文章编号：2095-2163(2022)12-0016-14 中图分类号：TP302 文献标志码：A

针对多线程应用程序的片上网络优化设计

胡海洋[1,2]，李悦瑶[1,2]，李 阳[1,2]，赵玉来[1,2]，李建华[1,2]
（1 合肥工业大学 计算机与信息学院，合肥 230601；
2 合肥工业大学 情感计算与先进智能机器安徽省重点实验室，合肥 230601）

摘 要：在多线程程序执行过程中，遇到同步屏障(Barrier)时，进度快的线程需要等待进度慢的线程，导致执行进度快的处理器处于等待状态，不仅影响性能且浪费功耗。对基于片上网络的多核处理器来说，数据包在网络中的路由过程所耗费的时间占据了较多的线程执行时间。为了均衡多线程的执行进度提升性能，本文提出了线程感知的虚拟通道分配和片上网络路由方案：一种动态的为执行进度慢的线程优先分配虚拟通道的机制 AVCA 和关键线程拥塞避免的路由策略 CTCAR。实验结果表明，与传统没有线程感知路由算法相比，本文所提出的组合方案可以有效降低多线程应用程序数据包在片上网络传输时的平均延迟。

关键词：片上网络；多线程应用程序；拥塞避免；路由算法；虚拟通道

On-chip network optimized design for multithreaded applications

HU Haiyang[1,2], LI Yueyao[1,2], LI Yang[1,2], ZHAO Yulai[1,2], LI Jianhua[1,2]
(1 School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601, China; 2 Anhui Province Key Laboratory of Affective Computing and Advanced Intelligent Machine, Hefei University of Technology, Hefei 230601, China)

【Abstract】 During the execution of a multithreaded application, when encountering a barrier, the thread with fast progress needs to wait for the thread with slow progress, therefore causes the processor with fast execution progress to be in a waiting state, which not only affects the performance, but also wastes power consumption. For multi-core processors based on the on-chip network, the routing process of packets in the network occupies more thread execution time. In order to balance the progress of multiple threads and improve the performance, this paper proposes a thread-aware virtual channel allocation and on-chip network routing scheme: a dynamic mechanism AVCA for assigning virtual channels to threads with slow execution progress in priority, and routing for key thread congestion avoidance strategy CTCAR. The experimental results show that, compared with the traditional thread-unaware routing algorithm, the combined scheme proposed in this paper can effectively reduce the average delay of multi-threaded application packets in the on-chip network transmission.

【Key words】 on-chip network; multithreaded applications; congestion avoidance; routing algorithm; virtual channel

0 引 言

随着半导体工业的不断发展，芯片上集成了越来越多的处理器核，这些集成在芯片上的处理器核在提高性能的同时，各处理核间的通信也带来了高延迟和高能耗的问题，因此芯片上多个处理器核之间的通信延迟成为制约芯片性能提高的重要因素。当下，片上互联网络正在发生巨变，计算机体系结构专家提出了一种叫做片上网络(NoC)的具有高扩展性的资源管理方案。NoC 逐渐取代了像总线(Bus)和交叉开关(Crossbar)这样的传统片上互联方案[1-6]。

同步屏障(Barrier)是并行计算的一种方法。对于一组线程，程序中的一个同步屏障意味着任何线程执行到此后必须等待，直到所有线程都到达此点才可以继续执行下面的程序。同步屏障(Barrier)是用来将程序分段而被设计出来的。举一个例子，在任何线程使用步骤 $t+1$ 中的值作为输入之前，步骤 t 中的共享矩阵中的值已经被更新[7-10]。

对于基于 NoC 的多核处理器来说，数据包在网络中的路由过程占据了较多的线程执行时间。有的线程由于缺失地址较多，需要在 NoC 传输的数据包较多，因此这个线程执行得较慢。而同步屏障

基金项目：安徽省重点研究与开发计划(202004d07020004)；安徽省自然科学基金项目(2108085MF203)。

作者简介：胡海洋(1994-)，男，硕士研究生，主要研究方向：片上网络；李悦瑶(1997-)，女，硕士研究生，主要研究方向：片上网络；李 阳(1996-)，男，硕士研究生，主要研究方向：机器学习；赵玉来(1998-)，男，硕士研究生，主要研究方向：机器学习；李建华(1985-)，男，博士，副研究员，主要研究方向：计算机体系结构。

通讯作者：李建华 Email：jhli@ hfut.edu.cn

收稿日期：2022-03-25

(Barrier)的存在,使得其他执行进度快的处理器需要等待这个执行进度慢的处理器,不仅影响性能、且浪费功耗。

同步屏障示例如图 1 所示。由图 1 可看到,线程 *A* 产生了一个蓝色的数据包 *A*,线程 *B* 产生了 2 个红色数据包 *B* 和 *C*。在传统没有线程感知的 NoC 中,数据包 *A* 和数据包 *B* 会平等地竞争路由器 10 和路由器 15 的端口和虚拟通道。实际上,由于线程 *B* 缺失 2 块地址,需要等待 2 个请求的回复都收到才能继续执行,所以这是一个进度较慢的线程。而同步屏障(Barrier)的存在,使得线程 *A* 需要等待线程 *B*。此时,数据包 *A* 不必与数据包 *C* 竞争路由器 10 和路由器 15 的端口和虚拟通道,因为即便数据包 *A* 优先到达终点,其线程依旧需要等待数据包 *C* 的线程,对程序的执行进度没有影响。

Das 等人[11]通过将影响应用程序执行的重要的数据包重新路由来解决这个问题,即挑选另外一条路由来避开拥塞区域,从而使影响应用程序执行的重要数据包可以更快地到达终点。这种方法可以使应用程序执行进度更快,同时解决饥饿、活锁、死锁等问题,但是当网络负载较高,由于没有可选的重新路由的路径,导致这种方法的效果达不到预期。

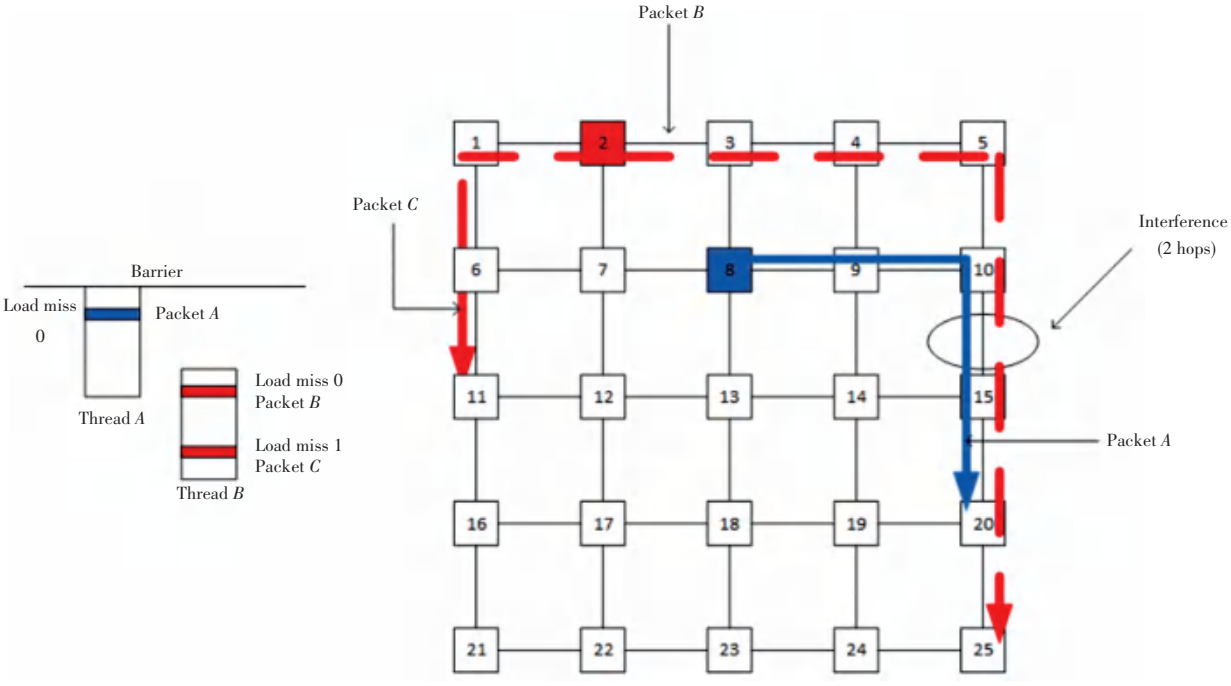


图 1　同步屏障示例

Fig. 1　Barrier demo

Das 等人[12]通过为每个数据包头部添加一个叫做 CFI(Critical Flit Identifier)的标签来使影响程序顺利执行的数据优先仲裁成功。其中,CFI 表示重要的数据存储在这个数据包的第几个 flit 当中。这种方法可以使程序执行进度更快,但是却会导致没有重要数据的 flit 发生饥饿。

