The Architecture of a Gb/s Multimedia Protocol Adapter

Erich Rütsche
IBM Research Division,
Zurich Research Laboratory
Säumerstrasse 4, 8803 Rüschlikon, Switzerland

Abstract

In this paper a new multiprocessor–based communication adapter is presented. The adapter architecture supports isochronous multimedia traffic and asynchronous data traffic by handling them separately. The adapter architecture and its components are explained and the protocol processing performance for TCP/IP and for ST–II is evaluated. The architecture supports the processing of ST–II at the network speed of 622 Mb/s. The calculated performance for TCP/IP is more than 30000 segments/sec. The architecture can be extended to protocol processing at one Gb/s.

Keywords: Multimedia Communication Subsystems; Network Protocols; Parallel Protocol Processing

1. Introduction

As data transmission speeds have increased dramatically in recent years, the processing of protocols has become one of the major bottlenecks in data communications. Current experimental networks provide a bandwidth in the Gb/s range. New multimedia applications require that networks guarantee the quality of service of bulk data streams for video or HDTV. The protocol processing bottleneck has been overcome by dedicated communication subsystems which off–load protocol processing from the workstation. Many of such communication subsystems proposed in the literature are multiprocessor architectures [Braun 92, Jain 90, Steenkiste 92, Wicki 90]. In this paper we present a new multiprocessor communication subsystem architecture, the Multimedia Protocol Adapter (MPA), which is based on the experience with the Parallel Protocol Engine *(PPE)* [Kaiserswerth 92] and is designed to connect to a 622 Mb/s ATM network. The MPA architecture exploits the inherent parallelism between the transmitter and receiver parts of a protocol and provides support for the handling of new multimedia protocols.

The goal of this architecture is to speed up the handling of multiple protocol stacks and of multimedia protocols such as the Internet Stream Protocol *(ST–II)* [Topolcic 90]. Multimedia traffic often requires isochronous transmission in contrast to conventional asynchronous traffic for file transfer or for remote procedure call. To guarantee the isochronous processing of multimedia data streams, the asynchronous and isochronous traffic are handled separately. A Header Parser scans incoming packets, detects the header fields and extracts the header information. This information is used to separate isochronous and asynchronous traffic and to split the header and the data portions of a packet. Dedicated header and data memories are used to store the header and data portions of a packet. The separation of receiver, transmitter and the dedicated memories decreases memory contention.

In Section 2 the concepts of the MPA architecture are presented. Section 3 explains protocol processing on the MPA. In Section 4 the performance of the MPA is evaluated by adapting the measurements of our TCP/IP implementation on the PPE to the MPA architecture. The last section gives the conclusions.

2. Architecture

The architecture of the MPA is based on our experiments with the PPE [Kaiserswerth 92],[Rütsche 92]. The PPE is a four–processor system based on the transputer T425 with a network interface running at 120 Mb/s. On the separate transmit and receive side two processors, the host system and the network interface use a shared memory for storing and processing protocol data. Transmit and receive side are only connected via serial transputer links.

2.1 Concept

The protocol processing requirements of multimedia protocols are very different from the requirements of traditional transport protocols. Isochronous multimedia traffic may require the processing of bulk data streams with low delay and low jitter but may accept bit errors or packet loss. Asynchronous traffic, such as file transfer or remote procedure call, requires more moderate throughput but tolerates no errors. In a file transfer between a file server and a client the throughput is limited by the I/O bus and the disk speed. Errors in the data are not acceptable, whereas a bit–error in uncompressed video is not visible.

To guarantee the requirements of multimedia connections the processing of multimedia data must be separated from the processing of asynchronous data. A Header Parser detects the connection to which an incoming packet belongs. Multimedia packets are then forwarded to dedicated multimedia devices while other packets go through normal protocol processing.

Protocol processing must be done in software to handle a multitude of protocols. Only functions that are common to all or most of the protocols are implemented in hardware. The MAC layer for ATM and the ATM Adaptation Layer *(AAL)* must be implemented in hardware or firmware to achieve the full network bandwidth of 622 Mb/s.

