第 42 卷 第 5 期 计 算 机 工 程 2016 年 5 月
Vol. 42 No. 5 Computer Engineering May 2016

·体系结构与软件技术· 文章编号:1000-3428(2016)05-0071-09 文献标识码:A 中图分类号:TP391

基于分布式模拟机制的片上网络硬件模拟系统

彭 毅,安 虹,金 旭,程亦超,迟孟贤,孙 荪

(中国科学技术大学计算机科学与技术学院,合肥 230027)

摘 要: 针对基于现场可编程门阵列的 DART 模拟器可扩展性较差和模拟精度较低的问题,提出一种硬件友好的分布式模拟机制。该机制在模拟中采用隐式同步方法,以节点内计数器和节点间缓冲队列取代集中式控制器,将时序同步和计数任务交给每个节点自行处理,从而提高模拟速度。基于该机制,设计并实现片上网络硬件模拟系统。实验结果表明,该系统能达到与业界权威 BookSim 模拟器同级别的模拟精度,模拟速度可达 BookSim 模拟器的 200 倍,相比 DART 模拟器能获得 21% 的速度提升,并且具有较好的扩展性。

关键词: 片上网络;分布式模拟;现场可编程门阵列;多核处理器;时钟精确;动态路障同步

中文引用格式: 彭 毅,安 虹,金 旭,等. 基于分布式模拟机制的片上网络硬件模拟系统[J]. 计算机工程,2016,42(5):71-79.

英文引用格式: Peng Yi, An Hong, Jin Xu, et al. Hardware Simulation System for Network on Chip Based on Distributed Simulation Mechanism[J]. Computer Engineering, 2016, 42(5):71-79.

Hardware Simulation System for Network on Chip Based on Distributed Simulation Mechanism

PENG Yi, AN Hong, JIN Xu, CHENG Yichao, CHI Mengxian, SUN Sun

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

【Abstract】 Aiming at the problems of poor scalability and low precision of DART simulator based on Field Programmable Gate Array (FPGA), this paper proposes a hardware-friendly distributed simulation mechanism. This mechanism uses implicit synchronization method, and replaces the centralized controller with intra-node counters and inter-node buffer queues. In this way, timing synchronization and counting can be handled by each node, which improves the simulation speed. Based on this mechanism, a Network on Chip (NoC) simulator is designed and implemented. Experimental results demonstrate that this simulator can achieve similar accuracy to widely-used BookSim simulator software ones and gain 200-fold speedup. Compared with DART simulator, it accelerates simulation speed by 21% at most and achieves better scalability.

【Key words】 Network on Chip (NoC); distributed simulation; Field Programmable Gate Array (FPGA); multi-core processor; cycle-accurate; dynamic barrier synchronization

DOI: 10.3969/j.issn.1000-3428.2016.05.013

1 概述

随着单个芯片上可集成的核数越来越多,IP 间的通信已成为影响多核、众核系统性能的重要因素,高带宽和良好的扩展性作为片上通信的 2 个关键问题显得越来越重要。然而,传统基于总线的互连方式因总线自身特性很难满足上述要求[1]。片上网络(Network on Chip, NoC)通过并行和分时复用节点连接通信流,能较好解决上述问题[2]。在 NoC 设计过程中,会涉及许多参数,例如拓扑、路由器微结构、缓冲大小等,它们会显著影响系统性能。所以,全系统模拟中必须考虑 NoC 模拟。与其他体系结构研究类似,软件模拟[3]最先被提出并广泛用于 NoC 设计。它为研究者提供了许多便利,例如使用灵活、编

基金项目: 国家自然科学基金资助项目(60970023);国家“973”计划基金资助项目(2011CB302501);国家“863”计划基金资助项目(2012AA010902,2012AA010901)。

作者简介: 彭 毅(1990-),男,硕士研究生,主研方向为片上多处理器;安 虹,教授、博士生导师;金 旭、程亦超、迟孟贤,硕士研究生;孙 荪,博士研究生。

收稿日期: 2015-04-10 **修回日期:** 2015-05-15 **E-mail:** peng1990@mail.ustc.edu.cn

译迅速、开发周期短以及易于调试等,但是其存在模拟时间过长的问题。为缩短模拟过程,研究者只能选择测试集中具有代表性的片段或者进行更高抽象级别的模拟,但却降低了模拟精度和结果可信度。

考虑到软件模拟器在模拟性能和正确性上的冲突,选择具有天然并行性的硬件平台用于 NoC 设计。其中,单片现场可编程门阵列(Field Programmable Gate Array,FPGA)由于拥有更多逻辑、片上线路以及存储单元,使它成为较有前景的 NoC 建模结构工具。目前,已有一些基于 FPGA 的 NoC 评估平台,如采用直接仿真方法,用一个 FPGA 周期仿真一个目标周期。在该方法下,若目标 NoC 不能很好地映射到 FPGA 上,则网络中复杂模块的仿真需要一个很长的 FPGA 周期,进而降低模拟系统的时钟频率,减缓仿真速度。此外,该仿真方法不具备较好的扩展性。为此,研究者提出了新的模拟方法,允许多个 FPGA 时钟周期模拟一个目标周期,加强了基于 FPGA 模拟的灵活性。例如,若要模拟一个运行频率为 40 MHz 的目标 NoC,则可选择频率为 200 MHz 的 FPGA,由5 个 FPGA 周期模拟一个目标周期。该策略能减轻仿真方式产生的问题,然而用于确定 FPGA 周期和目标周期关系的集中式时序控制会产生扩展性问题[4]。为此,本文设计并实现一个快速、可扩展、可精确到时钟的 NoC 硬件模拟系统,并提出一种硬件友好的分布式模拟机制,用于控制 NoC 模拟时序。该机制能使网络中的节点不受集中式时钟计数器约束,在模拟中能够处于不同的目标周期,以解决扩展性问题。

