PARALLEL IMAGE THINNING AND VECTORIZATION ON PASM

James T. Kuehn
Jeffrey A. Fessler
Howard Jay Siegel

PASM Parallel Processing Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907 USA

Abstract

Two key steps in many pattern recognition tasks are image thinning and vectorization. Parallel forms of image thinning and vectorization algorithms are developed and analyzed in the context of a complete image raster to vector conversion process. Two image thinning algorithms are presented: a "peeling" algorithm using Arcelli's masks and the other a hybrid scheme that combines a "distance-measurement" and "peeling" algorithm to achieve a better average performance. The vectorization algorithm scans a thinned binary image, identifies line junctions and endpoints using template matching, and performs vectorization of curves using Ramer's polygonal approximation algorithm. Parallel techniques are shown to offer the possibility of improved execution time as compared to serial algorithms.

I. Introduction

The processing of large amounts of data, the need for real-time computation, the use of computationally-expensive operations, or other demands that would make a task prohibitively expensive to perform on a conventional computer system have forced computer architects to consider parallel/distributed computer designs. Applications that have one or more of these characteristic "demands" include image analysis for automated photo reconnaissance, map generation, and robot (machine) vision. Image thinning and vectorization are key steps in many of these pattern recognition applications. In this paper, parallel forms of these algorithms are developed and analyzed in the context of a complete image raster to vector conversion process. These two algorithms are good candidates for parallelism because the image pixel data can be easily distributed among processors and because each pixel in the image must be repeatedly examined. While the parallel algorithms are more complex than their serial counterparts, they are shown to offer the possibility of improved execution time. Furthermore, interprocessor communication needed for these algorithms is limited to nearest-neighbor communication which allows a variety of parallel machines to use them.

In this paper, an *image raster* refers to a rectangular tessellation in which pixel positions are identified by a (row, column) coordinate. Each pixel has an intensity value (grey level). The first step in a raster to vector conversion task is *segmenting* image pixels into two classes: "object" and "background." Histogramming is used to determine the distribution of grey levels in the image. Depending on the characteristics of the histogram, the image may be able to be segmented by simple thresholding, by an adaptive technique such as edge-guided thresholding [13, 23], or by more complex classification methods [25]. Parallel algorithms for histogramming [10, 20], edge-guided thresholding [26], and other methods [19] have been studied. The result of such a segmentation is a *binary image raster* where the 1-valued points correspond to "object" and the 0-valued to "background."

The application dictates whether "outlines" of objects or "skeletons" of objects are to be found. Both transformations reduce the amount of data to be stored by characterizing the shape of the features in the image. Generally, this simplifies later procedures used for recognition and classification of features. A parallel contour tracing algorithm [26] can be used to find outlines. Thinning algorithms [2, 9] are used to reduce elongated objects to one-pixel thick figures. In either case, it is often desirable to reduce the resulting curved "lines" to a set of *vectors* that form a piecewise-linear approximation of the original curve. The vectors need not match the curve exactly; they may be an estimation of it assuming some tolerance, e.g., ± 1 pixel.

Two image thinning algorithms are presented; a "peeling" algorithm using Arcelli's masks [2] and the other a hybrid scheme that combines Rosenfeld's "distance-measurement" [17] and the peeling algorithm to achieve a better average performance. The vectorization algorithm scans a thinned binary image, identifies line junctions and endpoints using template matching, and performs vectorization of curves using Ramer's polygonal approximation algorithm.

This research is motivated by several ongoing projects at Purdue. One is the design of the PASM multimicroprocessor system and the implementation of a 30-processor prototype system [12]. Another is the study of the use of parallel processing for image processing and pattern recognition with special emphasis on automated mapping applications.

Parallel processing models are given in Section II. In Section III, parallel thinning algorithms are described. Parallel vectorization is discussed in Section IV.

II. Parallel Processing Models

Two types of parallel processing systems are single instruction stream - multiple data stream (SIMD) machines and multiple instruction stream - multiple data stream (MIMD) machines [7]. An *SIMD machine* typically consists of a *Control Unit (CU),* an interconnection network, and $N=2^n$ *Processing Elements (PEs),* where each PE is a processor/memory pair (Figure 1). The PEs are numbered from 0 to $N-1$ and each PE knows its number (address). The CU broadcasts instructions to the processors and all enabled processors execute the same instruction at the same

This research was supported by the Rome Air Development Center under contract number F30602-83-K-0119 and by an IBM Graduate Fellowship.

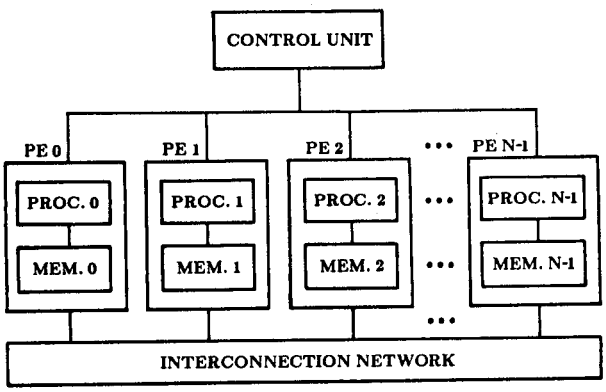

Figure 1. Model of an SIMD/MIMD machine.

time. Each processor operates on data taken from a memory to which only it is connected. The interconnection network allows interprocessor communication. Examples of such machines are the Illiac IV [5], STARAN [3], CLIP4 [6, 8], and MPP [4]. An *MIMD machine* has a similar organization, but each processor may follow an independent instruction stream. As with SIMD architectures, there is a multiple data stream and an interconnection network. Examples of such machines are C.mmp [27] and Cm* [24]. A *partitionable SIMD/MIMD system* is a parallel processing system which can be structured as one or more independent SIMD and/or MIMD machines (e.g., PASM [20], TRAC [18]).

