混合体系结构中有状态硬件加速器的优化

马宜科[1),2),3)] 常晓涛[2)] 范东睿[1)] 刘志勇[1)]

[1)](中国科学院计算技术研究所系统结构重点实验室 北京 100190)

[2)](IBM 中国研究院 北京 100193)

[3)](中国科学院研究生院 北京 100049)

摘 要 在诸多计算领域中,硬件加速器可以代替通用处理器上执行的软件完成专用功能,达到提高性能和降低功耗的目的.网络应用中,许多硬件加速器是无状态的,这就需要一个网络流的全部数据包到达后才能被处理.有状态加速器则可以确保每个数据包到达后即可被处理,因而具有更好的性能和灵活性.由于网络流的并发性,有状态加速器需要维护众多并发网络流的状态,并在需要时进行硬件状态切换,从而增加了加速器的性能开销.该文基于请求队列提出对不同网络流的请求进行动态重排序的方法,其中请求所在的队列可以在片上也可以在片外,从而有效减少加速器的状态切换次数.对多种流行的有状态加速器进行的实验结果表明,该方法可以有效降低加速器的平均响应时间并提高吞吐率.实验结果表明:与传统的 FIFO 设计对比,解压缩加速器的吞吐率最大提高了 26.7%,响应时间最大减少了 50%.

关键词 硬件加速;混合体系结构;有状态加速器

中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2011.01314

Optimization of Stateful Hardware Acceleration in Hybrid Architectures

MA Yi-Ke[1),2),3)] CHANG Xiao-Tao[2)] FAN Dong-Rui[1)] LIU Zhi-Yong[1)]

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

[2)](*IBM Research-China*, *Beijing* 100193)

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

Abstract In many computing domains, hardware accelerators can improve throughput and lower power consumption, instead of executing functionally equivalent software on the general-purpose micro-processors cores. While hardware accelerators often are stateless, network processing exemplifies the need for stateful hardware acceleration. The packet oriented streaming nature of current networks enables data processing as soon as the packets arrive rather than when the data of the whole network flow is available. Due to the concurrence of many flows, an accelerator must maintain and switch contexts between many states of the various accelerated streams embodied in the flows, which increases overhead associated with acceleration. This paper proposes to dynamically reorder the requests of different accelerated streams in a hybrid on-chip/memory based request queue to reduce the associated overhead. Through a simulation-based performance study, the effectiveness of the proposed mechanism for different popular stateful accelerators is shown. The experimental results shown the approach can help reduce the average response time sig-

收稿日期:2010-12-23;最终修改稿收到日期:2011-06-13.本课题得到国家自然科学基金重点项目(60736012)、国家"九七三"重点基础研究发展规划项目基金(2005CB321600)、国家"八六三"高技术研究发展计划项目基金(2009AA01Z103)及北京市自然科学基金(4092044)资助.**马宜科**,男,1980 年生,博士研究生,主要研究方向为计算机体系结构、片上存储系统等. E-mail: ykma@ict.ac.cn. **常晓涛**,男,1978 年生,博士,IBM 中国研究院研究员,研究领域包括计算机体系结构、处理器仿真、低功耗处理器设计. **范东睿**,男,1979 年生,博士,副研究员,主要研究方向为低功耗处理器设计. **刘志勇**,男,1946 年生,博士,研究员,博士生导师,主要研究领域为算法、计算机系统结构、并行处理、片上存储系统等.

nificantly and improve throughput up to 26.7% and response time reduction of upto 50% for decompression acceleration compared with the traditional FIFO order design.

Keywords　hardware acceleration; hybrid architecture; stateful acceleration

1　引　言

CMOS 技术的发展是计算机性能指标的主要推动力之一.通过在处理器内集成网络接口和数据解压加速器等部件,越来越多的处理器被设计成既有通用处理单元又有专用加速器的混合体系结构的芯片.网络应用的性能会受到数据传输速率、延迟、吞吐率及流特性等因素的限制.当前的网络应用已经增加到 10Gbits/s 的带宽量级,正在向 100Gbits/s 发展,这就需要能同时满足低延迟、高带宽和低功耗需求的新一代处理器的支持.这些系统的重要特性包括使用大规模硬件多线程以及集成对网络的硬件支持和特定应用的硬件加速器[1].

硬件加速器的优势体现在网络流处理的多个方面:一方面,当网络流的数据包到达的时候,系统对接收的全部数据包进行缓存和分析,再由硬件线程对数据包进行调度和进一步的处理.因为每个包都需要经过这个流程的处理,对这些步骤进行加速可以显著增加性能.另一个方面,网络数据解密[2]① 也可以体现加速器的优势,这是因为硬件实现的解密算法比软件更高效,特别是当需要解密的数据包的数量很大的时候,硬件加速器可以显著提高系统性能.数据解压缩方面,在虚拟专用网的网络层和应用层,大量的数据包经过压缩之后再传输以降低对广域网的带宽需求;在应用层,HTTP/1.1 通过压缩网页内容来减少带宽需求;通过邮件传输的大文件也经常使用压缩传输.因此减少接收方的解压缩开销,可以在很大程度上提高吞吐率.此外,正则表达式匹配(Regular eXpression,RegX)[3] 是实现网络安全领域中深度包检测的重要方面,通过对其进行硬件加速,也可以获得明显的性能收益.

