众核处理器中硬件支持的I/O虚拟化优化技术研究

郭御风[1] 郭诵忻[2] 邓 宇[1]
(国防科学技术大学计算机学院 长沙 410073)[1] (国家电网江西九江供电公司 九江 332000)[2]

摘 要 众核处理器中I/O资源被多个处理器核所共享。I/O虚拟化实现了I/O资源的高效共享和安全隔离,被越来越多的处理器设计所采用。硬件支持的I/O虚拟化从体系结构设计时就考虑对I/O虚拟化的支持,提供了一个全面、高效的I/O虚拟化的解决方案。深入研究了硬件支持I/O虚拟化的两大关键技术——DMA重映射技术和中断重定向技术,提出了基于Hint的IOTSB Cache管理方法和基于失效队列的失效方法来对DMA重映射进行优化,提出了多层可操控的中断模型和灵活可控的中断重定向实现方法来对I/O中断重定向进行优化。测试结果表明,提出的硬件支持的I/O虚拟化优化方法以很低的I/O性能开销实现了I/O资源的高效共享,提供了几乎接近无虚拟化环境下的I/O性能。

关键词 众核处理器,虚拟化技术,I/O虚拟化,DMA重映射,中断重定向
中图法分类号 TP302.1 **文献标识码** A

Study on Optimization Methods for Hardware Enhancements I/O Virtualization of Many-Cores Processor

GUO Yu-feng[1] GUO Song-xin[2] DENG Yu[1]
(School of Computer, National University of Defense Technology, Changsha 410073, China)[1]
(Jiangxi Jiujiang Power Supply Company, State Grid, Jiujiang 332000, China)[2]

Abstract I/O resources of many cores processor are shared by many cores. I/O virtualization supports I/O resources efficient share and secure isolation, and is adopted by more and more processors. Hardware enhancement for I/O virtualization is taken into account when it's architecture is designed, and it provides a general and efficient resolvent. This paper studied two key techniques of I/O virtualization-DMA remapping and Interrupt redirection. Firstly put forward a IOTSB cache management method based on hint and invalidation method based on invalidation queue to optimize DMA remapping performance, then a flexible and controllable interrupt redirection method was brought forward to support reliable and efficient interrupt routing. Experimental results show the hardware enhancement method for I/O virtualization which we put forward not only can support efficient sharing of I/O resource with little cost, but also can provide almost the same I/O performance as environment without I/O virtualization.

Keywords Many-cores processor, Virtualization technology, I/O virtualization, DMA remapping, Interrupt redirection

1 引言

虚拟化技术实现了计算能力、存储资源和I/O资源的高效管理和分配,精确匹配了应用所需要的软硬件资源,简化并平滑了并行应用的移植和优化过程。虚拟化技术能够提高硬件资源利用率,方便软件的跨平台移植,降低维护成本,同时提高了安全性和容错性,已成为未来高性能微处理器必须支持的特性之一。虚拟化主要包括处理器虚拟化、内存虚拟化和I/O虚拟化。众核处理器中,I/O资源被多个处理器核所共享,为了满足多个客户机操作系统的需求,必须通过I/O虚拟化的方式来复用有限的外设资源。众核结构使系统可以同时运行更多的虚拟机,共享I/O资源的竞争将变得更加激烈,I/O虚拟化的性能将直接影响整个处理器性能的发挥。

I/O虚拟化实现了众核处理器中I/O资源的高效共享和安全隔离。虚拟化技术从实现方式上可以分为纯软件实现的虚拟化和硬件辅助的虚拟化[1,2]。软件实现的虚拟化虽然跨平台性好,但是通常性能损失严重;硬件支持的虚拟化是在硬件体系结构设计时就考虑对虚拟化的支持,从而简化软件设计,减少虚拟化实现过程中的性能损失,加强虚拟机间的隔离。硬件支持的I/O虚拟化也会消耗部分系统资源,并产生一定的性能开销。因此,必须对硬件支持的I/O虚拟化的实现方法进行优化,使其在实现I/O资源高效共享的同时,保证I/O系统性能的高效。

Intel和AMD都在微处理器中实现了对虚拟化的硬件支持。Intel的VT技术是基于Intel处理器实现的虚拟化平台,目标是在独立的逻辑划分上运行多个操作系统和应用。Intel的VT技术添加了额外的处理器模式,增添了虚拟机运行指令和数据结果。Intel VT支持Directed I/O技术实现I/O虚拟化。Intel的VT技术根据涉及的处理器部件分为VT-x[3],VT-i[4]和VT-d[5],分别对应x86处理器执行部件、Itanium

到稿日期:2011-02-28 返修日期:2011-06-30
郭御风(1979—),男,博士,助理研究员,主要研究方向为微处理器设计和高性能计算,E-mail:yfguo21@yahoo.com.cn。

