# MVICH——一种基于 TH-net VIA 的 MPICH 下的设备层实现

谢 超 陈 渝 孙意嘉 刘昊飞 唐瑞春

(清华大学计算机系高性能研究所 北京 100084)

## MVICH——An Implementation of Device Layer Underlying MPICH Base on TH-net VIA

XIE Chao CHEN Yu SUN Yi-Jia LIU Hao-Fei TANG Rui-Chun

(Institute of High Performance, Department of Computer of Tsinghua University, Beijing 100084)

Email: xiechao00@mails.tsinghua.edu.cn xiechao@tirc.cs.tsinghua.edu.cn

**Abstract** In this article, we have given a elaborate introduction about MVICH implementation underlying MPICH, and illustrated its structure, working principle and performance. MVICH is an implementation of MPICH ADI2 base of VIA protocol, using VIA descriptors queue for transferring data in its low layer. It has some characters such as user level communications, without kernel trapping, memory binding and registering and so on. MVICH uses buffer (region) management, four transfer protocol, converting dynamically at runtime, and special flow control etc. For high speed LAN, the combination of VIA and MPI can give a greater improvement for the performance of MPI applications, and exert those applications' potential ability. It will become the main project of research and development in future.

**Keywords** MPICH, VIA, MVICH, ADI, Vbuf message buffers, Flow control

## 1. 背景介绍

### 1.1 MPICH 的概念[7]

众所周知，MPI(Message Passing Interface)[8]作为一种消息传递标准库规范，被许多并行计算机厂商和研究机构采纳，Message Passing 并行计算模型也作为一种有表现力的、高效率的、为并行编程易于理解的范例出现。很多 MPI 标准的实现都已经得到发展。

### 1.2 关于 VIA 和 MVICH[1]

VIA(Virtual Interface Architecture)作为一种网络间(现大多应用于高速局域网)通信的新型模式，首先由 Compaq，Intel 和 Microsoft 公司联合提出，主要特点在于在处理器、内存和 I/O 子系统之间建立直接的关联，以用户层的角度直接和远端通信，从而避免了用户/核心之间的频繁切换开销，大大提高了效率[4]。尤其值得一提的是，通信时间不再受限于并行系统中各个分立的元素，所以在工作站机群上效率可以成倍提高。清华大学高性能研究所实现了一个基于自行研制的网络设备硬件的 VIA—TH-net VIA，并在此基础上完成了 MVICH 的调试和测试。

MVICH 是基于 VIA 底层通信的一种 MPI 的实现，它建立在 MPICH 的实现之上。分为两个层次，上层实现 MPI 的抽象接口，负责通信和采集的操作，和底部真正的信息传递机制剥离开来；下层实现基础的基于特定设备和协议的点对点通信。一层具有普适性的接口被限定在这两层之中，称为 Abstract Device Interface(ADI)。MVICH 属于设备层实现，和 MVICH 一起协同工作的可以是任何基于 VIA 规范的 VIA 实现。

## 2. ADI 核心机制

如前所述，MPICH 实现结构上分为两层，上层包含 MPI 类库的基本抽象接口，下层面向特定设备和协议进行点对点间通信，在这两层之间引入一个很重要的概念：抽象设备接口 ADI(Abstract Device Interface)。上层所有的 MPI 函数都由 ADI 中定义的宏或者函数实现，这些规格化的 ADI 层的宏或者函数由下层根据具体设备或者协议的不同区别对待，保证了实现上的相对独立。有的情况下，ADI 下面又抽象出一层 Channel Interface(通道接口)。MVICH 相当于一种基于 VIA 的 MPICH ADI-2 实现，它可以实现点对点消息传递，没有被介入通道层，支持多设备。结构如图 1 所示。

