THE ORGANIZATION OF THE PASM RECONFIGURABLE PARALLEL PROCESSING SYSTEM

Howard Jay Siegel
hj@ecn.purdue.edu

Wayne G. Nation
wn@ecn.purdue.edu

Mark D. Allemang
allemang@ecn.purdue.edu

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

Abstract -- PASM is an architectural concept for a research tool for studying the design and use of reconfigurable large-scale parallel processing systems. The PASM organization will support over 1000 sophisticated processors and a small-scale 30-processor prototype has been built at Purdue. PASM can be partitioned into independent or communicating submachines of various sizes that can change dynamically. Two modes of parallelism are supported in a PASM system: the synchronous single instruction stream – multiple data stream (SIMD) mode and the asynchronous multiple instruction stream – multiple data stream (MIMD) mode. Each submachine in PASM can independently switch between the SIMD and MIMD modes at instruction level granularity ("mixed-mode" parallelism). The processing elements in PASM are interconnected with a flexible multistage cube network that allows connection patterns to be changed. Thus, PASM is dynamically reconfigurable along three dimensions: partitionability, modes of parallelism, and connections among processing elements. The PASM prototype hardware provides all three dimensions of reconfigurability and is currently supporting active experimentation. The PASM design and prototype are overviewed. A companion paper discusses the mapping of tasks onto the PASM system.

1. Introduction

Parallelism is one way to achieve the computational "need for speed" for tasks like image and speech understanding, where the exploitation of data parallelism and/or control parallelism is possible. Complex tasks that need parallel processing to meet their execution time constraints often consist of numerous subtasks, with various computational characteristics. It is generally difficult for a single system to be an optimal match for these different computational needs [Fre89]. One way to attempt to maximize performance in such situations is by using a reconfigurable system, such as PASM [SiS87]. This is an overview of the PASM parallel processing system architectural organization and the small-scale prototype constructed to validate design concepts and to serve as a tool for studying issues related to the use of reconfigurable parallel machines.

This work supported by the Naval Ocean Systems Center under the High Performance Computing Block, ONT, and by the Office of Naval Research under grant number N00014-90-J-1483.

Consider a system consisting of N processors, N memories, an interconnection network, and a control unit. In the *SIMD (single instruction stream – multiple data stream)* [Fly66] mode of parallelism, all active processors execute the same instructions, but each on its own data. The control unit broadcasts instructions in sequence to the N processors which execute these instructions in lockstep. The operand data for these instructions is fetched from the memory associated with each processor. The interconnection network provides inter-processor communication. Examples of SIMD systems that have been constructed are ASPRO [Bat82], CLIP4 [Fou81], DAP [Hun89], Illiac IV [BoD72], MasPar [Bla90], MPP [Bat80], and STARAN [Bat76, Bat77].

An *MSIMD (multiple-SIMD) system* can be structured as one or more independent SIMD machines of various sizes. The proposed MAP [Nut77] and existing Connection Machine 2 [TuR88] are examples of MSIMD systems. The Illiac IV was originally designed as an MSIMD system [BaB68]. A formal model of MSIMD machines is presented in [RiS88].

In the *MIMD (multiple instruction stream – multiple data stream)* [Fly66] mode of parallelism, each processor has its own instructions and data. MIMD systems are typically structured like SIMD systems with the exception of the control unit, i.e., N processors, N memories, an interconnection network, and multiple data streams. The BBN Butterfly [CrG85], Cm* [SwF77], Cosmic Cube [Sei85], iPSC cube [Nug88], NCube [HaM89], and IBM RP3 [PfB85], are examples of MIMD systems that have been constructed.

The *SPMD (single-program-multiple data)* mode of parallelism is a type of MIMD mode where the processors are all restricted to executing the same program (asynchronously with respect to one another). OPSILA [DuB88] is an existing system capable of switching between the SIMD and SPMD modes of parallelism.

A *partitionable-SIMD/MIMD system* can dynamically reconfigure to form independent or communicating submachines of various sizes, where each submachine can operate in the SIMD or MIMD mode of parallelism (e.g., TRAC [SeU80]). PASM is a partitionable-SIMD/MIMD system concept being developed at Purdue University as a design for a large-scale dynamically reconfigurable multimicroprocessor system [SiS81, SiS87]. Each submachine can independently switch between the SIMD and MIMD

NOTICE
THIS MATERIAL MAY BE PROTECTED BY
COPYRIGHT LAW (TITLE 17 U.S. CODE)

modes of parallelism at instruction-level granularity with negligible overhead (*mixed-mode* parallelism). PASM uses a flexible multistage interconnection network for interprocessor communication. Thus, PASM is dynamically reconfigurable along the three dimensions of partitionability, modes of parallelism, and connections among processors.

PASM was originally intended as a special-purpose system aimed at image understanding tasks and it was the algorithmic and computational characteristics of these tasks that guided design decisions. Furthermore, originally mode switching was considered only at the subtask level. Experience with the prototype has shown that the PASM reconfiguration structure is effective for a great variety of application domains and that the capability to perform mode switching at the instruction level rather than subtask level can be a powerful tool.

The goal of the PASM research team is to develop a unique dynamically reconfigurable research tool for studying large-scale SIMD, MIMD, and mixed-mode processing. The PASM concept can support 1024 *processing elements (PEs)*, processor/memory pairs, and attempts to incorporate the flexibility needed to investigate the many issues related to the design and use of reconfigurable parallel processing systems. The small-scale prototype built in-house at Purdue University (30 processors, 16 in the computational engine) allows actual experimentation with reconfigurable systems, including mixed-mode operation, and has produced insights not attainable from earlier simulations and theoretical studies [e.g., BrC90, FiC88, FiC91, SiA90]. In addition to allowing a system to adapt its architectural configuration to provide a better match to the computational structures to be executed, reconfigurability can also be exploited to facilitate fault tolerance.

This paper overviews the PASM architectural concepts (Section 2) and describes the prototype that has been built (Section 3). A companion paper addresses the mapping of a variety of tasks onto PASM [SiA90].

2. The PASM Architecture

This section describes the components of the PASM system architecture design concepts. Section 3 details how these components are implemented in the PASM prototype. As shown in Figure 2.1, the PASM system is comprised of six basic components. The *System Control Unit* is the overall system coordinator and is the part of PASM with which the user directly interacts. The *Micro Controllers (MCs)* serve as the PE control units in SIMD mode and may coordinate the PEs in MIMD mode. The *Parallel Computation Unit* contains the $N = 2^n$ PEs, physically addressed 0 to $N - 1$, and the interconnection network used by the PEs. The *Memory Storage System* provides secondary storage for the PEs; it is used to store data files for SIMD mode and both program and data files for MIMD mode. The *Memory Management System* controls the transferring of files between the Memory Storage System and the PEs. *Control Storage* is the secondary storage for the MCs and the System Control Unit.

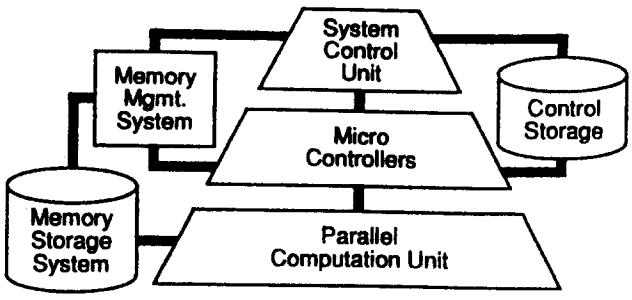

Figure 2.1: Block diagram of the PASM system.

2.1. System Control Unit

The tasks performed by the System Control Unit include support for program development, job scheduling, general system coordination, management of system configuration and partitioning, and the assignment of user jobs to submachines. The hardware needed to combine and synchronize the MCs and PEs to form SIMD submachines of various sizes resides in the System Control Unit. It is also responsible for coordinating the loading of PE memories from the Memory Storage System with the loading of MC memories from the Control Storage.

Appropriate distribution of tasks between the System Control Unit and the other system components (e.g., the MCs and the Memory Management System) will keep the System Control Unit from becoming a system bottleneck. In an $N = 1024$ system, the System Control Unit will likely be comprised of several processors. In the $N = 16$ prototype, the System Control Unit is a microprocessor, and program development is performed on the host computer network.