执行部件和 I/O 系统的虚拟化技术分支。AMD 的虚拟化技术 Pacifica[6,7] 又称为安全虚拟机 SVM(Secure Virtual Machine)。从 2006 年开始,AMD 在移动、服务期/工作站和桌面各个产品线上的处理器都将具有 SVM 能力。SVM 的目标是大大减少当前 x86/64 上虚拟化的复杂性和性能开销。在软件 Hypervisor[10-12] 之上,可以同时支持 16 位、32 位和 64 位的 Guest 操作系统与应用。AMD 虚拟化也是通过设计 IOMMU(I/O Mapping Management Unit)实现 I/O 虚拟化的。

硬件支持的 I/O 虚拟化是通过 I/O 设备的分配和指派、DMA 请求的重映射和中断的重定向实现的。其中 DMA 请求的重映射和中断的重定向对虚拟化性能造成了重要影响,因此,如何高效解决 DMA 重映射和中断重定向,成为硬件支持的 I/O 虚拟化所要解决的关键技术。本文将对这两个关键技术的实现进行优化,实现对硬件支持 I/O 虚拟化的高效支持。目前 DMA 重映射通常通过 IOMMU[8,9] 实现,IOMMU 通过查找软件维护的 IOTSB(I/O Translation Storage Buffer)表支持虚地址到物理地址的转换,其管理效率是影响 DMA 重映射性能的关键因素。中断重定向把 I/O 设备产生的中断准确及时地路由到指定客户机的虚拟处理器,灵活、可操控是确保中断能够正确、高效路由的关键因素。

本文首先介绍 I/O 虚拟化的研究背景和相关研究,并提出要研究的主要问题;然后,从 DMA 重映射方法进行研究,提出了高效的 IOMMU 实现方法、基于 Hint 的高效 IOTSB Cache 管理策略(IOTCMBH)和基于失效队列的失效处理策略(IMBINQ)对 DMA 重映射进行优化;提出了多层可操控的中断模型和灵活可控的中断重定向实现方法对 I/O 中断重定向进行优化;最后,对提出的 I/O 虚拟化优化方法进行了性能评估。评测结果表明,我们实现的 I/O 虚拟化方法很好地满足了虚拟化所要求的同质、高效和资源受控的要求,且产生的性能开销非常小。

2 DMA 重映射方法研究

2.1 高效 IOMMU 的实现方法

IOMMU 模块负责实现 DMA 操作的地址重映射,支持 I/O 设备采用虚地址直接访问系统物理存储空间。IOMMU 位于外设与系统存储器之间,将 I/O 设备 DMA 请求的虚地址转换成系统物理地址,并根据访问权限检查 DMA 操作的合法性。通常,IOMMU 实现在 I/O 的主机桥中,对桥下的所有设备进行地址转换。当系统中有多条设备树时,则需要实现多个 IOMMU。IOMMU 通过 DMA 地址重映射与权限检查,让客户机操作系统能够直接使用目标 I/O 设备。I/O 虚拟化可以根据设备号将 I/O 设备分成不同的 I/O 域,每个 I/O 域分别属于不同的虚拟客户机。实现时,I/O 域的划分粒度比较灵活,可以以总线号为单位,也可以以功能号为单位。每一个 I/O 域均被分配一个保护域,在保护域中定义了 I/O 地址转换页表和对每一个 I/O 页的读写权限。IOMMU 结构图如图 1 所示。

IOMMU 通过查询内存中的 IOTSB 表来实现客户机的虚地址到存储器物理地址的转换。IOTSB 由一组转换页表项 TTE(Translation Table Entries)组成,每个 TTE 项保存了对应的物理地址、访问权限等信息。IOTSB 保存在内存中,由软件负责初始化和维护。

由于 IOTSB 表存放在存储器中,每次查表都需要访问存储器,因此地址转换开销非常大,极大地影响了虚拟化性能。为了加速地址转换,IOMMU 中通常实现一个软件管理的一致性 cache 来保存最近使用的地址转换页表项。如果转换地址命中 cache,则无需访问系统存储器,而是通过直接访问 cache 获取地址转换页表,减少对存储器的访问次数。IOTSB Cache 表项由软件负责初始化和更新。当存储器中 IOTSB 表的表项发生失效或更新时,IOTSB Cache 中的对应表项也要失效或更新。

