Received 16 November 2022, accepted 1 December 2022, date of publication 5 December 2022, date of current version 8 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3226943

RESEARCH ARTICLE

Algorithms to Speed up Contour Tracing in Real Time Image Processing Systems

SONAL GUPTA AND SUBRAT KAR, (Senior Member, IEEE)
Department of Electrical Engineering, Indian Institute of Technology Delhi (IIT Delhi), New Delhi 110016, India
Corresponding author: Sonal Gupta (sonal.gupta@ee.iitd.ac.in)

This work was supported by the Department of Electrical Engineering, IIT Delhi.

ABSTRACT Contour tracing is an important pre-processing step in many image-processing applications such as feature recognition, biomedical imaging, security and surveillance. As single-processor architectures reach their performance limits, parallel processing architectures offer energy-efficient and high-performance solutions for real-time applications. Parallel processing architectures are thus used for several real-time image processing applications. Among the several interconnection schemes available, Cayley graph-based interconnections offer easy routing and symmetric implementation capabilities. For parallel processing systems with a Cayley graph-based interconnection scheme, torus, we developed three accelerated algorithms corresponding to three existing families of contour tracing algorithms. We simulated these algorithms on a parallel processing framework to quantify the normalized speed-up possible in any torus-connected parallel processing system. We also compared our best-performing algorithm with the existing parallel processing implementations for Nvidia GPUs. We observed a speed-up of up to 468 times using our algorithms on a parallel processing architecture in comparison to the corresponding algorithm on a single processor architecture. We evaluated a speedup of 194 (and 47) compared to the existing parallel processing contour tracing implementation on Tesla K40c (and Quadro RTX 5000 GPU hardware, respectively). We observe that for torus-connected parallel processing architectures used for image processing, our algorithms can speed up contour tracing without any hardware modification.

INDEX TERMS Accelerated contour tracing, image processing, parallel algorithms, multiprocessors, torus, GPU.

I. INTRODUCTION

As single processor architectures approach their performance limits, scaling and speed-ups become possible only through parallel processing architectures [1]. Over the last few years, these architectures have gained popularity for several applications, especially in data-intensive applications like scientific computing, machine learning, big data analytics, human-computer interaction, where they provide energy-efficient solutions along with high processing capability [2], [3].

Out of the several interconnection schemes used in parallel processing systems, Cayley Graph (CG) interconnection-based schemes – rings, toroids, and hypercubes – are the most attractive because of their symmetric and decomposable nature [4]. In recent years, CG interconnected systems have been used to improve the speed and performance of algorithms in applications like multimedia processing, image processing, and parallel computing [5], [6], [7], [8], [9], [10], [11], [12]. Hardware implementations like matrix computations on FPGA [13], contour extraction on partitionable SIMD/MIMD System - PASM [14], and edge detection and image resizing on Cell BE and Blue Gene\L HPC (High-Performance Computing) platforms [15] have been developed to improve the performance in the corresponding fields. To improve the speed of contour tracing (hence image processing) on existing parallel processing hardware, we developed distributed data processing based algorithms for implementation on CG interconnected parallel processors. We used the contour tracing paradigm to quantify the performance of a parallel processing system in comparison to a single processor system. We also compared our implementation with the existing parallel processing contour tracing implementations.

Contour tracing identifies the boundary pixels of the active region of an image [16]. The contour is then used, instead of the original image, to reduce the storage memory

The associate editor coordinating the review of this manuscript and approving it for publication was Shuihua Wang.

and computation time in image processing applications like biomedical imaging [17], [18], character recognition [19], [20], security and surveillance [21], [22], feature recognition using machine learning and deep learning [23]. Corresponding to the three existing families of contour tracing algorithms – pixel-following, vertex-following, run-data-based-following – we developed three parallel processing algorithms for two-dimensional torus-connected multiprocessor architectures [24]. We simulated these algorithms to quantify the number of time units the parallel processing architecture takes in comparison to a single processor system for the same algorithm. We observed a speed-up of $\frac{n^2}{p} \times \frac{\Delta_p}{\Delta_c}$, for n processors operating on an image with p pixels, where Δ_p is the processing time for one comparison operation on a single processor, and Δ_c is the per-hop communication delay in the multiprocessor system. We also compared the expected time taken by our best-performing algorithm with the existing parallel processing contour tracing implementations on GPUs (Graphics Processing Units). We observed a speed-up of minimum of 47 times using our algorithm.

We assert and prove that for a very large array of tiny processors used in image processing applications, torus-based architectures lead to significantly faster parallel algorithms for, *inter alia*, contour tracing. We have developed parallel algorithms which are mapped to an underlying torus architecture and can be used to speed up image processing without any hardware modification in any CG interconnected system of processors.

The structure of the rest of the paper is as follows. In Section II, we list the existing parallel contour tracing implementations and state our novelty. In Section III, we detail the hardware framework, mesh and torus interconnection, choice and partitioning of the test image, details of the three algorithms, segmentation, metric used for evaluation of our methodology, and our proposed implementation on the NVIDIA GPUs used in existing parallel processing algorithms. In Section IV, we present the comparison between the conventional algorithms and their multiprocessor versions, the comparison between the three presented algorithms, and the comparison between the existing parallel processing systems and our implementation. In Sections V and VI, we entail the discussion and conclusions of our research experiment, respectively.

II. RELATED WORK REVIEW

Parallel contour tracing on a segmented image has been an active area of interest, using both Supercomputers and Graphics Processing Unit (GPU) hardware, in recent times. In [25], Agi et al., designed a custom-made hardware to enable parallel processing. They divided the image into blocks and implemented a raster scan based algorithm to achieve parallelism. In [26], Ratnayake et al., interfaced a high-speed memory with contour tracing modules to improve throughput. They segmented the image into 3 × 3 windows and implemented the raster scan algorithm using pipeline architecture. They also made hardware changes in the memory and its interface to reduce the memory access time. In [27], Chia et al., used a one-dimensional array of processing elements for parallel contour tracing. They used a Moore neighbour tracing based algorithm for implementation.

In [28], Garcia et al., presented a modification of the Suzuki algorithm for parallel contour tracing on GPUs. The Suzuki algorithm is also based on the Moore neighbour tracing algorithm. They segmented the image into 32 × 32 or 64 × 64 rectangles and performed contour tracing in each of the rectangles using GPUs. In [29], Cao et al., presented a block searching algorithm using Compute Unified Device Architecture (CUDA) platform for GPUs. They divided the grid of pixels into four smaller cells and then identified the contour by detecting the edge crossing of the image at the boundaries of each of those cells. This parallel implementation used CUDA threads to perform block searching tasks. In [30], Zhao et al., developed a parallel strategy to trace contour in large-scale Digital Elevation Model (DEM) data using a collaboration of CPU and GPU. They sectioned the image row-wise and traced the contours, on the sections, in parallel, using GPU. The reconstruction of the sectioned contours was done using CPU.

Though these existing techniques lay a fertile foundation for parallel contour tracing systems, a complete solution for *all* the contour tracing algorithms using the *existing* hardware does not exist yet. Also, today, the segmentation of the image can be taken to a *single* pixel level, given the multitude of cores available in the state-of-the-art supercomputer and GPU hardware. Our methodology, thus, adapts three major families of contour tracing algorithms for parallel processing on an existing supercomputer and GPU hardware. The performance of our parallel algorithms benefits significantly from the close coupling with the mesh and torus interconnections used in supercomputer and GPU architectures.

III. METHODOLOGY

A. HARDWARE FRAMEWORK

In the most popular supercomputer and GPU architectures, the processing cores are directly coupled with a router (or a fixed number of cores are connected to a single router) as shown in Fig.1. Multiple core-connected routers are connected together using a particular interconnection topology. In our implementation, we have considered that each router is connected to a single core. We term this single core-router setup as a processing element (or a processor).

These processing elements have local registers, buffers and/or shared memory for storage (not shown explicitly in Fig.1). The system has a global RAM for bulk storage. The interconnected Processing Elements (PEs), Network Interface, and shared and global memory enable parallel processing on such hardware architectures. The complete image pixels are stored in the global RAM of the system. The pre-processing steps of grayscaling and thresholding are done by the control unit (or CPU) using state-of-the-art methods.

