General Routing on the Lowest Level of the Image Understanding Architecture

Martin C. Herbordt and Charles C. Weems
Computer and Information Sciences Department
University of Massachusetts at Amherst
Amherst, Massachusetts 01003

David B. Shu
Hughes Research Laboratories
3011 Malibu Canyon Road, Malibu, CA. 90265

Abstract

SIMD mesh connected computers have been found to be very useful in many applications, such as those often found in image processing and matrix arithmetic, where the communication among PEs is local, regular, or both. Low efficiency results, however, when the communication patterns are irregular or sparse, a situation we have found to exist in many computer vision applications. This paper describes a routing primitive which makes significant progress towards solving that problem for the CAAPP, a SIMD mesh connected computer enhanced with the broadcast capability of the coterie network. We show that general routing on the CAAPP can be executed with simplicity to the user and performance similar to that of a dedicated network. We present experimental results from several classes of permutations as well as from some common machine vision applications.

1 Introduction

One class of architectures that has proven popular for use in image processing and other domains that map readily onto fine-grained parallel computation is the SIMD mesh connected computer (SMCC). Some of the advantages of mesh connected topology are that it is regular and easy to lay out on a chip, has high bandwidth for data movement along its dimensions and low latency for local transfers; some disadvantages are its large diameter and limited bandwidth when the data movement is not regular. It is therefore apparent why SMCC architectures have found their greatest success in two areas: the first is in modeling certain physical phenomena, such as images, which map naturally onto a mesh and for which many relationships are local; the second is in executing regular mathematical computations, such as matrix operations. Conversely, mesh connected topologies are least efficient when confronted with computations, many of which occur in computer vision, that require communication between distant PEs, especially when the communication pattern is sparse and/or cannot easily be described in terms of nearest neighbor moves.

In this paper we present the ROUTE primitive, a collection of routing algorithms for the Content Addressable Array Parallel Processor (CAAPP)[17], which allow many more classes of interprocessor communication to be executed efficiently than could otherwise on machines with conventional mesh connected topologies. These algorithms use the coterie network to emulate wormhole routing [8, 4] in that packets are not queued at intermediate nodes when progress is not possible, rather they are left in place on their current PE. One of the algorithms, at the cost of extra overhead, also emulates cut-through routing in that it sends packets not just to the next PE each time step, but rather to the next *free* PE, no matter how far away that may be. ROUTE uses the global feedback capability between controller and array to dynamically select between routing algorithms, thus presenting the programmer or language designer with a transparent mechanism for interprocessor communication.

2 The CAAPP

The CAAPP is the low-level processing array of the three-level architecture called the Image Understanding Architecture (IUA) being developed at the University of Massachusetts and Hughes Research Laboratories. The IUA is designed to perform real-time machine vision by combining pixel level, token level, and symbolic computation in one machine. The pixel level processor (the CAAPP) consists of a 512×512 content addressable array of one-bit processing elements (PEs). Each processing element has several general purpose registers, 320 bits of on-chip cache memory, 32K bits of main memory, and an "Activity Register" which is used for branching control. The PEs form a single instruction stream, multiple data stream (SIMD) array, with control provided by the Array Control Unit (ACU) which broadcasts instructions, data addresses, and global data. The controller can also extract information from the array by associative polling, as hardware support is provided for Get-Some/None and Get-ResponderCount operations. The Get-Some/None operation is especially useful in determining whether a data dependent algorithm has completed, while Get-ResponderCount can be used for adaptive algorithms.

Communication between PEs themselves can take place in two different ways: by using the nearest neighbor mesh interconnection network, and via broadcast. In the second method, broadcasting PEs transmit information by writing to a specified register connected to the Some/None circuit. Receiving PEs then read a register which will have been set to the OR of the broadcast signals. Nearest-neighbor moves and broadcast are extremely efficient on the CAAPP: a 32-bit move takes around $3\mu s$ while broadcast from any set of PEs to the entire 512×512 array takes $35\mu s$.

