Report Number: WUCS-91-57

1991-12-01

Axon Host-Network Interface Architecture for Gigabit Communication

James P.G. Sterbenz and Gurudatta M. Parulkar

This paper describes the design of the Axon host-network interface architecture, and performance factors determining its design. The Axon project is investigating an integrated design of the host architecture, operating systems, and communications protocols to allow applications to utilise the high bandwidth provided by the next generation of communications networks. The Axon host architecture and network interface is designed to provide a path directly between the network interface with direct access to host memory, capable of delivering bandwidth in excess of 1 Gbps to applications. This provide the ability to support demanding applications such as scientific visualisation and imaging.

... Read complete abstract on page 2.

Axon Host-Network Interface Architecture for Gigabit Communication

James P.G. Sterbenz and Gurudatta M. Parulkar

Complete Abstract:

This paper describes the design of the Axon host-network interface architecture, and performance factors determining its design. The Axon project is investigating an integrated design of the host architecture, operating systems, and communications protocols to allow applications to utilise the high bandwidth provided by the next generation of communications networks. The Axon host architecture and network interface is designed to provide a path directly between the network interface with direct access to host memory, capable of delivering bandwidth in excess of 1 Gbps to applications. This provide the ability to support demanding applications such as scientific visualisation and imaging.

Axon Host-Network Interface Architecture for Gigabit Communications

James P. G. Sterbenz and Gurudatta M. Parulkar

WUCS-91-57

December 1991

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

This work is supported in part by Bellcore, BNR, Italtel SIT, NEC, Southwestern Bell and NSF grant DCI-8600947.

Axon Host–Network Interface Architecture for Gigabit Communications[‡]

James P. G. Sterbenz*†
jps@wucs1.wustl.edu
+1 314 726 4203

Gurudatta M. Parulkar*
guru@flora.wustl.edu
+1 314 889 4621

ABSTRACT

This paper describes the design of the Axon host–network interface architecture, and performance factors determining its design. The Axon project is investigating an integrated design of host architecture, operating systems, and communication protocols to allow applications to utilise the high bandwidth provided by the next generation of communications networks. The Axon host architecture and network interface is designed to provide a path directly between the network and host memory, without any store-and-forward of data. A pipelined communications processor (CMP) serves as a network interface with direct access to host memory, capable of delivering bandwidth in excess of 1 Gbps to applications. This provides the ability to support demanding applications such as scientific visualisation and imaging.

Appeared in Protocols for High Speed Networks, II, Marjory Johnson (editor), Elsevier Science Publishers. 1991.

*Computer & Communications Research Center and Dept. of Computer Science, Bryan 307 – Box 1115, Washington University, One Brookings Drive, St. Louis MO 63130-4899

*On leave of absence from IBM Corp., Rochester Minn.

†This work is supported in part by Bellcore, BNR, Italtel SIT, NEC, Southwestern Bell, and NSF grant DCI-8600947

Axon Host–Network Interface Architecture for Gigabit Communications[‡]

James P. G. Sterbenz[†*] and Gurudatta M. Parulkar[*]
jps@wucs1.wustl.edu *guru@flora.wustl.edu*

[*]Computer & Communications Research Center and Department of Computer Science,
Bryan 405 – Box 1115, Washington University, One Brookings Drive,
St. Louis MO 63130-4899 USA
[†]On leave of absence from IBM Corporation, Rochester, Minnesota, USA

Abstract

This paper describes the design of the Axon host–network interface architecture, and performance factors determining its design. The Axon project is investigating an integrated design of host architecture, operating systems, and communication protocols to allow applications to utilise the high bandwidth provided by the next generation of communications networks. The Axon host architecture and network interface is designed to provide a path directly between the network and host memory, without any store-and-forward of data. A pipelined communications processor (CMP) serves as a network interface with direct access to host memory, capable of delivering bandwidth in excess of 1 Gbps to applications. This provides the ability to support demanding applications such as scientific visualisation and imaging.

1. INTRODUCTION

The continuing trends in the computer and communications fields involve increasing host processor performance along with the desire to use demanding applications such as scientific visualisation across emerging communication networks which can support increasingly high data rates (above 100 Mbps). It is becoming generally recognised that the bottleneck in delivering this bandwidth to applications is in the host–network interface.

The future generation of internetwork, consisting of high speed subnetworks and fast packet switches operating at or above 1 Gbps is referred to as the *very high speed internetwork*. The VHSI provides access to a number of large mass storage facilities which contain data and images obtained from computation such as simulations, finite element analysis, and molecular modeling, as well as from real-time data acquisition such as satel-

[‡]This work is supported in part by Bellcore, BNR, Italtel SIT, NEC, Southwestern Bell, and NSF grant DCI-8600947

lite telemetry and medical scanning. The VHSI also provides access to supercomputers and other specialised processors which run these simulation and modeling programs.

Existing architectures support distributed processing by message passing at the system level using a stack of network protocols and mechanisms to treat the network as an I/O device. The problems with this approach are the following:

- There is a lack of integration of hardware, operating system, and communication protocols. This results in considerable inefficiency and complexity for several reasons. The functionality of operating system and communication system modules are not optimised for one another, thus the interaction and interface between them is inefficient and complex. There is a lack of correspondence between network and host data objects (*e.g.* packets and pages), which results in inefficient control and synchronisation between the network and host (*e.g. per* packet processing and page fault handling) and unnecessary control transfer, data buffering, and reformatting.
- The network interface is treated like an I/O device, and therefore, *per* packet processing involves servicing interrupts, context switches, and data copying to protocol and I/O buffers. Furthermore, since I/O processors and interfaces are designed to handle a wide diversity of I/O devices ranging from character and unit record devices to high speed mass storage, performance is suboptimal for VHSI rate communication.
- There is no way to directly use the shared variables paradigm for IPC across a wide area network, which leads to performance compromises for applications naturally suited for data sharing.
- Many existing and proposed transport protocols are general purpose, and are not designed to perform well for various classes of demanding applications. General purpose error and flow control schemes are complex to implement in hardware and do not exploit the improved functionality of emerging high speed networks. Flow and congestion control mechanisms are less able to respond to changing network conditions as data rates increase, *i.e.* by the time adjustments are made, the conditions that induced the adjustment may have drastically changed.
- Communication is handled through front end network interface or communication processors, which are stored-program processors that manipulate packets in a store-and-forward manner, resulting in latency due to programmed operation and buffering of data. The network interface must also communicate with the host system using the standard I/O interface, which is not optimised for high speed communication (as mentioned above).

A new communication architecture for distributed systems has been proposed called **Axon** [14]. The primary goal of the Axon architecture is to support a high performance data path delivering VHSI bandwidth directly to applications. The significant features of Axon are: (1) an integrated design of host and network interface architecture, operating systems, and communication protocols; (2) a network virtual storage facility which includes support for virtual shared memory on loosely coupled systems [13, 15]; (3) a high performance, lightweight object transport facility which can be used by both message passing and shared memory mechanisms [16]; (4) a pipelined network interface which can provide a high bandwidth low latency path directly between the VHSI and host memory. Axon may be viewed as a *second generation* high performance host–network interface architecture, because of the emphasis on integrated design and the exploitation of the

enhanced performance and functionality provided by the VHSI.

This paper presents a description of the Axon host and network interface architecture. Other aspects of the Axon architecture are described in [16, 13, 15]. Section 2 provides an overview of the Axon architecture as background for this paper. Sections 3–5 describe the host architectures in support of Axon, the host–network interface architecture, and the design of the communications processor (CMP). Section 6 discusses Axon performance and the partitioning of function between hardware and software. Section 7 presents related work and Section 8 gives concluding remarks.

2. THE AXON ARCHITECTURE

This section provides a brief introduction to the Axon architecture, with emphasis on functionality that must be supported by the Axon network interface. First, IPC (interprocess communication) primitives are discussed within the framework of the VHSI environment. Then, a brief description is presented for the Axon system level IPC support and transport protocol. The host architecture, host–network interface, and communications processor (CMP) are described in subsequent sections.

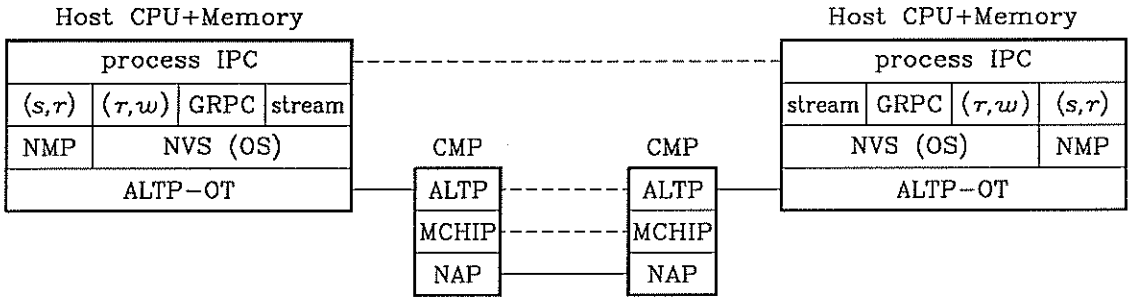

Figure 1: Logical Axon Protocol Hierarchy

2.1 IPC in the Axon architecture

A *logical* view of the Axon protocol hierarchy is presented in Figure 1. It is important to note that this is a logical view of functionality only, and does not imply that strict layering is adhered to in implementation. IPC is supported with shared variable *read/write* and message passing *send/receive* primitives. Axon supports a general form of remote procedure call, in which the code and data segments can be located on arbitrary and independent hosts, with execution specified for an arbitrary host, referred to as *generalised remote procedure call* (GRPC). Axon provides mechanisms to transfer *segment streams* at high bandwidth with low setup overhead to support the special demands of high performance visualisation and imaging applications. GRPC and segment streaming are described in [13, 15].

