143.474 Parallel Programming

Seminar Report

5/26/2009

Name: Siddharth Magazine ID NO: 04061535

TABLE OF CONTENTS:

ABSTRACT:

An important aspect of designing a parallel algorithm is *exploitation* of the data locality for minimization of the communication overhead. Aiming at this goal, we propose here a reformulation of a global image operation called the *watershed transformation*. The method lies among various approaches for image segmentation and performs by labelling connected components. Both serial and parallel programming models are presented and evaluated when running on SUN and DEC Alpha AXP workstations, and a Cray T3D, respectively. The watershed transformation is widely used in morphological image segmentation, and therefore, large amount of data and consequently involving complex analysis use parallel algorithms (Moga, 2009).

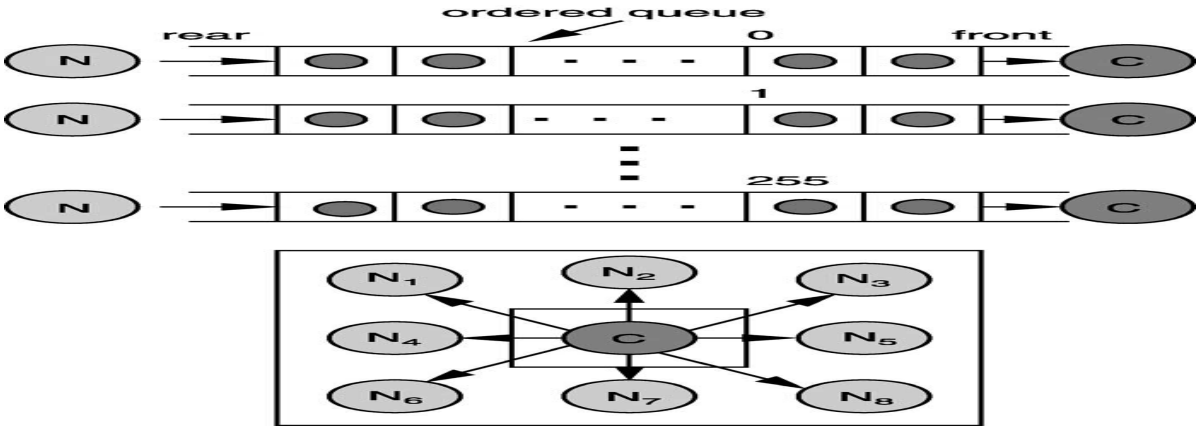
INTRODUCTION:

Watershed algorithm is an image processing segmentation algorithm that splits an image into areas based on the topology of the image. Also, there is watershed transformation which is basically a mid-level operation used in morphological image segmentation. *Watershed transformation* is a segmentation algorithm applied on gray-scale images, which detects and label objects, which are connected components of similar gray-level. It has been widely and successfully applied in different domains, especially in computer vision applications as a powerful segmentation tool. A more common image representation is provided by splitting the morphological gradient of an image into geodesic influence zones. In the initial stages, parallel realisation was based on image segmentation and sequential raster which was very expensive due to repeated scanning's. So, due to the result of recursive nature of watershed transformation, it can be said that parallelisation is not an easy task. Also, interactive water transformations always allows us to include and exclude points to create artificial watersheds, which therefore can enhance the result of the segmentation (Viero, 2009).

