筑波大学
University of Tsukuba

A Region Merging Approach for Image Segmentation On FPGA

Khac Trieu DANG BA and Tsutomu MARUYAMA

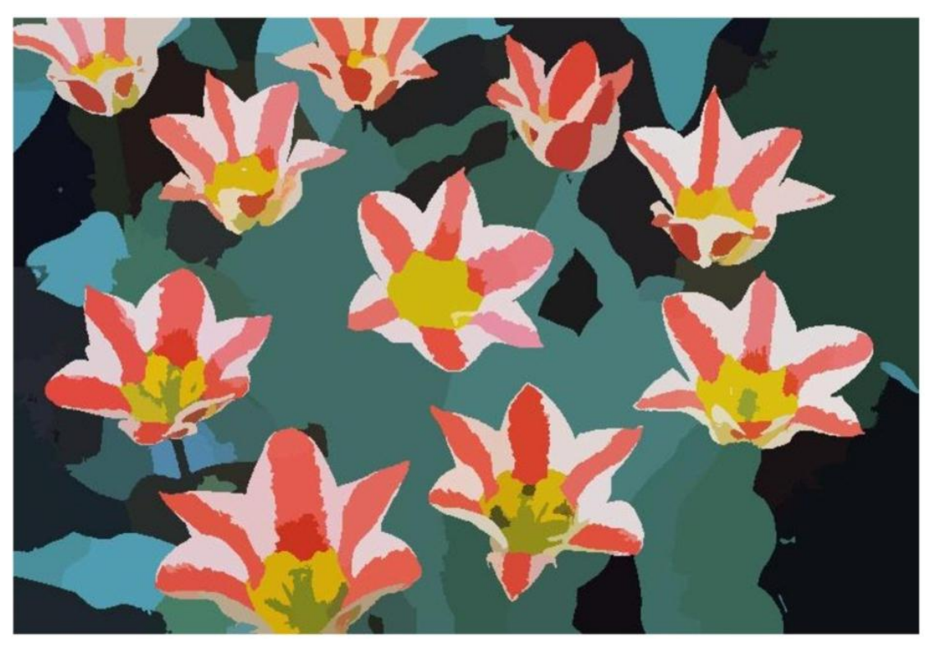
Introduction

- Image segmentation is one of the most important tasks in the image processing
- In the segmentation based on the mean-shift, k-means and so on, the image is once **over-segmented**, and then the small regions are merged
- We propose a **region merging algorithm for hardware systems**, in which the data are not managed strictly, and the redundant computation caused by this loose management is hidden by the pipelined and parallel processing.

Tulip (original)

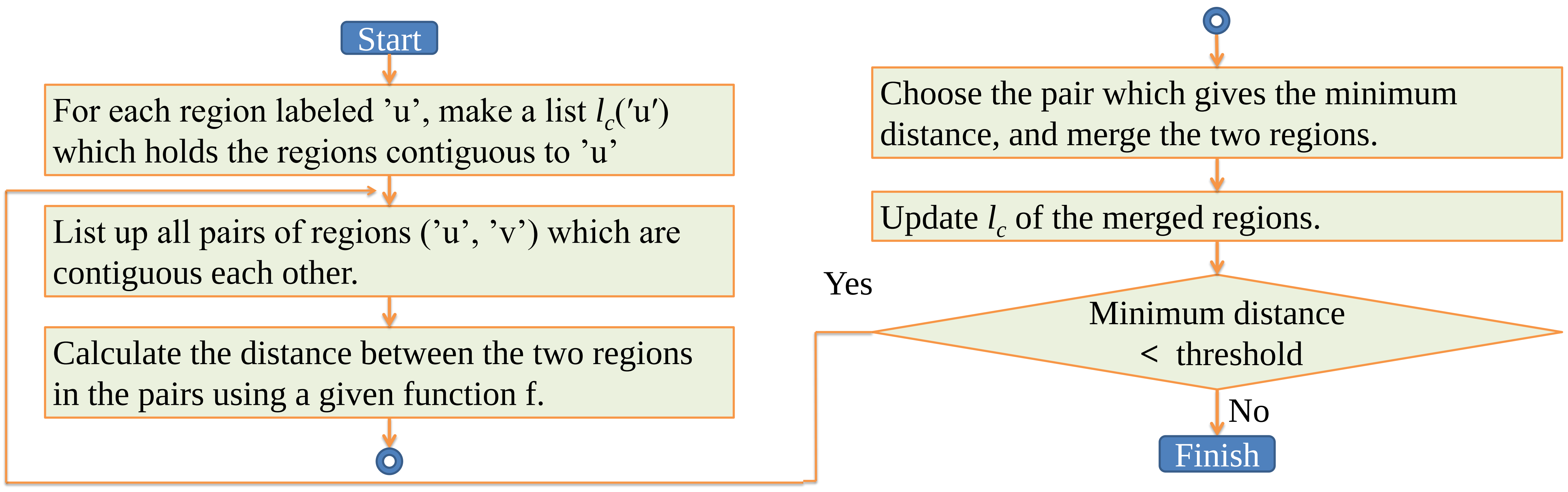
Tulip (mean-shift)

