Parallel Prefix and Reduction Algorithms Using Coterie Structures[*][†]

Martin C. Herbordt[‡] Charles C. Weems

Department of Computer Science
University of Massachusetts, Amherst, MA 01003

Abstract: The efficient computation of region parameters in image understanding by a SIMD array requires that those regions be processed simultaneously. The difficulty lies in orchestrating non-uniform data-dependent communication using only a single thread of control. We introduce a novel new technique we call *coterie structures* for the simultaneous parallel processing of connected components on reconfigurable broadcast meshes. Our technique differs fundamentally from those previously used for this class of networks in that each region is processed using only those PEs to which that region is mapped. Simultaneous and efficient region processing is thereby effected without remapping.

The primary theoretical result is that techniques similar to those used in PRAM graph contraction algorithms can be used to create an optimal randomized reduction algorithm and near optimal deterministic algorithms for reduction and parallel prefix in all regions simultaneously. The primary practical result is a reduction/prefix algorithm that, when used during segmentations of images of outdoor scenes, reduces the number of communication instructions by more than an order of magnitude over previous algorithms. In the process of obtaining these results, we derive basic properties of coterie structures and present algorithms for primitive operations such as information exchange and symmetry breaking within and among structures.

[*]A preliminary version of some of this work appeared in the *Proceedings of the 4th Conference on the Frontiers of Massively Parallel Computation.*

[†]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).

[‡]M.C. Herbordt was supported in part by an IBM Fellowship.

1 Introduction

A critical task in the analysis of sensory input is the reduction of the vast quantities of raw data to a manageable size and into a format that enables further processing. That is, the pixel data of one or multiple images are transformed into tokens representing edges, lines, corners, curves, blobs, regions, surfaces, etc. A common step in this process is the segmentation of the image into groups of pixels sharing some characteristics such as intensities in spectral bands and/or their derivatives.

Segmentation is typically an iterative process, the critical step of which is the characterization of candidate regions by size, border length, and mean and median of constituent pixel values. The problem thus has the following characteristics: it requires a large amount of computation, especially reductions; the input data are spatially mapped; there are numerous problem instances (regions to be characterized) enabling simultaneous parallel processing; the computation is dominated by fine-grained communication; and the communication patterns within each problem instance are non-uniform.

In this article we examine the particular problems of computing parallel prefix and reduction on sets of data mapped to non-uniform, contiguous aggregates of processing elements (PEs) in a SIMD processor array with a reconfigurable mesh network [18, 31, 19, 21]. We are specifically interested in the efficient computation of these function within all aggregates simultaneously while only using a single thread of control. This model is important because it represents an architecture in which it is possible for pixels to be mapped directly to unique PEs, affording the maximum parallelism if the computation can be done efficiently, and low communication overhead enabling excellent performance for fine grained communication.

In directly related work, Jenq and Sahni have published an $O(\log N)$ algorithm to compute simultaneously either the area or the perimeter of all components of an image using a reconfigurable mesh [14]. Beginning with individual pixels, the algorithm successively merges rectangles until only one, the entire image, remains. The algorithm requires 128 broadcast operations per iteration and $\log N$ iterations. Other communication networks for which non-uniform region processing has been studied are mesh connected arrays [30, 32] and pyramid processors [22]. Other work that has been done on vision computation using reconfigurable meshes can be found in [17, 27, 11].

The algorithms presented here use a very different approach than that normally used on a reconfigurable broadcast mesh: they are based on the concept of the *coterie structure*, first introduced in [11]. Each region is processed using only those PEs to which the pixels of the region have been mapped. Thus, although the underlying model is a reconfigurable mesh, the algorithms operate on arbitrary connected subgraphs of the mesh. The basic method is to use knowledge that PEs can obtain about the network configuration in constant time to dynamically repartition the connected subgraphs into various coterie structures as appropriate for the algorithm.

The primary theoretical results of this paper are that PRAM symmetry breaking and graph contraction techniques can be applied to the coterie structures model to obtain an optimal randomized reduction algorithm and near-optimal (within $O(\log^* N)$) deterministic parallel prefix and reduction algorithms. The primary practical result is a reduction algorithm that effectively runs in time equivalent to $7 \log n$ communication operations for a large number of segmentations and with a surprisingly small variance. In the process of obtaining these results, we develop the theory and practice of coterie structures: their definitions and properties (such as symmetry breaking), how they are created and transformed, and how they communicate.

The rest of this article is organized as follows. In the next section we introduce the idea of coterie structures and some useful definitions. In Section 3, we present basic algorithms dealing with communication, symmetry breaking, and transformation. There follow sections on the theoretical and experimental results, respectively.

2 Preliminaries

2.1 Coterie Network

The coterie network [31] is related to other reconfigurable broadcast networks. Members of this family of networks have two characteristics: they are describable using hypergraphs [1] and they are dynamic. In particular, the *time-t $n \times n$ Coterie Network graph* $C_n^{(t)}$ $(t = 0, 1, \ldots)$ has node-set Z_n^2. Each node of $C_n^{(t)}$ is incident to precisely one *coterie-hyperedge*: the coterie-hyperedge incident to node (i, j) of $C_n^{(t)}$ is a subset S of Z_2^n that 1) contains node (i, j) and 2) is *connected* in the sense that the induced subgraph of the $n \times n$ mesh on the set S is a connected graph. Note that at each time t, the coterie-hyperedges partition the node-set Z_n^2 of $C_n^{(t)}$. The coterie-hyperedges of $C_n^{(t)}$ do not depend in any way on the coterie-hyperedges of $C_n^{(t-1)}$.

The coterie network processor array is built on the coterie network graph. Each PE has an input and output port connecting it to precisely one *coterie*, i.e. a (possibly irregularly shaped) bus: PEs (i, j) and (k, l) of the coterie network array are connected at time t to the same coterie (bus) just when nodes (i, j) and (k, l) of the coterie network graph $C_n^{(t)}$ are incident to the same coterie-hyperedge. A coterie is a bus in the sense that, at any time, a single incident PE can "talk" while all other incident PEs "listen." The coterie network has the additional feature that if multiple PEs "talk," the listeners receive the bitwise logical OR of those messages. For a more on hypergraphs and coteries, see [24].

In the physical implementation, each PE controls a set of four switches, N, S, E, W, enabling the creation of coteries. These switches are set by loading the corresponding bits of the mesh control register, which is viewed as local storage within each PE. Thus coterie configurations can be loaded from memory or set from local data-dependent calculations. See Figure 1.

In this article we also assume that nearest neighbor mesh connections are available for

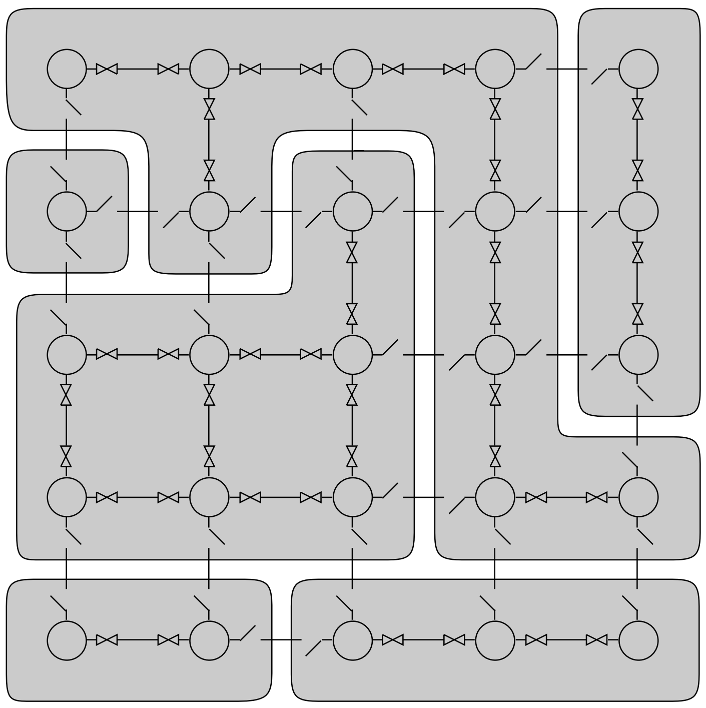

Figure 1: A 5×5 coterie network with switches shown in "arbitrary" settings. Shaded areas denote coteries (the sets of PEs sharing the same circuit).

local communication. These connections are not strictly necessary, either for algorithmic correctness, or for asymptotic complexity. However, they are useful conceptually and relatively inexpensive from a hardware point of view once the wires of the coterie network are in place.

2.2 Multiassociativity

Multiassociativity is an algorithmic framework we have developed in order to process regions in parallel. A sample application can be seen in Figure 2 where an outdoor image (a) has been partially segmented (b). In multiassociative processing, the associative operations (e.g. BroadcastQuery, LocalCompare, ResponseCount, and Some/NoneResponders) often supported in hardware at the array level are efficiently implemented for coteries [11]. For example, the well known associative algorithm SelectSingleResponder [9, 6] can be performed simultaneously for each connected component using only a single global controller using the following code.

```
*Algorithm* SelectSingleResponder
{Beginning with the high-order bit, transmit ID through}
{the Response register. If any PE has a 1 in that bit,}
{turn off activity in PEs with a 0 in that bit.}
FOR BitNum := PEIdLength - 1 DOWN TO 0 DO
    Response := PEId[BitNum]
    IF (Response = Some)
        THEN Activity := Response
```

2.3 Coterie Structures

Advantages of the coterie network are support of one-to-many communication within coteries, reconfigurability of the coteries, and propagation of information over long distances at electrical speeds. Disadvantages are that a circuit can carry only one datum per communication step, and that partitions of the network are constrained by the underlying geometry.

The fundamental strategy in using the coterie network is to orchestrate the partitioning of the array (or some previously determined subsets) into coteries such that the maximum number of PEs needing to exchange information on any algorithm step can do so. To do this efficiently in data dependent algorithms, each PE must determine in a small constant number of time-steps what role it is to play in the next phase or operation of the algorithm: whether it is a sender, receiver, or off; and how the section of the network controlled by that PE should be configured. The key is using the information available locally to each PE, for example: 1) the row and column ID, 2) the tag and the tags of the nearest neighbors, and 3) the OR of some bit of information in a given memory location of members of the coterie.

