端口间缓存和 vc 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

引言的写作思路：（1）通常的路由器结构是每个输入端口有自己的缓存；这些缓存资源被组织成不同的虚通道。（2）这种路由器的结构存在很大的问题，最关键是缓存的利用率不均衡而且很低。同一个端口内不同 VC 的利用率不同；不同端口间的缓存资源利用率也不同。（3）针对前一种现象，ViChaR 被提出；之后 Lai 提出另一种基于链表的等效而且低成本的实现方式；（4）针对后一种情况，xx 提出了 partial sharing 的方法，该方法假设某些路由端口之前有一部分 FIFO 是可以被两个端口共享的；但是，在任意时间该共享 FIFO 里只能存储一个端口的报文切片，当该报文的后续切片被阻塞时，该共享 FIFO 的利用率非常低；MH 提出了一种细粒度的端口间的缓存共享（两篇论文）的方法；（5）以上的两种方法由于支持的 VC 数可以很多，导致 VC 和 SA 仲裁非常复杂；另外这种方法都具有一个共同的问题：需要大量的控制存储资源来支持动态的 VC 分配和使用。（6）另外，以上提出的两种提高缓存利用率的方法都是基于输入缓存的路由器结构；在这种结构中通常采用 VC 和 SA 仲裁来消除冲突，由于 Allocator 的匹配效率很低，导致路由器的吞吐率不高；相反，输出缓存的路由器具有很高的吞吐率，但是需要 Crossbar 和 Memory 有一定的加速比，难以在片上实现。在 xx 中提出一种 DBS 结构，这种结构采用分布式的存储器来模拟输出排队，在 xx 情况下可以达到 97% 的吞吐率。但是，这种结构是另一种缓存共享的结构，可以极大的改善输入排队路由器的吞吐率。尽管分布的缓存资源可以被所有的端口使用，实现了端口间的缓存共享。但是，这种结构的缓存利用率依然不高，我们举例说明如下。

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.

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 $V_s:1$ 的仲裁器用于选出该输入端口胜出的 VC（在这一阶段解决了 FIFO 读冲突），每个输出端口有 1 个 $P:1$ 的仲裁器用于选择最终胜出的端口（在这一阶段解决了 FIFO 的写冲突）；在 MH 的方案中，每个输入端口有 1 个 $V_{max}:1$ 的仲裁器，每个输出端口有 1 个 $P:1$ 的仲裁器；而在我们的方案中第一阶段有 1 个 $V_s:1$ 和 N 个 $V_d:1$ 的小的仲裁器组成（$V_s+N*V_d=V_{max}$）；而第二阶段每个输出端口有 1 个 $(5+N):1$ 的仲裁器。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 P 个 $V_{max}:1$ 仲裁器，输出端采用 1 个 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。每个下游节点的 memory bank 配置发生变化时都会通知 VCA，VCA 据此来屏蔽不能用的 VC 段。这样就不会存在冲突了。

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。注意：VC 的重命名技术可能已经在 MH 和 Lai 的论文就已经实现。

我们的 VA 和 SA 都是基于优先级的，VA 阶段优先分配静态 VC，SA 阶段优先仲裁静态 VC。

有可能在为报文指定 Memory bank 时采用一定的策略，以提高其吞吐率。

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。

之前 MH 的方案还有下面的问题：由于每个端口的报文或者都存在一个 memory bank 里，或者与其它端口的报文存储在一起。这样的方法有以下问题：（1）去往不同端口的报文无法并行传输；我们的方案采用多个并行的存储体以减少这一影响。画图展示这一影响。实际上就是一种特殊的 HoL。假设端口 W 到达的切片目的地为 E 和 L，端口 N 和 S 都只产生目的端口为 E 的报文；在 RR 调度方式下，平均每 3 个周期服务 1 个 W 的 E 报文；而实际上，W 端口中向 L 端口的切片在高度向 E 端口的切片时是无法前进的，由于读冲突的存在。