本文提出 AVCA(Adaptive Virtual Channel Allocation)机制和 CTCAR(Critical Thread Congestion-Avoid Routing)算法来优化这个问题。AVCA 机制,即动态地为执行进度慢的线程分配更多的虚拟通道。CTCAR 路由算法选择策略会计算当下节点到目的节点每一条备选路径中,从当下路由器开始下 2 跳路由器中空的只能存储执行进度慢的线程数据包的缓冲区槽数之和,然后选择这 2 跳空闲缓冲区槽数之和最大的那条路径传输数据包。图 2 展示线程感知和非线程感知的比较,当 NoC 有线程感知时,会减少处理器 *A* 因为同步屏障(Barrier)等待处理器 *B* 的时间。

实验结果表明,本文方案在 4×4 mesh random 流量模式下,注入率为 0.023 packets/node/cycle 时相比 SAR[11]可以降低 19%的平均延迟。

本文内容第 1 节对相关工作进行介绍,包括本地感知自适应路由算法、区域感知自适应路由算法和全局感知自适应路由算法。第 2 节,对 AVCA 和 SSA 的设计与实现进行详细描述。第 3 节,提出了 CTCAR 路由算法选择策略。第 4 节,通过实验的方式将本文方案和对比方案进行比较。第 5 节,总结全文。

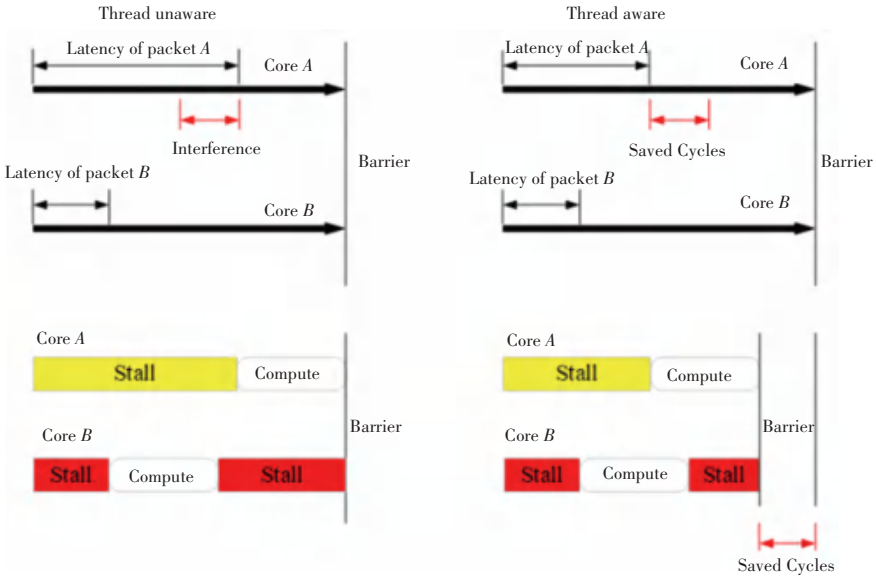


图 2 线程感知和非线程感知比较

Fig. 2 Thread-aware and thread-unaware comparison

1 相关工作

在 NoC 中,国内外学者设计了大量的路由算法来解决 NoC 拥塞的问题[13]。本文通过一种基于动态分配虚拟通道的路由算法,来平衡不同线程执行进度。

Jerger 等人[4]提到由于确定维度的路由算法(dimension-ordered routing)很简单,因而成为应用最为广泛的路由算法。然而,由于确定维度路由算法从源点到终点只有一条路径,所以不能有效绕开 NoC 中故障区域或者拥塞区域,从而降低了 NoC 的吞吐量,以及提高了 NoC 的延迟。自适应路由算法(Adaptive routing)有效解决了确定维度路由算法(Dimension-ordered routing)的缺点,给 NoC 增加了路径多样性,可以有效避开 NoC 中的拥塞区域,从而提高 NoC 的性能。自适应路由算法可以分为 3 类,分别是:本地感知自适应路由算法、区域感知自适应路由算法和全局感知自适应路由算法。

1.1 本地感知自适应路由算法

在本地感知自适应路由算法中,仅仅使用当前路由器所在节点的局部信息作为衡量 NoC 是否拥塞的标志。

Dally 等人[14]选择空的虚拟通道的数量作为某个端口是否拥塞的标志,当数据包到达 NoC 中某个节点时,会比较相邻节点的输入端口的空的虚拟通道的数量,随后选择具有最多空的虚拟通道的端口传输数据。

Kim 等人[15]选择空缓冲区的数量作为某个端口是否拥塞的标志。Singh 等人[16-17]使用输出队列长度作为某个端口是否拥塞的标志。当本地感知自适应路由算法被应用在不拥塞的区域时,由于本地感知自适应路由算法需要额外的逻辑来决定哪一条路径更好,从而导致本地感知自适应路由算法相比确定维度路由算法拥有更高的延迟。

Hu 等人[18]设计了一种叫做 DyAD 的路由算法,这种路由算法集合了确定维度路由算法和本地感知自适应路由算法的优点,能够根据 NoC 的拥塞情况在确定维度路由算法和本地感知自适应路由算法之间切换。当网络不拥塞时,路由器在确定维度的路由算法下运行;当网络变得拥塞下,路由器会转换为在本地感知自适应路由算法的情况下运行。

1.2 区域感知自适应路由算法

根据 NoC 局部拥塞情况做出的路由判断,可能会由于缺乏对更远处节点的拥塞情况的了解,做出不合理的判断。区域感知自适应路由算法的提出就是为了解决这个问题。

Gratz 等人[19]提出区域拥塞感知路由算法、简称 RCA,这是第一个跳出相邻节点观察更远处节点拥塞情况的路由算法。RCA 通过使用一个低带宽的监控网络在相邻路由器之间传输拥塞信息,在 NoC 的每一跳中,传输进入的拥塞信息与本地拥塞信息聚合后再一起传输到网络中。为了减少非最短路线上节点的拥塞信息的干扰,RCA 使用与本地节点的相对距离来对拥塞值进行加权。

在 RCA 中使用一个拥塞传递网络来综合本地拥塞信息和非本地拥塞信息，从而使 NoC 做出合理的路由选择。但是却因没有隔离不同的应用程序，从而导致过多的拥塞信息会对 NoC 路由选择做出干扰。

Ma 等人[20]设计了一种叫做 DBAR 的路由算法。在这种路由算法中，NoC 路由器做出路由选择时只会考虑起点路由器到终点路由器该二维象限的拥塞情况，不会考虑这个二维象限之外的拥塞情况。如此一来，当本应用程序做出路由选择时，就减少了被不同应用程序在 NoC 执行时拥塞情况的干扰。然而，在不分区的 NoC 中，DBAR 的性能和 RCA 的性能很接近。

区域感知自适应路由算法会形成一个用来专门传输非本地拥塞信息的拥塞传输网。当 NoC 的负载很大时，拥塞传输网将会以消耗不可忽视的线路和功耗开销为代价来提高 NoC 的性能。

Liu 等人[21]设计了一种把非本地的拥塞信息通过二进制位的形式嵌入到数据包的头 flit 中的路由算法来解决这个问题。这种叫做 FreeRider 的路由算法通过 3 步来选择输出线路。第一步，检查备选线路是不是热点，及这条线路是不是有一半以上的虚拟通道被占用。如果第一步没有选择成功，第二步会比较这 2 条备选线路的“后院”。如果第二步没有选择成功，第三步会比较这 2 条备选线路空的虚拟通道的数目，并选择空的虚拟通道最多的那条线路来传输数据。

1.3　全局感知自适应路由算法

区域感知自适应路由算法使数据包到达某个节点做出路由选择时，除了会考虑相邻节点的拥塞情况，还可以考虑更远处节点的拥塞情况，从而做出更合理的路由选择。但是因为区域感知自适应路由算法只考虑了 NoC 一部分节点的拥塞情况，就使得可能会因为对整个 NoC 的拥塞情况了解并不全面而做出不合理的路由选择。

全局感知自适应路由算法在选择路由路线时，综合考虑整个 NoC 每一个节点的拥塞情况，从而做出合理的路由选择。

Manevich 等人[22]提出自适应切换确定维度路由（ATDOR）的 NoC 架构，在这种 NoC 架构中通过使用一个二级网络，可将每一个节点的拥塞信息传输到一个专用节点，从而使每一个源点/终点对自适应地切换为 XY 确定维度路由算法或者 YX 确定维度路由算法。

Ramakrishna 等人[23]提出了 GCA 全局感知自适应路由算法。在这种路由算法中，拥塞信息被嵌入到数据包头部，NoC 的每一个路由器读取到达这个路由器头 flit 的拥塞信息，并嵌入到自己本地的图当中。GCA 通过为每个节点构建一个专属于该节点的图，从而使数据包到达某个节点时会根据全局网络的情况做出合理的路由选择。一个全局感知路由算法可以知道这条备选线路的拥塞值是多少，而不是只大概地了解这个方向的拥塞情况。

2　AVCA 和 SSA 的设计与实现

2.1　SSA 仲裁机制

文献[13]提到尽管系统中有大量未解决的负载缺失，但不是每一块负载缺失都会导致性能瓶颈。假设，一个线程有 2 个同时发出的网络请求，第一个向网络中较远的节点发送请求，第二个向网络中较近的节点发送请求。由于到更远处节点的数据包比到更近处节点的数据包有更高的延迟，所以到更近处节点的数据包是不重要的数据包。

