doi:10.3772/j.issn.1002-0470.2018.06.002

多核处理器系统 I/O 访存优化研究①

李 鹏②[*,**,***] 曾 露[*,**,***] 王焕东[****]

([*]计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

([**]中国科学院计算技术研究所 北京 100190)

([***]中国科学院大学 北京 100049)

([****]龙芯中科技术有限公司 北京 100195)

摘 要 本文提出了一种多核处理器自适应 I/O 直接缓存访问(ADCA)的方法以提升I/O访存的性能,降低对其他程序的影响。与传统直接缓存访问(DCA)不同的是,该方法利用了 LRU 栈特性,通过采样辅助标签目录的方式动态调整 DCA 可使用的 cache 空间,同时对 I/O 数据的替换和写内存策略进行优化。实验结果表明,与 DCA 方式相比,该方式使得 I/O 带宽提升了大约 10%,而与 SPEC 和采用直接内存访问(DMA)方式的网络测试程序同时运行相比,SPEC 定点和浮点性能分别提升了11.5%和8.9%。

关键词 直接缓存访问(DCA), LRU 栈特性, 自适应, 伪划分, 优先替换

0 引 言

长期以来,随着摩尔定律的发展,微处理器的集成度越来越高,于是更多的高速 I/O 控制单元和接口被集成到处理器芯片中,这使得处理器的计算、存储等需求和 I/O 性能的耦合度越来越密切相关。于是各种高性能的互联总线相继出现,比如 HT、PCI-E、QPI 以及 40G 和 100G 以太网接口等,这些总线的传输带宽能够达到数十 GB/s,几乎和内存的带宽相当,于是 I/O 控制器的访存性能变得越来越关键,其对整个系统性能有着至关重要的影响。

由于总线的传输速度已经足够快,但是处理器如何高效地处理 I/O 数据搬运、减少开销却并不容易。I/O 访存的多样性是造成性能瓶颈的一个重要原因,比如并行科学计算应用多是周期性猝发的I/O访问,数据量变化幅度很大,访存模式也包括顺序、交错和混合等多种类型。高并发的网络服务器,需要处理大量的并发请求,对处理器性能和网络I/O性能的要求都很大。特别是随着多核处理器成为主流,I/O 性能瓶颈问题变得更加突出。由于多核处理器显著提高了线程的并行度,提高了处理器的计算能力,使得 I/O 性能与处理器计算能力变得更加不平衡。更为重要的是,多核处理器中的 cache 和访存等资源被多个线程共享,也加剧了 I/O 和其他并行执行的程序之间的冲突,导致整体性能严重下降。

因此,在提供系统高 I/O 性能的同时,降低 I/O 对处理器性能的影响,从而更好地平衡 I/O 和其他应用之间的冲突,提高系统整体的性能成为一个热点问题。本文在传统的 I/O 直接缓存访问的基础上提出了一种自适应的 I/O 直接缓存访问(adaptive direct cache access, ADCA)方法,该方法相比已有的直接缓存访问(direct cache access, DCA)和直接内存访问(direct memory access, DMA)方式,可以更加合理地利用处理器资源,提高系统的整体性能。

① 国家"核高基"科技重大专项课题(2009ZX01028-002-003, 2009ZX01029-001-003, 2014ZX01020201, 2014ZX01030101),国家自然科学基金(61521092, 61232009, 61222204, 61432016)和中国科学院重点部署(ZDRW-XH-2017-1)资助项目。

② 男,1986 年生,博士生;研究方向:计算机系统结构;联系人,E-mail: lipeng-cpu@ict.ac.cn

(收稿日期:2018-01-30)

1　相关工作

I/O 访存通常会采用两种方式:传统的 DMA 方式和新型的 DCA[1]方式。

DMA 方式中,CPU 将 DMA 命令(描述符)写入内存中,随后通知 I/O 设备读取该命令,外设通过该命令指定的地址和数据大小进行数据搬运,并在完成时通知 CPU。由于实际数据搬运过程无需 CPU 参与,只需要 CPU 进行开始的任务分配和最后的数据处理,所以大大提升了整体性能。但是由于存在多层次的存储结构(内存和多级 cache),这也会导致数据一致性的问题。解决的办法一般有两种:一种是采用硬件的 I/O 侦听 cache 的方式,当 I/O 访问内存时同时向 cache 发出侦听访问,当 cache 中存在该地址的数据时进行无效或写回。另一种是采用软件维护一致性的方法,在进行 DMA 之前由软件对 cache 进行无效或写回操作。

随着高速 I/O 总线的发展,传统 DMA 方式带来的访存延迟已经严重制约了处理器充分利用高速总线带来的优势,于是让高速 I/O 设备直接访问缓存的 DCA 方式被 Intel 提出[1-3],并应用于其网络处理芯片中。文献[1]首次提出将网络接收数据直接写入 cache 中来减少随后处理器访问网络数据的延迟,作者通过分析不同大小的网络包采用 DCA 方式对访存带宽的降低和 CPU 利用率的降低说明 DCA 的有效性。实验表明,越小的网络包,NIC 写和 CPU 读的间隔时间越小,DCA 方式越合适。Cache 容量越大,DCA 方式越合适,性能提升越明显。文献[2]进一步通过实验验证同时运行 SEPC2000 和网络传输情况下是否开启 DCA 对 SEPC2000 的影响,结果表明开启 DCA 几乎对所有 SEPC2000 程序都有不同程度的性能提升。文献[3]针对之前仅在单核处理器进行实验的局限性,进一步在多核处理器以及 NUMA 结构上测试 DCA 的影响。实验表明,SPEC2000 和网络传输同时在两个核分别运行时性能都比单独运行时下降,这主要是由于对 cache 的竞争导致的。相对于传统 DMA 方式,DCA 的优势是利用 cache 降低 CPU 和 I/O 设备的访存延迟且无需软件维护数据一致性。但是文献[4]也指出,盲目进行 DCA 可能会导致性能变差,因为大量 I/O 数据集中到有限空间的 cache 中,如果不能被及时使用,会造成严重的 cache 污染。因此是否进行 DCA 取决于数据是否及时使用、数据量的大小以及数据使用的方式等。针对该问题,文献[5]提出了一种 DMA cache 的方法,并提出了两种实现方案:分离的 DMA 缓存(decoupled DMA cache,DDC)和基于划分的 DMA 缓存(partition-based DMA cache,PBDC),区别就是前者通过增加片内 cache 的方法区分 I/O 数据和 CPU 数据,后者则通过划分 cache 的某些路单独给 DMA 使用达到类似的效果,但都需要对 cache 结构和一致性协议进行较为明显的改动,实现的复杂度较高。文献[6]将网络适配器集成到芯片内部,进一步减少网络数据的传输延迟,同时结合 DCA 进一步提升了网络处理的性能。文献[7]根据重用距离将 I/O 数据和命令分开处理,对与 CPU 交互密切的描述符命令类请求采用 DCA 方式,而对重用距离较长的数据类请求采用 DMA 方式。但是由于描述符相比数据来说在 I/O 传输过程中占得比例很小,该方式对性能提升的效果也比较有限。