2.2. Micro Controllers

The *Micro Controllers (MCs)*, shown in Figure 2.2, are the multiple control units required to have an MSIMD system. There are $Q = 2^q$ MCs, physically addressed from 0 to $Q - 1$. Each MC controls a fixed group of N/Q of the PEs in the Parallel Computation Unit. An MC and its associated PEs is an *MC group*. The physical addresses of all N/Q PEs connected to an MC have the same low-order q bits. The value of these low-order q bits is the physical address of the MC. In an $N = 1024$ system, Q may be 32; for the $N = 16$ prototype, $Q = 4$.

Two memory units are provided in each MC so that computation and memory I/O can be overlapped; e.g., the MC processor can execute a job in one memory unit while the next job is preloaded into the other memory unit from MC secondary storage (the Control Storage). In MIMD mode, an MC fetches from its memory instructions and data used to coordinate the operation of its PEs. In SIMD mode, an MC fetches instructions and common PE data from its memory units; in general, control-flow instructions are executed in the MC, and data processing instructions are broadcast to the MC's group of PEs.

Submachines are formed by combining one or more MC groups. A submachine containing $R = 2^r$ MC groups (RN/Q PEs), where $0 \leq r \leq q$, is formed by combining the PEs connected to R MCs whose addresses

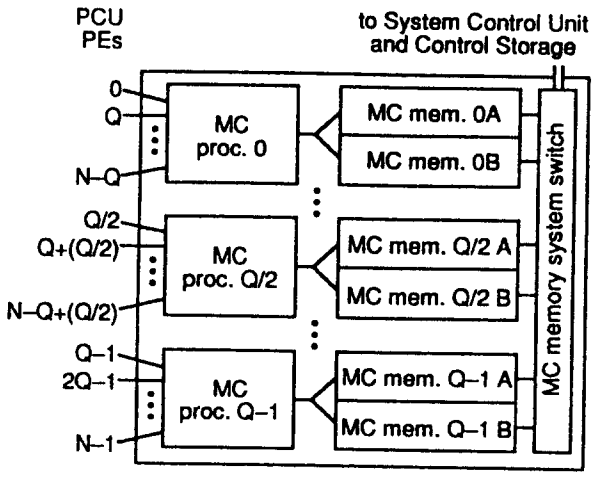

Figure 2.2: PASM Micro Controllers (MCs). "PCU" is the Parallel Computation Unit.

agree in their q − r low-order bits. The reconfiguration rule in PASM is that the addresses of all PEs in a submachine of size 2^p must agree in their n − p low-order bit positions. Thus, the addresses of all PEs in the submachine will agree in their q − r low-order bits. Consequently, there are a maximum of Q submachines of size N/Q each.

In an SIMD submachine, the R MCs must use the same instructions and all PEs in the submachine must be synchronized. The MCs can be given the same instructions simply by loading the memory units of the R MCs with the same program (a shared memory alternative is discussed in [SiS81]). The PEs can be synchronized by providing special simple circuitry in the System Control Unit to synchronize the MC instruction broadcasts.

The MCs within a submachine of RN/Q PEs are assigned *logical addresses*. The MCs are logically numbered (addressed) 0 to R − 1. For R > 1, the logical number of an MC is the high-order r bits of its physical number. Similarly, the PEs assigned to a submachine are logically numbered (addressed) 0 to (RN/Q) − 1, $R = 2^r$, $0 \leq r \leq q$. The logical number of a PE is the high-order r + n − q bits of its physical number.

Some advantages of this static PE to MC mapping as compared to a dynamic MC/PE interconnection (e.g., a crossbar switch [Nut77]) include: simpler (and less expensive) MC/PE interface hardware, the overhead of maintaining PE to MC assignments is eliminated, only Q MCs need to be scheduled instead of N PEs, the PE interconnection network may be partitioned into independent subnetworks (see Section 2.3), and efficient parallel primary to secondary memory connections may be designed (see Section 2.5). The main disadvantage of this approach is that the size of each submachine must be a power of two, with a minimum value of N/Q. However, for PASM's intended experimental environment, flexibility at "reasonable" cost is the goal, not maximum PE utilization.

Masking schemes are used in SIMD mode to enable and disable PEs. The PASM system uses PE-address masks and data conditional masks. A *PE-address mask* [Sie77] is used to enable a set of PEs based solely on PE addresses; thus, the decision to enable a PE does not depend on local PE data. A PE-address mask has n positions where each position contains either a 0, 1, or X ("don't care"). A PE is enabled only if the binary representation of its address matches the mask; e.g., for N = 64, [5{X}{0}] is equivalent to [XXXXX0] and enables all even-numbered PEs. A *negative PE-address mask* enables all PEs whose addresses do not match the mask.

Data conditional masks enable PEs based on some condition dependent on local PE data. Consequently, the resulting data condition may be true in some PEs and false in other PEs. An example use of data conditional masking is the *where statement*, the SIMD counterpart to the "if-then-else" statement. When

where <data-condition> *do* <where-part>
elsewhere <else-part>

is executed, each PE evaluates the <data-condition>. The PEs where the condition evaluated true then execute the <where-part> (the PEs where the condition evaluated false are idle). Next, the PEs where the condition evaluated false execute the <else-part> (the PEs where the condition evaluated true are idle). This type of masking is used in many SIMD machines (e.g., Connection Machine, DAP, MasPar). "Where" statements may be nested using a run-time control stack [ClS85].

Data conditional masking may be supported in hardware by providing each PE with a *local enable bit*, which is used to indicate whether the PE is conditionally enabled or disabled. PE-address masks may be supported in hardware by providing each MC with a *mask register* that contains a single *mask mode bit* that indicates whether the PEs' local enable bits should be used when determining PE enable status, and an N/Q-bit wide PE-address *mask vector* that is a decoded version of the PE-address mask [SiS81]. If bit i of the mask vector is one then the i-th PE of the MC group is enabled, and if bit i is zero then the i-th PE of the MC group is disabled. The mask mode bit and bit i of the mask vector are sent to PE i with each instruction. PE hardware can then combine its bit of the mask vector with the local enable bit according to the mask mode to decide if the PE is enabled. If the mask mode indicates unconditional masking, then the PE is enabled if its address matches the PE-address mask. If the mask mode indicates conditional masking, then the PE is enabled if its address matches the PE-address mask and its local enable bit is set to true. Such "hybrid" masking is examined in [NaF90]. Further information on PE-address masks and data conditional masks are found in [ClS85, NaF90].

There are situations in which the combined conditional status of all active PEs in an SIMD submachine is needed. For example, if each PE in the submachine is processing a section of an image looking for rain clouds, it is useful to know if any of the PEs have found anything yet. For instance, "if-none" has found a rain cloud then keep looking, "if-any" have found a rain cloud then report, and "if-all" have found a rain cloud then panic. Examples of machines that support operations of this type include MPP with its SUM-OR tree [Bat80, Bat82], ILLIAC IV with its ability for the control unit to read the enable state of all PEs [BoD72], and the Connection Machine with its GLOBAL line [Hil85]. Because PASM is a partitionable system, operations such as these require

conditional results to be communicated from the PEs to their MCs and then combined among the MCs comprising an SIMD submachine. These operations may be efficiently supported by providing simple special circuitry that combines MC group results according to the current system partitioning.

2.3. Parallel Computation Unit

The *Parallel Computation Unit*, shown in Figure 2.3, contains $N = 2^n$ PEs and an interconnection network. Two memory units are used in each PE so that computation and memory I/O can be overlapped; e.g., the PE processor can execute a job in one memory unit while the next job is preloaded into the other memory unit from PE secondary storage (the Memory Storage System). These memory units compose the primary memory of the system. In SIMD or MIMD mode, each PE can use its own PE address or local data as a basis for indirect addressing of its memory. The PE processors may be either standard microprocessors or custom designed for parallel processing and/or a particular application domain.

