据。但在实际应用中这种情况并不多见，一般是对部分数据条的写操作。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，写操作可分解为以下几个步骤：(1)读要改写的旧数据及旧校验信息；(2)旧数据和旧校验信息作异或计算；(3)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；(4)写回到校验块及新数据。

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。RAID5引入了冗余数据，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的性能。针对写Cache的优化显得尤其重要。由于数据的局部性，某些数据块在较短时间内会被频繁改写。上一次写入Cache的数据，还没有来得及写回到磁盘，而对该数据块新的写请求已经到来。尤其在数据库和文件管理等应用中，只需在Cache里用新的数据覆盖即可，从而消除对磁盘数据的频繁改写以及由此引发的校验数据的改写问题。

理想情况下，从Cache中将数据写回磁盘的操作对用户请求来说是透明的。目前通常采用定期写回法和伺机写回法，定期回写法主要是为了避免系统的突然断电，以克服数据的不稳定性，由于本系统采用了后备电源，且绝大多数情况下I/O请求并不饱和，因此本文主要讨论伺机回写法以提高性能。基本策略是：欲写到磁盘上的数据和校验可先暂存于Cache，并报告应用层操作已经完成，应用层任务不必由于写请求而阻塞，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Cache数据一次写回到磁盘上。首先从应用层角度来看，写响应时间变为一次读时间加上异或计算的时间，减少了一次写的时间。RAID5写响应时间几乎等于单个盘的写响应时间。其次可以尽量构成填充整个数据条的写入操作，从而避免了RAID5小块写操作中读、修改、写的复杂过程。

由于磁盘服务的非抢占性，用户读请求和写回指令不可避免地会有所冲突。为了避免该冲突，对回写算法实现中增加了线性阈值调度策略。其基本思想是：写回的速度与当前写Cache的空间使用率成正比，当其中的数据增加时逐渐加快写回的执行速度。当其中的数据量减少时放慢写回的执行速度，即写回的执行速度是写Cache空间使用率的线性函数。阈值的作用是减少写回和读操作之间的冲突，它定义了在进行写回时系统读请求的等待时间的上限。在阈值的设定上要考虑平衡的问题:如果阈值定得太低，则写回执行得太慢，会导致写Cache溢出；相反，如果阈值定得太高，写回执行得太快，在减少了系统读请求等待时间的同时也降低了写Cache的命中率。本文通过反复试验，采用不同负载的动态调整阈值方式，使写回算法即实现了对读请求服务透明性的要求，又保证在突发大量写请求情况下不引起缓存溢出。

3.2.3 数据和校验使用独立Cache

一般概念的Cache专指数据Cache，即Cache中仅仅存放数据而不包括校验块。这将导致RAID5性能受限。因为Cache中修改过的数据如需要写入磁盘，必须要读出与该块同属一个数据条的其它盘上的数据块，与该块的数据作异或运算，得到的新校验值，然后才能写回数据和新的校验块。这样的一次回写。仍然需要多次磁盘访问，也使得这时的回写Cache操作需要相当长的时间。本文提出了分别保存数据和校验的Cache的策略。根据异或运算的特点只需计算修改后数据、原始数据、原始校验和的异或值得到便是最终校验数据。此外校验部分Cache并不是越大越好，因为增大Cache空间同时意味着在同一管理策略条件下其命中率的相对降低。校验Cache最终大小要由RAID5系统磁盘数以及高速缓存大小共同决定。经过测试本系统中校验块Cache大小占读Cache总量的20%性能达到最优。虽然该方式导致保存数据的Cache空间减少，但是随着集成电路的发展相关芯片的容量不断上升而价格不断下降，用适当选取部分作为校验使用，也是提高写入性能的一个方法。

## 4 小结

本文基于Intel 80321 I/O处理器在MontaVista嵌入式Linux内存管理的基础上根据高速缓存读写策略进行了优化，采用bonnie测试了块操作的方式下顺序读、顺序写和随机定位3种模式数据传输速率和CPU占用率，其中磁盘用5个相同的Maxtor 6Y080M0，文件系统采用EXT2，结果均值如表1所示。

**表1 优化后磁盘性能结果**

| | 顺序写 | | 顺序读 | | 随机读写 | |
|---|---|---|---|---|---|---|
| | M/s | CPU(%) | M/s | CPU(%) | M/s | CPU(%) |
| 改进前RAID5 | 60.2 | 25.8 | 82.0 | 12.4 | 66.8 | 18.2 |
| 改进后RAID5 | 90.5 | 9.2 | 120.0 | 6.4 | 98.5 | 5.0 |
| 单磁盘 | 45.1 | 3.5 | 46.8 | 2.5 | 38.0 | 5.5 |

由表1可知，直接使用RAID技术，磁盘的读写性能虽然有了一定的提高，但是CPU占用率很高。通过充分利用I/O处理器的应用加速单元及高效读写高速缓存的管理策略的改进之后，CPU占用率在尽量减小系统影响的前提下较改进前已有明显减少，磁盘的性能得以进一步提高。系统性能的提高不仅仅取决于磁盘子系统，处理器的优化与文件系统等其他因素密切相关，磁盘读写性能没有达到理想情况下的性能。下一步工作将针对特定的应用，改进文件系统以便达到最优性能。

## 参考文献


1 Intel IQ80321 I/O Processor Evaluation Platform Board Manual[Z]. 2003.

2 Vadala D. Managing RAID on Linux[M]. O'Reilly Media, 2003.

3 MontaVista Software Inc. MontaVista Linux Professional(3.1) [EB/OL]. 2004. http://www.mvista.com/dswp/ds_pro3_1.pdf.