Associative, Multiassociative, and Hybrid Processing*†
(Extended Abstract)

Martin C. Herbordt‡ Charles C. Weems

Department of Computer Science
University of Massachusetts

Abstract: Multiassociative processing is the simultaneous associative processing of sets of elements. In the first part of this extended abstract, associative and multiassociative processing are defined in three different ways: 1) in terms of their characteristic primitive operations, 2) by the types of queries they are suited to answer, and 3) through their virtual machine models and instruction sets. The second part consists of a comparison of the performance of a generic, spatially-mapped, function on realizations of four different models of computation: the RAM, PRAM, associative processing, and multiassociative processing. For this class of functions, multiassociative processing is found to be superior in many cases and associative processing superior in the rest. The final part of this extended abstract contains the most significant and practical result: an analysis of the common set of function instances for which hybrid associative/multiassociative processing is superior. In particular, static and dynamic methods are presented for determining optimal partitions of these computations into associative and multiassociative phases.

*This paper also appears in the Proceedings of the 2nd Associative Processing and Applications Workshop.

†This work was supported in part by the Defense Advanced Research Projects Agency under contract DACA76-89-C-0016, monitored by the U.S. Army Engineer Topographic Laboratory; under contract DAAL02-91-K-0047, monitored by the U.S. Army Harry Diamond Laboratory; and by a CII grant from the National Science Foundation (CDA-8922572). Authors' address is Department of Computer Science; University of Massachusetts; Amherst, Massachusetts 01003; NetAd : herbordt@cs.umass.edu, weems@cs.umass.edu

‡M.C. Herbordt is supported in part by an IBM Fellowship.

1 Introduction

Multiassociativity is an additional level of parallelism: it is a flexible methodology for solving multiple problem instances simultaneously, and each using parallel/associative processing. In its most general form, multiassociativity resembles nested parallelism [1] to depth 2. However, it has the advantage of being based solely on primitives that can be implemented efficiently (using available hardware with some software emulation), while yet being general enough to be useful as both an algorithmic and a programming paradigm.

Multiassociative processing was introduced in [4] and has already found a place in the spatially mapped phases of machine vision computation. The performance is especially good for classes of problems that are characterized by the need for processing non-uniform, but proximate, data sets. It is also likely that multiassociativity will find more applications within machine vision, in other machine perception domains, and in for other tasks that use spatially mapped data.

We begin this paper by presenting the globally and multiassociative models of computation. We then discuss a generic function that subsumes many tasks in spatially mapped computation with respect to these associative processing paradigms, as well as to the standard serial and parallel computational models. We find the specific function instances for which the different models have superior performance, when hybrid algorithms are of use, and how this can be determined dynamically. We close with a brief description of some sample algorithms.

2 Global and Multiassociativity

The prototypical associative memory operation consists of the controller broadcasting a key to the cell array and then performing some action on those cells where a match occurs. The actions typically consist of reading (or writing) some value from (or into) the cell; receiving a response in the form of a SOME/NONE (global OR of the response tag bits) or a COUNT (of tag bits); or selecting a single responder for further processing. For example, the controller

may execute the instruction, "How many cells have variable GREEN set to TRUE?" or "All elements with BROWN set to TRUE set SKY to FALSE." The basic associative operations are summarized below (after [3, 13]).

1. Global broadcast/local compare/activity control
2. Some/None of responders from array to controller
3. Count of responders from array to controller
4. Select a single responder

In multiassociative processing, we add the concept of a *set*, which we define as a collection of elements that share some property. Set properties are typically either an *a priori* distinguishing characteristic of the elements, say that GREEN = TRUE, or an attribute of the set itself, say, that it contains over 100 elements. The fundamental principle of multiassociativity is to replace some of the function of the controller with a subset (often a single element) of the members of each set. The basic multiassociative operations are summarized below (after [4]).

1. Within each set: broadcast by a subset/local compare/activity control
2. Within each set: Some/None of responders to a subset
3. Within each set: Count of responders to a subset
4. Within each set: select a single responder
5. Split/Merge sets
6. Transfer data between sets

The first four operations are analogous to their globally associative counterparts, but are executed simultaneously in all selected sets. Of course sets are only useful if elements can added and removed with some efficiency. This is dealt with in the last two operations which together enable dynamic creation of sets and the implementation of divide-and-conquer algorithms.

The following is a sample multiassociative instruction sequence:

1. Each set of elements with the same COLOR selects a
   single element to be the accumulator.
2. Each set of elements with the same COLOR send a count of the
   number of elements to their accumulator.