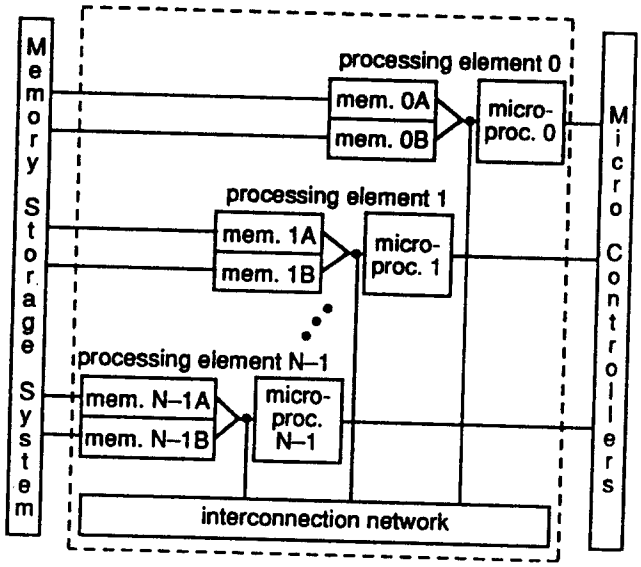

Figure 2.3: The Parallel Computation Unit.

The interconnection network allows the PEs to communicate with each other. Because PASM is a partitionable system, the interconnection network used must be a partitionable network. As described earlier, the reconfiguration rule in PASM requires that the physical addresses of all PEs in a submachine of 2^p PEs agree in their $n - p$ low-order bit positions. Thus, the p high-order bits of a PE's physical number form its logical number within a submachine of 2^p PEs. The low-order partitioning rule was chosen for PASM so that either the multistage cube network [Sie90] or the augmented data manipulator (ADM) network [Sie90] can be used in the system. The multistage cube network will be discussed to introduce many of the concepts and properties of multistage interconnection networks.

The *multistage cube* network [SiN89] is representative of the class of networks that includes the baseline [WuF80], delta [Pat81], generalized cube [Sie90], indirect binary n-cube [Pea77], multistage shuffle-exchange [ThN81], omega [Law75], and SW-banyan ($S = F = 2$) [GoL73] networks. This class of networks has been used in or proposed for use in many systems including BBN Butterfly [CrG85], Cedar [KuD86], dataflow machines [DeB80], RP3 [PfB85], STARAN [Bat76, Bat77], and Ultracomputer [GoG83].

The multistage cube network has N input ports, N output ports, and contains $n = \log_2 N$ stages of $N/2$ 2×2 interchange boxes per stage. Figure 2.4 shows the multistage cube network for $N = 8$. In the PE-to-PE configuration which is used in PASM, PE i is connected to network input port i and output port i. The stages are numbered consecutively from $n - 1$ at the input stage to 0 at the output stage. The interconnection pattern between stages is such that link labels that differ only in bit i are connected to the same interchange box. Each interchange box can be set to one of the four states shown in Figure 2.4. Figures 2.5, 2.6, and 2.7 show sample one-to-one (input 3 → output 5), broadcast (input 2 → outputs {4,5,6,7}), and permutation (input i → output i+1 mod 8) connections for an $N = 8$ multistage cube network.

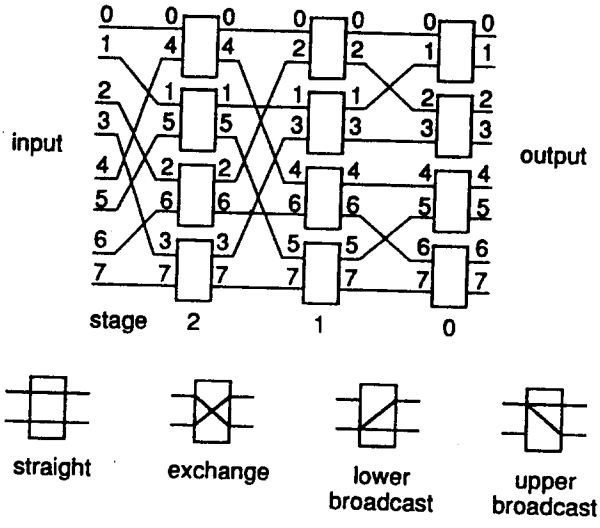

Figure 2.4: The multistage cube topology for $N = 8$, and valid interchange box states.

One advantage of the multistage cube network is that network control may be distributed — each network input device determines its own routing tag that is used as a header on messages. A simple way is to set the message header tag to the n-bit address of the message destination, [Law75]. Each interchange box encountered by the message examines the destination routing tag to determine how it should set itself. If a message is destined for output port $D = d_{n-1} \cdots d_1 d_0$, then at stage i it is routed to the lower box output if d_i is 1 and to the upper box output if d_i is 0. Several benefits result from this routing method: it is distributed, the routing tag can be used to verify that the message arrived at the correct destination, and tag generation is trivial. The destination tags described can specify one-to-one and permutation connections, and can be extended to handle broadcast connections by adding an n-bit broadcast mask to the tag

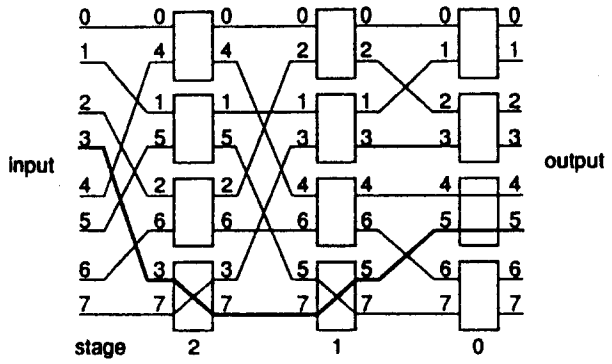

Figure 2.5: Example one-to-one connection for N = 8 for input 3 to output 5.

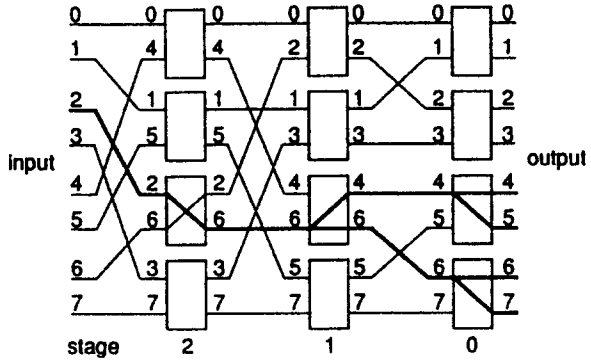

Figure 2.6: Example broadcast connection for N = 8 for input 2 to outputs 4, 5, 6, and 7.

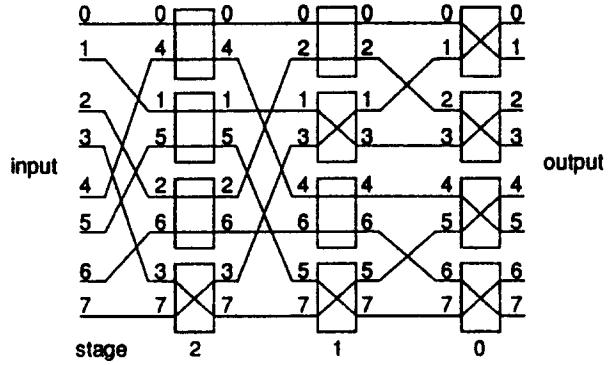

Figure 2.7: Example permutation connection for N = 8 for input i to output i+1 mod 8.

[Sie90].

A network is partitionable if it can be divided into independent subnetworks of smaller sizes that have all of the properties of the original network [Sie80, Sie90]. The multistage cube is a partitionable network. A multistage cube network of size N can be divided into two subnetworks of size N/2, each with the properties of a size N/2 multistage cube network. Because each subnetwork has all the properties of a multistage cube network, they can be further independently subdivided into networks of size N/4, N/8, etc. Thus, in general, a size N multistage cube network can be partitioned into multiple subnetworks of varying sizes where the size of each subnetwork is a power of two.

