端口间缓存共享技术研究

Baoliang Li, Zeljko Zilic, Wenhua Dou,

***Abstract*—本文提出了种基于端口间缓存共享的方法。**

***Keywords*—Networks-on-Chip (NoC), buffer sharing**

I. Introduction

Proper buffer sizing and organization are essential to achieving optimal network performance

画 2 张图表示静态缓存分配存在的问题：（1）3x3 mesh 的例子说明端口之间的利用率不平衡；(2) 同一个端口内 VC 之间的利用率不平衡。以 3x3 的 mesh 网络为例，在热点流量情况下（中间节点为热点）来说，热点的 N 和 S 端口接受的流量是 W 和 E 的 3 倍。因而如果能让 W 和 E 端口使用更多的 buffer，那么那么对于因减少流量控制造成的 stall 而引起的大的延迟是非常有好处的。最理想的办法是，令每个端口使用 $B_p = B_{total} \times \frac{C_p}{\sum_{p=1}^{N} C_p}$ 的 buffer. 然后对于每个端口所能使用的 B_p 个 slot 进行动态的管理。此外，端口间的 buffer 共享还有利于改进公平性以及不同数据流的吞吐率。最后指出，只有将端口间和 VC 间的缓存共享同时实现才能够提高和改进 buffer 的利用率，以实现用最少的 buffer 来实现最高的性能。因为 input buffers account for a large fraction of the overall area and power budget of typical Network-on-Chip。

再画一张图表示当前动态 VC 分配所面临的信约耗尽问题。并指出 Lai 等人提出的拥塞避免算法可以避免端口间的拥塞，但是端口内同一个 VC 的拥塞却无法解决。Standford 大学的方法可以解决这一问题。

当把这种共享缓存形式的路由器应用于 CPU-GPU 混合的系统中时，带来的问题会更大，主要原因在于：GPU 流量很大会很快将 VC 耗尽，而 CPU 流量很小由于信约太少而导致过大的延迟。而 CPU 程序通常是延迟敏感的。

另外，完全动态的 VC 分配还会增加 VCA 和 SWA 的压力，而这两个流水段通常又处于路由器的关键路径上。

再画 1 张图表示当前动态方案可能面临的死锁问题以及相应的解决办法。

对相关研究现状的描述：Lai 和 VichaR 的方法只解决了端口内的 VC 动态分配问题, 此外 DAMQ 的方法访存周期较长（3 拍），VCR 的方法也只适用于同一个端口内，而且所解决的是在故障情况下的性能平滑下降问题。对于端口间的动态缓存管理，有一类方法方法采用完全共享的 centralized buffer，这类方法属于精粒度的共享；还有一类方法采用不完全共享（DAC11），但是该方法不支持 VC。而 MH 的方法只有在某个端口发生故障的情况下其对应的 buffer 才能够被其它端口使用，而且这种方法是全动态的，因而硬件开销很大。采用链表的方式实现的完全 VC 共享会消耗大量的存储资源和芯片面积，standford 大学的研究结果表明这类方式虽然提高了资源的利用率但是开销是十分具大的，以 16 个 VC 为例，大概会引起 60% 的开销。另外一方面，完全静态的方式又会严重影响性能。

对我们的方案的简单介绍：因而我们的方法是在二者之间进行折衷，采用部分动态的 VC 共享方式，每个端口分配一定数量的容量较小的静态 VC（大小为可实现 100% 吞吐率的最小值），端口之间共享一定数量的动态 VC。每个端口只有当新到达的报文与被阻塞的报文不属于同一条流而且占用动态 VC 的数量小于其配额时才会启用动态 VC。

我们方案对上述几个问题的解决方法：（1）当某个端口的流量较少时，静态分配的 VC 足够支持切片数据的无阻塞移动。而当某些端口的流量较多时，共享的 buffer 则可以用来缓存大量的切片。由于静态分配的 VC 相当于是为每个端口预留的配额，因而可以有效的避免共享缓存被某个端口耗尽时对低速端口的影响。（2）同时，为了减轻 VCA 段的压力，我们可以还采用 VVC 映射的方法来动态的调整对动态缓存的使用。因为 VCA 通常处于关键路径上，这样的做法可以不影响路由器的关键路径。另外，我们所有的新增的硬件都可以并行的独立的操作，而而不对路由器的关键路径产生不利的影响。（3）由于对静态 VC 的访问消耗的能量要比动态 VC 要少，因而我们的

Baoliang Li and Wenhua Dou are with the College of Computer Science, National University of Defense Technology, Changsha 410073, P.R. China

Zeljko Zilic are with Department of Electrical & Computer Engineering, McGill University, Montreal H3A-2A7, Quebec, Canada

Manuscript received XX XX, 2014; revised XX XX, 2014.