After the pre-processing step, each pixel is represented by a single bit, which can be stored in the local registers or shared memory corresponding to the PEs. Depending on the segmentation of the image (detailed in Section III-K), each PE processes one pixel or a segment of pixels. From the global

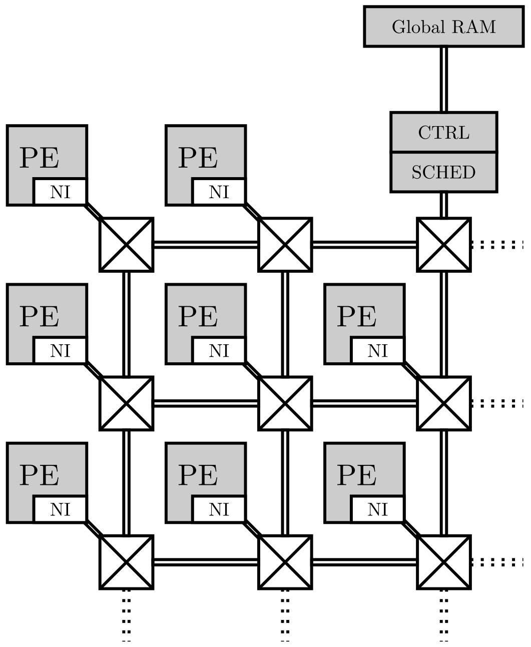


FIGURE 1. General supercomputer and GPU network architecture where PE is Processing Element, NI is Network Interface, CTRL is Control Unit, and SCHED is Scheduler. Boxes with cross are the Routers.

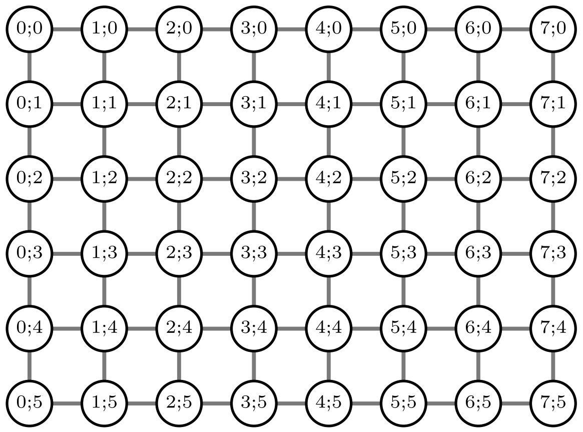


FIGURE 2. A $c \times r$ 2-D mesh connected network of processors with 8 columns and 6 rows, *i.e.,* a total of 48 processing elements.

RAM, the pixel values are routed to the corresponding PE. Then, contour tracing is performed on the distributed data using the algorithms presented in Sections III-F to III-K. The contour pixels are temporarily stored in the shared memory while executing the algorithm. After all PEs complete their contour tracing operations, the contour stack is transferred to the Global RAM. As a post-processing step, the stored contour stack is sorted from left to right and top to bottom order of pixels by the control unit.

B. ABOUT MESH AND TORUS INTERCONNECTION NETWORK

A $c \times r$ 2-D torus connected multiprocessor hardware is used for this experimentation, where c is the number of columns and r is the number of rows of the 2-D torus. In a 2-D mesh network topology, each node is connected with *four*

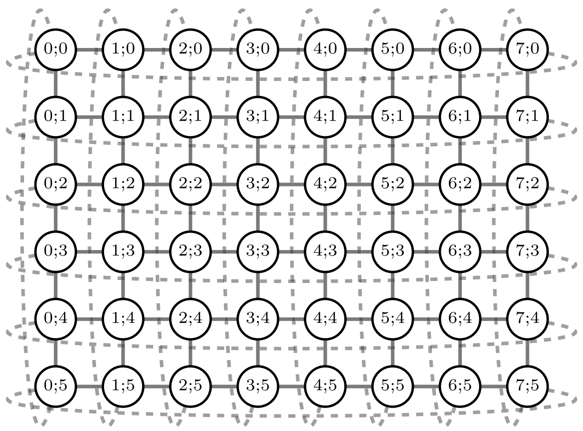


FIGURE 3. A $c \times r$, 2-D torus connected network of processors with 8 columns and 6 rows, *i.e.,* a total of 48 processing elements.

neighbours in north, east, south and west directions as shown in Fig.2. The 2-D torus has wraparound connections to form a symmetric network in addition to the basic topology of a mesh network, as shown in Fig.3.

C. OVERVIEW OF THE METHODOLOGY

First, the test image pixel values are distributed over the multiprocessor hardware. *Second*, the existing contour tracing algorithm is modified so that the per-pixel mathematical operations for contour tracing are distributed to multiple processors. *Third*, the multiprocessor array is segmented into symmetric sections. *Fourth*, the adapted contour tracing algorithm is run in each of these sections in parallel. These four steps are consecutively repeated for the three existing contour tracing algorithms to develop their respective accelerated versions.

D. CHOICE OF AN APPROPRIATE TEST IMAGE

The choice of a test image (or a set of images) is important, as it should represent the complete set of binary digital images. The test image, suggested by T. Miyatake et al., in [31], represents every local pattern encountered by two adjacent scan lines in a digital binary image. This image is shown in Fig.4a. Since we are considering only the nearest neighbour of a pixel to detect the contour of an image, the local patterns encountered by two scan lines cover all local patterns which can be encountered by our algorithms in digital binary images. Hence, the image suggested in [31] can be used as the test image for our algorithms. The ten local patterns contained in the test image are shown in Fig.5.

Since all the contours in the original image are at most one pixel wide, the image is scaled up by four times to illustrate the contour tracing operation. The scaled-up image, shown in Fig.4b retains all the 10 local patterns present in the original image. Thus, the results of contour tracing illustrated using this image can be applied to any binary digital image.

E. PARTITIONING THE TEST IMAGE AS A SQUARE PLANAR TESSELLATION WITH 4-CONNECTEDNESS TO MATCH UNDERLYING MULTIPROCESSOR HARDWARE

The pixel values of the standard image (Fig.4) are stored as a two-dimensional array such that the number of columns (c)

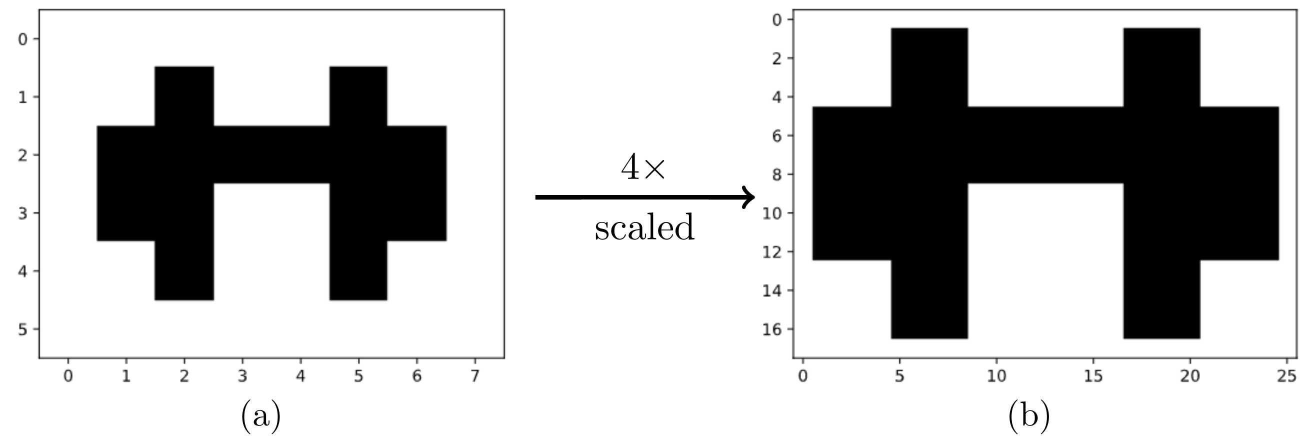


FIGURE 4. (a) Standard Image and (b) its 4× scaled version.

