Software Issues for the PASM Parallel Processing System

James B. Armstrong, Daniel W. Watson, and Howard Jay Siegel

Parallel Processing Laboratory, School of Electrical Engineering,
Purdue University, West Lafayette, IN 47907-1285, USA

Abstract: Partitionable mixed-mode systems can be dynamically reconfigured to form independent or communicating submachines of various sizes, where each submachine can switch between the SIMD and MIMD modes of parallelism at instruction level granularity. These systems allow software developers to tailor the system size and mode of parallelism to best match the problem. This chapter explores software issues related to one such system, the PASM parallel processing system. Specific topics addressed include a programming language, overlapped CU/PE operation, trade-offs between the SIMD and MIMD modes of parallelism, and aspects of operating systems for automatic system reconfiguration. A reading-list of PASM-related publications is appended.

Keywords: MIMD, mixed-mode, parallel processing, PASM, reconfigurable, SIMD.

1. Introduction

The parallel processing systems being designed today come in many "shapes" and "sizes." The diversity reflects the myriad of dissimilar application areas demanding greatly enhanced computing power [7]. In an effort to match machine configuration to program characteristics, system designers have created parallel systems that can switch between the SIMD and MIMD modes of parallelism. Although the goal in each of these systems is to exploit the advantages of each mode, there are different levels of granularity at which mode switching can be supported, such as: system level (heterogeneous networks of computers [4]), submachine level (Disputer [9]), program level (NETRA [2], TRAC [5]), subroutine level (PM^4 [1]), and instruction level (CM-5 [10], Opsila [3], PASM [8], Triton[6]). Furthermore, to better match different problem sizes and to exploit subtask parallelism, some of the above systems can be partitioned to form independent or communicating submachines of various sizes. The variety of machine types has necessitated the design of new programming languages, compilers, operating systems, and algorithm mapping techniques.

This research was supported by the National Aeronautical and Space Administration under grant number NGT-50961, the Office of Naval Research under grant number N00014-90-J-1937, and the National Science Foundation under grant number CDA-9015696.

This chapter considers some of the basic software issues that relate to partitionable systems whose submachines can dynamically switch between the SIMD and MIMD mode of parallelism at instruction level granularity; that is, submachines capable of mixed-mode parallelism. This is done by discussing some aspects of the research conducted in architecture, languages, algorithms, and operating systems for the PASM parallel processing system. Designed at Purdue, the PASM system can support 1024 processing elements (PEs - processor/memory pairs), and incorporates the flexibility needed to investigate many software issues affecting the design and use of a partitionable mixed-mode system. A 30-processor small-scale prototype (16 processors in the computational engine) has been built and is being used as a testbed for research studies. The PASM team currently includes the efforts of 12 faculty members and numerous graduate students, involved in a large number of inter-related activities. PASM is a constantly evolving research tool, providing a unique, dynamically reconfigurable environment for studying parallel processing. Although the software issues in this chapter are discussed with respect to the PASM system, the information presented can be applied to a variety of machines.

This chapter is organized as follows. Section 2 discusses the PASM system architecture. An explicitly parallel language designed for use on reconfigurable systems is described in Section 3. Section 4 treats in detail one SIMD/MIMD trade-off (SIMD control unit and PE computational overlap), and overviews the impact of multiple SIMD/MIMD trade-offs on mixed-mode implementations of an application. In Section 5, a model for an automatic reconfiguration system is outlined. Appended to the chapter is a reading list of PASM-related publications. The papers in the reading list also reference relevant work by others that have influenced PASM activities. References to the reading list will be in the form "[A-1]," which would refer to the first reference in part A of the reading list.

2. PASM Architecture

The PASM (partitionable SIMD/MIMD) system can dynamically reconfigure to form independent or communicating submachines of various sizes, where each submachine operates in mixed-mode. In addition, PASM uses a flexible multistage cube interconnection network, which allows the connection pattern between the processors to be varied. Thus, PASM is reconfigurable along three dimensions: partitionability, mode of parallelism (SIMD/MIMD), and inter-processor connections. The PASM prototype implements all three of these dimensions of reconfigurability.

The PASM conceptual design is comprised of six basic components, as shown in Figure 1. 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 PEs to form submachines of various sizes resides in the System Control Unit.

The Parallel Computation Unit contains $N = 2^n$ PEs physically numbered from 0 to $N-1$, and an interconnection network. Two memory units are used in each PE such that computation and memory I/O can be overlapped; for example, the PE processor can execute a job in one memory unit while the next job is preloaded into the other memory unit from secondary storage. 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 (as in the prototype) or custom designed for parallel processing and/or a particular application.

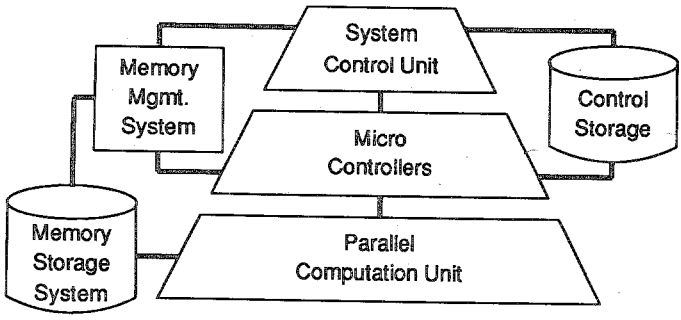

Figure 1. Block diagram of the PASM system.

The Micro Controllers (MCs) are the multiple control units (CUs) required to have multiple independent submachines capable of executing in SIMD mode. There are $Q=2^q$ MCs, physically numbered from 0 to $Q-1$. Each MC controls a fixed group of N/Q PEs in the Parallel Computation Unit. An MC and its associated PEs form an MC group. All N/Q PEs connected to MC i have i as the low-order q bits of their physical PE number. In an $N=1024$ system, Q may be 32; for the $N=16$ prototype, $Q=4$. MC groups can be combined to form submachines of different sizes.

