Global operations in SIMD cellular processor arrays employing functional asynchronism

Alexey Lopich
Piotr Dudek
School of Electrical and Electronic Engineering
The University of Manchester, UK;
E-mail: a.lopich@postgrad.manchester.ac.uk
p.dudek@manchester.ac.uk

Abstract

We present a new approach to execution of global image processing operations on massively parallel cellular processor arrays. Combining conventional synchronous processing with simple asynchronous propagations we achieve performance increase on global operations without additional hardware cost. By the example of watershed transformation we demonstrate the benefits of mixed synchronous/ asynchronous approach. In addition, we investigate asynchronous grey-scale data processing and its applications in image processing.

1. Introduction

Current development of pixel-parallel cellular processor arrays dedicated to image processing applications [1, 2] has raised an interest in adopting existing image processing algorithms to parallel architectures. The majority of image pre-processing routines have a straight-forward parallel implementation because operations based on local neighbourhoods are fundamental to low- and medium-level processing [3]. Nevertheless, the ability to perform local operations is necessary but not sufficient requirement for computer vision applications, since there are a number of image processing algorithms, such as segmentation, object reconstruction and recognition, morphological operators, etc. that involve global data-flow across pixel network. The difficulty of implementing global operations on a sequential architectures and the cost of the iterative data-flow has led researches to look for alternatives to SIMD approach [4-6]. To improve the performance of global operations that contain some degree of data-parallelism additional functionality has to be introduced into the architecture of elementary processing cell. A completely asynchronous approach to execution of global algorithms usually restricts a hardware implementation to a specific application and thus such devices have a lack of versatility [7, 8] . At the same time the SIMD paradigm has been proven to be efficient for local convolution operations. Retaining advantages from both approaches we were motivated to create a general-purpose solution so that a variety of algorithms (not only low-level processing) can be implemented while keeping the size of processing element reasonably small. We designed an asynchronous/synchronous processor array (ASPA) that can operate in both discrete- and continuous-time domains [9].

In this paper we demonstrate how this general purpose cellular processor array can be efficiently used for global image processing operations, exploiting asynchronism at functional and architectural levels and thus achieving significant performance increase together with optimised power consumption. Endowing processing elements with simple functional blocks we enable flexible control of the pixel network topology.

In the next section we briefly describe the ASPA [9] architecture. Sections 3 and 4 provide details on basic local and global operations. In section 5 we present a new implementation of watershed transformation algorithm on parallel architecture, following with the description of global data transferring and processing.

2. Architecture

The schematic diagram of the ASPA architecture is presented in Figure 1. Processing elements are placed onto a 4-connected rectangular grid. The interconnection between pixels allows exchanging both grey-scale and binary data. All the processors execute the same instruction word (IW) issued by central controller, while some degree of autonomy is provided by local 'flag' indicators so that conditional branching is enabled. By applying specific IWs it is possible to reconfigure the ASPA for continuous-time operation so that it will behave as a combinatorial circuit. The ASPA supports flexible pixel addressing

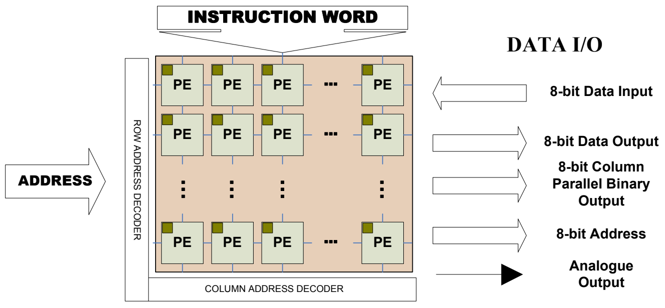

Figure 1: ASPA architecture

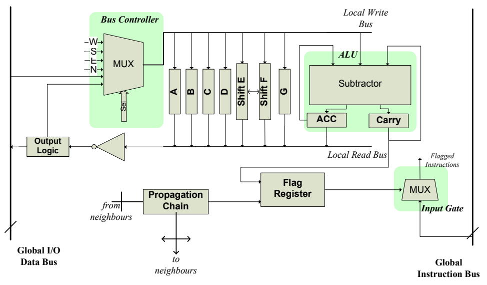

Figure 2: Processing Cell

by row and column selection. The readout circuit allows outputting 8 values of flag indicators in column-parallel manner, 8-bit pixel value or 8-bit address for fast search algorithms.

