

The BiProcessor: A merger of two architectures

by C. Berggren

The BiProcessor consists of an IBM System/370™ and a Personal System/2® and merges these two IBM architectures into a synergistic relationship. The two processing environments are connected by an internal high-speed pipe that allows each system to take advantage of the other's strengths as well as developed products, both hardware and software. This paper describes this closely coupled heterogeneous multiprocessor and its capability of concurrent coprocessing. Also discussed are the implementation, coupling architecture, and design considerations of the BiProcessor and its development objectives. Some of the intended applications are host off-loading of communications protocol processing, use as an applications coprocessor, and service as a platform for future clustering technology.

The BiProcessor is a heterogeneous multiprocessing system that has been developed to provide a platform for allowing nontraditional system solutions. It allows new applications to exploit the inherent goodness of proven architectures, even if they do not fit or conform completely to existing molds. The BiProcessor is different from the combined architectures of the past in that both processing environments are fully visible and accessible to the user. Each processor has access to the other's resources. In contrast, earlier configurations typically consisted of embedded controllers that were dedicated to a specific task with the controller's personality hidden. The BiProcessor is particularly suited for use as a toolkit for the experienced system developer who has a unique application or novel approach to system solutions. The BiProcessor may be thought of as a set of building blocks to support

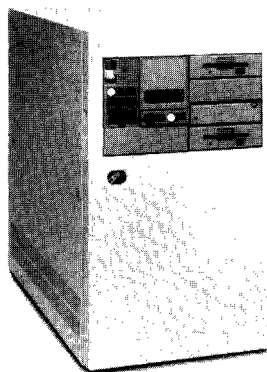
the rapid customization of computer systems and the easy creation of portable applications.

Several categories of new applications are envisioned for this hybrid multiprocessor, with focus on extending the IBM System/370* to other processing environments that exist today and as they may be realized in the near future. Development of the BiProcessor was undertaken with the recognition that heterogeneous coprocessing is likely to be a growing trend in future computational systems. The complex instruction-set architectures used in many of today's general-purpose processors are expected to be complemented with such coupled special-purpose computational units as transputers and neural processors. The design of the BiProcessor's interprocessor coupling mechanism reflects considerations of these future trends.

There was no existing architecture available for the coupling mechanism between heterogeneous multiprocessors, as we began development. Neither was there prior experience related to the merging of different IBM architectures. To a large extent, the BiProcessor design was driven by our vision of a product that would support future distributed object-oriented, event-driven, real-time applications.

©Copyright 1992 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

Figure 1 BiProcessor system enclosure



This paper discusses the intended uses for this hybrid architecture, both as the uses have already materialized and as they are envisioned for the future. Some of the applications addressed are synergistic coprocessing, protocol off-loading, adapter openness, and technology infusion and clustering. This paper also describes the implementation of the BiProcessor and the design of the coupling facility, including some of the design tradeoffs that were made during the development of the BiProcessor. Possible future improvements are also discussed.

System overview

The Micro Channel* 370 BiProcessor consists of two computer systems merged into a single small package containing a System/370 and a Personal System/2*. Each processor has its own private memory, DASD and hard disk, input/output, and separate controls. The two systems are connected by an internal high-speed pipe. Figures 1 and 2 show pictorial views of the system: the physical enclosure and a high-level schematic of the system internal block diagram.

Hardware. The System/370 is a fully functional System/370 processor, with capabilities functionally equivalent to an IBM 9370 CPU. The Personal System/2 (PS/2*) is a 20 MHz PS/2 Model 80 derivative, functionally equivalent to the PS/2 Model

8580-321. One of the eight available Micro Channel slots in the PS/2 processor is used by the System/370 coupling facility. The remaining seven slots can be used for other Micro Channel devices. Without the coupling facility between the System/370 and PS/2, each processor could be used independently. The two processing environments have full operating autonomy, apart from sharing a common power supply.

The coupling hardware connecting the two processing environments consists of a bus-to-bus protocol converter card that translates System/370 internal bus protocols to Micro Channel protocols and vice versa. The card plugs into a standard Micro Channel slot. The card houses Micro Channel mapped memory accessible to the PS/2 and a microprocessor that moves data between the System/370 and Micro Channel address spaces. The microprocessor can read and write all of the System/370 memory, a capability that is used for the high-speed pipe implementation.

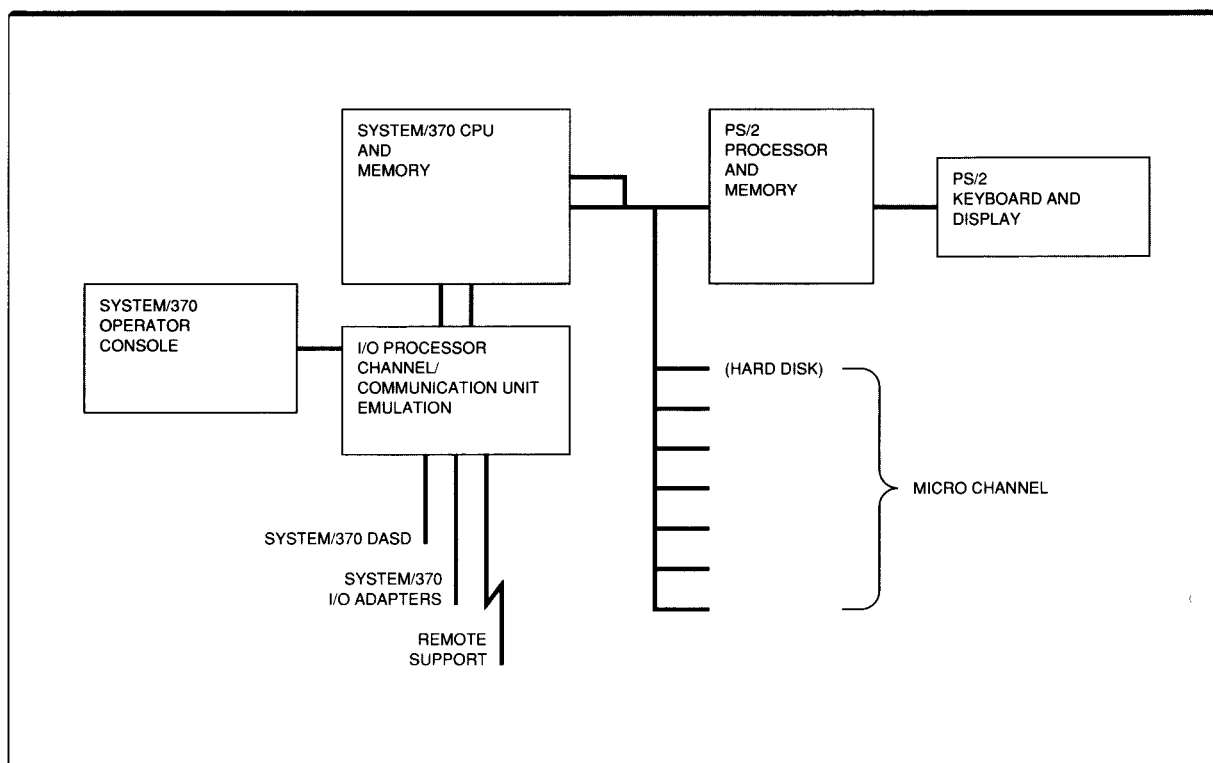
Software. Since the PS/2 is a fully functional PS/2, it can run all operating systems and applications intended for that platform. Several operating system combinations are supported for the BiProcessor interprocessor coupling facility. The PS/2 may execute DOS, OS/2*, AIX*/386, or proprietary operating systems such as Syzygy's HBX/370**. On the System/370 side, coupling support has been developed for the virtual machine (VM) and Virtual Storage Extended (VSE) operating systems. Applications that were developed for these operating systems will execute unchanged on the BiProcessor.

