Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 854241, 14 pages
doi:10.1155/2009/854241

Research Article

A CNN-Specific Integrated Processor

Suleyman Malki and Lambert Spaanenburg (EURASIP Member)

Department of Electrical and Information Technology, Lund University, P.O. Box 118, 22100 Lund, Sweden

Correspondence should be addressed to Suleyman Malki, suleyman.malki@gmail.com

Received 2 October 2008; Accepted 16 January 2009

Recommended by David Lopez Vilarino

Integrated Processors (IP) are algorithm-specific cores that either by programming or by configuration can be re-used within many microelectronic systems. This paper looks at Cellular Neural Networks (CNN) to become realized as IP. First current digital implementations are reviewed, and the memoryprocessor bandwidth issues are analyzed. Then a generic view is taken on the structure of the network, and a new intra-communication protocol based on rotating wheels is proposed. It is shown that this provides for guaranteed high-performance with a minimal network interface. The resulting node is small and supports multi-level CNN designs, giving the system a 30-fold increase in capacity compared to classical designs. As it facilitates multiple operations on a single image, and single operations on multiple images, with minimal access to the external image memory, balancing the internal and external data transfer requirements optimizes the system operation. In conventional digital CNN designs, the treatment of boundary nodes requires additional logic to handle the CNN value propagation scheme. In the new architecture, only a slight modification of the existing cells is necessary to model the boundary effect. A typical prototype for visual pattern recognition will house 4096 CNN cells with a 2% overhead for making it an IP.

1. Introduction

Over the past years, computer architecture has developed from general-purpose processing to provision of algorithm-specific support. Many signal-processing applications demand a large amount of processing elements (PEs) arranged in a 1- or 2-dimensional structure. In the video domain, it is well known that both structures are required for efficient operation, and a number of application-specific devices have been built. Nowadays, we see this experience reaching the embedded computing domain, where in-product supercomputing is the key to product quality. For instance, the NXP EPIC and the TI Leonardo da Vinci have a matrix of tiles as CPU [1].

The cellular neural network (CNN), as proposed by Chua and Yang [2, 3], is a typical example of a computational method that assumes a 2-dimensional structure. Each node has a simple function; but the input values need to be retrieved from all cells within a specified neighborhood for each nodal operation. Some years later, Harrer and Nossek introduce the discrete-time cellular neural network (DT-CNN) for digital implementations [4]. Initial applications are largely in the field of image processing, where the analog counterpart [5] suggests that an 8-bit number representation is more than enough. In case of doubt, the regular CNN structure allows for algorithmic pruning to establish the minimal word length requirements for a specific application [6].

In a two-dimensional DT-CNN, each cell c, which is identified by its position in the grid, communicates directly with its r-neighborhood, that is, a set of cells within a certain distance r to c. If $r = 1$, we have a 3×3 neighborhood while in the case of $r = 2$, a 5×5 neighborhood is obtained. Nevertheless, a cell can communicate with other cells outside its neighborhood due the network propagation effect. The state of a cell, x^c, depends mainly on the time-independent input u^d to its neighbors d and the time-variant output $y^d(k)$ of these neighbors. Equation (1) describes this dependence in a discrete time k. The control coefficients b_d^c only "scale" the inputs, while the feedback coefficients a_d^c are responsible for the nonlinear dynamical behavior. These coefficients are usually combined to compose matrices, which results in a so-called cloning template $T = \langle A, B, i \rangle$. The CNN equation

(1) implies linear transformations; by suitable application of linear templates, all 2-dimensional single data manipulations can be performed. Output of cell c at a certain time step is simply obtained by means of a squashing function. Three different types of nonlinear functions are frequently used [4]: threshold, hyperbolic tangent, and piecewise linear function.

$$x^c(k) = \sum_{d \in N_r(c)} a_d^c y^d(k) + \sum_{d \in N_r(c)} b_d^c u^d + i^c \qquad (1)$$

Both analogue (mixed-signal) and digital realizations of a CNN have been published [7, 8]. The former have a larger network capacity and allow for handling images of sufficient size. This is preferred as most work targets image processing in spite of the intrinsic ability of a CNN to solve complex nonlinear differential equations. On the other hand, digital implementations have been discarded as the massive amount of required multiplications is too area consuming. Furthermore, the digital CNN architecture is wiring dominated. Already 8 pairs of input and output values need to be communicated for the minimal 1-neighbourhood, one for each neighboring node. This is easily affordable in analogue architectures as each value is carried by a single wire only. But for digital architectures, in the simple case of 8-bit values, the simultaneous use of 8 values will need 64 wires to be routed. Obviously, the interconnection requirements are severely increased for larger neighborhood. Actually, establishing the connections within an arbitrary neighborhood is so area and/or time demanding that little research on large neighborhoods is made. Almost all known CNN templates are for a 1-neighbourhood, and all realizations are effectively restricted to that. The restriction is not fundamental, as a proper interconnect structure can extend a digital implementation to a larger neighborhood.

A related issue is the need for accessing the external image memory. In a typical system, the slow access of memory can only be balanced to the speed of the CPU by widening the memory bus [9] or by adding more functionality to the CPU [10]. As already shown in [11], complex CNN operations can help out also. Still the search remains open for the digital architecture that limits the memory access requirements.

A third problem is the handling of boundary effects. In a naive design, a network needs a frame of 2 cells in width to fix the boundary in a programmable way. This will severely decrease the usable capacity of the system. In other words, a proper handling of the boundary is basic for the development of a CNN integrated processor.

The paper goes through a number of such architectural issues. First, we review the early architectures and analyze their performance metrics. Then, we take a generic view and propose a word-serial mechanism. In Section 6, attention is given to the modeling of the boundary effects. Finally, we conclude the effect of such measures on the definition of a CNN as IP and see that we can prototype up to 4 k cells with 2% system overhead on a Xilinx Virtex-II 6000.

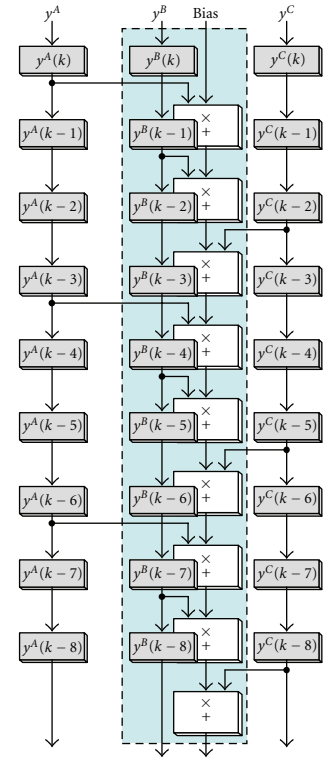

Figure 1: Data dependencies for a pipeline in naive architecture. Only the pipeline corresponding for the middle node is shown. White boxes represent functional blocks, consisting of a multiplier and an adder, while grey boxes represent registers. The middle node corresponds to a pixel sequence y^B. For sequences y^A and y^C, functional blocks are dropped for clarity. Identical architecture is used to calculate the contribution of pixel inputs.

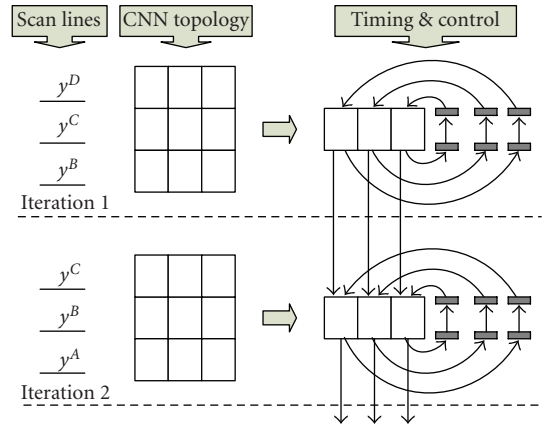

Figure 2: A pipelined CNN architecture with a pipeline three nodes.

2. CNN Architecture Spectrum

The mapping of mathematical CNN cells into physical network nodes can be done in several ways, depending on the adopted communication style. The approach first introduced in [12] inherits the benefits of superscalar computer

architecture, where values are retrieved from data memory, fed in series through a heavily pipelined processing unit and finally stored back in the data memory. The data represent a topographic map, often a natural image with pixel values. In a naive realization, data dependencies between scan lines in an image are stretched over a pipeline of single multiply-accumulate units (Figure 1). Each neighboring input value is evaluated separately in a pipelined fashion, doing in series as many multiply accumulates as there are cells in the neighborhood.

