Report Number: WUCS-90-07

1990-03-01

Axon: Host-Network Interface Design

James P.G. Sterbenz

This paper describes the Axon host-network interface architecture. The Axon project is investigating an integrated design of host architecture, operating systems, and communications protocols to allow applications to utilize the high bandwidth provided by the next generation of communications networks. The Axon host architecture and network interface is designed to provide a high bandwidth low latency path directly between the network and host memory. 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... **Read complete abstract on page 2.**

Axon: Host-Network Interface Design

James P.G. Sterbenz

Complete Abstract:

This paper describes the Axon host-network interface architecture. The Axon project is investigating an integrated design of host architecture, operating systems, and communications protocols to allow applications to utilize the high bandwidth provided by the next generation of communications networks. The Axon host architecture and network interface is designed to provide a high bandwidth low latency path directly between the network and host memory. 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 visualizations and imaging, requiring high bandwidth and low latency.

AXON: HOST-NETWORK INTERFACE DESIGN

James P. G. Sterbenz

WUCS-90-07

March 1990

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

Abstract

This paper describes the Axon host-network interface architecture. The Axon project is investigating an integrated design of host architecture, operating systems, and communications protocols to allow applications to utilize the high bandwidth provided by the next generation of communications networks. The Axon host architecture and network interface is designed to provide a high bandwidth low latency path directly between the network and host memory. 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 visualization and imaging, requiring high bandwidth and low latency.

James Sterbenz is on leave of absence from IBM Corporation at Washington University in St. Louis.

AXON: HOST–NETWORK INTERFACE DESIGN

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

1. Introduction

Ongoing research in the computer communication and telecommunications fields suggests two emerging trends which are complementary to one another. First, as time goes on we will continue to witness communication networks which can support increasingly high data rates. For example, networks with data rates of hundreds of Mbps are being prototyped, and networks with data rates above 1Gbps are being planned. The future generation of internetwork, consisting of these high speed subnetworks, is referred to as the *very high speed internetwork* (VHSI) [Pa90]. Second, a diverse application set having differing bandwidth, latency, and reliability requirements will have to be supported on the VHSI communication substrate. For example, video distribution, computer imaging, distributed scientific computation and visualisation, distributed file and procedure access, and multimedia conferencing are all target applications. These trends pose a number of new challenges and opportunities to researchers. One such challenge is how to support high performance interprocess communication (IPC) in this environment.

The existing approach of supporting IPC cannot deliver the underlying high bandwidth to newer and demanding applications because of a number of reasons: lack of integration among host architecture, operating system, and communication protocols; performance bottlenecks in the existing end-to-end protocols and their implementation; and almost no support for the shared memory paradigm in a loosely coupled or network environment.

A new communication architecture for distributed systems has been proposed called **Axon** [StPa89a, StPa90b]. 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 across networks [StPa89b, StPa90a, StPa90c]; [3] a high performance, lightweight object transport facility which can be used by both message passing and shared memory mechanisms [StPa89c]; [4] a pipelined network interface which can provide a high bandwidth low latency path directly between the VHSI and host memory.

This paper presents a description of the Axon host and network interface architecture, and is organised as follows: Section 2 describes the bandwidth and latency aspects of host–network interface design. Section 3 provides an overview of the Axon architecture as background. Section 4 describes the host architectures in support of Axon, Section 5 the Axon host–network interface, and Section 6 the design of the communications processor (CMP). Section 7 discusses Axon performance and the partitioning of function between hardware and software.

2. 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 or delay 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 1.

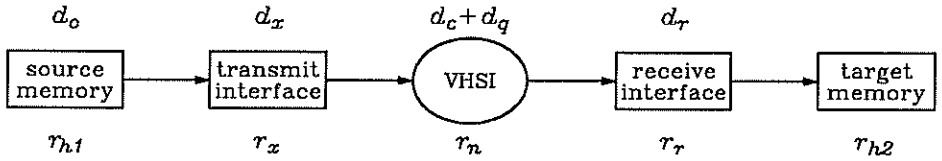

Figure 1: Interface Performance Parameters

2.1. 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 h_{r1}, 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 destination host rate h_{r2}.

Assume that the clock cycle of the sending and receiving hosts are equal: $r_h = r_{h1} = r_{h2}$. A host CPU processes data based on a word width of w with a clock cycle t_w. The rate at which data can be processed is then $r_h = 1/t_w$ [word/sec] which is w/t_w [bits/sec]. The VHSI data rate is $r_n = 1/\tau$ [bit/sec]. It will be assumed that the host is able to process data at the network bandwidth ($\tau_d = t_w/w$ where τ_d is the normalised data rate excluding header/trailer overhead), and therefore τ is used to determine the required clock cycle for the host–network interface interface. The data rate of the interface is $1/\tau_\omega$, where an internal data path width of ω allows the clock cycle to be $\tau_\omega \leq \omega\tau$.

For example, for a VHSI supporting connections of 1Gbps (τ = 1ns), a network interface with octet wide data paths ($\omega = 8$) needs to have a clock cycle of at most 8ns.

2.2. 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 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 pipeline delay at the receiving network interface d_r. Note that this excludes the latency of retransmitted packets.

Object transmission time. The object transmission time is a function of the object size $|o|$ [bits] and data rate $1/\tau$, giving $d_o = |o|\tau$. Examples for objects of various sizes at $1/\tau$ = 1Gbps are:

	o	$\|o\|$	d_o
bit	b	1b	1ns
byte	B	8b	8ns
packet	π	48+5B	424ns
		128+16B	1.2μs
page	p	1KB	9.2μs
segment	s	1MB	9.4ms
		1GB	9.6s

The packet structure and object hierarchy in Axon will be described later; this table is intended to provide an example rather than to specify the exact object sizes in Axon.

The overall goal is to minimise D_o, and the Axon architecture must minimise $d_x + d_r$, which are the 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 and host CPU power and memory size, d_o may remain a significant part of the latency.

Distance related latency. The speed of light delay is a function of the path length g and the velocity of light in fiber c_f, giving $d_c = g/c_f$. It is the responsibility of the VHSI to minimise queueing delay d_q. It is assumed that the VHSI provides rate based flow control, and therefore that the endpoint hosts need not be concerned with congestion control and are not able to directly control d_q.

The speed-of-light and queuing delays can be combined into a single *distance related latency* $d_{cq} = d_c + d_q$, which reflects the scope of the network in terms of both geographic distance and network hierarchy. The model for distance related latency is presented in Figure 2.

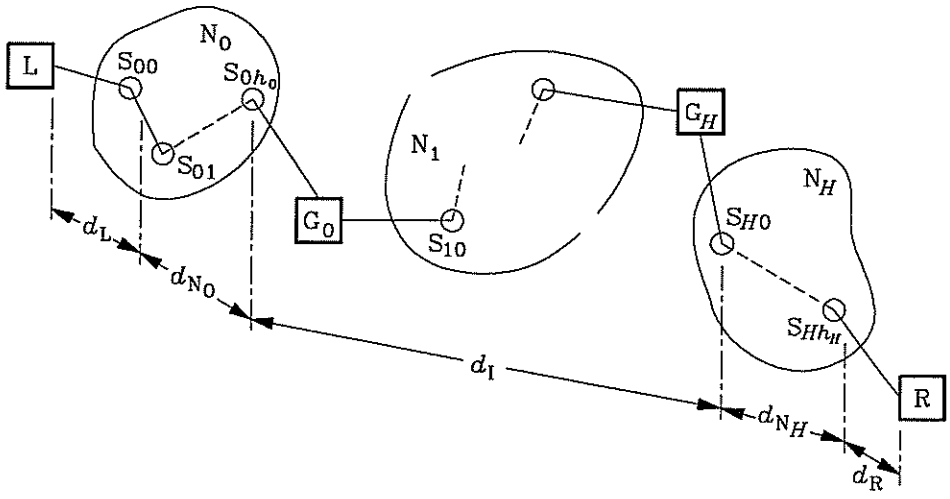

Figure 2: Distance Related Latency Model

Define the number of subnetwork N hops in an internetwork I as H, and the number of switch node S hops in subnetwork N_j as h_j. The endpoint hosts are L (local) and R (remote), and gateways G.

