Architecture of a VLSI cellular processor array for synchronous/asynchronous image processing

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*— This paper describes a new architecture for a cellular processor array integrated circuit, which operates in both discrete- and continuous-time domains. Asynchronous propagation networks, enabling trigger-wave operations, distance transform calculation, and long-distance inter-processor communication, are embedded in an SIMD processor array. The proposed approach results in an architecture that is efficient in implementing both local and global image processing algorithms.**

I. INTRODUCTION

Massively parallel, fine-grain processor arrays can provide a powerful solution for image pre-processing applications [1, 2]. Due to the regularity of image data, the pixel-per-processor approach is particularly efficient in low-level image processing applications, providing high performance, small area and low power consumption. This approach has led to the development of so-called 'vision chips', which combine a processing element (PE) with a photodetector [3, 4].

A conventional hardware realisation of a fine-grain processor array implies that PEs are controlled according to the Single Instruction Multiple Data (SIMD) paradigm and operate in a synchronous manner. Such an approach is very effective in many image pre-processing applications. However, synchronous operation is efficient only for local operators, such as convolutions with 3x3 kernels, where the result pixel value is a function of the neighbouring pixels and the current pixel value. There are, however, a number of global operations which involve a data-flow between pixels. This data-flow can be represented by a wave-propagation process in which pixel operations are triggered by a change of state (availability of new data) signalled by their neighbours. Such processes can be very efficiently employed in many image processing algorithms, such as morphological operations, e.g. object reconstruction, watershed segmentation, skeletonization, distance transform, centroid calculation etc. The execution of such a process on a synchronous system is performed in an iterative manner (one-pixel distance per iteration). On the other hand, asynchronous processing can provide significant performance increase, since the operation time is only determined by the global dataflow delays, and is not increased by side effects of synchronisation. Moreover, it reduces the power consumption, since fewer instruction-cycles per image frame are needed. In general, an asynchronous realisation provides a "single-instruction" implementation of the wave-propagation operation. Several architectures implementing asynchronous image processing have been proposed [5, 6], but while these solutions provide a significant performance increase compared to synchronous analogues, their practical application is limited. There are also a few general-purpose 'smart-sensor' systems that include some form of global, continuous-time operation [4, 7, 8].

In this paper we introduce a novel VLSI cellular processor array architecture, with a mixed asynchronous/synchronous operation. Each processing cell of the Asynchronous-Synchronous Processor Array (ASPA) has a universal synchronous digital architecture, and also enables generation and processing of asynchronous trigger-waves, continuous-time execution of a distance transform, global feature extraction, etc. When the ASPA operates synchronously, it resembles a SIMD array since every processing cell executes the same instruction. However, it is possible to reconfigure the circuits in such a way that the ASPA will behave like a combinatorial circuit. Thanks to this approach, both local and global operations are effectively executed on the ASPA.

II. ARCHITECTURE

The proposed chip architecture (Fig. 1a) contains a 2-D array of locally interconnected asynchronous-synchronous processing elements, placed on a rectangular grid. Every PE is connected to its four neighbours, which allows data exchange in both discrete- and continuous-time domains. The Instruction Word (IW) is broadcast to every PE, so that all PEs receive identical instructions and execute them in parallel (SIMD mode) while some degree of local autonomy is provided by 'flag' indicators. The IWs and local flags are also used to configure the array for continuous-time operation. Every instruction in the PE is interpreted as an opening of a transmission path between two or more memory elements so it is a register transfer operation. The array also supports random access, column and row parallel I/O operations.

A. PE Operation

Every PE is a simple microprocessor (Fig. 1b) with an Arithmetic and Logic Unit (ALU) and registers, providing flexible functionality and programmability. At the same time, it can perform global asynchronous operations with data-flow between PEs by adjusting local read/write operations. The PE contains general purpose registers (GPR) with some functioning as shift registers that are based on multiplexing data before register loading.

