用户层通信接口 VIA 的研究(二)*)

孙伟平　周敬利　余胜生
(华中科技大学计算机学院外存储国家专业实验室　武汉430074)

Study of User-Level Communication Interfaces VIA(2)

SUN Wei-Ping　ZHOU Jing-Li　YU Sheng-Sheng
(Computer College in Huazhong University of Science and Technology Wuhan,430074)

Abstract　Distributed applications require fast and reliable exchange of information across a network. Traditional network architectures do not provide the performance required by these applications, due to communication overhead incurred by messages to move through different layers of the TCP/IP stack in onperating system. VIA defines mechanisms that will bypass the intervention of the operating system layers and avoid excess data copying during sending and receiving of packets. This effectively reduces latency and lowers the impact on bandwidth. The descriptor processing models, VI status and some basic operation functions are studied here.

Keywords　Virtual interfaces, Descriptor, Doorbell, VI status

1. 介绍

SAN 作为一种大容量数据存储的解决方案,为用户通过低迟延和高带宽的通信。但是传统的通信模型由于网络软件协议的高开销而不能充分开发网络的性能。VIA[1~3]是为 SAN 设计的一个用户层内存映射通信模型,它将操作系统内核从通信路径中移出,从而减少了通信开销。VIA 的体系结构及其组成我们在文[8]中已经作了介绍。VIA 是从以前的用户层通信协议中得到灵感的一个工业标准,现在已经有多种硬件以及软件实现了 VIA[4~7],如 Giganet 在对 Linux 和 Windows NT 的驱动器上硬件实现了 VIA, ServerNet、Myrinet 固件实现了 VIA,M-VIA 为多种快速以太网卡提供 Linux 软件 VIA 驱动。本文将对 VIA 进行深入研究,包括描述符、虚拟接口状态以及主要操作等。

2. 虚拟接口的深入研究

2.1　描述符处理模型

数据传输请求是以描述符(Descriptor)来体现的。描述符包含了处理请求的所有信息,如传输的类型,传输的状态,队列信息以及分散一集中类型缓冲区指针列表等。

描述符分为发送/接收描述符和 RDMA 型描述符。每种描述符提供两种不同的功能,即出站数据传输和入站数据传输。一个描述符占用进程的虚拟地址空间的一个长度可变,事实上连续的区域,描述符必须位于 VI 使用者和 VI 提供者注册的内存区域。只要在一个内存区域中只有一个描述符,这个描述符就可以跨越物理页边界。任何时候只要在一个请求中用到描述符,它就必须伴以与描述符所在地内存区域的内存句柄。描述符的定位由 VI 使用者管理,向一个工作队列提交描述符就将此描述符的所有权移交给了 VI 提供者。描述符已经提交后就不允许 VI 使用者修改了。VI 提供者完成描述符后,此描述符的所有权就回到 VI 使用者手里。VI 提供者通过往描述符里写入一个完成状态值来完成一个描述符。如果队列与一个完成队列相连,则 VI 提供者还必须在完成队列中写入一个入口项。

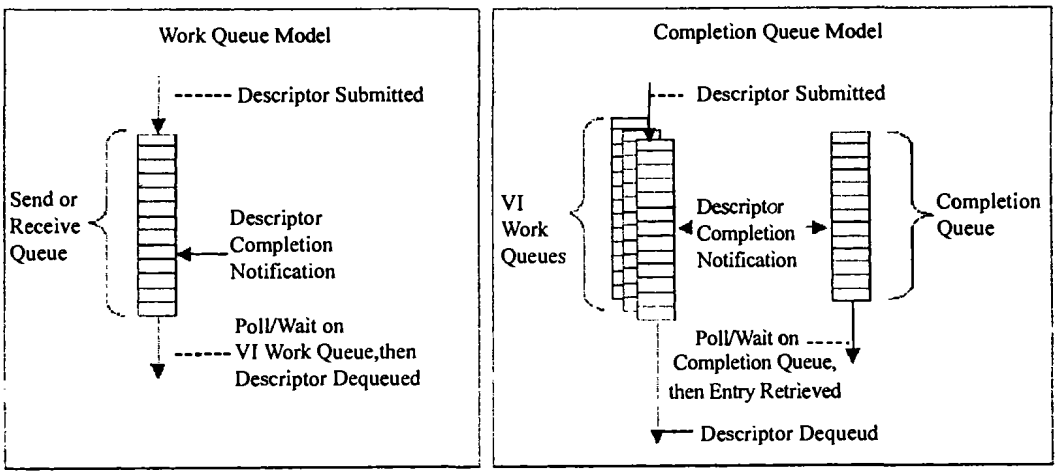

图1　工作队列模型与完成队列模型

*)本课题由国防预研基金413160502及华中科技大学博士后专项基金资助。孙伟平　博士后,主要研究方向为网络存储设备与性能分析。周敬利,余胜生　教授,博士导师,主要研究分析为网络存储与存储网络,多媒体等。