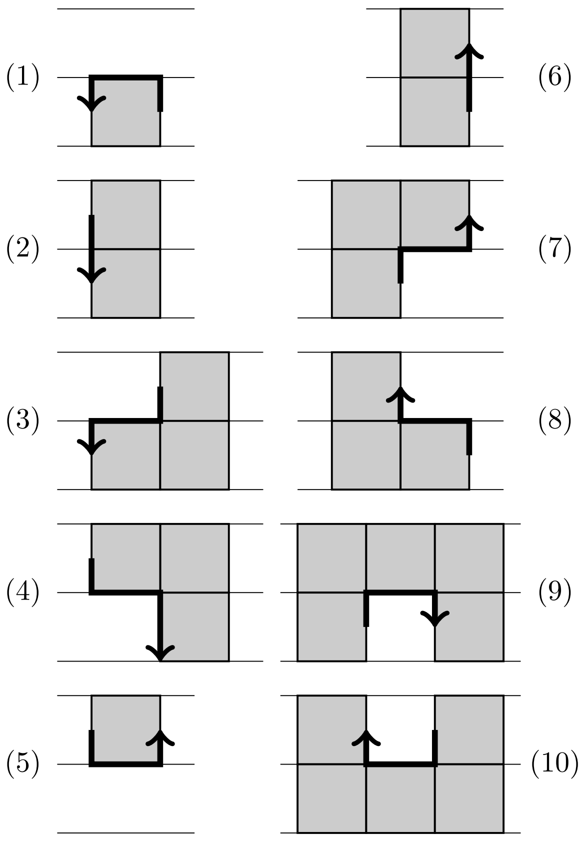


FIGURE 5. All ten local patterns which can be encountered by our algorithms in a digital binary image [31].

represents the number of pixels in the horizontal dimension and the number of rows (r) represents the number of pixels in the vertical dimension of the image, resulting in a $c \times r$ 2-D array.

Each of the pixels in this 2-D array has four neighbours, one each in the north, east, south, and west directions, *i.e.,* the pixel at (x, y) location in the 2-D array has the neighbours: $(x, y-1)$, $(x+1, y)$, $(x, y+1)$, $(x-1, y)$ $\forall x \in [1, c-1]$ and $y \in [1, r-1]$. It is to be noted that pixels $(x, y) \forall x \in \{0, c\}$ and $y \in \{0, r\}$ are exceptions as they lie on the corners

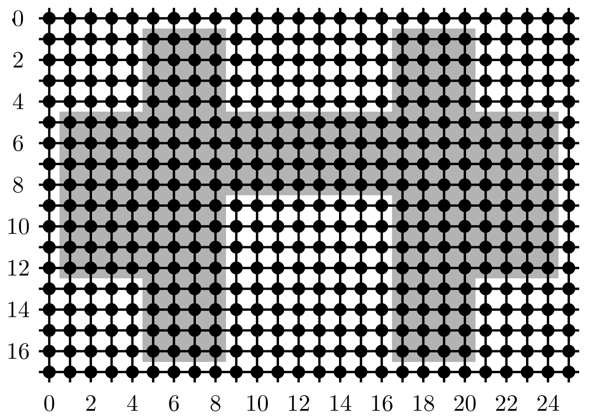


FIGURE 6. An $n \times m$ 2-D torus connected network of processors with 26 rows and 18 columns, *i.e.,* a total of 468 processing elements with the standard image overlaid – the wraparound connections (similar to Fig. 3) are present but are not explicitly shown in this figure.

of the image. This arrangement is equivalent to a square planar tessellation with four-connectedness [32], which is the same as the network connectedness of a $c \times r$ 2-D torus topology [33]. Hence, the image pixels can be overlaid on a $c \times r$ 2-D torus-connected multiprocessor system (Fig.6) to enable the implementation of our contour tracing algorithms.

F. DETAILS OF ALGORITHMS (PREPROCESSING AND CONVENTIONAL CONTOUR TRACING IN THE TEST IMAGE)

The contour of a digital image is precisely 1 pixel wide and can be used to reconstruct the original image with good accuracy. An active pixel in a coloured digital image is a pixel with non-zero RGB (Red, Green, Blue) values. However, in the case of a grey-scaled and thresholded image, an active pixel is the one with zero R, G and B values, while an inactive pixel is the one with 255 R, G and B values each [34]. We have used a grey-scaled and thresholded image in our implementation. We can thus inspect any one of the R, G or B values (say R) to detect an active pixel in the image, thus converting it into a binary digital image.

The conventional contour tracing algorithms [24] compare the value of each pixel in a grey-scaled and thresholded digital

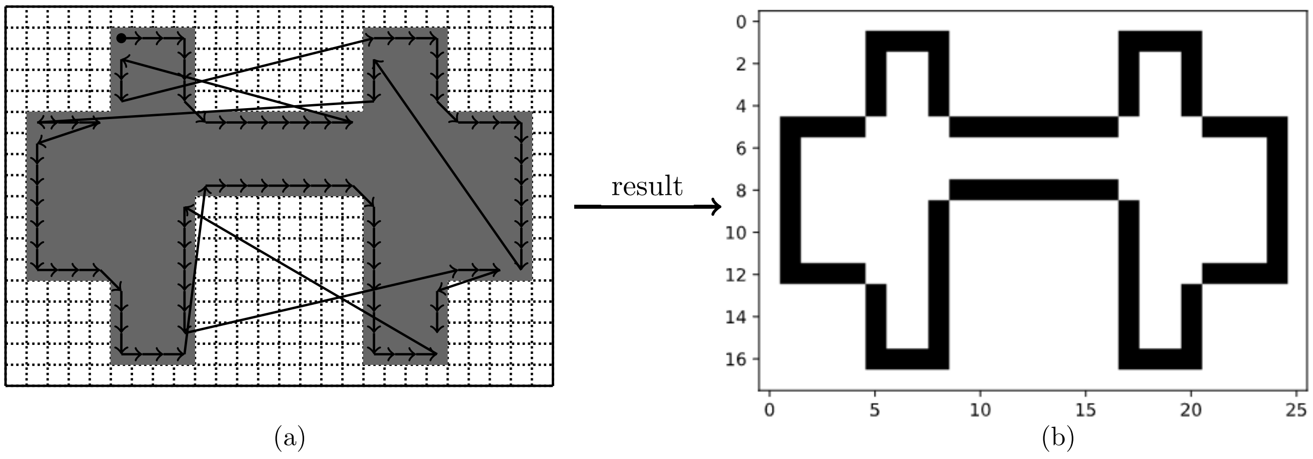


FIGURE 7. **(a) Tracing of the standard image using our implementation of the pixel-following algorithm; dot represents the first active pixel detected; (b) The resultant pixels in the contour stack.**

image with its four neighbours to detect whether that pixel is a pixel on the boundary of the image.

G. CONVERTING CONVENTIONAL CONTOUR TRACING ALGORITHM FOR A MULTIPROCESSOR IMPLEMENTATION

In our adaptation of the algorithms, by distributing the image pixels over the multiprocessor system (Fig.6), we convert the basic operation of comparison between pixels to communication between processing elements. Each pixel in our system is assigned to one processing element in the array of embedded systems. To detect an active pixel, the processor containing the pixel communicates with the four neighbouring processing elements and compares their pixel value with its own. The four operations of comparison for detecting each active pixel, thus, are converted into four operations of communication in the case of a 2-D torus-connected multiprocessor system.

Though a two-dimensional array of processors is the basic requirement for our implementation, the algorithms can also be ported to higher order torus topology by executing them in parallel on two-dimensional planes of processors for different images or image segments. We note that the number of processing elements (n) is equal to the number of pixels (p) in the test image, in each dimension, in our presented experiment. If, however, $n > p$, then only p number of processing elements will be (each containing the value of one pixel) *active*. In contrast, if $n < p$, then our algorithms can be used in two ways – (i) the image can be down-sampled to n number of pixels to execute the algorithms; (ii) the image can be divided into $\lceil \frac{p}{n} \rceil$ parts of n pixels each, and each of these parts can be processed sequentially by the n processors.

The conventional contour-tracing algorithms fall under three broad categories: (i) pixel-following, (ii) vertex-following and (iii) run-data-based-following [24] differentiated by the process of pixel scanning and detection of active pixels. The adaptation of each of these algorithms is described next.