片上网络中存在的几种形式的阻塞：因 VC 分配失败造成的阻塞对性能的影响最大，之间提出的动态 VC 分配实际上是在降低因为这一种形式的阻塞进而提高吞吐率降低延迟的。但是，这类动态 VC 分配方案的本质是为每个端口分配大量的 VC，并实现同一个端口内的不同 VC 的动态缓存分配（为了防止死锁，每个 VC 需要至少保留一个缓存空间, 剩余的缓存空间则可以在不同的 VC 之间进行动态的分配。）实际上，造成 VC 阻塞的原因是大量不完整的报文（即尾切片还没有到达）占据了所有的 VC。存在大量不完成报文的原因主要是因为阻塞造成的同一个报文的不同的切片跨跃存在于几个路由器中。换句话说，这些路由器中都会有一条 VC 被该报文所占据而无法被其它报文所利用。由于不同端口的流量情况不同。但是，某个端口的不完全报文的最大数量等于于上一级的所有路由器中需要向该路由器注入流量的 VC 的总数。VC 保证在同一个通道中的报文切片顺序不会交错。

我们的方案为每个端口预留一定量的 VC，然后剩余的 VC 被所有的端口共享。这样做的原因是为了为每个端口保证一定的吞吐率和带宽以避免当该端口出现猝发流量时没有可用 VC 的情况。预留不同数量的 VC 最终的性能是不同的，对于不同的流量模式可以进行多次实验找出最优的组合。目测发现这种部分虚通道共享方案对于热点流量效果会比较好，而热点流量又恰好对应于 CMP 系统中的 MC 或 home 节点。

之后 MH 提出的端口间缓存共享的方案实际上实现了端口间的缓存共享, 以减少因流量控制而造成的吞吐率下降和延迟增加。但是因为流量控制而造成的阻塞造成的影响相对虚通道不足造成的影响较小。

我们提出的方案同时实现了端口间的缓存共享和虚通道共享，特别是端口间的虚通道共享对改进系统的吞吐率是非常有好处的，因为 VC 阻塞通常要较多的时钟周期。我们在做实验的时候还可以考虑这样一种情况：对于 conventional router、damq 路由器和我们提出的路由器结构而言，为了达到同样的性能，我们的方案在需要的缓存资源方面的节省情况。由于实现了端口间的 VC 共享，我们的方案还可以实现的另一个目标是减少每个 VC 仲裁器的开销。因为这个仲裁器通常处于路由器流水线中的关键路径上。为了采用较小的分配器来实现大量动态 VC 的分配，我们的方案要求每个输入端口要有一个 VC 分发逻辑。该逻辑还要实现 VC 的重定向以实现流量隔离，避免某个恶意数据流占据所有的缓存空间。

我们的方案具体如下：对于之前的共享 VC 方案，假设原来的动态 VC 方案中每个端口有 x 个 VC，那么我们的方案每个端口设置 5x/6 个 VC，另外还有 5x/6 个 VC 为各个端口间共享的。与之前的方案，对于同样的 VC 数量，每个 VC 仲裁器的复杂度则有所下降，而 SW 仲裁器的复杂度暂时无法分析得之。采用 VC 共享方案的本质原因是为了尽可以避免 VC 阻塞造成的性能下降。对于共享的 VC 我们称之为影子 VC。原因是 VC 仲裁器没有专门的请求端口，而且与每个端口的对应 vc 共用一个请求端口，因为被分配的 VC 在尾切片没有到达之前是不会 grant 任何请求的，因而该端口可以被影子 vc 所使用。在