在传统应用中,同一个网络流的数据包必须被接收方全部接收,并在应用层重新组装成整块数据之后再进入硬件加速器进行处理.网络流中单个的数据包在到达之后并不能被硬件加速器处理,我们称这种处理方式的加速器为无状态的(Stateless)加速器.有状态(Stateful)加速器不需要等待一个网络流上的全部数据包到达,它可以在网络流的任一数据包到达之后立刻进行处理.为此,每处理完网络流中的一个数据包,加速器需要保存被加速网络流的当前状态,而当该网络流的下一个数据包到达之后,加速器首先需要恢复其前一个数据包处理完之后的状态,然后才能处理刚到达的数据包.因此,有状态加速器具有更好的实时性和吞吐率,更适合网络中的各种应用,例如压缩/解压缩、加密/解密、正则表达式匹配和 XML 处理等.

在带来灵活性的同时,有状态加速器的状态切换会引入性能开销.网络流的高并发性会导致频繁的状态切换,尤其在所切换的状态数据较多时,会付出显著的性能代价.一般来说,加速器按照先进先出(FIFO)的原则从加速请求队列中取出请求并进行处理.本文基于片上/主存混合请求队列,提出对队列中的请求进行重排序的方法,从而减少状态的切换次数即访存次数以提高性能,并对其进行仿真实验和性能评估.

本文第 2 节介绍硬件请求队列的一般性设计;第 3 节描述有状态硬件加速器设计的优化方法;第 4 节给出实验的模型和参数以及模拟器的实验结果;第 5 节是本文结论.

2　研究背景

通常,一个数据包通过网络协议栈的处理之后,处理器线程将为其创建一个协处理器请求块(Coprocessor Request Block,CRB[4]),并发往硬件加速器.CRB 至少包含源数据地址、目的数据地址和状态地址.其中,源数据地址指向需要硬件加速器进行处理的数据;目的数据地址指向由处理器分配给即将产生的结果数据的存储空间;状态地址所指向的空间则是用于维护当前网络流状态的专用区域.这里给每一个网络流分配唯一的 ID(Context ID,CID)进行状态检索.来自众多线程的大量并发请求将导致加速器十分繁忙,为了使处理线程可以不被阻塞从而继续进行其它处理,通常需要在硬件上设计能够缓存大量请求的队列(FIFO).

由于处理器芯片的面积有限,片上队列只能保存数量很有限的请求,例如当前工艺下,一个片上硬

① Advanced Encryption Standard. http://en.wikipedia.org/wiki/Advanced Encryption Standard.

件请求队列大约可以容纳数百项 CRB. 但是在当前网络中,每秒钟可能有上百万个包到达,因此片上的队列资源在短时间内很有可能被迅速耗尽. 一旦片上队列被填满,发送请求的线程将不能继续提交 CRB,从而进入阻塞状态. 由于当前片外存储(如主存)相比片上存储来说可以认为容量是无限大的,一种解决片上队列容量不足问题的方法是通过使用片外存储器中的空间维护一个片外队列(本文也称其为溢出队列,spill queue),从而达到增加请求队列容量的目的. 在图 1 中,放置于加速器之前的请求队列包含了片上队列和片外队列. 当片上队列已满的时候,溢出队列控制模块自动将新到达的 CRB 放入片外队列中. 一旦加速器处理完一项 CRB,片上队列将会空出一项位置,溢出队列控制模块将自动把片外队列中位于队首的 CRB 取到片上队列中,避免了处理线程上软件的干预.

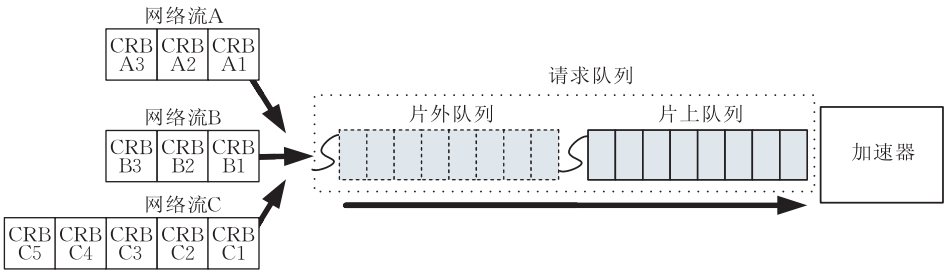


图 1 硬件加速器的请求队列

在硬件加速器中使用溢出队列管理技术之后,可以近似认为整个请求队列是无限长的,这是排队论[5]中标准的 M/M/1 模型. 在下面的章节中,我们将基于 M/M/1 模型生成不同并发程度的网络流进行实验.

因为大量并发网络流的 CRB 交错到达硬件加速器,网络应用需要加速器为每一个网络流保存状态(上下文). 由于受处理器芯片面积所限,特别是当某种加速器所需要保存的状态数据较大时,例如压缩加速器需要 32KBytes 空间保存状态,在片上保存全部网络流的状态是不现实的. 因此,通常选用主存来存储不同网络流的状态.