Interweaving three pipelines, corresponding to a row of three input values, reduces the latency (Figure 2). In other words, we let every node in the network contain image data from three pixels, that is, pixel values for the cell itself and for its upper and lower neighbors are stored in each node. A direct connection with the left and right nodes completes the communication between a node and its neighborhood. In short, one node contains three pixels and calculates the new value for *one* pixel and *one* iteration. One of such realizations is a design called ILVA [7]. The prescheduled broadcasting keeps the communication interface at minimum, which allows for a large number of nodes on chip. The performance is high as the system directly follows the line accessing speed, but the design suffers from a number of weaknesses. It supports 1 neighborhood only, where extension to larger neighborhoods requires a total overhaul. Furthermore, iterations are flattened on the pipeline, one iteration per pipeline stage. Consequently, the number of iterations is not only restricted due to the availability of logic, but it is also fixed. Operations that require a single iteration only have still to go through all pipeline stages. Lastly, actions between the pixels go only in one direction.

To remedy this, the concept of network on chip [13] is explored. The CNN equation is not unrolled in time but in space [14], and the nodes retain the result of the equation evaluation so that next iterations do not involve access to the external data memory. Two main alternatives for transferring the data between the nodes are *circuit switching* [15] and *packet switching* [16]. In our first attempt, called Caballero [7], circuit switching is used and it is studied how the data transfers between the nodes can be scheduled.

The principle of operation depicted in Figure 3 is as follows. Pixel lines come into the FIFO till it is fully filled. Then, these values are copied into the CNN nodes that subsequently start computing and communicating. Meanwhile new pixel lines come in over the FIFO. When the FIFO is filled again and the CNN nodes have completed all local iterations, the results are exchanged with the new inputs. This leaves the CNN nodes with fresh information, and the FIFO can take new pixel lines while moving the results out.

The schedule is still predetermined, but splitting the simple node into a processor and a router decouples the computation and communication needs. The nodes can theoretically transfer their values within the neighborhood in parallel. The number of simultaneous transfers is, however, reduced to four per node as Manhattan broadcasting is implemented. For the minimal 1 neighborhood, this requires two steps only (Figure 4(a)). In principle, the number of possible iterations is infinite and flexible. In order to avoid bus conflicts, nodes are grouped into 5 different activation groups indicated in Figure 4(b) by g_1, g_2, g_3, g_4, and g_5. Apparently, this adds heavily to the control and severely reduces the amount of potential parallelism. The amount of additional required logic is so big that a larger neighborhood is basically precluded.

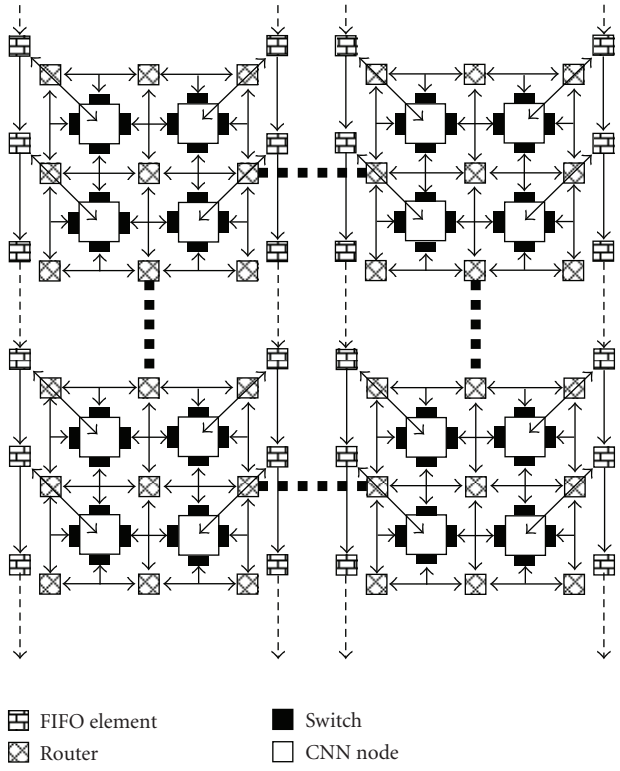

Figure 3: Caballero architecture uses a network-on-chip of CNN nodes, while the pixels are transported over a distributed FIFO.

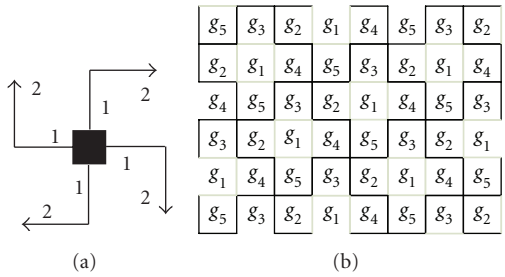

Figure 4: (a) Communication scheme and (b) activation groups in Caballero.

Having these prototype architectures available, it becomes interesting to have a better overview of the design space. An overview of the CNN implementation spectrum (VIND) is given in Figure 5. The similarity to Corporaal's 4-dimensional diagram about architecture design spectrum [17] reveals the importance of optimizing control and data flows in order to achieve a well-performing CNN system, as is always the case with hardware design.

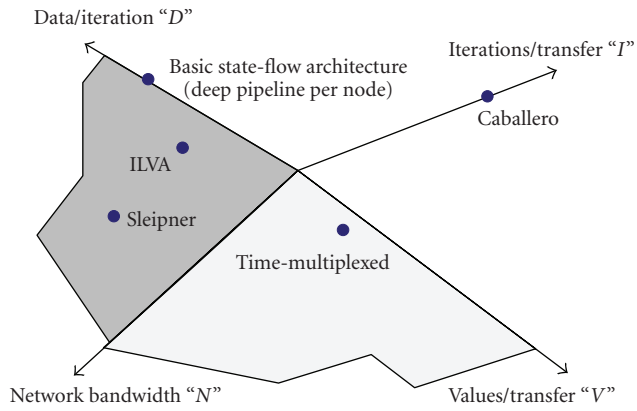

Figure 5: A 4-dimensional design space $\{V, I, N, D\}$ of CNN architectures.

The basic pipelined architecture (Figure 1) can be found on the D-axis as it aims to raise the number of data accesses to the external image memory. The individual accesses are threefold compared to ILVA (Figure 2). The bandwidth requirements can be even lowered when the intranetwork communication can handle an arbitrarily large neighborhood by virtue of a packet-switching technique, yielding a design called Sleipner [7]. This is shown by the move along the N-axis.

On the I-axis, we find the basic spatial architecture, Caballero (Figure 3). Every node interacts with its neighbors iteratively, constantly transferring data over the intranetwork. External memory access may be the limiting factor to system performance, but it depends to be seen for how many iterations the nodal computations become dominant. In Caballero, many values are transferred simultaneously. The effect may be counteracted by the scheduling needs.

These are only a few of the many CNN architectures. The algorithmic diversity is very large. Many technology mapping methods can be applied, next to temporal and spatial partitioning. As an example, we have already drawn in Figure 5 a version of Caballero with a time-multiplexed internal network that will be developed in Section 5. But there is much more, and therefore we do not claim that we present the most optimal. In fact, it appears that in the end the application decides on the quality of the implementation. The later that introduced generic structure helps compiling several networks from one description while fitting in the same box.

Though the connection pattern of the CNN structure is very regular and misleadingly easy to design, the network capacity needs to be very high to preclude bottlenecks. Therefore, we will analyze first the memory bandwidth requirements, taking the introduced archetypes (ILVA and Caballero) as example. Then, we take another approach to get grip on the algorithmic diversity of the implementation. The focus of that study is on the size and speed of the network interface (NI) that wraps any design part to become accessible through the network standard. It brings out the basic advantages of the time-multiplexed communication, and is fully in-line with the original Æthereal systematic [16].

3. Effect of Slicing

All known CNN implementations, both analogue and digital, are much smaller than a regular image frame. We may therefore rightfully assume that the network can handle only a part of the image at time. It has been suggested [7] that *slicing* the image solves the problem. Now, a smaller part of the image is fetched from memory which decreases the latency. In the following, a frame execution formula is derived to evaluate the effect of slicing for two of the digital realizations: ILVA and Caballero. We aim for a unified notation and make the following assumptions.

(i) Input values are brought per pixel line into a CNN column. Subsequent pixel lines will take subsequent columns.

(ii) Internodal broadcasting is instantaneous, that is, it does not add any delay to the system.

Memory time overhead, that is, the time needed to bring information from the external memory into the chip, is crucial for the overall elapsed time. Modern FPGA boards are equipped with off-chip memories of type DDR/DDR2 SDRAMs with different bandwidths. These memories are categorized due to their speed grade in "data transfers per second per pin." If memory bandwidth (in bits) and speed grade are denoted w_{mem} and s_{mem}, respectively, then the time required to fetch a (sub-) frame is given by equation (2). Here, w_d stands for the width, given in bits, of input/output values in the CNN, while r_{cnn} and c_{cnn} represent the number of rows and columns in the CNN, respectively.

$$t_{\text{fetch}} = \frac{w_d \cdot r_{\text{cnn}} \cdot c_{\text{cnn}}}{w_{\text{mem}} \cdot s_{\text{mem}}} \tag{2}$$

In Caballero architecture with a 1-to-1 mapping between digital nodes and CNN cells, the relation in equation (2) can be used straight forward, but it needs modification when ILVA is considered. Here, a fetched scan line is consumed directly, which has great influence on the overall performance of the system as will be seen soon. In this sense, if a scan line is mapped on a column of nodes (as in ILVA), the time needed to fetch one line from the external memory is obtained according to equation (3). It is easily seen that equations (2) and (3) are related by the number of columns c_{cnn}.

$$t_{\text{line\_fetch}} = \frac{w_d \cdot r_{\text{cnn}}}{w_{\text{mem}} \cdot s_{\text{mem}}} \tag{3}$$

In general, the nodal execution time for a certain template, t_{templ}, can be further divided into two parts.