我们的方案中，每个端口所能使用的 vc 数量在 5x/6 和 10x/6 之间，远远大于之前的设计方案，但是所用的 vc 分配器规模则远远小于之前的方案。实现这一目的的根本原因在于我们实现了 vc arbiter 的复用。为了区分这两类 vc，我们需要在原来的 channel 总线中增加一根线表明当时到达的切片的 vc id 是对应于影子 vc 还是本原 vc 的。另外，我们还需要在 channel 总线中增加 5x/6 根线来表明每个影子 vc 是否可以被该端口用于 vc 分配。这些线由下流路由器中一个集中的控制逻辑驱动，该控制逻辑会根据每个端口对应的私有 vc 的占用情况来决定对应的影子 vc 是否分配给这个端口。5x/6 个共享 vc 也是动态的分配缓存空间，为了避免写冲突，要求这些缓存空间被分成若干个小的存储体，这些存储体可以被并行的读写。每个存储体有自己的 sw arbiter，因而我们的方案中 sw allocator 也需要重新设计。由于 crossbar 的端口数量也有所增加，因而也会很大程度上提高吞吐率。总起来说，我们的方案与 conventional router 相比实现的性能提升源于三个方面：（1）实现了动态的 vc 分配，提高了缓存的利用率，可以利用较少的缓存资源实现同样的性能；（2）实现了 vc 分配器的共享，进而降低了 va 的复杂度和实现成本；（3）实现了端口间的 vc 共享，通过发现不同端口间对 vc 数量的不同需求和共享，我们的方案实现了端口间 vc 共享进而降低了因 vc 不足造成的阻塞；（4）我们的方案通过发现不同端口间对缓存容量的不同需要，动态的分配缓存资源到不同的端口，因而实现了资源的高效利用并以此来提高片上网络的性能。与之前提出的动态 vc 分配的方案相比，我们的方案由于利用了分配器的复用技术，进而实现了利用较小的 allocator 来实现对大量 vc 的分配和使用，在很大程度上降低了 allocator 的实现成本并降低了关键路径长度，对于提高系统的主频非常有好处。为了支持这种 vc 借用技术，我们的 sw 分配也需要进行相应的修改。修改后的 sw 仲裁器也是分为两人个附件，第一个阶段每个端口有 1 个 5x/6:1 的仲裁器用于从该端口中选择一个 vc 作为下一时钟的备选输出 vc；另外，共享的 5x/6 个 vc 也有一个仲裁器，因此第一阶段我们有 6 个同样的仲裁器。第二阶段我们在每个输出端口放置 1 个 6:1 的仲裁器，用于从每一组仲裁结果中选择出一个 vc 进行切片传输可以看出，我们的 sw 分配器第二阶段采用 5 个相对较大的 6:1 仲裁器 (原来的方案是 5:1)，但是在第一附件每个仲裁器的规模都比原来的方案要小（但是原来的方案需要 5 个我们的方案需要 6 个）。在具体的实现过程中，我们还可以为共享 vc 分配器的决策结果赋预较高的优先级以帮助其尽快排空队列以为其它端口所用。另外，我们的 vc 分配应该只适用于原子 vc 分配？因为我们的方案中集中的控制器需要根据每个端口的 vc 的占用情况来动态的分配 vc。在最终的实验比较中，我们可以考虑对于同样的性能，三种路由器的实现方案所需要的缓存资源各是多少。另外，我们还可以与 mh 的方案进行比较，mh 的方案为了实现 buffer 共享，每个端口的 next buffer slot ,header ptr 和 tail ptr 应该都是可以寻址到该路由器中任何一个 buffer slot 的。因此，这种共享方案的控制存储应该是非常耗资源的。

在我们的方案中，为了简化 VC 分配逻辑，我们在 Vc 分配和 sw 分配时给影子 vc 赋预较高的优先级，以让其尽快排空队列，这样方便其它端口使用。另外，共享的缓存区被分成 5 个或是 5 的整数倍数存储体，我们对 vc 的分配以存储体为粒度，如果某个存储体被分配给某个端口，那么对应于这个存储体的所有的 vc 也被赋予这个端口。当属于某个端口的共享存储中所有的 vc 都空闲时该存储体便可以被重新分配。之所以称之为影子 vc 是因为我们的方案中影子 vc 与端口的私有 vc 共享同一个 sw 端口和 va 端口，这样即实现了对大规模 vc 的支持双降低了设计成本。而实际上这样的设计并没有降低系统的性能。