Note that there are significant differences between the coterie structures model and other models (e.g. the PRAM and the data parallel model described by Hillis and Steele [12]). In particular, pointer jumping is not a viable algorithmic paradigm as the model does not support unit-time random access operations on non-local memory.

2.3.1 Node Classification

In coterie structure algorithms it is often convenient to classify PEs by whether a PE is an interior or a perimeter point and how the switches are set. A PE has different capabilities in controlling the flow of information through the network depending on its number of connections to its nearest neighbors. We call a PE with 0 connections *unconnected*, 1 an *endpoint*, 2 a *through-point*, 3 a *T-junction*, and 4 a *cross*.

We introduce some other terms that are used below. A closed switch that connects a PE to a coterie (implying that the opposing PE has its switch closed as well) is a *link*. A PE is a *node* if it is taking an active part in a computation, i.e. if it holds data and controls links. It is a *null node* if it takes part in a computation only for algorithmic convenience; i.e. the node itself is only carrying the identity element. And finally, it is sometimes useful for a PE to carry no data, but to control links; we refer to these as *helper nodes*.

2.3.2 Coterie Classification

It is also useful to classify entire coteries into categories (we call *coterie structures*). As is the case with data structures, certain algorithms are either available only on some coterie structures, or have different complexity depending on the structure. As we shall see later, an effective strategy is to partition a coterie that does not meet an algorithmic specification into a number of coteries that do, and then to combine the results. Before describing such algorithms, we first introduce the coterie structures that are used in the rest of this paper.

Coteries have already been defined, see Figure 2d for examples. *Horizontal (or vertical) lines* are defined to be coteries where all the North and South (or East and West) switches are open, see Figures 2e. A coterie is a *chain* if it is comprised of two endpoints joined by an arbitrary number of (0 or more) through-points. A *boundary component* is always defined with respect to a coterie: it is the subset of PEs having the property that at least one of its *eight* nearest neighbors (we include diagonals here) is not a member of the same coterie.

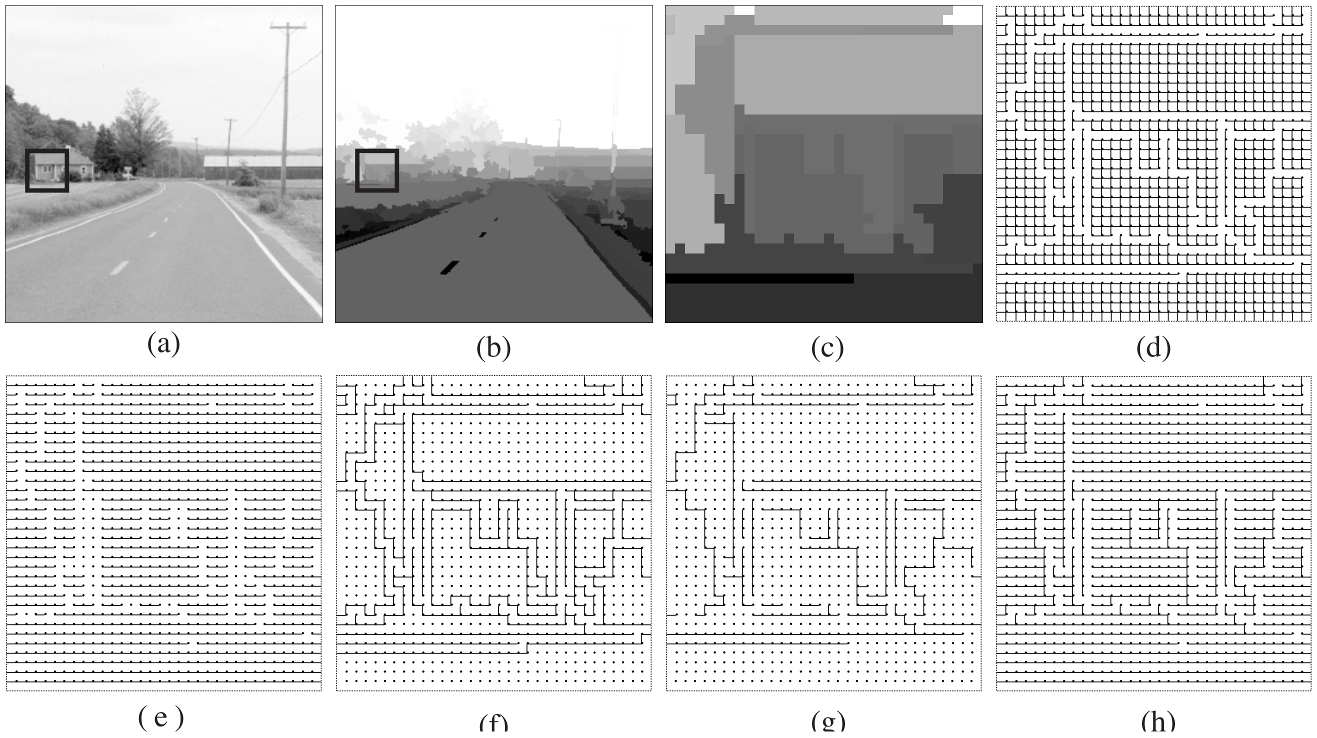

Figure 2: a) Image of a road scene b) segmentation of the image c) 32×32 sub-image of segmentation. (d-h) represent coterie structures derived from sub-image: d) coteries corresponding to regions, e) horizontal lines, f) boundary contours, g) singly vertically connected contours, h) spanning trees.

Not all connections among those PEs are closed, however: in order for two PEs to have a link, they must be mutually adjacent to at least one PE common from outside the region. Figure 2f contains boundary components derived from the coteries in Figure 2d.

PEs are members of the same *level* of a coterie if they are in the same row (column). PEs in the same level need not be in the same line. *Singly mono-dimensionally connected components* come in two varieties, vertical and horizontal. An SVCC is a coterie where at most one PE in each level has an up-link and one PE has a down-link (the same PE may have both). See Figure 2g for examples.

A *spanning tree* of a coterie contains all PEs in the original structure, but where links have been opened to remove all cycles. See Figure 2h for examples.

A coterie is a *C-graph* if all endpoints, T-junctions, and crosses, but not necessarily through-points are nodes.

2.4 Parallel Prefix on Simple Coterie Structures

The literature describing the properties, algorithms, and applications of parallel prefix (also known as scan) is vast; for a sample see [13, 16, 15, 3]. The definition is as follows: Given a set $[x_1, \ldots, x_i]$ of n elements with each element assigned to a different processor and a binary associative operator $*$, compute the n S_i's, where $S_i = x_1 * x_2 * \ldots x_i$, leaving the ith prefix sum in the ith processor. Besides usefulness in its own right, parallel prefix algorithms are usually also the best way of computing reductions, that is, operations identical to parallel prefix but where the partial sums need not be saved. The parallel prefix operation requires $2 \log n$ communication steps on a tree-connected parallel processor, but only $\log n$ are required for a reconfigurable bus. Refer to Figure 3 for an illustration.

The parallel prefix algorithm for an $n \times m$ rectangle is also well known and follows almost immediately from the line algorithm. It has three phases.

Algorithm Parallel Prefix Rectangle

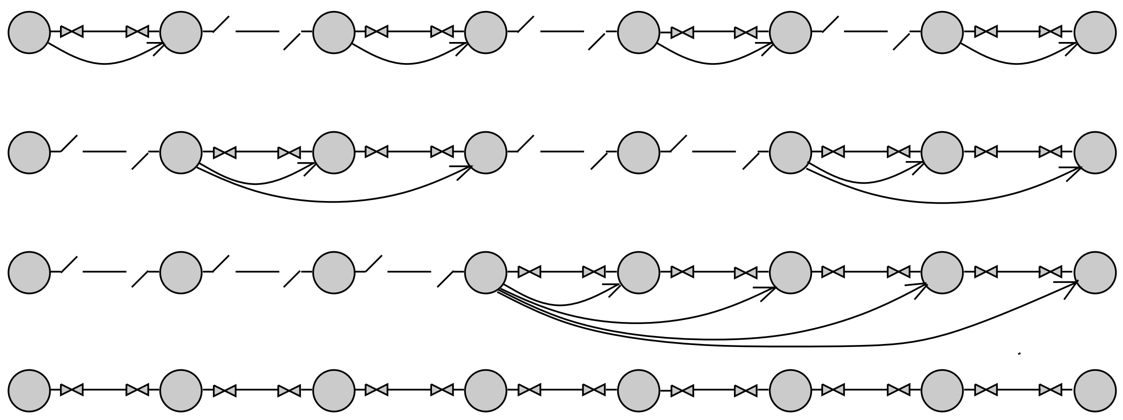

Figure 3: Three iterations and final result of parallel prefix on a line using the + operator. The arrows represent the broadcast operation to elements of the coteries formed when the switches are closed as indicated.

1. Partition the rectangle into m horizontal lines. Run parallel prefix on the lines.
2. Create a vertical line from the m right endpoints. Run parallel prefix on that line.
3. The endpoints update the lines beneath them.

Parallel prefix for a rectangle requires $\log n + \log m + 1$ communication and arithmetic operations.

Somewhat surprisingly, parallel prefix on an SMCC is no more complex and only slightly more complicated than parallel prefix on a rectangle. Again, begin by partitioning the structure into lines and computing parallel prefix. The second phase is slightly more complex because the right endpoints are not neatly aligned and therefore cannot be simply merged into a vertical line. Instead, we have up to three different PEs in each row perform the functions performed by the right endpoint PE in the rectangle algorithm: the right PE endpoint holds and processes the information as before, but two possibly different PEs control the links to the rows above and below respectively (up- and down-links). Recall from the definition of SMCC that (except for the top and bottom rows), exactly one PE in each row controls the up-link and one PE controls the down-link. These PEs recognize this fact in unit time by looking at their mesh control registers. See Figure 4 for an SMCC with those PEs labelled.

