The Parallel Processing Compute Server

by E. M. Ammann
R. R. Berbec
G. Bozman
M. Faix
G. A. Goldrian
J. A. Pershing, Jr.
J. Ruvolo-Chong
F. Scholz

The Parallel Processing Compute Server (PPCS) is a distributed-memory multiprocessing system consisting of System/370™ microprocessors (33 at present) interconnected through a matrix switch. This paper describes the hardware configuration, the extensions to the System/370 instruction set that are provided to support the distributed memory and interprocessor signaling, the modifications to the VM/SP operating system that allow it to run effectively on many closely coupled processors (most of which have no disks), and the application-support layer, which permits FORTRAN programs to take advantage of the highly parallel environment. Development of the PPCS is a joint effort of the IBM Böblingen Development Laboratory and the IBM Thomas J. Watson Research Center. A prototype PPCS has been installed at CERN.

Introduction

The Parallel Processing Compute Server (PPCS) is a *distributed-memory* multiprocessor that uses message passing for interprocessor communication. It comprises an IBM Enterprise System/9373 (ES/9373) Model 30[1] [1] integrated host (I-host) system and 32 (at present) System/370 satellite processors [2], all interconnected via a matrix switch designed to accommodate 62 ports. The I-host is the only processor with I/O capability; otherwise, the satellite processors are functionally identical to the I-host processor. Running under the control of Parallel VM, a modified form of the IBM Virtual Machine/System Product (VM/SP) operating system [3], the PPCS is suitable for a variety of numerically intensive computing programs.

Parallel VM features "diskless" operation on the satellite processors (but is designed to exploit I/O-capable satellite processors, if they exist), fast interprocessor communications, local and remote inter-virtual-machine message-passing, and, to a large degree, a single-system image as seen by the application programmer.

Distributed VS FORTRAN, a prototype programming environment for PPCS, has been developed. VS FORTRAN [4] application programs must be reorganized by the user to exploit parallel processing by originating, scheduling, and synchronizing subtasks. A subroutine library for task management and data transfer has been developed in the spirit of IBM Parallel FORTRAN [5]. The implementation of this Distributed VS FORTRAN library is based on a CMS [3] extension, called CS/X, that takes advantage of the Parallel VM functions.

A PPCS with 32 satellite processors, Parallel VM, and CS/X has been installed at the European Organization for Nuclear Research (CERN) and is running applications used by the high-energy physics (HEP) community. At CERN,

[1] The ES/9373 Model 30 is an entry-level processor of the Enterprise System/9370™ (ES/9370™), which is a family of IBM System/370™ processors that support many users and applications in the commercial, engineering, scientific, and industrial environments.

on-line data acquisition and preprocessing of experimental data is a major computing task. High-energy physics experiments can generate streams of data at a rate of several gigabytes per second. Once filtered and recorded, data are subject to further analysis off-line. Both on-line and off-line processing are numerically intensive computing jobs. The PPCS is primarily intended for the off-line processing, but it can be used for on-line work as well.

The PPCS provides a raw performance potential of 33 times the performance of a single processor. The actual performance depends on the application programs executed. In general, the greater the ratio of CPU time to I/O time and data-transfer time, the higher the throughput. Several CERN applications [6] that make extensive use of bulk data transfer have shown speedup factors of about 30, using 32 satellite processors.

With the established PPCS infrastructure (matrix switch, Parallel VM, and Distributed VS FORTRAN), the groundwork was put in place for the development of a more powerful PPCS with new Enterprise Systems Architecture/390™ (ESA/390™) microprocessors, as used in the IBM Enterprise System/9000™ (ES/9000™) Model 150[2] computer [7].

The following sections describe the background of PPCS-like systems, the PPCS hardware, its architecture, the Parallel VM operating system, and the Distributed VS FORTRAN environment.

Background

Since Paul Kunz of the Stanford Linear Accelerator Center (SLAC) constructed a "farm" of processors, each of which emulated, in part, the instruction set of the IBM System/370 computer [8], CERN, SLAC and the Massachusetts Institute of Technology have developed a considerable number of these emulators for use as computer farms [9–13], which use small processors for running applications written in FORTRAN. In these farms, only the host processor has I/O capability. Computer farms generally have a straightforward master-slave control structure, with each processor working on an independent unit of work and no requirement for any-to-any processor communication. A typical use of these farms in the HEP community is the analysis of "event data" from an experiment. Each processor of the farm is loaded with the same program. A scheduling program on the host then "feeds" the application programs with event data and receives the results upon completion. Since all of the events are disjoint, any event can be sent to any processor in the farm. Communication with the farm processors is typically via a VME bus [14] or something similar. A good review of these machines can be found in [6]. Similar farms based on machine architectures other than System/370 have been built at other laboratories [15–17]. These farms generally do not have operating systems running on the satellite nodes, but rather very primitive kernels sufficient to execute FORTRAN programs. For example, this kernel is several hundred lines of FORTRAN on the Fermi National Accelerator Laboratory Advanced Computer Program (ACP) [18]. Although this is a simple approach that performs well, it can be problematic for the development and debugging of applications.

PPCS development was initiated primarily because of HEP computing requirements. The PPCS is a parallel processing system that extends the concept of a processor farm. The matrix switch provides a high-speed interconnection network that allows any-to-any processor communication. This permits the PPCS to be used for more general parallel applications, such as lattice gauge calculations [16]. A consistent approach to System/370 architecture and system software (Parallel VM and Distributed VS FORTRAN) is used throughout the system and provides a large set of functions that operate on the I-host as well as on the satellite processors. The presence of a full-function operating system provides editing, compiling, testing, and debugging facilities throughout the farm.

Hardware system

A PPCS is primarily composed of "off-the-shelf" components from the IBM ES/9373 Model 30 system. It is housed in three standard IBM racks. The logical system structure is depicted in **Figure 1**. The matrix switch interconnects the I-host system, the 32 satellite processors, and, optionally, an external controller.

- *I-host system*

The I-host processor has 16 megabytes of main storage and contains two internal I/O buses. One bus links the I-host to the matrix switch; the other bus connects four IBM 9332 disk units, an IBM 9347 tape unit, a workstation controller, a communication controller, and a block multiplexer (BMPX) channel to the I-host processor. One can use the BMPX channel to connect the I-host to a large System/370 mainframe or to attach tape drives. The disk units are used to store the Parallel VM operating system, application programs, and data. The tape units serve as external input/output devices. An IBM PS/2® Model 30 attached to the I-host is used as a system console to control the entire PPCS, and as a support processor for IML (initial microprogram load), IPL (initial program load), error recovery, error logging, and console operations.

- *Satellite processors*

The satellite processors employ the System/370 mainframe architecture, including floating-point, square root, and

[2] The ES/9000 Model 150 computer is a rack-mounted processor of the ES/9000 family of processors. It is an advanced, general-purpose, intermediate computer that implements the ESA/390 and the System/370 architecture.

high-accuracy arithmetic [19]. To optimize performance, many frequently executed instructions are executed by the hardware without microcode.

As shown in **Figure 2**, each satellite processor consists of a processor card, an 8-megabyte memory card, and a switch-adapter card. The processor chips [2] have an 80-ns cycle time.

The switch-adapter card contains an electrically programmable read-only memory (EPROM) with bootstrap microcode to prepare for IML, an interface for displaying the status of the satellite processor, a bus switch adapter (BSA), which connects one internal I/O bus to the switch cable, and an interface to connect an optional service processor. The service processor is a tool that can be temporarily connected to a satellite processor for maintenance purposes. Its configuration is similar to that of the support processor.