PASM is a partitionable SIMD/MIMD system [20] being designed to have N=1024 PE. The PASM prototype's interconnection network is a circuit-switched implementation of the Extra Stage Cube network [1] which is a fault-tolerant version of the multistage Cube network [21]. When using a circuit switched network, prior to any message transmission between a source and a destination PE, a physical network connection between them must be established. Because the PEs do not share any global memory, all inter-PE communication is done through the interconnection network. In SIMD mode, data is exchanged between PEs during explicitly-programmed steps that all PEs perform simultaneously. In MIMD mode, messages are used to request data and to achieve synchronization. Incoming messages interrupt the destination PE and allow it to respond to asynchronous requests.

III. Line Thinning

Depending on the scan resolution of a map, picture, or camera image, lines may be from one to many pixels "thick." Practical vectorization algorithms require one-pixel-wide lines as input, thus, line thinning is an important step in the conversion process.

Two basic types of algorithms are being employed to thin lines. *Distance measurement* is a thinning algorithm that determines each pixel's distance from the edge of the line to which it belongs. Pixels with locally maximum distances are retained in the thinned line; non-local maximum pixels are removed [17]. On the first pass, the image is scanned row-wise from the upper left-hand corner to the lower right-hand corner. Each 1-valued pixel in the image I is assigned a new value according to:

$$I(row,column) = \min(I(row-1,column), I(row,column-1))+1$$

which calculates each pixel's distance from the left or top side of a line. On the second pass, the image is scanned row-wise from the lower right-hand corner to the upper left-hand corner. Each nonzero-valued pixel is assigned a new value according to:

$$I(row,column) = \min(I(row+1,column)+1, I(row,column+1)+1, I(row,column))$$

which calculates each pixel's distance from the right or bottom side of a line, or the previously-calculated distance from the left or top side if shorter. A third pass identifies and sets to zero non-local maxima points that do not cause the line to become disconnected. A point is not a local maxima if I(row, column) has a neighbor with value I(row, column)+1. Templates (3-by-3 windows) are used to determine if the line would become disconnected if the center point were removed. When the resulting image is used in later processing steps, pixels that are non-zero are taken to be 1-valued.

Peeling is another thinning algorithm in which pixels are "peeled" (removed) from line edges until a one-pixel-wide line remains. Arcelli's algorithm [2] peels pixels by comparing each 1-valued pixel and its neighbors in the image with a set of templates. The templates used to thin 8-connected lines are shown in Figure 2 [9]. In the templates, zeros must match 0-valued pixels, ones must match 1-valued pixels, and asterisks can match either 0- or 1-valued pixels in the image. The algorithm works on two images, the "current" image to which it compares templates and a "working" image of the same size which it updates when templates are matched. Initially, the current image and the working image are identical copies of the original input image. To begin, template A1 is compared with all 1-valued pixels and their neighbors in the current image. If a match is obtained, the corresponding central pixel of the working image is deleted (changed to a 0-valued pixel). After processing with template A1, the current image is discarded, the working image becomes the new current image, and a new working image is obtained by copying the new current image. The process is repeated with template B1, then with A2, B2, A3, B3, A4, and B4, in that order, forming a complete cycle. When no pixels are removed during the processing of a complete cycle, the procedure ends.

For a fixed image size, the processing time for the peeling algorithm is line-width dependent because thick lines require more passes over the data while the distance algorithm requires a fixed amount of time. The distance algorithm is much faster than the peeling algorithm due to the smaller number of operations involved for each 1-valued pixel. Both thinning algorithms leave extraneous pixels, i.e., those that could be removed without altering connectivity or shortening lines. The peeling algorithm leaves very few extraneous pixels, usually at complicated line junctions. The distance algorithm often leaves lines that are two pixels in width, which is unacceptable without further processing. A more sophisticated peeling algorithm which employs additional templates [9] to remove all extraneous pixels has been programmed for the CLIP-4 SIMD computer. Its execution time is slower than Arcelli's original procedure because many more templates must be applied.

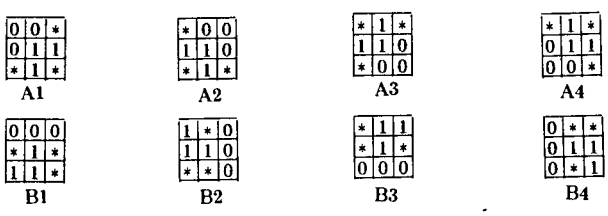

Figure 2. Arcelli's templates for thinning images [9].