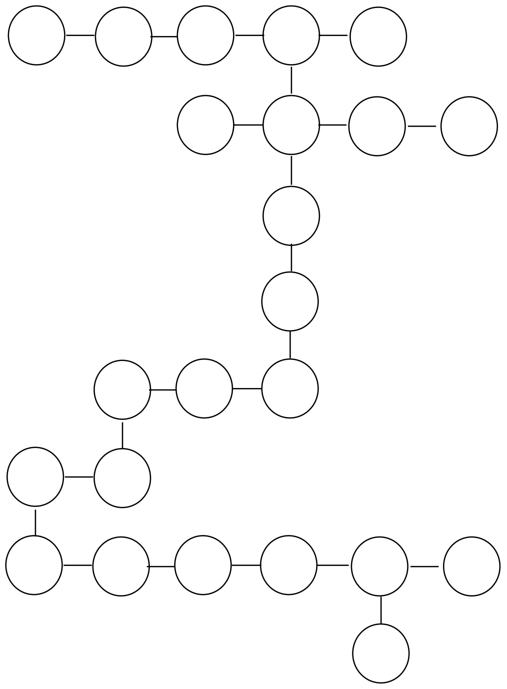

Figure 4: A singly vertically connected component (SVCC) with information-carrying, up- and down-link PEs indicated.

3 Basic Coterie Structure Algorithms

The algorithms presented here will be used as primitives later. The complexity is assumed to be $O(1)$ unless otherwise indicated.

3.1 Communication and Symmetry Breaking

Task: Transfer of data between two adjacent coteries.

Algorithm: Assume that a protocol has been decided upon between two neighbors. The PEs that are on the mutual borders of the communicating coteries close the switches toward each other. With a circuit comprising both coteries thus formed, the PEs of the sending coterie broadcast while the PEs of the receiving coterie listen. See Figure 5.

Task: Symmetry breaking (establishing precendence) between a pair of nodes in a coterie.

Randomized Algorithm: The nodes simultaneously broadcast a sequence of indepen-

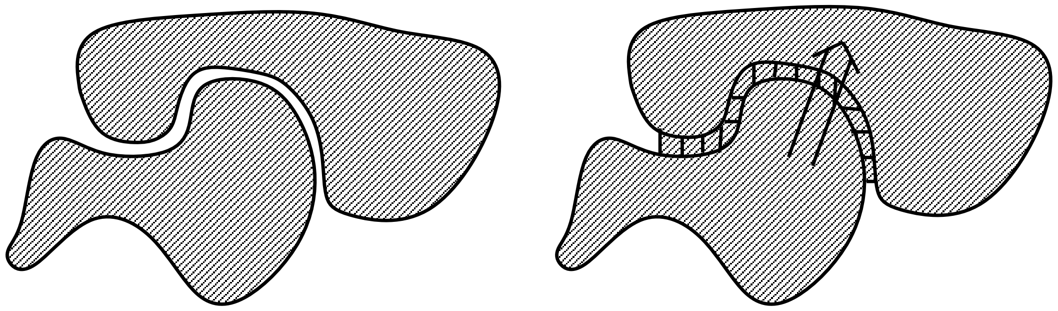

Figure 5: Coteries close mutual switches to communicate

dently generated random bits and their complements until the bit one node broadcasts is the complement of the bit the other node broadcast. Nodes can tell this has occurred by comparing the broadcast results (the two ORs of the two pairs of signals) with the bits they just broadcast. The node that generated the 1 is designated the sender and the other the receiver.

Deterministic Algorithm: Nodes broadcast their IDs and $\overline{ID}$s. After reading the two bitwise ORs and comparing them with the values they broadcast, each obtains its opposite's ID. The node with the higher ID becomes the sender and the other the receiver.

Task: Two nodes within a coterie exchange information.

Algorithm: Precedence is established by using one of the symmetry breaking algorithms immediately above. The task is completed with a pair of broadcasts.

Task: Each node in a C-graph exchanges information with all its neighbors.

Deterministic Algorithm: Recall that nodes are separated by chains of null nodes of arbitrary length and that therefore, all nodes have links either to other nodes, or to chains of PEs that have a node at the other end. The basic idea is to use the neighbor connections to break the symmetry; otherwise it would be impossible to tell deterministically which pair of nodes should communicate when. The cases where the length of the intermediate chain is 0, 1, and > 1 are handled separately (see Figure 6). In all three cases, the algorithm starts with the nodes using the nearest neighbor connections to transfer copies of their information to all the PEs toward which a link is established.

Case 1: Chain length $= 0$. The nearest neighbor move has already completed the transfer.

Case 2: Chain length $= 1$. The intermediate PE swaps the data from the two adjacent nodes and transfers them to their destinations.

Case 3: Chain length > 1. The two node swap algorithm is used to transfer the data between the two endpoints of the chain. Neighbor transfers complete the exhange.

Randomized Algorithm: We use a variation of the randomized symmetry breaking procedure presented above. It is assumed that nodes have a list of links to their neighboring nodes. Nodes randomly close switches in the direction of one of the links, opening those in all the other directions. Nodes then use one of the symmetry breaking procedures to check whether the opposite node has done the same. The nodes that did not form a correspondence with a neighbor again randomly select a link for another try. This continues for a constant number of iterations during which a constant fraction of the nodes in the C-graph will have established correspondence. Corresponding nodes exchange information and temporarily remove each other from their adjacency lists. This procedure continues until all nodes have completed the information exchange with all their neighbors.

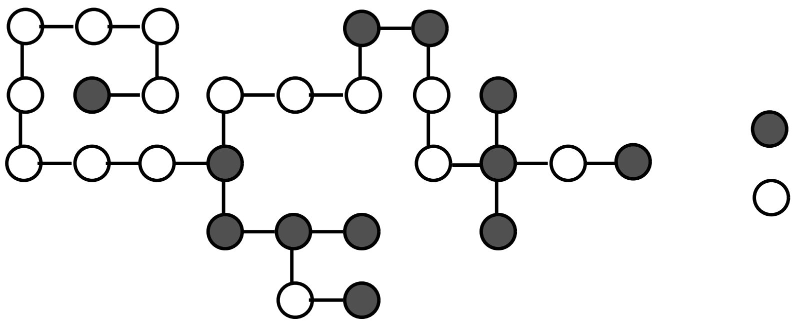

Figure 6: C-graph with nodes separated by chains of null nodes of varying length.

Task: $O(1)$ coloring a chain.

Algorithm: When using the standard PRAM $O(1)$ coloring algorithm [10] for constant degree graphs, the number of colors is small when operating on an ordered structure, say a list or rooted tree (see e.g. [5]), but exponential in the degree of the graph when not. The basic idea then, is to use information available locally to nodes on the chain to create such an ordering. Nodes obtain their neighbors' IDs which are compared with their own. Nodes label themselves $\nwarrow\nearrow$, $\nearrow\nwarrow$, $\nwarrow$, or $\nearrow$, depending on whether the ID "slope" is up or down in each direction. Nodes exchange slope labels with their neighbors. The neighbor on each link can have one of two possible labels: either the slope continues in the current direction (e.g. $\nearrow$ goes to $\nearrow$), or it can go to a local maximum or minimum (e.g. $\nearrow$ goes to $\nearrow\nwarrow$). Nodes that are $\nwarrow\nearrow$ and $\nearrow\nwarrow$ cannot be adjacent to other nodes that have that same slope label and so simply color themselves $\nwarrow\nearrow$ and $\nearrow\nwarrow$. The remaining nodes form monotonic ID sequences. These sequences, which now *have* direction information, can be colored using the previously mentioned 6-color algorithm and then spliced together with the $\nwarrow\nearrow$ and $\nearrow\nwarrow$ nodes to form an 8-colored chain. To reduce the number of colors to 6, the $\nwarrow\nearrow$ and $\nearrow\nwarrow$ nodes examine their neighbors' colors and choose any of those remaining to be their own. The 6-color PRAM list algorithm has $O(\log^* N)$ complexity, as does the 6-color coterie chain algorithm.

Task: Create Maximal Independent Set of nodes in a C-graph.

Deterministic Algorithm: Use the information exchange algorithm above to implement the $O(\log^* N)$ complexity MIS algorithm found in [10].

3.2 Minimum Partition of Coteries Into SVCCs

In this section we prove the somewhat surprising result that a minimum partition of a coterie into SVCCs can be constructed in constant time.[1] Besides the inherent usefulness of this result (see section 5), the significance is that the algorithm performs a complex function using only information that can be determined locally by each PE in constant time.

[1]A simple algorithm for a non-minimum, but useful, partition is presented in [11].

A requirement of an O(1) partitioning algorithm is to be able to determine (as yet unspecified) properties independently of distances. We have such a tool in the F-G algorithm (see below). In particular the following question can be answered in constant time: "For any PE P with property F on a line, is there a PE Q with property G between P and either the next PE with property F (to either the left or right) or the end of the line?" We define such a PE Q to be *adjacent* to PE P. If two adjacent PEs become members of the same coterie by closing switches towards each other and opening switches away, then those PEs are said to be *merged*.

```
Algorithm F-G
{For all PEs with property F, find out whether there}
{is a PE with property G between it and the next PE}
{to the left with property F, or the end of the line.}
SaveCoterieSettings()
Coterie[N,S] := OPEN
If (Tag = F) Coterie[E] := Open
If (Tag = G) Broadcast(TRUE)
If (Tag = F) G-AdjacentLeft := CoterieInput
RestoreCoterieSettings()
```

We start by observing that a greedy algorithm produces the desired partition: starting at the top level, draw circles around each down-link. Then level by level, expand each circle to include links at the lower levels while not letting the circles intersect. See Figure 7 for an example.