如图 2 所示,CRB C1 是网络流 C 的第一个请求,一旦它被处理完成,网络流 C 的当前状态就被写回主存中网络流 C 的状态空间,以备网络流 C 的下一个 CRB 使用. 随后,网络流 A 的 CRB A1 和网络流 B 的 CRB B1 相继被处理,其状态也随之被写回主存中自己的状态空间. 接着,加速器从请求队列中取出网络流 C 的 CRB C2,然后从主存中加载之前存储的状态来恢复 CRB C1 完成时的加速器状态,一直等到状态从主存中完全取回,才可以进行对源数据的处理. 图 2 中,CRB C3 和 CRB C4 相邻并且来自同一个网络流,因此在 CRB C3 完成后,就不需要进行状态切换. 频繁的状态切换意味着频繁的访存,由于当前计算机系统中存储访问带宽瓶颈的存在,必然会造成显著的延迟,从而降低性能. 我们将在下一节给出针对这一问题的优化方法.

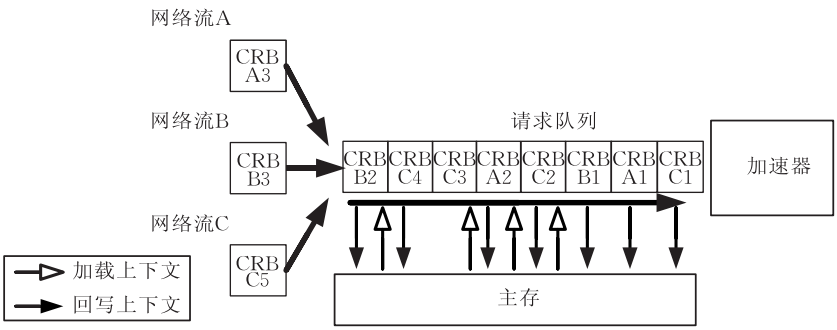


图 2 多个网络流请求的交错到达

3 有状态加速器的请求重排序技术

本节将提出对请求队列项进行动态重排序的方法,从而将来自相同网络流中等待处理的 CRB 放到相邻的位置上. 此方法通过减少加速器状态切换次数,来降低跟状态保存和恢复相关的内存操作次数,以达到提高加速器性能的目的,包括片上请求队列的重排序技术和对溢出队列的请求查找技术两部分. 这两种技术的重排序结果都保证了原先同一个网络流的多个 CRB 在重排序之后的相对次序不变.

3.1 片上请求重排序技术(On-chip Request Reordering,ORR)

ORR 技术的主要作用如下所述：将新到达的 CRB 的 CID 与片上请求队列中所有等待处理的 CRB 的 CID 进行比较. 如果没有找到相同的 CID，说明在请求队列中不存在来自同一个网络流的请求，新的 CRB 将被插到队尾. 如果找到，说明在请求队列中存在来自相同网络流的加速请求，新的 CRB 将通过片上请求重排续表(On-chip-request Reorder Table,ORT)将其插入到这一个网络流的最后，从而使得当加速器在完成了一个 CRB 并且还有多个 CRB 正在等待的时候，直接取到同一个网络流的下一个 CRB，避免了状态切换.

对比图 2，图 3 中给出了同一个 CRB 序列在采用 ORR 技术之后的处理过程. 图 3 中，CRB C2、C3 和 C4 由于进行了重排序，紧随同一个网络流的 CRB C1 之后，因此加速器在 CRB C1 完成之后不需要切换状态，可以直接处理 CRB C2、C3 和 C4，在 C4 完成后才需要进行一次状态切换. 与图 2 中的情况相比，ORR 大大减少了加速器的状态切换次数.

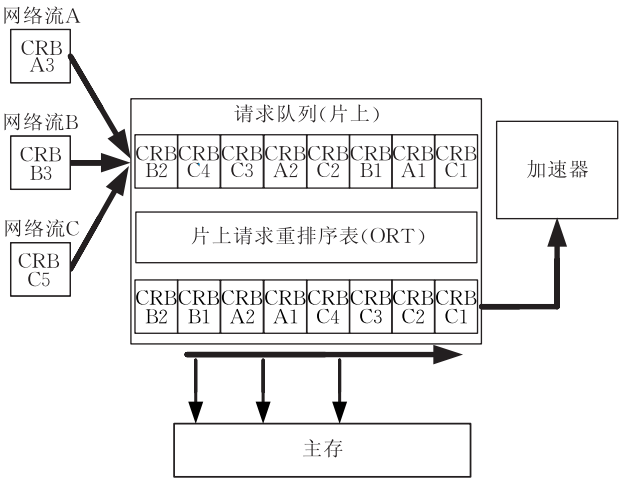


图 3　片上请求重排序(ORR)的结构示意图

当片上队列由于请求过量而耗尽时，可以使用溢出队列来维护更多的请求. 只要片上请求队列中出现空位，溢出队列控制模块将立即把溢出队列中位于队首的 CRB 取到片上队列中. 一旦这些请求进入片上请求队列，它们都将按 ORT 的信息重排序，从而减少加速器的状态切换开销. 但是，由于 ORR 技术只能对片上请求队列进行重排序，因此短时间内的大量并发网络流(Concurrent Streams,CS)可能会减少重排序的机会. 例如，如果 CS 数量大于片上队列容量，并且队列中多个网络流之间基本上都是交错到达的，那么 ORT 获得的收益将会非常有限. 因此，下一小节将给出对溢出队列中的 CRB 进行重排序的方法.

