Convergent micro-pipelines : A versatile operator for mixed asynchronous-synchronous computations

Valentin Gies
ENSTA
32, Bd Victor
75015 Paris, FRANCE

Thierry M. Bernard
ENSTA
32, Bd Victor
75015 Paris, FRANCE

Alain Mérigot
Institut d'électronique fondamentale
Université Paris Sud
Orsay, FRANCE

***Abstract*— Micro-pipelines are linear (1-D) structures for asynchronous communications. In retinotopic VLSI vision chips, communicating over 2-D image regions is a key to efficient mid-level vision computations. However, micro-pipelines are limited to 1-D communications only. In this paper, we introduce an extension of the micro-pipeline, called the *convergent micro-pipeline*, for implementing mixed asynchronous-synchronous regional computations over arbitrary shaped regions. This operator is exploited in programmable artificial retinas (PAR), that is, image sensors with a digital processor in each pixel, for low power vision applications. To illustrate its behavior and versatility, several regional computations are described.**

I. Introduction

Our motivation for micro-pipelines arose from their ability to support communications between distant pixels on a data-dependent VLSI communication network. The latter, is of interest in an artificial retina (a ”vision chip” [1]), i.e. an image sensor with a processing element (PE) in each pixel. For low power and versatile vision, we focused on digital PARs [2], 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.
SIMD is the basic operating mode of a PAR : at each clock cycle, the same instruction is simultaneously executed by each pixellic processor. Abandoned in the 90's, SIMD processing has come back into favor within commercial microprocessors in order to cope with frequency and power consumption limitations. The SIMD mode has the same virtues in PARs, but 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. In this paper, we introduce a novel structure based on a tree extension of micro-pipelines for computing regional mixed asynchronous-synchronous primitives. Called convergent *micro-pipelines*, it reduces the implementation cost and replaces dedicated by multi-purpose asynchronous hardware.

II. A case study for regional computation : the regional sum

As the visual process goes on, segmentation operators turn the image into a set of regions, each of which is a connected set of pixels. Efficient regional operators for simultaneously extracting features from each region, e.g. moments are needed. Regional operators need to communicate between sparse and distant pixels within each region. Programmable neighbor-to-neighbor connections [3] allow to implement data-dependant communication networks within the SIMD framework, but with very poor synchronous performances. In this 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 communication to efficiently handle regions. Is it enough ? To carry on the investigation, it is now worth considering the particular but exemplary case of the ”regional sum”. Given an integer number in each pixel, how to simultaneously compute the sum of these integers over each region ? There is no efficient solution to implement the regional sum in the SIMD framework. But it seems that programmable connections and asynchronous communications are not enough to help. In the next sections, we first review a few solutions that have been proposed by others to this problem and then introduce a novel one, based on the use of micro-pipelines.

A. Linear bit-serial multi-input adder

The difficulty for computing a sum with SIMD operators is due to the necessity of moving data. 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 a 3-input adder. The operator used is a full adder.
The principle of the global addition is explained in fig.1This solution has been proposed and implemented in [4] using dynamically reconfigurable chains of pixels set by external programming.
The algorithm principle is the following one. In each processor, full adder inputs are connected to local binary data (internal bit and carry) and to the preceding full adder less

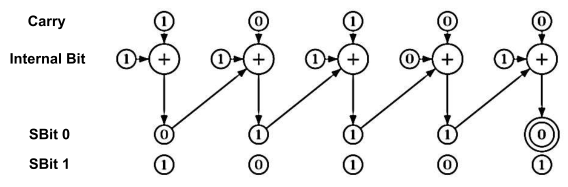


Fig. 1. Linear bit-serial multi-input adder process.

significant output bit (usually called the sum bit). The least significant output bit is connected to the 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.
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.

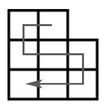


Fig. 2. Linear adder

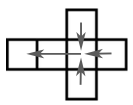


Fig. 3. Tree adder

B. Tree bit-serial multi-input adder

In this part, we recall a bit-serial multi-input adder which can be used to compute a sum on an arbitrary region. The region is considered in 4-connectivity, for which a pixel is connected to its four closest neighbors only. The asynchronous algorithm used to compute the regional sum 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.
What is a tree ? It is a direct acyclic graph [5]. 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 [6]. The root is used to collect the sum information computed on the graph. Every pixel in the region is connected to this root through a direct graph (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. 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 the 2 local binary data, a total of 5 binary inputs are needed for the local adder. A Wallace tree analysis shows that 2 full adders plus one half adder are needed to perform this task in a combinatorial way. Such a solution has already been implemented [7].
Then, the algorithm used to compute the sum is very similar to the one described in section II-A. The difference lies in the number of inputs of the adders. However, one main issue is still remaining : how can we install a spanning tree over a region using a fast enough procedure regardless of the region shape ?
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 its neighboring 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 an arbiter is needed in each pixel.
The different components needed to perform regional sum computation and spanning tree installation have been defined before. In addition to these functionalities, an SIMD part is present for the synchronization of the asynchronous phases, for the synchronous local computation and for data storage. This SIMD part has already been largely reported in the literature [2]. According to the specifications defined, the elementary

processor structure of the tree bit-serial multi-input adder is described in fig. 4.