Mean-shift (over-segmentation)
&
region merging



Tulip (region-merged)

Region merging



Start

For each region labeled ’u’, make a list l_c(’u’) which holds the regions contiguous to ’u’

List up all pairs of regions (’u’, ’v’) which are contiguous each other.

Calculate the distance between the two regions in the pairs using a given function f.

Choose the pair which gives the minimum distance, and merge the two regions.

Update l_c of the merged regions.

Minimum distance < threshold

Yes

No

Finish

Our Approach

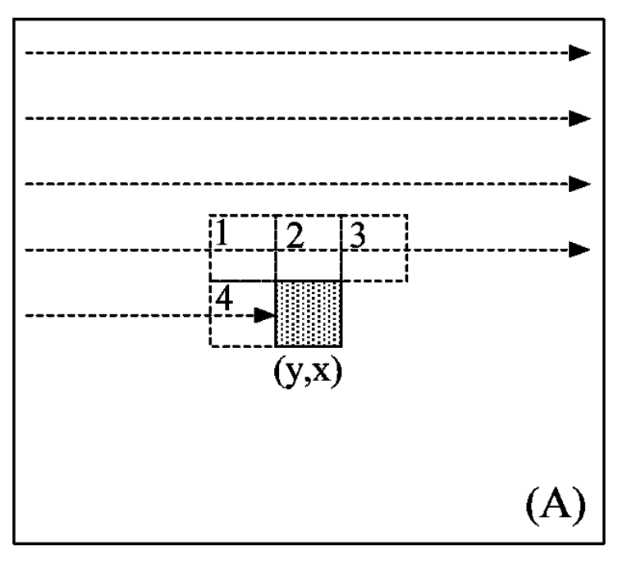
Scan image, and give unique labels to regions.
Put regions which consist of only one pixel aside.

Scan the image again, and for each region ‘u’, generate a list of its neighbor regions l_c(′u′).
Calculate the distance between two contiguous regions and sort the pairs according to their distances.

Merge two regions repeatedly according to their distances (from the closer one).
Update lc(′u′) when ’u’ and ’v’ (’u’<’v’) are merged.

Merge regions only one pixel to larger neighbor regions.