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 SIMD instruction fetch unit. A PE fetches SIMD instructions by reading an instruction word from the logical SIMD instruction space. This is a logical address space – no physical memory is provided. Logic in the PE detects read accesses to the SIMD instruction space, and any such request is interpreted as an SIMD instruction request. 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.

The interconnection network allows the PEs to communicate with each other. Because PASM is a partitionable system, the interconnection network used must be partitionable. The reconfiguration rule in PASM requires that the physical numbers of all the 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 the submachine. The extra stage cube, a fault tolerant variant of the multistage cube, was implemented in the PASM prototype and satisfies these requirements.

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 MSU 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. 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, allowing all N/Q MSUs to be used concurrently whenever one or more MC groups are loaded/unloaded.

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.

The 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 goal of current architecture research projects is to evaluate, enhance, and learn from the prototype implementation methodology so that a full 1024-PE system can be built effectively. For additional information about the PASM organization, see the papers listed in the "Architectural Issues" section of the reading list. Issues discussed include: barrier synchronization hardware, the control hierarchy in the prototype, enabling/disabling processors, fault tolerance, and interconnection networks. The architecture overview above is summarized from [A-10].

3. Parallel Language

The Explicit Language for Parallelism (ELP) is currently under development for PASM. ELP requires the user to explicitly indicate the parallelism to be employed.

ELP syntax is based on C, and provides constructs for specifying and using SIMD, MIMD, and mixed-mode parallelism. ELP distinguishes a subclass of MIMD mode where all processors execute the same program, but do so asynchronously with respect to one another; that is, each follows its own control path. This is referred to as SPMD (single program - multiple data) mode. A goal of ELP is to provide uniformity with respect to the SIMD and SPMD modes of parallelism by having interpretations for each element of the language in both modes that are identical in semantics. This is an important characteristic because it allows a data-parallel algorithm to be coded in a mode-independent manner, producing a data-parallel program for which: (1) the ELP compiler can be instructed to generate only SIMD code, (2) the ELP compiler can be instructed to generate only SPMD code, or (3) execution mode specifiers can be added easily by the user to facilitate mixed-mode experimentation.

An example of a feature of ELP that has the same semantics in SIMD and SPMD modes is the variable class associated with each variable. A variable defined to be of class mono always has a single value with respect to all PEs, independent of execution mode (i.e., a mono variable is scalar-valued); whereas a variable defined to be of class poly can have one or more values with respect to all PEs, independent of execution mode (i.e., a poly variable is vector-valued). Each mono variable has storage allocated for it on the CU and all PEs. If a mono variable is referenced while in SIMD mode, its CU storage is active. If a mono variable is referenced while in SPMD mode, its PE storage is active and all PE copies of the mono variable will have the same value (guaranteed by the compiler). For variables defined to be poly, each PE has its own copy with its own value, independent of execution mode.

In SIMD mode, operations on mono variables indicate work to be done on the CU, and they permit the overlapping of CU and PE computation to be explicitly specified. This, in turn, allows the user to experiment with load balancing between the CU and the PEs (discussed later in this chapter). In SPMD mode, mono variables can be used to force if, while, do, and for statements on different PEs to execute in the same fashion on all PEs; for example, mono

variables could be used as the index variable and as the common upper bound for a for loop with all PEs. All PEs must execute the same instructions, but not necessarily at the same time (as in SIMD mode). Mono variables also permit other SPMD operations to be performed in an identical fashion across all PEs, such as having each PE access the same element of an array.

The SIMD and SPMD modes of parallelism are supported by a full native-code compiler, under development for the PASM prototype, that permits these modes to be switched dynamically at instruction level granularity. ELP is being extended to include full MIMD capability and user specification for partitioning the system into submachines for subtask parallelism. The long term goal is to develop the technology that will allow the compiler to select the best mode of parallelism for each section of code automatically and to balance CU/PE overlap in SIMD mode.

More about PASM programming topics can be found in papers listed in the "Languages and Compiling Topics" section of the reading list. Topics examined include: barrier synchronization techniques, compilation methods, language constructs, memory usage, overlap of CU and PE operations, and SIMD/MIMD mode specification. The language overview above is summarized from [B-8].

4. Parallel Algorithm Studies

The effectiveness of a parallel algorithm can vary greatly depending on the mode of parallelism (SIMD/MIMD) the target machine supports. If the parallel programmer has the benefit of a heterogeneous computing environment [4] or is programming a mixed-mode system, he or she must evaluate the impact of any SIMD versus MIMD mode trade-offs to determine the mode of parallelism that is best suited for each portion of the algorithm. For some of the trade-offs illustrated in Figure 2, a quantitative analysis of the impact of parallel mode selection is not possible; some form of heuristic analysis (possibly, an experienced programmer's judgement) is needed. For example, the advantage of executing variable-time instructions in MIMD mode cannot be quantified at compile-time because it is data-dependent, but it can have an impact on performance and must be considered in some way. Fortunately, other trade-offs can be quantified statically (i.e., before executing the task), under certain circumstances. In this section, an example of how a programmer or compiler may quantify the SIMD advantage of overlapping CU and PE computation (known as CU/PE overlap) is presented in general terms. This is followed by a discussion of the interaction of several trade-offs in a bitonic sequence sorting application.

CU/PE overlap refers to the ability of some SIMD machines to overlap control flow instructions, as well as any scalar instructions that are independent of the processor data (e.g., array address calculations needed by all PEs), with instructions that are executed on the PEs. When CU/PE overlap occurs, the total execution time for a program is measured from the start of execution to the time when all the PEs and the CU have completed their execution. There may be an unequal amount of work on the CU and PEs, causing one to become idle. It has been shown that the best execution time is achieved by balancing the work between the CU and PEs (e.g., [C-17]).