Lemma 1 *The cardinality of the minimal set of SVCCs $|S|$ in a coterie is equal to the number of SVCCs that must be added with the addition of each new level.*

Proof: Assume a coterie partition contains fewer SVCCs than the number that need to be added at every level. Then one of the levels of the coterie would be partitioned into fewer SVCCs than by the above method. But that would either leave vertical links unaccounted for, or an SVCC with multiple vertical links. Both of these are contradictions. $\square$

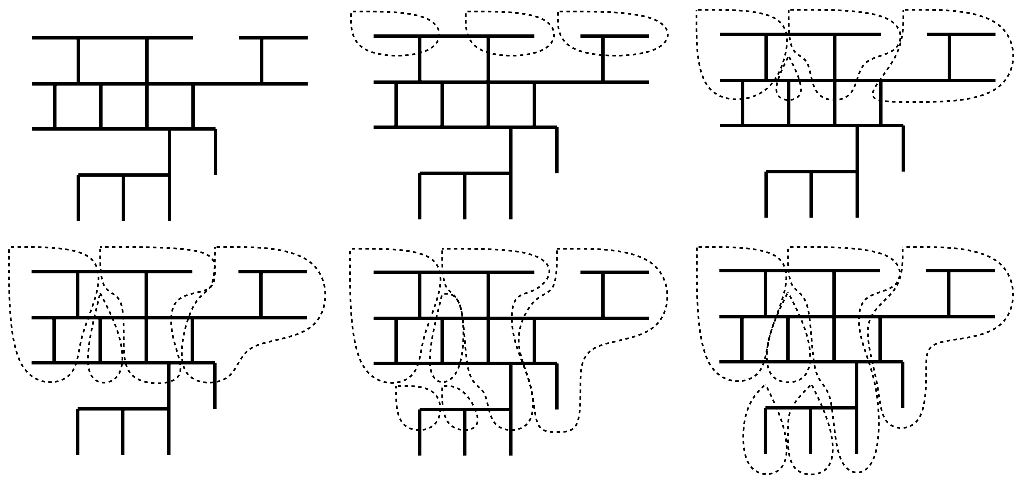

Figure 7: A coterie can be partitioned into a minimum set of SVCCs through a serial level-by-level growth algorithm.

Lemma 2 *To create a minimal set of SVCCs within a coterie, it suffices to create a maximal matching between adjacent up- and down-links and merge them together.*

Proof: The maximal matching leaves the fewest possible leftover vertical links, requiring that the fewest possible new SVCCs need to be created at that level. □

We now present the SVCC construction algorithm. First label the PEs with up- and down-links as U and D. All other PEs are ignored. Next, partition the coterie into lines. Within each line, create a maximal matching between adjacent U and D PEs, a sufficient condition for creating a minimal partition of a coterie into SVCCs. This is a multi-step process:

1. Each U (D) determines whether there are adjacent D (U) PEs to the left and/or right using the F-G algorithm.

2. Each U (D) PE labels itself with a 0, 1, or 2 depending on the number of adjacent D (U) PEs. We call those numbers the *incidences* of each PE.

3. The U (D) PEs send the number of incidences to their adjacent D (U) PEs, if any.

4. Each PE is labeled with an ordered triple composed of the number of incidences of the left adjacent PE, its own incidences, and the incidences of the right adjacent PE.

5. The value of the ordered triple is sufficient to determine whether each PE should open or close its East and West switches to create the optimal partition for all but the (2,2,2) case. PEs with (-,0,-) do nothing, with (*,1,*) merge with their one adjacent PE, with (1,2,1) arbitrarily merge with either left of right adjacent PE, and with (2,2,1) or (1,2,2) merge with the adjacent PE with one incidence.

6. PEs with (2,2,2) merge with adjacent PEs according to the procedure described below.

See Figure 8 for an illustration of the labeling of a single line. Before presenting the (2,2,2) case, we show the correctness of what we have described so far.

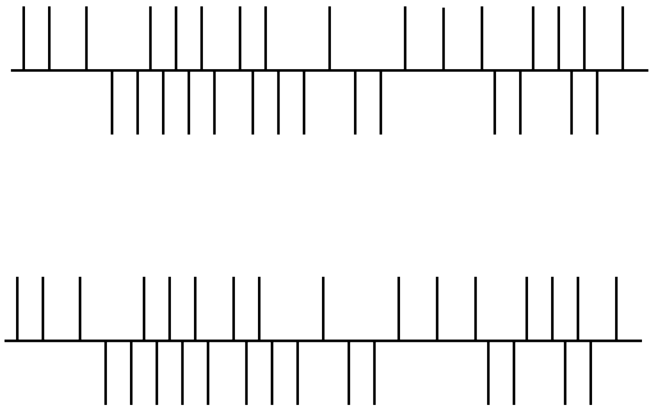

Figure 8: A line of one level of a coterie with a) up-links, down-links, and incidences labeled, and b) ordered triples of incidences of own and neighboring PEs.

Lemma 3 *A $U(D)$ PE can only form an SVCC with a single, adjacent, D (U) PE.*

Proof: Assume not. Then either two or more D PEs must be in the SVCC at that level. If the D PE is not adjacent, the line must be open circuited. Both are contradictions. □

Lemma 4 *The U-D merges effected by the SVCC construction algorithm are correct for the cases shown in step 5 (not the (2,2,2) case).*

Proof: From Lemma 4 we know we need only insure that each PE merges with the correct, adjacent PE. Cases (-,0,-), (*,1,*), and (1,2,1) are trivial: in the first nothing can be done, in the second there is no choice, and in the third the choice does not matter. In the cases (2,2,1) and (1,2,2), the PE should merge with the PE with one incidence: It is possible for the PE with two incidences to find another PE with which to merge, while the PE with one incidence has only this opportunity. □

The (2,2,2) case is critical: if the merger is not done correctly, the complexity of our parallel prefix algorithm increases from $O(\log N + |S|)$ to $O(N)$. We we formalize this fact in the following lemma.

Lemma 5 *If a PE P with label (2,2,2) is not guarenteed to merge with the correct PE, then it is possible for an adversary to force the creation of an $SVCC$ partition where $|S|$ is $O(N)$ greater than optimal.*

Proof: As previously, we are only concerned with U and D PEs. The neighborhood of the line around P necessarily has the following form: a PE with one incidence, followed by a series of PEs with two incidences, followed by a PE with one incidence (1,2,...,2,1). The series of 2 incidence PEs can have either an even or an odd number elements. If odd, and all the (2,2,2) PEs merge arbitrarily with either their right or left neighbors, then exactly one PE is left over, the best possible result. In the even case, however, such a uniform merging strategy leads to there being either zero or two left-over PEs. If no method existed to make the correct decision, it would thus be possible for an adversary to create $O(N)$ strings of constant length which after partitioning would each leave 2 unmatched links. □

We now solve the (2,2,2) case. From Lemma 6 we know that we only need to worry about the case where the series of PEs with 2 incidences is of even length. There are two possibilities: the (2,2,1) PE at the right end of the $2, \ldots, 2$ series is either a U or a D; the (1,2,2) PE at the left end is always the opposite. In the first case, all (2,2,2) PEs that are U should merge left, in the second case they should merge right.

A procedure for handling the (2,2,2) case follows. PEs that are (2,2,1) open their East switches and PEs that are (1,2,2) open their West switches to form lines with the pattern

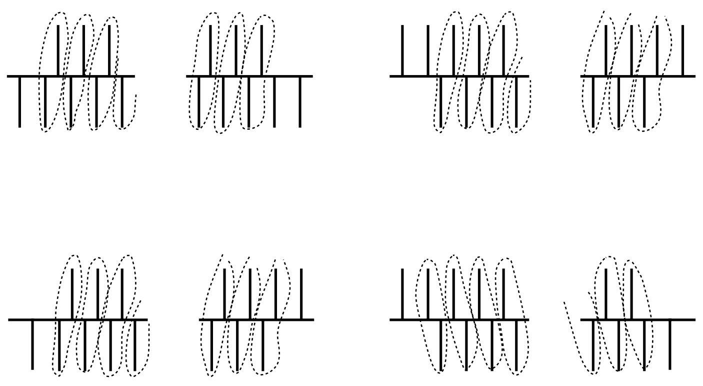

Figure 9: Four cases of merging adjacent PEs within strings (1,2,2,2,1). In a) and b), there are an odd number of (2,2,2) PEs and either the (1,2,2) or the (2,2,1) PE always remains unmerged. In c) and d), there are an even number of (2,2,2) PEs. The merges must all go in the correct direction or both (1,2,2) and (2,2,1) PEs will remain unmerged.

$(1, 2, \ldots, 2, 1)$. The PE that are (2,2,1) and U broadcast a one, those that are (2,2,1) and D broacast a zero. The procedure is repeated for the (1,2,2) PEs. The (2,2,2) PEs read these values and have four cases: (0,0), (0,1), (1,0), and (1,1). In the (0,0) and (1,1) cases, there are an odd number of U (D) PEs: these can thus merge arbitrarily (but uniformly) with their left or right (right or left) adjacent D (U) PE. In the (0,1) case the D (U) PEs merge right (left) and in the (1,0) case the D (U) PEs merge left (right). See Figure 9 for an illustration of the four cases with the merges done correctly.

Lemma 6 *The U-D merges effected by the SVCC construction algorithm are correct for the cases where all elements of the triple are 2's.*

Proof: Follows from previous two paragraphs. □

Theorem 1 *The SVCC construction algorithm partitions any coterie into a minimal set of SVCCs in constant time.*

Proof: That the time is constant is immediate: there are no loops and no dependencies on the number of PEs. The correctness follows from the preceding lemmas. □

4 Algorithms Based on Graph Contraction

In this section we again show how techniques used in other models can be applied to coterie structures. In particular, the communication and symmetry breaking algorithms of the previous section are integrated with PRAM contraction techniques (see [20, 26]) to produce an $O(\log N)$ randomized reduction algorithm and $O(\log^* N \log N)$ deterministic parallel prefix and reduction algorithms. Since the prefix algorithm follows almost immediately from the reduction algorithm, only the latter is presented.

```
*Algorithm* REDUCE
**While** |nodes| > 1 **do in parallel**
    Use MERGE to merge legal pairs of adjacent nodes
    such that the resulting vertices have degree $\leq d_{max}$.
```

Phillips shows that, for bounded degree planar graphs, there are always a constant fraction of nodes eligible for merger as long as $d_{max} \geq$ the degree of the graph [26]. That a constant fraction of nodes eligible for merger does so follows by arguments similar to those used in the previous section. The **While** loop thus executes $O(\log N)$ times.