文献[1-4]都提及通过限制 DCA 可使用的 cache 空间可以减少对 cache 的污染问题,但也仅仅是限定某些路给 I/O 数据使用,并没有具体实现方案。文献[5]虽然实现了一个基于划分的 PBDC 方案,但也仅仅是一个静态划分方案,不具有实时调整性。由于 I/O 访存多样性,分配过少的 cache 可能会使 I/O 占用的 cache 空间不足,导致 cache 中的 I/O数据还未被使用就发生替换,导致性能下降得更加严重。而分配过多的 cache 空间又会影响其他程序的性能,由于 I/O 传输一般具有周期性和猝发性,当 I/O 相对空闲时分配给它的 cache 空间应该能够被其他程序利用才能达到性能的最大化,所以单独为 I/O 增加 cache 空间或者静态划分 cache 的方式并不合适。

综上所述,为 I/O 数据的 DCA 方式找到一种合适的 cache 管理策略成为一个亟待解决的问题。一直以来,cache 的管理策略研究是一个热点问题,主要是通过研究 cache 的划分、插入和替换等策略来

达到最大吞吐率、提高公平性和服务质量。文献[8]提出了一种基于效用的 cache 划分(utility-based cache partitioning,UCP),通过为每个处理器核设置一个效用监控器(utility monitor,UMON),利用 LRU 替换算法的栈间距[9]特性来记录每个位置的命中计数,从而找到使得 cache 失效率最低的划分方法。文献[10]发现将第一次插入的块放在 LRU 位置可以有效防止 cache 颠簸,于是提出了双端插入策略(bimodal insertion policy,BIP),可以以一定概率将大部分新插入块放置在 LRU 位置,为了减少对 LRU 友好的程序产生不利影响,又提出了动态插入策略(dynamic insertion policy,DIP),该策略可以在 BIP 和传统 LRU 策略中间进行切换。文献[11]在文献[10]的基础上进一步提出了线程感知的动态插入策略(thread aware dynamic insertion policy,TADIP),对不同线程应用不同的动态插入策略。文献[12]综合了插入和提升策略,提出了一种插入提升伪划分策略(promotion/insertion pseudo-partitioning,PIPP),该策略为不同 cache 分区设置不同的插入提升策略,来自不同分区的块插入到不同优先级的位置,当发生命中时仅仅将命中块向 MRU 位置移动一小步。

这些方案大都是针对不同处理器核或者线程进行的 cache 优化策略,而对于与一般处理器运行程序的行为具有很大差异性的 I/O 指令和数据如何进行 cache 管理,则鲜有提及。

本文是在 DCA 的基础上对 I/O 数据在 cache 中如何分配和替换提出了一种更为完善的方案,该方案通过动态调整 I/O 数据在 cache 中占用的空间和替换策略,进一步提升了多核处理器的整体性能,提升了 cache 的利用率。

2 Cache 行为分析

2.1 LRU 栈特性

很多关于 cache 分区的理论基础都或多或少地利用了 cache 的栈特性。所谓 cache 的栈特性,就是指由 LRU 策略管理的 cache 中,在组数量不变的情况下,如果一个访问在包含 N 路的 cache 中命中,则该访问也会在大于 N 路的 cache 中命中。利用该特性,可以用一个 N 路的访问命中计数器,来统计 cache 路数从 1 到 N 的命中信息,从而对某个线程分配的 cache 路数进行决策,达到最大命中率。

本文以一个 4 路组相连 cache 为例,简单介绍如何利用 cache 的栈特性来统计命中信息,从而对 cache 分配进行决策。

如图 1 所示,cache 中每一组都对应有 4 个命中计数器,分别对应每个 cache 块,计数器的编号对应于 cache 块在 LRU 表中的位置,从 MRU 到 LRU 对应的位置我们依次称之为位置 0 ~ 3,对应的计数器编号分别为 0 ~ 3。

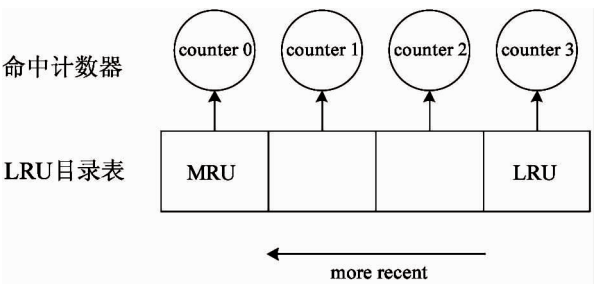

图 1 不同 LRU 位置命中计数器示例

当一个访问发生 cache 命中时,根据命中块在 LRU 表中的位置,对应的计数器加 1。表 1 给出了一个命中计数器统计信息的示例,该示例统计了 100 次 cache 访问,其中有 35 次在 MRU 位置命中,有 22 次在位置 1 命中,有 12 次在位置 2 命中,有 7 次在 LRU 位置命中,还有 24 次 cache 缺失。根据这个统计信息,可以知道,当 cache 的路数由 4 路减少为 2 路时,cache 的命中次数将下降为 57(35 + 22)次,因此可以根据一个 cache 路数较大的命中统计信息计算出任何小于该路数的命中信息。