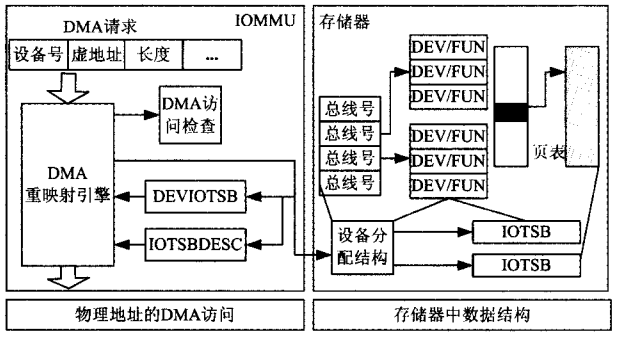

图 1 IOMMU 结构图

2.2 基于 Hint 的 IOTSB Cache 优化策略

从前面我们知道地址转换查表操作如果能够命中 IOTSB Cache,物理页号则直接可以从 cache 中取到,而无需访问存储器中的 IOTSB 表。要想提高 DMA 地址重映射的性能,就必须提高 IOTSB Cache 的命中率,避免 cache 失效开销。为了实现对 IOTSB Cache 的高效管理,提高 cache 访问性能,提出了一种基于 Hint 的高效 IOTSB Cache 管理策略 IOTCMBH(IOTSB Cache Management Method Based on Hints),它通过软件 Hint 配合硬件的 cache 管理策略,实现对 IOTSB Cache 的高效访问。

IOTSB Cache 采用全相联结构,cache 块大小为 64B,每个 TTE 表项大小为 8B,因此一个 cache 块对应 8 个 TTE 表项。系统中 cache 行的大小为 64B,因此一次 DMA 读操作可以返回整个 cache 块。cache 的更新、预取和替换都是以 8 个 TTE 表项为单位,利用访问的空间局部性实现了硬件固定预取相邻的映射表项,当软件分配连续的 DMA 虚地址空间时非常有效。同时,这种实现方法简化了 cache 管理和实现的难度。

表 1 支持的 Hint 列表

Hints	说明	硬件处理策略
顺序 Mapping	连续虚地址的地址映射,如流应用等;	主动预取后一个 cache 块; 替换采用 LRU 算法;
随机 Mapping	虚地址随机分布,空间局部性不明显;	硬件不主动预取; 替换采用 LFU,利用时间局部性;
大周期性 Mapping	虚地址周期性重复,且跳步较大;	硬件主动预取后面 n 个 cache 块; 替换采用 LRU 算法;
小周期性 Mapping	虚地址周期性重复,且跳步较小;	硬件主动预取后一个 cache 块; 替换采用 LRU 算法;
无	软件没有提供 Hint;	硬件不主动预取; 替换采用 LRU 算法;

为了减少 cache 的失效率,采用了基于软件 Hint 的替换算法和基于软件 Hint 的硬件预取算法。Hint 是软件向硬件提供的能反映某些 I/O 访问特性的提示,硬件根据 Hint 可以动态调整处理策略,以实现软硬件协作的高效 cache 管理。

软件可以根据使用要求编程 Hint，硬件则利用 Hint 改善 IOTSB Cache 效率。表 1 是目前支持的主要 Hint 以及硬件针对这些 Hint 的应对策略。

图 2 为 IOTCMBH 算法实现的硬件结构框图。IOTSB Cache 是内存中 IOTSB 表的 cache，以小的 cache 容量获得较高的 cache 命中率。每一个 cache 行对应 8 个 TTE 表项，因此每一次从内存中读 TTE 表项时，会同时预取同一行的其它 7 个 TTE 表项，既利用了 I/O 访问的空间局部性原理，又简化了 cache 设计和管理。Hint 管理模块管理 I/O 访问的 Hint，Hint 由软件根据 I/O 应用特性和用户喜好提供。Hint 管理模块根据软件 Hint 控制预取管理模块和替换管理模块，优化预取策略和替换策略。

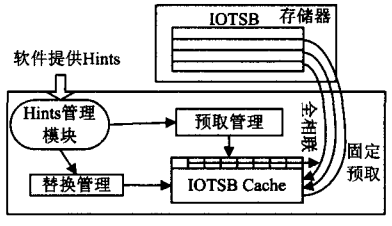

图 2 IOTCMBH 结构图

2.3 基于失效队列失效处理优化策略

IOMMU 中 cache 保存着内存中 IOTSB 表的部分副本，必须和 IOTSB 表内容保持一致。当处理器通过某些操作卸载或者变更 DMA 空间的虚实地址映射关系时，IOMMU 中 cache 的相应 TTE 表项需要被失效。通常的失效方法是通过 PIO 操作写 IOMMU 中的失效寄存器实现的，IOMMU 会把 PIO 写数据作为索引查找 cache。如果命中，则失效相应的 TTE 表项，并返回完成响应。这种方法一次只能发起一次失效操作，必须等收到失效响应后，后续的失效操作才能执行，因此失效开销比较大。为了提高失效处理效率，减少失效操作对 DMA 重映射性能影响，我们提出了基于失效队列的失效处理策略 IMBINQ(Invalidation Method Based on Invalidation Queue)。

