Parallel Edge Detection Using Uni-Directional MultiRing on Spiral Architecture

Xiangjian He

Department of Computer Systems
University of Technology, Sydney
PO Box 123, Broadway 2007
Australia
E-mail: sean@it.uts.edu.au

Hamid R. Arabnia

Department of Computer Science
University of Georgia
Athens, GA 30602
USA
Email: hra@cs.uga.edu

Abstract

Improving the computation efficiency is the key issue in image processing, especially in edge detection, because edge detection is very computationally intensive. With the development of real-time image processing application, fast processing response is becoming the major requirement in this area. In this paper, a parallel and distributed algorithm on Spiral Architecture for edge detection using a uni-directional MultiRing is proposed. The proposed algorithm is based on Master-Slave model. It guarantees a better load balancing among the processors (or nodes) and greatly reduces the traffic from and to the master node. It uses each MultiRing configuration more efficiently. It also improves the performance for message passing among the modes on the MultiRing.

Keywords: Spiral Architecture, MultiRing, Distributed Processing, Edge Detection

1. Introduction

Edge detection is based on the relationship a pixel has with its neighbors. It extracts and localizes points (pixels) around which a large change in image brightness has occurred.

Gaussian Multi-Scale theory introduced by Lindeberg is often applied to edge detection for images with noise (see, for example, [1]). In this theory, image representation (or image brightness function) is parameterized using a scale. The parameterized (or scaled) function for image representation is a convolution of the original image function with a Gaussian kernel. Image is blurred and noise is reduced or suppressed when the parameter (or scale) is positive. For a given scale, edge point of the blurred image with the scale is defined as a pixel at which the gradient magnitude of the scaled function reaches its maximum. Details of Gaussian scaled representation and edge definition can be found in another paper that will also appear in this conference proceedings.

The computation for Gaussian convolution is very massive. It often consumes most of CPU time in processing edge detection. The computation is conducted pixel by pixel sequentially across the entire image space, and several dozens of arithmetic operations are performed for each pixel. Furthermore, fast processing response is a major requirement in many image processing applications, such as in real-time processing where a sequence of image frames should be processed in a very short time. The above suggests parallel algorithms for fast processing.

The parallel algorithm to be proposed in this paper is based on a cost-effective MultiRing [2] computer network. The MultiRing network consists of eight (8) computers such as PCs. These computers are physically interconnected through a low-layer switch as displayed in Figure 1. The effectiveness of MultiRing system is

its *reconfigurability*. That is, according to the need, the eight computers (or nodes) can be configured as a single logical ring (network) having eight nodes, two separate ring networks of which each has four nodes, or four separate ring networks of which each has two nodes (see Figure 2).