The MERGE procedure is more complicated when applied to coterie structures than the equivalent PRAM algorithm. The reason is that in a distributed memory model, nodes cannot simply be eliminated once their data have been combined: the wires associated with the underlying PEs may still be needed to transmit information between the nodes remaining in the computation. Since these leftover PEs always through-PEs, MERGE operates on a C-graph.

The constraint that no node ever have degree > 4 creates a relatively small constant number of merge cases that can be handled separately. This number is decreased by imposing the convention that the node with smaller degree always transfer its data to the node with larger degree. We examine five cases (see Figure 10), the others are analogous.

1. An arbitrary node merges with a node with degree 1 or 2. If the lower degree node is d_1, the higher degree node eliminates the link to that node. If it is d_2, then the links remain intact, although that node is removed.

2. A d_3 node merges with another d_3 node with which it has one common neighbor. One node is selected to send data, the other to receive. The receiver retains its link to the common neighbor node while the sender removes it.

3. A d_3 node merges with another d_3 node with which it has no common neighbors. The resulting node has degree 4, but its functionality must be distributed over the two original nodes. One node holds the data and controls two links, while the other, which has now become a helper node, controls the other two links.

4. A d_4 node with a helper node merges with a d_3 node with which it has one common neighbor. The d_3 node is turned off and the link from the d_3 node to the common neighbor is eliminated.

5. Two d_4 nodes with helpers and two common neighbors merge. One of the nodes (and its helper) are turned off and its links to the common neighbors are removed.

For this list to be exhaustive (including analogous cases), it is necessary for the following propositions to be true:

Proposition 1: Only nodes of degree 4 can ever need a helper PE.

Proposition 2: No node ever needs more than one helper PE.

The truth of these propositions follows from the fact that for nodes that are created using the above procedures, the only possible configuration where a helper node can be introduced is in case 3. That is because compound nodes (nodes with helpers) can only be created when the node formed through merger has greater degree than either of the two constituents. We observe that after a merger where the degree does not increase—between any node and a d_2, a d_3 with one common neighbor, or a d_4 with two common neighbors—the merged node is

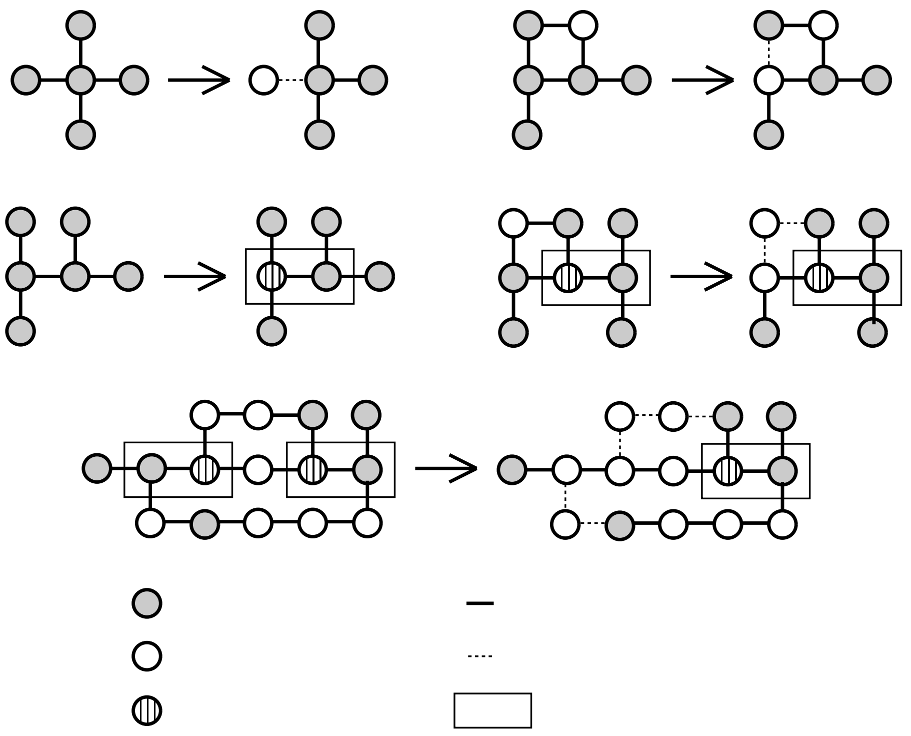

Figure 10: Five cases of nodes merging on C-graphs. In all cases, **A** merges into **B**.

a through-PE with no need to control information flow.

```
Procedure MERGE
∀ nodes do in parallel
1. Find adjacent nodes and create a neighbor list.
2. Exhange neighbor lists with all neighbors.
3. ∀ neighbor lists do sequentially
       Eliminate duplicates, determine neighbors with which merger is legal.
4. Execute either the random or deterministic version of SELECT-MATCH-PAIRS.
5. Combine nodes according to the cases presented above.
```

Since all the nodes are guaranteed to have bounded degree, steps 1, 2, 3, and 5 are $O(1)$.

```
Procedure DETERMINISTIC-SELECT-MATCH-PAIRS
1. Run the COTERIE-MIS algorithm from section 3.1.
2. Members of the MIS that are members of a legal merge pair choose a
   merge partner. The partner breaks ties if it is multiply selected.
```

The complexity of COTERIE-MIS is $O(\log^* N)$ which is also the complexity of the algorithm.

```
Procedure RANDOMIZED-SELECT-MATCH-PAIRS
While nodes in match pairs are unmatched and have an unmatched partner
        Nodes in match pairs randomly select a match partner.
        If the nodes agree, they declare themselves matched.
```

Since nodes have a constant probability of being matched during every iteration, the expected time of the algorithm is $O(1)$. Therefore, step 4 of MERGE is $O(1)$ if the randomized algorithm match select algorithm is used, and $O(\log^* N)$ in the deterministic case. Thus the overall REDUCE complexities are $O(\log N)$ and $O(\log^* N \log N)$ respectively.

The complexity constant improves drastically if the coterie is known to be a tree. The algorithm runs as follows:

```
Algorithm TREE-REDUCE
While |nodes| > 1 do in parallel
        Use TREE-MERGE to merge legal pairs of adjacent nodes
        where legal pairs have at least one node with degree 1 or 2.
```

Since trees have the property that at least half the nodes have degree 1 or 2, the **While** loop executes $O(\log N)$ times. The fraction of nodes available for merger in the general merge algorithm is likely to be substantially smaller.

```
Procedure TREE-MERGE
∀ nodes do in parallel
1. Find adjacent nodes and compute degree.
2. Exhange degree information with all neighbors.
3. Execute either the random or deterministic version of TREE-SELECT-MATCH-PAIRS.
5. Combine nodes according to case 1 presented above.
```

The tree version of randomized match pair selection is identical to the non-tree version. In the deterministic case, however, we only need to run chain MIS, rather than the coterie version.

```
Procedure TREE-DETERMINISTIC-SELECT-MATCH-PAIRS
1. Nodes adjacent to $d_1$ nodes select one of them for merger.
```

2. Nodes with degree 3 or 4 adjacent to d_2 nodes choose one
   for merger iwth the d_2 node breaking ties.
3. Form chains of the contiguous remaining d_2 nodes. Members
   of the MIS choose merger partners, with chosen nodes breaking ties.

The advantages of TREE-MERGE are that no neighbor lists need to be created, exchanged, or searched for duplicates; that symmetry breaking is much easier on a chain than on a bounded degree planar graph; and that only case 1 of the actual node combining needs to be executed. However, since we are not aware of any algorithm to create a spanning tree out of arbitrary coteries that is itself not $O(\log N)$, TREE-REDUCE will likely only be useful if multiple reductions on a coterie partition are to be computed.

5 Practical Algorithms and Experimental Results

In this section we investigate the performance of reduction algorithms on tasks arising during the execution of a working segmentation system. This system has been used extensively in image interpretation research in various image domains. We find that a combination of array-based and multiassociative techniques yields the best performance in practice.

5.1 A Segmentation System

Many segmentation systems are based either on image splitting guided by the analysis of feature histograms [25, 23], or region merging based on local region properties constrained by global criteria [4, 7, 29]. The system we have used to generate problem instances—the Nagin-Kohler-Griffith-Beveridge system—combines these two approaches [2]. In the first phase, spatially localized feature histograms are used to *over-segment* the image. That is, sensitivity parameters are set so that all region boundaries with even a modest likelihood of having semantic content are retained. In the second phase, a region merging algorithm is applied to remove those boundaries least likely to be meaningful.

The region merging phase begins by characterizing each region by the means and standard deviations of various spectral quantities, by its size, and by the lengths of its common borders with adjacent regions. Based on these values, merge scores are calculated with respect to the adjacent regions. Region pairs are merged if their merge score is both a local maximum and surpasses a global threshold. After the merge is completed, the newly created regions are again characterized and new merge scores calculated for region pairs. The process is repeated until no merge scores surpass the global threshold.

The bulk of the computation occurs during the original region characterization; thereafter, characterizations are computed from the attributes of the constituent regions. We therefore concentrate on the reduction of the regions of the oversegmented image.

The data set consists of 28 256×256 intensity images of which 13 are road scenes and 15 house scenes. Figure 11 shows a sample image from the road scene domain, the phase 1 output (oversegmented image), and the final segmentation. See [2] for more examples. The oversegmented images have an average of 1900 regions with a standard deviation of 788. See Figure 12 for details.

