一种低开销的面向节点内互连的网络接口控制器

苏勇[1),2)]，曹政[2)]，刘飞龙[1),2)]，王展[1),2)]，刘小丽[2)]，安学军[2)]，孙凝晖[2)]

[1)](中国科学院大学计算机与控制工程学院 北京 100190)

[2)](计算机体系结构国家重点实验室，中国科学院计算技术研究所 北京 100190)

摘 要 高性能计算和云计算的飞速发展对高性能互连网络的设计提出了越来越高的要求：除了要保证高带宽、低延迟和高可靠性等特性，还要面临成本和系统规模的挑战。本文针对这些特性和挑战提出了一种低开销的基于 cHPP 体系结构的超节点网络接口控制器：1）设计了兼容 PCIe 的网络通信协议，降低协议转换开销、减少通信延迟并增强系统可扩展性能；2）采用 PCIe 高速通信接口并支持用户级通信提高软硬件交互效率，面向 MPI 编程模型抽象出高效通信原语（如 NAP，PUT 和 GET）加速大数据传输；3）硬件支持 I/O 虚拟化实现超节点内对网络接口控制器的高效共享。为了对本文的设计进行功能和性能验证，文章基于 FPGA 实现了系统原型，实验结果显示最低延迟为 1.242μs，有效数据带宽可达 3.19 GB/s。

关键词 互连；网络接口控制器；直接存储器访问；PCI Express；I/O 虚拟化

中图法分类号： TP303

A Low Overhead Intra-node Interconnection Oriented Network Interface Controller

SU Yong[1),2)], CAO Zheng[2)], LIU FeiLong[1),2)], WANG Zhan[1),2)], LIU XiaoLi[2)], AN XueJun[2)], SUN NingHui[2)]

[1)] (Institute of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing 100190)

[2)] (State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

Abstract High performance computing and Cloud Computing are requiring the interconnection network: not only high bandwidth, low latency and high reliability, but also low cost and high scalability. To address these challenges, a low overhead hyper-node network interface controller (NIC) for cHPP architecture is proposed: 1) a PCIe-compliant network communication protocol is proposed to decrease the overhead of protocol conversion and the user-level communication is supported to decrease the software latency; 2) high efficient communication primitives such as NAP, PUT and GET are proposed and implemented to accelerate the MPI programming model and large data exchanging; 3) Hardware-based I/O virtualization is supported to enable the efficient NIC sharing in the hyper-node. The proposed hyper-node network interface controller design has been verified based on a FPGA prototype. Evaluations carried out on the prototype show that the lowest latency is 1.242 μs and the highest bandwidth is 3.19 GB/s.

Key words Interconnection; Network Interface Controller; Direct Memory Access; Peripheral Component Interconnect Express; I/O Virtualization

本课题得到国家自然科学基金 (No. 61100014) 资助.苏勇，男，1976年生，博士研究生，工程师，计算机学会(CCF)会员（E200020830G），主要研究领域为计算机体系结构、高性能互连网络，E-mail:sy.pass@163.com. 曹政，男，1982年生，博士，副研究员，主要研究领域为分布式计算、计算机体系结构、高性能计算互连网络等，计算机学会(CCF)会员（20509M）， E-mail:cz@ncic.ac.cn. 刘飞龙，男，1989年生，工学硕士，主要研究领域为计算机体系结构、高性能互连网络， E-mail:liufeilong@ncic.ac.cn. 王展，男，1986年生，博士研究生，主要研究领域为计算机体系结构、虚拟化技术，计算机学会(CCF)会员（E200020831G），E-mail：wangzhan@ncic.ac.cn. 刘小丽，女，1986年生，工学硕士，助理工程师，主要研究领域为高性能计算、虚拟化技术，计算机学会(CCF)会员（E200037123M）， E-mail:liuxiaoli@ncic.ac.cn. 安学军，男，1966年生，博士，高级工程师，博士生导师，主要研究领域为计算机体系结构、高性能互连网络等， E-mail:axj@ncic.ac.cn. 孙凝晖，男，1968年生，博士，研究员，博士生导师，主要研究领域为并行体系结构、分布式操作系统、高性能计算等，E-mail:snh@ict.ac.cn.

1 引言

大规模并行系统广泛应用于高性能计算和云计算领域，两个领域都对通信系统提出新的需求。高性能计算在科学研究，工程技术以及国防军工等方面的应用取得了巨大成功，计算能力突飞猛进。基于异构架构的“天河II”[1]超级计算机以峰值计算速度每秒 5.49 亿亿次、持续计算速度每秒 3.39 亿亿次双精度浮点运算的优异性能成为全球最快超级计算机。异构计算是一种高效利用各种计算资源的并行和分布式计算技术，在提升计算性能的同时降低成本和能耗，已经成为高性能计算发展的新趋势。因此，对异构计算模式的支持是高性能计算的现实需求，但是在传统结构中，协处理器仅仅挂载在I/O总线上作为加速部件使用，大量通信需要主处理器内存中转，难以获得等同的网络性能。计算能力的不断提高也要求互连网络必须具有超高带宽、超低延迟的高性能，因此，异构计算服务器中的通信性能亟待提高。此外，随着系统规模的不断增大，系统的成本和功耗越来越高，有效降低成本和功耗也是高性能互连网络面临的难点问题。云计算在助力企业发展，推动技术进步等方面发挥了重要作用。虚拟化技术在云计算领域得到了迅猛发展，特别是I/O虚拟化，就像光纤入户技术一样，成为虚拟化技术的“最后一公里”。在虚拟化环境下，大量并发的高吞吐率负载对网络接口控制器提出了严峻挑战，因此，迫切需要加强对I/O资源的合理高效的共享。

面对上述在性能和共享能力上的挑战，本文基于 HPP（Hyper Parallel Process）体系结构提出了cHPP（configurable HPP）体系结构，针对面向异构计算的通信加速和基于硬件的节点内 I/O 资源高效共享设计并实现了一种低开销的面向节点内互连的网络接口控制器。文章组织结构如下：第 2 节介绍了 cHPP 体系结构；第 3 节阐述了网络接口控制器设计的关键技术；第 4 节阐述了网络接口控制器的具体实现方法；第 5 节进行了性能评测并得出有关结论；第 6 节阐述了目前主流高性能计算机网络接口控制器的相关研究，对比了各自的特点和不足，最后是全文的总结和对未来工作的展望。

1 Top500.org (2013) TOP500 List. http://top500.org/lists/2013/06

2 超节点控制器结构

2.1 HPP体系结构

超并行(HPP)体系结构[1]是中国科学院计算技术研究所提出的一种基于超节点通信性能优化的高性能计算机体系结构，在保证分布式系统的高扩展性的同时基于硬件实现了全局物理内存共享，支持基于共享存储的编程模型。超节点是指多个处理单元通过一个超节点控制器连接起来构成超级节点，该结构能够有效降低超节点内部计算单元间的通信延迟，同时能够减少节点数量、降低互连网络规模，系统的平均通信延迟也随之降低。

如图 1 所示，HPP 体系结构实现了芯片、节点、系统多级并行结构：通过多核处理器实现核间并行、超节点内部采用异构加速器和通用处理器实现处理器间并行、超节点间通过互连网络构成机群系统实现超节点间并行；支持全局地址空间，对超节点内的内存和 I/O 资源统一编址，使超节点的处理器可对全局资源高效共享；支持多通道并发的核到核之间的通信；超节点操作系统具有单一系统映像并有效支持 MPI (Message Passing Interface)和 PGAS (Partitioned Global Address Space)编程模型。

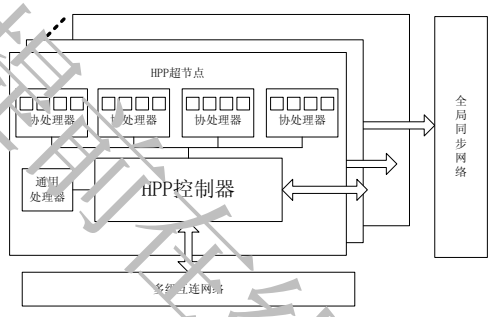

图 1 HPP 体系结构

2.2 cHPP体系结构

为满足高性能计算和云计算对性能的共同需求，同时兼顾异构计算和虚拟化对通信系统的差异性需求，在曙光 6000 HPP 体系结构的基础上提出了 cHPP 体系结构。cHPP 体系结构的首要目标是加速异构计算，提高节点计算密度，通过 cHPP 控制器，结合高效通信接口，实现了异构处理器间的直接通信。相比于传统结构，这种体系结构可以消除通信瓶颈，提高通信效率并有利于系统规模扩展。基于硬件支持资源的聚集和高效共享是 cHPP 体系结构的另一重要目标，因此 cHPP 控制器除了实现处理器对内存和 I/O 资源的高速访问，还支持灵活有效