2.2 System level IPC support and NVS

The system level support for the various application level IPC paradigms is provided by two components: NVS and NMP. Message passing IPC is supported by the *network*

message passing (NMP) interface which performs relatively straight-forward transformations from application level primitives (*e.g. send* and *receive*) to corresponding transport level operations (*e.g.* send-message and receive-message).

Network virtual storage (NVS) is the system level shared memory interface for shared variables, GRPC, and segment streaming IPC. NVS extends the typical virtual storage mechanisms to include systems throughout the VHSI. A segmented programming model is used, with underlying paging to facilitate storage management, as in the Multics operating system [2] and descendents. NVS provides the ability to easily use the shared variables paradigm across the VHSI.

In addition to the desirable functionality provided by NVS, its motivation is based on the need to avoid the latency associated with the store and forward of packets in host systems. This means that at the receiving end packets must be placed directly into memory, and thus the host must be able to map them into the appropriate application address space to avoid copying the data. The extension of the normal virtual store mapping mechanisms in NVS provide an efficient and elegant method for doing this.

Segment types. Axon segments are of two types: *memory* and *video*. Memory segments are either *code* or *data* subtype and are divided into pages (and may be organised into segment groups for performance reasons). Video segments are either *text* or *graphics* subtype. Graphics segments are bit-mapped image frames; text segments correspond to a text window on a workstation. Video graphics segments are divided into scanlines, and may be organised into multi-frame images (*e.g.* a color image of R+G+B frames).

Segments have attributes of *read, write, execute*, indicating the type of access allowed. These access bits in the segment descriptor may differ from the (more restrictive) capabilities that individual users possess, or the descriptors of individual processes. Code segments are assumed to be pure (refreshable), and therefore always have access attributes of execute-only. Data segments may be readable and/or writable.

Data structures. NVS extensions allow segments to be addressed when resident on a non-local host. This is accomplished by including a *host id* field in either the virtual address or *segment descriptor table* (SDT) entry. When a segment fault occurs for a nonlocal segment (indicated in the segment descriptor), the dynamic address translation facility invokes the transport level to get a copy of the segment from the appropriate system. As the segment is returned, corresponding page and segment descriptor presence bits are set so that program execution can resume with the normal fault recovery mechanisms.

The local storage management data structures are extended to allow the addressing of segments on other hosts. This is accomplished by adding a *host id* field to the *known segment table*, which holds the symbolic segment bindings. The host id is an index into the *per* process *known host table*, which holds the symbolic host name to address/path bindings. This binding is resolved by searching the *host address table* for each host, which gets its binding by invoking an internet name server, using the *host name database*. There are also tables to assist in n-way IPC using multipoint connections. As in the local environment, the *segment id* of the virtual address is the index into the *segment descriptor table* (SDT). The *per* process SDT descriptor points to the *per* system segment descriptor in the *active segment table* (AST). Multiple processes share segments by sharing AST entries. The AST descriptor points to the base of the *page descriptor table* (PDT) for the

segment, with the *page number* of the virtual address giving the index to the appropriate page descriptor. The page descriptor contains the auxiliary storage page slot location and real storage page frame address. The *offset* portion of the virtual address is added to the page frame address to obtain the desired real address. Depending on the method used for network-to-host object mapping, a packet presence bit vector may be in PDT entries. An associative translation lookaside buffer provides the typical performance benefits in avoiding table lookup most of the time.

Storage management policies. NVS in Axon involves extensions and additions to storage management policies. The *fetch* policy is not affected by NVS, except that demand-segment implies a degree of anticipatory-page movement across the network (which is desired to counter end-to-end latency effects). The *placement* policy is not affected by NVS at all, since placement is trivial for paged storage management and unaffected by remote access to segments.

An entirely new policy, the *remote placement policy*, is used to determine where remote segments are placed while being used by the local system. These include real store, auxiliary store, a combination, and frame buffer placement, with a number of sub-policy options (swappable, nailed, *etc*). Due to the presence of segments from remote hosts, the *replacement* policy is affected. In particular, if real store remote placement is used, an entire segment's pages are placed in real store, some of which are likely not in the process locality set. This can have significant impact on the availability of real storage, and indicates that the estimation of working sets must consider local and remote segments differently. NVS and its storage management policies are described in more detail in [13, 15].

2.3 VHSI underlying model

It is assumed that the underlying network is *quasi-reliable* [16], *i.e.* the probability of errors is low enough that protocols are *success-oriented*, designed with errors as the exceptional case (but they still must be handled). The probability of bit errors is extremely low due to the reliability of fiber optic links and modern fast packet switches; the probability of packet loss and mis-sequencing is low due to the connection-oriented substrate providing resource reservation [10]. These assumptions lead to a number of simplifications in the transport protocol design and its efficient high performance implementation, as described in this paper.

2.4 Transport protocol

At the transport level, the VHSI model is best supported by a set of simple application-oriented lightweight transport protocols for various classes of applications [11, 16]. These transport protocols can have their *critical path* functions implemented in the VLSI *communications processor* (CMP). The critical path consists of the data path and routine *per* packet processing allowing data to flow at VHSI rate once an operation has begun.

The transport protocol that is used by Axon is designed to support IPC by the transfer of objects (especially NVS segments), referred to as *application-oriented lightweight transport protocol* for *object transfer* (ALTP-OT) [16]. ALTP-OT uses rate based flow control, where the rate specification consists only of parameters important to IPC, and

Table 1
ALTP-OT Operations

Connection	join-ipc	join or establish connection
	respecify-rate	alter rate specification
	leave-ipc	leave or terminate connection
Receive	get-segment	obtain segment copy
	acquire-segment	acquire segment authorisation and locks
	get-page	obtain page (segment must be acquired)
	get-copy	obtain permanent segment copy
	get-stream	receive segment stream
	receive-message	receive IPC message
	retransmit-packets	selective retransmission request
Transmit	release-segment	release segment and return modified pages
	release-page	release and return modified page
	remote-execute	initiate remote process execution
	send-copy	send permanent segment copy
	send-stream	transmit segment stream
	send-message	send IPC message
	invalidate-segment	invalidate remote copies

efficient end-to-end error control, optimised to include only what is necessary for object transfer. The ALTP-OT requests and operations are listed in Table 1.

Packet structure and format. Information is transferred throughout the VHSI in packets. A group of packets corresponding to a single ALTP-OT semantic action is a *super-packet*, consisting of an initial control packet (which may also contain data), and optionally followed by data packets. The data packet format is shown in Figure 2.

Each data packet π_i corresponds to a fragment π_{d_i} of a page p_j of a segment s_k ($|s_k|$ pages long) of a segment-group g (containing $|g|$ segments), which are part of the superpacket σ. In the case of a video-graphics segment a page corresponds to a scanline, a segment to a frame, and a segment group to a complete image. The packet index fields ijk mark the location of the packet in the segment. The limit fields $|s_k||g|$ provide the length of each segment in the group and the group size, which is used for local storage allocation for the returning segments, as well as for bounds validity checking.

Identify each packet as $\pi_i p_j s_k \in g \subseteq \sigma$, indicating the i^{th} packet of the j^{th} page of the k^{th} segment in segment group g, corresponding to superpacket σ. The structure of a super-packet (with k segments in the group) is then:

$$
\begin{array}{llll}
\pi_0 p_0 s_0, & \text{[optional control packet]} & & \\
\pi_1 p_1 s_1, & \pi_2 p_1 s_1, & \cdots & \pi_{|p|} p_1 s_1, \\
\pi_1 p_2 s_1, & \pi_2 p_2 s_1, & \cdots & \pi_{|p|} p_2 s_1, \\
\vdots & \vdots & & \vdots \\
\pi_1 p_{|s_1|} s_1, & \pi_2 p_{|s_1|} s_1, & \cdots & \pi_{|p|} p_{|s_1|} s_1, \\
\\
\vdots & \vdots & & \vdots \\
\pi_1 p_{|s_k|} s_k, & \pi_2 p_{|s_k|} s_k, & \cdots & \pi_{|p|} p_{|s_k|} s_k
\end{array}
$$

field	subfield	
MCHIP	data	
	type	
ALTP	data	
	type OT	
connection id		c
request id		q
segment/frame	limit	$\|g\|$
	index	k
page/scanline	limit	$\|s_k\|$
	index	j
packet	index	i
data		π_d
checksum		Σ

Figure 2: Data Packet Format

Every data packet is sufficiently self-describing (connection id, request id, index, and limit fields) so that the CMP is able to place them directly into the proper host storage location regardless of packet arrival sequence and request and connection multiplexing.