描述符处理模型包括两种:工作队列模型和完成队列模型。这两种模型只在向 VI 使用者通知描述符完成的方式上有所不同。在工作队列模型中,VI 使用者通过检查队列头部的描述符的状态来获知某个工作队列是否完成。如果头部的描述符已经完成,VI 使用者就会将描述符从中删除,即出列。在完成队列模型中,VI 使用者通过检查完成队列头部的描述符来获知一组工作队列是否完成。如果一个描述符已经完成,它就写入完成队列。一旦 VI 使用者得到通知,描述符从一个完成队列完成,这个 VI 使用者必须将这个描述符从相关的工作队列中删除。VI 提供者应该提供一种机制,提供这种机制,VI 使用者能够等待描述符在一个 VI 工作队列中完成,或者等待一个通知将提交给一个完成队列。

描述符格式允许 VI 使用者在每个描述符中指定一个分散-集中列表,使得数据可由硬件直接发送或放置于一个想要的位置。这种分散-集中列表的长度在 NIC 段计数器的长度限制范围内任意。长度可以为0,这样可以进行直接数据传输或无数据传输。VI NIC 必须支持的最小段计数器的长度至少为252个数据片。

数据传输只要不超过任一 VI 提供者的 MTU,任何长度都允许。MTU 允许的最小值为32Kb,这向 VI 使用者保证每个 VI 提供者都能在一个描述符中传输至少32Kb 的数据。有些供应商支持的 MTU 超过32Kb。每个 VI 提供者都应该使 VI 使用者能识别它支持的 MTU。

描述符准备好后,VI 使用者就将其提交给适当的 VI 工作队列,并激活此队列的门铃,通知 VI NIC 来对其进行处理。门铃的格式和操作类型由每个 VI 提供者指定。发送和接收请求提交给它们各自的工作队列。远程 DMA 请求,无论是读还是写,都将提交给发送队列。接收描述符可能在连接建立之前就已经提交,但是它们不会完成直至虚拟接口连接,数据接收,除非出现错误。描述符的预提交可以避免错误的发生,特别是由于缺少接收描述符而导致的数据丢弃或拒绝。在连接建立之前提交发送和 RDMP 描述符会以错误完成。提交描述符不会妨碍对已提交的描述符的处理。

描述符一旦提交给了一个队列,VI NIC 就开始对它进行处理。当 VI NIC 处理一个描述符,或者当数据到达网络时,数据在两个连接的虚拟接口间传输,这时立即传输数据也在传输。如果在处理描述符时出现错误,VI NIC 会根据错误的类型和连接的可靠性等级采取行动。

当数据传输完成,描述符也必须完成。有两种方式,一种是 VI 提供者更新控制段的内容,如果队列与一个完成队列相连,完成队列增加一个入口项。另一种是 VI 使用者将描述符出列。

2.2 VI 状态

一个虚拟接口可能有4种状态,分别为:空闲(Idle),连接挂起(Pending Connect),连接(Connected)与错误(Error)。这几种状态的转换由 VI 使用者发出的请求和网络事件来驱动。

空闲状态 当 VI 被创建时,它便处于空闲状态。只有当处于这个状态时,VI 才能被删除。因为只有当 VI 的工作队列中没有描述符时这个 VI 才能被删除。

向 VI 提供者提出 ConnectRequest 请求时,VI 就由空闲状态切换为连接挂起状态。如果这个请求在规定的时间里不能完成,VI 则返回到空闲状态。当 VI 提供者发出了 ConnectAccept 请求,VI 为连接挂起状态。如果这个请求失败或超时,VI 返回到空闲状态。

