一种高效MPI设备层扩展库的设计与实现

方 铃,雷咏梅
(上海大学 计算机工程与科学学院,上海 200072)

摘 要:PCI Express作为新一代的总线接口,能支持各个设备之间并发的数据传输。每个设备在要求传输数据时各自建立专用的传输通道,这样的操作使得数据能够高效传输,因此PCI Express互联结构已被广泛地应用到高性能领域。用PCI Express总线代替以太网,不仅能实现不同设备之间高速的数据传输,而且能够缩减结构规模,并很好地应用于工业等领域。但基于这种互联结构来实现并行程序运行,需要设计和实现相应的MPI设备层扩展库,以实现不同板卡系统的进程间通信。结合PCI Express互联结构特点,基于MPICH的PMI KVS空间的功能,提出控制和数据双通道的通信模式。控制信息依靠可靠传输协议TCP进行传输,数据信息通过高速的PCI Express总线进行传输,以实现异构系统之间高效的数据传输,并通过模拟实验证明了设备层扩展库的可行性。

关键词:高性能;PCI Express互联结构;MPI设备层扩展库;双通道
中图分类号:TP302.1 **文献标识码**:A **文章编号**:1673-629X(2017)02-0006-05
doi:10.3969/j.issn.1673-629X.2017.02.002

Design and Implementation of an Efficient MPI Device Layer Extension Libraries

FANG Ling, LEI Yong-mei
(School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

Abstract: As a new generation of bus interface, PCI Express can support the concurrent data transmission between all kinds of devices, and each of these devices can establish their own transmission channel what is proprietary when it requires transmitting data, which makes the transmission of data more efficient. So PCI Express interconnect structure has also been applied to high performance areas. When Ethernet replaced by PCI Express, high speed transfer between different devices can be achieved, and the scale of the structure is effectively reduced, which can be well applied to the industry and other fields. But in order to run parallel programs based on this interconnect structure, it is necessary to design and implement the corresponding MPI device layer extension libraries, achieving that the process communication of different boards. The communication mode of control and data dual channel is presented taking advantage of the features of PCI Express interconnect structure and based on the function of MPICH PMI KVS space. Control information depends on reliable TCP to transmit and data information transmits via a high-speed PCI Express bus, so that efficient data transmission between heterogeneous systems can be implemented. The feasibility of device layer extension libraries is proved through the simulation experiment.

Key words: high performance; PCI Express interconnect structure; MPI device layer extension libraries; dual channel

1 概 述

近年来,随着高效能技术的发展,各种高性能总线互联技术层出不穷,包括嵌入式领域的RapidIO技术[1]、针对PC机的PCI Express技术[2]以及面向服务机群的InfiniBand技术[3]。其中,PCI Express作为新一代的总线接口,能够通过交换器实现多台设备之间通信的串行、点对点类型的互联,而PCI Express的高速数据传输使其广泛地应用于高性能计算等领域[4-6]。同时还有基于FPGA的PCI Express接口的设计[7-8]。目前已提出的PCI Express总线互联结构中,通过母板PCI Express的插槽连接不同的带有FPGA和ARM混合异构的嵌入式子板,这些子板上运行MPI[9]应用进程,而不同子板的应用进程之间通过PCI Express把数据传输给PC host,然后再由PC host把数据传输给目的板卡上的应用进程。用PCI Express代替以太网来传输数据,能很好地解决网络带宽限制的问题,而且该

收稿日期:2016-04-13 **修回日期**:2016-08-02 **网络出版时间**:2017-01-10
基金项目:国家"863"高技术发展计划项目(2009AA012201);上海市科研计划重大项目(08dz501600)
作者简介:方 铃(1989-),男,硕士研究生,研究方向为高性能计算;雷咏梅,博士,教授,研究方向为高性能计算。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20170110.1028.072.html

结构规模较小,能很好应用在工业等领域。然而基于这种互联结构实现并行程序运行,需要设计和实现相应的 MPI 设备层扩展库,实现不同板卡系统的进程间通信,软件架构如图 1 所示。

