Run-Processing: A Coherence-oriented Processing Method and its Hardware Architecture for Real-time Video Object Segmentation

CHIA-CHEUN LIANG, JING-YING CHANG, LIANG-GEE CHEN
DSP/IC Design Lab., Department of Electrical Engineering and Graduate Institute of Electronics Engineering
No. 1, Sec. 4, Roosevelt Road, Taipei, 10617 Taiwan (R.O.C.)
TEL: +886-2-2363-5251 ext. 332, Email:{rainstraw,jychang,lgchen}@video.ee.ntu.edu.tw

Abstract

A coherence-oriented run-processing method and its hardware architecture is proposed for multiple widely-used techniques in video object segmentation algorithms including connected component labeling, binary morphology, seeded region growing and bitwise operations of binary masks. Experimenterd result shows that this run-domain processing method allows significant computation acceleration, memory reduction and bus-bandwidth reduction as well in each operation with limited resources. Via hardware acceleration, a shadow-cancellation video object segmentation algorithm achieves 151fps on D1 (720x480) video. Run-processing reduces temporary data storage by 78% and takes 13.9% computation cycles compared to conventional pixel-based methods.

Keywords:
Video Segmentation, binary morphology, connected component labeling, hardware, image processing

Introduction

Video Object Segmentation

Video Object Segmentation (VOS) has been widely used in intelligent surveillance systems, video coding standards, smart car-driving aid and remote health care systems. Through separating active objects, such as humans in a room or moving cars on a road, from the whole video frame captured from a camera, meaningful region can be extracted, which then can be further analyzed by recognition algorithms to generate useful information. For example, an intelligent surveillance system determines whether a bank robbery is happening based on analysis on captured human activities analysis. Once a robbery in progress is detected, the system automatically calls the police and sets up siren. In order to realize these fast-responding systems, hardware acceleration is utilized to provide efficient computation for real-time VOS. On the other hand, the system should be flexible since there is no general solution for VOS. Different algorithms are adopted for different situations. In consequence, an hardware architecture capable of accelerating different segmentation algorithms is vital.

Common Functions in VOS

Widely-used functions in segmentation algorithms include:

Binary Morphology
Binary morphology is extensively used to refine features such as noisy foreground masks generated from background subtraction and frame-difference[3].

Connect Component Labeling (CCL)
CCL is one of the most fundamental processes before further recognition process is applied. In addition, it is the prior step of area-threshold utilized by several segmentation algorithms[21].

Full-frame Bitwise Operations
Bitwise operation between binary masks is used in feature processing and saliency test[21].

Seeded Region Growing (SRG)
SRG extends local features from seed position to corresponding region and has been widely applied by region growing approaches[22].

Other Functions
Other common functions include boundary extraction, bounding box extraction, and random searching of labeled object from a labeled image.

Related Works

Because binary morphology and CCL each requires large computation load, numerous hardware implementation has been proposed. Diamantaras and Kung, and Malamas et al. presented fast systolic architectures performing 1-D binary erosion or dilation[1-2]. Parallel processing extends the operation to 2-D, but the complexity of each 1-D module is proportional to the structure element (SE) width. In Consequence, it is not adequate for applications which require large SEs. Chien et al. used a partial-result-reuse architecture and delay-line buffers of size equal to video resolution width[3]. Though the delay-line approach achieved a parallel implementation of programmable morphology, the size of delay-line linearly increases with video resolution width, leading to impractically large area consumption in order to support modern high-resolution video. Hedberg et al. proposed a stall-free architecture to reduce memory requirement[4]. Despite reducing in
memory actualize high-resolution binary morphology, the computation time is still limited by video resolution due to its pixel-by-pixel processing nature, i.e., computation cycle proportionally rises with video resolution. For example, performing binary morphology on a D1 (720x480) video requires about 345,600 cycles, even though the image is actually empty.

