Efficient Masking Techniques for Large-Scale SIMD Architectures

Wayne G. Nation[*] wn@ecn.purdue.edu
Samuel A. Fineberg[**] safineberg@icaen.uiowa.edu
Mark D. Allemang[*] allemang@ecn.purdue.edu
Thomas Schwederski[***]
Thomas L. Casavant[**] tomc@eng.uiowa.edu
Howard Jay Siegel[*] hj@ecn.purdue.edu

[*]Parallel Processing Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907, USA

[**]Parallel Processing Laboratory
Dept. of Electrical and Computer Engineering
University of Iowa
Iowa City, Iowa 52242, USA

[***]Institute for Microelectronics Stuttgart
Allmandring 30a
D-7000 Stuttgart 80, West Germany

Abstract -- SIMD architectures require mechanisms that efficiently enable and disable (mask) processors to support flexible programming. Most current SIMD architectures employ special purpose (custom) processors incorporating masking logic that allow them to disable themselves based on data conditional results calculated at the processor's level (local masking). Global processor masks, specified by the control unit, are more efficient for tasks where the masking is data independent. An efficient hybrid masking technique is proposed that supports global masking as well as local masking for SIMD architectures constructed from standard microprocessors. A design for the proposed hybrid mechanism is described and its performance examined by experiments using the existing PASM prototype.

1. Introduction

Large-scale parallel processing systems, where the number of processors is in the range from 2^6 to 2^{16}, employing sophisticated microprocessors as the basic computational element are now feasible. When parallel processing systems are used in the SIMD (single instruction stream – multiple data stream) [17] mode of parallelism, where enabled processors simultaneously execute the same instructions on their own data sets, mechanisms that efficiently enable and disable (mask) the processors and ensure their synchronization are necessary to provide optimal performance. Most current SIMD architectures use special purpose processors incorporating synchronization and masking logic. This paper proposes a method for performing masking efficiently in large-scale SIMD parallel processing systems based on standard microprocessors. The results of this study and experimentation with existing hardware can be used to design SIMD architectures from either off-the-shelf or custom microprocessors.

Consider a system in which each *processing element* (*PE*) is composed of a processor/memory pair. A general SIMD system model in a *PE-to-PE configuration* consists of P PEs, a single control unit, and an interconnection network [28]. The PEs are numbered from 0 to P − 1, and each PE knows its own number. The *control unit* (*CU*) broadcasts the instructions that are executed by enabled PEs in lockstep.

The decision that a given PE is enabled to execute a set of one or more instructions is determined by two factors: the current location within a program and/or data dependent conditions arising on the PEs. *Global masking* is defined to be any masking operation where the set of enabled PEs is not a function of PE-resident data. It is static and known at compile time. *Local masking* is defined as any masking operation where the mask for a given PE is determined by PE-resident data (e.g., a PE i is enabled if the contents of PE i's variable A > 0). It is a dynamic process and the mask may not be known until execution time. In general, both schemes can perform masking based on the PE number (e.g., enable all even numbered PEs).

The CU is the obvious choice for specifying the enabled set of processors for global masking operations. The CU generally does not have direct access to PE data (although it can receive messages from the PEs). Thus, local masking operations may be best implemented if the PEs can enable or disable themselves. Algorithms exhibiting both forms of masking behavior executing on systems using only one of the masking types may not run as efficiently as possible because the system must emulate the form of masking not directly implemented. Most existing SIMD architectures employ mechanisms that implement either global masking or local masking, but not both. This paper proposes a hybrid masking scheme that directly implements both local and global masking, thus gaining the advantages of both while eliminating all emulation overhead.

A synthetic SIMD algorithm executing on an existing multi-microprocessor system (PASM) is used as a basis to experimentally examine the relative performance of different masking mechanisms. The proposed hybrid masking strategy is emulated on the PASM prototype to demonstrate its value. The results of this study are generally applicable to the design of microprocessor based SIMD architectures.

Another SIMD system model is the *processor-to-memory configuration* [28]. In the processor-to-memory configuration, the processors are not paired with memories but instead must access the memory modules through an interconnection network. The results of this paper are equally applicable to processor-to-memory SIMD systems.