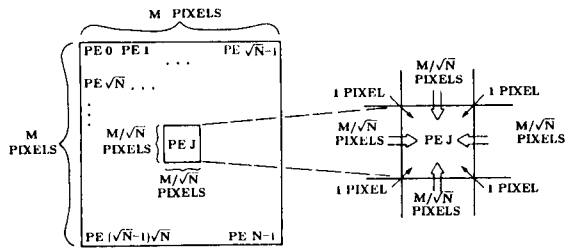

Figure 3. Data allocation and subimage border data transfers into PE J.

Parallel distance algorithm

Consider the implementation of the distance algorithm on an N-PE machine logically arranged as an array of $\sqrt{N}$-by-$\sqrt{N}$ PEs as shown in Figure 3. Each PE stores an $M/\sqrt{N}$-by-$M/\sqrt{N}$ *subimage* block of the original M-by-M image I. Specifically, PE 0 stores the pixels in columns 0 to $(M/\sqrt{N})-1$ of rows 0 to $(M/\sqrt{N})-1$, PE 1 stores the pixels in columns $M/\sqrt{N}$ to $2(M/\sqrt{N})-1$ of rows 0 to $(M/\sqrt{N})-1$, and so on.

During the first pass of the distance algorithm, the calculations for a 1-valued pixel at coordinate (row, column) depend on the values of the pixels above and to the left of it. Because each PE has only the data for its own subimage in its memory, results need to be transmitted as they are calculated to neighboring PEs below and to the right. Whenever a PE obtains a result for a pixel in the last column of a subimage, the result is passed to the PE on its right. By analogy, results obtained for pixels in the last row of a subimage are passed to the PE below. For the second (reverse) pass, results are passed to PEs above and to the left.

The straightforward implementation of the parallel distance algorithm is inefficient since at most $\sqrt{N}$ PEs can be doing calculations at once for the forward or reverse distance-calculating passes. Each pass requires M^2 calculation steps for a serial processor and

$$\frac{M^2}{\sqrt{N}} - \frac{M}{\sqrt{N}} + M$$

calculation steps for an N-PE machine. This can be derived by calculating the time (step number) at which each pixel is processed. If PE 0 begins processing the pixel in row 0, column 0 at time 0, it processes the pixel in row 0, column $(M/\sqrt{N})-1$ at time $(M/\sqrt{N})-1$ and the pixel in row 1, column 0 at time $M/\sqrt{N}$. Also at time $M/\sqrt{N}$, PE 1 may begin processing the pixel in row 0, column $M/\sqrt{N}$ because the results from the PE on its left (PE 0) have been obtained. It can be seen that the pixel in row M−1, column 0 is processed at time $(M/\sqrt{N})(M-1)$ and the pixel in row M−1, column M−1 is begun M−1 time units after this (and completed after an additional time unit). This yields the result stated above.

In the best case with $N=M^2$ PEs (one pixel per PE), each pass would take 2M−1 steps. The computational *speedup* (1-PE computation time / N-PE computation time) for this case would be

$$\frac{M^2}{2M-1} \simeq \frac{M}{2} = \frac{\sqrt{N}}{2}.$$

Because the ideal computational speedup is N, this indicates that the PEs are being used inefficiently. The speedup for practical numbers of PEs ($N \ll M^2$) is even worse.

Better speedups can be realized by allowing PEs to individually process pixels that do not depend on previously-calculated results from PEs above and to the left of them (or below and to the right). For example, note that 0-valued pixels remain 0-valued regardless of their neighbors and 1-valued pixels such as points A and B in Figure 4 remain 1-valued since one of the pixels above or to the left of them is known to be 0-valued.

An improved algorithm is stated as follows. Using the data allocation discussed earlier, each PE begins by passing its rightmost subimage column to the PE on its right and its bottommost subimage row to the PE below it. Transfers between different PEs occur simultaneously; e.g., when PE J−1 sends its upper right corner pixel to PE J, PE J sends its upper right corner pixel to PE J+1, PE J+1 sends its upper right corner pixel to PE J+2, etc. Cube-type networks can perform these transfers in a single step. After the $2(M/\sqrt{N})$transfers (the bottom right-hand corner is transferred twice), PEs independently calculate distances wherever they can do so. Points such as C, D, and E (Figure 4) require results from other PEs in order to be calculated; they are placed in a "do later" list in that PE. As described earlier, when results are obtained for rightmost columns or bottommost rows, these are passed to the PE to the right or below. Only nonzero-valued pixels need to be passed. As each updated result is received, items from the "do later" list are removed and calculated. An analogous procedure is performed in the reverse direction. For sparse images (relatively few 1-valued points and/or thin lines) the performance of this scheme approaches an ideal speedup.

A part-SIMD, part-MIMD mode algorithm is the best formulation of the improved algorithm: the initial passing of border points is done in SIMD mode while the calculations and passing of updated values are performed in MIMD mode. SIMD mode is preferred for the initial communication of border information because the interconnection network transfers are accomplished with explicitly programmed steps in the SIMD program which are executed synchronously. This is more efficient than asynchronous (MIMD) transfers because the destination PEs are not interrupted when incoming data arrives (eliminating the overhead due to interrupt-handling software) and there is no possibility of interconnection network conflicts. A conflict can occur when two or more data items wish to use the same internal network switching element or data link simultaneously. MIMD mode is preferred for the calculations and communication of updated values because it provides the flexibility to skip certain calculations and return to them later when the data becomes available. The ability of PASM to dynamically switch between the SIMD and MIMD modes of parallelism is used in this case to most efficiently perform each part of the algorithm.