Figure 3 shows the code segment that computes $\Sigma\Sigma\, T[i, j]\, A[i, j]$, for $0 \le i < C$ and $0 \le j < R$, where T is an $R_T \times C_T$ array and A is an $R_A \times C_A$ array ($R \le R_T \le R_A$, $C \le C_T \le C_A$). This is the same type of calculation that is in an inner loop of an image correlation algorithm, and is representative of the computation in many window-based image processing tasks. The numbers

SIMD/MIMD Trade-Offs

SIMD Advantages

- ease of programming and debugging
  - SIMD: single program, PEs operate synchronously
  - MIMD: multiple interacting programs, PEs operate asynchronously
- overlap loop control with operations
  - SIMD: control unit does increment and compare while PEs "compute"
  - MIMD: same processor does both
- overlap operations on common data
  - SIMD: control unit overlaps operations that all PEs need (e.g., common local array addresses)
  - MIMD: same processor does all
- reduced inter-PE transfer overhead
  - SIMD: "send" and "receive" automatically synchronized
  - MIMD: need explicit synchronization and identification protocol
- minimal synchronization overhead
  - SIMD: implicit in program
  - MIMD: need explicit statements (e.g., semaphores)
- less program memory space required
  - SIMD: store one copy of program
  - MIMD: each PE stores own copy
- minimal instruction decoder cost
  - SIMD: decoder in control unit
  - MIMD: decoder in each PE

MIMD Advantages

- more flexible
  - no constraints on operations that may be performed concurrently
- conditional statements more efficient
  - SIMD: "then" and "else" execution serialized
  - MIMD: each PE executes as if uniprocessor
- no SIMD control unit cost
- variable-time instructions more efficient
  - assume there is a block of instructions where the execution time of each instruction is data dependent
  - SIMD: waits for slowest PE to execute each instruction ("sum of maxs")
    $T_{SIMD} = \sum_{instr.} \max_{PE}$ (instr. time)
  - MIMD: waits for slowest PE to execute block of instructions ("max of sums")
    $T_{MIMD} = \max_{PE} \sum_{instr.}$ (instr. time)
  - example: execution of 3 instructions in SIMD mode and MIMD mode

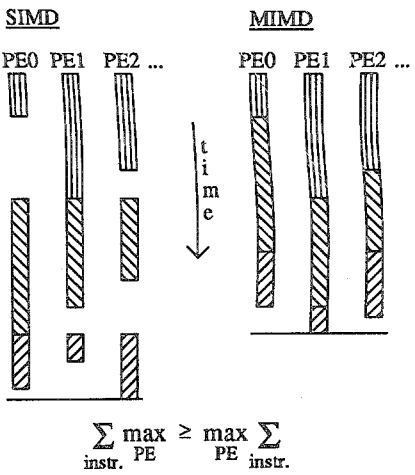

Figure 2. SIMD/MIMD trade-offs [C-18].

```
CU                                                                    PE
-    Tbase = T[];                    /* initialize pointers */
-    Abase = A[];

14   for (i = 0; i < R; i++) {       /* 4, 8, 6 */
32     Tptr = Tbase + C_T*i;         /* increment row ptrs */
32     Aptr  = Abase + C_A*i;
17     send_int(&Tptr, Tptr);        /* send ptrs from */              8
17     send_int(&Aptr, Aptr);        /* CU to PEs    */                8
14     for (j = 0; j < C; j++) {     /* 4, 8, 6 */
 6       simdbegin                   /* broadcast SIMD block */
           Tptr += 1;                /* increment column ptrs */      10
           Aptr  += 1;                                               10
           xysum += (*Tptr) * (*Aptr);                               34
         simdend
       }
     }
```

Figure 3. Code segment that shows CU/PE overlap.

along the left and right sides of the figure provide approximate statement execution times for the CU and PEs, respectively. All times in this section are in microseconds, and have been derived empirically on the PASM prototype. The comment adjacent to the for statement indicates the time to initialize the loop control variable (4 μs), to test for the end-of-loop condition (8 μs), and to increment the loop control variable (6 μs). This distinction is necessary because the test and increment operations are separate in the assembly code; that is, the increment operation is performed at the end of the loop. For each iteration after the first, only the test and increment are performed, which takes 8 + 6 = 14 μs. This is why 14 is marked in the left column.

The code syntax in Figure 3 has been chosen to show the interaction between the CU and the PEs. For the block of instructions between a simdbegin and a simdend, the CU CPU generates control signals that move the block from the CU RAM to an instruction queue. This takes only 6 μs. Independently of the CU CPU, the instruction queue is capable of broadcasting instructions to the PEs (in FIFO order) when all enabled PEs request the next instruction. Furthermore, for the statement send_int(), the CU generates and places in the instruction queue (17 μs) an instruction that, when executed by the PEs, loads the value of the CU's i variable into the PEs' i variable (8 μs). The other instructions listed are completely executed by either the CU or the PEs.

Now consider the overlap of CU and PE computation that occurs when the first iteration of the code segment of Figure 3 is executed. Assume the pointers Tbase and Abase have been initialized at compile-time. The CU takes 4 μs to set i = 0, 8 μs to test i for the end-of-loop condition, and 64 μs to update the pointers Tptr and Aptr. Then, the CU uses 17 μs to generate the instructions for the send_int(&Tptr,Tptr) statement. These instructions are placed in the instruction queue. The PEs take 8 μs to execute these instructions, while the CU generates and places in the instruction queue the instructions for the send_int(&Aptr,Aptr) statement (17 μs). Because 17 μs were required by the CU for the send_int(&Aptr,Aptr) statement, the PEs have finished receiving the Tptr variable and can immediately begin the 8 μs of computation required to receive the Aptr variable. The PEs' 8 μs are overlapped with the CU's computation of setting j = 0 (4 μs), and testing j for the end-of-loop condition (8 μs). At this point the PEs are idle once

again. Finally, the CU takes 6 μs to move the SIMD block of instructions from the CU RAM to the instruction queue. The PEs require 54 μs to fetch and execute the SIMD block of instructions queued for them. While the PEs execute the instructions from the instruction queue, the CU continues the succeeding iterations of the loop, performing control flow instructions and queueing instructions for the PEs.