3.2 溢出请求查找技术(Spill Request Lookup,SRL)

ORR 技术局限于对片上请求队列中的 CRB 进行重排序. 如果我们把片上队列看作一个窗口，显然，窗口的尺寸越大，可以重排序的机会就越多. 溢出队列中的所有 CRB 都在这个窗口之外，因此本节定义一个附加的窗口，即流查找表(Stream Lookup Table,SLT)，它基于内容可寻址存储器(Content Addressable Memory,CAM)①，扩展了溢出队列控制模块的功能，涵盖了溢出队列中部分最常用的信息，如图 4 所示. 首先，从逻辑上将溢出队列分成多个 CRB 组，相同组的全部 CRB 属于同一个网络流，每个组有多个用来存放 CRB 的槽，CRB 组中槽的数量是由软件配置的，图4中设置为8. 其次，SLT 中的每一个表项由 CID、在溢出队列中的组地址以及该组空槽的偏移量组成. 一旦一个新的 CRB 到达，它的 CID 将与 SLT 中所有的有效表项作匹配. 如果与其中一项匹配，这个 CRB 将被放入相应的 CRB 组的空槽中，同时，空槽偏移量循环增长，其中空槽的地址可以通过所匹配的组地址与空槽偏移量计算得到. 如果没有任何匹配，SLT 将给此 CRB 分配一个新表项，同时，这个 CRB 将被放到内存中该组的第一个槽中.

当一个 CRB 完成时，加速器将从片上队列中取出下一个 CRB 进行处理. 在片上队列满的情况中，溢出队列控制模块将立即把溢出队列中位于队首的 CRB 取到片上队列刚空出来的位置中. 然后，随着 CRB 持续的完成，在溢出队列同一组中的所有其它 CRB 都将被依次取到片上队列新空出来的位置中. 当这个组中的所有 CRB 都被取到片上之后，溢出队列中的下一组 CRB 将会继续被取到片上，以此类推. 由于大量连续的 CRB 都已经被排好了顺序，因此加速器状态切换次数大大降低.

SRL 技术进一步扩展了重排序窗口，后面的实验结果将表明，ORR 技术只能在小规模并发的情形下获得收益，而 SRL 技术在更高的并发规模下仍然能获得收益.

目前国内外鲜有有状态硬件加速器性能优化的相关工作，相关文献只有参考文献[6]. 但是文献[6]是利用解压缩算法的本身特性设计而成的有状态解

① CAM. http://en.wikipedia.org/wiki/Content-addressable memory.

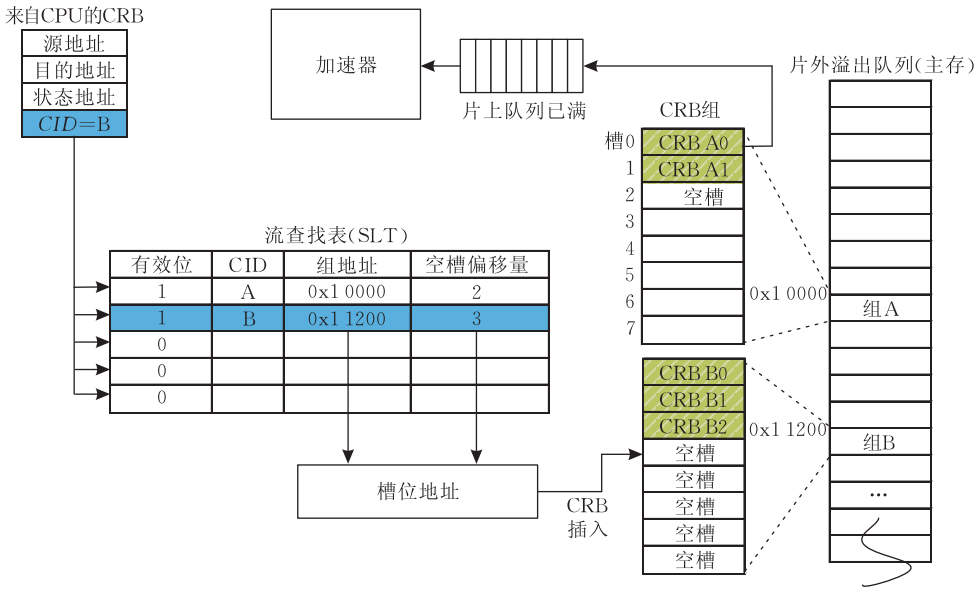


图 4 溢出请求查找(SRL)的结构示意图

压缩加速部件;而我们的研究成果适用于所有有状态加速器的通用优化方法,利用了对网络流预排序的方法来减少上下文交换.因此这两种方法是正交的,在使用了文献[6]所描述的部件上附加我们的部件,在不影响原有部件性能的基础上,仍然可以获得我们的部件所带来的性能提升.

3.3 硬件实现代价

为 ORR 技术设计的 ORT 和为 SRL 技术设计的 SLT 的核心部件都是内容可寻址存储器,它是一种在高速查找应用中广泛使用的电路,其主要工作机制是将一个输入数据项与存储在内容可寻址存储器中的所有数据项同时进行比较以判断是否匹配,并输出是否匹配的布尔信号及数据项的对应地址.由于大量使用 CAM 会显著增加芯片的面积和功耗,当前芯片上内容可寻址存储器通常最多只有几百项.因此,ORT 和 SLT 的容量可能会受到限制.

