Strip-based Watershed using Multiple-bank Memory Storage

Lee Seng Yeong, Li-Minn Ang, Kah Phooi Seng
School of Electrical and Electronic Engineering
The University of Nottingham
Jalan Broga, Semenyih, Selangor 43500, MALAYSIA
e-mail: {eyxlsy, kezklma, kezkps}@nottingham.edu.my

***Abstract*—In this paper we propose a single-pass strip-based watershed segmentation using multibank memory. This watershed segmentation is based on rainfall simulation and is targeted for low memory applications whereby only a small amount of memory is needed for processing at any one time. Using multiple memory banks for storage also ensures that clock cycles for memory accesses are kept to a minimum.**

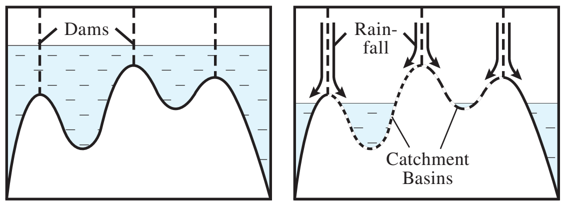
***Keywords*-strip-based, single-pass watershed, multibank memory**

I. Introduction

Watershed processing is a segmentation method based on an image's topography. Each grey level is treated as a gradient and the segmentation is based on how water accumulates in these regions. For immersion based watershed, region boundaries are formed by the dams built during flooding. Using the rainfall simulation method, the boundaries are formed between catchment basins. The differences are illustrated in Fig. 1. However, watershed processing typically requires a lot of memory. In many applications such as those whereby the watershed needs to be implemented on FPGA, a low memory version of the watershed is desired.

One such low memory method for immersion based watershed is given in [1]. In [2] new data structure for the output is proposed to minimize memory usage for storage. The approach to minimize memory usage used in this paper is named strip-based watershed, called so because it processes a strip (i.e. a few rows of pixels) at a time. It is loosely based on the rainfall watershed proposed in [3], [4] and has been modified for single-pass operation. The modifications includes merging of the arrowing and labelling stages of the watershed and to how plateaus are handled. In addition to processing the watershed in strips, a multibank memory as described in [4], [5] is used. This helps to reduce the number of clock cycles for memory access, especially when multiple values are required before processing can start by allowing these values to be accessed in parallel. The general overview of the system is shown in Fig. 2.

The main challenge for processing the watershed in strips is the way the plateaus are handled. In [4], the entire plateau is put into a queue and "eroded" one pixel at a time from the outermost pixel moving inwards. During each pass, these outermost pixels are labelled with the direction of it's lowest neighbour. In strip based processing, we no longer have the entire image in memory and consequently neither the whole plateau. Without knowing the extent of the plateau, it will not be possible to accurately determine the direction of steepest descent. This is illustrated in Fig. 3. The rest of the paper is organized as follows. Section II and III will briefly describe the watershed segmentation and multibank memory used in [4]. Section IV will describe our single-pass rainfall-based watershed using the multibank memory. Finally, Section V will summarize this paper.



(a) Flooding watershed. Boundaries are formed by dams between flooded regions.
(b) Rainfall watershed. Boundaries are formed between catchment basins.

Fig. 1. Flooding vs rainfall watershed.

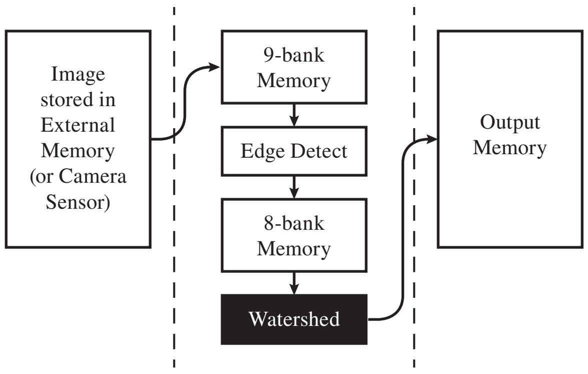


Fig. 2. The block diagram of the proposed strip-based watershed with multibank memory.