可以向一个处于空闲状态的 VI 的接收队列提交描述符,但是只有当 VI 转换成了连接状态,或者 VI 使用者发出断开连接请求,这些描述符才会被处理。向一个处于空闲状态的 VI 的发送队列提交描述符会立即报错。处于空闲状态的 VI 出现的错误将在 VI 使用者试图连接这个 VI 时被报告。通常会唤起异步错误处理机制。

连接挂起状态 表明连接请求已经提交给了 VI 提供者,但是连接还没有建立。

当 VI 处于空闲状态时,如果有 ConnectRequest 操作提交给了这个 VI 的提供者,那么这个 VI 就会变为连接挂起状态。当 VI 处于空闲状态时,如果有 ConnectAccept 操作已经提交给了这个 VI 的提供者,那么这个 VI 就会变为连接挂起状态。这个 VI 将一直保持这种状态直到连接操作已经完成。如果 ConnectRequest 请求得到确认和响应,这个 VI 就会转换成连接状态。ConnectAccept 请求成功完成也会将 VI 转换成连接状态。如果执行请求超时,或者拒绝连接,那么 VI 就会变为空闲状态。

由 VI NIC 产生的传输错误或硬件错误将使 VI 变为错误状态。可以向一个处于连接挂起状态的 VI 的接收队列提交描述符,但是只有当 VI 转换成了连接状态,或者 VI 使用者发出 Disconnect 请求,这些描述符才会被处理。向一个处于连接挂起状态的 VI 的发送队列提交描述符会立即报错。

连接状态 只有在这种状态下数据流才会形成。当 ConnectRequest 请求得到确认和响应,VI 就会从连接挂起状态转换成连接状态。ConnectAccept 请求成功完成也会将 VI 从连接挂起状态转换成连接状态。如果本地 VI 使用者向 VI 提供者提交 Disconnect 请求,那么 VI 就会变为空闲状态。硬件错误将使 VI 变为错误状态。出现的其它错误也可能使 VI 变为错误状态,这依赖于文[8]中讨论过的 VI 的可靠性等级。向一个处于连接状态的 VI 的发送队列或接收队列提交的描述符会得到正常的处理。

错误状态 当在正常的处理过程中出现错误,或者由 VI NIC 生成的事件会使得 VI 成为错误状态。

如果 VI 的可靠性等级为可靠传输或可靠接收,那么有几类错误会将 VI 从连接状态转换为错误状态。Disconnect 请求使 VI 转换为空闲状态。向一个处于错误状态的 VI 的发送或接收队列提交描述符会导致描述符错误地完成。

2.3 VIA 中的主要操作

这节介绍应用程序实现 VI 结构的几种主要的基本操作。

VI 的建立与破坏 为建立一个 VI 端点,应用程序需调用库函数 VipCreateVi,将信息传送给内核级驱动程序,内核级驱动程序将创建信息返回给 NIC。在这里 OS 可以拒绝应用程序访问网络接口或某端点。

为破坏一个 VI 端点,应用程序需调用库函数 VipDestroyVi 。一个 VI 实例只有当它处于空闲状态,它的工作队列中没有任何描述符时才能被破坏,否则调用会返回错误。

```
VipCreateVi(
IN VIP_NIC_HANDLE NicHandle,
IN VIP_VI_ATTRIBUTES * ViAttribs,
IN VIP_CQ_HANDLE SendCQHandle,
IN VIP_CQ_HANDLE RecvCQHandle,
OUT VIP_VI_HANDLE * ViHandle
)
```

参数

NicHandle: VI NIC 的句柄。

ViAttribs: 为这个新的 VI 设置的初始的属性。

SendCQHandle: 完成队列句柄。如果句柄有效,那么此 VI 的发送工作队列将与这个完成队列相连;否则,此 VI 的发送工作队列将与不与任何完成队列相连。

RecvCQHandle: 完成队列句柄。如果句柄有效,那么此 VI 的接收工

作队列将与这个完成队列相连；否则，此 VI 的接收工作队列将与不与任何完成队列相连。
ViHandle：新创建的 VI 实例的句柄。
返回值
VIP_SUCCESS - 操作成功完成。
VIP_ERROR_RESOURCE - 资源不够。
VIP_INVALID_PARAMETER - 输入的参数中有一个无效。
VIP_INVALID_RELIABILITY_LEVEL - 要求的可靠性级别无效或不支持。
VIP_INVALID_MTU - 最大传输单元无效或不支持。
VIP_INVALID_QOS - 服务质量无效或不支持。
VIP_INVALID_PTAG - 保护标志属性无效。
VIP_INVALID_RDMAREAD - 要求支持 RDMA 读，但 VI 提供者不支持这个属性。

```
VipDestroyVi(
IN VIP_VI_HANDLE ViHandle
)
```