Our measurements of TCP/IP on the PPE have shown that the processors were not equally loaded because of the different processing requirements of the protocol layers and because of the very high costs of the memory operations [Rütsche 92]. The loose coupling via serial links between the receive and a transmit part had only minor impact on the performance. An optimal speedup of 1.7 was calculated for two processors. Therefore we chose a two–processor architecture for the MPA. One processor on the transmit side is connected via serial links to one processor on the receive side. The processors are supported by an intelligent Direct Memory Access Unit and dedicated devices for header parsing and checksumming. The memory of both parts is split into a header memory and a data memory to lower memory contention. The two halves of the MPA are only connected by serial message links.

2.2 Main Building Blocks

The MPA is split into two parts, a receiver and a transmitter, as shown in Figure 1. The various components and their function are presented in the following.

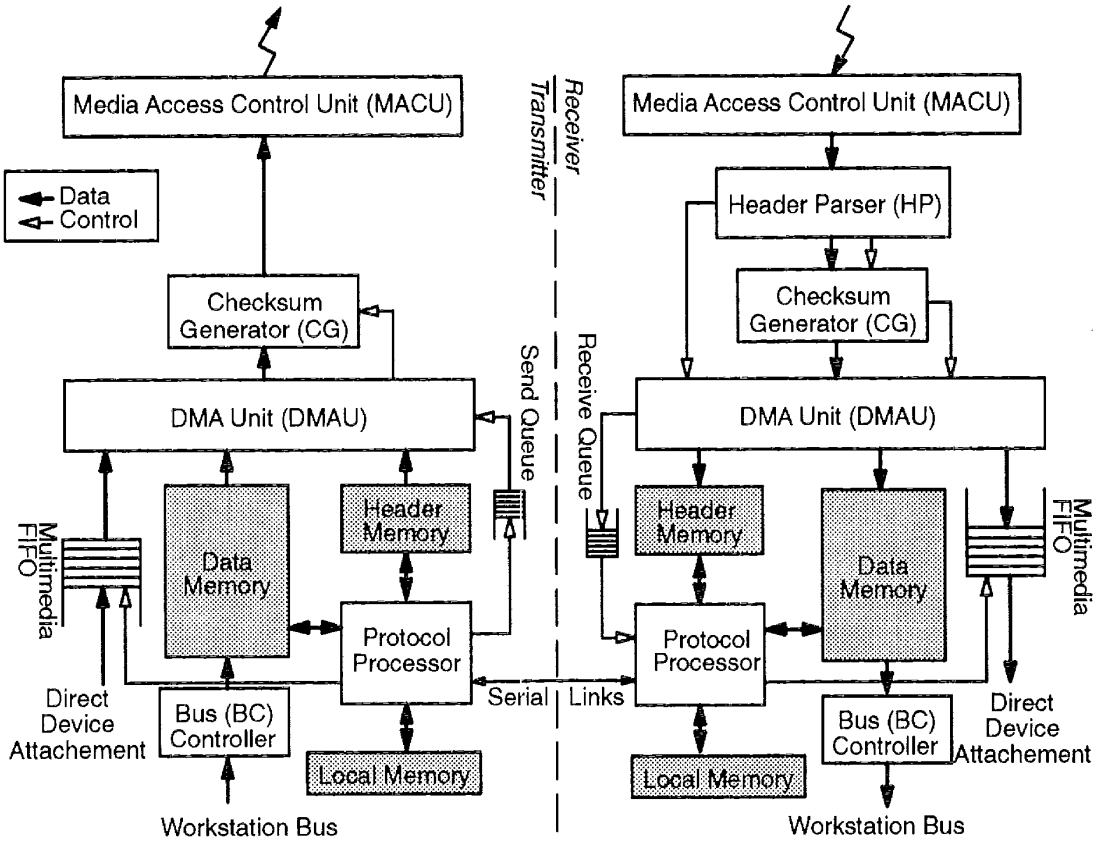

Figure 1. MPA Architecture

Media Access Control Unit (MACU): The MPA is designed to be connected to any high–speed network. The design of the MAC is beyond the scope of this paper. [Traw 91] for example describes an interface to the Aurora ATM network.

Header Parser (HP): The HP is similar to the ProtoParser[1] [Chin 92]. The HP detects on the fly the protocol type of an incoming packet and extracts the relevant header information. This information is forwarded to the DMA Unit and the Checksum Generator.