One powerful addition that the CAAPP has over conventional associative processors is the coterie network, used to isolate the propagation of broadcast to specified regions. Each PE in the CAAPP controls a set of switches in four different directions (north, south, east, west) that enable the creation of electrically isolated groups of PEs sharing a local associative Some/None feedback circuit. These switches are set by loading the corresponding bits of the mesh control register in each PE with the

appropriate mask. Because each PE views the mesh control register as local storage, coterie configurations can be loaded from masks stored in memory, or can be based on data dependent calculations. Isolated groups, or coteries, of processors can then respond to globally broadcast instructions in a locally data dependent fashion. For example, when a set of PEs executes a broadcast instruction, the receiving PEs will read the OR of precisely those PEs within the same coterie.

One way the switches of the coterie network can be set is so the columns or rows are isolated. Once this is done, the row or column "buses" can be arbitrarily segmented still further. The coterie network can thus emulate the mesh with reconfigurable buses [12], and the polymorphic-torus [10]. Another obvious use for the coterie network is in finding and labeling connected components. Each PE tests its four neighbors and compares the value with its own: if the values are equal, the PE closes its coterie switch in that direction; if not equal, the switch is opened. Using a standard leader election algorithm, a "master" PE from the newly formed component is selected. The leader broadcasts its unique ID to label the rest of the PEs in the component.

3 Routing on a Mesh

Much work has been done on the problem of routing on a mesh. Both SIMD and MIMD models have been used: In the former no queues are used; in the latter queue size becomes a variable to be minimized. In the following discussion, square meshes are assumed and N refers to the total number of PEs, while $n = \sqrt{N}$ is the number of processors in a row or column. The lower bound on mesh routing is $2n - 2$ on the MIMD model and $4n - 4$ on the SIMD model; this is the minimum number of routing steps required for processors in opposite corners to exchange packets. The difference occurs because, in the MIMD model, different sets of processors can send packets in different directions on the same time step, while in the SIMD model, the direction must be the same for every packet. When there is wrap-around, the lower bounds are halved.

One way to route using the MIMD model is to use a simple greedy algorithm: First route each packet along the column to the correct row, then along the row to the correct column. Packets arriving at the correct rows are ordered in queues so that the ones that need to travel the furthest are given priority. This algorithm takes $2n - 2$ steps, but requires queues of size $\theta(n)$. A randomized routing algorithm due to [16] is an extension of greedy routing. The algorithm consists of three phases; randomize packets within the columns, send packets to correct column along the row, and send packets to correct row along the column. This algorithm will result in routing in $\simeq 3n$ steps with a queue size of $O(\log N)$ with overwhelming probability. [9] has developed a more complex algorithm that is both optimal and uses constant size queues, although "it does not appear that the constant bound on the resulting queue size will be practical for moderate values of n (say, $n \leq 100$)."

Permutation routing can always be accomplished by sorting destination keys; currently this may be the best general-purpose on-line method for routing on the SIMD model. [15, 13] use variations of bitonic sorting to sort a mesh into various standard orderings in $14n+o(n)$ routing steps, which turns out to be a factor of 4.5 from optimal for *any* n. When off-line calculation is allowed, better results can be achieved. [14] presents an algorithm, that with $O(\log^2 N)$ preprocessing time, can route optimally the class of permutations that can be specified by permuting and complementing the bits in the PE ID. [1] presents an algorithm that with preprocessing can route any permutation in $3n$ routing steps on a mesh with wrap-around.

To determine the applicability of these routing algorithms in terms of forming the basis for a general construct, we must first examine more closely exactly what we are trying to accomplish. The routing primitive should:

- require minimal preprocessing;
- be able to route permutations, as well as support intermediate combining of results in many-to-one routing;
- route sparse as well as dense patterns efficiently;

- be able to take advantage of regularities, but not be too susceptible to congestion; and
- perform close to optimally.