2 相关工作

软件模拟器广泛用于 NoC 研究,BookSim[5-6],WormSim[7]和 Garnet[8]是 3 个典型代表。其中,BookSim 和 WormSim 是专用模拟器,设计过程注重灵活性和模块化思想;Garnet 是全系统模拟器 GEMS 和 Gem5 中负责通信的部分,同 BookSim 和 WormSim 一样,它进行精确到时钟的模拟,采用含虚通道的经典 5 阶段流水线以提高模拟精度。尽管软件模拟器拥有上述优点,但在模拟性能上却受到建模精细程度的制约。典型模拟速度从若干 KIPS(Thousands of Instructions per Second)到 100KIPS 不等。为提高模拟速度,WormSim 去除了虚通道和流水线,该方式降低了结果可信度。DARSIM[9]利用多处理器并行 NoC 模拟,但它在处理全局同步时会产生额外的时间开销。NOXIM 和 NIRGAM 是 2 个基于 SystemC 实现的模拟器。尽管 SystemC 使设计者能够模拟并发过程,但本质上更像一个系统级建模语言,而非硬件描述语言(Hardware Description Language,HDL)。所以,它在性能和精度上均弱于由 HDL 开发的基于硬件的模拟器。综上,由于各种限制导致软件模拟器速度缓慢而不能成为 NoC 建模的最好选择。

由于 FPGA 拥有专用功能单元和大量片上线路,因此能够提供细粒度并行性和可重配置能力,这一性质启发研究者选择其模拟 NoC。FIST[10]是基于 FPGA 的快速、轻量级 NoC 评估平台。它把网络中每个路由节点抽象成负载-延时曲线,通过记录节点负载和查询曲线来计算传输延时。然而,延时评估的正确性依赖于训练模型的代表性和保真度,负载-延时曲线也必须从一个已经存在的模型中获得,这导致无法利用该模拟器研究新式网络。文献[11-13]利用执行驱动模拟方式提供基于 FPGA 的模拟环境,能取得比 FIST 更可靠的模拟结果。文献[11]提出方案包含一些可编程组件,与 SystemC 实现的模拟系统相比可获得 4 个数量级的加速,但是改变网络配置需要重新综合模拟器,这个过程非常耗时且不易操作。DRNoC[12]利用其 FPGA 可重配置性缓解该问题。相对不可重配置方案,它在给定用例下可达到百倍加速。但是,部分可重配置需要特殊的设计流并产生额外的面积开销,可支持的设备也很有限。NoCem[13]通过简化路由节点微结构获得比 Genko[11]更大的模拟规模。

以上所有评估平台都没有对 FPGA 结构和目标 NoC 结构加以区分。这种耦合在目标 NoC 不能较好地映射到 FPGA 上时,会降低 FPGA 时钟频率,影响模拟性能,而且缺少灵活性且不利于权衡各种设计要素。DART 通过解耦合这 2 种结构,提出一种基于 FPGA 的 NoC 模拟器,具有超过 Booksim 100 倍的模拟速度,还借鉴了先前建模工作中的一些经验[14-15]。例如,利用虚拟化机制允许 DART 模拟更大规模的网络,采用划分机制缓解交叉开关的压力。表 1 比较了上述多个典型 NoC 评估平台的性能。表中每行代表研究者关注的不同指标。每一项用 D ~ A + 之间的值加以评估,D 代表最差,A + 代表最优。由表 1 可知,DART 总体上表现最优,但是在可扩展性和模拟精度上还有待提高。上文提到 DART 将 FPGA 结构和目标 NoC 结构解耦合,用多个 FPGA 周期模拟一个目标周期。这种思想的关键是网络中时序的控制和对应。Unit-Delay 技术和动态路障技术是 2 种常用方法,DART 实现时选择了后者,这种方式浪费了一些 FPGA 周期并且给集中式控制器带来很大负担。基于此,本文提出一种分布式模拟机制来解决该问题。

表 1　不同 NoC 评估平台性能比较

指标	BookSim	DARSIM	NIRGAM	FIST	DRNoC	DART
开发语言	C++		SystemC	System Verilog	VHDL	C++/Verilog
可扩展性	A	B+	A	A	B+	B+
花费	A+	A+	A+	A	B	A
可重配置性	A+	A+	A+	A+	A	A+
结果可信度	A	A	B+	C	B+	B+
时钟性能	D	C	D	A+	A	A

3　NoC 模拟系统基础架构设计

本文设计的 NoC 模拟系统在 DART 基础上实现。它继承了 DART 快速、易于重配置、模拟范围广泛等优点。本节描述搭建的模拟平台,并具体介绍了平台中片上网络各个要素(节点微结构、拓扑、路由算法和流控机制[16]等)的设计细节及特点。

3.1　模拟平台

NoC 模拟系统平台设计如图 1 所示。该平台以 BEEcube 公司的 BEE4 为框架,主要由嵌入式控制主机(Embedded Control Host,ECH)和主处理板(Main Processing Board,MPB)两部分构成。ECH 的核心是 PC 模块,它负责向 MPB 供电及监控 MPB 的状态。本文利用 ECH 向 FPGA 发送编程命令和配置文件,并处理接收到的模拟结果。MPB 的基本组件是 4 片 Xilinx Virtex-6 LX550T FPGA,它们通过通用异步收发器(Universal Asynchronous Receiver/Transmitter,UART)总线与 ECH 相连,并利用各自的串口与 ECH 通信。NoC 模拟器的核心部件实现于 FPGA 上,当前版本支持 1 片 FPGA,未来可扩展到 4 片 FPGA,如图 1中虚线范围所示。FPGA 上的模块用于临时存储配置文件和模拟结果。