II. Rainfall watershed

The watershed process described in [4] is a two-stage process, arrowing and labelling. During the arrowing stage, each pixel is labelled with the direction of its lowest neighbour unless all its neighbours are of a higher value and forms a

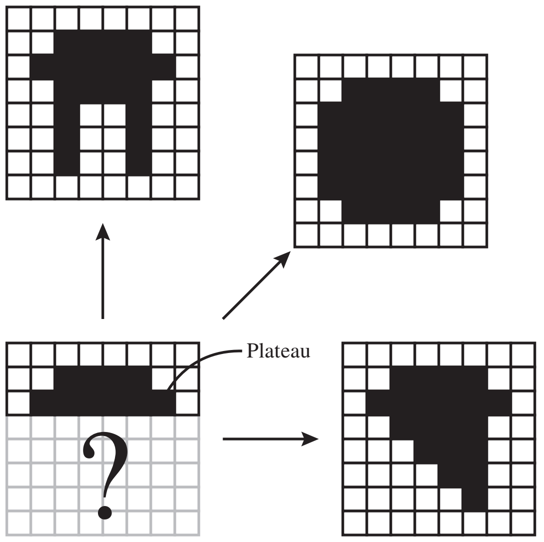


Fig. 3. In strip-based processing, the extent of a plateau is unknown until it is complete. Illustrated here are possible plateau shapes if only the first two rows of the plateau are known. Depending on the neighbouring pixels, these three possibilities can yield different segmentation results.

lower minima. This basically creates a chain code that points each pixel to its associated lowest minima. The labelling stage then labels each pixel using this chain code into their respective groups forming catchment basins. It is the boundaries between the catchment basins that form the edges of the segmented regions. In our proposed system, these two steps will be done in a single step eliminating the need to have to effectively go through the whole image again.

During the arrowing stage, the lowest neighbour is determined by comparing the values of the centre to the four neighbours to the north, east, south and west (using Cardinal directions) shown in Fig. 4(a). This is called 4-neighbourhood connectivity and it is the same one used for the strip-based watershed. This is good when there is only one lowest neighbour. In cases where there are two or more lower valued neighbours (but not the same value as the current pixel), the direction is chosen based on a predetermined order. See Fig. 4(b). In cases where the current pixel and at least one of the neighbours have the same value, they are considered as plateau pixels. As described earlier, this is processed by a recursive "erosion" process using two queues.

To perform this watershed processing, large amounts of memory is typically required with a minimum of the whole image stored in memory. We proposed a strip based method which allows processing of the watershed using less memory by processing only part of the image at any one time. This also eliminates the need to have large queues to process the plateau regions recursively.

III. Multibank Memory

In the proposed system, multiple memory banks as described in [4], [5] are used for parallel value access. Two different instances of this multibank memory is used for storage,

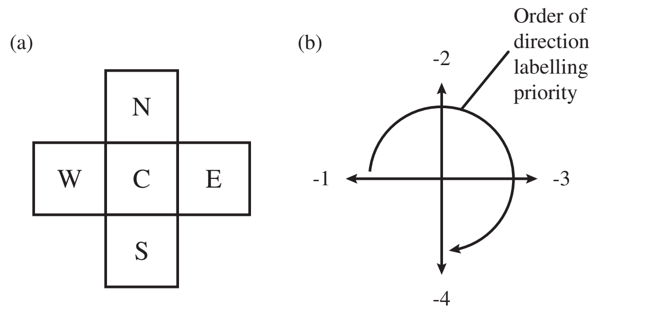


Fig. 4. (a) Values needed for 4-neighbourhood connectivity and (b) Direction precedence, starting from W(highest priority), N, E and ending with S(lowest priority). The negative numbers are the labels used during the arrowing stage when finding the direction of the lowest valued neighbour.

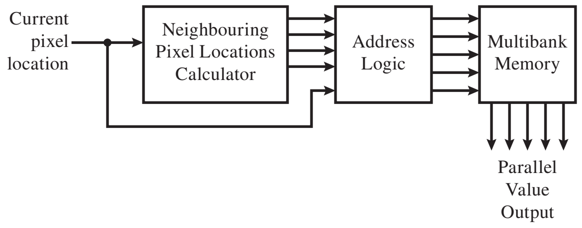
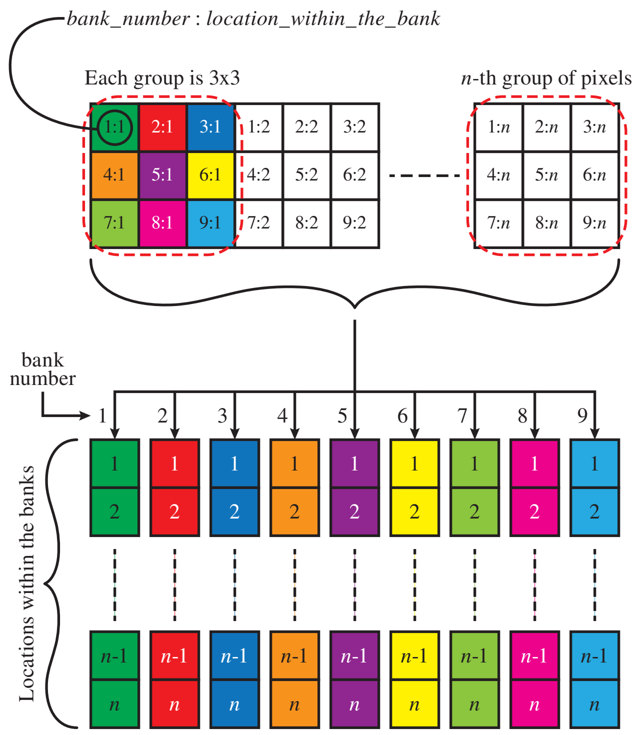