This structure is a generalization of the linear bit-serial multi-input structure. The impossibility of covering arbitrary pixel subsets with chains has been coped with, but the proposed solution has an important hardware cost. In 4-connectivity, the hardware cost of a 4-input arbiter is about 30 transistors, and 58 transistors for the 5-input adder, 12 additional transistors are needed for routing data. Even if this structure is adequate from an algorithmic viewpoint, the number of transistors needed ($\simeq 100$) forbids the VLSI integration in a large size PAR.

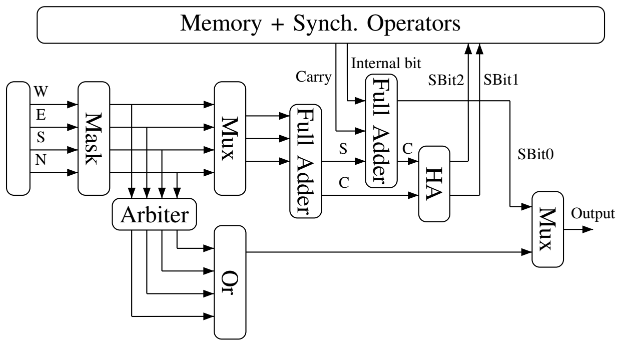


Fig. 4. Tree bit-serial multi-input adder processor

Another limitation of the tree bit serial multi-input adder is the use of a dedicated hardware structure for computing regional sum. Such a structure cannot be easily re-used for another computing task. With this kind of approach, every new asynchronous function require a specific hardware implementation. Less expensive and less dedicated alternative would be welcome.

III. Convergent micro-pipeline

To overcome the difficulties put in evidence in the previous section, asynchronous operators must be reconsidered in order to reduce them to the strict minimum needed to compute a sum over a spanning tree, and to set up this tree.

Preliminary step of the sum algorithm is the installation of the spanning tree. For this task, the arbiter is necessary and cannot be removed from the elementary processor. Consequently, let's make the most of it.

The arbiter's role is to choose one and only one signal out of the active ones. Only one output of the arbiter can be active. With this behavior, an arbiter can be seen as an automated multiplexor which can select the input corresponding to the first arrived data, and which can change the selected input when the selected data becomes inactive. In order to deactivate the selected input data when there is no need to keep it active, it is necessary to know when this data has been transmitted to the next processor. This functionality corresponds to the control structure of micro-pipelines introduced by I.E. Sutherland [8][9].

Once micro-pipelines adopted to connect neighbor pixels, getting a tree structure requires the ability of merging 2 or more micro-pipelines. Gathering an arbiter and micro-pipelines precisely yield this ability. The corresponding structure is shown on fig.5 and we call it *convergent micro-pipelines*.

This structure allows to propagate tokens from pixels to pixels through the spanning tree, heading for the root. A token is a set of one or several adjacent pixels where the micro-pipeline is at the logical active state. Unlike the global adder used in previous sections, no operation is performed during the propagation of a token through the tree. However, the number of tokens present in the spanning tree remains constant. The operator just propagates them towards the tree root. For example, if 2 tokens arrive simultaneously at 2 inputs of a convergent micro-pipeline, the arbiter blocks one token during the transmission of the other one. The blocked one is transmitted afterwards.

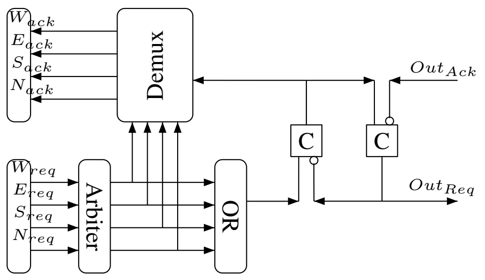


Fig. 5. Convergent micro-pipeline structure.

Since no operations are done during the propagation phase, computations are performed in a synchronous way after the token propagation. This approach reduces the hardware cost since it uses less dedicated operators inside the pixel.

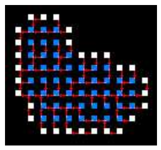
The hardware cost of the convergent micro-pipeline is about 52 transistors in 4-connectivity, which amounts to a 50% cut in transistor budget compared with the solution of fig. 4. This 50 transistors saving corresponds approximately to the hardware cost of the adder that has been removed.

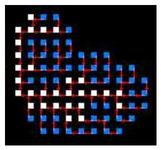
A. Region sum computation using convergent micro-pipelines

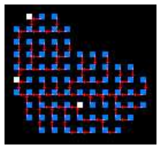
The ability of convergent micro-pipeline operator to gather tokens towards the spanning tree root, provide a way to compute the regional sum without a combinatorial adder in the pixel. First, the algorithm used to set up the spanning tree is the same as the one described above. As emphasized in section II-A, the least significant bit of a sum of tokens is the parity of this sum. Following this idea, an operation eliminating pairs of adjacent tokens will not change the parity of the set of tokens.