为了减少失效处理开销，在 IOMMU 中的 ptb 模块设置了一个 tag 阵列，保存 vtb 模块中所有 TTE 表项的物理地址，用于匹配失效请求。软件发起一次失效操作，无需和正常 DMA 请求竞争查询 vtb 模块，而是直接查询 ptb 模块，如果发现有物理地址匹配项，再失效掉 vtb 模块中对应的 TTE 表项。

为了提高失效处理的效率，在 IOMMU 中设置了失效队列，用于保存未完成的失效操作。软件发起一次失效操作后，不需要阻塞后面的失效操作等待失效响应，这些失效请求全部保存到 IOMMU 的失效队列中，IOMMU 依次处理所有的失效请求。

图 3 为基于失效队列的失效处理框图。软件通过非 cache 访问处理模块(NCU)中的失效寄存器提交 IOTSB Cache 表项失效请求，只要 DMA 管理模块(DMAU)中失效队列不满，就把失效请求写入失效队列。IOMMU 依次处理失效队列中的每一个失效请求，它首先在 ptb 模块的 tag 阵列中查找 cache 中是否有失效地址对应表项。如果有，则失效 vtb 中的 TTE 表项描述符和 tdb 中的 cache 数据，完成失效处理；如果 ptb 模块的 tag 阵列中没有对应项，则直接丢掉该失效请求，完成处理。

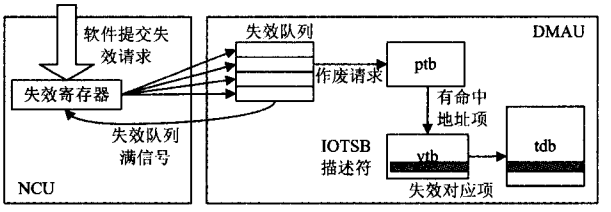

图 3 基于失效队列的失效处理框图

3 中断重定向方法研究

中断和异常是处理器提供给系统程序的重要功能，异常保证了系统程序对处理器关键资源的决定控制，而中断提供了处理器与外设之间更有效的一种交互方式。中断的及时、准确、不丢失的处理，保证了 I/O 设备的正常和高效的工作。在虚拟化系统中，每个 I/O 设备都被指派给特定的虚拟客户机，因此中断也必须重定向到相应的虚拟客户机。I/O 中断的重定向是 I/O 虚拟化的重要组成部分，中断重定向模型的性能和实现效率也将对 I/O 虚拟化的性能产生重要影响。

3.1 多层可操控中断处理模型

提出了一种多层可操纵的中断模型来支持 I/O 虚拟化的实现。图 4 为多层可操控中断处理模型。所有物理设备的中断都由 Hypervisor 首先接收并处理。Hypervisor 是硬件和系统软件的中间层，可以直接对硬件进行操作，系统软件所有的硬件操作都通过它完成。Hypervisor 负责系统初始化，被动式处理虚拟客户操作系统和上层系统软件的调用请求及中断转发；然后，Hypervisor 生成虚中断号，转发给目的虚拟客户机。从而实现了虚拟客户机设备中断的程序可操纵，极大地方便了设备虚拟化的实现，缩小了虚设备和物理设备之间的差异，简化了虚拟客户机操作系统的处理。

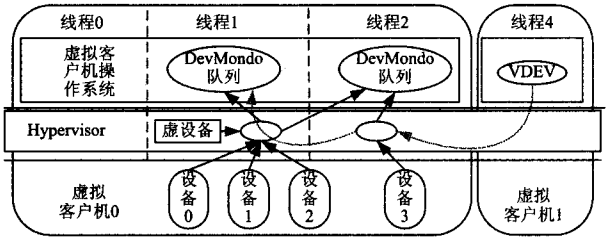

图 4 I/O 中断转发示意图

为了中断高效处理 I/O 设备，把所有 I/O 设备产生的中断都定义为 Mondo 中断，其中包括 PCI 体系结构定义的 MSI (Message Singaled Interrupts)[13] 和传统的 INTx 中断两种。所有虚设备和物理 I/O 设备中断都由 Hypervisor 通过操作 Dev Mondo 队列传递给虚拟客户机操作系统。虚设备和物理设备中断使用统一分配的虚中断号，虚拟客户机操作系统收到的虚设备和物理设备的中断没有差异，其处理流程也完全相同。