Checksum Generator (CG): The CG is triggered by the HP to calculate the appropriate checksum or Cyclic Redundancy Check *(CRC)* for the packet on the fly. The algorithms are implemented in hardware and selected by decoding the HP signal. On the sender side the CG is triggered by the DMA unit. [Birch 92], for example, describes a programmable CRC generator which is capable of processing 800 Mb/s.

The Protocol Processor T9000: The selection of the inmos[2] T9000 [inmos 91] is based on our good experience with the transputer family of processors in the PPE. The most significant improvements of the T9000 over the T425 for protocol processing are faster programmable link interfaces, a faster memory interface, and a cache. The serial message passing link provides a transmission speed of 100

1. ProtoParser is a trademark of Protocol Engines, Inc.
2. inmos is a trademark of INMOS Limited.

Mb/s plus a set of instructions to use the links for control purposes. The peek and poke instructions issue read and write operations in the address space of the second transputer connected to the other end of the link. These commands allow distributed 'shared memory' between transputers. Two transputers may allocate a block of memory at identical physical addresses in their local memory. Whenever a value is written into the local copy of the data structure, the address of the variable and its value are also sent via a control link to the second transputer.

The Memories: The memory is split into dedicated parts for each flow of data through the MPA to lower memory contention and to provide high bandwidth to those components that access the memory most. The following memory split is used:

***Header memory*:** stores the protocol headers. Fast static memory operating at cache speed is used to avoid wait cycles.

***Data memory*:** stores the data part of the packets. Inexpensive video memory *(VRAM)* is used. The serial port of the VRAM provides guaranteed access via the DMA Unit to the network. The parallel port of the VRAM is used in normal processing by the Bus Controller only. The processor can accesses the parallel port, e.g. for exception handling.

***Local memory*:** stores the program code of the processor and the control information of the connections.

***Multimedia FIFO*:** stores multimedia data and is the interface to a multimedia device. It can be controlled by the processor for synchronization with asynchronous data streams. Multiple multimedia FIFOs can be arranged in parallel.

The design does not employ physically shared memory between the transmitter and the receiver, because the implementation costs are too high compared to a software implementation using transputer links.

Memory Access; Processor to	Memory Type	Average Access Time
Data Memory	Video RAM	60 ns
Header Memory	Static RAM	30 ns
Local Memory	Dynamic RAM	60 ns

Table 1. Memory Access Time

Direct Memory Access Unit (DMAU): The DMAU directs the in– and outgoing data streams to the correct destination. The DMAU splits an incoming packet into its header and data part and moves the parts to the respective memories. A pointer to the header structure is written to the receive queue. To send a packet the DMAU gathers the data from the data memory and the header from the header memory. For multimedia traffic the data are gathered from the multimedia FIFO. The memory buffers are handled in a linked list format. The DMAU handles this linked list in hardware and thereby off–loads part of the memory management from the protocol processor.

Bus Controller (BC): The BC is a programmable busmaster DMA controller. It provides a small FIFO and a table for DMA requests. The FIFO contains a pointer to the linked list of source data and a connection identifier. The BC determines the destination memory address through the connection identifier in the table. The list format is the same for the BC and the DMAU. In the transmit BC the host writes to the

FIFO and the protocol processor to the table. In the receive BC the protocol processor writes to the FIFO and the host to the table.

2.3 Packet Processing

Packets are processed in a hardware pipeline which runs at network speed. The pipelined packet processing is shown in Figure 2.

Receiver

The MACU receives cells from the ATM network, processes the AAL, and triggers the receive pipeline to start. The receive pipeline is run by the DMAU. The HP and the CG process the data as they are copied from the MACU to the destination address in the memories or to the multimedia FIFO. The HP extracts the relevant header information from the packet and forwards the information to the DMAU and the CG. The CG uses this information to detect which checksum or CRC it must calculate. The CG calculates the checksum on the fly as the packet is copied by the DMAU and forwards the result to the DMAU. The DMAU uses the information generated by the HP to determine the format and the connection of the packet. For a multimedia connection the DMAU removes the header from the packet and writes the data part to the Multimedia FIFO.