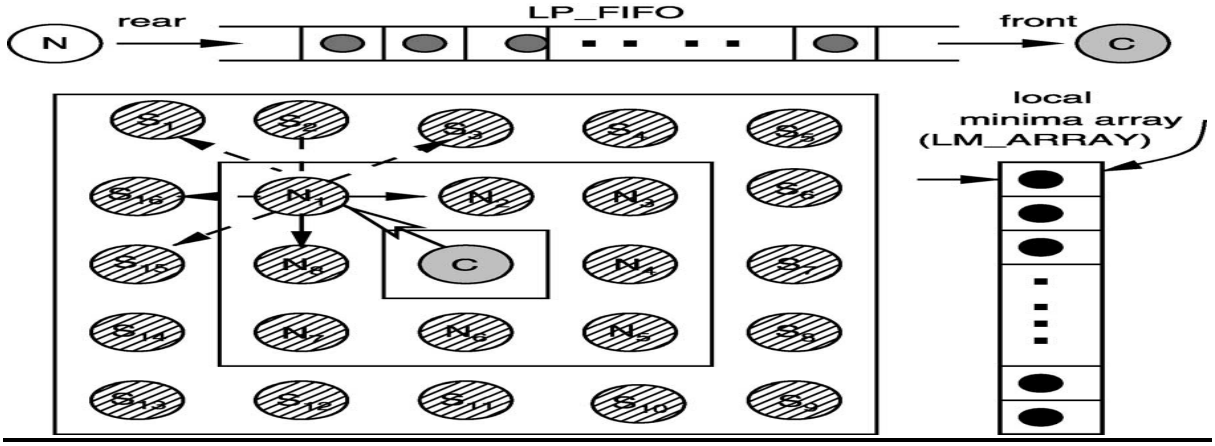
METHOD:

The watershed algorithm based on *rain falling simulation* performs segmentation by labelling connected areas within the gradient of an image. But one of the most common watershed algorithm we have is called as *Meyer's* watershed algorithm. This type of algorithm works on a grey scale image, where a set of markers are chosen , and each marker is given a different label. Here, the neighbouring pixels of each marked area are first inserted into a priority queue, with a priority level corresponding to the gray level of the pixel. Then, the pixel with the highest priority level is extracted from the priority queue, and if the neighbours of the extracted pixel have all the same labels, then the pixel is labelled with their label, and plus all the non-marked neighbours that are not yet in the priority queue are placed in the priority queue. This process is repeated over and over again until we get the priority queue to be empty. The non-labelled pixels are called the *watershed lines* (Watershed (algorithm), 2009).

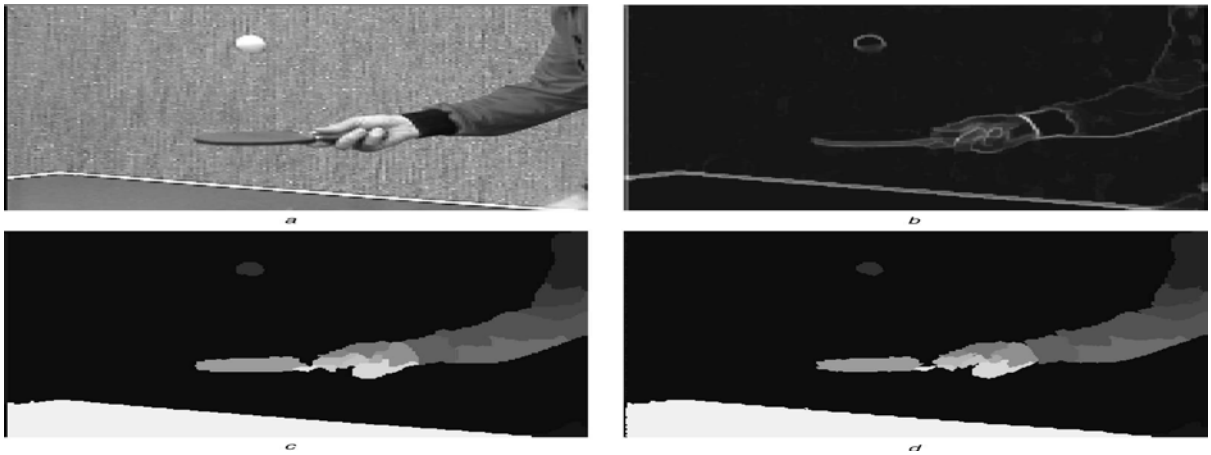
RESULTS:



Label propagation in Meyer's algorithm



Label propagation in proposed algorithm



Comparison of Meyer's algorithm and proposed algorithm for 'Tennis' image

DISCUSSION (Based on the above two algorithm's)