Standard high-level application programming interfaces (APIs) are provided to facilitate ease of use, and a low-level transport interface is made available for customized protocol implementations.

Communications architecture. The communications architecture of the coupling facility between the two processors is based on a tightly coupled multiprocessor model, with peer-to-peer interaction. Interprocess communications are implemented as if the processes were executed in the same environment.

The architecture is layered with three distinct interfaces, as shown in Figure 3. It provides the capability for PS/2 and System/370 programs to

Figure 2 BiProcessor hardware functional diagram



communicate directly at two interface levels—a high-level user access point and a low-level transport interface.

The high-level user access point in the communications architecture is the application programming interface (API), as shown. A protocol stack is built on top of the high-speed pipe implementation. The design is modular, so that any specific protocol can be supported, for example, the Systems Application Architecture* (SAA*) Common Programming Interface-Communications (CPI-C),¹ Sockets, Network Basic Input/Output System (NetBIOS), and Transmission Control Protocol/Internet Protocol (TCP/IP).

The low-level user access is directly to the BiProcessor high-speed pipe, shown as the low-level programming interface (LLPI) in Figure 3. It provides a low-overhead data send and receive function and rudimentary routing. The functional components of the high-speed pipe consist of data transport and inter-CPU signaling mechanisms, in-

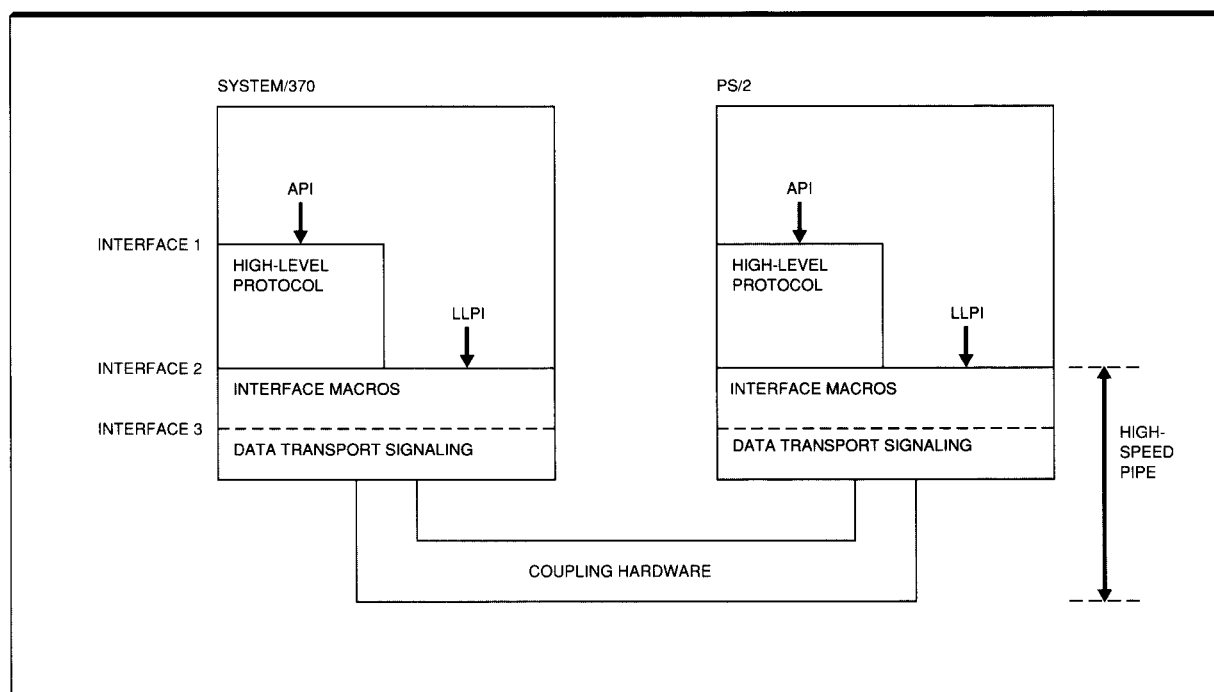
terface macros, and the supporting hardware, shown as everything below the LLPI interface. More than one logical pipe can be supported concurrently over the same set of hardware.

Intended applications and system realizations

The BiProcessor allows System/370 and PS/2 applications to communicate directly in a closely coupled multiprocessing configuration, thereby providing each with access to the other's resources. The BiProcessor system design objectives were to satisfy the needs of several major categories of intended applications that exploit this capability, as described next.

Applications coprocessing. An applications coprocessing category is intended to be one of a synergistic relationship between System/390- and PS/2-based applications, where one adds value to the other. These applications may communicate via either the CPI-C or LLPI interface, and specific

Figure 3 BiProcessor communications architecture



interface selections depend on the intended application's need and the operating systems used. Both LLPI and CPI-C can be supported concurrently, if desired. See Figure 4.

Coprocessing examples are numerous, many of which have traditionally been connected via local area networks (LANs). Included are PS/2 LAN servers, such as the IBM LAN Server, Microsoft's LAN Manager**, and Novell's NetWare**. Some more recent implementations include the coupling of Series/1 capabilities to the System/370 via PS/2-based Series/1 emulators as shown by the Syzygy implementation, as described later in this paper, and high-performance mathematics coprocessing applications via Micro Channel Reduced Instruction Set Computer (RISC) processor cards.