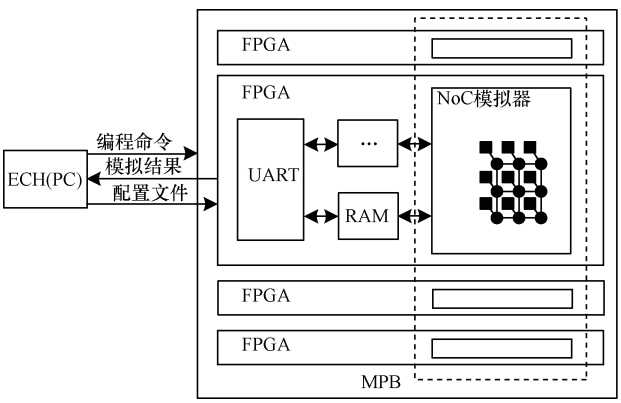


图 1　NoC 模拟系统平台设计

3.2　节点微结构设计

图 1 中 NoC 模拟器部分展示了一个 3×3 Mesh 结构的片上网络。该网络有 9 个路由节点(实心圆),每个路由节点经网络接口(斜线)与本地资源(实心方形)相连。其中,本地资源既可以是传统意义上的处理器核,也可以是 IP 核或用户自定义的功能模块。2 个路由节点之间通过物理通道(水平/竖直线)实现双向通信。这些路由节点、本地资源与物理通道是模拟目标 NoC 的基本物理组件。为了提高 NoC 模拟器的通用性、可配置性,本文设计了一种抽象节点,如图 2 所示。每个节点包含一个流量发生器(Traffic Generator)、一个路由节点(Router)和若干输入端口(Port)。通过抽象可提高节点的重用性,减少模拟器开发工作量。同时,抽象节点可提供很多可配置参数,保证了建模的多样性。

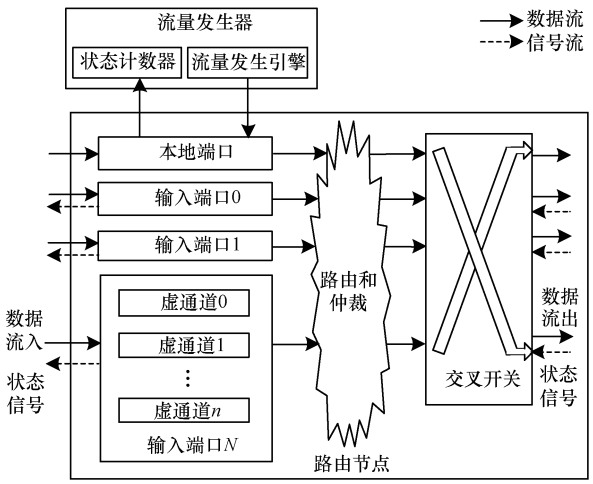


图 2　抽象节点结构

3.3　网络拓扑和路由算法

图 2 中每个 Router 都有一个本地端口和若干 Port。本地端口通过 2 个反向的单向通道(Channel)和 Traffic Generator 相连。每个 Port 也通过该通道和邻居节点的输出端口相连。为便于描述,本文将输出端口以及与该端口相连的输出通道抽象成先进先出(First In First Out,FIFO)模式,在逻辑上与真实网络没有差异,因此,不会影响实验结果。节点在物理上通过与 DART 中类似的划分和全局互联方式实现全连通,在逻辑上只能向拓扑配置文件中规定的邻居发送消息。通过更改拓扑配置文件以模拟不同的 NoC。模拟器路由节点中设置可配置的路由表,路由表的每项说明对应目的地的输出端口。通过配置每项不同的对应关系便能模拟各种路由算法。

3.4 流控机制

流控机制决定消息流经网络时的资源(缓存、通道带宽等)分配方式。模拟器采用经典的基于 flit 的虫孔流控机制,同时在每个输入端口设置多个虚通道(Virtual Channel, VC)[17]。在这种机制下,一个 packet 被分成若干大小相等的 flit。每个 packet 由一个头 flit 标示目的节点,一个尾 flit 标示 packet 结束,头 flit 与尾 flit 之间是携带数据的 flit。模拟中规定,若一个虚通道被某个 packet 占据,则直到这个 packet的所有 flit 都流过,它才能被其他 flit 使用。这种方式保证每个 packet 中的 flit 严格按照它们注入时的顺序在网络中传输,避免了重新排序的开销。

相邻节点间采用基于信度的机制,避免因缓存满或消息冲突引起 flit 丢失。在节点的每一个输出端口,增加一条流向下游节点输入端口的信度通道。在输入端口设置多个虚通道状态向量,每个向量记录端口中一个虚通道的状态。状态向量各项组成如图 3 所示。其中,Occupied(1 bit)表示虚通道是否已经被某个 packet 占用,将状态反馈给上游节点(面对不同方向的 flit 流,一个端口既是输入端口,又是输出端口);Port_No. 记录已占用虚通道的 packet 将向哪个输出端口发送消息;VC_No. 记录发送的消息将进入下游节点输入端口的哪一个虚通道(因为一个节点的某个输出端口只能有一条通向其他节点的通道,一个节点的某个输入端口也只能有一条来自其他节点的通道,所以 Port_No. 已经确定了消息将流向下游节点的哪个输入端口)。EVC_Num. 记录虚通道空闲缓存数。此外,可以增加优先级(Pri.)选项,设置 packet 按不同优先级传输。

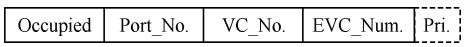


图 3 状态向量

在片上网络基本要素设计完成后,结合图 2,packet 中不同类型的 flit 依照不同行为流经一个节点:头 flit 确定路由信息,并将其保存在虚通道的状态向量中;头 flit 查询状态向量,依照其中信息流出节点;尾 flit 复原状态向量,释放当前 packet 对虚通道的占用。

3.5 模拟配置

NoC 模拟器运行前需要根据图 2 中 ECH 发送的配置文件进行初始化。配置文件有 2 种,分别是网络描述文件和通信描述文件。其中,网络描述文件描述用户需要模拟的目标 NoC 的基本特征,如网络拓扑,每个节点的路由表、通道和路由节点的各种属性;通信描述文件描述了目标 NoC 的流量注入情况,模拟器可以支持 2 种流量注入模式:合成模式和动态模式。合成模式主要对目标网络进行压力测试,在此模式下,图 2 中 Traffic Generator 的流量发生引擎(Traffic Engine)以固定时间间隔向本地路由节点发送固定长度的 packet。动态模式代表模拟器接入全系统评估平台后的流量注入情况。在此模式下,Traffic Engine 根据通信描述文件中的大量 packet 描述符不定间隔注入不同大小的 packet。每个 packet 描述符包括注入时间、packet 大小、源目的节点地址等信息。模拟结果由 Traffic Generator 中的状态计数器(Stats Counter)模块记录。模块中有若干计数器分别记录注入 packet 数、接收 packet 数和节点接收的 packet 总延时。

