4-1-1988

Prospectus for a Remote PASM Execution and Debugging Environment - PDB

Thomas L. Casavant
Purdue University

James E. Lumpp Jr
Purdue University

Thomas Schwederski
Purdue University

Wayne Nation
Purdue University

A Prospectus for a Remote PASM Execution and Debugging Environment - PDB

Thomas L. Casavant
James E. Lumpp, Jr.
Thomas Schwederski
Wayne Nation
the PASM Working Group

TR-EE 88-16
April 1988

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

A Prospectus fcr A Remote PASM Execution and Debugging Environment — PDB

Thomas L. Casavant, James E. Lumpp, Jr., Thomas Schwederski, Wayne Nation, and the PASM Working Group

PASM Parallel Processing Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

Abstract

This document describes four design alternatives for a remote debugging and execution environment for the PASM Parallel Processing System Prototype in the School of EE at Purdue. Two alternatives involve acquisition of modest hardware for system enhancement, while the others are software-only solutions. All solutions involve use of a high-resolution bit-mapped graphics device, mouse and keyboard input, and a broad-band Ethernet-like communication medium. These latter components are currently available. The goal of this environment is to support any type of debugging which is currently supported by using the front panel of the machine and several terminals which are manually multiplexed between PEs and other resource management processors of the system. The environment will support *voluntary* output of processor activity from, and input to, any of the 30 processors of the PASM prototype. This configuration represents a step toward multiprogramming of the machine and will support development of software tools, languages and additional applications. Debugging information will be in the form of textual (or other) output displayed on virtual windows of a high-resolution device such as a SUN 3/50.

1. Introduction

For the past year, the 30 processor PASM system prototype has been operational and experimental work has been carried out. While very fruitful in itself, this work has illuminated the need for a more sophisticated programming environment. The current environment requires a user to be present in the room with the (very noisy) machine, and to have numerous terminals connected to the front panel to support monitoring of activities inside more than one processor at the same time. However, the functional characteristics of this interface are quite adequate for many phases of program development and support many areas of experimental parallel processing research — in fact, our goal. This project represents the next evolutionary phase of development of an environment for programming machines such as PASM.

This document assumes general familiarity with the PASM system and its prototype [SiS81, SiS87].

The immediate goal of this work is to provide identical functionality in the interface to the machine as described above, while also supporting:

1. Remote program execution and debugging.
2. A greater level of programmer visibility into the dynamic operation of PASM.
3. Multi-user access to the resources of PASM.

The foundation of the 4 solutions presented here is a broad-band (e.g., Ethernet) connection of a high-resolution work-station (e.g., SUN 3/50 or better) to the Parallel Computation Unit (PCU) of PASM. The work-station will then use a virtual windowing capability to simulate the simultaneous monitoring of each of the processors in the PCU as well as the numerous management processors of the system. The nature of the connection of the channel to PASM represents some special problems and a variety of feasible solutions. The solutions described here range from software-only solutions which place the bulk of the user-interface maintenance load on the SCU to solutions which place this load on the work-station with the SCU only taking on a management and error-reporting role. The other major concern in comparing alternatives is the level of interference with the computations being monitored created by the monitoring process itself.

2. Design Alternatives

The basic environment described in section 1 may be achieved in a number of ways. The collection of alternatives explored and described here each have different requirements in terms of hardware and/or software. The solutions also differ in their potential for interference with the actual parallel computation. The more hardware-intensive solutions are superior with respect to their ability to provide debugging information and checkpointing without altering natural program flow. The software-intensive solutions will be cheaper to implement, but will utilize the control path from the PCU through the MCs and SCU to route data to and from the PCU and SUN work-station. This will greatly affect the behavior and performance of programs — especially programs with a significant SIMD component. However, since none of the solutions are completely passive in their monitoring capabilities, the question is one of a matter of degree of interference.