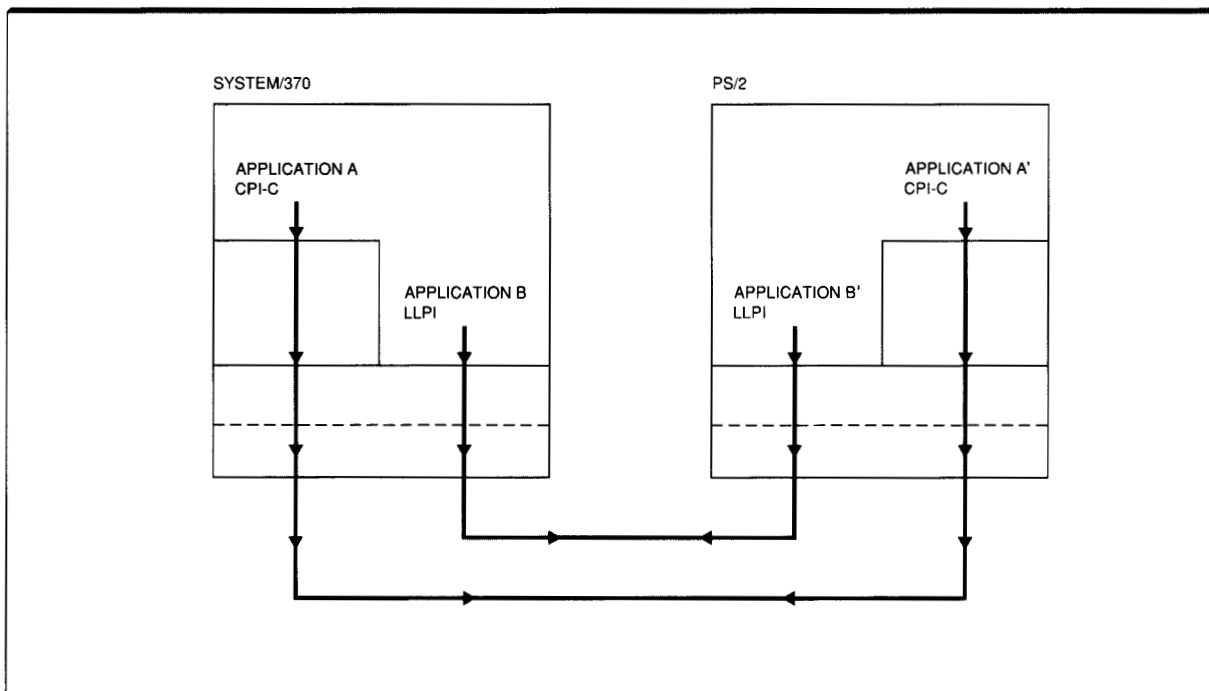
Using BiProcessor implementations, PS/2-based LAN servers can exploit System/370 functionality, thereby extending existing client-server offering capabilities. These extensions include systems management and control, data sharing and management (including backup and archiving), network monitor and control, and software distribution. The extensions also enable full Systems

Network Architecture (SNA) wide-area communications participation. PS/2 applications can benefit by utilizing Virtual Telecommunications Access Method (VTAM*) and thus gain access to the SNA network and remote resources.

Another client-server computing application that used the BiProcessor was demonstrated by Phaser Systems, Inc. at NetworkWorld. Phaser used the BiProcessor to demonstrate their NetWare for VM** product and how it provides for routing Novell NetWare's IPX/SPX across an SNA network. IPX/SPX are LAN protocols developed by Novell. The PS/2 was used as a communications gateway to provide access to the System/370-based NetWare for VM component to LAN-attached workstations.

A prime example of the platform's use as an applications coprocessor was developed by Syzygy Communications, Inc. in partnership with IBM. The objectives were to provide a Series/1 replacement and growth path using newer technology. IBM developed the pipe support for the VSE operating system, and Syzygy integrated Hummingbird** on the PS/2 side allowing it to communicate

Figure 4 Applications coprocessor configuration



across the pipe. Hummingbird is a Series/1 EDX (Event-Driven Executive) emulator that was developed by Computer Information Enterprises for IBM PCs and PS/2s. (EDX is a Series/1 operating system.) Configuration details are shown in Figure 5.

The Syzygy exploitation of the hybrid aspects of the BiProcessor has both immediate and long-term benefits. Existing Series/1 EDX applications can be ported to this interim platform, thus preserving developed applications software. At the same time, new applications can be System/370-based and use efficiently the capabilities of new technology at lower price and better performance. The System/370 with the VSE operating system offers an unlimited migration path from the Series/1. (The Customer Information Control System [CICS] transaction environment is ideally suited for Series/1-type applications.) This coprocessing implementation solves both Series/1 capacity and Series/1 future availability problems.

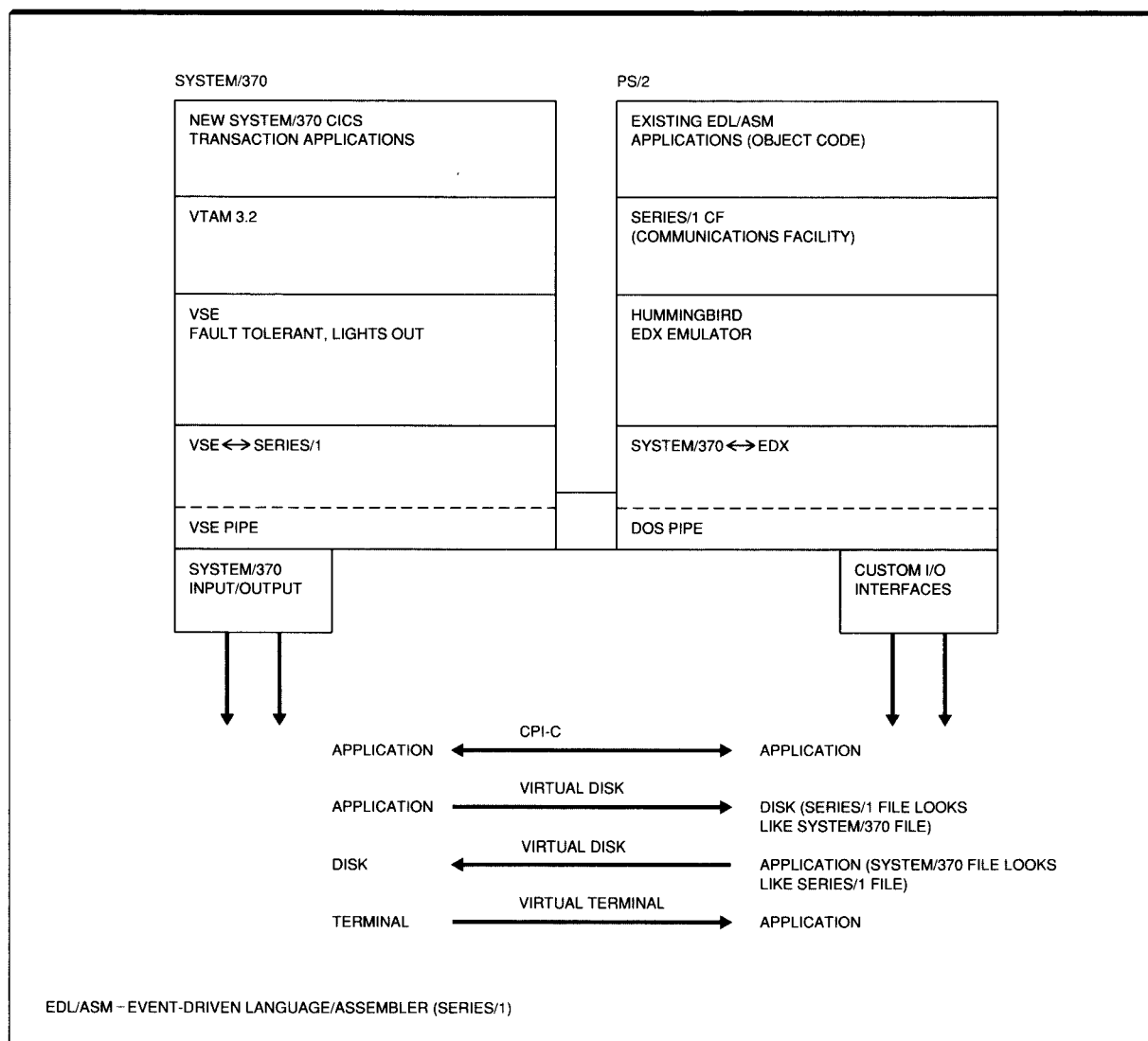
Protocol off-load. A second major intended category of BiProcessor usage was to open the System/370 to new processing environments through

