# Increasing interconnection network connectivity for reducing operator complexity in asynchronous vision systems

Valentin Gies and Thierry M. Bernard

ENSTA, 32 Bd Victor 75015, Paris, FRANCE,
contact@vgies.com,
WWW home page: http://www.ensta.fr/uer/uei/eng/index.html

**Abstract.** Due to the restriction of SIMD mode to local operations in VLSI massively parallel vision chips, using programmable connections and asynchronous communications are key ingredients to support regional computations. Asynchronism implies using combinatorial multi-input operators having an important hardware cost. To reduce it, we propose to use a connection network having a connectivity level greater than the mesh being mapped. This solution allows to use only 2-inputs asynchronous operators having a reduced hardware cost in each pixel. Examples and results will be presented on the examples of the regional sum algorithm computed over a 4-connectivity squared mesh connected with a 6-connectivity interconnection network, and the regional sum computed over a 6-connectivity squared mesh connected with a 8-connectivity interconnection network.

## 1 Introduction

An artificial retina is an image sensor with a processing element (PE) in each pixel. Such VLSI circuits are also called "vision chips" [1]. Motivated by the low power implementation of vision applications, we focus our research [2] on digital programmable artificial retinas (PAR), for which the PE is a tiny digital processor called the pixellic processor. The latter allows the on-site processing of data from the pixel or its neighbors, according to instructions provided by an external program.
The basic operating mode of a PAR is the SIMD mode (Single Instruction Multiple Data) : at a given time, the same instruction is simultaneously executed by each pixellic processor. SIMD mesh arrays for image processing were popular in the eighties as they allow the efficient implementation of local and shift-invariant operators (linear filtering, mathematical morphology, ...). But they were later abandoned due to several drawbacks. Nowadays, SIMD processing has come back into favor within commercial microprocessors in order to cope with frequency and power consumption limitations. While PARs fully benefit from the SIMD low power advantages, they are much less subject to SIMD drawbacks than the mesh arrays of the eighties. Still, the SIMD mode is only well adapted

to low-level vision.
Rather than processed images produced by low-level vision operators, a PAR should ideally output image descriptors, which can only result from higher levels of vision. These descriptors are based on regions resulting from a segmentation of the image. These regions need efficient regional operators for manipulating it. In contrast with neighbor-to-neighbor communications used in PARs for low-level vision, regional operators need to communicate between sparse and distant pixels.
Programmable neighbor-to-neighbor connections [3] allow to implement data-dependant communication networks within the SIMD framework, but with very poor synchronous performances. In the synchronous case, communication speed is limited to ”one pixel farther per clock cycle”.
Suppressing the above drawback leads to use asynchronous instead of synchronous communication. Thus PARs have to feature programmable connections and asynchronous communications and computations to efficiently handle regions.
This paper first presents some existing solutions for computing an exemplary asynchronous regional task, the ”regional sum” in a 4-connectivity network using a dedicated asynchronous adder in each pixel. Since this adder cost is prohibitive for very large scale implementations, we propose a new communication network based on 6-connectivity reducing the hardware cost of asynchronous operators by reducing their necessary inputs to the minimum possible. Algorithm for installing the communication network is presented, a hardware cost comparison is proposed.

## 2 Linear bit-serial multi-input adder

Computing a regional sum implies to collect data from all the pixels of the region. Collecting and adding these data in one chosen place implies moving data on long distances. In order to overcome this problem, data must be added locally. To do this, a possibility is to chain pixels with an adder operator inside the pixel. The operator will have to add the binary value provided by the preceding pixel, and the local value. For digital sum computation, bits have to be processed one after the other, from less significant bit to most significant bit. In this case, one also has to sum the carry stored in each pixel during the computation of the preceding bit sum. Finally, the local operator has to be an adder able to compute the sum of 3 binary inputs. The operator used is a full adder.
The principle of the global addition is explained in fig.1. This solution has been proposed and implemented by in [4] using dynamically reconfigurable chains of pixels set by external programming.