影子 vc 重新分配的前提是该共享存储体上的所有 vc 都空间（由于优先分配私有 vc 因此这种情况会经常出现），集中的控制器根据每个端口的需要进行动态的分配。每个集中的控制器提前若干周期预测每个端口的 vc 需求然后将 vc 回收的结果提前告知占用该 vc 的端口，在收到 vc 回收请求以后，该端口停止对相应共享 vc 的再分配，等该 vc 被排空后该端口向集中控制器告知该共享 vc 可以进行再分配。集中的控制器是一个有限状态机，该状态机负责对几个共享存储体的分配和调度，每个共享存储体有一个自己的状态机用于协调与各个端口的分配/回收请求。

利用 booksim 做实验验证每个端口的 vc 需求量与时间的关系，关画图。测试的方法是为每个端口设置一个极大的 vc 数量然后在仿真进行过程中记录每个端口实际使用的 vc 数量。注意，对于不同的 vc 预设数量所得的结论可能不同。而且极有可能是线性增长的，这个时间我们可以假设每个端口的 vc 数量为与我们的实验参数相同的静态 vc 数量，然后看看在不同的流量情况下每个端口实际需要的 vc 数量。由此引入本论文的主要目的，即通过动态共享 vc 来降低每个端口报文的 vc 分配失败造成的阻塞和停顿进而降低平均端到端延迟提升吞吐率。

我们需要强调 vc 数量与吞吐率间的关系：vc 数量越大吞吐率越高（这一点可以利用 booksim 的仿真结果进行佐证）。因此，我们参数动态共享不同端口的空闲 vc 资源可以动态的提高系统的吞吐率。我们的方法对吞吐率的改进还体现在我们的方法中共享的缓存有各自的 crossbar

端口，因而对提高 crossbar 的通过率有好处。当然，在最初实现的时候为了修改代码方便可以考虑采用 5x5 的 crossbar 然后在进行 sw 分配前对私有 vc 和共享 vc 进行复选。

端口间的缓存共享可以提高 buffer 的利用率，使得采用较少的 buffer 实现较高的性能成为可能。但是，完全的端口间缓存共享（例如 mh 和 mit 女人）需要非常复杂的逻辑支持以及大量的硬件支持，而这些复杂的硬件又会进一步增加系统的功耗。mit 的研究表明尽管共享提高了吞吐率但是功耗却大大增加这样也就在很大程度上降低了这类方案的可用性。我们的方案实际上是在不共享和完全共享之间的一个折衷，通过仅共享一部分缓存空间，我们使用对共享缓存的控制存储开销大大降低，列一个表给出我们的方案与 mh 方案需要的指针寄存器的数量。最终还可以列一个表给出我们的功耗与 mit 的方案比较的结果。而且之前的方案都是只共享了缓存而没有共享 vc，当某个端口的 vc 全部被占用时我们的方案还可以借助其它端口的空闲 vc。

实现 vc 共享的目的在于，受限于 vc 和 sw 分配器的实现复杂度，每个端口可用的 vc 数量是非常有限的。另外，受限于系统的设计成本，每个端口的缓存容量也是有限的。而系统所能达到的吞吐率与 vc 的数量以及缓存容量是正相关的，因此为了提高吞吐率需要较多的 vc 和缓存空间。但是不同端口的流量特点是不同的，在这种情况下，我们需要一种有效的机制来动态的共享 vc 和缓存并对其进行高效的分配和使用。之前的方案仅实现了对缓存的动态分配和共享而没有实现 vc 的共享。然而我们发现因 vc 不足造成的阻塞要远远比缓存不足造成的后果要严重。而且这种影响随着报文长度的增加而越发明显。vc 是维持报文内部切片以正确的顺序传输的有效手段。

在全文的主题上，我们的目标是提高吞吐率降低平均端到端延迟，而之前的很多方案都是提如何提高缓存利用率并降低延迟。

考虑某种情况，当某条流在某个路由器中发生 VC 阻塞时，该条流会在路由器中积压大量的切片，而切片在缓存区中的积压会将该数据流转变成为恶意数据流。而造成这一现象的原因则是因为虚通道分配失败。因而我们的方案通过避免这种虚通道分配失败来提高吞吐率将对减轻恶意流的影响也会有好处。

A High-Throughput Distributed Shared-Buffer NoC Router 应该值得参考。其提高吞吐率的方法与我们的方法很像。类似的文章还有 Achieving High-Performance On-Chip Networks with Shared Buffer Routers。但是，这两篇论文中给出的方法都不支持 VC，是典型的虫孔路由器。