The latencies for an internetwork, subnetwork, and switch (LAN) are:

$$\begin{aligned} d_{cq\mathsf{I}} &= d_{\mathsf{L}} + d_{\mathsf{N}_0} + d_{\mathsf{I}} + d_{\mathsf{N}_H} + d_{\mathsf{R}} \\ d_{cq\mathsf{N}} &= d_{\mathsf{L}} + d_{\mathsf{N}_0} + d_{\mathsf{R}}|_{H=0} \\ d_{cq\mathsf{S}} &= d_{\mathsf{L}} + d_c(\mathsf{S}_{00}, \mathsf{R}) \end{aligned}$$

where $d_c(\mathsf{X}, \mathsf{Y})$ is the speed of light delay between X and Y. Note that $d_{\mathsf{N}_0} = 0$ or $d_{\mathsf{N}_H} = 0$ implies an endpoint host connection to a gateway directly through a LAN.

The latencies associated with the endpoint hosts $\{\mathsf{L}, \mathsf{R}\}$ include one switch hop, and are

$$\begin{aligned} d_{\mathsf{L}} &= d_{q\mathsf{S}_{00}} + d_c(\mathsf{L}, \mathsf{S}_{00}) \\ d_{\mathsf{R}} &= d_{q\mathsf{S}_{Hh_H}} + d_c(\mathsf{S}_{Hh_H}, \mathsf{R}) \end{aligned}$$

where $d_{q\mathsf{S}}$ is the queuing delay through S. The latencies associated with the endpoint subnetworks $\{\mathsf{N}_0, \mathsf{N}_H\}$ are

$$\begin{aligned} d_{\mathsf{N}_0} &= \sum_{j=1}^{h_0} \left[d_{q\mathsf{S}_{0j}} + d_c(\mathsf{S}_{0(j-1)}, \mathsf{S}_{0j}) \right] \\ d_{\mathsf{N}_H} &= \sum_{j=0}^{h_0-1} \left[d_{q\mathsf{S}_{Hj}} + d_c(\mathsf{S}_{Hj}, \mathsf{S}_{H(j+1)}) \right] \end{aligned}$$

The latency associated with the entire internetwork is:

$$d_{\mathsf{I}} = \sum_{i=1}^{H} \left[d_{\mathsf{G}i} + \sum_{j=1}^{h_i} d_{q\mathsf{S}_{ij}} + \sum_{j=0}^{h_i-1} d_c(\mathsf{S}_{ij}, \mathsf{S}_{i(j+1)}) \right]$$

where the gateway latency (queueing and links to adjacent subnets) is

$$d_{\mathsf{G}i} = d_{q\mathsf{G}_i} + d_c(\mathsf{G}_i, \mathsf{N}_{i-1}) + d_c(\mathsf{G}_i, \mathsf{N}_{i+1})$$

The range of latency can be partitioned into regions, based on the geographic diameter g [km] and hierarchical structure of the network: {LAN,MAN,WAN,IAN_{tc},IAN_{ic}} (local network, metropolitan network, wide area network, transcontinental internetwork, and intercontinental internetwork). This results in a set of differing magnitudes of d_c and d_q for each network type.

For this discussion, a simple set of assumptions bounding distance for each network type and queueing delay through gateways and switches will be used. The speed of light latencies are computed assuming fiber optic transmission technology ($c_f = 0.7c$), and upper bounds on distances [km] of $g \in \{2, 100, 5\times10^3, 5\times10^3, 20\times10^3\}$. The queuing delays d_q are computed assuming a conservative 1ms delay *per* packet switch and 10ms *per* gateway. A LAN is assumed to involve one switch latency, a MAN subnet 3 switch latencies and a WAN 5 switch latencies. A transcontinental internet is assumed to consist of a MAN at each endpoint connected through gateways into a WAN. An intercontinental internet is assumed to traverse about 6 gateway–WAN pairs.

The sum of these is the distance related latency $d_{cq} = d_c + d_q$, and are approximated below for the five network types:

$$\begin{array}{lcccccc} d_c & \in & \{10\mu\text{s}, & 500\mu\text{s}, & 25\text{ms}, & 25\text{ms}, & 100\text{ms}\} \\ d_q & \in & \{1\text{ms}, & 3\text{ms}, & 5\text{ms}, & 25\text{ms}, & 100\text{ms}\} \\ d_{cq} & \in & \{1\text{ms}, & 3\text{ms}, & 30\text{ms}, & 50\text{ms}, & 200\text{ms}\} \end{array}$$

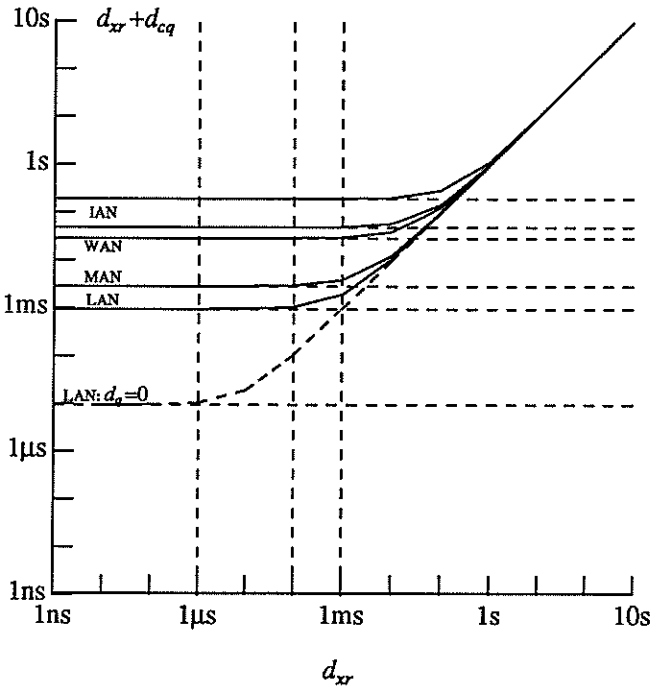

Figure 3: Host Interface Latency Sensitivity

Host interface latency. By assuming d_{cq} as a constant bound for the various kinds of networks, and plotting the end-to-end latency (without the object transmit time) $d_{cq}+d_{xr}$ *vs.* the host interface component $d_{xr} = d_x + d_r$, the sensitivity of end-to-end latency to the host interface can be seen in Figure 3.

This plot indicates when the host interface latency d_{xr} dominates the distance related latency d_{cq}, and assumes a data rate of $1/\tau = 1$Gbps. The result is that for wide area and internetworks, the requirement for interface latency is $d_{xr} < O(1\text{ms})$. For a the host interface to support LANs, the requirement increases to $d_{xr} < O(100\mu\text{s})$, and finally as an extreme lower bound, ignoring d_q we have $d_{xr} < O(1\mu\text{s})$.

The Axon host–network interface is pipelined, as will be described later. The number of acceptable stages in the host interface pipeline can be computed as $n_{xr} = d_{xr}/\tau_\omega$. Assuming an interface with octet wide data paths ($\omega = 8$), this allows a pipeline at most 125 stages long using the extreme lower bound. Note that realistically, considering $d_q = 1$ms increases this by two orders of magnitude, and that object size begins to dominate above 10KB ($d_o = 92\mu$s).