use of the PS/2 as a protocol engine. This is the outboard processing of communication protocols on the Micro Channel-based processor and adapter cards.

To communicate with the outside world, the System/370 usually uses either SNA or TCP/IP protocols. A System/370 application wishing to communicate with the outside world communicates with VTAM, for instance, which in turn does all the communications on behalf of the requesting application by executing the SNA protocols. By providing a direct path into the System/370, any PS/2-based communications protocol can be used to access System/370 applications with the BiProcessor. Figure 6 shows the BiProcessor in a protocol engine configuration. The System/370 executes user applications, and the protocol processing is done outboard. The System/370 application gets access to the protocol-specific API via the LLPI interface and high-speed pipe, with the communications verb set being pulled through.

A database application executing on the System/370 can be opened up to a large segment of

Figure 5 Syzygy Series/1 replacement

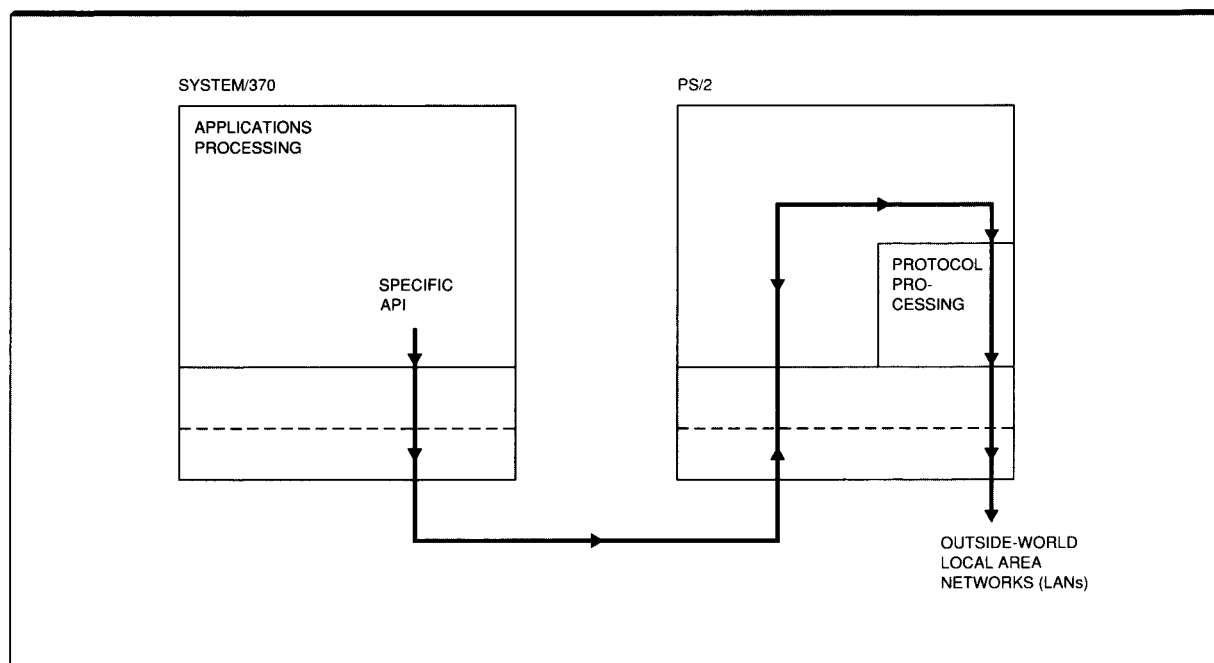


LAN-attached PCs by putting NetBIOS on the PS/2 side. NetBIOS is the prevalent method of communications for existing PCs and PS/2s. In another configuration, AppleTalk** protocols can be put on the PS/2 to enable Macintosh** computers access to the System/370. In still another configuration, DECnet** can be used if VAX** connectivity is desired. Another advantageous implementation of this protocol off-load is to substitute the System/370-based TCP/IP protocols, common with governments and universities, with the AIX PS/2 equivalent. This does not necessarily provide

new function for the System/370 operating systems that already have TCP/IP capability, but it improves the performance of existing function.

Viability of the BiProcessor protocol engine concept was proved by the ease with which the manufacturing messaging services (MMS) support was developed for the BiProcessor. MMS is the standard application-level interface for manufacturing, and it provides data collection and control for factory devices on the plant floor. MMS uses the seven-layer communication protocols defined by

Figure 6 Protocol engine configuration



the International Standards Organization's Open Systems Interconnection (OSI) Reference Model and the Manufacturing Automation Protocol (MAP). By integrating a PS/2-based MAP adapter and OSI protocols for OS/2 with the System/370 MMS server for VM/SP, the development of a manufacturing solution package that applies the Bi-Processor as an area or cell controller for the manufacturing floor was achieved in a very short time and with little effort.