In the case of a segment transfer (*e.g.* to satisfy a remote segment fault), a super-packet consists of the entire segment. The correspondence between host and network objects, and the resulting correspondence of control (*e.g.* segment fault resolution and super-packet processing) provides substantial performance benefits, in terms of reduced overhead of data buffering/reformatting and control synchronisation. For example, assuming a continuous data rate of 1 Gbps and packets based on ATM cell size of 53 bytes transporting 32 data bytes each, the transfer of a stream of 1MB segments will involve the host processing of super-packets every 13.9ms (32K packets/segment), rather than of packets every 424ns. Additionally, super-packet processing corresponds to segment fault processing, and therefore requires no additional CPU interrupts and context switches.

Flow control. ALTP-OT uses rate based flow control. When ALTP-OT opens a connection, it specifies attributes of the connection in terms of parameters such as average and peak bandwidth, and a factor reflecting the burstiness of the transmission. These parameters are used by all the intermediate systems, including various packet switches and gateways, as well as the endpoint hosts that the connection goes through, to make appropriate buffer and resource reservations. The rate specification is negotiated between ALTP-OT and the internetwork/network layers, to ensure that the requested rate does not exceed the capacity of internal network nodes (packet switches, gateways, and subnetworks). Furthermore, any adjustments to the rate specification should be infrequent, based on long term changes in application demands. It is assumed that the internet level below has the functionality to support connections with specified bandwidth requirements, and furthermore, that the probability of packet loss, errors, and resequencing is very low, which is referred to as *quasi-reliability*.

This results in very simple flow control at the host–network interface, involving clocking packets at the specified rate, which can realistically be designed into VLSI hardware. As long as both ends transmit subject to the rate specification, the probability of packet loss within the VHSI due to buffer overruns is very low. Since the internet level is responsible for resource allocation, ALTP-OT is not concerned with congestion control, further simplifying flow control and the network interface. Error control is decoupled from the rate based flow control, which allows considerable simplification as described below.

Error control. In the VHSI environment error control is performed, as much as possible, on an end-to-end basis, and is decoupled from flow (rate) control, as described above. The ALTP error control is as simple as possible, based on application characteristics; the packet handling is:

- duplicate packets are discarded
- corrupted packets are discarded with application based selective retransmission
- missing packets (timer detected) are selectively retransmitted (application based)
- packet sequence is irrelevant (see below): *sequence by placement*

Note that due to the orientation of ALTP-OT to object transfer, the handling of duplicate and out-of-sequence packets is considerably simpler and more efficient than would be the case for a general purpose transport protocol. Since data packets have sufficient header information to indicate the connection and request, and are placed directly into the proper location of target store, the overhead of sequence buffering is eliminated. The simplified error control of ALTP-OT can be efficiently implemented in VLSI hardware.

Retransmission strategies. Several options exist for the retransmission of packets: *granularity* of retransmission and timer values, retransmission *fetch policy*, and *preemption* by the retransmission.

- **granularity:** The granularity of retransmission refers to how many missing packet events are accumulated before a request for retransmission is made. Due to the knowledge of the super-packet structure of segment (groups) by ALTP-OT, a rich set of options can be exploited, that are based on the granularity of the data structure transmitted: *packet*, *page*, *segment*, and segment *group*.
- **fetch policy:** The retransmission strategies can be classified by whether packets are always requested for retransmission, or only if a page is referenced that contains them. If all packets corrupted or missing are retransmitted, this corresponds to *anticipatory retransmission* thus anticipating the future reference of all missing packets. In this case the timers indicate *when* a packet retransmission request should be made. If the only packets retransmitted are those corrupted or missing which are part of a page actually referenced, the policy is *demand retransmission*, and assumes that a number of packets in the segment will not necessarily ever be referenced. In this case, the timers indicate *how long to wait* before a referenced packet is assumed to be missing.
- **preemption:** Since error control is *in-band*, packets retransmitted use the same connection and allocated bandwidth as the primary data stream. The alternatives are to allow all of the original request to flow before any of the retransmission requests are serviced resulting in a *non-preemptive* policy, or to *preempt* the primary data stream and immediately retransmit.

The number of possible strategies is the cross-product of these orthogonal sub-policies: granularity, fetch, and preemption, *e.g.* a reasonable strategy is to retransmit a page of packets only when the page is referenced, and preempt the primary data stream.

Lower layer protocols. The underlying internet/network layers of function are provided by a *multipoint congram-oriented high performance internet protocol** (MCHIP) [9, 10], and *network access protocols.*

3. AXON HOST ARCHITECTURE

This section describes the Axon host architecture configurations. High performance computer systems typically consist of one or more central processors (CPU), which communicate with memory banks (M) and I/O processors (IOP) through an interconnection network (as shown in Figure 3 excluding the dashed-line boxes). In addition, various caches ($) may be present to utilise fine-grained locality and perform speed-matching of data rates. Note that CPU blocks may represent special purpose processors or coprocessors (such as array processors or simulation engines), as well as general purpose instruction processors. Additionally, the memory system may consist of a multi-level hierarchy including extended memory (EM) for high performance backing store. Communication is typically handled by front-end communications processors or network interfaces (NI), which use the I/O interface to the host system. The data stream is thus subject to the delays of both the network interface and I/O processor, as well as the additional operating system instruction path length and context switching overhead for each. In addition, since the IOPs are designed to handle a wide diversity of I/O devices ranging from slow unit record and character devices to high speed mass storage, it is likely that IOPs will not perform optimally for high VHSI rate communications (if at all).

For Axon to support communications in the VHSI environment, it is necessary to provide high bandwidth low latency data paths directly to memory, motivating modifications to current host architecture. In particular, the most significant requirement is that the objects being communicated are moved between host memory without any intervening store-and-forward hops. This is necessary to avoid extra latency, large amounts of buffer space, and the complexity of buffer management. Two configurations of host architecture meet these requirements.

3.1 Interconnect interface architecture (IIA)

The first Axon host architecture gives the CMP (communications processor) a relationship to the system similar to that of IOPs, interfacing directly to the processor–memory interconnection network. This is referred to as *interconnect interface architecture* (IIA), and is presented in Figure 3 (with the inclusion of the dashed box labeled IIA). In addition, an interconnection between CMPs and IOPs should be provided to allow direct, high-speed transfers between the VHSI links and I/O controllers (IOC) or devices (which

*A congram combines the desirable features of a datagram with those of a (soft) connection. For the purposes of this paper, it can be thought of a connection with the added attributes of rapid setup and survivability in the presence of network failures.

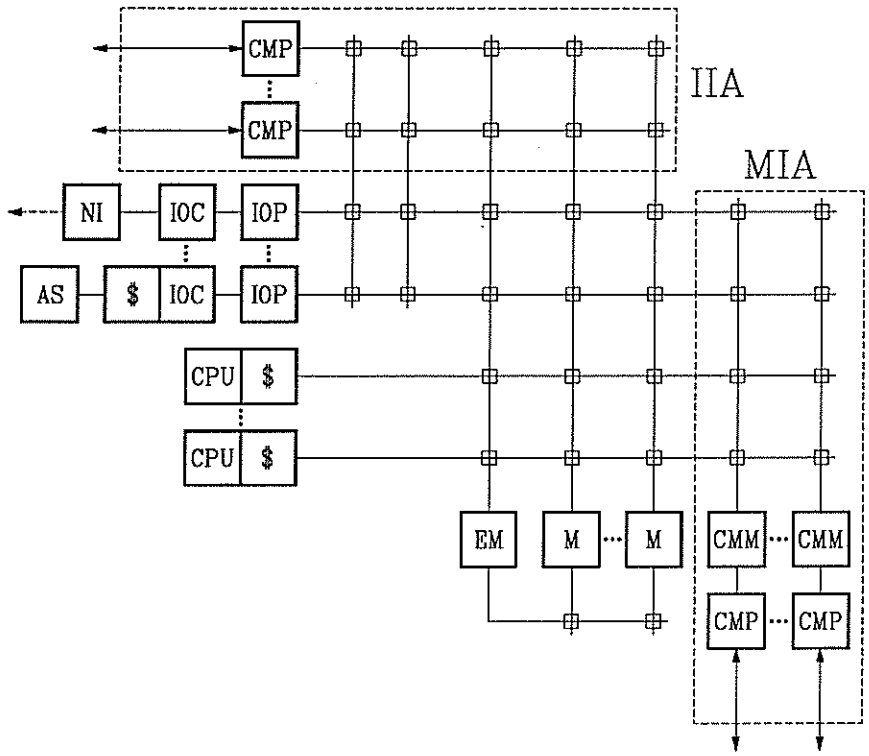

Figure 3: Axon Host Architecture

provides the path to auxiliary backing storage – AS). Note that the interconnection network is depicted as a crossbar for simplicity, but the actual structure will vary based on the particular host system architecture. Axon only imposes the requirement that the interconnection be rich enough to allow the added CMP connections, and has enough performance to sustain the additional VHSI communication traffic without significant blocking.

3.2 Memory interface architecture (MIA)

The second Axon host architecture interfaces the CMPs to a special multi-ported *communications memory module* (CMM), similar in concept to VRAM (video-RAM) design. This is referred to as *memory interface architecture* (MIA), and is presented in Figure 3 (with the inclusion of the dashed box labeled MIA). The CMM has a conventional random access port which appears like any other memory bank to the processor–memory interconnect, out of which the CPU may execute code and access data. The other ports are high speed sequential access interfaces to the CMP (transmit and receive), and must operate at a rate of the VHSI optical links ($\div$ CMP datapath width).

3.3 Comparison of IIA and MIA

This section will discuss some of the tradeoffs affecting the choice of host architecture.