2.3. Network interface performance

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). The preceding discussion has shown that this is feasible even for the transfer of large objects (1MB) over long distances distances (20×10^3km), if a network interface can be designed with the necessary performance. Thus the design of a host–network interface with a high speed clock (τ_ω) and reasonably low latency (d_{xr}) will be explored in the rest of this paper, after a brief introduction to the higher layer aspects of Axon.

3. 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 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.

3.1. IPC in the Axon architecture

A *logical* view of the Axon protocol hierarchy is presented in Figure 4. It is important to note that this is a logical view of functionality only, and does not imply that strict layering (in the ISO-OSI sense) is being adhered to.

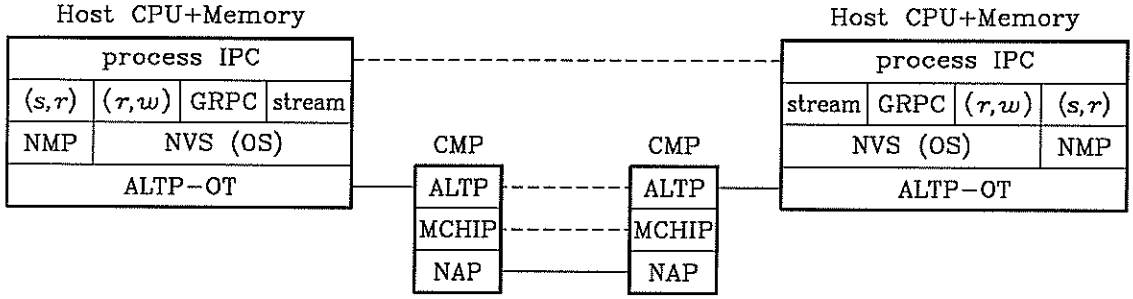

Figure 4: Logical Axon Protocol Hierarchy

IPC is supported with shared variable *read/write* (r, w) and message passing *send/receive* (s, r) 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 [StPa89b, StPa90a, StPa90c].

3.2. 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. The *network message passing* (NMP) interface performs relatively straightforward transformations from application level primitives (*e.g. send* and *receive*) to transport level message passing calls (*e.g.* send-message and receive-message). *Network virtual storage* (NVS) is the system 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 [Be72] operating system.

NVS extensions allow the 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 the segment descriptor. When a segment fault occurs for a nonlocal segment, the dynamic address translation facility invokes the transport protocol to get a copy of the segment from the appropriate system. When the segment

is returned, the appropriate page and segment descriptor presence bits are set so that program execution can resume with the normal fault recovery mechanisms.

NVS in Axon also involves extensions and additions to storage management policies. The replacement policy is affected as a result of pages from remote segments in the locality set, and therefore requires redefinition of the working set to account for non-local 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 (RS), auxiliary store (AS), a combination (RAS), or frame buffer (FB) placement, with a number of sub-policy options (swappable, nailed, *etc*). The NVS mechanisms, policies, and data structures are described in detail in [StPa89b, StPa90a, StPa90c].

3.3. Communication protocols

At the transport level, applications using the VHSI are best supported by a set of simple *application-oriented lightweight transport protocols* (ALTP) for various classes of applications [StPa89c, PaTu90]. Key features of ALTPs are the implementation of critical functions in hardware, rate based flow control, application-oriented error control, and structured collections of packets.

ALTPs are designed so that their functionality can be split into *critical* and *non-critical* paths. The critical path consists of the data path and routine packet processing, which are implemented in VLSI hardware to sustain data rates above 1 Gbps. The non-critical path function consists of everything else, specifically the control that must involve host interaction for connection setup and initiation of object transfer. By optimising the critical path functions, and by processing multiple packets in a single host transport level operation, the *per* packet processing is sustained at the full VHSI data rate. For the protocol to be efficiently implemented in hardware, its design must be well integrated with the host architecture and operating system.

ALTPs are optimised to provide the kind of performance guarantees and functionality the specific applications need. The ALTP type used in Axon is designed to support IPC *object transfer* (especially NVS segments), called ALTP-OT. The underlying internet/network layer of function is provided by a *multipoint congram-oriented high-performance internet protocol*† (MCHIP) [Pa90, MaPa89], and *network access protocols* (NAP).

MCHIP		connid	ALTP		reqid	segment (frame)		page (scanline)		pkt	data	cksum
Ø	type	c	Ø	type	q	$\|g\|$	k	$\|s\|$	j	i		Σ
2		2	2		2	1	1	2	2	1		2

Figure 5: ALTP-OT Data Packet Format

Packet structure and format. Information is transferred throughout the internetwork in packets. A structured 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 a small amount of data), and optionally followed by data packets. Bits in the packet header indicate whether the packet is control or data. ALTP-OT control packets require processing by the ALTP-OT logic in the CMP (communications processor), as well as by the host system hardware and operating system. Data packets require considerably less processing, all of which can be done in real time by the CMP hardware. The format of a data packet presented in Figure 5.

† 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.

Each data packet corresponds to a fragment π_i of a page p_j of a segment s_k of a segment-group g corresponding to a super-packet σ. For a video-graphics segment a page corresponds to a scanline, a segment to a frame, and a segment group to an image.

The benefits of this packet/super-packet hierarchy is that most of the usual *per packet* control processing is only performed *per super-packet* in Axon. A structuring of the data that is recognised by ALTP-OT allows the *per* packet processing to be simplified to the extent that VLSI implementation is reasonable and efficient. In addition, since ALTP-OT is tightly integrated with the host systems software it has direct access to the appropriate operating system facilities (*via* lightweight system calls) and data structures (such as virtual storage management tables), resulting in efficient coordination between ALTP-OT and conventional operating system operations. For example, assuming a data rate of 1Gbps and packet size of 144B (128B data + 16B header/trailer), the transfer of a stream of 1MB segments will involve the host processing of super-packets every 9.4ms, rather than of packets every 1.2μs. Additionally, super-packet processing corresponds to segment fault processing, and therefore requires no additional interrupts and context switches, as would *per* packet host processing.

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 (MCHIP) [MaPa89, Pa90] 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 the CMP 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, ALTPs are not concerned with congestion control, further simplifying the ALTP and 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. For ALTP-OT, the packet handling is:

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

Note that due to the orientation of ALTP-OT to this application, 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* (PKT), *page* (PGE), *segment* (SEG), and segment-*group* (GRP).

• **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* (AR) 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* (DR), 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* (NP) policy, or to *preempt* (PE) 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 error packets only when the page is referenced, and preempt the primary data stream (PGE-DRPE).

Operations. The ALTP-OT requests and operations are listed below. A detailed description of ALTP-OT is presented in [StPa89c].

Connection/congram	
join-ipc	join/establish connection/congram
respecify-rate	alter rate specification
leave-ipc	leave/terminate connection/congram
Receive	
get-segment	obtain segment copy
acquire-segment	acquire segment authorisation
get-page	obtain page (must be acquired)
get-copy	obtain permanent segment copy
get-stream	receive segment stream
receive-message	receive IPC message
retransmit-packets	selective retransmission

Transmit	
release-segment	release or return (if modified) segment
release-page	release or return (if 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

4. 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 6. In addition, various caches ($) may be present to utilize 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, video and image 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, all of which contributes to the interface latency d_{xr}. 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 $1/\tau$ VHSI rate communications (if at all).

