I. Introduction

This paper details the architecture of a parallel microcomputer that allows the straightforward representation and efficient execution of a "parallel" emulator. A parallel emulator refers to emulators not only for parallel machine languages but also those that emulate sequential machine languages in a parallel manner.

A parallel emulator can most naturally be structured as a set of microprocesses, each performing a small independent task, that interact in a closely-coupled manner [LES73]. Just as the level of an intermediate (virtual) machine language (i.e. an IML) was introduced into the interpretation process to more efficiently execute higher level languages, it is felt that to construct efficient and natural emulators for parallel IMLs a new level of variability must be introduced into the emulation process. This new level, which will be called the virtual process-memory-switch environment, allows the designer to structure an emulator in terms of an arbitrary configuration of microprocesses and their associated interaction patterns without directly dealing with how microprocess activity is mapped into microprocessor activity (see Figure 1).

These closely-coupled interaction patterns among microprocesses must be implemented efficiently, otherwise the overhead of interaction will significantly reduce the utilization of hardware parallelism. Thus, the unified design approach detailed in the following quote is especially required for the architecture of a parallel microprocessor:

"Adequate performance of parallel processing systems is...predicated on an appropriately low level of overhead. Allocation, scheduling, and supervisory strategies, in particular, must be simplified and the related procedures minimized to comprise a small proportion of the total activity in the system..........thus a unified and integrated design approach is required in which software and hardware, operating system and processing units, lose their separate identities and merge into one overall complex, for which allocation and scheduling procedures, for example, are as basic and as critical as arithmetic operations." [LEH66]

The concept of a virtual PMS environment viewed from a hardware orientation...

---

* This paper presents material from the fourth and fifth chapters of [LES72]. The research was carried on while the author was with the Department of Computer Science, Stanford University, Stanford, Calif. and was partially supported under AEC contracts AT(04-3)326,P.A.23 and AT(04-3)515 Stanford Linear Accelerator Center, Stanford, Calif.

** See [BEL70] for definition of "processor-memory-switch".

*** A more detailed justification of the need for this new level is contained in [LES73].
Figure 1. A New View of Emulation

Figure 2. A New View of Relationship between Emulation Process and Microcomputer Architecture
represents a unification of the concepts of residual control [FLY71, ROS71], virtual memory [WIL72, DENG66], and dynamic (restructurable) control structure [LES71]. The virtual PMS provides the microprogrammer with the capability for reconfiguring both the internal and the external environment of a microcomputer system. The concept of residual control allows the varying of the number and functions of the internal working registers of each microprocessor; the concept of virtual memory allows the varying of the structure of memory, e.g., its size and word length; the concept of a dynamic control structure allows the varying of the number of microprocessors and functional units, their interconnections and their interaction patterns.

A virtual PMS capability has been made an integral part of the architecture of this microcomputer (see Figure 2). This has been accomplished by adding a new level of control. There are two distinct hardware levels of control in the architecture that are structured in a hierarchical fashion: the conventional level of control, contained in each microprocessor, for the sequencing of microinstructions, and a new level of control for the sequencing of microprocessors and functional units; thus, the microcomputer system contains both local distributed control structures and a global, system-wide control structure.

The global control level performs the mapping of microprocessor activity into microprocessor activity. This control level can be thought of as a simple hardware operating system which manages the scheduling of and interactions among microprocessors. The control rules for this new level of control are implemented in hardware which is distributed in each microprocessor and in the controller for the bus(ses) which are used for inter-processor communication. As previously stated, this new level of control must be an integral part of the hardware organization because, otherwise, the overhead in implementing interaction patterns, where the closely-coupled parallel activity is of short duration, will overwhelm the inherent parallelism of the interaction patterns.

Microprocessor activity is specified through a data base, the control data structure (CDS), held in a separate memory called the Process Space Memory (M.PSM). The CDS, in a very general sense, can be considered a control structure definition program which, when interpreted by the global control rules of the microcomputer system, defines a particular sequential or parallel control structure for sequencing of microprocessors and functional units. The CDS consists of a collection of state vectors, each state vector defining a microprocess. A particular set of interaction patterns among microprocessors is dynamically defined by varying the number of and relationships among these state vectors.