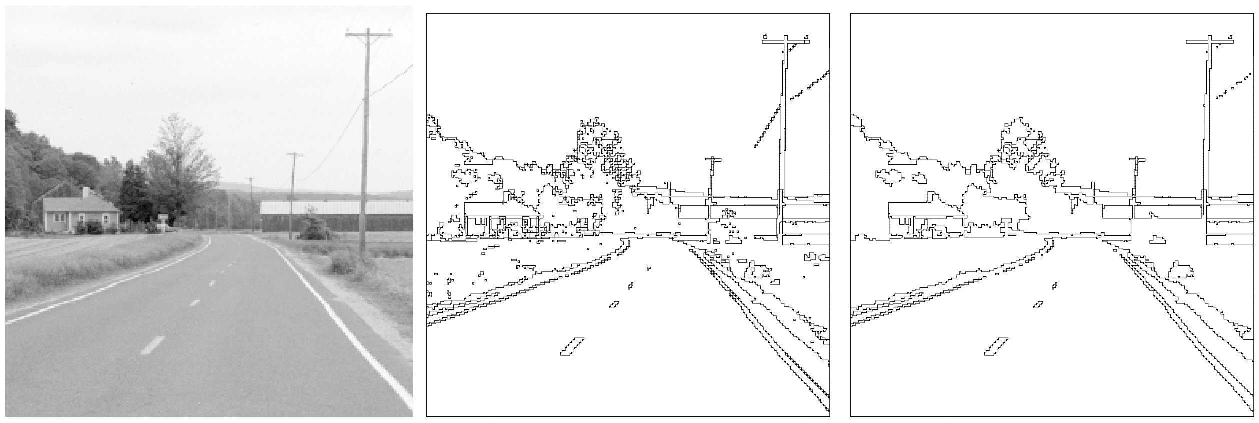

Figure 11: a) Sample image is Road Scene 16, b) over-segmentation of image, c) image after region merge.

5.2 SVCC-based Reduction Algorithms

We now present an SVCC-based reduction algorithm for arbitrary coteries that is the basis for our practical algorithm. The parallel-prefix algorithm follows as before with little added complexity.

```
Algorithm Local REDUCE
1. Partition the coterie into horizontal lines.
2. Reduce lines and leave the result in the right endpoint.
3. Partition the coterie into SVCCs.
4. Reduce SVCCs leaving the result in the bottom endpoint. The bottom
   endpoints of the SVCCs in each region form a set $S$ with cardinality $|S|$.
DO UNTIL $|S| = 0$ for all coteries
          6. Use SelectSingleResponder to select an element $s$ from $S$.
          7. $s$ broadcasts its partial sum to the elements in $S$ which
             combine that value with their own.
          8. $s$ removes itself from $S$.
```

Steps 1 and 3 are constant time operations, while 2 and 4 require $\log n$ communication and arithmetic operations.[2] In the DO loop (a phase we refer to a *local removal*), Step 6 uses an $O(\log N)$ algorithm (although with a small constant), while Steps 7 and 8 are again constant time operations. The overall complexity is therefore proportional to $\log N$ and depends on the number of SVCCs $|S|$ into which the worst behaved coterie has in the image been partitioned. Since it is relatively easy to construct a coterie where $|S| = O(N)$, we are left with an $O(N \log N)$ algorithm.

Since we are invesigating practical algorithms in this section, a quantity more important than the worst case of $|S|$ is its size during real applications. Results are presented in Figure 12. In summary, it is common for at least one region per image to be so badly behaved as to make the above algorithm impractical: the DO loop would have to be executed an average of 153 times and a value of 300 would not be unlikely. However, the number of badly behaved regions per image is small (with an average of only 13 regions per image

[2]For the rest of this article, the algorithm used in step 3 is assumed to be the one in [11].

having more than 10 SVCCs), signaling the possibility of large performance gains with a relatively small amount of additional work.

Image Name	number of regions	number of SVCCs	avg SVCCs per reg.	max SVCCs per reg.	count of regions this many SVCCs				
					1-10	11-20	21-30	31-40	41+
amroad0	1967	3159	1.61	198	1944	11	6	2	4
amroad10	1949	3246	1.67	206	1933	6	2	2	6
amroad14	1017	1627	1.60	120	1007	2	3	1	4
amroad15	1685	2654	1.58	202	1674	4	3	0	4
amroad16	546	908	1.67	44	537	4	1	3	1
amroad1	1294	2271	1.76	302	1284	5	1	0	4
amroad25	2915	4672	1.61	276	2895	8	4	4	4
amroad29	1352	2362	1.75	453	1341	8	0	0	3
amroad2	2821	3780	1.34	50	2804	11	4	1	1
amroad33	1848	3012	1.63	268	1830	13	2	0	3
amroad34	1399	2178	1.56	286	1391	3	1	1	3
amroad3	2732	4006	1.47	61	2707	14	4	3	4
amroad5	2195	3454	1.58	348	2185	3	3	2	2
avg road	1824.6	2871.5	1.60	216.5	1810.2	7.1	2.6	1.5	3.3
house10	1176	1647	1.41	74	1168	5	1	0	2
house14	1727	2276	1.32	48	1719	3	2	2	1
house15	1291	1721	1.34	37	1284	2	3	2	0
house16	1168	1555	1.34	48	1162	2	1	1	2
house17	1451	1927	1.33	64	1442	6	0	1	2
house19	861	1263	1.47	69	852	3	2	1	3
house1	1638	2252	1.38	158	1630	4	2	1	1
house20	1363	1848	1.36	47	1350	7	3	0	3
house2	2109	2923	1.39	82	2095	10	2	1	1
house3	1890	2628	1.40	106	1879	6	2	0	3
house4	2786	3838	1.38	313	2770	10	3	1	2
house5	3715	5171	1.40	75	3692	9	6	1	7
house6	2442	3540	1.45	81	2420	15	2	1	4
house7	2067	3219	1.56	84	2043	11	7	2	4
house8	3796	4955	1.31	188	3786	8	1	0	1
avg house	1965.3	2717.5	1.38	98.3	1952.8	6.7	2.5	0.9	2.4
avg all	1900.0	2789.0	1.48	153.1	1887.6	6.9	2.5	1.2	2.8

Figure 12: Image statistics for the data set after completion of the oversegmentation phase. For reference, the average number of regions per image is substantially greater than n, the average maximum number of SVCCs per region is only slightly less than n, and the average number of regions per image with more than 10 SVCCs is 13.4.

One possibility is to add one or more SMCC reduction steps, i.e. to repeat steps 3 and 4 alternating between the vertical and the horizontal dimension. After all, if SVCC reduction of endpoints eliminates a large fraction of the horizontal line accumulators, it seems likely that SHCC reduction of SVCC accumulators could also be helpful.

```
*Algorithm* SMCC REDUCE
1. Partition the coterie into horizontal lines.
2. Reduce lines and leave the result in the right endpoint.
DO I TIMES
    3.  Partition the coterie into SVCCs.
    4.  Reduce the SVCCs leaving the result in the bottom endpoint.
    5.  Partition the coterie into SHCCs.
    6.  Reduce the SHCCs leaving the result in the right-most endpoint.
DO UNTIL $|S| = 0$ for all coteries
    8.  Use SelectSingleResponder to select an element $s$ from $S$.
    9.  $s$ broadcasts its partial sum to the elements in $S$ which
        combine that value with their own.
    10. $s$ removes itself from $S$.
```

The distribution of $|S|$ as a function of I (see Figure 13) shows that the return diminishes rapidly. What happens is that some regions have a large number of SVCC-SHCC pairs that contain each other's accumulators. No progress was ever made after I reached 20.

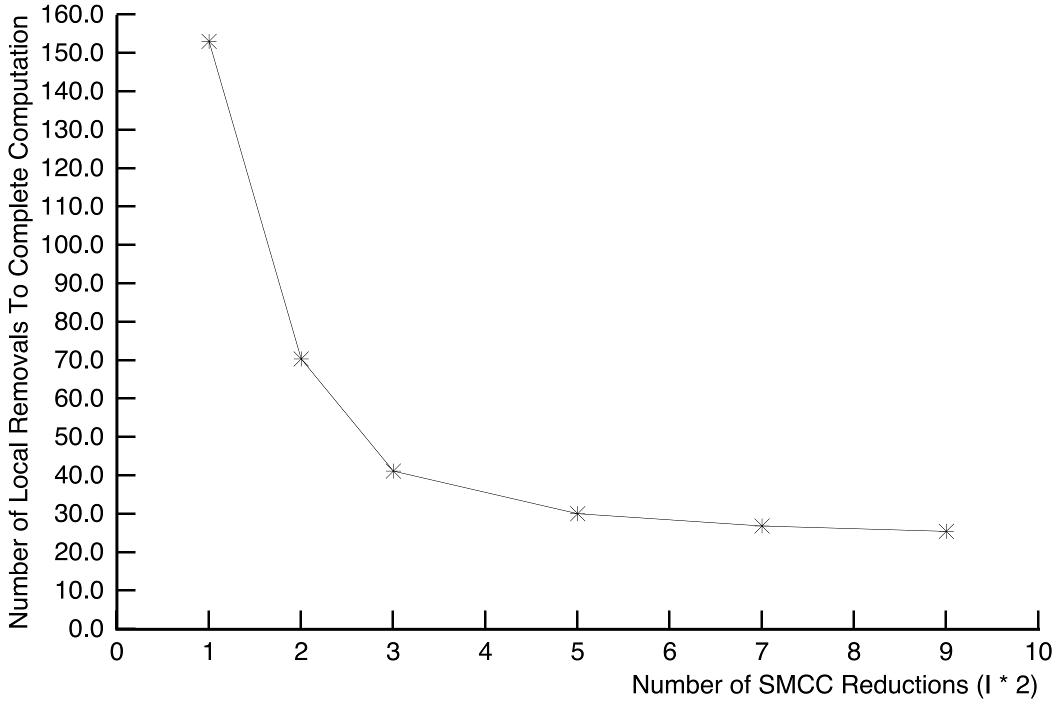

Figure 13: Graph of Max($|S|$) versus I. The reduction in the number of times local removal must be executed levels off after $I = 2$ (number of reductions = 4).

A more promising alternative is the hybrid algorithm in the following section.