3.6 可配置参数和虚拟化机制

模拟器为许多属性设置了可配置参数,表 2 列出了这些属性以及目前模拟器可支持的属性值,它们由网络描述文件定义。为节省硬件资源,本文模拟器基于 DART 的虚拟化机制。模拟时在 FPGA 上只实现一个物理节点,以时间换资源的方式,将多个逻辑节点的上下文依次在物理节点上执行来模拟目标网络。

表 2 模拟器的可配置参数

可配置参数	可选参数值
节点数目	2×2 ~ 8×8
节点端口数目	3 ~ 6
拓扑结构	Mesh,Torus,Butterfly 等
虚通道数目	1,2,4
虚通道深度	1 flit,2 flit,4 flit,8 flit
packet 大小	1 flit,2 flit,4 flit,8 flit

4 分布式模拟机制

在精确到时钟的模拟中,目标系统结构的模拟是基础,时序的模拟是关键。时序信息能够帮助研究者确定目标系统做某些动作时处于哪个时钟周期,或者完成某个任务经历了哪些时钟周期。将模拟系统和模拟目标分离导致模型系统本身时序和目标系统时序的不一致。因此,目标系统的时序主要利用系统中各个模块的同步特性,通过模拟或计算方式实现。

4.1 集中式控制

DART 采用动态路障方式同步各个模块并计算目标周期。如图 4 所示,DART 中设置一个全局时钟控制器,用来计数目标时钟周期。时钟控制器向 NoC 中每个节点发送启动信号,开始一个目标周期的模拟。每个节点完成当前目标周期模拟后,向控

制器发送结束信号,时钟控制器收到所有结束信号,标志一个目标周期结束,此时时钟控制器中计数器加 1。网络中每个 flit 含有一个时间戳,当 flit 流经节点时,时间戳值更新为当前值与节点延迟值之和。只有当目标周期与更新后的时间戳值相等时,该 flit 才能流出当前节点。这种方式通过集中的时钟控制器控制了节点间的同步与全局时序,减少了 Unit-Delay[18] 技术中因等待整个模拟过程中最慢模块造成的 FPGA 周期浪费。然而,集中式控制器带来了严重的可扩展性问题。随着模拟规模的增加,由控制器发出和接收的大量组合逻辑信号对 FPGA 上的布局和布线带来很大负担。而且,在每一个目标周期,网络中所有模块仍然受到控制器的制约,先模拟完的模块必须等待模拟速度慢的模块,这势必会损失一些模拟性能。为解决该问题,本文提出分布式模拟机制。

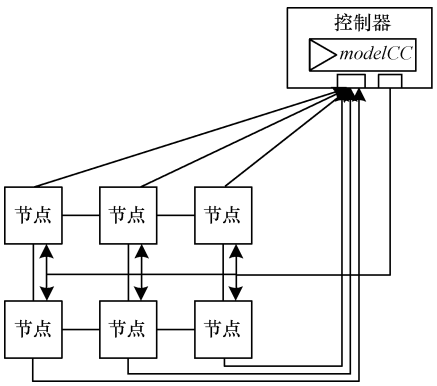


图 4　DART 中的集中式动态路障机制

4.2　分布式模拟原理与实现

4.2.1　节点控制时序

NoC 本质上是一个同步系统。DART 中节点间的同步由时钟控制器决定,如果去掉时钟控制器,则必须寻找一种新的方式模拟 NoC 的同步特性。如第 3 节所述,NoC 由很多基本的虚拟节点组成,宏观上整个网络的消息传输可以看成是微观上每一个虚拟节点从通道接收消息、处理消息、向通道发送消息这一流程的集合。在每一个目标周期,如果缓存充足且没有冲突,节点都应从所有输入端口接收带宽限定的 flit,并向所有输出端口发送本目标周期处理后的 flit。由此得到启发,本文利用节点读输入和写输出 flit 的行为来驱动目标系统时序前进。网络中每一个节点检查自己所有输入端口和输出端口 flit 的流入流出状况,然后决定是否进入下一目标周期。图 5 描述了分布式机制下每个节点模拟一个目标周期时的状态转换,本文假设通道带宽为 1。

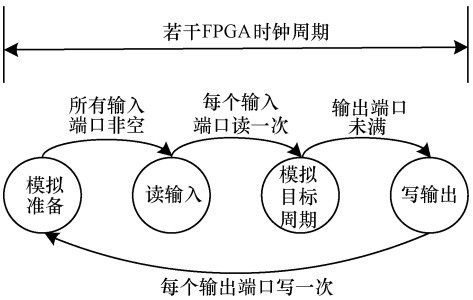


图 5　分布式模拟中的节点状态转换

在目标周期开始时,节点首先检查所有输入端口状态,只有所有输入端口均非空,才能进入读输入状态;然后从每个输入端口读取一个 flit,进入模拟目标周期状态,对每个 flit 进行路由与仲裁;最后当输出端口未满时,将路由与仲裁后的 flit 发往输出端口,当前目标周期结束,进入下一个目标周期。整个目标周期需要若干系统的 FPGA 周期模拟。在模拟过程中,对每个输出端口,若有多个 flit 需要流出,则按仲裁优先级将这些 flit 放入缓存。若没有任何 flit 从此端口流出,则会造成系统时序异常。因为分布式机制依靠 flit 的输入输出驱动时钟前进,若某输出端口在当前目标周期没有 flit 流出,则与此端口相连的下游节点无法收到应有的 flit 来模拟当前周期,这个周期便被遗漏,造成时序错误。为此,本文引入 Emptyflit,它不携带任何有意义的信息。当出现上述情况时,规定节点向相应输出端口输出一个 Emptyflit,下游节点便可由这个 flit 触发新的目标周期模拟,保证时序正确。