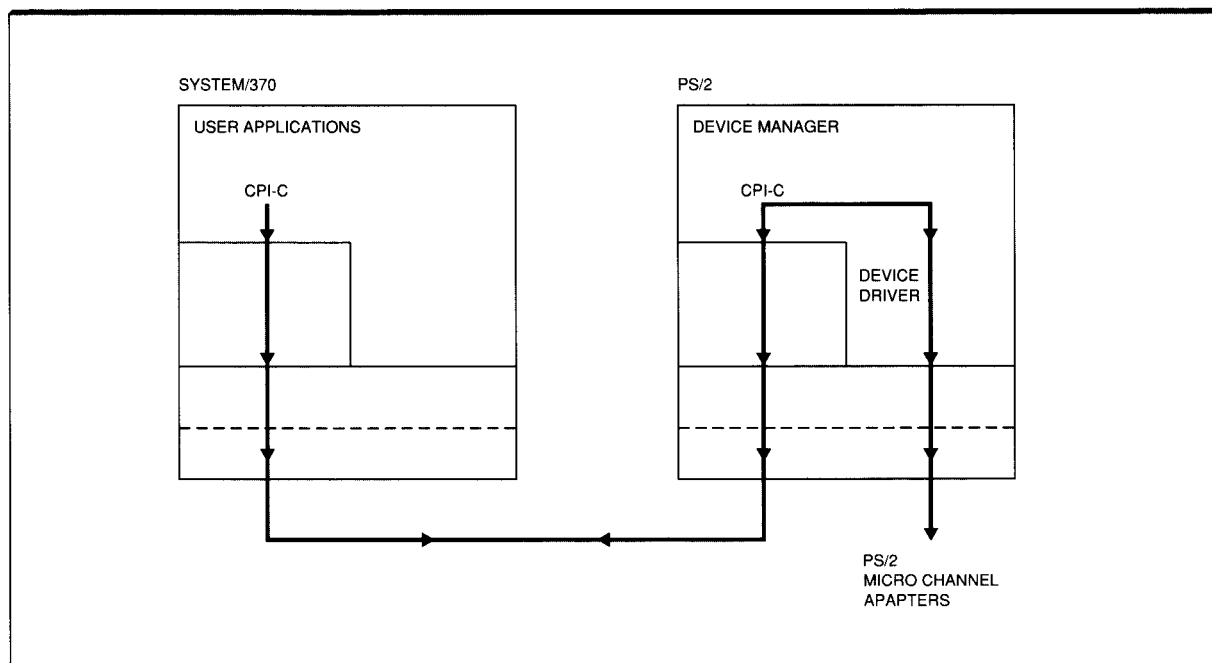
There are numerous advantages to the protocol engine application of the BiProcessor concept. It makes it possible for the System/370 to exploit a multitude of PS/2-family-developed protocols, as well as private low-overhead "roll-your-own" implementations. The protocol engine application provides for off-loading the System/370, thus allowing the host to dedicate its processing power to mainframe applications, while exploiting the PS/2's less expensive MIPs and hardware for communications protocol processing. In addition, it has a decided performance benefit in that the communication protocols can be processed in parallel with the application, rather than serially with the added processing overhead of context switching.

Adapter openness. Adapter openness can be viewed as a variation on the protocol engine application. However, rather than using the PS/2-developed communication protocols, Micro Channel adapters are being made available to the System/370. Besides IBM, there are several hundred independent vendors that develop adapters for the PS/2 family of processors. Adapter examples are manifold and include optical disks, CD ROMs, FAX adapters, scanners, and many more.

The adapter openness is implemented by a data-manager application on the PS/2 that controls the specific device on the Micro Channel and passes the data to and from the System/370 application, as shown in Figure 7. Data are passed from application to application, thus hiding the adapter specifics to the System/370, using either the CPI-C or LLPI interface depending on the specific needs.

By allowing the System/370 to interface to any adapter card developed for the Micro Channel without having the device defined to the System/370, the System/370 is given a previously unknown flexibility in available adapter options and in ease of implementation.

Figure 7 Adapter openness configuration



Network router. Yet another communications example is that of configuring the BiProcessor with the PS/2 as a network router. In this configuration, upper-layer protocols are resident on the System/370. The PS/2 functions as a low-level router to the data-link layer—for example the IEEE 802.2—and through it to the outside world, as shown in Figure 8.

The VM CPI-C support developed for the BiProcessor contains network routing support. Thus VM applications may also use CPI-C to communicate with PC-DOS, OS/2, or AIX workstations. These are attached via a token ring or Ethernet adapter to the PS/2 side of a BiProcessor that is executing OS/2. The following capabilities are provided:

- A workstation application (PC-DOS, OS/2, or AIX) can allocate a conversation with a System/370 CPI-C application, and through VTAM to remote hosts.
- A System/370 CPI-C application can allocate a conversation with the OS/2 server, an OS/2, PC-DOS, or AIX workstation, or another System/370 system.

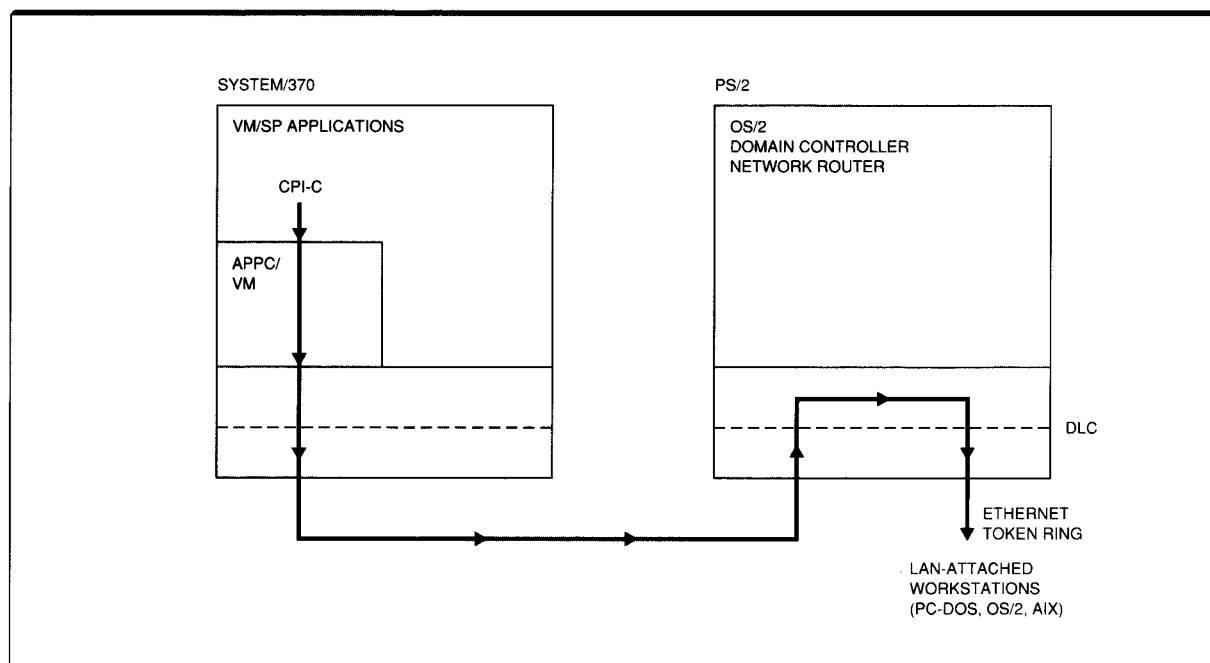
The BiProcessor implementation of the CPI-C support for VM is discussed in References 2–5.

Coupling implementation

BiProcessor communications architecture. The communications architecture of the coupling facility between the two processors is layered with three distinct interfaces, as shown in Figure 3. The functional components are: (1) the coupling hardware previously discussed, (2) data transport, (3) inter-CPU signaling mechanisms, (4) low-level interface macros, and (5) upper-layer protocols in support of the high-level interface, as described in later sections of this paper.