### 2.1 Sum algorithm

In each processor, the full adder inputs are connected to local binary data (internal bit and carry) and to the preceding full adder less significant output bit (usually called the sum bit). The least significant output bit is connected to the

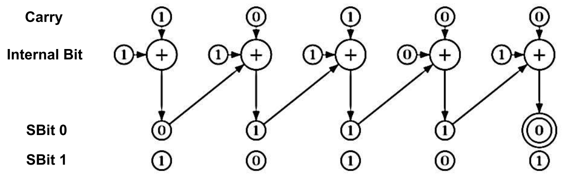


**Fig. 1.** Linear bit-serial multi-input adder process.

next full adder input in the chain. This bit can also be seen as the parity of the number of 1's input to the full adder in that pixel. By associativity of the parity operator, the result at the end of the chain can be interpreted as the parity of the number of 1's among all local binary inputs in the chain (fig.1). This value is also the least significant bit of the sum of local binary data of the whole region. To start the regional sum algorithm, the least significant bit of the operand in each pixel is placed in the internal bit while all carries are reset. The first step is to run the global combinatorial sum computation and to get the least significant bit of the sum in the adder at the end of the chain (displayed as a double circled *Sbit0* in fig.1). The second step is to move the most significant bit (called *SBit1* in fig.1) of each full adder in the carry bit. Besides, local values corresponding to the next bit of the operands are loaded in the internal bit of the pixel.
Then, the process is iterated to produce each bit of the sum, as the output of the processor in the pixel at the end of the chain.
This algorithm computes the regional sum in $N$ combinatorial operations where $N$ is the number of bits required to represent the sum. These combinatorial operations are executed in a synchronous sequence. Since the regional sum operator is based on the ripple propagation, from pixel to pixel of parity information from the beginning to the end of the chain, we consider it as an *asynchronous* operator.

## 2.2 Limitations

Although this implementation allows to compute the regional sum quickly, the main problem is that a chain is a linear structure (cf. fig.2), and it is impossible to cover arbitrary connected sets of pixels with chains. Figure 3 shows a simple example of this impossibility. In such a situation, a tree-based bit-serial multi-input adder is needed instead of a linear one.

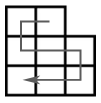


**Fig. 2.** Linear adder

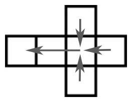


**Fig. 3.** Tree adder

## 3 Tree bit-serial multi-input adder

The asynchronous algorithm used to compute a sum over an arbitrary shaped region is an extension of the linear bit-serial multi-input one presented in previous section. The main difference is the tree structure of the global adder. Such a structure has been implemented in the Associative Mesh of Orsay [5] [6].
What is a tree ? It is a direct acyclic graph [7]. That means there is no loop (it is impossible to find 2 points connected by more than one direct subgraph) and there is only one root in it. The acyclic property is needed for using non idempotent associative operators (such as sum) on a graph [8]. The root is be used to collect the sum information computed on the graph.
Every pixel in the region is connected to this root through a spanning tree (fig. 3). As a consequence, inputs of the adder are connected to local binary data (internal bit and carry) and to *all* the directly preceding full adder least significant output bits in the tree. Consecutively, in 4-connectivity, each pixel different from the root can be connected to up to 3 neighbors as input of the adder, the fourth one being necessarily connected to the output of the adder. Taking into account 1 local binary data (both binary local data are added synchronously before the asynchronous sum), a total of 4 binary inputs are needed for the local adder. A Wallace tree analysis shows that 1 full adders and 2 half adder are needed to perform this task in a combinatorial way.
The algorithm used to compute the sum is very similar to the one described in section 2.1. The only difference lies in the number of inputs of the adders.
At this point, one main issue is still remaining : How to install a spanning tree over a region using a fast enough procedure regardless of the region shape ?

### 3.1 Asynchronous spanning tree installation

A spanning tree cannot be settled efficiently in a synchronous way, because the number of steps of the algorithm would grow linearly with the geodesic diameter, and it would take a long time. So, we have to perform this task in an asynchronous way.
The asynchronous algorithm used is the following one. At initialization, all connections between pixels of a same region are established. All pixels are inactive and a root is chosen, deterministically or at random. Then the root is activated, and communicate its state to the neighbor pixels. Each activated pixel keeps in memory the connection through which it was activated, and forward its active state to its neighbors. This process propagates asynchronously throughout the region until all pixels are active. The oriented spanning tree is obtained looking at the unique connections used for the activation of each pixel.
As explained before, a spanning tree is a direct acyclic graph, that means each pixel must have only one antecedent. During the algorithm, a pixel may have to choose between 2 or more antecedents if they want to activate the considered pixel at the same time. For this reason a 4 inputs arbiter is needed in each pixel.

### 3.2 Tree bit-serial adder asynchronous hardware cost

The different asynchronous components needed to perform regional sum computation and spanning tree installation have been defined before. According to the specification above, the elementary processor asynchronous part is composed of a 4-input arbiter (32 transistors), a 4-input adder(44 transistors), and 6 programmable connections (necessary for choosing 3 out of 4 inputs). The hardware cost of the asynchronous part is finally 82 transistors. Such an important cost is worth being reduced for a VLSI implementation.

