01 Jan 1998

Systolic Algorithm for Processing RLE Images

Hao Feng

Fikret Erçal
Missouri University of Science and Technology, ercal@mst.edu

Filiz Bunyak
Missouri University of Science and Technology

Recommended Citation

H. Feng et al., "Systolic Algorithm for Processing RLE Images," *Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation, 1998*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1998.
The definitive version is available at https://doi.org/10.1109/IAI.1998.666872

SYSTOLIC ALGORITHM FOR PROCESSING RLE IMAGES

Hao Feng[†], *Fikret Ercal*[‡], *and Filiz Bunyak*[§]

Computer Science Department and Intelligent Systems Center,
University of Missouri – Rolla

ABSTRACT

Image difference operation is commonly used in on-line automated printed circuit board (PCB) inspection systems as well as many other image processing applications. In this paper, we describe a new systolic algorithm and its system architecture which computes image differences in run-length encoded (RLE) format. The efficiency of this operation greatly affects the overall performance of the inspection system. It is shown that, for images with a high similarity measure, time complexity of the systolic algorithm is a small constant. A formal proof of correctness for the algorithm is also given in the paper.

I. INTRODUCTION

On-line automatic inspection of PCBs requires acquisition and processing of gigabytes of binary image data in a matter of few seconds. To meet the demands for high speed and accuracy, we have designed a fast modular RLE-based PCB inspection system. The system[1] uses run-length encoding (RLE) for storage and operations and an inspection scheme which exploits the availability of an artwork for comparison purposes. It is suitable for parallel processing and consists of four steps: (i) segmentation of artwork and feature extraction, (ii) image acquisition, (iii) inspection of blank areas, and (iv) inspection of trace areas. Steps (ii), (iii) and (iv) are time-critical online operations, therefore, they should be performed with high efficiency. Image difference is the most frequently used operation in steps (iii) and (iv), and hence, overall system performance critically depends on the speed of this operation. In this study, a parallel systolic algorithm is developed to compute the difference between the corresponding rows of two images which are represented in compressed form using RLE. The system performs its operations in compressed mode and can be effectively used to find the differences between two RLE images.

Systolic systems use cellular iterative computations and perform global tasks through exchange of local data in pipelined fashion[2]. Since most of the image processing operations exhibit high local dependencies among data elements, systolic machines are widely used in image processing applications such as filtering[3], thinning[4], convolution[5], etc. The straightforward parallel method for computing these iterative-convergent operators is through a globally synchronous updating mode: all variables are updated at once, based on the values calculated during the previous step, before another iteration step is initiated. Most systolic image processing algorithms proposed so far are based on operations on pixel data. Since systolic machines are designed to exploit spatial information and most of the spatial locality information is lost in compressed domain, it is extremely difficult to design systolic algorithms which operate on compressed image data. Fortunately, some compression techniques such as RLE preserve part of the information pertaining to spatial locality. Hence, in this paper, we are able to propose a systolic system that finds the difference between two binary images represented in RLE. Work is underway to extend our design to other image operations in compressed domain.

Figure 1 demonstrates the operation for finding

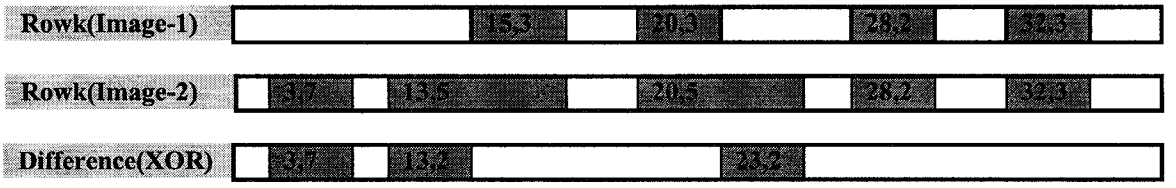


Figure 1: RLE-based image difference

[†] E-mail: feng@umr.edu WWW: http://www.umr.edu/~feng
[‡] E-mail: ercal@umr.edu WWW: http://www.cs.umr.edu/~ercal
[§] E-mail: bunyak@umr.edu WWW: http://www.umr.edu/~bunyak

the difference (XOR operation) between the corresponding rows of two images represented in RLE. Pseudo-code for the sequential algorithm is given below:

Algorithm: RLE-based Image Difference(XOR)

Repeat Until *end of row* is reached
{ **If** (end of current runs in both of the rows are equal)
Then Move to next runs in both of the rows; don't change *Difference;*
(as both of the rows switch color at the same point their ***xor*** *does not change since (0 xor 1) == (1 xor 0) and (1 xor 1) == (0 xor 0) result color remains the same)*
Else Add a new run to *Difference,*
(Result switches color)
with start point: last end point
and end point: smaller of the end points of current runs
Move to next run in the row having the smallest end point among the current runs
}