Different masking techniques and their inherent advantages and disadvantages are discussed in Section 2. An efficient masking technique for SIMD multi-microprocessor architectures is detailed in Section 3. Section 4 presents the synthetic algorithm used to exercise and examine the masking techniques, and discusses the results of the experiments conducted on the PASM prototype, pointing out the advantages of the proposed masking mechanism.

This work supported by the National Science Foundation under grant numbers CCR-8704826, CCR-8809600, by the Air Force Office of Scientific Research under grant number F49620-86-K-0006, in part by the NSF Software Engineering Research Center (SERC), and by the Naval Ocean Systems Center under the High Performance Computing Block, ONT.

CH2908-2/90/0000/0259/$01.00 © 1990 IEEE

2. Masking Techniques for SIMD Machines

One form of global masking uses a P-bit *mask vector* where bit i = 1 means PE i is enabled and bit i = 0 means PE i is disabled. The Illiac IV [1, 10, 18], where P = 64 and the words were 64 bits long, had instructions to perform local and a form of global masking. The Illiac IV had an instruction that broadcast a 64-bit mask vector to all PEs from the CU and each PE selected its own bit to set its active/inactive status. For a massively parallel system (P $\geq$ 1000), this method is impractical due to the difficulty associated with the user generation and handling of the large bit string.

A form of global masking appropriate for massively parallel systems is PE-address masks [26, 28]. An enabled set of PEs can be specified by a p-position *PE-address mask*, $P = 2^p$, where each position is a 0, 1, or X (don't care) and superscripts are repetition factors. For example, for P = 64 (p = 6), those PEs whose numbers (in binary) match the mask $[0^4X^2]$ = [0000XX] are enabled (i.e., PEs 0, 1, 2, and 3). Even though PE-address masks can be used to specify only $3^{\log_2 P}$ of the total 2^P different subsets of the PEs, they are quite useful because the subsets of PEs to be activated in actual algorithms are generally regular in form, not arbitrary. For example, if 1024 PEs are logically arranged as a 32×32 two-dimensional array to operate on an image, a PE-address mask can be used to efficiently enable those PEs operating on the outer edge pixels: $[0^5X^5]$ for the top, $[1^5X^5]$ for the bottom, $[X^50^5]$ for the left side, and $[X^51^5]$ for the right side.

A PE-address mask can be specified in an easily decodable form with 2p bits. The 2p bits are broadcast to all PEs and simple comparator logic at each PE determines if the PE matches the mask. By adding one bit to the mask encoding, the scheme can be extended to include *negative PE-address masks*, which disable all PEs matching the mask. For the previous example, $[\neg 0^5X^5]$ disables the top row of PEs.

In systems using local masking, PE hardware can enable or disable the execution of an instruction stream based on some data condition. This PE hardware can consist of a single bit *local mask bit register* settable by the PE processor that dictates whether a stream of instructions will be executed or not. Typically, PEs disabled under this scheme execute instructions but are prevented from updating memory contents or registers (excluding their local mask bit register). This allows disabled processors to enable themselves. BLITZEN [9], CM-1 and CM-2 [33], DAP [19, 20, 24], MPP [6, 7, 22], MasPar [8], PEPE [13, 14, 34] and STARAN [3, 4, 5] use local masking but have no direct support for global masking. Illiac IV implemented local masking in addition to a form of global masking.

The local masking strategy is efficient when PEs execute instructions conditionally based on PE data values. If a system contains local masking hardware but no global masking hardware, local masking can be used to emulate global PE-address masking. The PE-address mask is decoded into a P-bit mask vector $M = m_{P-1}...m_1m_0$ and bit m_i of the mask vector M is loaded into the memory of PE i, $0 \leq i < P$. Before the instructions associated with the PE-address mask are broadcast to the PEs, all PEs are instructed to move their corresponding mask vector bit m_i from their memory into their local mask bit registers. This preload scheme can be efficient when PE-address masks are known a priori and there is enough PE memory to accommodate the mask vector bits associated with each of the PE-address masks. The cost would be only one load instruction for each masking operation. This becomes more complex if the PE-address mask is not "fixed," i.e., depends on runtime information. For example, if the PE-address mask $[X^{p-j}0^j]$ is in a loop and index variable dependent, a different mask vector is generated for each iteration of the j-loop.