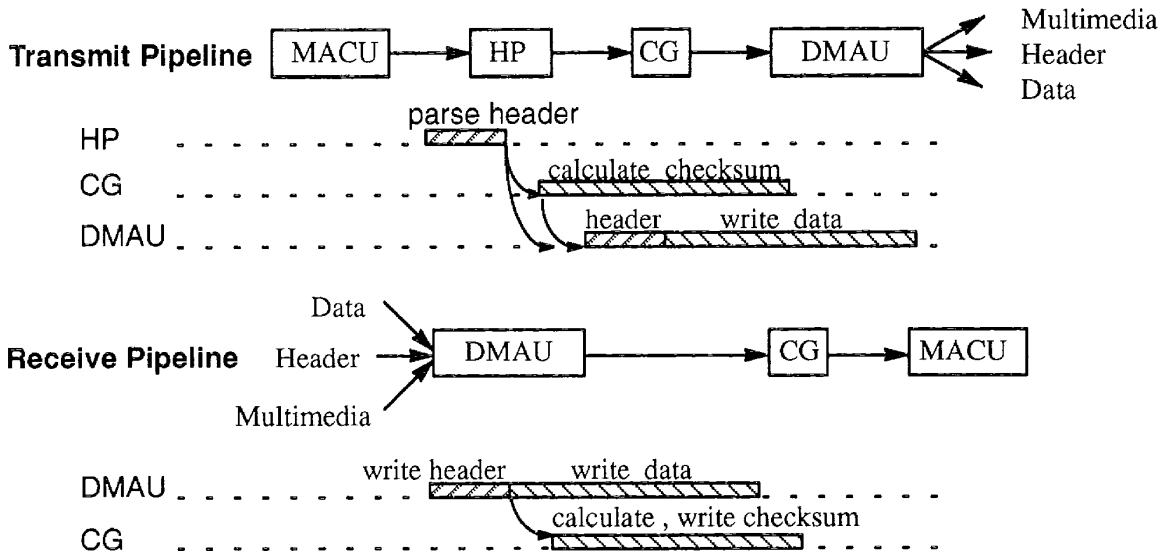

Figure 2. Pipelined Packet Processing

For asynchronous traffic the DMAU writes a structure to the header memory which holds the header, the header information extracted by the HP, the checksum calculated by the CG, and the pointer to the data in data memory. The data part of the packet is written to the data memory. The DMAU writes a pointer to the header structure to the receive queue. The protocol processor is then responsible for processing of the header structure. The addresses of free buffers in header and data memory are obtained from a linked list of free buffers.

If the HP does not recognize a packet header the entire packet is written to the data memory. In this case, the protocol processor performs the processing of the packet header in data memory. For a new connection the protocol processor builds up the connection and programs the HP to recognize the header.

Sender

On the transmit side the protocol processor builds the layered protocol header in the header memory. It builds a structure which holds the pointers to the header and to the data, the length of the header and data, and the connection type. This structure is written to the send queue. The DMAU runs the send pipeline. It interprets the structure and forwards the connection type to the CG. The CG calculates the checksum on the fly as the packet is written to the MACU memory. In the MACU the packets are stored to process the AAL and to segment the AAL frame into ATM cells. Once the CG has finished, it writes the checksum to its position in the packet frame and triggers the MACU to send the packet. Once the packet is sent, the DMAU appends the buffers to the corresponding free–lists.

3. Protocol Processing

3.1 Transport Protocol Stacks

Transport protocol processing on the MPA in the example of TCP/IP is shown in Figure 3. The socket layer is split into a lower half serving TCP and an upper half which interfaces to the application. A more detailed description of our parallel TCP/IP implementation can be found in [Rütsche 92].

Sending a packet: The send data are in a buffer allocated on the host. The application creates a socket and establishes a TCP/IP connection. The socket send call triggers the **`write`** process which copies the data to the MPA and gives the control over the data to **`xtask`**. The **`xtask`** process is then responsible for the transmission and possible retransmissions of the data. It builds the TCP packet and forwards the pointer to the packet to **`ip_send`**. Here the IP header is placed in front of the TCP segment. Then the pointer to the packet is written to the send queue and the DMAU sends the packet via the MACU to the network.