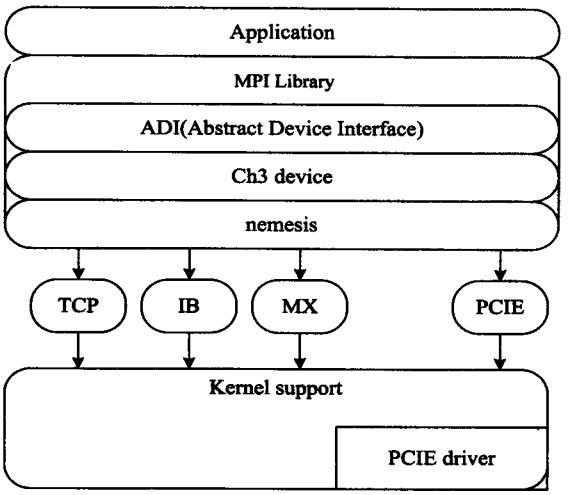


图 1　系统软件架构

通过对 MPI 各个层次结构的分析,文中提出一种构建 MPI 设备层扩展库的实现方法。利用 MPICH3 的 PMI KVS 空间的功能[10],每个进程通过 PMI put 操作保存本地节点端口等信息到本地 KVS 中,同时同步到远程服务端 KVS,而其他节点通过 PMI get 操作获取各节点所在端口等信息,从而建立通信关系。同时结合 PCI Express 互联结构的特点,提出控制和数据双通道的通信模式。一方面通过以太网传输必需的控制信息,另一方面使用高速的 PCI Express 传输数据信息,以实现这种嵌入式混合异构系统运算节点之间高效的数据传输。

2　MPICH 结构的分析

MPICH[11]作为 MPI 实现之一,也是目前主流的并行函数库,主要包括两个工作:实现进程管理;实现 MPI 库接口。第一部分主要通过 Hydra[12]的作业加载程序在每个节点创建一个进程管理代理 PMP,并通过 PMP 在本地节点创建和管理应用进程。第二部分是通过分层、模块化等方法实现,层次结构图如图 2 所示。

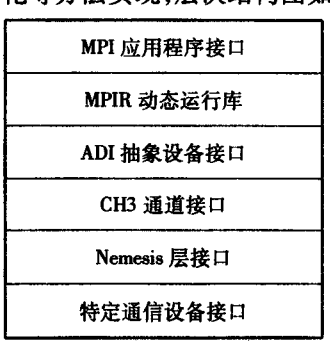


图 2　MPICH3 层次结构图

从分层结构中可看出,要实现设备层扩展库,需有 MPI 最后一层——特定通信设备接口层的相关实现。

2.1　数据结构

MPICH3 工程设计中体现出了面向对象思想。MPICH3 中有许多重要的数据结构,比如:ADI 层的进程(MPID_PerProcess)、虚连接(MPIDI_VC)结构和收发请求(MPID_Request)结构等。

设备层有两个重要的结构体:

(1)struct MPID_nem_netmod_funcs,该结构中定义了需要实现的初始化、发送、接收等回调函数,在 CH3 层初始化时,需要将实现好的 MPID_nem_netmod_funcs_t 指针传递到上层。而对于设备层已有实现的网络设备,用户要通过填写 Makefile 参数去选择需要的设备层。

(2)struct MPIDI_Comm_ops,包含主要的通信函数,ADI 层对函数指针进行判断,而且程序优先调用 struct MPIDI_Comm_ops 内函数,主要函数有发送函数 send、接收函数 recv_posted、标准通信模式 isend、同步通信模式 irsend 等。

由于双通道设计实现需要有初始化的过程,所以文中选择只实现带有初始化函数的结构体 MPID_nem_netmod_funcs 中的函数。

2.2　虚连接与请求管理

虚连接是 MPICH3 ADI 层的一个重要结构体,MPI 的每一个应用进程都会维护一张自己的虚连接表,在 ADI 层初始化的时候,会为虚连接表分配空间,也就是初始化虚连接的每一个表项,主要是根据进程所在节点,为各个进程之间建立通信方式,并且虚连接表是以数组方式进行存储的。例如,现在 0 号进程中要初始化 VC 数组的第二个下标对应的 VC,也就是 VC[1],这个 VC[1]就是 0 号进程和 1 号进程间的虚连接。