The following types of data are transported across the switch between connected processors, under the control of the BSA:

- Messages (from mailbox to destination message queue).
- Data blocks (from memory to memory).
- Control commands.

The BSA resolves message collisions with a minimum impact on performance.

The switch cable connects the BSA with its switch port. It is a transmission-line cable (2-meter flat cable) with 15 signal and five ground lines. The cable supports a bidirectional, 1-byte-wide, synchronous data transfer.

- *Switch*

The switch is the interconnection facility for the PCCS. It uses the switch element from the ESCON Director [20], used in the IBM System/390™ computer. The switch consists of 62 ports, the switch element itself, and a switch controller. Although the switch is designed for 62 ports, only 33 are used in the current PPCS system: to connect the I-host system and 32 satellite processors. Optionally an additional port can be used for an external controller. The switch is nonblocking and allows up to 16 two-party communication paths to exist simultaneously. In addition, it has a broadcasting mechanism that allows the I-host to IML all satellite processors simultaneously. The switch is controlled through a set of new *System/370 interprocessor communication instructions*, which are described in the next section.

PPCS machine architecture

The PPCS machine architecture is an extension of System/370 architecture [21] to support a switch-connected distributed-memory parallel processing system. To enable the processors to work together efficiently on a single job,

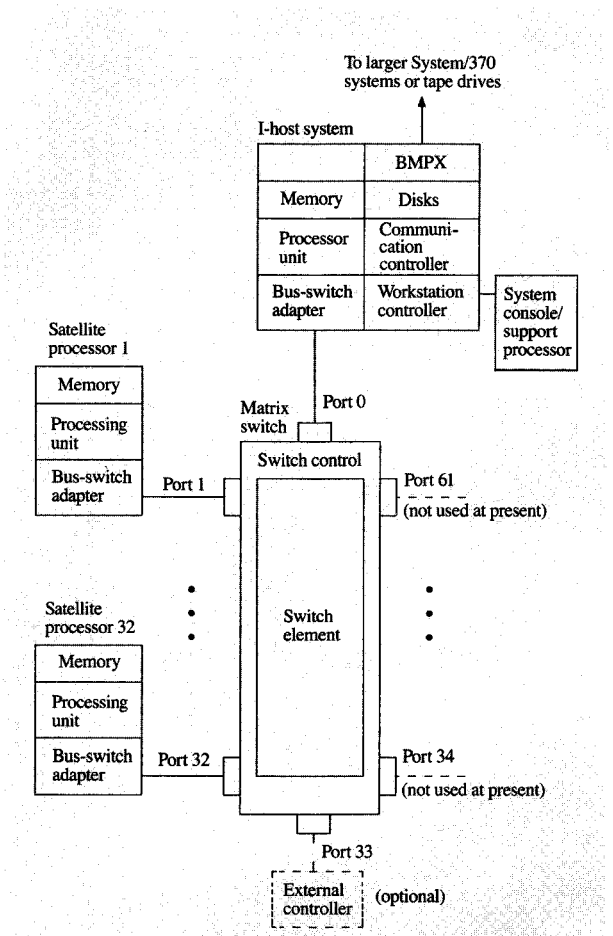

Figure 1

PPCS hardware structure.

the architecture includes synchronous instructions that allow a processor to access the memory of other processors and to signal other processors. These interprocessor memory accesses are key-protected in the same way as local memory access. In effect, this provides a message-passing architecture complemented with facilities that are usually included only in shared-memory multiprocessor systems. In addition, instructions are included that allow direct function invocation on a remote processor without the overhead of interrupt-handling routines.

The new instructions are listed below:

- *Synchronous data transfer*

IPPUT	Write data to destination processor memory.
IPGET	Read data from destination processor memory.

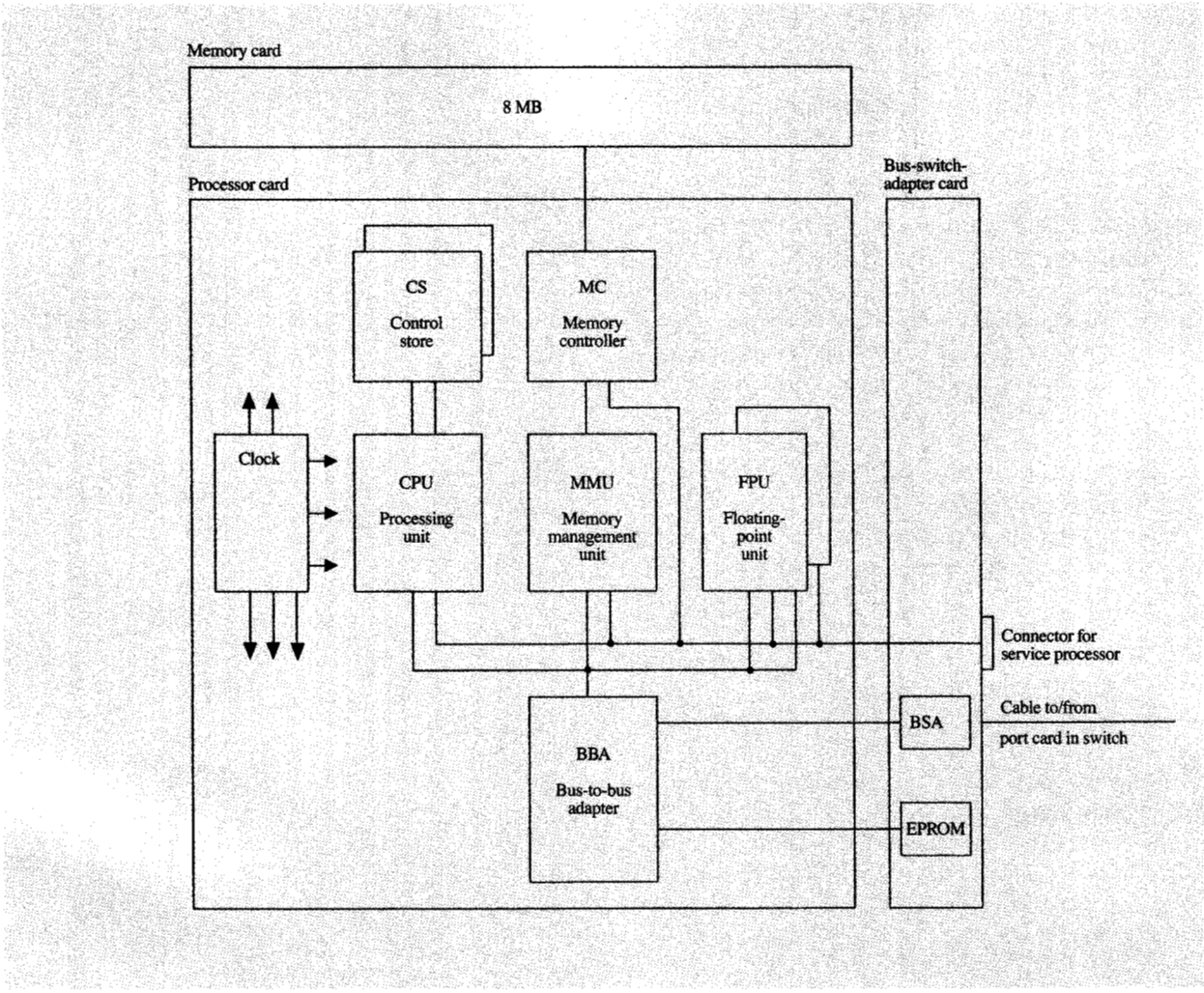

Figure 2

Satellite processor — interconnections and internal elements.