H. ADAPTATION OF PIXEL-FOLLOWING ALGORITHM TO MULTIPROCESSOR IMPLEMENTATION

The pixel-following (PF) algorithm is based on the wall-following concept of contour tracing [35]. In a room, to detect the contour of the room, *i.e.,* the continuous structure made by the walls, we start from any point in the room and keep tracing the next point on the wall to trace the walls (or contour) of the complete room. Similarly, in the case of an image, we do a standard scan (top to bottom and left to right) to detect the first active pixel and keep tracing for the neighbouring active pixels till we reach the first pixel of this contour. This is done repeatedly over the complete image till the full image is scanned to detect disconnected contours. To decide whether a pixel is active while scanning, the pixel R (Red component) value is compared with its four neighbouring pixel values. If the pixel value of the pixel being scanned is smaller (zero) than *any* of its neighbours, the pixel is denoted as an active (or boundary) pixel. The contour pixels thus detected are pushed onto a common shared stack to get a complete set of contour pixels at the end of the algorithm.

In our adaptation of the PF algorithm, each of the image pixel R values is assigned to the corresponding processor in the 2-D array, as shown in Fig.6. The PF algorithm is then run on the complete array of *active* processors to trace the contour. In this implementation, since, to compare values of any two neighbouring pixels, two processors need to communicate with each other, each comparison is converted into a communication between two processing elements of the network. This is a noteworthy salient feature of our adaptation.

Fig.7a shows the sequential tracing of contour pixels using our implementation of the PF algorithm and the resultant contour pixels in Fig.7b. Algorithm1 shows the pseudocode for our adaptation of the PF algorithm.

The contour tracing using the PF algorithm does not scale very well as the image scales – it requires large amounts of memory for large and dense images [24]. The discontinuities in the scan, at the corners of the test image (shown in Fig.7a),

Algorithm 1 Pixel-Following Algorithm

```
Result: Returns a stack of contour points (x,y)
processorNodesUsed = pixelsInImage;
for All nodes do
    for [neighbor] east and south sorted neighbors of node do
        if nodeIsInContour not set then
            if pixelValueNeighbor > pixelValueNode then
                push node to contourStack;
                set nodeIsInContour;
                searchNeighbor(node);
            if neighborIsInContour not set then
                if pixelValueNeighbor < pixelValueNode then
                    push neighbor to contourStack;
                    set neighborIsInContour;
                    searchNeighbor(neighbor);
searchNeighbor(neighbor)
for [neighbor1] east and south sorted neighbors of neighbor do
    if neighbor1IsInContour not set then
        for [neighbor2] All sorted neighbors (north, south, east, west) of neighbor1 do
            if pixelValueNeighbor2 > pixelValueNeighbor1 then
                push neighbor1 to contourStack;
                set neighbor1IsInContour;
                searchNeighbor(neighbor1);
return
```

is another limitation of the PF algorithm for 4-connected tessellations [24].

I. ADAPTATION OF VERTEX-FOLLOWING ALGORITHM TO MULTIPROCESSOR IMPLEMENTATION

A contour vertex is the line demarcation between an active and an inactive pixel in a digital image. The vertex-following (VF) algorithm scans these contour vertices (as shown in Fig.8a) and pushes the pair of pixels *containing* the contour line (vertex) onto the common shared stack. The scanning of the image pixels is done similar to the pixel-following algorithm, *i.e.,* from top to bottom and from left to right. The demarcation between any two pixels is a vertex if the value of the two pixels is *unequal* in the grey-scaled, thresholded image.

The VF algorithm need not be a recursive algorithm, since one sequential scan over all the pixels gives all the vertices of the image. To store a vertex, the pixel pair $\{(x_1, y_1);(x_2, y_2)\}$ is pushed onto the common shared stack. To optimize the algorithm further, we compare only the east and the south neighbours of each pixel with themselves to reduce the number of redundant comparisons. Similar to the adaptation of the PF algorithm, the VF algorithm is adapted for a multiprocessor system by assigning each pixel to each processing element and hence converting all the computations to communications between the processing elements.

Fig.8a shows the traced boundary in the test image using the VF algorithm, and Fig.8b shows the resultant boundary pixel pairs stored in the common shared stack. Algorithm2 shows the pseudocode for our adaptation of the VF algorithm.

In comparison to the PF algorithm, the VF algorithm needs a larger stack size since pixel pairs are stored. Also, additional processing is required while retrieving the image from the vertices stored as pixel pairs.

Algorithm 2 Vertex-Following Algorithm

```
Result: Returns a stack of corners (node,neighbor)
processorNodesUsed = pixelsInImage;
for All nodes do
    for east and south sorted neighbors of node do
        if node_neighborIsCorner not set then
            if pixelValueNeighbor != pixelValueNode then
                push (node,neighbor) to cornerStack;
                set node_neighborIsCorner;
```

J. ADAPTATION OF RUN-DATA-BASED-FOLLOWING ALGORITHM TO MULTIPROCESSOR IMPLEMENTATION

In the run-data-based-following (RDBF) algorithm, the scan of pixels is done in the form of horizontal scan lines from left to right and sequentially from top to bottom. Within each of these scan lines, small active line segments are detected, which are the active pixels of the image. The endpoints $\{(x_1, y_1);(x_2, y_2)\}$ of these active line segments are stored in the common shared stack representing the contour of the image. Since the comparison is done only along these horizontal scan lines, each pixel is supposed to be compared with the west and east neighbours only. To further optimize the algorithm, we have compared each pixel with only its east neighbour to avoid comparison redundancy.

Algorithm 3 shows the pseudocode for the adaptation of the RDBF algorithm for a multiprocessor system. Fig.9a shows the line segments scanned to detect the contour, and Fig.9b shows the resultant line segment ends representing the contour.

The stack required for the RDBF algorithm is very small in comparison to the PF and the VF algorithms, especially for dense images. However, additional processing is required to generate the original image from the ends of line segments representing the contour. The contour stack generated by the RDBF algorithm can easily be scaled to scale the size of the image.

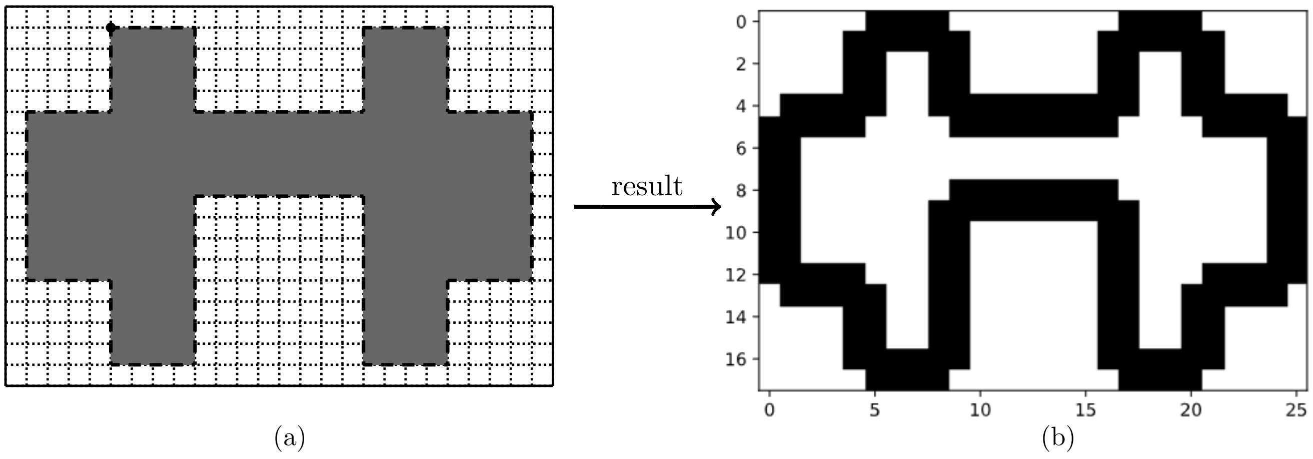


FIGURE 8. (a) Tracing of the standard image using our implementation of the vertex-following algorithm; dot represents the starting of the first vertex detected; (b) The resultant pixels in the contour stack.