A Microprocess State Vector (MSV), as pictured in Figure 3, consists of a number of components. Through manipulation of these components, a wide variety of different control (interconnection) and data environment (interaction) relationships can be defined among microprocesses (microprocessors). For example, a broadcast control structure could be defined among a group of microprocessors by each microprocessor having the same port registers,* or coroutine interaction by two microprocessors having the same value (operand) stack register, etc. The number and location of registers pointed to by the state vector components is definable by the microprogrammer. The registers pointed to are held together with state vectors in the Process Space Memory. A complete explanation of the structure of the state vector is contained in [LES72].

* The port of a microprocess is a block of registers specified in the MSV which can be used to directly communicate data to the microprocess while at the same time activating that process.
Figure 3. Structure of Microprocess State Vector

The main thrust of the remainder of this paper is to demonstrate that a logical hardware organization can be developed for this microcomputer architecture such that parallel activity expressed on the virtual PMS level can be mapped, without significant overhead, onto parallel activity on the physical PMS level. In addition, the hardware organization must guarantee that the mapping (scheduling) of virtual activity to physical activity neither introduces hardware resource deadlocks nor changes a deterministic virtual activity into a non-deterministic physical activity. In particular, the latter requirement implies that microprocess synchronizing primitives must work correctly, independent of the number of physical microprocessors and the particular mapping of microprocesses onto microprocessors.

II. A Hardware Organization

The hardware organizational problems of minimizing overhead, guaranteeing no resource deadlocks, and correctly implementing synchronization mechanisms are all faced in the design of a conventional multiprocessor [LOR72]. However, these design problems are significantly magnified in the context of this new microcomputer architecture since microprocessor interaction patterns can be very closely-coupled and can occur on a very fine time grain, i.e., the time between successive interactions can be of very short duration. Thus, there is a greater likelihood for microprocessors to interfere with each other when they 1) access the Microprogram Memory for microinstructions, 2) access the Process Space Memory for shared data items, and 3) communicate with one another. In addition, the time to perform a context switch in a microprocessor from one microprocessor to another is especially critical because of the potential for a high rate of context switches. A high rate of context switches may occur because 1) the time between successive interactions of the microprocess is generally of short duration, and 2) in order to avoid a deadlock in mapping virtual activity to physical activity, the microprocessor must be multiprogrammed when there are more active microprocesses than
microprocessors. Thus, when a microprocessor is connected to a microprocessor that is waiting for a response to a communication, the microprocessor is context switched if another microprocess exists that is active but not connected.

These conventional multiprocessor design problems are compounded with the problem of efficiently implementing the concept of a virtual PMS. The efficient implementation of such a concept requires that:

1) There is a built-in hardware scheduling algorithm.

2) The internal working registers of a microprocessor are dynamically reconfigurable so as to conform to the particular microprocess being executed.

3) Microprocess interaction patterns are directly implemented as microprocessor interaction patterns rather than indirectly implemented as modifications to the Process Space Memory whenever possible, dependent upon the particular connection between the virtual PMS and physical PMS at the time of the interaction.

The remainder of this section will be a discussion of the hardware organization for the microcomputer architecture. This organization demonstrates that there exists a feasible solution to the design problems previously outlined and, further, that this design is a coherent, integrated solution to these problems. However, this paper will not discuss the hardware technology required to implement the bus structure and memories specified in the design [BEL71, LOR70, MIL70], but rather, will focus on the interconnection patterns and interaction protocols among the "black boxes" that define the PMS environment of the microcomputer architecture, i.e., the logical hardware organization. Section III will, however, investigate the dynamic aspects of the organization using a detailed bus level simulation.

II.1 A PMS Configuration for the Microcomputer Architecture

An overview of the PMS configuration for the microcomputer architecture is pictured in Figure 4 (notation is that of Bell and Newell, [BEL70]). This configuration consists of NP microprocessors and ND devices (functional units) which can directly communicate with each other over an Interprocessor Bus (IB). Each microprocessor and functional unit has separate control hardware, K.IBP and K.IBD respectively, for interacting with the IB. The overall control of IB resides in the Virtual Interaction Controller, K.VIC. K.VIC, together with the K.IBP and K.IBD, represent the hardware for mapping microprocess interaction patterns into microprocessor interaction patterns.

