EURASIP Journal on Advances in Signal Processing

CNN Technology for Spatiotemporal Signal Processing

Guest Editors: David López Vilariño, Diego Cabello Ferrer, Víctor Manuel Brea Sánchez, Ronald Tetzlaff, and Chin-Teng Lin

CNN Technology for Spatiotemporal Signal Processing

EURASIP Journal on Advances in Signal Processing

CNN Technology for Spatiotemporal Signal Processing

Guest Editors: David López Vilariño, Diego Cabello Ferrer, Víctor Manuel Brea Sánchez, Ronald Tetzlaff, and Chin-Teng Lin

Copyright © 2009 Hindawi Publishing Corporation. All rights reserved.

This is a special issue published in volume 2009 of "EURASIP Journal on Advances in Signal Processing." All articles are open access articles distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Editor-in-Chief

Phillip Regalia, Institut National des Télécommunications, France

Associate Editors

Adel M. Alimi, Tunisia
Kenneth Barner, USA
Yasar Becerikli, Turkey
Kostas Berberidis, Greece
Jose Carlos Bermudez, Brazil
Enrico Capobianco, Italy
A. Enis Cetin, Turkey
Jonathon Chambers, UK
Mei-Juan Chen, Taiwan
Liang-Gee Chen, Taiwan
Huaiyu Dai, USA
Satya Dharanipragada, USA
Kutluyil Dogancay, Australia
Florent Dupont, France
Frank Ehlers, Italy
Sharon Gannot, Israel
M. Greco, Italy
Irene Y. H. Gu, Sweden
Fredrik Gustafsson, Sweden
Ulrich Heute, Germany
Sangjin Hong, USA
Jiri Jan, Czech Republic
Magnus Jansson, Sweden
Sudharman K. Jayaweera, USA
Soren Holdt Jensen, Denmark
Mark Kahrs, USA
Moon Gi Kang, South Korea
Walter Kellermann, Germany
Lisimachos P. Kondi, Greece
Alex Chichung Kot, Singapore
C.-C. Jay Kuo, USA
Ercan E. Kuruoglu, Italy
Tan Lee, China
Geert Leus, The Netherlands
T.-H. Li, USA
Husheng Li, USA
Mark Liao, Taiwan
Y.-P. Lin, Taiwan
Shoji Makino, Japan
Stephen Marshall, UK
C. F. Mecklenbräuker, Austria
Gloria Menegaz, Italy
Ricardo Merched, Brazil
Marc Moonen, Belgium
Vitor Heloiz Nascimento, Brazil
Christophoros Nikou, Greece
Sven Nordholm, Australia
Patrick Oonincx, The Netherlands
Douglas O'Shaughnessy, Canada
Björn Ottersten, Sweden
Jacques Palicot, France
Ana Perez-Neira, Spain
Wilfried Philips, Belgium
Aggelos Pikrakis, Greece
Ioannis Psaromiligkos, Canada
Athanasios Rontogiannis, Greece
Gregor Rozinaj, Slovakia
Markus Rupp, Austria
William Allan Sandham, UK
Bulent Sankur, Turkey
Ling Shao, UK
Dirk Slock, France
Y.-P. Tan, Singapore
João Manuel R. S. Tavares, Portugal
George S. Tombras, Greece
Dimitrios Tzovaras, Greece
Bernhard Wess, Austria
Jar-Ferr Yang, Taiwan
Azzedine Zerguine, Saudi Arabia
Abdelhak M. Zoubir, Germany

Contents

CNN Technology for Spatiotemporal Signal Processing, David López Vilariño, Diego Cabello Ferrer, Víctor Manuel Brea Sánchez, Ronald Tetzlaff, and Chin-Teng Lin
Volume 2009, Article ID 854806, 2 pages

Cellular Neural Networks for NP-Hard Optimization, Mária Ercsey-Ravasz, Tamás Roska, and Zoltán N´da
Volume 2009, Article ID 646975, 7 pages

Extended LaSalle's Invariance Principle for Full-Range Cellular Neural Networks, Mauro Di Marco, Mauro Forti, Massimo Grazzini, and Luca Pancioni
Volume 2009, Article ID 730968, 10 pages

Analysis of Synchronization Phenomena in Broadband Signals with Nonlinear Excitable Media, Anton Chernihovskyi, Christian E. Elger, and Klaus Lehnertz
Volume 2009, Article ID 582412, 7 pages

FPGA-Based Real Time, Multichannel Emulated-Digital Retina Model Implementation, Zsolt Vörösházi, Zoltán Nagy, and Péter Szolgay
Volume 2009, Article ID 749838, 10 pages

Simulation of Two-Dimensional Supersonic Flows on Emulated-Digital CNN-UM, Sándor Kocsárdi, Zoltán Nagy, Árpád Csík, and Péter Szolgay
Volume 2009, Article ID 923404, 11 pages

GPU Boosted CNN Simulator Library for Graphical Flow-Based Programmability, Balázs Gergely Soós, Ádám Rák, József Veres, and György Cserey
Volume 2009, Article ID 930619, 11 pages

APRON: A Cellular Processor Array Simulation and Hardware Design Tool, David R. W. Barr and Piotr Dudek
Volume 2009, Article ID 751687, 9 pages

A CNN-Specific Integrated Processor, Suleyman Malki and Lambert Spaanenburg
Volume 2009, Article ID 854241, 14 pages

An Analog Processor Array Implementing Interconnect-Efficient Reference Data Shift and SAD/SSD Extraction for Motion Estimation, Jonne Poikonen, Mika Laiho, Ari Paasio, Lauri Koskinen, and Kari Halonen
Volume 2009, Article ID 127630, 11 pages

Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 854806, 2 pages
doi:10.1155/2009/854806

Editorial

CNN Technology for Spatiotemporal Signal Processing

David López Vilariño,[1] Diego Cabello Ferrer,[1] Víctor Manuel Brea Sánchez,[1] Ronald Tetzlaff,[2] and Chin-Teng Lin[3]

[1] *University of Santiago de Compostela, Spain*
[2] *Dresden University of Technology, Germany*
[3] *National Chiao-Tung University (NCTU), Hsinchu, Taiwan*

Correspondence should be addressed to David López Vilariño, david.vilarino@usc.es

Received 18 October 2009; Accepted 18 October 2009

Copyright © 2009 David López Vilariño et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Cellular Neural Networks (CNNs) are a paradigm for nonlinear spatial-temporal dynamics and the core of the Cellular Wave Computing (also called CNN technology). Partial Differential Equations (PDE) or wave-like phenomena are the computing primitives of CNN. Besides, their suitability for physical implementation makes CNNs very appropriate for high-speed parallel signal processing.

Since its inception by Professor Chua in 1988, who defined a CNN as an arrangement of regularly spaced cells which usually communicate with each other through their nearest neighbors [1], many advances in theory, applications and implementation have emerged. Early CNN applications were mainly focused on image processing. The availability of cellular processor arrays with a high number of processing elements paved the way for the development of new applications and the recovery of techniques traditionally conditioned by the slow speed of conventional computers. Let us name, as example, image processing techniques based on active wave propagation, or applications within the medical image processing framework, where fast processing provides new capabilities for medical disease diagnosis. Also, emerging applications exploit the complex spatiotemporal phenomena exhibited by multilayer CNN and extend to the modeling of neural circuits for biological vision, motion, and higher brain function.

Besides, today one of the most decisive factors in CNN research is the appearance of many new exciting technological advances. The impressive pace of the Moore's Law has led to the introduction of ever more complex CMOS digital approaches like high-end FPGAs, multicore general purpose microprocessors, or general purpose Graphic Processing Units (GPU). All these platforms offer the performance of ASIC chips from the recent past, allowing for the rapid development of not only proof-of-concept or prototype approaches but also final solutions. At the same time, advances like CMOS-3D technology or new nanodevices fuel the design of custom analog or mixed-signal CNNs. All of these implementations are interwoven with the progress in theory and applications. The nine papers collected on this Special Issue gather advances in theory, applications and, implementation, giving a snapshot of the state-of-the-art of the CNN technology.

The first two papers of this Special Issue by Mária Ercsey-Ravasz et al. and by Mauro Di Marco et al. are placed midway between theory and applications. The first paper touches on a very well-known problem, yet a very hot topic in Artificial Intelligence, NP-hard optimization. The paper demonstrates that CNN computers outperform conventional digital computers. The second paper studies convergence and stability of the dynamics of the full-range model of CNNs, key in signal processing for real time.

The third paper of this Special Issue by Anton Chernihovskyi et al. shows an example of an application efficiently addressed by the spatial-temporal dynamics of CNN technology: the problem of synchronization phenomena in human electroencephalograms.

This Special Issue includes five papers of CNN implementations on digital platforms. The papers by Zsolt Vörösházi et al. and by Sándor Kocsárdi et al. address FPGA realizations of two different applications: a retina model and simulation of two-dimensional supersonic flows. The paper by Balázs Gergely Soós et al. utilizes a GPU

for CNN processing. The paper by David Barr and Piotr Dudek deals with the implementation of a software processor array on today, standard microprocessors. Although it does not discuss an actual digital implementation, the paper by Suleyman Malki and Lambert Spaanenburg tackles different strategies to design a CNN-specific integrated processor mainly focusing on the digital domain. All these papers are examples of a general trend of the today CNN technology: using cheap digital and easy programable platforms for the realization of massively fine-grained parallelism.

Nevertheless, when it comes to very demanding or specific applications, the design of more specific and analog or mixed-signal hardware becomes a reality. Moreover, the advent of new technological advances like the CMOS-3D technology or nanodevices will certainly push further the research in this direction. This might be the case of applications like visual attention, tracking, or motion estimation. The last paper of this issue by Jonne Poikonen et al. gives a glimpse of such implementations with the realization of an analog processor array for motion estimation in conventional CMOS technology.

Last, but not least, the guest editors would like to thank the numerous reviewers for their invaluable work during the preparation of this Special Issue as well as all the authors for their effort in submitting their papers. The guest editors would also like to mention that some of the articles on this Special Issue are extended versions of papers presented at 2008 11th International Workshop on Cellular Neural Networks and their Applications, CNNA2008, held in Santiago de Compostela, Spain.

David López Vilariño
Diego Cabello Ferrer
Víctor Manuel Brea Sánchez
Ronald Tetzlaff
Chin-Teng Lin

References

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

Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 646975, 7 pages
doi:10.1155/2009/646975

Research Article

Cellular Neural Networks for NP-Hard Optimization

Mária Ercsey-Ravasz,[1] Tamás Roska,[2,3] and Zoltán Néda[4]

[1] *Department of Physics, University of Notre Dame, Notre Dame, IN 46556, USA*
[2] *Faculty of Information Technology, Péter Pázmany Catholic University, 1083 Budapest, Hungary*
[3] *Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA-SZTAKI), 1111 Budapest, Hungary*
[4] *Faculty of Physics, Babeş-Bolyai University, 400084 Cluj-Napoca, Romania*

Correspondence should be addressed to Mária Ercsey-Ravasz, mercseyr@nd.edu

Received 24 September 2008; Accepted 26 November 2008

Recommended by David Lopez Vilarino

A cellular neural/nonlinear network (CNN) is used for NP-hard optimization. We prove that a CNN in which the parameters of all cells can be separately controlled is the analog correspondent of a two-dimensional Ising-type (Edwards-Anderson) spin-glass system. Using the properties of CNN, we show that one single operation (template) always yields a local minimum of the spin-glass energy function. This way, a very fast optimization method, similar to simulated annealing, can be built. Estimating the simulation time needed on CNN-based computers, and comparing it with the time needed on normal digital computers using the simulated annealing algorithm, the results are astonishing. CNN computers could be faster than digital computers already at 10×10 lattice sizes. The local control of the template parameters was already partially realized on some of the hardwares, we think this study could further motivate their development in this direction.

Copyright © 2009 Mária Ercsey-Ravasz et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

NP-hard optimization problems represent a key task when testing novel computing paradigms. These complex problems frequently appear in physics, life sciences, biometrics, logistics, database search, and so on, and in most cases they are associated with important practical applications as well [1]. The deficiency of solving these problems in a reasonable amount of time is one of the most important limitation of digital computers, thus all novel paradigms are tested in this sense. Quantum computing, for example, is theoretically well suited for solving NP-hard problems, but the technical realization of quantum computers seems to be quite hard. Here, we argue that cellular neural networks (CNNs) show good perspectives for fast NP-hard optimization. The advantage of cellular wave computing [2], and the CNN paradigm [3], relative to quantum computing is that several practical realizations are already available. After the idea of CNN appeared in 1988 [4], a detailed plan for a CNN computer was developed in 1993 [5]. Since then the chips had a fast developing trend [6, 7] and the latest—already commercialized—version is the Q-Eye chip with lattice size 176×144, included in the Eye-RIS system, mainly used as a visual microprocessor [8].

It has been proved in several previous studies that this novel computational paradigm is useful in solving partial differential equations [9, 10], studying cellular automata models [11, 12], doing image processing [13], and making Monte Carlo simulations on lattice models [14, 15]. Here, it is shown that NP-hard optimization algorithms could be also highly effective using the CNN-UM, if the parameters (templates) of each cell could be separately controlled. The local control of the parameters of each cell is already partially realized on some hardwares and the importance of this study consists also in motivating the further development of hardwares in such direction. In Section 2, we will shortly present the chosen NP-hard problem, namely, the optimization of spin-glass systems. After that, in Section 3, we prove that a CNN computer, on which the parameters of each cell can be separately controlled, is the analog correspondent of a locally coupled two-dimensional spin-glass system. Using these properties of CNN computers, a fast optimization algorithm can be thus

developed, presented in Section 4. In Section 5, we make a rough estimation of the speed of the algorithm, based on the properties of existing hardwares.

2. Spin-Glass Systems

The NP-hard problem we chose to solve using a locally variant CNN is a two-dimensional Ising-type (Edwards-Anderson) spin-glass system [16, 17]. A spin glass is a disordered material exhibiting high magnetic frustration. The typical origin of this frustration is the simultaneous presence of competing interactions and disorder. The Edwards-Anderson model places the interacting Ising-type spins (which can have two possible states $\sigma_{i,j} = \pm 1$) on a square lattice. They interact through an exchange interaction with their neighbors. The strength of the interaction is characterized by the $J_{i,j;k,l}$ coupling constants, separately defined for each connection. The typical frustration characteristic for spin glasses appears, when the coupling constants can take both positive (favoring the spins to align in the same direction) and negative values (favoring the spins to align in opposite directions).

The energy function of the system is the following:

$$H = -\sum_{\langle i,j;k,l\rangle} J_{i,j;k,l}\sigma_{i,j}\sigma_{k,l}, \tag{1}$$

$\langle i,j;k,l\rangle$ representing neighbors, $J_{i,j;k,l}$ is the coupling constant between the spins with coordinates (i,j) and (k,l). If external magnetic field is also added, then the energy is

$$H = -\sum_{\langle i,j;k,l\rangle} J_{i,j;k,l}\sigma_{i,j}\sigma_{k,l} - \sum_{i,j} B_{i,j}\sigma_{i,j}, \tag{2}$$

where $B_{i,j}$ is the strength of the external magnetic field at the lattice site with coordinates (i,j).

The energy landscape of spin-glasses is very complicated, there are many local minima, and finding the ground state is in most of the cases an NP-hard problem. The exceptions are the systems defined on planar graphs, for example, the two-dimensional lattice on which spins are only connected to their 4 nearest neighbors. This can be solved in polynomial time [18]. However, any two-dimensional graph or lattice structure, in which crossing bonds exist, is a nonplanar graph and the problem is already NP-hard (for details see [19]). In this paper, we concentrate on the two dimensional case in which connections with 8 neighbors (nearest and next-nearest neighbors) exist.

Besides its importance in condensed matter physics, spin-glass theory has in the time acquired a strongly interdisciplinary character, with applications to neural network theory [20], computer science [1], theoretical biology [21], econophysics [22], and so on. It has also been shown that using spin-glass models as error-correcting codes, their cost-performance is excellent [23].

3. The Analogy between CNN and Spin-Glass Systems

Here, we present a locally variant CNN, which is the analog correspondent of a spin-glass type system. All the local energy minimums of the two systems coincide, the only difference is that in CNN the state variables of the cells are continuous values between -1 and 1, not discrete values, ± 1, like in the Ising-type spin systems.

We consider a two-dimensional CNN, where the templates (parameters of the state equation in the Chua-Yang model [4]) can be locally controlled. The feedback matrix, A, is considered symmetric $A(i,j;k,l) = A(k,l;i,j)$, $A(i,j;i,j) = 1$ for all (i,j), and the elements are bounded $A(i,j;k,l) \in [-1,1]$((i,j) and (k,l) denote two neighboring cells). Matrix B, which controls the effect of the input array, will be taken simply as $B(i,j;i,j) = b$ and $B(i,j;k,l) = 0$, $\{i,j\} \neq \{k,l\}$. The parameter z is chosen as $z = 0$, so finally our template is defined by $\{A, b\}$ alone. The state-equation of the system writes as

$$\frac{dx_{i,j}(t)}{dt} = -x_{i,j}(t) + \sum_{\langle k,l\rangle \in N(i,j)} A_{i,j;k,l}y_{k,l}(t) + bu_{i,j}, \tag{3}$$

where $x_{i,j}$ is the state value, $y_{i,j}$ is the output, and $u_{i,j}$ is the input of the cell (i,j) with neighborhood $N(i,j)$ (8 neighbors and itself).

In their work Chua and Yang [4] defined the Lyapunov function for the CNN, which behaves like the "energy" (Hamiltonian) of the system. For the standard CNN model, this function was defined as follows:

$$\begin{aligned} E(t) = &-\frac{1}{2}\sum_{(i,j)}\sum_{(k,l)} A_{i,j;k,l}y_{i,j}(t)y_{k,l}(t) + \frac{1}{2}\sum_{(i,j)} y_{i,j}(t)^2 \\ &- \sum_{(i,j)}\sum_{(k,l)} B_{i,j;k,l}y_{i,j}(t)u_{k,l} - \sum_{(i,j)} z_{i,j}y_{i,j}(t). \end{aligned} \tag{4}$$

Inserting the template parameters defined above, this function can be written simply as

$$E(t) = -\sum_{\langle i,j;k,l\rangle} A_{i,j;k,l}y_{i,j}y_{k,l} - b\sum_{i,j} y_{i,j}u_{i,j}. \tag{5}$$

Note that taking all $A_{i,j;i,j} = 1$ the second term of (4) disappears, and in (5) $\langle i,j;k,l\rangle$ denotes pairs of neighbors ($(i,j) \neq (k,l)$), each pair taken only once in the sum. $y_{i,j}$ denotes the output value of each cell and $u_{i,j}$ stands for an arbitrary input image.

By choosing the parameter $b = 0$, the energy of this special CNN is similar with the energy of the spin-glass system shown in (1). The difference is that Ising spins are defined as ± 1, while here we have continuous values between $[-1,1]$. When $b \neq 0$ the Lyapunov function has a similar form to (2), the input image takes the role of the external magnetic field, and parameter b can globally influence the strength of this external field.

The $A(i,j;k,l)$ parameters take the role of the $J_{i,j;k,l}$ coupling constants, and they can be positive and negative as well. In the following, we will be especially interested in the

case when the $A(i, j; k, l)$ couplings lead to a frustration and the quenched disorder in the system is similar with that of spin-glass systems [16, 17].

For showing the analogy between this CNN and spin-glass system, we will use three important properties of the CNN. The first two concerns the Lyapunov function defined by Chua and Yang [4].

(1) It is always a monotone decreasing function in time, $dE/dt \leq 0$, so starting from an initial condition E can only decrease during the dynamics of the CNN.

(2) The final steady state is a local minimum of the energy $dE/dt = 0$.

We shortly prove this theorem for the energy function defined in our system. Making the derivative of the energy function, one gets

$$\begin{aligned}\frac{dE}{dt} &= -\sum_{\langle i,j;k,l\rangle} A_{i,j;k,l}\left(\frac{dy_{i,j}}{dx_{i,j}}\frac{dx_{i,j}}{dt}y_{k,l} + y_{i,j}\frac{dy_{k,l}}{dx_{k,l}}\frac{dx_{k,l}}{dt}\right) \\ &\quad - b\sum_{i,j}\frac{dy_{i,j}}{dx_{i,j}}\frac{dx_{i,j}}{dt}u_{i,j} \\ &= -\sum_{i,j}\sum_{k,l}\frac{dy_{i,j}}{dx_{i,j}}\frac{dx_{i,j}}{dt}A_{i,j;k,l}y_{k,l} - b\sum_{i,j}\frac{dy_{i,j}}{dx_{i,j}}\frac{dx_{i,j}}{dt}u_{i,j} \\ &= -\sum_{i,j}\sum_{k,l}\frac{dy_{i,j}}{dx_{i,j}}\frac{dx_{i,j}}{dt}\left(A_{i,j;k,l}y_{k,l} + b*u_{i,j}\right) \\ &= -\sum_{i,j}\sum_{k,l}\frac{dy_{i,j}}{dx_{i,j}}\left(\frac{dx_{i,j}}{dt}\right)^2 \leq 0.\end{aligned} \tag{6}$$

Note that the symmetry property of the A template was used $A(i, j; k, l) = A(k, l; i, j)$. From the properties of the output function defined as $y_{i,j} = (1/2)(|x_{i,j} + 1| - |x_{i,j} - 1|)$ in the Chua-Yang model [4], we know that

$$\frac{dy_{i,j}}{dx_{i,j}} \geq 1. \tag{7}$$

This way follows that $dE/dt \leq 0$ and in the final steady state when $dx_{i,j}/dt = 0$, the energy also reaches a minimum $dE/dt = 0$.

In addition to these, our CNN has also another important property. Due to the fact that A is symmetric and all self-interaction parameters are $A(i, j; i, j) = 1$, the output values of the cells in a final steady state will be either 1 or -1, so the local minima achieved by the CNN is an Ising-like configuration. This can be understood by analyzing the driving-point (DP) plot of the system. The derivative of the state value is plotted in function of the state value in Figure 1. The state equation of a cell can be divided in two parts, one depending on the state value of the cell, $g(x_{i,j}(t))$, and the other part depending only on the neighbors:

$$\frac{dx_{i,j}(t)}{dt} = g\left(x_{i,j}(t)\right) + w(t), \tag{8}$$

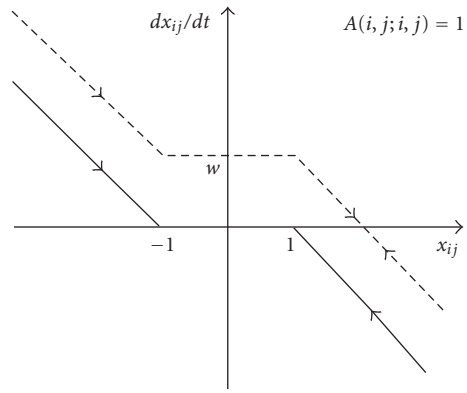

Figure 1: The DP plot of a cell. The derivative of the state value is presented in function of the state value, for $w(t) = 0$ (continuous line) and $w(t) > 0$ (dashed line).

where

$$\begin{aligned} g\left(x_{i,j}(t)\right) &= -x_{i,j}(t) + A(i, j; i, j) * f\left(x_{i,j}(t)\right), \\ w(t) &= \sum_{\langle k,l\rangle \neq \langle i,j\rangle} A_{i,j;k,l}y_{k,l}(t) + bu_{i,j}. \end{aligned} \tag{9}$$

Because we have a coupled template in which $w(t)$ changes in time, also the DP plot of the system changes in function of $w(t)$ (in Figure 1 the case of $w(t) = 0$ is plotted with a continuous line, and $w(t) > 0$ is represented with dashed line). We cannot predict the exact final steady state of the cell, but we can see that the equilibrium points cannot be in the $(-1, 1)$ region, so the output of the cell will be always ± 1. The single exceptions will be the cells (spins) with 0 local energy, in which case $w(t) = 0$. It can happen that these cells do not achieve a ± 1 output (although on real hardwares, in presence of real noise this is hardly probable), but the state of these does not affect the energy of the Ising configuration. So we can just randomly choose between $-1$and 1. The result given by the CNN is similar to finding more than one state at the same time.