Receiving a packet: Upon receipt of a packet the DMAU writes the pointer to the packet to the receive queue. **`ip_demux`** reads the receive queue, checks the header and, if no error or exception occurred, forwards the packet to **`tcp_recv`**, or else to **`icmp_demux`**. The **`tcp_recv`** process analyzes the TCP header and calls the appropriate handler function for a given protocol state. To send an acknowledgement or a control packet **`tcp_recv`** uses a Remote Procedure Call *(RPC)* to the transmit side. Correctly received packets are appended to the receive list. **`rtask`** forwards the received segments to the application process which is blocked in the socket receive procedure. This procedure then fills the user buffer with data from the receive list.

3.2 Multimedia Protocols

For multimedia traffic often real–time data and continuous data streams are required. ST–II is a good example of a protocol that supports this type of traffic. After a connection has been set up, the reception of data packets requires only the detection of the connection and the calculation of the header checksum. For sending, the header can be built only once in the header memory and then used for all the data packets of the connection. These functions are done in a hardware pipeline by the HP and the CG (see Figure 2). The DMAU scatters and gathers the header and the data without any interaction of the protocol pro-

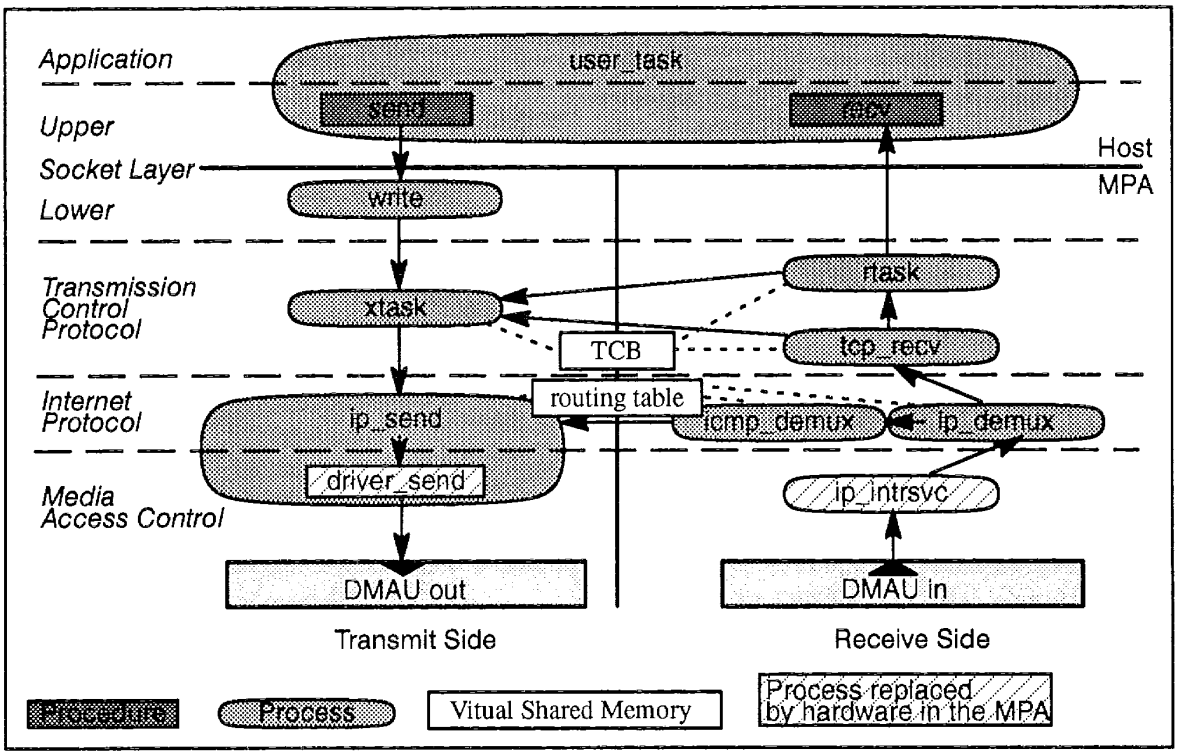

Figure 3. Parallel TCP/IP Implementation

cessor. Therefore real–time processing of ST–II at the network speed of 622 Mb/s is possible. The interaction of the processor is only required to handle the Stream Control Message Protocol *(SCMP)*, which is responsible for creating and keeping most of the state in a ST–II protocol connection.