Various hardware designs for CCL has been reported. A classical approach is proposed by Rosenfeld and Pfafitz[7], it employed a two-path algorithm and a global equivalence table, which is later improved by Lumia et al. by storing equivalence table of a single-row[21]. Chien et al. introduced a sub-word concept which processes four pixels at a time to fully-utilize bus-bandwidth[10]. Similar multi-pixel solution is also found in [8]. A block of 2x2 pixels is processed simultaneously and corresponding type of possible equivalences are analyzed to ensure its memory access reduction scheme. However, increasing the size of block implies more complicate connectivity conditions, preventing the solution from higher-resolution extension. Moreover, pixel-by-pixel scan restrains further acceleration.

Considering segmentation as a prior operation of intelligent video processing system, a better processing technique should have the following features:

1. It is somehow aware of video content rather than blind operation and ensures efficient computation, thus reserving the hardware resource for other essential processing.
2. It is memory-efficient and consumes bus-bandwidth to a limited extent.

In this paper, a run-processing method is proposed to support high-resolution real-time video object segmentation. Making use of coherence property of binary masks, run-processing possesses content-awareness and reduces memory usage by 67%. Binary morphological operations, connected components labeling, seeded region growing, bitwise operation, and random-search operations are accelerated. Furthermore, run-processing is suitable for hardware implementation. Simulation shows that based on the proposed hardware architecture, a shadow-cancellation video object segmentation algorithm achieves an average speed of 151 frames per second at D1 (720x480) resolution.

The structure of this paper is as follows: The succeeding section illustrates run-coding this paper adopts. Next, Run-processing Functions section describes the methods of each operation with run-coding. Then, the proposed architecture are demonstrated. Finally, experiment results are shown and conclusion is followed.

Run-coding

Coherence Property

Since foreground (FG) binary mask attempts to store the existence of active object, ‘1’ pixels (pixels at where a object is found) of a object tends to be spatially connected into a closed region. Identical characteristic applies to ‘0’ pixels. This is called the coherence property of foreground binary masks.

To achieve both coded-domain computation capability and storage reduction, row-based Run-coding is proposed.

Run-coding Format

Binary runs are of the form (X,Yₜ), which represents a series of pixel labeled as Yₜ starting from address X. Fig.1 shows the run coded data of a binary mask. The rows in a frame are then separately encoded into several runs. The number of runs depends on the mask content. \(N_{w,0}\) indicates the ‘end’ run of a row. Where \(N_w\) is the length of a row; in this example \(N_w=25\). The encoding processing can be done by single raster scan. Moreover, independence between rows allows further acceleration through processing multiple rows in parallel.

Figure 1 – Example of Run-coding (a)Pixel representation (b) Run-coded format

\(X = \log_2 N_w\) bits and \(Yₜ\) is 1 bit. Therefore, each run is \((\log_2 N_w+1)\) bits long and the storage requirement of a binary mask equals \(N_{run} \times (\log_2 N_w+1)\). \(N_{run}\) is the number of total runs in a frame. For example, run-coded representation for 720x480 binary masks of Fig.2(a) takes 1,858 runs. During segmentation process, erroneous patterns usually corrupt the masks, leading to increased run numbers. Noisy masks like Fig.2(b) contains 4,636 runs, which is equivalent to 50.9Kb storage. Even so, run-coding still achieves 85.3% storage reduction compared with 345.6Kb, the pixel-based storage size of a D1 image. Experiments have shown an average of 78% storage reduction in segmentation processes.

Figure 2 – General binary mask (a)refined binary mask (b)noisy binary mask
Run-processing Functions

In this section, the most frequently used functions in VOS are illustrated.

Binary Morphology

Binary morphology consists of two core functions, dilation and erosion[3]. Mathematically all of the morphological operations can be implemented by cascading these two functions. Dilation turns a pixel into ‘1’ if any ‘1’ pixel exists in a pre-defined window, structure element (SE), centered at the pixel. Conversely, erosion turns pixels into ‘0’ if none of the pixels in SE is ‘1’. Breuel[17] used a run-length representation and divided morphological operations into horizontal and vertical steps. However, vertical morphology requires transposing the entire frame, which needs to buffer nearly the entire frame. Instead of the impractical transposition, a multi-row processing is proposed to realize the vertical operation.