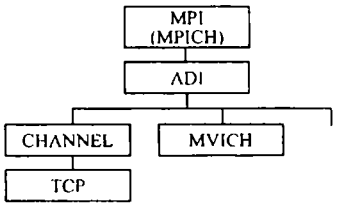


图 1 MPICH-MVICH 关系图

## 3. MVICH 的消息发送机制

在 MVICH 的消息传输机制中，采用固定大小的 vbuf 缓冲块结构来存放数据，给予传输，并以缓冲池的形式分配管理，目的是为了避免 VI 可能出现的资源饥饿现象[6]。程序将消息分成固定大小的 vbuf 块，在必要时候，分一系列发送出去。vbuf 内含有一个 VIA 描述子，形成一一对应(vbuf↔VIP_DESCRIPTOR)的关系；用户数据区 buffer 数组，负责存放控制信息及有效数据；此外还有一个指向 vbuf_region (缓冲区池)的指针。vbuf 在传输的时候不仅包含用户有效数据，而且利用其中缓冲区的头部存放必要的控制和通知信息。

MVICH 实现中定义了一个结构 vbuf_region，称之为缓冲区池。池与池之间以链表结构相连。每个 vbuf_region 结构

的 next 指针指向下一个 vbuf_region，构成统一的全局链，池内含有一定数量的 vbufs，每到数据需要传输的时候就向池里面申请 vbuf。在分配池空间的时候，首先给 vbuf_region 分配空间，然后在池内部分配 vbuf 数组，再链接起来所有池中的 vbufs，便于以后 vbuf 的获取和释放，再把新分配的 vbuf_region 置为池链表的头，全局变量 vbuf_region_head 指向新分配的 vbuf_region。另外，free_vbuf_head 指向新分配缓冲池中的数组首地址。如图 2 所示。

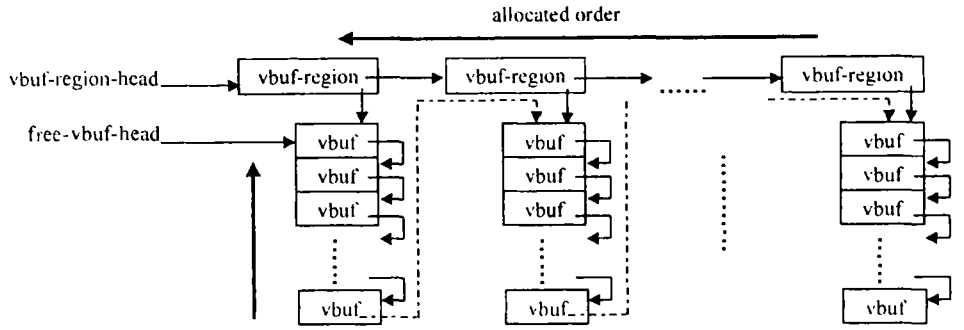


图 2　vbuf 分配管理图

在 MVICH 中，消息的传递采用四种点对点协议之一，具体情况依赖于消息大小和硬件环境。下面分别阐述这四种协议。

1)eager：eager 协议应用于短消息的传送。数据被送到一系列 vbuf 缓冲区里，接收方将其缓存起来直到有匹配的 MPI 接收操作被执行。在流控(flow control)机制防止一次性发送很多 vbufs 的情况下，协议会转化成为 R3。默认的短消息长度限定为 5000 字节，可以在运行时候通过指定 VIADEV_RENDEZVOUS_THRESHOLD 参数来修改。见 R3 模式后面图示(注有 EAGER)。

发送一方进入 eager 模式后，取下一个可用 vbuf，将其中数据区的头部封装进控制信息(VIADEV_PACKET_EAGER_START 包类型)，后面填充用户有效数据，将其发送；紧接着进入循环，在数据没有分块完成之前，继续取 vbuf，填充数据，这时包类型有的变成 VIADEV_PACKET_EAGER_NEXT，直到用户数据发完为止。对于接收一方，负责维护一 rhandle 结构指针，对于 eager 模式来说，接收方用 post 到接收队列里面的 vbufs 缓存发送过来的数据，而后如果找到匹配的接收操作，再行拷贝到 rhandle 指向的域 start，作为存放接收到数据的起始地址，将缓存释放，与发送方形成对应；如果找不到匹配者，则将缓存数据的 vbufs 挂到 rhandle 指向的缓存队列链表中去(以域 vbuf_head 为头，vbuf_tail 为尾)。