表 1 命中计数器统计示例

计数器	统计值
counter0	35
counter1	22
counter2	12
counter3	7
misses	24

从该命中统计信息中,还可以看出,越是靠近 MRU 位置的 cache 块,其发生命中的次数越多,反之越是靠近 LRU 位置,发生的命中次数越少。也就是说,大部分程序的 cache 命中位置都倾向于最近刚刚访问过的 cache 块,越是长时间没有访问的 cache 块,其再次发生命中的可能性越小。

由此可以推断,当 cache 路数较多时,在一小段有限的时间内,大部分的访问都会命中 cache 中靠近 MRU 位置的块,而对于靠近 LRU 位置的块,则很少发生访问。可以充分利用这一特性,当高速 I/O 数据相对于 LRU 位置的 CPU 数据具有较小的重用距离(见 2.2 节)时,可以将 I/O 数据直接写入 cache 从而达到更好的整体性能,达到硬件利用率的最大化。

2.2　I/O 数据的访存特性

数据的重用距离(也叫 LRU 栈间距)定义为对同一个数据发生两次访问的时间间隔,一般用处理器拍数或者中间访问的 cache 块数来衡量。I/O 数据具有明显的生产者消费者特性,也就是 CPU 写 I/O读或者 I/O 写 CPU 读。而对于 I/O 写 CPU 读的数据一般来说具有更小的重用距离,甚至比一些 CPU 数据的重用距离还要小,采用 DCA 方式更有合理性。

I/O 数据具有更好的流特性,也就是空间局部性较好,这意味着由多路组相连组织的 cache 结构可以发挥更大的空间利用率,因为 I/O 一般会发出地址连续的访存访问,使得 cache 的每个组都接受数目大致相当的 I/O 请求。这一特性也使得如果对 I/O 数据进行类似按路的 cache 划分也更有效,可以比对 CPU 线程的 cache 划分获得更大的收益,也避免了采用哈希索引寻址等更复杂的 cache 划分替换算法。

I/O 数据一般也只会被读取一次,之后就会被释放,因此对其采用传统的 LRU 替换算法并不合适,因为当 I/O 数据被读取后 LRU 策略会将其放置于 MRU 位置,而由于一般 I/O 数据被访问后直至被替换都不会再发生访问,这样使得该 I/O 数据从 MRU 位置移至 LRU 位置直至被替换,会浪费大量的 cache 空间,所以对 I/O 数据采用合适的替换策略也是很有必要的。

3　自适应直接缓存访问

3.1　自适应的 DCA 伪划分策略

根据以上的分析,需要为多核系统下 I/O 数据的 DCA 分配合适大小的 cache 空间。该空间既不能严重影响处理器核运行的其他线程,也要充分发挥高速 I/O 的性能,使 cache 的利用率最大化,达到整体性能最优。

对分配给 I/O 数据的 cache 空间以路为单位,为了实时监测分配多少路空间合适,本文利用辅助标签目录来统计一个 I/O 数据从写入 cache 到被读取的这段时间其他多核线程对 cache 的利用情况。辅助标签目录和 cache 中的标签目录具有相同结构,也采用 LRU 策略,区别是该标签只跟踪 CPU 访存请求在 cache 中的命中情况,对 I/O 访存请求直接忽略掉,这样可以用该标签目录模拟 CPU 程序在 cache 中的行为。

尽管如此,设置一个与 cache 中完全一样的标签目录硬件开销还是非常大的,因为这相当于将 cache 中的标签完全复制一份。为了节省硬件开销,本文采用了采样的方式,只对某些 cache 组进行跟踪采样,已有工作[9]表明,只对某些组进行采样也可以达到很高的准确度。比如对于一个 1024 组的 cache,只对其中的 16 组进行跟踪采样,这大大降低了硬件开销。而由于 I/O 数据的地址连续特性,只要合理设置采样的组(比如上面的例子每隔 64 组进行采样),基本上可以使得 I/O 访存请求落在采样空间内。

可以利用辅助标签目录来为 DCA 划分提供支持,为此本文为每个采样组增加了一个监测寄存器,该寄存器的字段定义如图 2 所示。

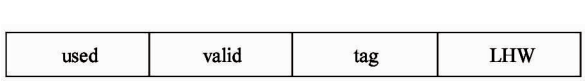

图 2　监测寄存器的字段定义

该监测寄存器的主要作用是记录一个 I/O 数据从写入 cache 到被读取的这段时间其他 CPU 线程对

cache 的利用情况。其中 LHW(lru hit way)用来表示这段时间其他 CPU 线程访存命中 cache 中的 CPU 数据时,该命中位置中最靠近 LRU 位置的路数。假定一个 W 路组相连 cache,如果在某段时间统计到的 LHW 的值为 $N(N \leqslant W)$,根据 LRU 栈特性,如果在这段时间只分配给 CPU 数据 N 路 cache 则命中率不会受到影响。我们之所以只记录 LHW 的值而没有统计不同 LRU 位置的命中数,是因为利用了 LRU 栈特性,假定如果第 N 路的数据发生命中,则 N 路之前的 CPU 数据也很大概率已经发生过命中,因此只需以最保守的方式给出 I/O 数据能使用的路数($W-N$),这样对其他线程的影响最小。tag 位的定义与 cache 完全一致,用来表示待监测的 I/O 数据的标签。used 位用来表示该监测寄存器是否正在被使用,也就是是否有 I/O 数据正在被监测,valid 位表示监测是否结束,结束的条件是该寄存器的 I/O 数据被访问,如果结束,则输出 LHW 的值,同时复位 used 和 valid 标志位,表明该寄存器可以用来监测下一个 I/O 请求,如此循环。由此可以设计出一个动态 LHW 生成算法,该算法流程图如图 3 所示。