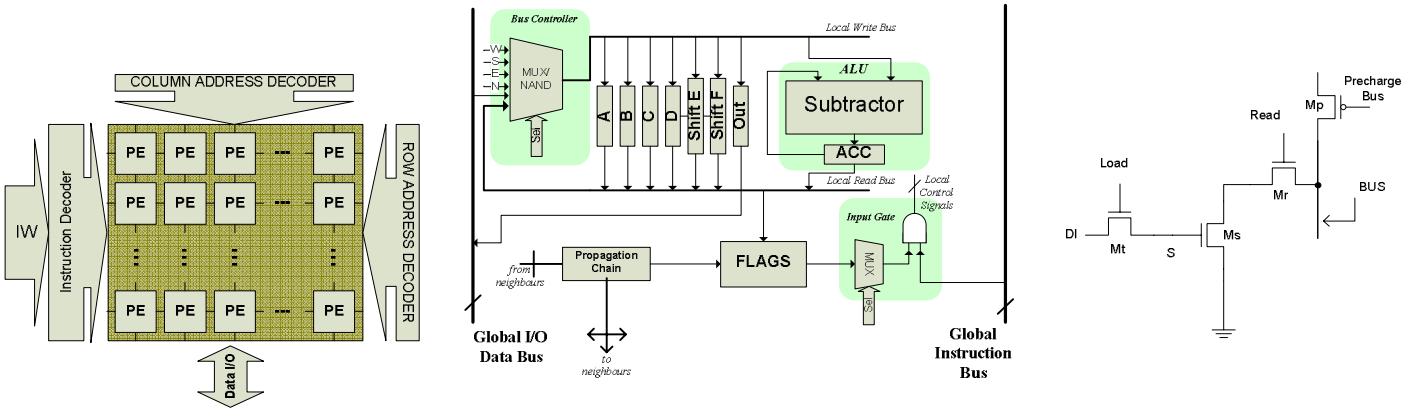


Figure 1: The chip architecture: (a) ASPA architecture; (b) PE architecture; (c) Dynamic memory element

Bi-directional shifting provides a flexible tool for shift-and-add multiplication and shift-and-subtract division operations. All the memory is based on dynamic latches (Fig.1c) with a shared pre-charged bus (i.e. level sensitive) so there is no edge-sensitive logic within a PE. A simple ALU and a FLAGS register (Zero, Carry, Propagation) enable a range of arithmetic and logic operations to be processed. By using a subtractor in the ALU we can execute both subtraction and addition without any additional hardware cost (if an adder was used then additional inversion would be required). The result of an arithmetic operation is stored in the accumulator (ACC). The inputs of the ALU are connected to the local write bus (LWB) and the ACC.

A single PE operation is executed as follows: 1) the data from a memory cell is read to the local read bus (LRB); 2) the local or external data, controlled by the Bus Controller (BC), is forwarded to the LWB; 3) the data from the LWB is loaded to a local memory.

B. Binary Propagations

The propagation chain (Fig. 2a) block is used for performing global trigger-wave propagations across the array [9]. The E_latch is used for storing the marker. Once the stored value is '0' a PE will not allow further signals to propagate through and its output will remain '0'. The P_Latch is assigned to store the propagation result for further processing. The minimum propagation routine consists of the following steps: 1) define the propagation space; 2) define the propagation start points; 3) set the 'start' bit to '1'; 4) load the result into P_Latch, which can then be used as a flag.

C. Local and Asynchronous Communication

The communication between PEs is organised via the shared LRB. Every PE can access the LRB of its four neighbours, so the result of any local operation can be directly transferred to the neighbourhood, eliminating intermediate local operations (i.e. both synchronous and continuous-time transfers are possible). The data in the LWB is managed by the BC. Essentially the BC is a multiplexer, which enables the set of following transfer operations: GPR → GPR, Global I/O Bus → GPR, Neighbour → GPR. Apart from multiplexing, the BC performs a NAND function, which is used to continuously compute a minimum value among nearest neighbours. The circuit for a single bit calculation is shown in (Fig. 2b). This 'minimum' function is used in continuous-time distance transform operation, as explained in the following section.