(i) t_{const}: the time needed to calculate the control contribution along with the bias, that is, $\sum Bu + i$.

(ii) t_y: the time needed to calculate the iterative part of the state equation, that is, $\sum Ay$, followed by discrimination.

The first part needs to be performed only once for the given input pattern, while the second part is repeatedly performed depending on the required number of iterations, n_{iter}, that is in principle infinte. Obviously, t_{const} and t_y depend on the r-neighborhood, and so does t_{templ} as well. For all digital realizations carried out so far, it has been shown that $t_{\text{const}} = t_y$. Therefore, the common notation t_{comp} is used when no ambiguity rises. In this sense, template execution time can basically be expressed as depicted in equation (4). Frame execution time is then calculated according to equation (5).

$$t_{\text{templ}} = t_{\text{const}} + n_{\text{iter}} \cdot t_y = (1 + n_{\text{iter}}) \cdot t_{\text{comp}} \tag{4}$$

$$t_{\text{frame}} = (1 + n_{\text{iter}}) \cdot t_{\text{comp}} + c_{\text{cnn}} \cdot t_{\text{line\_fetch}} \tag{5}$$

This is, however, true only if the size of the network is large enough to accommodate an entire frame, while slicing the frame introduces a number of complications. The number of slices depends on the size of both frame and CNN as shown in equation (6), where r_{frame} and c_{frame} stand for the number of rows and columns in the processed frame, respectively.

$$n_{\text{slice}}^{\text{cab}} = \frac{\text{frame size}}{\text{CNN size}} = \frac{r_{\text{frame}} \cdot c_{\text{frame}}}{r_{\text{cnn}} \cdot c_{\text{cnn}}} \tag{6}$$

Two cases may arise depending on the relation between template execution time and data fetch time.

(i) If $t_{\text{fetch}} \leq t_{\text{templ}}$, frame execution time is dependent on the number of slices as well as on the template execution time. All output values corresponding to the inputs of the entire frame have to be available before the next iteration is performed. In other words, a single iteration has to be completed on each slice until the whole frame is processed before the next iteration is performed on the first slice of the next frame and so on. As the procedure of fetching overlaps with the computational part, due to the usage of FIFO-structure, Caballero is idle only when the first slice is brought in and the last slice is moved out. In equation (7), frame execution time is given as function of frame size, CNN size, number of iterations, and data fetch time. Note that the obtained formula differs from the one in equation (5).

$$\begin{aligned} t_{\text{frame}}^{\text{cab}} &= n_{\text{slice}}^{\text{cab}} \cdot n_{\text{iter}} \cdot (t_{\text{const}} + t_y) + 2 \cdot t_{\text{fetch}}^{\text{cab}} \\ &= 2\left(\frac{r_{\text{frame}} \cdot c_{\text{frame}}}{r_{\text{cnn}} \cdot c_{\text{cnn}}} \cdot n_{\text{iter}} \cdot t_{\text{comp}} + c_{\text{cnn}} \cdot t_{\text{line\_fetch}}\right) \end{aligned} \tag{7}$$

(ii) $t_{\text{fetch}} > t_{\text{templ}}$, frame execution time depends only on data fetch time:

$$\begin{aligned} t_{\text{frame}}^{\text{cab}} &= n_{\text{slice}}^{\text{cab}} \cdot n_{\text{iter}} \cdot t_{\text{fetch}} \\ &= \frac{r_{\text{frame}} \cdot c_{\text{frame}}}{r_{\text{cnn}} \cdot c_{\text{cnn}}} \cdot n_{\text{iter}} \cdot c_{\text{cnn}} \cdot t_{\text{line\_fetch}} \end{aligned} \tag{8}$$

In contrast to Caballero, ILVA has an implicit bound on the number of iterations. As the nodes are arranged in pipeline stages, on which the iterations are mapped, the maximum number of performed iterations is one shorter than the number of pipelines n_{pipe}. The first pipeline stage is used to calculate the constant part, while each of the following stages completes the computation of state and corresponding output. In all stages, the operation is accomplished during time t_{pipe}. Therefore, ILVA's template execution time, given in equation (9) differs from the one previously obtained for Caballero as given in equation (4). The calculated time is precise in Caballero, while it is on average in ILVA.

$$t_{\text{templ}}^{\text{ILVA}} = \frac{n_{\text{pipe}} \cdot t_{\text{pipe}}}{n_{\text{pipe}} - 1} \tag{9}$$

The pipelining mechanism requires only one (sub-) line of the frame to be present prior to computation start. ILVA consumes the fetched line directly but still experiences a latency that equals three times t_{comp} per pipeline stage. An overall latency rises from the fact that the pipeline has to be filled before the first output values are produced. This is reflected in the last term of equation (10). However, when the pipeline is filled, a new output value is produced each t_{comp}. In other words, pipeline execution time t_{pipe} can be replaced by t_{comp} without loss of generality.

$$\begin{aligned} t_{\text{frame}}^{\text{ILVA}} &= c_{\text{frame}} \frac{n_{\text{pipe}} \cdot t_{\text{comp}}}{n_{\text{pipe}} - 1} + t_{\text{line\_fetch}} \\ &\quad + 3t_{\text{comp}} \cdot n_{\text{pipe}} \end{aligned} \tag{10}$$

Slicing of the processed frame is required when $r_{\text{frame}} > r_{\text{cnn}}$. The number of slices is then given as

$$n_{\text{slice}}^{\text{ILVA}} = \frac{r_{\text{frame}}}{r_{\text{cnn}}} \tag{11}$$

In line with the earlier discussion, two different cases are distinguished.

(i) $t_{\text{line\_fetch}} \leq t_{\text{templ}}$, frame execution time depends mainly on the t_{comp} and $n_{\text{slice}}^{\text{ILVA}}$:

$$\begin{aligned} t_{\text{frame}}^{\text{ILVA}} &= \frac{r_{\text{frame}}}{r_{\text{cnn}}} \left(\frac{c_{\text{frame}} \cdot n_{\text{pipe}} \cdot t_{\text{comp}}}{n_{\text{pipe}} - 1}\right) \\ &\quad + t_{\text{line\_fetch}} + 3t_{\text{comp}} \cdot n_{\text{pipe}} \end{aligned} \tag{12}$$

(ii) $t_{\text{line\_fetch}} > t_{\text{templ}}$, frame execution time depends mostly on data fetch time:

$$t_{\text{frame}}^{\text{ILVA}} = \frac{r_{\text{frame}}}{r_{\text{cnn}}} \cdot t_{\text{line\_fetch}} + 3t_{\text{comp}} \cdot n_{\text{pipe}} \tag{13}$$

Due to the different mechanisms employed in ILVA and Caballero architectures, a straightforward comparison of frame execution times is not feasible. A key factor is the number of iterations a given template is performed. In ILVA, this number is tightly coupled to the number of realized columns, that is, $n_{\text{iter}} = n_{\text{pipe}} - 1$. Allowing more iterations will render the comparison unfair as it violates the intrinsic

Table 1: The actual number of rows in ILVA as a function of the number of pipelines and number of columns in Caballero with respect to equation (14). Parameter r represents the total number of rows in Caballero.

Iter	# Pipe	Number of rows in ILVA						
		6	7	8	9	10	11	12
1	2	$3r$	$3r$	$4r$	$4r$	$5r$	$5r$	$6r$
2	3	$2r$	$2r$	$2r$	$3r$	$3r$	$3r$	$4r$
3	4	r	r	$2r$	$2r$	$2r$	$2r$	$3r$
4	5	r	r	r	r	$2r$	$2r$	$2r$
5	6	r	r	r	r	r	r	$2r$
6	7	r	r	r	r	r	r	r

limit of functionality in ILVA. However, if less iterations are required, that is, $n_{\text{iter}} < n_{\text{pipe}} - 1$, the superfluous pipeline stages should be removed and replaced, if possible, by nodes in such a way that the total number of rows in ILVA is increased. Equation (14) explains the relation between the number of rows in ILVA and Caballero. In the following, the comparison is arranged such that first a single iteration, $n_{\text{iter}} = 1$, and then several iterations, up to $n_{\text{iter}} = c_{\text{cnn}} - 1$, are performed on both architectures. This will, with respect to equation (14), yield the different settings given in Table 1.