Every PE (Figure 2) is a simple digital microprocessor with arithmetic and logic unit (ALU) and general purpose registers (GPRs). As a functional block PE operates as a datapath with register transfer operations, controlled by IW. Each PE contains seven 8-bit GPRs and an accumulator in the ALU so that it is capable of storing 64 bits of data. In addition to that it can store two flag values: carry (**C**) and propagation flag (**P**).

Thanks to a variety of shift operations provided by some GPRs it is possible to perform shift-and-add multiplication and shift-and-subtract division. ALU consist of bit-serial subtractor and thus can execute both addition and subtraction without additional hardware costs. The result of ALU operation is stored in 8-bit accumulator. There are two main local buses: local read bus (LRB) and local write bus (LWB). A bus controller (BC) multiplexes external and internal data to the LWB so that it is possible to execute local register transfer operations, data transfer between distant pixels and global data transfers. The BC is implemented as an OR gate, which facilitates asynchronous processing while transferring grey-scale data.

While having a synchronous digital architecture, the PE is capable of executing certain operations in an asynchronous manner, thus achieving significant performance increase when performing global operations. A functional block called Propagation Chain (PC) provides a simple tool for binary trigger-wave propagations across the entire array, controlled by flag and mask indicators. A detailed description of the architecture can be found in [9].

3. Local operation

Current design provides all the basic operations for local operators (such as convolution by 3x3 kernel): arithmetic and logical operations, bi-directional shift, access to the data of four local neighbours. It is also possible to chain several PEs so that data transfer between distant pixels (that exceed the scope of local neighbourhood) is performed in a single step. Grey-scale arithmetic operations contribute the major part of overall operations because of the bit-serial nature of the ALU. Therefore it is essential to decompose local operators so that the number of grey-scale operations is minimised. The basic Sobel edge detection with two kernels is illustrated in Figure 3. It has required 12 grey-scale arithmetic operations, 7 conditional branching and 8 register transfer operations. Implemented in 0.35 µm CMOS technology and operating at 300MHz clock frequency ASPA was capable of executing this operation in 1.3 µs (SPICE simulation results). In an identical way other local image pre-processing tasks such as smoothing, sharpening, averaging, median filtering and basic morphological routines (binary and grey-scale dilation and erosion) can be executed on the presented architecture.

4. Trigger-wave propagations

A binary trigger-wave propagation is a basic global operation that can be used for geodesic reconstructions, hole filling and as part of more complex algorithms. Although the binary propagation is easy to implement as an iterative process employing the SIMD paradigm, such implementation is inefficient in terms of overall performance and power consumption. The number of required iterations without prior knowledge about the image should cover the 'worst case' (possibly exceeding the dimension of the array).

Operation principles of the propagation chain are fully described in [10]. As a functional block, PC is described by the following equation:

$$y=(y_N \cdot sel_N + y_S \cdot sel_S + y_W \cdot sel_W + y_E \cdot sel_E) \cdot u + u \cdot pg \qquad (1)$$

where y_N, y_S, y_W and y_E represent the present state of the nearest neighbours, sel_N, sel_S, sel_W and sel_E are masking signals, *u* is a propagation space and signal *pg* initiates propagation. The average propagation speed is

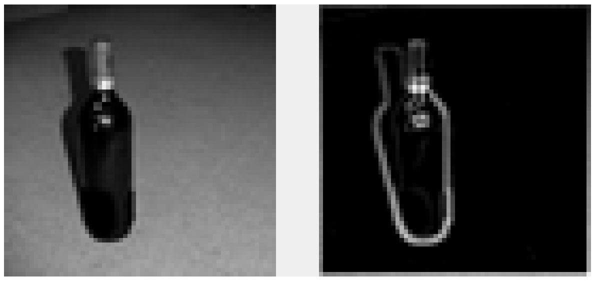

Figure 3: Sobel edge detection on ASPA

approximately 0.25 ns per pixel (Figure 4) consuming 0.4 pJ per cell. The equivalent synchronous implementation on ASPA in iterative manner would require operation at 10 GHz (two iterations per cell), providing over 163 trillion operations per second on 128x128 array, which corresponds to a maximum throughput of $24 \cdot 10^4$ fps. The application where such frame rate is required is hardly imaginable, but if we think about wave-propagation as a basic operation extensively used in more complex real-time computer vision, than we can minimize the overall processing time. The next section demonstrates an extensive use of binary propagations.

5. Watershed transformation

In order to demonstrate benefits of global asynchronous operations we have implemented a watershed transformation [11] on the presented architecture. Watershed transformation is one of the examples of computationally expensive routine which comprises a number of morphological operations (grey-scale reconstruction, regional minima extraction, etc.).