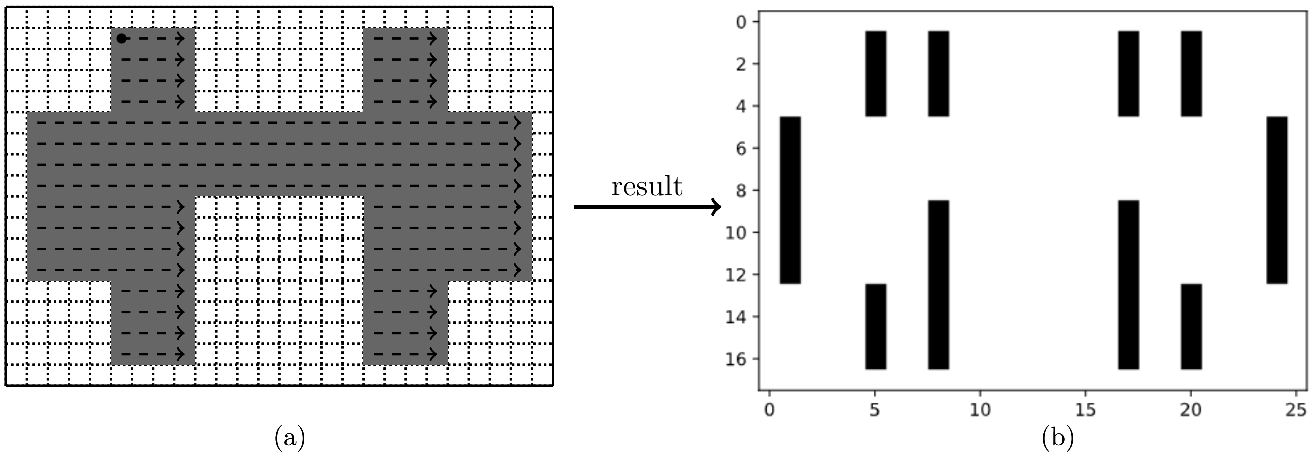


FIGURE 9. (a) Tracing of the standard image using our implementation of the run-data-based-following algorithm; dot represents the first active pixel detected; (b) The resultant pixels in the contour stack.

Algorithm 3 Run-Data-Based-Following Algorithm

```
Result: Returns a stack of line segments (x,y)
processorNodesUsed = pixelsInImage;
for All nodes do
    for east neighbor of node do
        if pixelValueNeighbor != pixelValueNode then
            if pixelValueNode == 0 then
                push (node) to lineSegmentStack;
            else
                push (neighbor) to lineSegmentStack;
```

K. EFFICIENT PARALLEL ALGORITHMS FOR A MULTIPROCESSOR SYSTEM USING SEGMENTATION

In the third step in our methodology, the array of microprocessors is segmented into symmetric parts. This leads to the segmentation of the image being processed, as shown in Fig.10(b). Since, in our implementation of the contour tracing algorithms, any active pixels on the corners of the image are not considered as a part of the contour, the resulting contour stacks from different segments of the image can be combined in *any order* to form the complete contour stack of the image. Any extra processing, to merge the boundary pixels of the segments of the image, is not required.

As the fourth step, one of the three algorithms discussed is run on these segmented hardware sections in parallel to achieve the hypothesized speed up in contour tracing. The resulting algorithms are termed Adapted and Segmented (AnS) algorithms for contour tracing.

L. METRIC USED FOR COMPARISON BETWEEN DIFFERENT ALGORITHMS AND THEIR ADAPTATIONS

In our adaptation of the PF, VF, and RDBF contour tracing algorithms, pixel value comparisons have been converted into communications between processors. *One* pixel value comparison has, thus, been converted into *one* hop communication between two neighbouring processors. To normalize the time taken for the comparison operation and the communication operation, we have defined one comparison or one communication operation as one time-tick. This

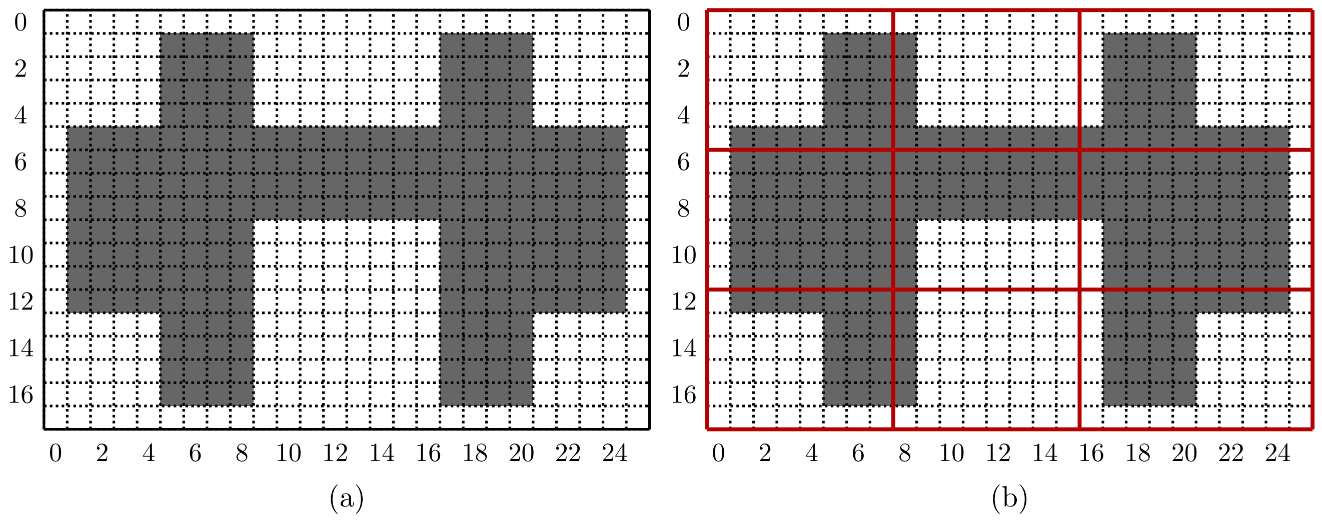


FIGURE 10. Pictorial explanation of the concept of the paper – (a) Distribution of image pixels in a multiprocessor system; Each dotted square in the 2-D array is a processing element; (b) The multiprocessor hardware system divided into 3 × 3 segments.

means that when one pixel value is compared with one neighbour value, one time-tick is used in both conventional implementation and our adaption of the PF, VF and RDBF algorithms. This metric of time-ticks is used for comparison between the algorithms and their adaptations.

To calculate the time taken for our algorithms to process the test image, we used python as the simulation tool. We used the *networkx* package [36] in python to model the torus interconnected parallel processor system. The memory connected to each of the PE was modelled as the local variables of each node in networkx. Based on the segmentation of the image, the pixel comparisons were performed in parallel or sequentially using the framework. To calculate the total time taken for an algorithm run on the test image, we increase the number of time-ticks as the comparison/communication proceeds sequentially. When the algorithm runs in parallel on different segments of the image, the time ticks are calculated for individual segments and normalized over the number of segments. Since the processing time taken for one comparison operation by a single processor, Δ_p, and the communication time taken for one bit communication between two neighbouring processors, Δ_c, will not be equal, we would have to multiply the normalized speed-up by a factor of $\frac{\Delta_p}{\Delta_c}$, corresponding to the hardware system being used for implementation.

The second metric used for comparison is the size of the common shared stack used for storing the contour pixels generated by different algorithms and their multiprocessor versions.

M. PROPOSED IMPLEMENTATION ON NVIDIA GPUs USED IN EXISTING PARALLEL PROCESSING ALGORITHMS

For performance analysis, we have designed a methodology to implement our algorithms using the NVIDIA GPUs used in [28] and [30]. The pre-processed image (after greyscaling and thresholding, respectively) is stored in the global memory of the GPU. Each byte of a coloured image leads to a single bit after pre-processing. To reduce the memory space and enable parallel processing, each word (32-bit long) is used to store 32 row-wise adjacent pixels (in bits). A single-bit comparison is modelled as a single-bit XOR operation. Since the AnS-RDBF algorithm is observed to have the best performance (detailed in Section IV), we implement the AnS-RDBF algorithm only on the GPU hardware. In case of complete parallelism (Number of Segments = Number of pixels in the image = 18×26 in Table 1), each PE compares a single pixel (32 pixels stored as a 32-bit word in this case) of the image. This leads to complete image contour tracing in just one time tick. However, the number of CUDA cores available in the GPU hardware is a fraction of the number of pixels in the image, thus, the image is segmented depending on the number of CUDA cores available. The segments of the image are operated on serially for complete contour tracing.