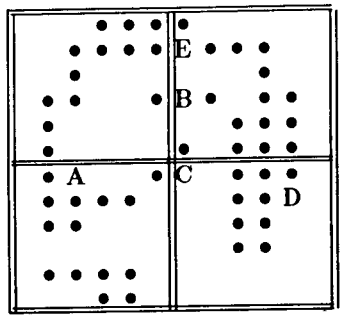

Figure 4. Enhanced distance algorithm; 1-valued pixels are denoted by dots or letters. Each of the four PEs has a subimage of six by six pixels. Pixels A and B can be processed immediately; C, D, and E require results calculated by other PEs.

The algorithm performance also depends on N, the number of PEs. For a fixed input image size, as N increases, the subimage size decreases. In general, the algorithm execution time will decrease as N increases because each PE processes fewer pixels. However, for smaller subimages, the ratio of pixels that are on subimage edges (and thus participate in inter-PE transfers) to the total number of pixels increases. This increases the proportion of time spent in overhead (transfers) and hence decreases the speedup. The optimal number of PEs is data dependent: larger N for sparse images; smaller N for less sparse images to reduce the number of points placed on the "do later" lists. In the worst case, the amount of parallelism available to the straightforward and the improved versions of the distance algorithm is comparable. However, the actual worst-case performance of the improved algorithm may be poorer than that of the straightforward algorithm due to the overhead in handling the "do later" lists.

The final pass of the distance algorithm involves the determination and removal of non-local maxima. To begin this pass, PEs exchange subimage border information in SIMD mode as shown in Figure 3. Then in MIMD mode, each nonzero-valued pixel is examined. If it is not a local maxima and its removal would not cause disconnection of the line, it is removed from the image. The speedup here is data-dependent: it approaches N for those images that result in an equitable distribution of the workload among the PEs.

Parallel peeling algorithm

The peeling algorithm uses the same data allocation and border information from adjacent PEs as shown in Figure 3. After the exchange of border information, each PE thins the lines in its subimage independently. PEs begin at the upper-left-hand corner of their subimages and scan along the columns of row 0, then row 1, and so on. Each time a 1-valued pixel is encountered, a template is applied to determine whether the 1-valued pixel can be removed. The templates were shown in Figure 2. The first set of templates matches pixels that are on the "top" of lines; the second set matches pixels that are on the "right side" of lines; and so on. The templates are applied in series, causing wide lines to be stripped of one layer of pixels at a time. After the complete cycle of templates (one layer of pixels removed from all "sides" of lines), PEs again exchange border information and the complete process begins again at the upper-left hand corner of the image. The communication of border information is required after each cycle because a border pixel removed by a PE in one cycle will affect the results an adjacent PE calculates in the next cycle. The process ends when no more pixels can be removed.

The algorithm can be structured for all-SIMD, all-MIMD, or part-SIMD, part-MIMD mode processing. At first glance, it would seem that the all-SIMD approach would be very inefficient since there are a differing number of 1-valued points in each PE, they occur at different locations within the subimage, and the templates need to be applied only at the 1-valued points. Because it is likely that *some* PE will have a 1-valued pixel at each local subimage coordinate and because PEs are completely synchronized in SIMD mode, PEs with 0-valued pixels will be idle while the PEs with 1-valued pixels apply the templates at each step. (In the MIMD algorithm, PEs would independently skip over 0-valued pixels.) However, in SIMD mode the PEs need not perform the control instructions needed to match the template, as would be required for the MIMD algorithm. (Control instructions include loop counting, branching, and local index calculations.) This is because the CU would perform these tasks, overlapping its operation with the PEs' computations. Furthermore, the exchange of border information is performed efficiently in SIMD mode using synchronized interconnection network transfers.

The protocol for communication and achieving synchronization between cycles for the all-MIMD case is as follows. As each PE completes the processing for a cycle, it transmits all of its border pixels to its neighbors. When a PE has received border pixels from all of its neighbors, it may begin processing the next cycle. When a PE completes a cycle that removed no pixels, it sends all of its neighbors a message indicating such and stops. Neighboring PEs will then know that the border points received from that PE are final results and will not change. Final results can be used over and over by the neighboring PEs that are still working. The algorithm is complete when all PEs stop.

In the part-SIMD part-MIMD scheme, the computation during each cycle is performed in MIMD mode and the communication is done in SIMD mode. PEs synchronize after each cycle and revert to SIMD mode to exchange border pixels.

The all-SIMD approach performs best for densely packed lines because a large fraction of the PEs are doing useful work at each step, the CU and PE operations may be overlapped, and the SIMD communication efficiency is better than that of MIMD. Sparse images in which the data are evenly distributed among the PEs are best processed by the part-SIMD part-MIMD algorithm. It combines the advantages of applying templates to only the small number of 1-valued points, high communication efficiency, and short waiting times encountered by PEs waiting to synchronize between cycles. The all-MIMD approach works best for sparse subimages with disparate numbers of object pixels.