2.1. Solutions Requiring New Hardware

2.1.1. Annex-Box Solution

The first solution described here is characterized by the use of an N-to-1 MUX/DEMUX hardware component which combines N RS-232 ports connected to the PASM PCU into a serial stream of data received and transmitted over an Ethernet channel (as shown in Figure 1). The hardware component is an **ANNEX Box** manufactured by Encore Computer Corporation [Enc86a]. This hardware will act as a de-multiplexor of data from windows on a SUN work-station to the processors of the PCU as input, and as a multiplexor/concentrator of data from the PCU processors back to the SUN. When tracing/debugging information is to be displayed, the programmer simply uses the RS-232 interface, and the output will be displayed in a window of the SUN. Any SUN work-station with access to the research Ethernet on ECN will be usable in the first implementation, and eventually any SUN on ECN will be usable. The software required on the SUN will be a program (labeled *smon* in Figure 1) using X-windows routines to maintain an up-to-date display of any information printed from each processor in the PCU. The program must interpret arriving data and display it in the correct window, and must also poll the keyboard and send data from the *active* window (identified by the mouse position) to the appropriate processor in the PCU. The initialization phase of this program will begin by requesting an allocation of processors from the SCU of PASM (this function is carried out by the program *pmon* shown in Figure 1). The SCU will then establish the sessions between itself, the ports of the ANNEX hardware, and the SUN smon program. The pmon block must then continue to monitor the parallel ports for control, status, and error conditions detected in the PCU and relay these to the smon block.

