An exploration of digital CNN implementations

Malki, Suleyman; Spaanenburg, Lambert

Published in:
Proceedings SSOCC

2004

An exploration of digital CNN implementations

Suleyman Malki and Lambert Spaanenburg

Lund University, Institute of Information Technology,
P.O.Box 118, 22 100 Lund (Sweden)
Email: suleyman@it.lth.se, lambert@it.lth.se

Abstract. *Computationally hard problems, such as operations on n-dimensional maps, are in need of efficient solutions. Cellular Neural Networks have this promise. This paper explores digital realizations of such computational paradigms through the case of real-time image processing. It is shown that a behaviour-based spatial unrolling of the time-critical numerical convergence loop outperforms a classical function-based architecture. An FPGA implementation on Xilinx Virtex-II illustrates a 50 times higher throughput, bringing intelligent vision into reach.*

Keywords *– Cellular Neural Network, Virtual Sensing, Real-Time System, Image Processing, Spatial Architecture*

I. Introduction

Their local connectivity gives Cellular Neural Networks (CNNs) a first-hand advantage to VLSI implementation with very high speed and complexity. A fully digital implementation relies on the field-programmable gate-array (FPGA) for reasons like explicit parallelism and reconfigurability. FPGA architectures provide large amounts of Configurable Logic Blocks with optimized Block Select RAM and multiplier macros. Such macros, specialized for popular digital signal processing functions, increase the flexibility to map modular neural structures.

However, the limitation of hardware resources in an FPGA makes it necessary to use every hardware element as much as possible. Scheduling the computational process is usually applied to achieve this. Such unravelling in time gives it the name "temporal computing", in contrast to "spatial computing", where the process is unravelled in space to reduce spurious latency [1]. The facility of spatial computing makes the FPGA already very popular as a hardware accelerator.

A DT-CNN (Discrete Time CNN), introduced by Harrer and Nossek [2], is a regular multidimensional grid of locally connected cells. Each cell *c* communicates directly with its *r-neighbours*, i.e. a set of cells within a certain distance *r* to *c,* where $r \geq 0$. E.g. if r = 1 we have 3 × 3 neighbourhood and if r=2 we have 5 × 5 neighbourhood. Still a cell can communicate with other cells outside its neighbourhood due the network propagation effect.

The state of a cell c, denoted x^c, depends mainly on two factors: the time-independent input u^d to its neighbours *d* and the time-variant output $y^d(k)$ of its neighbourhood. The neighbourhood always includes c itself. Equation 1 describes this dependency in a discrete time k, while Figure 1 gives an illustration.

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

Spatially invariant CNNs are specified by a control template A containing a_d^c, a feedback template B containing b_d^c, and the cell bias i = i^c. Node template, *T=<A, B ,i>*, determines together with input image *u* and an initial output *y(0)* completely the dynamic behaviour of the DT-CNN. Convergence to a stable state will be achieved after a number of iterations *n,* i.e. at time *k=n*.

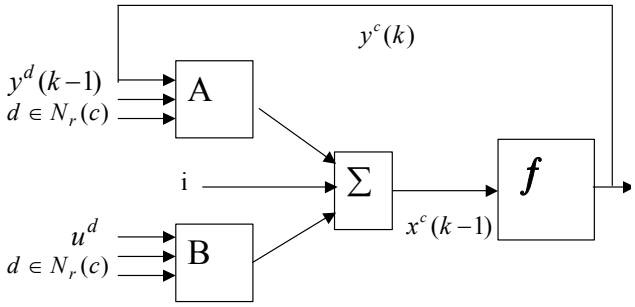

Figure 1 Block diagram of a DT-CNN cell.

This paper stresses the exploitation of the built-in macros to spatially unroll the local feedback. After some general consideration, we discuss in section 0 the image processing considerations that determine the architectural options. Then a classical pipelined architecture of a DT-CNN system is introduced. Ensuing section V reviews the ILVA architecture, followed by a short evaluation of these architectural alternatives.

II. General Considerations

Although neighbourhoods of any size are allowed in DT-CNN, it is almost impossible to realize large templates. Limited interconnectivity imposed by current VLSI technology demands that communication between cells is only local. In this work, we restrict ourselves to the use of 3 × 3 neighbourhood, where templates A and B are 3 × 3 matrices of real-valued coefficients. Additionally, the input range of a DT-CNN cell is restricted to [-1, +1], due to the fact that grey-scale level is commonly used to the purpose of image processing. A value of –1 represents a white pixel and a value of +1 represents a black pixel. Other values represent grey levels in-between. Although floating-point representation

of real-valued numbers is generally preferred, we employ fixed-point representation due to its friendliness in hardware realizations, especially for the use of multiplier primitives in FPGA. Furthermore, before any computations can be done, the image is assumed to be stored in an external storage space.