**Figure 3 – a (3x7) dilation example**

Morphological operations with (A x B) SE are divided into Intra-row and Inter-row morphology. Intra-row implements (Ax1) morphology for each row. Then, Inter-row executes (1xB) morphology by processing B rows together. A dilation process is shown in Fig.3. Intra-row scans each row, extending every ‘1’ run by A/2 in both its ends. Disappeared ‘0’ runs are erased and connected ‘1’ runs are merged. Inter-row scans B rows at a time. In (1xB) dilation, a ‘0’ run exists only if all the B rows is labeled ‘0’ at the address. Inter-row dilations processes as follows:

1. Read the first run of all B rows; if there is no ‘1’ run: push (0,0) into result, go to step2; otherwise, push(0,1) and go to step3
2. Each non-ended row reads next run and either a ‘1’ or ‘end’ is met. Go to step3.
3. Push \( A_{\min 1} \) and go to step4. \( A_{\min 1} \) is minimal address of current ‘1’ runs and the starting address of resultant ‘1’ run.
4. Forward all B rows until all current runs is ‘0’ or ‘end’. Go to step5.
5. If all current runs are ‘end’, push ‘end’. Stop the process. Otherwise, maximal address of current ‘0’ runs \( A_{\max 0} \) denotes the starting address of a ‘0’ run. Push \( A_{\max 0} \) and go to step 2.

Both Intra-row and Inter-row can be done by single scan, thus alleviating enormous buffers in previous works. Substituting address update scheme for pixel-based label assignment greatly reduces computation. Moreover, Intra-row process is regardless of SE width. Morphology of extreme large SE width requires exactly the same resource as that of a small SE.

According to duality property between dilation and erosion, erosion can be done by inverting input and output labels of dilation process.

Connected Component Labeling

Through run-coding, continuous part of a region is already grouped. Hence, run-processing CCL only needs to solve run connection via neighboring rows. Nicol introduced a two-path run-length CCL[11]. Appiah et al.[17] implemented CCL hardware with another run-length algorithm. It adopted a data structure storing serial identity number, equivalence label, starting and ending address, and corresponding row of a run. The hardware managed equivalence through global storage. Consequently, updating equivalence consumes large amount of bus-bandwidth. This serially numbered data organization can not support multiple functions. In addition to the storage issue, the use of iterative algorithm restricted it against further acceleration.

**Figure 4 – an example of Equivalence Checking (a) Top-down building equivalence table (b) a: after equivalence checking (c) Bottom-up building equivalence table (d) c: after equivalence checking**

Run-processing utilize a top-down-bottom-up two-path algorithm. Using single-row equivalence table avoids huge global equivalence table. Pseudo process is listed below:

**Top-down Process:**

1. Initialize new label number; scan first row, assign new label for ‘1’ runs; let i = 0 and go to step 2.
2. Scan the 1th and (i+1)th rows. Push \( <G,K> \) into equivalence table and assign G label to the lower-row run if upper-row G label run encounters a lower-row K label run(Fig.4(a)); assign lower ‘1’ runs the label of connected upper-run; assign new label for
isolated ‘1’ runs in (i+1)th row. Go to step 3.

3. Scan local equivalence table for each non ‘0’ runs in (i+1)th row; change run labels $G_i$ into $K_j$. $<G_j,K_j>$ is equivalent units in equivalence table. If (i+1)th row is the last row, go to Bottom-up Process, otherwise $i = i+1$ and return to step 2.

Bottom-up Process:

1. Scan the ith and (i+1)th rows. Push $<G,K>$ into equivalence table and assign G label to the upper-row run if lower-row G label run encounters a upper-row K label run; Go to step 2.