4.2.2　目标周期计算

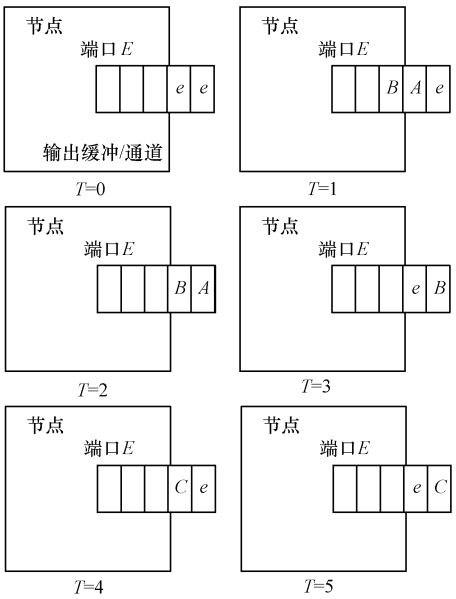
在实现分布式机制时,还需在 NoC 基础结构上额外增加 2 类组件:一类是相邻节点对应端口间具有 FIFO 特性的缓冲队列,用来缓冲节点间模拟速度的不平衡,可用虚拟节点中输出端口(输出通道)实现,不会增加 FPGA 硬件资源开销;另一类是节点内的目标周期计数器,用来标记节点所处目标周期。目前,NoC 的规模和消息传输延迟有限,因此,目标周期计数器设置为 32 bit,这样能满足模拟要求,相对其他模拟部分硬件开销也很小。

在真实 NoC 中,flit 流经某个节点的延时由两部分组成:节点流水线延时和冲突、阻塞造成的延时。前者通常是固定值,后者则因网络拓扑和流量而变化。结合上文 flit 在功能上流过节点的过程,说明分布式机制中如何计算这两部分的节点延时。采用在输出缓冲队列预设 Emptyflit 的方式模拟流水线延时。例如,某个节点的流水线延时为 L,则模拟开始前在节点所有输出端口的缓冲队列预设 L 个 Emptyflit。当真正有效的 flit 进入下一节点时,节点的目标时钟已经增加了 L,达到该 flit 在上一节点经

历 L 个目标周期的效果。通过控制端口输出缓冲队列的行为模拟由冲突、阻塞造成的延时。从输入端口虚通道取出并经过路由和仲裁后的 flit,按如下规定进入相应输出端口的缓冲队列:若缓冲队列已满,则等待;否则,按照仲裁结果将所有应从该输出端口输出的 flit 放入缓冲队列(若 flit 个数大于 1,则表明有冲突出现)。模拟中还可能出现某个输出端口没有有效 flit 流出的情况,此时需检查缓冲队列中 flit 的个数。若小于 L(模拟开始时,缓冲队列中预设的 Emptyflit 个数),则应向缓冲队列中发送一个 Emptyflit,这样才能保证流水线延迟模拟正确。图 6 举例说明了分布式模拟机制模拟节点延时的过程。

目标周期T	1	2	3	4	5	6
到达的flit	A, B			C		
流出的flit	e	e	A	B	e	C

(a)节点输出端口E的flit流动状况



(b)各目标周期输出的缓冲状态

图 6 通过分布式模拟机制模拟节点延时的过程

图 6(a)给出了某一节点在模拟前 6 个目标周期输出端口 E 时的 flit 流动状况。图 6(b)展示了分布式模拟机制下,每个目标周期端口输出的 E 缓冲状态。假设该节点流水线延迟 $L=2$,所以,当 $T=0$ 时,输出缓冲预先设置 2 个 Emptyflit,用 e 表示。当 $T=1$ 时,flit A 和 flit B 均要从此端口流出,即出现了冲突。因为仲裁决定先处理 A,所以按照 A,B 的次序将它们存入输出缓冲,而输出缓冲头部的一个 Emptyflit 也被下游节点取出。当 $T=2$ 时,没有有效 flit 流出,缓冲头部 Emptyflit 被取出后,缓冲中还有 2 个 flit 等于 L,所以,无需向队列中添加新的 flit。但是当 $T=3$ 时,处理完头部 flit 后,队列中只有一个 flit 小于 L,所以,需要向队尾添加一个 Emptyflit。当 $T=4$ 时,flit C 进入缓冲,flit B 流出,缓冲中 flit 数等于 L,无需其他操作。以此类推,flit C 在 $T=6$ 时流出输出缓冲。可以发现,flit A 和 flit C 从流入到流出均间隔了 2 个目标周期,而 flit B 因与 flit A 冲突且优先级较低,所以,经历了 3 个目标周期。这与图 6(a)的理论分析结论相符,说明本文提出的分布式模拟机制能正确模拟 flit 流经节点的延迟。

4.2.3 解耦合节点的重新同步

如前所述,本文提出的分布式模拟机制能够使 NoC 各节点摆脱集中式控制中时钟计数器的束缚。每个节点能自己确定目标周期,模拟快的节点无需等待模拟慢的节点,这样在某个 FPGA 时钟周期,不同节点可能处在不同的目标周期。实际模拟的一些场合需要对模拟器逐步调试或者使用户和模拟器实时交互,这时需要重新同步已经解耦合的节点,将网络中所有节点调整到同一个目标周期。在模拟开始时,网络中的各个节点处在同一目标周期,并且节点间的输出缓冲会因延迟的不同预先设置不同数目的 Emptyflit,记为 L。规定在模拟中的任何时刻,若输出缓冲中 flit 个数大于 L,则该缓冲处于过载(heavy)状态;若输出缓冲中 flit 个数小于 L,则该缓冲处于轻载(light)状态;若输出缓冲中 flit 个数等于 L,则该缓冲处于平衡(balanced)状态。因此,可以通过以下方式重新同步各个节点:如果某个节点有任何端口的输出缓冲处于 light 状态,或者有任何端口的输入缓冲处于 heavy 状态,则时钟计数器加 1,模拟下一个目标周期。