There are three external memories contained in the PMS: M.PSM, M.MPM, and M.MEM. M.PSM, which is the Process Space Memory, is directly accessed by a microprocessor in order to retrieve and modify the working registers of the microprocess it is executing. A microprocessor also indirectly accesses the M.PSM through K.VIC in order to obtain a state vector. The M.MPM, which is the Microprogram Memory, is accessed by microprocessors directly in order to fetch microinstructions. The M.MEM, which is the main Memory Subsystem and holds the state image of the emulated machine, is a bit addressable memory. The K.MEM is the control circuitry required to perform the appropriate shifting to align the desired bit string.

The PMS configuration presented in Figure 4 does not indicate the number of
These data paths are optional depending upon the bandwidth required by the device.

Figure 4. PMS for Microcomputer Architecture

Figure 5. A Detailed PPS for Microcomputer Architecture
independent communications that a bus can handle, nor the interleaving of a memory. These PMS characteristics have been purposely omitted because they can be varied in the simulator of the hardware organization. In addition, the simulator permits the reconfiguration of the bussing structures so that the bussing structure for interprocessor communication can have the additional function of being an access path to any one of the three external memories. In this manner, the PMS can be configured to have from 1 to 4 independent busses.

An example of a more detailed configuration of the PMS for the microcomputer architecture is pictured in Figure 5.

II.2 The Interprocessor Communication Structure and the Virtual Interaction Controller

This section discusses the major logical design issues involved in mapping virtual PMS activity to physical PMS activity:

1) The bussing structure for interprocessor communication.

2) The design requirements necessary to insure no hardware deadlocks are introduced which are not already present as software deadlocks.

3) The hardware algorithm for scheduling of microprocessors on microprocessors.

One of the major considerations in this design is to multiprocess rather than to multiprogram virtual microprocessor activity whenever possible. The design of the Interprocessor Bus (IB) strongly reflects this major design consideration. The IB is designed so that, whenever possible, microprocessor interaction patterns are directly implemented as microprocessor interaction patterns. The IB transforms a microprocessor interaction pattern directly into a microprocessor interaction pattern by acting in a manner similar to the common data bus on 360/91 [AND67] or equivalently the UNIBUS on PDP-11 [DEC69]; a similar technique for handling interprocessor communications has also been suggested by Lehman [LEH66].

A microprocessor P1, when executing a microprocess V1 that desires to initiate a communication with another microprocess V2, sends out a request on the interprocessor bus for the MSV of microprocess V2 so as to determine whether V2 is in an acceptable state to receive a communication from V1. Each request sent out on the bus is scanned by the control circuitry KIBP (associated with each microprocessor) in order to determine whether that microprocessor is currently connected to the microprocess V2. If a microprocessor P2 is connected to microprocess V2 and is not currently involved in a dialogue with another microprocessor, then microprocessor P2 will honor the request for the MSV of V2 and transmit this MSV directly over the bus to microprocessor P1. However, if there exists no microprocessor to honor the request, then the Virtual Interaction Controller will take over responsibility for fetching the MSV of V2 from the MPSM and then transmitting this MSV to microprocessor P1. Once a connection has been established between microprocessor P1 executing microprocess V1 and microprocessor P2 executing microprocess V2, the bus can be used to transmit a communication between microprocesses V1 and V2 directly. In particular, if a communication can be consummated between V1 and V2, then the bus can be used to transmit the new MSV of V2, and the data to be stored in the port of V2. After the communication has been completed, a signal is sent by P1 to P2 on the bus to break off the connection between the microprocessors.
so that microprocessor P2 can again directly receive communications from other microprocessors. This ability to implement microprocess communications directly significantly decreases the time required to perform a communication in comparison to the time required to implement the communication indirectly through multiple fetches and stores to the MPSM.

The Virtual Interaction Controller (K.VIC), when it receives a request for a microprocessor state vector (MSV), checks before fetching the MSV from the MPSM whether some other microprocessor is currently accessing the desired MSV, or if the desired MSV is currently connected to a microprocessor. The latter case implies that another microprocessor is already involved in a dialogue with the microprocessor connected to the desired MSV. If either case is true, the Virtual Interaction Controller will place on a FIFO queue associated with the locked-out MSV the address of the microprocessor requesting access. The request to access the locked-out MSV will then eventually get honored when that request is at the head of the queue and the microprocessor currently accessing the MSV again permits access to the MSV. Only one microprocessor can access an MSV at a time because of the semantics of microprocess interaction patterns which specify that a microprocess can only receive one communication at a time. Thus, a microprocessor must wait until the microprocessor which is currently attempting to communicate with the desired microprocess either decides that communication cannot be completed or completes the communication.

