

Virtual Interface Architecture Primer

Traditional operating system interfaces to network hardware prevent application programs from taking advantage of the performance improvements (i.e. high bandwidth and extremely low latencies) of gigabit-per-second interconnects like Fibre Channel. To eliminate this bottleneck, a group of independent hardware and software vendors have defined the Virtual Interface (VI) Architecture. VI is a standard and extremely efficient way to reduce software overhead between a high-performance CPU/memory subsystem and a high-performance network.

This primer introduces the VI Architecture and provides an overview of the key elements of VI. Using the clustering challenge as an example, it describes how these elements work to provide an efficient way of moving data between applications and network hardware.

A TECHNICAL OVERVIEW

Although gigabit-per-second interconnects like Fibre Channel provide network bandwidths with extremely low latencies, traditional operating system interfaces to network hardware prevent application programs from taking advantage of these performance improvements. To eliminate this bottleneck, a group of independent hardware and software vendors have defined the Virtual Interface (VI) Architecture. VI is a standard and extremely efficient way to reduce the software overhead between a high-performance CPU/memory subsystem and a high-performance network.

This primer is intended as an introduction to the VI Architecture and includes an overview of the key elements of VI as well as a description of how these elements work to provide an efficient way for moving data between applications and network hardware. The motivation for creating the VI specification was the clustering challenge - and that's where we begin.

THE CLUSTERING CHALLENGE

Clustering is the scaling of computer power to increase overall performance and availability. This is accomplished by connecting processors so that they can collaboratively perform computations while appearing as a single computing resource to the application or client (see Figure 1). From the application's point of view, a computer cluster is a virtual mainframe because distribution of processing is completely transparent. The performance gains realized by scaling with clusters are highly dependent on the underlying network and communication protocols. The goal is to obtain linear performance scaling (i.e. a return of a dollar of performance gain for every dollar of investment).

The overhead of standard communication protocols and the inefficiencies of their interaction with operating systems greatly reduce the performance benefits of scaling. For example, some of the best TCP/IP protocol stack implementations have significant throughput bottlenecks (see discussion in "Profiling and Reducing Processing Overheads in TCP/IP" by Kay and

Pasquale). Further, they are CPU intensive and extremely inefficient with respect to end-to-end latencies.

VIRTUAL INTERFACE ARCHITECTURE (VI)

As a standard mechanism to deliver high performance directly to applications, and to eliminate overhead and inefficiencies common to operating systems and network stacks, Intel®, Compaq® and Microsoft® have jointly defined and developed the Virtual Interface Architecture (VI). The development of VI now allows application developers and operating system vendors to code to a standard architecture for lightweight messaging. Further, it allows hardware vendors to supply products implementing it, thus creating a total end-to-end solution with multi-vendor support.

VI bypasses much of the overhead in traditional protocol stacks and provides a more direct access to the network interface hardware. VI gives a concise set of operations for moving data between network connec-



Figure 1. Clustered servers in a Fibre Channel Network.

AD PHAR LAP

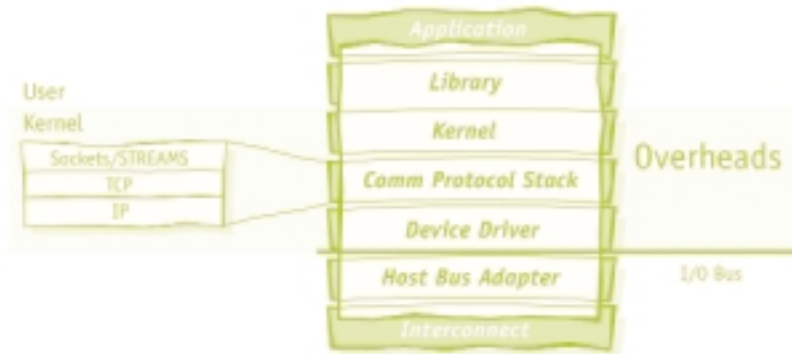


Figure 2. Contemporary Network Architecture.

tions with latency characteristics closer to memory movement than network operations. It eliminates most of the copying inherent in the movement of data between an application and the supporting operating system. It also reduces wasted CPU cycles typically required to manage the interaction between the operating system and the network interface hardware.