虚连接内部包含了一个重要的结构体 struct MPIDI_CH3I_VC,该结构体包含两个重要的函数,同时还包括虚连接自身的一个函数,分别是 iStartContigMsg、iSendContig、sendNoncontig_fn。它们都和发送相关。当应用程序调用 MPI_Send 时,MPICH 根据 rank 号取到对应的 VC,按照参数选择调用以上三个 VC 中的函数。这三个函数需要在设备层扩展库进行具体实现,当虚连接初始化时根据进程是节点间还是节点内选择是否对 VC 的函数指针进行赋值。

由于 MPI 存在非阻塞通信,而且在实际数据发送过程中可能出现信道阻塞,所以需要有缓存。MPICH3 设备层缓存和 CH3 层对接收请求的组织方式相同,都是以请求队列[13]的形式存在的。当出现阻塞时,将消息封装成请求并入队,等待出队发送,所以设备层需要在虚连接内维护一个请求队列。对一个进程来说,对

应所有其他进程都有一个虚连接就有一个请求队列，设备层扩展库的发送接口实现时需要利用请求队列作为缓存。

3 MPI设备层扩展库的设计

设备层扩展库主要包括虚连接初始化和通过MPICH PMI KVS方法实现对控制数据传输双通道的支持以及系统功能设计。

3.1 虚连接初始化和自定义结构体

虚连接初始化接口，主要包括两个步骤：第一是在作业初始化阶段为进程初始化与其他进程之间调用过程；第二是初始化设备层相关数据结构并保存到虚连接内。结构体MPIDI_CH3I_VC中有一个很重要的成员，该成员变量描述如下：

```
union
{
char padding[ MPID_NEM_VC_NETMOD_AREA_LEN];
void * align_helper;
} netmod_area;
```

netmod_area中的char padding[ ]用来保存设备层重要的数据结构到虚连接，文中需要保存的自定义结构体如下：

```
typedef struct{
fileopt_t * fo; //设备文件描述符
reqq_t send_queue;  //发送队列
} MPID_nem_file_vc_area;
typedef struct{
int fd;  //每一个进程获得的文件描述符
MPIDI_VC_t * vc;  //本进程和目的进程的虚连接
int endpoint;  //端口号
} fileopt_t;
```

通过这两个结构体可以保存需要操作的设备文件描述符和发送队列以及端口号等。

VC初始化的具体流程如下：

(1)初始化iStartContigMsg、iSendContig、sendNoncontig_fn三个函数指针。

(2)初始化发送队列send_queue。

(3)将文件描述符保存到对应的虚连接结构体。

(4)得到各个目的进程的business card，通过查询server端得到端口号保存在相应的虚连接结构体中。

3.2 控制和数据传输双通道模式

由于总线互联结构支撑软件提供的操作设备方法无法像传统TCP一样，以IP地址作为主机标识进行数据传输，而它一般以端口作为主机标识，如PCI Express，而在进程管理部分通过SSH等进行远程控制访问。另一方面，数据传输是通过PCI Express总线，所以整体数据的流通是以控制和数据相分离的，控制信息主要是进程管理相关的信息通过TCP进行传输，而数据信息主要是发送、接收等通过PCI Express总线传输。

但是发送数据时，发送进程需要知道接收进程所在运行板卡的端口，才能将数据准确地发送给接收进程。文中采用将端口号等信息和设定的key，如字符串“endpoint”，通过PMI_KVS_Put方法写入本地KVS中，再从整个树形结构中层层提交到远程的Launch node端KVS中，其他进程按照这个key值在对应的进程KVS中去查找需要的端口号等信息。因为每一块板卡有一个唯一的IP地址，也对应一个唯一的端口，本地初始化时可以通过kernel提供的方法获得端口号，然后通过上面的方法进行保存，在每个进程虚连接初始化时通过PMI_KVS_Get方法查询每个进程rank值对应的端口号等信息，并保存到虚连接中，流程如图3所示。