的资源聚集和共享。

cHPP 控制器用于构建面向高性能计算应用同时兼顾云计算需求的新型高性能服务器。因此 cHPP 控制器既要实现高性能的互连，还要提供虚拟化的相关支持。图 2 描述的是 cHPP 控制器用于节点内互连的场景，cHPP 控制器既可以连接通用处理器和协处理器，也可以直接连接 I/O 设备或 I/O 桥。cHPP 控制器提供全面的 I/O 虚拟化支持，节点内的 I/O 设备可以被虚拟成若干虚拟设备。此外，cHPP 控制器本身也可以被虚拟成若干虚拟控制器。所有的虚拟设备和虚拟控制器均可被直接分配给节点内的虚拟机，实现 I/O 设备被处理器的直接共享。

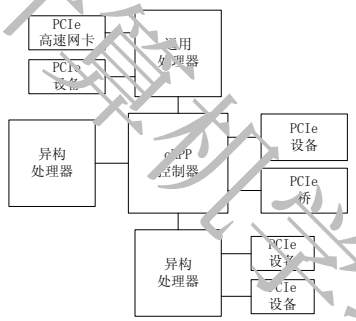

图 2 cHPP 超节点结构示意图

2.3 cHPP超节点控制器

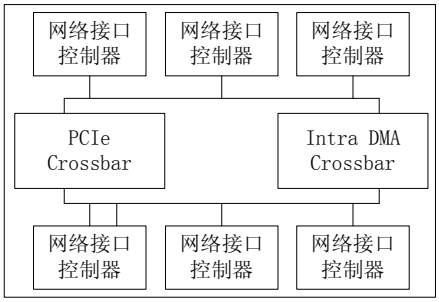

图 3 cHPP 控制器结构框图

为满足上述场景的需要，cHPP 控制器主要负责超节点数据通信和 I/O 高效共享的功能，支持全局地址空间和用户级通信，采用 PCI Express 标准（PCIe）的高速 I/O 接口提升链路性能，并增加了对单根虚拟化(SR-IOV)规范的支持，实现超节点内 I/O 资源的高效共享。如图 3 所示，每个 cHPP 控制器含多个网络接口控制器和交换模块，cHPP 控制器间通过 PCIe 链路直接互连，支持任意网络拓扑结构。系统规模可根据需求灵活配置，具有良好的伸缩性。PCIe 交叉开关模块用于实现处理器与 I/O 设备间 PCIe 消息的交换；Intra DMA 交叉开关模块用于实现节点内 DMA 数据的交换。因此，cHPP 控制器可支持多种拓扑的直接网络互连，可提供节点内多处理器间高速通信和 I/O 高效共享。网络接口控制器是保证节点内高性能通信和资源共享的关键部件，网络接口控制器的微体系结构和网络通信协议是本文的主要研究内容。

3 网络接口控制器设计

cHPP 控制器面向高性能计算，同时兼顾云计算的需求，其网络接口控制器需要支持高性能的通信和高效的共享。通信延迟决定了高性能互连网络的性能而可扩展性则决定了网络的规模，二者是衡量网络性能的关键因素。网络通信协议决定了网络接口控制器数据传输的效率，通信接口决定了软硬件的交互效率，通信原语的定义和实现机制则决定了控制器的功能和硬件效率，共享机制决定了控制器被共享的能力，这些方面共同构成了网络接口控制器的有机整体。因此，本节围绕低延迟与可扩展性能对节点内通信进行优化，从低开销通信协议，高性能通信接口，高效通信原语和高效的 I/O 共享 4 个方面对网络接口控制器进行了设计分析。

3.1 低延迟与可扩展性能分析

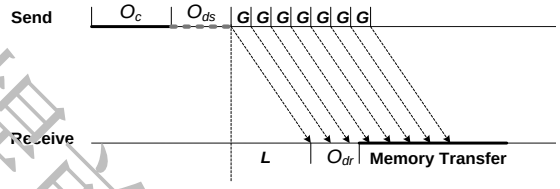

图 4 基于 Chain-DMA 引擎的数据传输时延

LogP[2]模型以很少的参数来反映并行计算的关键技术，但主要是针对短消息进行分析。LogGP[3]模型对长消息传输进行了分析。本文在 LogGP 模型的基础上对基于 Chain-DMA 引擎的网络接口控制器执行长消息传输进行了系统延迟分析。端到端传输时间定义为发送节点网络接口控制器开始 DMA 操作到接收节点接收最后一个字节为止。如图 4 所示，采用所设计的 Chain-DMA 引擎的网络接口控制器传输 M 字节消息所需的时间由公式（1）定义。

$$T(P)=O_c + O_{ds} + (M - 1) \times G + L(P) + O_{dr} \quad (1)$$

其中，O_c 为处理器启动消息发送的时间；O_{ds} 是 DMA 启动开销，包括门铃启动时间和描述符读取时间；M 是消息长度（字节）；G 是网络“间隔”（周期每字节），其倒数为网络带宽；$L(P)=H \times r$ 是消息包头通过网络的平均时间，H 是消息通过的平均跳数，r 是每跳处理时间和线路延迟之和，P 是处理器数目；O_{dr} 是 DMA 接收处理延迟。从公式（1）

可以看出：设计高效的软硬件接口可以加速消息发送，有利于压缩 O_c 开销；建立快速的 DMA 启动机制，有利于降低 O_{ds} 开销；构建高效的 DMA 引擎可实现数据的快速传递，设计低开销的通信协议都有利于减少 O_{dr}；提供更高的网络带宽可降低 G，因此采用高带宽的通信链路有利于减少通信延迟；压缩网络直径，降低平均跳数 H 也可减少网络延迟。因此，可以围绕上述各项指标设计网络接口控制器以提供高性能的通信能力。对于大规模互连网络来说，网络拓扑结构应具有良好的扩展性。理想拓扑结构应具有对称性、扩展粒度低、等分带宽大、网络直径短、节点度适度等优良特性。为此，比对了不同网络拓扑结构对延迟和可扩展性能的影响。为简化分析，根据以往的经验数据参数取值分别为 O_c=200ns, O_{ds}=528ns, L=240ns, O_{dr}=160ns, M=2K Bytes, G=1/4 (ns/byte)。根据文献[2]的统计，直接网络的平均距离 H 与处理器数目 P 的关系如表 1 所示。图 5 显示了在各种不同拓扑网络情况下的通信延迟性能。数据表明，网络接口控制器在 Hypercube 网络具有最好的延迟性能，而且随着规模的扩展缓慢的增长，说明具有良好的可扩展性。

表 1 网络平均距离和处理器关系

网络类型	平均距离 H(P)
Hypercube	$(1/2)\times\log(P)$
3D Torus	$(3/4)\times P^{1/3}$
3D Mesh	$P^{1/3}$
2D Torus	$(1/2)\times P^{1/2}$
2D Mesh	$(2/3)\times P^{1/2}$

图 5 基于 LogGP 模型的通信延迟性能

3.2 低开销通信协议

由于具有高带宽、低延迟、高吞吐率等特性，PCIe 已经成为实际上的 I/O 总线标准。PCIe 提供高速点对点的单/双工通信，物理链路采用差动信令以提高传输距离。最新的 PCIe 3.0 架构单信道(×1)单向带宽即可接近 8 GB/s。主流处理器都直接支持 PCIe 协议，通过 PCIe 可直接与 I/O 设备通信，减少协议转换开销，降低通信延迟；支持 SR-IOV 技术可实现多虚拟机间 I/O 资源的高效共享；基于信用的流控机制确保链路层的可靠传输；CRC 校验支持链路层错误检测，支持出错自动重传，链路可靠性高。因此，从带宽、传输距离、可靠性等方面看，PCIe 可用于高性能计算的系统级网络互连，实现大规模高性能互连网络的高速互连，但是作为 I/O 总线，PCIe 的典型应用是树形拓扑，处理机是树的根，而 I/O 设备则是树的叶子。标准 PCIe 交换机构成的网络，通常是若干个功能独立的子树的集合，子树之间并无数据交换。虽然这已经满足了在 I/O 扩展方面的需要，但是根与根之间不能通信，终端叶子节点之间也无法直接通信，难以构造复杂的拓扑网络，无法用于处理器间通信。因此，网络通信协议在 PCIe 协议的基础上进行了拓展，可充分利用 PCIe 标准的优良性能并拓展了处理器间直接通信能力。