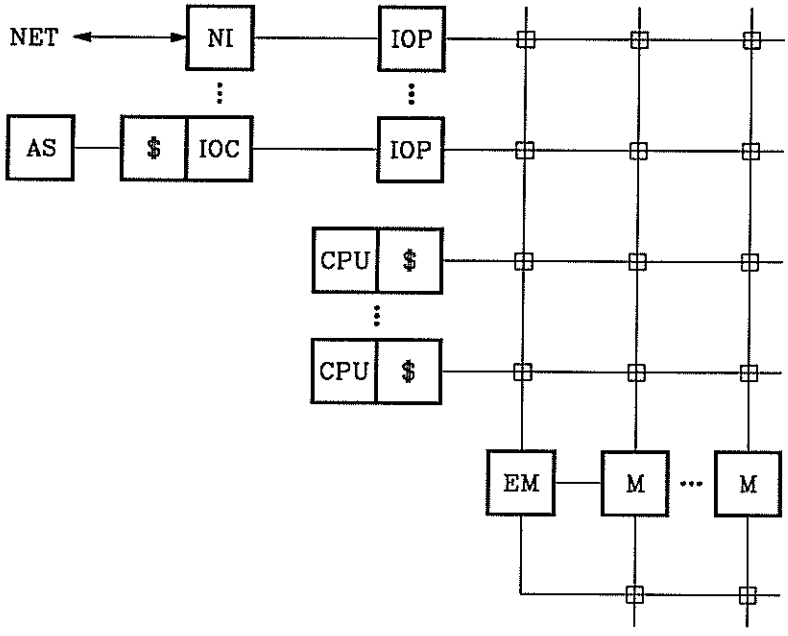

Figure 6: Conventional Host Architecture

For Axon to support communications in the VHSI environment, it is necessary to provide high bandwidth low latency data paths directly to memory, motiving 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.

4.1. 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 7.

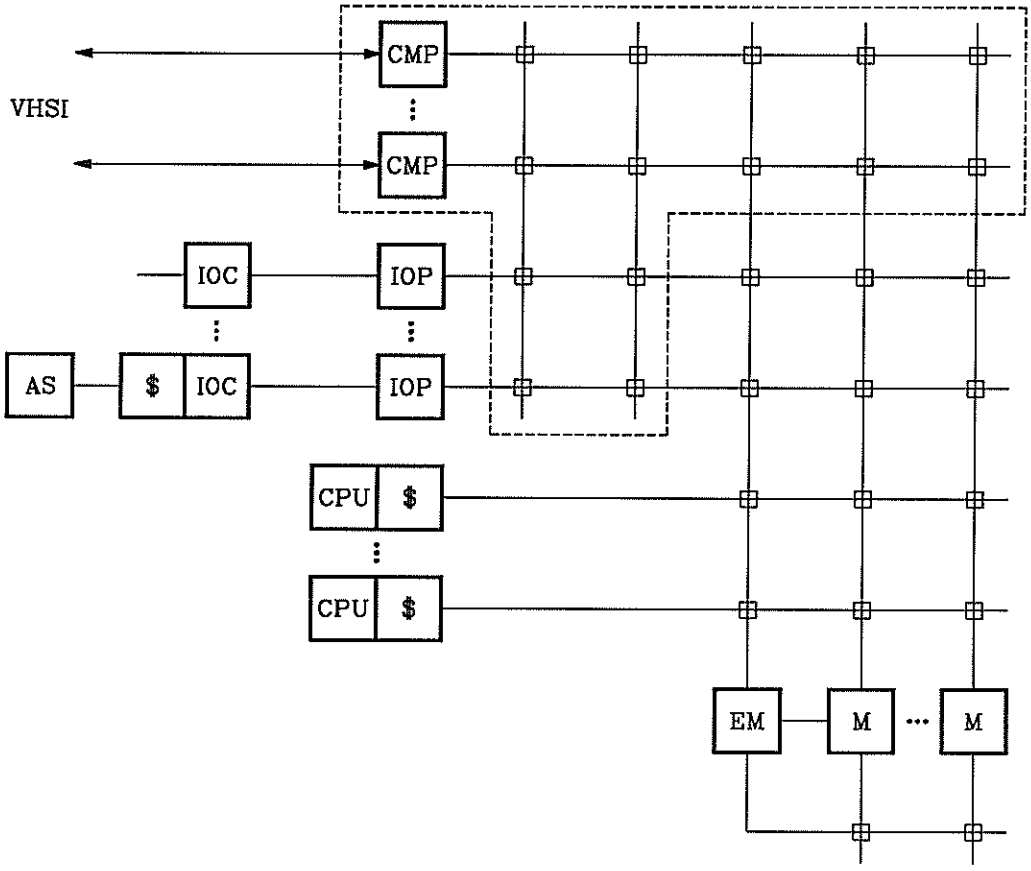

Figure 7: Interconnect Interface Architecture (IIA)

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 provides the path to auxiliary 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.

4.2. 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 8.

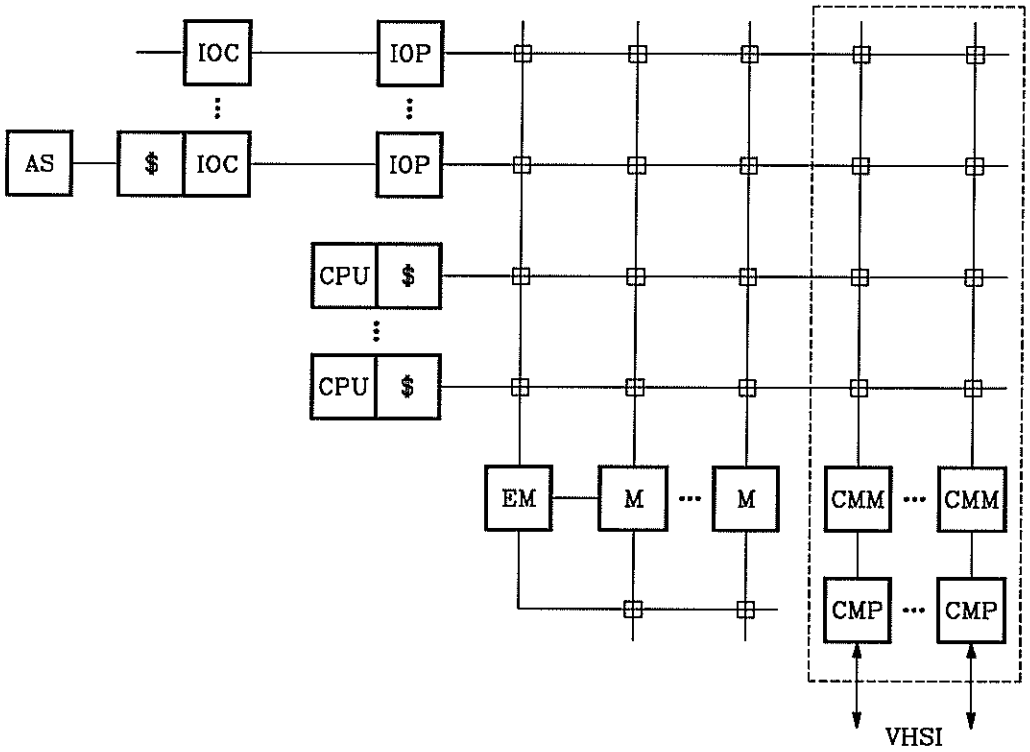

Figure 8: Memory Interface Architecture (MIA)

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 serial access interfaces to the CMP (transmit and receive), and must operate at a rate of $1/\tau_\omega$.

4.3. Comparison of host architectures (IIA and MIA)

This section discusses 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 $\{\mathrm{M}_1, \ldots, \mathrm{M}_n; \mathrm{CMM}_1, \ldots \mathrm{CMM}_m\}$. Furthermore, remote segments are clustered in fewer memory

modules. This scheme also makes memory access somewhat more difficult, particularly if there are multiple CPU memory modules for the purpose of interleaved memory access. Additionally, the fraction of real store devoted to CMM must be determined, based on communications requirements.

Blocking of traffic. In IIA, communications 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 communication 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 communications traffic can proceed at a rate of $1/\tau_\omega$ with no blocking using the serial 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 CMP connections and additional traffic at a rate of $1/\tau_\omega$ *per* CMP attached, assuming $\omega = w$ (CPU word length matches CMP datapath width). If $\omega \neq w$, additional complexity results in data width conversion.