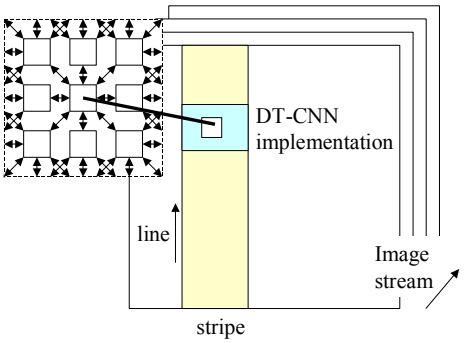

Figure 2 Dimensionality of CNN image processing.

The operation of a DT-CNN on images covers many dimensions. The local operation is performed in a two-dimensional plane (width & length) and iterates in time. Due to the limited capacity of the CNN implementation this has to be repeated over image slices and iterates over the surface to handle potential wave propagation. Finally, the operation is performed on sequences of images (Figure 2). All this has to be facilitated on the two-dimensions in a FPGA. Consequently, the dominating architectural question is: how to reduce the dimensions from the functional requirements to the platform facilities?

Figure 1 depicts the requirement of all data to be present simultaneously in order to compute the state, and then the output, of a certain neuron (cell). Due to the limited wiring of the FPGA, this is almost impossible to implement. There are two aspects of time that can be spatially unrolled: (a) the temporal ordering of the images parts, which we discuss in [3], and (b) the time steps in convergence of each nodal operation to reach consensus with its neighbours. The latter will be discussed here from both temporal and spatial view.

III. Principle of Image Processing

The RGB standard reigns in the field of image processing. It generates any colour by mixing the intensities of the three basic ones, red, green and blue. Unfortunately, the RGB standard does not make it easy to extract grey-level; the digital standard CCIR-601 is more suitable. It is defined in the YCbCr colour space, where the luminance Y-channel holds the grey intensity and channels Cb and Cr contain colour information. In this way, the grey level information is obtained without any need of conversion. An added advantage is that the CCIR-601 also provides timing information.

The processed image may originate from a camera or be artificially composed. A typical platform contains a PC-interface, and PAL-standard camera plus monitor that need an additional interface board to produce the map in the CCIR-601 standard. We used a Video Board from Axis Communications, Lund (Sweden), based on Philips Enhanced Video Input Processor (EVIP) version SAA711A and Digital Video Encoder (ConDENC) version SAA7121.

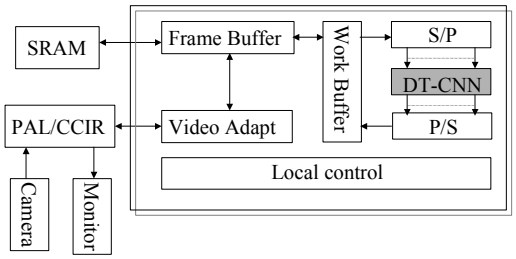

Figure 3 The CNN-based image processing system

The camera generates frames with a rate of 25 frames per second, where each frame has the actual size of 720x576 pixels. The grey level part of a pixel is 8 bits wide, which needs to be converted to a signed fixed-point number.

According to the PAL standard, a video frame is divided into two fields: an odd field containing all odd lines and an even field containing all even lines. All compliant PAL equipment processes the odd field first ("interlaced video"). A Frame Buffer is needed, because the 3x3 neighbourhood can't be obtained until both odd and even lines are available. Due to the big storage needed and the limited resources on the FPGA in use, an external SRAM is used. See Figure 3. Communication with this SRAM is provided by an SRAM I/O, which uses two one-directional data busses. SRAM I/O is shared between the DT-CNN and a Video I/O. The major drawback of the external SRAM is its low speed (27 MHz) compared to what our DT-CNN system must handle. This motivates the need of fast and flexible Work Buffer in-between. This buffer fits two stripes and uses Block Select RAM primitives on the FPGA.

IV. Temporal Architecture

The nodal equation (1) consists mainly of two accumulated sequences of multiplications. Every sequence contains nine multiplications, and therefore can be realized in nine stages. In this case, pipelining each sequence gives a simple design, which will reduce memory access required per pipeline step. Data dependencies between scan-lines in an image are then stretched to cover all nine stages of the pipeline. Pixels needed to perform the desired computation of the output for one cell, are fetched from three series of registers connected to the pipeline. Based on the observation that both multiplication sequences are independent, the desired

network behaviour is implemented as two 9-stage pipelines per one DT-CNN node. Output is obtained by thresholding the sum.