4. Performance Estimation

4.1 The Method

The measurements of the TCP/IP implementation on the PPE were used and adapted to the MPA architecture [Rütsche 92]. The execution times of program segments accessing local memory $T_{localmem}$ and data memory $T_{datamem}$ are calculated from the execution times on the PPE minus the time saved by the hardware devices replacing software functions. These execution times are multiplied by a speedup factor S, which is determined by the memory timing and the faster processor, and summed to get the execution time T_{MPA} on the MPA.

$$T_{MPA} = T_{localmem} * S_{localmem} + T_{datamem} * S_{sharedmem} \quad (1)$$

This approach is valid for protocol processing because most operations are memory operations to build a header or to compare header data with expected data in a control block. The control information is built with simple arithmetical and logical operations such as *add*, *multiply*, *and*, *or* etc.

4.2 Cost of Basic Operations

The transputer T9000 is downwards compatible with the T425 used in the PPE. The main differences are a higher link speed of 100 Mb/s, a sustained performance of more than 70 MIPS and a peak rate of

200 MIPS. A function call or process switch costs less than 1 µs. The sustained MIPS rate improves the performance at least seven times for the simple protocol processing operations. The memory functions are determined by the memory access time shown in Table 1. The access time to the header memory decreases by a factor of 19, the access time to the data memory by a factor of 9.5. Therefore the full power of the processor can be utilized, and the typical speedup factor $^1/_{10}$ [inmos 91] can be assumed for $S_{datamem}$. The speedup factor to the local memory is determined by the processor speedup, because the local memory and the cache provide an optimal memory interface to the processor. We assume a conservative speedup factor of $S_{localmem} = {}^1/_7$ for the local memory.

The costs of basic operations for protocol handling are listed in Table 2. The connection detection and the calculation of a CRC or a checksum are implemented in hardware. These operations run at network speed as the data is clocked in from the MACU. The T9000 improves the implementation of the distributed shared memory, because the peek and poke calls are already implemented in the microcode. Therefore the costs of the distributed 'shared memory' are only the issuing of a peek or poke instruction.

Processor Operation	Number of Processor Instructions	Estimated Time in ns
Queue read / write	3	180
Linked List add/remove	4	240
Distributed shared memory read/write	3	300 + size[word] * 460
Connection detection	0 (implemented in hardware)	network speed
Checksum/CRC calculation	0 (implemented in hardware)	network speed

Table 2. Cost of Basic Operations

4.3 TCP/IP Performance

The performance of TCP/IP is evaluated using the measurements of our TCP/IP implementation on the PPE. The TCP stack, the socket layer and a test application run on the MPA. The cost of the single processes of TCP/IP is calculated using (1). Table 3 lists the execution times on the PPE and the calculated execution times on the MPA. In the PPE implementation of the IP processes 60% of the accesses go to the shared memory, in **`tcp_send`** 47% and in **`tcp_recv`** 10%. In the MPA architecture all of these accesses are replaced by accesses to the header memory. In the **`user_task`** and the **`ip_intrsvc`** most processing is replaced by the list handling in the DMAU and the BC. However the write process is still needed to control the send queue. All copy operations are implemented in the BC. The **`ip_demux`** process is supported by the HP, which extracts the header information.

	PPE	MPA
Process (Procedure) on Receiver	**µs/Packet**	**µs/Packet**
tcp_recv	235	25
user_task (socket_recv/copy)	31+ 0.545µs/word	1
ip_intrsvc	9	–
ip_demux	23	1.7
Process (Procedure) on Transmitter	**µs/Packet**	**µs/Packet**
write	30+ 0.545µs/word	4.3
tcp_snd_data	147	9.8
ip_send	23	1
driver_send	17+ 0.27µs/word	0.7
Access to *Shared* Memory (poke call)	18.6+2.4µs/word	0.3 + 0.46µs/word

Table 3. Process Execution Times

The TCP/IP process pipeline for bidirectional traffic is shown in Figure 4. The throughput is determined by **`tcp_recv`** and **`ip_demux`**, which add up to 26.7 µs. The transmitter is less costly than in the PPE implementation because of the faster network speed.