We can conclude thus, that starting from any initial condition, the final steady state of the CNN template will be always a local minimum of the spin-glass type Ising spin system with local connections defined by matrix A. The fact that one single operation is needed for finding a local minimum of the energy gives us hope to develop fast optimization algorithms.

4. The Optimization Algorithm

As already emphasized, the complex frustrated case (locally coupled spin-glass-type system), where the A coupling parameters generate a nontrivial quenched disorder will be considered here. The minimum energy configuration of such systems is searched by an algorithm which is similar to the well-known simulated annealing method [24].

The noise is included with random input images ($u_{i,j}$ values in (5)) acting as an external locally variable magnetic field. The strength of this field is governed through parameter

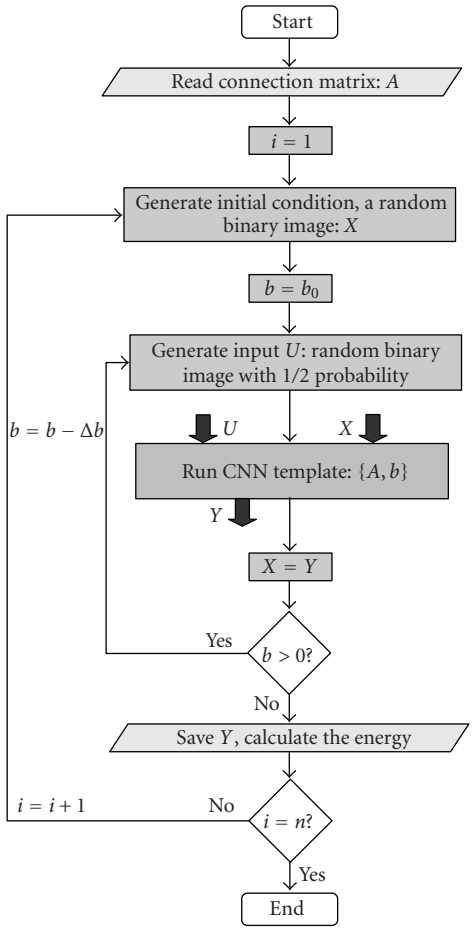

Figure 2: Flowchart of the optimization algorithm using n number of cooling processes.

b. Whenever b is different from zero, our CNN template minimizes the energy with form (5). The first part of it being the energy of the considered spin-glass-type model and the second part an additional term, which gets minimal when the state of the system is equal to the input image (the external magnetic field). If b is large, the result will be the input image itself, if $b = 0$ the result is a local minimum of the pure Ising-type system. For values in between, our result is a "compromise" between the two cases. Slowly decreasing the value of b will result in a process similar with simulated annealing, where the temperature of the system is consecutively lowered. First, big fluctuations of the energy are allowed, and by decreasing this we slowly drive the system to a low-energy state. Since the method is a stochastic one, we can, of course, never be totally sure that the global minimum will be achieved, but good approximations can be achieved.

The algorithm is as follows (see Figure 2).

(1) One starts from a random initial condition x, and $b = 5$ (with this value the result of the template is almost exactly the same as the input image).

(2) A binary random input image u is generated with 1/2 probability of black (1) pixels.

(3) Using the x initial state and the u input image, the CNN template is applied.

(4) The value of b is decreased with steps Δb.

(5) Steps 2–4 are repeated until $b = 0$ is reached. The results of the previous step (minimization) are considered always as the initial state for the next step.

(6) When reaching $b = 0$, the image (Ising spin configuration) is saved and the energy is calculated.

In the classical-simulated annealing algorithm, several thousands of steps for a single temperature are needed. Here, at each noise value one single CNN template is applied. The settling time of the template may vary (it is usually longer in the beginning and gets very fast at the end of the algorithm), but given the fact that after each step noise is introduced, it is acceptable to set a constant running time for our template (usually 5 times the time constant of the CNN dynamics).

Similarly, with choosing the cooling rate in simulated annealing, choosing the value of Δb is also a delicate problem. A proper value providing an acceptable compromise between the quality of the results and speed of the algorithm has to be found. For each system size, one can find an optimal value of Δb, but as one would expect this is rapidly decreasing by increasing the system size. It is much more effective, both for performance (meaning the probability of finding the real global optimum) and speed to choose a constant $\Delta b = 0.05$ step and repeat the whole cooling process several times. As a result, several different final states will be obtained, and we have a higher probability to get the right global minima between these. On the flowchart, the number of cooling processes is denoted with n (Figure 2).

For testing the efficiency of the algorithm, one needs to measure the number of steps necessary for finding the right global minima. To do this, one has to previously know the global minima. In case of small systems with $L = 5, 6$ this can be obtained by a quick exhaustive search in the phase space. For bigger systems, the classical-simulated annealing algorithm was used. The temperature was decreased with a rate of 0.99 ($T_{\text{final}}/T_{\text{ini}}$) and 1000 Monte Carlo steps were performed for each temperature.

In the present work, spin-glass systems with $A(i, j; k, l) = \pm 1$, local connections were studied. The P probability of the positive bonds was varied (influencing the amount of frustration in the system), and local interactions with the 8 Moore neighbors were considered. For several P densities of the positive links and various system sizes, we calculated the average number of steps needed for finding the energy minimum. As naturally is expected for the nontrivial frustrated cases, the number of simulation steps needed exponentially increases with the system size. As an example, the $P = .4$ case is shown in Figure 3(a). As observable in the figure, we could made estimates for relatively small system sizes only ($L < 12$). The reason for this is that we had to simulate also the CNN operations, and for large lattices a huge system of partial differential equations had to be solved. This process gets quite slow for bigger lattices.

The needed average number of steps to reach the estimated energy minima depends also on the P probability

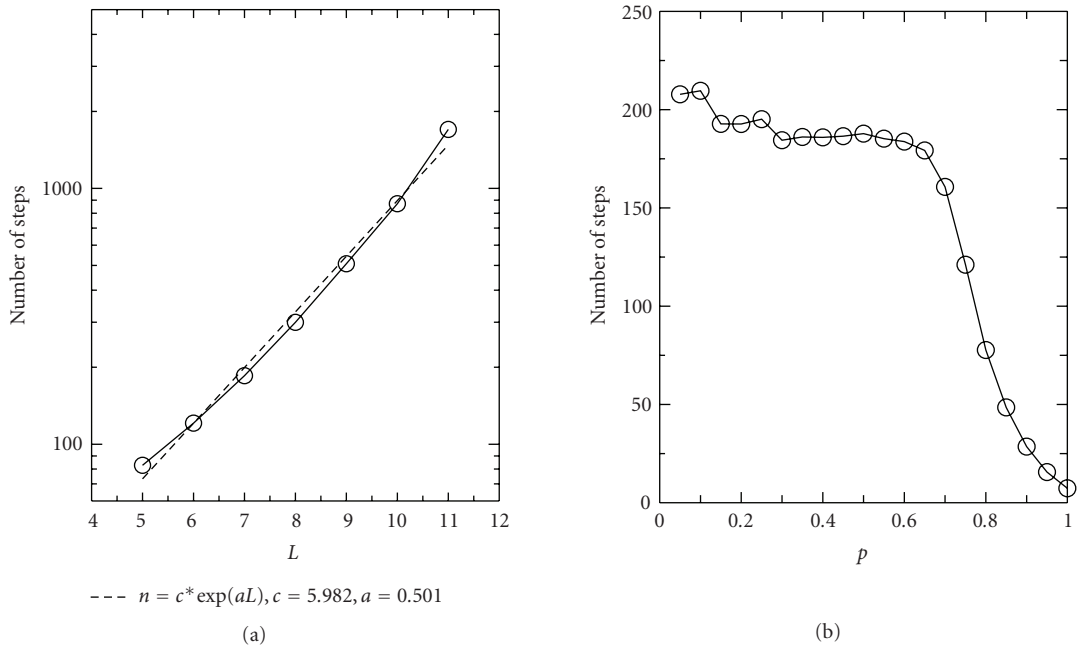

Figure 3: (a) The number of steps needed to find the optimal energy as a function of the lattice size L. The density of positive connections is fixed to $P = .4$, and the parameter $\Delta b = 0.05$ is used. (b) For a system with size $L = 7$, the number of steps needed for getting the presumed global minima is plotted as a function of the probability of positive connections p.

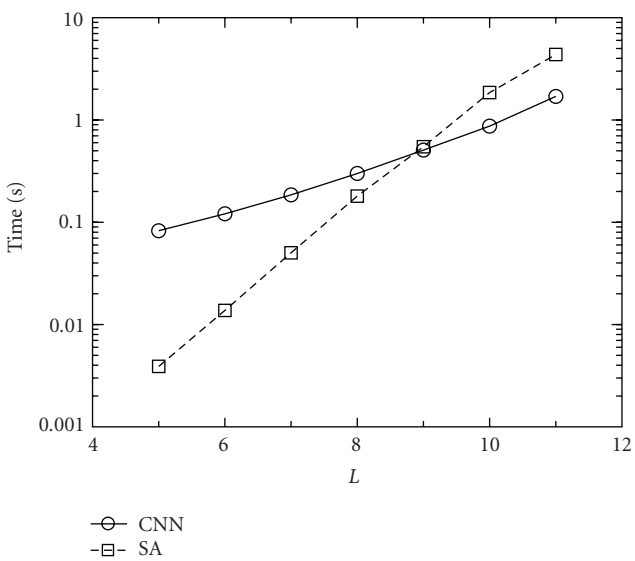

Figure 4: (a) Time needed to reach the minimum energy as a function of the lattice size L. Circles are for estimates on CNN computers and squares are simulated annealing results on 3.4 GHz Pentium 4. Results are averaged on 10 000 different configurations with $P = .4$ probability of positive bonds. For the CNN algorithm, $\Delta b = 0.05$ was chosen. For simulated annealing, the initial temperature was $T_0 = 0.9$, final temperature $T_f = 0.2$, and the decreasing rate of the temperature was fixed as 0.99.

of the positive connections in the system. In Figure 3(b), we illustrate this for a system with size $L = 7$. To obtain this data for each P-value, 5000 different systems were analyzed. As observable in Figure 3(b), the system is almost equally hard to solve for all P-values in the rage of $P \in (0, .6)$.

On real hardwares, one has to think about the fact that noise appears also in the template values, and it may introduce small asymmetries ($A(i, j; k, l) = A(k, l; i, j)$ will be only approximative). We also tested this case in our simulations. For $P = .4$ density of the positive connections and for 7×7 lattice size, 1000 systems were analyzed, introducing a relatively large 10% noise on the template values. Results show no significant difference, the average value of the number of steps needed was 264.0, while in the symmetric case, for the same system size this average was 265.8. Given the fact that our algorithm is similar to simulated annealing, and noise is introduced in the system at each step, these small asymmetries may only increase the noise, but have no significant effect.

5. Speed Estimation

Finally, let us have some thoughts about the estimated speed of such an optimization algorithm. On the CNN chips nowadays available, only parameter z can be locally varied, parameters A and B are 3×3 matrices, uniformly applied for all cells [6]. The reason for no local control of A and B seems to be more due to the lack of motivations. It is technically possible, of course, that the control unit and template memories of the CNN-UM would be more complicated. This modification would not change too much the properties of the hardwares. Introducing the connection parameters (the template) in the local memories of the chip would take a longer time, but in the specific problem considered here the

connection parameters have to be introduced only once for each problem, so this would not effect in a detectable manner the speed of calculations.

Based on our previous experience with the ACE16K chip (with sizes 128×128) [14, 15], we can make a rough estimation of the speed for the presented optimization algorithm. This chip with its parallel architecture solves one template in a very short time—of the order of microseconds. For each step in the algorithm, one also needs to generate a random binary image. This process is already 4 times faster on the ACE16K chip than on a 3.4 GHz Pentium 4 computer and needs around 100 microseconds (see [14]). It is also helpful for the speed that in the present algorithm it is not needed to save information at each step, only once at the end of each cooling process. Saving an image takes roughly 1 millisecond on the ACE16K, but this is done only once after several thousands of simulation steps. Making thus a rough estimate for our algorithm, a chip with similar properties like the ACE16K should be able to compute between 1000–5000 steps in one second, *independently of the lattice size*. Using the lower estimation value (1000 steps /second) and following up the number of steps needed in case of $P = .4$, the estimated average time for solving one problem is plotted as a function of the lattice size in Figure 4 (circles).

Comparing this with the speed of simulated annealing (SA) performed on a 3.4 GHz Pentium 4 (squares in Figure 4), the results for larger lattice sizes are clearly in favor of the CNN chips. For testing the speed of simulated annealing, we used the following parameters: initial temperature $T_0 = 0.9$, final temperature $T_f = 0.2$, decreasing rate of the temperature 0.99. Results were averaged for 10 000 different bond distributions. The necessary number of Monte Carlo steps was always carefully measured by performing many different simulations, using different number of Monte Carlo steps, and comparing the obtained results. From Figure 4, it results that the estimated time needed for the presented algorithm on a CNN chip would be smaller than simulated annealing already at 10×10 lattice sizes.

Spin-glass-like systems have many applications in which global minimum is not crucial to be exactly found, the minimization is needed only with a margin of error. In such cases, the number of requested steps will decrease drastically. As an example in such sense, it has been shown that using spin-glass models as error-correcting codes, their cost performance is excellent [23], and usually the system is not even in the spin-glass phase. In this manner, by using the CNN chip, finding acceptable results could be very fast, even on big lattices.

6. Conclusion

A cellular neural network with locally variable parameters was used for finding the optimal state of locally coupled, two-dimensional, Ising-type spin-glass systems. By simulating the proposed optimization algorithm for a CNN chip, where all connections can be locally controlled, very good perspectives for solving such NP hard problems were predicted. CNN computers could be faster than digital computers already at a 10×10 lattice size. Chips with 2 layers of cells were also produced (CACE1k [25]) and increasing the number of layers is expected in the near future. This way, achieving local control could further extend the number of possible applications. On two layers, it is possible to map already a spin system with any connection matrix (even globally coupled spins) and similar stochastic optimization algorithms could be developed, and also other important NP-hard problems (e.g., K-SAT) may become treatable.

Acknowledgments

This work is supported by a Hungarian ONR Grant no. (N00014-07-1-0350) and a Romanian Consiliul National al Cercetarii Stiintifice din Invatamantul Superior (CNCSIS) no.1571 research Grant (Contract 84/2007).

References

[1] H. Nishimori, *Statistical Physics of Spin Glasses and Information Processing: An Introduction*, Clarendon Press, Oxford, UK, 2001.

[2] T. Roska, "Computational and computer complexity of analogic cellular wave computers," in *Proceedings of the 7th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '02)*, pp. 323–338, Frankfurt, Germany, July 2002.

[3] L. O. Chua and T. Roska, "The CNN paradigm," *IEEE Transactions on Circuits and Systems I*, vol. 40, no. 3, pp. 147–156, 1993.

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

[5] T. Roska and L. O. Chua, "The CNN universal machine: an analogic array computer," *IEEE Transactions on Circuits and Systems II*, vol. 40, no. 3, pp. 163–173, 1993.

[6] A. 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.

[7] Á. Zarándy and C. Rekeczky, "Bi-i: a standalone ultra high speed cellular vision system," *IEEE Circuits and Systems Magazine*, vol. 5, no. 2, pp. 36–45, 2005.

[8] http://www.anafocus.com.

[9] T. Roska, L. O. Chua, D. Wolf, T. Kozek, R. Tetzlaff, and F. Puffer, "Simulating nonlinear waves and partial differential equations via CNN—part I: basic techniques," *IEEE Transactions on Circuits and Systems I*, vol. 42, no. 10, pp. 807–815, 1995.

[10] T. Kozek, L. O. Chua, T. Roska, et al., "Simulating nonlinear waves and partial differential equations via CNN—part II: typical examples," *IEEE Transactions on Circuits and Systems I*, vol. 42, no. 10, pp. 816–820, 1995.

[11] J. M. Cruz and L. O. Chua, "Application of cellular neural networks to model population dynamics," *IEEE Transactions on Circuits and Systems I*, vol. 42, no. 10, pp. 715–720, 1995.

[12] K. R. Crounse, T. Yang, and L. O. Chua, "Pseudo-random sequence generation using the CNN universal machine with applications to cryptography," in *Proceedings of the 4th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '96)*, pp. 433–438, Seville, Spain, June 1996.

[13] K. R. Crounse and L. O. Chua, "Methods for image processing and pattern formation in Cellular Neural Networks: a tutorial," *IEEE Transactions on Circuits and Systems I*, vol. 42, no. 10, pp. 583–601, 1995.

[14] M. Ercsey-Ravasz, T. Roska, and Z. Néda, "Perspectives for Monte Carlo simulations on the CNN universal machine," *International Journal of Modern Physics C*, vol. 17, no. 6, pp. 909–922, 2006.

[15] M. Ercsey-Ravasz, T. Roska, and Z. Néda, "Stochastic simulations on the cellular wave computers," *European Physical Journal B*, vol. 51, no. 3, pp. 407–411, 2006.

[16] S. F. Edwards and P. W. Anderson, "Theory of spin glasses," *Journal of Physics F*, vol. 5, no. 5, pp. 965–974, 1975.

[17] D. Sherrington and S. Kirkpatrick, "Solvable model of a spin-glass," *Physical Review Letters*, vol. 35, no. 26, pp. 1792–1796, 1975.

[18] F. Barahona, "On the computational complexity of Ising spin glass models," *Journal of Physics A*, vol. 15, no. 10, pp. 3241–3253, 1982.

[19] S. Istrail, "Statistical mechanics, three-dimensionality and NP-completeness. I. Universality of intractability for the partition function of the Ising model across non-planar lattices," in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, pp. 87–96, Portland, Ore, USA, May 2000.

[20] M. Mezard, G. Parisi, and M. A. Virasoro, *Spin Glass Theory and Beyond*, World Scientific, Singapore, 1987.

[21] G. Rowe, *Theoretical Models in Biology: The Origin of Life, the Immune System, and the Brain*, Clarendon Press, Oxford, UK, 1997.

[22] R. N. Mantegna and H. E. Stanley, *An Introduction to Econophysics: Correlations and Complexity in Finance*, Cambridge University Press, Cambridge, UK, 2000.

[23] N. Sourlas, "Spin-glass models as error-correcting codes," *Nature*, vol. 339, no. 6227, pp. 693–695, 1989.

[24] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[25] T. Roska and Á. Rodríguez-Vázquez, "Toward visual microprocessors," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1244–1257, 2002.

Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 730968, 10 pages
doi:10.1155/2009/730968

Research Article

Extended LaSalle's Invariance Principle for Full-Range Cellular Neural Networks

Mauro Di Marco, Mauro Forti, Massimo Grazzini, and Luca Pancioni

Department of Information Engineering, University of Siena, 53100 - Siena, Italy

Correspondence should be addressed to Mauro Di Marco, dimarco@dii.unisi.it

Received 15 September 2008; Accepted 20 February 2009

Recommended by Diego Cabello Ferrer

In several relevant applications to the solution of signal processing tasks in real time, a cellular neural network (CNN) is required to be convergent, that is, each solution should tend toward some equilibrium point. The paper develops a Lyapunov method, which is based on a generalized version of LaSalle's invariance principle, for studying convergence and stability of the differential inclusions modeling the dynamics of the full-range (FR) model of CNNs. The applicability of the method is demonstrated by obtaining a rigorous proof of convergence for symmetric FR-CNNs. The proof, which is a direct consequence of the fact that a symmetric FR-CNN admits a strict Lyapunov function, is much more simple than the corresponding proof of convergence for symmetric standard CNNs.

Copyright © 2009 Mauro Di Marco et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

The Full-Range (FR) model of cellular neural networks (CNNs) has been introduced in [1] in order to obtain advantages in the VLSI implementation of CNN chips with a large number of neurons. One main feature is the use of hard-limiter nonlinearities that constrain the evolution of the FR-CNN trajectories within a closed hypercube of the state space. This improved range of the trajectories has enabled us to reduce the power consumption and obtain higher cell densities and increased processing speed [1–4] compared to the original standard (S) CNN model by Chua and Yang [5].

In several applications for solving signal processing tasks in real time it is needed that a FR-CNN is convergent (or completely stable), that is, each solution is required to approach some equilibrium point in the long-run behavior [5–7]. For example, given a two-dimensional image, a CNN is able to perform contour extraction and morphological operations, noise filtering, or motion detection, during the transient motion toward an equilibrium point [8]. Other relevant applications of convergent FR-CNN dynamics concern the solution of optimization or identification problems or the implementation of nonlinear electronic devices for pattern formation [9, 10].

An FR-CNN is characterized by ideal hard-limiter nonlinearities with vertical segments in the i-v characteristic, hence its dynamics is mathematically described by a differential inclusion, where a set-valued vector field models the set of feasible velocities for each state of the FR-CNN. A recent paper [11] has been devoted to the rigorous mathematical foundation of the FR model within the framework of the theory of differential inclusions [12]. The goal of this paper is to extend the results in [11] by developing a generalized Lyapunov approach for addressing stability and convergence of FR-CNNs. The approach is based on a suitable notion of derivative of a (candidate) Lyapunov function and a generalized version of LaSalle's invariance principle for the differential inclusions modeling the FR-CNNs.

The Lyapunov method developed in the paper is formulated in a general fashion, which makes it suitable to check if a continuously differentiable (candidate) Lyapunov function is decreasing along the solutions of a FR-CNNs, and to verify if this property in turn implies convergence of each FR-CNN solution. The applicability of the method is demonstrated by obtaining a rigorous convergence proof for the important and widely used class of symmetric FR-CNNs. It is shown that the proof is more simple than the proof of an analogous convergence result in [11], which

is not based on an invariance principle for FR-CNNs. The same proof is also much more simple than the proof of convergence for symmetric S-CNNs. We refer the reader to [13] for other applications of the method to classes of FR-CNNs with nonsymmetric interconnection matrices used in the real-time solution of some classes of global optimization problems.

The structure of the paper is briefly outlined as follows. Section 2 introduces the FR-CNN model studied in the paper, whereas Section 3 gives some fundamental properties of the solutions of FR-CNNs. The extended LaSalle's invariance principle for FR-CNNs and the convergence results for FR-CNNs are described in Sections 4 and 5, respectively. Section 6 discusses the significance of the convergence results and, finally, Section 7 draws the main conclusions of the paper.

Notation. Let $\mathbb{R}^n$ be the real n-space. Given matrix $A \in \mathbb{R}^{n\times n}$, by A' we mean the transpose of A. In particular, by E_n we denote the $n \times n$ identity matrix. Given the column vectors $x, y \in \mathbb{R}^n$, we denote by $\langle x, y\rangle = \sum_{i=1}^n x_i y_i$ the scalar product of x and y, while $\|x\| = \sqrt{\langle x, x\rangle}$ is the Euclidean norm of x. Sometimes, use is made of the norm $\|x\|_\infty = \max_{i=1,2,\dots,n}|x_i|$. Given a set $D \subset \mathbb{R}^n$, by $\mathrm{cl}(D)$ we denote the closure of D, while $\mathrm{dist}(x, D) = \inf_{y\in D}\|x - y\|$ is the distance of vector $x \in \mathbb{R}^n$ from D. By $B(z, r) = \{y \in \mathbb{R}^n : \|y - z\| < r\}$ we mean an n-dimensional open ball with center $z \in \mathbb{R}^n$ and radius r.

1.1. Preliminaries

1.1.1. Tangent and Normal Cones. This section reports the definitions of tangent and normal comes to a closed convex set and some related properties that are used throughout the paper. The reader is referred to [12, 14, 15] for a more thorough treatment.