First we give some basic theoretical definitions

5.1 Basic Notions

Let us consider a 2D image as a planar graph $G=(I,U,f)$ where $I \subseteq Z^2$ is a coordinate grid and $U \subseteq Z^2 \times Z^2$ defines a connectivity between pixels. We label 4-connected and 8-connected grids by U_4 and U_8 correspondingly. So two pixels p and q such as $p,q \subset P$ are called neighbours if $(p,q) \subset U$. The neighbourhood of pixel p is defined as $N_U(p)=\{q:(q,p) \subset U\}$. The grey-level function $f:I \rightarrow Z$ can be interpreted as a pixels altitude.

Let us define the slope between two pixels as:

$$slope(p,q)=f(p)-f(q),\ \forall p \in I,\ \forall q \in N_U(p) \qquad (2)$$

Then the lowest neighbourhood is the set of neighbour pixels with maximum slope, provided they have a lower altitude:

$$LS(p) = \max_{q \in N_u(p), f(q) \le f(p)} (f(p) - f(q)) \qquad (3)$$

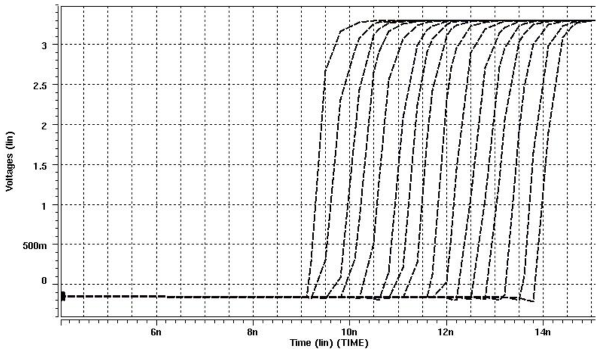

Figure 4: Simulated voltages at node *y* in a 16x16 array. Following the pre-charge phase the propagation was triggered at 9 ns at the pixel in the first column and first row.

We define the pixel p as local minimum if $h(p) \le h(q)$, $\forall q \in N_{U_8}(p)$. The set of local minima (our points of interest) is denoted as $\min_I$. The catchment basin $CB(m_i)$ of a minimum $m_i \in \min_I$ is defined as the set of pixels $x \in I$ that are topographically closer to m_i than to any other regional minimum m_j. Indices i and j belong to an index set $\mathrm{Ind}_{\min}$ of connected sets of local minima. Then the watershed of the given image $WL(I)$ is a set of points that does not belong to any catchment basin.

Considering equation (1) let us refer to $P_I(p_{ij})$ as to a set of pixels, whose propagation flag **P** is set after binary propagation operation executed on image I with the origin in pixel p_{ij}.

5.2 Algorithm Implementation

Our algorithm is based on asynchronous relaxation of "Hill-Climbing" algorithm which has been explored in a number of research works [6, 7]. Our implementation consists of two parts: initialisation and flooding.

Initialisation step is used for identifying local minima ($\min_I$) and determine the 'steepest neighbour' mask for every PE. As it was shown in section 4 every PE can locally control the direction of propagation. It means that it is possible to mask certain neighbours so that propagation signals issued by these neighbours will be ignored. Therefore for every processing element p_{ij} we define a local propagation mask, so that it can be triggered only by neighbours that belong to LS(p_{ij}).

Initialisation will consist of the following steps:

- *For every pixel p define LS(p). If LS(p)≠0 then assign appropriate bits of dedicated GPR (let say register D) to logic '1'.*
- *Identify if the pixel is a local minimum:* $min_I=min_I \cup \{p_{ij}\} : \nexists q \in N_{U_8}(p_{ij}), f(q)<(p_{ij})$.

Marking a pixel as a local minimum is done by setting a dedicated bit (say D<8>) to logic '0'. In case it is not a local minimum this bit is set to '1'.

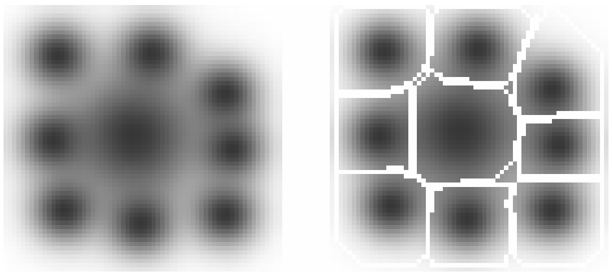

Figure 5: Watershed transformation of 64x64 image (8-bit grey-scale) with 9 catchment basins.