In order to express frame execution times in seconds, both ILVA and Caballero are assumed to run on 100 MHz, resulting in $t_{\text{comp}} = 10^{-7}$ seconds in both realizations. We assume further that a PAL frame of size 720×576 is stored on an external storage of type DDR266 with $s_{\text{mem}} = 266 \times 10^6$ and $w_{\text{mem}} = 16$ bits. With respect to equations (7) and (12), Figures 6 and 7 illustrate frame execution times with different sizes of the realized CNN.

$$r_{\text{cnn}}^{\text{ILVA}} = \begin{cases} r_{\text{cnn}}^{\text{cab}}, & \text{if } c_{\text{cnn}}^{\text{cab}} \le n_{\text{iter}} \\ r_{\text{cnn}}^{\text{cab}} \cdot \left\lfloor \dfrac{c_{\text{cnn}}^{\text{cab}}}{n_{\text{pipe}}} \right\rfloor, & \text{otherwise} \end{cases} \tag{14}$$

The figures show clearly that ILVA outperforms Caballero for all CNN-sizes when the larger number of iterations per template is required. Caballero is better when 1 or 2 iterations are needed. This is caused by the need to swap all slices in and out for each iteration. On the other hand, if a sequence of iterations is allowed on the same slice before the next slice is brought in, a different situation arises, which requires a slight modification of equation (7) as shown in equation (15). The resulting execution time is reflected in Figure 8. Here, it is noticed that Caballero performs better for more accommodated columns, almost regardless of the number of iterations.

$$\begin{aligned} t_{\text{frame}}^{\text{cab}} &= n_{\text{slice}}^{\text{cab}} \cdot t_{\text{templ}} + 2c_{\text{cnn}} \cdot t_{\text{line\_fetch}} \\ &= \frac{r_{\text{frame}} \cdot c_{\text{frame}}}{r_{\text{cnn}} \cdot c_{\text{cnn}}} \cdot (n_{\text{iter}} + 1) \cdot t_{\text{comp}} \\ &\quad + 2c_{\text{cnn}} \cdot t_{\text{line\_fetch}} \end{aligned} \tag{15}$$

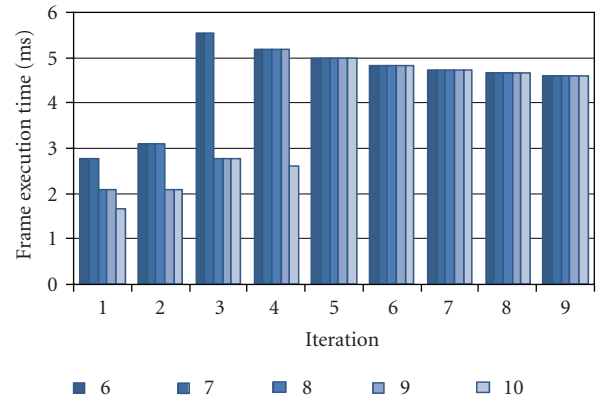

Figure 6: Frame execution time for ILVA with different CNN sizes, when slicing is required. The legends, 6 to 10, represent the number of pipelines, that is, the number of columns in the design.

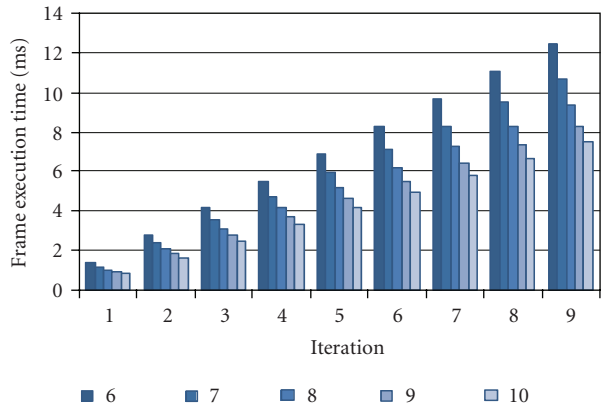

Figure 7: Frame execution time for Caballero with different CNN sizes, when slicing is required.

Any real-life application consists of a number of templates that are applied sequentially. In the extreme case, a new frame needs to be fetched from memory for each applied template. But for most applications, each template in the sequence needs to work on the same frame or on an intermediate modification of the frame from a previous template. This is valid if the frame and its intermediate copies are kept in the network, which is possible in Caballero only. Furthermore, the benefits of high throughput in ILVA are totally lost when the different templates in a single task vary in the number of iterations. In this sense, Caballero is preferred due to the provided iteration flexibility, especially when whole frames can be accommodated. As this is hard to achieve in the current implementation, *pixel sampling* seems to provide a way out. Here, each node will correspond to the average of a pixel block rather than just one pixel. This can initially be done for the entire frame and then repeated for smaller parts thereby gradually focusing into the region of interest.

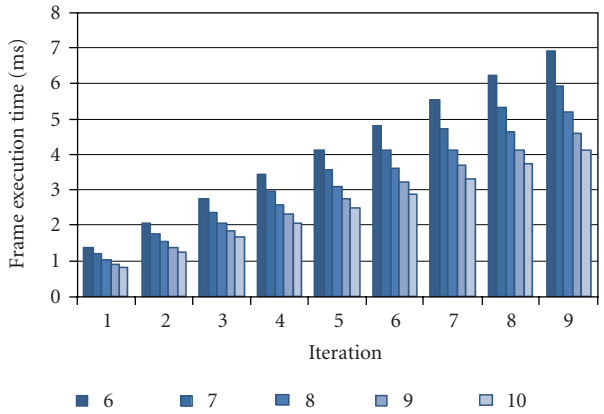

Figure 8: Frame execution time for Caballero is reduced when all the iterations are performed on a slice before next slice is brought in.

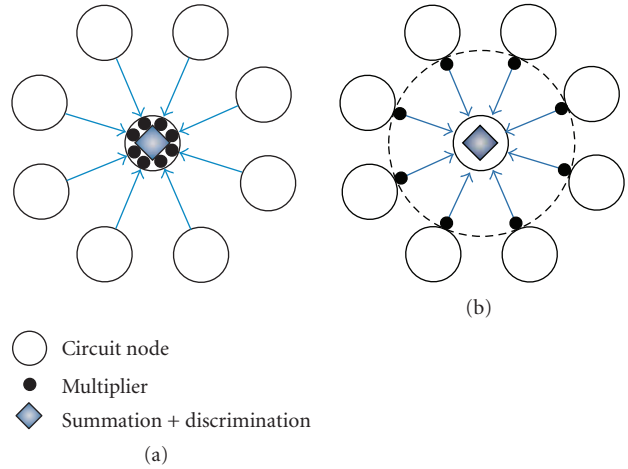

Figure 9: (a) Consumer and (b) producer cell to node mapping.

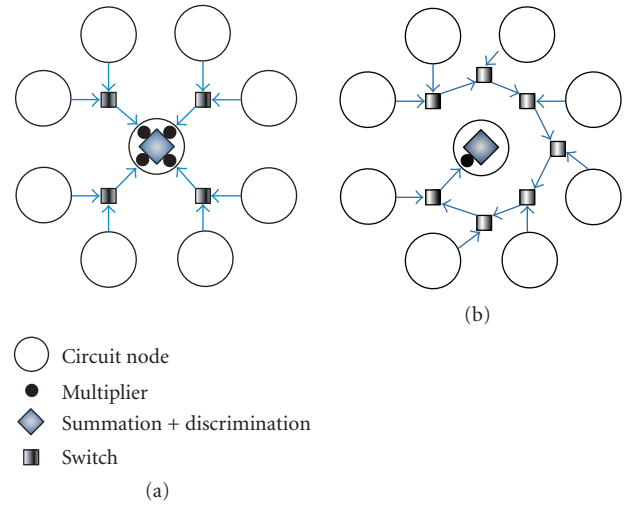

Figure 10: Value routing by multiplexing (a) in space and (b) in time.

The conclusion is that any Caballero-like overcomes memory latency if and only if

(i) the size of the CNN allows for a rapid determination of the region of interest, on which a succession of templates is applied:

(ii) the task consists of a number of templates, with a total number of iterations such that the total time exceeds, or at least equals, the time needed to fill the FIFO-structure.

In Section 5, we see how stretching the 2-step communication cycle in Caballero reduces local control demands and leads to smaller network interface (NI). The modified architecture accommodates more nodes such that pixel sampling is within reach.

4. Nodal Models

The computation of control and feedback contributions in the nodal equation (1) is identical by means of number and nature of the performed operations. The series of multiply-and-add operations have, however, to be explicitly scheduled in order to guarantee correct functionality and achieve the desired performance. The need for explicit scheduling on nodal activities works out differently for different CNN to network mappings.

(i) The *consumer node* is fully in accordance with equation (1). The discriminated output of a cell is also the node output and broadcasted to all connected nodes, where it will be weighted with the coefficients of the applied template before the combined effect is determined through summation (Figure 9(a)).

(ii) The *producer node* discriminates the already weighted inputs and passes to each connected node a separate value that corresponds to the cell output but weighted according to the applied template (Figure 9(b)).