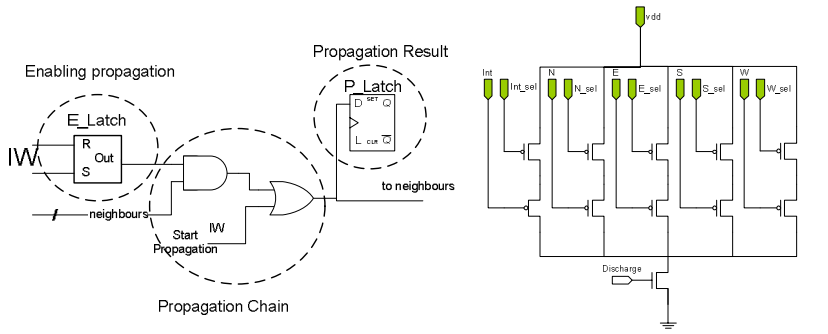
III. OPERATION

A. Distance transform

There are a number of algorithms that involve a distance transform: skeletonization, small object location, Hausdorff distance calculation (pattern matching, object recognition), Hough transform, etc. The ASPA provides fast and efficient implementation of distance transform, via asynchronous operation.

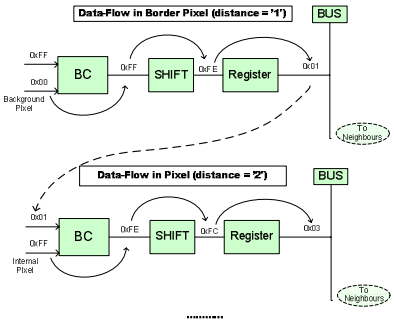
Consider the situation when the distance is calculated between object pixels and the background. Initially a shift register is loaded with 0xFF (background) or 0x00 (object). The shift register is set for "left shift" operation with carry-in signal set to '0'. The multiplexing unit of the BC is set to provide a neighbour value instead of the value from the LRB.

Let us consider the processing of a border pixel. After loading initial values to PEs the register 'E' is being read, therefore inverted values appear on the LRB accessible by neighbours. After that, the 'Load' signal for register 'E' in all object pixels goes high, configuring the register as a transition gate (with the only possible transition '0' to '1'). In this case the left-shifted value calculated by the BC passes through the register 'E' to the LRB and correspondingly to its nearest neighbours (Fig. 3) and so on. Since distance values are represented according to Fig. 4a, the NAND operator in the BC calculates the minimum distance value and left-shift performs an increment function. As the result of this propagation, we will achieve the exact Manhattan distances for pixels within the eight pixel distance range (Fig. 4b). This process is robust and is not sensitive to any non-uniformity of propagation velocity.



(a) (b)

Figure 2: Asynchronous Logic: (a) Propagation Chain; (b) Bus Controller with NAND operator.



(c)

Figure 3: Distance transform data-flow.

In order to calculate distances to all the object pixels, it is necessary to perform [R/8]+1 iterations, where R is the object radius. A single distance transformation requires 5 instructions (subject to initial conditions).

It is also possible to calculate the global minimum and to transfer data between distant pixels with a single instruction (a process similar to the one described above, but without shifting). In general, the architecture supports chaining pixels in various ways and performing other global operations such as those described in [7].

B. Matrix Multiplication

The advantage of using the proposed architecture for synchronous operations is demonstrated on matrix multiplication, the basis of linear discrete image transforms. The ability to perform such an operation will significantly simplify real-time compression procedures.