Fig. 5. How multiple values are accessed in parallel using only the current pixel location as input. Shown here is for a five parallel value access.

(1) a nine bank memory for nine parallel value access which used during edge detection and (2) an eight bank memory for five parallel value access which used during the watershed transform. These are shown in Fig. 6 and Fig. 7 respectively.

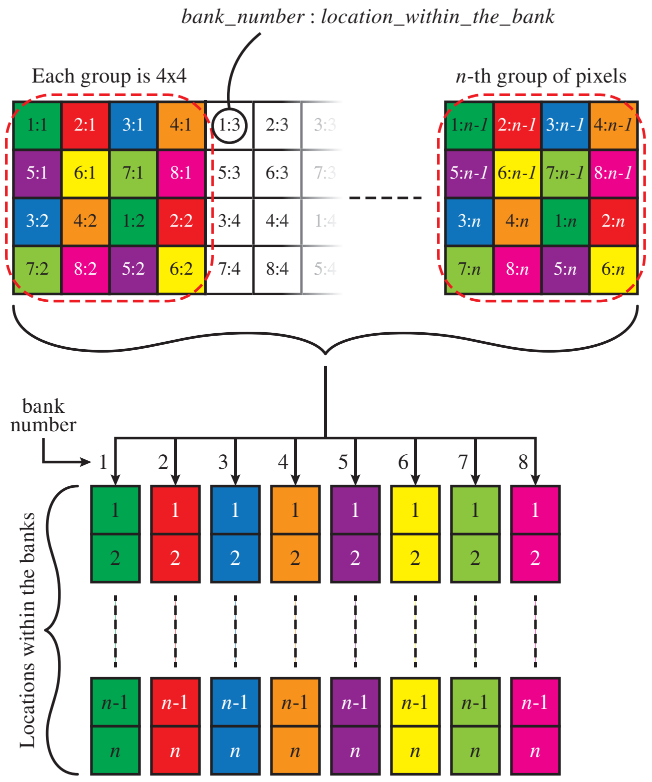
The multibank memory allows parallel values access from random pixel locations by calculating the memory addresses for the banks and location within the banks based on the current pixel position. This is normally some simple logic which can be derived by mapping the binary values of the pixel locations to their respective banks and locations within the banks. During writes to memory, typically only one or two values are written at a time and this is normally done in a fixed manner. For this purpose, it is typically better to use a lookup table. Parallel values are obtained by activating multiple banks concurrently. This is done by calculating the memory locations of as many parallel values as needed and using it as an input into the multibank system. In the case of edge and watershed processing, the parallel values are the current pixel and its neighbouring pixels, generating five or nine values respectively. This is shown in Fig. 5.

During operation, pixel values from the camera are stored in the nine-bank strip buffer in a sequential manner, one pixel at a time. An addressing scheme will select the correct bank and address to which the values read from the camera will be stored into. Values in this nine-bank strip buffer serves as the input to the edge detection module. After edge detection, the edge values are stored in the eight-bank memory. This is followed by the watershed which takes these edge values as input. This is shown in Fig. 2.



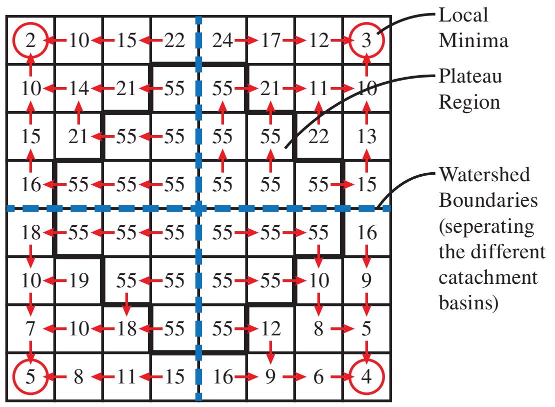
bank_number : location_within_the_bank
Each group is 3x3
n-th group of pixels
bank number
Locations within the banks

Fig. 6. Image from camera is stored into nine memory banks for nine parallel value access. Shown here for a 3xn-th group strip where n is the last group that can contain the whole width of the image.