II. RLE-BASED SYSTOLIC ALGORITHM FOR IMAGE DIFFERENCE

We assume that each row of a given image is expressed as a vector of run elements (tuples) as shown in Figure 1. In each tuple, first element is the starting position and the second element is the length of the run. For efficiency reasons, only the runs corresponding to the foreground pixels are used as input for the systolic computation. Let's assume that the maximum number of runs in any row is bounded by k. Then the result of image difference for two rows can have at most 2k tuples. Therefore, 2k systolic cells are needed in the system to be able to store the result data as shown in Figure 2. When the computation is completed, each systolic cell will hold either none or one tuple.

For each cell, I_1 and I_2 are used to input the original data (tuples from rows of image1 and image2) (see Figure 2(a)). Iin/Iout are used to pass the contents of Register-Big between neighboring cells. **Out** is for storing the final result back to memory. C signals the end of the process. **Termination is reached when none of the cells holds a tuple in Reg-Big.** Computation starts

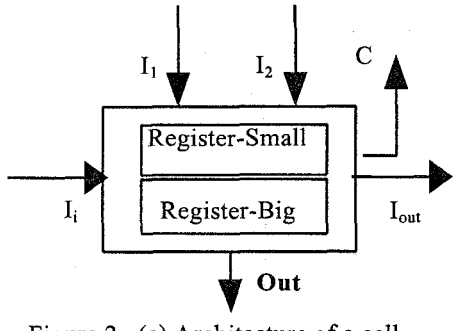
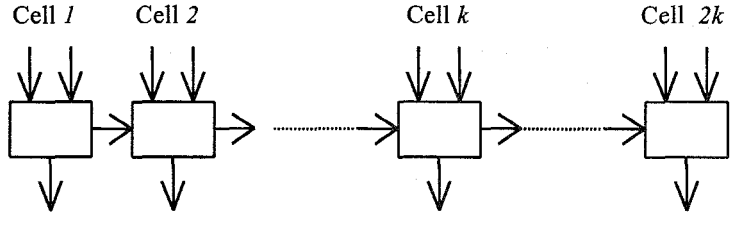


Figure 2. (a) Architecture of a cell



(b) 2k cells cascaded together to form the systolic system

Image-1	(15,3)	(20,3)	(28,2)	(32,3)					
Image-2	(3,7)	(13,5)	(20,5)	(28,2)	(32,3)				
		Cell1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell10
step1									
	Reg-Small	(15,3)	(20,3)	(28,2)	(32,3)	(32,3)*	//if there is only one input structure, put it into Register-Small during initialization.		
	Reg-Big	(3,7)	(13,5)	(20,5)	(28,2)				
	Reg-Small	(3,7)	(13,5)	(20,5)	(28,2)	(32,3)			
	Reg-Big	(15,3)	(20,3)	(28,2)	(32,3)				
step2									
	Reg-Small	(3,7)	(13,5)	(20,5)	(28,2)	(32,3)			
	Reg-Big		(15,3)	(20,3)	(28,2)	(32,3)			
	Reg-Small	(3,7)	(13,2)						
	Reg-Big			(23,2)					
Step3									
	Reg-Small	(3,7)	(13,5)	(20,5)	(28,2)	(32,3)			
	Reg-Big		(15,3)	(20,3)	(28,2)	(32,3)			
	Reg-Small	(3,7)	(13,2)		(23,2)				
	Reg-Big								

result (3,7) (13,2) (23,2), Final result after post-process (merging the adjacent runs) is (3,7)(13,2)(23,2).
(First two rows mean registers of each cell containing before cell computation, while last two rows mean registers of each cell containing after cell computation before tuple-move.)

Figure 3. Systolic computation steps for the image given in Fig.1(k=5)

by inputting the run tuples in sorted order into the cells through input ports I_1 and I_2 as shown in the example in Figure 3. Each cell executes the following algorithm repeatedly until the termination condition is reached.

Cell Algorithm (LXOR and Data Move):

Cell algorithm uses two operations: **LXOR** and **Move** which are defined in the next section. Depending on the application, a **post-processing** step may be needed to merge the result tuples that are adjacent with each other.

Repeat until (termination condition is not reached)
If there is no tuple or only one tuple in the cell
no computation is necssary.
Else { // LXOR the tuples
Switch {
Case 1: (adjacent) The end position of one tuple is the beginning position of another tuple, leave them as they were but reallocate their register position according to their beginning position. Ex. (20,3) and (15,5) will be put into Reg-Big and Reg-Small respectively.
Case 2: Two tuples have the same start position and different length, the XOR result will be put into Reg-Big.
Or Two tuples have the same end position and different length, the result will be put into Reg-Small.
Case 3: (Other) XOR two tuples and the resulted two tuples will be put into two registers based on their starting position.
}

Move the tuple in Reg-Big (if exits) into right neighbor. If the neighbor is empty, put the tuple into Reg-Small of the neighbor, else put it into Reg-Big.