Parallel hybrid thinning algorithm

Analysis of these parallel thinning algorithms prompted the development and testing of a "hybrid" algorithm consisting of both the distance and peeling approaches. In the hybrid algorithm, the "distance" algorithm would be used to locate the approximate "centerline" and remove the bulk of the unwanted pixels in a fixed number of passes. Then, the simple [2] or extended [9] peeling algorithm would be used to remove the remaining extraneous pixels. Because the peeling algorithm removes only one pixel layer at a time, it is inefficient in the early passes when lines are still thick. Therefore, the use of a more efficient algorithm for removing pixels in the early passes achieves a higher average performance. Unless *a priori* knowledge of the image characteristics dictate use of a certain thinning algorithm, the hybrid approach is preferred.

IV. Parallel Vectorization

Many computer image processing algorithms require that objects in images be represented by sets of straight line segments (vectors). Vector format is conceptually more familiar, commonly used for display devices, and requires significantly less storage space.

The vectorization algorithm described here assumes a thinned, binary image is available. If the thinning was not ideal so that some thick junctions and spurious points remain, the algorithm will produce extra short lines. Gap removal, kink straightening, and other "beautifying" algorithms are not considered here. The data allocation among PEs is just as described for the thinning algorithm; therefore, each PE can process the same subimage for the complete raster to vector task.

The vectorization algorithm consists of three main elements: line end/junction identification and line following [15], topological reconstruction [15], and iterative polygonal

approximation [16]. Line ends/junctions are identified by applying a template at each 1-valued point. A data structure which describes the graph model (vertices and edges) of the lines in the image is constructed. Finally, sets of vectors that approximate the lines in the image are obtained.

Line end/junction identification and line following

PEs begin with their thinned binary subimages and subimage border data of each of their neighbors. If the same PEs that performed the thinning are being used, each PE already has the necessary data. A PE's subimage with all of its neighboring borders is called an *augmented subimage.*

A set of templates has been constructed to allow the rapid identification of line ends and junctions. Each template is a 3-by-3 pixel window with a 1-valued center pixel and the other 8 pixels either 0- or 1-valued. Thus there are 2^8=256 different templates, each representing one of the following cases: (1) line end, (2) line junction, (3) middle point, or (4) indeterminate point. Figure 5 shows examples of each type of template for the 8-connected case. Figure 5a is an example of a line end. The line junction (Figure 5b) is defined as the meeting point of three or more distinct lines. The template of Figure 5c matches a point on a line that is neither an end or a junction. Figure 5d is a point that could be either a junction or a middle point; however, a 3-by-3 template is insufficient to determine the type. At this stage of the algorithm, indeterminate points are treated as junctions. Later, indeterminate points that are adjacent to other indeterminate points and junctions are examined and re-classified as middle points or true line junctions.

Line end/junction identification and line following are done in a single pass using the following procedure. In MIMD mode, each PE scans its thinned binary subimage by row starting from the upper left-hand corner. When a 1-valued pixel is encountered in the subimage, scanning stops and the pixel is treated as the center point of a 3-by-3 window. This pixel is called the *start point.* Data from the augmented subimage are needed to apply the template if the start point is on the edge of the subimage. The template is used to classify the start point as an end, junction (or indeterminate), or middle.

If the start point is classified as an end, the line is followed until another end or junction is encountered. For "middle" points, the line is followed in each direction until ends or junctions are encountered. Junction (or indeterminate) start points have several lines emanating from them. Each line is followed until ends or junctions are encountered. When a line is followed, the coordinates of all 1-valued pixels in the line are recorded. Also, each point encountered is marked so it will not be retraced. This is especially important when tracing closed figures; if the start point is re-encountered, its classification is changed from "middle" to "closed figure entry point."

Due to the data allocation among processors, PEs will encounter lines that continue into an adjacent PE. With the augmented subimage data, PEs can determine whether a line stops on the edge of their subimage (a 0-valued pixel in the augmented subimage) or whether it continues (a 1-valued pixel in the augmented subimage). When a line continues into another PE, the pixel on the subimage edge is known as a "border connection point."

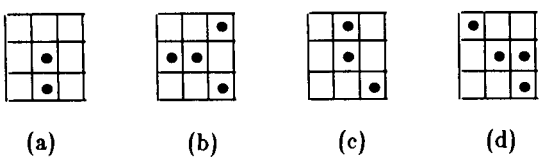

Figure 5. Templates for identification of line ends and junctions for 8-connected imagery: (a) line end; (b) line junction; (c) middle point; (d) indeterminate point.

The desired output of the vectorization algorithm is a two-level structure. At the *graph level,* an edge and vertex representation of the image is formed. The graph's edges are the lines that emanate from vertices (end points, junctions, and closed figure entry points). This representation describes which vertices are adjacent (connected to each other by a single edge) and is useful for pattern matching or analysis of the connectivity of lines. At the more detailed *vector level,* each graph edge is modeled by vectors which give a piecewise-linear representation of the image line. This level is required for cartographic analysis and visual interpretation.