By using the statement execution times given, the exact amount of overlapped computation for the code segment in Figure 3 can be computed [C-17]. Because the execution time of the code segment is not data-dependent, one objective for an SIMD compiler is to determine the optimal distribution of work between the CU and the PEs (as mentioned at the end of Section 3). A theoretical model for CU/PE overlap that assumes a general SIMD architecture is given in [C-14].

If the loop in Figure 3 were executed in MIMD mode, the send_int(), simdbegin, and simdend instructions would not be needed. The PEs would execute all other instructions (no work is done by the control unit in MIMD mode). An analysis of the code segment in Figure 3 has shown that the SIMD mode implementation outperforms the corresponding MIMD mode implementation by 28 percent [C-17]. This illustrates that CU/PE overlap can be quantified and has a significant impact on execution time, especially when array-intensive and loop-intensive computations are involved. Consequently, it must be considered when comparing the SIMD, MIMD, and mixed-mode implementations of an algorithm.

As another example, consider the bitonic sorting of sequences on the PASM prototype. This is not a study of sorting *per se* but a study of the interaction of algorithm characteristics with modes of parallelism.

Assume there are M numbers and $N = 2^n$ PEs, where M is an integer multiple of N, and that M/N numbers are stored in each PE, initially sorted. The goal of the bitonic sequence sorting algorithm is to have each PE contain a sorted list of M/N elements, where the elements in PE i are less than or equal to the elements in PE k, for $i < k$. A version of Batcher's regular bitonic sorting algorithm, where $M = N$, is modified in Figure 4 to accommodate the M/N element sequence in each PE. Instead of performing a pairwise comparison at each step, an ordered-merge is done between the local PE sequence X and the transferred sequence Y using local data-conditional statements ("merge(X,Y)"). The lesser half of the merged sequence is assigned the pointer X and the greater half is assigned the pointer Y. The pointers to the two lists are then swapped, based on a pre-computed data-independent mask ("swap(X,Y)").

```
for k = 1 step + 1 until log2N do
    for i = k − 1 step − 1 until 0 do
        for q = 1 step + 1 until M/N do
            load X[q] into network
            send to PE whose number differs in bit i
            Y[q] ← network output
        merge(X, Y)
        swap(X, Y)
```

Figure 4. Bitonic sequence sorting algorithm executed by each PE.

When choosing the mode of parallelism, the programmer must consider two salient characteristics of the algorithm. First, the ordered-merge involves many comparisons that can be more efficiently computed in MIMD mode (due to the multiple control paths). Second, the

algorithm requires many network transfers, which are better performed in SIMD mode (due to simpler communication protocols resulting from the implicit synchronization). To evaluate different approaches to this algorithm, a pure SIMD, a pure MIMD, and two mixed-mode implementations have been executed on the prototype.

In the S/MIMD (SIMD/MIMD) mixed-mode implementation, the ordered-merge and swap routines were executed in MIMD mode, while the rest of the operations, including network transfers, were performed in SIMD mode. This algorithm has an advantage over pure SIMD and pure MIMD implementations because all comparisons are done in MIMD mode and all network transfers are done in SIMD mode. Additionally, there is potential for significant CU/PE overlap in the SIMD instructions.

The BMIMD (barrier MIMD) mixed-mode implementation uses MIMD mode, but employs barrier synchronization to synchronize all inter-PE transfers. On PASM this is accomplished by fetching a word from the SIMD address space, thus using the SIMD instruction fetch synchronization hardware to implement the barrier. Consequently, the PEs can perform the transfer without the overhead normally involved with MIMD network transfers. Thus, the BMIMD implementation has the advantage of performing data-dependent conditionals in MIMD mode, but performs barrier synchronization to reduce inter-PE data-transfer overhead. Therefore, its performance would be expected to be better than pure SIMD or pure MIMD.

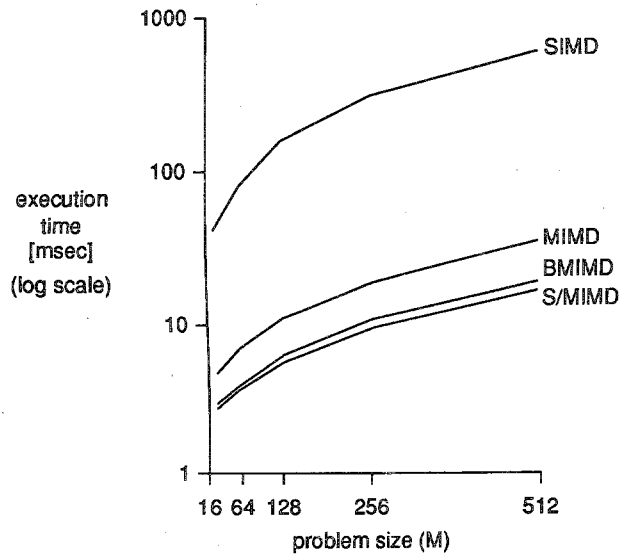

Figure 5. Bitonic sequence sorting execution time versus problem size for N = 16 PEs.