Looking at the label propagation of *Meyer's algorithm*, from the figure, we have a label propagation, in which C is the centre pixel and N1 –N8 are the eight-connected nearest neighbours. Also, we can see that the ordered queues are initialised with the labelled regional minima which at least have one non-labelled neighbouring pixel. Pixels are dequeued from the current FIFO queue, one at a time, and then its labels are assigned to the non-labelled neighbouring pixels of higher altitude, which in turn are inserted in the corresponding FIFO queues. When all the queues have been emptied, it means that every pixel in the image has been labelled and the flooding procedure stops. Here the label propagation requires 256 queues, which makes the hardware realisation of Meyer's algorithm complex and costly (Moga, 2009).

Now, looking at the label propagation in proposed algorithm, from the figure, we keep the identification of local minima and their labelling as same in Meyer's original algorithm. Here, we have a label propagation, which decides the labels of neighbouring (N1 – N8) pixels of the centre pixel C, but it also requires 16 additional supporting pixels (S1 – S16) as shown in the figure. Also, we note that only one single queue, is used against the 256 queues required in Meyer's label propagation scheme. In the first step, we detect the labelled regional minima which have at least one non-labelled neighbouring pixel and that is stored in an array named LM_ARRAY , and then we initialise the queue LP_FIFO with the array of labelled regional minima one at a time, and at last when all the local minima of LM_ARRAY have been visited , it signifies that each pixel in the image has been labelled and the entire flooding procedure stops. Also, it should be noted that to decide the label of a pixel Ni, i = 1,....8, the four connected neighbours of Ni, and the eight-connected supporting pixels of Ni, are considered and processed in different conditional steps (Moga, 2009).

DISCUSSION (Based on the parallel algorithm)

Divide-and-conquer parallel implémentation is the type of algorithm chosen here, and one of the methods I have mentioned is *detection and labelling of the minima pixels* algorithm which is as follows:

Detection of minima in each processor P_i

Initialization: /* MAX_LABEL < NARM < OUTBOARD */

$\forall p \in D_{oi}^{d}$, $o_i(p)$= NARM and $\forall p \in D_{oi} \setminus D_{oi}^{d}$, $o_i(p)$= OUTBOARD

$\forall p \in D_{di}^{d}$, $d_i(p)$= MAX_DIST (=∞) and $\forall p \in D_{di} \setminus D_{di}^{d}$, $d_i(p)$=0

Input:

$f_i.o_i.d_i.\lambda_i = 1.current\_label_i = i \times \frac{\text{MAX\_LABEL}}{P}$

;

Output: o_i,d_i,λ_i,λ.

(1) Raster scan($p \in D_{fi}^{d}$ not visited before) {

(1.1)Inquiry the neighbourhood of *p*

(1.2)if *p* is a regional minimum then

$o_i(p) \leftarrow current\_label_i$;$current\_label_i \leftarrow current\_label_i$+1;

(1.3)else if *p* is an inner pixel then explore the plateau originating in *p*

}

(2)*state*← ON

(3)while(*state* = ON) {

(3.1)*state*← OFF

(3.2)for each($P_j \in N_G(P_i)$) Send Receive($E(o_i,o_j)$,$E(d_i,d_j)$,$B(o_i,o_j^{'})$,$B(d_i,d_j^{'})$)

(3.3)for each($P_j \in N_G(P_i)$) UpdateEdge(j,f_i,o_i,d_i,$state_i$)

(3.4)UpdateDistributionSubdomain(f_i,o_i,d_i,λ_i)

(3.5)All_Reduce($state_i$,*state*,OR)

}

(4)All_Reduce(λ_i,λ,MAX)

Step (1) comprises the pseudocode for the serial execution of this stage. Thus, at step (1.1), by checking the graylevels of the neighbouring pixels in four- or eight-connectivity, *p* can be classified as a local minimum (strictly higher neighbourhood), case in which *p* is labelled with the current label at step (1.2), an inner pixel (higher or equal neighbourhood), or a non-minimum (otherwise).