3. Efficient Masking Mechanisms for SIMD Multi-Microprocessors

The proposed design facilitates global PE-address masking and local masking for massively parallel SIMD architectures. An added feature is the ability to execute PE instructions conditionally within a PE-address mask (i.e., a hybrid of local and global PE-address masking). The general SIMD multi-microprocessor architecture used as a model for the proposed hybrid masking strategy consists of a CU and PEs that are based on standard microprocessors but the results are directly applicable to custom processors.

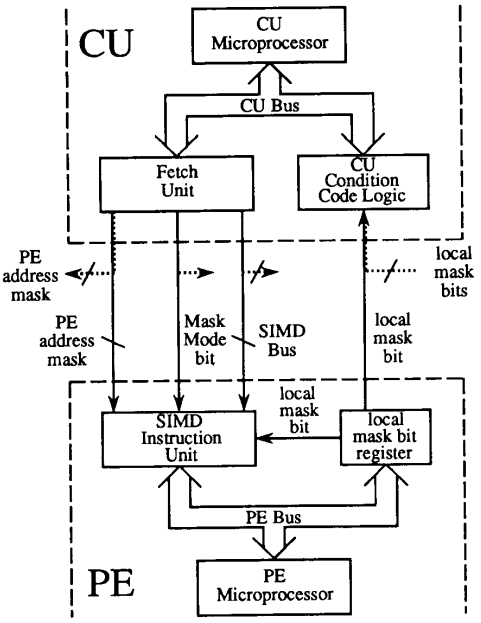

Figure 1: Proposed architectural support for hybrid masking.

Figure 1 shows the CU-PE connections and components of the system supporting instruction broadcast and PE masking. Dotted lines indicate connections to/from the other PEs. The *Fetch Unit* is responsible for broadcasting instructions to the PEs. Each of these instructions is accompanied by a PE-address mask and a *Mask Mode* bit. The CU commands the Fetch Unit to broadcast a block of PE instructions on the SIMD Bus. The CU also specifies the PE-address mask and Mask Mode bit associated with those instructions by writing their value into the Fetch Unit *Mask Register*. In general, the Fetch Unit may broadcast one word at a time directly to the PEs or may enqueue blocks of PE instructions into a FIFO queue, where they are broadcast as needed to the PEs.

The PE's *SIMD Instruction Units* (*SIUs*) interpret the Mask Mode bit as follows. If an SIMD instruction word is broadcast to the PEs and a given PE's SIU detects that the broadcast PE-address mask does not match its PE number, then that PE is disabled for that instruction word. If a PE's SIU receives an SIMD instruction word whose PE-address mask does match its PE number and the Mask Mode bit is cleared

(=0), the PE is unconditionally enabled for that instruction word (the local mask bit is ignored). However, if a receives an SIMD instruction word whose PE-address mask matches the PE number and the Mask Mode bit is set (=1), the PE is enabled for the instruction only if its local mask bit is also set. A PE's *local mask bit register* is set (or cleared) by the PE microprocessor to represent true (or false) data conditions present on the PEs.

To summarize, when the Mask Mode bit is cleared, the system is running as a pure global PE-address masking system. If the Mask Mode is set and the PE-address mask is always set to all X's (all PEs enabled), the system is running as a pure local masking system. If the Mask Mode bit is set and a PE-address mask is allowed to vary, the system is employing a hybrid of the the two masking strategies. No inefficient emulation of one scheme with another is necessary, the advantages of both masking techniques are exploited, and, furthermore, the combination of the two schemes provides additional capabilities (e.g., enabling all PEs on the top edge of a logical array ($[0^5X^5]$) where A > B inside that PE).

The proposed masking hardware configuration may be used in many different ways. For example, consider one way the SIU can be used to facilitate the implementation of nested if-then-else data conditional statements in an SIMD program, where the conditions are based on local PE data. The format is shown in Figure 2, where the "then-block" and/or "else-block" may contain if-then-else statements. Also, each PE instruction may be associated with an arbitrary PE-address mask R.

if (conditional involving local PE data)
    then "then-block" PE instructions
    else "else-block" PE instructions

Figure 2: Typical format of a data conditional statement.