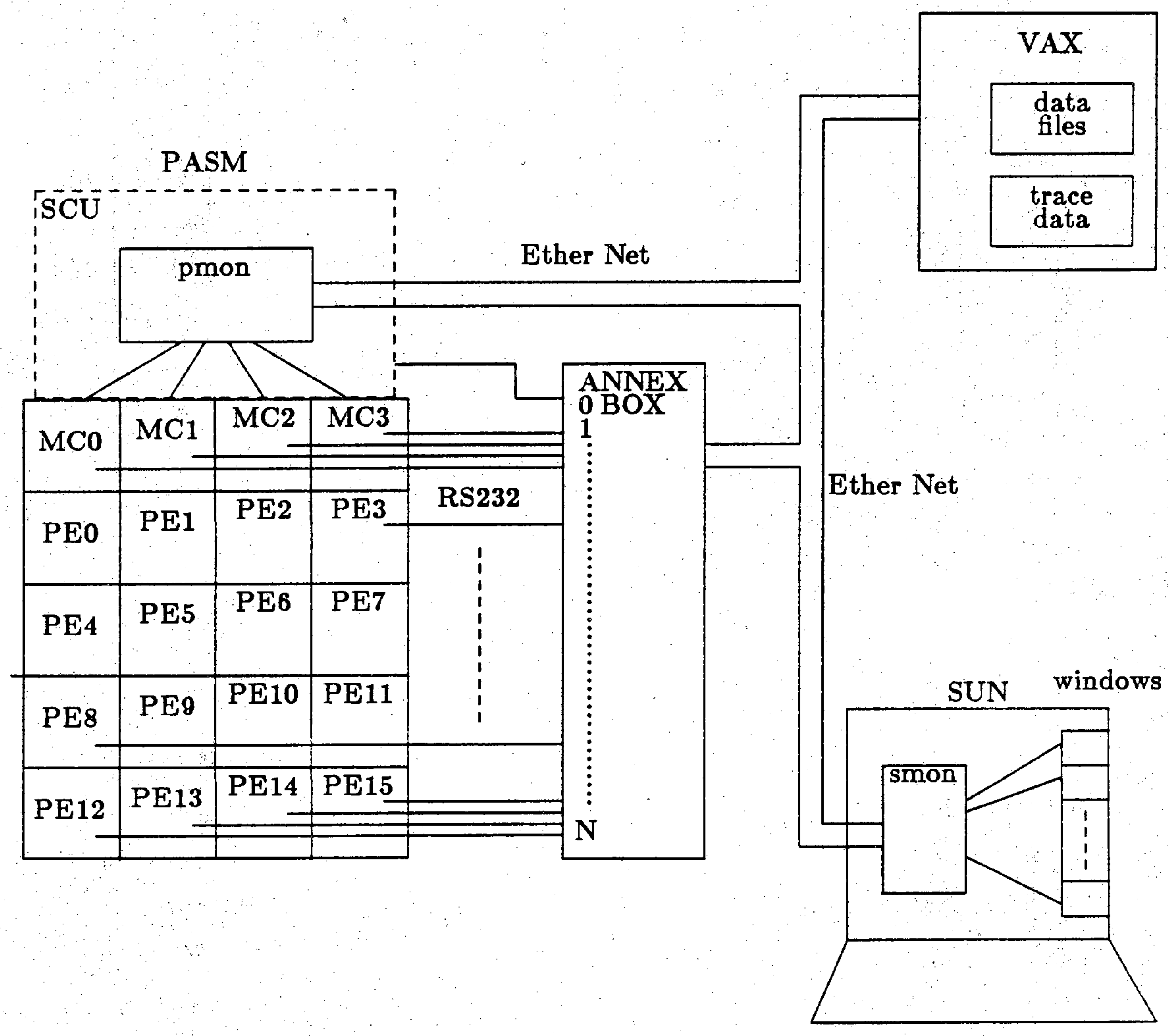

Figure 1.
ANNEX Solution

2.1.2. SMM Solution

An alternative to accessing the serial ports of all PASM CPU boards is by means of a system monitoring module (SMM), which is an enhancement to the PASM prototype I/O processor. Each SMM contains 8 MC68681 DUART chips, each of which implements two serial ports. Thus, 16 serial ports are available per SMM, and two

SMM boards are required to have a connection to each of the 30 PASM processors. The DUARTs are accessed by the CPU through the I/O-Channel, a local bus provided by the MVME110 CPU boards that are used in the PASM prototype. The I/O-Channel has the advantage of very simple interfacing logic. Below is a simplified block diagram of the SMM:

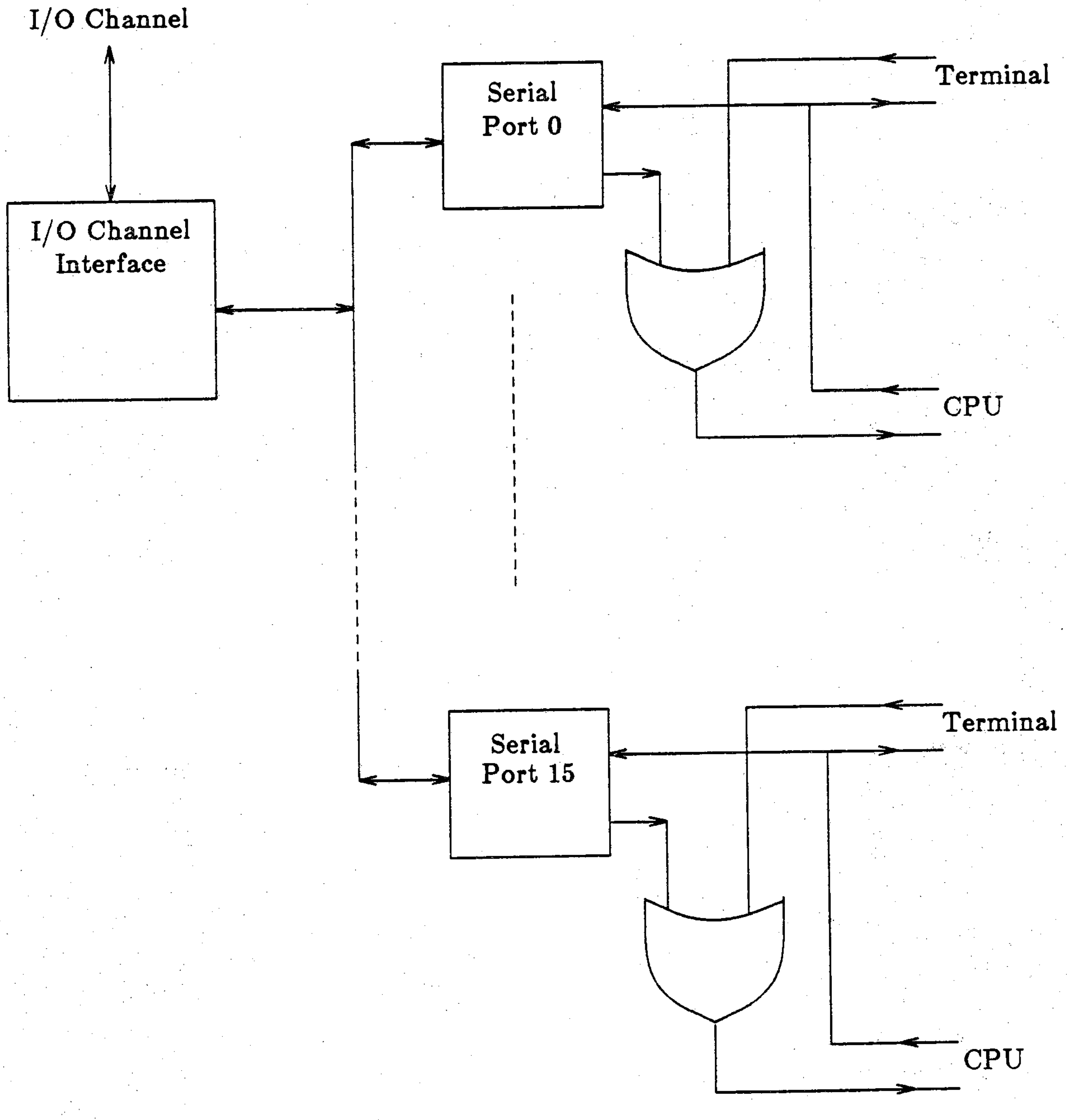

Figure 2.
Basic Block Diagram of SMM Solution

The I/O processor has a parallel port connection to the SCU. This connection should be of sufficiently high speed to keep up with messages to and from serial CPU ports. The SCU ethernet link can be used to connect PASM to the SUN workstation, similar to the other proposals. Figure 3 shows the relationship of the SMM to the rest of PASM and the display device.