The next step, flooding, forms watershed lines. The flooding is performed sequentially basin by basin. In order to flood all the basins simultaneously like in [7] it would be necessary to label them individually and then in every pixel perform minimum function asynchronously. Such an approach benefits from complete parallelism, but requires additional time for labelling the markers, additional hardware for real-time minimum computation and increased the number of interconnections. Using sequential flooding doesn't require labelling and is identical to a simple trigger wave propagation described in section 4. This step consists of the following operations:

1. *Select any pixel* $p_{ij} \in min_I$;
2. *Initiate trigger wave propagation with the source in the chosen pixel* p_{ij};
3. *For any propagated pixel:*
   a. *Remove the local minimum mark for all the pixels activated by current propagation:* $min_I = min_I \setminus \{q\}$, $\forall q \in P_I(p_{ij})$.
   b. *Remove the watershed mark for all the pixels activated by current propagation:* $WL(I) = WL(I) \setminus \{q\}$, $\forall q \in P_I(p_{ij})$.
   c. *Mark every activated pixel that has a non-activated neighbour as a watershed line:* $WL(I) = WL(i) \cup \{q\}$, $\forall q \in P_I(p_{ij})$: $\exists q_n \in N_{U'}(q)$, $q_n \notin P_I(p_{ij})$.

Current approach implies possibility that some flooding can be initiated from a non-minimum plateau which will produce incorrect watershed line. However when the real regional minimum marker will be selected the incorrect part of watershed line achieved by flooding from non-minimum plateau will be erased (step 3.b) and the correct result will be achieved. The initialisation and step 3 of flooding procedure are performed synchronously since all pixels operate with local data. In order to select a pixel that is marked as a local minimum we use a row/column address extraction circuitry so that this step requires a fixed amount of operations independently from the image size. It should be emphasized that definition of local minimum given in section 5.1 will also label so called plateaux [7]. Therefore overall performance can be affected because a number of floodings can be initiated from a non-minimum plateau.

There are several ways to anticipate this problem. The first involves preliminary pre-processing (filtering, smoothing, morphological operators) in order to smooth shapes of slopes and thus reduce number of 'undesired' markers. The success of this operation always depends on the original image, since for some images with low dynamic range important information can be lost. The second involves grey-scale reconstruction that can be decomposed to D_I binary reconstructions, where D_I is a dynamic range of the input image. Although this procedure will extract 'real' local minima, but for a wide range images it can be computationally expensive as it will require the number of iterations, proportional to D_I. Another approach involves marker selection process that starts from pixels with lowest brightness, so that non-minimum plateaux will be processed before their marker can be chosen.

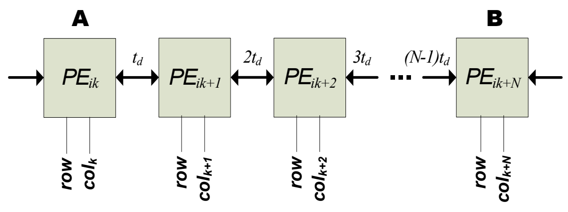

Figure 6: Asynchronous data transferring.

It should be noted that, without additional pre-processing, using watershed transformation directly on 'real' images often results in "oversegmentation". Such pre-processing usually involves filtering with opening by reconstruction, edge detection, background subtraction and other operations that have a straightforward synchronous implementation on a CPAs' because they require only local neighbourhood information.

At implementation level, the watershed transformation of a 64x64 image presented in Figure 5 was achieved with: 478 primitive instructions (one clock cycle per instruction) for initialisation step; 9 global instructions, 18 global readout instructions and 261 primitive instructions during flooding. Simulating the circuit at 300MHz frequency the total processing time was approximately 3 µs. Apart from storing the actual grey-scale value of pixel the whole processes requires only 6 bits of memory (4 for storing neighbourhood mask, 1 for watershed mark, 1 for local minima mark).