For example, consider Figure 2.8 that shows an N = 16 network partitioned into two subnetworks by setting stage 0 to straight. Because stage 0 is straight, even inputs cannot reach odd outputs, and odd inputs cannot reach even outputs. The two resulting subnetworks are subnetwork A that contains physical ports 0, 2, ..., 14 and subnetwork B that contains physical ports 1, 3, ..., 15. Because each of these subnetworks is an independent multistage cube network of size N/2 = 8, either or both subnetworks may be further partitioned by forcing all interchange boxes in stage one of the subnetwork to straight. Figure 2.9 shows group B of the Figure 2.8 partitioning further divided by forcing stage 1 to straight to form two new size four subnetworks: C (physical ports 1, 5, 9, and 13) and D (physical ports 3, 7, 11, and 15).

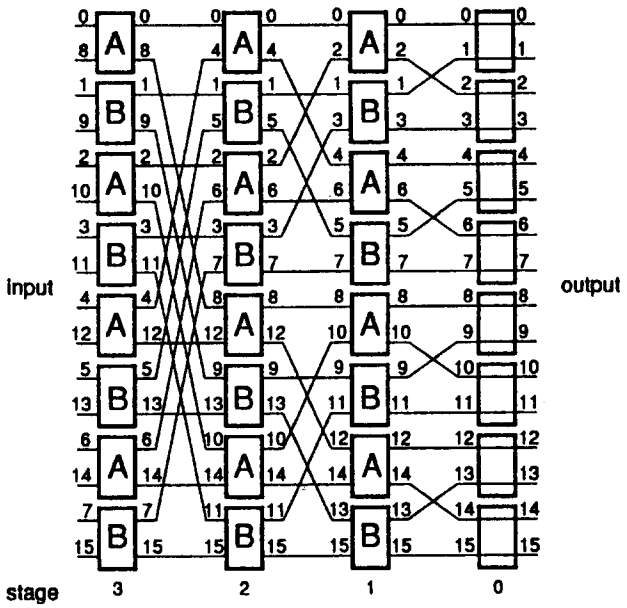

Figure 2.8: N = 16 multistage cube network network partitioned into two size eight subnetworks.

The operating system can use routing tags to enforce multistage cube network partitioning as follows. Consider a partition of 2^k PEs. Let the partition mask M be formed, where bits 0 to n−k−1 of M are 0 and bits n−k to n−1 are 1. When source PE $S = s_{n-1}...s_1s_0$ sends data to destination PE $D = d_{n-1}...d_1d_0$, the effective routing tag $E = D \cdot M + S \cdot \overline{M} = d_{n-1}...d_{n-k+1}d_{n-k}s_{n-k-1}...s_1s_0$ is used. This forces stages 0 to n−k−1 to straight. For example, consider subnetwork C of the Figure 2.9 partitioning. The partition mask is M = 1100, and the effective tag is $E = d_3d_2s_1s_0 = d_3d_201$. Using E as the routing tag for all communications in the subnetwork will cause the partitioning to be enforced.

In summary, the multistage cube network's main advantages include its ability to perform up to N simultaneous transfers, to be partitioned into multiple independent subnetworks of various sizes, and to perform one-to-

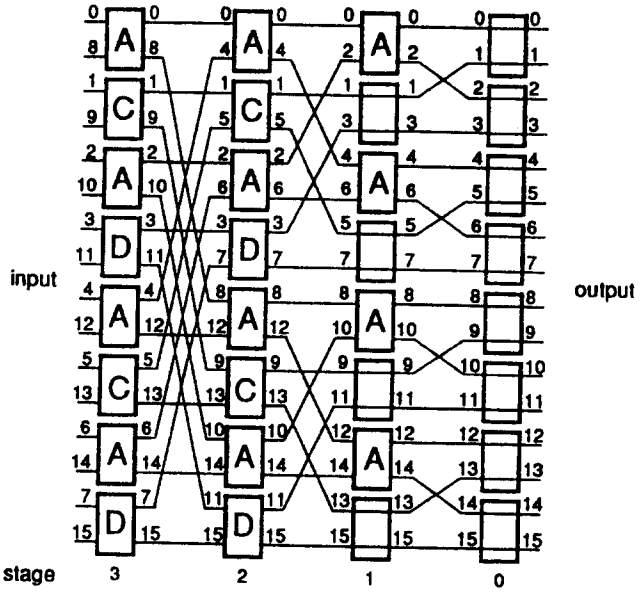

Figure 2.9: N = 16 multistage cube network partitioned into subnetworks of sizes eight, four, and four.

one, permutation, and broadcast transfers using routing tags. It can support efficient global as well as local (nearest neighbors) inter-PE communications. Finally, there are a variety of network implementation options (e.g., circuit switched vs. packet switched, 2×2 vs. 4×4 vs. 8×8 interchange boxes, etc.), and the network may be used in both the SIMD and MIMD modes of parallelism.

The multistage cube network as presented here has the disadvantage that only a single path exists for any source-destination pair. Thus, a single fault in the network makes many source-destination paths unavailable. The Extra Stage Cube network [AdS82] is a single-fault tolerant variation of the multistage cube network. Figure 2.10 shows an Extra Stage Cube for N = 8 ports. PE i connects to both input port i links and both output port i links so a single failure cannot deny PE i access to the network. The input (extra) stage pairs links that differ in bit positions 0, as does stage 0. The input and output stages can be bypassed as shown in Figure 2.10. In normal fault-free operation, the extra stage is bypassed (i.e., the bypass circuitry is used), and the network operates exactly as the multistage cube network. If a fault is present in a box in the extra stage, the bypass circuitry is employed as in the no-fault case and the network is again equivalent to the multistage cube network. If a fault is present in a box in an intermediate stage (stages 1 through $n - 1$) or in any internal network link, then both stage 0 and the extra stage are enabled. This establishes two paths for every source-destination pair, and these pairs never share the same link or interchange box except at the input and output stages. In the presence of a single fault, one path is guaranteed to be fault-free for any source-destination pair. If a fault occurs in a box in stage 0, the extra stage is enabled and stage 0 is bypassed. As described in [AdS82] and [Sie90], the Extra Stage Cube network is partitionable and may be controlled using

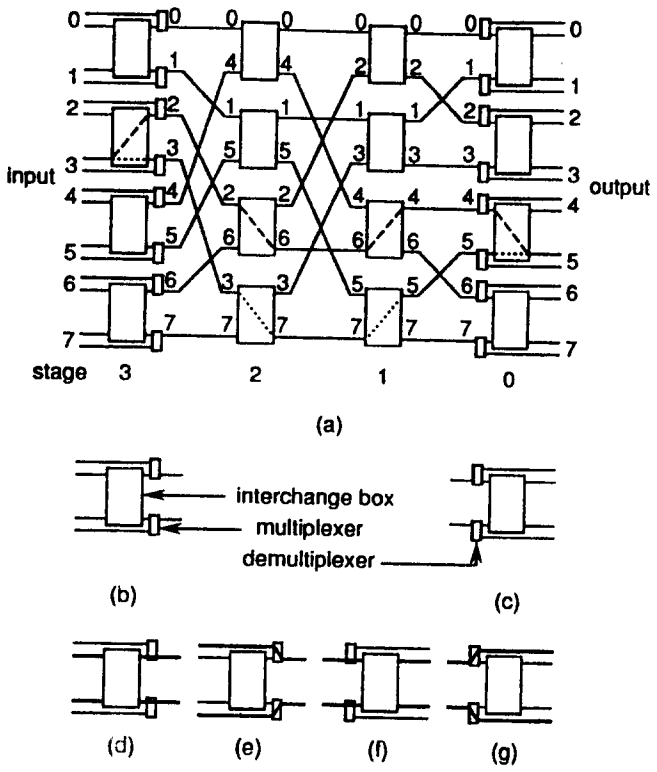

Figure 2.10: (a) The Extra Stage Cube network topology, shown for N = 8. (b) Detail of input interchange box. (c) Detail of output interchange box. (d) Input stage enabled. (e) Input stage disabled. (f) Output stage enabled. (g) Output stage disabled.

routing tags.

2.4. Memory Storage System

The *Memory Storage System* is the secondary storage for the Parallel Computation Unit, storing data files in SIMD mode and both program and data files in MIMD mode. The Memory Storage System is comprised of N/Q independent *Memory Storage Units (MSUs)*, numbered from 0 to (N/Q) − 1. Each Memory Storage Unit contains a mass storage unit and a processor to manage the file system and to transfer files to and from its associated PE memory units.