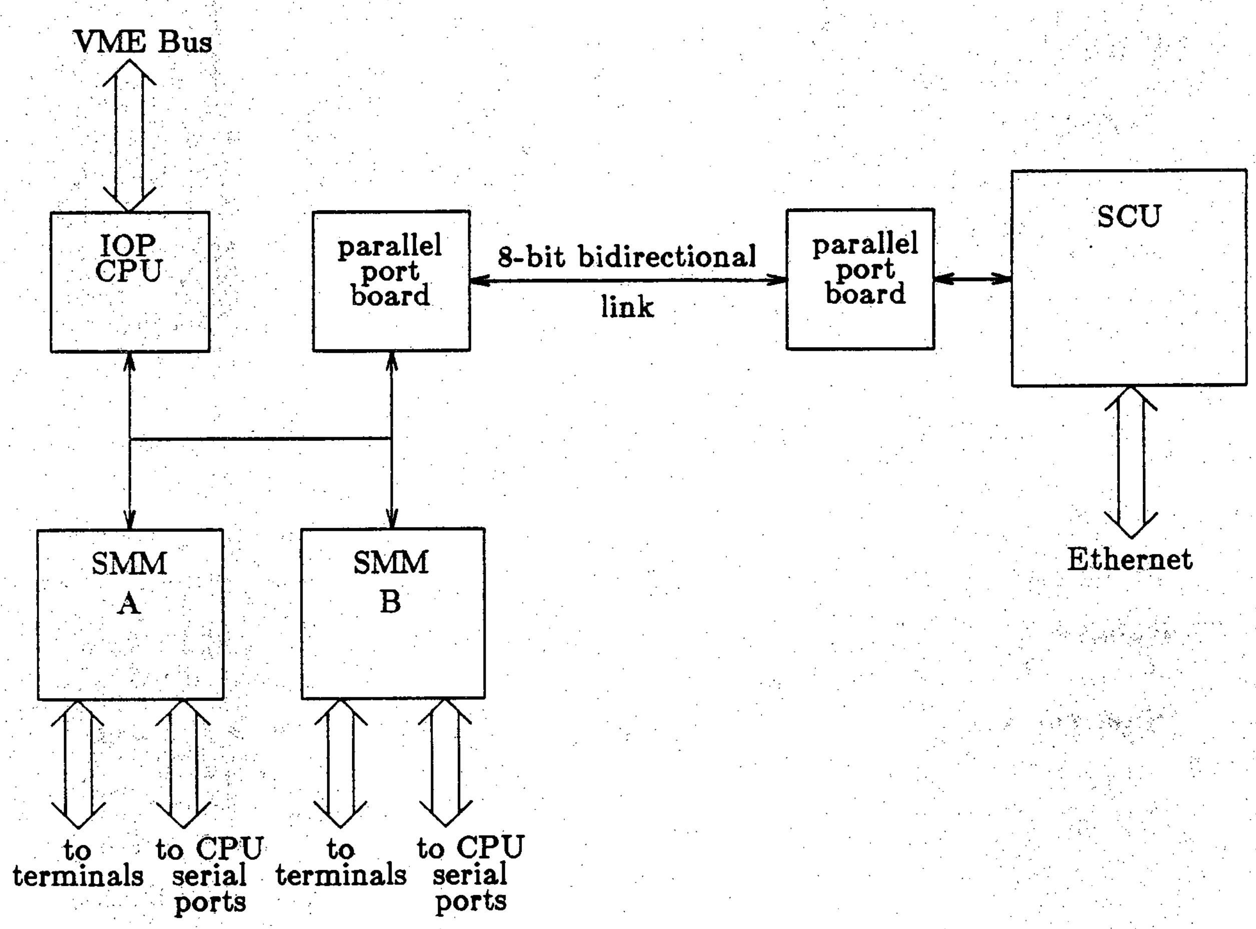

Figure 3.
Integration of SMM Hardware into PASM

Another advantage of the SMM is that the "Blue Box", i.e., the central serial port distribution box, can still be used, even without software support from the I/O processor. The SMM will automatically intercept all messages from a terminal connected to the blue box, and pass it on to the appropriate CPU board. It can also read such a message and pass it on to the SCU. On the other hand, messages from the SCU (or SUN) are passed to the Blue Box as well as the connected CPU. This way, a terminal connected to the Blue Box will display all traffic to and from the CPU serial port, whether originating from the terminal, the CPU, or the SMM. Also, the current wiring used to connect the Blue Box to PASM can be used, which will make it very simple to hook up the SMM boards. No new terminal cables need to be manufactured (which is always a pain).

Software Requirements

The software requirements are minimal. Because the PASM monitor supports downloading through the serial ports, no resident software needs to be developed, but all required programs can be downloaded via SCU to the I/O processor. The I/O processor will then constantly monitor the serial ports of the two SMM boards, either by polling them or in an interrupt driven mode. If a serial port requests service, the IOP will form a two-byte packet. The first byte will be identified by a '0' in its MSB, and will contain the source of the request. The second byte has a '1' in the MSB and contains the 7-bit ASCII character transmitted to the SMM. Software very similar to this has been written by PASM personnel and should not present any difficulties.

Consider the required data rates. If all processors of the Parallel Computation Unit (i.e., 16 PEs and 4 MCs) send data at maximum speed (9600 baud), a data rate of 24K byte/second results. The parallel port therefore needs twice this rate (approx 50Kbyte/sec) to keep up with the incoming characters. To the best of my knowledge this is far below the throughput provided by the parallel port links.

On the other hand, the processor must be capable of handling these data rates. If polling is employed (until the IOP is used in its original capacity there is no reason why polling should not be used), no interrupt overhead need be considered. Each polling action requires one memory access (for the test) and a branch, followed by either the next test access or by the sending to the parallel port. When 50K bytes have to be transmitted per second, and an average memory access is pessimistically estimated at 1 microsecond, 50 memory accesses are allowed per transferred byte. This seems within easy reach of efficient assembly level programming of the required task. In addition, this worst case scenario of 20 sending CPUs is rather unlikely to be sustained.

Hardware and Development Costs

The most expensive parts of the SMM are the DUART chips. These we got as donations from Motorola a long time ago; most of the other parts we have as well, leftovers from other boards. The only costs will be approx. $50 for some additional parts we do not have in sufficient quantities, and another $50 for artwork and PC board manufacturing.

Status

The circuit diagrams of the SMM have been designed and a circuit board layout will most likely be finished within one week. Manufacturing of the blank board (i.e., artwork plotting, photographic reduction, board etching, drilling of holes) should take another week. The time to debug hardware and software is always difficult to predict. However, large parts of the board are reused from previous designs and are therefore known to work properly. The serial port connections are very simple. Thus, a feasible implementation time for the first board to work should be approximately one month from now (April 15th).