文献[13]定义了一个参数 *Slack*，用来表示在不影响程序整体执行时间的情况下，一个数据包的最大延迟是多少个周期。因此，网络中 *Slack* 等于 0 的数据包越多，线程的执行进度越慢。

Slack 流程如图 3 所示。图 3 中，数据包 *A* 到路由器 20 有 4 跳，数据包 *B* 到路由器 21 有 5 跳。只有节点 8 收到请求数据包 *A* 和请求数据包 *B* 的回复，线程才可以继续执行。因此数据包 *A* 的 *Slack* 是 1，数据包 *B* 的 *Slack* 是 0。

本文引入 SSA（Slack Switch Allocation）仲裁机制，将每个数据包的 *Slack* 嵌入到数据包头 flit 的空闲位中，2 个数据包不再是平等地竞争同一个端口，而是根据数据包的 *Slack* 仲裁同一个端口。

2.2　数据包格式

传统数据包[21]中，头 flit 通常含有 2 bits flit 种类信息，31 bits 路由信息，3 bits 请求种类信息和 40 bits 物理地址信息。数据包格式如图 4 所示。图 4 表明头 flit 至少还有 52 bits 空闲位。

本文在数据包头 flit 的空闲位中添加 1 位线程识别（Thread Identification，*TI*）信息和 6 位 *Slack* 信息。为了区分不同种类的线程，本文将线程随机地分为 2 类，将一类线程数据包的 *TI* 信息标记为 0，另一类线程数据包的 *TI* 信息标记为 1。

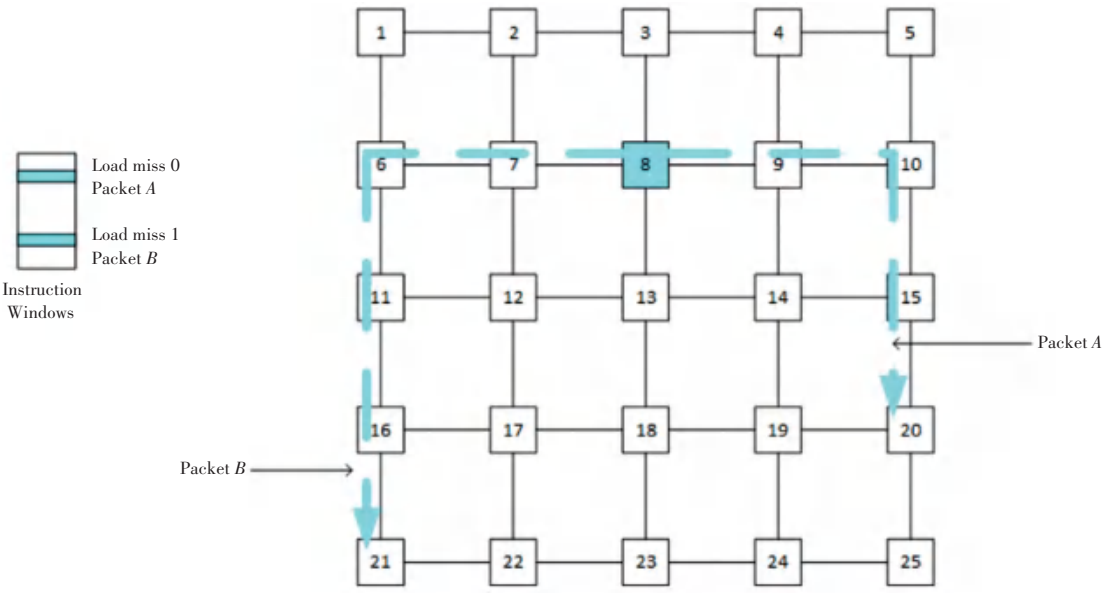


图 3 *Slack* 流程图

Fig. 3 *Slack* flow chart

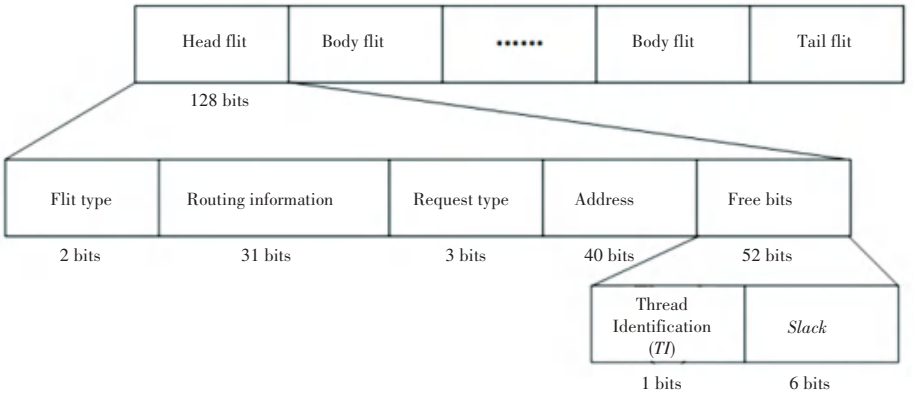


图 4 数据包格式

Fig. 4 Packet format

2.3 AVCA 机制

由于已经为不同线程的数据包标有不同的标记,虚拟通道分配如图 5 所示。图 5 中,白色的虚拟通道只传输 *TI* 值为 0 的数据包,紫色的虚拟通道只传输 *TI* 值为 1 的数据包, 蓝色的虚拟通道会根据 NoC 中实际流量情况动态地传输 *TI* 值为 0 或者 *TI* 值为 1 的数据包。

假设开始情况下 2 个蓝色的虚拟通道传输 *TI* 值为 0 的数据包,当 NoC 中某一路由器某一个端口紫色虚拟通道满载,这说明网络中 *TI* 值为 1 的数据包不再是低优先级的数据包,此时网络中蓝色的虚拟通道需要做出调整。

发生拥塞的紫色虚拟通道所在的路由器,通过信号线发送 VCAI(VC Adjustment Information) 虚拟通道调整信息到网络中所有其他的路由器,所有其他路由器收到 VCAI 后,将 2 个蓝色的虚拟通道转换为只能传输 *TI* 值为 1 的数据包。

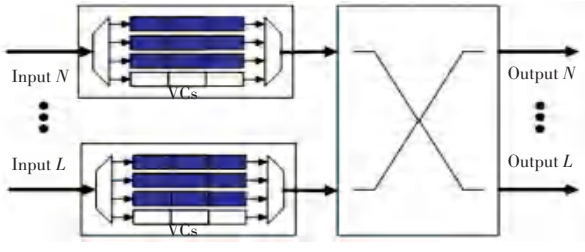


图 5 虚拟通道分配图

Fig. 5 Virtual channel allocation diagram

2.4 流水线变更

传统流水线如图 6 所示。由图 6 可知,传统的五级流水线有固定的虚拟通道分配阶段,及头 flit 经过路由计算确定输出端口以后,还需要仲裁成功一个与该输出端口相应的虚拟通道。

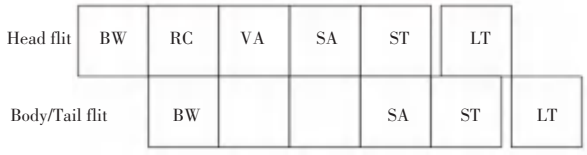


图 6　传统流水线

Fig. 6　Traditional pipeline

由于本文为数据包做了不同的标记，故而也得匹配不同的虚拟通道。在本文中，与传统的虚拟通道分配阶段相比，本文添加了一个数据包识别（Packet Identification，PI）阶段。如果识别出来的数据包是低优先级线程数据包，由于低优先级线程数据包只匹配一个虚拟通道，所以将没有 VA（VC Allocation）阶段，可以直接进行 ST（Switch Traversal）阶段。如果识别出来的数据包是高优先级线程数据包，由于高优先级线程数据包匹配 3 个虚拟通道，所以就还要进行 VA 阶段。图 7 展示了本文设计的流水线。

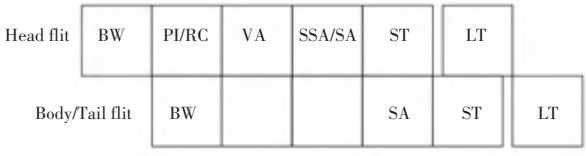


图 7　本文流水线

Fig. 7　The pipeline in this paper

仲裁策略是指，当多个程序的数据包都要请求使用相同的输出端口时，哪一个线程的数据包可以优先使用这个输出端口。传统的仲裁策略包括 Round-robin 和 Age-based，都是没有程序感知的。

本文在数据包头 flit 中嵌入 *Slack* 信息，使数据包仲裁时变成程序感知，及 *Slack* 小的数据包可以优先使用输出端口。本文的头 flit SA（Switch Allocation）阶段变成了 SSA（Slack Switch Allocation）阶段，即根据 *Slack* 数值的大小进行仲裁。