Assume that each PE has a one-bit wide local mask bit stack that has the local mask bit register as its top-of-stack value and that all PEs initially set their local mask bit register (i.e., push a one on the local mask bit stack). Also, each PE can send instructions to its memory-mapped SIU to manipulate the local mask bit stack. The three instruction are "push-then-else," "push-then," and "pop." In [12], similar operations are described. The "push-then-else" instruction has an immediate one-bit operand that is the value of the conditional. The "push-then-else" instruction commands the SIU to copy (not remove) the top of the local mask bit stack into "temp," logically AND "temp" with the complement of the one-bit operand and push the result onto the local mask bit stack. Then, "temp" is logically ANDed with the one-bit operand and the result is pushed onto the local mask bit stack. The "push-then" instruction logically ANDs its one-bit operand with the top-of-stack and pushes the result. "Push-then" is used when the data conditional statement has no "else-block." Finally, the "pop" instruction commands the SIU to pop and discard the top-of-stack value.

To implement the data conditional statement of Figure 2, the CU must perform the following sequence of actions:

(1) set the Fetch Unit Mask Register to the arbitrary PE-address mask R with the Mask Mode bit cleared;
(2) command the Fetch Unit to broadcast the PE instructions that perform the condition followed by the "push-then-else" instruction that writes the true/false result to the SIU;
(3) set the Mask Mode bit of the Fetch Unit Mask Register and command the Fetch Unit to broadcast the "then-block" instructions;
(4) clear the Mask Mode bit of the Fetch Unit Mask Register and command the Fetch Unit to broadcast the "pop" instruction to the PEs;
(5) set the Mask Mode bit of the Fetch Unit Mask Register and command to Fetch Unit to broadcast the "else-block" instructions;
(6) clear the Mask Mode bit of the Fetch Unit Mask Register and command the Fetch Unit to broadcast the "pop" instruction to the PEs.

All or a subset of the above instructions can be loaded into the Fetch Unit queue before any of the steps are executed by the PEs (i.e., the CU does not have to wait for the PEs to execute one step before any subsequent steps are moved into the queue). Thus, there is no overhead due to PE-to-CU synchronization. The added execution of PE instructions needed to control the local mask bit register are minimal. Also recall that the data conditional statement was executed efficiently under a PE-address mask R; this combination of local and global masking is more flexible than SIMD machines with pure local masking strategies.

The steps above for implementing nested data conditional statements under an arbitrary PE-address mask R can be directly extended to support the nesting ("scoping") of PE-address masks within nested data conditional statements. Just as the PEs manage local mask bit stacks, the CU can manipulate a PE-address mask stack that maintains a stack of CU-specified "scopes" [12].

In Figure 1, the CU has *CU Condition Code Logic* hardware. This hardware is necessary for the CU to efficiently gather data conditional information from its PEs to alter program flow (e.g., "repeat until A > 0 in all enabled PEs").

4. Experimental Evaluation

In this section, the proposed hybrid masking technique is examined experimentally. The vehicle for this evaluation is a modified SIMD bitonic sequence sorting algorithm programmed in assembly language on *PASM,* a partitionable SIMD/MIMD parallel processing system designed to support up to 1024 PEs [30, 31]. A 30-processor prototype with 16 MC68000 based PEs in the parallel computation unit was constructed at Purdue and is being used for a variety of parallel processing studies (e.g., [11, 16]).* The PEs are interconnected by a multistage cube type of network [27, 29]. For this study, PASM was used as a single SIMD machine (i.e., its partitioning and MIMD capabilities were not used).

Bitonic-based ordered merging is an appropriate platform for testing masking techniques because of its inherent use of global and local masking strategies. The algorithm is especially appropriate because the average size of PE instruction blocks is very small and there are frequent data conditional statements, making the performance very sensitive to any added overhead in manipulating data conditional information. Thus, it is the computational characteristics of this algorithm that are important here; the fact that it happens to sort (and how well it compares to other approaches to sorting) is irrelevant in this con-

* The prototype currently does local masking by having the CU read a bit from each PE, forming a global mask vector, and sending bit i to PE i. This approach to local masking was taken for hardware simplicity and is not used in this study.

text. The algorithm is simply a vehicle for evaluating SIMD masking strategies; this paper does not attempt to explain why the algorithm sorts [see 15, 16].