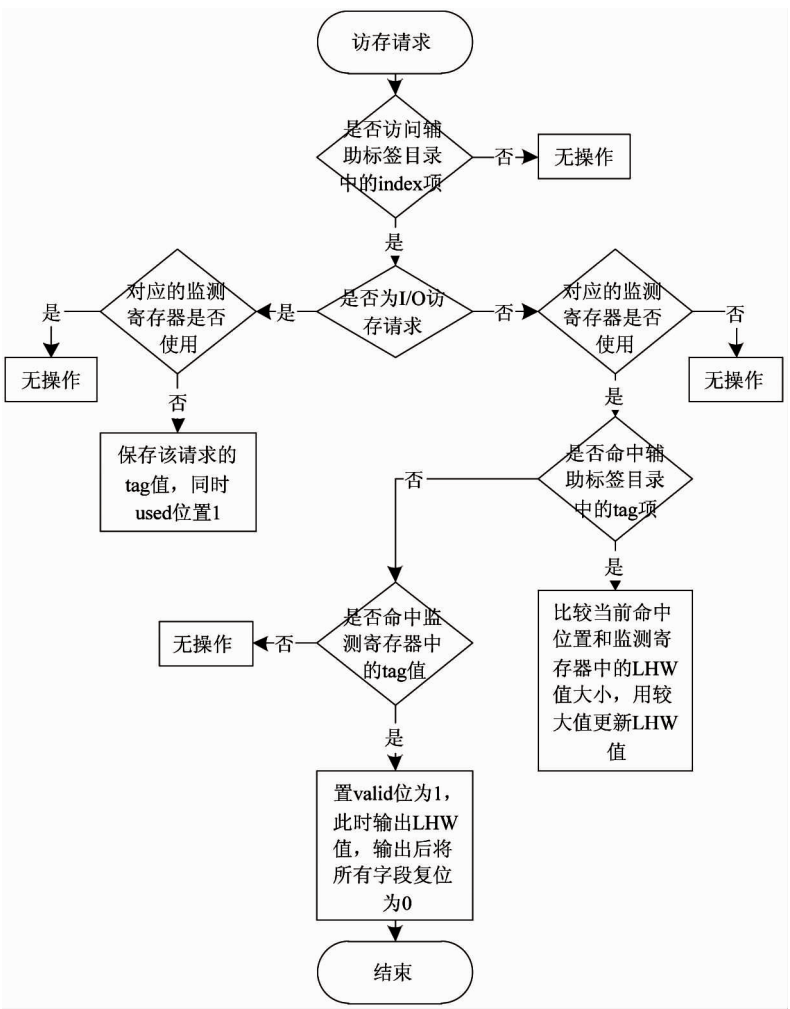

图 3　动态 LHW 生成算法流程图

通过该流程输出的 LHW 的值即可得到 I/O 数据 DCA 方式可以占据的 cache 路数。将 DCA 的路数记为 P,输出的 LHW 值为 N,总的 cache 路数为 W,则 $P = W - N$(P 的变化范围即为 $0 \sim W$)。由于每个监测寄存器都会输出一个 LHW 值,而不同寄存器输出的值很可能并不一定相同,因此需要选择合适的值。为此本文可以为 I/O 数据的 DCA 方式选择不同的策略,有激进型(选取 P 的最大值)、保

守型(选取 P 的最小值)和正常型(选取 P 的平均值)。激进型更有利于发挥高速 I/O 的性能,但是也会影响其他线程的运行,保守型则正好相反,而正常型则兼顾了二者的公平性。由于 I/O 数据的流特性,实际情况 3 种类型对应的 P 值差别并不大,可以通过采用软件配置的方式对 3 种类型进行选择切换。方法就是首先将不同监测寄存器输出的 P 值翻译成独热码,独热码的位数等于 cache 路数,然后增加一个位数与 cache 路数相同的寄存器,该寄存器保存的是所有独热码 P 值按位或的结果,该结果中最左边 1 所在位置即为 P 的最小值,最右边 1 所在的位置即为 P 的最大值,而 P 的平均值只需要通过简单将最大值和最小值求平均即可大致估计。该寄存器会输出 P 的 3 个结果,然后通过软件配置选择其中某个值即可达到 DCA 3 种不同的策略。由于激进型会伤害处理器其他线程的性能,而保守型则不利于 I/O 数据充分利用 cache 空间,在实际中可以根据性能需求侧重点来选择不同的策略。为了系统整体性能最大化和公平性,后面实验中当有其他处理器线程运行时都采用正常型。

通过得到的 P 值,可以设计出自适应的 DCA 伪划分策略,不过在此之前,需要对 cache 中原有的 tag 做一点修改。为了区别 I/O 数据和 CPU 数据,本文对 cache 中原有的 tag 新增了两个标识位,如图 4所示。

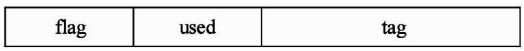

图 4　新增 tag 寄存器的字段定义

其中 flag 位用来区分 CPU 数据和 I/O 数据,flag 为 1 表示 I/O 数据,为 0 表示 CPU 数据。used 位表示该 I/O 数据是否被访问,为 1 表示已经被访问,为 0 表示未被访问。

本文以 I/O 写 CPU 读为例介绍该策略,该策略的流程图如图 5 所示。

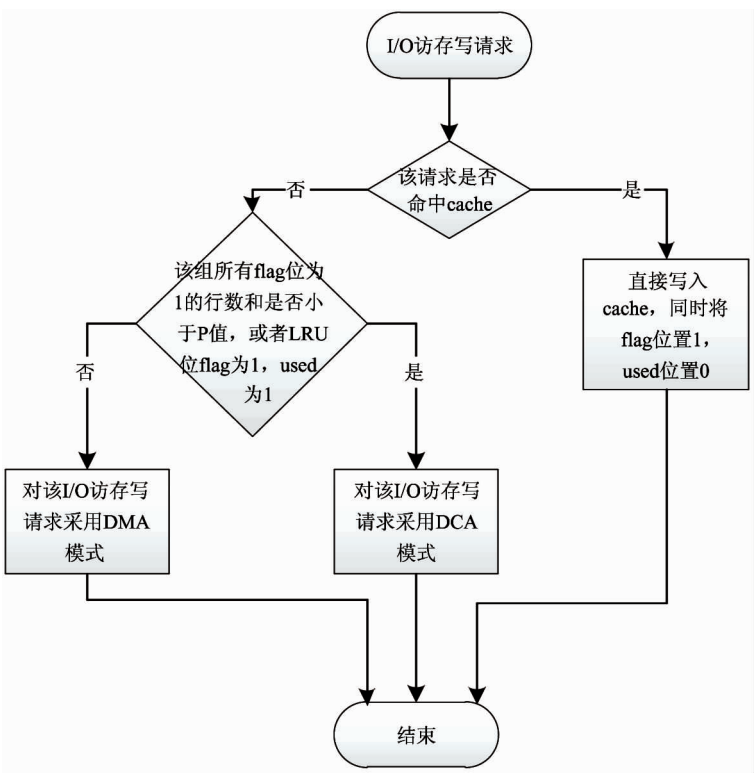

图 5　自适应的 DCA 伪划分策略流程图

图 5 的流程图介绍了该策略,总的来说就是当 I/O 写请求命中 cache 时直接写 cache,而不命中时则根据 I/O 数据是否已经超过允许占有的路数决定是否采用 DCA 模式。