5.3 A Hybrid Reduction Algorithm

For any particular region r from the set of regions R, it is usually faster to execute reduction/prefix algorithms by applying the resources of the entire array to that problem instance than it is to use only those PEs to which the region is mapped. There follows a simple reduction algorithm based on array reduction procedures (which we call *global removal*).

```
Algorithm Global REDUCE
DO UNTIL $|R| = 0$
        1. Use GlobalSelectSingleResponder to select an arbitrary region $r$ from $R$.
        2. Use the resources of the array to reduce $r$.
        3. Remove $r$ from $R$.
```

Depending on the operation being used during the reduction, the complexity of step 2 can range from $O(\log N)$ in the general case, down to $O(1)$ when executing the CountSelectedResponders operation with appropriate hardware support [28]. Similarly, step 1 can also have complexity from $O(1)$ using hardware described in [8] to $O(\log N)$ if only a global OR circuit is available. In any case, the algorithm is efficient if and only if $|R|$ is small. We have seen, however, that the number of regions in an oversegmentation can be huge: in the case where regions are relatively small, $|R| = O(N)$.

The idea behind hybrid reduction is to combine the global algorithm with the local removal operation used in the SMCC-based algorithm. After two reduction passes (steps 1-4), an as yet undetermined number of local removals is executed. Global removals are then used to finish the reduction.

```
Algorithm Hybrid-REDUCE
1. Partition the coterie into horizontal lines.
2. Reduce lines and leave the result in the right endpoint.
3. Partition the coterie into SVCCs.
4. Reduce SVCCs leaving the result in the bottom endpoint. The bottom
   endpoints of the SVCCs in each region form a set $S$ with cardinality $|S|$.
DO an optimal number ($O$) TIMES
   6.   Use SelectSingleResponder to select an element $e$ from $S$.
```

```
    7.  $e$ broadcasts its partial sum to the elements in $S$ which
        combine that value with their own.
    8.  $e$ removes itself from $S$.
DO UNTIL $|R| = 0$
    9.  Use GlobalSelectSingleResponder to select a region $r$ from $R$.
    10. Use a global reduction algorithm to reduce $r$.
    11. Host broadcasts result back to $r$.
    12. $r$ removes itself from $R$.
```

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 remaining regions during local and global removal (see Figure 14). Local removal is equivalent to moving the Y-axis to the right one unit per iteration. 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.

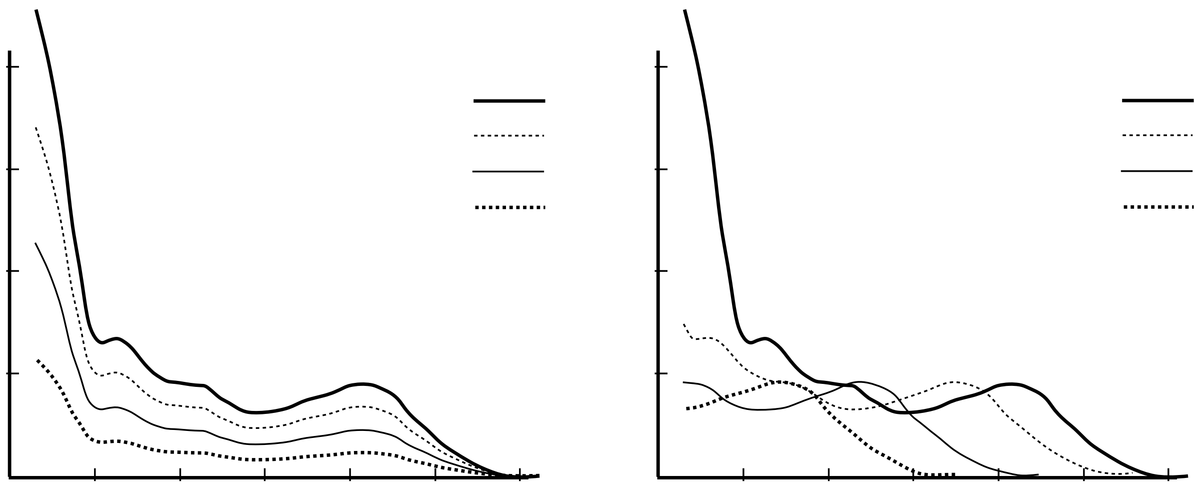

Figure 14: Local and global removal have very different effects on the distribution of the number of regions with certain values of $|S|$. Data shown is synthetic.

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

$$K{*}O \;+\; \int_{O}^{\infty} dist(t)\, dt,$$

where K is the ratio of local to global removal execution times and $dist(t)$ is the distribution of the number of regions with certain values of $|S|$. The integral is the number of regions 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:

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

This procedure has been used to find optimal O's for the data set; the results are presented in Figure 15.

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. We see from Figures 15 and 16 that for each ratio of global to local removal time, O has a relatively small variance. This indicates that O can be fixed *a priori*, perhaps once for each image domain. Alternatively, we can use the monotonicity assumption: determining whether the local removal phase should be terminated reduces to obtaining $|R|$ after every (perhaps ith) iteration. This can be done efficiently by using global count on the remaining region accumulators.

5.4 Performance and Comparison

In this section we examine how the Hybrid-REDUCE algorithm is likely to perform using existing technology, and how its relative performance is likely to change with technological advances. Throughout this subsection the operation is assumed to be CountSelected PEs,

Image Name	Ratio of Local to Global Removal Execution Times									
Name	1	2	3	4	5	6	7	8	9	10
amroad0	24	11	11	8	6	6	6	6	6	6
amroad10	14	8	8	8	8	8	6	6	6	6
amroad14	10	7	7	4	4	4	4	4	4	4
amroad15	12	12	8	8	6	6	6	6	6	6
amroad16	10	6	6	3	3	3	3	3	3	3
amroad1	11	9	8	6	5	5	5	5	5	5
amroad25	13	11	11	10	8	7	7	7	6	6
amroad29	15	9	9	7	5	5	5	5	4	4
amroad2	15	13	13	8	7	6	6	6	6	5
amroad33	20	7	7	7	6	6	5	5	5	5
amroad34	12	7	7	7	6	5	5	4	4	4
amroad3	20	15	10	8	8	7	7	7	7	7
amroad5	12	11	9	9	9	7	7	6	6	6
avg road	14.5	9.7	8.8	7.2	6.3	5.8	5.6	5.4	5.3	5.2
house10	9	9	9	5	5	5	5	5	4	4
house14	10	10	8	6	6	5	4	4	4	4
house15	11	6	5	5	5	5	5	5	5	5
house16	12	10	5	5	5	3	3	3	3	3
house17	14	8	7	5	5	5	5	5	5	4
house19	6	6	4	4	4	3	3	3	3	3
house1	10	10	5	5	5	4	4	4	4	4
house20	7	5	5	5	5	5	5	4	4	3
house2	13	12	12	6	6	6	6	6	5	5
house3	14	12	8	7	6	6	5	5	4	4
house4	18	7	6	6	6	5	5	5	5	5
house5	12	12	9	7	7	7	7	7	6	6
house6	15	15	9	8	8	8	6	6	6	6
house7	15	13	13	8	8	8	6	6	6	5
house8	17	9	9	9	9	9	6	6	6	6
avg house	12.2	9.7	7.6	6.1	6.0	5.6	5.0	5.0	4.7	4.5
avg all	13.3	9.7	8.2	6.6	6.2	5.7	5.3	5.2	5.0	4.8

Figure 15: Optimal numbers of local removal operations (O) for different images and ratios of local to global removal execution times (K).

the data to be 32 bit integers, and the array to consist of a 256×256 grid of PEs.

The relative performance of primitive operations on existing technology was obtained by using results from the CAAPP prototype, a 64×64 array of bit-serial processing elements, and through hardware simulation of the full size array. The instruction durations in machine cycles are as follows: coterie communication instructions $t_{Cot} = 10$, GlobalCount $t_{GC} = 20$, and all other instruction, including ALU broadcast and PE arithmetic instructions, $t_{PE} = 1$. Since a single iteration of local and global removal take about 100 and 580 cycles respectively, the ratio $K = 6$. We have used average case values for the local and global loop counts from Figure 16: $O = 4$ and $R = 17$. These are justified because of the small variance of O. Bookkeeping added up to less than 100 cycles and was ignored.

$$FirstTwoReductions \Rightarrow 2 * \log n * (32 * t_{Cot} + 64 * t_{PE}) = 2624 \; cycles$$

$$LocalRemoval \Rightarrow O * \begin{cases} SelectSingleResponder \Rightarrow \log N * (t_{Cot} + 2 * t_{PE}) \\ BroadcastPartialSum \Rightarrow 32 * t_{Cot} \\ CombinePartialSum \Rightarrow 64 * t_{PE} \end{cases} = 3480 \; cycles$$

$$GlobalRemoval \Rightarrow R * \begin{cases} GlobalSelectSingleResponder \Rightarrow 3 * \log N * t_{PE} \\ GlobalCount \Rightarrow t_{GC} \\ BroadcastResult \Rightarrow 32 * t_{PE} \end{cases} = 2652 \; cycles$$

Significant is that the local and global removal phases—the parts of the computation that are not asymptotically optimal—take a similar number of cycles as the reductions executed during the first phase. This means that executing extra reductions as in the SMCC-Reduce algorithm is not likely to improve performance.

Two likely changes in hardware performance will alter the relative performance of the algorithms: the latency of the coterie network broadcast, and the path width of the coterie network bus. The expression

$$cycles = \frac{512}{w}(t_{Cot} + 2) + O(\frac{32}{w} + 16)(t_{Cot} + 2) + R(70 + \frac{32}{w})$$

	Ratio of Local to Global Removal Execution Times (K)									
	1	2	3	4	5	6	7	8	9	10
avg of O	13.3	9.7	8.2	6.6	6.2	5.7	5.3	5.2	5.0	4.8
Sigma of O	4.0	2.8	2.4	1.7	1.6	1.6	1.2	1.2	1.1	1.2
avg of R	9.3	13.4	16.7	21.7	23.6	25.8	28.4	29.2	30.9	32.2
Sigma of R	3.6	4.7	4.9	6.2	7.1	6.9	7.6	7.2	8.0	8.6

Figure 16: Average optimal values over the entire data set for local and global removal operations (O and R) and their standard deviations for different ratios of local to global removal execution times (K).

relates the performance of Hybrid-REDUCE to the two parameters, width $= w$ and coterie communication latency $= t_{Cot}$. The loop counter values O and R are again obtained empirically from Figure 16 where K is determined as follows:

$$K = \frac{(\frac{32}{w} + 16)(2 + t_{Cot})}{70 + \frac{32}{w}}.$$

The resultant changes in performance of the hybrid algorithm are illustrated in Figure17. The improvement due to increasing path-width levels off because SelectSingleResponder is inherently bit-serial.