参数
ViHandle：要破坏的 VI 实例的句柄。
返回值
VIP_SUCCESS - 操作成功完成。
VIP_INVALID_PARAMETER - 无效 VI 句柄。
VIP_ERROR_RESOURCE - VI 不是空闲状态，或在工作队列中仍有描述符。

内存的注册与取消 所有的数据缓冲区和描述符都必须位于一个已注册的内存区域中，内存注册通过调用 VipRegisterMemory 来实现。注册的内存通常对应于物理上连续的一些内存页，允许 NIC 直接内存访问，并会将注册内存区域的地址告诉 NIC。

```
VipRegisterMem(
IN VIP_NIC_HANDLE NicHandle,
IN VIP_PVOID VirtualAddress,
IN VIP_ULONG Length,
IN VIP_MEM_ATTRIBUTES * MemAttribs,
OUT VIP_MEM_HANDLE * MemoryHandle
)
```

参数
NicHandle：当前打开的 NIC 的句柄。
VirtualAddress：将注册的内存区域的开始地址。
Length：将注册的内存区域的长度(以字节计)。
MemAttribs：内存属性。
MemoryHandle：如果注册成功，这个内存区域的新的内存句柄；否则为 NULL。
返回值
VIP_SUCCESS - 内存注册成功完成。
VIP_ERROR_RESOURCE - 由于资源限制注册操作失败。
VIP_INVALID_PARAMETER - 有无效输入参数。
VIP_INVALID_PTAG - 保护标志位属性无效。
VIP_INVALID_RDMAREAD - 属性要求内存区域允许 RDMA Read，但是 VI 提供者不支持。

VI 的连接与断开 VIA 是一个面向连接的协议，当一个 VI 第一次被创建时，它没有与任何 VI 连接。如果要进行数据传输，VI 必须连接到一个远程的 VI。这个调用传到内核级的驱动器，此驱动器构建内部状态来保护这个 VI，然后 NIC 通知端点进行连接。数据传输结束后，连接的 VI 要断开，出现设置 VI 的状态并释放描述符。

为建立与远程 VI 的连接，VI 使用者需要调用 VipConnectRequest，发送请求给 VI 提供者。VI 端点的连接模型是客户机/服务器模型。服务器端等待传入的连接请求，任何根据远程 VI 的属性来决定接受还是拒绝。为断开连接，VI 使用者需向 VI 提供者发送一个 Disconnect 请求。Disconnect 请求通过设置合适的错误位来完成 VI 端点中的所有未完成的描述符。VI 提供者检测 VI 是否断开连接，并通知 VI 使用者。

```
VipConnectRequest(
IN VIP_VI_HANDLE ViHandle,
IN VIP_NET_ADDRESS * LocalAddr,
IN VIP_NET_ADDRESS * RemoteAddr,
IN VIP_ULONG Timeout,
OUT VIP_VI_ATTRIBUTES * RemoteViAttribs
)
```

参数
ViHandle：本地 VI 端点的句柄。
LocalAddr：本地网络地址。
RemoteAddr：远程网络地址。
RemoteViAttribs：远程端点的属性。
返回值
VIP_SUCCESS - 连接成功建立。
VIP_TIMEOUT - 连接操作超时。
VIP_ERROR_RESOURCE - 由于资源限制连接失败。
VIP_INVALID_PARAMETER - 有无效参数。
VIP_REJECTED - 连接被远程端点拒绝。

```
VipDisconnect(
IN VIP_VI_HANDLE ViHandle
)
```

参数
ViHandle：一个已连接的 VI 端点的实例。
返回值
VIP_SUCCESS - 成功断开连接。
VIP_INVALID_PARAMETER -参数 ViHandle 无效。