多层可操控 I/O 中断模型处理机制如下：

- 所有物理 I/O 中断都由 Hypervisor 首先接收并处理；
- 虚拟客户机操作系统收到的物理设备中断及虚设备中断都是由 Hypervisor 通过操纵虚拟处理器的 Dev Mondo 队列产生的虚中断；
- Hypervisor 提供 2048 个虚中断向量号；

·硬件配置的中断向量号只对 Hypervisor 可见，虚拟客户机操作系统使用虚中断向量号。

3.2 灵活可控的中断重定向方法

DMAU 使用事务队列排队 MSI 中断，每个事务队列都和特定的处理器绑定，一次或多次对事务队列的写只能产生一个未完成的 Mondo 中断。多个事务队列实现多个处理器间的基于硬件的中断分布。每个事务队列对应一个 Mondo 号，INTa/INTb/INTc/INTd 分别对应一个 Mondo 号。处理器进入中断处理后，对于 MSI 中断必须读取 NCU 中的 Mondo 中断数据，才能获取 I/O 设备发送的中断向量号。由于 INTx 中断一般为多个设备共享，因此必须查询所有共享某个 INTx 中断的中断源，才能获取真正的中断源。通过这种方式实现了 I/O 中断的多层可操控，并支持了 I/O 中断的重定向。

图 5 为 I/O 中断重定向处理流程图。Mondo 中断处理流程为：

1)外部 I/O 设备产生 MSI 或 INTx 中断，送到 DMAU 的中断处理模块；

2)DMAU 把收到的中断写入相应的事务队列中；

3)如果事务队列非空，则 DMAU 生成对应 Mondo 号的 Mondo 中断，发送给 NCU；

4)NCU 根据中断目的线程号查 Mondo 中断的 Busy 表，判定目标是否已有中断未处理完，并返回 ACK/NACK；如果返回 NACK，则把中断报文丢弃，由 DMAU 超时后重发；

5)如果 Mondo 中断被接收，则更新 NCU 中 Mondo 中断 Busy 表和 Mondo 中断数据表(把 MSI 中断向量保存在 Mondo 数据中)；

6)NCU 从 Mondo 中断向量表中取出中断向量，并生成中断报文，送到处理器核；

7)收到中断后，虚处理器的中断接收寄存器的对应中断位置位；

8)虚处理器进入中断处理程序，读取 Mondo 数据表，获取设备中断向量；

9)中断处理完后，更新 Mondo 中断 Busy 表，允许后续中断送到目的线程；

10)中断处理程序读取中断源的中断状态寄存器，获取中断原因，进行中断处理。

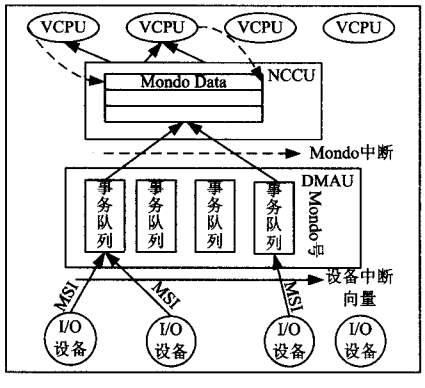

图 5 I/O 中断重定向处理流程

4 性能测试

4.1 测试环境

I/O 虚拟化优化方法应用到了自主设计的众核处理器芯片 HMCP 中，并基于 HMCP 设计了测试板。如图 6 所示，HMCP 通过 PLX8648 Switch 芯片扩展出 3 个 PCI Express 插槽，分别插了 LSI1068e SAS 卡、Intel82576 网卡和我们自主设计的高性能接口通信卡 PDP。HMCP 一共有 8 个核，总共 64 个线程，每个线程都可以虚拟成一个独立的 CPU，总共可以虚拟成 64 个独立 CPU。每个虚拟 CPU 为一个 Domain，可以成独立的操作系统，不同 Domain 在软件看来完全独立。我们在测试时分成 3 个独立 Domain，每个 Domain 单独成一个操作系统。单个 I/O 设备分别分配给不同的 Domain，实现了 I/O 资源的共享。PDP 卡支持用户级通信，主机接口为 16 通道的 PCI Express2.0 接口。网络接口为 10GB/s 的光纤接口，双向通信峰值带宽为 9.8GB/s。

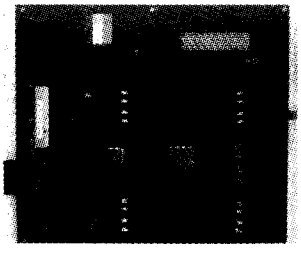

图 6 测试环境

不同 Domain 独立运行测试程序，最后统计 I/O 性能。SAS 卡挂接一个 SATA2 硬盘，测试程序主要采用 Iozone[14]；网络测试采用 netperf[15] 网络吞吐量测试程序；PDP 采用通信带宽和延迟测试程序。测试的主要指标是带宽、延迟和 CPU 利用率。