## 4 Network topology for regional sum computation over a 4-connectivity region

Asynchronous dedicated operators used for sum computation are expensive mostly because they have 4 inputs. To cut this transistor expense, a lighter structure is desirable. When looking at an example of computing network, we notice that most of the cells exploits their 4-input convergent operators as simple 1-input operators only. This is a waste of resources. Is there any way to better distribute the network, that allows the use of k-input convergent operators with $k$ smaller than 4 ?
First, let's recall that $k = 1$ corresponds to 1-input operators and is therefore insufficient. What about using a 2-input asynchronous operator? Let's call it a *2-input convergent operator*. Let's consider a computation network over a region with $m$ pixels. Connecting these $m$ pixels together in a tree structure requires exactly $m - 1$ operators, convergent or not. How to settle them among $m$ pixels ? Only one 2-input convergent operator in each pixel could be enough. There are 2 main issues for implementing such a structure. The first one is the network topology needed to implement it, the second one is how to install the spanning tree.

### 4.1 Network topology

We recall that a 4-connectivity network combined with the use of 2-input convergent operators is insufficient to set-up a network over an arbitrary shaped region. Let's consider a cross-shaped region of 5 pixels, such a region is an example of this impossibility (Fig. 5). A solution is to increase the connectivity level used. An 8-connectivity interconnection network could be an obvious solution to the problem, allowing vertical, horizontal and both diagonal connections. However, the hardware cost would be rather expensive. A better solution is to use 6-connectivity and 2-input convergent operators. First, let's show that this solution fits our needs. For this, let's consider the pixel matrix as a hexagonal mesh (Fig. 4). Thanks to hexagonal mesh properties, an arbitrary pixel configuration can be connected with only 2-input convergent operators. For example, installation of a spanning tree using only 2-input convergent operators over a 5 pixels cross-shaped region is proposed (Fig. 5),whereas it was impossible to

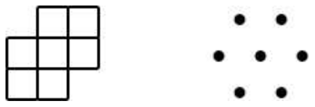


**Fig. 4.** Transformation from square to hexagonal mesh

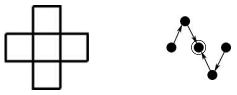


**Fig. 5.** Spanning tree over a cross-shaped region

map in 4-connectivity. 6-connectivity is the lowest connectivity level allowing the connection of an arbitrary shaped region into a tree structure with only binary operators, and it leads to the lowest possible hardware cost using asynchronous computation operators.

### 4.2 Asynchronous spanning tree installation

Using 4-inputs convergent operators, installation of a spanning tree is a rather simple task. Starting from a fully connected network, a signal propagates from the tree root through the network until all pixels have been reached (cf. 3.1). Using 2-inputs convergent operators, this task is much more difficult because at the initialization of the algorithm, each pixel can be connected to 2 pixels only, and not to all its neighbors. Connecting all the neighbors is something simplifying the construction of the spanning tree but fortunately it is not really necessary. One as only to ensure that propagation starting from one pixel will reach all the other pixels of the region. That means every pixel of the region has to be connected to all other pixels. Such a region is called a strongly connected component (SCC). The issue is how to build a SCC in 6-connectivity using only 2-inputs convergent operators.

**Algorithm principles** A way to solve this problem is to connect all the boundary pixels of the region into a clockwise oriented chain and then to connect all the pixels not connected yet and the boundary rings together. Fig.6 shows the original region on the top, and its corresponding hexagonal representation after the initialization of the connections on the bottom. As explained before, pixel inputs are connected to a maximum of 2 other pixels. According to this connection method, boundary pixels are connected in a SCC (a ring is a simple SCC), and other pixels are added to the SCC thanks to bi-directional connections, this ensuring them to be part of the SCC.
The final step of the proposed method is to extract a spanning tree from the SCC by propagating a token from the root as explained before in section 3.1.

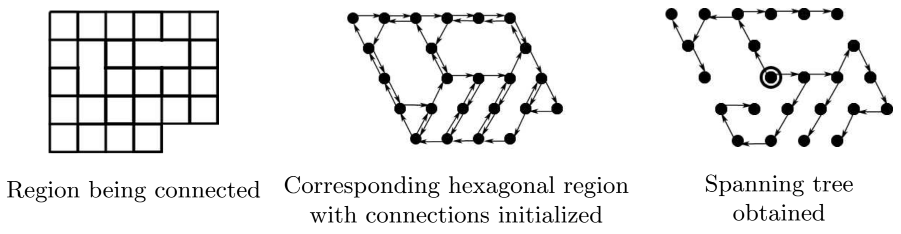


**Fig. 6.** Example of spanning tree installation using a 6-connectivity interconnection network and 2-inputs convergent operators over a 4-connectivity region