The MIMD algorithms can be eliminated immediately because of their need for queues or heaps; simulating dynamic structures on a SIMD processor with no indirect addressing or index capability increases the routing complexity by a factor on the order of the maximum queue length or index offset. Since the MIMD algorithms all require queues of length greater than 4.5, the SIMD sorting methods would be preferable. But SIMD sorting and off-line routing also do not fulfill the need: they do not extend well to combining, nor do they take advantage of sparse routing. More comparisons will be made later, but first we will present a primitive that meets all of the criteria stated above.

4 The ROUTE Primitive

The ROUTE primitive consists of three similar, but distinct, algorithms which are selected dynamically through the use of the ACU get-count command. The basic idea of all three algorithms is to route greedily without the use of queues. Taking wormhole routing for inspiration [8, 3], every PE simulates two channels, X and Y, that are chosen arbitrarily to run in directions parallel to the rows and columns respectively. We will first present three routing algorithms, then the method they are selected, followed by a description of how they have been modified to create a combine operation, and end the section with a randomized version of one of the algorithms.

The mesh greedy routing algorithm (MGRA) runs as follows: A PE uses the nearest neighbor connections to send a packet along the X-channel a distance of one PE per routing step, until the correct X coordinate (column) is reached. At this point the PE moves the packet from its X-channel to its Y-channel. The packet then continues along the Y-channel until the destination is reached. The X- and Y-moves are interleaved so that each occurs on every iteration of the algorithm. Packets travel in only one direction in each channel and wraparound is used, as in

[4], so there will be no deadlock. If the packet has reached the correct X coordinate but the Y-channel at that PE is occupied, then the packet is "blocked", as are all the other packets contiguously behind that packet in the X-channel. Y-channels are never blocked, so overall progress is assured. The critical question in running this algorithm without queues is how to inform those packets which are contiguously behind a blocked packet, that they too are blocked. In normal meshes, this notification step would require n steps, the maximum possible number of blocked packets in a channel, and would yield a $\theta(N)$ algorithm. But by using the broadcast buses of the coterie network communication can be accomplished in $n/50$ machine cycles (the time it takes a broadcast signal to traverse the longest possible row bus), or on the order of a microsecond for a 512×512 array. The details of the notification step are as follows: all contiguous packets in X-channels form coteries or electrically isolated islands. The blocked packets open their left switches so that only packets behind them will receive the message, and then broadcast "blocked bits" to these coteries.

In the second algorithm, the four channel MGRA, two more simulated channels, X2 and Y2, are added to route packets in the opposite direction of X1 and Y1 respectively. The advantage is that packets can now be routed by a shortest path, minimizing the distance packets must travel. The number of iterations should be cut roughly in half. The disadvantage of the four channel MGRA is that the overhead is slightly more than doubled as there are now four ways that the X- and Y-channels can interact, instead of one. In the rest of the paper, the two channel version will be the default unless the number of channels is explicitly mentioned.

The third algorithm, the Coterie Greedy Routing Algorithm (CGRA), again uses two channels. The CGRA differs from the MGRAs in that the coterie network will be used to transfer packets, rather than just status bits. The key difference is that here, rather than moving packets just one PE at a time, all of the open space between occupied PEs is traversed in a single iteration of the algorithm. The mechanism is to create coteries which have the property that the rightmost PE (bottommost if these are Y-channels) contains a packet, while all other PEs in the coterie do not. The occupied PE then broadcasts its packet to the coterie, where it is read either by the

destination or the furthest PE. Clearly the overhead per iteration is much higher for the CGRA than for the MGRA; exactly how much will be discussed below.

Selection among the three algorithms to create the unified ROUTE primitive is effected through use of the get-count directive of the ACU. The CGRA has much higher overhead than the MGRAs, but is very effective if the route is sparse and packets can be sent long distances during most iterations; therefore the CGRA will be called if the density of PEs sending packets is low. Similarly, the four-channel MGRA is used when the maximum distance that any packet must travel is somewhat less than the half the diameter of the torus. The density and the maximum distance can both be calculated in under 20 microseconds, or about one iteration of the MGRA. ROUTE is tuned so that the MGRA is the default; the more data dependent CGRA and four channel MGRA are only selected if it is virtually certain there will be a speed-up.