- *Signaling*

  SIGP Signal destination processor.

- *Remote function invocation*

  IPCALL Invoke remote function at destination processor; save original environment in a save area.

  IPSS Retrieve environment from local save area (for inspection).

  IPRET Resume environment kept in local save area.

The IPPUT and IPGET instructions are used for *data transfer* between two processors.

The *signaling* instruction, SIGP, extends the standard System/370 SIGP instruction to a distributed-memory parallel processor environment. The SIGP instruction accepts as one of its operands an order code, which determines the type of signal to be presented at the destination processor. For PPCS, new orders are provided with SIGP; for instance, the *external call with parameter* order passes a 32-bit parameter along with the "external call" signal.

The IPCALL instruction permits *function invocation* on a remote processor. The issuing processor supplies an entry-point address (on the remote processor) and a set of parameters, and the specified routine is invoked in much the same way as an interrupt-handling routine. This routine may retrieve the interrupted state of its processor into main memory with the IPSS instruction, and may "return to" this interrupted state with IPRET.

Standard System/370 machine instructions are implemented partly in hardware and partly in microcode. The new PPCS-unique instructions are all implemented in microcode and additional hardware in the switch subsystem.

The microcode load (IML) for the PPCS is initiated by the I-host support processor and is performed for all processors, in parallel. Thus, the IML time for the entire PPCS is only slightly longer than for a single processor.

Figure 3 illustrates the interfaces between the System/370 layer, the PPCS microcode, and the switch subsystem. PPCS microcode is invoked when one of the new instructions is issued or when a message arrives from another processor over the switch. Requests from the PPCS microcode in a processor are given to the switch hardware (to send a message or to transfer data) by placing them in a special "mailbox" location in the internal memory of the processor and then raising a signal line to notify the switch hardware of the request. The switch hardware delivers a message by placing it on the inbound message queue in internal memory and by raising a signal line to notify the PPCS microcode in that processor. Data transfer is performed by the switch hardware by moving data directly between the main memories of the appropriate processors.

For example, invoking the IPPUT instruction leads to a data-transfer request (of type *write* in this case) to the switch hardware. The instruction is completed after the hardware has performed the data transfer and dropped the signal line. The PPCS microcode of the destination processor is not involved.

Consider another example: invoking a SIGP instruction. This causes the PPCS microcode to send a request message to the destination processor. The PPCS microcode of the destination processor performs the order (e.g., by presenting an external interrupt to the System/370 layer for the order *external call with parameter*) and sends back a response message containing status information. The microcode of the issuing processor synchronously waits for the response message and then completes the SIGP instruction with the appropriate condition code.

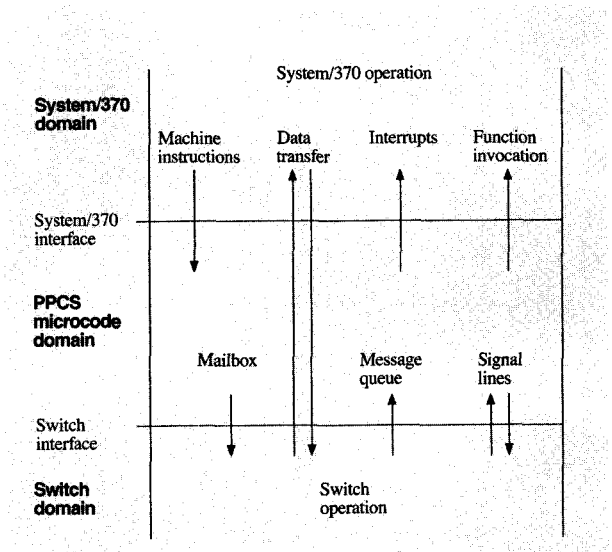

Figure 3

PPCS machine interfaces.

Parallel VM

The control program (CP) of VM/SP manages the resources of a single computer so that multiple systems appear to exist. Each "virtual" computing system, or virtual machine, is the functional equivalent of a System/370 computer. CP also manages the communications among virtual machines, and between a virtual machine and the real system. It seemed natural to extend virtual-machine-to-virtual-machine communication across processors. By using VM/SP as the base, we developed a message-passing parallel system, Parallel VM.

Processor-to-processor communication is the most critical component of Parallel VM. Communication is required at all levels of the Parallel VM system, ranging from "stand-alone" communication to support IPL and dump through higher levels to support CP operations (e.g., paging and spooling) and the end users running under CMS[3] (e.g., minidisks[4] and message passing).

There is a single version of the operating system for the PPCS, so Parallel VM must dynamically determine its environment (e.g., is an I/O operation for a local or remote device?). I/O requests from satellites to remote devices (i.e., those attached to the I-host) are redirected to a file server on the I-host.

Figure 4 shows the Parallel VM components and their interrelationship, for the I-host and two of the 32 satellites. The components are described in the following subsections.

- *Communication*

Native CP Communication component

Parallel VM includes a basic processor-to-processor communication facility as a native part of CP. The communication component provides CP (and, by extension, the various virtual machines) with an arbitrary

[3] The Conversational Monitor System (CMS) is a single-user operating system that executes in a virtual machine. It runs under CP to provide a general-purpose, conversational, time-sharing facility.

[4] A minidisk is a contiguous subset of a real disk. In VM, a minidisk is used by the CMS file system to contain a set of files and their associated directory.