之所以称为自适应的,是因为该策略可以根据 CPU 和 I/O 实时运行的情况动态调整合适的 P 值,当 CPU 相对 I/O 需要更多的 cache 空间时,P 值会自动减小从而通过限制 DCA 来提高 CPU 线程的性能,反之亦然。

之所以称为伪划分策略,是因为该策略并没有指定某些路必须为 I/O 数据所用,而仅仅限定了使用的最大路数,当 I/O 相对空闲或者并没有占用一个完整的路时,这些未使用的空间都可以被 CPU 使用,从而达到 cache 利用的最大化,因此该策略并非严格意义上的 cache 划分。

3.2 自适应的 DCA 替换策略

前面已提到,很多 I/O 数据从写入到读取只会被使用一次,因此传统的 LRU 替换算法并不合适。比较直观的是对 I/O 数据采用读取后优先替换策略,即将使用后的 I/O 数据直接置于 LRU 位置,但是这样简单的策略很可能并不适合所有的 I/O 数据,比如网络包会存在修改包头后转发的行为,这样的行为使得有些 I/O 数据会产生不止一次访问。考虑到 I/O 应用行为的多样性,为其设计一种自适应的替换策略也是很有必要的。

本文设计了一种能够在传统 LRU 替换和优先替换策略之间动态切换的策略。该策略通过饱和计数器统计 I/O 数据被访问的次数,通过统计结果来决定使用哪种策略。

下面以 I/O 写 CPU 读为例介绍该策略。开始时并不清楚 I/O 数据的行为,因此对 DCA 的 I/O 数据采用传统的 LRU 替换方式,然后通过饱和计数器对 I/O 数据的行为进行统计。饱和计数器是一个 N 位的计数器,初始值为全 0,利用新增 tag 寄存器字段进行计数,将 used 位扩展为 2 位,低位含义未变,当发生第二次访问时高位置 1。当一个 I/O 数据所在的 cache 行(flag 位为 1)被替换时,如果 used 高位为 0(该行直到替换都未发生二次访问),则饱和计数器加 1,当一个 I/O 数据所在的 cache 行被访问时,如果 used 值为 01,则将其改为 11(该行发生了两次访问),同时饱和计数器的值减 1。当饱和计数器变为全 1 时,将 I/O 数据的替换策略变更为优先替换。

而对于如何由优先替换变更为 LRU 替换,则需要增加一个 3.1 节中提到的辅助标签目录,该标签目录中的 DCA 替换策略一直保持 LRU 替换,其他字段定义与 cache 中的 tag 完全保持一致,当采用优先替换策略时,饱和计数器对该辅助标签目录进行跟踪,计数器增减的条件与上面完全一致,当饱和计数器的值变为全 0 时,将 I/O 数据的替换策略变更为 LRU 替换策略。为了减少硬件开销,该辅助标签目录同样采用采样的方式,只设置有限的几组即可达到较高的准确度。

3.3 自适应的 DCA 写内存策略

内存地址一般会被组织为堆地址、行地址和列地址的三维结构,由于其物理特性,连续访问同一行地址效率最高,可以很好地减少内存的行冲突,对外表现就是连续地址的读写访存带宽较高,延迟也更小。当高速 I/O 采用 DMA 方式进行访存时,由于 I/O 数据的连续性,可以充分利用这一优势。但是当采用 DCA 方式时,如何利用这一优势则需要进行分析优化。

写策略一般分为写直达和写回两种,写直达就是写 cache 时同时写内存,而写回则是只有当 cache 替换时才写内存。为了减少频繁的内存读写,现代微处理器大都采用写回的方式,这对于空间局部性不是很强而且被反复使用的 CPU 数据是合适的。但是如果将同样的方式用于 I/O 数据,可能会大大降低 DCA 方式带来的好处。

考虑这样一种情况,采用 DCA 方式的 I/O 写请求地址连续并且数据仅仅被 CPU 读取使用一次,这样的系统稳定运行后会采用本文前面提到的优先替换策略。但是由于 cache 的过滤性,不同组内连续地址的 I/O 数据被替换的时机并不一定连续,这就造成了本来连续地址的写被分成了许多不同地址的写,这些写请求在不同的时间被写回内存,造成内存严重的行冲突。而如果采用写直达策略,将采用 DCA 方式的 I/O 写 cache 的同时也写入内存,由于该数据随后并不会发生写操作,因此替换时可以直接从 cache 中删除而无需再写回内存,大大提高了访存效率。

基于这种考虑,本文将 DCA 替换策略和写内存

策略结合起来综合考虑,设计了一种自适应的 DCA 写内存策略。该策略根据 3.2 节中的替换策略调整写内存策略,当替换策略为 LRU 替换时,采用写回策略;当替换策略为优先替换时,采用写直达策略。

4　实验平台和实验结果

4.1　实验平台

本文采用 EVE 仿真加速器[13]对修改后的龙芯 2K1000 处理器 RTL 代码进行仿真。EVE 仿真加速器是一款基于 FPGA 的仿真加速器平台,可以周期级精确地实现处理器的功能,并提供有调试接口和验证工具集成环境,是处理器流片前验证的重要工具。

龙芯 2K1000 处理器是 2017 年发布的龙芯 2 号处理器系列的最新产品,主要面向于网络应用。该处理器的配置如表 2 所示。

龙芯 2K1000 处理器集成了 2 个千兆网接口[14](GMAC-IP),我们通过修改该 IP 使得 I/O 描述符和数据具有不同的 ID,对描述符固定使用 DCA 策略,对数据使用 ADCA 策略。本实验在 EVE 上将两个处理器通过 GMAC 连接,然后通过处理器的两个核分别运行 Netperf 基准测试程序[15]和 SPEC CPU2000 测试程序来进行实验。Netperf 是由惠普公司开发的测试网络栈,是一种测试不同类型的网络性能的 benchmark 工具。