2.2. Software Only Solutions

The two *software-intensive* solutions arise from the possibility of parallel I/O through the hierarchy of PASM itself. It is possible for input and output of the PCU to be routed through GPIB ports to the MCs and through the parallel ports to the SCU. From there , data packets would be forwarded through the Ethernet to a SUN or other monitoring device. The principle requirements are a program on the SCU to take data from the MCs, a means to forward that data to the SUN, and routines on the SUN to handle the I/O. The main drawbacks to each of these approaches are:

1. The possible bottle-neck of data at the MC and SCU.
2. The inevitable degradation of system performance due to interference with MC operation in SIMD or Hybrid SIMD/MIMD mode, and
3. The complexity of user-driven debugging in SIMD mode. This arises since in SIMD mode, data/variables which reside in the PEs must be directed back to the MC under the control of the MC.

Each of these considerations will be discussed in the descriptions following.

The program running on the SCU (pmon), in addition to providing the error and program control outlined in section 2.1, will now be responsible for the identification and packaging of I/O to and from the PCU elements. Since the most common path for I/O will be PE→MC→SCU→SUN this will serve as an illustrative example.

Information from a specific PE will be sent through a GPIB port to its corresponding MC. For the sake of performance preservation, it will be necessary to *packetize* output from the PE. It would be very inefficient to forward individual characters due to the overhead of identification. This requirement places the first restraint on the *free-flow* of I/O as provided by the current multiple terminal configuration. Next, the MC will process the packet by labeling it as to which PE in its group the data is from and forwarding via parallel ports to the SCU. Here, then, is where the bottle-neck may occur (i.e., it will not be possible for multiple PEs to output in SIMD mode) and additional measures may have to be taken to prevent loss of data if more than one PE attempts to output in MIMD mode. These problems are not present in the hardware approach of section 2.1. Next, the SCU will identify from which MC the data originated, determine if it is I/O from the MC itself or one of its PEs and label it as such.

At this point, all that remains to be done is to send the packet to the appropriate window on the sun. This set of steps is handled differently in the two software approaches proposed and comprises the principle difference between them. These differences lie in the Ethernet and X-window interfaces.

2.2.1. SCU-Intensive Solution

The SCU-intensive solution would handle the communication through XLIB software running as an application library extension to the SCU system software. This library includes features which emulates a terminal across the net and supports TCP sockets. In this case, pmon interfaces with the X-window library routines and maitains the multiple virtual terminals by stripping the origination information from each packet, and sending the data through the appropriate virtual terminal. Much of the development of the virtual terminal manipulation routines could be done on a SUN or VAX. This routine would then be ported to the SCU and would receive information from the parallel ports through a routine developed on the SCU itself which polled the parallel ports.. Note, that in Figure 4, this is illustrated by the movement of smon from the SUN to the SCU. The functionality of pmon and smon may be merged in this case, thus eliminating one level of message passing between clients of the tool.

This approach would require no code to be written for the SUN, however the major drawback here is the need to port the XLIB routines needed by smon to run on System-V (currently native on the SCU). X-windows was developed on 4.3 BSD and the port may pose a considerable challenge. A second possible drawback is the performance degradation of the SCU which could further heighten the data bottleneck phenomenon.