数据的发送与接收 在 VI 结构中发送数据要用到3个关键的数据结构。

1. 数据缓冲区。由应用程序在注册内存区域中指定，从中取数据。通常一个注册的内存区域用于多个信息。

2. 描述符。对每个消息，应用程序将数据发送到缓冲区并在内存区域中为这个缓冲区建立一个描述符。这个描述符包括数据的地址与长度，保护以及状态等信息。

3. 门铃。应用程序提交一个 doorbell 来表示这个描述符以及准备好，可以将这个描述符的地址写入 NIC 中的一个寄存器。NIC 包括一列 doorbell，当这个 doorbell 到达队列的前面，NIC 就通过这个 double indirection 来传输数据。最后，NIC 更新描述符的状态域，应用程序从这个状态就能决定完成传输还是以及出现错误。

为接收一个消息，应用程序在注册的内存区域中指定一个空的数据缓冲区，并为它建立一个描述符。然后，它向 NIC 提交一个 doorbell 告诉 NIC 接收新的数据包的空间以及准备好。所有这些步骤都必须在数据包到达之前完成。如果数据包到达时，NIC 还没有接到接收数据的 doorbells ，那么就丢弃这个数据包。如果数据包已到达，NIC 就将 doorbells 作为开始指针，遍历 the double indirection，等待被接收的数据包到达事先指定的缓冲区。然后更新描述符的状态域，表明缓冲区里已经有了新接收的数据。

下面以两个函数为例来说明数据的传输：

```
VipPostSend(
IN VIP_VI_HANDLE ViHandle,
IN VIP_DESCRIPTOR * DescriptorPtr,
IN VIP_MEM_HANDLE MemoryHandle
)
```

参数
ViHandle：虚拟接口实例。
DescriptorPtr：将要提交给发送队列的指向描述符的指针。
MemoryHandle：将要提交的描述符所在的内存区域的句柄。
返回值
VIP_SUCCESS - 发送描述符成功提交。
VIP_INVALID_PARAMETER -VI 句柄无效。

```
VipSendDone(
IN VIP_VI_HANDLE ViHandle,
OUT VIP_DESCRIPTOR * * DescriptorPtr
)
```

参数
ViHandle：虚拟接口的实例。
DescriptorPtr：已完成的描述符的地址。
返回值
VIP_SUCCESS - 返回一个已完成的描述符。
VIP_NOT_DONE - 没有找到已完成的描述符。
VIP_INVALID_PARAMETER - VI 句柄无效。

3. VI 结构的发展趋势

SAN(Storage Area Network)是一种针对大容量数据的

(下转第53页)

配资源，所以其恢复时间最短；而后者正好相反，在错误发生之后才计算备份路径和分配资源，所以错误恢复时间最长。对于 PSSRM，Hold-On-Time 值越大，错误恢复时间就越大。如果 Hold-On-Time 值大到一定程度，PSSRM 的恢复时间可能比 End-to-End 的预留式恢复策略的恢复时间还要长，在图4中未表示出来。对于错误恢复率，理所当然的是保护式的错误恢复策略最高。随着 Hold-On-Time 值的增大，PSSRM 的错误恢复率将会增加，并且从总体上都高于预留式的错误恢复策略。

结束语 从仿真的实验结果，可以得出结论：PSSRM 的资源利用率、恢复时间和错误恢复率的综合平衡明显优于其他的策略。在本文中，算法使用了如 Hold-On-Time 和带宽等简单的 QoS 约束条件来优化网络的性能。PSSRM 同样可以使用其他的 QoS 约束条件来优化网络。

参 考 文 献

1 Banerjee A, et al. Generalized Multi-Protocol Label Switching: An Overview of Signaling Enhancements and Recovery Techniques. IEEE Communication Mag., 2001, 39(7): 144～151

2 Mannie E, Papadimitriou D, et al. Recovery (Protection and Restoration) Terminology for Generalized Multi-Protocol Label Switching (GMPLS). work in progress, Internet draft, draft-ietf-ccamp-gmpls-recovery-terminology-02. txt, May 2003

3 Mannie E, et al. Generalized Multi-Protocol Label Switching (GMPLS) Architecture. work in progress, Internet draft, draft-ietf-ccamp-gmpls-architecture-02. txt, March 2002

4 Sharma V, et al. Framework for MPLS-based Recovery. Internet Draft, draft-ietf-mpls-recovery-framwork-03. txt, Feb. 2002

5 Shimano K, et al. RSVP extensions for GMPLS restoration signaling. Internet Draft, draft-shimano-imajuku-gmpls-restoration-00. txt, Feb. 2003