At step (1.3), a plateau is scanned in a breadth-first order, and visited pixels are labelled with the current label value. The examination always starts from an inner pixel which introduces in the list of candidates neighbouring pixels of equal altitude (recall that an inner pixel which is not a local minimum has such a neighbourhood). A currently investigated candidate pixel *q* may still satisfy the inner condition; otherwise, it has lower neighbours, and *q* is an outer pixel. In the latter case, according to the definition, the lower distance value is 1, and *q* is stored in a list. If, after scanning the plateau, the list of outer pixels is not empty, another breadth-first scan is performed to reset the label of the plateau to NARM, compute the lower distance function, and update the value $\lambda_i = \max_{r \in {}^{d}_{Ddi}\{d_i(r)\}+1}$. For this purpose, a wave front, comprising at the beginning all

outer pixels, drifts iteratively and exhaustively across the plateau. Pixels swept by a wave set the lower distance to the time stamp of the wave, which is initially 1 and is incremented at each iteration. The partial labels and lower distances inside non-minima plateaus for the image example in Fig. 1(a) are illustrated in Fig. 5, when four processors are used (‘M' stands for MAX_DIST and ‘N' for NARM). Let us notice that for non-minima pixels which are not on a non-minima plateau, as well as for minima pixels, the lower distance remained MAX_DIST.

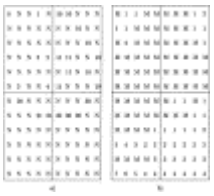

Full-size image (19K)

Fig. 5.

(a) Local labels for minima plateaus; (b) local lower distances for non-minima plateaus.

Since in the parallel implementation, due to the domain decomposition, a plateau may extend over more than one subdomain, step (1) solely does not produce either a unique label, in the case of a plateau of minima (processors use disjoint ranges of labels) (see the plateau of graylevel 7 in Fig. 1(a)), or the lower distance is inaccurate because lower brims of the plateau exist in remote subdomains. Moreover, in the latter case, when such lower brims are located only remotely, the part of the plateau analyzed locally will be wrongly classified as a minima plateau (see the plateau of altitude 8 which has two NARM pixels in the top left subimage and minima labelled pixels in the other subimages).

Therefore, further steps are performed to update the partial results of step (1) and produce global values.

At step (3.2), the *Send Receive* routine combines in one call sending of the edge $E(o_i,o_j)$ and $E(d_i,d_j)$ to processor P_j and receiving of labels and distances in $B(o_i,o_j')$ and $B(d_i,d_j')$ from another processor P_j'. A Message Passing Interface (MPI) function [20] is used, which actually shifts values across the regular grid topology of processors.

At step (3.3), parts of the same plateau from different subimages are merged by adjusting first the label and lower distance of pixels in each subimage edge based on the non-local neighbourhood replicated by message passing in the associated boundaries. Thus, if neighbouring pixels pertain to the same minima plateau, but have different labels, relabeling with the minimal label is performed. For example in Fig. 5(a), pixels (3,4) and (3,5) of altitude 7 labelled 2 and 11, respectively; consequently pixel (3,5) will be relabelled 2. Furthermore, the plateau of altitude 8 has been classified as minima in three subimages and, at the same time, it has NARM pixels in the top left subimage, pixel (0,4) and (1,4). Consequently, the latter pixels become outer pixels in the top left subimage, and hence their lower distance equals 1, while pixel (0,5) resets its label to NARM and sets its lower distance to 2. Finally, in other situations corrections of the lower distance inside plateaus of non-minima are performed, such that the distances are computed according to the entire set of distributed outer pixels. Therefore, for each edge pixel p, the lower distance is update based on the values of its neighbours in the extension area: $d(p)= \min_{r \in NG(p)\cap B(oi,oj)}\{d(p),d(r)+1|f_i(r)=f_i(p)\}$. Moreover, if $o_i(r)=$ NARM and $o_i(p)\neq$ NARM, then $o_i(p)\leftarrow$ NARM. Changes in label and/or distance are then propagated in every subdomain by a breadth-first scan at step (3.4). Whenever modifications in the distance subimage occur, the value of λ_i is also updated. Modifications in both distance and label are marked by setting the variable

$state_i$ to ON. If the merging operation leaves all pixels unchanged, $state_i$ remains unchanged, too.