4.2 测试结果

HMCP 支持一种 I/O 虚拟化旁路模式——DirectIO 工作模式。所有 DMA 地址都不经过虚地址到物理地址的转换，屏蔽了 I/O 虚拟化逻辑层的影响。首先测试支持 I/O 虚拟化和 DirectIO 情况下磁盘、网络和 PDP 卡的性能与 CPU 利用率的对比情况，及硬件支持的 I/O 虚拟化对 I/O 性能造成的影响。

图 7 为磁盘 Iozone[14] 的读写带宽测试结果。测试数据集的大小为 16GB，块大小从 64kB 到 16MB。分别测试了 I/O 虚拟化模式和 DirectIO 模式下磁盘顺序读和顺序写，以及随机读和随机写的带宽。从图中可以看出，我们实现的硬件支持的 I/O 虚拟化方法对磁盘的读写带宽几乎没有造成什么影响，性能损失可以忽略不计。

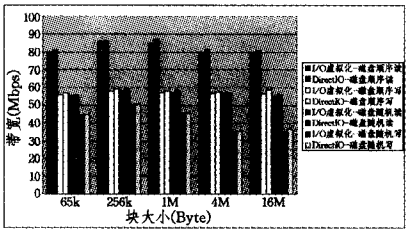

图 7 磁盘 IOZONE 测试性能对比

图 8 为网络测试程序 netperf[15] 的吞吐量测试结果。netperf 是一种网络性能的测量工具，主要针对基于 TCP 或 UDP 的传输。HMCP 作为服务器端运行 netserver 程序，一台 Xeon 服务器作为客户端运行 netperf 测试程序进行测试。测试均在千兆位网络环境下进行，使用同一个交换机。分别测试 TCP 和 UDP 模式下支不支持 I/O 虚拟化的网络吞吐量

情况。从图中可以看出，实现的硬件支持的 I/O 虚拟化对网络吞吐量损失非常小。

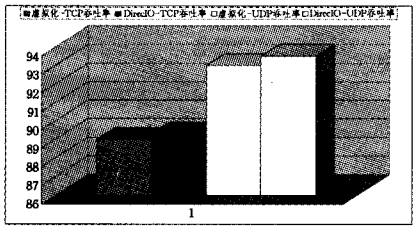

图 8　网络 netperf 吞吐量对比

我们还对接口通信卡 PDP 的单、双向带宽和通信延时情况进行了测试。从图 9 中看出，I/O 虚拟化模式下和 DirectIO 模式下单向、双向带宽基本相当，而对于小粒度的数据传输延时 DirectIO 模式下要略小一些。但当数据传输粒度大于 64Byte 后，传输延迟相当，这是由于 IOTSB cache 的预取屏蔽了地址转换的延迟。

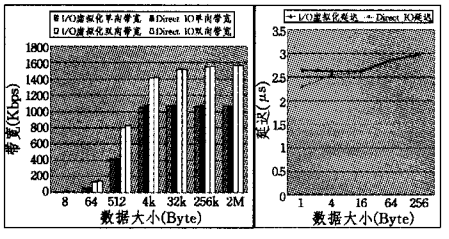

图 9　PDP 卡测试性能对比

图 10、图 11 和图 12 分别为磁盘、网络和 PDP 卡测试时 CPU 利用率的对比情况。从图中可以看出，实现硬件支持的 I/O 虚拟化带来的 CPU 额外开销也非常小，这主要得益于 IOTSB Cache 失效并不是太频繁以及对失效操作的优化。

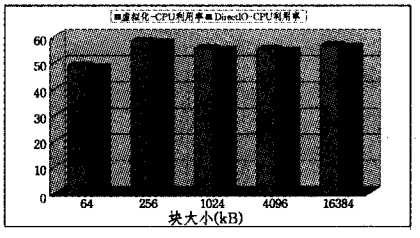

图 10　Iozone CPU 利用率对比

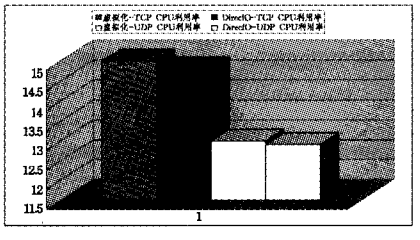

图 11　Netperf CPU 利用率对比

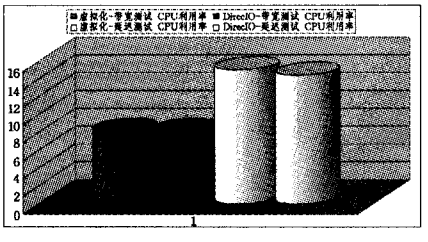

图 12　PDP 测试 CPU 利用率对比

前面对实现的硬件支持的 I/O 虚拟化模式下和不支持 I/O 虚拟化的 DirectIO 模式下的 I/O 性能和 CPU 利用率进行了测试和对比。下面将对 IOTSB Cache 的管理策略优化方法进行性能评测，分别分析 IOTCMBH 方法和 IMBINQHS 方法对 I/O 虚拟化性能的优化情况。