III. PROOF OF CORRECTNESS

The correctness of the algorithm can be shown by proving that XOR operation is applied once and only once to any two overlapping tuples in the given rows. That is, any two overlapping tuples meet in a cell once and only once at some step of the systolic execution and the result of the operation will be the same as that obtained in the sequential algorithm (XOR operation).

In order to prove this, we first give a formal definition of the operations in a cell between two data movements. Later, we show two important properties of the operation that are essential for our proof. The actual proof follows from the definition of the cell operation and the given algorithm.

1) Definitions For Relative Tuple Positions:

Given two tuples, T1 (begin1, length1) and T2 (begin2, length2), they may either be disjoint or overlapped. If the two tuples are adjacent (i.e. their end points meet), by definition, they will be considered as overlapped.

I) Disjoint tuples

If T1 and T2 are disjoint, T1 may be either "less than" (T1 < T2) or "greater than" T2 (T1 > T2). Formally,

a) T1 < T2, if (begin1+length1) < begin2
b) T1 > T2, if (begin2+length2) < begin1

II) Overlapping tuples

We use the notation (T1 $\Leftrightarrow$ T2) to denote that T1 and T2 have some overlap or they are adjacent with each other. Using formal terms,

c) T1 $\Leftrightarrow$ T2,
if (begin1 $\leq$ begin2 $\leq$ begin1+length1), OR
if (begin2 $\leq$ begin1 $\leq$ begin2+length2).

Based on the relative positions of T1 and T2, we define the following special relationships under this category:

c1) T1 $\Leftarrow$ (Left Clearly Contained in, LCC) T2,
when (begin1= begin2) and (length1 < length2).
c2) T1 $\Rightarrow$ (Right Clearly Contained in, RCC) T2,
when (begin1 < begin2) and
(begin1+ lengh1 = begin2+length2) .
(Note that T1 $\Leftarrow$ T2 and T2 $\Rightarrow$ T1 are different cases)
c3) T1 $\leftarrow$ (left adjacent with) T2,
when (begin1+ length1 = begin2).
c4) T1 $\rightarrow$ (right adjacent with)T2,
when (begin2+ length2 = begin1).

2) Definition of LXOR Operation:

Computation of the image difference between two rows is based on the LXOR operation performed synchronously by each cell of the systolic array shown in Figure 2. LXOR (less-condition XOR) operation is performed on two input tuples T1 and T2. As a result of this operation, two tuples are produced; Ts (small tuple) and Tb (big tuple), either or both of which may be null. By definition, if both Ts and Tb exist, then (Ts < Tb) or (Ts $\leftarrow$ Tb). Operation LXOR can be formally defined as follows:

T1 **LXOR** T2 = (Ts, Tb) where
Case (T1 $\leftarrow$ T2): Ts = T1, Tb = T2,
Case (T1 $\Leftarrow$ T2):

Ts = null, Tb = (begin1+length1, length2-length1)

Case (T1 ⇒ T2):

Ts = (begin1, length1-length2), Tb = null.

Other: Standard XOR is applied to two input tuples. Resulting tuples will have a "<" relationship between them, smaller tuple is called Ts, bigger one Tb.

Like standard XOR, LXOR has the following two properties:

a) LXOR is commutative; T1 LXOR T2 = T2 LXOR T1.

b) T1 LXOR (T2 ∪ T3) = (T1 LXOR T2) LXOR T3, where T2 and T3 are non-overlapped.

Property (a) implies that the result of the operation does not depend on the physical location of the two tuples which are operands to the LXOR. In other words, we need not pay attention to which registers the two tuples in a cell are stored in.

Property (b) is interesting in that, for a tuple T1, if it is overlapped with an aggregation of non-overlapped (two or more) tuples, the result of LXOR operation between T1 and the aggregation can be obtained by operating T1 with the tuples in the sequence one at a time. Obviously, in our systolic implementation, the result of this operation will possibly be stored in multiple cells of the systolic system.

In summary, between every move, each cell repeatedly performs LXOR operation on the tuples stored in Reg-Big and Reg-Small and then moves data in Reg-Big to the right neighbor.