当需要仲裁的 2 个数据包的 *Slack* 不同时，用 *Slack* 进行仲裁。当需要仲裁的 2 个数据包 *Slack* 相同时，用 Round-robin 进行仲裁。同时，如果数据包经过 SSA 阶段，仲裁成功的输出端口会专供这个数据包来使用，因此 Body 和 Tail flit 无需再仲裁这个输出端口，直到相应的 Tail flit 离开这个端口。而如果没有经过 SSA 阶段，那么 Body 和 Tail flit 还需要继续再仲裁这个输出端口。

2.5　AVCA 和 SSA 总体设计与实现

路由器基础架构如图 8 所示。由图 8 可知，本文提出的路由器基础架构，主要由 5 部分组成，分别是：输入/输出端口、输入缓冲区、AVCA、交叉开关（Crossbar）和 SSA。

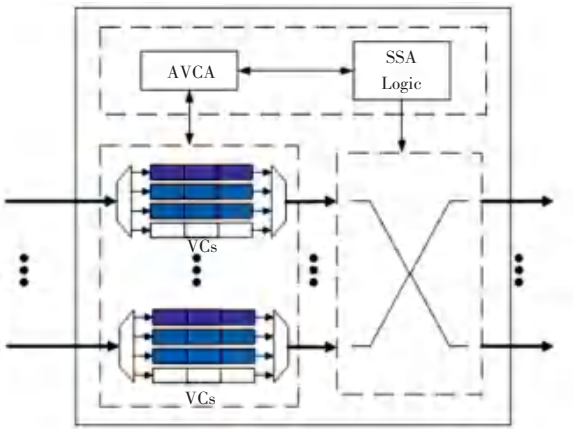


图 8　路由器基础架构

Fig. 8　Router architecture

AVCA 的灵活分配虚拟通道和 SSA 的基于 *Slack* 的仲裁策略，相互作用，促使制约程序执行进度重要的数据包在路由器中传输时得到更多资源。

图 9 是有关 AVCA 的详细设计，本文为每个路由器添加了一个虚拟通道控制器（VC Controller，VCC）。VCC 会通过多路复用器（Multiplexer）收集每个端口 4 个虚拟通道的信息。例如，北端口的紫色虚拟通道满载，北端口会将 VCAI（VC Adjustment Information）信息传输到本路由器 VCC 中，VCC 通过信号线将 VCAI 传输到网络所有其他路由器中。其他路由器的 VCC 收到 VCAI 后，将 VCAI 传输到每个端口的蓝色虚拟通道中，蓝色虚拟通道收到 VCAI 做出调整，改为只传输 *TI* 值为 1 的数据包。

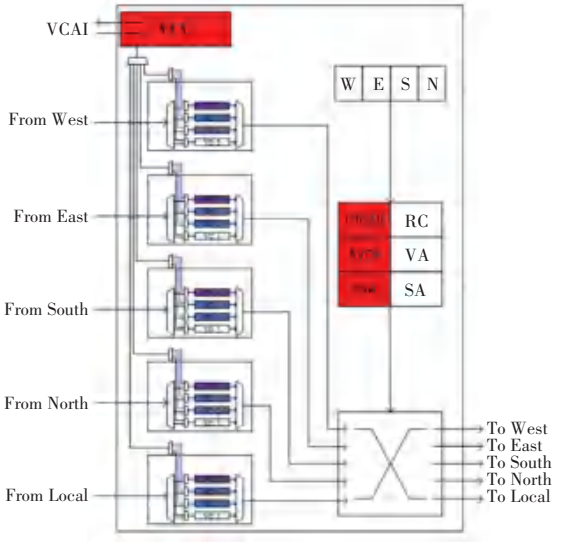


图 9　AVCA 详细设计

Fig. 9　AVCA detailed design

2.6　AVCA 举例

数据包传输实例如图 10 所示。在图 10（a）中，假设有一个低优先级线程数据包 *A* 的路由路径是从

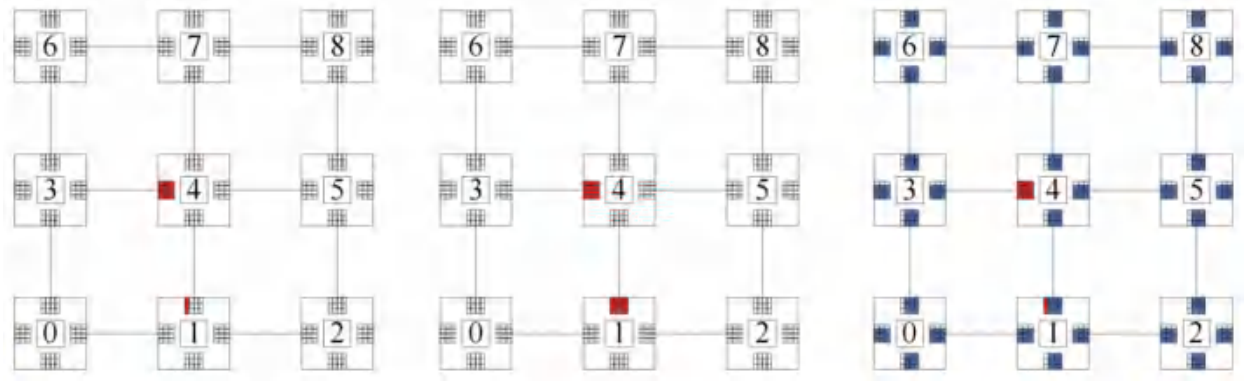
起点路由器 1 通过中间路由器 4 传输到终点路由器 3,预期则应从路由器 1 的北端口传输到路由器 4,再从路由器 4 的西端口传输到路由器 3。

但是此时,路由器 4 的西端口没有空闲的缓冲区可以传输数据,所以数据包 *A* 不能传输到路由器 4,只能在路由器 1 的北端口停留。在图 10(a)中,红色的虚拟通道,代表这个虚拟通道已经被数据包占据,不能传输数据。

随后,路由器 1 又需要分别传输 3 个和数据包 *A* 的路由路径相同的低优先级线程数据包 *B*,*C* 和 *D*,由图 10(b) 可知,这时数据包 *A*,*B*,*C* 和 *D* 将占据路由器 1 北端口的 4 个虚拟通道。

当路由器 1 需要把高优先级线程的数据包 *E* 从路由器 1 传输到路由器 7 时,即便路由器 4 的北端口有空缓冲区可以传输数据,但是因为路由器 1 的北端口的 4 个虚拟通道已经被 4 个低优先级线程数据包占据,所以路由器 1 此时不能传输数据包 *E*。

如果使用本文方案,数据包传输具体见图 10(c)。在图 10(c)中,每个端口只有一个虚拟通道可以传输低优先级线程的数据包。每个端口蓝色的虚拟通道表示只能传输高优先级线程数据包的虚拟通道,白色的虚拟通道表示只能传输低优先级线程数据包的虚拟通道。当数据包 *A* 占据了路由器 1 北端口唯一一个可以传输低优先级线程数据包的虚拟通道时,由于数据包 *B*,*C* 和 *D* 都是低优先级线程的数据包,因此数据包 *B*,*C* 和 *D* 不会再占据路由器 1 北端口剩余 3 个空的虚拟通道。随后,当路由器 1 需要传输高优先级线程的数据包 *E* 时,因为路由器 1 的北端口有 3 个空的虚拟通道,所以路由器 1 可以将数据包 *E* 顺利地传输到终点。



(a) 实例 1 (b) 实例 2 (c) 实例 3

图 10 数据包传输实例图

Fig. 10 Packets transmission instance diagram

3 CTCAR 路由算法选择策略

3.1 CTCAR 概述

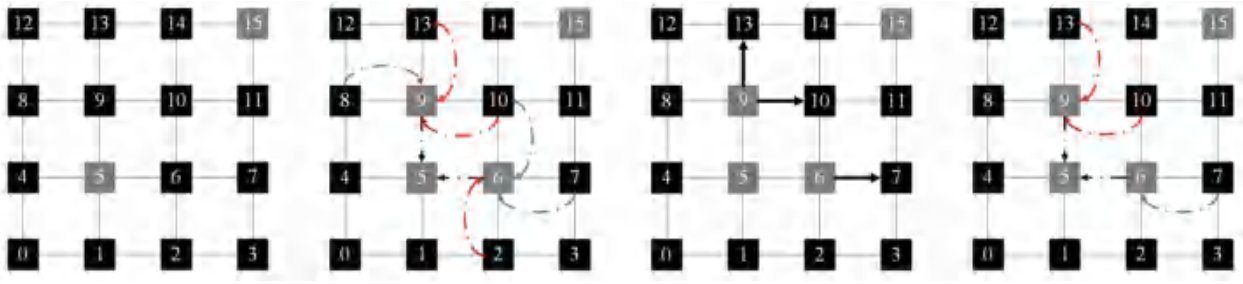
本文提出一种专门为高优先级线程优化的路由算法选择策略,即重要线程拥塞避免(Critical Thread Congestion-Avoided Routing,CTCAR)路由算法选择策略。