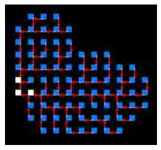
Sum computation is performed with the following algorithm. A token is placed in each of pixel participating to the sum computation (fig.6a). After propagation of these tokens towards the root (fig.6b), pairs of tokens are eliminated by pairs in a synchronous way (fig.6c). This elimination does not depend on tree topology : considering the whole array of pixels as a checkerboard, each white pixel tries to couple its own token with the token located on e.g. its northern black neighbor. That allows an efficient simple pixel pair

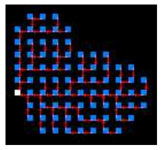
synchronous SIMD elimination. The tokens remaining after the elimination are then propagated again (fig.6d) and then eliminated by pairs. The process is iterated until there is only 1 or 0 token remaining in the spanning tree. Necessarily, the remaining token will find itself at the root of the tree (fig.6e). This value is the least significant bit of the number of tokens (by eliminating pairs of tokens, the parity of the global number of tokens has not changed).
Actually, pairs of tokens are not eliminated as suggested in


(a) Token initialization


(b) End of tokens propagation


(c) Tokens after pairs eliminations


(d) End of tokens propagation


(e) End of LSB computation

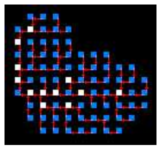

(f) 1-weighted tokens loading

Fig. 6. Region perimeter computation

the previous paragraph, but replaced by a single token to be used at the next step of the algorithm. Now each step of the algorithm is meant to produce one bit of the sum from LSB to MSB. Then it is meaningful to consider that tokens have a weight. At the first step of the algorithm, the token weight is 1, then each time 2 tokens are coupled, they produce a token of which the weight is twice as large. While computing the i^{th} bit of the sum, the weight of the tokens moving in the spanning tree is 2^i.
Proposed algorithm allows to compute a sum over a region without using a combinatorial adder in each pixel. This method allows to reduce the hardware cost of the elementary processor. However, the algorithmic cost of this method is increased. Using tree bit-serial multi-input adder (cf. section II-B), computation of one bit of the global sum is performed with only one propagation through the spanning tree. Instead of that, using convergent micro-pipelines, requires a few propagations (typically 2 to 4). Still, the number of propagation remains linear with the number of bits needed to represent the sum.

B. Other algorithmic capabilities

Beyond the reduction of the hardware cost, sum computation is done in a synchronous way, and without using a dedicated adder. Compared with existing solutions, the only hardware structure remaining is the propagation network. Dedicated hardware resources have been removed from the circuit. Computation operations on the tokens being performed by standard synchronous operators after propagation phases, consequently it is possible to find many other algorithmic capabilities.
The most simple one is the global OR operator. A token is placed in each active pixel. Tokens are then propagated, the presence of this token or not in the root is the result of the global OR.
Another more complex one is the ternary sum. If elimination is performed on sets of 3 pixels instead of pairs of pixels, one token being generated for each set of 3 pixels deleted, we can compute directly a sum in ternary base instead of a binary sum Using versatile convergent micro-pipelines allows to overcome some limitations of dedicated functions such as binary adders.

C. Validation

A 2 input convergent micropipeline circuit has been validated by SPICE simulation. Fig.7 shows the convergence of token described in III. Input tokens arrive by $RI1$ and $RI2$. Output tokens exits through RO.

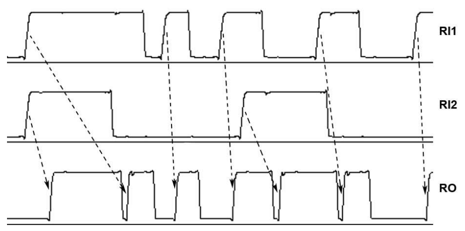


Fig. 7. 2 input convergent micro-pipeline oscillogram

IV. Conclusion

In this paper, we proposed an novel approach for region-based computations using convergent micro-pipelines. This versatile operator allows to reduce the number of transistors in the asynchronous part of the pixel by removing dedicated hardware, and provides new algorithmic capabilities. A large scale implementation is now on its way, and will lead to a retina chip allowing to perform medium level image processing with a wide range of efficient regional operators.

References

[1] A. Moini, *Vision Chips*. ISBN: 0-7923-8664-7: Kluwer Academic Publishers, 2000.
[2] F. Paillet, D. Mercier, and T. 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*. Englewood Cliffs, NJ: Prentice-Hall, 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] B. Ducourthial and A. Merigot, "Graph embedding in the associative mesh model, Tech. Rep. TR-96-02, 1996. [Online]. Available: citeseer.ist.psu.edu/ducourthial96graph.html
[6] B. Ducourthial and A. Mérigot, "Parallel asynchronous computations for image analysis," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1218–1228, 2002.
[7] A. Merigot, "Associative nets: A graph-based parallel computing net," *IEEE Transactions on Computers*, vol. 46, no. 5, pp. 558–571, 1997.
[8] I. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, 1989.
[9] S. Hauck, "Asynchronous design methodologies: An overview, Tech. Rep. TR-93-05-07, 1993. [Online]. Available: citeseer.ist.psu.edu/17380.html