The results of the SIMD, MIMD, S/MIMD, and BMIMD mode algorithms for the bitonic sequence sorting problem with N = 16 PEs are shown in Figure 5. There is a significant improvement in execution time for both mixed-mode algorithms. S/MIMD performed better than BMIMD, with the difference increasing with M, mainly because of the CU/PE overlap. The

research described in [C-12] shows that the relative performance of the four approaches are the product of properties inherent to the modes of parallelism and not artifacts of the prototype construction.

The goal of the algorithm research activities is to explore ways to exploit the flexibility of a reconfigurable parallel processing system. This knowledge can then be codified in new programming techniques and compiler technology.

PASM-related algorithm studies have included theoretical analyses, simulations, and experiments on the PASM prototype. These studies have examined issues such as mapping tasks onto reconfigurable parallel processing systems, trade-offs between the SIMD and MIMD modes of parallelism, CU/PE overlap in SIMD mode, and the impact of partitioning on performance. Applications considered include: bitonic sorting, edge-guided thresholding, FFTs, global histogramming, image correlation, image smoothing, matrix multiplication, range-image segmentation, scaling and rotational registration, and word recognition. Papers about these algorithm research efforts are listed in the "Algorithm Studies" section of the reading list.

5. Automatic Reconfiguration System

Many tasks (e.g., image processing) involve having a collection of algorithms operate on a given data set, some of which can execute concurrently and some of which must execute in sequence. An assignment of resources (e.g., PEs) to each algorithm (i.e., subtask) needs to be made with the goal of minimizing execution time for the entire task. A model for an Automatic Reconfiguration System (ARS) that would do this dynamically is overviewed in this section.

In the case of image understanding applications, a single task may consist of a number of subtasks (e.g., median filtering, edge detection, texture analysis, boundary tracing, region formation, object recognition). The temporal ordering of subtasks follows a data-dependent precedence graph that allows for the concurrent execution of certain subtasks. Each node of the graph represents one of the subtasks that is conditionally executed, depending upon the results from previous subtasks. An image understanding system must select an appropriate algorithm for each subtask (node), based upon image analysis properties as well as the properties of the input images (e.g., signal-to-noise ratio). The control-flow graph is then passed to the ARS from the image understanding layer.

The ARS requires information about the system configuration to schedule the subtasks effectively. The state of a reconfigurable system is the division of the PEs into submachines and the state of any jobs executing or awaiting execution at the given point in time. To minimize overall task execution time, as subtasks complete, their associated PEs may be assigned to waiting subtasks or joined with PEs assigned to currently executing subtasks to complete the subtask on a larger partition. Relevant parameters used to make resource allocation decisions include: precedence constraints among subtasks, assignment of algorithms to submachines, execution time expended and the amount of working memory consumed by each algorithm, relationship of execution time to number of PEs used in each submachine and to input size, expected execution time, expected memory requirements, and data allocation schemes among the PEs of the submachine for both input and output data.

From the control-flow (data-dependent precedence) graph, the ARS schedules the execution of the algorithms by determining a system state for each point in time. Because some algorithms may have non-deterministic execution times, the ARS is responsible for dynamically updating

the system state, when necessary, to maximize performance. Thus, currently executing subtasks may have their resource allocation changed.

Associated with each algorithm may be several parallel implementations from which the ARS can choose the one that results in the best overall task execution. A parallel implementation may be chosen on the basis of performance/system-requirements characteristics such as: resolution of data conditionals, current allocation of resources, algorithm execution time as a function of resources (PEs), subtask precedence constraints, and impact of temporal juxtaposition of algorithms on data allocations.

The ARS may incorporate fault-tolerant capability and allow the "concentration" of computational resources for subtasks of high priority in a real-time process. Also, the same ARS can be used for other applications (e.g., speech understanding).

This high-level model for the ARS provides a framework for ongoing research. The long term goal is to develop the techniques to calculate, represent, and use the dynamic information needed to implement the ARS.

Further details about automatic reconfiguration and information about other areas of system software for PASM can be found in papers listed in the "Operating System Aspects" section of the reading list. Aspects explored include: automatic system reconfiguration, image understanding environments, partition management, performance measurements, secondary storage service rate, synchronization techniques, system models, system reconfiguration, and task migration. The automatic reconfiguration overview above is summarized from [D-9].

6. Summary

The flexibility of partitionable mixed-mode systems makes the efficient execution of a wide-range of applications possible. However, the software challenges for partitionable mixed-mode systems are a superset of those for MIMD and SIMD systems.

This chapter overviewed some of the software issues being addressed by the PASM research team. The PASM prototype acts as a testbed for new software ideas, and much of the research can be extended to other systems. For instance, the ELP language is a general language for partitionable mixed-mode parallel systems that is being developed on the PASM prototype. Several algorithm studies implemented on the prototype (e.g., bitonic sequence sorting) have given insights into the interaction of general program characteristics with modes of parallelism. Consequently, SIMD versus MIMD trade-offs (e.g., CU/PE overlap) have been better understood. A thorough understanding of the effect of system configuration on an application's performance is necessary when designing a compiler and an operating system that will be mapping tasks onto reconfigurable machines. ARS was presented as a model for automatically assigning processors to subtasks to minimize the execution time of a task. The reading list at the end of the chapter references publications that provide more information on the subject of each section as well as related topics.

References

[1] F. A. Briggs, K. S. Fu, K. Hwang, and J. H. Patel, "PM^4 - a reconfigurable multiprocessor system for pattern recognition and image processing," *National Computer Conference*, pp. 255-265, June 1979.

[2] A. N. Choudhary, J. H. Patel, and N. Ahuja, "Architecture and performance evaluation of NETRA," in *Parallel Architectures and Algorithms for Image Understanding*, V. K. Prasanna Kumar, ed., Academic Press, New York, 1991.

[3] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, "Image Processing on SIMD/SPMD Architecture: OPSILA," *Ninth International Conference on Pattern Recognition*, pp. 430-433, Nov. 1988.