Let $Q \subset \mathbb{R}^n$ be a nonempty closed convex set. The tangent cone to Q at $x \in Q$ is given by [14, 15]

$$T_Q(x) = \left\{v \in \mathbb{R}^n : \liminf_{\rho\to 0^+} \frac{\mathrm{dist}(x + \rho v, Q)}{\rho} = 0\right\}, \quad (1)$$

while the normal cone to Q at $x \in Q$ is defined as

$$N_Q(x) = \{p \in \mathbb{R}^n : \langle p, v\rangle \le 0,\ \forall v \in T_Q(x)\}. \quad (2)$$

The orthogonal set to $N_Q(x)$ is given by

$$N_Q^\perp(x) = \{v \in \mathbb{R}^n : \langle p, v\rangle = 0,\ \forall p \in N_Q(x)\}. \quad (3)$$

From a geometrical point of view, the tangent cone is a generalization of the notion of the tangent space to a set, which can be applied when the boundary is not necessarily smooth. In particular, $T_Q(x)$ is the closure of the cone formed by all half lines originating at x and intersecting Q in at least one point y distinct from x. The normal cone is the dual cone of the tangent cone, that is, it is formed by all directions with an angle of at least ninety degrees with any direction belonging to the tangent cone. It is known that $T_Q(x)$ and $N_Q(x)$ are nonempty closed convex cones in $\mathbb{R}^n$, which possibly reduce to the singleton $\{0\}$. Moreover, $N_Q^\perp(x)$ is a vector subspace of $\mathbb{R}^n$, and we have $N_Q^\perp(x) \subset T_Q(x)$. The next property holds [11].

Property 1. If Q coincides with the hypercube $K = [-1, 1]^n$, then $N_K(x)$, $T_K(x)$, and $N_K^\perp(x)$ have the following analytical expressions.

For any $x \in K$ we have

$$N_K(x) = H(x) = (h(x_1), h(x_2), \dots, h(x_n))', \quad (4)$$

where

$$h(\rho) = \begin{cases} (-\infty, 0], & \rho = -1, \\ 0, & \rho \in (-1, 1), \\ [0, +\infty), & \rho = 1, \end{cases} \quad (5)$$

whereas

$$T_K(x) = H_T(x) = (h_T(x_1), h_T(x_2), \dots, h_T(x_n))', \quad (6)$$

with

$$h_T(\rho) = \begin{cases} [0, +\infty), & \rho = -1, \\ (-\infty, +\infty), & \rho \in (-1, 1), \\ (-\infty, 0], & \rho = 1. \end{cases} \quad (7)$$

Finally, for any $x \in K$ we have

$$N_K^\perp(x) = H^\perp(x) = (h^\perp(x_1), h^\perp(x_2), \dots, h^\perp(x_n))', \quad (8)$$

where

$$h^\perp(\rho) = \begin{cases} 0, & \rho = -1, \\ (-\infty, +\infty), & \rho \in (-1, 1), \\ 0, & \rho = 1. \end{cases} \quad (9)$$

The above cones, evaluated at some points of the set $K = [-1, 1]^2$, are reported in Figure 1.

Let $Q \subset \mathbb{R}^n$ be a nonempty closed convex set. The orthogonal projector onto Q is a mathematical operator which associates to any $x \in \mathbb{R}^n$ the set $\mathcal{P}_Q(x)$, composed by the points of Q that are closest to x, namely,

$$\|x - \mathcal{P}_Q(x)\| = \mathrm{dist}(x, Q) = \min_{y\in Q}\|y - x\|. \quad (10)$$

Under the considered assumptions, $\mathcal{P}_Q(x)$ always contains exactly one point. The name derives from the fact that $x - \mathcal{P}_Q(x) \in N_Q(x)$.

2. CNN Models and Motivating Results

The dynamics of the S-CNNs, introduced by Chua and Yang in the fundamental paper [5], can be described by the differential equations:

$$\dot{x}(t) = -x(t) + AG(x(t)) + I, \quad (\mathrm{S})$$

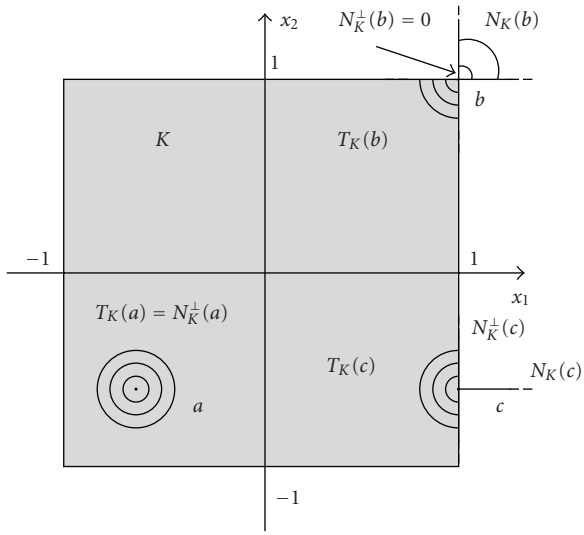

Figure 1: Set $K = [-1,1]^2$ and cones T_K, N_K, and $N_K^\perp$ at points a, b, and c of K (the cones are shown translated into the corresponding points of K). Point a belongs to the interior of K, and hence $T_K(a)$ is the whole space $\mathbb{R}^2$, while $N_Q(a)$ reduces to $\{0\}$. Point b coincides with a vertex of K, and so $T_K(b)$ corresponds to the third quadrant of $\mathbb{R}^2$, while $N_K(b)$ corresponds to the first quadrant of $\mathbb{R}^2$. Finally, point c belongs to the right edge of the square and, consequently, $T_K(c)$ coincides to the left half plane of $\mathbb{R}^2$, while $N_K(c)$ coincides with the nonnegative part of x_1 axis.

where $x \in \mathbb{R}^n$ is the vector of neuron state variables; $A \in \mathbb{R}^{n\times n}$ is the neuron interconnection matrix; $I \in \mathbb{R}^n$ is the constant input; $G(x) = (g(x_1), g(x_2), \ldots, g(x_n))' : \mathbb{R}^n \to \mathbb{R}^n$, where the piecewise-linear neuron activation g is given by

$$g(\rho) = \frac{1}{2}(|\rho+1| - |\rho-1|) = \begin{cases} 1, & \rho > 1, \\ \rho, & -1 \le \rho \le 1, \\ -1, & \rho < -1, \end{cases} \tag{11}$$

see Figure 2(a). It is convenient to define

$$\mathcal{A} = A - E_n, \tag{12}$$

which is the matrix of the affine system satisfied by (S) in the linear region $|x_i| < 1$, $i = 1, 2, \ldots, n$.

The improved signal range (ISR) model of CNNs has been introduced in [1, 2] with the goal to obtain advantages in the electronic implementation of CNN chips. The dynamics of an ISR-CNN can be described by the differential equations:

$$\dot{x}(t) = -x(t) + AG(x(t)) + I - mL(x(t)), \tag{I}$$

where $m \ge 0$, $L(x) = (\ell(x_1), \ell(x_2), \ldots, \ell(x_n))' : \mathbb{R}^n \to \mathbb{R}^n$ and

$$\ell(\rho) = \begin{cases} \rho - 1, & \rho \ge 1, \\ 0, & -1 < \rho < 1, \\ \rho + 1, & \rho \le -1, \end{cases} \tag{13}$$

see Figure 2(b). When the slope m of the nonlinearity $m\ell(\cdot)$ is large, $m\ell(\cdot)$ plays the role of a limiter device that prevents the state variables x_i of (I) from exceedingly enter the saturation regions where $|x_i(t)| > 1$. The larger m, the smaller the neighborhood of the hypercube:

$$K = [-1,1]^n, \tag{14}$$

where the state variables x_i are constrained to evolve for all large t.

A particularly interesting limiting situation is that where $m \to +\infty$, in which case $m\ell(\cdot)$ approaches the ideal hard-limiter nonlinearity $h(\cdot)$ given in (5); see Figure 2(c). The hard-limiter $h(\cdot)$ now constrains the state variables of (F) to evolve within K, that is, we have $|x_i(t)| \le 1$ for all t and for all $i = 1, 2, \ldots, n$. Since for $x \in K$ we have $x = G(x)$, (I) becomes the FR model of CNNs [1, 2, 11]:

$$\dot{x}(t) \in -x(t) + Ax(t) + I - H(x(t)), \tag{F}$$

where $H(x) = (h(x_1), h(x_2), \ldots, h(x_n))'$, and h is given in (5).

From a mathematical viewpoint, $h(\rho)$ is a set-valued map assuming the entire interval of values $[0, +\infty)$ (resp., $(-\infty, 0]$) at $\rho = 1$ (resp., $\rho = -1$). As a consequence, the vector field defining the dynamics of (F), $-x+Ax+I-H(x)$, is a set-valued map assuming multiple values when some state variable x_i is saturated at $x_i = \pm 1$, which represent the set of feasible velocities for (F) at point x. An FR-CNN is thus described by a differential inclusion as in (F) [11, 12] and not by an ordinary differential equation.

In [16], Corinto and Gilli have compared the dynamical behavior of (S) ($m = 0$), with that of (I) ($m \gg 0$) and (F) ($m \to +\infty$), under the assumption that the three models are characterized by the same set of parameters (interconnections and inputs). It is shown in [16] that there are cases where the global behavior of (S) and (I) is *not* qualitatively similar for the same set of parameters, due to bifurcations in model (I) occurring for some positive values of m. In particular, a class of completely stable, second-order S-CNNs (S) has been considered, and it has been shown that, for the same parameters, (I) displays a heteroclinic bifurcation at some $m = m_\beta > 0$, which leads to the birth of a stable limit cycle for any $m > m_\beta$. In other words, (I) is not completely stable for $m > m_\beta$, and the same holds for (F), which is the limit of (I) as $m \to +\infty$.

The result in [16] has the important consequence that in the general case the stability of model (F) cannot be deduced from existing results on stability of (S). Hence, it is needed to develop suitable tools, which are based on the theory of differential inclusions, for studying in a rigorous way the stability and convergence of FR-CNNs.

The goal of this paper is to develop an extended Lyapunov approach for addressing stability and convergence of FR-CNNs. The approach is based on a suitable notion of derivative and an extended version of LaSalle's invariance principle for the differential inclusion (F) modeling a FR-CNN.

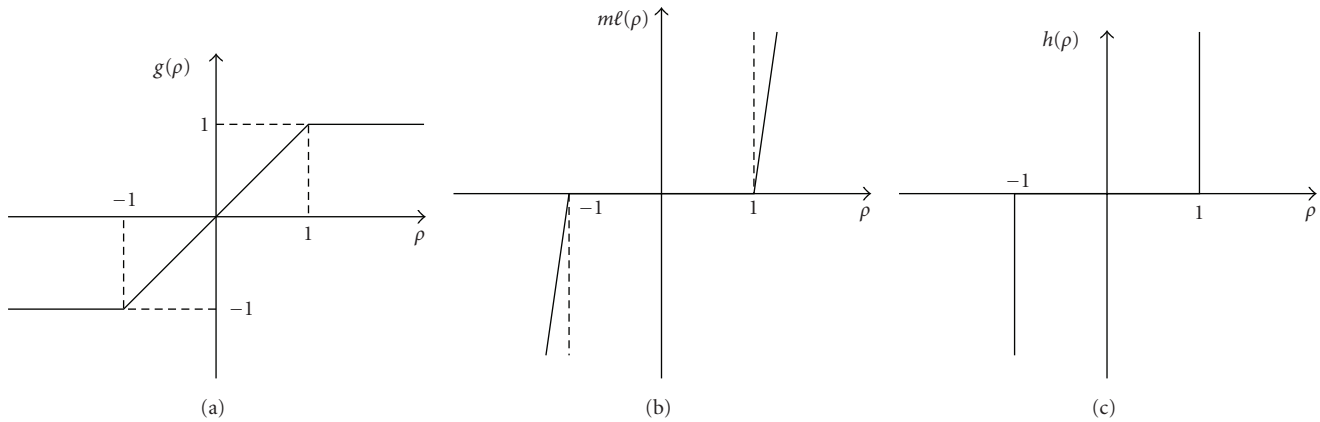

Figure 2: Nonlinearities used in the CNN models (S), (I), and (F).

3. Solutions of FR-CNNs

To the authors knowledge, [11] has been the first paper giving a foundation with the theory of differential inclusions of the FR model of CNNs. One main property noted in [11] is that we have

$$H(x) = N_K(x), \tag{15}$$

for all $x \in K$, that is, $H(x)$ coincides with the normal cone to K at point x (cf. Property 1). Therefore, (F) can be written as

$$\dot{x}(t) \in -x(t) + Ax(t) + I - N_K(x(t)), \tag{16}$$

which represents a class of differential inclusions termed differential variational inequalities (DVIs) [12, Chapter 5].

Let $x_0 \in K$. A solution of (F) in $[0, \tilde{t}]$, with initial condition x_0, is a function x satisfying [12]: (a) $x(t) \in K$ for $t \in [0, \tilde{t}]$ and $x(0) = x_0$; (b) x is absolutely continuous on $[0, \tilde{t}]$, and for almost all (a.a.) $t \in [0, \tilde{t}]$ we have $\dot{x}(t) \in -x(t) + Ax(t) + I - N_K(x(t))$. By an equilibrium point (EP) we mean a constant solution $x(t) = \xi \in K$, $t \geq 0$, of (F). Note that $\xi \in K$ is an EP of (F) if and only if there exists $\gamma_\xi \in N_K(\xi)$ such that $0 = -\xi + A\xi + I - \gamma_\xi$, or equivalently, we have $(A - E_n)\xi + I \in N_K(\xi)$.

By exploiting the theory of DVIs, the next result has been proved in [11].

Property 2. For any $x_0 \in K$, there exists a unique solution x of (F) with initial condition $x(0) = x_0$, which is defined for all $t \geq 0$. Moreover, there exists at least an EP $\xi \in K$ of (F).

We will denote by $E \neq \varnothing$ the set of EPs of (F). It can be shown that E is a compact subset of K.

It is both of theoretic and practical interest to compare the solutions of the ideal model (F) with those of model (I). The next result shows that the solutions of (F) are the uniform limit, as the slope $m \to +\infty$, of the solutions of model (I).

Property 3. Let $x(t)$, $t \geq 0$, be the solution of (F) with initial condition $x(0) = x_0 \in K$. Moreover, for any $m = k = 1, 2, 3, \ldots$, let $x_k(t)$, $t \geq 0$, be the solution of model (I) such that $x_k(t) = x_0$. Then, $x_k(\cdot)$ converges uniformly to $x(\cdot)$ on any compact interval $[0, T] \subset [0, +\infty)$, as $k \to +\infty$.

Proof. See Appendix A. □

4. LaSalle's Invariance Principle for FR-CNNs

Consider the system of ordinary differential equations:

$$\dot{x} = f(x), \tag{17}$$

where $x \in \mathbb{R}^n$, and $f : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable. Let $\phi : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable (candidate) Lyapunov function, and consider the vector field:

$$\delta(x) = \langle f(x), \nabla\phi(x) \rangle, \tag{18}$$

for all $x \in \mathbb{R}^n$. From the standard Lyapunov method for ordinary differential equations [17], it is known that for all times t the derivative of ϕ along a solution x of (17) can be evaluated from δ as follows:

$$\frac{d}{dt}\phi(x(t)) = \delta(x(t)). \tag{19}$$

Such a treatment cannot be directly applied to the differential inclusion (16) modeling the dynamics of a FR-CNN, since the vector field at the right-hand side of (16) assumes multiple values when some component x_i of x assumes the values ± 1. In what follows our goal is to introduce a suitable concept of derivative, which generalizes the definition of δ, for evaluating the time evolution of a candidate Lyapunov function along the solutions of the differential inclusion (16). Then, we prove a version of LaSalle's invariance principle generalizing to the differential inclusions (16) the classic version for ordinary differential equations [17]. In doing so, we need to take into account that the limiting sets for the solutions of (16) enjoy a weaker invariance property with respect to the solutions of the standard differential equations defined by a continuously differentiable vector field.

We begin by introducing the following definition of derivative.

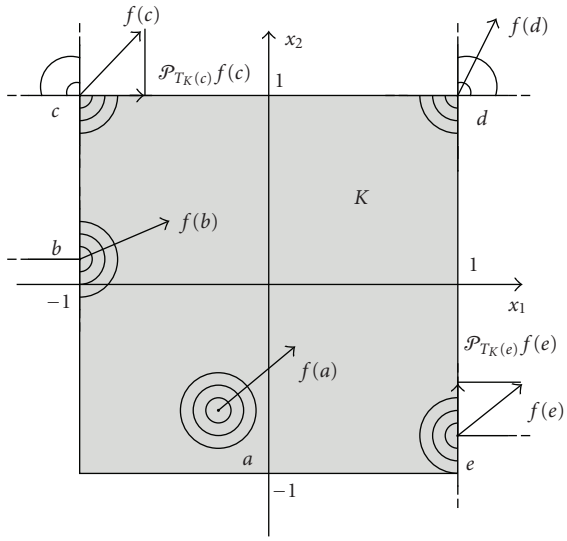

Figure 3: Vector fields involved in the definition of the derivative $D\phi$ for a second-order FR-CNN. Let $f(x) = \mathcal{A}x + I$. We have $\mathcal{P}_{T_K(x)}f(x) \in N_K^{\perp}(x)$, hence $D\phi(x)$ is a singleton, when x is one of the points $a, d, e \in K$. On the other hand, $\mathcal{P}_{T_K(x)}f(x) \notin N_K^{\perp}(x)$ and then $D\phi(x) = \varnothing$, when x is one of the points $b, c \in K$.

Definition 1. Let $\phi : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function in $\mathbb{R}^n$. The derivative $D\phi(x)$ of function ϕ at a point $x \in K$ is given by

$$D\phi(x) = \langle \mathcal{P}_{T_K(x)}(\mathcal{A}x + I), \nabla\phi(x)\rangle, \tag{20}$$

if $\mathcal{P}_{T_K(x)}(\mathcal{A}x + I) \in N_Q^{\perp}(x)$, while

$$D\phi(x) = \varnothing, \tag{21}$$

if $\mathcal{P}_{T_K(x)}(\mathcal{A}x + I) \notin N_K^{\perp}(x)$.

We stress that, for any $x \in K$, $D\phi(x)$ is either the empty set or a singleton. These two different cases are illustrated in Figure 3 for a second-order FR-CNN. Moreover, if $\xi \in E$, then we have $D\phi(\xi) = 0$. Indeed, we have $\mathcal{A}\xi + I \in N_K(\xi)$, and then $\mathcal{P}_{T_Q(\xi)}(\mathcal{A}\xi + I) = 0 \in N_Q^{\perp}(\xi)$. Moreover, $\langle \mathcal{P}_{T_Q(\xi)}(\mathcal{A}\xi + I), \nabla\phi(\xi)\rangle = \langle 0, \nabla\phi(\xi)\rangle = 0$ and so $D\phi(\xi) = 0$.

Definition 2. Let $\phi : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function in $\mathbb{R}^n$. We say that ϕ is a Lyapunov function for (F), if we have $D\phi(x) = \varnothing$ or $D\phi(x) \le 0$, for any $x \in K$. If, in addition, we have $D\phi(x) = 0$ if and only if x is an EP of (F), then ϕ is said to be a strict Lyapunov function for (F).

The next fundamental property can be proved.

Property 4. Let $\phi : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function in $\mathbb{R}^n$, and let $x(t)$, $t \ge 0$, be a solution of (F). Then, for a.a. $t \ge 0$ we have

$$\frac{d}{dt}\phi(x(t)) = D\phi(x(t)). \tag{22}$$

If ϕ is a Lyapunov function for (F), then for a.a. $t \ge 0$ we have

$$\frac{d}{dt}\phi(x(t)) = D\phi(x(t)) \le 0, \tag{23}$$

hence $\phi(x(t))$ is a nonincreasing function for $t \ge 0$, and there exists the $\lim_{t\to+\infty}\phi(x(t)) = \phi_\infty > -\infty$.

Proof. The function $\phi(x(t))$, $t \ge 0$, is absolutely continuous on any compact interval in $[0, +\infty)$, since it is the composition of a continuously differentiable function ϕ and an absolutely continuous function x. Then, for a.a. $t \ge 0$ we have that $x(\cdot)$ and $\phi(x(\cdot))$ are differentiable at t. By [12, page 266, Proposition 2] we have that for a.a. $t \ge 0$

$$\dot{x}(t) \in \mathcal{P}_{T_K(x(t))}(\mathcal{A}x(t) + I). \tag{24}$$

Let $t > 0$ be such that x is differentiable at t. Let us show that $\dot{x}(t) \in N_K^{\perp}(x(t))$. Let $h > 0$, and note that since $x(t)$ and $x(t+h)$ belong to K, we have

$$\operatorname{dist}(x(t) + h\dot{x}(t), K) \le \|x(t) + h\dot{x}(t) - x(t+h)\|. \tag{25}$$

Dividing by h, and accounting for the differentiability of x at time t, we obtain

$$\lim_{h\to 0^+} \frac{\operatorname{dist}(x(t) + h\dot{x}(t), K)}{h} = 0, \tag{26}$$

and hence we have $\dot{x}(t) \in T_K(x(t))$.

Now, suppose that $h \in (-t, 0)$. Since once more $x(t)$ and $x(t+h)$ belong to K, we have

$$\begin{aligned} 0 &\le \frac{\operatorname{dist}(x(t) + (-h)(-\dot{x}(t)), K)}{-h} \\ &\le \frac{\|(x(t) + h\dot{x}(t) - x(t+h)\|}{-h}. \end{aligned} \tag{27}$$

Let $\rho = -h$. Then,

$$\lim_{\rho\to 0^+} \frac{\operatorname{dist}(x(t) + \rho(-\dot{x}(t)), K)}{\rho} = 0, \tag{28}$$

and hence, by definition, $-\dot{x}(t) \in T_K(x(t))$. Now, it suffices to observe that $T_K(x) \cap (-T_K(x)) = N_K^{\perp}(x)$ for any $x \in K$. In fact, if $v \in T_K(x) \cap (-T_K(x))$ and $p \in N_K(x)$, then $\langle v, p\rangle \le 0$ and $\langle -v, p\rangle \le 0$. This means that $\langle v, p\rangle = 0$, that is, $v \in N_K^{\perp}(x)$. Conversely, if $v \in N_K^{\perp}(x)$ and $p \in N_K(x)$, then we have $\langle v, p\rangle = 0$ and $\langle -v, p\rangle = 0$. Hence $v \in T_K(x) \cap (-T_K(x))$.

For a.a. $t \ge 0$ we have

$$\begin{aligned} \frac{d}{dt}\phi(x(t)) &= \langle \dot{x}(t), \nabla\phi(x(t))\rangle \\ &= \langle \mathcal{P}_{T_K(x(t))}(\mathcal{A}x(t) + I), \nabla\phi(x(t))\rangle, \end{aligned} \tag{29}$$

and hence, by Definition 1,

$$\frac{d}{dt}\phi(x(t)) = D\phi(x(t)). \tag{30}$$

Now, suppose that ϕ is a Lyapunov function for (F). Then, for a.a. $t \geq 0$ we have

$$\frac{d}{dt}\phi(x(t)) = D\phi(x(t)) \leq 0, \tag{31}$$

and hence $\phi(x(t))$, $t \geq 0$, is a monotone nonincreasing function. Moreover, ϕbeing a continuous function, it attains a minimum over the compact set K. Since we have $x(t) \in K$ for all $t \geq 0$, the function $\phi(x(t))$, $t \geq 0$, is bounded from below, and there exists the $\lim_{t\to+\infty}\phi(x(t)) = \phi_\infty > -\infty$. □

It is important to stress that, as in the standard Lyapunov approach for differential equations, $D\phi$ permits to evaluate $d\phi(x(t))/dt$ for a.a. $t \geq 0$ directly from the vector field $\mathcal{A}x+I$, without involving integrations of (F) (see Property 4).