Each of the N/Q MSUs is connected to the memory modules of Q PEs. MSU i is connected to and stores files for the Q PEs whose n − q high-order address bits are equal to i. This high-order mapping is used so that each of the N/Q PEs connected to an MC is connected to a different MSU. Recall that the PE to MC mapping is a low-order mapping. An example for N = 16 and Q = 4 is shown in Figure 2.11.

One benefit of this MSU to PE configuration is that the memories of the N/Q PEs connected to any one MC can be loaded or unloaded in one parallel block transfer. For example, in Figure 2.11, MC 2's group of PEs (PEs 2, 6, 10, and 14) are loaded by having MSU 0 load PE 2, MSU 1 load PE 6, MSU 2 load PE 10, and MSU 3 load PE 14. All four of these transfers may be done in parallel so that only one parallel block transfer is

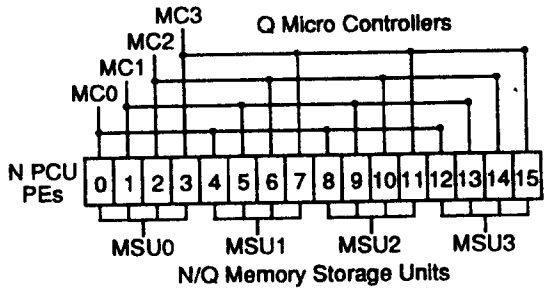

Figure 2.11: Organization of the Memory Storage System ($N = 16$ and $Q = 4$). "PCU" is Parallel Computation Unit.

required to load the PEs connected to any one MC. If there are only $N/(QD)$ distinct MSUs, where $1 \leq D \leq N/Q$, then this scheme can be scaled so D parallel block transfers are required to load or unload the memories of the PEs connected to any one MC.

Similarly, a submachine formed by combining the RN/Q PEs controlled by R MCs can be loaded or unloaded in R parallel block transfers if there are N/Q MSUs and RD parallel block transfers if there are $N/(QD)$ distinct MSUs. For example, in Figure 2.11, a submachine comprised of the PEs of MC 0 and MC 2 can be loaded in $R = 2$ parallel block loads as follows: first MC 0's PEs are loaded in one parallel block transfer as described above, then MC 2's PEs are loaded. Thus, the full bandwidth of the Memory Storage System is always used when transferring files between the Memory Storage System and the Parallel Computation Unit.

Data for logical PE i in a submachine is stored in the MSU whose number is the PE's n—q high-order logical address bits (which equal its n—q high-order physical address bits). As a result, no matter which MCs are assigned to a task, the data will be in the appropriate MSUs. For the example in Figure 2.13, for any submachine of size $N/Q = 4$, MSU 0 is connected to logical PE 0 (which could be physical PE 0, 1, 2, or 3).

2.5. Memory Management System

The *Memory Management System* supervises file transfers between the N PE memory modules in the Parallel Computation Unit and the N/Q secondary storage devices in the Memory Storage System. As described in the discussion of the Memory Storage System, multiple PEs are connected to a single Memory Storage Unit through a switch. The Memory Management System controls this switch to determine which of the PEs connected to a MSU will be the source or destination for file transfers.

The Memory Storage System is composed of multiple processors that perform the tasks required in a distributed manner. The main functions that must be performed include (1) passing file system requests from the System Control Unit, MCs, and PEs to the appropriate MSUs; (2) controlling the data transfers between the Memory Storage System and the PEs; (3) controlling input/output operations involving peripheral devices; and (4) enforcing consistent file naming and placement across the multiple Memory Storage System disks.

2.6. Control Storage

Control Storage is the mass storage for the MCs and the System Control Unit. It consists of a secondary storage device and a processor for managing the file system on the device. The Control Storage processor acts as a file server by responding to requests from the MCs and the System Control Unit.

2.7. Advantages of Reconfigurability

Numerous advantages derive from PASM's reconfigurability along the three dimensions of partitionability, modes of parallelism, and variable connectivity among PEs. These include:

1. *Multiple Simultaneous Users* — Because there can be multiple simultaneous independent submachines, there can be multiple simultaneous users of the system, each executing a different program.
2. *Program Development* — Rather than trying to debug a new parallel program on, for example, 1024 PEs, it can be debugged on a smaller size submachine of 32 PEs, and then extended to 1024 PEs.
3. *Variable Submachine Size for Increased Utilization* — If a task requires only N′ of N available PEs, the other $N - N'$ can be used for another task.
4. *Variable Submachine Size for Decreased Execution Time* — There are some algorithms where the minimum execution time is obtained when less than N PEs are used; thus, it is desirable to create a submachine consisting of the optimal number of PEs.
5. *Subtask Parallelism* — Two independent subtasks that are part of the same job can be executed in parallel, sharing results if necessary, which may result in improved overall task execution time.
6. *Multiple Processing Modes* — An algorithm can be executed by using a combination of SIMD and MIMD control with the same set of PEs (mixed-mode parallelism).
7. *Matching Inter-PE Connectivity to the Task* — The multistage cube allows different connection patterns among PEs to be established depending on the task (as opposed to, for example, having a fixed mesh network).
8. *System Fault Tolerance* — If a single PE fails, only those submachines that include the failed PE are affected.
9. *Submachine Fault Tolerance* — If a PE in a submachine fails, it may be possible to redistribute data and make use of mixed-mode parallelism (e.g., changing from SIMD to MIMD mode) and the variable connectivity (e.g., to establish connection patterns that do not include the faulty PE) so that the job executing on the submachine may continue on that submachine with negligible degradation.

Examples of varying machine size for utilization and speed, exploiting subtask parallelism, using mixed-mode parallelism, and establishing different types of inter-PE connection patterns are given in [SiA90].

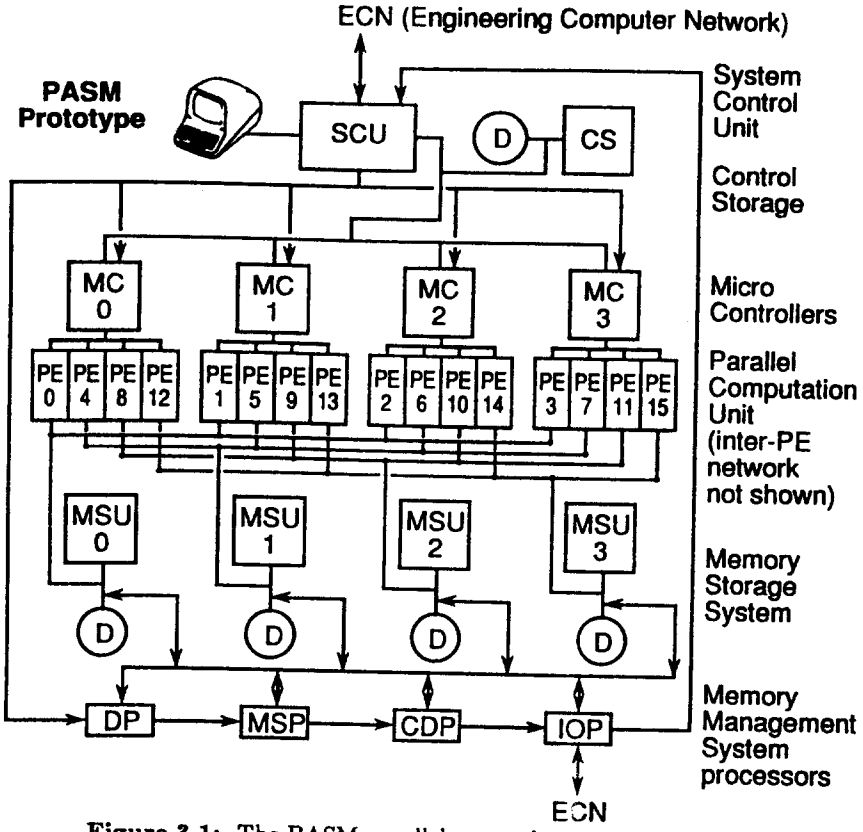

Figure 3.1: The PASM parallel processing system prototype.