理论上,SLT 的容量与并发网络流的数量相当是最合适的.由于网络流有突发性[7],即同一个网络流的不同数据包,会在一个相对较短的时间内到达,因此使用较少容量的 SLT 就可以满足需求.后面的章节将表明,数百项的 ORT 和 SLT 对性能的提高已经非常明显.这里假设 ORT 和 SLT 的项数都是 128,与整个 IBM WSP 处理器[8]的面积进行比较,用以支持 ORR 和 SRL 的全部硬件代价不超过 0.05%.

3.4 公平性

上面所述的重排序方法可能会改变某些 CRB 的响应时间,从而影响某些网络流的请求延迟,造成不公平现象.我们可以通过限制重排序的时间跨度来使这个影响降到可承受的范围之内.例如,通过软件设定一个时间跨度,新到达的 CRB 相比等待最久的 CRB 已经超出了这个时间跨度,这个 CRB 将不会被重排序,而是被放到当前请求队列的队尾,从而不会影响到先于它发出的那些请求.本文中,我们仅给出在不加入公平性限制的情况下,重排序对网络流最大响应时间的影响.我们会在将来的工作中对加入公平性策略后的重排序方法进行实现和评估.

4 实验结果与分析

对本文方法进行性能评估的指标包括两个方面:吞吐率和响应时间.我们在解压缩加速器上进行了详细的性能研究,对于其它的加速器,我们只给出性能提高的峰值结果.

我们在实现了 IBM Wire-Speed PowerTM Processor(WSP)的 Mambo 模拟器上进行实验,其主要参数如表 1 所示.Mambo[9]是一个 IBM 全系统模拟器,而 IBM WSP[4,8]是一个集成了 16 个 IBM

表 1 模拟器的配置参数

参数	配置
主频	2GHz
处理器核数/线程数	16/64
一级指令/数据 Cache	16x (16KB+16KB) SRAM
二级 Cache	4x 2MB eDRAM
硬件加速器	压缩/解压缩 XML RegX 加解密(AES,DES,ARC4,Kasumi,SHA,MD5)
内存带宽	2x DDR3 controllers 4 Channels@800~1600MHz
系统 I/O 带宽	4x 10GB Ethernet, 2x PCI Gen2

PowerPC 核(每核含 4 硬件线程)和多个专用加速器的处理器.

我们首先参照加速器的状态切换开销得到一个近似的服务时间.假设平均的负载为每个数据包1200Bytes,平均压缩率因子为 4.91[6],状态信息大小取为 2.5KB(2KB 历史状态[6]和 0.5KB 动态哈夫曼表[10]①).对解压缩加速器来说,状态写回开销可以忽略,这是因为其状态就是上一次输出的数据,它的写回过程已经包含在结果数据(平均 5892Bytes,1200Bytes×4.91)的写回阶段.

无论是否存在加载或者写回停顿,加速器的解压缩性能都约为每时钟周期 1 字节.因此,处理完成一个加速请求大约需要 1200 个时钟周期.因为在 WSP 中写回 8 个 Cache 行(64Bytes)需要花费 300 个时钟周期,所以写回 5892 个字节的输出结果需要 3452 个时钟周期.对于一次状态切换,我们需要 1464 个时钟周期来加载 2.5KB 的状态信息.基于服从泊松分布的 M/M/1 的排队论模型,我们生成了多组不同到达速率以及多种并发流数(从 $CS=8$ 到 $CS=2048$)的模拟网络流,用来研究在不同的 ORT 和 SLT 大小以及传统 FIFO 下处理性能.

4.1　吞吐率优化

我们首先分析了 ORT 从只有 1 个表项(FIFO)到有 128 个表项的情况,从而得到在各种不同配置下可以进行重排序的总次数,如图 5 所示.FIFO 意味着只有相邻到达的两个 CRB 恰好是来自同一网络流的时候,才不需要加速器切换状态.随着 CS 的增加,相同的窗口大小下,可以重排序的次数是逐渐减少的.同时,随着窗口大小的增加,可以重排序的数量是逐渐增加的.在某些情况下,可重排序的 CRB 数量能达到总 CRB 数量的 90%,这也反映了请求重排序技术具有较大的优化空间.

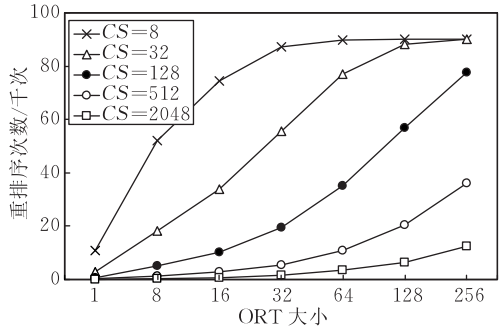


图 5　使用 ORR 之后的重排序次数

图 6 给出了仅使用 ORR 技术的解压缩加速器得到的吞吐率优化结果.随着重排序表大小的增加,吞吐率最大可以提高 26.7%.当 CS 远大于窗口大小的时候,最差的情况($CS=2048$)将获得 2%的吞吐率提升.