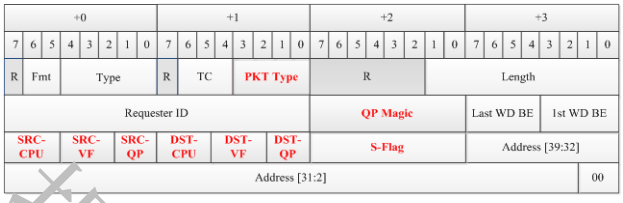

图 6 网络包硬件包头格式

完整的网络包是由硬件包头、有效数据载荷和软件包尾构成。基于 PCIe 事务层包格式，本文的网络包头格式如图 6 所示，其中红色加粗字段为重定义字段，包括：PKT Type 定义了网络包的类型（包括 NAP/PUT/GET）；SRC_ID 包含了源数据的处理器号，虚功能号和队列对号；DST_ID 包含了目标处理器号、目标虚功能号和目标队列对号；增加了 QP Magic 域用于保护信息，并通过 S_Flag 域进行控制（如完成事件发送控制等）。延用了 PCIe 类型域(Type)、流量类型域、长度（Length）、地址（Address）、请求 ID（Requester ID）等 PCIe 的链路层关键域，其含义与 PCIe 协议一致。网络接口控制器根据数据长度域（Length）、地址域（Address）、源节点和目标节点的数据及控制信息进行数据传输。

基于 PCIe 协议定义的网络消息格式更适合于局部互连：1）充分发挥 PCIe 高速可靠的链路层性能，使网络消息直接面向内存操作，不仅减少了 I/O 总线与网络间的协议转换开销，还无缝兼容 PCIe 设备，实现 I/O 设备访问与网络通信功能的融合；2）

提高了有效负载率，PCIe 包头为 16 字节，数据负载 0-4096 字节，相比面向大规模系统的互连网络（如 InfiniBand[4]的包头最大为 94 字节，IBM Bluegene /Q[5][6]和 K Computer Tofu[7]网络包头均为 32 字节），在相同物理链路带宽条件下，可有效提高有效负载带宽。

此外，为使主机可以识别全局资源，实现处理器对节点内资源的高效访问，通信协议支持全局统一地址空间，对超节点内全局物理内存和 I/O 地址空间统一编址，使得内存及 I/O 资源均拥有全局唯一的地址。全局地址空间避免了系统中复杂的地址映射和变换关系，用户可以直接访问系统资源，能够简化系统设计，提高系统性能。

由于地址空间全局可见，为防止误操作或恶意进程造成的非法访问，DMA 引擎还采用了简单高效的地址保护机制：通过绑定密钥来限制访问权限，接收方会检验网络包包头的 QP Magic 字段（如图 6 所示），只有与通信建立阶段协商一致的密钥匹配才允许数据传输，否则视为非法访问，拒绝数据请求，进而实现安全、隔离的用户访问。

3.3 高性能通信接口

高速PCIe接口

网络接口控制器不仅要实现多处理器间的互连通信，也要实现 I/O 设备的扩展。本文的网络接口控制器端口选择 PCIe 总线，可充分利用 PCIe 高带宽、低延迟的传输性能，高可靠的链路层通信和丰富的服务质量支持等方面的优势。通过实现兼容 PCIe 的网络通信协议（见第 3.2 节），在无缝兼容现有 PCIe 设备的前提下，将 PCIe 总线扩展至处理器间互连领域。

用户级通信

本文设计了用户级通信接口降低通信过程中的软件开销。用户级通信[8]也被称为操作系统旁路，通过消除操作系统转发引入的内存拷贝，实现应用程序对底层硬件设备的直接访问。

本文实现了基于 QP（Queue Pair：队列对）的用户级通信接口，QP 包括发送队列（Send Queue）、接收队列（Receive Queue）和控制队列（Control Queue），其中发送队列缓存需要执行的发送请求，接收队列缓存接收到的数据，控制队列用于缓存发送队列和接收队列所需的完成事件通知。需要说明的是，控制队列仅实现了硬件向应用程序的通知机制（发送或接收完成），应用程序向硬件的通知（启动一次发送操作）采用了门铃（Doorbell）机制（详见第 3.4 节）。QP 的队列存在于主机内存，与应用程序或通信进程绑定，因此应用程序通过操作其独占的 QP 实现对硬件的访问。

3.4 高效通信原语

通信原语定义

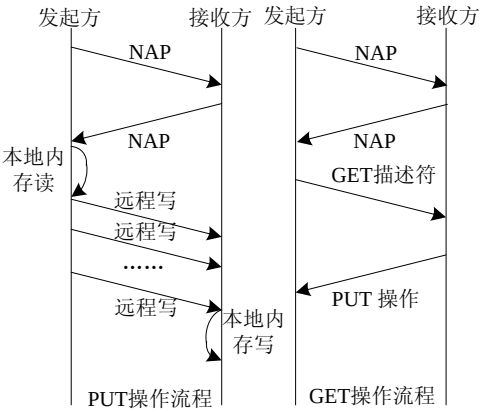

图 7 PUT、GET 操作示意图

通过定义高效的硬件原语，网络接口控制器可实现通信协议的卸载（Offloading），提高通信效率。消息传递编程模型（MPI）是目前高性能计算应用中最常见的编程模型，是高性能计算领域事实上的标准，因此 cHPP 控制器主要针对 MPI 的两种通信方式进行加速：适用于小消息的 eager 通信模式和适用于大消息传递的 rendezvous 模式，抽象出 NAP、PUT、GET 等通信语义。

无地址包（Non-address Packet: NAP）是为了加速小消息通信抽象出来的通信原语。NAP 命令只需知道目标节点信息，不需具体地址信息，因此叫做无地址包。DMA 引擎将接收到的 NAP 消息存放在内存中事先分配好的接收缓冲区内，再将其复制到用户程序空间，主要用于少量数据和控制信息的传递,可以用来实现 MPI eager 通信和同步操作。NAP 操作有两种：立即数和间接数。NAP 立即数是指描述符中直接包含待传输的数据，而 NAP 间接数是指描述符中只包含有需要传输数据的地址和长度信息，两种类型是通过描述符中的类型域区分的。

PUT 通信原语用于大数据量消息的传输，可看作是大块内存写操作，需要事先通过握手协商获得内存地址信息，可以直接将数据写入到用户程序的内存空间。PUT 通信原语支持 MPI 的 rendezvous 通信和 MPI-2 的 put 操作，采用 Chain-DMA 描述符，可支持复杂的通信模式。GET 通信语义和 PUT 类似，只是数据流是反向的，用于大数据量消息的读取和写回。

图 7 为 PUT 和 GET 操作的示意图，在启动 PUT 和 GET 操作之前，发送方均需要 2 次 NAP 操作协商传输的源数据和目标地址等信息。对于 PUT 操作，从发送方读取的数据将封装为若干 PUT 数据包发往目标节点，待数据传输完成接收方发送 NAP 消息告知通信完成；而 GET 操作中，发送方则直接将协商获取的地址等信息打包为 GET 数据包发送，接收方对 GET 数据包进行解析，根据所得信息，将发送方 GET 操作重构为接收方的 PUT 操作后，执行该 PUT 操作流程。

原语启动机制

通信原语的启动，依赖于主机向 DMA 引擎提供包含源数据和目的地址等相关信息。DMA 启动方式可以分为描述符启动和门铃启动两种。在描述符启动方式中，处理器直接将描述符写往 DMA 引擎，DMA 引擎根据描述符信息去读取内存数据。描述符启动的优点在于启动速度快，DMA 引擎接收到描述符后可以直接发起内存读取操作，缺点是写描述符过程占用处理器时间，且原子性差，为保证写描述符过程中不被其他 DMA 描述符打断，增加了软件的加锁操作开销。