CTCAR 实例如图 11 所示。在图 11(a)中,这是一个 4 行 4 列的 mesh 拓扑结构,其中每一个节点中的数字表示这是第几个节点。图 11 中,实例的数据包传输起始路由器为节点 5,终点路由器为节点 15。

当高优先级线程的数据包要从起点路由器 5 传输到终点路由器 15 时,有路由器 6 和路由器 9 两个路由器可以选择。本文通过计算路由器 6 和路由器 6 的下一跳路由器的空闲缓冲区之和以及路由器 9 和路由器 9 的下一跳路由器的空闲缓冲区之和,然后比较这两者的大小,最终指定这两跳缓冲区之和最大的那条线路为选择成功的线路。在图 11(b)中的黑色虚线表明,路由器 5 在选择下一跳是路由器 6、还是路由器 9 时,会根据路由器 6 和路由器 9 发回来的信息作为判断。

本文定义了 2 个参数,分别是 *Vc_reservation* 和 *Slot_number*。因为本文为高优先级线程数据包分配了 3 个虚拟通道,所以一个路由器在收集相邻路由器的信息时,会得到相邻路由器专门存储高优先级线程数据包的虚拟通道是否被其他数据包预定的信息,及 *Vc_reservation* 是 1、还是 0。如果 *Vc_reservation* 是 1 代表这个虚拟通道已经被预定,不能传输数据;如果 *Vc_reservation* 是 0,代表这个虚拟通道没有被预定,可以传输数据。

Slot_number 代表这个没有被预定的虚拟通道有多少个空闲缓冲区槽。信息传输示意如图 12 所示。在图 12 中，路由器 9 收集与其相邻的 3 个路由器，即路由器 8、路由器 10 和路由器 13 的每个专门存储高优先级线程数据包虚拟通道的 *Vc_reservation* 和 *Slot_number*。



（a）实例 1　（b）实例 2　（c）实例 3　（d）实例 4

图 11　CTCAR 实例图

Fig. 11　CTCAR instance diagram

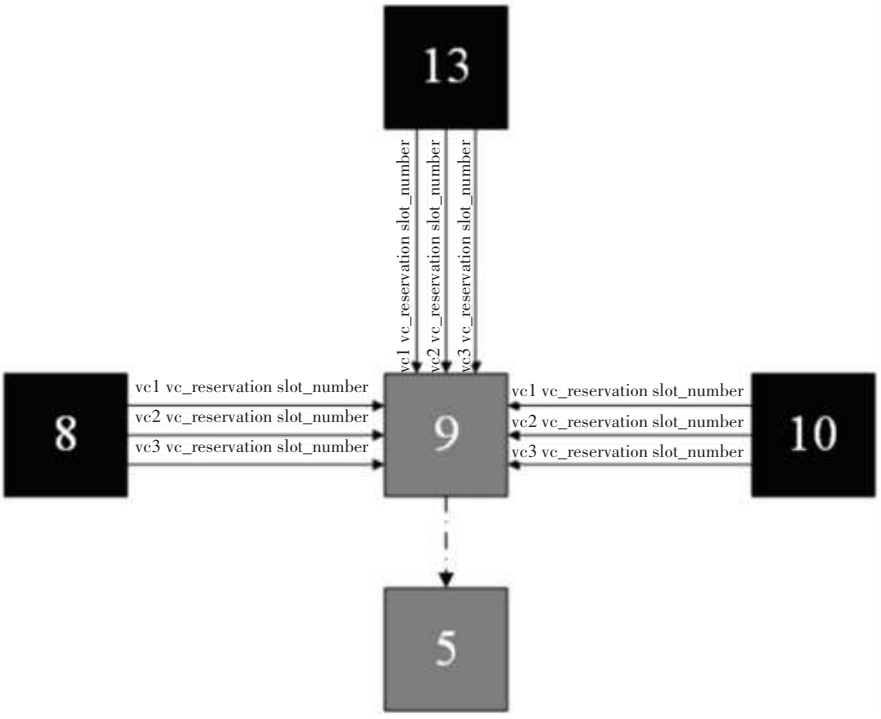


图 12　信息传输图

Fig. 12　Information transmission diagram

图 11(b)中，蓝色虚线表示下一跳存在没有被预定的专门传输高优先级线程数据包的虚拟通道，可以传输数据。图 11(b)中，红色虚线表示下一跳的 3 个专门传输高优先级线程数据包的虚拟通道已经被预定，不能传输数据。

图 11(c)中，3 条深黑色实线表示路由算法计算出路由器 6 和路由器 9 的下一跳可以到达哪个路由器。

在本文中，用的路由算法是基于奇偶转向模型的自适应路由算法。当高优先级线程的数据包由路由器 5 传输到路由器 9 时，因为这是奇数列不能做由向北到向西的转向和由向南到向西的转向，及路由器 9 的下一跳可以到达路由器 10 和路由器 13。因为路由器 6 位于偶数列，不能做由向东到向北的转向和由向东到向南的转向，及路由器 6 的下一跳只能到达路由器 7，不能到达路由器 10。

在图 11(d)中，由于路由器 10 和路由器 13 专门传输高优先级线程数据包的虚拟通道，已经被其他数据包预定。而路由器 7 专门传输高优先级线程的虚拟通道没有被其他数据包预定，所以本例应该选择路由器 6，因为路由器 6 的下一跳路由器 7 可以传输高优先级线程的数据包。

3.2 CTCAR 详细设计

以图 11 为例 CTCAR 流程如图 13 所示。首先，确定当下网络中，*TI* 值为 1 是高优先级线程数据包，还是 *TI* 值为 0 是高优先级线程数据包。如果数据包是高优先级线程数据包，使用 CTCAR。如果数据包不是高优先级线程数据包，退出流程，使用 XY 路由算法。