在后面的研究中可以考虑在动态 vc 分配的基础上实现流量的隔离和服务质量保证。

有关队列空满的信号由 free buffer tracker 模块产生。

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。

在做实验时可以考虑不同的报文长度以模拟 cache 一致性协议的需求，例如将报文长度取双峰分布，一个长度为 1(对应读请求或者写响应消息) 一个长度为 16（对应 cache 行）。

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。

我们的共享虚通道技术为每个端口预留少量的 VC 以减少 VC 仲裁和分配的压力，为了保证最差情况下的延迟性能，我们需要一种技术来确定每个虚通道所需要的最少的 buffer slot 数量。这就是我们之前基于 RTC 的论文所要解决的问题，由于在动态 VC 共享方案中每个 VC 只需要保留 1 个 buffer slot 便可以保证无死锁。我们的方案采用 RTC 的方法来确定每个 VC 所需要分配的最少缓存区。对于热点流量来说，特别是热点处于 mesh 网络的四个角上或者四条边上的时候，热点路由器和一些其它的路由器总有某些端口的缓存是没有被利用的。不光这些缓存区是没有被利用的，就连这些端口所对应的 VC 也是无法被有效使用的。因此我们可以考虑共享一部分缓存区和 VC，之前 MH 的方案中 VC 是每个端口所独有的，没有被共享；为了保证性能，需要为每个端口预留大量的 VC 才能保证性能。但是这无疑增加了硬件成本降低了资源利用率。共享缓存技术除了增加资源利用率以外，还增加了实现了动态 VC 分配中间出现的流量隔离技术。

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_wrap 是路由器的顶层模块，根据不同的配置参数，该模块可以实例化不同的路由器实现类型，例如虫孔路由器和虚通道路由器以及混合路由器。由于我们的研究是基于虚通道路由器的，因此 router_wrap 会被实例化为 vcr_top 模块。在 vcr_top 模块中，为每一个输入端口实例化化一个 vcr_ip_ctrl_mac 子模块，并为每一个输出端口实例化一个 vcr_op_ctrl_mac 子模块，整个跌幅器还会实例化一个 vcr_alloc_mac 模块用于进行 vc 和 sw 的分配。另外，整个路由器还会实例化一个 crossbar 模块。

在 vcr_ip_ctrl_mac 模块中包括的子模块有 rtr_channel_input 和一个 rtr_flit_buffer 模块以及一个 rtr_flow_ctrl_output 模块，并为每个 vc 实例化一个 vcr_ivc_ctrl 子模块。

vcr_channel_input 模块的作用是：在路由器之间进行连接的时候将所有的控制信号和数据信号都汇总在一起用 channel 总线来表示这样可以简化路由器连接的实现，而这个子模块的作用则是在路由

器内部将 channel 总线中的信号剥离出来以便进行相关的操作。这个子模块的输入信号只有 channel 总线而输出信号包括：flit_valid_out, flit_head_out, flit_head_out_ivc, flit_data_out 和 flit_sel_out_ivc。flit_data_out 和 flit_valid_out 都是经过锁存的输出信号，采用锁存器的目的是实现切片的流水传输。另外，关于切片的 vc id 需要先对 channel 的相关位进行译码才能得到选择器能使用的位向量信号。

rtr_flit_buffer 会根据存储的分配模式确定是采用 damq 或者 FIFO 队列的形式来组织缓存。

rtr_flow_ctrl_output 模块中，flow_ctrl_width 等于 vc 寻址宽度 +1（对应于 credit_valid 信号）. 这个模块输入信号有流量控制有效信号 flow_ctrl_valid 和该事件对应的 vc，这两人个信号一个对应于 sw 分配的授权信号和 vc 选择信号。输出信号 flow_ctrl_out 连接到上级路由器的 rtr_flow_ctrl_input 模块中。flow_ctrl_out 的第 0 位即 credit_valid 由锁存器驱动。信约所对应的 vc 由 select 信号译码后得到，该信号也是由锁存器驱动。