在门铃启动方式中，用户进程先将描述符存储在指定内存中，通过向对应的门铃 FIFO（First Input First Output）写入门铃信息启动 DMA 操作，门铃承载描述符在内存空间的位置信息，由地址域和长度域构成：地址域是描述符所在的物理内存空间首地址，长度域表示描述符的长度信息。门铃操作触发描述符命令单元读取描述符内容，DMA 引擎得到描述符后再根据描述符信息去读取数据进行传输。

门铃启动的优点在于操作的原子性，由于描述符内容较少，因此主机只需写一次门铃寄存器（在网络接口控制器中）即可，不会被其他操作所打断。同时可以使描述符的格式更加灵活和复杂。门铃启动的缺点在于启动速度慢，相对于描述符启动方式，需要先读取描述符，额外引入了一次内存读取操作。通常 DMA 引擎是被所有应用程序所共享的，当多个进程或线程希望同时使用 DMA 引擎时会引起竞争。因此，DMA 启动操作应该是“原子”的，是指一次 DMA 操作不会被其它的 DMA 操作打断。通过门铃启动实现 DMA 引擎的虚拟化可以避免竞争。门铃启动机制可支持更多的进程高效地进行 DMA 操作，既保证了 DMA 启动的原子性，又实现了良好的扩展性。而且，在高性能计算中，进行大数据量的传输时，门铃启动增加的额外开销相比软件加锁开销并不显著。特别是在多核多进程的情况下，门铃启动的原子性尤为重要，因此本文的网络接口控制器 DMA 引擎选用门铃启动的方式。

描述符定义了 DMA 操作的必要信息，由用户进程事先写入指定内存空间。描述符命令单元根据门铃信息，将描述符从主存读取到 DMA 引擎，DMA 引擎根据描述符判断 DMA 操作的类型，获得数据传输的控制和数据信息。如表 2 所示，Chain-DMA 型描述符在硬件包头信息域定义了 DMA 操作的类型、控制信息和地址保护等信息；源数据信息域和目标数据信息域指定了数据的源地址和长度信息以及接收方数据存储信息,采用队列形式的数据结构可以实现灵活的数据传输；软件包尾由上层软件使用并对硬件透明。Chain-DMA 型描述符优化了虚地址连续、物理地址离散的数据块的传输，只需读取一次描述符即可实现任意源地址、任意长度、任意目标地址的数据传输。同时 Chain-DMA 只需源数据信息域队列中的总长度与目标数据信息域队列中的总长度相等即可，并不要求队列中每一项均相等，因此可灵活的支持多个页面的大量数据传输，对于大数据通信有加速作用。

表 2 Chain-DMA 描述符

硬件包头
源数据信息[ ]
目标数据信息[ ]
软件包尾

执行流程及死锁避免

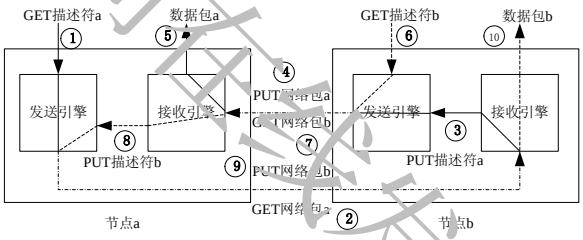

图 8 GET 通信死锁过程

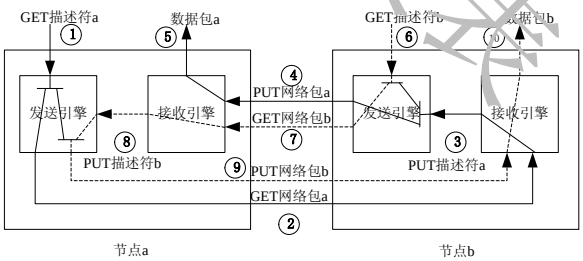

图 9 GET 通信死锁避免方法

通信原语的执行流程开始于 DMA 根据门铃信息读取描述符信息之后，DMA 引擎根据描述符类型

域信息执行相应的操作：

- **NAP：**将描述符中的数据（立即数）或根据描述符中源地址读回的数据（间接数）封装成一个 NAP 网络包发往网络，由于 NAP 网络包不含目标地址信息，接收方的接收引擎接收到 NAP 网络包后，先向本地申请缓存地址，再将数据写入数据接收缓冲区；
- **GET：**图 8 中标号①至⑤是节点 a 发送 GET 描述符向节点 b 读取数据的流程——发送引擎将 GET 描述符直接作为数据封装为 GET 网络包发往节点 b（图 8①和②），节点 b 接收引擎从网络包提取到 GET 描述符后转换为本地发起的 PUT 操作（图 8③），节点 b 的发送引擎将所请求的数据以 PUT 网络包的形式发往节点 a（图 8④），最后节点 a 根据 PUT 网络包包头所携带的目标缓冲区地址域信息（在握手过程中获取）将数据直接写入目标进程的接收缓冲区中（图 8⑤）；
- **PUT：**上述 GET 是转换为 PUT 实现，因此 PUT 的主要执行流程已在 GET 中描述（图 8④和⑤）。

当通信双方同时发起 PUT 或 GET 操作时，图 8 中的执行流程（将 GET 转换为接收方 PUT 执行）将存在死锁。如图 8 中的②至④（节点 a 发起的 GET 操作）和⑦至⑨（节点 b 发起的 GET 操作）形成环路，而死锁的原因在于 GET 网络包与 PUT 网络包形成的网络资源竞争。针对该问题，本文采用虚通道来避免死锁，如图 9 所示，通过为 GET 设置独立的虚通道，解除了 GET 网络包与 PUT 网络包争用。

3.5 I/O高效共享

在云计算环境下存在大量并发通信请求，这对通信接口提出了更高要求：高性能通信，即承载大量通信请求的能力；其次是高效共享，在极小损失通信效率的前提下，实现多虚拟机对 I/O 资源的共享访问。高性能计算与云计算对于网络接口控制器在性能方面的需求是一致的，即包括更高的峰值能力和更高的实际通信能力。同时，还应兼容现有 PCIe 设备，设计支持 SR-IOV 协议，实现超节点内部的 I/O 资源在多虚拟机间的高效共享。

DMA 引擎支持 SR-IOV 规范，可被虚拟为若干个 DMA 引擎，每个虚拟 DMA 引擎都可以被当作独立的设备分配给虚拟机使用，实现虚拟机间对 DMA 引擎的充分共享。每个虚拟机可通过 DMA 引擎直接访问 IO 资源，从而实现多个虚拟机的用户级 I/O 高效共享。每个虚功能（VF）拥有多个 QP，对应一个独立的门铃 FIFO 来缓存 DMA 请求。通过仲裁模块对每个门铃 FIFO 的请求进行仲裁，基于优先级仲裁可实现 VF 之间的差异化服务。

4 网络接口控制器实现

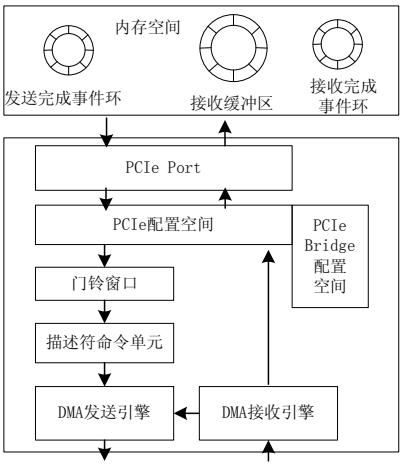

图 10 网络接口控制器结构框图

如图 10 所示，网络接口控制器主要由 PCIe 端口、PCIe 配置空间、门铃启动窗口、描述符命令单元和 DMA 数据发送及接收引擎构成。PCIe 端口采用标准 PCIe Gen2 链路。PCIe 配置空间则实现了 PCIe SR-IOV 功能。门铃窗口模块用于接收启动 DMA 操作所需的门铃信息，门铃将被描述符命令单元解析，并根据解析结果从主存中获取 DMA 描述符。DMA 描述符则提交给 DMA 发送引擎，DMA 发送引擎根据描述符获取源目的信息后，进行数据的读取及发送。DMA 接收引擎则负责数据的接收，解封装并将数据送到相应的内存空间。DMA 操作需要在内存空间开辟 3 块缓冲区：发送完成事件缓存用于通知主机本次数据发送完毕；接收完成事件缓存用于通知数据成功接收；而接收数据缓存用于存放接收的数据，只用于 NAP 操作。每个缓存都是逻辑上的环形结构，每完成一次 DMA 操作都会将指针指向下一项。