表 2　龙芯 2K1000 处理器参数

参数	配置
处理器核	64 位双发射,800MHz,双核
指令集	MIPS64
L1 缓存	私有,32kB 数据,32kB 指令
L2 缓存	共享,1MB,16 路组相连
缓存一致性	MESI 协议,位向量目录
片上网络	交叉开关
内存	DDR3,533MHz

4.2　实验结果

本实验通过单独运行 Netperf 基准测试程序,测试不同包大小情况下的带宽,然后测试同时运行 SPEC CPU2000 测试程序下分别采用 DMA、DCA 和 ADCA 情况下对带宽的影响,结果如图 6 所示。

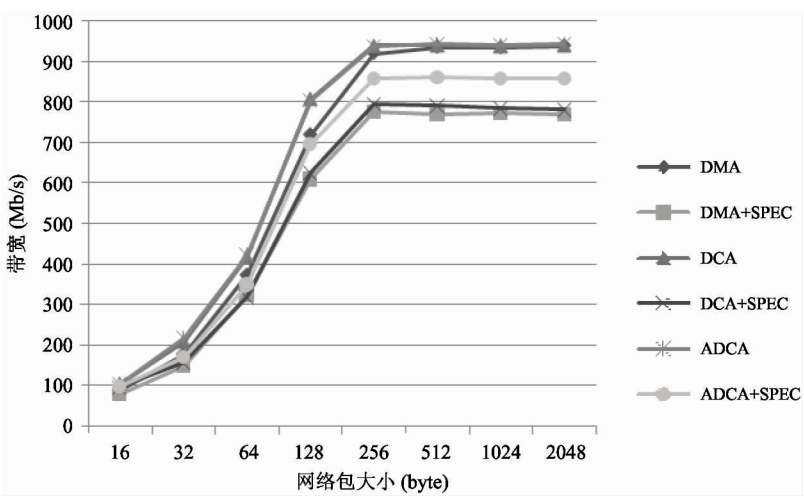

图 6　各种组合情况下的网络带宽

从图中可以看出,当不存在 SPEC 程序干扰情况下,ADCA 方式和 DCA 方式达到的带宽基本一致,从图中基本看不出区别,这主要是因为当处理器仅仅运行 Netperf 基准测试程序时,cache 空间相对于网络数据来说比较充足,所以此时大部分网络数据都是直接写入 cache 中,ADCA 方式退化为 DCA 方式。而采用 ADCA 方式或 DCA 方式比 DMA 方式大概有 10% ~15% 的性能提升,尤其是在网络包小于 256byte 时比较明显,当网络包大于 512byte 时,受限于网络硬件瓶颈,都可以达到 940Mb/s 左右的

带宽。当存在 SPEC 程序干扰时,两种方式的带宽都有明显下降,其中 DMA 方式带宽下降了 20% 左右,而 DCA 方式带宽下降更严重一些,达到 25% 左右,此时 DCA 与 DMA 方式的带宽基本相同,已经体现不出 DCA 方式的优势,原因主要是 SPEC 与 DCA 竞争 cache 导致 DCA 写入的数据未被使用而发生替换。而当采用 ADCA 方式时,相比 DCA 方式,带宽提高了 10% 左右。

单独运行 SPEC CPU2000 测试程序,测试不同程序的分数,然后同时运行 Netperf 基准测试程序,测试采用 DMA、DCA 和 ADCA 方式对 SPEC CPU2000 不同程序的影响。测试中采用的网络包大小为 512byte,为的是使得网络接近最大带宽,同时以 DMA 方式下 SPEC 的性能为基准,测量其他方式相对于该方式性能增加或者降低的比例,结果如图 7 和图 8 所示。