The FIFO queues associated with the locked-out MSVs are contained in a storage area associated with K.VIC. In addition, this local storage contains the current activity status of each microprocessor and functional unit, and the MPSM addresses of the MSVs that are currently connected to microprocessors. The maximum number of requests that can be queued is \( N + N - 1 \), since a microprocessor or functional unit can only initiate a single communication at a time. Thus, the size of the local storage area required by K.VIC is only a small linear multiple of the number of microprocessors and functional units.

A request for an MSV, if it cannot be immediately honored, is queued rather than reissued by the requesting microprocessor at some later time for two reasons. The first and most important reason is that queuing eliminates the potential for a microprocessor resource deadlock. A simple example where microprocessor resource deadlock can occur when there is no queuing of requests is the following.

Example 1: Consider four microprocessors A, B, C, and D, where microprocessors B, C, and D are all connected to microprocessors and are all simultaneously attempting to initiate a communication with A. Suppose that microprocess A is not in the appropriate execution state to receive a communication from B or C, but is ready to receive a communication from D. Given this situation there then exists the possibility that every time microprocess D attempts to fetch the MSV of A, microprocess B or C has locked the MSV of A in an attempt to determine whether A is in an appropriate execution state to receive a communication. Thus, microprocess D never consummates a communication with A and A never alters its execution state so that B or C can consummate a communication. Therefore, a hardware deadlock is introduced which is not present as a software deadlock.

The situation described in the example is not unrealistic when there are closely-coupled interaction patterns among a large number of microprocessors. In addition to the queuing mechanism contained as part of the K.VIC, there is also a need for a bus control mechanism, e.g., a bus commutator, to guarantee that eventually a microprocessor will be able to get a free bus cycle in order to send out a request for an MSV.
II.2.1 Microprocessor Scheduling Strategy

The Virtual Interaction Controller is also responsible for dynamic hardware resource allocation in the microcomputer system. It dynamically allocates microprocessors in a manner analogous to the way the data elements of a cache are allocated [Gilb67]. In particular, the Interprocessor Communication Structure is analogous to a cache in the following ways:

1) The relationship between a data element in the cache and its corresponding data element in the large primary memory is analogous to the relationship between a microprocessor contained in the physical PMS and its connected microprocess defined in the virtual PMS.

2) A direct hit to the cache is analogous to a microprocess interaction pattern being directly implemented over the Interprocessor Bus.

3) A miss in the cache which then requires access to the large primary memory is analogous to a microprocess interaction that cannot be implemented directly over the Interprocessor Bus but instead must be indirectly implemented through modifications to the Process Space Memory.

4) Associating a data word in the cache with a different word in the large primary memory, is analogous to context switching a microprocessor to another microprocess.

5) An empty data word in the cache is equivalent to a microprocessor which is not connected to any microprocess; a read-only data word in the cache is equivalent to a microprocessor which has performed all the work necessary for a context switch, but has not yet been switched to another microprocess; and, a read-write data word in the cache is equivalent to a microprocessor which is currently executing a microprocess.

This analogy has been explored in depth because it provides a convenient framework within which to think about how to schedule microprocesses on microprocessors. In addition, this analogy leads to some interesting ways of looking at the relationship between the number and structure of microprocesses, and that of microprocessors. In particular, the concept of "working set" [DEN68] which is normally applied to data, seems applicable also to microprocesses, i.e., control working set; and, the concept of fetching consecutive data items for each line of a cache can also be applied to context switching a group of microprocesses rather than a single microprocess. Though these ideas have not been explored further in their implication on the hardware organization, simulation data will be presented which explores the phenomenon of the control working set.