Batcher's bitonic sorting method is described in [2, 21, 23, 27, 32]. For the experimental examination of the different masking techniques, it is assumed that the number of items to be sorted is $N = 2^n$, $N > P$, and each PE initially contains a sorted list of N/P data items, each of which is an (N/P)-element subset of N unsigned uniformly distributed random 8-bit integers. The elements within a subset are sorted in advance and placed in a PE's memory in ascending order. The algorithm shown in Figure 3 is executed by all enabled PEs in the SIMD machine. At the algorithm's completion, the N/P data items in each PE are in ascending order within and across PEs (i.e., if data item i is in PE x and data item j is in PE x+1 then $i \leq j$, $0 \leq x < P-1$).

Each PE processor includes a *data transfer register* (*DTR*) and a register containing the PE's address (*ADDR*). ADDR(i) denotes the i-th bit of ADDR. When an *interconnection function* f is executed, the contents of the DTR of PE i are copied into the DTR of PE f(i), for all i, $0 \leq i < P$, simultaneously. A Cube network will be defined using $b_{p-1}...b_1b_0$ as the binary representation of an arbitrary PE-address and $\overline{b_i}$ as the complement of b_i. The *Cube* network consists of p functions: $cube_i(b_{p-1}...b_1b_0) = b_{p-1}...b_{i+1}\overline{b_i}b_{i-1}...b_0$, for $0 \leq i < p$ [26, 28].

```
for j = 1 step +1 until p do
    if j = p or ADDR(j) = 0 then type ← 0 else type ← 1
    for i = j−1 step −1 until 0 do
        for q = 1 step +1 until N/P do
            DTR = X[q]
            cube_i
            Y[q] = DTR
        merge (X,Y)
        if (type ≠ ADDR(i)) then swap(X,Y)
```

Figure 3: Algorithm for sorting sorted lists.

Referring to Figure 3, in each PE, the portion of the list to be sorted is held in an N/P element array X. For the "q" loop, each PE α receives a copy of the X array of PE $cube_i(\alpha)$, and stores it in Y. Then, in each PE, merge(X,Y) merges the sorted lists X and Y into the sorted list $X \cup Y$ and places the lesser half in X and the greater half in Y. This is a simple O(N/P) "merge" routine for merging two sorted lists requiring O(N/P) if-then-else data conditional operations.

After completing the "merge," the PEs use swap(X,Y) to swap the two lists (pointers) depending on each PE's value of "type" and "ADDR." The global PE-address masks required for the "swap" routine are precomputed and the second line of Figure 3 is never executed.

The code for the algorithm was altered to produce three different versions: (1) an SIMD/hybrid version, (2) an SIMD/no-CC version, and (3) an SIMD/local version. The SIMD/hybrid version emulates the required actions assuming the proposed hybrid masking strategy is incorporated in an off-the-shelf microprocessor based system where the manipulation of the local mask bit register and SIU is through memory-mapped I/O. In this version, for conditional statements dependent on PE data (e.g., in the merge() routine), the CU emulates setting the Mask Mode bit when broadcasting the "then" and "else" instruction blocks for execution by the PEs and the PEs emulate the execution of the instruction necessary to manipulate the local mask bit register. The emulation is done by the CU writing to a dummy memory location when setting the Mask Mode bit and the PEs manipulating a dummy memory location instead of the SIU. The PEs enabled by the PE-address mask ignore the Mask Mode bit (because there is no such hardware in place) and execute both the "then" and the "else" instructions. This version does not sort correctly, but the execution times are accurate.

The SIMD/no-CC version is an emulation of the algorithm execution time assuming changing the local mask bit register in the hybrid masking mechanism has zero time cost, indicating a lower bound on the execution time. This version assumes a custom PE design where every "push-then," "push-else," and "pop" can be replaced by an internal register access that is incorporated into the instruction set. Thus, there is no need for the memory access cycle to manipulate the local mask bit register and SIU as in the off-the-shelf CPU approach for the SIMD/hybrid version.