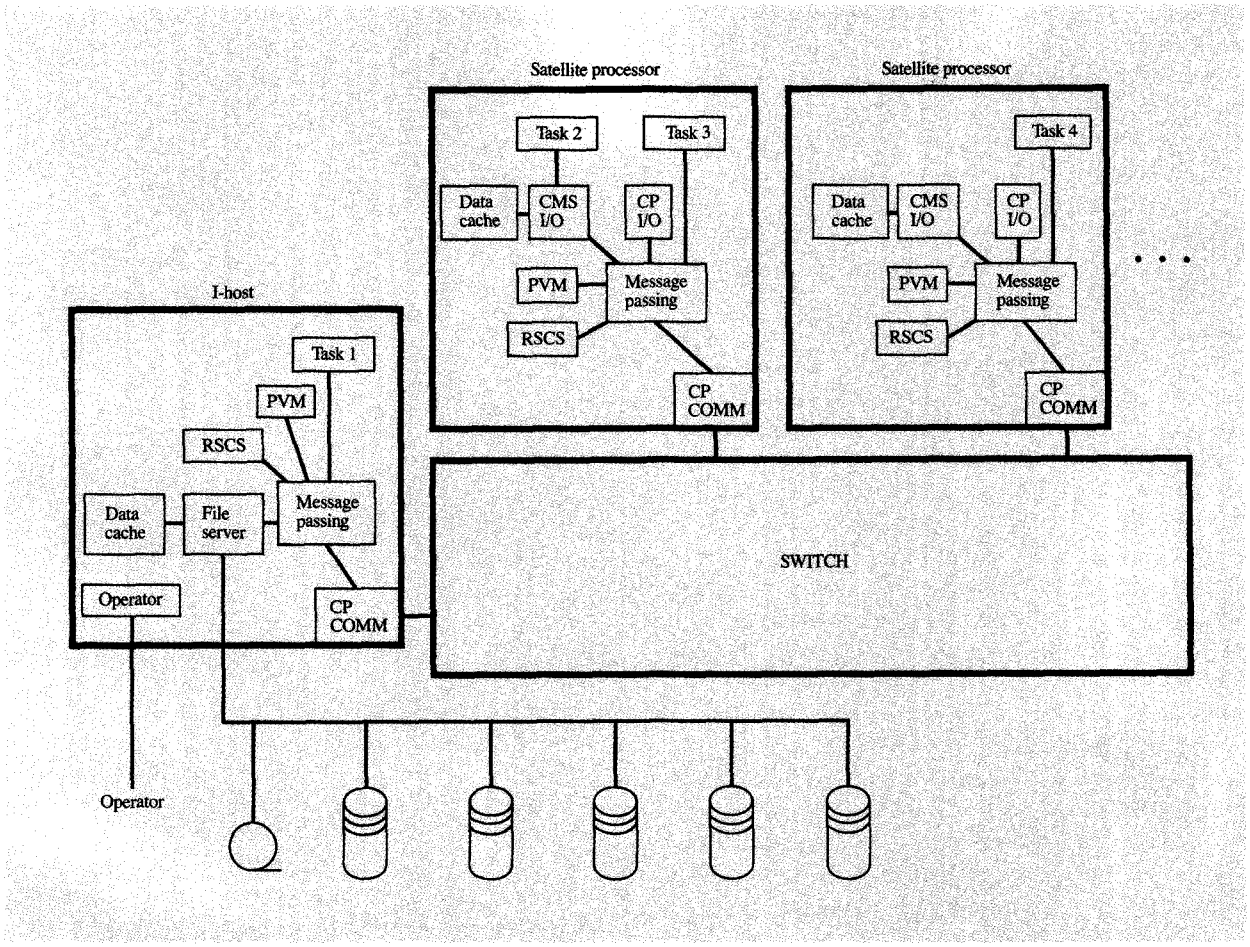

Figure 4

Parallel VM structure.

number of logical *sessions* that provide connections to CP components in the other processors of the PPCS. Each session provides reliable and orderly movement of data between "peer" CP components on a pair of processors. If an error prevents this orderly movement of data, the two components are notified of the failure.

We could have implemented an *ad hoc* communication protocol that would have worked adequately. However, even a specialized, *ad hoc* protocol must address the basic problems of communication: flow control (pacing), error control, data transparency, naming, addressing, and multiplexing/routing. Once these issues have been tackled, the implementer finds that he has reinvented techniques that have already been developed for the various "standard" communication protocols. We therefore decided to produce an implementation of an existing communication protocol: the IBM Advanced Peer-to-Peer Networking (APPN, also known as PU-2.1) [22], which provides Logical Unit 6.2 (LU-6.2) sessions [23] to the various components of CP.

For expediency, only a subset of the full APPN and LU-6.2 protocols has been implemented. The native CP Communication component does *not* provide the "Presentation Services" function of LU-6.2, because our immediate client is system code (as opposed to application code), and because the Presentation Services function already exists in the Advanced Program-to-Program Communication component of VM/SP. Routing in the PPCS machine is trivial because each processor has direct access to every other processor; therefore, a trivial subset of APPN routing has been implemented. We have merged the *address* of each processor into the *names* used on that processor (e.g., `SYS0005` on processor 5), obviating the need for a directory function.

Since so much of the system depends on communication, the implementation must be fast. All communication processing is event-driven to minimize dispatching and queuing overhead. Excess copying of data within the memory of each processor has been eliminated by a special I/O-buffer-tracking discipline: Data being sent and received are copied at most once. Pointers to processing routines are stored in control blocks, and these pointers are changed to point to different routines to reflect major state changes in the communication system; this eliminates the need for extraneous conditional tests and branches in the mainline code. Since the various PPCS-unique instructions are relatively expensive operations, considerable work has gone into minimizing the use of these instructions. The net communication cost to *send* a request and *receive* a response is approximately 1200 instructions, plus two PPCS-unique instructions.

Virtual Machine Communications Facility (VMCF)

VMCF [3], a component of CP, allows a virtual machine to send messages and data to another virtual machine on the same processor. We modified VMCF so that a virtual machine on one processor can also communicate with a peer on a different processor in the PPCS cluster.

SEND/RECV is an example of a VMCF protocol. It consists of the SEND/RECV, RECEIVE, and REPLY functions. SEND/RECV directs data to another virtual machine and requests a reply. RECEIVE (on the target virtual machine) accepts data sent. REPLY directs data back to the originator of the SEND/RECV function. **Figure 5** depicts the local SEND/RECV protocol, and **Figure 6** depicts the remote one. The originating virtual machine is called the *source* virtual machine, and the destination virtual machine is the *sink* virtual machine.

When converting local VMCF into remote VMCF, we tried to maintain the VMCF interface and structure. The VMCF functions to be executed are the same, but in the remote case some are executed on the source processor while others are executed on the sink processor. For example, for a SEND/RECV, validity checking of the SEND and REPLY buffer addresses and "locking" of the virtual pages into real storage frames must occur on the source processor. Ensuring that the sink virtual machine is logged on and authorized for VMCF occurs on the sink processor.

Remote VMCF uses CP Communications to exchange control information. For a remote data transfer, IPGET and IPPUT are used to move data directly between virtual machines instead of the intraprocessor MOVE LONG instruction.

Virtual channel-to-channel (VCTC)

To interconnect the satellites to the I-host for spool file interchange and "remote" log-on, virtual channel-to-

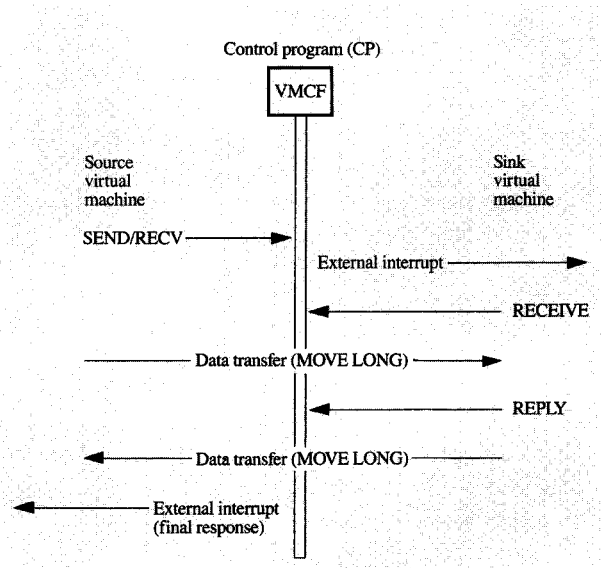

Figure 5

Protocol for a local SEND/RECV request.

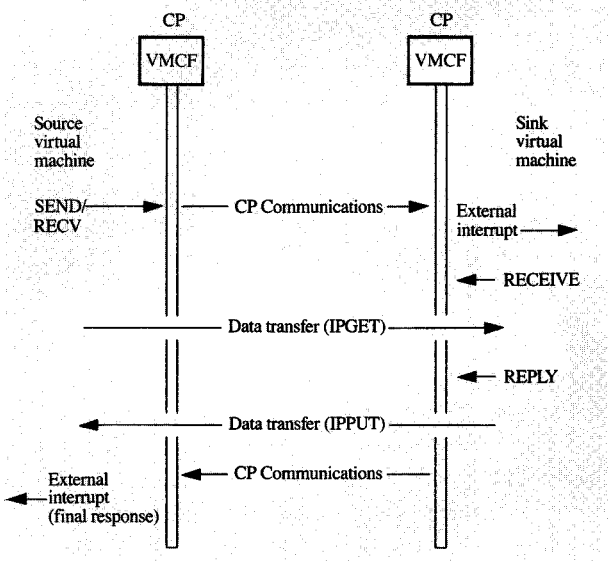

Figure 6

Protocol for a remote SEND/RECV request.

channel links are established over sessions provided by CP Communications. These links are used by the Remote Spooling Communications Subsystem (RSCS) and the VM/Pass-Through Facility (PVM) to provide an intra-PPCS "network" and a gateway to other systems [3]. For example, a user at CERN can log on to any satellite; edit, compile, test, and run a FORTRAN program; and then have any output forwarded to another system.