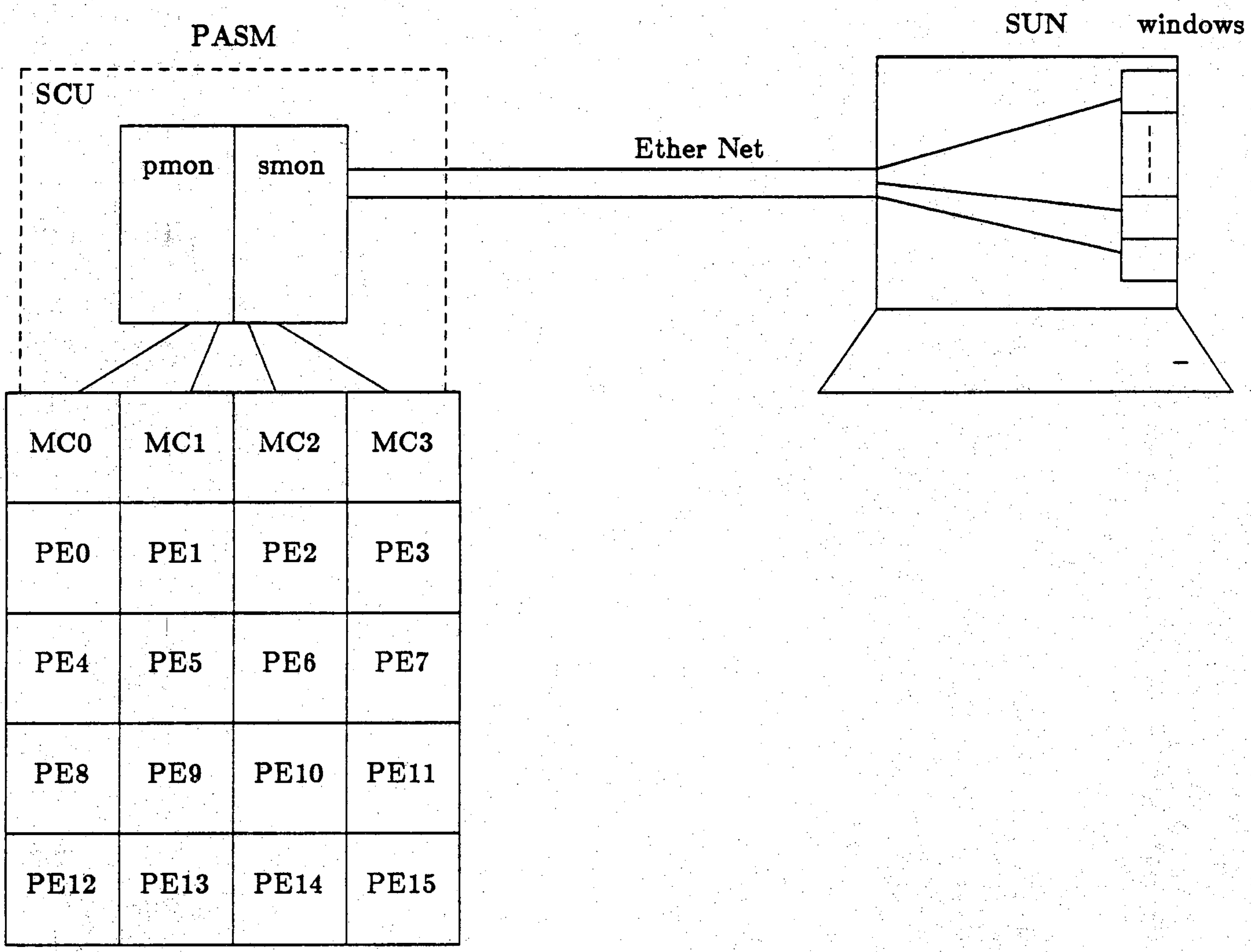

Figure 4.
SCU-Intensive Solution

2.2.2. SUN-Intensive Solution

This alternative would alleviate some of the load on the SCU by moving the X-window support (i.e., smon) from the SCU to the SUN while still utilizing the PE⟶MC⟶SCU⟶SUN path for data. In this case, pmon must multiplex MUX all the packets from the MCs into a single pipe or socket for transmittion across the Ethernet. The packets would be received by the smon routine on the SUN, the origination information would be stripped, and the data displayed in the correct window through X-window calls by smon. Input from the user would be packetized and labeled with a destination and forwarded to the appropriate PCU element based on the mouse location determined by smon.

The SCU is spared the burden of managing the XTERMs (i.e., smon now runs on the SUN, not the SCU) but must still interpret the packets from the SUN, strip off the control bits, and send the data to the correct part of the PCU. In the SCU-intensive case pmon/smon needs to packetize data coming from the virtual terminals (PCU). The virtual terminals would automatically send characters across as they are typed while the smon would have to packetize to avoid a performance degradation. With this SUN-intensive case the packetization and lableing would take plac on the SUN. A SUN 3/50 has a Motorola 68020 running at 16 MHz while the SCU is a 68010 running at 10 MHz. Thus, the packetization in the SUN-intensive solution could be done much more quickly on the SUN thus avoiding the potential bottle-neck there.

The most important point of comparison between these two software solutions would be the porting of the X-window libraries to System-V as opposed to the potentially *easier* manipulation of windows in the SCU-intensive version. The disadvantage of the second software solution would be the extra code to be written for the SUN to manipulate the windows. However, performance is expected to be significantly greater.