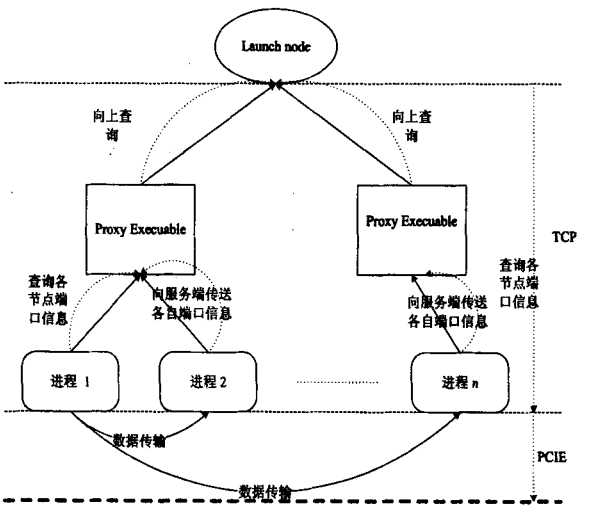


图3 控制与数据双通道结构图

3.3 系统功能实现

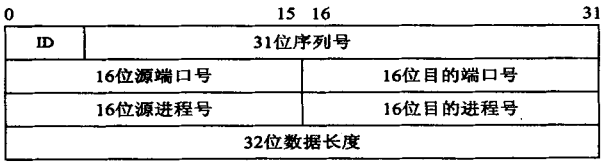
系统功能实现主要包括发送和接收功能。在上一节介绍的数据结构中已说明要实现struct MPID_nem_netmod_funcs中发送相关的函数，其中三个函数的不同点在于调用iStartContigMsg时，上层没有创建request对象，而调用iSendContig和sendNoncontig_fn时已经创建好request对象。因为MPICH3考虑到request对象管理频繁，造成资源的浪费，同时由于大量非阻塞通信的使用，需要对request对象进行出队入队耗时的操作，就区分了两种发送数据的方式。iStartContigMsg、iSendContig都是发送连续数据，sendNoncontig_fn是发送非连续数据。和接收相关的需要在设备层扩展库实现的是MPID_nem_poll函数。

当进程数据传输时发送进程得到目的进程的所在节点端口号，先将端口以及要发送的数据大小等信息发送给PC host，然后将需要发送的数据发给PC host，最后由PC host转发给相应端口的节点，所以这种互联

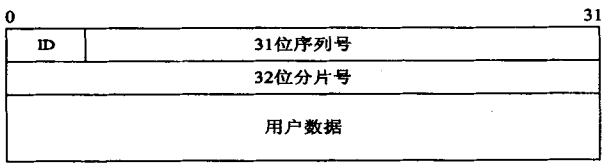
结构的数据发送是通过PC host去中转,并非点对点的数据传输。文中针对这种结构定义了相应的数据包格式以及数据的发送和接收方式。

3.3.1 数据包格式

发送方要向PC端发送的消息有两种:一种是需要预先向PC发送接收方的端口号、进程号等控制信息;另一种是向接收方发送的用户数据。所以通过消息头的ID字段将两种消息进行区分,同时定义两种数据包的格式,如图4所示。



(a)控制信息数据包格式



(b)用户数据包格式

图4　数据包格式

ID字段:ID字段只有一位,当ID为0时,表示控制信息数据包,当ID为1时表示用户数据包。

序列号:根据进程号为每个进程分配连续的且不同的序列号,保证同时一个端口不同进程向PC端发送数据时,可以以不同的序列号进行区分。

分片号:当数据以分片发送,分片号从1开始,以分片号标识每片数据,当数据不以分片发送,分片号字段为0。

3.3.2 数据的发送与接收

当发送方向接收方发送数据时,先为本次通信过程分配唯一的序列号,再填充源端口号和目的端口号,源进程号和目的进程号以及数据长度信息,将控制信息数据包发送给PC端;PC端接收到控制信息数据包,得到目的端号和序列号等信息,并等待发送方发送数据;发送方将数据发送给PC端;当PC端接收到所有数据时,将数据发送到目的端号。