We are now in a position to prove the next extended version of LaSalle's invariance principle for FR-CNNs.

Theorem 1. *Let $\phi : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function in $\mathbb{R}^n$, which is a Lyapunov function for* (F). *Let $Z = \{x \in K : D\phi(x) = 0\}$, and let M be the largest positively invariant subset of* (F) *in* $\mathrm{cl}(Z)$. *Then, any solution $x(t)$, $t \geq 0$, of* (F) *converges to M as $t \to +\infty$, that is,* $\lim_{t\to+\infty} dist(x(t), M) = 0$.

Proof. Consider the differential inclusion

$$\dot{x} \in F_r(x) = \mathcal{A}x + I - [N_K(x) \cap \mathrm{cl}(B(0,r))], \tag{32}$$

where $+\infty > r > \sup_K \|\mathcal{A}x + I\|$ and F_r from K into $\mathbb{R}^n$ is an upper-semicontinuous set-valued map with nonempty compact convex values. By [11, Proposition 5] we have that if $x(t)$, $t \geq 0$, is a solution of (F), then x is also a solution of (32) for $t \geq 0$.

Denote by ω_x the ω-limit set of the solution $x(t)$, $t \geq 0$, that is, the set of points $y \in \mathbb{R}^n$ such that there exists a sequence $\{t_k\}$, with $t_k \to +\infty$ as $k \to +\infty$, such that $\lim_{k\to+\infty} x(t_k) = y$. It is known that ω_x is a nonempty compact connected subset of K, and $x(t) \to \omega_x$ as $t \to +\infty$ [18, pages 129, 130]. Furthermore, due to the uniqueness of the solution with respect to the initial conditions (Property 2), ω_x is positively invariant for the solutions of (F) [18, pages 129, 130].

Now, it suffices to show that $\omega_x \subseteq M$. It is known from Property 4 that $\phi(x(t))$, $t \geq 0$, is a nonincreasing function on $[0, +\infty)$ and $\phi(x(t)) \to \phi(\infty) > -\infty$ as $t \to +\infty$. For any $y \in \omega_x$, there exists a sequence $\{t_k\}$, with $t_k \to +\infty$ as $k \to +\infty$, such that $x(t_k) \to y$ as $k \to +\infty$. From the continuity of ϕ, we have $\phi(y) = \lim_{t_k\to+\infty}\phi(x(t_k)) = \phi(\infty)$, hence ϕ is constant on ω_x.

Let $y_0 \in \omega_x$ and let $y(t)$, $t \geq 0$, be the solution of (F) such that $y(0) = y_0$. Since ω_x is positively invariant, we have $y(t) \subseteq \omega_x$ for $t \geq 0$. It follows that $\phi(y(t)) = \phi(\infty)$ for $t \geq 0$ and hence, by Property 4, for a.a. $t \geq 0$ we have $0 = d\phi(y(t))/dt = D\phi(y(t))$. This means that $y(t) \in Z$ for a.a. $t \geq 0$. Hence, $y(t) \in \mathrm{cl}(Z)$ for all $t \geq 0$. In fact, if we had $y(t^*) \notin \mathrm{cl}(Z)$ for some $t^* \geq 0$, then we could find $\delta > 0$ such that $y([t^*, t^* + \delta)) \cap Z = \varnothing$, which is a contradiction. Now, note that in particular we have $y_0 = y(0) \in \mathrm{cl}(Z)$. y_0 being an arbitrary point of ω_x, we conclude that $\omega_x \subset \mathrm{cl}(Z)$. Finally, since ω_x is positively invariant, it follows that $\omega_x \subseteq M$. □

5. Convergence of Symmetric FR-CNNs

In this section, we exploit the extended LaSalle's invariance principle in Theorem 1 in order to prove convergence of FR-CNNs with a symmetric neuron interconnection matrix.

Definition 3. The FR-CNN (F) is said to be quasiconvergent if we have $\lim_{t\to+\infty}\mathrm{dist}(x(t), E) = 0$ for any solution $x(t)$, $t \geq 0$, of (F). Moreover, (F) is said to be convergent if for any solution $x(t)$, $t \geq 0$, of (F) there exists an EP ξ such that $\lim_{t\to+\infty} x(t) = \xi$.

Suppose that $A = A'$ is a symmetric matrix, and consider for (F) the (candidate) quadratic Lyapunov function

$$\phi(x) = -\frac{1}{2}x'\mathcal{A}x - x'I, \tag{33}$$

where $x \in \mathbb{R}^n$.

Property 5. If $A = A'$, then for function ϕ as in (33) we have

$$D\phi(x) = -\left\|\mathcal{P}_{T_K(x)}(\mathcal{A}x + I)\right\|^2 \leq 0, \tag{34}$$

if $P_{T_K(x)}(\mathcal{A}x + I) \in N_K^{\perp}(x)$, while

$$D\phi(x) = \varnothing, \tag{35}$$

if $\mathcal{P}_{T_K(x)}(\mathcal{A}x + I) \notin N_K^{\perp}(x)$. Furthermore, $\mathcal{D}\phi(x) = 0$ if and only if x is an EP of (F), that is, ϕ is a strict Lyapunov function for (F).

Proof. Let $x \in K$ and suppose that $\mathcal{P}_{T_K(x)}(\mathcal{A}x + I) \in N_K^{\perp}(x)$. Observe that $\nabla\phi(x) = -(\mathcal{A}x + I)$. Moreover, since $N_K(x)$ is the negative polar cone of $T_K(x)$ [12, page 220, Proposition 2], we have [12, page 26, Proposition 3]

$$\mathcal{A}x + I = \mathcal{P}_{T_K(x)}(\mathcal{A}x + I) + \mathcal{P}_{N_K(x)}(\mathcal{A}x + I), \tag{36}$$

with $\langle \mathcal{P}_{T_K(x)}(\mathcal{A}x + I), \mathcal{P}_{N_K(x)}(\mathcal{A}x + I)\rangle = 0$.

Accounting for Definition 1, we have

$$\begin{aligned} D\phi(x) &= \langle \mathcal{P}_{T_K(x)}(\mathcal{A}x + I), \nabla\phi(x)\rangle \\ &= \langle \mathcal{P}_{T_K(x)}(\mathcal{A}x + I), -\mathcal{P}_{T_K(x)}(\mathcal{A}x + I)\rangle \\ &\quad + \langle \mathcal{P}_{T_K(x)}(\mathcal{A}x + I), -\mathcal{P}_{N_K(x)}(\mathcal{A}x + I)\rangle \\ &= -\left\|\mathcal{P}_{T_K(x)}(\mathcal{A}x + I)\right\|^2 \leq 0. \end{aligned} \tag{37}$$

Hence, ϕ is a Lyapunov function for (F). It remains to show that it is strict. If x is an EP of (F), then we have $\mathcal{P}_{T_K(x)}(\mathcal{A}x + I) = 0$ and hence $D\phi(x) = 0$. Conversely, if $D\phi(x) = 0$, then we have $\|\mathcal{P}_{T_K(x)}(\mathcal{A}x + I)\| = 0$. Thus, x is an EP for (F). □

Property 5 and Theorem 1 yield the following.

Theorem 2. *Suppose that $A = A'$. Then,* (F) *is quasiconvergent, and it is convergent if the EPs of* (F) *are isolated.*

Proof. Since ϕ is a strict Lyapunov function for (F), we have $Z = E$. Let M be the largest positively invariant set of (F) contained in Z. Due to the uniqueness of the solutions for (F) (Property 2), it follows that $E \subseteq M$. On the other hand,

E is a closed set and hence $E = \mathrm{cl}(E) = \mathrm{cl}(Z) \supseteq M$. In conclusion, $M = E$. Then, Theorem 1 implies that any solution $x(t)$, $t \geq 0$, of (F) converges to E as $t \to +\infty$. Hence (F) is quasiconvergent. Suppose in addition that the equilibrium points of (F) are isolated. Observe that ω_x is a connected subset of $M = E$. This implies that there exists $\xi \in E$ such that $\omega_x = \xi$. Since $x(t) \to \omega_x$, we have $x(t) \to \xi$ as $t \to +\infty$. □

6. Remarks and Discussion

Here, we discuss the significance of the result in Theorem 2 by comparing it with existing results in the literature on convergence of FR-CNNs and S-CNNs. Furthermore, we briefly discuss the possible extensions of the proposed Lyapunov approach to neural network models described by more general classes of differential inclusions.

(1) Theorem 2 coincides with the result on convergence obtained in [11, Theorem 1]. In what follows we point out some advantages with respect to that paper. It is stressed that the proof of Theorem 2 is a direct consequence of the extended version of LaSalle's invariance principle in this paper. The proof of [11, Theorem 1], which is not based on an invariance principle, is comparatively more complex, and in particular it requires an elaborate analysis of the behavior of the solutions of (F) close to the set of equilibrium points of (F). Also the mathematical machinery employed in [11] is more complex than that in the present paper. In fact, in [11] use is made of extended Lyapunov functions assuming the value $+\infty$ outside K and a generalized version of the chain-rule for computing the derivative of the extended-valued functions along the solutions of (F). Here, instead, we have analyzed convergence of (F) by means of a simple quadratic Lyapunov function as in (33).

(2) Consider the S-CNN model (S) and suppose that the neuron interconnection matrix $A = A'$ is symmetric. It has been shown in [5] that (S) admits the Lyapunov function:

$$\psi(x) = -\frac{1}{2}G'(x)(A - E_n)G(x) - G'(x)I, \tag{38}$$

where $x \in \mathbb{R}^n$. One key problem is that ψ *is not a strict Lyapunov function* for the symmetric S-CNN (S), since in partial and total saturation regions of (S) the time derivative of ψ along solutions of (S) may vanish in sets of points that are larger than the sets of equilibrium points of (S). Then, in order to prove quasiconvergence or convergence of (S), it is needed to investigate the geometry of the largest invariant sets of (S) where the time derivative of ψ along solutions of (S) vanishes [7]. Such an analysis is quite elaborate and complex (see [19] for the details). It is worth to remark once more that, according to Theorem 2, ϕ as in (33) *is a strict Lyapunov function* for a symmetric FR-CNN, hence the proof of quasiconvergence or convergence of (F) is a direct consequence of the generalized version of LaSalle's invariance principle in this paper.

(3) The derivative $D\phi$ in Definition 1 and the extended version of LaSalle's invariance principle in Theorem 1 have been inspired by analogous concepts previously developed by Shevitz and Paden [20] and later improved by Bacciotti and Ceragioli [21].

Next, we briefly compare the derivative $D\phi$ with the derivative $\widetilde{D}\phi$ proposed in [21]. If we consider that ϕ is continuously differentiable in $\mathbb{R}^n$, then we have

$$\widetilde{D}\phi(x) = \{\langle v, \nabla\phi(x)\rangle,\ v \in \mathcal{A}x + I - N_K(x)\}, \tag{39}$$

for any $x \in K$. Note that $\widetilde{D}\phi$ is in general set valued, that is, it may assume an entire interval of values. Since $\mathcal{P}_{T_K(x)}(\mathcal{A}x + I) \cap N_K^{\perp}(x) \subseteq \mathcal{P}_{T_K(x)}(\mathcal{A}x + I) \subseteq \mathcal{A}x + I - N_K(x)$, we have

$$D\phi(x) \subseteq \widetilde{D}\phi(x), \tag{40}$$

for any $x \in K$. An analogous inclusion holds when comparing $D\phi$ with the derivative in [20].

Consider now the following second-order symmetric FR-CNN:

$$\dot{x} = -x + Ax + I - N_K(x) = f(x) - N_K(x), \tag{41}$$

where $x = (x_1, x_2)' \in \mathbb{R}^2$,

$$A = \begin{pmatrix} 2 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix}, \qquad I = \begin{pmatrix} 0 \\ \frac{2}{3} \end{pmatrix}, \tag{42}$$

whose solutions evolve in the square $K = [-1, 1]^2$. Also consider the candidate Lyapunov function ϕ given in (33), namely,

$$\phi(x) = -\frac{1}{2}x'\mathcal{A}x - I'x = -\frac{1}{2}x_1(x_1 - x_2) - \frac{2}{3}x_2. \tag{43}$$

Simple computations show that, for any $x = (x_1, x_2)' \in K$ such that $x_2 = 1$, it holds $\mathcal{P}_{T_K(x)}f(x) \in N_K^{\perp}(x)$. As a consequence, if a solution of the FR-CNN (41) passes through a point belonging to the upper edge of K, then the solution will slide along that edge during some time interval.

Now, consider the point $x^* = (0, 1)'$, lying on the upper edge of K. We have $f(x^*) = (-1/2, 2/3)'$, $\nabla\phi(x^*) = -f(x^*) = (1/2, -2/3)'$ and, from Definition 1,

$$\begin{aligned} D\phi(x^*) &= \langle P_{T_K(x^*)}(f(x^*)), \nabla V(x^*)\rangle \\ &= \left\langle \left(-\frac{1}{2}, 0\right), \left(\frac{1}{2}, \frac{2}{2}\right)\right\rangle = -\frac{1}{4} < 0. \end{aligned} \tag{44}$$

On the other hand, we obtain

$$\begin{aligned} \widetilde{D}\phi(x^*) &= \{\langle v, \nabla\phi(x^*)\rangle,\ v \in f(x^*) - N_K(x^*)\} \\ &= \left[-\frac{25}{36}, +\infty\right). \end{aligned} \tag{45}$$

It is seen that $\widetilde{D}\phi(x^*)$ assume both positive and negative values; see Figure 4 for a geometric interpretation.

Therefore, by means of the derivative $D\phi$ we can conclude that ϕ as in (33) is a Lyapunov function for the FR-CNN, while it cannot be concluded that ϕ is a Lyapunov function for the FR-CNN using the derivative $\widetilde{D}\phi$.

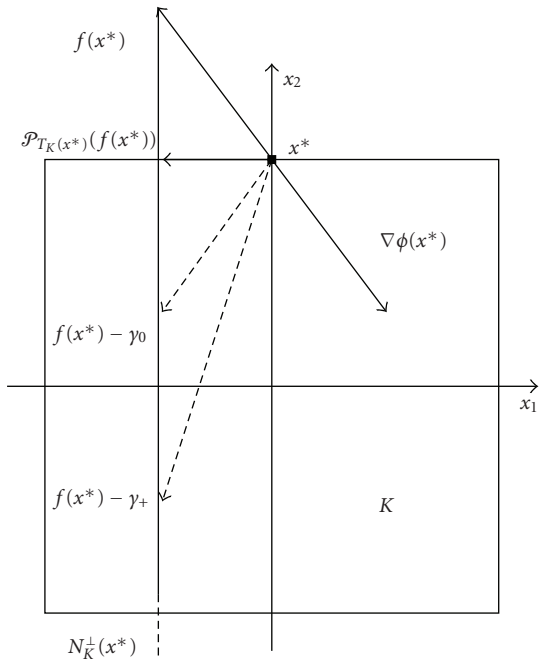

Figure 4: Comparison between the derivative $D\phi$ in Definition 1, and the derivative $\widetilde{D}\phi$ in [21], for the second-order FR-CNN (41). The point $x^* = (0,1)'$ lies on an edge of K such that $T_K(x^*) = \{(x_1,x_2) \in \mathbb{R}^2 : -\infty < x_1 < +\infty,\ x_2 \le 0\}$, $N_K(x^*) = \{(x_1,x_2) \in \mathbb{R}^2 : x_1 = 0,\ x_2 \ge 0\}$ and $N_K^{\perp}(x^*) = \{(x_1,x_2) \in \mathbb{R}^2 : -\infty < x_1 < +\infty,\ x_2 = 0\}$. We have $\mathcal{P}_{T_K(x^*)} f(x^*) \in N_K^{\perp}(x)$ and $D\phi(x^*) = \langle \mathcal{P}_{T_K(x^*)} f(x^*), \nabla\phi(x^*)\rangle = -1/4 < 0$. The derivative $\widetilde{D}\phi(x^*)$ is given by $\widetilde{D}\phi(x^*) = \{\langle v, \nabla\phi(x^*)\rangle,\ v \in f(x^*) - N_K(x^*)\} = [-25/36, +\infty)$, hence it assumes both positive and negative values. For example, the figure shows a vector $\gamma_0 \in N_K(x^*)$ such that we have $\widetilde{D}\phi(x^*) \ni 0 = \langle f(x^*) - \gamma_0, \nabla\phi(x^*)\rangle$, and a vector $\gamma_+ \in N_K(x^*)$ for which we have $\widetilde{D}\phi(x^*) \ni \langle f(x^*) - \gamma_+, \nabla\phi(x^*)\rangle > 0$.

(4) The Lyapunov approach in this paper has been developed in relation to the differential inclusion modeling the FR model of CNNs, that is, a class of DVIs (16) where the dynamics defined by an affine vector field $\mathcal{A}x + I$ are constrained to evolve within the hypercube $K = [-1,1]^n$. The approach can be generalized to a wider class of DVIs, by substituting K with an arbitrary compact convex set $Q \subset \mathbb{R}^n$, or by substituting the affine vector field with a more general (possibly nonsmooth) vector field. In the latter case, it is needed to use nondifferentiable Lyapunov functions and a generalized nonsmooth version of the derivative given in Definition 1. The details on these extensions can be found in the recent paper [13].

7. Conclusion

The paper has developed a generalized Lyapunov approach, which is based on an extended version of LaSalle's invariance principle, for studying stability and convergence of the FR model of CNNs. The approach has been applied to give a rigorous proof of convergence for symmetric FR-CNNs.

The results obtained have shown that, by means of the developed Lyapunov approach, the analysis of convergence of symmetric FR-CNNs is much more simple than that of the symmetric S-CNNs. In fact, one basic result proved here is that a symmetric FR-CNN admits a strict Lyapunov function, and thus it is convergent as a direct consequence of the extended version of LaSalle's invariance principle.

Future work will be devoted to investigate the possibility to apply the proposed methodology for addressing stability of other classes of FR-CNNs that are used in the solution of signal processing tasks in real time. Particular attention will be devoted to certain classes of FR-CNNs with nonsymmetric interconnection matrices. Another interesting issue is the possibility to extend the approach in order to consider the presence of delays in the FR-CNN neuron interconnections.

Appendices

A. Proof of Property 3

Let $M_i = \sum_{j=1}^{n}(|A_{ij}| + |I_i|)$, $i = 1,2,\ldots,n$, and $M = \max\{M_1, M_2, \ldots, M_n\} \ge 0$. We have $\|Ax + I\|_\infty \le M + 1$ for all $x \in K$.

We need to define the following maps. For $k = 1,2,3,\ldots$, let $H_k(x) = (h_k(x_1), h_k(x_2), \ldots, h_k(x_n))'$, $x \in \mathbb{R}^n$, where

$$h_k(\rho) = \begin{cases} -M-1, & \text{if } \dfrac{\rho < -1-(M+1)}{k}, \\ k\ell(\rho), & \text{if } \dfrac{|\rho| \le 1+(M+1)}{k}, \\ M+1, & \text{if } \dfrac{\rho > 1+(M+1)}{k}, \end{cases} \tag{A.1}$$

and $\ell(\cdot)$ is defined in (13). Then, let $H_\infty(x) = (h_\infty(x_1), h_\infty(x_2), \ldots, h_\infty(x_n))'$, $x \in \mathbb{R}^n$, where

$$h_\infty(\rho) = \begin{cases} -(M+1), & \text{if } \rho < -1, \\ [-M-1, 0], & \text{if } \rho = -1, \\ 0, & \text{if } |\rho| < 1, \\ [0, M+1], & \text{if } \rho = 1, \\ M+1, & \text{if } \rho > 1. \end{cases} \tag{A.2}$$

Finally, let $B_M(x) = (b_m(x_1), b_m(x_2), \ldots, b_m(x_n))'$, $x \in \mathbb{R}^n$, where

$$b_m(\rho) = \begin{cases} -(M+1), & \text{if } \rho < -(1+M), \\ \rho, & \text{if } |\rho| \le 1+M, \\ M+1, & \text{if } \rho > 1+M. \end{cases} \tag{A.3}$$

The three maps h_m, h_∞ and b_m are represented in Figure 5.

The proof of Property 3 consists of the three main steps detailed below.

Step 1. Let $x(t)$, $t \ge 0$, be the solution of (F) such that $x(0) = x_0 \in K$. We want to verify that x is also a solution of

$$\dot{x}(t) \in -B_M(x(t)) + AG(x(t)) + I - H_\infty(x(t)), \tag{A.4}$$

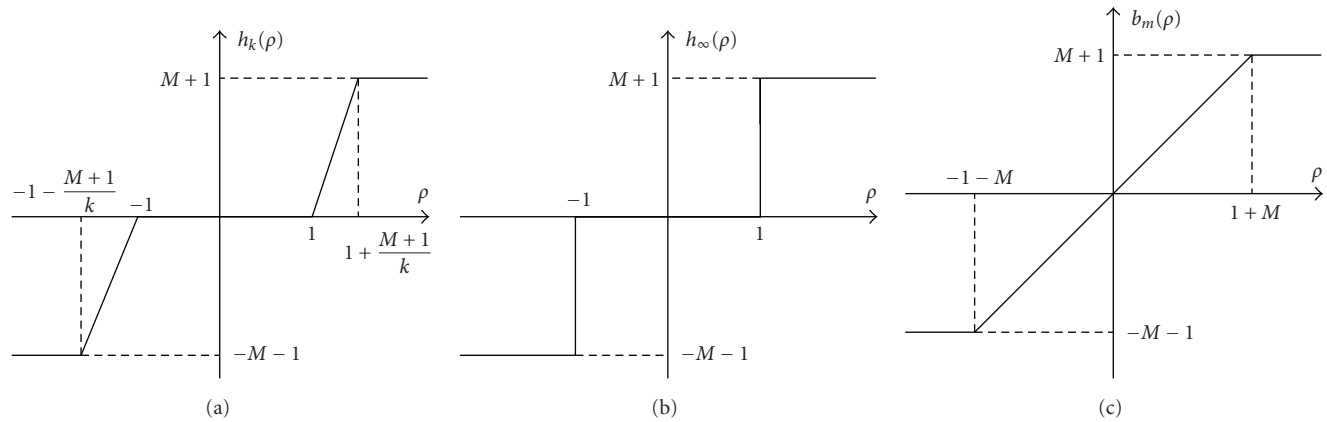

FIGURE 5: Auxiliary maps (a) h_m, (b) h_∞, and (c) b_m employed in the proof of Property 3.

for $t \geq 0$, where $G(x) = (g(x_1), g(x_2), \ldots, g(x_n))'$, $x \in \mathbb{R}^n$, and $g(\cdot)$ is given in (11).

On the basis of [12, page 266, Proposition 2], for a.a. $t \geq 0$ we have

$$\begin{aligned} \dot{x}(t) &= \mathcal{P}_{T_K(x(t))}(\mathcal{A}x(t) + I) \\ &= -x(t) + Ax(t) + I - \mathcal{P}_{N_K(x(t))}(\mathcal{A}x(t) + I), \end{aligned} \tag{A.5}$$

where $\mathcal{P}_{N_K(x(t))}(\mathcal{A}x(t) + I) \in N_K(x(t))$ [12, page 24, Proposition 2; page 26, Proposition 3]. Since for any $t \geq 0$ we have $\|\mathcal{A}x(t) + I\|_\infty \leq M + 1$, by applying the result in Lemma 1 in Appendix B, we obtain $\mathcal{P}_{N_K(x(t))}(\mathcal{A}x(t) + I) \in H_\infty(x(t))$. Furthermore, considering that for any $t \geq 0$ we have $x(t) \in K$, it follows that $B_M(x(t)) = x(t) = G(x(t))$. In conclusion, for a.a. $t \geq 0$ we have

$$\dot{x}(t) \in -B_M(x(t)) + AG(x(t)) + I - H_\infty(x(t)). \tag{A.6}$$

Step 2. For any $k = 1, 2, 3, \ldots$, let $x_k(t)$, $t \geq 0$, be the solution of (I) such that $x_k(0) = x_0 \in K$. We want to show that x_k is also a solution of

$$\dot{x}(t) \in -B_M(x(t)) + AG(x(t)) + I - H_k(x(t)), \tag{A.7}$$

for $t \geq 0$. For any $i \in \{1, 2, \ldots, n\}$ and $t \geq 0$ we have from [2, equation 12]

$$\begin{aligned} |x_i^k(t)| &\leq \frac{M+k}{k+1} + \left(1 - \frac{M+k}{k+1}\right) \exp(-(k+1)t) \\ &= 1 + \frac{M-1}{k+1}(1 - \exp(-(k+1)t)) \\ &\leq 1 + \frac{|M-1|}{k+1} \leq 1 + \min\left\{M, \frac{M+1}{k}\right\}. \end{aligned} \tag{A.8}$$

Then, $B_M(x(t)) = x(t) = G(x(t))$ and $H_k(x(t)) = kL(x(t))$, for $t \geq 0$.