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 require a completely new memory design, specifically the high performance multi-ported CMM, which must support simultaneous asynchronous serial ports of rate $1/\tau_\omega$ along with a random access port with a cycle time of t_w.

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.

5. 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 9. The interface is bidirectional, but the functions have been labeled for communications in the left-to-right direction for clarity.

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 serial 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, rate specification timing, as well as the *per* packet congram multiplexing and control.

The CMM is a multiported memory, with serial 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. Examples include

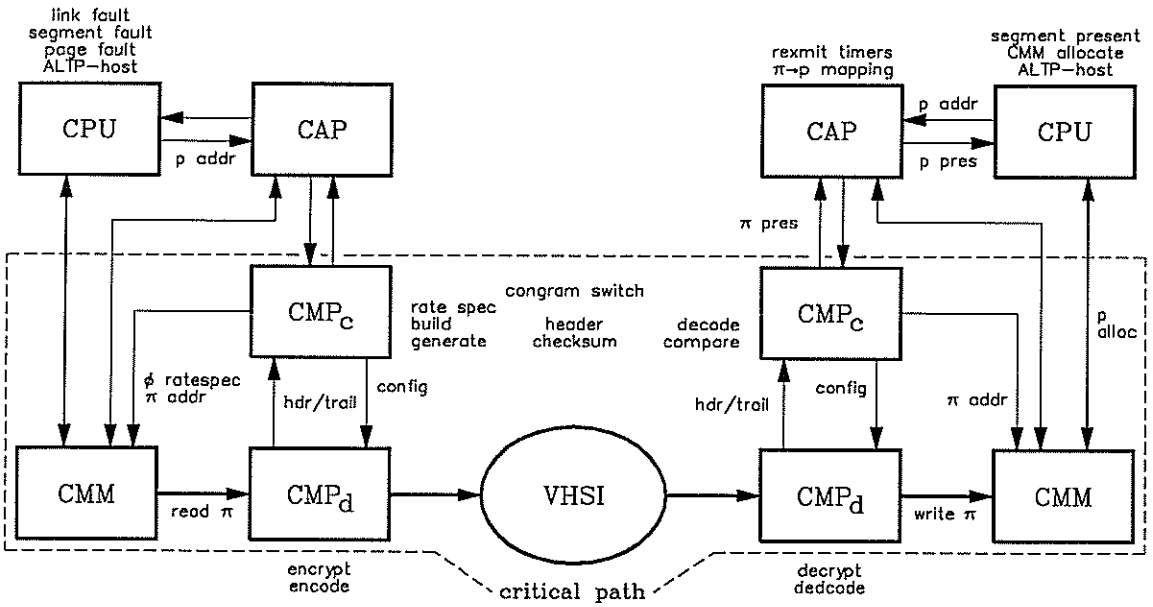

Figure 9: Axon Host–Network Interface

the packet arrival to page presence mapping ($\pi \rightarrow p$), and packet retransmission timer management functions.

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

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 rate. For example, it is unreasonable to expect a host CPU to process incoming and outgoing packets every 424ns ($|\pi| = 53$B, $1/\tau = 1$Gbps), and to concurrently execute the application needing this bandwidth. It should be noted that the design of a simple critical path can remain constant with control optimisations that affect non-critical path software in the CAP and host CPU.

Other control functions may or may not need to be 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 ($1/\tau$) for a given CMP processing rate ($1/\tau_\omega$). 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 function between the host CPU and CAP is also important. This is dictated by the desire to engage the host in minimal interaction with the communication that is not directly related to application execution. Thus, it is reasonable to expect the host to initiate an ALTP-OT segment transfer as the 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 the segment has fully 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 what control function should be implemented as part of the critical path involves a time–space complexity tradeoff, and will be discussed in Section 7.

6. Communications Processor

This section describes the CMP (communications processor) design. First an implementation model is presented, followed by a high level functional description of the CMP design implementing ALTP-OT for an MIA host.

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.

6.1. Implementation model

The CMP implementation model (Figure 10) consists of a set of *datapath modules* (DMs) $\boldsymbol{D}=\{D_0, D_1, \ldots D_{N+1}\}$, and control modules (CMs) $\boldsymbol{C}=\{C_1, C_2, \ldots C_M\}$. The DMs perform data manipulation and transformation on packets as they pass through the CMP. In general, each DM should be designed to perform its function without buffering a packet, except for the pipeline delay as the packet passes through. One of the CMs (C_0) is responsible for pipeline configuration and control.

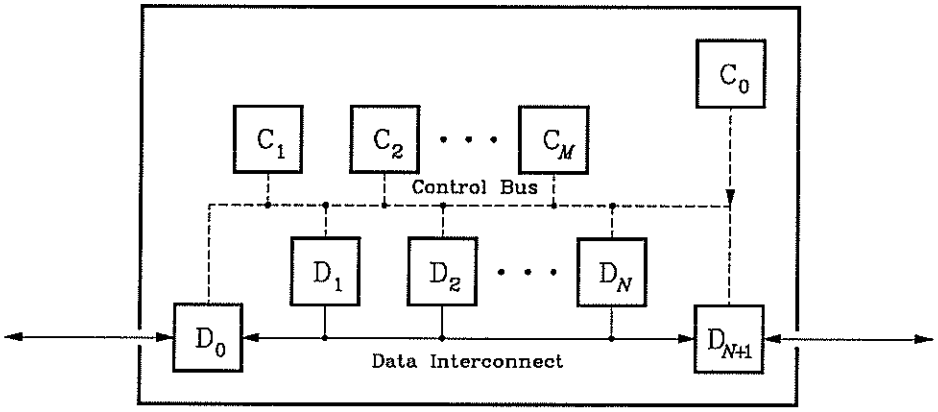

Figure 10: Communications Processor Implementation Model

A particular configuration of the pipe C_i, consists of a permuted sequence of data modules $\boldsymbol{d} = \langle d_0, d_1, \ldots d_n \rangle \subseteq \boldsymbol{D}$, forming a logical pipeline, along with a set of control modules controlling them $\boldsymbol{c} = \{c_0, c_1, \ldots c_m\} \subseteq \boldsymbol{C}$. The DMs are connected by a high speed interconnection network, which is capable of transferring data between the modules subject to the configuration C_i. Each type of ALTP requires a particular configuration of the pipeline, *e.g.* ALTP-OT induces a configuration C_{OT} of the pipeline. Note that while this model allows for reordering of the DMs, a fixed sequence pipeline may be sufficient in most implementations.