Ideally all nodes are directly coupled, and therefore bandwidth is maximal. In practice, the space is limited, and the value transfer has to be sequenced over a more limited bandwidth. This problem kicks in first with the producer type of network, where we have $2n$ connections for n neighbors. The network-on-chip approach is meant to solve such problems. However, as the cellular neural network is a special case for such networks, having identical nodes in a symmetric structure, such a NoC comes in various disguises.

In the consumer architecture, scheduling is needed to more optimally use the limited communication bandwidth. Switches are inserted to handle the incoming values one by one. To identify the origin of each value, one can either schedule this hard to local controllers that simply assume the origins from the local state of the scheduler (circuit switching, Figure 10(a)), or provide the source address as part of the message (packet switching, Figure 10(b)). The former technique is simple. It gives a guaranteed performance as the symmetry of the system allows for an analytical solution of the scheduling mechanism. The latter is more complicated but allows also for best effort.

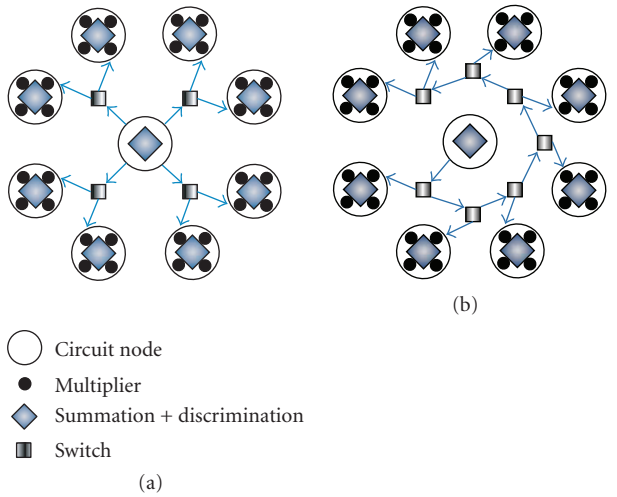

Figure 11: More value routing by multiplexing (a) in space and (b) in time.

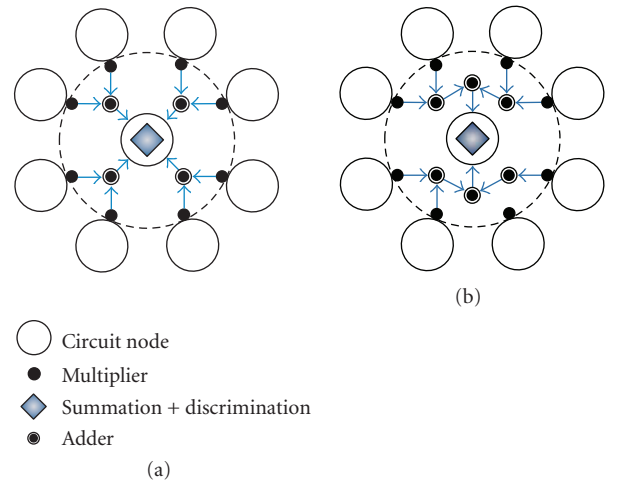

Figure 12: Adder trees combine the network in the producer architecture.

The counterpart of consumption is production. Every node gives values that have to broadcast to all the neighbors. Again where the communication has a limited bandwidth, we need to sequence the broadcast and this can be done in the same way as for the value consumption (Figure 11).

In the case of producer architectures, the nodal output is already differentiated for the different target nodes. Each target node will combine such signals to a single contribution. This combining network is an adder tree that will reduce the n values to 1 in a pipeline fashion. Consequently, this tree can also be distributed, allowing for a spatial reduction in bandwidth. This can be seen from the simple rewrite of the CNN equation as

$$x^c(k) = \sum_{d\in N_r(c)} [a^c y(k)]_d + \sum_{d\in N_r(c)} [b^c u]_d + i^c \qquad (16)$$

The overall processing scheme as shown in Figure 12 is then similar to what has been discussed for the consumer architecture. The main difference is that the communicated values will be larger as they represent products and are therefore of double length. Where the consumer architecture is characterized by "transfer and calculate," the producer architecture is more "calculate and transfer." Furthermore, they both rely on a strict sequencing of the communication, simultaneously loosing a lot of the principle advantage of having a cellular structure.

Also here, we have to look at the way values are broadcasted. In contrast to the consumer architecture, we have as many output values as there are neighbors. This makes for an identical situation and no additional measures are needed, except for the fact that we will not be able to generate all the different products at the same and the sequencing issue pops up again.

In a word-serial/bit-parallel approach, all nodes are broadcasting packaged values simultaneously over a set of "rotating wheels" (Figure 11(b)). For a 1 neighborhood, the cell execution time is $c + d$, where c is the amount of neighbouring cells and d is the core cell cycle. The packet that passed through the network is comprised by the values and for both the row and the column address 2 bits each. So, for an 8-bit value, a packet of 12 bits is needed. The network interface comprises of the packet switch, an input buffer, and an output register. The core node will iterate a parallel multiplication plus addition, followed by discrimination. Characteristic for this approach is the need for a parallel multiplier; furthermore, it can only work on fixed-point integer. The state of a cell is contained in the output register. For a multilayer CNN implementation, the state is salvaged in the local memory. Therefore, the overhead in performing the same operation on an image sequence or different operations on a CNN sequence is moderate.

On the other hand, in a word-parallel/bit-serial approach, all nodes are serially forwarding their values to all neighbors directly (Figure 12(b)). Being circuit switched rather than packet switched, no addresses are transmitted. For a 1 neighborhood, the cell execution time is given by $n+d+\log_2(c)$, where n is the number of bits, d is the core cell cycle, and c is the amount of neighbouring cells. There is no network interface. The local multiplications are done bitwise and are followed by an adder tree that gradually increases in size. Characteristic for this approach is the reduction of the multiplier to a mere AND gate; furthermore, it can be easily adapted to scaled arithmetic and therefore allows a large dynamic range with limited precision.

It appears that the two architectural varieties differ mostly in the balance between wiring and logic, and are therefore dependent on the realization technology. They both show the ability to pass state and output data via the local memory, effectively mapping a levelled hierarchy of CNNs into a single implementation.

5. Wheeled Networks

The attraction of CNNs lies in the feature of local connections. But bandwidth limitation prevents full connectivity, as already addressed in Section 4. Value routing is

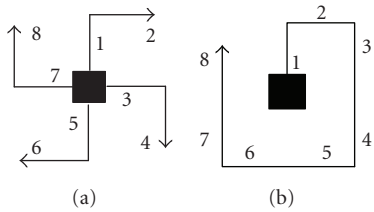

Figure 13: (a) Semiparallel and (b) serial switched broadcasting.

then unavoidable, both in consumer and producer models. Obviously, not all nodes can be active at the same time. In the existing implementations, this is solved by handling one value at time, where a strict sequencing of value transfers is enforced. All nodes in ILVA perform the sequence of compute-and-transfer operations in an identical predefined order. But, as the values flow over the pipeline, they are actually in different states of the iterative procedure. One may say that corresponding nodes are acting out of phase. On the other hand, the active nodes in Caballero are in the same operative phase, but far from all nodes are active simultaneously. Instead, stretching the communication cycle, so that it overlaps with the sequence of operation, reduces routing demands considerably. A side effect, but a highly desired one, is the simplification of the local controller. In this basic concept, values that come into the nodes are immediately absorbed, which allows for evaluation of the nodal equation on the fly.

Looking back at the switched broadcasting employed in Caballero, we see that all nodes send their own values to the orthogonal neighbors that copy the data and forward it in a perpendicular direction to the received one. Theoretically, all nodes will have access to the values of the entire neighborhood after two steps only. But as the CNN grid is grouped into subgroups of 5 nodes (Figure 4), a latency of 10 clock cycles is then introduced. Hence, the actual communication cycle, during which a node is idle, is coupled to the number of nodes in each subgroup. In other words, the short communication pattern of two steps does not boost the performance. On the contrary, it affects the final throughput negatively due to larger routing units and thereby smaller network. Stretching the communication cycle of a 1 neighborhood to 10 clock cycles leads to a semiparallel broadcasting scheme (Figure 13(a)) with reduced routing demands. Table 2 shows the order of sending and forwarding packets. The possible directions are always: North, East, South, and West. Received packets are labeled in accordance with the position of the source node with respect to the current (destination) node. Obviously, the computation needs can be plaited together with the communication cycle.

We propose here a word-serial scheme (Figure 13(b)), where values are sent out in one direction only, but are forwarded to all nodes within the neighborhood serially. Tables 2 and 3 show that stretching the broadcasting of packets yields the same sequence of computation calculation, regardless of the broadcasting scheme. The received packets are consumed directly and overridden by subsequent packets.

Table 2: The semiparallel broadcasting scheme interlaces computation with communication. Characters N, E, S, and W stand for the four main directions on which packets are sent, received, or forwarded. The output value y_{sw}, for example, originates from the southwest neighbor.