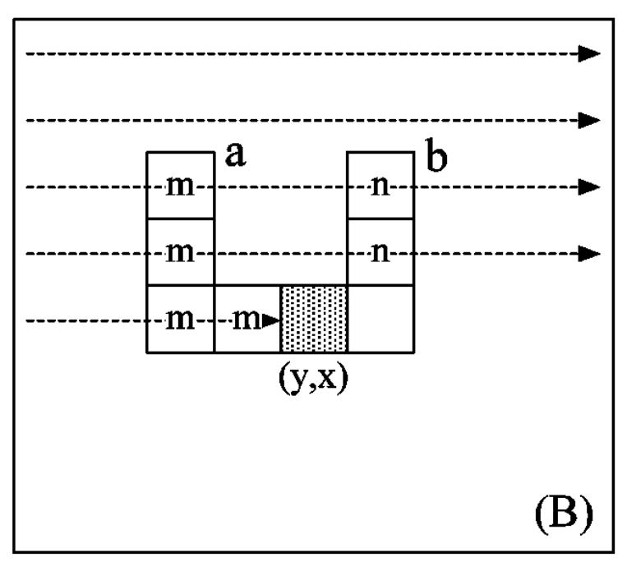
First scan

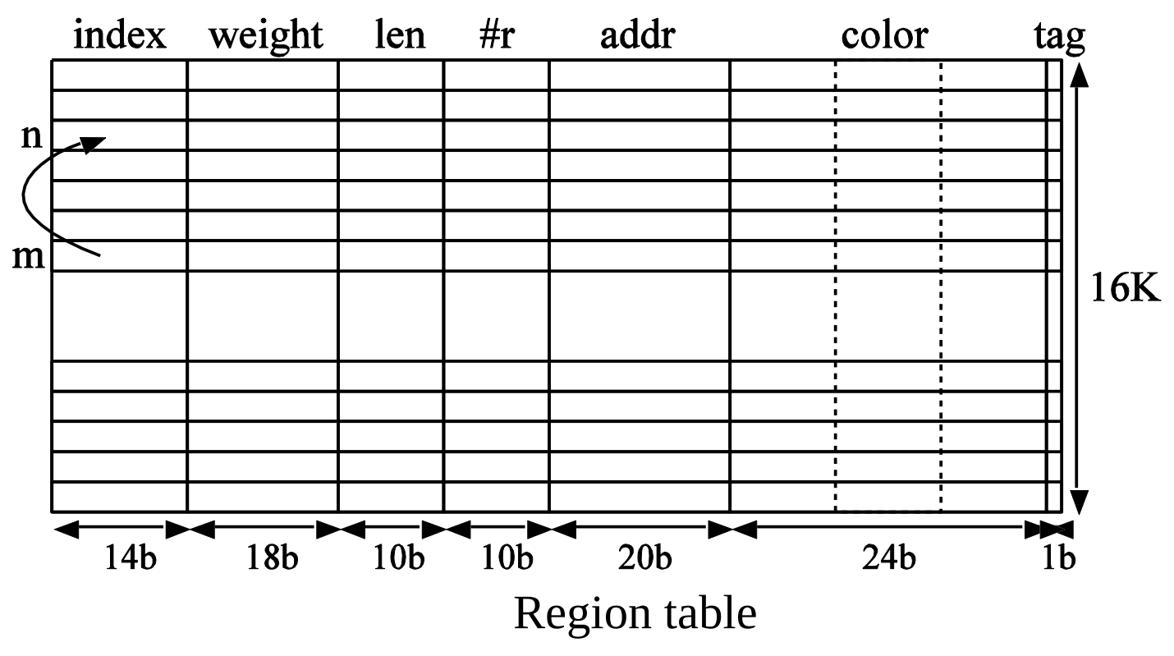


1. Unique labels are given to regions.
2. The length of l_c(u) is estimated.

(1a) the color of the pixel at (y, x) (I(y, x)) is compared with the color of its four neighbors.

- All are different => give a new label to I(y, x)
- At least, one is the same => copy its label to I(y, x)