Remote segment placement. In IIA, the interface is uniform to all memory modules in the real address space, and to IOPs giving access to auxiliary storage. Thus, segments fetched across the network can be easily placed anywhere in real storage, auxiliary storage, or both (corresponding to the NVS remote segment placement policy).

In MIA, however, all memory modules are not directly available to the CMP. Segments received are placed in the CMMs connected directly to the CMP that has received them. Segments to be transmitted must be present in the appropriate CMMs. This creates a partition of the real address space $R = \{M_1, \ldots, M_n; CMM_1, \ldots CMM_m\}$ and makes memory access somewhat more difficult, particularly if there are multiple interleaved CPU memory modules. Additionally, the fraction of real store devoted to CMM must be determined based on communication requirements, and remote segments must be clustered in these memory modules.

Blocking of traffic. In IIA, communication traffic uses the main host interconnect and is therefore subject to blocking based on congestion due to traffic between memory (M) and CPUs or IOPs. Since the CMP does not buffer packets, two approaches may be taken: blocked packets are dropped or CMP traffic blocks local interconnect traffic. Either method is acceptable if the blocking probability is sufficiently low, and the latter has the advantage of requiring fewer packet retransmissions.

In MIA, the CMP–CMM interface is designed such that communication traffic can proceed at VHSI data rates with no blocking using the sequential CMM ports.

Pragmatics. The IIA dictates a rich, complex host interconnect structure, and may require some redesign of current host hardware architecture. The interconnect must support the additional connections and additional traffic for each CMP attached.

The MIA requires little redesign of host architecture, other than dealing with the physical address configuration of M and CMM for proper real address space partitioning and memory interleaving. The MIA does, however, require a completely new memory design, specifically the high performance multi-ported CMM, which must support the two simultaneous asynchronous sequential ports attached to the CMP, along with a random access port attached to the processor interconnect.

Additionally, in MIA the connection between CMPs and IOPs needs to be treated as a special case. Segments could be staged through the CMM, but this violates the principle of avoiding store-and-forward of data. The alternative is to cut-through on a CMM bypass path into the host interconnect. Note that if a cut through is used, the blocking issues discussed in the context of IIA must be considered.

4. HOST–NETWORK INTERFACE

This section gives a brief description of the organisation of the Axon host–network interface design for an MIA (memory interface architecture) host.[†] A block diagram is presented in Figure 4. The interface is bidirectional, but the functions have been labeled for communication in the left-to-right direction for clarity.

[†]Note that this does not imply that MIA is superior to IIA, but rather that MIA is chosen as the example for this paper.

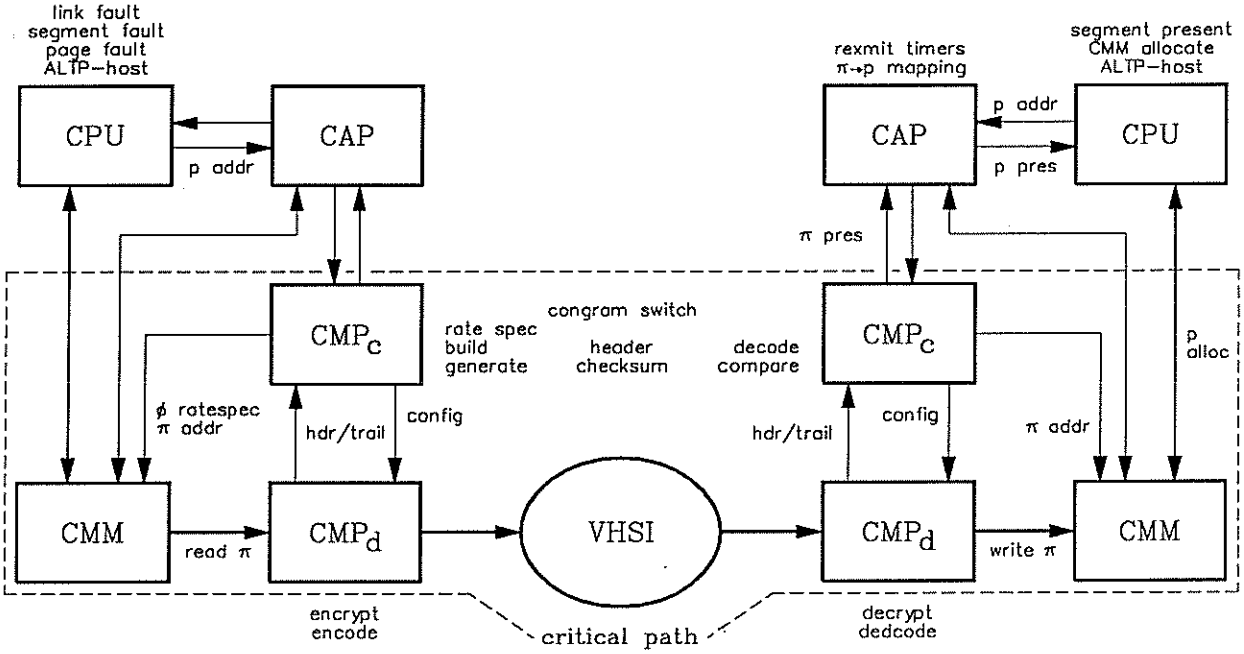

Figure 4: Axon Host–Network Interface

4.1 Architecture and organisation

The ALTP-OT critical path, consisting of the data path and *per* packet (π) processing, is implemented in the CMP (communications processor). The CMP consists of datapath (CMP_d) and control (CMP_c) portions. The CMP datapath interfaces to the VHSI optical links and the sequential ports of the CMM (communications memory module), and performs such functions as encryption/decryption and format conversion (encode/decode). The CMP control functions are those directly related to the datapath such as header build/decode, checksum generate/compare, CMM address generation, rate specification timing, as well as the *per* packet congram multiplexing and control.

The CMM is a multiported memory, with sequential ports connected to the CMP transmitting and receiving data paths, and the random access port available to the host CPU for program execution.

A high performance microprocessor, the *CMP assist processor* (CAP), performs functions that are not part of the critical path, but require high performance that would be inadequately provided by the host CPU and would adversely impact the performance of other host processes.[‡] The CAP is responsible for building control packets and passing them to the CMP for transmission and checksumming. Similarly, control packets received by the CMP are passed to the CAP for full decoding and subsequent action, which may involve interaction with the host CPU. The CAP is involved in the timer management for error packet retransmission, and in the packet arrival to page presence mapping ($\pi \rightarrow p$), in particular setting page and segment presence and subsequent host notification for fault recovery.

The host CPU is responsible for link/segment/page fault handling (NVS), and *per* congram functions (ALTP-host) directly corresponding to application requests.

[‡]The CAP was omitted from Fig. 3 for clarity.

4.2 Functional partitioning

A key issue in the implementation of high performance architectures such as Axon is to determine the proper partitioning of function between the critical and non-critical path.

Some function, such as the data path from network to memory and associated *per* packet control is clearly part of the critical path and must reside in CMP hardware, to support small packets at high data rates. For example, it is unreasonable to expect a host CPU to process incoming and outgoing packets every 424ns (ATM cell size at 1 Gbps), and to concurrently execute the application needing this bandwidth. It should be noted that the design of the simple critical path can remain constant even if control functions need optimisations that affect non-critical path software in the CAP and host CPU.

Other control functions may or may not need to be (fully) part of the critical path, depending on the data rate and time–space complexity tradeoffs. Examples include packet retransmission timers and host–network object mapping (packet arrival to page and segment presence). Parallelism in the data path can be used to provide a speed advantage, allowing higher data rates for a given CMP clock cycle. In addition, pipeline delay is used to allow control functions the necessary time to operate at high data rates.

The partitioning of non-critical path functions between the host CPU and CAP is also important. This is dictated by the desire to avoid host interaction with the communication operation that is not directly related to application execution. Thus, it is reasonable to expect the host to initiate an ALTP-OT segment transfer as a result of a segment fault; the application process has already been interrupted and a context switch taken to system state. But the host CPU should not be involved in the protocol processing until all of the packets in the first addressed page of the segment have arrived and the suspended process can resume.[§] The CAP handles all of the asynchronous events that would otherwise cause the CPU to be interrupted and suspend other processes, reducing application efficiency.

The determination of critical path control function implementation involves a time–space complexity tradeoff, and will be discussed in Section 6.

5. COMMUNICATIONS PROCESSOR

This section gives a high level functional description of the CMP (communications processor) for an MIA host, and an operational example for an ALTP-OT request.

The goals for the design of the CMP include the ability to perform critical path functions in real time with no packet buffering and to incorporate the necessary function in VLSI. This may be realised by organising the CMP as a dynamically reconfigurable pipeline, based on the ALTP type and options for a particular congram. The pipeline organisation allows packets to be processed at the VHSI data rate.

The CMP (Figure 5) consists of a set of datapath modules and control modules. The datapath modules perform manipulation and transformation on packets as they pass between the point-to-point VHSI optical links and the CMM (communications memory

[§]This is similar to the motivation for relieving the CPU of programmed I/O by providing IOPs.