Clock cycles	Send	Receive	Forward	Hold	Calculate
1				y_{own}	$a_{own} \cdot y_{own}$
2	N	S		y_s	$a_s \cdot y_s$
3		W	E	—	$a_{sw} \cdot y_{sw}$
4	E	W		y_w	$a_w \cdot y_w$
5		N	S	—	$a_{nw} \cdot y_{nw}$
6	S	N		y_n	$a_n \cdot y_n$
7		E	W	—	$a_{ne} \cdot y_{ne}$
8	W	E		y_e	$a_e \cdot y_e$
9		S	N	—	$a_{se} \cdot y_{se}$
10					$f(\cdot)$

Table 3: The serial broadcasting scheme yields in a same sequence of computation as in the semiparallel one.

Clock cycles	Send	Receive	Forward	Hold	Calculate
1				y_{own}	$a_{own} \cdot y_{own}$
2	N	S		y_s	$a_s \cdot y_s$
3		W	E	y_{sw}	$a_{sw} \cdot y_{sw}$
4		N	S	y_w	$a_w \cdot y_w$
5		N	S	y_{nw}	$a_{nw} \cdot y_{nw}$
6		E	W	y_n	$a_n \cdot y_n$
7		E	W	y_{ne}	$a_{ne} \cdot y_{ne}$
8		S	N	y_e	$a_e \cdot y_e$
9		S	N	y_{se}	$a_{se} \cdot y_{se}$
10					$f(\cdot)$

Consequently, the need of a local memory to hold the values of all neighboring nodes is removed. A single register is used to hold the current packet before it is multiplied by corresponding template coefficient that resides in a local memory (BRAM). Traditionally, the same memory is used to hold a look-up table representing the discrimination function.

Looking back at equation (1), we see that the part involving u^d-values together with the bias remains unchanged during the iterative process of computing the new nodal state and thereby the new output. Thus, the broadcast will first handle the inputs u^d and the bias, and the resulting constant is locally stored. On every next iteration, the result of broadcasting the cell outputs will be added to the stored constant to give the new cell output. There is no need anymore for a global control, and the network interface is very simple.

In order to simplify the control demands, the addressing of template coefficients is obtained through a base-address register that holds the higher address part, and indexing of the lower address part is carried out by the nodal controller itself. As the BRAM has the configuration of a 2 K entries memory, the base-address register does not need to be wider than 6 bits as shown in Figure 14. For a 1

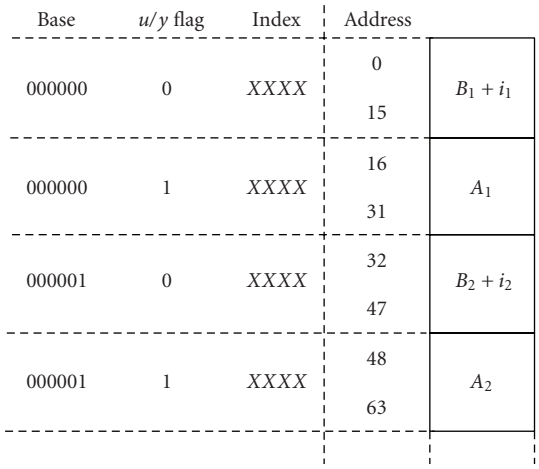

Figure 14: Address space of the nodal template memory.

neighborhood, 19 coefficients need to be stored for each template: 9 control coefficients, 9 feedback coefficients, and a bias. As the control coefficients b_c^d, a_c^d, and the bias are used in the first iteration to compute the constant, they are stored sequentially and can be addressed by 4 bits. A u/y-flag, set by the nodal controller, allows the addressing of the corresponding feedback coefficients a_d^c. The base address picks out the correct template.

Also a number of templates are prestored in the local memory. But other templates can be sent by the user to every node in the network through the FIFO elements. These FIFO elements serve originally to bring the external inputs u into the nodes, but their functionality can easily be extended to cover the handling of template transmission. At first glance, this additional mechanism seems to add on the complexity of the nodal controller, but a proper usage of information stored in the header of the received FIFO packets keeps the complexity at minimum.

In principle, control demands are reduced to a mux-enable signal and addressing of the template memory. A single register is used to hold one value only according to Table 3. The content of the register is overridden as a new value is received or locally produced. The schematic design of the node is shown in Figure 15. Here, the local memory is merged with the discriminator, as it also holds a table of precomputed values to map the state onto a certain output.

6. Boundary Nodes

The functional correctness of any CNN system depends on the handling of the boundary nodes, as these nodes lack a complete neighborhood. Traditionally, the effect of boundary conditions is modeled by adding virtual nodes on the edge of the network. The problem here is further complicated by the asymmetry of the prescheduled communication pattern: boundary nodes experience different needs depending on their position in the network. Figure 16(a) illustrates the disturbed communication cycle for top boundary nodes. The situation is even worse for the corner nodes (Figure 16(b)).

Table 4: Additional actions in boundary nodes remove the need of virtual nodes.

Step	Top boundary node	Bottom boundary node
(1)	Send E (instead of N); store W value locally	Use own value; do not update u/y register
(2)	—	—
(3)	Use W value (instead of u/y-register value)	—
(4)	Use W value	—
(5)	Use own value	—
(6)	Forward own value W	—
(7)	Forward own value S	Forward W; receive E
(8)	—	Forward own value W; receive E

Actually, not only boundary nodes are affected by the incompleteness of broadcasting but even close-to-boundary nodes as well (Figure 16(c)).

Employing the traditional approach of adding virtual nodes is not as simple as it may seem. Besides being unable to solve the problem completely, it adds on the network size. In any prescheduled communication scheme, virtual nodes should follow the sequence of sending (and eventually forwarding) of values that is accommodated by all regular nodes in the network. This works fine for close-to-boundary nodes (Figure 17(a)), but the communication path is still incomplete for boundary nodes. It is clear from Figure 17(b) that top boundary nodes will not receive any data in steps (4), (5), and (6). In other words, the partially asymmetric transfer cycle necessitates the existence of two (!)layers of virtual nodes to achieve completion.

We aim here for a total removal of the need for virtual nodes. This is possible by slightly changing the communication pattern of boundary nodes. Let us consider top and bottom boundary nodes. Then, the actions listed in Table 4 have to be performed in addition to the regular functionality of the node, mainly when a zero-flux boundary condition is used. For fixed boundary condition, most of the sending/forwarding is redundant as all boundary nodes will need to store a single fixed value only that can be used instead of the received value. Implementing the actions in Table 4 introduces the need for boundary nodes to, sometimes, send or receive two packets simultaneously, which requires a remarkable redesign of the nodal controller and the router in addition to the need of an extraregister that keeps one value (W value in the table). Once again, different boundary nodes will require different refinements. This is of course better than the virtual nodes approach, but still increases the area considerably. A better solution makes use of the existing routing mechanism to forward boundary conditions. We call it "swing boundary broadcasting" as each boundary node will send its own value to one neighboring boundary node and then to the other boundary node in the opposite direction. Due to the use of duplex lines between the nodes, the internodal connections have to be idle for one time step in between (Figure 18). In this case, all boundary nodes will

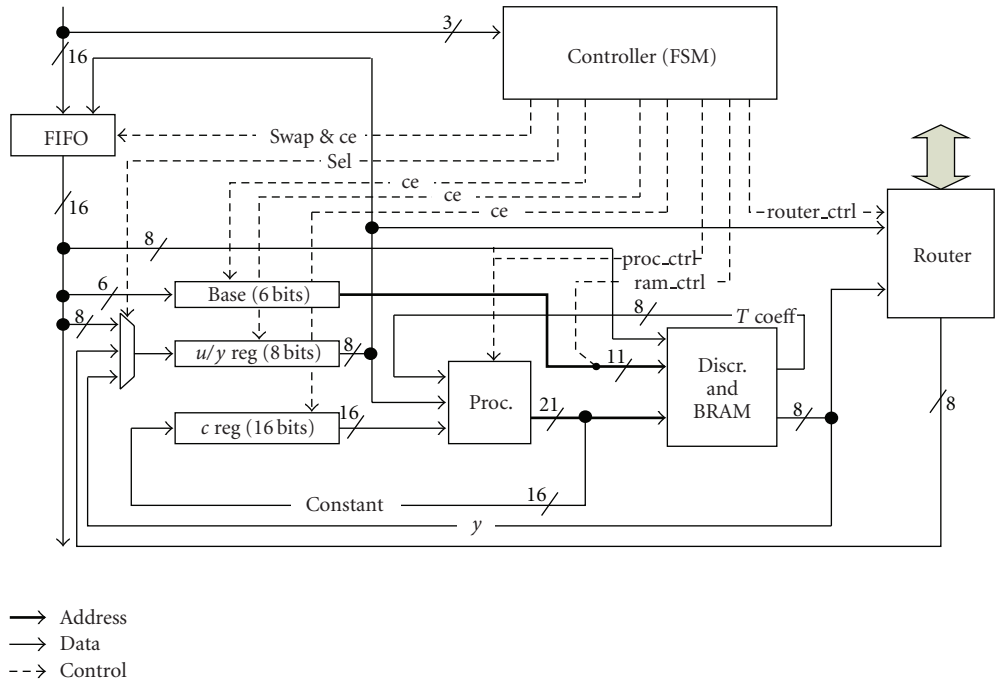

Figure 15: Schematic view of a serial CNN node.