6. Global data transferring

Apart from binary trigger-wave propagations some global operations involve transferring and processing of grey-scale data: data-transfer between distant pixels, distance transform, transferring of a pattern image inside an original image as in Hough transform, etc. Proposed design facilitates these operations in such a

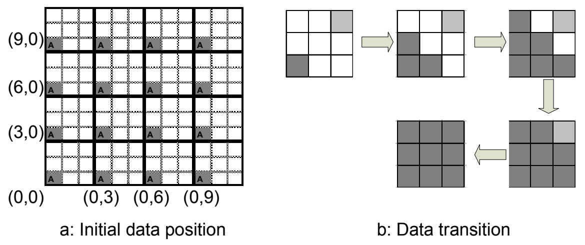

Figure 7: Data transferring within 3x3 blocks

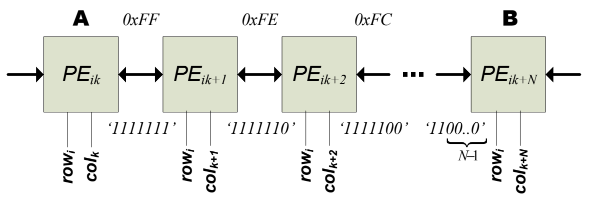

Figure 8: Asynchronous distance transformation

way that global transfers are akin to local read-write operations.

6.1 Data Transferring

Assume it is necessary to transfer grey-scale data from register D of pixel A to register D of pixel B as depicted in Figure 6. The transfer has to be performed in a way that the data from PE_{ik-1} and PE_{ik+N} should not pass to their east neighbours. Synchronous implementation would require 2N iterations (2 iterations per pixel-to-pixel transfer). Our architecture allows performing such transform in asynchronous manner by executing the following steps:

1. *Mask PE_{ik} (so that the following operations will be ignored by PE_{ik});*
2. *Load register D with 0x00;*
3. *Remove mask; Mask PE_{ik-1}, PE_{ik+N};*
4. *Read and load register D with the value from west neighbour and LRB;*
5. *Remove mask;*
6. *Load register D with value from LWB.*

By the end of sixth step the value from **A** appears in register D of PE_{ik+N}. It should be noted that step 4 implies asynchronous data propagation through the array and the time necessary to accomplish this step is equal to Nt_d, where t_d is propagation delay in processing cell. Generally t_d is much smaller that the instruction supply time t_I. The total number of required iterations in this case is $9+N\times t_d/t_I$. It is therefore clear that the acceleration of the processes depends on the ratio t_d/t_I, which is always less than one.

To transfer data within a 2D block of pixels similar steps have to be taken. In case when an image is split into 3x3 blocks (Figure 7) and it is necessary to distribute a value from bottom left corner PE within each block, the sequence of steps will be similar to the above. In step 3 it is necessary to mask 3n+2 columns and rows (n=1,2,…). This operation can be used to propagate some characteristic value (e.g. region mean value) within a fixed block.

6.2 Distance transform

Utilising the features of BC and shift registers it is possible to perform simple processing while transferring data, in particular it is possible to perform a simplified version of increment operation.

Let us consider the value 0xFF being stored in one of two shift registers presented in PE. A shift left operation will translate this value to 0xFE or '11111110' in binary code. By applying this operation *m* times ($m<9$) we will achieve *m* zeroes in least significant bits. Let us consider the chain of processing elements presented in section 2 with the shift register F of PE **A** loaded with 0xFF. Making modification in the algorithm so that in step 4 we will execute left-shift (carry-in is logical '0') we will perform simple distance transform (Figure 8). An additional step is required to interpret the value into a binary number. If we achieve 0x00 it indicates that processing cell **B** is out of range of 8-pixel radius.

Taking into account that BC executes a bitwise OR operation [9] it is easy to notice that this operation provides a minimum function in the context of operated numbers. Let us consider the processing of a binary image. If we assign 0xFF to all the background pixels, 0x00 to foreground pixels and perform the operation described above well will achieve a distance map using city block distance within 8 pixel distant range. If necessary the full distance map can be calculated applying the above procedure [R/8]+1 times where R is a maximum object radius.

Distance maps are widely used for image representation. Among applications are skeletonization, object feature extraction and other algorithms used for reducing the image data.

The asynchronous algorithm described in previous section is suitable for transferring a value of only one pixel in the fixed space (block, row, column). If we try to transfer the value of some register (say E) from PE_{ik} and PE_{ik+1} to PE_{ik+N-1} and PE_{ik+N} correspondingly, at the same time, we will obtain a result being a bitwise OR operation of PE_{ik}.E and PE_{ik+1}.E (which can be used for calculation of global minima in a similar way as distance transform). Yet, a parallel translation of the entire image by more than one pixel can be done if the image is binary. Assume we have a binary image loaded to a least significant bit of the shift register E. Then by following the steps required for distance transform we can translate the original image by up to 8 pixels in one of the four directions (all the translations, i.e. by 1,2,…,8 pixels are executed in parallel for all pixels, so that 8 binary images are obtained as a result). To extract the shifted image the appropriate bit of the shift register have to be examined (e.g. for image shifted by 4 pixels it is necessary to check the fourth bit of the shift register).