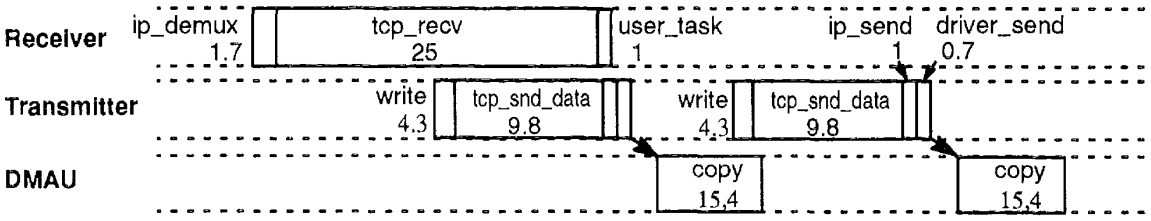

Figure 4. TCP/IP Processing

The throughput calculated for unidirectional TCP/IP traffic between two MPA systems is 35720 TCP segments/s. For bidirectional traffic, the throughput is 20290 segments/s, if for every eight packets an acknowledgment packet is sent. The throughput numbers are independent of the packet size because all data copying is done in hardware overlapped to protocol processing. However for large packets, the network becomes the bottleneck (4 kByte segments would result in more than 1 Gb/s). If we assume a segment size of 1024 bytes, the throughput is 292 Mb/s which is more than most current workstation can handle.

5. Conclusion

The separation of isochronous and asynchronous traffic permits processing of isochronous multimedia traffic at the network speed. The separate header and data memories provide optimized access to the critical components. The parallelism between the transmitter and receiver is the most suitable form of moderate parallelism to speed up protocol processing and to lower hardware contention.

The hardware components such as the HP, CG and DMAU can be built to process one Gb/s. The MPA could then process multimedia streams at one Gb/s. A multimedia application could for example look as

follows. The multimedia interface handles 700 Mb/s in hardware. The protocol processors perform transport protocol processing at a throughput of 300 Mb/s and forward the data to the workstation. This split of the bandwidth would make sense, because applications which require reliable transport connections in the Gb/s range do not seem feasible in the near future because of the I/O bottleneck of the workstations. However transport protocol processing at one Gb/s is already possible with an architecture based on the MPA.

The efficient attachment of the subsystem to the workstation is yet unsolved. To take advantage of the high bandwidth available on the network and on the MPA, the current workstation hardware and software interfaces must be changed. Designing these interfaces especially for multimedia will be one of the goals of future work.

6. References

[Birch 92] Birch, J., Christensen, L. G., Skov, M., "A programmable 800Mbit/s CRC check / generator unit for LANs and MANs", Computer Networks and ISDN Systems, Nr. 24, North–Holland 1992.

[Chin 92] Chin, H. W., Edholm, Ph., Schwaderer, D. W., "Implementing PE–1000 Based Internetworking Nodes, Part 2 of 3", Transfer, Volume 5, Nr 3, March/April 1992.

[Braun 92] Braun, T., Zitterbart, M., "Parallel Transport System Design", IFIP Conference on High Performance Networking, Liege (Belgium), 1992.

[inmos 91] "The T9000 Transputer Products Overview Manual", inmos 1991.

[Jain 90] Jain, N., Schwartz, M., Bashkow, T. R., "Transport Protocol Processing at GBPS Rates", Proceedings of the SIGCOMM '90 Symposium , Sept. 1990.

[Kaiserswerth 92] Kaiserswerth, M., "The Parallel Protocol Engine", IBM Research Report, RZ 2298 (#77818), March 1992.

[Rütsche 92] Rütsche, E., Kaiserswerth, M., "TCP/IP on the Parallel Protocol Engine", Proceedings, IFIP Conference on High Performance Networking, Liège (Belgium), Dec. 1992.

[Topolcic 90] Topolcic, C. (Editor), "Experimental Internet Stream Protocol, Version 2 (ST–II)", RFC 1190, Oct. 1990.

[Traw 91] Traw. B., Smith, J., "A High–Performance Host Interface for ATM Networks", Proceedings ACM SIGCOMM '91, Zürich, Switzerland, Sept. 1991.

[Steenkiste 92] Stennkiste, P., et al., "A Host Interface Architecture for High–Speed Networks", Proceedings IFIP Conference on High Performance Networking, Liège (Belgium), Dec. 1992.

[Wicki 90] Wicki, T., "A Multiprocessor –Based Controller Architecture for High–Speed Communication Protocol Processing", Doctoral Thesis, IBM Research Report, RZ 2053 (#72078), Vol 6, 1990.