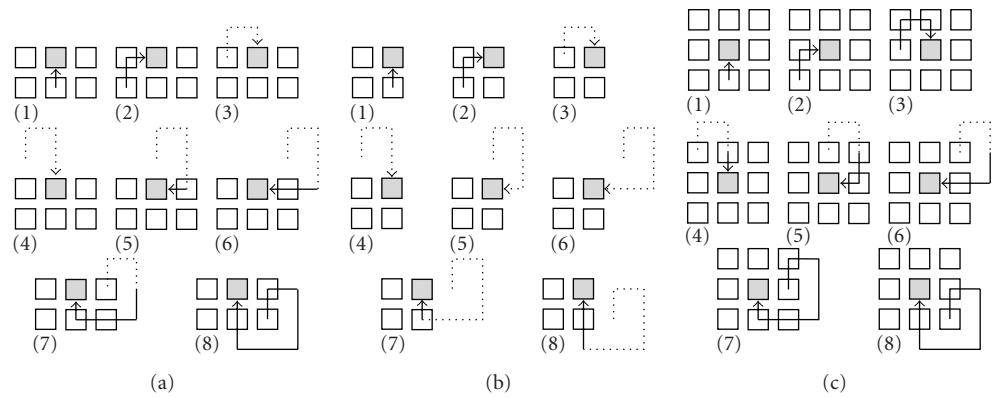

Figure 16: (a) Top edge, (b) right upper corner nodes, and (c) even close-to-boundary nodes have an incomplete communication cycle. Squares represent regular nodes; while the dotted lines show which part of the packet path is missing. The node of interest is shaded.

have the value of their neighboring boundary nodes available locally. This requires two additional buffering elements to store the values, but the effect on area utilization is kept at a minimum. Overall, 3 time steps are introduced for each newly calculated y value.

7. System Architecture

The development of classical computer architecture has shown an emphasis on computing acceleration by pipelining the central processing unit [18]. More and more the memory bottleneck became a concern. Of late, attention moved to more spatially distributed methods, such as networked tiles, which offer inherent parallelism and local storage. The underlying assumption is that sequencing instructions over the local node takes the pressure away from the memory access by the many parallel executing tasks.

We see the same principle back in the research reported in this paper. On one hand, we aim to have as much nodes executing in parallel as possible. This poses a severe burden on the memory bandwidth. Therefore, it is required to do more locally. From inspection of existing CNN applications, one finds that data is accessed in memory more than once. Therefore, bringing the amount of accesses down to 1 or less will easily pay the bill.

The newest digital implementations reach a "close to analogue" capacity by merging temporal distribution of many nodes inside a single CNN cell implementation and spatial distribution of many cells within a network. This

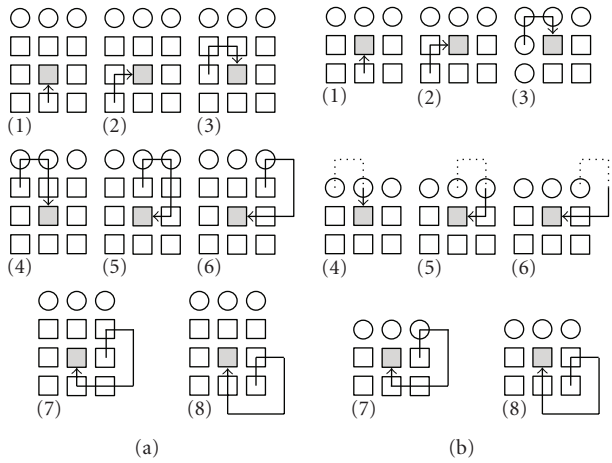

Figure 17: (a) The absence of neighbors is salvaged by adding a single layer of virtual nodes. (b) For boundary nodes more than one layer of virtual nodes is needed.

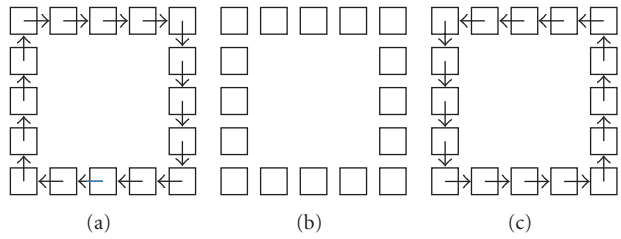

Figure 18: Swing broadcasting allows distributing of boundary conditions in (a) three steps clockwise and (c) anticlockwise. (b) For proper functionality on the duplex lines, a separating idle step is introduced.

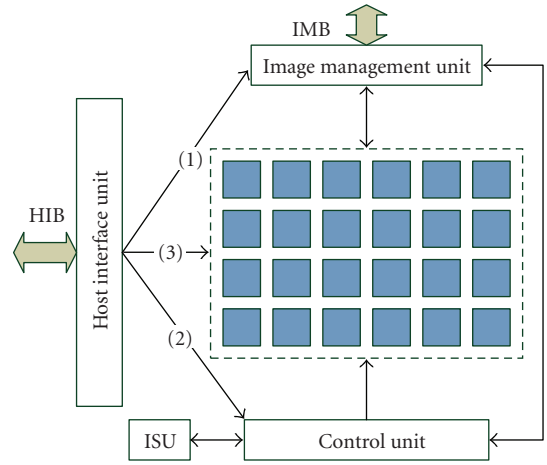

Figure 19: External view of the CNN architecture.

allows entire CNN programs to be handled with minimal memory access. This invites to the definition of a system architecture and an appropriate application programming interface.

The CNN Instruction Set Architecture defines the exterior of the CNN Image Processor in terms of signals and visible memory locations. The overall CNN ISA is depicted in Figure 19. Overall we find two separate bus systems: the image memory bus (IMB) and the host interface bus (HIB), both with an R/W signal and strobe address and data bus.

A host must be able to control the overall functionality of the system by sending instructions and cloning templates and by setting a number of configuration parameters. The communication is handled by a host interface unit (HIU) that receives the requests from the host over the HIB and forwards them to the system using a wishbone bus. The HIU is as well responsible for data delivery to the host. Two different FIFOs are used, one for acquiring host requests and one for putting out data to the host.

The instructions cover four modes of operation. Firstly, it sets the image window parameters, which allows zooming in and out of an image while applying the CNN transformations. Secondly, it sets the CNN structure by loading the program, template, and discrimination tables. The program can contain logical, arithmetic, and CNN-specific operations. Thirdly, it can insert debug facilities. Finally, it can also execute the stored instructions in a variety of run steps.

The control unit (CU) has direct communication to the CNN core and the HIU through wishbone buses. It is built with the concept of pipelining in mind and consists of two main components: instruction fetch and a controller. The latter is in turn composed of two major components. One is the actual instruction decoder for the proper template, while the other generates CNN-enable and instruction-done signals depending on the number of iterations and whether equilibrium is reached or not.

8. Discussion

The moral of the serialized broadcasting approach is that the transfer needs to be sequenced when the communication bandwidth is limited due to area restrictions. The nodal control demands are kept at a minimum by interlacing communication and computation needs. Local storage needs are reduced as well, due to the need of holding one value only locally at any time step. A realization of the serial broadcasting approach, hosted on a Xilinx Virtex-II FPGA, shows reduced area utilization (Figure 20). Special attention should be paid to the smaller network interface of the switched serial approach compared to the 2-step approach employed in Caballero. The simplicity of the serial scheme eliminates the complexity of the router, which affects the total size of the node. Note that the serial architecture occupies fewer slices than the almost-interface-less ILVA architecture in spite of that the latter requires less flip-flops and equal number of LUTs. This has probably to do with a more balanced logic usage among the functional components (Figure 21).

We have also found that using state machines not only leads to architectural rigidity but also to degraded performance. For instance, 30% of the utilized area in the serial broadcasting scheme is occupied by the controller, that

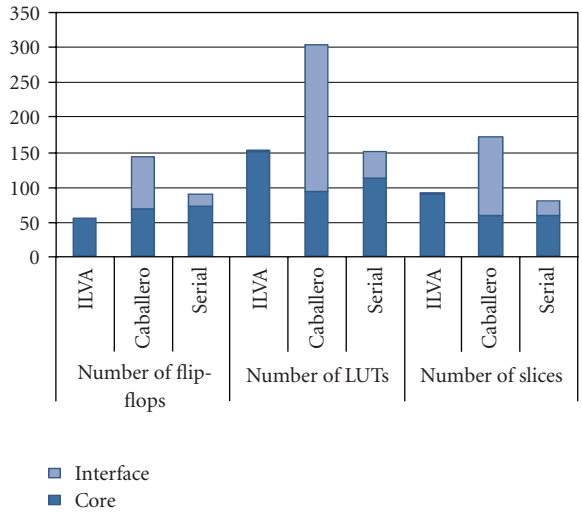

Figure 20: Area utilization per node compared to ILVA and Caballero architectures shows that nodal interface is kept at minimum which improves the overall logic utilization. The vertical axis reflects the number of flip-flops, LUTs, or slices of the three different architectures.