VI defines a combination of hardware, firmware and operating system driver interaction to improve the efficiencies of network communications. In fact, one of the major tenets of VIA is that the architecture be simple enough to be integrated in silicon. VI-compliant hardware, such as a Fibre Channel NIC with hardware assist for VI connections and compatible operating system drivers, off-loads the host CPU by performing much of the work involved in transferring data directly in and out of application-based buffers. At the same time, VI drivers are fully compatible with both general purpose and real-time operating systems such as Windows NT™, Linux, VxWorks® and LynxOS™.

NETWORK ARCHITECTURE

The contemporary network architecture (see Figure 2) consists of many layers of software interaction that sit between application software and the network hardware.

Overhead in this system includes:

- virtual to physical memory mapping of data
- operating system context switches
- interrupt latencies associated with hardware interrupts from NIC

duplicate/redundant data copying as it moves between application and network hardware

inefficient protocol stacks resulting in costly CPU overhead (e.g., IP checksum handled by CPU).

The VI network architecture (see Figure 3) eliminates this CPU overhead by providing direct communication between application software and the network hardware, including:

- direct access from NIC to application memory, eliminating virtual to physical memory mapping overhead
- user-level access to NIC via a Kernel Agent reducing user to kernel context switches
- elimination of operating system driver interrupts
- direct data copies between application memory and NIC
- NIC-based protocol processing, off-loading the host card CPU.

VIA MODEL

This section introduces VIA nomenclature and describes the major elements of the VI Architecture Model. Figure 4 illustrates the relationship between these major elements.

User vs. Kernel Space

When describing the elements of VI, User space and Kernel space are often distinguished. User space refers to application processes while operating system functions (including device drivers, described below) operate in Kernel or protected mode. Application processes operate in their own User space that is pro-

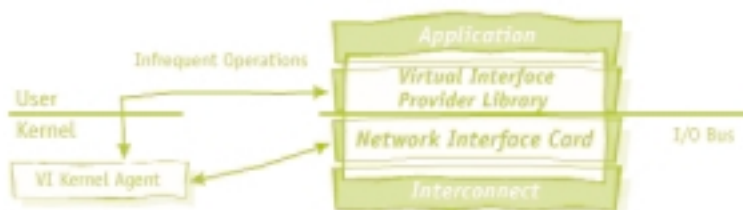


Figure 3. VI Network Architecture.

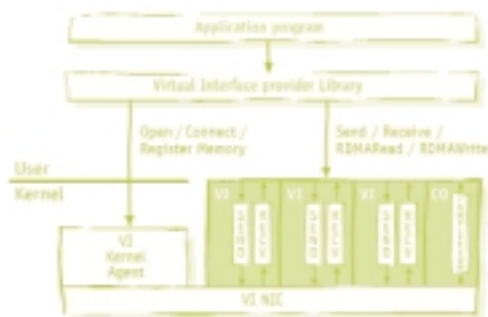


Figure 4. VI Architecture Model.

tected from other applications. In User mode, the operating system controls application access to network hardware through operating system extensions known as device drivers. Device drivers provide a set of function calls that can be used to send and receive data using the network hardware. In traditional operating system interfaces to network hardware, these driver-level calls result in intermediate data copies from User to Kernel space and also in operating system context switches. Context switches and intermediate data copies result in increased CPU overhead and increased latencies in data movement that VI attempts to reduce or eliminate.

Application Program

An application program is any software process or processes that will initiate the movement of data over the network to another process. An application operates in a protected region of memory, where it maintains data that can be communicated over network interface hardware.