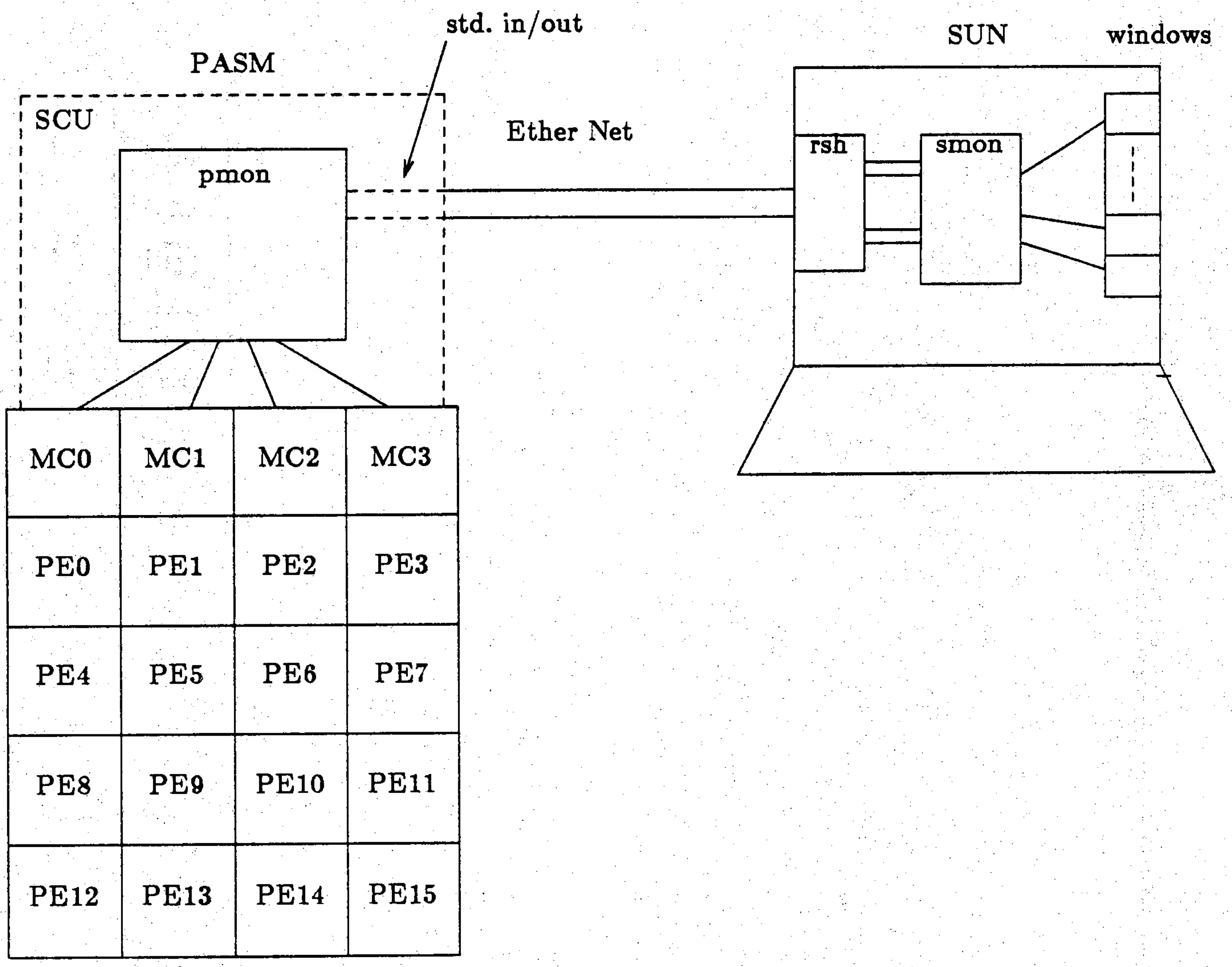

Figure 5.
SUN-Intensive Solution

References

[Enc86] Encore Computer Corporation, "ANNEX Hardware Installation Guide," and "ANNEX User's Guide," Document #'s 716-02887 and 716-02886, 1986.

[SiS81] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: a partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Transactions on Computers,* Vol. C-30, December 1981, pp. 934-947.

[SiS87] H. J. Siegel, T. Schwederski, J. T. Kuehn, and N. J. Davis IV, "An overview of the PASM parallel processing system," in *Computer Architecture,* D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furht, eds., IEEE Computer Society Press, Washington, D.C., 1987, pp. 387-407.