A combine operation has been created by augmenting the routing algorithms as follows: Instead of simply moving the packets that have arrived at their destinations from the Y-channel(s) to the output buffer, a binary operator is interposed. For example sum-combine adds the value in the packet to the value already in the output buffer. Many-to-one routing is implicit in the combine operation; more congestion is therefore likely to occur than in permutation routing. To deal with this situation, intermediate combining at the point of collision may optionally be executed. The cost is an increase in overhead of an extra compare and arithmetic operation for each cycle, but there are certainly situations where intermediate combining is worthwhile. One example is the degenerate case where the entire array is combined at one destination: the complexity of the combine operation is reduced from $O(N)$ to $O(n)$.

Some results from later in this paper are that the MGRA routes random permutations in slightly more than $2n$ iterations with very small standard deviation, while on non-trivial permutations arising from specific applications the number of iterations ranges from $2n$ to $3n$ iterations. It may be possible, however, that an application exists where the worst case takes longer than $3n$, and for which a highly predictable completion time is required. In this case we can take advantage of the small variance

of the MGRA on random permutations by first randomizing the input packets along one dimension (as in [16]), and then routing to the destination. Since randomizing in one dimension takes n and random permutations take about $2n$ routing steps with very small variance, this algorithm routes all permutations in $3n$ data moves with extremely high probability.

5 Performance

In order to predict the performance of the MGRA and CGRA, variations of these algorithms were simulated at two levels of granularity. The first is a coarse simulation which disregards the number of time-steps needed to perform each iteration. The second is a complete assembly language implementation that was run on the IUA simulator [18].

5.1 Coarse Simulation

The coarse simulation was used to discover relationships among the following parameters: the algorithm used, the number of iterations needed for completion, the width of a side of the torus (n), the number of channels, the density of the route (that is, the percent of the N PEs which send a packet), and the number of times that individual packets were blocked. By disregarding the details of how the PEs actually carry out the bit-serial operations, it was possible to generate a larger number of trials than would be possible using the full simulator, thereby reducing the standard deviation in the averages. Some of these experiments are now described.

- How many iterations do the various algorithms need to complete the route on random permutations, and how does this relate to the width of the torus?

  The MGRA needed a number of iterations approximately equal to the diameter of the torus ($2n$); e.g when $n = 64$ the average number of iterations was 134, when $n = 256$, the average number of iterations was 523. The four-channel MGRA needed slightly more than half that many iterations. In all cases, the

standard deviation was less than 3. The performance of the two channel CGRA was sublinear: the relation $iters = n^{.89}$ fits the curve well, but no underlying structure was found.

- How much speed-up does each algorithm achieve when the number of packets to be routed in random permutations is reduced?

  The performance of the MGRAs remains roughly the same as the density is decreased: there is still a high probability that some packet will need to travel close to the maximum distance. The CGRA achieves a significant speed-up because it takes advantage of the empty space between PEs. The number of iterations needed decreases very rapidly when the density is less than 20%; the break-even point between the CGRA and the MGRA occurs when the density is around 10%. For a 512×512 array, fewer than 60 iterations are required by the CGRA, whereas over 1000 are needed by the MGRA.

- How often is the most-blocked packet blocked during random permutations? What is the average number of times that a packet is blocked?

  The average maximum number of times that any packet is blocked during the running of the MGRA is asymptotic to about 30. The average number of total blocks for all packets is roughly linear with the total number of packets; the average number of blocks per packet stays under 1 for the range tested, i.e. $n \leq 512$.

- How does the MGRA perform on particular permutations?

  The MGRA was tested on numerous particular permutations and some basic results are presented above. The first table contains permutations of the type described in [14], that is, permutations "that can be specified by the permutation and complementing of bits in a PE address." The same notation is also used. The second table contains some permutations that cannot be thus specified.