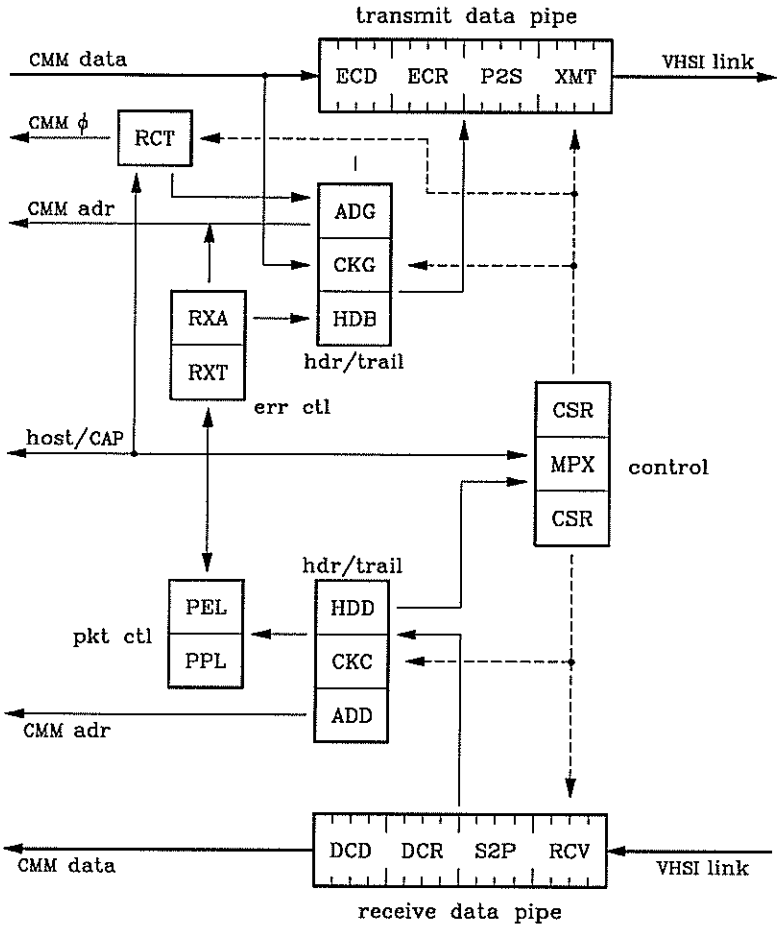

Figure 11: Communications Processor Block Diagram

6.2. Functional description

A block diagram of the CMP is shown in Figure 11. The CMP consists of a set of datapath modules and control modules. The datapath modules perform manipulation and transformation on packets as they pass through the CMP, without buffering (except for the pipeline delay).

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

ECD – Encode performs any required data 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 prepares packet information for transmission on the network link.

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

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 the transmit pipeline for each congram. The r_i values for each congram are obtained from the corresponding CSR (congram state register – see below).

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., CSR, and control information from ALTP-host to build the header for the packet. All protocol levels of encapsulation (ALTP, MCHIP, and NAP) are done at one time. The congram and request id.s (q, r) and packet id. (segment id. k, page number j, packet number i) are inserted into the header template from CMM.

The receive pipe datapath 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.

Associated with the receive data pipe are control modules:

HDD – Header decode determines the congram id. for CMP configuration, and determines the packet address in CMM from the packet index (ijk) and the base address of the page from the corresponding CSR (congram state register). The congram id. is used 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 the each page (from the CSR), and the packet index (ijk) to form the CMM address for the writing of each packet.

Connection/congram multiplexing is handled by the *congram control logic*:

MPX – Congram multiplexing control logic performs the hardware context switching of the CMP in response to transmission requests and received congram id.s.

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 CSR set for both the transmitting and receiving side of the CMP.

The *packet control* logic is responsible for recording packet arrivals and missing/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 PDT (page descriptor table) and SDT (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 retransmission requests and packet addresses.

RXA – Retransmit address generates the CMM addresses for packets to be retransmitted.

RXT – Retransmit timers determine when packet retransmission requests should be made.

7. Performance of Remote Access

This section discusses the performance of Axon in terms of the time to access remote storage. First the partitioning of functions between the critical and non-critical path is described. Then the determination of remote access time is discussed with respect to the operations involved and functional partitioning.

7.1. 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 clock cycles) *vs.* the time taken for a software implementation (in host CPU or CAP instruction cycles).

For the host, the time that control function C_i takes to complete t_i is based on the instruction cycle t_w and the number of instructions to execute m_i, as well on any extra overhead m_i' such as context switches and system calls: $t_i = (m_i + m_i')t_w$.

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, $\tau = 1$ns). Define the CMP *major cycle* τ_ω as the clock cycle internal to the CMP within the parallel data path ω bits wide. By allowing n_i stages of pipeline delay for a particular control function C_i to take place, the time it takes to complete is then $\tau_i = n_i\tau_\omega$. Thus, the *critical path time savings* for C_i is $\mathsf{CP}_i = t_i - \tau_i$.

Note that by implementing a control function in the critical path, there may be an associated cost in increased the latency through the CMP $\tau_\omega \Delta n_i$, due to Δn_i more pipeline stages required for C_i.

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 [m^2] required to implement C_i. Control function complexity can be classified by the sensitivity to other parameters: [1] *Fixed-cost control functions* utilise a fixed area, relatively independent of other parameters (an example of this is the rate control logic). [2] *Datapath-width sensitive control functions* have an area that is a function of the data path width of the CMP: $a_i = f(\omega)$, and thus constrain the possible speedup of the CMP by greater parallelism, in the same manner that total available chip area limits the possible datapath width (an example of this is the checksum logic). Note that a_i need not be linear in ω. [3] *State-sensitive control functions* are those that utilise on chip memory in maintaining state (an example is the packet presence function). 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 area for a given process technology: $\sum_i a_i \leq a_{\text{CMP}}$. It will be assumed that a complementary logic family will be used so that power dissipation is not a dominant constraint.

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 ω is clearly a significant factor in this case.

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). Particularly as data rates scale upward, the decision can be made determining which functions should be included into the critical path.

7.2. Remote storage access

The performance metrics of principal interest to Axon are the time that a process is blocked due to a remote segment fault T_s, and a page fault on a remote segment T_p; these are the cost penalty of non-local NVS object access. Note that this is similar to the primary performance metric in Memnet [De88, DeSe88] $\mathcal{T}$, which also provides a shared-memory view of the network, but that the Axon measure is based on virtual rather than real memory access. The time involved in the processing of a remote segment fault (and the resulting page faults) is indicated in Figure 12. Time increases downward; the horizontal position indicates where the processing is taking place, as labeled on the top of the figure.

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

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

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

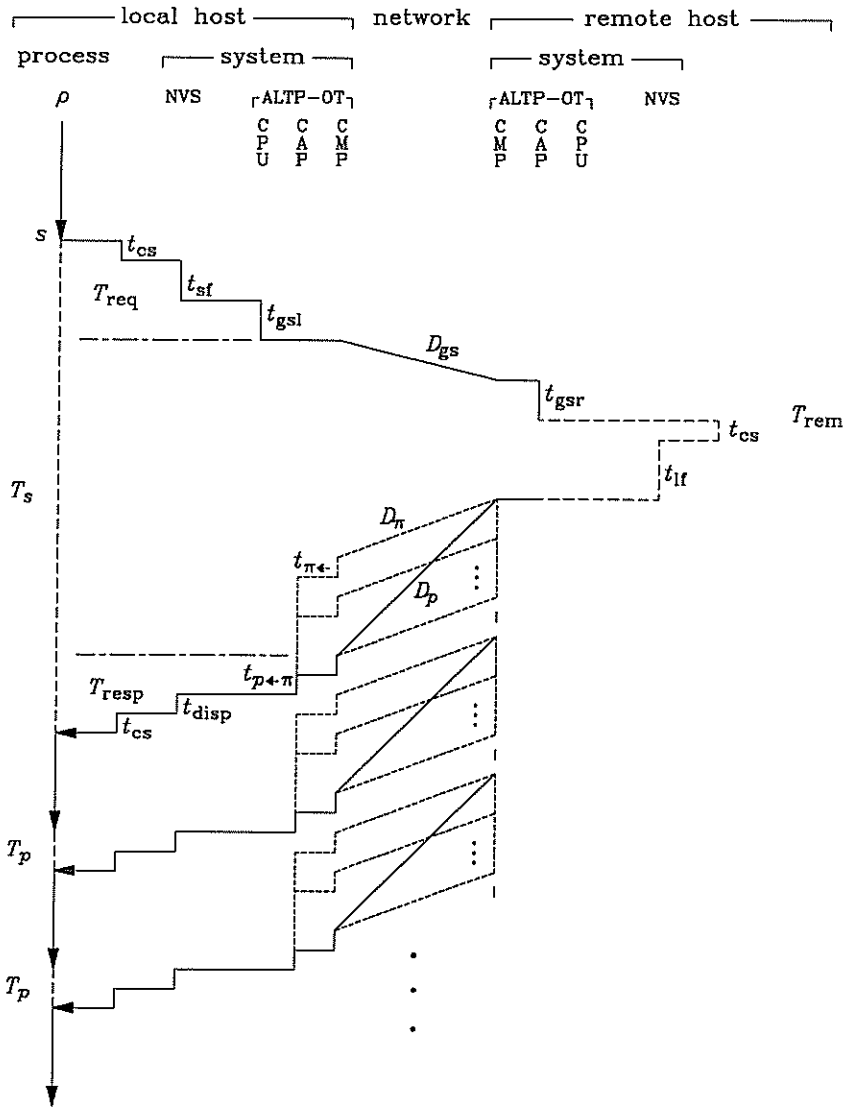

Figure 12: Segment Fault Processing

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.