3. PASM Prototype Overview

This section is a brief overview of the implementation of the PASM prototype built at Purdue University. Figure 3.1 shows a block diagram of the PASM prototype with $N = 16$ and $Q = 4$. Each rectangle is a processor and is based on the Motorola MC68000 family microprocessor. Detailed descriptions of the interactions of the different components of the PASM prototype can be found in [ScN87]. Table 1 summarizes the PASM design parameters.

3.1. Prototype System Control Unit and Secondary Storage System

The System Control Unit is an MC68010 microprocessor running UNIX and has an Ethernet connection to the Purdue Engineering Computer Network (ECN), a local network connecting several hundred micro-, mini-, and supermini-computers and workstations. A PASM user logs into the System Control Unit and controls the prototype by issuing commands from the System Control Unit to initialize the system and load and execute programs. The System Control Unit communicates with the Control Storage and Micro Controllers through dedicated parallel port connections.

Mass storage for the MCs and PEs is provided by the Control Storage and the Memory Storage Units, respectively. The Control Storage and the four Memory Storage Units each contain a Winchester technology disk drive. The Memory Storage Units are managed and controlled by the Memory Management System, consisting of a *Directory Lookup Processor (DP)*, a *Memory Scheduling Processor (MSP)*, a *Command Distribution Processor (CDP)*, and an *I/O Processor (IOP)*. The I/O Processor is responsible for handling interactions with external I/O devices and for distributing data to the Memory Storage Units and the Control Storage through highly efficient DMA links with their memories.

Previous papers have described the Control Storage as the mass storage for both the MCs and the System Control Unit. While such an organization is still being considered for a full 1024-PE PASM system, the use of a stand-alone computer system (i.e., a microcomputer or workstation) for the System Control Unit of the small-scale prototype is possible. Thus, the prototype System Control Unit contains its own mass storage device and the prototype's Control Storage is used only by the MCs.

3.2. Micro Controller Implementation

Each prototype MC is based on an MVME-110 single-board MC68000 microcomputer on a private VMEbus backplane with 2M bytes of dynamic RAM, double buffered as described earlier in Section 2.2. An MC is augmented with additional logic to support necessary interfaces to the System Control Unit, Memory Management System, other MCs, and its associated PEs. The System Control Unit, Memory Management System, and remote MC connections are implemented with simple parallel ports used for passing control information. The remaining special-purpose connections to the PEs support mixed-mode operation and are overviewed here. These connections include the mechanism to perform SIMD instruction broadcasts, the logic to allow the MC to access

	general	full PASM	PASM prototype
Number of PEs	N	1024	16
Number of network stages (Extra Stage Cube, if 2 × 2 boxes)	$\log_2 N + 1$	11	5
Number of MCs	Q	32	4
Number of PEs per MC	N/Q	32	4
Number of Memory Storage Units	N/Q	32	4
Number of Memory Management System processors	fixed	≥4	4
Smallest size partition	N/Q	32	4
Maximum number of partitions	Q	32	4

Table 1: The PASM design parameters, based on current plans.

PE conditional results, and a channel for the PEs to send data to their MCs.

In SIMD mode, the MC CPU fetches and executes SIMD control flow instructions from its memory (e.g., loop control, branch instructions). The MC *Fetch Unit* is the mechanism that allows the MCs to broadcast SIMD instructions to their associated PEs. Figure 3.2 shows the basic organization of an MC with the Fetch Unit organization emphasized. The MC CPU commands the Fetch

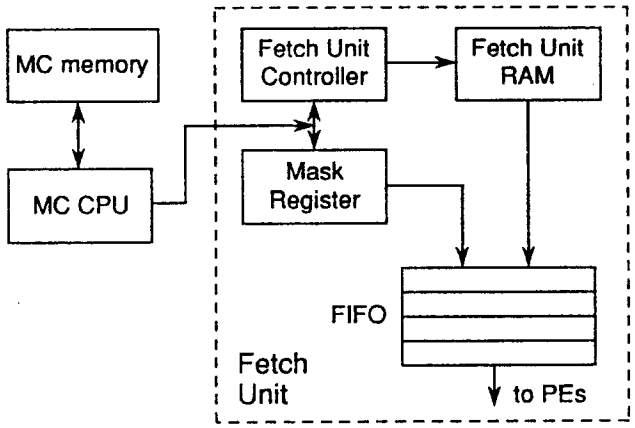

Figure 3.2: Simplified block diagram of an MC with emphasis on the Fetch Unit structure.

Unit to broadcast SIMD data processing instructions to the PEs. A 64-word FIFO instruction queue is provided in the Fetch Unit to decouple MC and PE execution. Other machines, such as the Illiac IV [BoD72] and the Connection Machine [TuR88], also use a FIFO to decouple control unit execution from PE execution.

In the prototype, the MC handles all masking. Information for data-conditional masking is sent to the MCs from the PEs through dedicated hardware [ScN87]. Masking is accomplished in the prototype by providing a *Mask Register* in the Fetch Unit containing an $N/Q = 4$-bit mask that is broadcast with each SIMD instruction. If the i-th bit of the mask is zero, then the i-th PE associated with that MC is disabled for the instruction. It is planned that the hybrid masking scheme described in Section 2.2 and [NaF90] will be incorporated into the prototype in the near future.

The final MC-PE control channel is an 8-bit IEEE 488-standard General Purpose Interface Bus (GPIB). This channel's main function is to allow PEs to communicate information to their MC: to relate PE error conditions (i.e., divide by zero) in SIMD mode, to pass PE data to MCs in both SIMD and MIMD mode, to allow the MC to coordinate its PEs in MIMD mode, etc.

3.3. Processing Element Implementation

Each PE consists of a MVME-110 MC68000 microcomputer connected via a VMEbus backplane to 2M bytes of dynamic RAM and logic to interface the PE with its MC and the prototype interconnection network. The dynamic RAM is physically double-buffered, as described in Section 2.3, with two 1M byte memory modules. The memory modules themselves are dual-ported with one port connected to the PE VMEbus and the other port connected to the appropriate MSU with a high-speed DMA link. PEs access the interconnection network through a set of parallel ports mapped in the I/O space of the PE's address space.

In MIMD mode, PEs fetch instructions and data from their local RAM. In SIMD mode, PEs still fetch data from their local RAM but SIMD instructions are received from the stream of SIMD instructions broadcast from the associated MC's Fetch Unit. A PE fetches SIMD instructions by reading an instruction word from the *SIMD instruction space* of the PE's memory. This is a logical address space — no physical memory is provided. Figure 3.3 illustrates the address space of a PE, showing the SIMD instruction space, physical RAM, and I/O space. Logic in the PE detects read accesses to the SIMD instruction space, and any such access is interpreted as an SIMD instruction request. In the PASM prototype, accesses to the 4M bytes of SIMD instruction space are determined by the values of the two high-order bits of the 24-bit program counter. The remaining 22 bits of the program counter value are ignored during SIMD instruction fetches. Because the program counter is still incremented after each SIMD instruction fetch (the PE CPU still views such accesses as "normal" memory reads), it is possible for the PE CPU program counter value to increment out of the SIMD instruction space during a very long SIMD program. This problem is avoided by having the MC periodically broadcast a "jump to start of SIMD space" instruction to the PEs. Because the SIMD instruction space is large (4M bytes), the frequency with which this is required is small. Furthermore, because a single MC68000 branch instruction is sufficient to perform this operation, the overhead incurred by the operation is negligible.