Assume we have an array **P**, and we need to derive a new matrix **B** as a product of **P** and another matrix **A**, i.e. **B**=**P**×**A**, where

$$B_{ij} = \sum_{k=1}^{n} P_{ik} \times A_{kj}$$

Let us refer to a pixel associated with a PE at grid location (i,j) as P_{ij}, and to a register A (B,C,..) of this PE as A_{ij} (B_{ij},C_{ij},..). Then to get the product of two matrices it is necessary to accomplish the three following steps: 1) load to A_{ij} the corresponding value of matrix $\mathbf{A}^T$; 2) perform multiplication $C_{ij}=P_{ij}\times A_{ij}$; 3) n times perform $B_{ij}=C_{ij}+C_{ij+1}$; $C_{ij}=C_{ij+1}$. As a result of this operation we will achieve the 1^{st} column of matrix **B**. In total, we have n multiplications, and n^2 additions and register–transfer operations. In the case of a transform such as DCT with 8x8 image blocks, a significant performance increase will be achieved because all the operations are performed in parallel. Moreover, with an advanced addressing mechanism (e.g. the address xxxx000 will indicate all the columns out of 128, divisible by 8) [10] it is possible to have a fixed amount of operations (for the above case only 8 multiplications) irrespectively of image size. The multiplication, based on shift-and-add operations, is implemented by 40 instructions. The calculation of a single row takes 64 instructions. So the 8x8 linear discrete transform will require 1152 instructions (including loading two matrices) for any image size, using only 3 memory elements (8-bit registers).

C. Watershed Transform

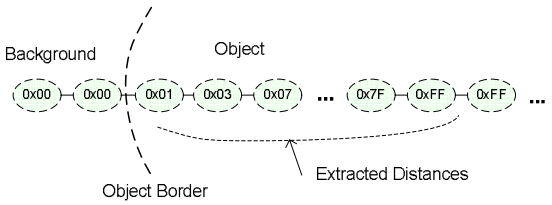
The watershed transform is an example of a complex, computationally expensive image processing algorithm that comprises different morphological operations (binary and grey-scale geodesic reconstruction, regional minima extraction, etc.). Watershed segmentation helps to extract objects of interest from the background on grey-scale images.

Essentially, watershed segmentation consists of two main procedures: basin marking and flooding. For efficient implementation we have decomposed both of them into simple wave-propagation operations. Basin marking, which is based on a grey-scale reconstruction, has been implemented as a set of binary reconstructions (254 single-step iterations). Thanks to a propagation unit in ASPA, it is possible to perform binary reconstruction over an entire image in a single-iteration. After the basin marking operation, an additional step is required. During this step it is necessary to find, for every PE, the neighbour(s) with the minimum brightness value. Then an appropriate mask is applied to the propagation chain. As a result of this operation, utilising the local autonomy feature, the propagating signal in each cell will be accepted only from the lowest neighbours. This will implement the required "drop effect", i.e. if we leave a drop of water on a physical landscape, at each point it will roll downhill along the maximum decline, finally reaching the basin's origin.

After this procedure, propagation is initiated at the basin markers (one at a time). Due to the configuration of the processing array, propagation will only spread within a single basin and along the watershed lines. After each propagation operation the border pixels will form a watershed line. By sequentially performing such operation, basin by basin, the watershed line of the entire image is obtained. By using asynchronous propagations, the number of operations involved in the flooding is reduced and is proportional to the number of basins.

Distance	Binary Representation
0	00000000
1	00000001
2	00000011
…	…
8	11111111

(a)



(b)

Figure 4: Distance Transform: (a) Distance Correspondence Table; (b) Extracted Distances.

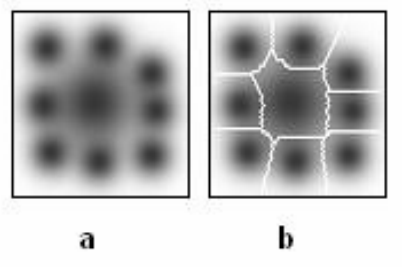


Figure 5: Watershed Transform: a) Original Image b) Transformed Image.

In our simulation experiments, the complete processing of a 64x64 image with nine areas of interest required around 2900 instructions on the ASPA. Assuming the iteration time to be 100 ns the total processing time is 2.9 μs. The simulation results are presented in Fig. 5. It is shown that the actual flooding is accomplished by a single instruction in a wave-propagating manner. The achieved results are comparable to application specific arrays [6, 11]

In general, we can see that an appropriate decomposition of complex algorithms into simple operations can provide a significant performance increase when implemented on the proposed architecture.