我们基于 PDP 卡进行通信测试，采用了随机地址测试(Random)、循环地址测试(Loop)和大粒度顺序(Sequence)测试 3 种。由于 IOTSB Cache 对应的最大 DMA 空间大小为 4MB，因此传输数据大小采用 16MB，分别比较 IOTCMBH 和 LRU cache 管理算法的性能。

从图 13 中可以看出，IOTCMBH 方法相对于 LRU 方法性能得到了很大改善，特别是对激励的特征比较明显的应用(比如 Loop 测试)，性能提升明显。主要原因是硬件基于 Hint 的预取和替换，大大降低了 IOTSB Cache 的失效率，虚实地址转换绝大部分都能命中 cache，这样地址转换的开销基本可以忽略。

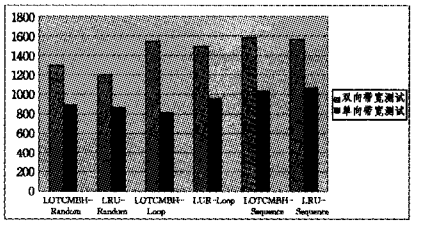

图 13　IOTCMBH 和 LRU 性能对比

下面将测试 IMBINQHS 失效方法和基于失效寄存器(INVR)方法的失效开销。由于实际芯片对失效操作单独测试比较困难，因此我们采用模拟的方法，模拟环境为 HMCP 芯片最终逻辑的系统软模拟环境。一方面让处理器连续发出多个 IOTSB Cache 失效请求，另一方面不停发起上行 DMA 请求(虚实地址转换需要查 IOTSB Cache)，以查看所有失效请求的最终完成时间。测试结果采用相对于单次失效的时间比值。

图 14 为 IMBINQHS 和 INVR 两种失效方法的性能对比。从图中可以看出，INVR 失效处理时间比失效请求数目增长要快，主要原因是失效处理和 cache 查找操作冲突。而 IMBINQHS 方法的失效相对处理时间只有略多于失效请求个数的一半，这主要得益于失效处理的流水化和失效操作、cache 查找操作的并行化。因此可以看出，我们提出的 IMBINQHS 方法可以大大提高失效处理效率，降低 cache 失效带来的性能损失。

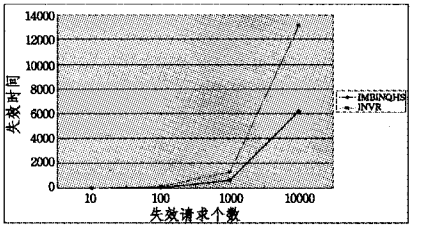

图 14　IMBINQHS 和 INVR 性能对比

结束语　I/O 虚拟化提高了系统的资源利用率、隔离安全性、使用方便性等，已经广泛应用于当前微处理器中。硬件支持的 I/O 虚拟化通过增加专门的虚拟化硬件，大大减少了虚拟化对系统性能的影响。我们对硬件支持的 I/O 虚拟化技术进行了优化。测试结果表明，我们提出的硬件支持的 I/O

虚拟化优化方法以很小的性能开销实现了I/O资源高效共享和安全隔离，并在我们设计的众核处理器芯片的应用中，取得了很好的效果。

参考文献

[1] Tang Hong, Gulbeden, Zhou Jing-yu, et al. The Panasas ActiveScale Storage Cluster-Delivering Scalable High Bandwidth Storage[C]//Proc of Supercomputing'04. 2004:53-53

[2] Adams K, Agesen O. A comparison of software and hardware techniques for x86 virtualization PPT[EB/OL]. http://www.cs.wisc.edu/areas/os/schedules/archive/vmware_osseminar.ppt

[3] Pratt I, Fraser K, et al. Xen 3.0 and the Art of Virtualization [C]//Proceedings of the Ottawa Linux Symposium. 2005

[4] Intel Corp. Intel Virtualization Technology Specification for the Intel Itanium Architecture[EB/OL]. www.intel.com/technology/vt/

[5] Intel Corp. Intel Virtualization Technology for Directed I/O Architecture Specfication[S]. 2006