For each segment of the image, the number of CUDA processors×32 single-bit XOR operations are performed in parallel. These operations are programmed to be SIMD threads in CUDA [37], [38]. The complete data required for contour tracing on one segment of the image is stored in the shared memory of the GPU to minimise latency. After the contour tracing of one segment completes, the contour stack is transferred to the global memory and the pixels corresponding to the next segment are transferred to the shared memory. This continues till the complete image has been traced.

IV. RESULTS

A. COMPARISON BETWEEN THE CONVENTIONAL ALGORITHMS AND THEIR MULTIPROCESSOR VERSIONS

In Table 1, we compare the execution time of these three new algorithms to their single processor counterparts to quantify the speed-up. We use two metrics for these comparisons – (a) the amount of time taken (in normalized time-ticks) and (b) the size of the stack generated for contour tracing in a standard image.

TABLE 1. Performance speed-up of the three algorithms for the different number of segments.

Number of Segments	Pixel-Following (PF)		Vertex-Following (VF)		Run-Data-Based-Following (RDBF)	
	Time Ticks	Speed-up	Time Ticks	Speed-up	Time Ticks	Speed-up
1 (Single Processor)	1320	1	936	1	468	1
$1\times1^* = 1$	1320	1	936	1	468	1
$2\times2^* = 4$	694	2	234	4	117	4
$3\times3^* = 9$	404	3	96	10	48	10
$6\times6^* = 36$	222	6	24	39	12	39
$9\times9^* = 81$	212	6	8	117	4	117
$13\times13^* = 169$	212	6	4	234	2	234
$18\times18^* = 324$	210	6	2	468	1	468
$18\times26^* = 468$	210	6	2	468	1	468
*with 18 x 26 processors						

The number of time-ticks observed for adaptation of each of the algorithms, for a multiprocessor system was the same as that of the single processor implementation because each comparison got converted into a single hop communication. We have normalized the speed-up by a factor of $\frac{\Delta_c}{\Delta_p}$, where Δ_c is the single hop communication time, and Δ_p is a single comparison process time, to quantify the speed-up obtained by exclusively parallelizing the algorithms. As we introduced data-level parallelism through segmentation, the number of time-ticks decreased (speed increased) with an increase in the number of segments, as shown in Table 1 and Fig.11. The speed-up is calculated as $\frac{TT_s}{TT_m}$, where TT_s is the number of time ticks taken by the single processor implementation and TT_m is the number of time ticks taken by the multiprocessor implementation. Also, as the number of segments increases, the contour points are pushed onto the stack in random order because of the parallel running algorithm instances. This might require additional buffer hardware to access the stack. Pre-ordering of the stack before reconstruction of the image from the contour pixels might also be needed in some cases.

In Table 2, we show the comparison of time-ticks and the contour stack size generated by the single processor implementation of the PF, VF and RDBF algorithms. The percentage of memory saved is equal to $\frac{p_i - p_c}{p_i} \times 100$, where p_i is the number of pixels in the image, and p_c is the number of pixels in the contour.

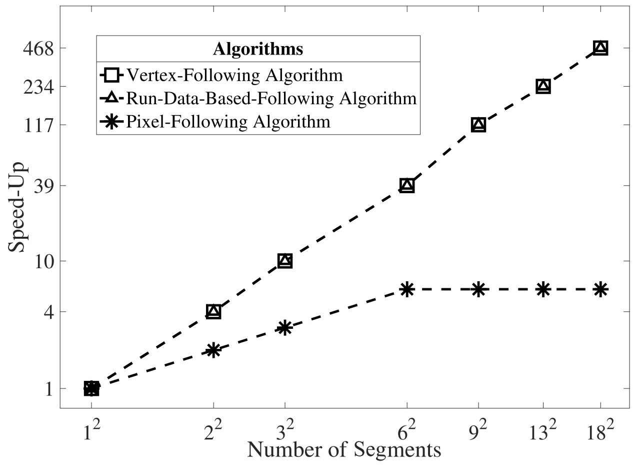


FIGURE 11. Speed-up as a function of number of segments in log-log scale.

TABLE 2. Comparison between the adaptation of the three algorithms on the basis of various parameters.

	Pixel Following	Vertex Following	Run Data Based Following
Time-Ticks	1320	936	468
Number of pixels in Contour	92	188	56
Number of pixels in Image	350	350	350
Percentage of Memory saved	74%	46%	84%

B. COMPARISON OF THE INCREASED EFFICIENCY IN THE THREE ALGORITHMS

The comparative results obtained from our experimentation are -

1) AnS algorithms, using a multiprocessor system corresponding to the PF, VF, and RDBF algorithms, are faster than the corresponding conventional implementations for contour tracing when the number of segments is greater than one.
2) a) If the number of processors (n) is equal to the number of pixels (p), as in our experiment, the speed of contour tracing using our algorithms increases to up to n-times as the number of segments is increased because, when the number of segments is equal to the number of processing elements (maximum segmentation), each processing element (or segment) performs only basic comparison operation with the neighbours of only *one* pixel.
   b) However, if $n > p$, then speed up will be limited by the number of pixels (also the number of active processing elements) and can be up to p-times.
   c) Also, if $p > n$, then there can be two cases. (I) If the image is down-sampled to match the number of processors, the case becomes the same as ii)(a) and a speed of up to n-times is observed. (II) If the image is divided into $\lceil \frac{p}{n} \rceil$ parts of n length each, processed sequentially (details in section III-G), the maximum speed-up reduces to $\frac{n}{p/n} = \frac{n^2}{p}$. Thus, in general, we can say that the speed up using our algorithms can be up to $\frac{n^2}{p}$ times.

TABLE 3. Performance of our methodology using GPU hardware used by existing parallel processing implementations.

Total Image Pixels (in bits)	GPU used	# of Multiprocessors	Device Memory size (in GB)	Compute Capability	# of Results per clock cycle	Base Frequency (in MHz)	Total # of results in a clock cycle	# of clock cycles for total image computation	Time taken for total computation (in ns)
P		M			R	f	$R_c = MR \times 32$	$C = \frac{P}{R_c}$	$T_c = \frac{f}{C}$
1×2^{20}	Tesla K40c	15	12	3.5	160	745	76800	13.65	18.33
1×2^{20}	Quadro RTX 5000	48	16	7.5	64	1620	98304	10.67	6.58
6.11×2^{30}	Tesla K20c	13	5	3.5	160	745	66560	98,566.14	132,303.55
21.44×2^{30}	Tesla K20c	13	5	3.5	160	745	66560	345,868.76	464,253.37
37.81×2^{30}	Tesla K20c	13	5	3.5	160	745	66560	609,948.59	818,722.94

TABLE 4. Memory access time and total time of our methodology using GPU hardware used by existing parallel processing implementations.

Total Image Pixels (in bits)	GPU used	# of SM	Memory bus width (in bit)	Shared memory per SM (in KB)	Memory Bandwidth (in GB/s)	Total # of results in a clock cycle	# of clock cycles for total image computation	Total time taken for memory access (in ns)	Total time taken (in ns)
P		M	W	S	B	$R_c = MR \times 32$	$C = \frac{P}{R_c}$	$T_m = \frac{R_c C}{WB \times 8}$	$T = T_c + T_m$
1×2^{20}	Tesla K40c	15	384	48	288.4	76800	13.65	1.18	19.51
1×2^{20}	Quadro RTX 5000	48	256	64	488	98304	10.67	1.05	7.63
6.11×2^{30}	Tesla K20c	13	320	48	208	66560	98,566.14	12,320.77	144,624.32
21.44×2^{30}	Tesla K20c	13	320	48	208	66560	345,868.76	43,233.60	507,486.97
37.81×2^{30}	Tesla K20c	13	320	48	208	66560	609,948.59	76,243.57	894,966.52

3) Converting the comparison between pixels to communication between processing elements, *i.e.*, just the adaptation of the single processor algorithms for a multiprocessor system, is not sufficient to increase the speed of contour tracing. Segmentation is additionally required for parallelization and thus speed-up.
4) The AnS-RDBF algorithm was found to be the fastest and the most memory-efficient contour tracing algorithm, amongst our algorithm implementations, for digital images.
5) The pixel-following algorithm does not offer much improvement in the speed with the increase in segmentation.
6) We also observed that the improvement in speed for both the VF and the RDBF algorithms is exactly the same, for the same number of segments, as seen in Fig.11 and Table1. This happens because the number of time-ticks for the RDBF algorithm comes out to be exactly half of that for the VF algorithm in both the adapted algorithms and the AnS algorithms since these both are constant comparison algorithms. In the VF algorithm, two neighbours (east and south) of each pixel are compared with itself, while in the RDBF algorithm, only one neighbour (east) is compared. This makes the time-ticks for the RDBF algorithm exactly half of the VF algorithm for all cases.