Data transport. The BiProcessor data transport mechanism is implemented through buffer-pool technology, which is an interlocked memory management scheme with shared direct access to memory. Data transfer is not under the control of a System/370 channel program nor are the data presented to the PS/2 in a serial manner as a result of IN/OUT instructions. Instead, controls for shared buffer pools are located in a communica-

Figure 8 Network router configuration



tion area in System/370 memory, with a separate communication area for each logical pipe.

Definitions for circular buffer pool queues are located in this communication area. The buffer pool queues contain memory pointers to storage locations into or out of which data are to be transferred, thus minimizing data moves by moving the data directly to and from application space. The definitions include information on storage locations of the buffer pointer lists, number of entries, and size of the data buffers that the queue entries point to.

The BiProcessor pipe is full duplex. Thus there are two circular buffer pool queues defined in the communication area for each logical pipe, one for each direction of data transfer—System/370 to PS/2 and PS/2 to System/370. One processor controls the read pointer, and the other pointer controls the write pointer in a queue, with the read pointer chasing the write pointer. The content of the pointers may be updated dynamically, thereby allowing the application flexibility, with regard to memory locations of the data buffers.

The communication area also contains control fields for the buffer pools. These control fields

determine where in the circular buffer queue list the current read and write pointers are located. These pointers indicate the locations of next available empty input buffer or data buffer to be transferred out. Control fields also indicate whether a queue is empty or full (or neither empty nor full), information which is used to determine signaling need.

Information with regard to buffer list formats is contained in the communication area control fields. The high-speed pipe supports transfers of packets larger than the buffer size specified in the control fields. The buffer list format fields provide the controls to allow concatenation of individual data buffers. These buffers do not have to be located in contiguous memory.

Inter-CPU signaling. In addition to the buffer pool queue and controls definition described above, an inter-CPU signaling mechanism is required for the operation of the BiProcessor pipes. The inter-CPU signaling is used for the following:

- *Communication area definition.* The storage location of a specific communication area and its size must be communicated by the System/370

to the PS/2 agent responsible for buffer pool queue manipulation. This is done when a pipe is initially enabled and after a reset and/or program load.

- *Pipe termination.* An application or detected errors may close a pipe.
- *Buffer pool queue state changes.* The peer processor is not interrupted, as a rule, for normal data transfers, in order to optimize data transfer performance. A shoulder-tap notification is required, however, when a circular buffer queue goes from empty to nonempty or from full to nonfull, indicating that action should be taken. If processors are speed-matched, data transfers can go on indefinitely without interruption to either the System/370 or the Micro Channel-based peer processor. This is because one processor reads from a queue at the same rate as another processor writes into it.
- *Requests for status.*

Although the design of the data transport mechanism could easily be optimized, development of the interprocessor signaling was less straightforward. With no prior implementation of peer-to-peer coupling between heterogeneous processors, there was no given mechanism to implement this interprocessor signaling. What was needed was an instruction and a corresponding asynchronous external interrupt that could be used between the two processors symmetrically with each one switching roles. This was not available to us in either system. Therefore, we evaluated possible instructions and interrupts, based on what could be made to meet our objectives within one year. After discarding several options, the choice of signaling method was narrowed down to one based on multiprocessing or to an I/O subset as described below.

The multiprocessing option was based on using the existing System/370 multiprocessing architecture for tightly coupled System/370 processors. Inter-CPU signaling for homogeneous System/370s is implemented by a Signal Processor (SIGP) instruction. The SIGP instruction and associated external interrupts with the various order codes fill the BiProcessor pipe signaling needs naturally. A logical pipe would be implemented as an emulated System/370 CPU and interact as such with the real System/370. The order codes associated with the SIGP instruction include commands that allow one processor to retrieve status from, and to start, stop, and reset the

other. This is in addition to the basic shoulder tap needed for queue manipulation.

The second option for implementing the signaling between the CPUs was based on using a thin layer of the System/370 I/O architecture to emulate subsets of the System/370 I/O channel and control unit function. The I/O-subset approach to interprocessor signaling was chosen, based on the time required for development. The multiprocessing facility was not implemented for the specific hardware, and development of the required functions would not be a trivial undertaking. Modification of the I/O architecture to meet our needs, on the other hand, was fairly simple, and we could exploit existing microcode. The System/370 I/O architecture subset was used as follows:

- *Start I/O (SIO)* with a Prepare command was used to enable the pipe communication area. The Channel Command Word (CCW) data address was used for the address of the communication area, and the data length was used to denote its size.
- *Test I/O (TIO)* was used for the shoulder tap from the System/370 to notify the PS/2 agent of a buffer queue state change. TIO was chosen because of its low operating system overhead, and performance was a major design criterion.
- *Asynchronous attention interrupt (ATTN)* was used by the PS/2 to signal the System/370 about buffer queue state changes. ATTN plus unit check was used to signal errors or the closing of a pipe.
- *Start I/O (SIO)* with a Sense command was used by the System/370 to retrieve status from the PS/2.

The chosen approach to interprocessor signaling performed well, and it can be extended in the future. With the layered approach to the design and specifics of the pipe implementation details hidden, there is nothing to preclude changing it later when standard coupling mechanisms emerge for heterogeneous multiprocessors, without impact on developed application software.

When the implementation choice was made, the standard System/370 I/O architecture was evaluated at length for applicability to the BiProcessor concept, as were different channel protocols. Although it was recognized that many future applications could be developed for a System/370 with a channel-attached PS/2 and that some of the func-

tional objectives would be met with this approach to coupling (albeit with a performance penalty), the I/O approach was discarded as an option as it did not meet the BiProcessor long-term objectives for clustering and technology infusion.

Pipe operation. The following is a discussion of the features of pipe operation, which is a major function of the BiProcessor architecture.

Pipe initialization. Before data transfers can take place, a pipe has to be initialized by determining the location of its communication area. The System/370 defines a communication area. The System/370 then opens its end of the pipe by communicating the location and size of the communication area to the PS/2 agent responsible for buffer pool queue manipulation. When a pipe has been initialized by the System/370, it is in a *connection-pending state*. The pipe is not operational and enabled for data transfers, however, until end-to-end connection has been established. The application that owns the corresponding Pipe Connection End Point (PCEP) still has to open its end of the pipe, which, when completed, puts a pipe in the *operational state*.

Pipe data transfer. Data transfer is symmetric, with each partner switching roles for the opposite queue. Queue 1 is used for data flowing from System/370 storage to the Micro Channel address space, and Queue 2 is used for data flowing from the Micro Channel address space to the System/370 storage. Consequently, the System/370 is responsible for maintaining the Queue 1 input sequence number and the Queue 2 output sequence number. At the same time, the PS/2 agent process is responsible for maintaining the Queue 2 input sequence number and the Queue 1 output sequence number.