设计的功耗效率也要比 MH 的方法要好。（4）对于死锁，我们的方案由于存在静态配置的 VC，因而采用合理的分配方案则可以实现死锁避免，而在 MH 的方案中，死锁是通过逃逸通道来避免的。

当前几乎所有的路由器都是输入排队的，因为输出排队需要每个端口以 P 倍的加速比运行才能达到 100% 的吞吐率，而输入排队的路由器通常只有不到 80% 的吞吐率。另外，输出排队由于需要复杂的仲裁等机制，功耗通常是输入排队系统的 1.5 倍以上。

利用链表的好处是可以充分利用存储资源，但是缺点是硬件开销太大。因此对于这些共享的存储体我们有以下几种组织方式。对每个 memory bank 设置一 1 个存储控制模块这样使用的总的控制寄存器的数量要比为每个端口所有的存储体设置一个共用控制器要节省很多的硬件资源。因此，我们的方法从两方面节省了开销，一方面是几个小的控制器；另一方面是我们提出的混合式的存储管理方法的动态存储容量要比同等大小的完全动态的管理方式要小很多。因，最后可以画一张表并给出一张图列不这些比较的结果。

我们的 SWA 要进行修改，由于属于同一个输入端口的不同的存储体可以支持并行的读写，因此 SA 段可以进行修改以支持这个特性。在通常的路由器中，每个输入端口有 1 个 $V_s:1$ 的仲裁器，每个输出端口有 1 个 $P:1$ 的仲裁器；在 MH 的方案中，每个输出端口有 1 个 $V_{max}:1$ 的仲裁器，每个输出端口有 1 个 $P:1$ 的仲裁器；而我们的方案采用：第一阶段由 5 个 $V_s:1$ 和 N 个 $V_d:1$ 的小的仲裁器组成（$V_s+N*V_d=V_{max}$）；而第二阶段则与之前的方案相同。通常逻辑综合可以发现我们的方案由于每个仲裁器都比较小，因而可支持的最高时钟频率也最大，另外硬件开销也可以进行一些比较。

还有就是硬件开销，由于 MH 的方案每个端口有一个较大的缓存器和一套控制逻辑，假设每个端口有 B 的缓存资源；那么 free buffer tracker 的开销为 $B\times log_2B$，header pointer 的开销是 $V_{max}\times log_2B$，tail pointer 的开销是 $V_{max}\times log_2B$，下一跳指针的开销 $B\times log_2B$；5 个端口总的控制开销为 $10\times(B+V_{max})\times log_2B$；而在我们的方案中，由于总的缓存资源中有一部分会被独立分配给每个端口做为静态 VC，而且剩下的动态缓存资源又被分为几个小的 memory bank, 每个 memory bank 需要的控制资源比较少，我们假设有一半的缓存分配给静态 VC，那么我们的方法控制开销为 $5\times(B+2V_{max})\times log_2(B/2)$；

另外，我们采用的 VC 重命名技术也在一定程度上简化了 VCA 的实现。关于 VCA 的实现，传统的方法在输入端采用 V:1 仲裁器，在输出端采用 PV:1 仲裁器；MH 的方法输入端采用 $V_{max}:1$ 仲裁器，输出端采用 P:1 仲裁器；而我们的方法，在输入端采用 n 个 $V_i:1$ 仲裁器，其中（$\sum_i^n V_i=V_{max}$）, 在输出端采用 nP:1 仲裁器。采用几个小的仲裁器并行工作来代替一个较大的仲裁器的好处是降低硬件复杂度并易于提高主频。（考虑到同一个端口之间不需要进行匹配，能否利用 n(P-1):1 来代替）由于我们的方法在输入端有更多的请求胜出，因而更加有利于提高匹配的成功率。由于我们的方法在 VCA 以及 SWA 上都有较高的匹配成功率，可以预见我们的路由器可以提供较高的吞吐率。