3) Correctness Proof:

Let's first make proof for general case where there are two input tuples in each cell. For any two neighboring cells C_A and C_B, assume that the tuples in them are labeled as T_{a1}, T_{a2}, T_{b1} and T_{b2}, respectively. Initially, (1) holds since input tuples are ordered. During the process, this order is preserved since T_1 is always replaced by previous T_s while T_2 replaced by T_b of its left neighbor.

$$T_{a1} < T_{b1}, \quad T_{a2} < T_{b2} \tag{1}$$

The results of LXOR operation between them are T_{as}, T_{ab}, T_{bs} and T_{bb} respectively, where T_{xs} stands for the small tuple in cell C_x, and T_{xb} stands for the big tuple in C_x. None of the four tuples is null in general.

Right after the LXOR operation in the cell and before the data move, we will have

$$T_{as} < T_{ab} \text{ or } T_{as} \leftarrow T_{ab}\ ;$$
$$T_{bs} < T_{bb} \text{ or } T_{bs} \leftarrow T_{bb} \tag{2}$$

From condition (1), we can see

$$T_{as} < T_{bs},\ T_{ab} < T_{bb} \tag{3}$$

From (2) and (3), we have

$$T_{as} < T_{bb}. \tag{4}$$

From (4), we conclude that T_{as} and T_{bb} are disjoint, while it is possible that T_{ab} and T_{bs} may overlap with each other. So, moving T_{ab} to C_B and T_{bb} away from C_B guarantees that two possibly overlapped tuples will not move away from each other and eventually meet later. Since this process is repeated till there is no tuple left to move in Reg-Big of any cell, any two overlapping tuples will eventually meet some time during the process. Also note from (1) that, at termination, the tuples in Reg-Small (disjoint or adjacent) will be in (sorted) order.

We can see that, at any step in the algorithm, (4) always holds before any data move. Also, from the property 2) of LXOR operation, we can see that the final result after all the data moves and after the post-processing step for merging the adjacent tuples, is equivalent to the result obtained by the *algorithm RLE-based Image Difference(XOR)*. This concludes our proof for the general case.

It is easy to see that the proof above still holds for the case that a cell might only have one tuple or no tuples between data moves.

IV. COMPLEXITY ANALYSIS

For the sequential algorithm, the time complexity of the operation is O(2k) where k is the maximum number of tuples in one row. For systolic algorithm, originally there will be k cells with two tuples each, and at each step, data will move to right one position at a time. At the end, the tuples of the result vector will be stored in the cells; one tuple per cell. The length of a result vector may range from 0 to 2k. Therefore, in the best case, the computation will finish in 1 step, and it will take k steps in the worst case. On the average, the running time is expected to be k/2. The time complexity is highly dependent on the similarity of two input images. Indeed, it is proportional with the amount and the distance of run-length differences in two image rows. Since, in an application such as PCB inspection, the image rows are so much alike, the time complexity is expected to be almost constant.

V. CONCLUSIONS

In this paper, we have presented an efficient systolic algorithm to find the differences between two run-length encoded images. The algorithm terminates in k/2 steps on the average, where k is the number of runs in an image row. The time complexity is highly dependent on the similarity of two input images. Since, in an application such as PCB inspection, the image rows are so much alike, the time complexity is expected to be almost constant.

Currently, we are in the process of improving our design such that i) the cells numbered between k and 2k can be utilized in the beginning of the computation and ii) the termination condition may be reached even if some cells have two tuples but that all tuples are disjoint and in order. We are also gathering statistical data on the distribution of image rows that are being compared. This kind of analysis will help us to obtain a more accurate average running time for the systolic computation. Furthermore, we plan to develop similar systolic designs for other commonly used image operations (e.g. dilation, erosion, convolution, etc.) in compressed domain.

References

1. F. Ercal, F. Bunyak, H. Feng, and L. Zheng, *A fast modular RLE-Based inspection scheme for PCBs*, Proc. of SPIE - Architectures, Networks, and Intelligent Systems for Manufacturing Integration, Pittsburg, Oct. 1997, Vol. 3203, pp. 49-59.

2. Vipin Kumar, Ananth Grama, A. Gupta, and G. Karypis, *Parallel Computing Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Inc.

3. F. Robin, M.Renaudin, G. Privat and N. Van Den Bossche, *Functionally asynchronous array processor for morphological filtering of greyscale images*, IEE Pro-computer Digit Tech, Vol 143, No.5.

4. N. Ranganathan and K. B. Doreswamy, *A Systolic Algorithm and Architecture for Image Thinning*, Proc. Of Fifth Great Lakes Symoisium on VLSI, Bufflo, NY, Mar. 1995

5. N. K. Ratha, A. K. Jain and D. T. Rover, *Convolution on Splash 2*, Proc. Of IEE Symposium on FPGAs for Custom Computing Machies, Napa Valley, CA, April, 1995