The *Instruction Broadcast Unit (IBU)* is the PE logic that monitors each memory access made by the PE CPU. When a read to the SIMD instruction space is detected, the IBU notifies the Fetch Unit of its MC that it is awaiting an SIMD instruction. For submachines con-

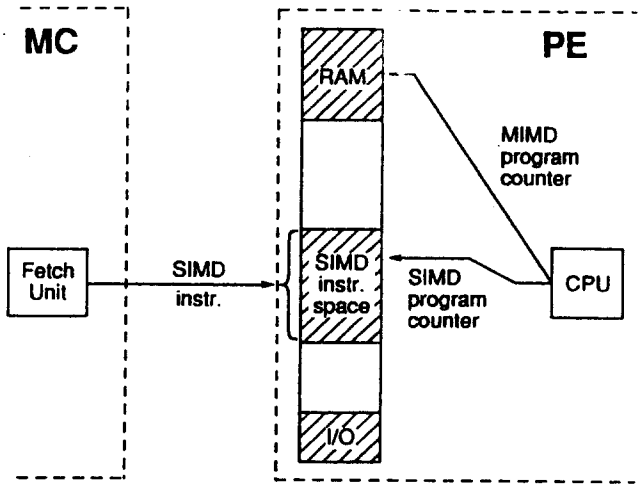

Figure 3.3: Prototype PE address space.

sisting of more than one MC group, synchronization logic located in the System Control Unit determines when all PEs of a given submachine have requested a SIMD instruction and instructs the MCs to broadcast the next SIMD instruction. This logic is a simple AND-tree [ScN87] of depth $\log_2 Q$ that can be short-circuited into smaller trees to synchronize multiple submachines of various sizes. Due to the use of non-latched combinatorial logic in the tree, the added latency of SIMD instruction accesses to SIMD space in the prototype are, on average, one cycle (125 ns) faster than PE RAM accesses because of the faster and smaller memory used in the FIFO instruction queue. To further hide the latency of instruction fetches (in both SIMD and MIMD modes) the PE processors pre-fetch instruction words wherever possible.

When all PEs in a submachine have requested an instruction, the Fetch Units in the submachine dequeue an instruction and its enable bits from their queue and broadcast them to the PE IBUs. If an IBU determines from the broadcast enable bits that its PE is enabled for the instruction, it uses the received instruction to satisfy the pending read of its PE. However, if the IBU determines that its PE is disabled for the current instruction, then the PE's read request is not satisfied and the PE continues to wait for an instruction for which it is enabled. Accesses made to PE RAM are satisfied by the physical RAM so that accesses to local data resident in PE RAM are handled correctly.

One advantage of this scheme is the ease of switching between SIMD and MIMD modes. A PE in MIMD mode can enter SIMD mode simply by executing a branch to the SIMD instruction space. Similarly, a PE currently in SIMD mode can switch to an MIMD program by executing a branch to the PE's physical RAM space. Thus, while in SIMD mode, MIMD subroutines may be executed by causing the PEs to execute a "branch to subroutine" instruction. Such an instruction will cause the current program counter (pointing to SIMD space) to be saved on the stack. Then, when a "return from subroutine" instruction is executed in MIMD mode, the old program counter will be restored, and the PE will be back in SIMD mode. Such flexibility in mode switching allows mixed-mode programs to be written that change modes at instruction level granularity with negligible overhead. The SIMD instruction fetch mechanism can also be used to support a form of MIMD barrier synchronization [DiS89].

The evaluation of "if-any" and "if-none" data conditional statements requires MCs to obtain conditional data information from other MC groups in the same submachine. This is accomplished through the use of an AND-tree similar to that used for SIMD instruction broadcast synchronization across MC groups. When an "if-any" or "if-none" type statement is encountered each MC in the submachine collects the one-bit conditional result from each of its PEs and forms the appropriate one-bit result for its MC group. This one-bit result is then fed into the AND-tree where a one-bit global (submachine) result is formed and sent to all MCs in the submachine.

3.4. Interconnection Network Implementation

The PASM prototype interconnection network is an Extra Stage Cube [AdS82, Sie90]. To simplify the hardware, circuit switching (as opposed to packet switching) is used. To transfer data between a network source-destination pair, a physical path must first be made connecting the pair. The source PE establishes the path by writing the appropriate routing tag to its memory-mapped network interface logic and is informed when the virtual circuit through the network is complete. Once the path is established, data transfers can proceed. The source PE writes the individual words of the message to the network interface (parallel ports) that automatically transfer the data to the network interface of the destination PE. The destination PE reads the message word-by-word. At the conclusion of transmission, the source PE relinquishes the path by writing a control word to its network interface.

The PASM prototype network is constructed from off-the-shelf TTL SSI/MSI components and implements a 16-bit wide data path plus two parity bits. The use of standard TTL components in the prototype network yields an inexpensive network that does not limit throughput on established paths. The operating speed of the PE CPU is therefore the limiting factor in determining how fast the network must transfer data. Hardware in the network interface provides correct destination verification on each path established and performs parity checking on all transmitted data.

Assuming no conflicts, the time to establish a path is approximately 2 μs. Once a path is established, the network hardware can support a transfer rate of 24M bits/second, but is limited to a sustained transfer rate of 3.8M bits/second due to the processing rate of the PE CPU.

3.5. Prototype Functionality

The prototype provides all three dimensions of PASM's reconfigurability. It is being actively used to conduct experiments studying how to exploit this flexibility.

4. Summary

The PASM parallel processing system is dynamically reconfigurable along three dimensions: partitionability, modes of parallelism, and communication among processors. Much knowledge related to reconfigurable systems is to be learned from experience with the prototype. There is a large group of faculty and students exploring the design and use of PASM concepts through a variety of application studies and enhancements to the prototype.

Current projects associated with PASM include: developing a parallel C language for explicit specification of parallelism [NiS90], examining the trade-offs involved with mixed-mode parallelism (e.g., [FiC91]), performing application experiments on the prototype and extrapolating to larger machines (e.g., [BrC90]), exploring ways to use partitioning [Sie90], mapping tasks onto reconfigurable parallel machines [Jam87], investigating instrumentation for performance monitoring and debugging [LuF89, MaL90], exploiting visualization for understanding system behavior, studying an automatic reconfiguration system for improving performance and fault tolerance [ChD89], and researching hardware design improvements (e.g., [NaF90]). The PASM prototype is a constantly evolving tool for understanding the programming and design of parallel processing systems.

Acknowledgments: We thank Mark A. Nichols and Daniel W. Watson for their helpful comments.

References

[AdS82] G. B. Adams III and H. J. Siegel, "The extra stage cube: a fault-tolerant interconnection network for supersystems," *IEEE Trans. Computers,* Vol. C-31, No. 5, May 1982, pp. 443-454.

[BaB68] G. H. Barnes, R. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, "The Illiac IV computer," *IEEE Trans. Computers,* Vol. C-17, No. 8, Aug. 1968, pp. 746-757.

[Bat76] K. E. Batcher, "The flip network in STARAN," *1976 Int'l Conf. Parallel Processing,* Aug. 1976, pp. 65-71.

[Bat77] K. E. Batcher, "STARAN series E," *1977 Int'l Conf. Parallel Processing,* Aug. 1977, pp. 140-143.

[Bat80] K. E. Batcher, "Design of a massively parallel processor," *IEEE Trans. Computers,* Vol. C-29, No. 9, Sep. 1980, pp. 836-844.

[Bat82] K. E. Batcher, "Bit serial parallel processing systems," *IEEE Trans. Computers,* Vol. C-31, No. 5, May 1982, pp. 377-384.

[Bla90] T. Blank, "The MasPar MP-1 architecture," *IEEE Compcon,* Feb. 1990, pp. 20-24.

[BoD72] W. J. Bouknight, S. A. Denenberg, D. E. McIntyre, J. M. Randall, A. H. Sameh, and D. L. Slotnick, "The Illiac IV system," *Proc. of the IEEE,* Vol. 60, No. 4, Apr. 1972, pp. 369-388.

[BrC90] E. C. Bronson, T. L. Casavant, and L. H. Jamieson, "Experimental application-driven architecture analysis of an SIMD/MIMD parallel processing system," *IEEE Trans. Parallel and Distributed Systems,* Vol. 1, No. 2, Apr. 1990, pp. 195-205.

[ChD89] C. H. Chu, E. J. Delp, L. H. Jamieson, H. J. Siegel, F. J. Weil, and A. B. Whinston, "A model for an intelligent operating system for executing image understanding tasks on a reconfigurable parallel architecture," *J. Parallel and Distributed Computing,* Vol. 6, No. 3, June 1989, pp. 598-622.

[ClS85] C. L. Cline and H. J. Siegel, "Augmenting Ada for SIMD parallel processing," *IEEE Trans. Software Engineering,* Vol. SE-11, No. 9, Sep. 1985, pp. 970-977.