One pipeline, consisting of 18 multipliers and additional logic, is needed for every column of the image. Due to the organisation of multiplier macros in the target FPGA, only six nodes can be mapped. Figure 4 shows a node net consisting of 4 nodes.

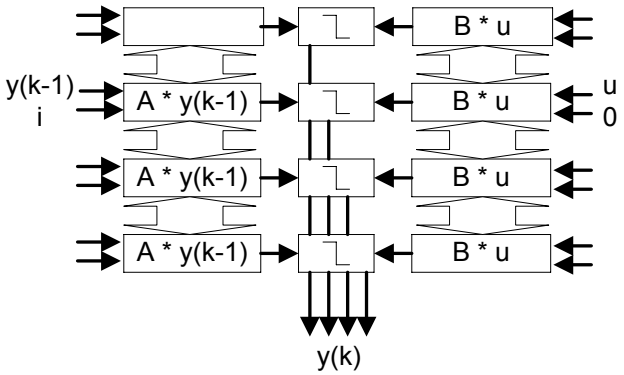

Figure 4 An overview of a temporal node net.

V. Spatial Architecture

The early ILVA implementation [4] is focussed on single images. To delineate the temporal effects of line-oriented image inspection and CNN iteration, it unrolls the iteration on the pixel timing axis [5]. The principle of operation is illustrated in Figure 5. The local interaction between CNN nodes in a 1-neighbourhood requires 3 lines of 3 pixels to be present. As the pixel lines come in one-by-one, this can be implemented as an array of CNN nodes with a feedback that stores the history of the passed two lines, assuming one RAM available per CNN node.

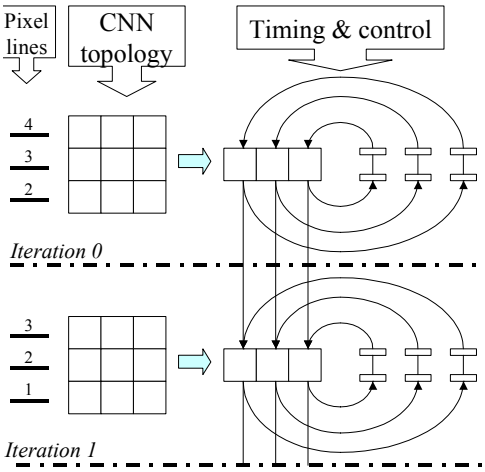

Figure 5 Unrolling the iteration loop.

This makes all values for 3 lines to be available for the computation of a single discrete iteration. The result is passed to the next array for CNN nodes, synchronized with the arrival of new pixel lines, where the next iteration is performed with the same compute & store principle as in the first array.

The floor plan of the FPGA realization shows this concept as follows. We let each column represents a line in the image stripe and the columns will then form iterations performed on the image. In this way grid structure and number of iterations are implemented by using columns of nodes, and image size is handled by slicing the image. Coupling scheme of multiplier and Block Select RAM macros in the target FPGA limits the number of iterations to 5. In other words, our CNN system consists of six columns of nodes where the maximum of 24 pixels can be handled in parallel.

The nodal formula consists of two parts: the constant part $\sum A.x + i$ (implemented as U cell) and the state dependent remainder $\sum B.u$ (implemented as Y cell). At every iteration, the constant part is added to the newly computed state dependent part. Or in other words, the incoming line can be consumed directly in the computation of the constant that in turn will be passed to every node at the moment that corresponds with the consumed pixel. This way, the line-by-line pixel flow becomes synchronized with unrolling the convergence loop (Figure 6).

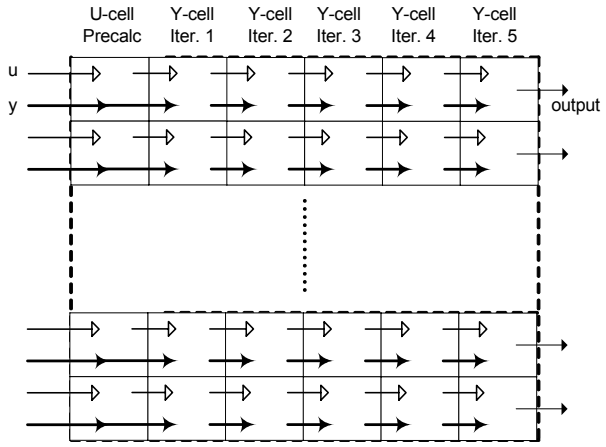

Figure 6 Dataflow of U and Y matrix.

Intuitively, resources of the FPGA should be grouped in nodes or Processing Elements (PEs), where each node corresponds to a cell in the CNN, which in turn represents a pixel in the image stripe. This model is preferred since it maps very well to the mathematical model of a DT-CNN. The major drawback seemed that loading data to and from the FPGA contributes with a tangible loss in performance and bandwidth. In [3] it is discussed how packet switching techniques can solve this problem.