IV. Hardware Implementation

The presented ASPA design has been initially implemented on a Spartan3 xc3s1500 FPGA. Each PE (an 8-bit processor with 7 local registers) occupies a block of 10x10 slices. The 10x10 array design with periphery uses 74% of available resources. The propagation delay across the array is 2.612 ns. The operating frequency is 37.5 MHz. Calculation of a single row in matrix multiplication requires 80 instructions, which corresponds to 2.15 μs. We estimate that the IC implementation requires not more than 700 transistors per PE. Using a 0.35 μm 4 metal layer CMOS process the cell area should not exceed 100x100 μm^2. Hence a 128x128 array can be fabricated on a 200 mm^2 chip. Using a 90nm technology with 6 metal layers it should be possible to achieve 25x25 μm^2 per PE and 512x512 array on the same size chip.

V. Conclusions

A new architecture for a general-purpose cellular processor array has been designed and demonstrated. The ASPA design is based on a mixed asynchronous and synchronous operation. Such an approach provides an efficient solution for both local and global image processing operations. Significant performance increase can be achieved in global morphological operations, due to the ability to perform global wave-propagation operations in an asynchronous single-iteration manner. An efficient decomposition of complex algorithms into simple trigger-wave operations is introduced. The operation of the ASPA has been verified by simulations and FPGA implementation. Currently, we are working on a full-custom VLSI circuit design.

VI. Acknolegement

This work has been supported by the EPSRC. The authors would like to thank Dr. S.J.Carey for valuable remarks.

VII. References

[1] J.C.Gealow, C.G.Sodini, *Pixel-parallel image processor using logic pitch-matched to dynamic memory.* IEEE Journal of Solid-State Circuits, 1999. **34**(6): p. 831-839.

[2] M.J.B.Duff, D.M.Watson, *The cellular logic array image processor.* Computer Journal, 1977. **20**(1): p. 68-72.

[3] P.Dudek, P.J.Hicks, *A general-purpose processor-per-pixel analog SIMD vision chip.* IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications, 2005. **52**(1): p. 13-20.

[4] J.-E.Eklund, C.Svensson, A.Astrom, *VLSI Implementation of a Focal Plane Image Processor - A Realization of the Near-Sensor Image Processing Concept.* IEEE Trans. on VLSI Systems, 1996. **4**(3): p. 322-335.

[5] A.Lopich, P.Dudek, *Architecture of Asynchronous Cellular Processor Array for Image Skeletonization*. in *ECCTD'05*. 2005. Cork, Ireland.

[6] A.Manzanera, *Morphological segmentation on the programmable retina:towards mixed synchronous/asynchronous algorithms*. in *6th International Symposium on Mathematical Morphology*. 2002. Sydney, Australia.

[7] T.Komuro, S.Kagami, M.Ishikawa: *A dynamically reconfigurable SIMD processor for a vision chip.* IEEE Journal of Solid-State Circuits, 2004. **39**(1): p. 265-268.

[8] G.Linan, S.Espejo, et al, *Architectural and Basic Circuit Considerations for a Flexible 128x128 Mixed-Signal SIMD Vision Chip.* Analog Integrated Circuits and Signal Processing, 2002. **33**: p. 179-190.

[9] P.Dudek, *Fast and Efficient Implementation of Trigger-Wave Propagation on VLSI Cellular Processor Arrays*. in *CNNA2004*. 2004. Budapest.

[10] P.Dudek, *A flexible global readout architecture for an analogue SIMD vision chip*. in *ISCAS 2003. Int. Symp. on Circuits and Systems, 25-28 May 2003*. 2003. Bangkok, Thailand: IEEE.

[11] B.Galilee, F.Mamalet, M.Renaudin, P.-Y.Coulon, *Watershed parallel algorithm for asynchronous processors array*. in *ICME '02*. 26-29 Aug. 2002.