若 NoC 中所有节点都按此方式执行,则整个系统将达到稳定状态,即所有节点处于同一目标周期。这是因为在某一 FPGA 时钟周期,网络中必然存在一个节点集合,集合中的节点模拟速度最快,处在最远的目标周期。按照定义,这些节点将没有任何输入端口缓冲处于 heavy 状态,没有任何输出端口缓冲处于 light 状态,因此,它们无法模拟下一个目标周期。此时,任何进入该集合的缓冲队列必然处于 light 状态,从该集合流出的缓冲必然处于 heavy 状态。因此,与这些缓冲相连的集合外节点有机会模拟下一个目标周期,并且网络中一定存在一个模拟速度最慢的节点,处在最靠后的目标周期。这个节点的所有输入端口都准备就绪,因此,能够模拟下一目标周期。又因为 NoC 中的节点是全连通的,所以能够继续模拟的节点会相互传递,使模拟速度最快的节点集合越来越大,最终包括网络中所有节点。

4.3 性能分析

分布式机制的主要思想是向符合条件的输出缓冲队列发送 Emptyflit。Emptyflit 从输出缓冲队列流入下游节点时,只是作为驱动下游节点时序前进的

标记,下游节点收到并处理后直接丢弃,因此,不会因额外占用缓存对结果产生影响。又因为流控机制作用在相邻节点输入端口的虚通道间,输出缓冲队列只是为了实现分布式模拟机制额外添加的部件,所以其大小和所包含的 Emptyflit 只会影响模拟的性能,而不会影响模拟的正确性。

分布式模拟机制不仅能提高系统的可扩展性,节点间添加的缓冲队列还允许不同节点模拟不同目标周期,从而提高模拟性能。模拟中阻塞、冲突等因素的出现会导致节点在模拟不同目标周期时消耗不同数目的 FPGA 周期。假设有 4 个连续的目标周期 a,b,c,d,其中,模拟 a 和 c 需要 4 个 FPGA 周期;模拟 b 需要 1 个 FPGA 周期;模拟 d 需要 2 个 FPGA 周期。图 7 展示了延迟为 1 的相邻 2 个节点模拟这 4 个目标周期时,动态路障技术和本文提出的分布式模拟机制各自的 FPGA 时钟时序。可以发现,动态路障方式需要 18 个 FPGA 周期模拟这 4 个目标周期,而本文提出的分布式模拟机制只需 15 个 FPGA 周期。还可观察到,在分布式模拟机制中,当系统处于第 5 个 FPGA 周期时,节点 A 模拟第 3 个目标周期,节点 B 模拟第 2 个目标周期。

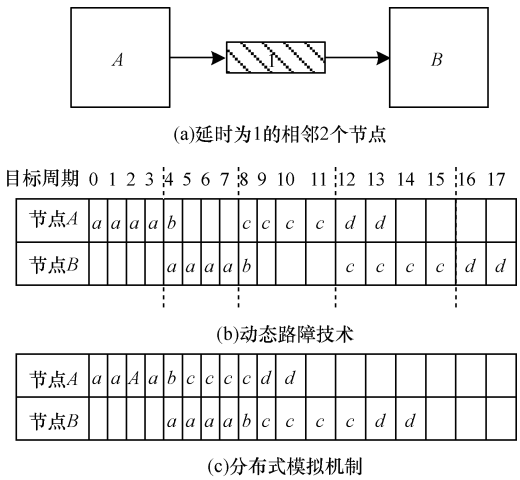


图 7　模拟延时为 1 时相邻 2 个节点的 FPGA 时序

5　实验与结果分析

5.1　实验设计与评价标准

为验证基于分布式模拟机制的 NoC 模拟器设计效果,本文采用 2 种方式实现:一种是C ++ 编写的结构模拟器,它可以灵活配置参数、快速验证正确性以及间接估算模拟性能;另一种是使用 Verilog 硬件描述语言实现的模拟器,它能反映模拟器在真实硬件系统上的性能。目前的版本已经能够针对单片 Xilinx Virtex6 lx550t FPGA,使用 Xilinx ISE 14.2 将设计综合实现并生成网表文件,分析模拟结果。

实验中选择 BookSim 和 DART 作为参照标准。BookSim 是 NoC 研究者广泛使用的软件模拟器,DART 是最近提出的基于 FPGA 的模拟器。通过将本文模拟器(目标 NoC)与 BookSim,DART 进行比较,期望提高模拟器在模拟正确性和性能方面的可信度。以上的 DART 和 BookSim 运行在拥有 3.5 GHz Intel Xeon 处理器的 Linux 服务器上,而硬件描述语言实现的版本则运行在如图 1 所示的平台上。模拟的目标 NoC 基本配置如表 3 所示。其中,网络消息由合成模式产生;Traffic Generator 通过 Bernoulli 过程在规定时间间隔向网络注入固定大小的 packet。

表 3　模拟参数配置

参数	描述
拓扑结构	Mesh
路由算法	Dimension-order 路由算法
路由器微结构	输入缓冲(虚通道)
交换技术	Wormhole 技术
链路延时/(flit cycle)	1
流水线延时/(flit cycle)	5
仲裁机制	轮询机制
流量模式	置换模式
packet 大小/flit	2

本文用 $CPS_{\text{simulator}}$(Simulated Model Cycles Per Second)指标评估 NoC 模拟器的模拟性能。$CPS_{\text{simulator}}$较大表示模拟器每秒可模拟较多的目标周期,具有更好的模拟性能。研究者之前提出的 $IPS_{\text{simulator}}$计算公式如式(1)所示。

$$IPS_{\text{simulator}} = \frac{frequency_{\text{FPGA}}}{CPI_{\text{model}} \times FMR} \tag{1}$$

经变换可得,$CPS_{\text{simulator}}$的计算如式(2)所示。

$$CPS_{\text{simulator}} = \frac{frequency_{\text{FPGA}}}{FMR} \tag{2}$$

其中,$frequency_{\text{FPGA}}$表示 FPGA 可达到的时钟频率;FMR 表示整个模拟过程所消耗的 FPGA 周期数与目标时钟周期数的比值,可表示如下:

$$FMR = \frac{\sum_{i=0}^{numModelCC} cycles_{\text{FPGA}}(i)}{numModelCC} \tag{3}$$