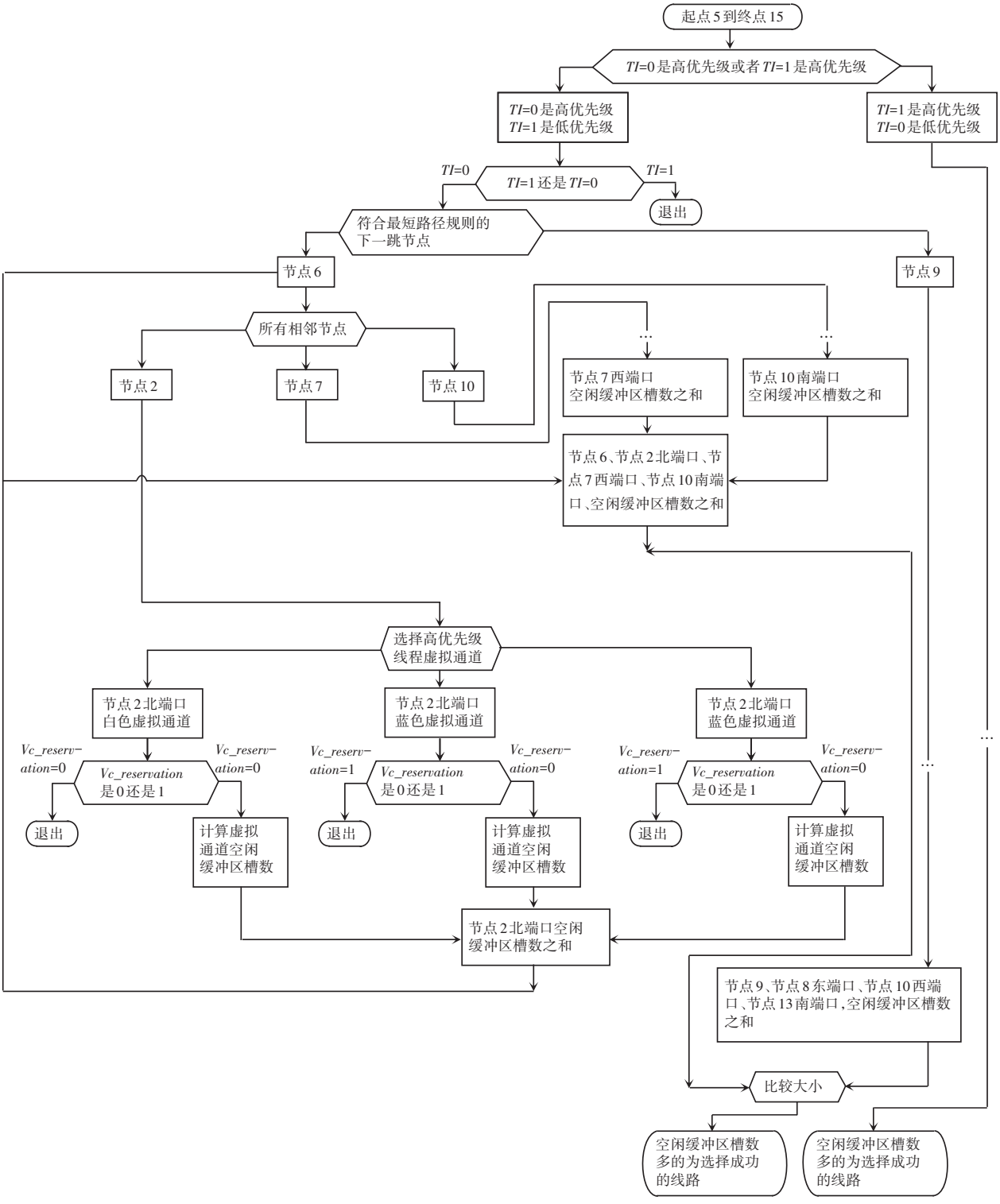


图 13 CTCAR 流程图

Fig. 13 CTCAR flow chart

其次，计算当下节点到目的节点每一条备选线路中，从当下路由器开始下 2 跳路由器中，空的只能存储高优先级线程数据包的缓冲区槽数之和，再选择这 2 跳空闲缓冲区槽数之和最大的那条线路传输数据包。CTCAR 算法设计代码见如下。

```
Algorithm: CTCAR
Definition:  Route          Alternate path
             Ti             Thread information
             Hpti           High priority thread information
             Number[ ]      Sum of free buffer slot in a path
             Path           Selected one of the route
-------------------------------------------------------------------------------
             Node           The current node
             Nextnode       The next node
             Neighbor_node  Adjacent node of the current node
             Vc             Virtual channel
             Vc_number      The number of virtual channels for a port
             Vc_reservation Whether a virtual channel is reserved
             Total_slot     Sum of all free buffer slots on a port
             Slot_number    Number of free buffer slots
             Selected_route The route that was eventually chosen

Input:  Route, Ti, Hpti
Output: Selected_route
if (Ti == Hpti) {
  for (Path î Route) {
    for (Neighbor_node î Path.Nextnode) {
      for (Vc î Neighbor_node.Vc) {
      if (Vc.Vc_reservation != 1) {
          Total_slot = 0
          Total_slot = Total_slot + Vc.Slot_number
          }
      }
    Number[Path] = 0
    Number[Path] = Number[Path] + Total_slot
    Number[Path] = Number[Path] + Path.Node.Slot_number
    }
    Selected_route = max(Number[Path])
}
}
```

4　实验

4.1　仿真环境配置

本文通过扩展的 Noxim[24] 模拟器对 AVCA、SSA 和 CTCAR 进行分析。本文设定的仿真环境配置见表 1。

在本文中，每经过 10 000 个周期设置一个同步屏障(Barrier)，即低优先级的线程每仿真10 000个周期就会停下等待高优先级的线程。

在本文中为每个路由器的输入端口设置了 4 个逻辑大小为 8 个 flit 的虚拟通道。

表 1　实验参数表

Tab. 1　Experimental parameters table

设置	参数
拓扑结果	4×4 mesh，8×8 mesh
Packet 大小	8~12 fits
缓冲区大小	8 flits
Flits 大小	32 bits
系统 Warm-up 时间	1 000 cycles
运行时间	20 000 cycles
流量模式	Random、Transpose1、Transpose2

本文采用 3 种流量模式进行仿真，分别是

Random、Transpose1 和 Transpose2。对此可做阐释分述如下。

(1) Random 流量模式中,NoC 中每个节点向 NoC 中任意其他节点随机地发送数据包。

(2) Transpose1 流量模式中,假设准备发送数据包的起始节点的坐标是 (x,y),那么这个数据包将会被发送到的终点是 $(mesh\_dim\_x - 1 - y, mesh\_dim\_y - 1 - x)$,其中 $mesh\_dim\_x$ 和 $mesh\_dim\_y$ 表示这个 NoC 设置的横坐标和纵坐标上分别有多少个节点。这种流量模式的特点是:越靠近中间区域的路由节点注入网络的数据包的跳数就越小,但是从整体上来看,长距离多跳数据包占的比重更大。

(3) 在 Transpose2 流量模式中,假设准备发送数据包的起始节点的坐标是 (x,y),那么这个数据包将会被发送到的终点是 (y,x)。在本文中,每次仿真包括1 000个周期的预热阶段和20 000个周期的仿真阶段。

4.2 性能分析

4.2.1 平均延迟

图 14~图 16 是 4×4 mesh 拓扑结构 3 种流量模式下的延迟对比图。图 17 是 8×8 mesh 拓扑结构 random 流量模式下的延迟对比图。4 幅图中,横坐标是网络的注入率,纵坐标是以周期为单位的数据包平均延迟。

本文对 4 种方案进行对比分析。方案一是 Baseline 方案及没有采用 AVCA 机制、SSA 机制和 CTCAR 路由算法选择策略。方案二采用了传统的没有程序感知的路由算法,即 Freerider 路由算法。方案三采用了程序感知的路由算法,及 SAR 路由算法。方案四采用了本文方案,及应用了 AVCA、SSA 和 CTCAR。各对比方案的比较结果见表 2。

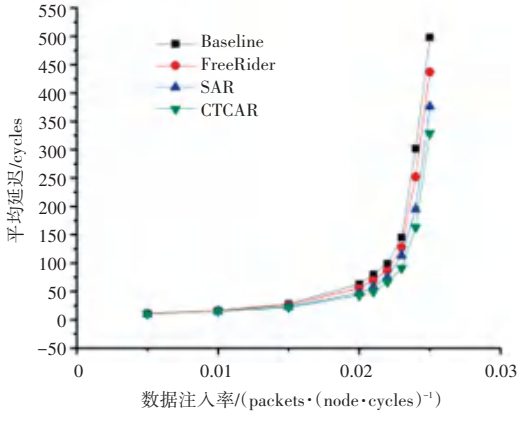


图 14 random 流量模式下 4×4 mesh 网络延迟

Fig. 14 Latency of 4 × 4 mesh in random traffic

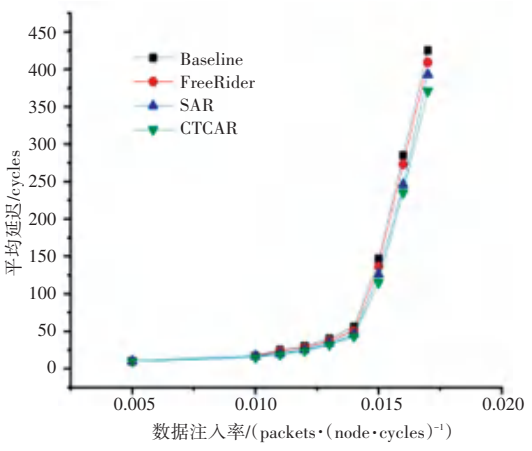


图 15 Transpose1 流量模式下 4×4 mesh 网络延迟

Fig. 15 Latency of 4 × 4 mesh in Transpose1 traffic

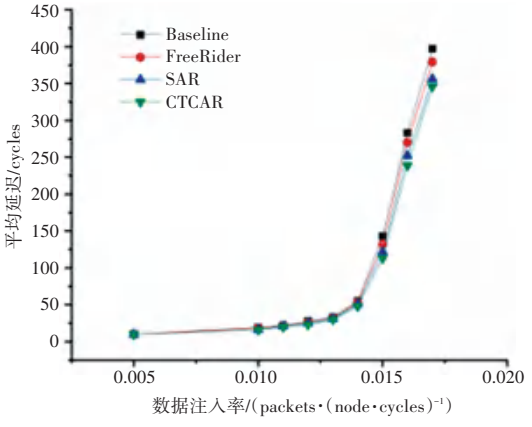


图 16 Transpose2 流量模式下 4×4 mesh 网络延迟

Fig. 16 Latency of 4 × 4 mesh in Transpose2 traffic

表 2 各对比方案比较

Tab. 2 Comparison of different schemes

	最小路由	程序感知	动态虚拟通道分配	线程感知路由算法
Baseline	×	×	×	×
Freerider	√	×	×	×
SAR	√	√	×	×
CTCAR	√	√	√	√

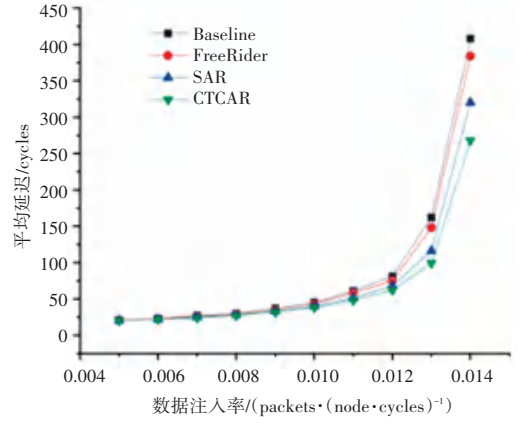


图 17 random 流量模式下 8×8 mesh 网络延迟

Fig. 17 Latency of 8 × 8 mesh in random traffic

SAR 和本文所提出的 CTCAR 都是程序感知的路由算法。从图 14~图 17 可以看出，在不同的流量模式下 CTCAR 较 SAR 在高注入率或低注入率下都有一定的延迟改善。

在 4×4 mesh random 流量模式下，注入率大于 0.021 packets/node/cycle 小于 0.025 packets/node/cycle 时，CTCAR 与对比方案相比有较大的延迟改善。其中，注入率为 0.023 packets/node/cycle 时，CTCAR 效果达到最佳，相比 SAR 可以降低 19% 的平均延迟。