Based on all local states, the moment of termination of the loop (3) is detected at step (3.5). The new state is computed by a global OR reduction operation such that the result is available at every processor: $state$=$state_0$ OR $state_1$ OR … OR $state_{P-1}$ (see Ref. [20]). Consequently, if $state$ is ON in at least one processor, all processors keep exchanging messages; otherwise, the computation has stabilized and the processors stop intercommunicating. The global images of labels and lower distances inside non-minima plateaus can be observed in Fig. 6

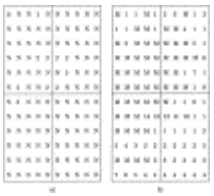
 Full-size image (18K)

Fig. 6.

(a) Global labels for minima plateaus; (b) global lower distances for non-minima plateaus.

The global value $\lambda = \max_{0 \leqslant i<P}\{\lambda_i\}$ is computed by a global reduction operation at step (4), using the MAX operator [20], such that the value of λ will be available at every processor (in our example λ=15). Each processor P_i applies next the rule to transform its subimage f_i into a lower-complete version l_i based on the lower distance subimage d_i: $l_i(p)=\lambda \times f_i(p)+d_i'(p)$, where $d_i'(p)=0$ if $o_i(p)<$ NARM, $d_i'(p)=d_i(p)$ if $d_i(p)<$ MAX_DIST and $o_i(p)=$ NARM, otherwise $d_i'(p)=1$. The values in l_i, $0 \leqslant i<4$ can be observed in Fig. 4(b). In the following, the algorithm acts on the graph representation of the image. Pixels within the image are vertices in the graph, and there is a directed arc from a pixel q to a

neighbouring pixel p, if $l(q)>l(p)$, and $l(p)=\min\{l(t)|t\in N_G(q)\}$ (p is the steepest neighbour of q). The directed graph thus defined is acyclic (it is not possible to return to the same point following only descending paths) and its arrows actually coincide with the ones in Fig. 1(a), but imposed on the lower-complete image in Fig. 4(b). Therefore, the graph is a forest whose trees contain non-minima as internal nodes and minima which have non-minima in their neighbourhood as leaves (Parallel watershed transformation algorithm for image segmentation, 2009) .

CONCLUSION:

According to the various referred sources, we come to know that the present-day image analysis algorithms incorporate a watershed algorithm as an important component. In this report, a improved technique of rain falling simulation watershed algorithm is described. The proposed algorithm exhibits equivalent performance at reduced computational complexity compared to the conventional watershed algorithm. The complexity of the proposed algorithm has been analysed in detail. Moreover, a quantitative measure of accuracy of the segmentation results produced on various images by the algorithm has been provided. Apart from demonstrating its satisfactory performance on a number of images, the report also presents a description about the working of the *Meyer's* algorithm, and the *proposed algorithm* from the given figure. Now, after comparing the Meyer's algorithm and proposed algorithm for the tennis image, we can certainly say that image segmentation as a *watershed algorithm* plays an important role in the image processing (Moga, 2009). Also, one of the parallel watershed algorithm was also presented here for image segmentation, which is detecting and labelling of the minima pixels. This means parallel watershed algorithms are quite efficient but also time-consuming.

REFERENCES:

- Moga, A. (2009, May). Parallel watershed algorithm based on sequential scanning, 1. retrieved from Google.
- Moga, A. (2009, May). An efficient watershed segmentation algorithm suitable for parallel implementation , 1. retrieved from Google.
- Rambabu, C. (2009, May). Flooding-based watershed algorithm, 1-2, 6, 11. retrieved from Google.
- Parallel watershed transformation algorithms for image segmentation. (2009). Retrieved May 21, 2009, from http://www.sciencedirect.com.