从式(2)可看出,模拟器性能与$frequency_{\text{FPGA}}$和 FMR 有关。选用不同的 FPGA 会使模拟系统具有不同的$frequency_{\text{FPGA}}$。目前,本文的硬件系统频率可达到 70 MHz。若$frequency_{\text{FPGA}}$固定,则模拟器的性能与 FMR 直接相关,即模拟一个目标周期所需的 FPGA 周期数目越少,模拟系统的性能越好。

5.2　结果分析

5.2.1　正确性

本节选择 DART,BookSim 和本文模拟器分别

模拟 3×3 Mesh 结构的网络,网络基本配置如表 3 所示。正确性验证分 2 个方面:(1)通过对实验结果的收集分析,发现本文模拟器能够按照目的地描述将 packet 送往正确的目的节点。(2)实验中计算了不同注入率下每种模拟器的平均 packet 延时,如图 8 所示。可见,在 flit 注入率较低时,3 种模拟器的平均 packet 延时相近,随着 flit 注入率的提高,本文模拟器的模拟结果介于 BookSim 和 DART 之间。

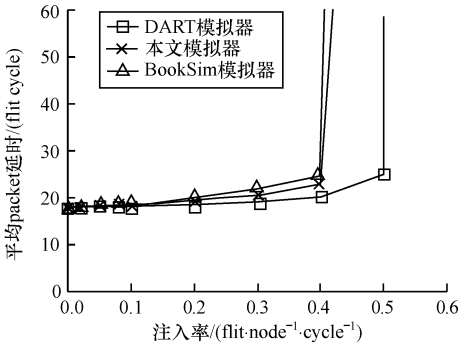


图 8 3×3 Mesh 网络下 3 种模拟器的平均 packet 延时

BookSim 和 DART 模拟结果的差异可归结于两方面:(1)对流水线延时的模拟,BookSim 将其细分为路由延时、交换开关分配延时和虚通道分配延时,每个部分可独立配置参数值,而 DART 只是粗略地将流水线延时设为某固定值。(2)对冲突和阻塞(由缓冲、虚通道、交换开关分配等引起)的模拟,DART 将这部分集中在一个阶段处理,而 BookSim 在模拟中能实时记录冲突情况。所以,实验结果显示在注入率较高时 DART 模拟的平均 packet 值较低,这是因为它低估了 flit 申请资源时冲突发生的概率。为在模拟精度和性能间寻求平衡,本文模拟器在模拟流水线延时时选择了和 DART 一样的策略。而由上文分析可知,分布式模拟机制能够实时模拟冲突的发生。所以,模拟器得到的平均 packet 延时介于 DART 和 BookSim 之间,并且总的曲线走势和 BookSim 十分接近,这说明本文模拟器得到的模拟结果可信度较高。

5.2.2 可扩展性

图 9 比较了 flit 注入率为 0.4 时,不同 Mesh 规模下 2 种模拟器的 *FMR* 值。结果显示,DART 的 *FMR* 值随网络规模的扩大迅速上升,即模拟每个目标周期所需的 FPGA 周期越来越多,这意味着模拟较大规模的网络时模拟器速度会急剧下降。而本文模拟器的 *FMR* 值只是随着网络规模的增大缓慢上升,这样能保证模拟大规模网络时的模拟器性能能够维持在可接受的范围内。出现这种现象是因为集中式方式在节点增多时会因同步和等待产生大量额外开销,而分布式模拟机制给每个节点自主控制时序的权利,节点之间无需相互等待,能更充分地利用 FPGA 周期。

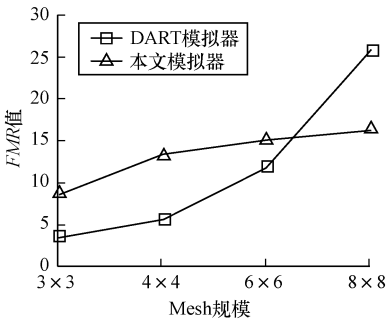


图 9 不同 Mesh 规模下 2 种模拟器的 *FMR* 值比较

5.2.3 模拟性能

本文选择如表 3 所示的 4×4 Mesh 网络作为模拟目标,比较 3 种模拟器在不同注入率下的模拟性能,结果如图 10 所示。图中左侧纵轴表示 BookSim 的模拟速度,即模拟器每秒模拟的目标周期数。右侧纵轴表示 DART 和本文模拟器相对 BookSim 的加速比。可以看出,BookSim 的模拟速度随着 flit 注入率的提高而显著下降。基于 FPGA 实现的 DART 能够利用硬件特性发掘更多的细粒度并行性,因此,其模拟速度下降较少,从而获得了 100 倍~170 倍的速度提升。本文实现的模拟器采用分布式模拟机制取代 DART 的集中式控制方式,使网络中的节点无需相互等待,能够更充分地利用 FPGA 周期。实验结果显示,本文模拟器相比 DART 最多能获得 21% 的性能提升。

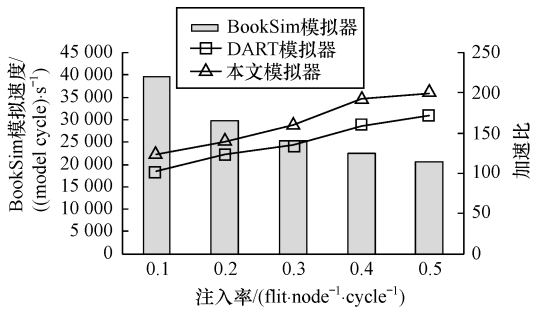


图 10 4×4 Mesh 网络下 3 种模拟器的模拟速度比较

6 结束语

本文提出一种用于 NoC 模拟的分布式时序控制机制。利用 NoC 中节点在每个目标周期从端口接收、发送消息的规律,通过添加节点内计数器和节点间缓冲队列计算目标周期数目,解决了扩展性问题,避免了模拟过程中节点间的相互束缚,能更充分利用 FPGA 周期并提高模拟性能。实验结果表明,采用该机制实现的模拟系统在模拟精度和速度方面相比已有模拟器均有较好提升。未来需要研究不同

模拟目标下缓冲大小的最优选择以及如何在性能和资源间作较好的折衷。随着 NoC 研究者对功耗的日益关注,还需考虑在模拟器中加入功耗模块(如 ORION[19])以拓宽模拟器的适用范围。

参考文献

[1] Bjerregaard T, Mahadevan S. A Survey of Research and Practices of Network-on-chip [J]. ACM Computing Surveys, 2006, 38(1): 1-51.