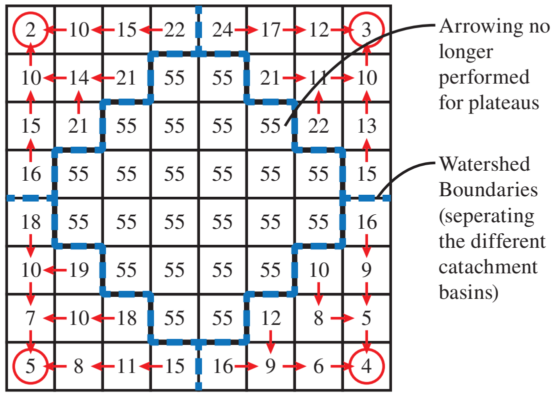
bank_number : location_within_the_bank
Each group is 4x4
n-th group of pixels
bank number
Locations within the banks

Fig. 7. Storage for the edge detected image. Shown here is a 4xn-th group memory which is arranged in an eight-bank configuration.



Local Minima
Plateau Region
Watershed Boundaries (seperating the different catchment basins)

(a) Result of original watershed using edge erosion for the plateau. The plateau is split into four. The result is four segmented regions (or four catchment basins) with the boundaries of each shown by the detached line.



Arrowing no longer performed for plateaus
Watershed Boundaries (seperating the different catchment basins)

(b) Result of the strip-based watershed by treating plateau regions as a complete catchment basin. The result is five segmented regions.

Fig. 8. Results of the rainfall-based watershed in [4] and our proposed system

IV. Single-pass Strip-based Watershed

As mentioned in the introduction, the main challenge to implement the rainfall simulation based watershed proposed in [4] is the way the plateaus are handled. Since the data coming in is "incomplete" there is no way to know to what extent the plateaus go. To overcome this, we propose a modification to this method and treat the entire plateau as a catchment basin as done in [6]. By doing so, a plateau does not require arrowing but treated as a catchment basin (one complete region). For this, each plateau is processed using the connected components processing described in [7].

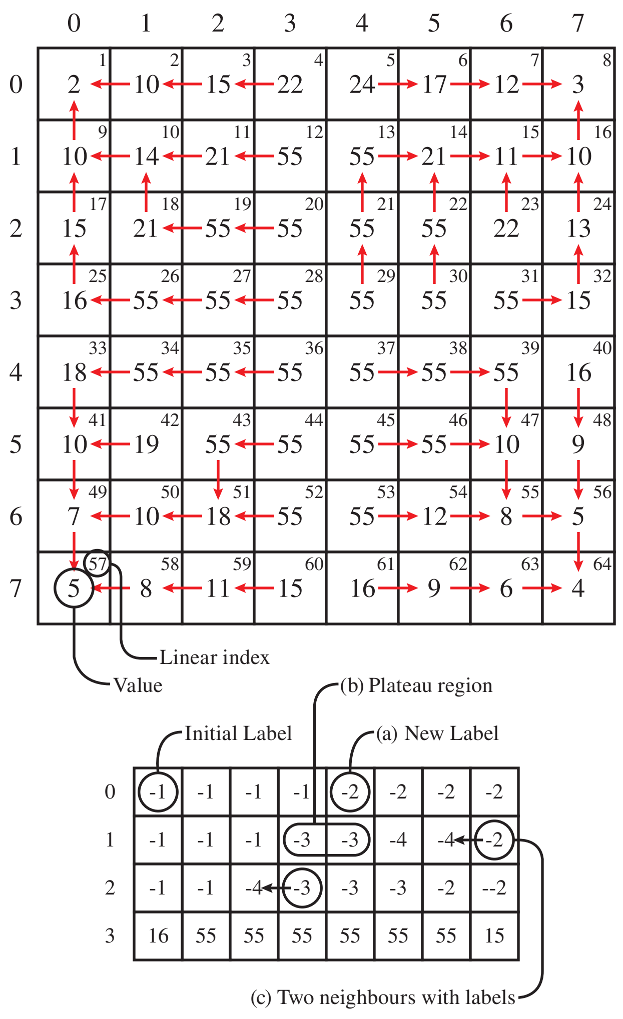
An example of this watershed labelling for the first three lines is shown in Fig. 9. The single-pass watershed works by immediately assigning a label to the first pixel processed. The next pixel points toward an already labelled pixel and will assume the same label. When a pixel breaks this connectivity, a new label is assigned and the process continues as before, labelling each connected pixel with this new label. In the given example this occurs at pixel location 5. At this point, it points to a different direction and a new label is given. When a labelled pixel points to an unlabelled pixel, both will get labelled at the same time. This occurs at pixel 5 which points to the unlabelled pixel 6. Both pixels will be assigned the same label at the same time. When pixel 6 is processed the same thing occurs and pixel 7 will get labelled with the label of pixel 6.