VI Provider Library (VIPL)

The VI Provider Library (VIPL) is the interface between the application program and the network hardware (called the VI Network Interface Card or VI NIC). The VIPL is the standard API that provides a common set of function calls independent of the underlying hardware implementation.

The VIPL provides two communication paths to the VI NIC, one through the VI Kernel Agent and the other directly to the VI NIC. VIPL commands used to create and manage VI tasks do this through the VI Kernel Agent. The more common VIPL operations that support performance-critical data movement operations bypass the VI Kernel Agent overhead and communicate directly with the hardware on the VI NIC. Because both the application program and VIPL reside in User space, data transfers between the application and VI NIC can be performed without incurring context switches between User and Kernel space.

Virtual Interface (VI)

A virtual interface is the interface between a VI NIC and an application program process that gives the VI direct access to application program memory. Each VI consists of a Send queue and a Receive queue. The queues hold descriptors that are data structures used by the application to communicate Send and Receive

requests to the NIC.

VI Kernel Agent

The VI Kernel Agent is a privileged part of the operating system responsible for management functions for registering communication memory, and setting up and breaking down VIs. The VI Kernel Agent is implemented as an operating system driver and operates in Kernel rather than User space. Functions handled by the Kernel Agent are infrequent because VIs are typically established once and have long lives; thereby minimizing the overhead in switching between user and kernel space. In fact once the VI is established, application programs can communicate using VIPL Send and Receive calls that communicate directly to the hardware, bypassing the operating system overhead.

Completion Queues

A completion queue stores information used to notify a VI of the completion of a Send or Receive operation. Completion queues reside in VI NIC memory and simplify synchronization activities by combining completion notifications from multiple interfaces into a single completion queue. Completion queues are created by an application through a VIPL call.

Hardware Doorbells

Each Send and Receive queue has an associated work notification mechanism called a "doorbell." A doorbell is a mechanism for a process to notify the VI NIC that work has been placed on a Send or Receive request. A doorbell is typically implemented as a memory-mapped register on the VI NIC.

VI NIC

A VI NIC is a network interface controller that complies with the VIA specification. The VI NIC incorporates Send and Receive queues, completion queues, doorbells and all of the processing logic for mapping VI data to a network such as Fibre Channel. For an example of a VI NIC implementation, see DY 4's PMC-642

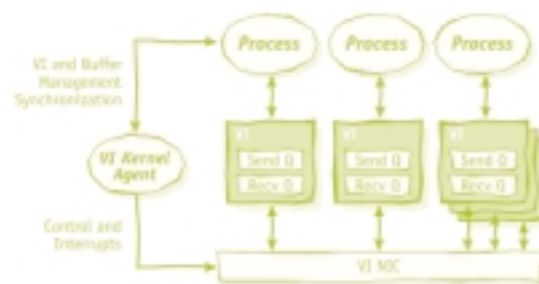


Figure 5. VI Dataflow.

Fibre Channel module at <http://www.dy4.com/ps-datasht/pdf/pmc642.pdf>.

WHAT'S A VI?

Conceptually, VIA provides multiple independent application processes with their own direct control of the network hardware. Each VI is a communication end-

point that can be logically connected to another VI to support point-to-point communication of data between processes (see Figure 5). Each VI consists of one Send queue and one Receive queue, each NIC can support as many as 64K separate VIs, and a process can have multiple VIs.

CREATING A VI

The VIPL provides an application call that in turn commands the VI Kernel Agent to initiate the creation of a VI. The creation process consists of associating a doorbell and Send/Receive queues with a VI and returning addresses of these to the calling application. There is no connection established upon creation of a VI and no data will flow between processes until a connection is established with a pair of VIs.

ESTABLISHING A CONNECTION

VI provides a flexible connection management scheme with support for both a client-server and peer-to-peer model. The client-server model offers the traditional blocking connection scheme; one side of the connection waits for connection requests, and the connection is established after another remote process issues a connection request. The peer-to-peer scheme provides a non-blocking connection scheme that allows a connection request to time out if there is no matching connection request from a peer after a fixed time period.