[2] Benini L, de Micheli G. Networks on Chips: A New SoC Paradigm[J]. Computer, 2002, 35(1): 70-78.

[3] 喻之斌,金　海,邹南海. 计算机体系结构软件模拟技术研究[J]. 软件学报, 2008, 19(4): 1051-1068.

[4] Wang Danyao, Jerger N E, Steffan J G. DART: A Programmable Architecture for NoC Simulation on FPGAs[C]// Proceedings of the 5th ACM/IEEE International Symposium on Networks-on-chip. Washington D. C., USA: IEEE Press, 2011: 145-152.

[5] Dally W J, Towles B P. Principles and Practices of Interconnection Networks [M]. San Francisco, USA: Morgan Kaufmann Publishers Inc., 2003.

[6] Jiang N, Becker D U, Michelogiannakis G, et al. A Detailed and Flexible Cycle-accurate Network-on-chip Simulator[C]//Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software. Washington D. C., USA: IEEE Press, 2013: 86-96.

[7] Hu Jingcao. Design Methodologies for Application Specific Network-on-chip [D]. Pittsburgh, USA: Carnegie Mellon University, 2005.

[8] Agarwal N, Krishna T, Peh L S, et al. GARNET: A Detailed On-chip Network Model Inside a Full-system Simulator[C]//Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software. Washington D. C., USA: IEEE Press, 2009: 33-42.

[9] Lis M, Shim K S, Cho M H, et al. DARSIM: A Parallel Cycle-level NoC Simulator[C]//Proceedings of the 6th Annual Workshop on Modeling, Benchmarking and Simulation. Saint Malo, France: [s. n.], 2010: 1-9.

[10] Papamichael M K, Hoe J C, Mutlu O. FIST: A Fast, Lightweight, FPGA-friendly Packet Latency Estimator for NoC Modeling in Full-system Simulations [C]// Proceedings of the 5th ACM/IEEE International Symposium on Networks-on-chip. Washington D. C., USA: IEEE Press, 2011: 137-144.

[11] Genko N, Atienza D, de Micheli G, et al. A Complete Network-on-chip Emulation Framework[J]. Data, 2005, 1: 246-251.

[12] Krasteva Y E, Criado F, de la Torre E, et al. A Fast Emulation Based NoC Prototyping Framework[C]//Proceedings of International Conference on Reconfigurable Computing and FPGAs. Washington D. C., USA: IEEE Press, 2008: 211-216.

[13] Schelle G, Grunwald D. Onchip Interconnect Exploration for Multicore Processors Utilizing FPGAs [C]//Proceedings of the 2nd Workshop on Architecture Research Using FPGA Platforms. Austin, USA: [s. n.], 2006.

[14] Wolkotte P T, Holzenspies P K, Smit G J. Fast, Accurate and Detailed NoC Simulations[C]//Proceedings of the 1st International Symposium on Networks-on-chip. Washington D. C., USA: IEEE Press, 2007: 323-332.

[15] Papamichael M K. Fast Scalable FPGA-based Network-on-chip Simulation Models [C]//Proceedings of the 9th ACM/IEEE International Conference on Formal Methods and Models for Codesign. Washington D. C., USA: IEEE Press, 2011: 77-82.

[16] Jerger E N, Peh L. On-chip Networks[M]. San Francisco, USA: Morgan and Claypool Publishers, 2009.

[17] Dally W J. Virtual-channel Flow Control [J]. IEEE Transactions on Parallel and Distributed Systems, 1992, 3(2): 194-205.

[18] Pfister G F. The Yorktown Simulation Engine [C]// Proceedings of the 19th Conference on Design Automation. Washington D. C., USA: IEEE Press, 1982: 51-54.

[19] Wang Hangsheng, Zhu Xinping, Li S P, et al. Orion: A Power-performance Simulator for Interconnection Networks[C]//Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture. Washington D. C., USA: IEEE Press, 2002: 18-22.

编辑　陆燕菲

(上接第 70 页)

[9] Nayak N, Mohapatra D P. Automatic Test Data Generation for Data Flow Testing Using Particle Swarm Optimization[M]. Berlin, Germany: Springer, 2010.

[10] Zhu Xiaomei, Yang Xianfeng. Software Test Data Generation Automatically Based on Improved Adaptive Particle Swarm Optimizer [C]//Proceedings of the 4th International Conference on Computational and Information Sciences. Washington D. C., USA: IEEE Computer Society, 2010: 1300-1303.

[11] 陈　翔,顾　庆,王子元,等. 一种基于粒子群优化的成对组合测试算法框架[J]. 软件学报, 2011, 22(12): 2879-2893.

[12] 史娇娇,姜淑娟,韩　寒,等. 自适应粒子群优化算法及其在测试数据生成中的应用研究[J]. 电子学报, 2013, 41(8): 1555-1559.

[13] 时贵英. 改进 PSO 算法在软件测试数据生成中的应用[J]. 计算机工程与科学, 2012, 34(1): 86-89.

[14] 胥小波,郑康锋,李　丹,等. 新的混沌粒子群优化算法[J]. 通信学报, 2012, 33(1): 24-30.

[15] 陈　刚,简华阳,龚　啸. 自适应混沌粒子群算法在 PSS 设计中的应用[J]. 电力系统及其自动化学报, 2012, 24(4): 82-87.

编辑　陆燕菲