This sequence might be followed by a globally associative routine to obtain a histogram of the largest sets:

While Count (SetAccumulator = TRUE $\wedge$ Count $>$ 500 $= 0$
1. SelectFirst(SetAccumulator = TRUE)
2. Read(Count)
3. Read(Color)
4. Write(FALSE $\rightarrow$ SetAccumulator)

In order to present a more formal description of global and multiassociativity, we define assembly language level machine models for associative and multiassociative memory processors. In both cases, the associative memory cells consist of some number of bits that are not distinguished in terms of function: that is, any cell location can be referenced in any cell operand.

The associative instruction consists of five fields as shown below:

1. OPCODE – Operations are: Compare Read Write Count SelectFirst
2. OPERAND 1 – Controller operand
3. OPERAND 2 – Cell operand
4. ACTIVITY – Indicates cells taking part in the computation
5. RESULT – I/O for some instructions

Field 2 is a value generated by the controller and broadcast to the array. Fields 3-5 are cell locations. As an example, the Compare instruction compares the data broadcast by the controller as indicated by OPERAND 1 with the data given by OPERAND 2 in each cell whose bit pointed to by ACTIVITY is on. If the data match, then the cell location in the RESULT field is set to TRUE.

The multiassociative instruction word consists of seven fields:

1. OPCODE – Operations are: Compare Read Write Count SelectFirst
2. PARTITION – Partitions multiassociative memory
3. OPERAND 1 – Cell operand 1
4. OPERAND 2 – Cell operand 2
5. ACTIVITY 1 – Indicates cells taking part in the computation as operand 1
6. ACTIVITY 2 – Indicates cells taking part in the computation as operand 2
7. RESULT – I/O for some instructions

In this case, all fields besides the OPCODE are cell locations. The PARTITION field divides the memory into sets: cells with identical values are members of the same set. Two activity fields are needed: one to indicate which cells are senders and one for the receivers. As an example, the Compare instruction is now applied in parallel to each set as given by the PARTITION location. Within each set, cells with ACTIVITY 1 turned on distribute the values at location OPERAND 1 to cells with ACTIVITY 2 turned on. If there are multiple writers, then the value becomes the bitwise logical OR of the send values. The readers compare the received value with their own value at OPERAND 2 and set RESULT according to whether there is a match.

Multiassociative *processing* extends the concept of multiassociative memory in the same way that associative processing extends associative memory. Additional computing capability is added to each cell to enable the generation of symbolic tags to constrain further processing. Rather than forcing an operand to be an existing value, it can be the result of a complex computation. A simple example of associative processing is "All elements with $G > 128$ and $R < 128$ and $B < 128$ set GREEN to TRUE." An example of a more complex multiassociative routine is "All elements with the same color that are contiguous form connected components."

3 A Generic Function Template

In order to discuss a real implementation of a subset of multiassociative processing, we restrict our attention to a generic function of the type that frequently arises in spatially mapped vision computation. This function will be used to compare global and multiassociative

Model	Complexities of Basic Operators		
	Count	Update	Select
Serial	$O(N)$	$O(N)$	$O(\lvert\text{total number selected}\rvert)$
Parallel	$O(\log N)$	$O(\log^2 N)$	$O(\log^3 N)$
Associative	$O(\lvert S\rvert)$	$O(\lvert S\rvert)$	$O(\lvert S\rvert \log N)$
Multiassociative	$O(\log N)$	$O(1)$	$O(\log N)$

Table 1: Big-O complexities of basic operators on various computational models. $|S|$ denotes the number of sets into which the array has been partitioned. It is assumed that at least one set has $O(N)$ elements, which is the worst case for the parallel and multiassociative models.

processing with each other and with the familiar serial and parallel processing paradigms.

Pixels are mapped to PEs. Those that are contiguous and have certain values in common are formed into regions. These regions are then processed using some number each of three basic operators: SelectSubset, Characterize (often a Count of PEs within a component that have a certain property), and UpdateSet. Depending on the task and the processing model, some preprocessing might be helpful to set up communication paths. Examples of tasks that use the generic function template are region-parallel versions of Count, Histogram, FindMedian, FindMean, SelectConvexHull, and many others from computational geometry.

4 Performance of Basic Operators

In this section we use big-O notation to make broad comparisons. For the parallel and multiassociative cases, it is assumed that there is at least one set (region) with $O(N)$ elements, a likely occurence in practice. The results are summarized in Table 1.