Establishing a VI connection is typically an infrequent task that results in a long-life connection for many data exchanges. The connection will remain until an explicit disconnect request is issued by an application on one side of the connection.

MEMORY REGISTRATION AND PROTECTION

Most modern operating systems support virtual memory management schemes that force a virtual-to-physical address map translation so that network hardware can access user data. These address translations are required for every data transfer request introducing a significant overhead for every network transfer.

The VI architecture requires that data blocks used for data transfer over a VI be tied to a physical memory address rather than a virtual address in User space. This provides the network hardware with free access to application memory without participation of the operating system. This reduces software overhead in the performance-critical data transfer path, and eliminates the need for intermediate data copying from User to Kernel space.

The VI architecture enforces a memory registration scheme to tie down physical memory. Memory registration requires that application processes allocate a region of memory in User space and pass the address of that region to the VI Kernel Agent as a parameter of the VIPL register-memory command. A memory handle returned by the VIPL function can then be used in subsequent data transfer requests.

By eliminating operating system level memory management from the critical path, there is a risk that one

also eliminates the protection mechanisms that ensure that applications can't interfere with each other. VI provides applications with direct access to the network interface hardware while retaining access protection. It offers a mechanism for memory protection in the form of protection tags that ensures that a user process cannot send out of or receive into memory locations that it does not own.

Sending and Receiving Data

A pair of processes connected by a VI can use that channel to communicate data with minimal host processor and operating system overhead. Two different data transfer mechanisms are supported, a traditional Send/Receive model and the Remote Direct Memory Access (RDMA) model.

Send Receive Model

The Send/Receive model is the traditional method of transferring data between two end-points. With this model, the receiving end-point executes a Receive command and the transmitting end-point executes a Send command. The Receive-side process specifies the memory location where data will be placed and the Send-side process specifies the memory location from which data will be sent. For synchronization purposes, both the Send-side and Receive-side processes are notified when their respective requests have completed.

RDMA Model

Somewhat unique to VI is the RDMA model, whereby the initiator of the data transfer specifies both the source memory region and the destination memory region. The RDMA model includes both an RDMA write and an RDMA read capability. This is a very useful operation because it allows a node to transfer data to or from remote memory without intervention by the remote CPU. RDMA operations do require that the remote end pre-registers memory regions to be used for RDMA operations.

SYNCHRONIZATION AND COMPLETION

VIA supports both polled and interrupt mechanisms to synchronize and inform the application program that an operation has completed. By using a VIPL call to poll on the head of each VI queue, an application can check on completion of a request. If the descriptor in the work queue is complete, the VIPL call removes the descriptor from the head of the queue and returns the descriptor address to the calling process; otherwise, it returns a unique status and the head of the queue does not change.

The user may also use a blocking call to wait on the completion of a request. When the request has completed, an operating system interrupt will be generated. VIA supports both an interrupt to awaken a blocked process as well as a callback function.

As an alternative to synchronizing on individual VI queues, there is a construct that supports coalescing completion notifications from multiple work queues into a single completion queue. This simplifies the synchronization activities and reduces overhead for appli-

*AD NATIONAL
INSTRUMENTS*

cation software with multiple VIs and multiple outstanding requests.

LOOKING FORWARD

VIA introduces open standard networking concepts, data structures, architecture and API that allow hardware and software vendors to build compatible, interoperable, high performance, and competitive networking products. It allows application programs to take advantage of the high bandwidth and low latency capabilities of high speed network technologies, such as Fibre Channel, by reducing the overhead and inefficiencies of traditional operating systems and networks stacks.

FOR MORE INFORMATION ON VIA:

Virtual Interface Architecture Specification

This is the primary VIA specification that defines the VI architecture and all related elements. This document can be downloaded from the Intel developer's site at http://developer.intel.com/design/servers/vi/developer/ia_imp_guide.htm. This site also provides the VI Conformance Tests - a set of tests that can be used by independent hardware and software vendors to demonstrate conformance of VI-related products to the VIPL specification.