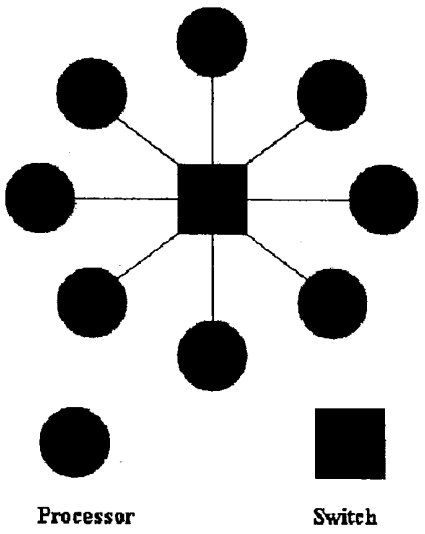

Figure 1. Physical topology of an 8-node MultiRing

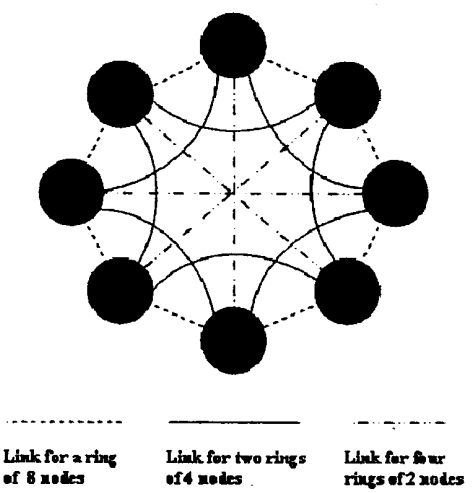

Figure 2. Configurations of an 8-node MultiRing

The computers in each ring can be configured to sends messages or data in a unique direction or in both directions. In this paper, a unidirectional MultiRing is used for our edge detection. That is, every ring created from the MultiRing and configured for edge detection sends messages and data in the same single direction. This provides a better balance of using the links for computer connections on the MultiRing, and hence further increases parallel processing speed for edge detection. By using the unidirectional MultiRing network in this paper, the traffic to and from the master and the burden of master node are greatly reduced. The use of the unidirectional MultiRing makes it possible to overlap the processing of two consecutive images. Each ring configuration can be used by two images simultaneously. This greatly improves the efficiency of using ring configurations of the MultiRing, and hence improves the parallel performance.

The detection algorithm to be presented in this paper uses a recently discovered image structure inspired from a primate's vision system, called Spiral Architecture [3]. The image in this relatively new image structure is represented by a collection of hexagons of the same size. The importance of the hexagonal representation is that it possesses special computational features that are pertinent to the vision process. In this image structure, image can be easily and uniformly separated into many similar sub-images through an operation called Spiral Multiplication. By uniformly partitioning image and assigning sub-images to various computer nodes, working load can be balanced among all the nodes in a distributed processing system. So the system performance can be very well improved.

The context of this paper is arranged as follows. In Section 2, we construct a MultiRing for the edge detection algorithm in this paper. Methods for data flow on the MultiRing are demonstrated in Section 3. The algorithm for parallel and distributed edge detection is presented in Section 4. We conclude in Section 5.

2. MultiRing for Edge Detection

In this section, we will construct a MultiRing topology for the edge detection algorithm to be presented in next section.

2.1. The MultiRing

The MultiRing in this paper consists of eight nodes connected to a MultiRing switch as shown in Figure 1.

One node acts as the master node, and other seven nodes act as slave nodes. The message from one node can only be sent in a unique direction (for example, in

the counter clockwise direction which will always be used in this paper) to next node in a ring configured by the switch. We assume that the nodes in the MultiRing can only send and receive messages when the MultiRing has been configured to 1 ring of 8 nodes, 2 rings of 4 nodes, or 4 rings of 2 nodes as requested. This requests each node to keep a *Neighbour Table* that maintains the information of its previous node and next node in a ring, and a *Routing Table* that tells the address of the next node for a message to be forwarded to the destination on the MultiRing. The information contains the balanced ring to reach the previous and next nodes, and the addresses of the nodes.

2.2. Neighbour Tables and Routing Tables

The neighbour table on each node has *n* entries for a 2^n-processor MultiRing. Each entry corresponds to a ring configuration. Each entry has the fields of *ring configuration* (e.g., 2 rings of 4 nodes), *previous node* and *next node* as shown in Figure 3.

Ring Configuration	Previous Node Address	Next Node Address

Figure 3. Entry on a Neighbour Table

The routing table on each node has 2^n entries for a 2^n-processor MultiRing. The number of entries can be reduced to *n* for scalability. We will show this improvement in another paper for MultiRing implementation. Each entry on the routing table contains the destination address and the next node address to reach the destination as shown in Figure 4.

Destination Node Address	Next Node Address

Figure 4. Entry on a Routing Table

When a message is created, it contains its source and destination addresses. The sending node checks the routing table for the next node address and then finds the ring configuration to forward the message to its next node based on its neighbour table. It is then sent to the next node on a ring configured. The source and destination addresses are detected by the next node, which decides to forward the message to its next node during the next ring configuration or keep the message based on the destination address, and finds the ring configuration for forwarding the message based on the source address.

For example, as shown in Figure 5, Processor (or node) P_0 has three logical links to P_1, P_2 and P_4 respectively. When P_0 wants to send a message to P_6, it looks at its routing table and finds that the next node to reach P_6 is node P_2 as displayed in Figure 3. It then checks its neighbour table and finds that in order to reach P_2 the ring configuration must be 2 rings of 4 nodes. Hence, P_0 waits for the ring configuration and sends the message to P_2. When P_2 receives the message, P_2 checks the destination for the message and forwards the message to P_6 trough a ring of 2 nodes based on the information on its neighbour table.