Serial Processing Model: In the serial model, processing time depends not on the size of the sets, but rather on the number of elements in the entire array that are involved in each computation. SelectSubset requires $O(|\text{total elements selected}|)$ operations, while the other two operators require that all elements be examined and so have $O(N)$ complexity.

Parallel Processing Model: In the parallel model we assume a controller, PE array with N processors, and a global OR circuit. The best possible performance for combining routing networks with a non-trivial number of processors is $O(\log N)$. We assume that all sets are

processed simultaneously. The best way to execute the generic function is to select a PE per set to be the accumulator. Once this has been done, Count takes one combining communication operation, or $O(\log N)$. A region-parallel Update uses a similar propagation method as a connected components labeling algorithm: it requires $O(\log N)$ communication operations for a total complexity of $O(\log^2 N)$ [9]. Select uses Update $\log N$ times (using the algorithm in [2]) and so has $(\log^3 N)$ complexity.

Global Associative Model: In this model sets are necessarily processed serially. The basic operations have the following complexity *per set*: Select is $O(\log N)$, Count and Update are $O(1)$. If $|S|$ is the number of sets, the total complexities are then $O(|S| \log N)$, $O(|S|)$, and $O(|S|)$ respectively. The complexities are derived in [13].

Multiassociative Model: Here, as in the parallel model, sets are processed simultaneously. The Count and Select operations are each $O(\log N)$ while the update operation is $O(1)$. These values are justified by results in [8, 4, 7].

5 Performance of the Generic Function

If we are to assume bounded size arrays, the big-O results of the previous section cannot be taken entirely at face value. But when taken together with empirical results (not included here) two general conclusions can be made. The first is that multiassociative processing is faster than parallel processing as long as Count operations do not dominate. The second is that global associative processing is superior to both parallel and multiassociative processing when $|S|$ is small, but inferior when $|S|$ is large.

To compare global and multiassociative models in greater detail, we must determine the constants within the big-O and examine their performance as several parameters are varied. These are the proportion of Counts, Updates, and Selects in the generic function instantiation; the input image (upon which the number of sets depends); and the choice of multiassociative Count algorithm. The last choice is significant because there exist several algorithms that are not asymptotically optimal, but which are fast in practice.

We do not go into great detail here, but the following results give a flavor for the relative performance of the models. Timings for the basic operators on a 256×256 CAAPP

Model	Timimg of Basic Operators		
	Count	Update	Select
Associative	$5\|S\|$ overhead, $2\|S\|$ per Count	$5\|S\|$	$2\|S\|$
Multiassociative	5000	20	20

Table 2: Estimated times in microseconds for basic template function operations on a 256×256 CAAPP. $|S|$ is the number of sets in the array. Multiassociative algorithms represented are data independent.

are given in Table 2. Graphs giving the behavior of functions with different proportions of operators are given in Figure 1. The multiassociative Count procedure used is a largely data-independent implementation of the general communication operation [4] and thus gives consistent, though suboptimal, performance.

6 Hybrid Algorithms

The generic function we have been examining computes a function F for each of a number of sets s_i in an array. We make the following observations:

- If we apply the resources of the entire array, including those of the controller, to compute F for *any particular set* s_i of S, then a result will almost certainly be obtained more quickly than if F is computed using only those PEs to which s_i is mapped.
- In the case where F is calculated multiassociatively for all s_i of S, there is often a wide variation in elapsed time for the different s_i's. Such a distribution is shown in Figure 2.

From these observations it follows that there may be some function/partition combinations for which a hybrid of a globally associative algorithm A_G and a multiassociative algorithm A_M may be preferable to either by itself.

Algorithm Hybrid-GENERIC
DO an optimal number (O) TIMES
1. Execute an iteration of A_M.

DO UNTIL F is done for all sets
2. Use GlobalSelectSingleResponder to select a set s_i from S, the set of sets for which A_M did not run to completion.