6 Liu, et al. OSPF-TE Extensions in Support of Shared Mesh Restoration. Internet Draft, draft-liu-gmpls-ospf-restoration-00. txt, Oct. 2002

7 Berger L. Generalized MPLS Signaling-RSVP-TE Extension. RFC3473, Jan. 2003

8 Papadimitriou D, Mannie E. Analysis of Generalized MPLS-based Recovery Mechanisms (Including Protection and Restoration). Internet Draft, work in progress, draft-ietf-ccamp-gmpls-recovery-analysis-01. txt, Nov. 2003

9 Mannie E. Recovery (Protection and Restoration) Terminology for generalized Multi-Protocol Label Switching (GMPLS). Internet Draft, work in progress, draft-ietf-ccamp-gmpls-recovery-terminology-02. txt, Nov. 2003

10 Lang J P. Generalized MPLS Recovery Functional Specification. Internet Draft, work in progress, draft-ietf-ccamp-gmpls-recovery-functional-00. txt, July 2003

11 Papadimitriou D, et al. Shared Risk Link Groups Encoding and Processing. Internet Draft, draft-papadimitriou-ccamp-srlg-processing-01. txt, May 2003

12 Awduche D, Rekhter W. Multiprotocol Lambda Switching: Combining MPLS Traffic Engineering Control with Optical Crossconnects. IEEE Comm. Mag., March 2001

13 Czezowski P. Optical Network Failure Recovery Requirements. Internet Draft, draft-czezowski-optical-recovery-reqs-00. txt, April 2003

14 Lang J P, et al. RSVP-TE Extensions in Support of End-to-End GMPLS-based Recovery. Internet Draft, work in Progress, draft-lang-ccamp-gmpls-recovery-e2e-signaling-00. txt, Aug. 2003

15 Doshi B T, Dravida S, Harshavardhana P, et al. Optical Network Design and Restoration. Bell Labs Technical Journal, vol. January-March, 1999. 58～84

（上接第33页）

存储解决方案，具有高数据传输率、高可用性和可扩展性。VI 结构作为对存储区域网(SANs)体系结构的发展，与传统的结构相比，它在带宽、迟延以及主机负担的处理上更有效。

VI 结构已经越来越被业界所接受。应用以及 Internet 网络的进一步发展将会使这项技术受益。VI 结构不要求 TCP/IP 广域的、分布式的能力就能给所有的应用带来性能与效率上的提高。从理论上来说，所有的拥有数据中心的网络骨干通信都可以从 VI 的实现中受益。

VI 结构是今天高效、高性能的互联技术技术的第一代技术，在接下来的几年里，InfiniBand 结构(IBA)会取代基于总线的互联，而成为主导的互联技术。IBA 是基于光纤的互联技术，几乎具有无限的可扩展性，在高带宽、低迟延的环境下允许多个设备并行通信。IBA 具有 VI 结构的许多特征，如 VI 结构的虚拟通道、远程访问能力等。然而，与 VI 结构不同的是，IBA 互联技术并不依赖于服务器的速度较慢的 PCI 总线结构。

参 考 文 献

1 Compaq, Intel and Microsoft Corporations. Virtual Interface Architecture Specification. Version 1. 0, 1997. http://www.viarch.org

2 Dunning D, et al. The Virtual Interface Architecture. IEEE Micro, Mar. -Apr. 1998. 66～76

3 Buonadonna P, Geweke A, Culler D. An Implementation and Analysis of the Virtual Interface Architecture. In: Proc. of SuperComputing Conf. Nov. 1998

4 Giganet Inc., cLAN Performance. http://www.giganet.com/products/performance.htm

5 Dubnicki C, Bilas A, Li K, Philbin J F. Design and Implementation of Virtual Memory-Mapped Communication on Myrinet. In: Proc. of the 11th Intl. Parallel Processing Symposium, April 1997

6 Eicken T V, Basu A, Buch V, Vogels W. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In: Proc. of the 15th Symposium on Operating Systems Principles, 1995. 40～53

7 National Energy Research Scientific Computing Center (NERSC), M-VIA: A High Performance Modular VIA for Linux. http://www.nersc.gov/research/FTG/via

8 孙伟平，周敬利，余胜生. 用户层通信接口 VIA 的研究(一). 计算机科学，2003，30(11)：125～128