vcr_ivc_ctrl 模块中实例化的子模块包括 rtr_route_filter,rtr_routing_logic 和 rtr_next_hop_addr. 这个模块的实现非常复杂，它的会根据当前的路由器的地址来计算输出端口。这个模拟接受来自 rtr_channel_input 模块译码产生的 flit_valid_in 等信号，另外，这些经过译码的信号还会连接到 rtr_flit_buffer 模块. 这个模块用于计算前瞻路由信息并更新头切片的 lar 域，经过这个模块以后的切片会直接穿过 switch 并进入 rtr_channel_output 模块。在综合的过程中，这个模块中 rtr_flit_type_check 模块需要注释掉。这个模块的主要功能包括：（1）跟踪每个 ivc 的 ovc 分配情况;(2) 为该 ivc 生成 header, tail 的 indicator 信号.（3）更新该 vc 的头信息寄存器；（4）解码头切片中的路由信息；（5）计算下一跳的路由信息；（6）跟踪每个信约的使用情况. 对相关信号的说明：flit_sel_in 信号由 rtr_channel_input 模块译码得到，该信号结合 flit_valid_in 信号可以说明当前到达的切片可以属于该 ivc。该模块同时还输出 flit_valid 信号，当对应的 vc 不为空或者该 vc 有新到达的切片时这个信号有效。表示对应 ivc 已经被分配的 allocated 信号在这个模块中被锁存。flit_tail 和 flit_head 信号的产生方法是：如果当前队列为空那么就看新到达该 vc 的切片是否是尾 / 头切片，否则就根据 flit_buffer 的 flit_tail 和 flit_head 信号。在这个模块中 flit_sent 信号非常重要，因为他是决定各个信号取值的关键。这个信号产生的方法是根据 vc 和 sw 分配的结果来进行的。当新到达的切片是头切片时，header_info_in 信号有效，其它包括了该切片的路由等信息，该信号在 rtr_ip_ctrl_mac 模块中从头切片的前面若干字段中取出得到。该模块中的信约跟踪实现的是对 ivc 空满状态的跟踪，

rtr_next_hop_addr 模块接受的输入包括当前的路由器地址，lar 信息和目的地址信息。输出的是下一跳的路由器地址。这个下一跳地址会作为 rtr_routing_logic 模块的输入用于计算当前切片在下一跳路由器处的输出端口。该输出端口信息在经过译码之后写入切片的 lar_header 中。

代码中所有以 *_active 结尾的信号实际上都相当于是模块使能信号。

vcr_op_ctrl_mac 模块中实例化的子模块包括 rtr_flow_ctrl_input，rtr_fc_state, rtr_channel_output, 并为每一个输出 vc 实例化一个 rtr_ovc_ctrl。rtr_flow_ctrl_input 接受下游路由器的信约反馈信息并对其进行译码和缓存，该模块的两人个输出信号是 valid 和 sel。这两个信号会被输入到 rtr_fc_state 模块，而 rtr_fc_state 模块的作用是跟踪每个 ovc 的信约使用情况。

rtr_ovc_ctrl 模块用于跟踪下流路由器的 vc 使用情况。这个模块实际上只有两个输出信号：flit_sel 和 elig。flit_sel 信号最终会连接到 rtr_fc_state 模块上用于跟踪对应 vc 的信约使用情况。到达该 ovc 的切片来自正确的端口和 ivc 时这个信号 assert 而且被锁存。这个模块根据输入的切片类型以及下流路由器的信约情况来确定对应的 ovc 是否可用以及对应的空满状态 (elig)。这个模块还接受 vc 和 sw 分配的结果并依据这些信息进行决策。

rtr_channel_output 的作用是将各种信号汇总成为 channel 总线，并实现锁存功能。其作用与 rtr_channel_input 相反。

vcr_alloc_mac 模块的实现比较简单，其主要功能就是根据配置参数来生成合适的 vc 和 sw 分配器。具体的 vc 和 sw 分配器实现我们可以不去考虑而直接采用 clib 库中已经实现的。