- *Diskless operation*

Server

I/O operations (CP and CMS) from satellites for devices attached to the I-host (and consequently remote to a satellite) are handled by a server running in a virtual machine on the I-host that has access to PPCS minidisks. This server is a modified and extended version of a prototype developed at the IBM Thomas J. Watson Research Center[5] [24].

Communication between the server and its clients on the satellites is performed through a special interface to VMCF, using the SEND/RECV protocol. The server recognizes two types of requests: input/output requests (read or write a block on a minidisk) and command requests (e.g., LINK or DETACH a minidisk).

The server performs I/O to CMS minidisks using the IUCV *BLOCKIO [3] system service; multiple blocks can be read or written with one request. The server is designed to support multiple concurrent I/O operations.

Minidisk cache

File caching is done on the I-host and satellites. The satellite file cache stores data previously received from the I-host server. This reduces the load on the I-host server, alleviating a potential bottleneck. The file cache is a modification of the VM/XA® minidisk cache [25]. It was modified to use processor memory page frames instead of expanded storage frames. An arbiter dynamically adjusts the size of the cache in response to contending demands for processor memory.

CMS

CMS was modified to exploit remote minidisk access[5] [24] and local caching of remote minidisk data.

A prototype of remote minidisk access was developed at the IBM Thomas J. Watson Research Center in 1984 by Xavier de Lamberterie[5]. The prototype intercepted remote minidisk I/O requests at their entry to the lowest layer of the CMS file system (DMSDIO), which services requests to read sets of blocks on a minidisk. The intercepted requests used a special communication facility which, for performance reasons, was deemed inappropriate for the PPCS because the communication path went through several virtual machines. Therefore, while retaining the intercept point in DMSDIO, CMS in Parallel VM uses a special interface with VMCF for file system I/O.

Modifications were made to the CMS initialization routines to support a completely diskless environment.

To preserve the behavior of CMS minidisk sharing among readers and writer, a diagnose[6] interface was provided to the local minidisk cache. The diagnose is used during a CMS ACCESS[7] to verify the consistency of the minidisk in the local cache. If it is not consistent, the cache is purged of all data from this minidisk. The cache is not purged frequently, since in the PPCS environment the files with the greatest probability of being rereferenced are either on read-only libraries or private, unshared minidisks.

CP I/O

The I/O that CP does for itself or when providing a virtual machine image (i.e., all I/O not directly initiated by a virtual machine) is normally directed to system disk areas. In a sense, CP can be considered to have a primitive file system. In Parallel VM, CMS minidisks managed by the server on the I-host replace these CP files. Since CP does I/O in 4-kilobyte (4KB) blocks, it is easy to map these files into 4KB-blocked CMS minidisks.

Each satellite has access to two system "volumes." The first one, shared by all satellites as well as the I-host, is the system residence volume; it contains all of the read-only data common to all processors. For example, it contains the operating systems (e.g., CP and CMS) and the user directory.[8] The second volume contains all of the read-write areas required by each satellite (e.g., warm-start, checkpoint, and error-recording areas and page and spool space). Therefore, the I-host server manages one read-only minidisk for all shared read-only system data and as many read-write minidisks as there are satellites without local read-write system volumes.

The page manager performs almost all CP I/O operations. When the page manager is called, its parameter list specifies the system volume and the block number on that volume. If the volume is remote, the page manager builds a VMCF parameter list specifying the equivalent block number within the I-host server CMS minidisk. VMCF then calls CP Communications to redirect the I/O request to the server on the I-host. Once the request is complete, the page manager is resumed.

[5] Xavier de Lamberterie, *Remote Disk Access Design Notes*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1984.

[6] A diagnose is a special instruction-level interface between a virtual machine and CP. Each diagnose has a unique identifying code and typically points to a set of parameters.

[7] The ACCESS command identifies the minidisk to CMS. Also, it reads the file directory and, if the user has write capability, the allocation bit map into the user's virtual memory.

[8] The *user directory* in VM is a list of all users of the system along with their resources—in particular, minidisks.

Satellite initialization (IPL) and satellite failure recovery (dump) do not use the page manager for their I/O operations. These components were also modified to run in a diskless environment.

The I-host is IPLed in the traditional System/370 way. The IPL is started by specifying *load* on the processor console, causing a channel program to be read from the selected input device. The channel program specifies that the CP nucleus initialization program (NIP) is to be read into memory. Control then passes to the NIP, which proceeds to read the rest of the CP modules into memory.

The satellites cannot be IPLed in the traditional manner, because they have neither I/O devices nor a processor console. Instead, as part of the I-host initialization, IPPUT instructions inject the NIP into the memory of all on-line satellite processors. Once the NIP is loaded, a SIGP instruction starts each satellite.

The satellite must now read in portions of the CP operating system, but it lacks I/O capability. The satellite requires that CP Communications forward the I/O requests to the server on the I-host, but it is too early in the satellite initialization to support CP Communications (e.g., the satellite does not yet have storage management services and an external interrupt handler). To solve this problem, a "thin-layer" communication routine, a very small subset of CP Communications, is included in the satellite NIP. The thin layer does its own storage management; it statically allocates storage from an area that is not yet needed by the operating system. It has its own external interrupt handler for notification by the I-host when data have arrived or when a buffer has become available.

Requests from the first IPLing satellite cause the server to read the remainder of the CP modules from the system residence volume. Since IUCV *BLOCKIO is a cachable interface, these pages are also inserted into the I-host minidisk cache, from which the needs of subsequently IPLing satellites can be satisfied without further disk I/O. When initialization is complete, CP Communications takes over the function of the thin layer.

- *Systems management*

The systems management of 33 processors could have been unwieldy, but packaging the PPCS with only one processor console has simplified its management. The I-host operator serves as the PPCS cluster operator through the use of the Programmable Remote Operator Program (PROP) [3] and additional commands to control satellites and display their memory. With PROP, important satellite messages are redirected to the I-host operator. The I-host operator can broadcast commands (e.g., IPL a satellite) or direct them to a specific satellite. Displaying satellite memory from the I-host is particularly useful when debugging a problem on a satellite.

Having one disk copy of the system residence volume also simplifies the management of the PPCS. If changes are made to the operating system, only one system generation is required instead of 33. Since there is a common user directory and all minidisks are accessed through a server on the I-host, file data are accessible by any processor (subject to normal password protection).

- *Future plans*

As a result of using Parallel VM at several IBM sites and CERN, we intend to extend the existing prototype. These extensions will further enhance the single-system image and broaden the I/O transparency.

The separate spools on the PPCS cluster result in an inefficient use of space and reduce the illusion of a single system. Spool files for a single user may be located on several processors within the cluster. Although it is relatively straightforward to collect them in one place (e.g., the I-host), this can be confusing to users. Also, global spool file maintenance is complicated, because 33 different spools are involved. As the number of processors in the PPCS grows, the maintenance becomes more complicated; therefore, any future work would solve this by creating a true single spool shared by all systems in the PPCS.