The problem with this is that if a new group of connected pixels is connected to a previous group, it will require label corrections. Fortunately, this only needs to be done for a selected number of pixels, namely the pixels which came before the new found connectivity. In the example given, this occurs twice, at pixel location 16 and 20. At pixel 16, it will discover that it has two neighbours which have been labelled. A flag is used determine which pixels have been labelled. In the example, it is denoted by a negative sign. For pixel 16, it will assume the smallest group number. The system will then backtrack through the row following the direction of the neighbours with the "wrong" group label. Only pixels 14 and 15 require relabelling. Correction is done immediately once it is discovered.

The is correction can only be performed on the strip currently in memory. Without complete correction over segmentation will occur. Fortunately, this can be rectified during the stage which follows segmentation. This is done by using an association tree which is generated when the correction marker moves to the boundaries of a strip. This association tree is used each time an operation is performed on the segments, such as size or average value calculations and again during the actual segmentation. Instead of using only one group of pixels, other groups are included in the calculations and processing depending on the association tree.

V. Summary

In this paper, we have proposed an efficient single-pass strip-based rainfall simulated watershed segmentation. By treating the plateau as a catchment basin it allows us to eliminate the queues used for plateau processing in [4]. Clock cycles required for memory access are also kept to a minimum by allowing parallel value access through the use of multibank memory storage.



(a) A new label is given to the pixel once the connectivity between the pixels is lost.
(b) A plateau region will be given a new label.
(c) A pixel with two neighbours which have already been labelled will change the state of the system to "label correction". The smaller valued label is chosen. The system backtracks on the same row and relabel the mislabelled ones.

Fig. 9. Example of the single-pass strip-based watershed. The linear index is given for easier reference during the explanation and the arrows represent paths of steepest descent for each pixel. (These paths are not used for the plateau processing)

References

[1] I. Pitas and C. I. Cotsaces, "Memory efficient propagation-based watershed and influence zone algorithms for large images," *IEEE Transactions on Image Processing*, vol. 9, 2000.

[2] J. De Bock and W. Philips, "Line segment based watershed segmentation," in *Computer Vision/Computer Graphics Collaboration Techniques*, ser. Lecture Notes in Computer Science, A. Gagalowicz and W. Philips, Eds. Springer Berlin / Heidelberg, 2007, vol. 4418, pp. 579–586.

[3] V. Osma-Ruiz, J. I. Godino-Llorente, N. Sáenz-Lechón, and P. Gómez-Vilda, "An improved watershed algorithm based on efficient computation of shortest paths," *Pattern Recogn.*, vol. 40, no. 3, pp. 1078–1090, 2007.

[4] L. S. Yeong, C. W. H. Ngau, L.-M. Ang, and K. P. Seng, "Efficient processing of a rainfall simulation watershed on an FPGA-based architecture with fast access to neighbourhood pixels," *EURASIP Journal on Embedded Systems*, vol. 2009, no. 318654, p. 19, 2009.

[5] D. Noguet and M. Ollivier, "New hardware memory management architecture for fast neighborhood access based on graph analysis," *Journal of Electronic Imaging*, vol. 11, no. 1, pp. 96–103, 2002.

[6] J. De Bock, P. De Smet, and W. Philips, "A fast sequential rainfalling watershed segmentation algorithm," in *Advanced Concepts for Intelligent Vision Systems*, ser. Lecture Notes in Computer Science, J. Blanc-Talon, W. Philips, D. Popescu, and P. Scheunders, Eds. Springer Berlin / Heidelberg, 2005, vol. 3708, pp. 476–482.

[7] L. S. Yeong, L.-M. Ang, and K. P. Seng, "Efficient connected component labelling using multiple-bank memory storage," *Proceedings of the 2010 3rd IEEE International Conference on Computer Science and Information Technology*, vol. 9, pp. 75–79, July 2010.