在 8×8 mesh random 流量模式下，注入率大于 0.012 packets/node/cycle 小于 0.014 packets/node/cycle 时，CTCAR 与对比方案相比有较大的延迟改善。其中，注入率为 0.014 packets/node/cycle 时，CTCAR 效果达到最佳，相比 SAR 可以降低 16% 的平均延迟。

这是由于 CTCAR 引入 *Slack*，在仲裁阶段是程序感知的。同时 CTCAR 考虑到同步屏障的存在，可以动态为不同线程数据包分配不同的虚拟通道，从而平衡各个线程的执行进度。CTCAR 在灵活分配虚拟通道的基础上，还引入了区域拥塞感知的思想，从而进一步降低了网络的平均延迟。

4.2.2　吞吐率

4×4 mesh 和 8×8 mesh 拓扑结构 random 流量模式下，CTCAR 和对应的 3 种对比方案的吞吐率比较结果如图 18、图 19 所示。图 18、图 19 中，横坐标表示网络注入率，纵坐标表示网络中平均每个节点的吞吐率。

从图 18、图 19 中可以看出，CTCAR 具有更高的网络吞吐率，这是因为本文提出的 AVCA 机制可以动态地为不同优先级线程数据包分配不同的虚拟通道，同时 CTCAR 可以有效避免高优先级线程数据包发生拥塞。

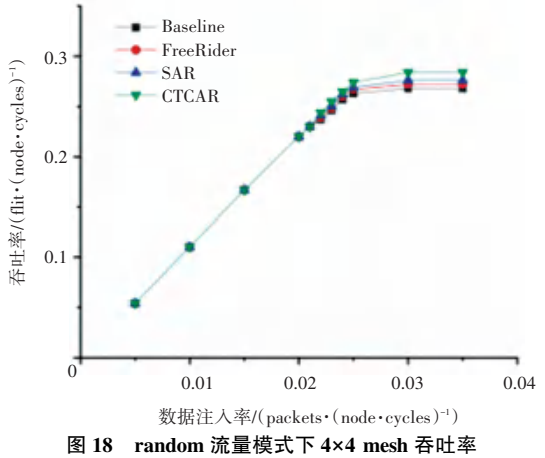


图 18　random 流量模式下 4×4 mesh 吞吐率

Fig. 18　Throughput of 4×4 mesh in random traffic

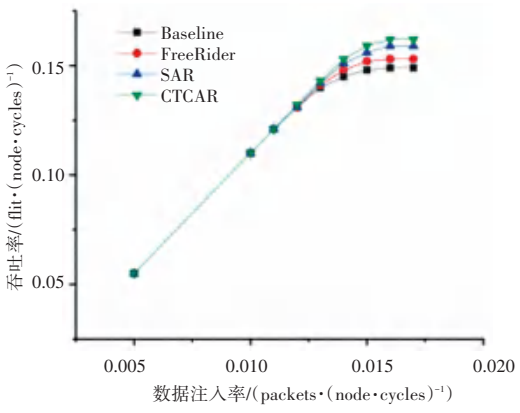


图 19　random 流量模式下 8×8 mesh 吞吐率

Fig. 19　Throughput of 8×8 mesh in random traffic

图 18 中，当注入率为 0.03 packets/node/cycle 时，4×4 mesh random 流量模式下 4 种方案均达到饱和状态。在饱和注入率下，CTCAR 与 SAR、DBAR 和 Baseline 相比分别降低了 2.8%、4.2% 和 5.6% 的饱和吞吐率。

图 19 中，当注入率为 0.016 packets/node/cycle 时，8×8 mesh random 流量模式下 4 种方案均达到饱和状态。在饱和注入率下，CTCAR 与 SAR、DBAR 和 Baseline 相比分别降低了 1.9%、5.6% 和 8% 的饱和吞吐率。

4.2.3　面积及功耗分析

本文采用的实验工具是 Synopsys 公司的 Design Compiler，所采用的工艺库为 45 nm 标准单元库，对相同规模的 4 种方案进行逻辑综合。

表 3 为本文方案与对比方案的硬件开销。由于 CTCAR 中添加了 AVCA 模块、SSA 模块和 VCC 模块，使得本文的面积和功耗开销都有微小的增加。但是，通过 SSA 仲裁机制，AVCA 动态为高优先级线程分配虚拟通道，CTCAR 防止高优先级线程数据包发生局部拥塞，可以有效降低网络的平均延迟，以及提高网络的吞吐量。考虑到 CTCAR 整体性能优势，CTCAR 中额外面积和功耗开销所带来的影响是可以接受的。

表 3　面积与功耗开销

Tab. 3　Area and power consumption overhead

Network platforms	*Area*/ mm^2	*Avg. Power*/ w	*Max. Power*/ w
Baseline	6.56	2.18	3.09
FreeRider	6.78	2.39	3.28
SAR	6.71	2.35	3.23
CTCAR	6.81	2.42	3.29

5　结束语

对基于 NoC 的多核处理器来说，数据包在网络中的路由过程占据了较多的线程执行时间。有的线

程由于缺失地址较多,需要在 NoC 传输的数据包较多,因此这个线程执行得较慢。而同步屏障(Barrier)的存在,使得其他执行进度快的处理器需要等待这个执行进度慢的处理器,不仅影响性能且浪费功耗。

本文首先引入文献[13]中 *Slack* 参数,提出了基于 *Slack* 的仲裁机制 SSA。其次,本文通过在数据包头 flit 中嵌入标签的方式,将不同线程的数据包分类,用 VCC 为不同类别的线程分配不同数量的虚拟通道。最后,本文提出了 CTCAR 路由算法选择策略,可以有效降低高优先级线程数据包发生拥塞的可能性。

实验结果表明,与 SAR 相比,本文方案在增加可接受的硬件开销、功耗开销下,可以有效降低网络平均延迟,提高网路吞吐率。

参考文献

[1] DALLY W J, TOWLES B. Route Packets, Not Wires: On-Chip Interconnection Networks[C]//the 38th annual Design Automation Conference. Las Vegas, Nevada, USA: ACM, 2001: 684-689.

[2] GRATZ P, KIM C, MCDONALD R, et al. Implementation and evaluation of on - chip network architectures [C]//2006 International Conference on Computer Design. San Jose, CA, USA: IEEE, 2007: 477-484.

[3] BENINI L, MICHELI G D. Networks on chip: A new paradigm for systems on chip design[C]//2002 Design, Automation and Test in Europe Conference and Exhibition. Paris, France: IEEE, 2002: 418-419.

[4] JERGER N E, KEISHNA T, PEN L. On-chip networks[M]. Second Edition. Morgan & Claypool Publishers, 2017: 43-51.

[5] SANKARALINGAM K, NAGARAJAN R, MCDONALD R, et al. Distributed microarchitectural protocols in the TRIPS prototype processor [C]//2006 39th Annual IEEE/ACM International Symposium on Microarchitecture. Orlando, FL, USA: IEEE, 2006: 480-491.

[6] LIANG J, SWAMINATHAN S, TESSIER R. aSOC: A scalable, single-chip communications architecture[C]// 2000 International Conference on Parallel Architectures and Compilation Techniques. Philadelphia, PA, USA: IEEE, 2000, 37-46.

[7] MELLOR-CRUMMEY J, SCOTT M L. Algorithms forscalable synchronization on shared-memory multiprocessors[J]. ACM Transactions on Computer Systems, 1991, 9(1): 21-65.

[8] CHENG L, CARTER J B. Fastbarriers for scalable ccNUMA systems [C]//2005 International Conference on Parallel Processing. Oslo, Norway: IEEE, 2005: 241-250.

[9] CHEN Jie, WATSON W. Software barrier performance ondual quad-core opterons [C]//2008 International Conference on Networking, Architecture, and Storage. Chongqing, China: IEEE, 2008: 303-309.

[10] MICHAEL M M, SCOTT M L. Implementation ofatomic primitives on distributed shared memory multiprocessors[C]// 1995 1st IEEE Symposium on High Performance Computer Architecture. Raleigh, NC, USA: IEEE, 1995: 222-231.

[11] DAS A, BABU S, JOSE J, et al. Critical packet prioritisation by slack-aware re-routing in on-chip networks[C]// 2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS). Turin, Italy: IEEE, 2018: 1-8.

[12] DAS A, JOSE J, MISHRA P. Data Criticality in multithreaded applications: An insight for many-core systems[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2021, 29(9): 1675-1679.

[13] DAS R, MUTLU O, MOSCIBRODA T, et al. Aérgia: Exploiting packet latency slack in on-chip networks[C]//ISCA'10. Saint-Malo, France: ACM, 2010: 106-116.

[14] DALLY W J, AOKI H. Deadlock-free adaptive routing in multicomputer networks using virtual channels [J]. IEEE Transactions on Parallel and Distributed Systems, 1993, 4(4): 466-475.

[15] KIM J, PARK D, THEOCHARIDES T, et al. A low latency router supporting adaptivity for on-chip interconnects[C]//42nd Design Automation Conference. Anaheim, CA, USA: IEEE, 2005: 559-564.