Some PE ID "Bit" Permutations

Name	Formulation	Iters.
Bit Reverse	[0,1,...,p-1]	$< 2n$
Unshuffle	[p-2,p-3,...,0,p-1]	$< 2n$
Shuffle	[0,p-1,...,1]	$< 2n$
Transpose	[p/2-1,...,0,p-1,...,p/2]	$< 2n$
Shuffled Row-Major	[p-1,p/2-1,...,p/2,0]	$< 3n$
Bit Shuffle	[p-1,p-3,...,1,p-2,...,0]	$< 3n$
Vector Reverse	[-(p-1),-(p-2),...,-0]	$< 2n$
Random Bit	[arbitrary orderings]	$< 3n$
with flipping	[and arbitrary flippings]	$< 3n$

Some Other Common Permutations

Name	Iters.
Random	$\simeq 2n$
Reflection in X	$< 2n$
Reflection in Y	$< 2n$
Snakelike Row-Major Ordering	$\simeq n$
Snakelike Column-Major Ordering	$< 2n$
$90° * k$ rotation	$< 2n$
$45° + 90° * k$	$< 3n$
P-ordered vectors	$\simeq 2n$

Trials were run for $n = 4, 8, 16, ..., 256$ on all particular permutations. For the random permutations, from 100 ($n = 256$) permutations to many thousands were run for each n value.

Whenever the value in the "Iters" column states that less than $2n$ iterations are required to complete the MGRA, this indicates that no packets are blocked for any n tested. For values of $\simeq 2n$ iterations, a small constant more than $2n$ was required, as with the random permutations mentioned above.

The 45° rotation route was calculated by using the standard rotation matrix and is a special case as it is not a permutation. The factors of 45° are the worst cases of all rotations tested ($0° \ldots 360°$ in increments of 5°).

5.2 Implementation and Timing

Three variations of the greedy algorithm were implemented on the full CAAPP simulator [18], the MGRA, the 4-channel MGRA, and the CGRA. Some implementation details are worth mentioning before timing is discussed.

- The CAAPP is a bit-serial processor with a memory hierarchy: Each PE contains 320 bits of on-chip memory, any of which can be transferred to a nearest-neighbor PE in a single cycle, and 32K bits of off-chip memory. Therefore, execution times for the algorithms are linear in the number of bits in the message up to 120; for longer messages, the execution time per message length increases slightly.

- The execution time of the broadcast instruction is linear with respect to the diameter of the region in which the message is being sent. On the CAAPP, each bit propagates 50 PEs per instruction cycle. Therefore, while the broadcast instruction is technically $O(\sqrt{n})$, in practice it requires at most 6 cycles per bit on a 256×256 array. To simplify timing comparisons between simulations on different sized arrays, it is assumed that all broadcast instructions require the worst case of 6 cycles per bit, regardless of the diameter.

- Global feedback between ACU and array is used to determine whether the algorithms have completed and to increase performance. At the end of every iteration, the ACU executes get-SOME/NONE operations on the X- and Y-channels (each taking 2 microseconds); if the X-channels are empty, then they need no longer be simulated; if the Y-channels are also empty, then the algorithm has finished.

Presented are average times in milliseconds for random permutation routes.

2-channel MGRA

number of bits	width of torus	time
1	64	1.31
1	256	4.65
8	64	1.74
8	256	6.29
16	64	2.31
16	256	8.42

The 4-channel MGRA yielded results similar to the 2-channel MGRA, but has the advantage that the times are proportional to the maximum distance that any packet must travel, rather than to the diameter.

4-channel MGRA

number of bits	maximum distance	time
16	10	.50
16	15	.72

The CGRA is dependent both on the width of the torus and the density, so results are given with respect to both of these quantities. The last entry indicates that for very low densities, however, the expected time is roughly independent of the torus width.