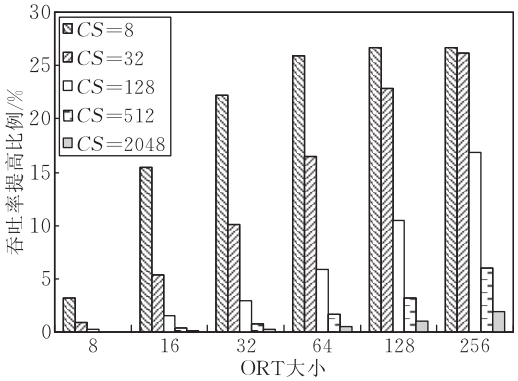


图 6　相对传统 FIFO、ORR 对硬件解压缩吞吐率的提升

根据参考文献[6]中所收集的大量来自热门门户网站的网页测试集,单个网页压缩后的平均长度大约为 7 个网络包,因此我们设置槽数为 10,既能满足基本需求,也便于展示我们的设计.本文后续实验中 CRB 组的槽数都设置为 10.我们将 ORT 的大小固定为 128,并且设定 SLT 的大小在 128～512 个表项之间变化,以观察溢出队列在使用 SRL 之后的效果.

图 7 表明,采用 SRL 技术之后,吞吐率得到了提高,尤其是随着 CS 增加的时候将更加明显.当并发流数小于或等于 SLT 大小的时候,这种机制非常有效,带来了超过 20%的性能提高;当并发流数 4 倍于 SLT 表大小时,相对 FIFO,我们仍然得到了 7.2%的性能提升.

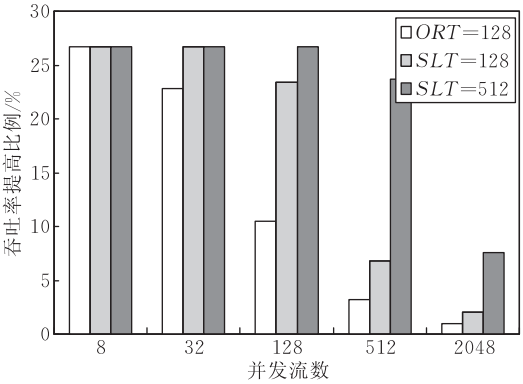


图 7　相对传统 FIFO,SRL 对硬件解压缩吞吐率的提升

除了解压缩加速器之外,我们也分析了 IBM WSP 处理器中的其它几种硬件加速器,如表 2 所

① Deutsch L P. RFC 1951: Deflate Compressed Data Format Specificaiton V1.3. http://www.ietf.org/rfc/rfc1951.txt, by Internet Engineering Task Force, 1996

示,我们给出了请求重排序的峰值吞吐率收益.这里给出的峰值依据各种不同配置下所观察到的最大性能收益,包括($CS=8$,$ORT=64$),($CS=32$,$ORT=256$)和($CS=128$,$SLT=256$).其中,加解密加速器的吞吐率收益大约只有4%.这是因为加解密加速器的状态信息一般最多只有64Bytes,其状态切换开销相比处理时间来说并不是限制吞吐率的关键因素.

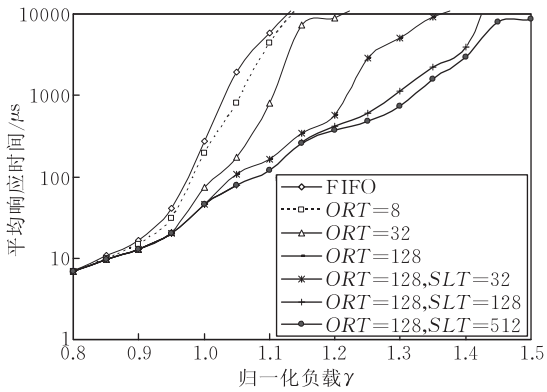
表2 不同加速器的吞吐率峰值收益

加速器	IBM WSP中的上下文大小/Bytes	吞吐率提高比例/%
解压缩	2500	26.7
XML	256	15.8
RegX	192	9.9
加解密	64	4.0

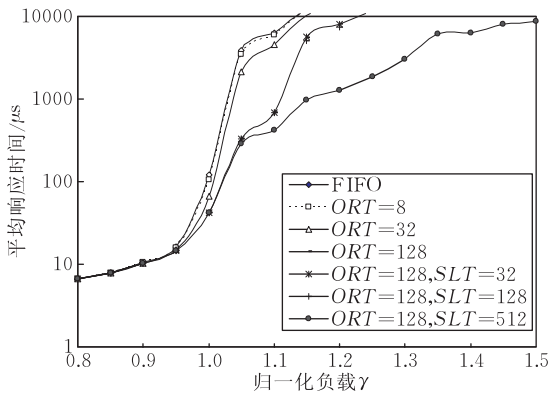
4.2 响应时间优化

本节将评估请求重排序对平均和最大响应时间的收益.在很多应用中,响应时间是应用程序性能的关键因素.一些对响应时间敏感的应用,例如入侵检测,我们可能无法让加速器运行在最大的服务速率上.基于已经生成的模拟网络流,我们提高请求到达速率λ并且评估加速器在不同λ下的平均响应时间.我们把平均响应时间作为归一化负载γ的一个函数,其中μ_{FIFO}表示在没有重排序的情况下的平均服务速率,这里的$\gamma=\dfrac{\lambda}{\mu_{\text{FIFO}}}$,即相对于FIFO情况下的服务速率进行归一化.