D_{gs} is the end-to-end propagation delay of the get-segment request between hosts, and since the control packet is small, is primarily a function of the latencies $d_x + d_c + d_q + d_r$.

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 only on the first get-segment for each 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.

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 congrams). It is likely that the usual course of action for the CMP will be to transmit packets in page bursts.

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.

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 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 id. Each time the last packet in a page has arrived, (initially or after retransmission), the page is marked present in the PDT (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.

In the implicit case, when packets arrive, packet presence bits are set in the corresponding host PDT entry. When the host page faults, the packet presence vector is examined to determine if the whole page is present. If the entire page is not present (or on the page fault immediately after a segment fault), the host will have to periodically check the packet presence vector for page presence. Note that while relieving the CMP of a certain amount of complexity (especially memory for the packet presence state), this restricts the ability to request missing or corrupted packets to page fault events rather than based on CMP or CAP timers, and imposes additional overhead on the host CPU. 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 PDT 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 $\pi \in sp_0$, and mis-sequenced packets to arrive. Note that a packet error for a segment roughly doubles the access time T_s, assuming that the object transmission delay d_o does not dominate the latency. The most complex aspect

of the error retransmission involves packet timers. A counter must be maintained for each active congram, 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. A more detailed discussion of retransmission policies is presented in [StPa89c].

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.

Assume a sequential program address reference trace, uniform rate specification, and sequential packet arrival. If the packet arrival rate exceeds the rate at which the program execution proceeds ($\tau_d \geq w/t_w$), page faults will not occur in the segment, and $T_s = 0$. The advantage of segment granularity object movement has eliminated the end-to-end latency for each page fault.

If the rate of program execution exceeds the packet arrival rate ($w/t_w > \tau_d$), the process will page fault and be blocked for $T_p = |p|(w/\tau_w - \tau_d)$. 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 satisfying the average rate. This allows for a simple implementation of the CMP rate control, and more efficient use of the CMM.

Remote execution. One additional consideration is important when an Axon host must deal with unsolicited data. In the case of send operations, a large amount of data may follow a request, but without waiting for an intervening acknowledgement response. (An acknowledgement is still used to ensure that the initial control packet was received by the destination host.) In this case it important to predict the necessary delay following the initial request, to allow the remote host to prepare for incoming data.

As an example, the processing involved in a remote-execute operation is presented in Figure 13. Many of the time parameters are similar to the get-segment described. Define the time a process must wait for a (remote) segment fault as

$$T_u = T_{\text{req}} D_{\text{re}} + T_{\text{rem}} - D_b$$

with components:

T_{req}	$\triangleq$	local host request processing time
D_{re}	$\triangleq$	latency of remote-execute request propagation
T_{rem}	$\triangleq$	remote host request processing time
D_b	$\triangleq$	latency of sending the first bit in p_0

This trace is similar to the semgent fault trace (Fig. 12), but with data flowing just after the initial request, and in the same direction.

D_b is the the end-to-end delay of sending the leading edge of the first packet in the first page of the first segment for remote execution $\pi_0 p_0 s_0$.

$$D_b = d_x + d_c + d_q + d_r$$

and is just the end-to-end VHSI and host delay.

The delay between initiation and sending of unsolicited data is the time for the request to propagate to the remote system and the remote processing time, minus the time for the first bit in the data to traverse the network.

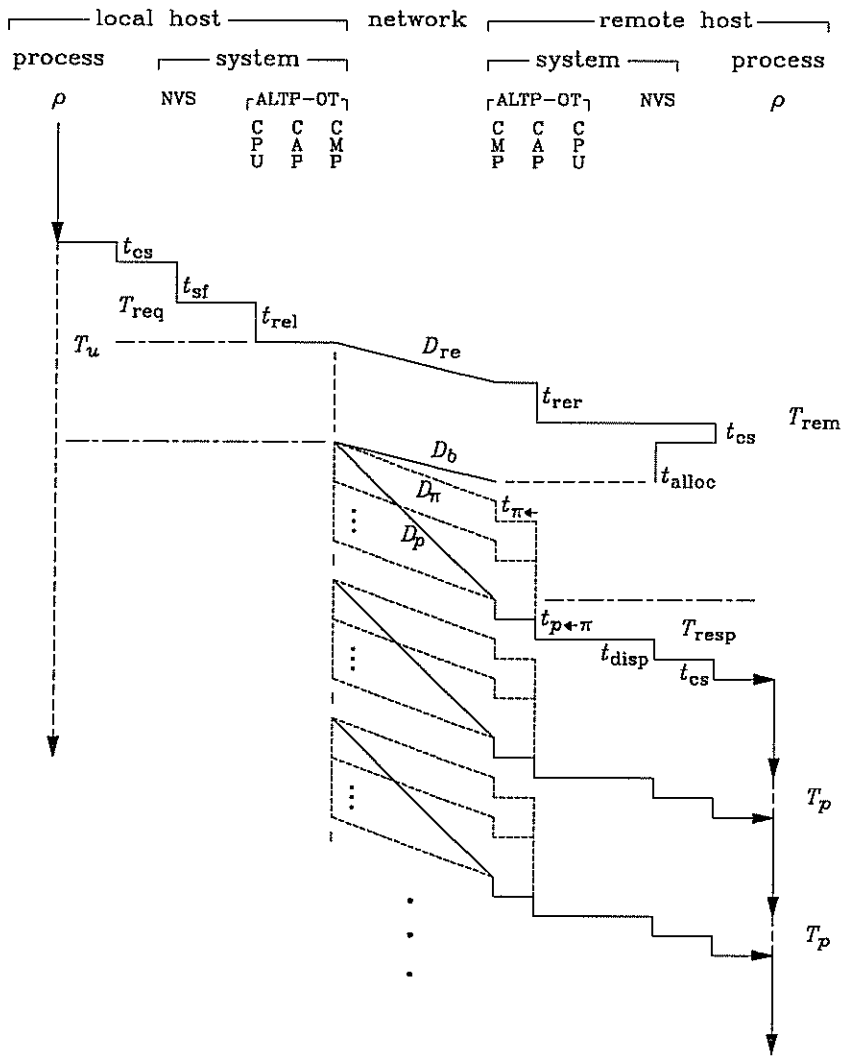

Figure 13: Remote Process Execution

8. Related Work

Several recent efforts have been underway to provide high performance host–network interface architectures. The NAB (network adapter board) [KaCh88] is a custom host-interface designed to support VMTP [Ch86a]. 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 is designed as a general purpose transport protocol.

Another approach to the performance problem is to implement existing transport protocol mechanisms in hardware, as in XTP (express transport protocol) and the PE (protocol engine) [Ch86b, ChEi88]. The XTP approach is to streamline existing protocol mechanisms and packet formats for pipeline processing, and implement each step in the pipeline using a customised VLSI processor.

Axon may be viewed as a *second generation* high performance host–network interface architec-

ture, whose design 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 rate-based communications substrate, 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 of increased functionality by the NVS mechanism.

9. Conclusions

A new host communication architecture for the 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 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.

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. Given the appropriate functional partitioning it is reasonable to implement critical 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 will be pursued with appropriate simulation and implementation as this work continues.