[CrG85] W. Crowther, J. Goodhue, R. Thomas, W. Milliken, and T. Blackadar, "Performance measurements on a 128-node butterfly parallel processor," *1985 Int'l Conf. Parallel Processing,* Aug. 1985, pp. 531-540.

[DeB80] J. B. Dennis, G. A. Boughton, and C. K. C. Leung, "Building blocks for data flow prototypes," *7th Ann. Symp. Computer Architecture,* May 1980, pp. 1-8.

[DiS89] H. G. Dietz, T. Schwederski, M. T. O'Keefe, and A. Zaafrani, "Static synchronization beyond VLIW," *Supercomputing '89,* Nov. 1989, pp. 416-425.

[DuB88] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, "Image processing on SIMD/SPMD architecture: OPSILA," *9th Int'l Conf. Pattern Recognition,* Nov. 1988, pp. 430-433.

[FiC88] S. A. Fineberg, T. L. Casavant, T. Schwederski, and H. J. Siegel, "Non-deterministic instruction time experiments on the PASM system prototype," *1988 Int'l Conf. Parallel Processing,* Aug. 1988, pp. 444-451.

[FiC91] S. A. Fineberg, T. L. Casavant, and H. J. Siegel, "Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting," *J. Parallel and Distributed Computing,* to appear, 1991.

[Fly66] M. J. Flynn, "Very high-speed computing systems," *Proc. of the IEEE,* Vol. 54, No. 12, Dec. 1966, pp. 1901-1909.

[Fou81] T. J. Fountain, "CLIP4: progress report," in *Languages and Architectures for Image Processing,* M. J. B. Duff and S. Levialdi, eds., Academic Press, London, England, 1981, pp. 281-291.

[Fre89] R. F. Freund, "Optimal selection theory for superconcurrency," *Supercomputing '89,* Nov. 1989, pp. 699-703.

[GoL73] G. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," *1st Ann. Symp. Computer Architecture,* Dec. 1973, pp. 21-28.

[GoG83] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer — designing an MIMD shared-memory parallel computer," *IEEE Trans. Computers,* Vol. C-32, No. 2, Feb. 1983, pp. 175-189.

[HaM89] J. P. Hayes and T. Mudge, "Hypercube supercomputers," *Proc. of the IEEE,* Vol. 77, No. 12, Dec. 1989, pp. 1829-1841.

[Hil85] W. D. Hillis, *The Connection Machine,* MIT Press, Cambridge, MA, 1985.

[Hun89] D. J. Hunt, "AMT DAP - a processor array in a workstation environment," *Computer Systems Science and Engineering,* Vol. 4, No. 2, Apr. 1989, pp. 107-114.

[Jam87] L. H. Jamieson, "Characterizing parallel algorithms," in *The Characteristics of Parallel Algorithms,* L. H. Jamieson, D. B. Gannon, and R. J. Douglass, eds., MIT Press, Cambridge, MA, 1987, pp. 65-100.

[KuD86] D. J. Kuck, E. S. Davidson, D. H. Lawrie, and E. H. Sameh, "Parallel supercomputing today and the Cedar approach," *Science,* Vol. 231, Feb. 1986, pp. 967-974.

[Law75] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Computers,* Vol. C-24, No. 12, Dec. 1975, pp. 1145-1155.

[LuF89] J. E. Lumpp, S. A. Fineberg, W. G. Nation, T. L. Casavant, E. C. Bronson, H. J. Siegel, P. H. Pero, T. Schwederski, and D. C. Marinescu, "CAPS - a coding aid used with the PASM parallel processing system,"

Workshop on Experiences with Building Distributed and Multiprocessor Systems, Oct. 1989, pp. 269-288.

[MaL90] D. C. Marinescu, J. E. Lumpp, T. L. Casavant, and H. J. Siegel, "Models for monitoring and debugging tools for parallel and distributed software," *J. Parallel and Distributed Computing,* Vol. 9, No. 2, June 1990, pp. 171-184.

[NaF90] W. G. Nation, S. A. Fineberg, M. D. Allemang, T. Schwederski, T. L. Casavant, and H. J. Siegel, "Efficient masking techniques for large-scale SIMD architectures," *Frontiers '90: The 3rd Symp. Frontiers of Massively Parallel Computation,* to appear, Oct. 1990.

[NiS90] M. A. Nichols, H. J. Siegel, and H. G. Dietz, "Data management and control-flow constructs in a SIMD/SPMD parallel language/compiler," *Frontiers '90: The 3rd Symp. Frontiers of Massively Parallel Computation,* to appear, Oct. 1990.

[Nug88] S. F. Nugent, "The iPSC/2 direct-connect communications technology," *3rd Conf. Hypercube Computers and Applications,* January 1988, pp. 51-60.

[Nut77] G. J. Nutt, "Microprocessor implementation of a parallel processor," *4th Ann. Symp. Computer Architecture,* Mar. 1977, pp. 147-152.

[Pat81] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Computers,* Vol. C-30, No. 10, Oct. 1981, pp. 771-780.

[Pea77] M. C. Pease III, "The indirect binary n-cube microprocessor array," *IEEE Trans. Computers,* Vol. C-26, No. 5, May 1977, pp. 458-473.

[PfB85] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): introduction and architecture," *1985 Int'l Conf. Parallel Processing,* Aug. 1985, pp. 764-771.

[RiS88] M. D. Rice, S. B. Seidman, and P. Y. Wang, "A formal model for SIMD computation," *Frontiers '88: The 2nd Symp. Frontiers of Massively Parallel Computation,* Oct. 1988, pp. 601-607.

[ScN87] T. Schwederski, W. G. Nation, H. J. Siegel, and D. G. Meyer, "Design and implementation of the PASM prototype control hierarchy," *2nd Int'l Supercomputing Conf.,* May 1987, pp. 418-427.

[Sei85] C. L. Seitz, "The Cosmic Cube," *Communications of the ACM,* January 1985, pp. 22-33.

[SeU80] M. C. Sejnowski, E. T. Upchurch, R. N. Kapur, D. P. S. Charlu, and G. J. Lipovski, "An overview of the Texas Reconfigurable Array Computer," *AFIPS 1980 Nat'l Computer Conf.,* June 1980, pp. 631-641.

[Sie77] H. J. Siegel, "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," *IEEE Trans. Computers,* Vol. C-26, No. 2, Feb. 1977, pp. 153-161.

[Sie80] H. J. Siegel, "The theory underlying the partitioning of permutation networks," *IEEE Trans. Computers,* Vol. C-29, No. 9, Sep. 1980, pp. 791-801.

[Sie90] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies, 2nd Edition,* McGraw-Hill, New York, NY, 1990.

[SiA90] H. J. Siegel, J. B. Armstrong, and D. W. Watson, "Mapping tasks onto the PASM reconfigurable parallel processing system," *1990 Parallel Computing Workshop,* sponsored by the Computer and Information Science Department at The Ohio State University, this proceedings.

[SiN89] H. J. Siegel, W. G. Nation, C. P. Kruskal, and L. M. Napolitano, "Using the multistage cube network topology in parallel supercomputers," *Proc. of the IEEE,* Vol. 77, No. 12, Dec. 1989, pp. 1932-1953.

[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 Trans. Computers,* Vol. C-30, No. 12, Dec. 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, DC, 1987, pp. 387-407.

[SwF77] R. J. Swan, S. Fuller, and D. P. Siewiorek, "Cm*: a modular multimicroprocessor," *AFIPS 1977 Nat'l Computer Conf.,* June 1977, pp. 637-644.

[ThN81] S. Thanawastien and V. P. Nelson, "Interference analysis of shuffle/exchange networks," *IEEE Trans. Computers,* Vol. C-30, No. 8, Aug. 1981, pp. 545-556.

[TuR88] L. W. Tucker and G. G. Robertson, "Architecture and applications of the Connection Machine," *Computer,* Vol. 21, No. 8, Aug. 1988, pp. 26-38.

[WuF80] C.-L. Wu and T. Y. Feng, "On a class of multistage interconnection networks," *IEEE Trans. Computers,* Vol. C-29, No. 8, Aug. 1980, pp. 694-702.