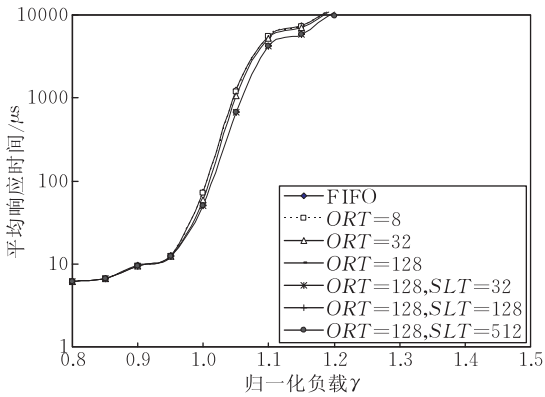
我们分别研究了并发流数为128、512和2048时平均响应时间的情况.图8(a)、8(b)、8(c)分别显示了对于不同配置的ORT和SLT,归一化负载(γ)带给平均响应时间的影响.图8(a),当并发流数比较低($CS=128$)时,在归一化负载γ到达0.8之前,平均响应时间上基本上没有收益.在$\gamma>0.8$之后,增加ORT和SLT的大小才会得到收益,例如$\gamma=0.95$的情况下,$ORT=128$在所有的SLT配置下平均响应时间都减少了50%.更大的SLT使得在$\gamma>1.0$时平均响应时间获得更多收益,例如当$\gamma=1.2$时,在ORT项数为128的条件下,$SLT=512$使得平均响应时间减少了36%.图8(b)为中等并发规模($CS=512$),尽管可以观察到平均响应时间的减少,但是单独的ORT基本上没有效果.而当$\gamma>1.05$时,可以观察到$SLT=512$时使平均响应时间减少了12%.图8(c)显示了高并发的情况($CS=2048$),



(a) $CS=128$



(b) $CS=512$

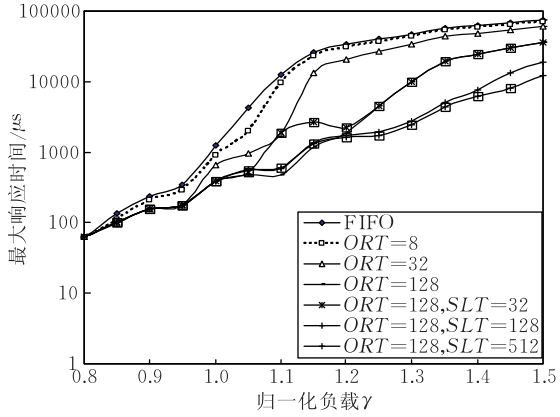


(c) $CS=2048$

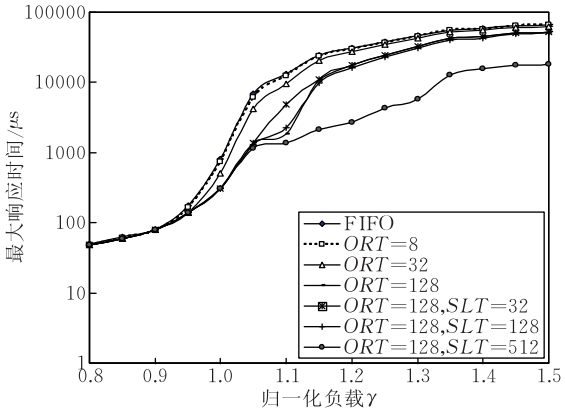
图8 γ对平均响应时间的影响

我们发现 $\gamma<0.95$ 时，没有任何的收益. 在 $\gamma=1.0$ 时，我们观察到对于所有的 ORT 和 SLT 配置都使平均响应时间减少了 31%. 而当 $\gamma=1.2$ 时，平均响应时间减少了 10%. 这个结果表明，如果并发流数高于 ORT 或者 SLT 的大小，我们的方法所获得的平均响应时间收益将会减少.

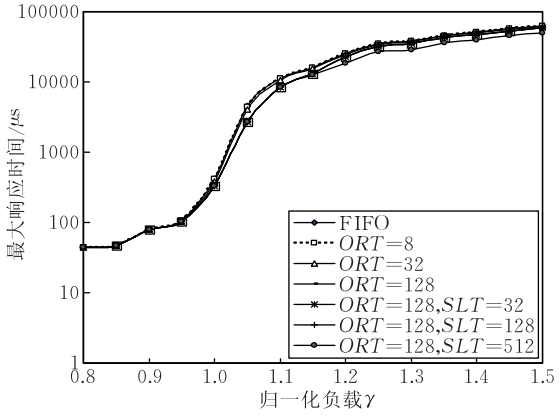
对于最大响应时间，我们也研究了并发流数为 128、512 和 2048 的情况. 图 9(a)、9(b)、9(c)分别显示了对于不同配置的 ORT 和 SLT，归一化负载(γ)带给最大响应时间的影响. 图 9(a)，当并发流数比较低($CS=128$)时，在归一化负载 γ 到达 0.8 之前，最大响应时间上基本没有收益. 在 $\gamma>0.8$ 之后，增加 ORT 和 SLT 的大小才会得到收益，例如 $\gamma=1$ 的情况下，$ORT=128$ 在所有的 SLT 配置下最大响应时间都减少了 68.6%. 更大的 SLT 使得在 $\gamma>1.0$ 时最大响应时间获得更多收益，例如当 $\gamma=1.2$ 时，在 ORT 项数为 128 的条件下，$SLT=512$ 使得最大响应时间减少了 95%. 图 9(b)为中等并发规模($CS=512$)，尽管可以观察到最大响应时间的减少，但是单独的 ORT 效果不明显. 而当 $\gamma>1.05$ 时，可以观察到 $SLT=512$ 时使最大响应时间减少了 91%. 图 9(c)显示了高并发的情况($CS=2048$)，我们发现 $\gamma<1$ 时，基本没有收益. 在 $\gamma=1.05$ 时，我们观察到所有 $ORT=128$ 的配置都使最大响应时间减少了 40.8%. 而当 $\gamma=1.2$ 时，$SLT=512$ 的配置使得最大响应时间减少了 26.9%，而其它的几种配置的收益都不明显. 这个结果表明，如果并发流数高于 ORT 或者 SLT 的大小，我们的方法所获得的最大响应时间收益将会减少. 从图 9 的上可以看出，尽管重排序可能会影响到某些网络流的请求延迟，但是由于减少了最大响应时间，在总体上，重排序的方法并不会对请求延迟造成很大影响.