CGRA

number of bits	width of torus	density	time
16	64	10%	1.95
16	64	4%	1.41
16	256	10%	4.83
16	256	4%	2.81
16	≤ 300	$\leq 5 * n$	.74

We compare these results to a machine with a dedicated routing network: The 256×256 version of the Connection Machine II has running times of 500 micro-seconds for 32-bit permutations, and 80 micro-seconds for nearest neighbor permutations [11]. Therefore, the CAAPP running the greedy routing algorithms on random permutations has running times from roughly equivalent to an order of magnitude slower than

the CM-2. When the CAAPP executes nearest neighbor permutations, the running time is less than 2 microseconds for 16 bits, or more than an order of magnitude faster than the CM-2.

6 Applications

In this section we will present some applications on the CAAPP for which inter-processor communication plays a significant role. This will give an idea of where the ROUTE primitive is useful, where the nearest neighbor moves are sufficient, and which algorithm of ROUTE was used. One result of this presentation is that the results from the previous section on random permutations extend to particular cases; in all applications tried so far, ROUTE terminates after at most $3n$ iterations.

Window operations. Local thinning and the convolutions used in differential edge detection are some of the many applications that use communication between PEs and a well specified neighborhood or 'window'. It is these sorts of operations that SMCCs perform extremely efficiently using nearest neighbor moves, and that is also the best way to execute window operations on the CAAPP. For example, a 3×3 Sobel operator can be executed, using a standard optimization, in 6 nearest neighbor moves, each taking only a few micro-seconds.

FFT. The FFT actually has two distinct sections, the initial phase where data is combined, and the final phase where the results are routed back, or unscrambled. The first phase requires very dense long-distance communication, as every PE in the array sends data 1, then 2, then 4, ... up to $n/2$ PEs away in both dimensions. This phase requires a total of $4n$ nearest neighbor moves. The unscramble phase uses the MGRA. Packets are never blocked for diameters tested ($n \leq 512$) so the number of iterations is always less than $2n$.

Some Matrix Operations. Matrix transpose, and reflection in the X and Y axes all used the MGRA part of ROUTE. As in the FFT unscramble, these permutations are non-blocking and so require less than $2n$ iterations.

Image Rotation. Although rotating an image is not trivial because of aliasing problems, effective methods can be constructed where most of the complexity consists of moving packets to locations generated by the rotation matrix. Rotations about the center were tested in increments of 5 degrees on several diameters and three basic results obtained. First, for small rotations, the 4 channel MGRA was used and the number of iterations was small: for a 64×64 mesh the number of iterations requires for 5, 10, 15, and 20 degrees is 10, 16, 20, and 27 respectively. Second, for rotations of 90, 180, and 270 degrees, the MGRA does not block, and therefore less than $2n$ iterations are required. Third, the worst case for the MGRA occurrs for angles of 45, 135, 225, and 315, where up to $3n$ iterations are needed.

Ray Tracing. In lens design evaluation the problem arises of determining the distribution in the focal plane of rays passing through the lens from a point source. The PEs represents a point both on the lens and in the image plane. Each PE computes the path that a ray will take through "its" point on the lens and determines the address in the image plane where that ray will strike. The CAAPP then uses a sum-combine to route this information and obtain the result. In the case of an ideal lens, where all rays converge on one point, ROUTE takes $2n$ iterations with intermediate combining. In real designs, 140 to 150 iterations were needed on a 64×64 image.

Hough Transform. One algorithm for performing the Hough transform on the CAAPP runs as follows: The input consisting of an image plane (X,Y), usually a bit plane of thresholded edge pixels, and the Hough plane (ρ,θ) are both mapped to the rows and columns, respectively, of the mesh of PEs. For every θ from 0 to 180 degrees for I increments $d\theta$, $\cos(\theta_i)$ and $\sin(\theta_i)$ are broadcast by the controller. Each PE which is occupied in the (X,Y) plane must calculate the ρ value $\rho = X\cos(\theta_i) + Y\sin(\theta_i)$ and then send a bit to the PE at (ρ,θ_i), which adds it to the total already there. The Hough transform can be viewed as a series of I column-histogram operations, one for every θ_i. Each column-histogram is executed in two steps; first a sum-combine operation is executed to get values from (X,Y) along the X axis to the correct ρ value, that is, to (ρ,Y). Then, since all values in the various (ρ,Y) will be going to the same column, that is, the same θ, a row parallel prefix is executed in $\log n$ routing steps.