由于一块板卡上运行多个应用进程,而每一个进程会从同一个端口读取数据,从而使得数据读取混乱,所以文中采用为板卡上每一个进程预设内存缓冲池的方式存放PC端发送来的数据。当设备层扩展库读取数据时,按进程号读取对应缓冲池上的数据,再拷贝到用户数据区。发送与接收数据过程如图5所示。

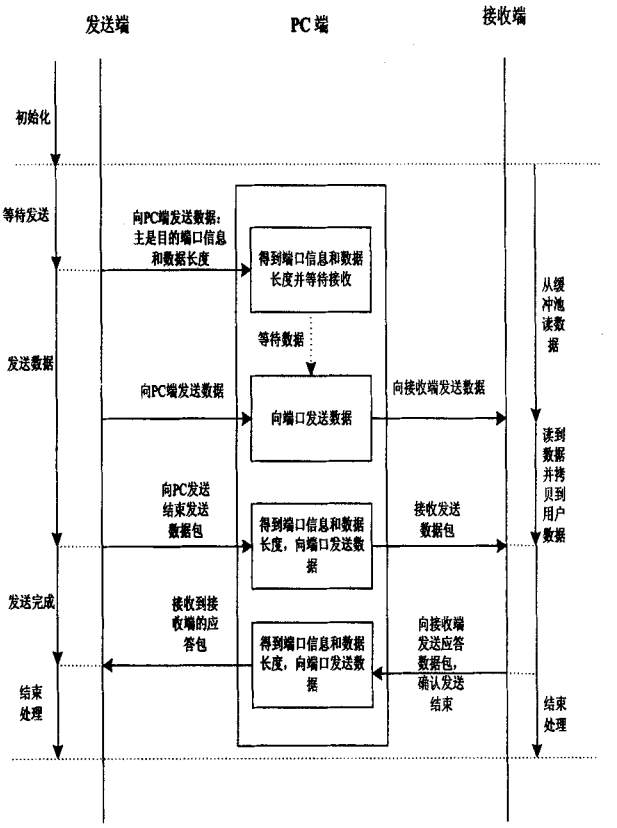


图5　发送与接收数据过程

实验环境:处理器为Intel(R) Core(TM) i3-2130 CPU @3.4 GHz;内存为4GB;操作系统为Ubuntu 14.04.1。对用梯形积分法求面积的应用进行测试,应用实例并行计算过程的通信关系如图6所示。

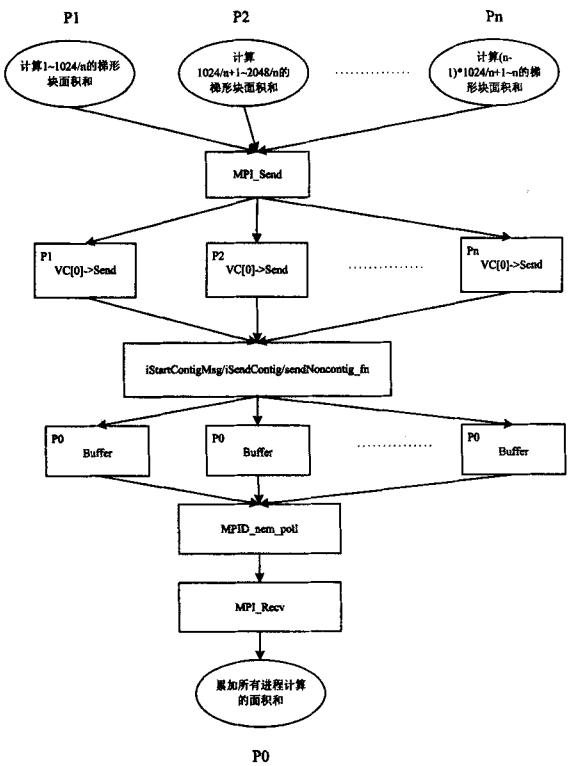


图6　进程通信关系和调用过程

4　实验结果及分析

实验对MPI设备层扩展库的通信模式进行验证。

采用普通文件的方式进行模拟实验，测试功能包括 MPI 应用程序接口中初始化、发送、接收功能的正确调用，以验证 MPI 设备层扩展库支持多进程并行程序的正确运行。

梯形积分法求面积思想是将整个面积区域划分成多个近似矩形区域，然后求解所有矩形面积之和。实验中计算 $y = x^2$ 与 $x = 0$，$x = 3$ 围成的区域面积。算法中预先将整个区域分成 1 024 小块，再将这些小块按照进程数均等划分。