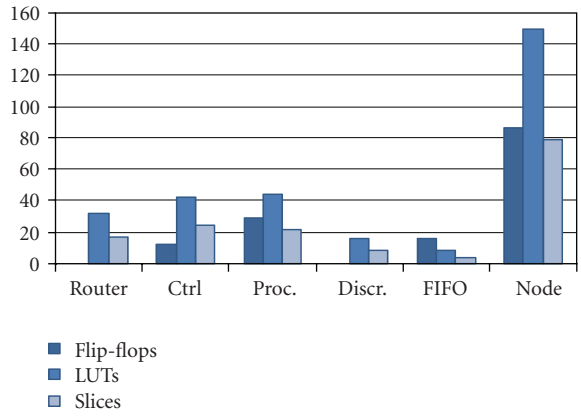

Figure 21: Area utilization of the different components with serial broadcasting scheme.

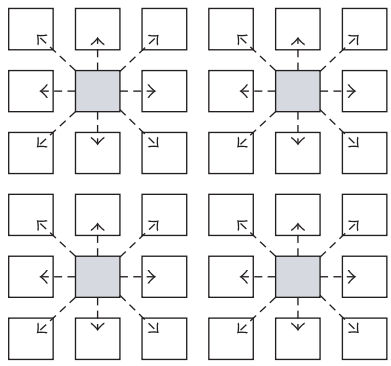

Figure 22: Semiglobal control with one controller per group of nodes.

is, state machine (Figure 21). One way to eliminate the need of the nodal controller, at least partially, is by transferring all values in a source-addressed packet. The original data-only packet used previously is padded with a small header containing the position of the source node in the grid. Hence, the packets carry their addressing information in the header, which can be exploited in two different ways.

In a traditional approach, the packets will be stored in distinct destination registers accordingly. In this case, the number of required registers equals the number of neighbouring nodes. For the minimal 1 neighborhood, this means 9 registers. This is not as bad as it sounds. Registers are mapped on flip-flops only and no LUTs are used. The present imbalance in the number of LUTs and flip-flops, shown in Figure 21, allows for more flip-flops without affecting the overall number of slices. In this way, an eventual architectural rigidity is removed with no impact on area utilization.

A better approach uses the intrinsic positioning information in the header to address the local template memory of the current node. The nodal equation, (1), performs then in the order the packets come in. The logic required for the addressing of the value/coefficient pairs is greatly reduced through the use of a mirrored binary numbers of both the rows and the columns. In case of a 1 neighborhood only 2 bits for the row and 2 bits for the column address are required. In general, we need only $2(r + 1)$ bits, where r is the neighborhood.

It is also possible to merge the local controllers. The network is divided into small groups with each a single controller (Figure 22). This semiglobal control approach does not affect the guaranteed performance but will lead to logic optimization. It adds some wiring overhead and therefore slow down the system but the gained amount of logic from reducing the number of nodal controllers is far much larger. Attention has to be paid; so the average wire length is not increased to such a limit that the potential benefits of the CNN are lost [19]. The rate of one controller per neighborhood seems to be a good tradeoff.

By virtue of the slim network interface and limited need for run-time storage within the cell, the word-serial implementation can easily be extended to handle multilevel structures. In principle, for every node the run-time status is stored in the connected BRAM and loaded from there when the execution moves from one level to the other. This makes that the network is virtually much larger than the actual number of implemented nodes sets as a limit.

Wrapping the CNN in the ISA scheme takes only a 1–5% overhead. At this price, it becomes feasible to execute a program of standardized instructions on a variety of CNN implementations. It can as well work as coprocessor (Caballero) and as image streamer (ILVA). Especially noteworthy is the rich set of debug facilities to support work in the development phase of a project.

The ISA is especially helpful when the actual parameterization for the network is not clear during development, but should not influence the application at hand. It has become practical because the virtual network size has been raised from a meager 144 cells to 4096, and can probably be raised even higher. This makes a digital CNN a practical

alternative for image processing. As such networks do not have a global control, their intrinsic speed is much higher than the usual, and speeds of 400 frames per second have been demonstrated.

In a typical hand-vein application [20], we find that different templates need to be applied to the same image, and the two results need to be used in a next dyadic operation to bring a single result on which again two different templates are applied, and so on. With the current implementation, we can reduce the amount of external memory access, as each frame only has to be loaded once. Additional registers are simply added for each suboperation. Factually, in this application we need to go through 7 subsequent CNN layers and never have to reload from external memory. This provides us with an amazing 20x higher performance, making real-world, real-time, and real-power product applications possible.

References

[1] R. Schiffelers, R. van den Berg, J. van den Braak, H. S. Bhullar, S. T. de Feber, and M. Klaarwater, "Epics7B: a lean and mean concept," GSPx, April 2003.

[2] L. O. Chua and L. Yang, "Cellular neural networks: theory," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1257–1272, 1988.

[3] L. O. Chua and L. Yang, "Cellular neural networks: applications," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1273–1290, 1988.

[4] H. Harrer and J. A. Nossek, "Discrete-time cellular neural networks," *International Journal of Circuit Theory and Applications*, vol. 20, no. 5, pp. 453–467, 1992.

[5] G. Liñán, S. Espejo, R. Domínguez-Castro, and Á. Rodríguez-Vázquez, "ACE4k: an analog I/O 64×64 visual microprocessor chip with 7-bit analog accuracy," *International Journal of Circuit Theory and Applications*, vol. 30, no. 2-3, pp. 89–116, 2002.

[6] F. Wenhai, W. Cheng, and L. Spaanenburg, "In search for a robust digital CNN system," in *Proceedings of the 10th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '06)*, pp. 1–6, Istanbul, Turkey, August 2006.

[7] S. Malki, *Discrete-time cellular neural networks implemented on field-programmable gate-arrays to build a virtual sensor system*, Lic. thesis, Lund University, Lund, Sweden, 2006.

[8] Á. Rodríguez-Vázquez, G. Liñán-Cembrano, L. Carranza, et al., "ACE16k: the third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs," *IEEE Transactions on Circuits and Systems I*, vol. 51, no. 5, pp. 851–863, 2004.

[9] B. Khailany, W. J. Dally, U. J. Kapasi, et al., "Imagine: media processing with streams," *IEEE Micro*, vol. 21, no. 2, pp. 35–46, 2001.

[10] Á. Rodríguez-Vázquez, R. Domínguez-Castro, and F. Jiménez-Garrido, "The eye-RIS CMOS vision system," in *Analog Circuit Design*, H. Casier, M. Steyaert, and A. H. M. Van Roermund, Eds., pp. 15–32, Springer, Dordrecht, The Netherlands, 2008.

[11] M. H. ter Brugge, *Morphological design of discrete-time cellular neural networks*, Ph.D. thesis, Rijksuniversiteit Groningen, Groningen, The Netherlands, 2005.

[12] Z. Nagy and P. Szolgay, "Configurable multi-layer CNN-UM emulator on FPGA," in *Proceedings of the 7th IEEE Workshop on Cellular Neural Networks and Their Applications (CNNA '02)*, pp. 164–171, World Scientific, Frankfurt, Germany, July 2002.

[13] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[14] A. DeHon, "Density advantage of configurable computing," *Computer*, vol. 33, no. 4, pp. 41–49, 2000.

[15] D. Wiklund and D. Liu, "SoCBUS: switched network on chip for hard real time embedded systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IDPDS '03)*, pp. 1–8, Nice, France, April 2003.

[16] K. Goossens, J. Dielissen, and A. Rădulescu, "Æthereal network on chip: concepts, architectures, and implementations," *IEEE Design& Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.

[17] H. Corporaal, "Transport triggered architectures used for embedded systems," in *Proceedings of the International Symposium on New Trends in Architectures*, Gent, Belgium, December 1999.

[18] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, Calif, USA, 4th edition, 2007.

[19] V. Zhirnov, R. Cavin, G. Leeming, and K. Galatsis, "An assessment of integrated digital cellular automata architectures," *Computer*, vol. 41, no. 1, pp. 38–44, 2008.

[20] S. Malki and L. Spaanenburg, "Hand veins feature extraction using DT-CNNS," in *VLSI Circuits and Systems III*, vol. 6590 of *Proceedings of SPIE*, pp. 1–8, Maspalomas, Spain, May 2007.