Region Characterization. (This example and those from the next few sections do not indicate the way that these algorithms are implemented on the IUA, but rather show how ROUTE could be used on a processor with reconfigurable buses but not coteries.) In order to decide whether to merge regions it is necessary to collect, in a "master" PE of that region, information such as number of points, number of border points, average and extremal spectral values, and other quantities. Characterization can be implemented using a simple leader election algorithm ($\log n$ steps) to select a region master, and the four channel MGRA with sum-combining. The number of iterations is very close to the maximum distance from any point of a region to its master PE as most collisions are handled with intermediate combining.

Region Merging. Assume a region merging algorithm, with preliminary regions already selected. Assume further that each region has elected a master PE which has gathered information about its own region and has determined the leaders of the neighboring regions. It is then necessary for the master PEs to communicate with all of the surrounding masters in order to determine whether or not to merge. If a few thousand initial regions are selected in a 256×256 image, then the density of communicating PEs will be small and the CGRA efficient. As merging progresses, the density will become smaller and smaller and only a few iterations of the CGRA will be needed for communication.

Local Histogramming. Obviously global histogramming can be implemented using sum-combine of the MGRA. But in one segmentation algorithm [2], local histogramming is used to extract information about subgrids of an image, typically 32×32, and create regions on the basis of that information. Sum-combine with the four channel MGRA can also be used for this procedure.

Convex Hull. On the CAAPP it is possible to find the convex hull of any number of sets of points simultaneously, as long as they are all members of the same region. A variation on the Graham scan [5] can be implemented which requires communication between a master PE and hull elements. In most cases, the density of hull points to total points is small, so that the CGRA will be efficient.

The applications just presented represent only a small fraction of the possible

uses for the ROUTE primitive, but we emphasize three points. First, in dense, long distance, permutations the MGRA usually terminates in $2n$ iterations. Second, in no case did the MGRA take more than $3n$ iterations. And third, many applications have been found which use sparse or dense, nearby communication, and for which therefore the CGRA and four channel MGRA are preferable. Times for these algorithms are highly data dependent, but will always be less than those for the MGRA on the same application; often the times are substantially better.

7 Conclusions and Future Work

We have presented a routing primitive that is easy to use and that performs favorably to other available methods in many respects. One drawback is that these are not optimal algorithms: therefore, if maximum performance is required for a permutation known a priori, and for which an optimal route can be derived (e.g. bit permutations of PE IDs [14]), then naturally the optimal route should be implemented. However, ROUTE has also performed well on the bit permutations tested, as well as for many other applications including those requiring irregular, sparse, and many-to-one communication. Also, the randomized routing method (described at the end of section four) bounds the communication time to $3n$ MGRA iterations, although its use has not yet proved necessary. Perhaps just as important as performance, ROUTE gives the programmer and language designer a unified construct to handle all interprocessor communication that cannot easily be coded with nearest-neighbor moves.

A research problem that remains is a formal method for determining when to use nearest neighbor moves and when to use ROUTE. This issue is tied to a basic question of SIMD parallel processing, whether or not the mapping of data to PEs should be transparent to the programmer. In the primary CAAPP application of low-level machine vision, where the data often consist of pixels corresponding directly with PEs, we much prefer to know how our data are mapped. Corollaries of knowing the mapping are that nearest neighbor moves are easy to conceptualize and implement, and that is almost always always obvious when the nearest neighbor connections and when general routing should be used. Needless to say, we view this flexibility as a

great advantage.

One overall conclusion that can be drawn from this work is that the coterie network extends the applications the CAAPP can handle efficiently beyond SMCCs and into the range of SIMD machines with dedicated general routing networks. Obviously a dedicated routing network will better handle some classes of communication, the CM II routes random dense permutations more than ten times faster than ROUTE on the CAAPP does. But conversely, sparse and nearby permutations are handled with similar speed, and most importantly, no sacrifice is made in the areas where the CAAPP excels: nearest neighbor and broadcast communication. Therefore, in applications domains such as low-level machine vision where most of the inter-PE communication is sparse or involves neighborhoods, the cost of a dedicated routing network may not be justified if you have coteries.