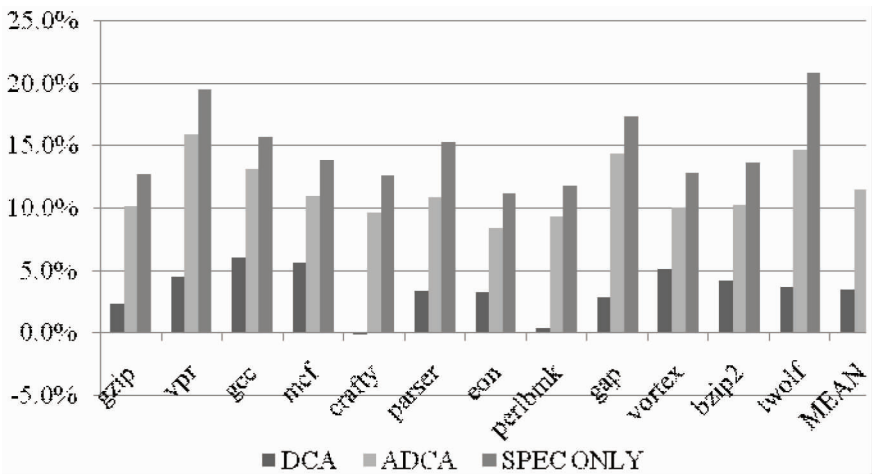

图 7　不同方式下 SPEC 定点程序性能

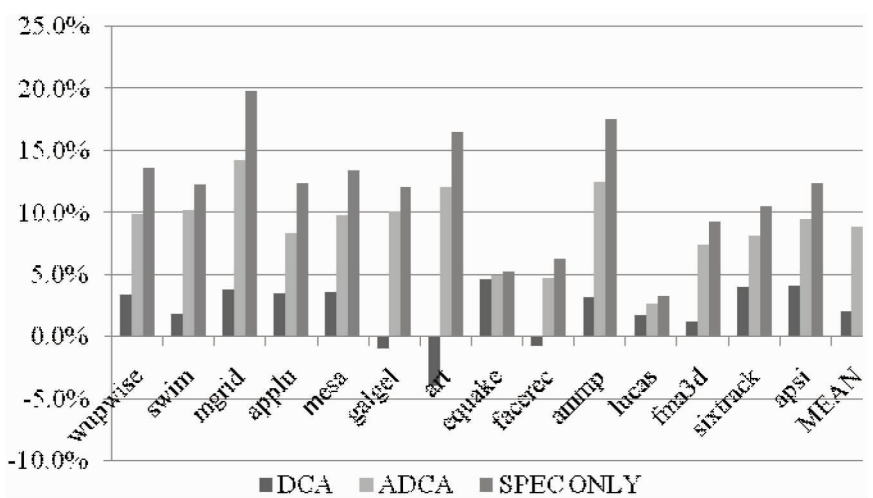

图 8　不同方式下 SPEC 浮点程序性能

从图中可以看出,相对于采用 DMA 方式下同时运行 Netperf 和 SPEC,单独运行 SPEC 定点程序性能和浮点程序性能分别提升了14.8% 和11.8%,这说明 Netperf 测试程序对 SPEC 的性能影响是比较明显的。当采用 DCA 方式时,大部分的程序性能相比较 DMA 方式出现小幅度的提升,也有一些出现下降,总体性能定点提高了3.4%,浮点提高了2.1%,这可能是因为网络处理程序对于写入 cache 的数据读取比较及时,使得 DCA 写入的数据大部分在替换前被使用,没有造成严重的 cache 污染。可以预计,当网络包更大或者处理不够及时时,DCA 方式很可能对 SPEC 程序性能造成不利影响。当采

用 ADCA 方式时,相比较 DMA 方式,定点性能和浮点性能分别提升了11.5%和8.9%,更加接近单独运行 SPEC 时的性能。但是无论采用何种方式,均无法达到单独运行 SPEC 的性能,这主要是因为 DCA 方式通过盲目抢占 cache 来提高 I/O 性能,这样挤占了 CPU 其他线程的 cache 空间,从而影响了其他线程的性能。而 ADCA 方式则是通过更合理的分配方式来达到 cache 利用率的最大化,其原理本质上是将 I/O 数据和 CPU 数据公平看待,将重用距离更小的数据存入 cache 中,从而达到整体性能的最优,该方式也会占用其他 CPU 线程的 cache 空间,所以也会造成其他线程性能下降。

I/O 访存延迟也是一个重要的性能指标,由于 ADCA 方式主要针对的是 I/O 数据写 cache 的情况,而对于 I/O 读请求,当其在 cache 中不命中时不会为其在 cache 中分配空间,所以该方式并不会优化 I/O 读请求的延迟。Netperf 基准测试程序 request/response 模式可以用来测试请求应答的响应时间,但是该过程综合了 I/O 读取写入时间以及传输路上的延迟等。为了获得比较纯净的 I/O 写入时间,本实验在 GMAC-IP 上增加了一个写响应统计模块,该模块通过统计发出写请求到收到响应经历的时间(时钟拍数计数)来计算写延迟。本文采用了和带宽测试相同的 6 种情况,但是由于延迟和数据包的大小直接相关,为了更好地显示出不同情况的延迟对比,以 Netperf 采用 DMA 方式运行时的写延迟为基准,测试其他情况下节省时间百分比,结果如图 9 所示。本实验又测试了开启 SPEC 程序的影响,同样以 DMA 方式为基准,结果如图 10 所示。

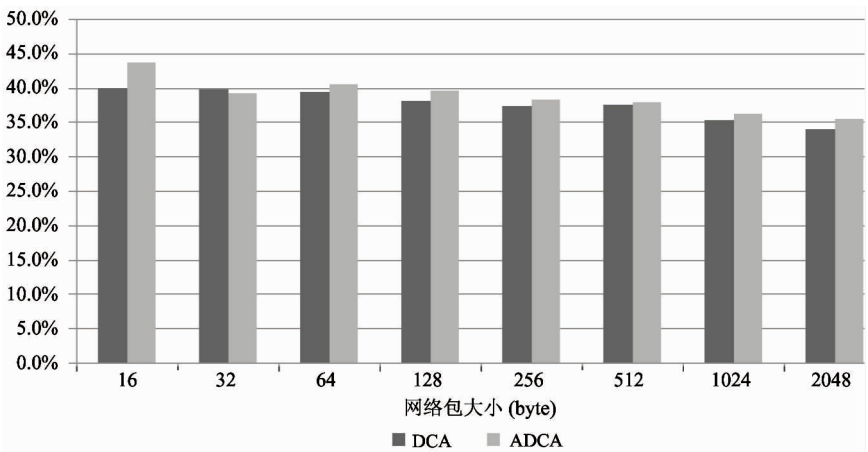

图 9　相对 DMA 方式延迟降低比例

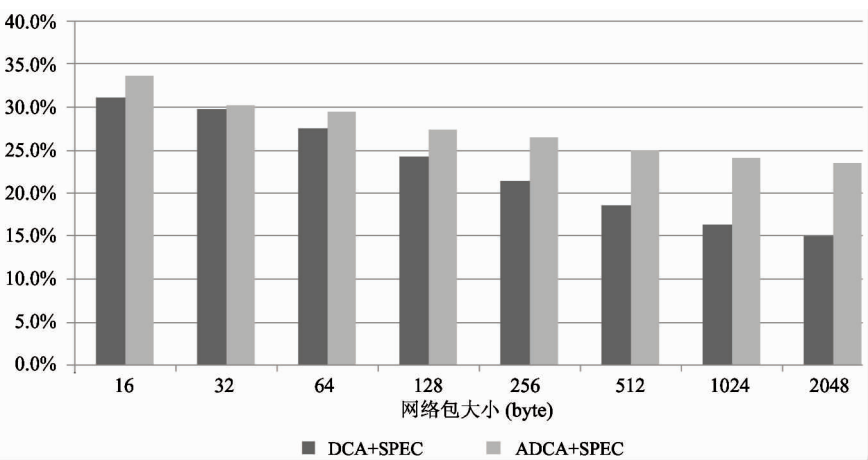

图 10　相对 DMA + SPEC 方式延迟降低比例

从图 9 可以看出,单独运行 Netperf 程序时,DCA 和 ADCA 方式相对于 DMA 方式的写延迟均有较大幅度的降低,两者降低的比例几乎相同,这主要是因为单独运行 Netperf 时 cache 空间相对比较充足,所以 I/O 数据几乎全部被写入 cache,此时 ADCA 方式退化为 DCA 方式。至于 ADCA 延迟更低一点的原因可能是因为其采用了优先替换策略,但是总体来说差别不大。两者都随着网络包的增大延迟降低比例略有下降,这是因为随着网络包增大 cache 发生更多替换导致的。