Step 3. Consider the map $\Phi_\infty(x) = -B_M(x) + AG(x) + I - H_\infty(x)$, $x \in \mathbb{R}^n$, and for $k = 1, 2, 3, \ldots$, the maps $\Phi_k(x) = -B_M(x) + AG(x) + I - H_k(x)$, $x \in \mathbb{R}^n$, which are upper semicontinuous in $\mathbb{R}^n$ with nonempty compact convex values.

Let $\text{graph}(H_\infty) = \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^n : y = H_\infty(x)\}$ and $\text{graph}(H_k) = \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^n : y = H_k(x)\}$. Given any $\delta > 0$, for sufficiently large k, say $k > k_\delta$, we have

$$\text{graph}(H_k) \subseteq \text{graph}(H_\infty) + B(0, \delta). \tag{A.9}$$

By applying [12, page 105, Proposition 1] it follows that for any $\tilde{\epsilon} > 0$, $T > 0$, and for any $k > k_\delta$, there exists a solution $\hat{x}_k(t)$, $t \in [0, T]$, of (A.4), such that $\max_{[0,T]} \|\hat{x}_k(t) - x_k(t)\| < \tilde{\epsilon}$.

Choose $\tilde{\epsilon} = \epsilon \exp(-\|A\|_2 T/2)/2$, where $\epsilon > 0$, $\|A\|_2 = (\lambda_M(A'A))^{1/2}$ and $\lambda_M(A'A)$ denotes the maximum eigenvalue of the symmetric matrix $A'A$. Then, we obtain

$$\begin{aligned} \|\hat{x}_k(0) - x(0)\| &= \|\hat{x}_k(0) - x_k(0)\| \\ &\leq \max_{[0,T]} \|\hat{x}_k(t) - x_k(t)\| \\ &< \frac{\epsilon}{2} \exp(-\|A\|_2 T). \end{aligned} \tag{A.10}$$

By Property 6 in Appendix C we have $\max_{[0,T]} \|\hat{x}_k(t) - x(t)\| < \epsilon/2$. Then,

$$\begin{aligned} \max_{[0,T]} \|x_k(t) - x(t)\| &\leq \max_{[0,T]} \|\hat{x}_k(t) - x(t)\| \\ &\quad + \max_{[0,T]} \|\hat{x}_k(t) - x_k(t)\| \\ &< \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon, \end{aligned} \tag{A.11}$$

for all $t \in [0, T]$.

B. Lemma 1 and Its Proof

Lemma 1. *For any $x \in K$, and any $v \in \mathbb{R}^n$ such that $\|v\|_\infty \leq M + 1$, we have $\mathcal{P}_{N_K(x)}(v) \in H_\infty(x)$.*

Proof. For any $i \in \{1, 2, \ldots, n\}$ we have

$$[\mathcal{P}_{N_K(x)}(v)]_i = \begin{cases} v_i, & \text{if } |x_i| = 1,\ x_i v_i > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{B.1}$$

If $\mathcal{P}_{N_K(x)}(v)_i = 0$, we immediately obtain $[\mathcal{P}_{N_K(x)}(v)]_i \in h_\infty(x_i)$. If $x_i = 1$ and $x_i v_i > 0$, we may proceed as follows. We have $h_\infty(x_i) = h_\infty(1) = [0, M+1]$. On the other hand, $0 < v_i \leq M+1$ and so $[\mathcal{P}_{N_K(x)}(v)]_i = v_i \in [0, M+1] = h_\infty(x_i)$. We can proceed in a similar way in the case $x_i = -1$ and $x_i v_i > 0$. □

C. Property 6 and Its Proof

Property 6. Let $\epsilon > 0$. For any $y_0, z_0 \in \mathbb{R}^n$ such that

$$\|z_0 - y_0\| < \epsilon \exp\left(-\frac{\|A\|_2 T}{2}\right), \tag{C.1}$$

we have $\max_{[0,T]} \|z(t) - y(t)\| < \epsilon$, where y and z are the solutions of (A.4) such that $y(0) = y_0$ and $z(0) = z_0$, respectively.

Proof. Let $\varphi(t) = \|z(t) - y(t)\|^2/2$, $t \in [0, T]$. Due to (C.1), for a.a. $t \in [0, T]$ we have

$$\begin{aligned}\dot{\varphi}(t) &= \langle z(t) - y(t), \dot{z}(t) - \dot{y}(t)\rangle \\ &= -\langle z(t) - y(t), B_M(z(t)) - B_M(y(t))\rangle \\ &\quad + \langle z(t) - y(t), A(G(z(t)) - G(y(t)))\rangle \\ &\quad - \langle z(t) - y(t), \gamma_y(t) - \gamma_z(t)\rangle,\end{aligned} \tag{C.2}$$

where $\gamma_y(t) \in H_\infty(y(t))$ and $\gamma_z(t) \in H_\infty(z(t))$. It is seen that B_M is a monotone map in $\mathbb{R}^n$ [12, page 159, Proposition 1], that is, for any $x, y \in \mathbb{R}^n$ and any $\gamma_x \in B_M(x)$, $\gamma_y \in B_M(y)$, we have $\langle x - y, \gamma_x - \gamma_y\rangle \geq 0$. Also H_∞ is a monotone map in $\mathbb{R}^n$. Then, we obtain

$$\begin{aligned}\dot{\varphi}(t) &\leq \langle z(t) - y(t), A(G(z(t)) - G(y(t)))\rangle \\ &\leq \|A\| \, \|z(t) - y(t)\|^2 \\ &= 2\|A\|\varphi(t).\end{aligned} \tag{C.3}$$

Gronwall's lemma yields $\varphi(t) \leq \varphi(0)e^{\|A\|T}$, and so

$$\|z(t) - y(t)\| = \sqrt{2\varphi(t)} \leq \sqrt{2\varphi(0)e^{\|A\|T}} < \epsilon, \tag{C.4}$$

for $t \in [0, T]$. □

Acknowledgment

The authors wish to thank the anonymous Reviewers and Associate Editor for the insightful and constructive comments.

References

[1] A. Rodríguez-Vázquez, S. Espejo, R. Domínguez-Castro, J. L. Huertas, and E. Sánchez-Sinencio, "Current-mode techniques for the implementation of continuous- and discrete-time cellular neural networks," *IEEE Transactions on Circuits and Systems II*, vol. 40, no. 3, pp. 132–146, 1993.

[2] S. Espejo, R. Carmona, R. Domínguez-Castro, and A. Rodríguez-Vázquez, "A VLSI-oriented continuous-time CNN model," *International Journal of Circuit Theory and Applications*, vol. 24, no. 3, pp. 341–356, 1996.

[3] G. L. Cembrano, A. Rodríguez-Vázquez, S. E. Meana, and R. Domínguez-Castro, "ACE16k: a 128 × 128 focal plane analog processor with digital I/O," *International Journal of Neural Systems*, vol. 13, no. 6, pp. 427–434, 2003.

[4] A. 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.

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

[6] L. O. Chua, *CNN: A Paradigm for Complexity*, World Scientific, Singapore, 1998.

[7] M. W. Hirsch, "Convergent activation dynamics in continuous time networks," *Neural Networks*, vol. 2, no. 5, pp. 331–349, 1989.

[8] L. O. Chua and T. Roska, *Cellular Neural Networks and Visual Computing: Foundations and Applications*, Cambridge University Press, Cambridge, UK, 2005.

[9] M. Forti, P. Nistri, and M. Quincampoix, "Convergence of neural networks for programming problems via a nonsmooth Łojasiewicz inequality," *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1471–1486, 2006.

[10] L. O. Chua, Ed., "Special issue on nonlinear waves, patterns and spatio-temporal chaos in dynamic arrays," *IEEE Transactions on Circuits and Systems I*, vol. 42, no. 10, pp. 557–823, 1995.

[11] G. De Sandre, M. Forti, P. Nistri, and A. Premoli, "Dynamical analysis of full-range cellular neural networks by exploiting differential variational inequalities," *IEEE Transactions on Circuits and Systems I*, vol. 54, no. 8, pp. 1736–1749, 2007.

[12] J. P. Aubin and A. Cellina, *Differential Inclusions*, Springer, Berlin, Germany, 1984.

[13] M. Di Marco, M. Forti, M. Grazzini, P. Nistri, and L. Pancioni, "Lyapunov method and convergence of the full-range model of CNNs," *IEEE Transactions on Circuits and Systems I*, vol. 55, no. 11, pp. 3528–3541, 2008.

[14] J. P. Aubin and H. Frankowska, *Set-Valued Analysis*, Birkhäuser, Boston, Mass, USA, 1990.

[15] T. Rockafellar and R. Wets, *Variational Analysis*, Springer, Berlin, Germany, 1997.

[16] F. Corinto and M. Gilli, "Comparison between the dynamic behaviour of Chua-Yang and full-range cellular neural networks," *International Journal of Circuit Theory and Applications*, vol. 31, no. 5, pp. 423–441, 2003.

[17] J. K. Hale, *Ordinary Differential Equations*, Wiley Interscience, New York, NY, USA, 1969.

[18] A. F. Filippov, *Differential Equations with Discontinuous Right-Hand Side*, Mathematics and Its Applications (Soviet Series), Kluwer Academic Publishers, Boston, Mass, USA, 1988.

[19] S.-S. Lin and C.-W. Shih, "Complete stability for standard cellular neural networks," *International Journal of Bifurcation and Chaos*, vol. 9, no. 5, pp. 909–918, 1999.

[20] D. Shevitz and B. Paden, "Lyapunov stability theory of nonsmooth systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 9, pp. 1910–1914, 1994.

[21] A. Bacciotti and F. Ceragioli, "Stability and stabilization of discontinuous systems and nonsmooth Lyapunov functions," *ESAIM: Control, Optimisation and Calculus of Variations*, no. 4, pp. 361–376, 1999.

Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 582412, 7 pages
doi:10.1155/2009/582412

Research Article

Analysis of Synchronization Phenomena in Broadband Signals with Nonlinear Excitable Media

Anton Chernihovskyi,[1,2] Christian E. Elger,[1] and Klaus Lehnertz[1,2,3]

[1] *Department of Epileptology, University of Bonn, Sigmund-Freud Street 25, 53105 Bonn, Germany*
[2] *Helmholtz-Institute for Radiation and Nuclear Physics, University of Bonn, Nussallee 14-16, 53115 Bonn, Germany*
[3] *Interdisciplinary Center for Complex Systems, University of Bonn, Römerstr. 164, 53117 Bonn, Germany*

Correspondence should be addressed to Anton Chernihovskyi, cheant@web.de

Received 23 September 2008; Revised 28 February 2009; Accepted 18 August 2009

Recommended by Ronald Tetzlaff

We apply the method of frequency-selective excitation waves in excitable media to characterize synchronization phenomena in interacting complex dynamical systems by measuring coincidence rates of induced excitations. We relax the frequency-selectivity of excitable media and demonstrate two applications of the method to signals with broadband spectra. Findings obtained from analyzing time series of coupled chaotic oscillators as well as electroencephalographic (EEG) recordings from an epilepsy patient indicate that this method can provide an alternative and complementary way to estimate the degree of phase synchronization in noisy signals.

Copyright © 2009 Anton Chernihovskyi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

Excitable media (EM) are spatially distributed excitable systems that can rapidly propagate excitations (suprathreshold impulses) over long distances without damping. Such traveling waves have been observed in many scientific contexts and—as it is now widely accepted—play an important role in information processing in many biological systems. EM are usually modeled as a collection of locally coupled units with excitable or threshold dynamics [1]. Neighboring units of the medium interact with each other via diffusion-like transport processes which, in many cases, results in a spatially homogeneous steady state. Recent studies extended the notion of excitability of EM to spatially distributed systems, whose unperturbed steady state resembles spatiotemporal chaos. This results in a steady state that actively destroys long-range spatiotemporal correlations and thus prevents the propagation of a single excitation. To generate and then to facilitate a wave-propagation phenomenon, the EM have to be driven with perturbations of appropriate amplitudes and certain (resonant) frequencies. Thus, the phenomenon of frequency-selective excitation waves in EM [2–5] provides a possible way to extend the conventional amplitude-selective notion of excitability in dynamical systems. Applying an unknown signal as a local perturbation to the first unit of EM and measuring the number of excitations in the last unit provide a possibility to detect the presence of rhythmic components in the applied perturbation. Moreover, by increasing the spatial size of the medium it is possible to suppress accidental spikes induced in the first unit and thus to enhance frequency-selectivity of this approach.

The concept of Cellular Neural Networks (CNNs) provides a solid framework to model and to study pattern formation and wave propagation phenomena in many spatially distributed dynamical systems [6–9]. An important subclass of CNN allows one to approximate the behavior of reaction-diffusion partial differential equations (PDEs) and is thus well suited as a framework to model various EM [10]. Probably the most tractable mathematical model of EM is the FitzHugh-Nagumo (FHN) PDE. The discretized version

of this equation provides a generic mathematical model of EM and can in principle be implemented in form of reaction-diffusion CNN.

Despite the many advantages, applications of the method of frequency-selective excitation waves in EM are limited. A permanent propagation of excitations in the media requires—by definition—the presence of rhythmic components in the applied perturbation, which limits the applicability of the method to either monochromatic signals or signals with sharp peaks in the power spectrum well above the noise floor. In many field applications, however, one is often confronted with noisy broadband signals.

The human brain is a complex and nonstationary dynamical system. It is capable to generate a vast variety of spatiotemporal patterns that reflect physiologic as well as pathophysiologic states. Electroencephalography is an important tool in neuroscientific research and especially in clinical practice to measure the aforementioned patterns at a high temporal resolution. Electroencephalography is used for diagnostic purposes and in the presurgical evaluation of epilepsy patients. Epilepsy represents one of the most common neurological disorders, second only to stroke. The cardinal symptom of this disease is recurrent seizures that are usually characterized by an abnormal synchronized firing of neurons involved in the epileptic process. But even between seizures, the epileptic brain is different from normal and accompanying alterations seen on the EEG are often associated with complex characteristic spectral patterns and changes in the degree of synchronization between EEG signals. Since these alterations are usually not rigorously defined and, moreover, are patient specific it is rather difficult to detect them automatically in EEG recordings. There is now growing evidence that an improved understanding of the epileptic process can be achieved through the analysis of interactions in epileptic brain networks [11] and further improvements can be expected from time series analysis technique that allow one to estimate the degree of synchronization in the human EEG especially in cases where a high level of noise contaminations cannot be easily avoided.

In this work we apply the method of frequency-selective excitation waves in EM to characterize synchronization phenomena in time series from two coupled chaotic oscillators as well as in multichannel, multiday EEG recordings. For this purpose we relax the property of frequency-selectivity of EM by considering media that are composed of only a few excitable units. With two such media we estimate the coincidence rate of excitations induced in the last units of the media due to a local perturbation of the first units with broadband signals. Our preliminary findings indicate that this approach provides a reliable estimate of the degree of phase synchronization between signals.

2. Methods and Data

2.1. Frequency-Selective Induction of Excitation Waves in Excitable Media. We consider an EM as a diffusively coupled network of resonators, that is, oscillators exhibiting a transition from their resting states to an oscillatory state via a supercritical Hopf-type bifurcation. The FHN oscillator is a generic example of an excitable oscillator that resides (in its parameter space) either near a supercritical or a subcritical Hopf bifurcation [4, 12, 13]. We here model an EM as a chain of only $L = 3$ diffusively coupled FHN oscillators with zero-flux boundary conditions. The equations of motion of the ith unit read:

$$\begin{aligned}\frac{1}{\epsilon}\frac{dX_i}{dt} &= X_i(a - X_i)(X_i - 1) - Y_i + I \\ &\quad + D(X_{i+1} + X_{i-1} - 2X_i) + \frac{\theta_i(t)\delta_{i1}}{\epsilon} \\ \frac{1}{\epsilon}\frac{dY_i}{dt} &= bX_i - mY_i,\end{aligned} \tag{1}$$

where ϵ is a time scaling coefficient, δ_{ij} is the Kronecker delta, and $\theta_i(t)$ denotes an external perturbation. Following [4] the equations were integrated using a fourth-order Runge-Kutta algorithm ($X_i(0) \in \mathcal{N}(0, 0.1)$) with step size $dt = 0.05$, followed by a downsampling to $dt = 0.5$. The parameters were set in the *Canard* region near a supercritical Hopf bifurcation ($a = 0.1$, $b = 0.015$, $m = 0.015$, $I = 0.062$). With this choice and without spatial diffusion ($D = 0$) each unit exhibits low amplitude (subthreshold) oscillations with a characteristic frequency $w_{\text{int}} \approx 0.2$. The time scaling coefficient ϵ is used to adjust the characteristic frequency of the model with respect to the external perturbation. A supra-threshold perturbation leads to an excited state, which is defined by oscillations (i.e., spikes) that exhibit a significantly higher amplitude (≈ 0.9) than those of an unperturbed unit (<0.3), as already shown in [4]. A nonzero diffusive coupling ($D = 0.5$) leads to an active desynchronization of all units [14]. Such a spatiotemporally disordered state of the EM prevents wave propagation that is induced by a single short-lasting local supra-threshold perturbation ($\theta_i(t) = A\delta(t - t_0)$) . However, wave propagation is facilitated if the medium is perturbed locally with a frequency located near the characteristic frequency of an unperturbed FHN unit (e.g., $\theta_i(t) = A\sin(wt)$). The medium thus represents a sort of narrow band-pass frequency filter exhibiting frequency-selective firing of the last unit X_L. The periodic firing of the last unit can thus be considered as a detection of rhythmic components in the perturbation signal applied to the first unit X_1; see (1). To quantify detection of rhythmic components in the local perturbation of a medium we define the spike rate C in the dynamics of the last unit X_L:

$$\begin{aligned}C &= \frac{M}{M_{\max}}, \\ M &= \sum_{n=1}^{N}\Theta(X_L(n) - S),\end{aligned} \tag{2}$$

where Θ is the Heaviside step function ($\Theta(v) = 0, \forall v \leq 0$ and $\Theta(v) = 1, \forall v > 0$), and S denotes a threshold for the spike amplitude, which we set to $S = 0.9$ in order to match the spike amplitude of an uncoupled FHN unit. M counts the number of spikes induced in the last unit of the medium within a time window that consists of N data

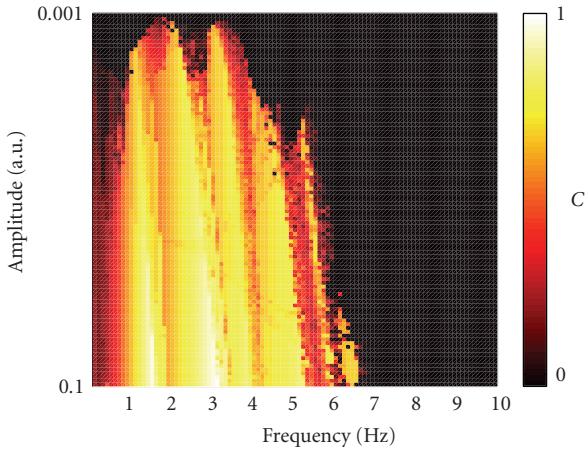

Figure 1: Spike rate C in the last unit of a medium composed of $L = 3$ FHN units. The first unit was perturbed with sine waves of varying amplitudes and frequencies. The time scaling coefficient ϵ was set to 10 and time series consisted of $N = 5000$ data points.

points. The normalization factor $M_{\max}$ defines the maximum possible number of spikes (i.e., during permanent firing of the last unit) within the given time window, that is, $M_{\max} = N w_{\text{int}}/\epsilon$. Thus, $C \to 1$ indicates detection of rhythmic components in the local perturbation.

In Figure 1 we show the spike rate C in the last unit of a medium as a function of amplitude and frequency of a sine wave that is applied as a local perturbation to the first unit. In contrast to the results obtained in the original works on this method [2–5] we observe a rather smooth response over the frequency range $w \in (0.5, 7)$ Hz. By increasing the number of units within the medium and/or decreasing the diffusive coupling strength D between units it is possible to obtain a rather sharp resonant peak and thus increase the frequency-selectivity of the method [4].

Since we aim at approximating the degree of phase synchronization between two signals (consisting of N data points each) we here consider two identical EM, say A and B, each tuned to the same characteristic frequency $w_{\text{int}}^A = w_{\text{int}}^B = w_{\text{int}}$ [15]. According to (1) only the first units ($i = 1$) of the media are perturbed by the signals. Let us assume that the signals contain rhythmic components with frequency $w \approx w_{\text{int}}$. A perturbation of the media will then lead to waves propagating throughout the media. The coincidence between spikes generated by the last units ($i = L$) of media A and B per time step can be used to approximate the degree of phase synchronization between rhythmic components in the applied signals. We define

$$C(A,B) = \frac{2M_c(A,B)}{M_{\max}(A) + M_{\max}(B)}, \qquad M_c(A,B) = \frac{1}{2}\sum_{n=1}^{N} \delta\left(X_L^A(n) + X_L^B(n) - PS\right) \tag{3}$$

where S again denotes a threshold for the spike amplitude, δ is the Dirac delta function, and P is a coincidence threshold with $1 < P \leq 2$. This definition implies that by setting the limiting case $P = 2$ the coincidence rate $C(A, B)$ will provide an estimate of zero-lag phase synchronization only. By decreasing the coincidence threshold P it is possible, however, to cover the regime of lagged phase synchronization. In the following we set $P = 1.3$ to allow a certain temporal mismatch between spikes in the last units of the media. Preliminary investigations using time series from different model systems indicate that with this setting it is indeed possible to detect a lagged phase synchronization in broad band signals. However, setting P too close to 1 increases the risk of detecting accidental coincidences; that is, the situation where accidentally induced spikes will contribute to $C(A, B)$. This results in a certain bias in the regime of weak synchronization as will be shown later in Figure 3. The normalization factors $M_{\max}(A)$ and $M_{\max}(B)$ are defined as above. The coincidence rate $C(A, B)$ is confined to the interval $[0, 1]$ and approaches 1 for identical spike sequences. For large N, $C(A, B) \to 0$ if the sequences of spikes are completely uncorrelated or if the perturbations (even identical perturbations) do not contain rhythmic components matching the characteristic frequencies.