(a) $CS=128$



(b) $CS=512$



(c) $CS=2048$

图 9　γ 对最大响应时间的影响

在响应时间方面，我们还没有在其它的加速器上进行分析，但是我们预期其它几种加速器上的收益会少于解压缩的加速器. 我们会在将来的工作中对其进行实现和评估.

5　结论和下一步的工作

本文提出了对硬件加速器请求重排序的方法，

同时对片上和片外存储的队列进行优化,可以显著减少有状态加速器状态切换的开销,从而减少处理器的响应时间并增加其吞吐率.实验结果表明,与传统的 FIFO 设计对比,解压缩加速器的吞吐率最大提高了 26.7%,响应时间最大减少了 50%.实验结果还表明,并发网络流数大于 SLT 表的容量时,优化效果将会减小.重排序方法可能会改变某些 CRB 的响应时间,从而影响某些网络流的请求延迟,造成不公平现象,对于这个问题,我们将在下一步的工作中通过限制重排序的时间跨度来使这个影响降到可承受的范围之内,但仅从本文中给出的最大响应时间来看,与无重排序的方法相比,在相同最大响应时间要求的前提下,重排序已经可以提供更高的处理能力.

参 考 文 献

[1] LaPotin D P, Daijavad S, Johnson D P et al. Workload and network-optimized computing systems. IBM Journal of Research and Development, 2010, 54(1), Paper 1: 1-12

[2] Menezes A J, van Oorschot P C, Vanstone S A. Handbook of Applied Cryptography. Boca Raton, FL: CRC Press, 1996

[3] van Lunteren Jan. High-performance pattern-matching for intrusion detection//Proceedings of the IEEE INFOCOM. Barcelona, Spain, 2006: 1-13

[4] Franke H, Nelms T, Yu H et al. Exploiting heterogeneous multicore-processor systems for high-performance network processing. IBM Journal of Research and Development, 2010, 54(1), Paper 2: 1-14

[5] Cooper R B. Introduction to Queueing Theory. 2nd Edition. North Holland, 1981

[6] Yu H, Franke H, Biran G et al. Stateful hardware decompression in networking environments//Proceedings of the 4th ACM/IEEE Symposium on Architecture for Networking and Communications Systems. San Jose, CA, 2008: 141-150

[7] Siriwong K, Lipsky L, Ammar R. Study of bursty Internet traffic//Proceedings of the 6th IEEE International Symposium on Network Computing and Applications (NCA 2007). 2007: 53-60

[8] Johnson C, Allen D H, Brown J et al. A wire-speed PowerTM processor: 2.3GHz 45nm SOI with 16 Cores and 64 threads//Proceedings of the 2010 IEEE International Solid-State Circuits Conference. 2010: 104-105

[9] Bohrer P, Peterson J, Elnozahy M et al. Mambo: A full system simulator for the PowerPC architecture. ACM Sigmetrics Performance Evaluation Review, 2004, 31(4): 8-12

[10] Ziv J, Lempel A. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 1977, 23(3): 337-343

MA Yi-Ke, born in 1980, Ph. D. candidate. His research interests include computer architecture and memory systems.

CHANG Xiao-Tao, born in 1978, Ph. D.. His research interests include computer architecture and low power design.

FAN Dong-Rui, born in 1979, Ph. D., associate professor. His research interests include low power design and processor micro architect.

LIU Zhi-Yong, born in 1946, Ph. D., professor, Ph. D. supervisor. His research interests include algorithm, parallel processing and memory systems.

Background

Network-optimized applications are constrained by ingress and egress data rates, latency and throughput requirements, and the temporal or streaming nature of the data as the current link rates are approaching 40Gb/s. To address the computational and I/O demands of this domain, processor chips are increasingly built as Systems on a Chip (SoC), integrating massively multi threading cores with network interfaces and generic and application specific accelerators onto a single chip. Acceleration opportunities present themselves at various points in packet processing. Compared to stateless accelerators, stateful accelerators do not need to wait for all the packets of a flow to be reassembled. It can process any packet of the flow as it arrives. Dependent on the accelerator, frequent context switching can introduce performance overhead, especially when the context size is large. In general, requests are fetched from a queue and processed by the accelerator in FIFO order. The approach to reducing this overhead is to focus on the acceleration request queue and reordering requests to reduce the number of required context switches. In particular, our contributions are the reordering design of a hybrid on-chip/memory based request queue with limited resources, and the simulation based performance evaluation of this design. This work is under the support of the National Natural Science Foundation of China (No. 60736012), National Grand Fundamental Research 973 Program of China (No. 2005CB321600) and Beijing Natural Science Foundation(No. 4092044).