The graph level of the data structure is built as follows. When an end point, junction, or closed figure entry point is first encountered, a *vnode* (vertex node) structure is allocated to store information about it. Border connection points are not true vertices; however, they are given vnodes until the PEs complete the topological reconstruction phase. The vnodes for a vertex has entries for its global row and column positions, its type (end, junction, border connection point, closed figure entry point, indeterminate point), a "busy" semaphore to prevent access to the vnode by more than one process at a time [22], and a pointer to a list of *adjnodes* (adjacent node structures). Each adjnode gives the global row and column positions of adjacent vertices and has a pointer to a *coordinate list* that describes the edge between the vnode point and the adjnode point. Vnodes are kept sorted by row and column index so that they can be accessed easily.

At the end of the line following phase, the graph level of the output data structure is complete for all edges that do not cross subimage boundaries. The next step, topological reconstruction, completes the graph level of the data structure.

Topological reconstruction

In this phase, PEs exchange information about edges that cross subimage boundaries. Although reconnecting lines at subimage boundaries is an extra step in the parallel algorithm, even with serial techniques, images are frequently subdivided due to central memory restraints [14]. Such a serial scheme would have the same complications as the parallel approach.

Several examples of edges processed during this phase are shown in Figure 6. Consider the edge connecting ver-

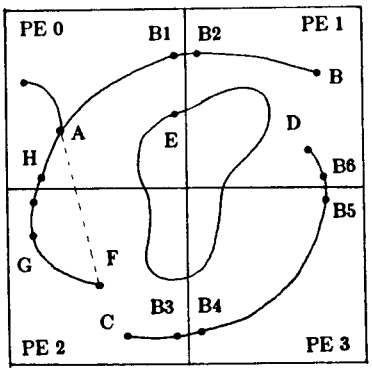

Figure 6. Connection of curves across subimage boundaries. F and G are the maximally distant points from line segment FA for PEs 2 and 0, respectively.

tices A and B which is split between PE 0 and PE 1. B1 and B2 are the corresponding border connection points. The result to be obtained is to have the data structure indicate that A is adjacent to B and vice versa. However, the data structure currently indicates only the adjacency within a subimage; that is, adjacency between A and B1 in PE 0 and adjacency between B2 and B in PE 1.

To begin this phase, each PE scans its list of border connection point vnodes. Suppose PE 0 first encounters the B1 vnode. When it does, it marks the B1 vnode entry busy by setting the semaphore. This prevents other PEs from accessing B1 while the update operation is in progress. PE 0 knows that B2 is the corresponding border connection point in PE 1 because B2 appears in PE 0's augmented subimage. PE 0 sends a message to PE 1 requesting the coordinates of the vertex point adjacent to B2. In this case, PE 0 is known as the *initiating* PE. PE 0 also sends the coordinates of A, the vertex adjacent to B1, with the request as well as the address of the initiating PE (PE 0). PE 1 is interrupted upon receipt of the request and processes the message. It marks vertex B2 busy so that it does not try to initiate any queries of its own about the edge AB. PE 1 looks up B2's vnode and determines its adjacent vertex, B. PE 1 then returns B's coordinates in a message to PE 0. PE 0 updates the vnode entry for A indicating that B is adjacent to it; PE 1 updates vnode B indicating that A is adjacent to it. B1 and B2 are kept as "placeholder" vnodes for later stages of the algorithm.

As an alternative to the exchange of information outlined above, PE 1 could have initiated the query with a request to PE 0. The final result would be identical.

The "locking" of the vnode entries was necessary in the example above since both PE 0 and PE 1 could have been trying to work on the same edge simultaneously. Only one PE should be allowed to connect it.

Suppose that PE 0 had initiated its request first and locked B1, but before PE 1 received the message and locked B2 due to PE 0's request, it also initiated a request (locking B2 due to its own request). In this case, PE 0 would return the message "locked" to PE 1 in response to the query about B1. Similarly, PE 1 would return the "locked" message to PE 0 because B2 is locked. To prevent *deadlock* [22], that is, a situation in which each participating PE needs a resource that the other is unwilling to provide, only one PE must be allowed to continue. The situation can be resolved by having PEs unlock their vnode entries when they receive a "locked" message. Unfortunately, this results in the edge AB not being processed. The solution is to adopt a rule that prioritizes requests based on some criterion; for example, requests initiated by lower-numbered PEs are given priority. A variation on this method of resolving deadlock was discussed in [26].

The edge CD in Figure 6 is processed similarly to the way described earlier. If PE 2 initiates the query, PE 3 cannot determine the other vertex (D) itself and passes the request along to PE 1. PE 1 processes the request and returns vertex D's coordinates to PE 3 which in turn returns them to PE 2. However, if PE 3 initiates the query, it has neither of the coordinates of the vertices. Thus when it is returned one vertex's coordinates, for example, "C" as a result of querying PE 2, it passes them to PE 1 when querying in the other direction. When PE 1 returns the coordinates of D, PE 2 still lacks them and must be sent an additional message. This additional message can often be avoided if PEs delay processing the "double border point" cases (such as B4-B5 in PE 3).

The closed figure in Figure 6 has no real vertices; each PE sees part of an edge (a *subedge*) with two boundary connection points. No PE knows whether its subedge is part of a closed figure or is part of an edge that crosses multiple subimage boundaries (like edge CD). Some PE queries about the vertex in one of the two directions. Eventually, the query will come back to the initiating PE which will realize that a closed figure has been found. The initiating PE designates one of its border connection points as a closed figure entry point vertex and returns the coordinates of this vertex along the path of queries.