The allocation (scheduling) mechanism of the Virtual Interaction Controller is invoked when a request is received to activate a microprocess which is not currently connected to a microprocessor. If there exists a microprocessor which is not currently connected, then
the microprocess is connected to this processor. Otherwise, the microprocess is connected to the least-recently-used microprocessor whose associated microprocess is in a busy-waiting or stopped-execution state. If there exists no such microprocessor, then the address of the MSV of the process is placed on a FIFO queue stored in MPSM until a processor is available. Note, however, that a microprocess is not a candidate for rescheduling if it is waiting to access the state vector of another microprocess because the queing mechanism guarantees that the wait is finite and short. This is in contrast to the busy-waiting state where there is no guarantee a microprocess will ever change its state to accept the requested communication.

In summary, the microprocessor scheduling strategy, implemented in the Virtex Interaction Controller is simple, is independent of the number of microprocessors, and maps virtual PMS activity into physical PMS activity without introducing deadlocks. In addition, this scheduling strategy attempts to maximize the number of microprocess interaction patterns that can be directly implemented as microprocessor interaction patterns. As will be seen in the hardware simulation results, this scheduling strategy will take advantage of additional microprocessors added to the microcomputer system to 1) increase the parallel activity of the microcomputer system, 2) increase the number of microprocess interaction patterns that can be directly implemented, and 3) decrease the number of context switches.

II.3 The Microprocessor Organization

The microprocessor organization is based on the following goals:

1) to configure the internal registers of a microprocessor so as to match the particular internal register configuration of the virtual microprocessor (microprocess) being executed.

2) to reduce the interference among microprocessors caused by accessing the Microprogram Memory and the Process Space Memory.

3) to make the overhead time required to context switch small.

The first goal is necessitated by the ability to define an arbitrary number of registers that will be contained in the local and global data environment, port, value stack and program counter stack of a virtual microprocessor. Thus, the conventional solution of assigning a fixed set of microprocessor hardware registers for each of these data structures cannot be used in this context. The second goal follows from the fact there may be 32 or more microprocessors all attempting to access the Process Space Memory and Microprogram Memory, possibly some of these microprocessors attempting to access the same microinstruction. Conventional solutions of interleaving memory and multiple access paths to memory are only partial solutions because of the very high traffic to these memories. Thus, some technique is required for cutting down the total traffic to these memories. The third goal is a result of the requirement to multiprogram microprocess activity in order to avoid deadlocks. This requirement to multiprogram combined with the short length of the computational activity of a microprocess between successive waits for a communication from other microprocesses leads to a high number of context switches.

* A busy-waiting state occurs when a microprocess repeatedly attempts to initiate a communication with a microprocess which is not ready to receive a communication.
Thus, the requirement for a context switch to be done with low overhead is quite important. These three design goals can all be satisfied through the use of a cache for each microprocessor.

The microprocessor organization contains an extremely small number of dedicated (specific function) internal hardware registers. The remainder of the internal storage of the microprocessor is structured as a memory cache. The memory cache is used to hold either microinstructions or M.PSM registers which contain the local and global data environment, port, value stack and program counter stack of a virtual microprocessor. The cache per microprocessor concept satisfies the three design goals mentioned previously in the following ways:

1) It permits the internal hardware registers to be configured so as to match the register configuration of a particular virtual microprocessor; the cache accomplishes this configuring by attaching tag information to each internal register of the cache. These internal registers are associatively addressed, based on the tag information, in order to access the contents of a particular M.PSM register.

2) It reduces the traffic to the Microprogram Memory and the Process Space Memory by allowing a percentage of the accesses and stores of M.PSM registers and the accesses of microinstructions to be accomplished without interaction with the Microprogram Memory and Process Space Memory.

3) It reduces significantly the time to context switch because only the M.PSM registers that have been changed need be stored, and only the registers that are required to execute the microprocess need be loaded into the microprocessor.

The implementation of the cache and its associated control in each microprocessor differs from conventional implementations because this microcomputer system contains multiple microprocessors. These multiple microprocessors can be simultaneously executing microprocesses that are communicating through a shared data area in the Process Space Memory. A copy of the contents of these M.PSM registers contained in a shared data area cannot be held in multiple caches simultaneously. The problem of deciding whether a M.PSM register should be stored in the cache is solved by requiring all accesses and stores of M.PSM registers to be indirect through a descriptor. A M.PSM descriptor specifies, among other things, the "sharability attribute" of the register to be accessed or modified. There are three modes for the sharability attribute: global, coroutine, and local. A register which has the global attribute will never be held in the cache; a register which has the coroutine attribute may reside in the cache until the microprocess either is about to enter the stopped-execution state or is disconnected from its microprocessor. The coroutine attribute indicates that only one microprocess at a time will access the shared data. Thus, while the microprocess is in a running execution state, the coroutine data may reside in the cache. The local attribute indicates that the data may reside in the cache as long as the microprocess is connected to a microprocessor since no one else can access the data.