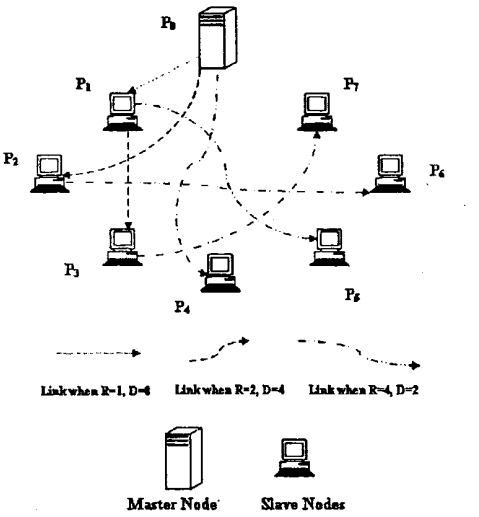

Figure 5. Logical Links for a MultiRing

2.3. The MultiRing Switch

MultiRing switch periodically configures the MultiRing in the order of 1 ring of 8 nodes, 2 rings of 4 nodes and 4 rings of 2 nodes. It signals all nodes the current ring configuration. It may re-set the duration for each ring configuration based on the need. Each ring configuration is allowed for multiple messages to be forwarded from one node to its next node. After the MultiRing is configured, the MultiRing switch receives the messages sent from any node in a configured ring and forwards them to the next node on the same ring. All nodes on the MultiRing can send and receive messages at the same time (or simultaneously) according to the ring configuration. The detailed discussion for how the MultiRing switch configures a ring is beyond this paper, and will be presented in another paper.

3. Data Flow in the MultiRing

Before we present our new algorithm for edge detection in next section, we present here image data flow in the MultiRing created in the previous section. The master node separates image into seven uniform parts through a Spiral multiplication. Each part is executed by a slave node for edge detection based on Gaussain theory. Communications among the nodes including the master node and the seven slave nodes for choosing scale t for Gaussian convolution are necessary and are performed through message passing in the MultiRing.

3.1. Assignment of Image Parts to Slave Nodes

Figure 5 shows the logical connections of the MultiRing for the assignment of image parts to slave nodes. It can be seen easily that after maximum three node-to-node hops, each of the seven slave nodes will receive an image part to execute. As each part looks similar with any other part, it is expected that all slave nodes will complete their execution for initial edge detection at almost the same time. This greatly increases the global edge detection speed.

3.2. Message Passing for Gaussian Scales

The master node sets an initial Gaussian scale t_0 and sends it with the image parts to the slave nodes for initial edge detection [1]. At the end of edge detection computation at each slave, the node has also decreased the original Gaussian scale to its final value at the node. Let us assume the final Gaussian scales at P_1, P_2, ..., P_7 by t_1, t_2, ..., t_7 respectively. When every slave node has done the edge detection for the image part allocated, it needs to let other slave nodes know its final Gaussian scale. When each slave node has received the final Gaussian scales from all other slave nodes, it compares the scales received with its own final scale. The smallest scale among the seven final scales created in the seven slave nodes is selected in each node as the global final Gaussian scale. The global scale is used by each slave node for its final round edge detection algorithm. The following procedure shows the steps for passing Gaussian scales from one slave node to another on a uni-directional MultiRing.

1) Each slave node sends its final Gaussian scale to its next node using 1-ring-of-8-nodes configuration (see Figure 6) in counter-clockwise direction. Meanwhile, the master node sends P_1 the message (denoted by *msg*) about whether there will be more image frames to come. This message is for slave nodes to know when the connections between nodes can be terminated.