从图 10 可以看出,同时运行 SPEC 和 Netperf 测试程序时,DCA 和 ADCA 方式相对于 DMA 方式的写延迟也有较大幅度的降低,但是两者之间降低的比例随着网络包的增大出现了明显的差别。DCA 方式随着网络包的增大延迟降低的比例显著下降,而 ADCA 方式的下降幅度则缓和许多,究其原因是因为 SPEC 程序的影响,当同时运行 SPEC 程序时,cache 空间变得紧张,此时 DCA 方式将所有 I/O 数据写入 cache,导致 cache 中大量数据发生替换,造成 cache 访问延迟的增加,从而导致了 DCA 方式下写延迟的增加。而 ADCA 方式则更加合理地利用 cache 空间,只是占用了 CPU 数据暂时不用的空间,因此延迟降低比例下降的幅度比较缓和。

综上所述,在多核处理器运行多个程序时,ADCA 方式可以在增大 I/O 带宽的同时减小对其他处理器程序造成的干扰,提高了 cache 空间的利用率,提升了系统的整体性能。

5 结 论

本文总结了 I/O 访存模式的进展,并在传统的 I/O 直接缓存访问的基础上提出了一种自适应的 I/O直接缓存访问方法。该方法优化了 I/O 直接缓存访问的分配、替换和写内存策略,相比已有的 DCA 和 DMA 方式,在硬件开销增加很小的情况下可以更加合理地利用 cache 资源,提高系统的整体性能。

未来的工作包括进一步发掘 I/O 访存的特性,对 I/O 访存的不同负载进行更细粒度划分,进而使用不同的策略;同时研究 cache 更细粒度的划分方法,探索更精确的 DCA 分配策略,在提高 I/O 性能的同时进一步降低对其他程序的影响。

参考文献

[1] Huggahalli R, Iyer R, Tetrick S. Direct cache access for high bandwidth network I/O[C]. In: Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05), Madison, USA,2005. 50-59

[2] Kumar, Huggahalli R. Impact of cache coherence protocols on the processing of network traffic[C]. In: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), Chicago, USA, 2007. 161-171

[3] Kumar, Huggahalli R, Makineni S. Characterization of direct cache access on multi-core systems and 10GbE [C]. In: Proceedings of the 2009 IEEE 15th International Symposium on High Performance Computer Architecture, Raleigh, USA, 2009. 341-352

[4] Leon E A, Ferreira K B, Maccabe A B. Reducing the impact of the memory wall for I/O using cache injection [C]. In: Proceedings of the 15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007), Stanford, USA, 2007. 143-150

[5] Tang D, Bao Y, Hu W, et al. DMA cache: using on-chip storage to architecturally separate I/O data from CPU data for improving I/O performance[C]. In: Proceedings of the 16th International Symposium on High-Performance Computer Architecture, Bangalore, India, 2010. 1-12

[6] Su W, Zhang L, Tang D, et al. Using direct cache access combined with integrated NIC architecture to accelerate network processing[C]. In: Proceedings of the IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, Liverpool, UK, 2012. 509-515

[7] 刘苏. 异构多核片上系统访存优化研究:[博士学位论文][D]. 北京:中国科学院大学,2014. 40-44

[8] Qureshi M K, Patt Y N. Utility-based cache partitioning: a low-overhead, high-performance, runtime mechanism to partition shared caches[C]. In: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), Orlando, USA, 2006. 423-432

[9] Mattson R L, Gecsei J, Slutz D R, et al. Evaluation techniques for storage hierarchies[J]. *IBM Systems Journal*, 1970, 9(2):78-117

[10] Qureshi M K, Moinuddin K, Patt Y N, et al. Adaptive insertion policies for high performance caching[C]. In: Proceedings of the 34th International Symposium on Computer Architecture (ISCA 07), San Diego, USA, 2007. 381-391

[11] Jaleel A, Hasenplaugh W, Qureshi W, et al. Adaptive insertion policies for managing shared caches[C]. In: Proceedings of the 2008 International Conference on Parallel Architectures and Compilation Techniques (PACT), Toronto, Canada, 2008. 208-219

[12] Xie Y, Loh G H. PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches[J]. *ACM Sigarch Computer Architecture News*, 2009, 37(3):174-183

[13] Synopsys. EVE[EB/OL]. http://www.eve-team.com: Synopsys, 2010

[14] Synopsys. GMAC IP[EB/OL]. http://www.synopsys.com/dw/dwtb.php? a = ethernet _ mac: Synopsys, 2010

[15] Jones R. NetPerf: a network performance benchmark[EB/OL]. http://www.netperf.org: Hewlett-Packard Company, 1995

Memory access optimization of I/O devices on multi-core processor systems

Li Peng[*,**,***], Zeng Lu[*,**,***], Wang Huandong[****]

([*] State Key Laboratory of Computer Architecture (Institute of Computing Technology, Chinese Academy of Sciences), Beijing 100190)

([**] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***] University of Chinese Academy of Sciences, Beijing 100049)

([****] Loongson Technology Corporation Limited, Beijing 100195)

Abstract

This paper presents a method of adaptive direct cache access (ADCA) for chip multi-core processors to improve memory access performance of I/O device and reduce the impact on other programs. Unlike traditional direct cache access (DCA), this approach takes advantage of the LRU stack property to dynamically adjust the cache space available to DCA by sampling the auxiliary tag directory, and simultaneously optimizes the replacement strategy and write memory strategy for I/O data. The experimental results show that compared with the DCA method, the proposed method improves the I/O bandwidth by about 10%. Compared with SPEC running with network in the direct memory access (DMA) method, SPECint _ rate and SPECfp _ rate gain by 11.5% and 8.9% respectively.

Key words: direct cache access (DCA), LRU stack property, adaptive, pseudo-partition, priority replacement