Polygonal Approximation

The basic goal when converting images from raster to vector format is to represent curved lines (graph edges) by sets of vectors that approximate the curve shapes. Approximation algorithms require calculation of some numerical criterion that quantifies the approximation's "fit." This calculation can be computationally expensive for algorithms involving minimum perimeter polygons or least squares line fitting [16]. For many applications, conversion speed is more important than making rigorously mathematical approximations and finding the absolute minimum number of vertices.

Ramer [16] described an algorithm for approximating plane curves by polygons that is computationally efficient and produces near minimum numbers of vertices. The following recursive procedure is used: a curve is approximated by a straight line through its endpoints; a calculation is made to see if the approximation is acceptable and if not, the curve is broken into two curves at the point most distant from the straight-line segment. This procedure is repeated until the entire curve is approximated by a set of straight-line segments that satisfy the fit criterion.

There are various criteria for quantifying the approximation of curves by polygons [16]. For our algorithm, the square of the maximum absolute distance of a curve to the straight line segment connecting its endpoints was chosen because of its computational simplicity. Furthermore, sharp spikes in an image (that may be important image information) will be preserved. Some methods of approximation such as mean squared error would not preserve this information.

One disadvantage of the polygonal approximation is that long curves require many calculations. Ramer suggested arbitrarily breaking up curves every P points (where P is perhaps 50 or 100) to minimize this problem [16]. In the parallel environment, long curves are automatically segmented at every subimage boundary.

There are two ways the PEs can perform the polygonal approximations. In the first way, there is no inter-PE communication: each PE vectorizes the graph subedges within its subimage independently. Edges are arbitrarily broken at the subimage boundaries, which does not substantially affect the visual results although it may increase the total number of vectors created. If edges are generally much longer than the dimensions of a subimage, they cross multiple subimage boundaries may be split more often than necessary. This simple approach may be acceptable for many applications. Unfortunately, the results depend on the number of PEs used to perform the algorithm, the data allocation among the PEs, and the actual image.

To obtain results comparable to the serial algorithm, the second and more complex approach must be taken. Here, PEs independently vectorize graph edges that lie entirely within their subimages and cooperate on edges that cross subimage boundaries. When processing an edge like AB in Figure 6, each PE knows the two vertex endpoints (exchanged during the topology reconstruction phase) and is responsible for determining the maximum distance between its part of the edge and the straight line segment AB. The PEs exchange their maximum distances and coordinates so each can decide where the edge should be split. Since each is executing the same algorithm, they must come

to the same conclusion.

For example, consider the edge FA in Figure 6. Suppose that PE 0 determines that point H is maximally distant at 8 units and PE 2 determines that point G is maximally distant at 10 units. After the exchange of the coordinates and distances, each PE determines that point G is maximally distant from line segment FA and that the edge should be split into two vectors, FG and GA. Vector FG can now be subdivided if necessary by PE 2 on its own. However, the two PEs must cooperate again to determine if GA meets the tolerance criterion. The process begins again: PE 0 determines the maximum distance from the curve to vector GA on its side of the subimage; PE 2 does so on its side. If after the exchange, the tolerance criterion has not been met, vector GA is subdivided further using the same procedure.

To avoid use of the deadlock resolution protocol during this phase, the exchange of maximum distance information about a given edge is initiated by the PE having the vertex point on that edge with the lower-valued row coordinate (if tie, lower-valued column coordinate). For closed figures, the PE having the closed figure entry point initiates the exchange.

The communication of the maximum distances for edge CD of Figure 6 is more complex but it mimics the protocol used in the topology reconstruction phase. Suppose that subedge C-B3 has maximum distance 10, B4-B5 has 20, and B6-D has 8. The initiating PE, PE 1, sends the distance 8 and the coordinate of the point at which it occurred to PE 3. PE 3 compares this distance to its own and passes the maximum of the two (distance 20) to PE 2. PE 2 compares the distance to its own, records the maximum (distance 20), and returns it to PE 3. PE 3 in turn returns the maximum distance to PE 1. Each now knows the maximum distance which it compares with the tolerance to determine if a split is necessary.

Communication of the distances for the closed curve of Figure 6 is similar to that described above. Each PE calculates the maximum distance of its subedge from the initiating PE's "closed curve entry point." The first split is made at the maximally-distant point.

V. Conclusions

Preliminary results have been obtained by writing serial simulations of these parallel thinning and vectorization algorithms and by examining their performance for a number of test images. In general, these results demonstrate that parallel processing systems can be used to significantly reduce the execution times for converting images from raster to vector format. Simulations of the hybrid thinning algorithm consisting of the part-SIMD, part-MIMD distance and peeling algorithms showed improvements in average performance as compared to the distance or peeling algorithms used alone. Speedups approaching N, the number of PEs, are obtained for the vectorization algorithm so long as the subimage size does not become too small in comparison with the average length of lines in the image. For a given machine size, arbitrarily breaking lines into vectors at PE subimage boundaries results in better speedups than those obtained for the more complex approach of iterative polygonal approximation across subimage boundaries. More detailed discussion of the parallel algorithms, including specific techniques used to match templates, follow lines, link subedges across subimage border points, and so on, are found in [11].