2.2. Measuring Phase Synchronization. In order to demonstrate the applicability of our method to characterize the degree of synchronization between two signals we compare it with a conventional time series analysis technique. Traditionally, phase synchronization is defined as the locking of the phases ϕ of two oscillating systems 1 and 2 [16]:

$$\phi_1(t) - \phi_2(t) = \text{const.} \tag{4}$$

In order to quantify the degree of phase synchronization we used the *mean phase coherence* R [17, 18]

$$R = \left| \frac{1}{N}\sum_{j=0}^{N} e^{i\left[\phi_1(j\Delta t) - \phi_2(j\Delta t)\right]} \right| = 1 - V, \tag{5}$$

where $1/\Delta t$ is the sampling rate of the discrete time series of length N. V denotes the circular variance of an angular distribution obtained by transforming the differences in phase onto the unit circle in the complex plane [19]. By definition R is confined to the interval $[0, 1]$ where $R = 1 (V = 0)$ indicates fully synchronized systems. In order to determine the phases $\phi_{1,2}(t)$ of two signals $s_1(t)$ and $s_2(t)$, we followed the analytic signal approach [20, 21] which renders an unambiguous definition of the instantaneous phase for an arbitrary signal $s(t)$:

$$\phi(t) = \arctan \frac{\tilde{s}(t)}{s(t)}, \tag{6}$$

where

$$\tilde{s}(t) = \frac{1}{\pi}\mathcal{P}\int_{-\infty}^{+\infty} \frac{s(\tau)}{t - \tau} d\tau \tag{7}$$

is the Hilbert transform of the signal ($\mathcal{P}$ denoting the Cauchy principal value). Application of the convolution theorem turns (7) into

$$\tilde{s}(t) = -i\mathcal{F}^{-1}\left[\mathcal{F}\left[s(t)\right] \operatorname{sgn}(\omega)\right], \tag{8}$$

where $\mathcal{F}$ denotes Fourier transform and $\mathcal{F}^{-1}$ inverse Fourier transform, respectively.

An important property of the analytic signal approach is that the instantaneous phase $\phi(t)$ always relates to the predominant frequency in the Fourier spectrum [22] of a broadband signal. A frequency-selective analysis of phase synchronization thus either necessitates prefiltering of signals before applying the Hilbert transform or an alternative definition of phases, for example, by the complex wavelet transform [23]. Both definitions of a phase are mathematically related to each other [24]. Quantifying the degree of phase synchronization via the coincidence rate C is frequency-selective by definition. Note that in contrast to the coincidence rate C the mean phase coherence R always approaches 1 for two identical signals.

2.3. Model Systems. In order to test our approach we analyzed time series generated by two diffusively coupled Rössler systems:

$$\begin{aligned} \frac{dx_{1,2}}{dt} &= -\omega_{1,2}\ y_{1,2} - z_{1,2} + \kappa(x_{2,1} - x_{1,2}), \\ \frac{dy_{1,2}}{dt} &= \omega_{1,2}\ x_{1,2} + 0.165\, y_{1,2}, \\ \frac{dz_{1,2}}{dt} &= 0.2 + z_{1,2}(x_{1,2} - 10) \end{aligned} \tag{9}$$

with a small mismatch of the natural frequencies $\omega_1 = 0.84$ and $\omega_2 = 0.89$. The parameter κ denotes the coupling strength. The differential equations were integrated using a fourth-order Runge-Kutta algorithm with step size $dt = 0.05$ [25]. We generated twenty realizations (using different initial conditions, normally distributed with zero mean and unit variance) of time series consisting of $N = 10^5$ data points. In order to eliminate transients we discarded the first 10^4 points. The data were then normalized to zero mean and standard deviation $\sigma = 0.1$ and eventually contaminated with additive uniform white noise at a signal-to-noise ratio of SNR $= -6$ dB. The amplitude of the noise is thus twice as high as the amplitude of the signal. In order to measure the degree of synchronization when varying the coupling strength κ we analyzed the x-components of the Rössler systems using the coincidence rate C and the mean phase coherence R.

2.4. Field Data. In epilepsy monitoring units, EEG is recorded over an extended period of time for diagnostic purposes or for the presurgical evaluation of candidates for resective therapy. Successful surgical treatment of so-called focal epilepsies requires exact localization of the brain region that can generate seizures and its delineation from eloquent cortex that is indispensable for defined cortical functions (see [26] for a comprehensive overview). This task mostly relies on the observation of typical seizures on the video-EEG, which is currently regarded as the gold standard. Epileptic seizures, however, often represent a rather infrequent phenomenon, and thus the question arises as to what extent information obtained from EEG data recorded during the seizure-free interval can help to identify and to delineate the seizure-generating brain region.

We here retrospectively analyzed long-term, multichannel EEG recordings from an epilepsy patient suffering from temporal lobe epilepsy. For this patient the presurgical workup indicated an epileptic focus in the left mesial temporal lobe and after surgical resection of the seizure generating structure the patient is seizure-free. Intracranial EEG were recorded continuously over a longer period from electrodes implanted bilaterally along the longitudinal axis of the hippocampal formation (cf. Figure 2). The data were sampled at 200 Hz using a 16-bit analog-to-digital (A/D) converter and filtered within a frequency band of 0.5–85 Hz. The recording lasted 104 hours during which ten seizures occurred. We used a moving-window approach (nonoverlapping segments of $N = 2^{14}$ data points) to measure synchronization between all pairs of EEG signals. Since we were interested in recordings from the seizure-free interval only, we excluded data recorded 90 minutes prior to seizures, during actual seizure activities, and from periods of 30 minutes duration immediately following seizures. Prior to estimating the coincidence rate C signals were normalized to zero mean and standard deviation $\sigma = 0.1$.

The intrahippocampal EEG does not have a white spectrum but instead has a predominant spectral content in the so-called δ (0.5–4 Hz) and ϑ (4–7 Hz) range [27]. For a characterization of synchronization phenomena in EEG recordings we therefore concentrated on these frequency bands and adjusted the internal frequency w_{int} of the FHN-units (by setting the time scaling coefficient to $\epsilon = 10$ (cf. (1) and Figure 1)) such as to cover this frequency range.

3. Results

3.1. Model Systems. In Figure 3 we show the dependence of the coincidence rate C and of the mean phase coherence R on the coupling strength κ for the noise-contaminated diffusively coupled Rössler systems. In addition, we show the dependence of R for time series that we low-pass filtered (3rd-order Butterworth characteristic; cutoff-frequency set to $2\omega_2$) prior to analysis. Despite the strong noise contamination (SNR $= -6$ dB) C allowed to detect the transition to the regime of phase synchronization ($\kappa > 0.05$) between the Rössler systems. Due to the noise contamination the resolvability of the different synchronization regimes using the mean phase coherence R was drastically reduced but could be recovered with an appropriate filtering of the data.

Note that even in the regime of no or weak coupling ($\kappa \in (0.0, 0.03)$) we observed $C \approx 0.45$, which resulted from accidental spikes propagating throughout the media. This bias can be reduced by increasing the spatial size of the media, however, at the expense of limiting the applicability of the method to narrow band signals.

3.2. Field Data. In Figure 4 we show, as an example, for all intra- and interhemispheric channel combinations (cf. Figure 2) the averaged coincidence rate $\langle C \rangle$, obtained from averaging C-values from the seizure-free periods only.

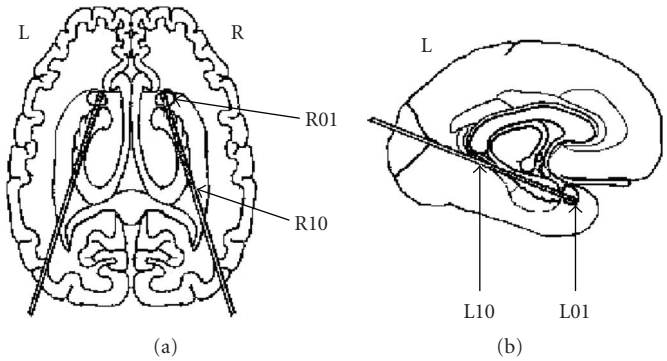

Figure 2: Schematic of intracranially implanted depth electrodes. (a) Axial view; (b) sagittal view. Each catheter-like, 1-mm-thick silastic electrode contains 10 cylindrical contacts of a nickel-chromium-alloy (2.5 mm) every 4 mm.

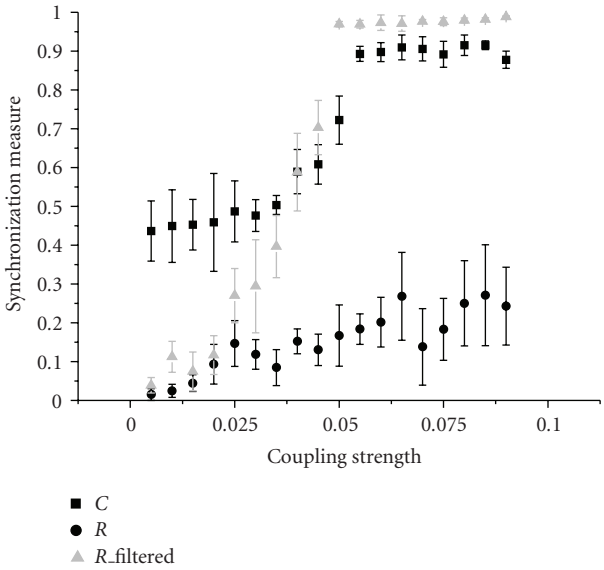

Figure 3: Means and standard deviations of the mean phase coherence R (circles and triangles) and the coincidence rate C (squares) estimated from the noisy (SNR = -6 dB) x-components of coupled Rössler systems for varying coupling strength κ. Triangles indicate values of R from low-pass filtered data. To calculate the coincidence rate C the internal frequency w_{int} of FHN units in both media was chosen to match the natural frequencies of the Rössler oscillators ($w^A_{\text{int}} = w^B_{\text{int}} \approx 1$ by setting $\epsilon = 5$).

We performed the same analysis with the mean phase coherence R and findings are presented in Figure 5. Despite apparent limitations of our method in the regime of weak phase synchronization already observed with model systems, the coincidence rate C captures spatial patterns of synchronization similar to those observed with the mean phase coherence R. With both approaches we observe that the average degree of synchronization from intrahemispheric channel combinations exceeds the one from interhemispheric channel combinations. Interestingly, even a crude temporal averaging (i.e., from data covering more than four days of recording) and spatial averaging (i.e., from data from intrahemispheric channel combinations only) the degree of

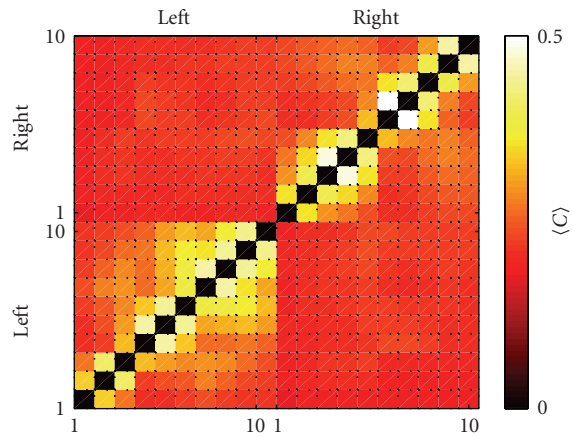

Figure 4: Averaged coincidence rates $\langle C \rangle$ estimated for all channel combinations of EEG data recorded during the seizure-free intervals. Note the difference in the degree of intrahemispheric synchronization between the focal (left) hemisphere and the nonfocal (right) hemisphere.

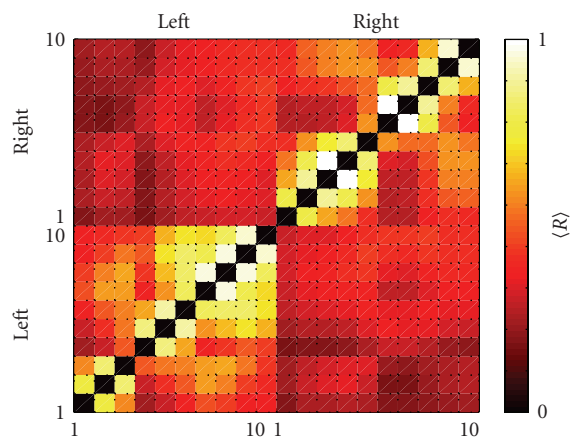

Figure 5: Same as Figure 4 but for the averaged mean phase coherence $\langle R \rangle$. EEG data were low-pass filtered with a 3rd-order Butterworth filter and the cutoff-frequency set to 7 Hz.

synchronization within the left brain hemisphere containing the epileptic focus (ipsilateral hemisphere; $\langle C \rangle_i = 0.25 \pm 0.04$) was slightly higher than in the right (contralateral) brain hemisphere ($\langle C \rangle_c = 0.22 \pm 0.05$). Analyses using the mean phase coherence revealed qualitatively similar results (cf. Figure 5) with $\langle R \rangle_i = 0.53 \pm 0.10$ and $\langle R \rangle_c = 0.49 \pm 0.12$.

4. Conclusion

We presented a biologically inspired approach to estimate the degree of phase synchronization in noisy signals. It is based on the method of frequency-selective excitation waves in excitable media whose coincident outputs are used as an estimate for the degree of phase synchronization in locally applied perturbations. By considering EM that are composed of only a few excitable units and thus by relaxing the frequency-selectivity of EM it was indeed possible to apply the method to signals with broadband spectra. From our preliminary investigations using model systems with well-known properties we could show that the method appears sensitive to intermediate and strong couplings even in cases where other approaches (e.g., the mean phase coherence R) require prefiltering of signals. However, our investigations also indicate that the method appears to be insensitive to detect weak couplings due to the propagation of accidental spikes. Moreover, a careful selection of several important parameters—such as the number L of excitable units constituting the medium, the diffusive coupling D or the coincidence threshold P—is required when analyzing broadband signals from dynamical systems with only poorly understood properties.

Despite these limitations, preliminary analyses of synchronization phenomena in multichannel, multiday electroencephalographic recordings from an epilepsy patient indicate that our method provides results that are qualitatively similar to those obtained with the mean phase coherence R. In line with findings reported in [17, 27–31] we here observed a higher degree of phase synchronization in brain electrical activity recorded from the hemisphere containing the epileptic focus as compared to the nonaffected hemisphere. Although promising, these findings need to be validated on the data from a larger group of patients. A more detailed study including other frequency bands (>7 Hz) on a larger data base is currently underway and will be presented elsewhere.

Extending our previous findings [14, 15] and alternative approaches [32–35] we consider a future implementation of excitable media as analog or digital electrical circuits (e.g., as a FitzHugh-Nagumo cellular neural network) as an attractive perspective for the development of miniaturized detectors of synchronous activities in noisy field data.

Acknowledgments

The authors would like to thank Ronald Tetzlaff and his group for close support. This work was supported by the Deutsche Forschungsgemeinschaft (Grant no. LE 660/2-4).

References

[1] J. Murray, *Mathematical Biology*, Springer, New York, NY, USA, 1989.

[2] G. Baier, R. S. Leder, and P. Parmananda, "Human electroencephalogram induces transient coherence in excitable spatiotemporal chaos," *Physical Review Letters*, vol. 84, no. 19, pp. 4501–4504, 2000.

[3] G. Baier and M. Müller, "Nonlinear dynamic conversion of analog signals into excitation patterns," *Physical Review E*, vol. 70, no. 3, Article ID 037201, 4 pages, 2004.

[4] G. Baier and M. Müller, "Frequency-selective induction of excitation waves near sub- and supercritical hopf bifurcation," *Physics Letters A*, vol. 330, no. 5, pp. 350–357, 2004.

[5] A. Chernihovskyi, F. Mormann, M. Müller, C. E. Elger, G. Baier, and K. Lehnertz, "EEG analysis with nonlinear excitable media," *Journal of Clinical Neurophysiology*, vol. 22, no. 5, pp. 314–329, 2005.

[6] L. O. Chua and L. Yang, "Cellular neural networks: theory," *IEEE Transactions on Circuits and Systems*, vol. 35, pp. 1257–1272, 1988.

[7] L. O. Chua and L. Yang, "Cellular neural networks: applications," *IEEE Transactions on Circuits and Systems*, vol. 35, pp. 1273–1290, 1988.

[8] L. O. Chua, *CNN: A Paradigm for Complexity*, World Scientific, Singapore, 1998.

[9] L. O. Chua and T. Roska, *Cellular Neural Networks and Visual Computing*, Cambridge University Press, Cambridge, UK, 2002.

[10] L. O. Chua, M. Hasler, G. S. Moschytz, and J. Neirynck, "Autonomous cellular neural networks: a unified paradigm for pattern formation and active wave propagation," *IEEE Transactions on Circuits and Systems*, vol. 42, no. 10, pp. 559–577, 1995.

[11] K. Lehnertz, S. Bialonski, M.-T. Horstmann, et al., "Synchronization phenomena in human epileptic brain networks," *Journal of Neuroscience Methods*, vol. 183, no. 1, pp. 42–48, 2009.

[12] E. M. Izhikevich, "Neural excitability, spiking and bursting," *International Journal of Bifurcation and Chaos in Applied Science and Engineering*, vol. 10, no. 6, pp. 1171–1266, 2000.

[13] V. M. Eguiluz, M. Ospeck, Y. Choe, A. J. Hudspeth, and M. O. Magnasco, "Essential nonlinearities in hearing," *Physical Review Letters*, vol. 84, no. 22, pp. 5232–5235, 2000.

[14] A. Chernihovskyi, C. E. Elger, and K. Lehnertz, "Effect of inhibitory diffusive coupling on frequency-selectivity of excitable media simulated with cellular neural networks," in *Proceedings of the 10th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '06)*, V. Tavsanoglu and S. Arik, Eds., pp. 292–296, Piscataway, NJ, USA, August 2006.

[15] A. Chernihovskyi and K. Lehnertz, "Measuring synchronization with nonlinear excitable media," *International Journal of Bifurcation and Chaos in Applied Sciences and Engineering*, vol. 17, no. 10, pp. 3425–3429, 2007.

[16] C. H. Huygens, *Horologium Oscillatorium*, Apud F. Muguet, Paris, France, 1673.

[17] F. Mormann, K. Lehnertz, P. David, and C. E. Elger, "Mean phase coherence as a measure for phase synchronization and its application to the EEG of epilepsy patients," *Physica D*, vol. 144, no. 3, pp. 358–369, 2000.

[18] A. Pikovsky, M. Rosenblum, and J. Kurths, *Synchronization: A Universal Concept in Nonlinear Sciences*, Cambridge University Press, Cambridge, UK, 2001.

[19] K. V. Mardia, *Statistics of Directional Data*, Academic Press, London, UK, 1972.
[20] D. Gabor, "Theory of communication," *Journal of the Institute of Electrical Engineering (London)*, vol. 93, pp. 429–457, 1946.
[21] P. Panter, *Modulation, Noise, and Spectral Analysis*, McGraw-Hill, New York, NY, USA, 1965.
[22] B. Boashash, *Time Frequency Signal Analysis: Methods and Applications*, Longman Cheshire, Melbourne, Australia, 1992.
[23] J.-P. Lachaux, E. Rodriguez, J. Martinerie, and F. J. Varela, "Measuring phase synchrony in brain signals," *Human Brain Mapping*, vol. 8, no. 4, pp. 194–208, 1999.
[24] R. Quian Quiroga, A. Kraskov, T. Kreuz, and P. Grassberger, "Performance of different synchronization measures in real data: a case study on electroencephalographic signals," *Physical Review E*, vol. 65, no. 4, Article ID 041903, 14 pages, 2002.
[25] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK, 1992.
[26] J. Engel Jr. and T. A. Pedley, *Epilepsy: A Comprehensive Textbook*, Lippincott, Williams & Wilkins, Philadelphia, Pa, USA, 2nd edition, 2007.
[27] H. Osterhage, F. Mormann, M. Staniek, and K. Lehnertz, "Measuring synchronization in the epileptic brain: a comparison of different approaches," *International Journal of Bifurcation and Chaos in Applied Sciences and Engineering* , vol. 17, no. 10, pp. 3539–3544, 2007.
[28] S. Bialonski and K. Lehnertz, "Identifying phase synchronization clusters in spatially extended dynamical systems," *Physical Review E*, vol. 74, no. 5, Article ID 051909, 2006.
[29] E. Ben-Jacob, S. Boccaletti, A. Pomyalov, I. Procaccia, and V. L. Towle, "Detecting and localizing the foci in human epileptic seizures," *Chaos*, vol. 17, no. 4, Article ID 043113, 2007.
[30] C. A. Schevon, J. Cappell, R. Emerson, et al., "Cortical abnormalities in epilepsy revealed by local EEG synchrony," *NeuroImage*, vol. 35, no. 1, pp. 140–148, 2007.
[31] G. J. Ortega, L. Menendez de La Prida, R. G. Sola, and J. Pastor, "Synchronization clusters of interictal activity in the lateral temporal cortex of epileptic patients: intraoperative electrocorticographic analysis," *Epilepsia*, vol. 49, no. 2, pp. 269–280, 2008.
[32] R. Sowa, A. Chernihovskyi, F. Mormann, and K. Lehnertz, "Estimating phase synchronization in dynamical systems using cellular nonlinear networks," *Physical Review E*, vol. 71, no. 6, Article ID 061926, 6 pages, 2005.
[33] D. Krug, H. Osterhage, C. E. Elger, and K. Lehnertz, "Estimating nonlinear interdependences in dynamical systems using cellular nonlinear networks," *Physical Review E*, vol. 76, no. 4, Article ID 041916, 2007.
[34] D. Krug, C. E. Elger, and K. Lehnertz, "A CNN-based synchronization analysis for epileptic seizure prediction: inter- and intraindividual generalization properties," in *Proceedings of the 11th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '08)*, D. L. Vilarino, D. C. Ferrer, and V. M. B. Sanchez, Eds., pp. 92–95, Piscataway, NJ, USA, July 2008.
[35] A. Chernihovskyi, D. Krug, C. E. Elger, and K. Lehnertz, "Time series analysis with cellular neural networks," in *Seizure Prediction in Epilepsy: From Basic Mechanisms to Clinical Applications*, B. Schelter, J. Timmer, and A. Schulze-Bonhage, Eds., pp. 131–148, Wiley-VCH, Berlin, Germany, 2008.

Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 749838, 10 pages
doi:10.1155/2009/749838

Research Article

FPGA-Based Real Time, Multichannel Emulated-Digital Retina Model Implementation

Zsolt Vörösházi,[1] Zoltán Nagy,[2] and Péter Szolgay[2, 3]

[1] *Department of Image Processing and Neurocomputing, University of Pannonia, 8200 Veszprém, Hungary*
[2] *Cellular Sensory and Wave Computing Laboratory, Computer and Automation Institute of HAS, 1111 Budapest, Hungary*
[3] *Faculty of Information Technology, Pázmány Péter Catholic University, 1083 Budapest, Hungary*

Correspondence should be addressed to Zsolt Vörösházi, voroshazi@vision.vein.hu

Received 5 September 2008; Accepted 30 January 2009

Recommended by Diego Cabello Ferrer

The function of the low-level image processing that takes place in the biological retina is to compress only the relevant visual information to a manageable size. The behavior of the layers and different channels of the neuromorphic retina has been successfully modeled by cellular neural/nonlinear networks (CNNs). In this paper, we present an extended, application-specific emulated-digital CNN-universal machine (UM) architecture to compute the complex dynamic of this mammalian retina in video real time. The proposed emulated-digital implementation of multichannel retina model is compared to the previously developed models from three key aspects, which are processing speed, number of physical cells, and accuracy. Our primary aim was to build up a simple, real-time test environment with camera input and display output in order to mimic the behavior of retina model implementation on emulated digital CNN by using low-cost, moderate-sized field-programmable gate array (FPGA) architectures.

Copyright © 2009 Zsolt Vörösházi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

The most important sensory organ for both humans and mammals is the retina, which is a sophisticated visual preprocessor system. This well-known part of the eye sends visual information to the higher brain center (visual cortex) across several parallel stacked channels (visual pathway). These parallel operating ganglion cell populations are feature detectors (such as orientation or direction of movement, of form, color, etc.) that are built from a series of complex spatiotemporal transformations generated at different retinal levels.

The retina contains a dense mosaic of light-sensitive cells: the rod and the cone photoreceptors. The photoreceptors are in the deepest layers of the retina, and the light must first pass through rows of other transparent cell layers (ganglion, amacrine, bipolar, and horizontal cells) before reaching the receptors. Functionally, there are these five main cell types in the retina, and in the retinal transmission of information the receptors are the first units and the ganglion cells are the last [1].