由于每个 memory bank 有独立的操控逻辑，因而我们支持的最大 VC 数可以比 MH 的方法要大。

VCA 也需要修改。假设每个端口有 m 个静态 VC，那么 VCA 的实现规则是对于所有请求，优先分配这 m 个静态 vc，对于暂时没有被分配给静态 VC 的请求，则为其分配一个各不相同的标识符 VCID，通过该 VCID 在下一级路由器中再进行动态的 VC 分配。我们称这一个步骤为 VC 重命名技术，下一级可分配的动态 VC 数量与其可使用的动态 bank 数有关。通过使用 VC 重命名技术，我们的设计极大的简化了 VC 分配器的复杂度，这也为提高系统的主频带来很大的好处。因为 VCA 通常在关键路径上。这样一来，我们的 VCA 的实现电路得到了极大的简化，画一张图给出对应的电路设计方法。为了支持这一改变，信约控制模块也要进行相应的改变，有 m+1 个信约计数器。

这些共享的存储体每一个都独立的检测不同输入端口的流量状态（这些流量状态由若干个统计计数器来实现），并通过协商来确定每个存储体的分配方式以及分配的实际地址段。再加上我们的 SWA 段支持同一个端口的多个 bank 并行输出，因而有希望得到更高的吞吐率。在设计的过程中需要注意 free-buffer-tracker 的实现方法，这涉及到某个 bank 能否很快被别的端口使用。但是同时也会对系统的吞吐率有一些影响。

A. 流量控制模块

1) 最初的方法: 我们的流量控制模块有两根线，每根线负责其中模块中一个 buffer 段的信约，信约控制器自动进行信约维护。

我们的方法最重的一个特点是不增加 VA 和 SA 段的复杂性，因为这两段都处在路由器的关键路径上。

TAMS 的硕士论文中指出，ViChaR 的方式提高了一个端口内部的 buffer 利用率，但是也提高了设计的复杂性和功耗。

DAMQ 的思路与 ViChaR 的区别在于 DAMQ 采用固定数量的 VC，因而会出现 HoL。

我们提出的是一种运行的 VC 自适应技术，之前基于 RTC 的研究属于面向应用的 buffer sizing 方法，属于设计时的方法。

ViChaR 和 RAVC 的方法都需要对整个路由器进行重新设计，因而成本高。而我们的新方法则主要是在原来的基础上增加了些硬件来提高 buffer 的利用率。在做实验的时候我们可以统计每个 buffer 的利用率。

最终我们要与 conventional router 进行比较。而且重新设计和实现 sink 端以统计各种信息。

inter-port buffer sharing 是我们论文的主题。

MH 提出的 RAVC 方法虽然支持动态 VC 数量以避免 HoL，但是控制罗男过于复杂，硬件成本过高，需要大量的额外的存储器和寄存器文件来存储控制信息。而我们的方法增加的硬件几乎都是简单的组合逻辑电路，因则硬件成本更低。由于我们采用可扩展的 VC 结构，因则降低了拥塞端口发生拥塞的可能，在一定程度上对避免 HoL 也有帮助。

buffer 借用在热点流量情况下可能很有用，效果会很明显。而热点流量恰是 Cache 一致性能及 MC 访问的重要应用场景。

为了简化硬件设计，我们只允许每个 VC 被偷一次。

当需要进行 VC 间缓存共享时，通常的情况是这个端口已经很堵，因则借助端口内的缓存共享来改进性能的性能提升已经非常有限，而端口间缓存共享则没有这个问题。因为应用通常会呈现出流量在同一个路由器的不同端口间的不均衡分布情况，因而采用端口间共享对改进吞吐率有好处。当然，通道间共享和 MH 的工作都可以有效的防止 HoL。因而二者熟优熟劣需要再进行考虑。当然，对于 HoL 问题，我们的方案可以在选择待偷端口时适当的考虑可能发生的阻塞。对于非均匀流量，特别是热点流量情况下，端口间的流量分布是严格不均匀的。