Although the I/O-server approach combined with cacheing has worked well, it would be better if I/O were completely transparent on the satellites. Then any device, such as a tape drive, could be attached to any virtual machine in the PPCS cluster. This can be accomplished by intercepting remote I/O requests in CP.

The simple canonical naming used on the current prototype will also be relaxed by introducing a more sophisticated routing function. This constraint has not proved to be very annoying, but it does diminish the image of a single system.

Distributed VS FORTRAN environment

For the distributed-memory, parallel processing architecture of PPCS, a programming environment was needed to serve the requirements of the HEP community at CERN. The prevailing requirement at CERN [6, 17] was for control over individual processors, interprocessor data transfer, VS FORTRAN support, and facilities to program for high performance. The Multitasking Facility of VS FORTRAN and Parallel FORTRAN were evaluated with respect to PPCS requirements.

The Distributed VS FORTRAN prototype includes those capabilities of Parallel FORTRAN that are suitable for coarse-grained parallel programming on the PPCS system in the style of the computer farm [6, 13] model. The programming facilities of Distributed VS FORTRAN are provided by the Compute Server Library (CS/L), a collection of subroutines. There are no modifications either

to the VS FORTRAN language or to any of the compilers and libraries in the VS FORTRAN family. A distinct software component, named Compute Server Executive (CS/X), is loaded into all virtual machines that execute computationally intensive tasks.

• *Tasking model*

The tasking model for PPCS is based on an abstraction of computational resources that is represented by a set of virtual machines. One of these virtual machines acts as the host processor, where the application execution originates. This host virtual machine is typically on the I-host, since the I-host provides a direct connection to the application data. The remaining virtual machines on the satellites act as "slave" processors for computationally intensive work. A configuration file in the host environment defines a set of satellite virtual machines for the execution of an application. That configuration file is supplied by the application programmer as a specification of the processor requirements of a particular application.

Distributed VS FORTRAN distinguishes between a main task running on the I-host and subtasks running on satellite processors. The main task initiates subtasks on satellite virtual machines. Only the main task may schedule VS FORTRAN subroutines for execution in subtasks.

The tasking functions are summarized as follows:

NPROCS	Returns the number of available satellite virtual machines.
ORGTSK	Creates a new task and initializes its virtual machine.
TRMTSK	Deletes a task and frees the associated resources.
SCHDTSK	Orders a specified task to execute a named subroutine.
QYTSK	Returns the status of a task.
BRKTSK	Causes a task to wait to rendezvous with its parent.
RSMTSK	Resumes the execution of a task after a BRKTSK rendezvous.
WTTSK	Causes a parent to wait for the completion of a specified subtask.
WTANY	Causes a parent to wait for the completion of any of its subtasks.
WTALL	Causes a parent to wait for the completion of all of its subtasks.

• *Model of data transfer*

Because PPCS is a distributed-memory system, it was determined that all data movements would be explicitly programmed. Except for the remote subtask invocation, synchronous semantics were chosen for most of the CS/L subroutines.

Data exchange between the host and subtasks may be done only by copying FORTRAN COMMON blocks to and from virtual machines on satellite processors. (Copying data from subtask to subtask has not been implemented in the first CS/X prototype.) Subsets of COMMON blocks may be copied if the programmer provides offsets or addresses and lengths that can be checked against COMMON block boundaries.

The copy functions in CS/L are decoupled from the schedule function that assigns work to a subtask. In this way, CS/L deviates from the Parallel FORTRAN semantics in which copying of data can be expressed as part of the schedule statement. The CS/L copy functions require synchronization between the main task and the subtask to ensure the integrity of COMMON block data. Data copy always occurs under control of the main task, independent of the direction of data flow. If a subtask executed a VS FORTRAN subroutine at the time of an incoming copy request, the addressed data could be in an undefined state. Only in the case of "break" points invoked by BRKTSK does the subtask participate in controlling the "rendezvous."

In that respect, the CS/L copy functions differ from classical schemes of synchronous message passing [26], in which the send request can be issued at any time, at the expense of blocking the sender until the receiver is willing to accept the data.

The copy functions are summarized as follows:

CPYIN	A named COMMON block or subset thereof is copied from the main task to the workspace of a specified subtask.
ACPYIN	A variant of CPYIN that accepts addresses of variables as the starting place of the copy.
UCPYIN	A set of contiguous COMMON blocks is copied to the workspace of a specified subtask.
CPYOUT	The analog of CPYIN for copying from a subtask to the parent.
ACPYOUT	The analog of ACPYIN for copying from a subtask to the parent.
UCPYOUT	The analog of UCPYIN for copying from a subtask to the parent.

The variants UCPYIN and UCPYOUT implement "bulk data transfer" by reducing the overhead associated with data transfer. Since the setup time involved in sending a message is largely independent of the message size, it is more efficient to send one large message than a sequence of smaller ones that sum to the same size. If these COMMON blocks are consecutively allocated in memory, either UCPYIN or UCPYOUT can be used to transfer them in one single CS/L request.

• *Sample of an event-processing application*

The feasibility of the CS/L approach is demonstrated by a sample derived from CERN event-processing applications [6, 17]. Physical interactions of particles are transformed into "event data blocks" by signal-processor hardware during high-energy physics experiments. These event data blocks are either saved for off-line analysis or processed on-line by a computer farm.

Figure 7 sketches a sample program of a host application that executes M units of work with the help of N parallel subtasks. The subroutine CRUNCH is invoked on all satellite processors to perform the analysis of event data. Arrays of M input event data blocks (EDATA) and M corresponding result data blocks (RDATA) are maintained by the host program.

The STATUS variable associated with CS/L calls is a result parameter, reflecting the operational conditions upon completion of the call. An application running in production mode must test that status information.

• *Implementation*

The software component for managing distributed applications, the Compute Server Executive (CS/X), is not linked to the application program but resides as a CMS nucleus extension in the virtual memory of host and satellite virtual machines. The separation of application and run-time support software is equivalent to the relationship of user processes and their invocation of operating system functions.

CS/X implements distributed protocols based on passing messages between host and satellite virtual machines in the client-server [26] style. User-defined subtasks are mapped in a one-to-one manner to satellite virtual machines to avoid resource contention.

CS/X is multithreaded [27, 28] to respond in a timely fashion to requests from remote clients. There are separate threads for VS FORTRAN execution and satellite monitoring running in the same CS/X address space.