4.1 DMA发送引擎

DMA 发送引擎整体框图如图 11 所示，它负责接收主机发来的门铃，并对门铃请求进行响应，根据门铃读取描述符，然后再根据描述符的相关信息读取数据，并打包成网络包进行传输。

门铃模块包含 8 个 FIFO 分别用于存储 8 个功能（PF 和 7 个 VF）对应的门铃，每个 FIFO 深度为 32，即每个功能最多能同时支持 32 个 DMA 请求。

DMA 发送端的流控则交由软件负责，保证每个功能同时发起的 DMA 请求不超过 32 个。

描述符读取模块负责读取门铃模块中的 FIFO，并根据门铃中的内容生成读取描述符的 PCIe 读包。描述符提取模块负责接收并处理读取描述符的返回包，在接收到返回包后，首先从 PCIe 包中提取出描述符，在描述符提取出来后再根据描述符的种类进行处理。对于 PUT 描述符和 NAP 描述符，模块将描述符中的控制信息、源数据信息和目的信息分别提取出来并缓存到相应的 FIFO 中。对于 GET 描述符，模块则直接将其打包为 GET 网络包并发往 GET 交叉开关传输。除了本地的描述符外，描述符读取模块还接收由上传模块接收到的 GET 描述符，并将 GET 描述符转换为 PUT 描述符后存入相应的 FIFO 中。

数据读取模块根据从描述符中提取出来的源数据信息，生成相应的 PCIe 内存读包。具体的生成规则同描述符读取模块。数据提取模块负责接收返回的包含源数据的 PCIe 返回包，并从中提取出相应的源数据。数据重排序模块负责数据整合，方便网络包打包模块使用。由于 PCIe 协议对于单次内存读取的限制，一个源数据项可能会对应多次内存读取，而一次内存读取也可能对应多次返回。因此数据提取模块提取出的数据是不规则的，为了减轻网络包打包模块的负担，加快打包速度，源数据重排列模块负责将提取出的源数据按照网络包打包模块的要求进行重排列。网络包打包模块负责根据描述符控制项和目的项的信息，将读取的源数据进行打包并发往交叉开关（intra DMA 交叉开关）。

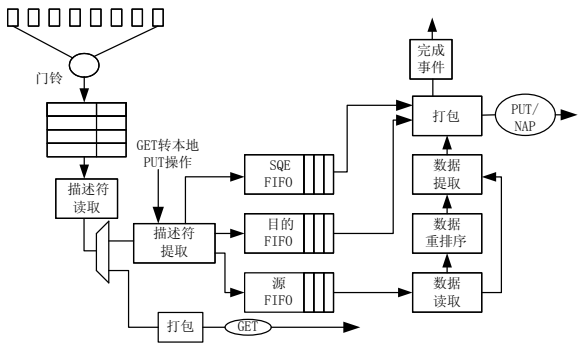

图 11 DMA 发送引擎结构框图

4.2 DMA接收引擎

DMA 接收引擎主要负责接收网络包，并根据网络包的类型进行相应的处理。主要由接收分发模块，数据上传模块和缓冲区管理等模块构成，具体结构如图 12 所示。

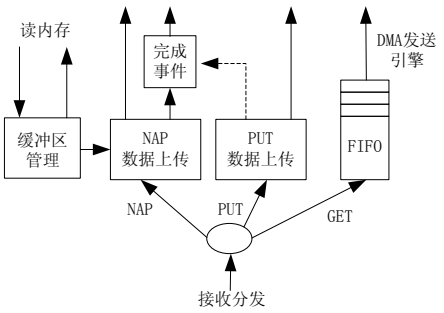

图 12 DMA 接收引擎结构框图

接收分发模块负责从 intra DMA 交叉开关中读取网络包，并根据网络包类型交给相应的模块进行数据处理。对于 PUT 包和 NAP 包，分发模块将其交给上传模块中的 PUT 包处理模块和 NAP 包处理模块进行处理，而对于 GET 包，则从中提取出描述符并写往本地发送引擎的描述符提取模块进行处理。

PUT 数据上传模块负责处理 PUT 网络包，从缓冲区中读取 PUT 包将其拆分成多个 PCIe 内存写包上传。在遵循 PCIe 协议规定及地址对齐要求等前提下，用尽可能少的逻辑资源实现高效上传。PUT 包和 NAP 包共用接收完成事件环，若需要上传接收完成事件，则需向缓存管理模块申请接收完成事件环地址；如果不要求，则可以直接将 PUT 包转换为 PCIe 内存写包上传。

NAP 数据上传模块负责处理 NAP 网络包，并将其转换为 PCIe 标准写包发往缓存空间。NAP 包上传过程与 PUT 包上传过程相似，只是 NAP 包中无地址信息，因此需要从缓冲区管理模块中读取相应的缓存地址。NAP 包必须上传接收完成事件，因此必须先申请得到接收完成事件地址才可以开始上传；PUT 包目的地址可以是任意的，NAP 包目的地址必须是 2K 字节对齐的。

缓冲区管理模块负责仲裁所有 NAP 数据接收缓冲区地址的读请求，采用请求/应答机制。系统初始化期间分配一定数目的缓冲区地址进行缓存，当接收缓冲区不足时，向缓冲区管理模块发出读地址请求，从内存中读取新的 NAP 接收缓冲区地址并更新 RAM 中缓存的 NAP 地址。缓冲区管理模块还负责维护接收完成事件环的流控。

完成事件缓冲区：NAP 与 PUT 共享一个接收完成事件缓冲区，缓冲区首地址在初始化时由主机告知网络接口控制器，缓冲区逻辑上呈环形结构，可循环使用。完成事件队列是一个有限项数的环形缓冲区，每上传一个 NAP 消息，需要向接收完成事件

缓冲区写入一个接收完成事件；每上传一个 PUT 包则根据标志位决定是否向接收事件缓冲区写入接收完成事，通过流控计数器防止接收事件队列的溢出。

4.3 I/O共享实现

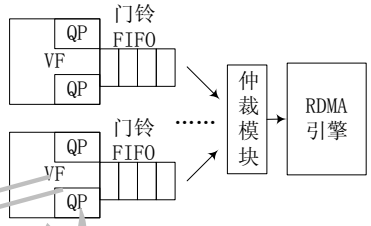

图 13 DMA 仲裁结构

DMA 引擎支持 SR-IOV 功能： DMA 引擎将虚拟出 1 个物理功能（PF）和 7 个虚功能（VF）与相应的配置空间和用于通信所需的资源“Queue Pair”（QP）建立映射关系（每个功能对应 4 个 QP），使每个虚功能可以分配给不同的虚拟机使用，实现 I/O 设备的高效共享，如图 13 所示。同时通过 QP 和虚功能配置空间使处理器获取若干“虚拟 DMA 引擎”，这些虚拟 DMA 可供不同虚拟机使用，并实现安全隔离功能，也即实现了“DMA 虚拟化”功能，处理器可以通过DMA引擎对全局统一编址的I/O资源进行直接访问，实现 I/O 设备的高效共享。

5 原型系统和性能评测

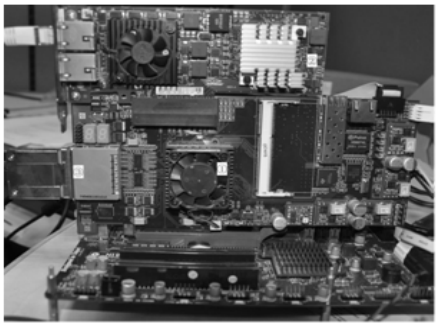

图 14 FPGA 原型系统

实验基于 Xilinx Virtex6 X365T 实现了原型系统，如图 14 所示，中间风扇下即为网络接口控制器的 FPGA 原型芯片，顶部插卡为具备 SR-IOV 功能的 Intel 82599 以太网卡，紧邻 FPGA 的 PCIe 线缆连接到另一个主机，作为处理器节点与控制器连接。