2)R3：此协议类似于标准的三次握手协议。MPI 发送方送去“要求发送”的控制信息到达接收方进程。此要求进入接收方队列，直到匹配的 MPI 接收操作被处理。在这种情况下，“可以发送”的控制信息被传送回发送方。于是，发送方开始传递一系列包含用户数据的 vbufs 给接收方。流控有可能使得发送方暂时挂起额外数据的发送，直到有更多的接收方 vbufs 可以利用。本协议在发送和接收方面均要求数据拷贝。此模式图示如图 3(右下为 EAGER 模式)。

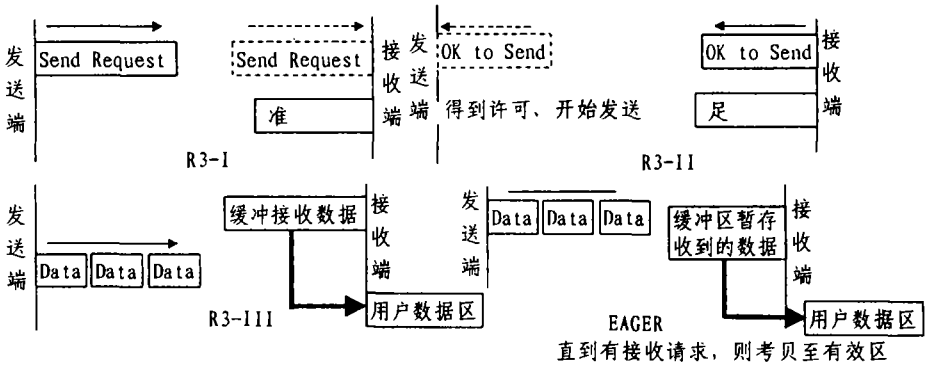


图 3

具体实现中，发送一方发送类型为 VIADEV_PACKET_RENDEZVOUS_START 的包通知远端，(MVICH 里，R3，RGET 和 RPUT 协议统一用 rendezvous 类型包代表，只是在附加的协议字中区分它们)接收一方得知此次发送按照该协议后，首先计算当前连接额外需要 post 的 vbuf 数目(因为初始化时双方都有预先 post 一些 vbufs 入队列)，再返回消息 VIADEV_PACKET_RENDEZVOUS_REPLY，发送方收到此反馈后调用 RENDEZVOUS_IN_PROGRESS 宏把任务挂起到队列中，后续处理时把原先要发送的数据分割到一系列 vbufs 里面发出。

3)RGET：此协议利用 VIA 网卡的 RDMA READ(远端 DMA 读)功能(如果具备)。除了比 RPUT 协议少了一些控制信息之外，基本类似。像 RPUT 一样，将用户发送方缓冲注册并与网卡绑定，发出“要求发送”的控制信息，同时携带发送方将要传送的数据内存虚拟地址。在接收方有匹配的 MPI RECV 操作发生后，它将绑定用户接收方缓冲，并将其注册到网卡。而后接收方进程请求 VIA 网卡执行从发送方用户内存区直接到接收方用户内存区的 RDMA READ 操作。如果发

送或接收一方注册内存失败，协议转化为 R3。此模式图示如图 4。

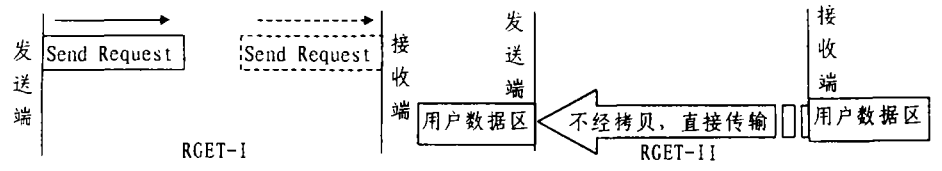


图 4

具体实现中，发送一方所作的工作很简单，依然发送 rendezvous 类型控制包（包含 RGET 协议字）通知远端；而与前面几种协议不同的地方是，接收方收到以后，不必反馈信息，直接进行内存注册，如果成功，则把要传输的大块数据按照 RDMA 一次可以处理的最大字节数将数据分块，分别传递块数据（实际几乎都是接收方在工作，发送方只是提出一个申请而已）。