[4] R. F. Freund and D. S. Conwell, "Superconcurrency: a form of distributed heterogeneous supercomputing," *Supercomputing Review*, Vol. 3, pp. 47-50, Oct. 1990.

[5] G. J. Lipovski and M. Malek, *Parallel Computing: Theory and Comparisons*, John Wiley & Sons, New York, 1987.

[6] M. Philippsen, T. Warschko, W. F. Tichy, and C. Herter, "Project Triton: towards improved programmability of parallel machines," Technical Report, Universitat Karlsruhe, Jan. 1992.

[7] H. J. Siegel, S. Abraham, et al., "Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing," *Journal of Parallel and Distributed Computing*, Vol. 16, No. 3, Nov. 1992, to appear.

[8] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller Jr., H. E. Smalley Jr., and S. D. Smith, "PASM: a partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Transactions on Computers*, Vol. C-30, No. 12, pp. 934-947, Dec. 1981.

[9] T. Theoharis and I. Page, "Polygon rendering on a dual-paradigm parallel processor," *Computing & Graphics* Vol. 13, No. 2, pp. 207-216, 1989.

[10] Thinking Machines Corp., "The Connection Machine: CM-5 technical summary," Technical Report, Thinking Machines Corp., Jan. 1989.

Reading List for 1/86 to 9/92

Below is a reading list of PASM-related publications since 1986. For a list of PASM-related publications prior to 1986, please see reference [A-4].

A. Architectural Issues

[1] N. J. Davis IV and H. J. Siegel, "Performance Analysis of Multiple-Packet Multistage Cube Networks and Comparison to Circuit Switching," *1986 Int'l Conf. Parallel Processing*, pp. 108-114, Aug. 1986.

[2] T. Schwederski, W. G. Nation, H. J. Siegel, and D. G. Meyer, "The Implementation of the PASM Prototype Control Hierarchy," *2nd Int'l Conf. Supercomputing, Vol. I*, pp. 418-427, May 1987.

[3] M. Jeng and H. J. Siegel, "The Use of a Dynamic Redundancy Network to Enhance the Reliability of PASM," *2nd Int'l Conf. Supercomputing, Vol. I*, pp. 311-320, May 1987.

[4] H. J. Siegel, T. Schwederski, J. T. Kuehn, and N. J. Davis IV, "An Overview of the PASM Parallel Processing System," in *Computer Architecture*, edited by D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furht, IEEE Computer Society Press, Washington, D.C., pp. 387-407, 1987.

[5] G. B. Adams III, D. P. Agrawal, and H. J. Siegel, "A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks," *Computer*, Special Issue on Interconnection Networks for Parallel and Distributed Processing, Vol. 20, No. 6, pp. 14-27, June 1987.

[6] H. J. Siegel, W. T.-Y. Hsu, and M. Jeng, "An Introduction to the Multistage Cube Family of Interconnection Networks," *The Journal of Supercomputing*, Vol. 1, No. 1, pp. 13-42, 1987.

[7] M. Jeng and H. J. Siegel, "Design and Analysis of Dynamic Redundancy Networks," *IEEE Transactions on Computers*, Vol. C-37, No. 9, pp. 1019-1029, Sept. 1988.

[8] H. G. Dietz, T. Schwederski, M. T. O'Keefe, and A. Zaafrani, "Static Synchronization Beyond VLIW," *Supercomputing 1989*, pp. 416-425, Nov. 1989.

[9] H. J. Siegel, W. G. Nation, C. P. Kruskal, and L. M. Napolitano, Jr., "Using the Multistage Cube Network Topology in Parallel Supercomputers," *Proceedings of the IEEE*, Special Issue on Supercomputer Technology, Vol. 77, No. 12, pp. 1932-1953, Dec. 1989.

[10] H. J. Siegel, W. G. Nation, and M. D. Allemang, "The Organization of the PASM Reconfigurable Parallel Processing System," *1990 Parallel Computing Workshop*, pp. 1-12, Mar. 1990.

[11] 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. on the Frontiers of Massively Parallel Computation*, pp. 259-264, Oct. 1990.

[12] T. Schwederski, E. Bernath, G. Roos, W. G. Nation, and H. J. Siegel, "Fault Side-Effects in Fault-Tolerant Multistage Interconnection Networks," *1991 Int'l Conf. Parallel Processing, Vol. I*, pp. 313-317, Aug. 1991.

[13] M. T. O'Keefe and H. G. Dietz, "Static Barrier MIMD: Architecture and Performance Analysis," *Journal of Parallel and Distributed Computing*, accepted to appear.

B. Languages and Compiling Topics

[1] H. G. Dietz, M. T. O'Keefe, A. Zaafrani, "An Introduction to Static Scheduling for MIMD Architectures," in *Advances in Languages and Compilers for Parallel Processing*, edited by A. Nicolau, D. Gelertner, T. Gross, and D. Padua, The MIT Press, Cambridge, MA, pp. 425-444, 1991

[2] T. L. Casavant, H. G. Dietz, T. Schwederski, P. C.-Y. Sheu, and H. J. Siegel, "Software Plans for PASM," *2nd Int'l Conf. Supercomputing, Vol. I*, pp. 428-439, May 1987.

[3] T. L. Casavant, H. G. Dietz, P. C-Y. Sheu, and H. J. Siegel, "The PARSE Approach to Programming Non-Shared Memory, Reconfigurable, Parallel Computers," *4th Int'l Conf. Supercomputing, Vol. I*, pp. 380-389, May 1989.

[4] M. J. Phillip and H. G. Dietz, "Toward Semantic Self-Consistency in Explicitly Parallel Languages," *4th Int'l Conf. Supercomputing, Vol. I*, pp. 398-407, May 1989.