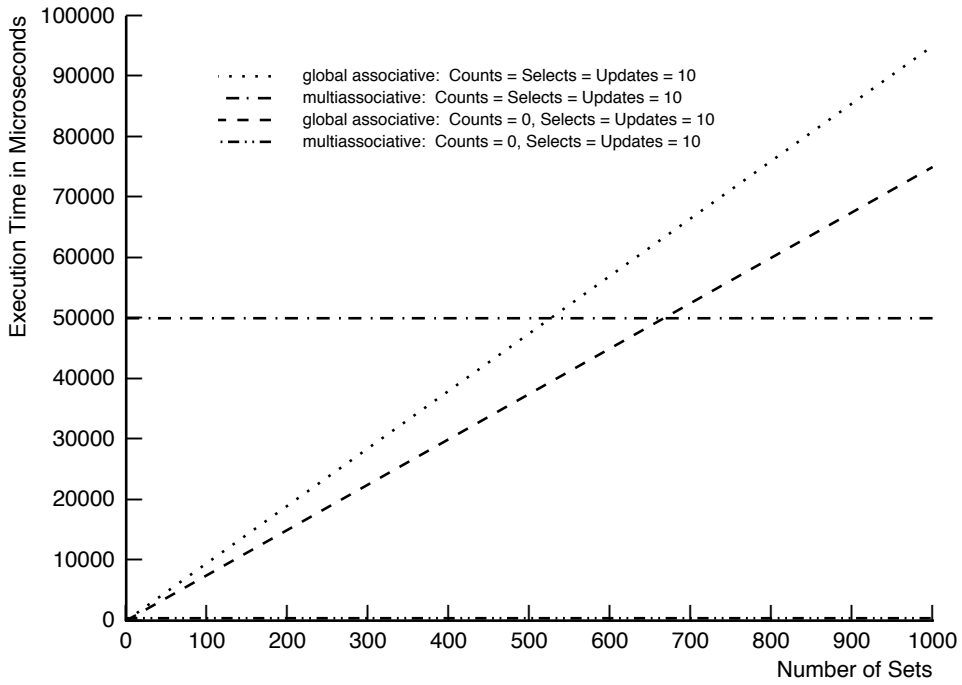

Figure 1: Plot of performance of global and multiassociative instantiations of the generic function versus the number of sets in the image. Times from Table 2 are assumed. For reference, during the early stages of region-merging segmentation algorithms, there are commonly several thousand regions (sets) to be processed.

3. Use A_G to process s_i.
4. Remove s_i from S.

We refer to the execution of the first loop as local removal and the second as global removal. To get some intuition as to what O should be and how it can be determined dynamically, we show graphically what happens to the distribution of the remaining sets during local and global removal (see Figure 3). Global removal is roughly equivalent to reducing the area under the graph by one unit per iteration; the graph of the expected new distribution is generated by reducing the height of the graph at each point by a constant fraction of the height at that point. Local removal is equivalent to moving the Y-axis to the right one unit per iteration.

In general, the optimal value for O minimizes the following expression:

$$K*O + \int_O^\infty dist(t)\,dt,$$

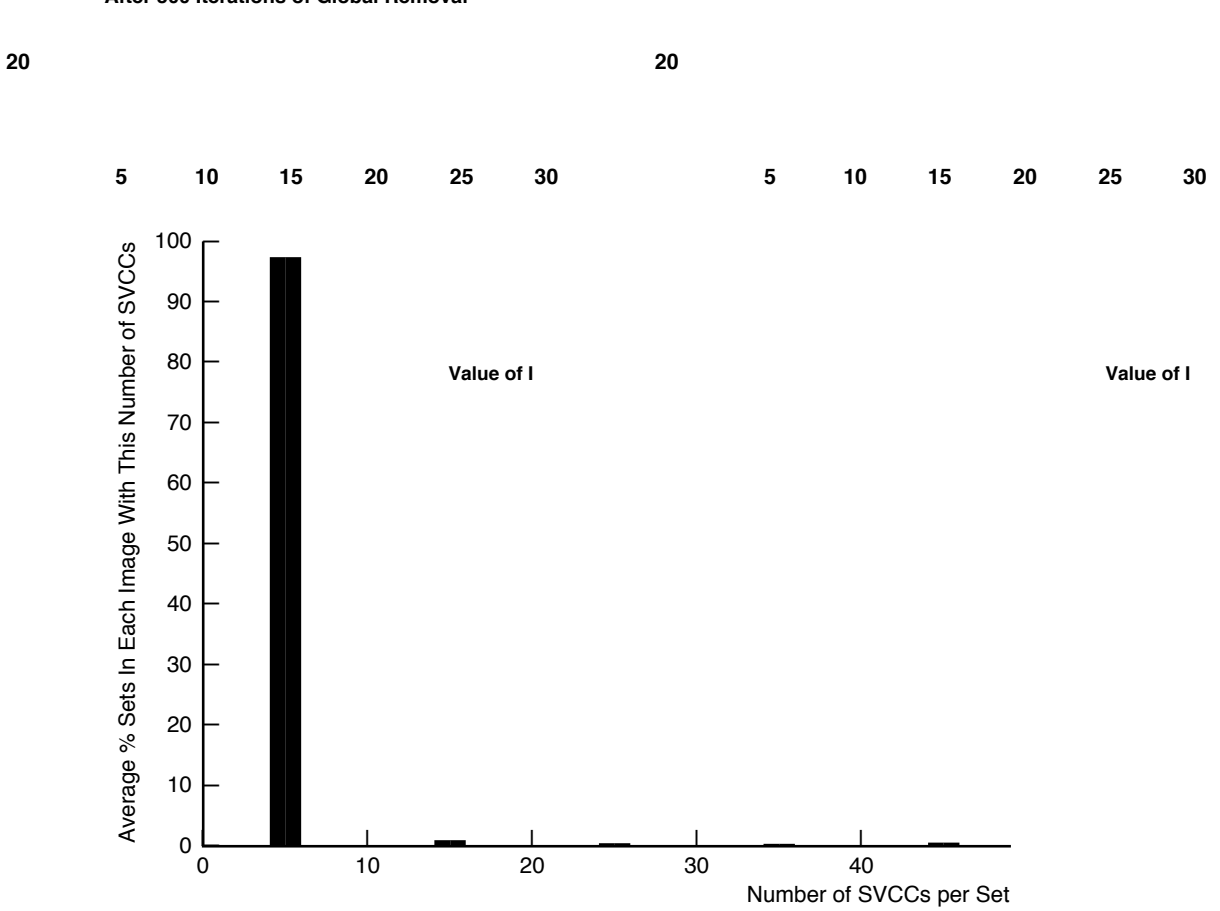

Figure 2: This histogram, taken from work on a multiassociative count algorithm, shows the fraction of sets that fall within given intervals of SVCC counts. The significance is that the performance of the algorithm in a set is proportional to the number of SVCCs in that set. The histogram shows that most sets tend to be well-behaved, but a significant fraction are not.

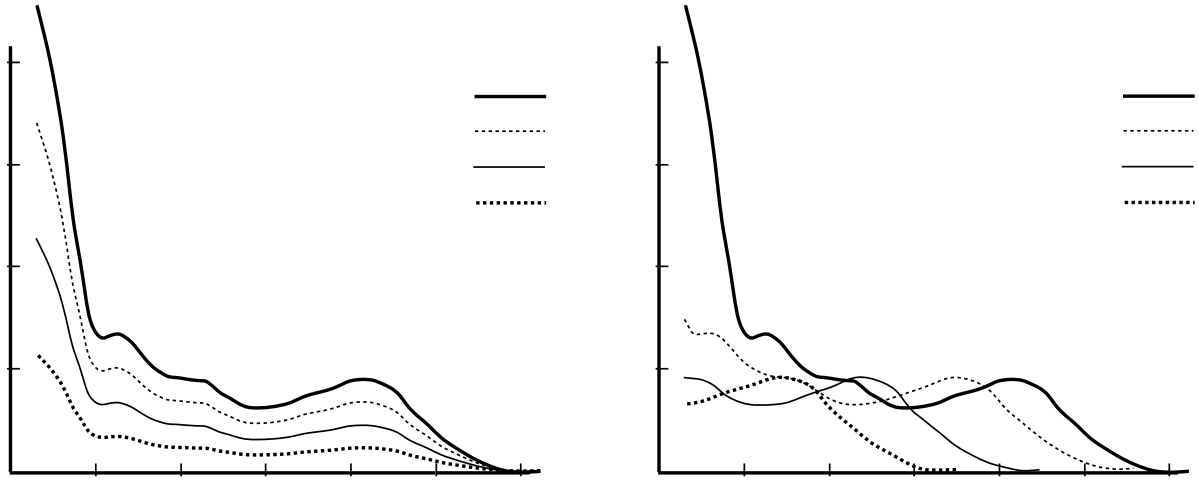

Figure 3: Local and global removal have very different effects on the distribution of the number of sets with certain values of I. I is the number of iterations of A_M left for the processing of a particular set to be completed.

where K is the ratio of local to global removal execution times and $dist(t)$ is the distribution of the number of sets which require a certain number of iterations of local removal to run to completion. The integral is the number of sets remaining after the local elimination phase has been completed.