4）RPUT：此协议利用 VIA 网卡的 RDMA WRITE（远端 DMA 写）功能（如果具备）。MVICH 将用户发送方缓冲注册并与网卡绑定。发送方发出“要求发送”的控制信息到接收方进程。这个控制消息进入处理队列等待，直到接收方 MPI RECV 状态被处理。这时进程绑定用户接收方缓冲，本端注册到网卡，接下来发送“可以发送”的反馈信息到发送方，同时携带接收方用于接收的用户缓冲虚拟地址。发送方进程则请求 VIA 网卡执行从发送方用户内存区直接到接收方用户内存区的 RDMA WRITE 操作，不需要任何中介拷贝。类似于 RGET，本协议在注册内存失败的情况下会转化为 R3。此模式图示如图 5。

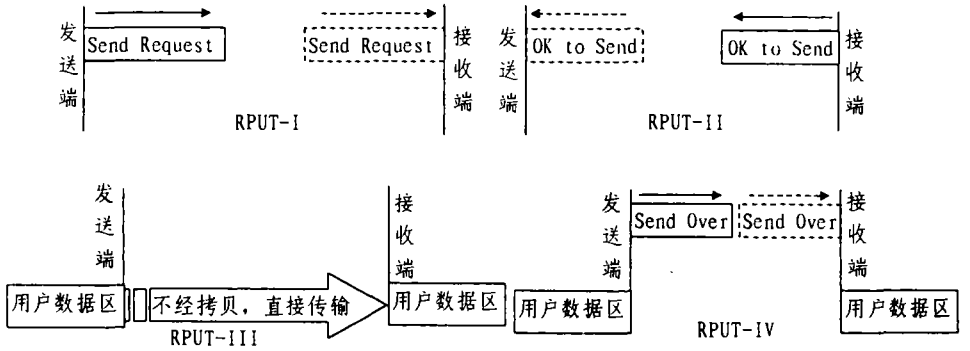


图 5

具体实现中，发送方发出 rendezvous 类型包（含 RPUT 协议字）被远端收到后，接收方反馈消息 VIADEV_PACKET_RENDEZVOUS_REPLY，对方收到后将此任务入栈等待后继处理。最后还要发送 VIADEV_PACKET_RPUT_FINISH 控制信息通告接收方发送完毕，远端收到后作一些善后工作，如释放内存句柄等等。

协议的选择基本属于以上几种情况。如果消息较短，则采用 Eager 协议，如果消息较长，超出了 eager 的允许范围，而且硬件支持远端 DMA 读写，RGET 和 RPUT 协议就被采用。其他情况下，采用 R3 协议。

## 4. MVICH 的流控机制

在 MVICH 中，流控（Flow Control）是一个独特而且至关重要的环节。VIA 下，一个分布式程序必须确保远端进程发送消息之前，在 VI 接收队列里面存在一个大小合适的缓冲。在 MVICH 中，所有 VIA 层的发送和接收都被限制在固定长度的 vbufs 里，故主要关心的情况就是：在一个消息发送之前是否至少存在一个 vbuf 在远端接收 VI 队列里面。为解决此问题，MVICH 采用基于 credit 方案的流控机制：每一连接维护两个计数器，“remote-credit”和“credit-update”。remote-credit 表示已经 post 进远端 VI 接收队列中的 vbufs 数目，credit-update 则表示自上一次消息发送到远端进程之后，重新 post 进入本端 VI 接收队列的 vbufs 数目。初始化阶段每个进程预先分配且 post 固定数目（设为 M）的 vbufs 进入 VI 接收队列，将 remote-credit 设置为 M，credit-update 设置为 0。每当进程发送一条消息，它对应的连接中的 remote-credit 都要减少（因为本端发送方认为对方接收它送出的消息，肯定要消耗一定数目的 vbufs，所以将自己的 remote-credit 减低）。当 remote-credit 降为 0 的时候，本地不再被允许发送任何消息。另一方面，每次进程 post 一个 vbuf 进 VI 接收队列时，本连接的 credit-update 递增。下一次传送消息到远端的时候，发送方将更新过的 credit-update 包含在 MVICH 包头里面送出，然后重置 credit-update 为 0，接收方则把包头中的 credit-update 值叠加到自己的 remote-credit 进行更新，从而允许可能处于阻塞状态的发送操作继续进行。