The input sequence number points to the buffer-list entry for the next available input buffer. The output sequence number points to the buffer-list entry for the next output buffer. The queue is empty if the input sequence number is equal to the output sequence number. The buffer list is circular and the output sequence number "chases" the input sequence number.

Each sequence number is a 2-byte value between 0 and 65535, which is used to locate an entry in a buffer list. The sequence number wraps from 65535 to 0. A buffer list entry is selected by di-

viding the current sequence number by the number of entries in the list and using the remainder as an index into the list. The input sequence number is incremented each time a buffer is filled, and the output sequence number is incremented each time a buffer is emptied. The queue is empty when the input sequence number is equal to the output sequence number.

A "wake-up" interrupt is reflected to the partner whenever the queue status changes from empty to not-empty or from full to not-full.

Pipe termination. The pipe interface will remain enabled until either application wishes to terminate the connection or until an error is detected and/or a reset is issued. Upon receipt of a disconnect order, all pending read and write requests are terminated for this pipe, and the application that owns the corresponding Pipe Connection End Point is notified that the pipe interface has been disabled.

Interface macros. In order to isolate the user from the specifics of pipe implementation details and to allow future upgrades without application software impact, we defined a low-level programming interface (LLPI). The LLPI is provided as a set of interface macros (Open, Close, Read, Write, and Status) supporting the communication between two cooperating application processes, each owning one end of a particular pipe realization.

The low-level interface is intended for development of private protocols by users who prefer to customize their protocol implementation and for applications with performance requirements that dictate the more direct interface. The LLPI macros, therefore, support high-speed, point-to-point data transfers between the cooperating processes, without any processing overhead for upper-layer communication protocols. That is, the macros' functions are strictly data-move operations between the two address spaces. There is no error checking on the data content or multiplexing over a pipe between different users. These functions, if desired, must be implemented by the user application. They are also provided by the higher-level CPI-C interface, which could be considered by a system developer, based on a specific application's need.

The major functions performed by the LLPI macros' communication verbs are outlined below.

- *Enable_Pipe* is the function that enables the pipe and makes it available to the calling application. If the pipe is already enabled, a return code of busy indicates that condition.
- *Get_Output_Status* obtains the current status of the outbound side of the pipe. The return value indicates the amount of data that can be accepted by the pipe queue.
- *Write_Output_Message* writes the next message to the pipe, up to the maximum message size of 32 767 bytes. The function return value is the size of the message sent to the peer application. If space is not available in the pipe queue, control is not returned until the message has been input to the queue. The *Get_Output_Status* function should be used to check for available buffers before issuing a *Write_Output_Message* request, if the application does not want to wait for a message to be sent.
- *Get_Input_Status* is the function used by the application to obtain the current status of the inbound side of the pipe. For example, this function can be used to poll for incoming messages. The return value indicates the amount of data waiting to be read.
- *Read_Input_Message* reads the next message from the pipe. (The maximum message size is 32 767 bytes.) The function return value is the size of the message received from the peer application. If there are no pending messages when the application issues a read, control is not returned to the caller, until a message is received in the queue. For polling-type transfers, the *Get_Input_Status* function is used to check for pending messages in conjunction with this synchronous read, before issuing a *Read_Input_Message* request. This assumes that the application does not want to wait for a message to be received.

We have been considering the desirability of future pipe implementations to provide additional transfer modes to complement the initial synchronous mode. Examples are the asynchronous modes in support of polling and signaling operations. For the latter, the data received interrupt the target application, and are used for the distribution of control, remote interrupts, and for event-driven applications.

- *Disable_Pipe* disables the pipe, and data pending in the high-speed pipe are discarded. An indication of *Pipe Closed* is reported to the peer application, to indicate that the pipe has been disabled. The Pipe Closed indication is also reported to all applications that have logical pipes enabled when the peer processor goes through an IPL while the pipe is operational.
- *Get_Installed_Status* determines whether the high-speed pipe device driver has been installed. If the device driver is installed, the version number of the driver is the return value. Otherwise, an error code is returned.

The LLPI API depends on the specific operating systems used on the two environments. On the PS/2 side, support has been developed for OS/2, DOS, and AIX, and the LLPI is provided by a set of interface routines that may be linked to application programs. These routines support function calls from programs compiled by means of the IBM C/2 compiler and any of the valid memory modules. To support languages other than C and to maintain consistency with the OS/2 system services API, these interface routines use the Pascal calling sequence, instead of the normal C calling sequence. Format of the PS/2 C-language LLPI interface macros, including the required parameters, are illustrated in Figure 9.

A similar macro set has been constructed for use by the cooperating System/370 program(s) for the VM/SP and VSE operating systems with callable subroutines that use System/370 assembler language, standard I/O commands, and (for VM/SP) CMS commands.

Further details with regard to the implementation and use of the BiProcessor LLPI macros are described in the BiProcessor programmer's reference manual.⁶

High-level application programming interface

The high-level user access point in the communications architecture for the BiProcessor high-speed pipe is the application programming interface (API), as shown in Figure 3. Several specific protocol options (APIs) are possible, depending on the operating systems that are executing in the two processors.

Figure 9 Format of C-language LLPI interface macros

```
int far pascal pstatus(ph);          /* Get installed status */
int far pascal popen(ph);           /* Open a logical pipe */
int far pascal pclose(ph);          /* Close a logical pipe */
int far pascal pread(ph,buffer,count); /* Read input message */
int far pascal pwrite(ph,buffer,count); /* Write output message */
int far pascal pinstat(ph);         /* Get input status */
int far pascal pinsize(ph);         /* Get input buffer size */
int far pascal poutstat(ph);        /* Get output status */
int far pascal poutsize(ph);        /* Get output buffer size */
int far pascal popensys(ph);        /* Open pipe for system use*/

int ph;                             /* Pipe handle (0-1) */
char far *buffer;                   /* Data buffer address */
int count;                          /* Data count (1 to 32767) */
```

CPI-C is the first API to be implemented. The BiProcessor design is modular, and new protocol machines that support additional APIs can be added when required.

The CPI-C interface conforms to the Common Programming Interface-Communications component of the IBM Systems Application Architecture (SAA).¹ It provides guaranteed data delivery and supports the Advanced Program-to-Program Communications (APPC) functions including message routing, error checking, and session management. The *SAA—Common Programming Interface Communications Reference Manual*¹ describes in detail the rich characteristics and extensive functionality of the CPI-C interface, and References 2–5 describe the BiProcessor VM program offering.

The VM/SP applications may also use CPI-C to communicate with PC-DOS, OS/2, or AIX workstations attached via a token-ring or Ethernet adapter to the PS/2 side of a BiProcessor that is executing OS/2, as described earlier in this paper.

The PC-DOS CPI-C implementation supports a single user on the BiProcessor PS/2. It does not support a local area network (LAN) attached to the BiProcessor system as the OS/2 implementation does.

Future considerations

BiProcessor objectives, in addition to satisfying applications immediately realizable, were to

serve as an enabler for the future. The design was driven by the recognition that systems architectures are going through a generational change, with novel system concepts emerging that will allow computing applications that were previously possible only on large super computers to become commonplace.