选取 8、64、128 个进程分别进行测试，运行结果如下：

```
root@ node1 :/home/fangling/github# /usr/fl1/mpi/bin/mpirun -np 8 ./compute
use time:0.000 653,process size:8,our estimation=
9.000 004 291 534 424e+00
root @ node1 :/home/fangling/github # /usr/fl1/mpi/bin/mpirun -np 64 ./compute
use time:0.043 264,process size:64,our estimation=
9.000 004 291 534 424e+00
root @ node1 :/home/fangling/github # /usr/fl1/mpi/bin/mpirun -np 128 ./compute
use time:0.057 931,process size:128,our estimation=
9.000 004 291 534 424e+00
```

程序中以 0 号进程作为 master 进程，其他进程作为 slave 进程，每一个 slave 进程计算自己的矩形面积，然后把结果发给 master 进程。分别以 3 组进程数启动 MPI 并行程序，过程中控制命令和数据传输运行正确，并行程序运行结果证明数据传输正确，程序结果运行正确。

上述实验验证了文中所设计的数据通信模式、数据包格式及数据的发送和接收方式准确可行，所构建的 MPI 通信库，可以有效实现基于 PCI Express 的嵌入式混合异构系统之间的数据传输。

5 结束语

文中利用 PCI Express 结构的特点和 MPICH 的 PMI KVS 空间，提出数据和控制双通道的通信模式，并通过模拟实验验证了设备层扩展库接口的正确使用。下一步将开发和移植典型应用，并且进行完善的性能测试以及性能优化。

参考文献：

[1] RapidIO Trade Association. RapidIO technology overview and application[EB/OL]. 2008. http://www. rapidio. org/.

[2] Ravi Budruk R, Anderson D, Shan T. PCI Express 系统体系结构标准教材[M]. 北京：电子工业出版社，2005.

[3] InfiniBand Trade Association. An InfiniBand technology overview[EB/OL]. 2008. http://www. infinibandta. org/.

[4] 张伟达，黄芝平，唐贵林. 基于 PCI Express 的高速数据传输系统研究与开发[J]. 计算机测量与控制，2009，17(12)：2555-2557.

[5] 马 萍，唐卫华，李绪志. 基于 PCI Express 总线高速数采卡的设计与实现[J]. 微计算机信息，2008，24(25)：116-118.

[6] 王 伟，傅其祥. 基于 PCIe 总线的超高速信号采集卡的设计[J]. 电子设计工程，2010，18(5)：43-45.

[7] 李木国，黄 影，刘于之. 基于 FPGA 的 PCIe 总线接口的 DMA 传输设计[J]. 计算机测量与控制，2013，21(1)：233-235.

[8] Kavianipour H, Muschter S, Bohm C. High performance FPGA-based DMA interface for PCIe[J]. IEEE Transactions on Nuclear Science, 2012, 61(2): 1-3.

[9] Message Passing Interface Forum. MPI: a message-passing interface standard version 3. 1[EB/OL]. 2015. http://www. mpi-forum. org/docs.

[10] Grahm L, Shipman M, Barren W, et al. Open MPI: a high-performance heterogeneous MPI[C]//Proceedings of 2006 IEEE international conference on cluster computing. Piscataway, NJ: IEEE, 2006: 1-9.

[11] Gropp W, Lusk E, Doss N, et al. MPICH: a high-performance, portable implementation for the MPI message-passing interface [J]. Parallel Computing, 1998, 22(6): 789-828.

[12] Argonne National Laboratory. Hydra process management framework[EB/OL]. 2009. http://wiki. mpich. org/mpich/in-dex. php/Hydra_Process_Management_Framework.

[13] Keller R, Graham R L. Characteristics of the unexpected message queue of MPI applications[C]//Recent advances in the message passing interface-European Mpi users group meeting. Stuttgart, Germany: [s. n.], 2010: 179-188.