但是，当消息发送不均匀的时候工作变得不很理想。设想进程 X 发送大量消息而远端进程 Y 不发送消息，那么 X 很快将没有 credit 可用，也收不到 Y 的更新通知消息。这在 R3 协议发送大量数据状况下是有可能出现的，数据需要接收方提供大数量的 vbufs。为解决这个问题，Y 在一定情形下发送“noop”（空消息）给 X，起到通知 credit 更新的作用，这在 MVICH-1.0 里面已经得到了优化，以防止无关的消息通过。又注意到 RDMA 读写操作不消耗接收 vbufs，直接对进程地址空间进行访问。正是数据重拷贝的消除和无需流控局限的特性使得 RPUT 和 RGET 协议成为传输大块数据的首选。

在 MVICH 实现里，很多地方涉及到流控的计算和判断。发送过程开始时，如果需要传送数据长度小于 eager 方式的上限，则判断连接的远端可消耗 credit 减去本次 eager 将要消耗的 vbufs 数是否大于系统要始终保持的 viadev_credit_preserve 数目（这些 vbufs 用于非用户数据的控制包，作为底限），如果大于，调用 eager 模式发送，；否则换作 rendezvous

（下转第 39 页）

给网络计算环境下的任务执行器，并令其进行并行计算，从而达到提高数据挖掘速度的目的。并且，由于本方法本质上支持动态增长的、分布式的、异质的数据，因此可适用于大数据库的数据挖掘。需要指出的是，本文提出的方法同样适用于单机系统。在具有多个处理器的单机系统中，本方法也能够有效地提高数据挖掘的速度。本方法经实验证明，利用丰富的计算机网络计算资源进行数据挖掘，是提高粗糙集规则挖掘系统性能的一个有效途径。

本文仅讨论粗糙集规则的并行挖掘方法，至于在这些规则基础上的规则约简等相关内容，将另文发表。

## 参考文献


1 Chen Ming-Syan, et al. Data Mining: An Overview from a Database Perspective. IEEE Trans. on Knowledge and Data Engineering. 1998,8(6):866～883

2 Pawlak Z. Rough sets. Intl. J. of Computer and Information Science, 1982,11(5):341～356

3 Ziarko W. Variable Precision Rough Set Model. J. of computer and system sciences. 1993,46:39～59

4 Pawlak Z. Rough set: Theoretical Aspects of Reasoning About Data. Kluwer Academic, Dordrecht, The Netherlands, 1991

5 Guan J W, Bell D A. Rough computational methods for information systems. Artificial Intelligence. 1998,105:77～103

6 Pawlak Z, et al. Rough Sets. Communication of The ACM. 1995, 38(11):89～91

7 Chan Chien-Chung. A rough set approach to attribute generalization in data mining. Journal of Information Sciences, 1998,107: 169～176

8 Bonikowski Z, et al. Extensions and intentions in the rough set theory. Journal of Information Sciences. 1998,107:149～167

9 苏健，高济．基于元信息的粗糙集规则增式生成方法．模式识别与人工智能，2001,14(4):428～433


（上接第 23 页）

方式。再有，每次最终的发送都要落实到发送单个 vbuf 的底部函数 viadev_post_send（除 RDMA 传输方式 RGET 和 RPUT 以外），调用宏定义 PACKET_SET_CREDITS 来设置流控数据，其中包括：把本地连接的域 local_credit（每次带过去远端的表示 credit 改变量，相当于 credit_update）赋值给发送包头的 vbuf_credit 成员，而后重置为 0，而且包头的 remote_credit 存贮本地 remote_credit 计数。最后，本地 remote_credit 自减 1。除此之外，流量控制和分析在处理接收过程时也有所涉及，获取发送过来的包以后，将包头里面携带过来的 vbuf_credit 叠加到接受端的 remote_credit 以更新。

为了避免由于一方不断大量发包，一方只接收但不反馈消息而造成的 credit 拥塞现象和程序死锁，在两种情况下会发送 noop 消息：

1. 所有完成队列里面的包都处理完毕，表示所有要到来或者要发出的消息已经完成。此情况下，本端发送 credit 的更新信息给远端，表示本方重新 post 进新的可用 vbufs。