Several different retina models have been developed till now: some are low-complexity black-box models [2], while others are high-complexity detailed neuromorphic models [3–5], from which one is described and implemented on FPGA here. The cellular neural/nonlinear network- (CNN-) based neuromorphic model of the vertebrate retina has been continuously improved and refined through the years. First, a number of highly complex space-time transformations were unraveled [4] and only the outer retina functions were explored. The main problems came from the lack of adequate analyzing tools. The detailed framework of mammalian retinal modeling via two-dimensional and multilayer CNN was published in [5], which provides us suitable model background to the real-time implementation. Definition of the neuromorphic model elements was based on retinal anatomy and electrophysiology measurements. The complexity of this channel-based model is moderate enough, therefore, it can be adapted to different hardware implementations using the CNN computational paradigm.

These CNN-based retina models can be realized in different forms, such as software simulator, analog VLSI

chips, and emulated-digital hardware (ASIC or FPGA) [6, 7]. However, the determination of the model parameters requires very high computing power and accurate solution. On one hand, using the complex cell analog VLSI chips, only the basic building blocks of the retina model can be implemented [6]. On the other hand, the software simulators provide high flexibility, but low computing speed [8, 9], hence simulating these detailed neuromorphic structures require powerful workstations, but simulations still require several hours or days to complete.

By using an emulated-digital CNN universal machine (CNN-UM) architecture on FPGA [10] we can overcome the limitations of the analog VLSI CNN chips such as a few numbers of layers (1-2), relatively small array size (128×128), difficulties with implementing nonlinear templates, and low accuracy (limited to 6-7 bit).

The emulated digital architectures based on CNN computational paradigm can be used very efficiently for two-dimensional, spatiotemporal signal processing, solving partial differential equations (PDEs) and state equations of complex dynamical systems [11] as well.

A neuromorphic two dimensional but multilayer retina model was successfully developed and tested comparing the computed and measured values. By using reconfigurable FPGAs, we could handle the inherently multilayer structure of a retina model, and the parameters and values can be represented with scalable accuracy [10, 11]. The previously elaborated single-channel model [7] was extended to a multichannel one with higher hardware requirements and optimizations. The complete real-time system (about 30 frame/sec) based on FPGA, video camera, and monitor was built up to verify the model's behavior and to analyze the effect of parameter accuracy.

Considering the structure of this paper, in Section 2, we show the computational background of the mammalian retina model. Details about the optimized Falcon processor and the distributed arithmetic unit of the multilayer retina model architecture are introduced in Section 3. In Section 4 real-time image processing test systems are presented while in Section 5 the device utilization and the speedup on different FPGA architectures are analyzed. In Section 6 our results are compared to the original neurobiological measurements and the implemented model is demonstrated by an example. Finally, some conclusions are drawn in Section 7.

2. Background: The Mammalian Retina Model

2.1. Governing Equations. The basic building blocks of the biological retina model are the abstract neurons which are organized into two-dimensional layers [2]. The model seems to be complex in one hand, but can be decomposed into multi-layer CNN structure and can be implemented on a parallel processing array. The main components of the abstract neuron are the cell body, the synapses, and the output transfer function. The cell body has a first- or second-order dynamics, which is described by the following differential equations deduced from [5]:

$$\tau_n^l \dot{x}_n^l = -x_n^l + \sum_{kl\in \mathrm{Sn}} C_{n,kl}^{\lambda} x_{n,kl}^l + \sum_{\forall m\in Sy1} \sum_{kl\in Sm} f_{nm}^r (G_{nm,kl}^{\sigma} y_m^l) + \sum_{\forall m\in \mathrm{Sy2}} x_{nm}^r + \sum_{\forall m\in \mathrm{Sy3}} (x_{nm}^r - r x_{nm}^d) - s x_n^j, \tag{1}$$

$$\tau_n^j \dot{x}_n^j = -x_n^j + x_n^l, \tag{2}$$

$$y_n^l = f_n^o (x_n^l), \tag{3}$$

$$C_n^{\lambda} = \lambda \begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \tag{4}$$

$$G_{nm,ij}^{\sigma} = \frac{1}{\sum_{\forall k,l} G_{nm}^{\sigma}(k,l)} g e^{-\sqrt{i^2+j^2}/\sigma}, \tag{5}$$

$$\begin{bmatrix} G_{-1,-1}^{\sigma} & G_{-1,0}^{\sigma} & G_{-1,1}^{\sigma} \\ G_{0,-1}^{\sigma} & G_{0,0}^{\sigma} & G_{0,1}^{\sigma} \\ G_{1,-1}^{\sigma} & G_{1,0}^{\sigma} & G_{1,1}^{\sigma} \end{bmatrix}, \tag{6}$$

where

(i) x_n^l and x_n^j are the states of the second-order abstract neuron,

(ii) y is the nonlinear output of the abstract neuron,

(iii) τ^l and τ^j are the time constants of the different layers in various retina channels (see [5]),

(iv) f^r and f^o are the receptor and the output transfer function,

(v) Sn and Sm are the local neighborhood of the neuron,

(vi) C is the diffusion type template (see (4)), receptive field- (RF-) type spatial weighting at intralayer synapses,

(vii) λ is the coupling parameter (space constant) of a given layer,

(viii) G is the Gauss-type template (see (5) and its matrix-form (6)), RF-type spatial weighting at intra-/interlayer synapses, where g is the gain parameter of the receptor

(ix) σ is the sigma (spatial property) parameter of the receptor,

(x) Sy1, Sy2, and Sy3 are the plain, delayed, and desensitized receptor set, respectively,

(xi) r is the ratio parameter, and s is the feedback gain (strength of the interaction).

Subsequent layers supply the input of the next layer through synapses. The abstract neuron has receptors to implement these synapses. The three different types of receptors are plain, delayed, and desensitizing. The differential equations of the receptors are the following:

$$\tau_{nm}^r \dot{x}_{nm}^r = -x_{nm}^r + \sum_{kl\in \mathrm{Sm}} f_{nm}^r (G_{nm,kl}^{\sigma} y_m^l), \qquad \tau_{nm}^d \dot{x}_{nm}^d = -x_{nm}^d + x_{nm}^r, \tag{7}$$

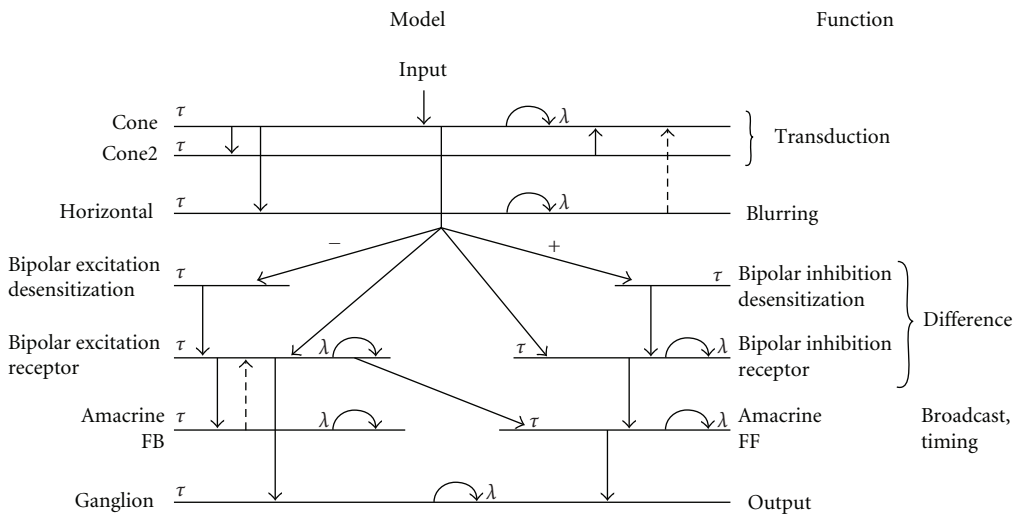

Figure 1: General neuromorphic structure of the light-adapted mammalian retina model. The horizontal lines represent different layers showing their proper cell names and functions. The vertical dashed arrows denote the inhibitory synapses while continuous arrows show the excitatory synapses. These connections can be varied for each specific retina channels. Each diffusion layer can be associated with a diffusion constant (called space constant) and a local time constant.

where

(i) x^r_{nm} and x^d_{nm} are the states of the receptor and the desensitized receptor,

(ii) τ^r and τ^d are the delay and the desensitizing speed (time constants) of the receptor.

2.2. Requirements. The light-adapted mammalian retina model consists of several main parts. These main parts are the outer retina model and the different ganglion models. The outer retina is a 3-layered uniform block (generally called outer plexiform layer), which transforms the input stimulus to the different, parallel ganglion models (channels).

The ganglion models contain three functional blocks: the excitation pattern generator, the inhibitory subsystem, and the ganglion cell model. This middle abstraction-level model can be transformed into a low-level multilayer CNN structure as shown in Figure 1. Each layer has its own time constant and connection weights. The interlayer connections are zero neighborhood neuron to neuron links that have linear- or rectifier-type, nonlinear transfer functions.

Constructing a single channel of the retina model at least 10 vertically stacked CNN layers are required. The whole structure cannot be directly implemented on the two-layer complex-cell analog VLSI CNN-UM chip (e.g., CACE1k) [6]. Additionally, the size of the array is just 32×32 cells, which is not sufficient for most of practical applications where larger image size is required (e.g., 174×144 sized cell array in QCIF format). Another analog VLSI implementation is a handy camera exploration device implemented on a stand-alone Bi-I system [12], which is capable of computing four retina channels in real time. However, the relatively low accuracy and moderate array size of these analog chips (128×128) do not make it possible to utilize their high computing power during the correct model building and parameter tuning steps of the retina modeling.

3. Objective: Retina Model Implementation on FPGA

The elaborated Falcon-multilayer (ML) emulated-digital CNN-UM architecture based on FPGA can be used to emulate a globally connected multilayer CNN network. It has a modular structure, for example, the number of layers and processors elements, the size and the number of templates can be adjusted moreover, the bit widths of state, constant, and template values are configurable [10]. Large number of these parameters makes it easy to synthesize the Falcon architecture, which is optimal for the retina channel computations. The control unit can be tailored to these configuration parameters.

The hardware complexity of Falcon-ML core increases in quadratic manner as the number of layers is increased. In the simplest case, for modeling one channel of the retina at least 10 layers should be emulated which means that 100 inter and intralayer connections and 300 multipliers (in case of nearest neighborhood templates where 3 multipliers are used to compute the template operation in 3 clock cycles) are required. Although implementing such a huge arithmetic unit of the single-channel retina model is already possible on the latest available reconfigurable FPGA chip, it consumes a lot of dedicated resources, therefore, some optimizations are required. This complex arithmetic unit can compute the derivatives and update the cell's state value in one clock cycle but it requires pipelining to achieve high clock speeds.

The main blocks of the optimized p-layered Falcon-ML architecture are shown in Figure 2. Depending on the template size, each mixer unit stores the surrounding cells of the currently processed one, while the memory units store

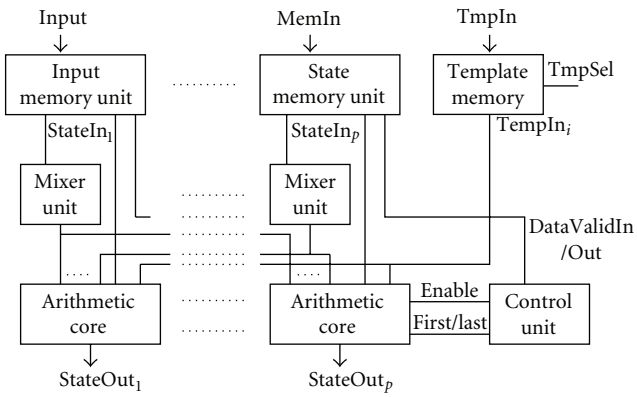

Figure 2: The structure of the optimized multilayer Falcon-ML architecture (p denotes the number of layers).

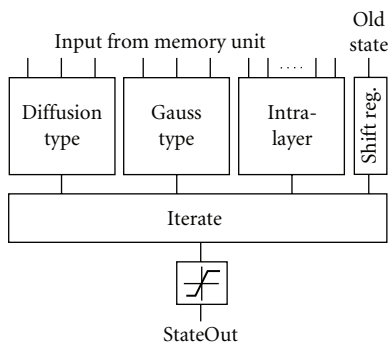

Figure 3: The block level structure of the optimized arithmetic core.

a one or two row-high belt from the given layer. Using this structure the I/O requirements of the processor are reduced to p load and p store operations per cell update. The optimized template memory contains only the parameters which are necessary to perform the computations, while the modified arithmetic units make efficient computation of the different type multilayer dynamics possible.

If we consider the different specific channels of the mammalian retina model, most of the interlayer links are zero, some of them are zero neighborhood templates and only a few feedback connections require a nearest neighbor template [5]. Additionally, the diffusion- and Gauss-type templates (see (4) and (6)) are symmetrical, which makes further optimizations possible. The block level structure of the optimized arithmetic unit is shown in Figure 3. To reduce the clock cycle time of the arithmetic unit, pipeline technique is used. According to (1), (2), and (7) the computation of the derivatives can be divided into the following three parts:

(i) computation with the zero neighborhood (intra- and interlayer) connections,

(ii) computation with the diffusion-type template,

(iii) computation with the Gauss-type templates.

Each layer has a separated and optimized arithmetic core which is connected to the mixer and memory units of the

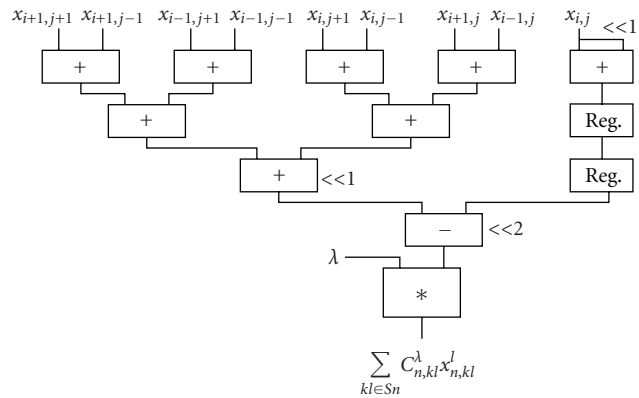

Figure 4: Structure of the optimized arithmetic unit to compute the diffusion-type template (pipeline registers are not shown for simplicity).

other layers according to the existing connections between the layers (see Figure 1). The structure of the arithmetic unit is shown in Figure 3.

The simplest element of the arithmetic unit is the intralayer block, which computes the inter- and intralayer zero neighborhood connections. This unit contains one multiplier for each connection, and the multiplied values are summed by an adder tree.

Due to the symmetry properties of the diffusion-type template (see (4)), the computation can be performed by the optimized circuit shown in Figure 4.

Multiplication with 2 and -12 is carried out by shifting operations and only one multiplier is required to compute the 3×3 template operation. This solution reduces the number of required multipliers from 3 to 1. Additionally, the number of clock cycles required to compute a new value is also reduced from 3 to 1 clock cycle, which significantly increases the computing performance of the processor.

The Gaussian template (see (6)) is also symmetrical but the ratio of the coefficient values is not an integer number. Therefore, at first the equally weighted state values are summed then these partial results are multiplied; finally the multiplied values are summed again. By using this optimized circuit shown in Figure 5, the number of multipliers is still three but the length of the computation cycle is reduced to one clock cycle.

The optimized Falcon-ML processor uses the discretized Forward-Euler method to compute the dynamics in each layer of the retina model. The Iterate block of the arithmetic unit (in Figure 3) sums the computed parts of the derivative and computes the new state value of each cell. The time-step value h is restricted to be an integer power of two. This method allows us to perform the multiplication with h by shifts and does not require an additional multiplier.

Implementing process and testing the previously described arithmetic unit can be very time consuming, but using rapid prototyping techniques and high-level hardware description languages (such as Handel-C [13] from Agility Inc.) makes it possible to implement the optimized arithmetic units within shorter development

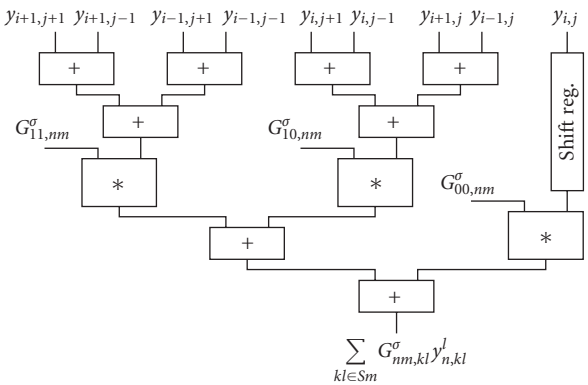

Figure 5: Structure of the optimized arithmetic unit to compute the Gauss-type template (pipeline registers are not shown for simplicity).

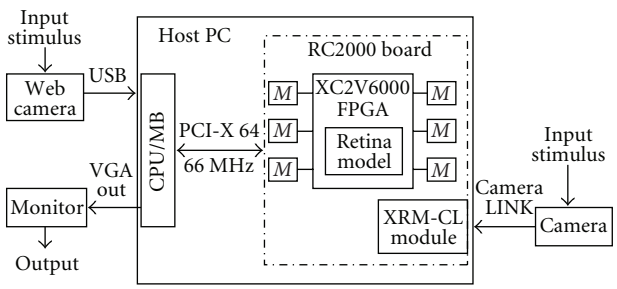

Figure 6: Test environment for RC2000 prototyping board.

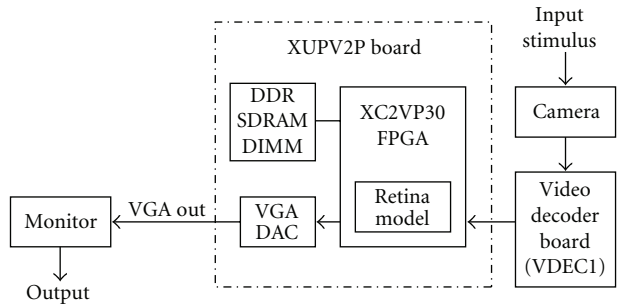

Figure 7: XUPV2P-based experimental system.

Table 1: General single-channel retina model requirements and available resources on different FPGAs.

Device utilization and speed	1-channel (10 layer) retina model @ 22 bit	Available on XC2V6000	Available on XC2VP30	Available on XC5SX240T
Num. of occupied slices	8496	33 792	13 696	37440*
Num. of BRAMs	69	144	136	516*
Num. of MULTs	35	144	136	1056*
Max. Core Freq. [MHz]	—	220	280	550

*Virtex-5 FPGAs are differently organized: a slice contains four 6-input LUT, 4-bit carry logic, and four registers; size of the Virtex-5 BRAM is 36 Kbit and the multipliers (DSP48E slice) are 25 × 18-bit wide.

time compared to the conventional HDL-based RTL level approaches, where the entire model hierarchy should be treated.

4. Real-Time Test Environments

The proposed specialized Falcon-ML architecture with single-channel and the extended multichannel model were implemented both on the RC2000 prototyping board (in Figure 6) from Agility Inc. [13] and on the XUPV2P development board (in Figure 7) from Digilent Inc. [14].

The PCI-based RC2000 board contains a Virtex-II FPGA (XC2V6000) and 24 MB ZBT SSRAM memory, which is organized in six 36-bit wide independent banks [13]. Indirectly, a simple web camera can be connected to the host PC with a USB cable and it can communicate with the RC2000 board across the PCI-64 bit/66 MHz bus. Alternatively, using the XRM CameraLINK module, we can attach an external digital camera directly to the standard XRM-CL interface. This test environment is represented in Figure 6.

Our another experimental system was the XUPV2P prototyping board based on Virtex-II Pro FPGA (XC2VP30) which is supplied with 256 MB DDR SDRAM memory [14]. Using the Digilent VDEC1 video decoder add-on board [15] makes it possible to connect an NTSC camera directly to the FPGA board and it sends input video frames to the processor with 25 fps, shown in Figure 7.

In both cases a monitor displays the computed results of the various channels of retina model.

5. Device Utilization and Performance

The available resources and the device utilizations for some different FPGA processors are summarized in Table 1. The single-channel retina model implementation consume large amount from the available chip resources, even at 22-bit state precision. This is the required accuracy to provide qualitatively correct responses for input stimulus (as we will see in Section 7).

Architecture of the Xilinx Virtex-5 devices is different from the previous Virtex families, therefore, area requirements of the retina model implementation is smaller on these devices. Additionally, these new devices have considerably more general logic and dedicated resources. Due to these considerations, the number of implementable Falcon-ML processor elements can be increased, which makes it possible to emulate multiple retina-channels in video real time.

As can be seen in Figure 8, the logic resource occupancy of the Xilinx Virtex-II Pro-30 FPGA is between 35.1% and 73.6% depending on the computing precision, while the Xilinx Virtex-II 6000 has significantly more logic resources and only 14.2% to 29.8% of the device is consumed by the single-channel retina model. These moderate-sized FPGAs

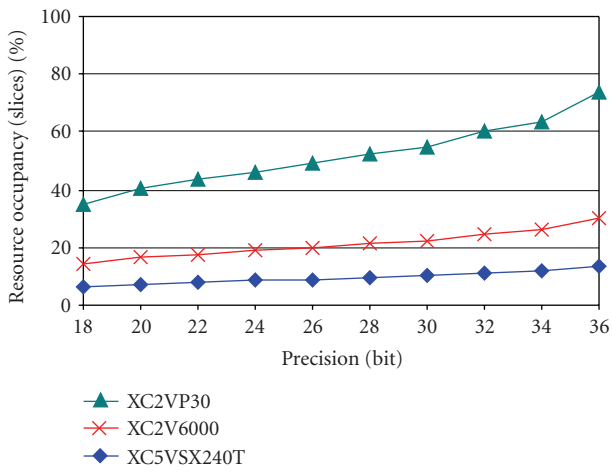

Figure 8: Logical resource (slice) requirements on different FPGAs.

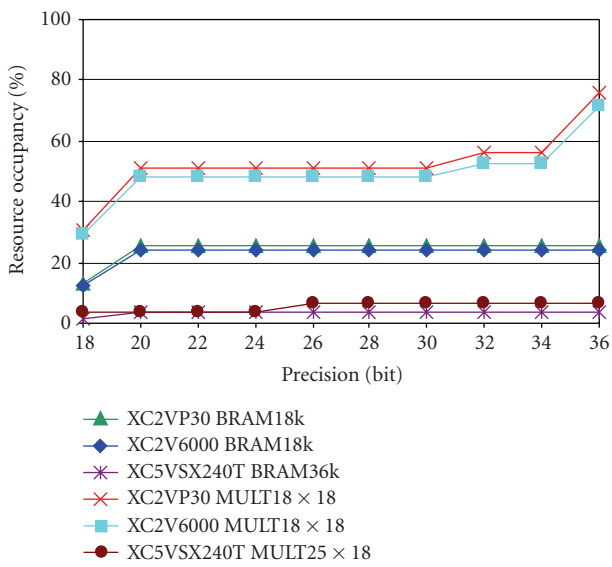

Figure 9: Dedicated resource (multiplier, block-ram) requirements on different FPGAs.

have similar amount of dedicated resources and about 50% of Mult18 × 18 components and about 25% of BRAM18k blocks are required for the implementation as shown in Figure 9, considering the qualitatively correct 22–24 bit state precision.

If we use the state-of-the-art Virtex-5 SX240T, DSP-specialized FPGA, only a small fraction of the available resources are used at 22–24 bit precision: 8.2% of slices, 3.4% of BRAM36Ks block memories, and 4% of DSP48E multipliers are utilized.