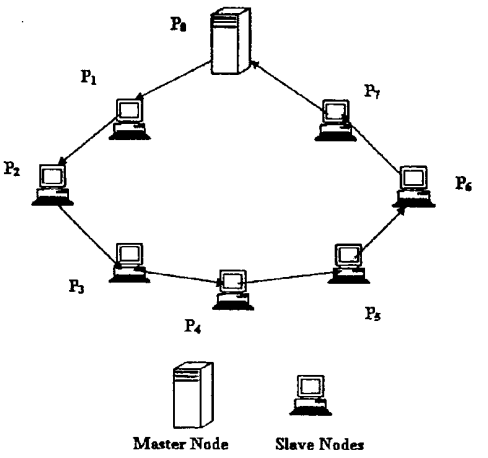

Figure 6. A Uni-directional ring of 8 nodes

2) Each node including master node sends the final scales of all slave nodes that it has, and msg if it has, to its next node using 2-rings-of-4-nodes configuration (see Figure 7) in counter-clockwise direction.

3) Each node including master node sends the final scales of all slave nodes that it has, and msg if it has, to its next node using 4-rings-of-2-nodes configuration (see Figure 8) in counter-clockwise direction. Note that each ring has only two nodes. So each node in a ring of two nodes is the next node of the other node in the ring in counter-clockwise direction.

After these three steps, each node including the master node has the final scales of all slave nodes and the *msg* from the master node. See Table 2 for the final scales and/or the *msg* every node has after each above step. The master node uses the final scales collected as a reference to determine the initial scale for next image.

3.3. Final Scale for Edge Detection on a Slave Node

Each slave node calculates the minimum final scale from the final scales obtained. Let us denote the minimum scale t_{final}, i.e.,

$$t_{final} = \min\{t_i \mid i = 1,2,\ldots,7\}.$$

The final procedure for edge detection on each slave node for an image is shown as follows.

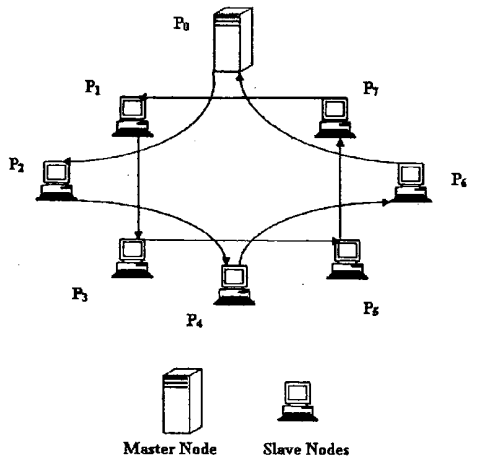

Figure 7. Two uni-directional rings of 4 nodes

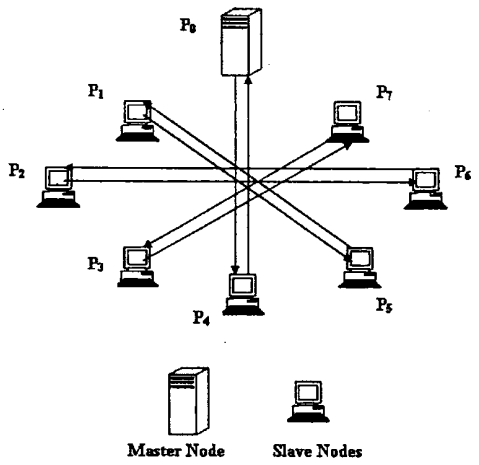

Figure 8. Four uni-directional rings of 2 nodes

1) Each slave node works on edge detection for the image part it has using the Gaussian scale t_{final}.

2) When the slave node finishes its edge detection, it sends the edge detection results to the master node following the procedures to be presented in Subsection 3.4.

3) The slave node terminates all its connections to all its previous nodes on the MultiRing if *msg* saying 'No more images to be detected'; or

4) The slave node waits for an image part for the next image to be assigned to it if *msg* saying 'There will be more image to come.'

msg can be implemented using a single bit: 1 for more image frames and 0 for no more frames, or implemented using multiple bits: a positive integer for the frame count of the coming image (next image) and 0 for no more frames (or images).

3.4. Report Results to Master Node

After each node completes the edge detection for the image part assigned to its node with Gaussian scale t_{final}, it reports the detection results (edge map) for the image part to the master node through the MultiRing. The details for all slave nodes to report the detection results are presented as follows.