2. 在我们正在处理接收描述子时。如果本地 credit 值足够大，但我们又不发送消息通知远端，连接的远端可能就会停止下来等待我们处理完所有接收任务从而发送一个 noop 使得它们能够继续工作。

## 5. MVICH 的现状和规划

比起一般的 TCP 通信，或是 LAM、MPICH 通信（建立在 TCP/TP 上）的性能，建立在 VIA 协议上的 MVICH 实现由于在传输数据上避免陷入系统核心，防止用户/核心间频繁切换，大大提高了网络速率[2]（例如在 Gigabit 以太网采用不同网卡，GNICII、Alteon 或 SysKonnect 进行比较），是将来高速局域网络应用程序的发展方向之一。用 MPI 与 VIA 的结合取代 MPI 和 TCP 或 UDP 的结合，将更能增强 MPI 执行性能，并且有进一步优化自身的开发潜力[1]。目前我们在 Th-net VIA 上调试通过的 MVICH1.0 版本经过改进，性能有了进一步提高，点到点的通信带宽达到 83MB/s（基于 33MHz/32bit 的 PCI 总线）。它和 MPICH 相互间配合，利用后者的 mpirun 脚本运行程序；在运行时动态调整参；可用 vipl 库点对点连接管理；采取更有效的流控算法，使得空消息的发送数目降至最低；和底层 VIA 协议相一致，使用动态（懒惰）注册内存机制（可降低系统负载）；vbuf 池的动态分配消除了资源不足问题[3]。

今后的工作旨在深入研究各方面因素。首先，拟将系统移植到 Linux2.4 内核上运行，实现 MVICH 的多线程版本，提高系统对于异步事件的敏感度和灵活性，其次，完善流控机制，支持不可靠的网络底层环境和异步通信，在提高稳定性的基础上提升传输速率，最后，进一步消除资源饥饿，从而优化 MVICH 性能。

**总结** 这篇文章里，结合我们清华大学高性能研究所自行开发的 VIA 网络接口卡和 TH-net VIA 底层环境，重点介绍了建立在其上，利用 VIA 传输协议的一种 MPICH 的设备层实现 MVICH。从其背景 MPI 标准入手，说明 MPICH 的核心机制 ADI 和 MVICH 在整个体系中的位置。然后深入到 MVICH 内部，较为详细地阐述其管理消息，实现消息传递的基础（vbuf 固定缓冲机制）。介绍四种传输协议 Eager，R3，RGET，RPUT，包括它们的模式和具体实现方法。接着描述 MVICH 里面的流控环节，依靠这个 MVICH 得以顺利发送大额数据而不造成拥塞。文章的最后概述了我们对提高现有 MVICH 性能所做工作和今后计划。

## 参考文献


1 Berkeley Lab VIA Software, the Regents of the University of California. M-VIA and MVICH, Virtual Interface Architecture Software for Low-Latency, High-Bandwidth, Inter-Process Communication, 2001

2 Ongg H, Farrelly P A. Performance Comparison of LAM/MPI, MPICH, and MVICH on a Linux Cluster connected by a Gigabit Ethernet Network. 2000

3 Lawrence Berkeley National Lab. M-VIA and MVICH Status and Future Plans, 2001

4 Compaq Computer Corp, Intel Corporation, Microsoft Corporation. Virtual Interface Architecture Specification Draft Revision 1.0, 1997

5 Speight E, Abdel-Shafi H, Bennett J K. Realizing the Performance Potential of the Virtual Interface Architecture, 1999

6 Brightwell R, Maccabe A B. Scalabilty Limitations of VIA-Based Technologies in Supporting MPI, 2000

7 Gropp W, Lusk E. A high performance, portable implementation of the mpi message passing interface standard, 1996

8 International J. for Supercomputing Applications. The Message Passing Interface(MPI)standard, 1995. & 都志辉，高性能计算之并行编程技术——MPI 并行程序设计

9 Gropp W, Lusk E. An Abstract Device Definition to Support the Implementation of a High-Level Point-to-Point Message-Passing Interface, 1995