References

[Be72] Bensoussan, A., C.T. Clingen, and R.C. Daley, "The Multics Virtual Memory: Concepts and Design", *Communications of the ACM*, Vol.15 #5, ACM, New York, May 1972, pp. 308–318.

[Ch86a] Cheriton, David, "VMTP: A Transport Protocol for the Next Generation of Computer Systems", *SIGCOMM '86 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol.16 #3, ACM, New York, 1986, pp. 406–415.

[Ch86b] Chesson, Greg, "Protocol Engine Design", *Proceeding of the Usenix Conference*, 1986.

[Ch88a] Cheriton, David, "VMTP: Versatile Message Transaction Protocol", *DARPA Internet Program Protocol Specification*, Defense Advanced Research Projects Agency – Information Processing Techniques Office, RFC-1045, Arlington Va., Feb. 1988

[Ch88b] Chesson, Greg, "XTP/PE Overview", Protocol Engines, Inc., PEI 88-63, Santa Barbara, Calif., 1988.

[ChEi88] Chesson, Greg, Brendan Eich, Vernon Schryver, Andrew Cherenson, and Al Whaley, "XTP Protocol Definition", Revision 3.1, Protocol Engines, Inc., PEI 88-13, Santa Barbara, Calif., 1988.

[ChGr88] Chesson, Greg, and Larry Green, "XTP - Protocol Engine VLSI for Real-Time LANs", *EFOC/88 Amsterdam*, Protocol Engines, Inc., PEI 88-53, Santa Barbara, Calif., 1988.

[Cl87a] Clark, David D., Mark L. Lambert, and LiXia Zhang, "NETBLT: A High Throughput Transport Protocol", *SIGCOMM '87 Symposium: Frontiers in Computer Communications Technology (Computer Communication Review)*, Vol.17 # 5, ACM, New York, 1987, pp. 353–359.

[Cl87b] Clark, David D., Mark L. Lambert, and Lixia Zhang, "NETBLT: A Bulk Data Transfer Protocol", *DARPA Internet Program Protocol Specification*, Defense Advanced Research Projects Agency – Information Processing Techniques Office, RFC-998, Arlington Va., Feb. 1988.

[Co88] Comer, Douglas E., *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Prentice-Hall, Engelwood Cliffs, N.J., 1988.

[Cy78] Cypser, R.J., *Communication Architecture for Distributed Systems*, Addison-Wesley, Reading, Mass, 1978.

[De88] Delp, Gary S., *The Architecture and Implementation of Memnet: A High-Speed Shared-Memory Computer Communication Network*, University of Delaware Department of Electrical Engineering, #88-05-1, Newark, Delaware, May 1988.

[DeSe88] 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 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol.18 #4, ACM SIGCOMM, New York, 1988, pp. 165–174.

[IBM85a] *Systems Network Architecture: Concepts and Products*, IBM Corporation, GC30-3072-2, 1985.

[IBM85b] *Systems Network Architecture: Technical Overview*, IBM Corporation, GC30-3073-1, 1985.

[Ka89] Kanakia, Hemant Ratubhai, *High Performance Host Interfacing for Packet Switched Networks*, Stanford University Department of Electrical Engineering, Ph.D. thesis, Palo Alto, California, Nov. 1989.

[KaCh88] Kanakia, Hemant and David R. Cheriton, "The VMP Network Adapter Board (NAB): High Performance Network Communication for Multiprocessors", *SIGCOMM '88 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol.18 #4, ACM, New York, 1988, pp. 175–187.

[MaCh87] Martin, James and Kathleen Kavanagh Chapman, *SNA: IBM's Networking Solution*, Prentice-Hall, Engelwood Cliffs, N.J., 1987.

[MaPa89] Mazraani, Tony Y. and Gurudatta M. Parulkar, "Specification of a Multipoint Congram-Oriented High Performance Internet Protocol", *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'90)* IEEE Computer Society, Washington D.C., June 1990, abridged from: Washington University Department of Computer Science, technical report WUCS-89-20, St. Louis, Aug. 1989.

[Or72] Organick, Elliot I., *The Multics System: An Examination of Its Structure*, MIT Press, Cambridge, Mass., 1972.

[Pa90] Parulkar, Gurudatta M., "The Next Generation of Internetworking", *Computer Communication Review*, Vol.20 #1, ACM SIGCOMM, New York, Jan. 1990, pp. 18–43, also: Washington University Department of Computer Science, technical report WUCS-89-19, St. Louis, May 1989.

[Pa90a] Partridge, Craig, "How Slow is One Gigabit Per Second?", *ACM SIGCOMM CCR*, Vol.20 #1, New York, Jan. 1990, pp. 44-53.

[PaTu90] Parulkar, Gurudatta M. and Jonathan S. Turner, "Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment", *IEEE Network*, Vol.4 #2, IEEE, New York, March 1990, pp. 19–27, also: *Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'89)*, IEEE Computer Society, Washington, D.C., Vol.II, pp. 655–667, also: Washington University Department of Computer Science, technical report WUCS-88-7, St. Louis, 1988.

[RFC793] "Transmission Control Protocol", *DARPA Internet Program Protocol Specification*, Defense Advanced Research Projects Agency – Information Processing Techniques Office, RFC-793, Arlington Va., Sep. 1981

[St87] Stallings, William, *Handbook of Computer Communication Standards, Volume 1: The Open Systems Interconnection (OSI) Model and OSI-Related Standards*, McMillan, New York, 1987.

[St88] Stallings, William, Paul Mockapetris, Sue McLeod, and Tony Michel, *Handbook of Computer Communication Standards, Volume 3: Department of Defense (DOD) Protocol Standards*, McMillan, New York, 1988.

[StPa89a] Sterbenz, James P.G. and Gurudatta M. Parulkar, *Axon: A High-Speed Communication Architecture for Distributed Applications*, Washington University Department of Computer Science, technical report WUCS-89-36, St. Louis, Sept. 1989, presented at the Fourth IEEE Communications Society Workshop on Computer Communications, Dana Point, California, Oct–Nov 1989.

[StPa89b] Sterbenz, James P.G. and Gurudatta M. Parulkar, *Axon: Network Virtual Storage Design*, Washington University Department of Computer Science, technical report WUCS-89-13, St. Louis, May 1989.

[StPa89c] Sterbenz, James P.G. and Gurudatta M. Parulkar, *Axon: Application-Oriented Lightweight Transport Protocol Design*, Washington University Department of Computer Science, technical report WUCS-89-14, St. Louis, Sept. 1989.

[St90] Sterbenz, James P.G., *Axon: Host–Network Interface Design*, Washington University Department of Computer Science, technical report WUCS-90-7, St. Louis, March 1990.

[StPa90a] Sterbenz, James P.G. and Gurudatta M. Parulkar, "Axon: Network Virtual Storage Design", *Computer Communication Review*, Vol.20 #2, ACM SIGCOMM, New York, April 1990, pp. 50–65.

[StPa90b] Sterbenz, James P.G. and Gurudatta M. Parulkar, "Axon: A High-Speed Communication Architecture for Distributed Applications", *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'90)* IEEE Computer Society, Washington D.C., June 1990.

[StPa90c] Sterbenz, James P.G. and Gurudatta M. Parulkar, "Axon Network Virtual Storage for High Performance Distributed Applications", *10th International Conference on Distributed Computing Systems*, IEEE, Washington D.C., June 1990, pp. 484–492.

[Tu85] Turner, Jonathan S., *Design of a Broadcast Packet Switching Network*, Washington University Department of Computer Science, Washington University Computer Science Department, technical report WUCS-85-4, St. Louis, March 1985

[Tu86] Turner, Jonathan S., "Design of a Broadcast Packet Network," *IEEE Transactions on Communication.*, IEEE Vol.36 #6, New York, June 1988, pp. 734–743.