[5] A. Zaafrani, H. G. Dietz, and M. T. O'Keefe, "Static Scheduling for Barrier MIMD Architectures," *1990 Int'l Conf. Parallel Processing, Vol. II*, pp. 187-194, Aug. 1990.

[6] M. A. Nichols, H. J. Siegel, H. G. Dietz, R. W. Quong, and W. G. Nation, "Eliminating Memory Fragmentation within Partitionable SIMD/SPMD Machines," *IEEE Transactions on Parallel and Distributed Systems*, Special Issue on Parallel Languages and Compilers, Vol. 2, No. 3, pp. 290-303, July 1991.

[7] M. A. Nichols, H. J. Siegel, and H. G. Dietz, "Execution Mode Management and CU/PE Overlap in an SIMD/SPMD Parallel Language/Compiler," *15th Annual Int'l Computer Software and Applications Conf. (COMPSAC '91)*, pp. 392-397, Sept. 1991.

[8] M. A. Nichols, H. J. Siegel, and H. G. Dietz, "Data Management and Control-Flow Aspects of an SIMD/SPMD Parallel Language," *IEEE Transactions on Parallel and Distributed Systems*, accepted to appear.

[9] H. G. Dietz, M. T. O'Keefe, and A. Zaafrani, "Static Scheduling for Barrier MIMD Architectures," *The Journal of Supercomputing*, Vol. 5, pp. 263-289, 1992.

[10] M. T. O'Keefe and H. G. Dietz, "Loop Coalescing and Scheduling for Barrier MIMD Architectures," *IEEE Transactions on Parallel and Distributed Systems*, accepted to appear.

C. Algorithm Studies

[1] J. T. Kuehn and H. J. Siegel, "Simulation Based Performance Measures for SIMD/MIMD Processing," in *Evaluation of Multicomputers for Image Processing*, edited by L. Uhr, K. Preston, Jr., S. Levialdi, and M. J. B. Duff, Academic Press, Orlando, FL, pp. 139-158, 1986.

[2] T. A. Rice and L. H. Jamieson, "Scaling and Rotational Registration," in *Evaluation of Multicomputers for Image Processing*, edited by L. Uhr, K. Preston, Jr., S. Levialdi, and M. J. B. Duff, Academic Press, Orlando, FL, pp. 203-218, 1986.

[3] L. H. Jamieson, P. T. Mueller, Jr., and H. J. Siegel, "FFT Algorithms for SIMD Parallel Processing Systems," *Journal of Parallel and Distributed Computing*, Vol. 3, No. 1, pp. 48-71, Mar. 1986.

[4] L. H. Jamieson, H. J. Siegel, E. J. Delp, and A. Whinston, "The Mapping of Parallel Algorithms to Reconfigurable Parallel Architectures," *ARO Workshop on Future Directions in Computer Architecture and Software*, pp. 147-154, May 1986.

[5] 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, pp. 65-100, 1987.

[6] E. C. Bronson, J. T. Kuehn, and L. H. Jamieson, "Simulation of SIMD Signal Processing Algorithms on the PASM Parallel Processing System," *1986 Int'l Conf. Parallel Processing*, pp. 796-798, Aug. 1986.

[7] 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, Vol. I*, pp. 444-451, Aug. 1988.

[8] M. A. Yoder and L. H. Jamieson, "Simulation of a Word Recognition System on Two Parallel Architectures," *IEEE Transactions on Computers*, Vol. C-38, No. 9, pp. 1269-1284, Sept. 1989.

[9] H. J. Siegel, J. B. Armstrong, and D. W. Watson, "Mapping Tasks onto the PASM Reconfigurable Parallel Processing System," *1990 Parallel Computing Workshop*, pp. 13-24, Mar. 1990.

[10] E. C. Bronson, T. L. Casavant, and L. H. Jamieson, "Experimental Application-Driven Architecture Analysis of an SIMD/MIMD Parallel Processing System," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, pp. 195-205, Apr. 1990.

[11] G. B. Adams III, E. C. Bronson, T. L. Casavant, L. H. Jamieson, and R. A. Kamin III, "Experiments with Parallel Fast Fourier Transforms," in *Parallel Algorithms and Architectures for DSP Applications*, edited by M. A. Bayoumi, Kluwer Academic Publishers, Norwell, MA, pp. 49-75, 1991.

[12] S. A. Fineberg, T. L. Casavant, and H. J. Siegel, "Experimental Analysis of a Mixed-Mode Parallel Architecture Using Bitonic Sequence Sorting," *Journal of Parallel and Distributed Computing*, Vol. 11, No. 3, pp. 239-251, Mar. 1991.

[13] T. B. Berg and H. J. Siegel, "Instruction Execution Trade-Offs for SIMD vs. MIMD vs. Mixed-Mode Parallelism," *5th Int'l Parallel Processing Symp.*, pp. 301-308, May 1991.

[14] S.-D. Kim, M. A. Nichols, and H. J. Siegel, "Modeling Overlapped Operation Between the Control Unit and Processing Elements in an SIMD Machine," *Journal of Parallel and Distributed Computing*, Special Issue on Modeling of Parallel Computers, Vol. 12, No. 4, pp. 329-342, Aug. 1991.

[15] E. C. Bronson and L. H. Jamieson, "Experimental Verification of the Critical Path Simulation of an SIMD/MIMD Parallel Processing System," *1991 Int'l Conf. Parallel Processing, Vol. II*, pp. 113-116, Aug. 1991.

[16] T. B. Berg, S.-D. Kim, and H. J. Siegel, "Limitations Imposed on Mixed-Mode Performance of Optimized Phases Due to Temporal Juxtaposition," *Journal of Parallel and Distributed Computing*, Special Issue on Massively Parallel Computation, Vol. 13, No. 2, pp. 154-169, Oct. 1991.