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 是指跟 flit_data 相关的控制信号，如 flit_head,flit_tail 等信号。

flit_buffer 的寄存器文件的实现方案与 mh 和 lai 的方法相同，都是 damq 的实现策略。这种策略在存储空间较大时需要 3 个时钟周期才能完成操作，之所以在二者的方案中得到使用是因为他们采用复选器，但是这种方式只适用于规模较小的情况。

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

代码中资源和消息类的含义：为了防止协议死锁，通常需要将网络中的报文分为若干个消息类，并将网络中的 vc 分几一些子集，然后将不同消息类的报文分配不同的 vc 子集中。另外，在一个消息类的报文中，为了防止因为资源循环依赖造成的死锁。还可以将每个消息类中的报文分为几个资源类别，不同资源类别的报文在竞争共享资源时形成一个偏序，因而可以避免这种死锁。注意到不同的消息类之间不存储资源死锁，因为他们是被映射到不同的硬件资源上的。在代码中假设每个类别使用的 vc 数量相同，这样整个路由器中每个端口需要的 vc 数量便等于资源类型的数量乘以消息类型的数量再乘以每个类型所能使用的 vc 数量。在具体的实现过程中，我们还要以考虑简化情况即消息类型和资源类型均为 1，在这种情况下每个类型的 vc 数量便等于端口所能使用的最大的 vc 数量。

IV. 代码修改过程

（1）源端与路由器的接口处：在 testbench.v 模块中实例化了 router_wrap,flit_sink.v 和 packet_source.v 三个模块。为此，我们需要在 packet_source 和 router_wrap 之间的接口处增加一些信号线以表示这些动态的 vc 变化信息。具体的方法是：（1）增加 M 根线从 router_wrap 到 packet_source（memory_bank_allocated）来表示每个 memory bank 的可用性；只有当相应的 memory bank 可用时 packet_source 模块可以利用相应的 vc。router_wrap 模块按照一定的策略选择该 memory bank 的下一个使用者并修改 memory_bank_grant 信号，packet_source 发现该信号无效以后便不再分配该 vc。当这个 memory bank 上的所有 vc 被释放时 router_wrap 回收该 memory bank 并将其分配给其它端口使用。（2）增加一根从 router_wrap 到 packet_source 的信约反馈信号线 credit_for_shared 用于表示当前反馈的信约是共享 vc 的还是私有 vc 的。(3) 在 packet_source 中增加 5 个流量控制模块用于跟踪 5 个共享的存储器的使用情况，这 5 个共享的存储体上的 vc 指针只能够寻址 1 个 memory bank。（4）增加一根从 packet_source 到 router_wrap 的信号线用于表明当前的 vc id 是针对私有 vc 的还是共享 vc 的。由于在源端对输出 vc 的选择是随机进行的，没有相应的分配模块，因此源端的修改还是非常简单的。另外，由于每个源端只有一条数据流，而且这种数据流是按顺序注入网络的，因此，我们的方案假设在源端不会占用共享的 vc。对共享 vc 的占用仅在路由器内部。由于不会占用共享的 vc，因此我们的方案在源端也不需要使用 5 个额外的流量控制模块。

channel_width ＝链路控制信号的宽度＋切片数据信号的宽度＋切片控制信号的宽度。为了实现我们的设计方案，我们需要增加两人个额外的控制信号 memory_bank_grant 和 credit_for_shared。

（2）路由器与目的端之间：由于目的端有足够的消耗能力，因此在目的端不存在 vc 不足的情况。因此，我们的方案在目的端也不会占用共享的 vc。对于共享 vc 的占用仅发生在路由器内部。因此，在目的端也不需要使用 5 个额外的流量控制模块。

（3）路由器与路由器之间：

通过这种方式，我们的方案所使用的控制存储资源要比 mh 小，同时我们的方案所用的控制存储资源实际上也比 lai 的方案要少。在做实验进行比较的过程中我们可以考虑两种情况：（1）同样的 flit buffer 大小的情况下比较性能；（2）同样的存储控制开销的情况下比较性能。（3）同样的性能情况下两人种方案需要的存储器和控制存储资源的多少。

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).