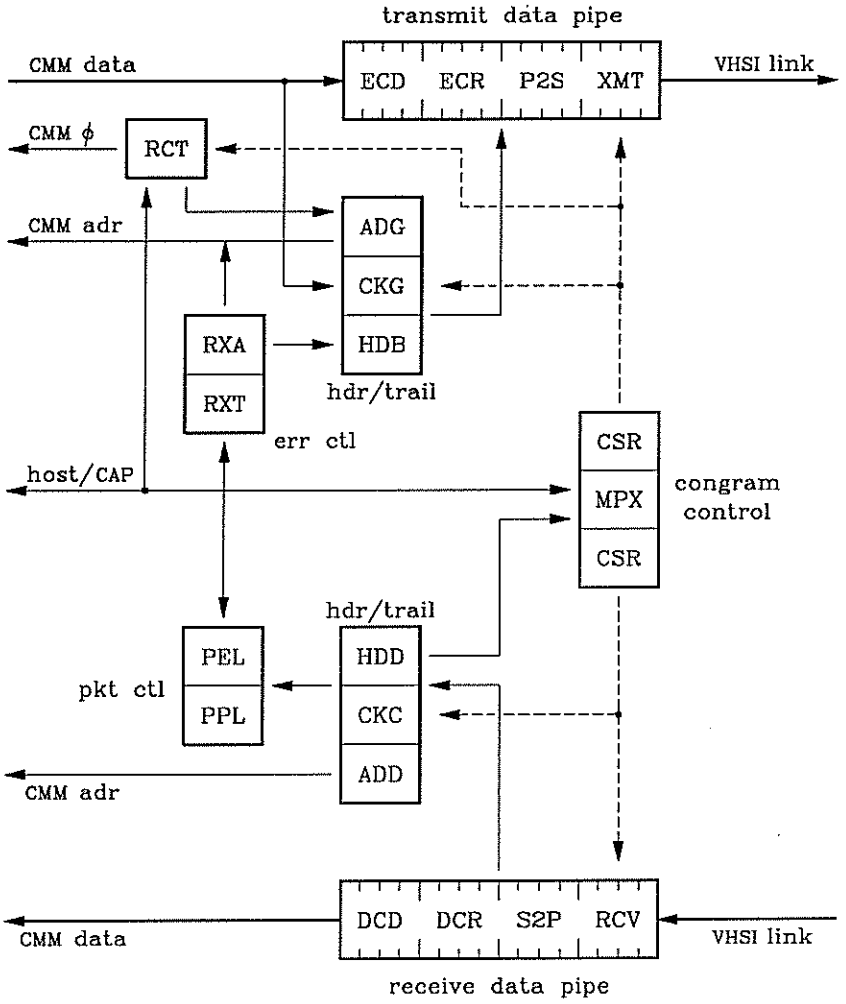

Figure 5: Communications Processor Block Diagram

module) sequential access ports, without buffering (except for the pipeline delay).

5.1 Congram control

The multiplexing of congrams (which may be considered to be soft connections) is handled by the *congram control logic*:

MPX – MULTIPLEXING control logic performs the hardware context switching of the CMP in response to transmission requests and received congram ids.

CSR – CONGRAM STATE REGISTERS hold all of the state information for each active congram, to allow rapid control and pipeline configuration changes for multiplexed congrams using the CMP. There is a set of CSRs for both the transmitting and receiving side of the CMP.

5.2 Packet data paths

The *transmit data pipe* and *receive data pipe* are the main data paths of the CMP. The transmit data pipe modules are:

ECD – ENCODE performs any required transformations to the data to correspond to

internetwork data format standards and accommodate heterogeneous hosts (such as byte ordering).

ECR – ENCRYPT performs data encryption of the data and internal control fields of the packet as it passes through the data pipeline.

P2S – PARALLEL-TO-SERIAL datapath conversion reduces the ω bit wide data path for transmission on the bit serial network link.

XMT – TRANSMIT performs the line coding and transmission functions for the optical transmitter.

The receive data pipe modules are:

RCV – RECEIVE takes the bit stream derived from the optical receiver, eliminates line coding, and derives the clock for the receive data pipeline.

S2P – SERIAL-TO-PARALLEL datapath conversion forms an ω wide data path to give a speed advantage and octet access to packet header/trailer fields for CMP control and data pipeline manipulation.

DCR – DECRYPT performs data decryption of the data and internal control fields of the packet as it passes through the data pipeline.

DCD – DECODE performs any required data transformations to the data from internetwork data format standards to the local host format.

5.3 *Per* packet control

Associated with the transmit data pipe are control modules:

RCT – RATE CONTROL uses the rate specification r_i for each congram i to determine the timing of data reads from the CMM, and thus the rate at which packets are clocked out of the transmit pipeline for each congram. The r_i values for each congram are obtained from the corresponding CSR.

ADG – ADDRESS GENERATE uses the initial CMM address of each page to form the addresses for each packet read from the CMM.

CKG – CHECKSUM GENERATE sums the packet data fields as they pass through the data pipeline. The computed checksum is then inserted in the packet trailer.

HDB – HEADER BUILD uses the congram id and control information from the corresponding CSR to build the header for the packet. All protocol levels of encapsulation (ALTP, MCHIP, and network) are done at one time. The congram and request ids (c, q) and packet id (segment id k, page number j, packet number i) are inserted into a header template.

Associated with the receive data pipe are control modules:

HDD – HEADER DECODE determines the congram id for CMP configuration, and determines whether the packet type is control or data. Control packets are passed to the CAP. For data packets, HDD determines the packet address in CMM from the packet index (ijk) and the base address of the page from the corresponding CSR. The congram and request ids (cq) are used by MPX to select the appropriate CSR.

CKC – CHECKSUM COMPARE sums the packet data fields as they pass through the data pipeline. The computed checksum is then compared with the actual checksum in the packet trailer. If a mismatch is found, the PPL and PEL (packet presence and error logic – see below) are notified to indicate that the packet has been discarded after initial receipt.

ADD – ADDRESS DECODE uses the initial CMM address of each page (from the CSR), and the packet index (ijk) to form the CMM address for the writing of each packet into CMM.

The *packet control* logic is responsible for recording packet arrivals and missing or corrupted packets.

PPL – PACKET PRESENCE LOGIC keeps track of packet arrival to allow the CAP and host to determine the presence of complete pages and segments, so that the appropriate page descriptor table and segment descriptor table presence bits can be set, and host CPU application resumed.

PEL – PACKET ERROR LOGIC keeps track of corrupted (from CKC) and missing (from RXT) packets so that retransmission requests can be made, and also invalidates corrupted packets to the PPL.

The *error control* logic is responsible for generating the appropriate selective retransmission requests and packet addresses at the receiving end, and retransmitting packets on the sending end.

RXT – RETRANSMIT TIMERS determine (in conjunction with the CAP) when packet retransmission requests should be made, based on the retransmission policy. Timer values are accumulated to the proper granularity (packet, page, segment, or group). The fetch and preemption options are then used to determine when the retransmit-packets control packet should be sent, and the PEL is used to construct the retransmit packet bit map.

RXA – RETRANSMIT ADDRESS generates the (local) CMM addresses for packets to be retransmitted, using the base address of the corresponding page and the retransmit packet bit map in the retransmit-packets control packet received.

5.4 Example operation

This section describes the network interface processing to handle a remote segment fault initiated by NVS, to indicate how the various blocks of the CMP and network interface interact in an operational manner. A description of a similar process concentrating on the NVS and ALTP-OT host level operations is presented in [14]

When a segment fault occurs, NVS determines if the request is for a local segment. If not, a copy of the segment must be fetched from a remote host, and an ALTP-OT get-segment operation is initiated. The host CPU resident ALTP-OT transfers control to the CAP, passing the segment name, remote host address, and other necessary information. The CAP then builds a get-segment control packet by inserting this information into an appropriate template in its local memory. The control packet is then passed to the CMP which sends it to the VHSI *via* its serial output link.

At the remote end, the CMP header decode logic determines that it has received a

control packet, and passes it to its CAP. The control packet checksum is computed, and if a mismatch occurs, the CAP is notified so that the request can be rejected. The CAP fully decodes the control packet, making the determination that a link fault is required to locate and authenticate the get-segment request. It passes this information to the host CPU using an interrupt. The host CPU and operating system use the normal mechanisms to locate, authenticate, (if necessary) lock, and page the segment into real store. The CAP is then notified of the segment location in terms of the base page addresses. The CAP passes these to the CMP, which stores them in the congram state registers (CSR) for the corresponding request. Using the base page addresses, the CMP reads from the sequential output port of the CMM in packet quantities. The header build logic inserts the congram and request id from the CSR, as well as the page and segment id jk, and increments the id for each packet i. The rate control logic is used to clock the packets out of CMM based on the rate specification for the congram. As the packet passes through the data path, a checksum is computed, which is inserted into the packet trailer. Furthermore, data encryption and format conversion are performed if required. After the entire page has been read from the CMM, under control of the congram multiplexing logic, and subject to the rate specifications of active congrams, a CMP hardware context switch may take place to output a page from another congram.