The range of coterie latency to PE instruction ratios was selected with the following scenarios in mind. The curent ratio is about 10. An increase is likely as the VLSI process improves, while the basic packaging configuration remains as it is. We estimate a factor of 3 maximum relative speedup here over the short term as the design improves, but no greater since clock distribution problems for massively parallel arrays are significant. A decrease in the ratio is likely in future generation machines when the entire array will fit on a wafer (or chip). However, it is very unlikely that the coterie latency will ever be less than the global signal propagation, bounding the minimum of the ratio at 1.

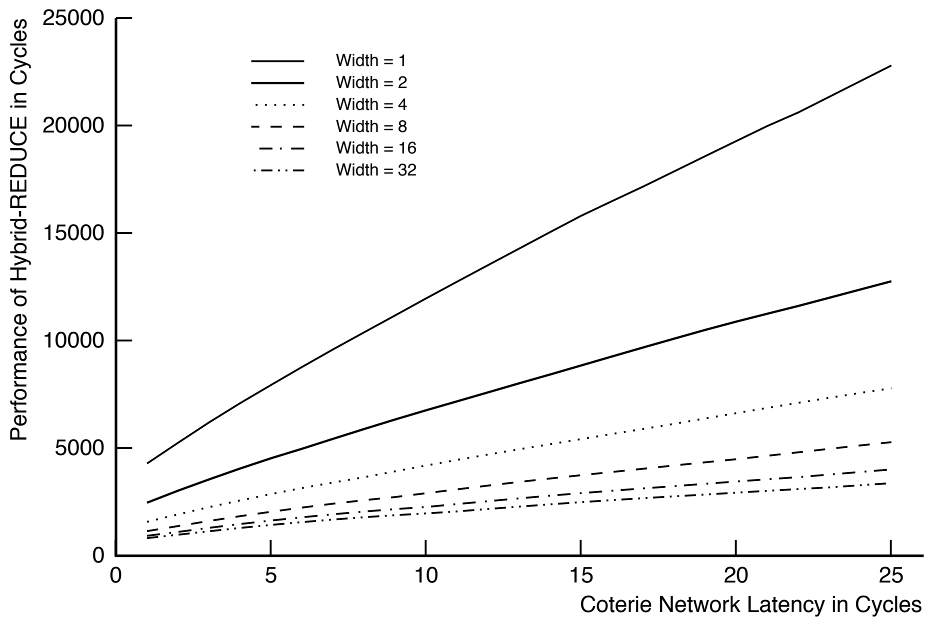

Figure 17: Expected performance of Hybrid-REDUCE given changes in network latency and path width.

6 Conclusion

We have presented a set of new computation techniques for solving problems on reconfigurable broadcast meshes. The method does not use either of the common approaches in this area, i.e. data independent recursive splitting and merging of the original image, or using the network to simulate one or more arithmetic components. Rather, for each subproblem, we use only those PEs to which the pixels of the region have been mapped, and thus avoid the need for remapping that sometimes arises in the latter approach.

We have shown how the coterie structures model can make up for the absence of an efficient general distributed memory access capability through the orchestrated repartition of the induced subgraphs. In this way, certain PRAM results were shown to hold for arbitrary connected components of the reconfigurable mesh as well. We have also shown how to take advantage of coterie broadcast to break symmetries and to obtain information independent of distance.

The algorithms presented are significant from both a theoretical and a practical point of view. The former because they show the possible richness of this computational model, the latter because they are likely to be the fastest available. In fact Hybrid-REDUCE runs within a factor of four of the rectangle reduction algorithm and more than an order of magnitude faster than previous methods. Perhaps the most significant result, however, is that we have shown it is possible to create efficient algorithms "without leaving the coterie"; that irregular graphs need not preclude good solutions for problems on reconfigurable broadcast networks.

We assume that the definitions, basic algorithms, and methodology will be useful in solving many other multiassociative problems. Work is under way using the coterie structures model to solve problems from computational geometry. Other future work includes generalizing the relationship between local and global associative processing.

Acknowledgment

We would like to thank Seth Malitz for many useful conversations and especially for his encouragement in this project.

References

[1] C. Berge, *Graphs and Hypergraphs.* Amsterdam: North-Holland, 1973.

[2] J.R. Beveridge, J. Griffith, R.R. Kohler, A.R. Hanson, and E.M. Riseman, "Segmenting Images Using Localized Histograms and Region Merging," *International Journal of Computer Vision,* Volume 2, Number 3, pp. 311-347, Jaunuary, 1989.

[3] G.E. Blelloch, "Scans as Primitive Parallel Operations," *IEEE Transactions on Computers,* Volume C-38, pp. 1526-1538, November, 1989.

[4] C.R. Brice and C.L. Fennema, "Scene Analysis Using Regions," *Artificial Intelligence,* Volume 1, pp. 205-226, 1970.

[5] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms,* Cambridge, MA: The MIT Press, Cambridge, 1990.

[6] A.D. Falkoff, "Algorithms for Parallel Search Memories," *Journal of the ACM,* Volume 9, Number 4, pp. 488-511, October, 1962.

[7] J.A. Feldman and Y. Yakimovsky, "Decision Theory and Artificial Intelligence I: A Semantics-Based Region Analyzer," *Artificial Intelligence,* Volume 5, Number 4, pp. 349-371, Winter, 1974.

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

[9] E.J. Gauss, "Locating the Largest Word in a File Using a Modified Memory," *Journal of the ACM,* Volume 8, pp. 418-425, 1961.

[10] A.V. Goldberg and S.A. Plotkin, "Parallel $(\delta + 1)$ coloring of constant degree graphs," *Information Processing Letters,* Volume 25, Number 4, pp. 241-245, June, 1987.

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

[12] W.D. Hillis and G.L. Steele Jr., "Data Parallel Algorithms," *Communications of the ACM,* Volume 29, pp. 1170-1183, December, 1986.

[13] K.E. Iverson, *A Programming Language,* New York: John Wiley and Sons, Inc., 1962.

[14] J.-F. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for the Area and Perimeter of Image Components," *Proceedings of the 20th International Conference on Parallel Processing,* Volume III, pp. 280-281, 1991.

[15] R.M. Karp and V. Ramachandran, "A Survey of Parallel Algorithms for Shared-Memory Machines," in *Handbook of Theoretical Computer Science,* Amsterdam: North Holland, 1988.

[16] R.E. Ladner and M.J. Fisher, "Parallel Prefix Computation," *Journal of the ACM,* Volume 27, Number 4, pp. 831-838, October, 1980.

[17] H. Li and M. Maresca, "The Polymorphic-Torus Architecture for Computer Vision," *IEEE Transactions on Pattern Anaylysis and Machine Intelligence,* Volume PAMI-11, pp. 233-243, March, 1989.

[18] H. Li and M. Maresca, "Polymorphic Torus Network," *IEEE Transactions on Computers,* Volume C-38, pp. 1345-1351, September, 1989.

[19] M. Maresca, "Polymorphic Processor Arrays," *IEEE Transactions on Parallel and Distributed Systems,* Volume PDS-4, pp. 490-506, May, 1993.

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

[21] R. Miller, V.K. Prasanna Kumar, D. Reisis, and Q.F. Stout, "Parallel Computations on Reconfigurable Meshes," *IEEE Transactions on Computers,* Volume C-42, pp. 678-692, June, 1993.

[22] A. Montanvert, P. Meer, and A. Rosenfeld, "Hierarchical Image Analysis Using Irregular Tessellations," *IEEE Trans. PAMI,* Volume PAMI-13, pp. 307-316, April, 1991.

[23] P.A. Nagin, A.R. Hanson, and E.M. Riseman, "Studies in Global and Local Histogram-Guided Relaxation Algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Volume PAMI-4, pp. 263-276, May, 1982.

[24] B. Obrenić, M.C. Herbordt, A.L. Rosenberg, C.C. Weems, F.S. Annexstein, and M. Baumslag, "Using Emulations to Construct High-Performance Virtual Parallel Architectures," *Technical Report TR91-40, Department of Computer Science, University of Massachusetts,* 1991.

[25] R. Ohlander, K. Price, and D.R. Reddy, "Picture Segmentation Using A Recursive Region Splitting Method," *Computer Graphics and Image Processing,* Volume 8, pp.

313-333, December, 1978.

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

[27] V.K. Prasanna and D. Reisis, "Image Computations on Meshes with Multiple Broadcast," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Volume PAMI-11, pp. 1194-1202, November, 1989.

[28] D. Rana and C.C. Weems, "The IUA Feedback Concentrator," *Proceedings of the International Conference on Pattern Recognition,* Volume II, pp. 540-544, 1990.

[29] J.M. Tenenbaum and H.G. Barrow, "Experiments in Interpretation Guided Segmentation," *Artificial Intelligence,* Volume 8, Number 3, pp. 241-274, June, 1976.

[30] J.C. Tilton, "Image Segmentation by Iterative Parallel Region Growing With Applications to Data Compression and Image Analysis," *Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation,* pp. 357-360, 1988.

[31] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, J.G. Nash, and D.B. Shu, "The Image Understanding Architecture," *International Journal of Computer Vision,* Volume 2, Number 3, pp. 251-282, January, 1989.

[32] M. Willebeek-LeMair and A.P. Reeves, "Solving Nonuniform Problems on SIMD Computers: Case Study on Region Growing," *Journal of Parallel and Distributed Computing,* Volume 8, Number 2, pp. 135-149, February, 1990.