Although today's typical data center application with TP-line-connected processing units that are used primarily for management of corporate data and time sharing of computing resources will not disappear, new applications will exploit recent advances in parallel systems technology. Clusters of processing units sharing in the task-execution load will also be joined by various specialized processing units the architectures of which are different and dedicated to specific tasks, such as data sorting and filtering, signal processing, and inferencing. These coprocessors may simply be executing reduced instruction sets for scientific visualization, or they may be collections of transputers cooperating on the processing of the different layers in a communications protocol. Other examples of future coprocessor categories include learning machines, that is, neural processors that emulate the behavior of the human brain, and massively parallel processors.

With the transition from centralized to fully distributed architectures, where single complex applications are distributed onto parallel clusters of heterogeneous CPUs, the communications architectures required to support this distribution, by

necessity, also are going through a generational change. With the high data rates realized by fiber optics and new system concepts exploiting these high data rates, the technology trend is to not move the data as communication protocols are processed, but rather to deposit the data directly in the application buffer. Therefore, implementations based on shared memory approaches with buffer-pool technology as an integral component in the path between the application and I/O processors are likely to be the prevalent choice for future low-overhead protocols. Some of these emerging I/O processors are in themselves highly parallelized.

Just as the applications and communications are changing so is the form of the information that is to be transferred between processing units. Information has attributes that require different treatment. This, in turn, translates into parallel logical data paths implemented with different protocols based on specific attributes between the execution units.

Whereas, there will always be requirements for the type of corporate data we are familiar with today, with its guaranteed integrity, for some information categories bit errors can be ignored. For image and voice information, for example, it may not make sense to do extensive error checking and correction on the data that would increase the associated processing costs, if the human senses cannot perceive the error. Neither does it make sense to do corrections of bit errors on information collected by sensors such as radar that have relative low resolution. Corporate-type data, on the other hand, must not be corrupted, because bit errors can lead to economic loss and chaos. For these information forms, the relative costs associated with the protocol processing overhead to ensure the accuracy of the data are necessary.

For other information forms, latency (or relative latency) is the primary protocol driver. For some process control applications, the length of the control loop forces transfer latencies between applications down to less than a millisecond, as compared to the order to several orders of magnitude higher latencies common today. For concurrent voice and video transmission on different channels, synchronization between the two, the relative latency, is essential and drives the protocol implementation.

A large category of new applications are cyclical in nature, for instance, computer vision, weather and storm tracking, air traffic control, and other monitor and control systems. Their information transfers are characterized by highly periodic traffic. For these transfers, no error checking or retries are performed because redundancy is intrinsic to the processes themselves. Given a 60 Hz system, new data will be available in 16 milliseconds, or in less time than it would take to resend the old erroneous information. For this type information, jitter (that is, variance in the periodicity), may be a much more stringent requirement than any other attribute. Periodic information as described for these type applications would be deposited directly at the application level in swinging buffer pairs, a two-deep FIFO (first-in-first-out). The application works out of one buffer while the other is being refreshed by the communications processor. Other applications, such as some filter applications, require the most recent five or six data points. Their pipe would be six points deep, with the oldest one updated by the communications processor every cycle.

Another example of an information form that dictates special communications protocol treatment is that of remote interrupts used for synchronization purposes and to distribute control, especially in event-driven architectures, where an external event may initiate the execution of a dormant task.

Although the subject of this paper is not that of the communication architectures required to support future distributed systems, a major objective in the development of the BiProcessor was to provide a platform that could be used for prototyping different clustering concepts and the intelligence required outboard before committing these protocols to silicon.

Concluding remarks

The BiProcessor provides a platform that supports the rapid customization of computer systems by merging the two processing environments, thereby allowing both the System/370 and the PS/2 to exploit the strength of the other, as evidenced by the ease with which new applications were developed. Examples of innovative system solutions that take advantage of the complementing resources, both hardware and soft-

ware, include concurrent applications coprocessing, protocol off-load, and System/370 use of Micro Channel adapters.

Communications architecture is, at present, a very rapidly evolving technology. Silicon implementations of parallel executions for low-overhead protocols are emerging with some functions done in hardware that used to be application software. This paper has described how BiProcessor design objectives were to provide a vehicle to allow the infusion of this technology into the System/370 processor family. The BiProcessor pipes are the first steps toward a long-term goal to come up with a product to support future distributed object-oriented, event-driven, real-time applications.

At the same time we are meeting our objectives to provide a toolkit for rapid customization, the BiProcessor development allows us to gain valuable experience in the coupling of heterogeneous architectures. The development of future processor-to-processor interfaces for the coupling of host processor to foreign architectures can take advantage of the lessons learned from the BiProcessor pipe.

Acknowledgments

The author wishes to acknowledge Ronald W. Hoffman and Robert C. Will for their role in the development of the prototype and G. W. (Bill) Wilhelm, Jr., who was instrumental in making that prototype into a product.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Syzygy Communications, Inc., Microsoft Corp., Novell, Inc., Phaser Systems, Inc., Computer Information Enterprises, Inc., Apple Computer, Inc., MacIntosh Laboratories, Inc., or Digital Equipment Corp.

Cited references

1. *Systems Application Architecture—Common Programming Interface Communications Reference*, SC26-4399-0, IBM Corporation; available through IBM branch offices.
2. *Installation Checklist for VM Personal Workstation Communication Facility*, SX24-5268, IBM Corporation; available through IBM branch offices.
3. *Managing VM Personal Workstation Communication Facility*, SC24-5585, IBM Corporation; available through IBM branch offices.

4. *Programming for VM Personal Workstation Communication Facility*, SC24-5586, IBM Corporation; available through IBM branch offices.
5. *VM Personal Workstation Communication Facility Host Guide and Reference*, SC24-5593, IBM Corporation; available through IBM branch offices.
6. *9371 Model 14 Data Exchange Adapter Programmer's Reference Guide*, SA24-4136, IBM Corporation; available through IBM branch offices.

Accepted for publication January 9, 1992.

Christina Berggren IBM Enterprise Systems Division, Route 17C and Glendale Drive, Endicott, New York 13760. Ms. Berggren joined IBM in 1973. She earned a B.S. in chemical engineering and an M.S. in polymer science from the Royal Institute of Technology, Stockholm, Sweden. Ms. Berggren is currently a senior engineer responsible for coprocessors and heterogeneous coupling for midrange Enterprise Systems. Her interests lie in clustering and communications architectures for distributed real-time computing, and she has given numerous papers and lectures on the subject. Ms. Berggren has been contributing for many years to the international technical organization that develops clustering standards for military and aerospace applications. She is chairperson of the Systems, Applications, and Requirements Subcommittee of the SAE Interconnect Networks Committee and also chairperson of the Real-Time Model Task Group in the same organization.

Reprint Order No. G321-5485.