在进行仲裁时，应当给借用的 buffer 空间切片以较高的优先级，以方便其清空队列。借用冲突可以通过与 victim 端口的协商来实现，theif 端口和 victim 端口之间可以通过 req/ack 来交互。

buffer 的读写冲突可以通过独立的两对读写指针来实现，高位读写指针地址以 1 开头，低位读写指针地址以 0 开头。

本文的一个重要的目标是改进现在的 NoC 的 buffer 利用率，增强性能的同时避免不可接受的硬件开销以及影响关键路径。

我们的设计中，每个 VC 由两个 FIFO 组成，prime_readd_ptr 和 prime_write_ptr 在非共享状态下用最高位做片选，在共享状态下由 share 信号选择高位地址，其自己的最高位置零。

我们的设计的目的在于同时不引入较大的硬件开销

提出一种新的流量控制方法，两根线，每根线负责维护信约模块中一个 FIFO 段的可用 buffer 空间。

之前的 buffer 共享策略没有考虑到恶意流占用很多 buffer 的情况，因而会对系统的加速比产生很大的影响。在具体做实验的时候我们可以通过综合流量发送 1000 个报文所用的时间来进行分析和说明。而我们的方法由于限制每个 VC 所用的 buffer 空间最大是 3/2，因而会在一定程度上减轻这一影响。

模块划分：

1. 流量控制模块：信约生成、信约跟踪。

2. 借用 buffer 的管理模块：处理借用和非借用状态下的地址译码以及读写，仲裁，队列空满检测模块。

3. 借且 buffer 读写控制模块。

4. 借用 buffer 跟踪模块。

5.buffer 借用状态机：buffer 选择，协商，确认和释放。

2) 新想法: 共享的 buffer 分成几个 bank，用以支持动态的 VC 分配和端口间的缓存调整。用一个状态机或者什么东西来定期的将空闲的 bank 分配给指定的比较忙碌的端口使用。

如果到达的报文不属于静态 VC 队头报文的数据流，那么就为他分配一个新的 VC，并将该切片写入新的 VC，多静态 VC 处移到动态 VC 后应当立即向源端返回一个信约，除非动态 VC 的缓存空间已经已经用完。之后每到达一个报文先看其是否属于某个数据流，如果都不属于，那么就为其分配新的 VC，否则就写入相应的 VC 中去。这样做的目的是防止报文序。

II. 实验与评估

缓存资源的使用比较：（1）列表比较资源；（2）实验比较性能，包括吞吐率和延迟；（3）综合结果比较面积和功耗。

III. 代码阅读重要提醒

Router_Checker 在做综合的过程中要去掉。

VCR 是带虚通道的路由器，RTR 是 SA 与 VA 结合的实现方法，WHR 是没有虚通道的实现方式。

代码中的 buffer_size 是一个端口的 buffer 总量，每个 VC 的 buffer 数量还需要除以 VC 数量。

rtr_flit_buffer.v 输入端口的管理模块

原子分配的时候这种优化策略可能没有效果。一般的 VC 路由器吞吐率比较高，原子 VC 分配只有当切片的信约返回时该 VC 才可用，而一般的 VC 则在尾切片离开

VC 时即可以再次进行分配。原子 VC 分配当 VC 很多时可以达到很高的性能。

在进行路由器综合时，mesh 型拓扑结构中某些边缘路由器的某些端口可以不生成。

代码中的 message_class 和 resource_class 都是指什么意思？

拓扑结构的连接性包括线、环和全互连。

路由器的端口数量等于每个维度的邻居数乘以网络维度再加上每个路由器的节点数。

代码中的 flow_control_bypass 是什么意思？

link_ctrl 是指功耗控制相关的。flit_ctrl 呢？

看一下 flit_buffer 的寄存器文件的实现方案。

随机拓扑的生成可以采用 TGFF 工具。

注意在 VC 和 SW 仲裁时 VC 非空和满时的选项

VC 分配是否偏向空队列的选项

almost full 的含义是只剩下 1 个 buffer 空间

Acknowledgement

The authors thank the reviewers for their suggestions and comments, and all the experiments are carried out at the Integrated Microsystem Lab (IML) of McGill University. This research is supported by High Technology Research and Development Program of China (Grant No. 2012AA012201, 2012AA011902).