为了平衡网络接口控制器的逻辑规模和工作频率，根据 FPGA 的结构特点，网络接口控制器的内部总线设为 128 位，工作频率 250MHz。为与内部总线带宽相匹配，原型系统的所有 PCIe 接口均选择 PCIe2.0 ×8（峰值 5 GB/s，使用 8b/10b 编码机制，即有效传输带宽为 4 GB/s）。Xilinx Virtex6 Lx365t 芯片共包含 56880 个 Slice，416 块 Block RAM，网络接口控制器实际设计消耗资源 Slice 消耗量为总量的 8%，Block RAM 为总量的 6%。

5.1 峰值带宽测试

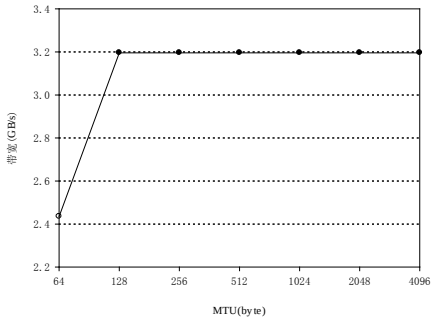

图 15 带宽随 MTU 的变化情况

在 PCIe 协议中，单个 PCIe 包所能携带的数据量，即最大传输单元 MTU（Maximum Transfer Unit），是受设备限制的。对于不同的 MTU，PCIe 包头所占据的开销比例会受影响。在本实验中，将单次内存读取所能请求的数据量大小与 MTU 设为相同，即通过改变 MTU 的值，研究 PCIe 协议对于 DMA 引擎峰值带宽的影响。

测试在请求 2M Bytes 数据情况下，MTU 变化对带宽利用率变化情况，采用带宽效率最高的 PUT 包测试峰值性能。从图 15 中可以看出，在 MTU 小于 128 Bytes 时，DMA 引擎的峰值带宽随着 MTU 呈线性增加，而在 MTU 为 128 Bytes 时达到顶峰，接近 3.2 GB/s 的实际带宽性能上限：16 Bytes PCIe 包头占 1 个周期，返回最大传输数据 128 Bytes 占 8 个周期，DMA 引擎处理需要 1 个周期，数据返回的效率为 80%。因此 PCIe 协议的限制传输带宽上限为 4 GB/s×80%，即 3.2 GB/s。当 MTU 大于 128 Bytes 之后，峰值带宽却不再增加，这是由于 PCIe 协议是通过包头 Tag 域来分辨返回包所对应的是哪个内存读请求所发起的，因此，在发送前要申请到 Tag 号才能上传。PCIe 协议默认 Tag 域的高 3 位是保留的，只有低 5 位有效，即可用的 Tag 数目为 32 个，不能随着 MTU 的增大而相应增加，进而导致带宽难以达到设计的理论性能。

DMA 发送引擎的描述符读取模块，数据读取模块和接收引擎中用于读取缓存区地址的模块需要申请 Tag 号进行内存读取操作。因此在设计中，需要对 Tag 进行统一的分配和调度等方式以免 Tag 被重用。对于读取内存的模块，满足持续提交 PCIe 内存

读请求所需的最少 Tag 数目如公式（2）所示。

$$n = \min\left(\left\lceil t/\left(\frac{Size_{payload}}{16} + 2\times\left\lceil\frac{Size_{payload}}{MTU}\right\rceil\right)\right\rceil, 32\right) \quad (0 \le n \le 32) \qquad (2)$$

其中 t 为从内存读取模块得到 Tag 号并生成 PCIe 读内存包开始，到响应数据包完全返回所需要的时间，单位为时钟周期，此后，相应的 Tag 号被释放，回收后可循环使用。$Size_{payload}$ 则为单个 PCIe 包所能请求的最大数据量（Byte 为单位），$Size_{payload}$ /16 是所请求的数据返回所需要的时钟周期数（位宽 16 Bytes），$Size_{payload}$ / MTU 是指所请求的数据被封装为 PCIe 包的个数，而每个数据包都需要一个时钟周期封装包头和一个周期的处理时间，因而要乘以系数 2。公式表明在这段时间内 n 个 Tag 足够使用，可以连续发送数据请求而不必等待 Tag 返回。当 $Size_{payload}$ 增加时，维持最高性能所需的 Tag 数目也会随之减少，但是在实际的系统中，由于内存返回数据是与 $Size_{payload}$ 成比例的，且存在竞争等不确定性因素，t 可能会需要成百上千个周期，仅 32 个 Tag 很难满足持续高效的数据读取。为了增加读取效率，可以通过两种方法提高性能：1）将 PCIe 设备中的 PCI Express Capability Structure 中 Device Control 寄存器的 Extended Tag Field Enable 位置位，使 Tag 数目扩大为 256 个，但是该功能需要 PCIe 核的支持；2）增大 $Size_{payload}$，需要设备本身及其 PCIe 桥支持大的数据请求量。

5.2 带宽随负载变化情况

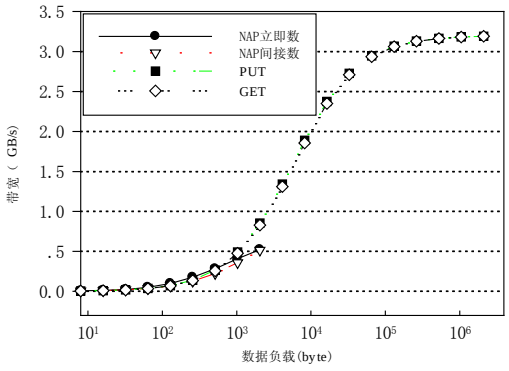

图 16 带宽随负载变化情况

由于实际系统 MTU 为 128 Bytes，Tag 数目为 32 个，因此，在该条件下测试了带宽利用率随负载变化的情况。如图 16 所示，当数据量较小时，PUT 包的带宽随负载的增加而显著增加，这是因为包头所占比例迅速减小所致。但当负载增加到一定的程度之后，带宽增加变缓，此时负载所占比例的增加已经不够明显。当负载的数据量增加到 512K Bytes 时，实际速度已达 3.19 GB/s。GET 包类似 PUT，但数值略小，是因为接收方会将 GET 描述符提取给发送 DMA 引擎，转化为本地 PUT 操作，延迟会增加。

由于 NAP 包长最大为 2K Bytes，因此只分析到传输数据最大 2K Bytes。如图 16 所示，NAP 包的带宽均随负载而增加。在负载很小时 NAP 立即数包的带宽最大，这是因为 NAP 立即数将待传输的源数据封装在描述符中，可直接提取发送，相比其他操作减少一次内存读取。而随着负载的增加，两种 NAP 操作的差距开始减小，这是因为描述符为 64 位，每两个周期才能提取一个周期的 NAP 立即数，而对于 NAP 间接数，内存直接返回 128 位的 PCIe 包，二者开销近似，因而带宽效率开始接近。

5.3 延迟随负载的变化情况

表 3 DMA 延迟（μs）

Payload（Byte）	NAP 立即数	NAP 间接数	PUT	GET
1	1.242	1.846	1.762	1.838
64	1.254	1.862	1.778	1.854
256	1.458	2.018	1.838	1.914
1024	2.514	2.858	2.078	2.154
2048	3.922	3.978	2.398	2.474
4096			3.054	3.13
16384			6.894	6.97
65536			22.254	22.33
262144			83.694	83.77
1048576			329.454	329.53

表 3 描述了各种网络包在不同负载情况下的延迟情况。考虑到描述符大小对于延迟的影响，将源数据以 2M Bytes 为单位传输，即使用最小的描述符。从表 3 可知，对于 PUT 包，当数据量最小 1 字节时，延迟最小，达到 1.762μs，而延迟随着数据量的增加而线性增加。对于 GET 请求，最小延迟为 1.838μs，是因为接收方会将 GET 请求转化为本地的 PUT 操作，因而略延迟大于 PUT 包。NAP 立即数包的延迟最小仅 1.242μs, 这是因为其源数据直接包含在描述符中，相对其他类型减少了一次内存读取，因此延迟最小。而 NAP 间接数最小延迟为 1.846μs。延迟略大于 PUT 包，这是因为 NAP 包没有指定目标地址，在接收端需要申请缓存地址，也需要一次内存读取，因此延迟会大于 PUT 包。