In summary, this section has indicated a coherent and simple design for the microcomputer system which allows parallel activity on the virtual PMS level to be mapped correctly and without significant overhead into parallel activity in the physical PMS level. The next section will demonstrate these assertions by examining the results of simulating this microcomputer organization while executing an emulator of a parallel machine language.
III. The Performance Characteristics of the Hardware Organization

In a previous paper [LES73], it was shown that parallel activity on the virtual PMS can be mapped into an equivalent amount of parallel activity on the physical PMS. This section will discuss, in detail, how the overhead of performing this mapping varies as a function of the amount of parallel activity and the number of microprocessors. This data has been generated from a detailed simulation of the execution of a parallel emulator on this proposed hardware organization.

There are two components of overhead: hardware interference and program interference. Hardware interference arises from the sharing of physical resources such as memories or microprocessors, while program interference arises from the sharing of virtual resources such as two microprocessors attempting to simultaneously activate the same microprocess. These two components of overhead are analyzed by comparing three measures of hardware parallelism: allocated, effective and throughput hardware parallelism. Allocated parallelism measures the actual parallelism at the hardware level, i.e., the average number of microprocessors executing on each bus cycle. Effective parallelism measures how much of the allocated parallelism is being used to perform useful work, i.e., the amount of allocated parallel activity normalized to the average execution time of a microinstruction for a single microprocessor. Throughput parallelism measures the relative speedup of the emulator compared to the time to execute for a single microprocessor. These three measures are plotted as a function of the number of microprocessors for the emulation of a sample parallel machine language program in Figure 6. The PMS configuration used in the simulation is diagrammed in Figure 5.

The first observation which can be made from Figure 6 is that the hardware interference and program interference increase as a function of the number of microprocessors. This observation on hardware interference can be explained in the following way: the more microprocessors, the more virtual PMS parallel activity that can be exploited as physical PMS parallel activity; in turn, the more physical PMS activity increases the likelihood of overloading the bussing and memory structures; this overloading leads to a longer time to execute microinstructions in the more highly parallel sections of the emulator activity; therefore, the allocated parallelism, which measures the amount of parallel microprocessor activity, increases in relation to the effective parallelism, which measures the amount of parallel microprocessor activity normalized to the average execution time of a microinstruction.

The increase in program interference can be explained in a similar way, except in this case, the resource causing interference is a microprocess: in a computation involving closely-coupled interaction patterns, the more parallel activity leads to a greater likelihood that a microprocess will attempt to communicate with another microprocess which is busy; thus, there will be an increase in busy-waiting time caused by repeatedly attempting to consummate a communication. This busy-waiting time is reflected as an increase in the number of microinstructions executed, which causes the increasing difference between the throughput and effective parallelism curves.

The second observation which can be made from Figure 6 is that the difference between the allocated and effective parallelism above 16 microprocessors decreases rather than increases or remains constant, as would be expected. This anomaly occurs because the difference between the allocated and effective parallelism, pictured as curve 1 in Figure 7, not only is a measure of hardware interference but also measures the hardware overhead functions involved in mapping virtual PMS activity into physical PMS activity. The major components of this hardware overhead are the time to context switch
Figure 6. The Effect of Hardware and Program Interference on Sub-Eighth-Power Graph Program (10)

\[ TA(n) = \frac{(\text{Time to run with } n \text{ microprocessors}) \times \text{(Average allocated parallel activity)}}{2} \]

1. (Average instruction execution time for single microprocessor) / (Average instruction execution time for n microprocessors)
2. (Time required to fetch data on cache miss for n microprocessors) / TA(n)
3. (Time caused by PSM interference for n microprocessors) / TA(n)
4. (Time caused by delay in accessing locked MSV for n microprocessors) / TA(n)
5. (Time required to context switch for n microprocessors) / TA(n)