At the local end, storage has been allocated for the returning segment, based on the estimated segment size. The data packets contain the actual segment size, which will be used to adjust the allocation if necessary. A segment superpacket consists of a sequence of data packets, all of which are completely self-describing by congram and request id (cq), and packet id (ijk). When a packet enters the CMP input from the VHSI link, the header decode logic uses cq to switch to the corresponding CSR. This contains the base address of the pages for CMM, allowing the address decode logic to compute the appropriate addresses so that the packets can be piped directly into the CMM input port. The packet presence logic is used to determine when entire pages are present, and signals the CAP to interrupt the host CPU to set page presence and recover from the page fault. Timers and counters in the CAP and CMP are used to determine when a packet is to be considered missing, and this causes the CAP to build and initiate a retransmit-packets request to the remote host. The CAP determines when the entire segment has arrived based on presence of all its pages, and the CSR information corresponding to this request may be freed for reuse. The CAP then notifies the host system that remote segment fault recovery is complete.

6. ANALYSIS

This section provides a framework to evaluate the design and performance of the Axon host–network interface architecture. First, the required performance measures are described. Then, a criterion for the partitioning of functions between the critical and non-critical path is discussed. Finally, the performance of Axon in terms of the time to access remote storage is discussed with respect to the operations involved and functional partitioning.

6.1 Performance measures

High performance is characterised by high data rate and low latency, with sufficient predictability. The data rate of a connection R is constrained by the minimum data rate r_i of components along the connection path: $R = \min(r_i)$. The total end-to-end latency D is the sum of the delays d_i through the various components: $D = \sum d_i$. The requirements for a high performance host–network interface will now be considered in terms of date rate (clock cycle) and latency (delay). These parameters are shown in Figure 6.

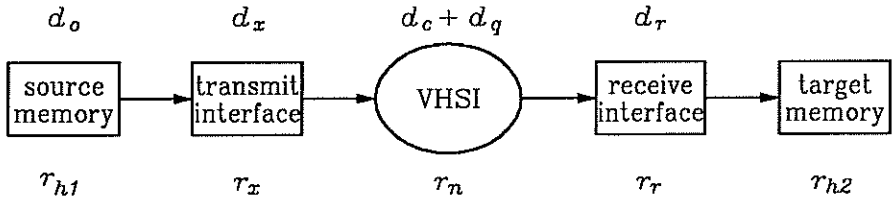

Figure 6: Interface Performance Parameters

Network interface clock cycle. Define the maximum data rate for transmission by $R = \min(r_{h1}, r_x, r_n, r_r, r_{h2})$ with components: the source host rate r_{h1}, the rate of the transmitting network interface r_x, the VHSI (network) rate r_n, the rate of the receiving network interface r_r, and the target host rate r_{h2}.

Assume that the rates of the hosts and VHSI are matched: $r = r_n = r_{h1} = r_{h2}$. This means that the network interface must be able to communicate at a data rate $r = 1/\tau$ [bit/sec]. Parallelism in the CMP data path provides a speed advantage; for a data path ω bits wide the required major cycle time is $\tau_\omega = \omega\tau$. Only the serial ports and serial/parallel conversion logic is required to operate at the minor cyle τ.

Table 2 shows the required CMP cycle time τ_ω for various required data rates r and datapath widths ω (which are multiples of 8 to allow octet access to packet fields). Cycle times on the order of 5ns are reasonable for a single chip VLSI CMP implementation in emerging CMOS or GaAs technology.

Network interface delay. Define the total end-to-end transmission delay of an object o by $D_o = d_o + d_x + d_c + d_q + d_r$ with components: the object transmit time d_o, the CMP pipeline delay at the transmitting network interface d_x, the speed-of-light latency d_c, the subnetwork and gateway queuing delay d_q, and the CMP pipeline delay at the receiving network interface d_r. Note that this excludes the latency of retransmitted packets.

The speed of light delay d_c is a function of the path length and the velocity of light in fiber. It is the responsibility of the VHSI to minimise queueing delay d_q. It is assumed that the VHSI provides guarantees on d_q through the use of statistical resource reservation by congrams, and therefore the endpoint hosts need not be concerned with d_q.

The object transmission time d_o is a function of the object size $|o|$ [bits] and data rate $1/\tau$. The object transmit time for some example object sizes at 1 Gbps is shown in Table 3.

Note that the actual page size is dictated by a particular host architecture, and the segment size by the application; these numbers give an idea of typical cases. The packet

Table 2
CMP Cycle Times

r	ω	τ_ω
1Gbps	8b	8.0ns
1Gbps	16b	16.0ns
1Gbps	32b	32.0ns
10Gbps	32b	3.2ns
10Gbps	64b	6.4ns
10Gbps	128b	12.8ns

Table 3
Object Transmit Times

	o	$\|o\|$	d_o
bit	b	1b	1ns
byte	B	8b	8ns
packet	π	48+5B	424ns
page	p	1KB	13.5μs
segment	s	1MB	13.9ms
		1GB	14.2 s

size is given as ATM cell size to give an *extreme* lower bound on packet size, and allows 32B of application data and 16B of header in the cell. A particular implementation of Axon may use a larger packet size, the upper bound being the smallest page size of any Axon host in the VHSI.

The overall goal is to minimise D_o, and the Axon architecture must minimise $d_{xr} = d_x + d_r$, which are the CMP interface delays between the VHSI boundary and host memory addressable by the application. Note that as d_q, d_x, and d_r scale downward, d_c dominates. Also note that if $|o|$ scales up with application demands along with host CPU power and memory size, d_o may remain a significant part of the latency.

By assuming a latency component consisting of the queuing delay and speed of light delay $d_{cq} = d_q + d_c$ as a constant for various kinds of networks (LAN, MAN, WAN, internetwork), a bound on the acceptable network interface delay can be determined [12]. A LAN will have the lowest latency, and provides the lower bound. To support a LAN with latency $d_{cq} = 10\mu$s, the required host interface latency is then $d_{xr} \leq O(10\mu\text{s})$.

The number of acceptable stages in the CMP data pipeline can be computed as $n_{xr} = d_{xr}/\tau_\omega$. Assuming an interface with octet wide data paths ($\omega = 8$), this allows the pipeline to be at most 625 stages long at each end, which is far more than will ever be needed by the CMP. This analysis shows that the Axon CMP can be designed to match even the low latency of a high performance LAN.

One of the performance targets of the Axon architecture is to support high bandwidth interprocess communication with low latency, specifically sub-second round trip delay (including object transmission). An internet will have the highest latency component d_{cq}, on the order of 200ms across an intercontinental internetwork [12], or a round trip delay of 400ms. The CMP pipeline delay is constrained by LAN latencies to $d_{xr} = 10\mu$s, as described above. At 1 Gbps, a 1 MB segment will have $d_o = 13.9$ms, from Table 3. This gives a total $D_o < 1$s, and indicates that given a sufficiently fast CMP and VHSI with highly reliable transport (requiring few retransmissions) the potential exists for applications to achieve good interactive performance across long haul networks, even when the transfer of large objects is involved.

6.2 Functional partitioning

A key issue in the implementation of the Axon architecture is to determine the partitioning of function $\mathcal{F}$ between the critical and non-critical path, which involves a time–space complexity tradeoff.

Time complexity. The impact on time complexity of a control function is the time taken for a hardware implementation (in CMP clock cycles) *vs.* the time taken for a software implementation (in host CPU or CAP instruction cycles).

For the host CPU and CAP, the time that control function C_i takes to complete t_i is based on the instruction cycle times t_{CPU} and t_{CAP}, and the number of instructions required on each $m_{i\text{CPU}}$ and $m_{i\text{CAP}}$, as well as extra overhead $m'_{i\text{CPU}}$ such as context switches and system calls. Thus in the worst case, with no overlap between the CAP and CPU: $t_i = m_{i\text{CAP}} t_{\text{CAP}} + (m_{i\text{CPU}} + m'_{i\text{CPU}}) t_{\text{CPU}}$

For the CMP, define the *minor cycle* τ to be the inverse of the serial data rate on VHSI communications links, (*e.g.* for 1 Gbps, $\tau = 1\text{ns}$). Define the CMP *major cycle* τ_ω as the clock cycle internal to the CMP within the parallel data path ω bits wide. Thus, an octet wide data path gives a CMP clock cycle of $\tau_\omega = 8\text{ns}$, which is feasible with current technology, since the operations performed within each CMP pipeline stage are fairly simple. If a slower clock is desired, the datapath width can be increased, but this trades against area complexity as described in the next section. Given n_i stages of pipeline delay for a particular control function C_i, the time it takes is $\tau_i = n_i \tau_\omega$. Thus, the *critical path time savings* for C_i is $\text{CP}_i = t_i - \tau_i$.

Note that by implementing a control function in the critical path, there may be an associated cost in increased latency through the CMP $\tau_\omega \Delta n_i$, due to Δn_i more pipeline stages required for C_i. The discussion in the proceeding section has indicated, however, that the latency bound on pipeline stages is extremely high, and therefore this is not of significant concern.

Space complexity. For the host and CAP, space complexity consists of memory used for software implementation of the function. This will be assumed to be a sufficiently small fraction of total memory that it will be ignored for the purposes of this paper.

For the CMP, space complexity has two measures: chip area and off-chip interconnect lines. Define a_i as the area required to implement C_i. The constraint to be met is that the sum of all the control and datapath functions implemented on the CMP must not exceed the available chip area for a given process technology: $\sum_i a_i \leq a_{\text{CMP}}$. The datapath width ω is a significant contribution to a_{CMP}, which limits the achievable speedup factor mentioned in the context of time complexity.

Note that although power issues are also of concern, for a complementary logic family power dissipation is related to the clock frequency and transistor count (related to a), and is therefore indirectly reflected by the area complexity.