Communication between CS/X instances in different virtual machines is accomplished with the interprocessor implementation of VMCF by Parallel VM. CS/X does not interface directly with interprocessor facilities of the PPCS hardware; therefore, it can be run on a collection of virtual machines under a nonparallel VM/SP system (e.g., for debugging purposes).

```
      program EventScheduler
      integer p,m,le,lr,forever,fpsize
      parameter (p=32,m=100,le=3000,lr=1000,forever=0,fpsize=8)
c     These blocks contain the event and result data:
      common /EVENTS/ edata
      common /RESULTS/ rdata
      real*8 edata(m,le), rdata(m,lr)
      integer n,i,tsk,status,tskid(p)
      integer Levent,Lresult
      character*8 cpuid(p)
c     Determine number of available processors:
      call NPROCS (n)
      n = MIN (n,m)
      Levent = le*fpsize
      Lresult = lr*fpsize
c     Start one task per processor:
      do 10 i = 1,n
        call ORGTSK (tskid(i),cpuid(i),status)
10    continue
c     Prime and start n tasks:
      do 20 i = 1,n
        call ACPYIN (tskid(i),'EVENTS',edata(i,1),Levent,status)
        call SCHDTSK (tskid(i),'CRUNCH',status)
20    continue
c     As tasks complete, get results back, supply more events,
c     and schedule again:
      do 30 i = 1,m-n
        call WTANY (tsk,forever,status)
        call ACPYOUT (tsk,'RESULTS',rdata(i,1),Lresult,status)
        call ACPYIN (tsk,'EVENTS',edata(n+i,1),Levent,status)
        call SCHDTSK (tsk,'CRUNCH',status)
30    continue
c     Wait for all n tasks of the last round to complete,
c     retrieve final results, and shut down n tasks:
      do 40 i = 1,n
        call WTANY (tsk,forever,status)
        call ACPYOUT (tsk,'RESULTS',rdata(m-n+i,1),Lresult,status)
        call TRMTSK (tsk,status)
40    continue
      end
```

Figure 7

Sample of an event-processing application.

Performance

The performance of the computer farms used by the HEP community depends largely on the I/O bandwidth of the host (i.e., how fast it can read event data and write output data) and the communication bandwidth from the host to the farm processors (i.e., how rapidly the host can send data to and receive data from the farm processors). A balanced system matches the I/O and communication speed to the computing speed of the processors for the intended set of applications. This allows speedups (comparing the amount of work per unit of time that can be done on a single system to that done on a farm of similar processors) that approach the number of processors in the farm.

In October 1988 CERN and IBM agreed on a joint project that would construct and evaluate several farm prototypes based on System/370 microprocessors. Two of these systems have been delivered to CERN to date: One is the PPCS described in this paper, and the other is a VME-bus-connected system very similar to the traditional

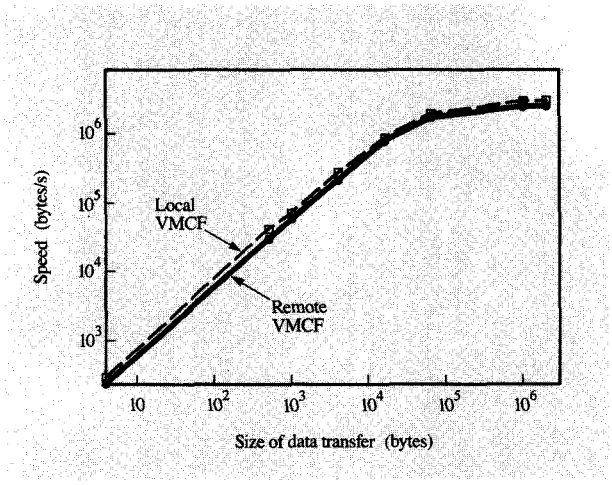

Figure 8

VMCF speed: local vs. remote.

farm. Like the other farms, the latter system does not have an operating system running on the farm processors, but rather a small kernel merely sufficient to support the execution of a FORTRAN program. This approach was taken primarily for simplicity (it is difficult to initialize and run an operating system such as VM/370 in a processor without I/O capability) but also for performance (it was thought that an operating system might introduce overhead that would reduce the efficiency of the farm).

CERN has analyzed several of its typical farm applications (e.g., TRIDENT [29] and a Monte Carlo proton-antiproton event generator, ISAJET) on both of these systems and found them to have essentially identical performance characteristics. Both the PPCS and the VME-bus system obtain speedups that approximate the number of processors in the farm. Details and performance comparisons with traditional HEP farms are given in [6].

One of the major reasons for the performance attained in the PPCS is the low-overhead interprocessor communications. **Figure 8** gives the speed of data transfer as a function of the message size. (The saturation effect is due to the bandwidth limits of memory and the switch.) The figure compares the speeds of local and remote VMCF transfers. Remote VMCF speed varies between 80% of local (with 4-byte messages) to about 90% (with large messages). This is a significant achievement, as the effective intraprocessor memory bandwidth is approximately double the effective switch bandwidth.

Conclusion

The switch-connected PPCS provides, in a way fully compatible with standard System/370 architecture, the numerically intensive computing capabilities of a System/370 system. This coherent system structure allows application program parts to be offloaded from the host system and executed on the satellite processors. This is accomplished through the Parallel VM operating system and a Distributed VS FORTRAN programming environment.

The PPCS is suitable for a variety of numerically intensive computing problems with a high content of partitionable tasks. It provides facilities for coarse-grain parallelism, with the full support of a general-purpose operating system. In addition, task-management and data-transfer support are provided to the application programmer.

The PPCS is running at a few IBM sites and at CERN. Various applications have been ported to the PPCS, including TRIDENT [29] and magnetohydrodynamics [30]. Several of the CERN applications that use bulk data transfer extensively have shown speedup factors of about 30 with 32 satellite processors.

Prototype development work continues at the IBM Böblingen Development Laboratory and the IBM Thomas J. Watson Research Center for a PPCS based on ESA/390 CMOS microprocessors, which delivers enhanced performance and function.

Acknowledgments

We gratefully acknowledge the contributions to this project by our colleagues in IBM, especially W. G. Spruth's advanced technology team and H. Diel for their work on the basic concept of the PPCS. Special thanks goes to the entire PPCS development team.

References

1. *IBM ES/9370: Introducing the System*, Order No. GA24-4030, 1989; available through IBM branch offices.
2. W. G. Spruth, *The Design of a Microprocessor*, Springer-Verlag, New York, 1989.
3. *Virtual Machine/System Product Introduction*, Order No. GC19-6200-4, 1986; available through IBM branch offices.
4. *VS FORTRAN Version 2 Language and Library Reference*, Order No. SC26-4221-2, 1987; available through IBM branch offices.
5. *Parallel FORTRAN Language and Library Reference*, Order No. SC23-0431-0, 1988; available through IBM branch offices.
6. D. Lord, A. Fucci, P. Sphicas, P. Favre, J. P. Ikonen, M. Koratzinos, H. Masuch, A. Paton, C. Pirotte, and S. Tether, "CERN, Emulators and Parallel Processing Compute Servers," *Proceedings of SHARE Europe Anniversary Meeting*, Paris, Oct. 1–5, 1990, SHARE Europe HQ, Geneva, 1990, Vol. II, pp. 929–941.
7. *IBM ES/9000: Introducing the System*, Order No. GA24-4186-0, 1990; available through IBM branch offices.