The simulations do not model some of the subtle aspects of parallel processing such as the number of interconnection network conflicts during MIMD message passing, and the number of messages involved in resolving potential deadlock situations. As the PASM prototype hardware becomes available, we will investigate the data-dependent behavior of the algorithms of a complete raster to vector conversion process and undertake the processing of realistically-sized imagery.

References

[1] G. B. Adams III and H. J. Siegel, "The extra stage cube: a fault-tolerant interconnection network for supersystems," *IEEE Trans. Comp.*, Vol. C-31, May 1982, pp. 443-454.

[2] C. Arcelli, L. Cordella, and S. Levialdi, "Parallel thinning of binary pictures," *Electronic Letters*, Vol. 11, July 1975, pp. 148-149.

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

[4] K. E. Batcher, "Bit serial parallel processing systems," *IEEE Trans. Comp.*, Vol. C-31, May 1982, pp. 377-384.

[5] W. J. Bouknight, S. A. Denenberg, D. E. McIntryre, J. M. Randall, A. H. Sameh, and D. L. Slotnick, "The Illiac IV system," *Proc. IEEE*, Vol. 60, Apr. 1972, pp. 369-388.

[6] M. J. B. Duff, "CLIP 4: a large scale integrated circuit array parallel processor," *Third Int'l. Conf. Pattern Recognition*, 1976, pp. 728-732.

[7] M. J. Flynn, "Very high-speed computing systems," *Proc. IEEE*, Vol. 54, Dec. 1966, pp. 1901-1909.

[8] T. J. Fountain, "CLIP4: progress report," in *Languages and Architectures for Image Processing*, M. J. B. Duff and S. Levialdi, eds., Academic Press, London, England, 1981, pp. 281-291.

[9] C. J. Hilditch, "Comparison of thinning algorithms on a parallel processor," *Image and Vision Processing*, Vol. 1, Aug. 1983, pp. 115-132.

[10] J. T. Kuehn and H. J. Siegel, "Simulation studies of a parallel histogramming algorithm for PASM," *Seventh Int'l. Conf. Pattern Recognition*, July 1984, pp. 646-649.

[11] J. T. Kuehn, *The PASM Parallel Processing System: Design, Simulation, and Image Processing Applications*, Ph.D. Dissertation, School of Electrical Engineering, Purdue University, in preparation.

[12] D. G. Meyer, H. J. Siegel, T. Schwederski, N. J. Davis IV, and J. T. Kuehn, "The PASM parallel system prototype," *IEEE Comp. Soc. Compcon*, Feb. 1985, pp. 429-434.

[13] D. L. Milgram, "Region extraction using convergent evidence," *Computer Graphics and Image Processing*, Vol. 11, 1979, pp. 1-12.

[14] D. J. Peuquet, "Raster processing: an alternative approach to automated cartographic data handling," *The American Cartographer*, Vol. 6, Oct. 1979, pp. 129-139.

[15] D. Peuquet, "An examination of techniques for reformatting digital cartographic data. Part 1: The raster-to-vector process," *Cartographica*, Vol. 18, 1981, pp. 34-38.

[16] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing*, Vol. 1, 1972, pp. 244-256.

[17] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *Journal of the Association for Computing Machinery*, Vol. 13, Oct. 1966, pp. 471-494.

[18] M. C. Sejnowski, E. T. Upchurch, R. N. Kapur, D. P. S. Charlu, and G. J. Lipovski, "An overview of the Texas Reconfigurable Array Computer," *AFIPS 1980 Nat'l. Comp. Conf.*, June 1980, pp. 631-641.

[19] H. J. Siegel and P. H. Swain, "Contextual classification on PASM," *IEEE Comp. Soc. Conference on Pattern Recognition and Image Processing*, Aug. 1981, pp. 320-325.

[20] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: a partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Comp.*, Vol. C-30, Dec. 1981, pp. 934-947.

[21] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, Lexington Books, Lexington, MA, 1985.

[22] H. S. Stone, "Parallel computers," in *Introduction to Computer Architecture*, H. S. Stone, ed., Science Research Associates, Inc., Chicago, IL, 1980, pp. 363-425.

[23] R. E. Suciu and A. P. Reeves, "A comparison of differential and moment based edge detectors," *1982 IEEE Comp. Soc. Conference on Pattern Recognition and Image Processing*, June 1982, pp. 97-102.

[24] R. J. Swan, S. Fuller, and D. P. Siewiorek, "Cm*: a modular multimicroprocessor," *AFIPS 1977 Nat'l. Comp. Conf.*, June 1977, pp. 637-644.

[25] P. H. Swain, J. C. Tilton, and S. B. Vardeman, "Contextual classification of multispectral image data," *Pattern Recognition*, Vol. 13, 1981, pp. 429-441.

[26] D. L. Tuomenoksa, G. B. Adams III, H. J. Siegel, and O. R. Mitchell, "A parallel algorithm for contour extraction: advantages and architectural implications," *1983 IEEE Comp. Soc. Symp. Computer Vision and Pattern Recognition*, June 1983, pp. 336-344.

[27] W. Wulf and C. Bell, "C.mmp--A multi-miniprocessor," *AFIPS 1972 Fall Joint Computer Conference*, Dec. 1972, pp. 765-777.