The other space complexity measure is the number of off-chip interconnect lines (pinout using conventional packaging techniques). Define l_i as the number of off-chip interconnect lines needed to implement control function C_i on the CMP. Thus, l_i is constrained by the threshold of available interconnect on a chip using a given packaging technology: $\sum_i l_i \leq l_{\text{CMP}}$. The datapath width is also a significant contribution to l_{CMP}, since there will be 2ω data lines between the CMP and CMM (read/transmit and write/receive paths).

Thus, in determining if a function should be in the critical path, the tradeoff is in time saving CP_i, *vs.* acceptable chip complexity (a_i, l_i). This criterion is being used to determine a good functional partitioning in the Axon implementation. Some function,

such as the data path from network to memory and associated *per* packet control is clearly part of the critical path and must reside in CMP hardware, to support small packets at high data rates. Other control functions may or may not need to be (fully) part of the critical path, depending on these tradeoffs. Examples include packet retransmission timers and host–network object mapping (packet arrival to page and segment presence). This analysis is being supplemented by detailed simulation studies.

6.3 Remote storage access

Given that Axon provides a virtual shared memory view of the VHSI, the time to gain access to remotely referenced objects is of prime concern. The performance metrics of principal interest to Axon are:

T_s: the time that a process is blocked due to a remote segment fault (which is is resolved when the local system may begin executing the first page referenced in the segment).

T_p: the time that a process is blocked on subsequent page faults, which will be the case if a page is referenced before it is present as part of the arriving segment.

These are the cost penalty of non-local NVS object access. The time involved in the processing of a remote segment fault (and the resulting page faults) is indicated in Figure 7. Time increases downward; the horizontal position indicates where the processing is taking place, as labeled on the top of the figure. T_s and T_p are the intervals indicated by the dashed line under the process address reference trace ρ.

These delays can be partitioned into three classes: (1) Host dependent delays are those directly related to the application (such as object size) and host architecture and operating system (such as the time to perform a context switch). The network interface latency is excluded from this class. These delays fall in the process, NVS, ALTP-OT/CPU, and network (for object size) columns of Figure 7. (2) Network dependent delays include the VHSI queuing and speed-of-light distance latencies, and are incorporated in the network column. (3) Network interface dependent delays are those that are included in the CMP and CAP processing, and are of primary concern in the following discussion.

The performance metrics are a function of the functional partitioning $\mathcal{F}$ and input performance parameters $\mathbf{P}$, thus $T = f(\mathcal{F}, \mathbf{P})$. The metrics T_s and T_p will now be discussed.

Remote segment fault delay. Define the time a process must wait for a (remote) segment fault as

$$T_s = T_{\text{req}} + D_{\text{gs}} + T_{\text{rem}} + D_p + T_{\text{resp}} + T_{s\leftarrow\sigma}$$

with components:

T_{req}	$\triangleq$	local host request processing time
D_{gs}	$\triangleq$	latency of get-segment request propagation
T_{rem}	$\triangleq$	remote host request processing time
D_p	$\triangleq$	latency of returning the first page p_0
T_{resp}	$\triangleq$	local host time in processing page response
$T_{s\leftarrow\sigma}$	$\triangleq$	superpacket to segment mapping

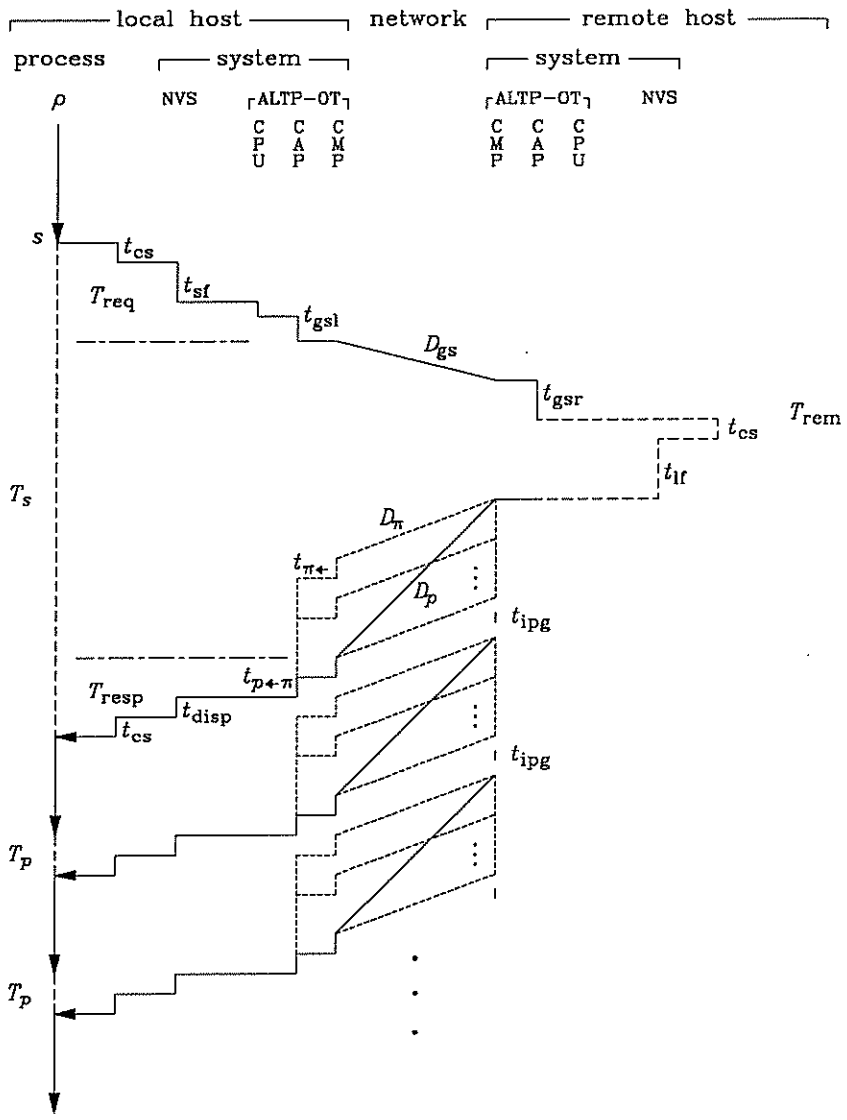

Figure 7: Segment Fault Processing

Each of these time components will be described in greater detail, with particular consideration of implementation options that affect the partitioning among host CPU, CAP, and CMP.

T_{req} is the local host request processing time

$$T_{req} = t_{cs} + t_{sf} + t_{gsl}$$

and consists of a context switch (t_{cs}), segment fault processing (t_{sf}), and get-segment local request initiation (t_{gsl}). Clearly the context switch is a function of the scheduler and the segment fault a function of NVS on the local host CPU. The request initiation must occur in coordination with ALTP-OT on the host CPU or CAP and the CMP. Note that latency involved in a name server request for a symbolic host name binding is not considered here, because this will only be added with probability that the appropriate NVS tables do not have this information cached.

D_{gs} is the end-to-end propagation delay of the get-segment request between hosts, and since a single control packet is small, is primarily a function of the latencies $d_x + d_c + d_q + d_r$. As discussed before, a CMP with a fairly large number of pipeline stages provides acceptably small d_{xr}.

T_{rem} is the remote host request processing time

$$T_{\text{rem}} = t_{\text{gsr}} + t_{\text{cs}} + t_{\text{lf}}$$

and consists of remote get-segment request processing (t_{gsr}), and if this is the first get-segment for this segment, a context switch (t_{cs}) and link fault (t_{lf}). As indicated before, the context switch is a function of the remote host scheduler, and the link fault a function of the remote host NVS. The remote request processing is an ALTP-OT function that must be recognised by the CMP, and then passed to the CAP for processing. Thus, the main consideration here is that the CAP perform control processing reasonably quickly.

D_p is the the end-to-end delay of returning the first page p_0 in segment s. This consists of the propagation of each packet in the page.

$$D_p = d_x + d_c + d_q + d_r + \left\lceil \frac{|p|}{|\pi_d|} |\pi| \tau \right\rceil$$

where $|\pi_d|$ is the size of the data portion of the packet. Note that this expression assumes that the stream of packets in each page is contiguous (*e.g.* not interleaved with other congram packets), since the CMP transmits pages as bursts of contiguous packets.

T_{resp} is the local host time in processing the page response

$$T_{\text{resp}} = t_{\pi\leftarrow} + t_{p\leftarrow\pi} + t_{\text{disp}} + t_{\text{cs}}$$

and consists of marking packet presence ($t_{\pi\leftarrow}$), packet to page mapping ($t_{p\leftarrow\pi}$), the process dispatch (t_{disp}), and a resulting context switch (t_{cs}). The dispatch and context switch are a function of the local host CPU and operating system.

Recognising the arrival of packets must clearly take place on a *per* packet basis, and at least be initiated by the CMP critical path. The packet to page presence mapping may be either explicit or implicit. The CMP will place packets directly into the proper locations of the target page in CMM. In the explicit case, the CMP critical path records packet arrivals, and determines page presence when all of the page's packets have arrived. One option is for the CMP to store packet presence state on chip. A full array implementation involves far too much memory to consider. Since the probability is high that packets will arrive in sequence, the CMP only needs to track expected packets that are missing, tagged by congram and request id. Each time the last packet in a page has arrived, (initially or after retransmission), the page is marked present in the page descriptor table. Note that explicit mapping allows the CMP or CAP to request retransmission of missing or corrupted packets based on the best retransmission policy, *e.g.* whenever the timers fire for each packet or page.