2. Scan local equivalence table for each non ‘0’ runs in ith row; change run labels $G_j$ into $K_j$. If ith row is the first row, end the process, otherwise $i = i-1$ and return to step 1.

The adopted CCL algorithm also contributes to bus-bandwidth reduction, which is later explained in Hardware Architecture section.

Full-frame Bitwise Operations

Recall that in binary morphology Inter-row process, a ‘0’ run exists only if all of the corresponding B rows is ‘0’ at the address. This is in essence the “OR” operation. Programmable input/output label inversion of Inter-row process allows all of the bitwise combinations such as $\text{And/Or/Nand/Nor}((\text{Mask}_A \& \text{Mask}_B) \& \neg\text{Mask}_C)$. For B input binary masks, $2^{B+1}$ modes are supported. This implies a reusable hardware module.

Seeded Region Growing (SRG)

SRG is done utilizing CCL and seed insertion. First a binary mask to be seeded is labeled by CCL. Then the seed mask is inserted into the label image, i.e., runs encountered with seed are labeled as ‘seed’ label (a fixed new label number). After the insertion, a confined CCL which only propagates ‘seed’ label is executed. Finally unwanted regions are erased by label extraction and a binary mask of seeded region is obtained.

Other Functions

Other functions such as boundary extraction, bounding box extraction and random search can be done in similar procedures, which is out of the scope of this paper.

Hardware Architecture of Run Processor

The architecture of Run Processor is shown in Fig.5. Processing Element (PE) is implemented by two multi-functional modules because of the highly correlation between functions. Instruction Decoder module decodes the input instructions and assigns the process. External Memory Allocation module stores external starting address of data frames and currently available address of external memory data banks.

Run Data I/O

Process Control module manages Row-to-PE linker according to function requirements. Each of the 10 Row -storage can buffer $N_{\text{pagerun}}$ runs. $N_{\text{pagerun}}$ is set to 11. The Row-storage serves as temporary buffer for read and write commands. For instance, Inter-row morphology uses one row as output buffer and the others as input row buffer. After Inter-row morphology completes a row, output buffer is written out to internal storage and external storage and the next row of the B rows should be read in for the process. In addition, the other rows should be rewound to the first run. Process Control module requests a read-in command to the new row and sends B-1 rewind commands to the other rows. Memory Access Unit reads data from external memory to the corresponding internal SRAM bank. Fig.6 shows the internal SRAM and Row-storage structure. Each bank of internal SRAM comprises 4 pages. The number is empirically chosen for the reuse scheme. The first page of read data is then transferred to the new row buffer, which completes the data preparation for next process.

If the row consists of more runs than row bank can hold, the address to the succeeding runs is kept in External Memory Allocation module for the next read-in. Using 10 Row Sub-storage, the maximum SE size of a single morphology is $(N_w \times 9)$.

Internal Data Reuse Scheme

Data are reused in two phases—page-reuse phase and bank-reuse phase. During each operation, Process Control requests mostly rewind than read-in command because of the access pattern of windows-based functions. If a Row-storage did not request a page in previous operation, the data in Row-storage is still the first page of the row, thus alleviating an internal read-in for rewind command. Likewise, page-reuse phase works if the row bank is not overwritten by further read-in, rewinding the row can be done simply by internal read-in instead of external memory access. Furthermore, temporary data such as local equivalence table are not written out to external memory if possible. 42% reduction external transmission is reduced by the data-reuse scheme.
Label Storage Arrangement

Consider the adopted two-path algorithm. In each path, continuity is check in two steps: (i) Neighboring rows, the upper-row and lower-row, are scanned to build local equivalence table. (ii) The lower row is scanned again and labels are updated according to equivalence table.

In step (i), continuity is checked and the required information is the address and label of runs in both rows, while step (ii) reads lower-row runs and equivalence regardless of run address. This insight suggests a divided storage format, i.e., storing labels and address separately. We empirically select run label size $Y_L = (\log_2 N_r + 1)$ bits. Thus corresponding run labels of a row are stored in the same size of the same order, allowing identical access methods. Application simulation justifies that $Y_L$ size is enough for related functions.