If the distribution is known *a priori*, then O can be found using the following procedure. Find the minimum point t between 0 and T (where T is the maximum non-zero value in the distribution) such that the following condition holds:

$$\forall(t')(t < t' < T) \left[\int_t^{t'} dist(t)\, dt > K(t' - t)\right].$$

The fact that $dist(t)$ is usually not known *a priori*, plus the complexity of the algorithm, make it impractical for dynamic use, however. But if the distribution decreases monotonically, as has proved to be the case in practice, then the procedure can be simplified substantially: the only t' between t and T that needs to be checked is t itself.

There are two methods for determining O in practice. O can be computed off-line for a data set known to be similar to that being processed. If the variance turns out to be small, as has often proved to be the case, then O can be fixed *a priori*, at least for that domain. Alternatively, we can use the monotonicity assumption: determining whether the local removal phase should be terminated reduces to obtaining $|S|$ after every (perhaps ith) iteration. This can be done efficiently by using global count on the remaining set accumulators.

7 Sample Application Algorithms

These algorithms can be executed simultaneously in all sets using a single control thread. Many more multiassociative algorithms can be found in [4].

Voronoi Diagram and Delauney Triangulation [12]. The brushfire method is used to create Voronoi Diagrams within each set. These regions form subsets. Then, using adjacent set communication (see Table 2), border PEs in each subset get the point addresses of their neighbors. These are then selected one-by-one, and the adjacent point locations collected. The Voronoi Diagram has complexity on the order of the furthest distance between

adjacent points. The Delauney Triagulation has a complexity of $\log N$ times the maximum adjacent sets to any set in the Voronoi Diagram.

Spanning Tree. A variation of graph contraction [10, 11] is used. The algorithm takes $O(\log(|\text{vertices}|)$ time.

References

[1] G.E. Blelloch and G.W. Sabot (1990): "Compiling Collection-Oriented Languages onto Massively Parallel Computers," *Journal of Parallel and Distributed Computing,* 8, pp. 119-134.

[2] A.D. Falkoff (1962): "Algorithms for Parallel Search Memories," *Journal of the ACM,* 9 (4), pp. 488-511.

[3] C.C. Foster (1976): *Content Addressable Parallel Processors,* Van Nostrand Reinhold Co. New York.

[4] M.C. Herbordt, C.C. Weems, M.J. Scudder (1992): "Non-Uniform Region Processing on SIMD Arrays Using the Coterie Network," *Machine Vision and Applications,* 5 (2), pp. 105-125.

[5] M.C. Herbordt, C.C. Weems (1992): "Computing Reduction and Parallel Prefix Using Coterie Structures," *Proceedings of the 4th Symposium on the Frontiers of Massively Parallel Computation,* pp. 141-149.

[6] M.C. Herbordt, J.C. Corbett, J. Spalding, C.C. Weems (1993): "Practical Algorithms for Online Routing on Meshes and Meshes with Reconfigurable Buses," To appear in *Journal of Parallel and Distributed Computing,* 19 (1).

[7] M.C. Herbordt, C.C. Weems (1993): "Parallel-Prefix and Reduction Using Coterie Structures," submitted to *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

[8] J.-F. Jenq and S. Sahni (1991): "Reconfigurable Mesh Algorithms for the Area and Perimter of Image Components," *Proceedings of the 1991 International Conference on Parallel Processing,* Vol. III, pp. 280-281.

[9] J.J. Little, G.E. Blelloch, T.A. Cass (1989): "Algorithmic Techniques for Computer Vision on a Fine-Grained Parallel Machine," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* PAMI-11 (3), pp. 244-257.

[10] G.L. Miller and J.H. Reif (1985): "Parallel Tree Contraction and its Applications," *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science,* pp. 478-489.

[11] C.A. Phillips (1989): "Parallel Graph Contraction," *Proceedings of the 1st ACM Symposium on Parallel Algorithms and Architectures,* pp. 148-157

[12] F.P. Preparata, M.I. Shamos (1985): *Computational Geometry: An Introduction,* Springer-Verlag, New York.

[13] C.C. Weems (1984): "Image Processing on a Content Addressable Array Parallel Processor," *COINS TR 84-14 and Ph.D. Dissertation,* University of Massachusetts.

[14] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, J.G. Nash, D.B. Shu (1989): "The Image Understanding Architecture," *International Journal of Computer Vision,* 2, pp. 251-282.