VI. Discussion

Depending on interpretation of the nodal equation, temporal and spatial approaches use different data representations. Common for both approaches is the use of fixed-point instead of float-point representation of real-valued numbers, which usually is preferred in the related literature. Table 1 shows how different data are represented, comparing the temporal approach with Ilva implementation of the spatial approach. First part is the number of bits in use. The notation [x+y] means that the number consists of x-bits integer part and y-bits fractional part.

	Temporal	*Spatial / Ilva*
Input Data	8 , [8+0]	8 , [1+7]
Template Coefficients	16 , [7+9]	8 , [4+4]
Bias	36 , [18+18]	8 , [8+0]
Feedback	Not defined	20 , [9+11]
Constant Output	8 , [8+0]	8 , [1+7]

Table 1 Representation of data.

Both temporal and spatial implementations target a Virtex-II 6000 FPGA from Xilinx. Three different implementations of the spatial architecture have been considered: Ilva, Halvar and Wickie. They illustrate a different balance between cycles per iteration and maximum clock speed. Table 2 shows a comparison between the implementations. These results were obtained after synthesis with Synplify. For Ilva, the simulation was done with a design of 22 rows only.

Spatial and temporal architectures differ in a number of ways. Firstly the modular structure of the spatial design offers a better usage of the distributed memory than the sliced structure of the temporal design. Modules based on the distribution of multiplier and SRAM macros provide for localization of concerns, which in turn opens the way for reconfiguration of the module hardware in order to boost performance further. Secondly, a sliced design with pipelines for the succession of multiplying-adder operations for a single DT-CNN node needs a frequent access to the external SRAM, while the modular design allows unrolling the design for the required computational iterations. This eases the demands on external SRAM access and therefore leads to an intrinsically better performance. Thirdly, the need for a system controller within the iteration loop for the temporal approach complicates the implementation considerably. As further the pipeline slice need to be located in the width of the FPGA, while the slices of the modules can use the length of the FPGA, the maximum capacity of a spatial architecture in terms of parallelized pixels is about 5-6 times higher than that of a temporal architecture.

The last point of attention is the accuracy of the operations. At present we have used 8 bits representations for pixels input and output, since it has been applied with success in the analogue alternative architecture. As noted in [6], there are reasons to believe that more accuracy is required. In both architectures, this cannot be easily accommodated because the multiplier macros are fixed at maximum 18 x 18 bits.

The low clock rate of 17 MHz, mainly caused by the complexity of the pixel address generation, is a major drawback for the temporal architecture. There is of course room for more optimization, but maximum theoretical performance remains low, about 70 Mpixels/sec. Together with other observations, this leads to the conclusion that the spatial architecture brings clear benefits.

	Halvar	*Ilva*	*Wickie*	*Temporal*
Slice utilization	50%	37%	47%	13%
LUT utilization	28%	28%	36%	12%
Multipliers/BSRAMs	144/144	132/132	144/144	78/0
Clock rate(MHz).	106	100	103	17,3
Cycles per iteration	15	10	13	17
Throughput	170/180	205/220	185/190	4.1

Table 2 Comparison of different implementations.

Acknowledgments

We are indebted to the 2002 RedBeard Team [7] for the development of the spatial architecture and Sebastian Rasmussen and Tor Silfverberg for the development of the temporal architecture.

References

[1] A. deHon, "Reconfigurable Architectures for General-Purpose Computing", *Ph.D. thesis,* MIT, Cambridge (USA), 1996.

[2] H. Harrer and J. A. Nossek, "Discrete-Time Cellular Neural Networks", *International Journal of Circuit Theory and Applications*, **20**, 453-467, September 1992.

[3] S. Malki, L. Spaanenburg and N. Ray, "Neural vision sensors for surface defect detection", to be presented at *IJCNN* (Budapest, July 2004).

[4] S. Malki, and L. Spaanenburg, "CNN Image Processing on a Xilinx Virtex-II 6000", *Proceedings ECCTD'03* (Krakow, Poland, 1-4 September 2003) pp. 261-264.

[5] Z. Nagy and P. Szolgay, "Configurable Multi-Layer CNN-UM Emulator on FPGA", *Proceedings 7th IEEE Int. Workshop on Cellular Neural Networks and their Applications*, 164 – 171, 2002.

[6] R. Tetzlaff, "Solitons in Cellular Non-linear Networks", *Proceedings ECCTD*, Vol. II, 434-439, 2003.

[7] L. Spaanenburg, "*The RedBeard files, Experiments in Application-Configurable VLSI Architectures*". 2003. URL: http://www.it.lth.se/lambert/RedBeard (2004-03-18).