7. Conclusions

In this work we have shown that early vision processing on CPA can significantly benefit from functional asynchronism. In fact, this way of processing introduces new possibilities for vision chip applications. The ability to perform global operations in a single iteration manner makes it possible to execute more advanced algorithms than simple convolution filters.

The SIMD approach is very effective for low-level image processing algorithms, because their complexity does not depend on image size. But when the algorithm implies extraction of specific data, for example objects of interest in segmentation routines, global operations are unavoidable. It turns out that computation of such problems in iterative manner on SIMD array is very inefficient, and can sometimes be inferior to using a general-purpose RISC microprocessor. This drawback of inefficient processing of global operations can be minimised by introducing asynchronous processing into CPAs. Indeed asynchronous processing can provide impressive results in computationally expensive algorithms [6, 8], but usually at high hardware cost. On the other hand basic but frequently used global operations such as binary trigger-wave propagations, asynchronous data transferring and simple processing do not require any extra hardware and can be accomplished on general-purpose processing cells. Because propagation delay through pixel is much smaller than the clock period, global operations can usually be executed within few clock cycles.

By the example of watershed transformation we have shown that appropriate decomposition into simple global propagations and local operations (local minimum) can significantly improve overall throughput. The benefits of asynchronism can also be observed in distance transform operations. Not only the data is transferred at high speed but a simple processing is performed on grey-scale values.

Presented algorithms have been implemented and simulated (SPICE) on the ASPA, designed in 0.35 µm CMOS technology. The operational speed of the chip is 300MHz and computational times for algorithms implementation were estimated at this speed. As compared to synchronous realisation, asynchronous operational performance is estimated to be 2900 MOPS per cell for binary propagations. The power consumption during binary-propagations is 0.4 pJ per cell. Such impressive performance characteristics mean that simple global propagations can be used as basic operations in more complex algorithms.

We believe that mixed synchronous/asynchronous operation can enhance overall performance of CPAs in complex applications. Adhering to strict area constrains, computational capabilities of individual processing cell can be extended with simple hardware asynchronism.

8. Acknowledgment

This work has been supported by the EPSRC grant number EP/D029759.

References

1. Eklund, J.-E., Svensson, C., Astrom, A., *VLSI Implementation of a Focal Plane Image Processor - A Realization of the Near-Sensor Image Processing Concept.* IEEE Tr.VLSI Systems, 1996.4(3): p. 322-335.
2. Dudek, P., Hicks, P.J., *A general-purpose processor-per-pixel analog SIMD vision chip.* IEEE Tran. Circuits and Systems I: Fundamental Theory and Applications, 2005. 52(1):p.13-20.
3. Serra, J., *Image Processing and Mathematical Morphology.* 1982, London New York: Academic Press.
4. Robin, F., Renaudin, M., Privat, G., *Functionally asynchronous VLSI cellular array for morphological filtering of images.*Trait.du Signal,1997.14(6):p.655-664.
5. Galilee, B., Mamalet, F., *et al. Watershed parallel algorithm for asynchronous processors array*. in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME), 26-29 Aug.* 2002. Lausanne, Switzerland: IEEE. p.793-796
6. Manzanera, A., *Morphological Segmentation on The Programmable Retina: Towards Mixed Synchronous/ Asynchronous Algorithms.*Proc.of ISMM2002,p.389-399
7. Noguet, D. *A massively parallel implementation of the watershed based on cellular automata*. in *Proc. International Conference on Application - Specific Systems, Architectures and Processors, 1997*. p.42-52
8. Lopich, A., Dudek, P. *Architecture of Asynchronous Cellular Processor Array for Image Skeletonization*. in *ECCTD'05*. 2005. Cork, Ireland.
9. Lopich, A., Dudek, P. *Architecture of a VLSI cellular processor array for synchronous/asynchronous image processing*. in Proc. of ISCAS2006. p.3618-3621.
10. Dudek, P., *An Asynchronous Cellular Logic Network for Trigger-Wave Image Processing on Fine-Grain Massively Parallel Arrays.* IEEE Transactions on Circuits and Systems II (accepted); IEEExplore: DOI 10.1109/TCSII.2006.869916.
11. Beucher, S. *Segmentation tools in mathematical morphology*. 1990. San Diego, CA, USA: Publ. by Int. Soc. for Optical Engineering, Bellingham, WA, USA. p.70-84