VIA Developer's Guide

The specification defines the standard application program interface. This API is called the VI Provider Library. It is defined in an operating system-independent manner. The VIPL specification can be downloaded from the VI Architecture web-site at www.viarch.org

FC-VI Specification (Fibre Channel Mapping to VI)

The mapping of the VI architecture onto the Fibre Channel framing protocol is defined by the FC-VI (ANSI T11) standards working group. A copy of the standard can be downloaded from the T11 web site at www.t11.org. The goal of the FC-VI group is to provide a mapping between Fibre Channel and VI that fully exploits the potential of both. Included within the recommended scope of this project is the creation of mappings to Fibre Channel for the transport of VIA and support for the full range of Fibre Channel topologies, including loops and fabrics, to enable scalable clustering solutions.

VI Developers Forum (VIDF)

The VI Developers Forum (VIDF) is an organization of independent software and hardware vendors that works to define and maintain VI as an open independent standard. VIDF working group topics include:

- Operating System support extensions for VI
- VIPL standardization
- socket APIs on top of VIPL
- fault tolerance

The defining standard for multi-pathing and fail-over

Information on the VIDF can be requested by sending a request to via@nersc.gov.

VIA implementation for Linux (M-VIA)

M-VIA (Modular-VIA) is a complete high-performance implementation of the VIA for Linux. It was written at the NERSC center at Lawrence Berkeley National Laboratory. It is essentially shareware that can be ported to new hardware platforms. The NERSC web site provides a downloadable image as well as information on performance numbers and hardware support for M-VIA. <http://www.nersc.gov/research/ftg/via/>.

MPI layered on top of VIA

Message Passing Interface (MPI), a de facto standard for parallel communications, has been implemented on VIA to provide an efficient interconnect for clusters of workstations in a gigabit/second system area network. MPI has become the programming interface of choice in the parallel computing world and is used to interconnect digital signal processors.

For information on MPI Softech's implementation of MPI on VIA see <http://www.giganet.com/cluster/whitepapers/MPIimpforvi.html> ■

Wilf Sullivan is the Product Marketing Manager of Channel 1 networking products and Pentium* single board computers at DY 4 Systems Inc. Building on his Master's degree in Computer Science, Wilf has over 10 years' experience as a designer and developer of software for the real-time embedded environment.*

REFERENCES

1. Dunning et al., "The Virtual Interface Architecture," IEEE Micro, March/April 1998, pp. 66-73.
2. Kay and Pasquale, "Profiling and Reducing Processing Overheads in TCP/IP" IEEE/ACM Transaction on Networking, Vol. 4, No. 6, December 1996, pp. 817-828.
3. Mukherjee et al., "Making Network Interfaces Less Peripheral," IEEE Computer, Oct. 1998, pp. 70-76.
4. "Next Generation I/O: A New Approach to Server I/O Architectures," Aberdeen Group Technical White Paper, February 1999, <http://www.ngioforum.org/events/02991357.html>.
5. Skjellum et al., "An Efficient MPI Implementation for Virtual Interface (VI) Architecture-Enabled Cluster Computing," <http://www.giganet.com/cluster/whitepapers/MPIimpforvi.html>.
6. "Virtual Interface (VI) Architecture Developer's Guide," Revision 1.0, September 1998, www.viarch.org.
7. Virtual Interface Architecture for Clustered Systems, Dell Computer Corporation White Paper, September 1998, <http://www.dell.com/r&d/whitepapers/wpvia/wpvia.htm>.
8. "Virtual Interface Architecture Specification," Version 1.0, December 1997, http://developer.intel.com/design/servers/vi/developer/ia_imp_guide.htm.
9. Von Eicken et al., "Evolution of the Virtual Interface