[17] J. B. Armstrong, M. A. Nichols, H. J. Siegel, and L. H. Jamieson, "Examining the Effects of CU/PE Overlap and Synchronization Overhead when Using the Complete Sums Approach to Image Correlation," *Symp. Parallel and Distributing Processing*, sponsored by IEEE Computer Society and ACM, pp. 224-232, Dec. 1991.

[18] H. J. Siegel, J. B. Armstrong, and D. W. Watson, "Mapping Computer Vision Related Tasks onto Reconfigurable Parallel Processing Systems," *Computer*, Special Issue on Parallel Processing for Computer Vision and Image Understanding, Vol. 25, No. 2, pp. 54-63, Feb. 1992.

[19] W. G. Nation, A. A. Maciejewski, and H. J. Siegel, "Exploiting Concurrency Among Tasks in Partitionable Parallel Processing Systems," *6th Int'l Parallel Processing Symp.*, pp. 30-38, Mar. 1992.

[20] N. Giolmas, D. W. Watson, D. M. Chelberg, and H. J. Siegel, "A Parallel Approach to Hybrid Range Image Segmentation," *6th Int'l Parallel Processing Symp.*, pp. 334-342, Mar. 1992.

D. Operating System Aspects

[1] J. T. Kuehn and H. J. Siegel, "Multifunction Processing with PASM," in *Intermediate-Level Image Processing*, edited by M.J.B. Duff, Academic Press, London, pp. 209-229, 1986.

[2] D. L. Tuomenoksa and H. J. Siegel, "Determining an Optimal Secondary Storage Service Rate for the PASM Control System," *IEEE Transactions on Computers*, Vol. C-35, No. 1, pp. 43-53, Jan. 1986.

[3] T. Schwederski and H. J. Siegel, "Adaptable Software for Supercomputers," *Computer*, Special Issue on Design for Adaptability, Vol. 19, No. 2, pp. 40-48, Feb. 1986.

[4] T. Schwederski, H. J. Siegel, E. J. Delp, A. Whinston, and L. H. Jamieson, "Modeling the PASM Parallel Processing System," *SIAM 1986 Nat'l Meeting*, abstract, pg. A86, July 1986.

[5] T. Schwederski and H. J. Siegel, "Performance Measurements on the PASM Prototype," *IEEE/ACM Workshop on Instrumentation for Distributed Computing Systems*, pp. 49-50, Jan. 1987.

[6] C. H. Chu, E. J. Delp, and H. J. Siegel, "Image Understanding on PASM: A User's Perspective," *2nd Int'l Conf. Supercomputing, Vol. I*, pp. 440-449, May 1987.

[7] F. Weil, L. H. Jamieson, and E. J. Delp, "An Algorithm Database for an Image Understanding Task Execution Environment," in *High-Level Vision with Multicomputers*, edited by S. Levialdi, Academic Press, London, pp. 35-51, 1988.

[8] T. Schwederski, H. J. Siegel, and T. L. Casavant, "A Model of Task Migration in Partitionable Parallel Processing Systems," *Frontiers '88: The 2nd Symp. the Frontiers of Massively Parallel Computation*, pp. 211-214, Oct. 1988.

[9] 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," *Journal of Parallel and Distributed Computing*, Vol. 6, No. 3, pp. 598-622, June 1989.

[10] T. Schwederski, H. J. Siegel, and T. L. Casavant, "Task Migration Transfers in Multistage Cube Based Parallel Systems," *1989 Int'l Conf. Parallel Processing, Vol. I*, pp. 296-305, Aug. 1989.

[11] M. Jeng and H. J. Siegel, "A Distributed Management Scheme for Partitionable Parallel Computers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 1, pp. 120-126, Jan. 1990.

[12] J. E. Lumpp, Jr., T. L. Casavant, H. J. Siegel, and D. C. Marinescu, "Specification and Identification of Events for Debugging and Performance Monitoring of Distributed Multiprocessor Systems," *10th Int'l Conf. Distributed Computing Systems*, pp. 476-483, May 1990.

[13] D. C. Marinescu, J. E. Lumpp, Jr., T. L. Casavant, and H. J. Siegel, "Models for Monitoring and Debugging Tools for Parallel and Distributed Software," *Journal of Parallel and Distributed Computing*, Special Issue on Software Tools for Parallel Programming and Visualization, Vol. 8, No. 6, pp. 171-184, June 1990.

[14] T. Schwederski, H. J. Siegel, and T. L. Casavant, "Optimizing Task Migration Transfers Using Multistage Cube Networks," *1990 Int'l Conf. Parallel Processing, Vol. I*, pp. 51-58, Aug. 1990.

[15] F. J. Weil, L. H. Jamieson, and E. J. Delp, "An Analysis of Fixed-Assignment Hypercube Partitioning," *1990 Int'l Conf. Parallel Processing, Vol. I*, pp. 222-225, Aug. 1990.

[16] 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," *Communications of the ACM*, Vol. 34, No. 11, pp. 104-117, Nov. 1991.

[17] F. J. Weil, L. H. Jamieson, and E. J. Delp, "DISC: A Method for Dynamic Intelligent Scheduling and Control of Reconfigurable Parallel Architectures," *Journal of Parallel and Distributed Computing*, Vol. 13, No. 3, pp. 273-285, Nov. 1991.

[18] L. H. Jamieson, E. J. Delp, C.-C. Wang, J. Li, and F. J. Weil, "A Software Environment for Parallel Computer Vision," *Computer*, Special Issue on Parallel Processing for Computer Vision and Image Understanding, Vol. 25, No. 2, pp. 73-77, Feb. 1992.

[19] G. Saghi, H. J. Siegel, and J. A. B. Fortes, "On the viability of a quantitative model of system reconfiguration due to a fault," *1992 Int'l Conf. Parallel Processing, Vol. I*, pp. 233-242, Aug. 1992.