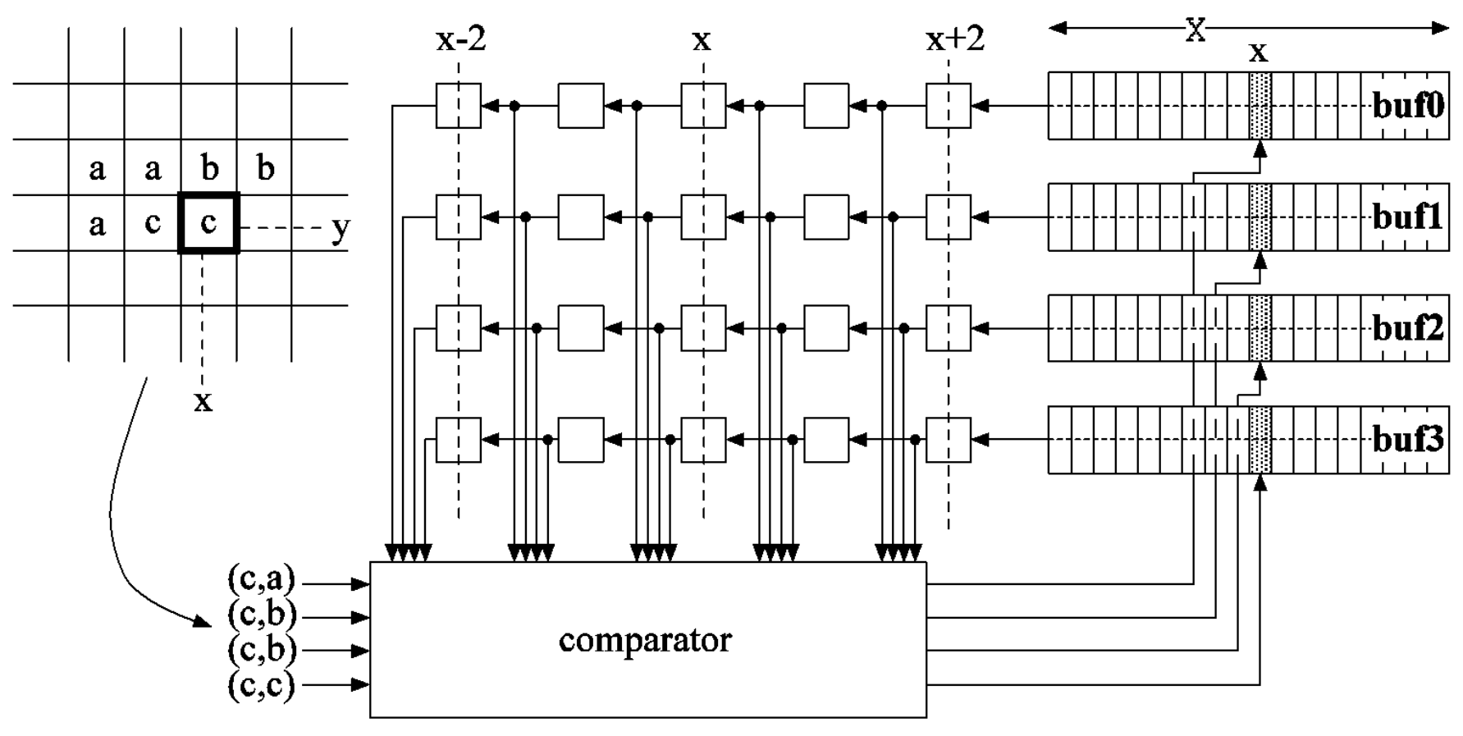
Region table

(1b) U-shaped region:

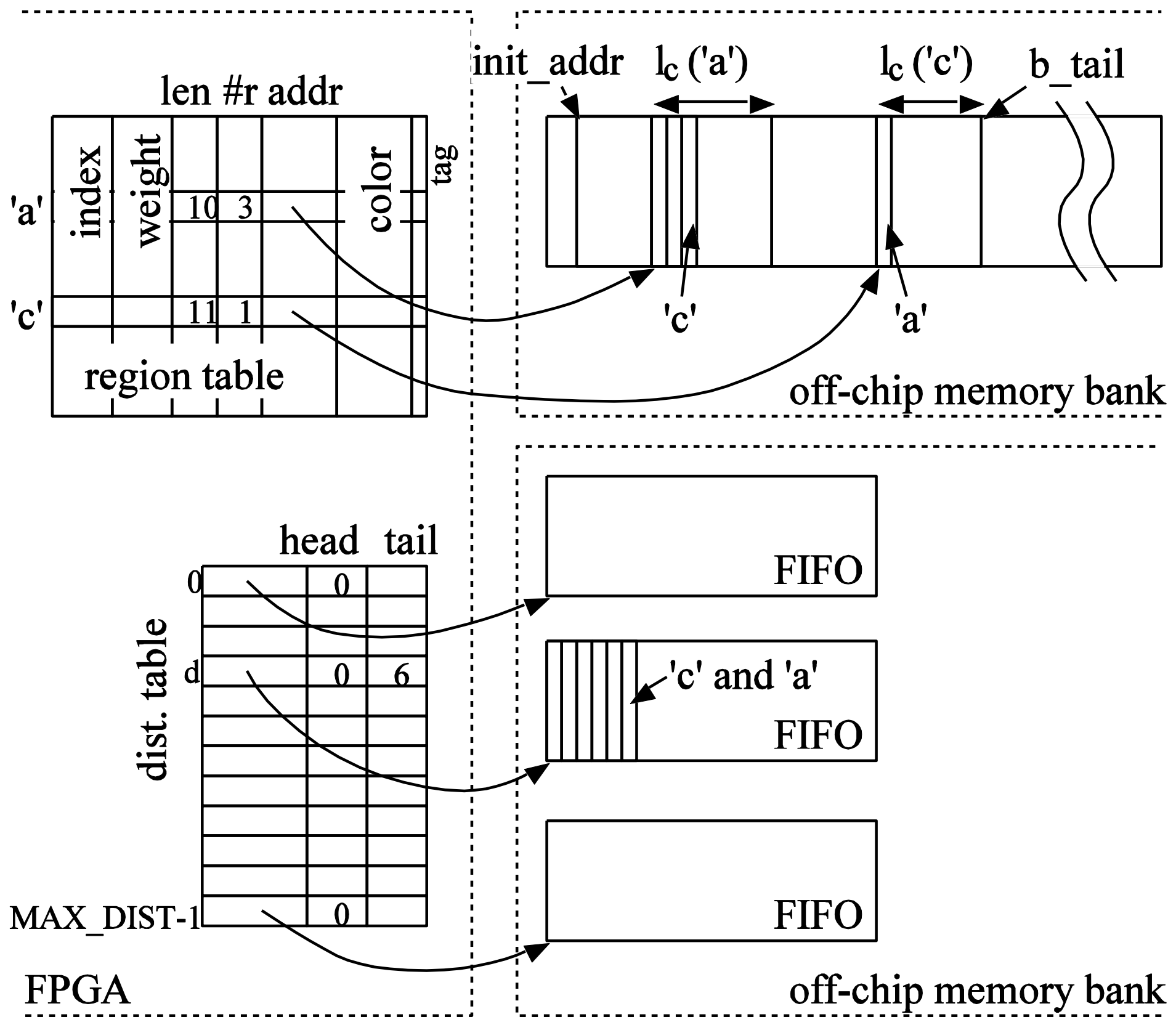
- ‘a’ and ‘b’ have the same color, but different labels (‘m’>‘n’) have been given to them.
  => set a pointer from ‘m’ to ‘n’ in the region table.
- When the first scan is finished, the index in the region table is scanned from the top, and the pointers are dereferenced.

(2a) for I(y, **x-1**) whose label is ’c’, l_c(‘c’) is counted up by 2 (’c’ is contiguous to ’a’ and ’b’).

(2b) (‘c’,’a’) and (‘c’,’b’) are held in the register array and the buffers, and are compared with new pairs for preventing redundant count-up.