In the implicit case, when packets arrive, *packet presence bits* are set in the corresponding host page descriptor table entry. When the host page faults, the packet presence vector is examined to determine if the whole page is present. Note that while relieving

the CMP of a certain amount of complexity (especially memory for the packet presence bit vectors), this restricts the ability to request missing or corrupted packets to the times at which page faults occur, rather than based on CMP or CAP timers. An alternative implicit scheme uses additional structure in the CMM to tag packet chunks of memory with presence bits, which are used by the host (or CAP) to set page descriptor table presence bits.

$T_{s \leftarrow \sigma}$ is the superpacket to segment mapping. In the absence of packet errors (including mis-sequence), a single super-packet transmission corresponds exactly to a segment transfer, and $T_{s \leftarrow \sigma} = 0$. In the presence of errors, $T_{s \leftarrow \sigma}$ is the delay waiting for all retransmitted and mis-sequenced packets to arrive for the first referenced page in the segment. Note that a packet error in the first referenced page roughly doubles the access time T_s. The most complex aspect of the error retransmission involves packet timers. A counter must be maintained for each active congram request, which is incremented for every expected packet arrival. If a packet is not yet present once the corresponding value has been reached, it is a candidate for retransmission. It is reasonable to expect that the CAP should be involved in this process in cooperation with the CMP.

Page fault delay. Define the time a process is blocked on a remote page fault (after remote segment fault and T_s) as T_p.

Define the host CPU clock cycle as t_w per word of length w bits, and the normalised bit time for data without packet header overhead as τ_d. Assume a sequential program address reference trace and uniform rate specification. If the packet arrival rate exceeds the rate at which the program execution proceeds ($\tau_d \leq t_w/w$), page faults will not occur in the segment, and $T_p = 0$. The advantage of segment granularity object transfer is that pages are cached based on application structure and the end-to-end latency for each page fault has been eliminated (after the first).

If the rate of program execution exceeds the packet arrival rate ($t_w/w < \tau_d$), the process will page fault and be blocked for $T_p = |p|(\tau_d - t_w/w)$. Thus it is important for the packet rate to exceed instruction rate if possible. Clearly as program locality of reference decreases, the probability of blocking for a page increases, unless there is a corresponding increase in the network data rate.

Note that the same effect of a uniform rate specification can be obtained by allowing a page length burst at the peak rate, with an inter-page gap (t_{ipg}) satisfying the average rate. This allows for a simple implementation of the CMP rate control, and more efficient use of the CMM. Multiplexing congrams at the transmitting end then occurs at page granularity.

7. RELATED WORK

Several recent efforts have been underway to provide high performance host–network interface architectures. The NAB (network adapter board) [8] is a custom host-interface designed to support VMTP [3]. The NAB protocol processor is a general purpose microprocessor and uses VRAM for communication, but packets are buffered in a store-and-forward manner for sequencing, since VMTP requires packet sequence to be maintained.

The Nectar CAB (communication accelerator board) [1] provides a workstation-network interface of 10 MBps, and avoids a store and forward hop when CAB memory is mapped into the host address space. The CAB is connected to the host through a VME-bus port.

The protocol engine (PE) is a design using multiple VLSI RISC processors to implement the streamlined protocol and packet formats of XTP (express transport protocol) [4, 5].

Other emerging efforts include the HOPS network interface [7], which uses horizontally organised packet formats to allow multiple layer packet processing in parallel.

The use of time to access remote storage as the primary performance metric is related to the performance metric in Memnet [6], which also provides a shared-memory view of the network, but the Axon measure is related to virtual rather than real memory.

Axon is based on underlying assumptions and tradeoffs that are very different than these other efforts. Specifically, these include the *quasi-reliability* provided by the underlying congram-oriented internet protocol (MCHIP) and subnetworks that make resource reservations and provide guarantees on delay and packet loss, and the much higher data rates of the VHSI. Furthermore, there is a greater emphasis on the integrated design of host architecture, protocols, and operating systems, as well as on the systematic evaluation of the division of functionality between hardware and software. Finally, there is the provision for increased functionality using the NVS mechanism.

8. CONCLUSIONS

A new host communication architecture for distributed systems has been proposed called Axon, which can support IPC with high throughput and low latency across the VHSI. The significant features of Axon are the network virtual storage facility, which includes support for virtual shared memory on loosely coupled systems, a high performance object transport facility which can be used by both message passing and shared memory mechanisms, and a pipelined network interface. The emphasis in the design of Axon has been to provide a direct data path between communicating applications, using an integrated design of host architecture, operating systems, and communication protocols.

The design of the Axon architecture has been presented, along with alternatives and tradeoffs in the host architecture giving the network interface direct access to host memory without store-and-forward packet buffering. The design of the MIA (memory interface architecture) host–network interface and CMP (communications processor) has also been presented.

The bandwidth and latency requirements of a high performance network interface have been described. In particular, there is reasonable latitude in the interface delay with respect to the overall end-to-end latency, as long as a sufficiently high data rate is maintained in the pipeline, and no full packet buffering takes place. It is reasonable to expect sub-second round trip latency, even across a long distance internetwork transporting large objects.

Given the appropriate functional partitioning it is reasonable to implement critical path functions directly in the network interface hardware. The CAP provides a useful role in the control processing hierarchy, allowing the CMP to only implement critical path function while relieving the host CPU of much of the overhead of protocol processing.

The methodology for functional partitioning and determination of Axon performance has been presented. The actions associated with remote memory access in Axon have been described, and their evaluation is being pursued with detailed simulations to be followed by implementation as this work continues.

9. REFERENCES

[1] Arnould, Emmanuel, *et al*, "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers", *ASPLOS-III (ACM SIGOPS OS Rev.)*, Vol.23, ACM, New York, April 1989, pp. 205–216.

[2] Bensoussan, A., C.T. Clingen, and R.C. Daley, "The Multics Virtual Memory: Concepts and Design", *CACM*, Vol.15 #5, May 1972, pp. 308–318.

[3] Cheriton, David, "VMTP: A Transport Protocol for the Next Generation of Computer Systems", *SIGCOMM'86 (CCR)*, Vol.16 #3, ACM, New York, 1986, pp. 406-415.

[4] Chesson, Greg, *et al*, "XTP Protocol Definition", Revision 3.4, Protocol Engines, Inc., PEI 89-103, Santa Barbara, Calif., July 1989.

[5] Chesson, Greg, "XTP/PE Design Considerations", *IFIP WG6.1/6.4 Workshop on Protocols for High Speed Networks*, May 1989, reprinted as: Protocol Engines, Inc., PEI 90-4, Santa Barbara, Calif., 1990.

[6] Delp, Gary S., Adarshpal S. Sethi, and David J. Farber, "An Analysis of Memnet: An Experiment in High-Speed Shared-Memory Local Networking", *SIGCOMM'88 (CCR)*, Vol.18 #4, ACM, New York, 1988, pp. 165–174.

[7] Haas, Zygmunt, "A Communication Architecture for High Speed Networking", *INFOCOM'90*, Vol.II, IEEE Comp.Soc., Wash., D.C., June 1990, pp. 433–441.

[8] Kanakia, Hemant and D.R. Cheriton, "The VMP Network Adapter Board (NAB): High Performance Network Communication for Multiprocessors", *SIGCOMM'88 (CCR)*, Vol.18 #4, ACM, New York, 1988, pp. 175–187.

[9] Mazraani, Tony Y. and G.M. Parulkar, "Specification of a Multipoint Congram-Oriented High Performance Internet Protocol", *INFOCOM'90*, Vol.II, IEEE Comp.Soc., Wash., D.C., June 1990, pp. 450–457.

[10] Parulkar, Gurudatta M., "The Next Generation of Internetworking", *ACM SIGCOMM CCR*, Vol.20 #1, ACM, New York, Jan. 1990, pp. 18–43.

[11] Parulkar, Gurudatta M. and J.S. Turner, "Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment", *INFOCOM'89*, IEEE Comp.Soc., Wash., D.C., Vol.II, pp. 655–667.

[12] Sterbenz, James P.G., *Axon: Host–Network Interface Architecture Design*, Wash. U. CS Dept., WUCS-90-7, St. Louis, March 1990.

[13] Sterbenz, James P.G. and G.M. Parulkar, "Axon: Network Virtual Storage Design", *ACM SIGCOMM CCR*, Vol.20 #2, ACM, New York, April 1990, pp. 50–65.

[14] Sterbenz, James P.G. and G.M. Parulkar, "Axon: A High Speed Communication Architecture for Distributed Applications", *INFOCOM'90*, Vol.II, IEEE Comp.Soc., Wash., D.C., June 1990, pp. 415–425.

[15] Sterbenz, James P.G. and G.M. Parulkar, "Axon Network Virtual Storage for High Performance Distributed Applications", *10th ICDCS*, IEEE Comp.Soc., Wash., D.C., June 1990, pp. 484–492.

[16] Sterbenz, James P.G. and G.M. Parulkar, "Axon: Application-Oriented Lightweight Transport Protocol Design", *ICCC'90*, ICCC, Narosa Publishing House, New Delhi, India, Nov. 1990, pp. 379–387.