C. PERFORMANCE COMPARISON BETWEEN THE EXISTING PARALLEL PROCESSING ALGORITHMS AND OUR IMPLEMENTATION

We have compared our implementation with parallel processing implementations of Garcia-Molla et al., [28] and Zhou et al., [30]. Garcia-Molla et al., have used Server 1 with a Tesla K40c GPU card and Server 2 with an Nvidia Quadro RTX 5000 for CUDA-based implementation. Zhou et al., have used an Nvidia Tesla K20c device for three sizes of the DEM data set. Detailed performance calculations and respective comparisons, using the methodology detailed in Section III-M, are shown in Table 3, 4 and 5.

TABLE 5. Time comparison between the performance of our methodology and the existing parallel processing methodologies by Garcia-Molla et al., [28] and Zhou et al., [30].

Image Size	GPU used	Existing implementation	Our implementation	Speed-Up
(in MB)		T_e (in ns)	T (in ns)	SU = $\frac{T_e}{T}$
1	Tesla K40c	3.79×10^3 [28]	1.95×10^1	194
1	Quadro RTX 5000	3.60×10^2 [28]	7.63	47
6.11×10^3	Tesla K20c	1.16×10^{11} [30]	1.45×10^5	800,695
21.44×10^3	Tesla K20c	3.19×10^{11} [30]	5.07×10^5	628,982
37.81×10^3	Tesla K20c	4.97×10^{11} [30]	8.95×10^5	555,775

In comparison to the parallel processing framework developed by Garcia-Molla et al., we observe a speed-up of 194 and 47 for Tesla K40c and Quadro RTX 5000 respectively. The Suzuki algorithm used by them is based on the Moore Neighbour Tracing algorithm which is a part of the Pixel-Following family of contour tracing algorithms. We observed that the RDBF algorithms, used for our implementation, are faster for parallel processing than the PF algorithms. This contributes to the speed-up observed using our GPU hardware implementation framework.

In comparison the implementation by Zhou et al., we observe a speedup of 8×10^5, 6.3×10^5 and 5.6×10^5 corresponding to the DEM data-set of 6.11, 21.44 and 37.81 GB respectively, for Tesla K20c GPU. One of the reasons for this speed-up is the reduction of data size owing to the pre-processing performed in our methodology. Zhou et al., have processed the large DEM data set using close interaction of the CPU and GPU, but, we have transformed the data into single-bit pixels and used only the GPU for parallel processing. This led to reduced time-intensive communications between the global and the shared memory in our implementation. They have also used the CPU for post-processing the broken contours, which further added up to the processing time of their implementation.

Though the speed-up observed in comparison to the methodology presented by Zhou et al., is very optimistic, we speculate that our implementation would lead to less accurate contour tracing for the same DEM data set. This loss of accuracy is adequate for our use case – real-time image processing applications like tracking a moving object, character recognition etc.

V. DISCUSSION

The results obtained from our experimentation of adapting and segmenting conventional contour tracing algorithms can be used to increase the speed of image processing or pre-processing of an image, without any hardware modifications. Our AnS algorithms can be easily plugged into any torus-connected multiprocessor system. The speed-up observed will be especially high for systems optimized for faster communication between processors, *i.e.,* the systems with $\frac{\Delta_p}{\Delta_c}$, where Δ_p is one bit comparison time and Δ_c is one bit communication time, greater than one. However, even in the case of systems with $\frac{\Delta_p}{\Delta_c}$ less than one, our algorithms provide a good factor of speed-up.

Based on the application, any of these three AnS algorithms can be used to obtain this speed up. In case there is no constraint on the choice of algorithm, the AnS-RDBF algorithm is preferred. The speed-up is directly proportional to the number of segments for the AnS-VF and the AnS-RDBF algorithms, though segmentation incurs additional software costs. The trade-off between the software cost of segmentation and the speed-up can be decided based on the application requirement and hardware-software constraints.

These algorithms are especially effective for applications where an array of embedded systems is used for capturing the image. An interesting application can be in detecting a fast-moving object in a sports video sequence (ex: a football or a baseball moving across a playfield) - with overlapping fields of view from networked cameras covering the entire playing field. Since tracing of a moving object across a sequence of images is similar to contour tracing in an image, our AnS algorithms can significantly speed up the processing in this application.

The algorithms presented in this paper can be used to improve the efficiency of image processing in various applications without any hardware upgrade given that affordable multi-processor cards are easily available.

VI. CONCLUSION

We have adapted three existing contour tracing algorithms for a multi-processor, 2-D mesh or torus-connected system to introduce data-level parallelism. The system was further segmented to run the adapted algorithm in parallel for different sections of the image. We observed a speedup of up to $\frac{n^2}{p}$ times, where n is the number of processing elements and p is the number of pixels in the system. This speedup was observed for two of the three algorithms, *i.e.,* vertex-following and run-data-based-following. Out of the three algorithms developed, run-data-based-following is observed to have the fastest speed and lowest memory consumption for the same image.

In future, this methodology of introducing data-level parallelism can be extended to different algorithms and network topology for potential speed up. Application specific parameters like communication delays and software overhead can also be evaluated in detail by implementing the algorithms on Field Programmable Gate Arrays or Application-Specific Integrated Circuits.

ACKNOWLEDGMENT

The authors would like to thank the Electrical Engineering Department, Indian Institute of Technology, Delhi, for providing the resources to enable these experiments. They would also like to thank Bhavya Jain, Ayush Kumar, and Shubh Goel from the Electrical Engineering Department, IIT Delhi, for helping with implementing the codes used in this article.

STATEMENTS AND DECLARATIONS

The authors declare that they have no conflict of interest. The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] S. Zhang, "Review of modern field effect transistor technologies for scaling," *J. Phys., Conf. Ser.*, vol. 1617, no. 1, Aug. 2020, Art. no. 012054.

[2] Y. W. Kim, S. H. Choi, and T. H. Han, "Rapid topology generation and core mapping of optical network-on-chip for heterogeneous computing platform," *IEEE Access*, vol. 9, pp. 110359–110370, 2021.

[3] F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads," *IEEE Trans. Comput.*, vol. 70, no. 11, pp. 1845–1860, Nov. 2021.

[4] M. C. Heydemann, "Cayley graphs and interconnection networks," in *Graph Symmetry: Algebraic Methods and Applications*, G. Hahn and G. Sabidussi, Eds. Dordrecht, The Netherlands: Springer, 1997, pp. 167–224.

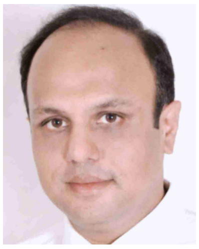
[5] G. M. Almeida, S. Varyani, R. Busseuil, G. Sassatelli, P. Benoit, L. Torres, E. A. Carara, and F. G. Moraes, "Evaluating the impact of task migration in multi-processor systems-on-chip," in *Proc. 23rd Symp. Integr. Circuits Syst. Design (SBCCI)*, 2010, pp. 73–78.

[6] S. Ranka and S. Sahni, *Hypercube Algorithms: With Applications to Image Processing and Pattern Recognition*. New York, NY, USA: Springer, 1990.