The SIMD/local version emulates the actions required assuming a pure local masking strategy. The PE-address mask is ignored (set to all X's and never altered). The updating of the Mask Mode bit and the local mask bit register for data conditional masking is emulated as in the SIMD/hybrid version. The execution time for performing a global PE-address mask operation using local masking is measured by emulating a memory load of a precomputed mask vector bit into the local mask register as described in Section 2.

Timings were made using the PASM prototype's internal performance timers (MC68230-based). The execution times shown in Figures 4 and 5 were measured for $N = 2^n$, $5 \leq n \leq 9$, with $P = 16$. The relative times, not the absolute times, are what are relevant to this study. As far as the different versions of the program compared in Figures 4 and 5 are concerned, the only difference between the SIMD/no-CC and SIMD/hybrid versions is that the SIMD/no-CC does not execute PE instructions to manipulate the local mask bit register. Recall that the SIMD/no-CC version emulates a "zero cost" condition code logic. Therefore, the time difference between these two versions is directly related to the memory access required to manipulate the local mask bit register.

The execution time of the SIMD/local version in Figure 4 appears coincident with the SIMD/hybrid version because it is

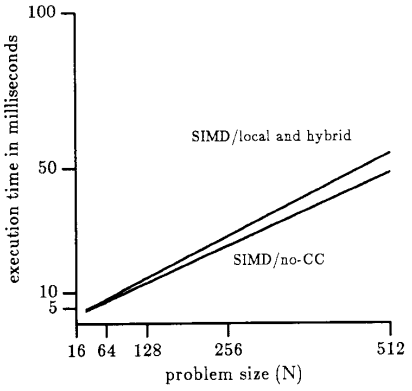

Figure 4: Local masking dominates global masking.

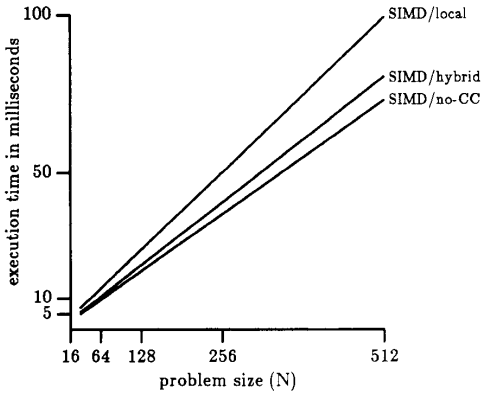

Figure 5: Balanced local and global masking.

only greater than the SIMD/hybrid results by approximately 2%. In general, the algorithm is dominated by local masking operations and the hybrid masking scheme does not perform much faster than the local scheme as a result. The "swap" is the only part of the algorithm that uses global PE-address masking operations. In the algorithm, the swap and merge routines are each called $O(p^2)$ times. O(1) global PE-address masking operations are performed on each call to swap, and O(N/P) local masking operations are performed on each call to "merge." Thus, $O(p^2)$ global PE-address masking operations and $O(p^2N/P)$ local masking operations are performed.

Not all algorithms are dominated by local masking and, thus, there are algorithms that should execute faster on machines supporting hybrid masking. As an illustration, consider a synthetic modified bitonic sorting algorithm where the "swap" operation is moved into the inner q loop. This new synthetic algorithm no longer has any functionality, other than to exercise the prototype in a certain way. The synthetic algorithm contains a more balanced mixture of masking types: $O(p^2N/P)$ local masking operations and $O(p^2N/P)$ global PE-address masking operations. As seen in Figure 5, the synthetic algorithm performs better on the hybrid system than on the local system. This disparity is even more evident for applications where global PE-address masking operations dominate local masking operations. While Figure 5 is based on a synthetic algorithm, it is a representation of computational situations where there is a balance between global and local masking, and is useful for the purposes of this study.

5. Conclusion

This paper has presented an efficient masking technique for large-scale SIMD architectures. This masking technique implements a hybrid of the two masking strategies inherent in SIMD algorithms: global masking, where the CU sets an explicit Global Mask independent of PE data, and local masking, where PEs enable and disable themselves based on conditions dependent on PE resident data.

Most of the constructed SIMD systems mentioned in Section 2 use local masking techniques, but not global. This work has shown how a hybrid of the two masking techniques can be supported in SIMD architectures using standard off-the-shelf or custom-designed microprocessors. The hybrid masking technique was detailed in Section 3.