**Algorithm for connecting a SCC** The algorithm used is very simple, and can performed in a very cheap and fast synchronous way. Initialization of the connections can be done by only considering 6 local pixel configurations as described in Fig. 7. Configurations 1 to 6 are used for connecting boundary pixels. Configurations 5 and 6 also allow to connect all other pixels diagonally. In the different configurations of Fig. **??**, the pixel to connect is double-circled. Black pixels are pixels belonging to the region while white ones are pixels outside the region.
An important fact is the non-isotropy of the local transformation. Configuration 5 and 6 are not rotated versions of configurations 1 and 2 or 3 and 4. This is a consequence of mapping a 4-connectivity square mesh onto an hexagonal network. Instead of configurations 5 and 6, using a $2\pi/3$ rotated versions of configurations 1 and 2 would lead to connect diagonal configurations of pixels not connected in a 4-connectivity squared mesh. Actually, the diagonal connection, not present in 4-connectivity, is used here for establishing all the non-boundary connections.

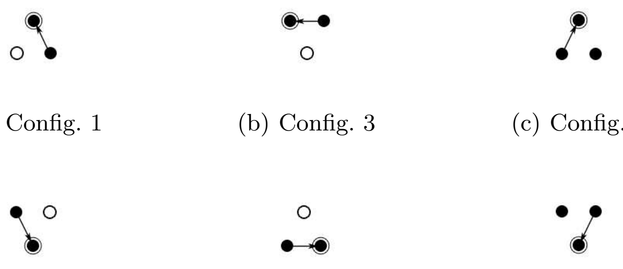


**Fig. 7.** Local configurations for SCC initialization in 4-connectivity using a 6-connectivity interconnection network

**Validity of the algorithm** Having presented the principles and operation of the algorithm, let's demonstrate its validity. By construction, all pixels are connected in a same SCC. By construction, all pixels of a same region are connected into one SCC. The only point to check is that pixel inputs do not have to be connected to more than 2 neighbor pixels, to allow the use of 2-inputs operators. As shown in Fig.7, each configuration sets-up one input connection. We have to verify that if a pixel neighborhood corresponds to 2 configurations, all the other configurations are false. For this let's consider the mutual exclusions of the configurations. Configuration 1 excludes configuration 3 and 5. Configuration 3 excludes 1 and 5, and configuration 5 excludes 1 and 3. Finally only one odd-numbered configuration can be true at one time. It is the same for even configuration. Only one even configuration can be true for a given neighborhood. There are no exclusions between odd and even configurations. Finally, a pixel neighborhood can only match one even and one odd configuration. That means a maximum of 2 configurations can be valid at one time, and maximum 2 input connections will be set-up in the pixel.

**Performance of the algorithm** The proposed algorithm for initializing the SCC can be performed very efficiently in a synchronous non-iterative way. This allows using this algorithm on a massively parallel synchronous machine having only limited resources for synchronous computation. There is no hardware dedicated to the spanning tree initialization task, which means that the reduction of hardware cost due to the use of 2-input convergent operators does not imply additional costs for installing the spanning tree.

### 4.3 Hardware reduction

Using 6-connectivity connections over a squared mesh allows to reduce the hardware cost dedicated to asynchronous regional sum to one 2-input arbiter (8 transistors) and one 3-inputs adder (20 transistors) in each pixel. 3 inputs are necessary for the adder : one for the local bit, and two for the neighbor connections. However, the necessary number of programmable connections increases. 2 out of 6 neighbors have to be connected at one time. Consequently, the minimal number of programmable connections necessary is 5 connections for each input. Finally, the number of transistors needed is $8 + 20 + 10 = 38$ transistors.
44 transistors are saved in each pixel by using a 6-connectivity topology. This leads to a reduction of 54% of the asynchronous hardware expense. However, the algorithmic capabilities are remaining the same.