[6] AMD Inc. Live Migration with AMD-V Extended Migration Technology[S]. 2007

[7] AMD Inc. Putting Server Virtulization to Work [R]. AMD White Paper, No32915B

[8] Ben-Yehuda M, Mason J, et al. Utilizing IOMMUs for Virtualization in Linux and Xen[C]//Proceedings of the Ottawa Linux Symposium. 2006

[9] Xen IOMMU trees[EB/OL]. http://xenbits.xensource.com/ext/xen-iommu.hg, 2007

[10] Sun Microsystems, Inc. The T1 Hypervisor and the sun4v Architecture/API[S]. 2008:123-128

[11] Chisnall D. The Definitive Guide to the Xen Hypervisor[M]. Prentice Hall, 2007

[12] Russel R. lguest: Implementing the little Linux hypervisor[C]//Proceedings of the 2007 Ottawa Linux Symposium. 2007

[13] PCI-SIG. PCI Express Base Specification Revision2.0[S]. 2009

[14] IOzone Filesystem Benchmark[EB/OL]. http://www.iozone.org

[15] Blum R. Network Performance Open Source Toolkit[M]. Wiley Publishing, Inc

(上接第289页)

表4 CAF程序性能(规模二:秒)

映像数	yg	la	md	mf
16	987.3	1731.2	763.22	954.61
32	418.98	864.4	393.21	388.01
64	209.12	443.1	203.01	128.05
128	149.63	218.9	109.84	67.98
256	75.31	104	55.59	42.06

表5 MPI程序性能(规模二:秒)

任务数	yg	la	md	mf
16	858.7	1510.8	919.3	892.07
32	426.5	747.9	467.8	382.51
64	205.2	385.4	237	104.3
128	99.2	185.5	119.4	53.93
256	54.3	89.3	58.6	31.86

对于规模一，CAF和MPI程序性能相当。尤其是md程序，CAF性能明显好于MPI。这是因为采用数据垫塞，使md程序的Cache利用率得到提高。对于yg和mf，当映像为1时，性能低于MPI。

对于规模二，当映像/进程数在16、32和64时，CAF和MPI的程序性能基本相当，当任务数达到128时，对于yg程序，CAF程序性能明显低于MPI。其原始是因为CAF规范中目前没有聚合操作，例如广播、规约等。广播操作通过所有映像读取同一个Co-Array数据实现，当规模扩大时，造成存储访问竞争，严重影响性能。归约操作通过临界区实现，规模扩大时，性能远低于采用树方式实现的MPI归约。yg程序中，一次迭代执行4次广播、6次归约，因此，当存在256个映像时，CAF性能低于MPI。CAF目前正在讨论的规范，已经将聚合操作加入[8]。

结束语 CAF编程简单，可扩展性好，极大地简化了SPMD模式的并行程序设计，是PGAS语言的典型代表，在科学计算领域已经逐步被人们接受。尤其是CAF已成为Fortran语言的一部分，极大地促进了CAF的发展和应用。

基于软件虚拟共享存储，在分布式系统上实现了一个CAF编译器。典型科学计算程序测试表明，其能够获得和MPI相当的性能。在下一步的工作中，将实现包含聚合操作的CAF的新标准，通过数据流分析，删除冗余的Co-Array的通信，提高CAF程序的性能。

参考文献

[1] Numrich R W, Reid J. Co-array Fortran for Parallel Programming[J]. ACM Fortran Forum, 1998, 17(2):1-31

[2] Fortran J3 Committee. Fortran 2008 Working Draft, J3/09-007r1[EB/OL]. Oct. 8, 2011. ttp://www.j3-fortran.org/doc/standing/links/007.pdf

[3] Cray Fortran Reference Manual. Volume 3[EB/OL]. http://docs.cray.com/books/S-3694-54/S-3694-54.pdf

[4] Adhianto L, Jin G, Krentel M, et al. Coarray Fortran 2.0 Pre-alpha Release Notes[EB/OL]. Oct. 8, 2011. http://caf.rice.edu

[5] Subset Co-Array Fortran translator[EB/OL]. Oct. 8, 2011. http://neptune.ce.ncsu.edu/~sarat/sc07/PGAS-SC2007/CAF/caf2omp.htm

[6] Coarrays via GNU Fortran's Coarray Communication Library [EB/OL]. http://gcc.gnu.org/wiki/CoarrayLib, 2011-10-08

[7] Distributed Memory Coarray Fortran with the Intel Fortran Compiler for Linux: Essential Guide[EB/OL]. Oct. 8, 2011. http://software.intel.com/en-us/articles/distributed-memory-coarray-fortran-with-the-intel-fortran-compiler-for-linux-essential-guide/

[8] Sottile M J, Rasmussen C E, Graham R L. Co-Array Collectives: Refined Semantics for Co-Array Fortran [C] // International Conference on Computational Science (2). 2006:945-952