图 17 显示了 NAP 立即数和 PUT 两种网络包在发送 1 字节数据负载条件下的延迟情况，时间从 DMA 接收引擎接收到主机发起的门铃开始，到 DMA 发送引擎将网络包转换为 PCIe 数据包上传数据为止，其数据流程与第四节所描述的一致，其中 Crossbar 延迟是节点内的交换延迟。对于 NAP 数据包需要请求缓存地址，而虚线框表示 DMA 会预先读取并保存一定数量的缓存地址，以避免频繁发起地址请求。其中描述符和数据读取和返回的延迟受内存控制器的响应速度和资源竞争的影响是动态变化的，图 17 中的“130”是经验值。若不考虑内存数据读取的延迟，DMA 引擎的处理延迟仅 35（NAP）到 48（PUT）时钟周期。尤其在 Crossbar 后的 DMA 上传阶段，NAP 操作（当缓存地址足够时）和 PUT 操作均将数据直接打包为标准 PCIe 数据包上传，简化了繁琐的协议转换。

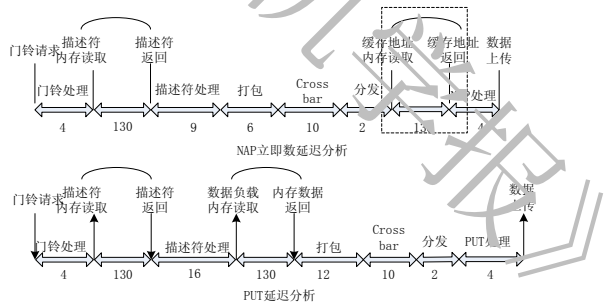

图 17 传输延迟分析

5.4 吞吐率分析

在不同端口相互通信的过程中，随着通信量的增加，连接通信接口的交叉开关会变得拥堵，进而导致 DMA 传输请求响应时间变长，即延迟增加。增加虚通道数量可有效缓解“队头阻塞”的影响，因此，实验测试了不同端口数配置不同虚通道条件下的通信吞吐率情况。为避免请求和响应导致的死锁规定 GET 数据包需要走专用虚通道。为提高吞吐率，本节采用传输效率高的 PUT 数据包进行测试，因此，这里不考虑 GET 专用虚通道。定义虚通道的使用策略 Dest-Mod：如果虚通道数量为 n，而消息的目的端口是 d,则消息将会被缓存在虚通道[d mod n]内。图 18 为 4 端口 2VC（Virtual Channel：虚通道）（每个端口 2VC，下同）、4 端口 4VC、8 端口 2VC、8 端口 4VC 和 16 端口 4VC 条件下采用所设计的基于 PCIe 标准的通信协议进行互连通信时，延迟和带宽的关系。延迟从写门铃开始计算，到接收端全部接收并处理完网络包结束，最后对所有请求的延迟进行平均。吞吐率定义为传输稳定之后，对每个端口的带宽求平均并除以理论峰值。

从图 18 中可以看出，在端口数目一定的情况下，吞吐率随着虚通道数目增加而增大；在虚通道数目一定的情况下，吞吐率则随着端口数目的增加而减小。4 端口时，在每端口 2VC 的情况下，当吞吐率接近最大理论带宽的 65%时，延迟开始迅速上升，此时通信系统的吞吐率为 2.6 GB/s。在每端口 4VC 的情况下，系统的吞吐率达到 2.8 GB/s。带宽较 2VC 有所提高是由于在 4 端口 4VC 的情况下构成 VOQ 结构[9]，可消除队头阻塞问题。没有达到实际性能上限，是因为仿真环境的随机性不足，没有达到完全均匀随机分布。考虑到 4VC 相对于 2VC 仅提高了 5%的性能，综合资源消耗和性能的考虑，4 端口下 2VC 的设计在性能和资源上都能达到较好的平衡。

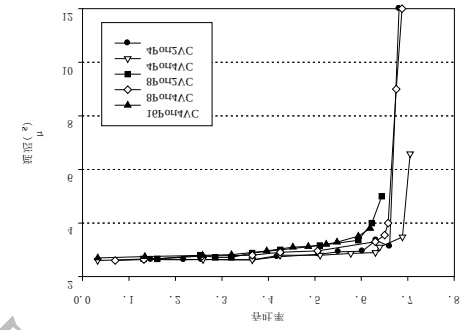

图 18 延迟和吞吐率关系

5.5 I/O虚拟化性能分析

在设计中，虚拟机通过 QP 可以同时使用多个网卡，为实现合理高效的资源共享，使每个虚功能都能公平的使用系统资源，采取了公平的仲裁策略。针对各个虚功能的门铃请求，实现了公平的调度策略。图 19 为各个 VF 分配的带宽随时间的变化情况。测试中使用随机大小的 PUT 包，并随机的将请求分配给各个功能。测试结果如图 19 所示，不同颜色曲线描述的是 VF0-VF6 和 PF 所分配到的带宽情况。纵轴曲线之间的间隔表示不同功能所占带宽的百分比，横轴为仿真时间，单位为时钟周期。从图中可以看出，传输刚开始时，带宽的分配变化很剧烈，这是因为 DMA 引擎刚开始工作时，已传输的数据量较小，因此单次请求传输的数据占总数据量的比例比较大，因而对带宽的分配有着很大的影响，但很快各个 VF 之间带宽分配的比例就基本达到均衡。大概在 150 万个时钟周期带宽分配达到稳定状态，很好的达到了公平分配带宽的目的。

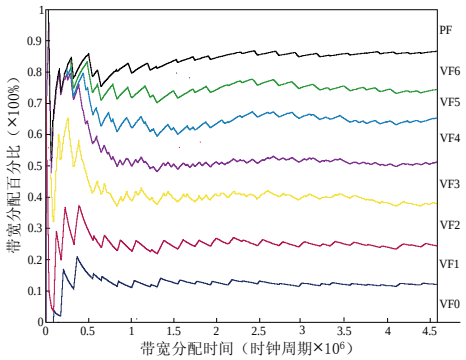

图 19 带宽分配随时间的变化

6 相关研究

表 4 带宽和延迟

Networks	Bandwidth (GB/s)	Latency (μs)
InfiniBand QDR	3.2	1<
Cray Gemini	6.1	1.0
Cray Aries	9.5	1.2
BlueGene/Q	2	2.0
PEARL	1.1	1.2
Dolphin Express	1.27	14
cHPP	3.19	1.242

节点控制器是高性能互连网络的核心组件，是提供高性能通信的重要保障。全局地址统一编址、提供高效通信原语、用户级通信等多种关键技术被许多高性能计算机的大规模互连网络所采用，例如：IBM BlueGene 系列[10][5][6]在处理器内集成了高性能网络路由器，其网络接口支持直接 PUT 和远程 GET；Cray 的 Gemini [11]和 Aries[12]系列互连网络的网络接口也在硬件层次提供了支持小消息传输的快速消息访问（FMA）和长消息传输的块传输引擎（BTE）；商业网络如 Infiniband[4]，其主机通道适配器（HCA）提供硬件支持的 RDMA PUT/GET 操作、采用 IPV6 兼容的 128 位全局地址以支持远程直接内存访问。由于这些网络接口控制器的设计均面向大规模互连网络，因此一次端对端通信要涉及 I/O 总线协议-网络协议-I/O 总线协议间的转换流程（解析→打包→解析→拆包）,其消息格式定义也需要涵盖大规模数据传输所需的子网管理、路由、拥塞控制等信息（例如 Infiniband 的包头最大可达到 94 字节），限制了网络处理的效率和有效负载带宽的提升。然而在规模限定的局部互连中，上述开销均为无效开销，本文正是通过面向局部通信的互连协议和 DMA 引擎结构设计，实现对上述开销的优化。

此外，在局部互连网络中，存在着网络互连和 I/O 设备扩展两种功能需求，本文通过扩展 PCIe 协议实现的互连网络，实现了在物理层上两种功能的融合，这是面向大规模互连网络的网络接口控制器所不具备的。正是上述功能融合的需求，国际上已有利用 PCIe 实现处理器间互连的工作，例如日本瑞萨公司的 PERAL[13]使用 PCIe 作为通信链路设计了功耗感知的，高可靠和高性能的互连芯片，最低传输延迟 1.2μs，但带宽仅为 1.1GB/s（理论峰值的 55%）。Dolphin 公司使用增强的 PCIe 互连方案[14]来实现多主机间通信和主机到 IO 通信的功能，由于同时面向大规模机群系统，因此它并没有进行针对局部互连的优化，其传输延迟为 14μs，带宽为 1.27GB/s。