Acknowledgments

We would like to thank Arny Rosenberg, Jim Burrill, Mike Rudenko, and Mike Scudder for their helpful comments. This work was supported in part by the Defense Advanced Research Projects Agency under contracts DACA76-86-C-0015, and DACA76-89-C-0016, monitored by the U.S. Army Engineer Topographic Laboratory; by the Air Force Office of Scientific Research, under contract F49620-86-C-0041; and by a Coordinated Experimental Research grant from the National Science Foundation (DCR 8500332).

References

[1] F. Annexstein and M. Baumslag (1990): "A Unified Approach to Offline Permutation Routing," *2nd ACM Symp. on Parallel Algorithms and Architectures.*

[2] J.R. Beveridge, J. Griffith, R.R. Kohler, A.R. Hanson, E.M. Riseman (1989): "Segmenting Images Using Localized Histograms and Region Merging," *International Journal of Computer Vision,* 2.

[3] W.J. Dally and C.L. Seitz (1986): "The Torus Routing Chip," *Distributed Computing,* 1 (3).

[4] W.J. Dally and C.L. Seitz (1987): "Deadlock Free Routing in Multiprocessor Interconnection Networks," *IEEE Trans. on Comp.,* 36 (5).

[5] R.L. Graham (1972): "An Efficient Algorithm for Determining the Convex Hull of a Planar Set," *Information Processing Letters,* 1.

[6] M.C. Herbordt, C.C. Weems, D.B. Shu (1990): "Routing on the CAAPP," *Proceedings of the 10th Int. Conf. on Pattern Recognition.*

[7] M.C. Herbordt, C.C. Weems, J.C. Corbett (1990): "Message Passing Algorithms on a SIMD Torus with Coteries," *Proceedings of the 2nd ACM Symposium on Parallel Algorithms and Architectures.*

[8] P. Kermani and L. Kleinrock (1979): "Virtual Cut-Through: A New Computer Communication Switching Technique," *Comp. Networks,* 3.

[9] F.T. Leighton, F. Makedon, I. Tollis (1989): "A $2n-2$ Step Algorithm for Routing in an $n \times n$ Array With Constant Size Queues," *1st ACM Symp. on Parallel Algorithms and Architectures.*

[10] H. Li and M.Maresca (1987): "Polymorphic-Torus Network," *Proc. International Conference on Parallel Processing.*

[11] J.J. Little, G.E. Blelloch, T.A. Cass (1989): "Algorithmic Techniques for Computer Vision on a Fine-Grained Parallel Machine," *IEEE Trans. on PAMI,* 11 (3).

[12] R. Miller, V.K. Prasanna Kumar, D. Reisis, Q.F. Stout, (1988): "Meshes With Reconfigurable Buses," *Proc. of the MIT Conference on Advanced Research in VLSI.*

[13] D. Nassimi and S. Sahni (1979): "Bitonic Sort on a Mesh-Connected Parallel Computer," *IEEE Trans. on Comp.,* 28.

[14] D. Nassimi and S. Sahni (1980): "An Optimal Routing Algorithm for Mesh-Connected Parallel Computers," *J. of the ACM,* 27.

[15] C.D. Thompson and H.T. Kung (1977): "Sorting on a Mesh Connected Computer," *Comm. of the ACM,* 20 (4).

[16] L.G. Valiant and G.J. Brebner (1981): "Universal Schemes for Parallel Computation," *13th ACM Symp. on the Theory of Computing.*

[17] 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.

[18] C.C. Weems and J.R. Burrill (1990): "The Image Understanding Architecture and its Programming Environment," in *Parallel Architectures and Algorithms for Image Understanding,* V.K. Prasanna Kumar, ed. Academic Press, Orlando, Florida.