[16] SINGH A, DALLY W J, GUPTA A K, et al. GOAL: A load-balanced adaptive routing algorithm for Torus networks[C]//30th Annual International Symposium on Computer Architecture. San Diego, CA, USA: IEEE, 2003: 194-205.

[17] SINGH A, DALLY W J, TOWLES B, et al. Globally adaptive load-balanced routing on Tori[J]. IEEE Computer Architecture Letters, 2004, 3(1): 2.

[18] HU Jingcao, MARCULESCU R. DyAD - Smart routing for networks - on - chip [C]//41st annual Design Automation Conference. San Diego, CA, USA: ACM, 2004: 260-263.

[19] GRATZ P, GROT B, W. KECKLER S. Regional congestion awareness for load balance in networks-on-chip[C]//2008 IEEE 14th International Symposium on High Performance Computer Architecture. Salt Lake City, UT, USA: IEEE, 2008: 203-214.

[20] MA Sheng, JERGER N E, WANG Zhiying. DBAR: An efficientrouting algorithm to support multiple concurrent applications in networks-on-chip[C]// Proceedings of the 38th annual international symposium on Computer architecture. San Jose, California, USA: ACM, 2011: 413-424.

[21] LIU Shaoli, CHEN Tianshi, LI Ling, et al. FreeRider: Non-local adaptive network-on-chip routing with packet-carried propagation of congestion information[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(8): 2272-2285.

[22] MANEVICH R, CIDON I, KOLODNY A, et al. A cost effective centralized adaptive routing for networks-on-chip[C]//2011 14th Euromicro Conference on DigitalSystem Design. Oulu, Finland: IEEE, 2011: 39-46.

[23] RAMAKRISHNA M, KODATI V K, GRATZ P V. GCA: Global congestion awareness for load balance in networks-on-chip[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 27(7): 2022-2035.

[24] CATANIA V, MINEO A, MONTELEONE S, et al. Cycle-accurate network on chip simulation with Noxim[J]. ACM Transactions on Modeling and Computer Simulation, 2016, 27(1): 1-25.

[25] FARROKHBAKHT H, KAO H, JERGER N E. UBERNoC: Unified buffer power-efficient router for network-on-chip[C]// 13th IEEE/ACM International Symposium on Networks-on-Chip.

New York，USA：ACM，2019：1-8.
[26] CHEN Lizhong，ZHU Di，PEDRAM M，et al. Power punch：Towards non-blocking power-gating of NoC routers[C]//2015 IEEE 21st International Symposium on High Performance Computer Architecture（HPCA）. Burlingame，CA，USA：IEEE，2015：378-389.
[27] CHEN Peng，LIU Weichen，LI Mengquan，et al. Contention minimized bypassing in SMART NoC [C]//2020 25th Asia andSouth Pacific Design Automation Conference（ASP-DAC）. Beijing，China：IEEE，2020：205-210.
[28] CHIU G. The odd-even turn model for adaptive routing[J]. IEEE Transactions on Parallel and DistributedSystems，2000，11（7）：729-738.
[29] KIM J，BALFOUR J，DALLY W J. Flattened butterfly topology for on-chip networks[C]//40th Annual IEEE/ACM International Symposium on Microarchitecture（MICRO 2007）. Chicago，IL，USA：IEEE，2007：172-182.
[30] RAMANUJAM R S，LIN B. Destination-Based AdaptiveRouting on 2D Mesh Networks[C]//2010 ACM/IEEE 6th Symposium on Architectures for Networking and Communications Systems（ANCS）.La Jolla，California：ACM，2010：1-12.
[31] 欧阳一鸣，贾博远，李建华，等. WiNoC 中无线通信拥塞与故障感知的容错路由算法[J]. 电子学报，2020，48（04）：662-669.
[32] LAN Y，LO S，LIN Y，et al. BiNoC：A Bidirectional NoC architecture with dynamic self-reconfigurable channel[C]//2009 3rd ACM/IEEE International Symposium on Networks-on-Chip. La Jolla，CA，USA：IEEE，2009：266-275.
[33] 欧阳一鸣，孙成龙，李建华，等. 针对瞬时故障和间歇性故障的 NoC 链路容错方法[J]. 计算机研究与发展，2017，54（05）：1109-1120.
[34] 付斌章，韩银和，李华伟，等. 面向高可靠片上网络通信的可重构路由算法[J]. 计算机辅助设计与图形学学报，2011，23（03）：448-455.
[35] KRISHNA T，CHEN C O，KWON W C，et al. Breaking the on-chip latency barrier using SMART [C]//2013 IEEE 19th International Symposium on High Performance Computer Architecture（HPCA）. Shenzhen，China：IEEE，2013，378-389.
[36] 蔡源，罗伟，向东. 基于列分转弯模型的片上网络路由算法[J]. 清华大学学报（自然科学版），2018，58（12）：1051-1058.
[37] NGUYEN S T，OYANAGI S. A Low Cost Single-cycle router based on virtual output queuing for on-chip networks[C]//2010 13th Euromicro Conference on Digital System Design：Architectures，Methods and Tools. Lille，France：IEEE，2010：60-67.
[38] 欧阳一鸣，陈静雯，梁华国，等. NoC 中负载均衡的 AVOQ 路由器设计[J]. 电子测量与仪器学报，2017，31（01）：92-98.
[39] PARASAR M，FARROKHBAKHT H，JERGER N E，et al. DRAIN：Deadlock removal for arbitrary irregular networks[C]//2020 IEEE International Symposium on High Performance Computer Architecture（HPCA）. San Diego，CA，USA：IEEE，2020：447-460.
[40] 欧阳一鸣，王悄，梁华国，等. 基于故障粒度划分的 NoC 链路自适应容错方法[J]. 电子测量与仪器学报，2015，29（08）：1102-1113.
[41] FARROKHBAKHT H，TARAM M，KHALEGHI B，et al. TooT：An efficient and scalable power-gating method for NoC routers [C]//2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip（NOCS）. Nara，Japan：IEEE，2016：1-8.
[42] FARROKHBAKHT H，KAO H，HASAN K，et al. Pitstop：Enabling a virtual network free network-on-chip[C]//2021 IEEE International Symposium on High-Performance Computer Architecture（HPCA）. Seoul，Korea：IEEE，2021：682-695.
[43] 郑小富，顾华玺，杨银堂，等. 基于提前分配路径的低时延片上路由器结构[J]. 电子与信息学报，2013，35（02）：341-348.
[44] DAS A，JOSE J，MISHRA P. Data Criticality in multithreaded applications：An insight for many-core systems[J]. IEEE Transactions on Very Large Scale Integration（VLSI）Systems，2021，29（9）：1675-1679.
[45] CHEN Y，CHANG E，HSIN H. Path-diversity-aware Fault-tolerant routing algorithm for network-on-chip systems[J]. IEEE Transactions on Parallel and Distributed Systems，2017，28（3）：838-849.
[46] RAMRAKHYANI A，GRATZ P V，KRISHNA T. Synchronized Progress in Interconnection Networks（SPIN）：A New Theory for Deadlock Freedom [C]//2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture（ISCA）. Los Angeles，CA，USA：IEEE，2018：699-711.
[47] PARASAR M，JERGER N E，GRATZ P V. SWAP：Synchronized weaving of adjacent packets for network deadlock resolution[C]//The 52nd Annual IEEE/ACM International Symposium on Microarchitecture. Columbus，OH，USA：ACM，2019：873-885.
[48] 李丽，万健，王佳文，等. 基于“包-电路交换”的片上网络回退转向路由算法[J]. 电子与信息学报，2011，33（11）：2759-2763.
[49] CHEN L Z，WANG R S，PINKSTON T M. Critical bubble scheme：An efficient implementation of globally-aware network flow control[C]//2011 IEEE International Parallel & Distributed Processing Symposium. Anchorage，AK，USA：IEEE，2011：592-603.
[50] MOSCIBRODA T，MUTLU O. A Case for bufferless routing in on-chip networks[C]//The 36th Annual International Symposium on Computer Architecture.Austin，TX，USA：ACM，2009：196-207.
[51] CHEN L Z，PINKSTON T M. Worm-bubble flow control[C]//2013 IEEE 19th International Symposium on High Performance Computer Architecture（HPCA）. Shenzhen，China：IEEE，2013：366-377.
[52] KIM J，DALLY W J，TOWLES B，et al. Microarchitecture of a high-radix router [C]//32nd International Symposium on Computer Architecture（ISCA）. Madison，WI，USA：IEEE，2005：420-431.
[53] KIM J，DALLY W J，ABTS D. Flattened butterfly：A cost-efficient topology for high-radix networks [C]//34th annual international symposium on Computer architecture（ISCA）. San Diego，California，USA：ACM，2007：126-137.
[54] 欧阳一鸣，张鹏，王奇，等. WiNoC 中交叉开关仲裁及路由算法设计[J]. 电子学报，2021，49（03）：518-526.
[55] PARASAR M，KRISHNA T. BINDU：Deadlock-freedomwith one bubble in the network [C]//13th IEEE/ACM International Symposium on Networks-on-Chip. New York：ACM，2019：1-8.