<table>
<thead>
<tr>
<th>Operation</th>
<th>averaged computation time(ms)</th>
<th>fps</th>
<th>label required per frame</th>
<th>required time per frame(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morphological Operation 9x9</td>
<td>0.58</td>
<td>2605</td>
<td>6</td>
<td>2.3</td>
</tr>
<tr>
<td>Seeded Region Growing</td>
<td>0.67</td>
<td>1498</td>
<td>1</td>
<td>0.67</td>
</tr>
<tr>
<td>Connected Component Labeling</td>
<td>0.65</td>
<td>1535</td>
<td>1</td>
<td>0.65</td>
</tr>
<tr>
<td>Seed Insertion</td>
<td>0.41</td>
<td>2443</td>
<td>1</td>
<td>0.41</td>
</tr>
<tr>
<td>Label extraction</td>
<td>0.2</td>
<td>4949</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>Full-frame Bitwise Operation</td>
<td>0.4</td>
<td>2512</td>
<td>6</td>
<td>2.39</td>
</tr>
<tr>
<td>Segmentation Algorithm</td>
<td>0.7</td>
<td>151</td>
<td>1</td>
<td>6.62</td>
</tr>
</tbody>
</table>

Experiment Results

Proposed hardware architecture is tested on a shadow cancellation video segmentation algorithm[22]. In addition, post-processing techniques including morphological refinement, CCL and region-area-threshold is applied to generate object masks[21]. The synthesized 7.8ns clock cycle leads to 151fps segmentation efficiency on D1 (720x480) resolution. Table 1 shows the computation time of tested algorithm. Fig. 8 illustrates the comparison with published architectures. In terms of gate-count, computation cycle, and GxC merit(gate-count x computation cycle/1Meg), Run Processor outperforms the existing architectures. The reported gate count is of overall multi-function architecture and is therefore slightly more than that of CCL module of [10]. Binary Morphology operation requires 13.9% computation cycles relative to [19] and CCL takes approximately 16% cycles compared to [10].

Although Run Processor supports multiple functions regardless of data content, boarding it to other applications necessitates analysis of data characteristics. Without the coherence property, the benefit from processing multiple pixels at a time would be lost.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Computation Cycle</th>
<th>Gate Count</th>
<th>CCL Cycle</th>
<th>Region Area Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morphological</td>
<td>7.8</td>
<td>500000</td>
<td>13.9%</td>
<td>16%</td>
</tr>
<tr>
<td>Seeded Region</td>
<td>7.8</td>
<td>500000</td>
<td>13.9%</td>
<td>16%</td>
</tr>
<tr>
<td>Connected</td>
<td>7.8</td>
<td>500000</td>
<td>13.9%</td>
<td>16%</td>
</tr>
<tr>
<td>Component</td>
<td>7.8</td>
<td>500000</td>
<td>13.9%</td>
<td>16%</td>
</tr>
<tr>
<td>Label</td>
<td>7.8</td>
<td>500000</td>
<td>13.9%</td>
<td>16%</td>
</tr>
<tr>
<td>Full-frame Bitwise</td>
<td>7.8</td>
<td>500000</td>
<td>13.9%</td>
<td>16%</td>
</tr>
<tr>
<td>Segmentation</td>
<td>7.8</td>
<td>500000</td>
<td>13.9%</td>
<td>16%</td>
</tr>
</tbody>
</table>

Figure 8 – Comparison with existing architectures, (a) Connected Component Labeling (b) Binary Morphology

Conclusion

A run-processing method for utilizing coherence property of
binary masks of video object segmentation is proposed. Multiple widely-used functions such as binary morphology, connected component labeling and seeded region growing is supported. A shadow-cancellation video object segmentation algorithm run by the presented hardware implementation achieves 151 fps for D1 (720x480) video. The presented hardware is shown to be more cost-effective than existing architectures.

References