## 5 Network topology for regional sum computation over a 6-connectivity region

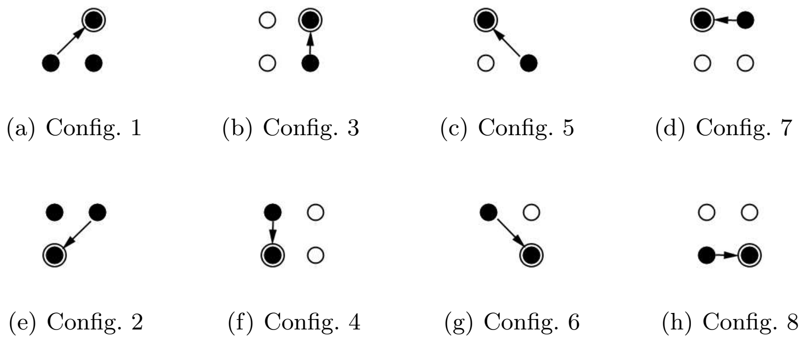
### 5.1 Network topology

Increasing the connectivity level of the interconnection network for reducing hardware cost of regional computation using only 2-input convergent operators

can be extended to 6-connectivity meshes. A 8-connectivity interconnection network is used. The interconnection network will be represented in a squared mesh because as shown in Fig. 8, in a squared mesh, a 8-connectivity neighborhood is more regular ( central symmetry) than in a hexagonal mesh (only 2 axial symmetries).
The principles of the algorithm remain the same as in 4-connectivity. Boundary

**Fig. 8.** Transformation from squared to hexagonal 8-connectivity mesh

pixels are connected into a clockwise oriented chain and non-boundary pixels are connected to the SCC formed by the boundary ring using a synchronous algorithm (cf. Fig. 9).There are 8 configurations to consider for setting the inter-



(a) Config. 1 (b) Config. 3 (c) Config. 5 (d) Config. 7

(e) Config. 2 (f) Config. 4 (g) Config. 6 (h) Config. 8

**Fig. 9.** Local configurations for SCC initialization in 6-connectivity using a 8-connectivity interconnection network

connection network. All theses configurations are used for connecting boundary pixels. Configurations 1 and 2 are also used for connecting non-boundary pixels to the boundary ring. With the proposed algorithm, an 6-connectivity network can be connected using 2-input convergent operators.

## 5.2 Hardware reduction

Using an 8-connectivity interconnection network over a 6-connectivity hexagonal mesh allows to reduce the hardware cost dedicated to asynchronous regional sum to one 2-input arbiter (8 transistors) and one 3-inputs adder (20 transistors) in

each pixel. 3 inputs are necessary for the adder : one for the local bit, and two for the neighbor connections. Considering interconnections, 2 out of 8 neighbors have to be connected at one time. Consequently, the minimal number of programmable connections necessary is 7 connections for each input. Finally, the number of transistors needed is $8 + 20 + 14 = 42$ transistors.
Using 7-input asynchronous adders and 6-input arbiters leads to an hardware cost of 156 transistors (72 for the adder and 84 for the arbiter). Finally, 114 are saved in each pixel by using a 8-connectivity interconnection network. This leads to a dramatic reduction of 73% of the asynchronous hardware expense, without lowering algorithmic capabilities.

## 6 Conclusion

We first presented implementations for regional sum computation through subsets of pixels in the image. After evaluating the hardware cost of dedicated operators needed for computing the sum asynchronously, we proposed using a 6-connectivity interconnection network and 2-input operators for reducing the hardware cost of asynchronism in 4-connectivity squared meshes. This leads to a important reduction of more than half the original transistor cost. The reduction is even more important (73%) when using an 8-connectivity interconnection network and 2-input convergent operators for mapping 6-connectivity region. Such a reduction offers an opportunity for achieving a dense large scale implementation of this circuit. Such an implementation is now on its way, and will lead to a vision chip allowing to perform medium level image processing.

## References

1. A. Moini, *Vision Chips*, Kluwer Academic Publishers, ISBN: 0-7923-8664-7, 2000.
2. F. Paillet, D. Mercier, and T.M. Bernard, “Second generation programmable artificial retina,” in *IEEE ASIC/SOC Conf.*, Sept. 1999, pp. 304–309.
3. H. Li and Q. Stout, *Reconfigurable Massively Parallel Computers*, Prentice-Hall, Englewood Cliffs, NJ, 1991.
4. T. Komuro, S. Kagami, and M. Ishikawa, “A dynamically reconfigurable simd processor for a vision chip,” *IEEE Journal of Solid-State Circuits*, vol. 39, no. 1, pp. 265–268, 2004.
5. A. Merigot, “Associative nets: A graph-based parallel computing net,” *IEEE Transactions on Computers*, vol. 46, no. 5, pp. 558–571, 1997.
6. D. Dulac, S. Mohammadi, and A. Merigot, “Implementation and evaluation of a parallel architecture using asynchronous communications,” in *CAMP*, 1995, pp. 106–111.
7. B. Ducourthial and A. Merigot, “Graph embedding in the associative mesh model,” Tech. Rep. TR-96-02, 1996.
8. B. Ducourthial and A. Mérigot, “Parallel asynchronous computations for image analysis,” *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1218–1228, 2002.