1) Nodes P_1, P_3, P_5 and P_7, send their results to their next nodes (they are P_2, P_4, P_6 and P_0 respectively) using 1-ring-of-8-nodes configuration (see Figure 9). Then P_1, P_3, P_5 and P_7 remove the edge maps that they have, and terminate their connections to their previous nodes in the 8-node MultiRing if *msg*=0.

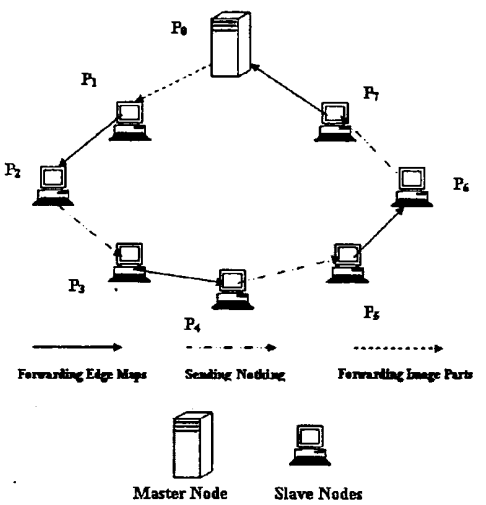

Figure 9. A 1-ring-of-8-nodes configuration for reporting edge detection results and forwarding consecutive image parts

2) Node P_2 sends its edge detection results (edge map) together the results received from P_1 to P_4 using 2-rings-of-4-nodes configuration (see Figure

10). Meanwhile, P_6 sends its edge detection results (edge map) together the results received from P_5 to P_0 using the same 2-rings-of-4-nodes configuration (see Figure 10). Then P_2 and P6 remove the edge maps that they have, and terminate their connections to their previous nodes in the 8-node MultiRing if msg=0.

3) Nodes P_4 sends its edge detection results (edge map) together the results (edge maps for the parts assigned to P_1, P_2 and P_3) received from P_2 and P_3 to P_0 using 4-rings-of-2-nodes configuration (see Figure 11). Then P_4 removes the edge maps that it has, and P_4 and P_0 terminate their connections to their previous nodes in the 8-node MultiRing if msg=0.

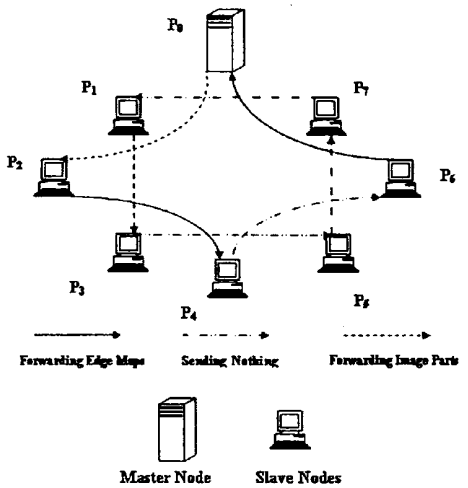

Figure 10. A 2-rings-of-4-nodes configuration for reporting edge detection results and forwarding consecutive image parts

After these three steps, the master node will have received all edge detection results from all slave nodes. See Table 3 for the edge maps that every node involved in the above three steps has after each above step by assuming that the results (edge maps) produced by P_1, P_2,..., and P_7 are e_1, e_2, ..., and e_7 respectively.

3.5. Assignments of Image Parts of Consecutive Image

If $msg \neq 0$, then there will be more images required for edge detection. The procedure for assignments of the image parts for the next image can overlap with the procedure for reporting the detection results of the current image. That is, while in step 1 of reporting results, the MultiRing is also in step 1 for the assignments of image parts of the next image as shown in Figure 9. Steps 2 and 3 of reporting results of current image are also overlapped with steps 2 and 3 of the assignments of the next image as shown in Figure 10 and Figure 11.