表 4 比较了cHPP网络接口控制器与上述经典互连网络 [2] 以及采用PCIe标准的商业网络[13][14]的性能。从表 4 可以看出，cHPP网络接口控制器的FPGA原型系统在带宽的绝对性能方面（绝对带宽与实现工艺相关）低于Cray的高性能互连网络，但超过同类型的PCIe互连网络（PERAL和Dolphin Express），同时可以获得与峰值带宽相同的InfiniBand（QDR）相近的性能。上述网络接口控制器在延迟方面的性能相似，根据图 17 的延迟分析可知，若提高cHPP控制器的工作频率，其硬件延迟还可进一步降低。

7 结论和展望

cHPP 体系结构采用超节点设计可支持多种网络拓扑，超节点结构可减少通信路径，缩减网络规模，并且充分利用了超节点内部通信的局部性，可有效降低通信延迟。超节点控制器采用 PCIe 高速接口提供高带宽和高可靠的链路性能。网络接口控制器支持用户级通信和高效通信原语加速大数据量传输，降低处理器的占用率，进一步提高计算能力。基于硬件支持超节点内 I/O 资源的高效共享在极小性能损失的条件下可满足大量并发的通信请求。

下一步工作主要是在节点内通信的基础上进行系统的级联扩展，支持高维度网络拓扑的大规模直接网络，实现全系统节点间处理器的高速通信；针对 MTU 和 Tag 等限制因素进一步优化 DMA 引擎结构，改善通信性能；拓展 I/O 虚拟化性能，实现不同节点间的处理器对全局 I/O 资源的高效共享；并根据不同层次的通信需求，通过调整 QP 资源在虚

2 network bandwidths and latencies for some networks, http://www.euroben.nl/reports.php

拟机和VF之间的动态分配来实现丰富的QoS服务。

致　谢　感谢李强博士在文章撰写过程中关于通信原语和上层软件库之间关系的深入有益的讨论。

参 考 文 献

[1] Sun Ninghui, Li Kai, Chen Mingyu. HPP: An Architecture for High Performance and Utility Computing. Chinese Journal of Computers, 2008, 31(9)：1503-1508 (in Chinese)
(孙凝晖, 李凯, 陈明宇. HPP：一种支持高性能和效用计算的体系结构. 计算机学报, 2008, 31(9)：1503-1508)

[2] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation//Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, USA, 1993: 1-12

[3] A. Alexandrov, M. Jonescu, K. E. Schauser and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model—one step closer towards a realistic model for parallel computation//Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures, Santa Barbara, USA, 1995: 95-105

[4] M. J. Koop, Wei Huang, K. Gopalakrishnan, D. K. Panda. Performance Analysis and Evaluation of PCIe 2.0 and Quad-Data Rate InfiniBand//Proceedings of High Performance Interconnects Symposium, Stanford, USA, 2008: 85-92

[5] Dong Chen, Noel A. Eisley, Philip Heidelberger, et al., The IBM Blue Gene/Q Interconnection Network and Message Unit//Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, USA, 2011: 1-10

[6] R. A. Haring, M. Ohmacht, T. W. Fox, et al., The IBM Blue Gene/Q Compute Chip. IEEE Computer Society, 2012, 32(2):48-60

[7] Ajima, Y., Sumimoto, S., & Shimizu, Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers. Computer, 2009, 42 (11): 36-40

[8] R. A. F. Bhoedjang, T. Ruhl, H. E. Bal. User-level network interface protocols. Computer, 1998, 31(11):53-60

[9] M. Mehmet Ali and H. Tri Nguyen. A Neural Network Implementation of an Input Access Scheme in a High-Speed Packet Switch//Proceedings of the Global Telecommunications Conference and Exhibition 'Communications Technology for the 1990s and Beyond'. Dallas, USA, 1989: 1192 -1196

[10] The Blue Gene/P Team. An Overview of the BlueGene/P Project. IBM Journal of Research and Development, January, 2008, 52(no. 1/2):199 -220

[11] Bob Alverson, Duncan Roweth and Larry Kaplan, Cray Inc. The Gemini System Interconnect//Proceedings of High-Performance Interconnects Symposium, Mountain View, USA, 2010:83-87

[12] Faanes, Greg et al,. Cray Cascade: A Scalable HPC System Based on a Dragonfly Network//Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, USA, 2012:1-9

[13] Toshihiro Hanawa, et al. PEARL and PEACH: A Novel PCI Express Direct Link and Its Implementation//Proceedings of IEEE International Parallel & Distributed Processing Symposium Anchorage, Alaska, USA, 2011: 871-879

[14] V. Krishnan. Towards an Integrated IO and Clustering Solution Using PCI Express//Proceedings of the IEEE International Conference on Cluster Computing, Austin, USA, 2007: 259-266

SU Yong, born in 1976, Ph.D. candidate, engineer, E-mail: sy.pass@163.com, His main research interests include computer architecture and high performance interconnection networks.

Cao Zheng, born in 1982. Ph.D., Associate professor, E-mail: cz@ncic.ac.cn, His research interests include high performance computer architecture, high performance interconnection, and optical interconnection.

Liu FeiLong, born in 1989, Master, E-mail: liufeilong@ncic.ac.cn, His main research interests include computer architecture and high performance interconnection.

Wang Zhan, born in 1986, Ph.D. candidate, E-mail: wangzhan@ncic.ac.cn, His main research interests include virtualization technology and high performance interconnection networks.

Liu Xiaoli, born in 1986. Master, engineer, E-mail: liuxiaoli@ncic.ac.cn, Her major interests focus on I/O Virtualization and high performance interconnection networks .

An Xuejun, born in 1966, Ph.D., Professor level senior engineer. E-mail: axj@ncic.ac.cn, His research interests include computer architecture, high performance interconnection.

Sun Ninghui, born in 1968, Ph.D., Professor, Ph.D. supervisor. E-mail: snh@ict.ac.cn, His main research interests include computer architecture, high performance computing and distributed OS.

Background

High performance interconnect network is one of the key technologies of high performance computing (HPC), which is monopolized by commercial InfiniBand fabric, high speed Ethernet and custom interconnection. InfiniBand fabric and high speed Ethernet are usually used to structure indirect network of cluster system, while custom interconnection, used in MPP system mainly, dominates in the high end system. Both of the cluster and MPP system have to use PCIe bus to connect the IO resource, which needs protocol conversion and increases communicate latency, power and cost. So, if PCIe could be used to interconnect the whole computing and IO resource, the overhead for protocol conversion could be eliminated, which improves the performance of interconnect network greatly.

Recently, HPC interconnecting network based on PCIe has attracted a lot of interests of both industry and academe. Dolphin provides a clustering solution using an enhanced PCIe interconnect thus obviating the need for a dedicated clustering interconnects, while the extra cluster-I/O switch increases the system latency. PLX has built a PCIe fabric for data center and cloud computing through its ExpressFabric, which is targeted at small to medium-sized cloud clusters. IDT use PCI Express as the Primary System Interconnect in Multi-root Compute, Storage, Communications and Embedded Systems, while the system scale is limited.Renesas's Hanawa used PCIe as a communication link that provides power-awareness, high reliability, and high performance. Results show a latency of 1 μs and maximum bandwidth of 1.1 GB/s, 55% of the theoretical peak performance. While our experiment results show that the maximum bandwidth is 3.1 GB/s, 80% of the theoretical peak bandwidth and the lowest latency is 1.242μs only. Meanwhile, we implement the efficient IO share in a node. Other famous networks are compared together, such as BlueGene Q/P, Cray Gemini/Aries and InfiniBand, the results show each has its own merits.

This project is supported by National Natural Science Foundation of China (No. 61100014), which focuses on HPC interconnect networks issues and I/O virtualization. Our project team had researched deeply in the multiplayer interconnect network during the development of Dawn 5000 and Dawn 6000 high performance interconnect network. Now we focus on the next HPP system controller：cHPP, which proved high speed interconnect between the Heterogeneous CPU based on PCIe. Moreover, cHPP controller fully supports SR-IOV, with the virtual port to implement the efficient share of cHPP controller by virtual machine or process. This research, which is the core component of cHPP controller, implements the key technique of DMA engine and PCIe high speed port and IO efficient native sharing.