Figure 7. Distribution of Hardware Interference for Sub-Eighth-Power Graph Program (10)

Figure 8. Effect of Varying the Number of Microprocessors on the Number of Context Switches, Interprocessor Communication, and Microprocessors Queued for Sub-Eighth-Power Graph Program (10)
and the time to fetch data for a miss in the cache. As seen from Figure 7, these two components of hardware overhead (see Curves 2 and 4) decrease above 16 microprocessors, counter-balancing the effect of increasing hardware interference (see Curves 3 and 5). The proportional decrease in time spent doing hardware overhead functions can be explained in terms of the "control working set" phenomenon previously discussed.

The phenomenon of the control working set is graphically displayed in Figure 8: above 16 microprocessors there is a sharp decrease in the number of context switches, the number of microprocessors queued, and a correspondingly sharp increase in the number of direct activations over the interprocessor bussing structure. The decrease in the time spent on hardware overhead functions is thus easily explained:

1) The decrease in the time spent on context switching occurs because there are significantly fewer context switches.

2) The decrease in time spent on fetching data for a microprocessor's cache occurs because there is a much higher probability that a microprocess can remain connected to a microprocessor while waiting for a communication. The data working set of microprocessors will thus have to be reassembled many fewer times. This phenomenon is substantiated on another level through looking at the cache miss ratio which significantly decreases with more than 16 microprocessors, i.e., 13 percent to 10 percent.

This control working set phenomenon, characterized by a sharp decrease in hardware overhead above a certain number of microprocessors, is directly analogous to the data working set phenomenon of thrashing which occurs when there are too few physical data pages to hold the data working set of the program.

In addition, the following are some tentative conclusions, based on a small set of samples, about using a cache in a multiple microprocessor:

1) The cache policy of never storing-through unless necessary seems to be optimal. The mode of only storing-through when there are available bus and memory cycles is surprisingly not the best. The explanation seems to be that though the memory is available, the storing into memory will tie up the memory for cycles in the future, thus causing interference for future accesses. However, if the storing-through is restricted to some fraction of the available bus cycles, then this policy approaches (or could be possibly be better than) the never store-through mode.

2) A cache of 128 32-bit words configured as 32x4 seems appropriate for handling microprograms of short duration with small data working sets (i.e. a cache hit ratio of 85-90%). A cache configuration of 64x2 has a significantly lower cache hit ratio.

IV. Summary

This paper presents the architecture of a multiple microprocessor that can be dynamically reconfigured such that its structure is appropriate for the particular parallel machine language to be emulated. This capability for reconfiguring the microprocessor system is accomplished by introducing a new global control level into the microcomputer.
architecture. This new level, called the virtual PMS (Processor-Memory-Switch) environment, permits the emulator designer to dynamically vary the number and function of the working registers of each microprocessor, the structure of main memory (its size and word length), and most importantly the interconnection and interaction patterns among microprocessors.

The remainder of the paper details a logical hardware organization for this architecture that contains as an integral part of its design the concept of a virtual PMS environment. The examination of this organization has indicated that a coherent and simple design can be constructed which allows parallel activity on the virtual PMS level to be mapped correctly and without significant overhead into parallel activity in the physical PMS level. The main design techniques applied were:

1) A built-in hardware scheduler for allocating microprocesses to microprocessors. This scheduler contains a FIFO queue mechanism for ordering access to a locked-out microprocess state vector so as to guarantee that no resource deadlock will be introduced in the mapping of virtual activity to physical activity.

2) An Interprocessor Bus and its associated control which works like a common data bus so as to map microprocess interaction patterns into microprocessor interaction patterns whenever possible.

3) A memory cache per microprocessor which allows the internal registers of the processor to be configured so as to match the particular microprocess being executed and to reduce the memory interference among microprocessors and the time to context switch.

Finally, the dynamic aspects of this logical organization have been studied through a detailed simulation. The emphasis of this study has been on how the overhead of mapping virtual PMS activity to physical PMS activity varies as a function of the number of microprocessors. This study has shown in detail the phenomenon of the "control working set" which is characterized by a sharp decrease in hardware overhead above a certain number of microprocessors, counter-balancing the effect of increased inference caused by more accessing of shared physical resources.

BIBLIOGRAPHY