[7] S. Lakshmivarahan and S. K. Dhall, "Ring, torus and hypercube architectures/algorithms for parallel computing," *Parallel Comput.*, vol. 25, nos. 13–14, pp. 1877–1906, Dec. 1999.
[8] K. Day and M. H. Al-Towaiq, "A parallel Gauss-seidel algorithm on a 3D torus network-on-chip architecture," in *Proc. 9th Int. Workshop Interconnection Netw. Archit., Chip, Multi-Chip*, Jan. 2015, pp. 13–16.
[9] S. K. Jha, "An improved parallel prefix computation on 2d-mesh network," in *Proc. Int. Conf. Comput. Intell., Modeling Techn. Appl., Proc. Technol.*, 2013, pp. 919–926.
[10] A. Datta, M. De, and B. P. Sinha, "Fast parallel algorithm for prefix computation in multi-mesh architecture," *Parallel Process. Lett.*, vol. 27, nos. 3–4, Dec. 2017, Art. no. 1750009.
[11] A. Gupta, "Optimal parallel prefix sum computation on three-dimensional torus network," *Nat. Acad. Sci. Lett.*, vol. 44, no. 5, pp. 423–426, Oct. 2021.
[12] A. Kazlouski and R. K. Sadykhov, "Plain objects detection in image based on a contour tracing algorithm in a binary image," in *Proc. IEEE Int. Symp. Innov. Intell. Syst. Appl. (INISTA) Proc.*, Jun. 2014, pp. 242–248.
[13] Y. Murakami, "FPGA implementation of a SIMD-based array processor with torus interconnect," in *Proc. Int. Conf. Field Program. Technol. (FPT)*, Dec. 2015, pp. 244–247.
[14] D. L. Tuomenoksa, I. Adams, H. J. Siegel, and O. R. Mitchell, *Parallel Algorithm for Contour Extraction: Advantages and Architectural Implications*. Silver Spring, MD, USA: IEEE Computer Society Press, 1983, pp. 336–345.
[15] A. A. El-Moursy and F. N. Sibai, "Image processing applications performance study on cell BE and blue gene/L," *Concurrency Comput., Pract. Exper.*, vol. 23, no. 4, pp. 351–371, Sep. 2010.
[16] N. L. Narappanawar, B. M. Rao, T. Srikanth, and M. Joshi, "Vector algebra based tracing of external and internal boundary of an object in binary images," *J. Adv. Eng. Sci.*, vol. 3, no. 1, pp. 57–70, 2010.
[17] Y. Cao, J. Mao, H. Yu, Q. Zhang, H. Wang, Q. Zhang, L. Guo, and F. Gao, "A novel hybrid active contour model for intracranial tuberculosis MRI segmentation applications," *IEEE Access*, vol. 8, pp. 149569–149585, 2020.
[18] A. Niaz, A. A. Memon, K. Rana, A. Joshi, S. Soomro, J. S. Kang, and K. N. Choi, "Inhomogeneous image segmentation using hybrid active contours model with application to breast tumor detection," *IEEE Access*, vol. 8, pp. 186851–186861, 2020.
[19] J. Zhang, J. Liu, X. Xu, P. Gong, and M. Duan, "TSER: A two-stage character segmentation network with two-stream attention and edge refinement," *IEEE Access*, vol. 8, pp. 205216–205230, 2020.
[20] Y. Sun, X. Mao, S. Hong, W. Xu, and G. Gui, "Template matching-based method for intelligent invoice information identification," *IEEE Access*, vol. 7, pp. 28392–28401, 2019.
[21] S. Pei, L. Li, L. Ye, and Y. Dong, "A tensor foreground-background separation algorithm based on dynamic dictionary update and active contour detection," *IEEE Access*, vol. 8, pp. 88259–88272, 2020.
[22] L. Xu, S. Yan, X. Chen, and P. Wang, "Motion recognition algorithm based on deep edge-aware pyramid pooling network in human–computer interaction," *IEEE Access*, vol. 7, pp. 163806–163813, 2019.
[23] H. Li, J. Liu, and X. Zhou, "Intelligent map reader: A framework for topographic map understanding with deep learning and gazetteer," *IEEE Access*, vol. 6, pp. 25363–25376, 2018.
[24] J. Seo, S. Chae, J. Shim, D. Kim, C. Cheong, and T.-D. Han, "Fast contour-tracing algorithm based on a pixel-following method for image sensors," *Sensors*, vol. 16, no. 3, p. 353, Mar. 2016.
[25] I. Agi, P. J. Hurst, and A. K. Jain, "A VLSI processor for parallel contour tracing," *IEEE Trans. Signal Process.*, vol. 40, no. 2, pp. 429–438, Feb. 1992.
[26] K. Ratnayake and A. Amer, "A real-time implementation of chaotic contour tracing and filling of video objects on reconfigurable hardware," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Oct. 2007, pp. 1089–1094.
[27] T. Chia, "A parallel algorithm for generating chain code of objects in binary images," *Inf. Sci.*, vol. 149, no. 4, pp. 219–234, Feb. 2003.
[28] V. M. Garcia-Molla, P. Alonso-Jordá, and R. García-Laguía, "Parallel border tracking in binary images using GPUs," *J. Supercomput.*, vol. 78, no. 7, pp. 9817–9839, May 2022.
[29] S. I. Editor, "A parallel approach for contour extraction based on CUDA platform," *Int. J. Simulation: Syst., Sci. Technol.*, Jan. 2016.
[30] C. Zhou and M. Li, "A systematic parallel strategy for generating contours from large-scale DEM data using collaborative CPUs and GPUs," *Cartography Geographic Inf. Sci.*, vol. 48, no. 3, pp. 187–209, May 2021.
[31] T. Miyatake, H. Matsushima, and M. Ejiri, "Contour representation of binary images using run-type direction codes," *Mach. Vis. Appl.*, vol. 9, no. 4, pp. 193–200, Feb. 1997.
[32] A. Rosenfeld, "Connectivity in digital pictures," *J. ACM*, vol. 17, no. 1, pp. 146–160, 1970.
[33] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, "Blue Gene/L torus interconnection network," *IBM J. Res. Develop.*, vol. 49, no. 2.3, pp. 265–276, Mar. 2005.
[34] D. W. Capson, "An improved algorithm for the sequential extraction of boundaries from a raster scan," *Comput. Vis., Graph., Image Process.*, vol. 28, no. 1, pp. 109–125, Oct. 1984.
[35] C.-H. Cheong and T.-D. Han, "Improved simple boundary following algorithm," *J. KIISE, Softw. Appl.*, vol. 33, no. 4, pp. 427–439, 2006.
[36] (2022). *NetworkX: Network Analysis in Python*. Accessed: Nov. 16, 2022. [Online]. Available: https://networkx.org
[37] (2022). *CUDA C++ Programming Guide*. Accessed: Nov. 9, 2022. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[38] (2022). *CUDA C++ Best Practices Guide*. Accessed: Nov. 9, 2022. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html

SONAL GUPTA received the B.E. degree (Hons.) in electronics and electrical communication from the Punjab Engineering College, Chandigarh, India, in 2011, and the M.Tech. degree in electrical engineering, with a specialization in electronic systems, from the Indian Institute of Technology Bombay (IIT Bombay), Mumbai, India, in 2015. She is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, Indian Institute of Technology Delhi, New Delhi, India.

She has worked with the Embedded Controller Group, Texas Instruments, Bengaluru, India, from 2011 to 2018, during which she also completed her sponsored M.Tech. degree from IIT Bombay. Her research interest includes enhancing the connectivity of richly connected embedded systems.



SUBRAT KAR (Senior Member, IEEE) received the degree (Hons.) in electrical and electronics engineering from the Birla Institute of Technology and Science, Pilani, in 1987, and the Ph.D. degree in electrical communication engineering from the Indian Institute of Science, Bengaluru, in 1991.

From 1991 to 1994, he was with the International Center for Theoretical Physics, Trieste, Italy, as a Postdoctoral Fellow. He is currently a Professor at the Department of Electrical Engineering, Indian Institute of Technology Delhi, where he is also the Ram and Sita Sabnani Chair Professor. As a member of the Optoelectronics and Optical Communication Research Group, he works in the area of non-linear optical CDMA networks, free-space optical communication (ground-satellite and inter-satellite), and ultra-fast optical LSI and fault-tolerant integrated optical switching architectures. He has designed and holds patents in the field of large-scale sensor networks, routing algorithms, macro languages, large-scale repository design for sensor data and localization issues in sensor networks. His research interests include optical communication, switching, access technologies, telecom protocols, embedded systems, and high-speed networks. His research interests also involve formalisms in embedded system design, hardware-software co-design, and telecom protocol design and verification tools for telecommunication protocols.