8. P. F. Kunz, "The LASS Hardware Processor," *Nuclear Instrum. & Meth.* **135,** 435–440 (1976).
9. S. Cittolin, M. Demoulin, A. Fucci, W. Haynes, B. Martin, J. P. Porte, and P. Sphicas, "Third-Level Trigger and Output Event Unit of the UA1 Data-Acquisition System," *Proceedings of the International Conference on Computing in High-Energy Physics*, Oxford, England, April 10–14, 1989, pp. 370–374.
10. P. Rankin et al., "The 3081/E Processor and Its On-Line Use," presented at the Conference on Real Time Computer Applications in Nuclear and Particle Physics, Chicago, 1985.
11. A. Fucci and K. M. Storr, "Using the 3081/E Emulators in On-Line and Off-Line Environments," *Proceedings of the Three-Day In-Depth Review of the Impact of Specialized Processors in Elementary Particle Physics*, INFN, Padua, Italy, March 1983, pp. 213–228.
12. A. J. G. Hey, "Experiments in MIMD Parallelism," *Future Generation Computer Systems*, Vol. 6, North-Holland, Amsterdam, pp. 185–196.
13. P. A. Sphicas, "UA1 Experience with 3081/E Systems," *Proceedings of the Conference on Computing in High Energy Physics*, Asilomar, CA, February 2–6, 1987, pp. 339–343.
14. "The VMEbus Specification," Revision C.1, *Series in Solid-State Electronics*, Motorola Corporation, Inc., October 1985.
15. T. Nash, H. Areti, R. Atac, J. Biel, G. Case, A. Cook, M. Fischler, I. Gaines, R. Hance, D. Husby, and T. Zmuda, "The ACP Multiprocessor System at Fermilab," presented at the 23rd International Conference on High-Energy Physics, Berkeley, CA, July 16–23, 1986.
16. I. Gaines, H. Areti, R. Atac, J. Biel, A. Cook, M. Fischler, R. Hance, D. Husby, T. Nash, and T. Zmuda, "The ACP Multiprocessor System at Fermilab," *Proceedings of the Conference on Computing in High-Energy Physics*, Asilomar, CA, February 2–6, 1987, pp. 323–329.
17. T. Nash, "Event Parallelism: Distributed-Memory Parallel Computing for High-Energy Physics Experiments," *Proceedings of the Conference on Computing in High-Energy Physics*, Oxford, England, April 10–14, 1989, pp. 47–56.
18. J. Biel, H. Areti, R. Atac, A. Cook, M. Fischler, I. Gaines, C. Kaliher, R. Hance, D. Husby, T. Nash, and T. Zmuda, "Software for the ACP Multiprocessor System," *Proceedings of the Conference on Computing in High-Energy Physics*, Asilomar, CA, February 2–6, 1987, pp. 331–338.
19. *High-Accuracy Arithmetic Subroutine Library General Information Manual*, Order No. GC33-6163-2, 1986; available through IBM branch offices.
20. C. J. Georgiou, T. A. Larsen, P. W. Oakhill, and B. Salimi, "The ESCON Director: A Dynamic Switch for 200-Mb/s Fiber-Optic Links," *Research Report RC-16475*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1991.
21. *IBM System/370 Principles of Operation*, Order No. GA22-7000-10, 1987; available through IBM branch offices.
22. *SNA Type 2.1 Node Reference*, Order No. SC30-3422, 1991; available through IBM branch offices.
23. *SNA LU-6.2 Reference: Peer Protocols*, Order No. SC31-6808, 1990; available through IBM branch offices.
24. Noah Mendelsohn, "Transparent Remote Disk Access for the CMS File System," *Technical Report ZZ20-6467*, IBM Palo Alto Scientific Center, Palo Alto, CA, 1984.
25. G. P. Bozman, "VM/XA SP2 Minidisk Cache," *IBM Syst. J.* **28,** 165–174 (1989).
26. D. R. Cheriton, "The V Distributed System," *Commun. ACM* **31,** 314–333 (1991).
27. R. Rashid, D. Julin, D. Orr, R. Sanzi, R. Baron, A. Forin, D. Golub, and M. Jones, "Mach: A System Software Kernel," *Proceedings of IEEE Spring COMPCON 89*, IEEE Computer Society Press, San Francisco, February 1989, pp. 176–178.
28. Th. E. Anderson, H. M. Levy, B. N. Bershead, and E. D. Lazowsky, "The Interaction of Architecture and Operating System Design," *Proceedings of the ACM ASPLOS-IV Conference*, Santa Clara, CA, 1991, pp. 108–120.
29. J. C. Lasalle, F. Carena, and S. Pensotti, "TRIDENT: A Track and Vertex Identification Program for the CERN OMEGA Particle Detector System," *Nuclear Instrum. & Meth.* **176,** 371–379 (1980).
30. W. Gentsch, F. Szelenyi, and V. Zecca, "Use of IBM Parallel FORTRAN for Some Engineering Problems on the IBM 3090 VF Multiprocessor," *Technical Report ICE-0023*, IBM Rome Scientific Center, Rome, Italy, 1988.

Received October 2, 1990; accepted for publication January 22, 1991

Eckhard M. Ammann *IBM Data Systems Division, Development Laboratory, Schönicherstrasse 220, 7030 Böblingen 1, Germany.* Dr. Ammann received a diploma (M.S.) in mathematics in 1980 and a Ph.D. in information sciences in 1983, both from the University of Tübingen, Germany. His doctoral research focused on graph-theoretical models for self-diagnosis and fault tolerance of computer systems. In 1984 he joined IBM at the Böblingen laboratory, where he participated in development work for the IX/370 operating system. Since 1988 Dr. Ammann has been working on various aspects of PPCS, especially in the communication area. His current research interests include parallel systems and fault tolerance. He is a member of the Gesellschaft für Informatik (GI), Germany.

Robert R. Berbec *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Berbec received his M.S. in statistics from Stanford University in 1968. After joining IBM in 1968, he participated in the development and extension of the MVS operating system. Since 1988 he has worked on VM for PPCS. His research interests focus on operating systems and data structures.

Gerald Bozman *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Bozman is a programmer with a special interest in operating systems.

Michael Faix *IBM Data Systems Division, Development Laboratory, Schönicherstrasse 220, 7030 Böblingen 1, Germany.* Mr. Faix joined IBM Germany in 1962 and worked in the areas of programming of engineering/scientific applications and OS/360. In 1968 he joined the IBM Development Laboratory Böblingen and worked on computer architecture and evaluation, and System/370 processor and system design. In his more recent work he has participated in the development of the System/370 channel-attached IBM 4994 ASCII Device Control Unit, the System/370 CMOS II microprocessor used in the IBM systems ES/9370, 9371, and 3092 Model 4/5, and the dual-processor-based IBM 3092 Model 4/5 processor controller element (PCE) for large IBM

mainframes (ES/3090 and ES/9021). Mr. Faix's major interest is in architecture and system design. He is currently a Senior Technical Staff Member involved in the PPCS system design.

Gottfried A. Goldrian *IBM Data Systems Division, Development Laboratory, Schönicherstrasse 220, 7030 Böblingen 1, Germany.* Mr. Goldrian received his Diploma in electrical engineering from the Polytechnikum in Munich, Germany, in 1964. He joined IBM in 1967 and participated in the invention and the design of the first digital recorder for computer problem analysis. Since then he has worked in many different development projects in the Böblingen and San Jose laboratories. Mr. Goldrian is currently a Senior Engineer with major interests in analog and digital electronics, especially in computer interfaces. His contribution to the PPCS is the design of the interconnection network.

John A. Pershing, Jr. *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Pershing received his B.S. and M.S. degrees from MIT in 1975 and 1978, respectively. He worked for Bolt, Beranek, and Newman before joining IBM Research in 1981 as a Research Staff Member. His main interest is system coupling: specifically, the interaction of operating systems with communication network protocols and implementations.

Joann Ruvolo-Chong *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Ms. Ruvolo-Chong is a programmer.

Frank Scholz *IBM Data Systems Division, Development Laboratory, Schönicherstrasse 220, 7030 Böblingen 1, Germany.* Dr. Scholz received his diploma thesis and his Ph.D. in computer science from the University of Karlsruhe, Germany, in 1976 and 1981, respectively. After joining IBM in 1981, he participated in the development of the VSE and IX/370 operating systems. Since 1988 he has been working on programming environments for PPCS. His research interests focus on operating systems and parallel processing. Dr. Scholz is a member of the ACM, the IEEE Computer Society, and the Gesellschaft für Informatik (GI), Germany.