Experience with the PASM prototype is the basis for this study and it was on the PASM prototype that the hybrid masking technique was examined through the emulation of the various masking strategies for the purpose of comparison (Section 4). This examination centered on a synthetic algorithm that exhibits both global and local masking characteristics. It was seen that the hybrid masking technique can increase the utilization of PEs and thus increase performance, the degree of improvement being algorithm dependent. The SIMD/no-CC results demonstrate that the hybrid technique used with off-the-shelf microprocessors can operate at near the performance of custom processors that internally incorporate some degree of hybrid masking support. This study also shows how particular details of the PASM prototype implementation can be stripped away to make the results of this research more generally applicable to any SIMD architectures built from off-the-shelf microprocessors.

The proposed hybrid masking scheme provides a highly efficient environment for performing masking operations over a range of applications on massively parallel SIMD systems. The results of this work should aid system designers in constructing these architectures. The results may also be used by system architects planning custom processors that will incorporate both modes of masking.

References

[1] 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.

[2] K. E. Batcher, "Sorting networks and their applications," *AFIPS 1968 Spring Joint Computer Conf.,* 1968, pp. 307-314.

[3] K. E. Batcher, "STARAN parallel processor system hardware," *AFIPS 1974 Nat'l Computer Conf.,* May 1974, pp. 405-410.

[4] K. E. Batcher, "The multidimensional access memory in STARAN," *IEEE Trans. Computers,* Vol. C-26, No. 2, Feb. 1977, pp. 174-177.

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

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

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

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

[9] D. W. Blevins, E. W. Davis, R. A. Heaton, and J. H. Reif, "BLITZEN: a highly integrated massively parallel machine," *J. Parallel and Distributed Computing,* Vol. 8, No. 2, Feb. 1990, pp. 150-160.

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

[11] 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 Computing,* Vol. 1, No. 2, Apr. 1990, pp. 195-205.

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

[13] B. A. Crane, M. J. Gilmartin, J. H. Huttenhoff, P. T. Rux, and R. R. Shively, "PEPE computer architecture," *IEEE Computer Soc. Compcon 72,* Sept. 1972, pp. 57-60.

[14] P. H. Enslow, Jr., "Appendix A: Parallel Element Processing Ensemble PEPE," in *Multiprocessors and Parallel Processing,* John Wiley and Sons, New York, NY, 1974, pp. 139-149.

[15] S. A. Fineberg, T. L. Casavant, and H. J. Siegel, "Experimental analysis of communication/synchronization aspects of a mixed-mode parallel architecture via synthetic computations," *Supercomputing '90,* to appear, Nov. 1990.

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

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

[18] R. M. Hord, *The Illiac IV, the First Supercomputer,* Computer Science Press, Rockville, MD, 1982.

[19] D. J. Hunt, "The ICL DAP and its application to image processing," in *Languages and Architectures for Image Processing,* M. J. B. Duff and S. Levialdi, eds., Academic Press, London, England, 1981, pp. 275-282.

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

[21] D. E. Knuth, *The Art of Computer Programming: Vol. 3 Sorting and Searching,* Addison-Wesley, Reading, MA, 1973.

[22] J. L. Potter, "MPP architecture and programming," in *Multicomputers and Image Processing: Algorithms and Programs,* K. Preston and L. Uhr, eds., Academic Press, New York, NY, 1982, pp. 275-290.

[23] M. J. Quinn, *Designing Efficient Algorithms for Parallel Computers,* McGraw-Hill, New York, NY, 1987.

[24] S. F. Reddaway, "DAP - A distributed array processor," *1st Symp. Computer Arch.,* Dec. 1973, pp. 61-65.

[25] 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.

[26] 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.

[27] H. J. Siegel, "Partitionable SIMD computer system interconnection network universality," *16th Allerton Conf. Communication, Control, and Computing,* Oct. 1978, pp. 586-595.

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

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

[30] 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.

[31] 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.

[32] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Computers,* Vol. C-20, No. 2, Feb. 1971, pp. 153-161.

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

[34] C. R. Vick and J. A. Cornell, "PEPE architecture - present and future," *AFIPS 1978 Nat'l Computer Conf.,* June 1978, pp. 981-992.