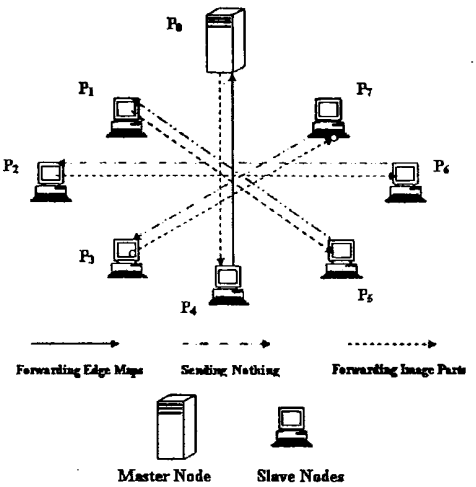

Figure 11. A 4-rings-of-2-nodes for reporting edge detection results and forwarding consecutive image parts

The advantages of overlapping the reporting procedure with the assignment procedure are the following.

1) More efficient use of MultiRing configuration. As each step of the two procedures is using the same MultiRing configuration such as 1-ring-of-8-nodes configuration and sending a block of data to next node in a ring, the time needed for each node for proceeding the step is almost the same.

2) More efficient use of memory spaces for storing image data. At each overlapped step, any node who is receiving new image data is sending some other image data, or has already sent some other image data.

4. Parallel Algorithm for Edge Detection

This work is done using master-slave model. When all nodes that constitute the MultiRing have arrived and are connected to the same MultiRing switch, routing

information and neighbouring information are collected, and hence a routing table and a neighbouring table on each node are created. The switch then periodically configures the MultiRing. Message is sent and received through a configured ring. Source and destination addresses must be included in the message header. The destination address is used by sending node to find the route, and by receiving node to determine whether to keep or forward the message. The communication between any two nodes uses *socket* mechanism in C computer language. It is uni-directional communication. We now summarize the parallel algorithm for edge detection on the MultiRing as follows.

1) For each image that the master node captured, the master node uniformly separates it into seven equal parts.

2) The master sends each image part to a slave node.

3) Each slave node pre-processes the edge detection for the image part using Gaussian theory. A final Gaussian scale on the node is obtained.

4) Each slave node sends the final Gaussian scale to all the other nodes including the master node, while the master node sends a special message, *msg*, to all the slave nodes..

5) Each slave node calculates the minimum of all final scales and processes the edge detection using the minimum Gaussian scale.

6) Each slave node sends its detection results to the master. Meanwhile, the master node sends the image parts of a new image to all slave nodes if there are more images to process. Then go to Step 3 for next iteration for the new image, and Step 7 below for final detection results of the current image. All slave nodes close its connections to their previous nodes of all ring configurations of the MultiRing if there are no more images to process.

7) The master creates final edge detection results based on the results collected from the slave nodes.

5. Conclusions

In this paper, we have presented a parallel and distributed algorithm for image edge detection on Spiral Architecture using a uni-directional MultiRing. We conclude the following.

1) The traffic from and to the master node has been greatly reduced compared with the work done in [4].

2) If a separate child process is created on each node for message passing between any two adjacent nodes of a ring configuration, there are maximum 6 child processes needed on each node. Hence the overall process time is less than the one in [4].

3) If we ignore the time needed for sockets connections between nodes, the time needed for transfer image data from the master node to all slave nodes is exact the same as that using the algorithm in [4]. Hence, the overall time actually needed for the image data transmission is much less.

4) In our algorithm, the minimum Gaussian scale is calculated at the same time on each node. This eliminates the time required for the master node to broadcast the minimum scale to all slave nodes.

6. References

[1] Xiangjian He, Tom Hintz, and Ury Szewcow, *Replicated shared object model for edge detection with Spiral Architecture*, Lecture Notes in Computer Science, Vol.1388, 1998, pp.252-260.

[2] H. R. Arabnia and M. A. A. Oliver, *A Transputer Network for Fast Operations on Digitised Images*, International Journal of Eurographics Association (Computer Graphics Forum), Vol. 8, No. 1, 1987, pp.3-12.

[3] P. Sheridan, Spiral architecture for machine vision, PhD thesis, University of Technology, Sydney, 1996.

[4] Qiang Wu, Xiangjian He and Tom Hintz, *Distributed Image Processing on Spiral Architecture*, Proceedings of the 5th International Conference on Algorithm and Architectures for Parallel Processing (IEEE), Beijing, October 2002, pp.84-91.