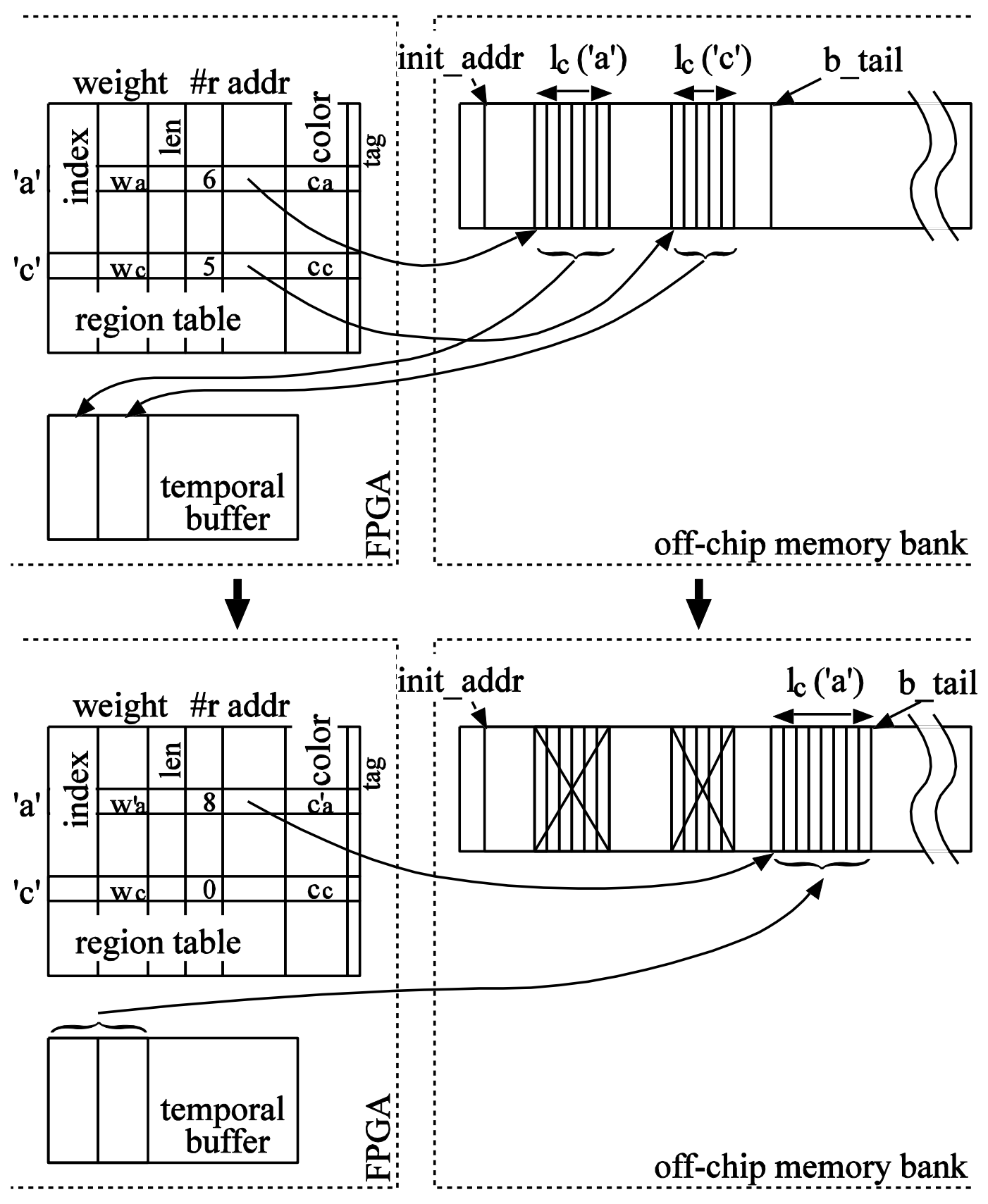


Second Scan



1. Read back the region label at (y, x), which are stored in the off-chip memory bank, and dereference it using the region table in order to obtain the true label.
2. Detect which region is contiguous to which region. Suppose that two contiguous regions (**’c’**, **’a’**) is newly detected and this is the first pair for ’**c**’ and the third for **’a’**
   - For **‘c’**: allocate a block of *len(l_c(c))+δ* to store all contiguous regions to **’c’**
     - the address of the block is *addr(c)*
     - ’a’ is store as the first contiguous region
     - *#count(c)* = 1
   - For **’a’**,
     - Store **’c’** using its *addr(a)+#count(a)* as the address,
     - Increment *#count(a)* to 3
   - At the same time, calculate the **distance d** between **’c’** and **’a’**, and if the distance is smaller than a given threshold MAX_DIST, store the pair (**’c’** and **’a’**) in the **dist.table** (in the queue pointed by **d).**

Merging region

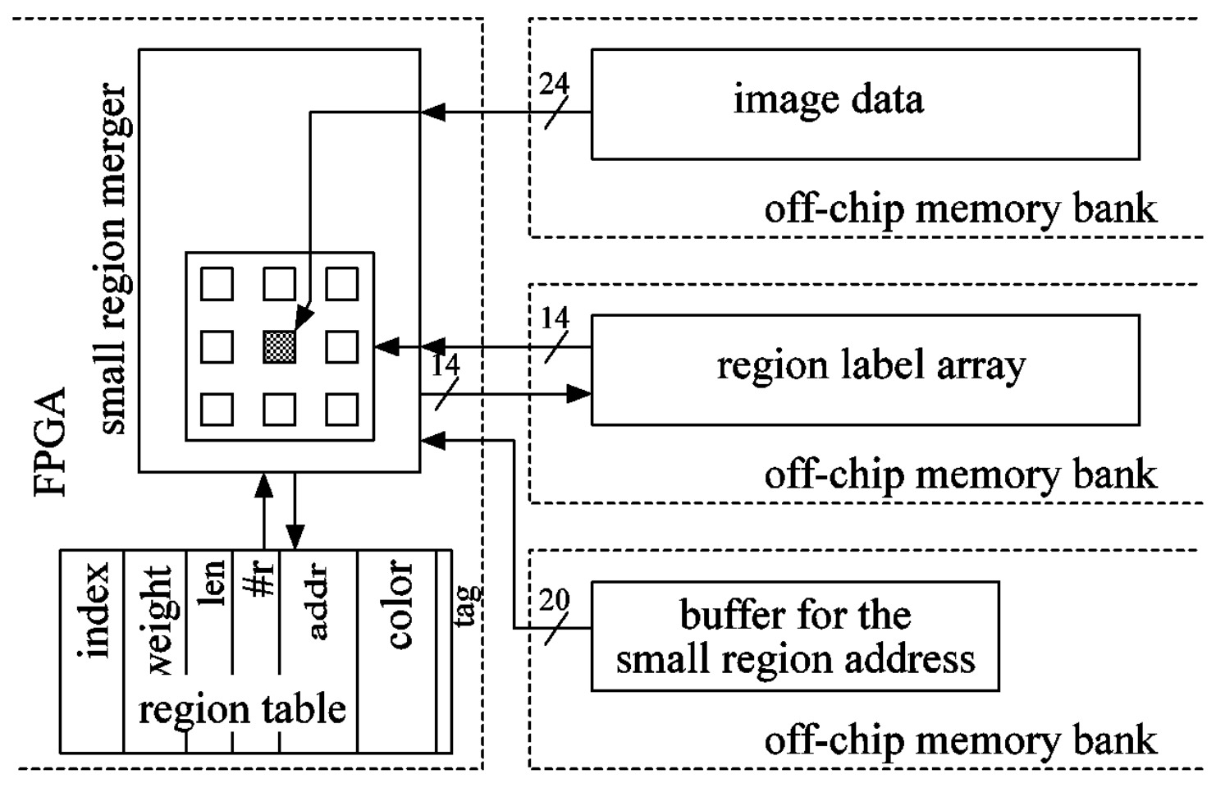


First, the **dist.table** is scanned from d = d_{min}, and a pair with the minimum distance is read out from its queue. Suppose that the pair is (’c’,’a’) and ’a’ < ’c’.

- Merge ‘c’ to ‘a’
  1. Update information of ‘a’ in the region table.
  2. Create a new l_c(‘a’) from old l_c(‘a’) and l_c(‘c’), and add new region pairs into dist.table (old ones are not deleted here).
  3. Old ones are skipped when they are read out from the **dist.table.**

Then, the **dist.table** is scanned from the current minimum distance (d_{min}), while pairs whose distance is less than MAX_DIST exist.

Region merging of the small regions



1. Read out address of a small region.
2. Read out
   a. its color,
   b. its eight neighbor’s region labels and colors
   sequentially.
3. Chose the closet neighbor and merge the small region to it.

Implementation Results

Device: Xilinx XC4VLX160

Operational Frequency: **166MHz**

Hardware Usage: 9144 LUTs, and 96 block RAMs

Experimental Results

Images	#Region			fps
	input	output	%	
Pepers	8995	175	1.95	123.3
Tulips	13810	352	2.55	82.3
Monarch	14842	300	2.02	82.7
Serrano	9194	501	5.45	101.9

- The execution time is about **7 - 9 times** faster than Intel Core2 Extreme Q6850 (3GHz)
- The important point here is that we can achieve higher performance than microprocessors for the problems which seems to be inherently sequential by relaxing the data management policy, and enabling the pipeline processing of the data.

Conclusions and future work

We have shown that we can achieve **real-time processing** by **relaxing the data management** in the algorithm and enabling the **pipeline processing** of the data.

In our current implementation, the distance between two regions is calculated using only **local relation** of the two regions.
→ use the **global relation** of the regions for obtaining better segmentation.