The number of the implementable processor cores is strongly related to the resource occupancies for different reconfigurable Xilinx chips. This is summarized in Figure 10. By using our Virtex-II Pro30 and Virtex-II 6000 moderate-sized, low-cost (below $1.000) FPGAs only 1-2 processor core(s) of the general, single-channel, multilayer retina

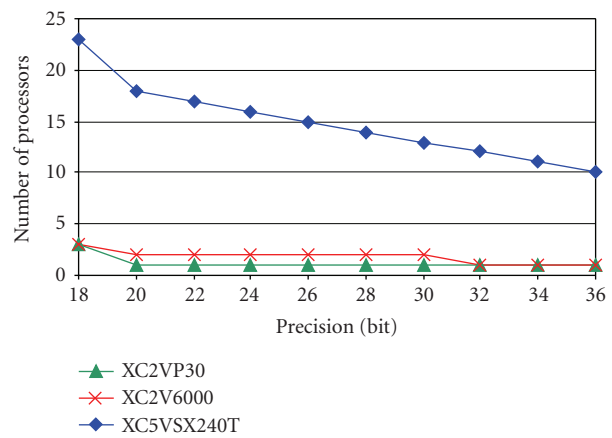

Figure 10: Number of implementable processor cores on different FPGAs.

model can be implemented at acceptable precision (~22–24 bit) that provides qualitatively correct output results. The main limitations in these two cases are the moderate number of dedicated multiplier (signed, 18-bit 2's complement MULT18 × 18) resources.

However, constructing a specific retina model for each different channel instead of using the general model, we can save additional resources by eliminating some unwanted connections. On the currently available largest Virtex-5 FPGA (XC5VSX240T) which is optimized for DSP- and memory-intensive applications even 17 Falcon-ML processor cores can be implemented to speed up the computations [16]. In this case, the number of available logical slices determines the number of implementable Falcon-ML processor elements. Since Virtex-5 family contains a huge number of 25×18-bit wide DSP48E multiplier blocks (1056), therefore, in the optimized arithmetic core of the Falcon-ML processor only a few number of dedicated multiplier blocks are required at the 22–24 bit precision. For this reason, more (or all) channels of the CNN-based mammalian retina model can be emulated simultaneously.

The performance of the elaborated architecture is scaled linearly according to the number of the processors. Computing performance compared to the speed of an Intel Core2 Duo T7200 [17] microprocessor running on 2 GHz clock frequency is shown in Figure 11. During the comparison proprietary-optimized software was run on the microprocessors which directly solved (1), (2), and (7) using the discretized Forward-Euler method.

The results of the general retina model show that even in the case of the moderate sized XC2V6000 FPGA at least 150 times higher computing performance can be achieved at the adequate 22–24-bit precision compared to the software simulation. If the state-of-the-art specialized XC5VSX240T FPGA is used, the computations can be carried out more than 3200 times faster.

The processing time increases in linear manner as the number of pixels is increased. The maximum size of the processed image is determined by the available on-chip

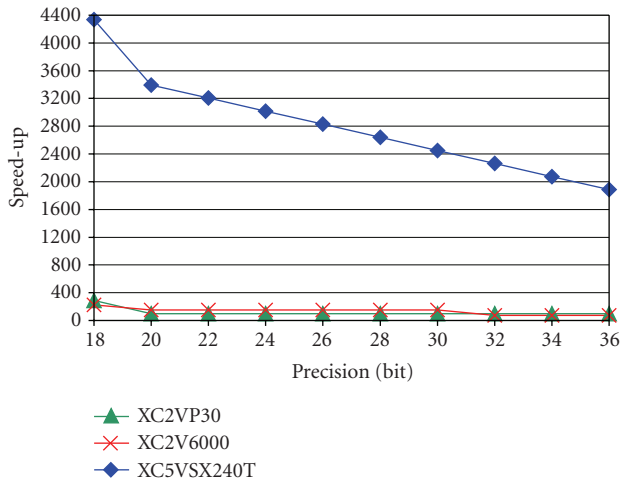

Figure 11: Performance of different implementations compared to an Intel Core2 Duo processor running on 2 GHz.

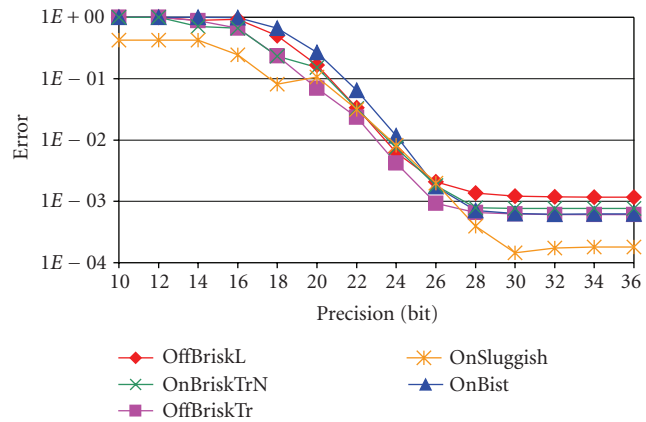

Figure 12: Differences of the 64 bit floating-point and the fixed-point result of the various retina channels.

BRAM memories on a given FPGA. In case of single-channel retina model and without using external memory the maximum size of the processed images can reach 64×64 on Virtex-II and Virtex-II Pro FPGA, while about 256×256 on the Virtex-5 FPGA architecture. To process high-resolution images the external onboard memory is required to store the state values of the retina model. So the processing time will be about an order of magnitude slower due to the memory bandwidth limitation.

The high computing performance of the largest available XC5VSX240T FPGA makes video real-time emulation of multiple retina channels possible. The clock frequency of the architecture is 550 MHz and only one clock cycle is required to compute a new cell value. The cumulative performance of the 17 implementable processor cores at qualitatively acceptable 22–24 bit precision can reach 9.35 billion cell iteration/sec. The high performance of this processor is capable of computing 256×256-sized cell array in real-time (at 30 frames/sec) and the different channels of the retina can be emulated by using 2^{-7} millisecond timestep.

Additionally, the large number of implementable Falcon processors makes it also possible to emulate more than one specific channel of the retina. In this case only one three-layered, uniform outer retina block is required and the processor is extended to emulate additional excitation, inhibition, and ganglion blocks. The modular structure of the memory and the arithmetic unit makes this extension possible easily. The Handel-C [13] high-level hardware description language makes implementation process and testing of the new processor much faster and simpler than the conventional RTL-based HDL level approaches.

Unfortunately, expansion of the neuromorphic retina model with any new layer increases the required I/O bandwidth significantly and several clock cycles are necessary to load the state values and save the results. Although the arithmetic unit can update a value in every clock cycle this high computing power cannot be utilized because several dummy cycles must be inserted until the I/O operations are carried out. Due to the architectural properties of the moderate sized Virtex-II and Virtex-II Pro FPGAs the memory units must be configured 1024 element-wide. In most cases much smaller cell array size is required hence the memory unit is also underutilized. To improve the utilization of the memory and the arithmetic unit virtual processors are implemented. These virtual processors are chained one after another and each of them works on different Forward-Euler iteration. In case of the memory unit the data of the different virtual memory units are stored on subsequent addresses. Similarly the arithmetic unit is used by the different virtual arithmetic units in subsequent clock cycles. Implementation of the virtual processors incurs minimal overhead in area. In most cases only shift registers are required, which can be very efficiently implemented on the FPGA. By using virtual processors the utilization of the memory and the arithmetic unit can be improved which results in higher performance.

6. Accuracy and Results

Performance of the processor can be significantly improved by decreasing the computational precision, but low precision gives inaccurate solution. We should find a balance between computing precision and solution accuracy. The results of the different fixed-point computations are compared to the 64-bit floating point results. Not knowing the exact analytical solution of a complex numerical problem, the double precision floating point implementation is considered as the accurate solution. The absolute maximum difference between the different solutions is represented in Figure 12.

A simple test case is computed by using different computing precision to approximate the accuracy of the results as it can be seen in Figure 13.

In case of very low precision, under 14 bit, the error values for all retina channels are very high because the model does not respond to the input. At least 16–18 bit precision is required to get some response on the output of the model. If the precision is improved further, the error of the solution decreases quickly (see Figure 12). If the dynamics of the retina model should be computed accurately, 28–30 bit

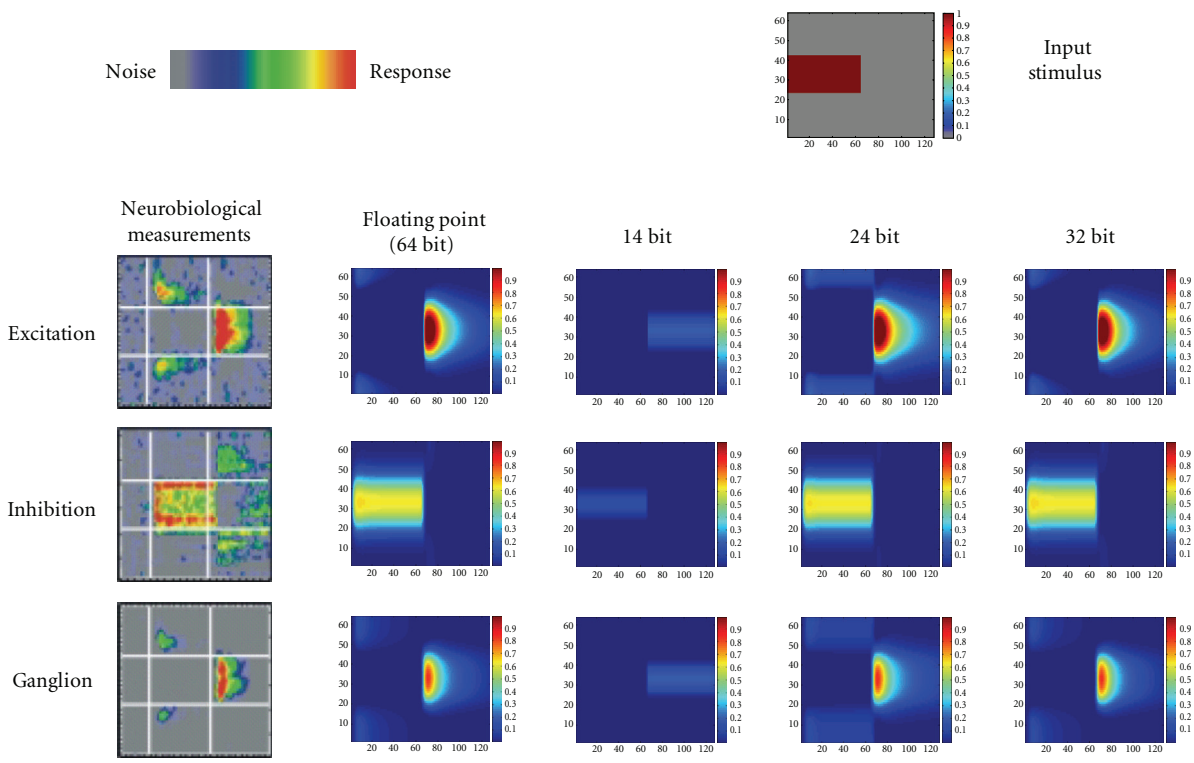

Figure 13: Accuracy of our solution at different bit widths (in case of OffBrisk-Linear channel type) compared to the results of neurobiological measurements derived from [5]. The simple test input is a commonly used flashed square stimulus. The time runs horizontally while space is on the vertical axes. The total examined time period is 2 seconds, where the flashing white stimulus is "turned-on" for 1 second.

precision is required while qualitatively correct results can already be obtained by using 22–24 bit precision. Input test stimulus of the model is the commonly used white flashed square, where the square is white for 1 second and uniform gray for 1 second. During the simulation the responses of different channels are computed by an FPGA of our experimental systems and finally the results are projected on the monitor. The spatiotemporal distributions of activity at different computing precisions are shown in Figure 13. The results qualitatively reproduce the inhibition, excitation, and spiking patterns of the neurobiological measurements for the flashed square stimulus.

The main goal of our emulated-digital implementation is to reproduce the results of neurobiological measurements and we are not focused on solving the computation of the dynamics as accurately as possible. The integration step size (h) is tightly related to the timing parameters (τ) and to the accuracy of the solution. Proper settings of the τ parameters are deduced from [5] (the details are shown in Table 1). The timing parameters can be of very different values of the subsequent layers within a unique retina channel and the smallest one will determine the value of the time-step.

Several higher order numerical methods are examined during the computation of the CNN dynamics and summarized in [18, 19]. On the one hand, if we use the higher-order Runge-Kutta (RK-4) formula the step-size can be doubled compared to the Forward-Euler method, but it requires four times more computations as the explicit Euler formula. It will imply that the Falcon architecture based on the higher-order Runge Kutta method requires four times more multiplier blocks, therefore, the area occupancy on FPGA will also be significantly increased. So this method is not worth to be implemented on an FPGA in the case of the retina model computations, because four-time larger area, which results in four-time smaller performance, is required to halve the number of steps.

On the other hand, during our computations the Forward-Euler method is used which is commonly applied in the calculations of the CNN dynamics. Although the Forward-Euler method employed in the Falcon-ML architecture computes the dynamic with smaller and fixed step-size as the Runge Kutta-4 method and it has a limited precision, the computations will be numerically stable if the step-size is selected carefully.

7. Example

Multiple channels of the neuromorphic, multilayer retina model can be implemented on our experimental systems. The incoming video flow (at 25 fps) is captured frame-by-frame by a standard camera, then these frames are processed according to the various retina channels of the model in real-time by using FPGA, and finally are displayed on a monitor. Snapshot of the output of the system for two different retina

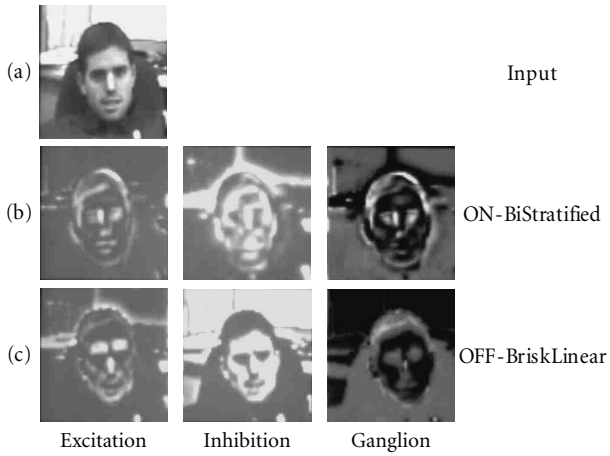

Figure 14: In this example two different (ON-, and OFF-type) retina channels are computed in video real-time by using our experimental system.

channels is shown in Figure 14. The size of the processed picture is set to 64×64, while time-step value h is adjusted as 2^{-7} millisecond. Row (a) shows the natural input stimulus captured by a camera. Row (b) denotes the first retina channel called OnBiStratified, while Row (c) demonstrates the second channel called OffBrisk-Linear. Each column represents the biologic name of the cells: bipolar excitation, inhibition, and ganglion cell output (spiking), respectively.

8. Conclusion

Our experimental systems with extended emulated-digital CNN-UM architecture based on the optimized Falcon-ML processor core were successfully designed to solve the state equations of the multilayer retina model. Due to the modular structure, it rapidly provides implementable specialized processors to emulate different retina models and multiple retina channels on the low-cost (referring to the ASIC implementations), moderate-sized FPGA prototyping boards as well. The computing precision of the processor is configurable in wide range making a trade-off between area and performance. Even using moderate precision (at 22–24 bit) the fixed-point results are very close to the floating-point result, which is considered as analytical result. If the precision is increased further to 30-bit, the accuracy of the fixed-point solution is comparable to the accuracy of the floating-point result. This provides for using our proposed architecture during the rapid model building and parameter tuning steps of various retina modeling where fast and accurate solution of the model is required.

On the currently available largest FPGAs these computations can be performed approximately three orders of magnitude faster than an Intel Core2 Duo microprocessor running at 2 GHz clock frequency. On the proposed architecture the cumulated performance of the Falcon-ML processors makes video real-time emulation possible in case of multiple retina-channels and large-sized (e.g., 256×256) cell arrays.

To sum it up, our proposed emulated-digital implementation on FPGA, based on CNN framework of the mammalian retina model, is capable of reproducing qualitatively the same full set of space-time activity patterns as the living retina in response to the different input stimulus.

References

[1] A. Valberg, *Light Vision Color*, John Wiley & Sons, New York, NY, USA, 2005.

[2] M. Meister and M. J. Berry II, "The neural code of the retina," *Neuron*, vol. 22, no. 3, pp. 435–450, 1999.

[3] B. Roska and F. Werblin, "Vertical interactions across ten parallel, stacked representations in the mammalian retina," *Nature*, vol. 410, no. 6828, pp. 583–587, 2001.

[4] F. Werblin and A. Jacobs, "Using CNN to unravel space-time processing in the vertebrate retina," in *Proceedings of the 3rd IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '94)*, pp. 33–40, Rome, Italy, December 1994.

[5] D. Bálya, B. Roska, T. Roska, and F. S. Werblin, "A CNN framework for modeling parallel processing in a mammalian retina," *International Journal of Circuit Theory and Applications*, vol. 30, no. 2-3, pp. 363–393, 2002.

[6] Cs. Rekeczky, T. Serrano-Gotarredona, T. Roska, and A. Rodríguez-Vázquez, "A stored program 2nd order/3-layer complex cell CNN-UM," in *Proceedings of the 6th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '00)*, pp. 213–217, Catania, Italy, May 2000.

[7] Z. Nagy, Z. Vörösházi, and P. Szolgay, "An emulated digital retina model implementation on FPGA," in *Proceedings of the 9th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '05)*, pp. 278–281, Hsinchu, Taiwan, May 2005.

[8] D. Bálya, B. Roska, T. Roska, and F. S. Werblin, "A CNN model framework and simulator for biological sensory systems," in *Proceedings of the 15th European Conference on Circuit Theory and Design (ECCTD '01)*, pp. 357–360, Espoo, Finland, August 2001.

[9] "RefineC—Receptive Field Network Calculus reference manual (version 2.5)," http://lab.analogic.sztaki.hu/Candy/refinec_UsersGuide.pdf.

[10] Z. Nagy and P. Szolgay, "Configurable multilayer CNN-UM emulator on FPGA," *IEEE Transactions on Circuits and Systems I*, vol. 50, no. 6, pp. 774–778, 2003.

[11] Z. Nagy and P. Szolgay, "Numerical solution of a class of PDEs by using emulated digital CNN-UM on FPGAs," in *Proceedings of the 16th European Conference on Circuits Theory and Design (ECCTD '03)*, vol. 2, pp. 181–184, Kraków, Poland, September 2003.

[12] D. Balya and B. Roska, "A handy retina exploration device," in *Proceedings of the 9th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '05)*, pp. 223–226, Hsinchu, Taiwan, May 2005.

[13] Agility Design Solutions Inc., 2008, http://www.agilityds.com.

[14] Digilent Inc. XUPV2P, "Virtex-II Pro Development System," 2008, http://www.digilentinc.com/XUPV2P.

[15] Digilent Inc., "VDEC-1 Video Decoder Add-on Board," 2008, http://www.digilentinc.com.

[16] Xilinx Products, 2008, http://www.xilinx.com.

[17] "Intel Core2 Duo and Intel Core2 Solo for Intel Centrino Duo Processor Technology—Specification," November 2007, http://www.intel.com/design/core2duo/documentation.htm.

[18] Z. Szlávik, D. Bálya, and T. Roska, "Properties of the adaptive integration formula to compute the CNN dynamic equations," in *Proceedings of the 16th European Conference on Circuits Theory and Design (ECCTD '03)*, vol. 1, pp. 105–108, Kraków, Poland, September 2003.

[19] H. Harrer, A. Schuler, and E. Amelunxen, "Comparison of different numerical integration methods forsimulating cellular neural networks," in *Proceedings of IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '90)*, pp. 151–159, Budapest, Hungary, December 1990.

Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 923404, 11 pages
doi:10.1155/2009/923404

Research Article

Simulation of Two-Dimensional Supersonic Flows on Emulated-Digital CNN-UM

Sándor Kocsárdi,[1] Zoltán Nagy,[2] Árpád Csík,[3] and Péter Szolgay[2,4]

[1] *Department of Image Processing and Neurocomputing, Faculty of Information Technology, University of Pannonia, Egyetem 10, 8200 Veszprém, Hungary*
[2] *Cellular Sensory and Wave Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences, 1518 Budapest, Hungary*
[3] *Department of Mathematics and Computational Sciences, Széchenyi István University, 9026 Győr, Hungary*
[4] *Faculty of Information Technology, Pázmány Péter Catholic University, 1083 Budapest, Hungary*

Correspondence should be addressed to Sándor Kocsárdi, skocso@vision.vein.hu

Received 25 September 2008; Accepted 7 January 2009

Recommended by Victor M. Brea

Computational fluid dynamics (CFD) is the scientific modeling of the temporal evolution of gas and fluid flows by exploiting the enormous processing power of computer technology. Simulation of fluid flow over complex-shaped objects currently requires several weeks of computing time on high-performance supercomputers. A CNN-UM-based solver of 2D inviscid, adiabatic, and compressible fluids will be presented. The governing partial differential equations (PDEs) are solved by using first- and second-order numerical methods. Unfortunately, the necessity of the coupled multilayered computational structure with nonlinear, space-variant templates does not make it possible to utilize the huge computing power of the analog CNN-UM chips. To improve the performance of our solution, emulated digital CNN-UM implemented on FPGA has been used. Properties of the implemented specialized architecture is examined in terms of area, speed, and accuracy.

Copyright © 2009 Sándor Kocsárdi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

The CNN paradigm is a natural framework to describe the behavior of locally interconnected dynamical systems which have an array structure [1]. Therefore, it possesses an inherent potential in the fields of computational fluid dynamics and numerical analysis [2]. Unfortunately, analog CNN-UM chips suffer from technical limitations diminishing their efficiency in such practical applications. Their most notable deficiencies are the low precision (8 bits) and restricted usability in applications requiring nonlinear, space-variant templates in a multilayered structure. However, by implementing the concepts behind the CNN-UM technology on reconfigurable architectures, the cell model can be modified according to the numerical simulation of the physical phenomena under consideration [3, 4]. Simulation of a 2D compressible flow on CNN-UM was reported in [5] but this solution used customized floating-point number representation inside the arithmetic unit. Unfortunately, area requirements of the floating-point arithmetic units are quite high, therefore, parallelism of the arithmetic unit needs to be reduced which has a negative impact on computing performance.

In this paper, we focus on the numerical solution of the same hyperbolic system of the nonlinear Euler equations but using fixed-point numbers. Our aim is to find some optimal computational architecture satisfying the functional requirements with minimal required precision, while driving computing power toward its maximum level. Thus, we intend to perform the operations with the highest possible parallelism.

The structure of the paper is the following. In Section 2, we recall the theoretical bases of compressible, adiabatic fluid flows. The details of the numerical discretization technique are described in Section 3. The optimized Falcon processor with the CNN templates and the optimized fixed-point arithmetic unit are given in Sections 4 and 5. In Section 6, the

accuracy analysis of the fixed- and floating-point solutions is presented and the features of their implementation on FPGA units are investigated. Finally, conclusions are drawn in Section 7.

2. Fluid Flows

A wide range of industrial processes and scientific phenomena involve gas or fluids flows over complex obstacles, for example, air flow around vehicles and buildings and the flow of water in the oceans or liquid in BioMEMS. In engineering applications, the temporal evolution of nonideal, compressible fluids is quite often modeled by the system of Navier-Stokes equations. It is based on the fundamental laws of mass, momentum, and energy conservation, extended by the dissipative effects of viscosity, diffusion, and heat conduction. By neglecting all these nonideal processes and assuming adiabatic variations, we obtain the Euler equations [6, 7], describing the dynamics of dissipation-free, inviscid, compressible fluids. They are a coupled set of nonlinear hyperbolic partial differential equations, in conservative form expressed as

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) &= 0, \\ \frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla\cdot(\rho\mathbf{v}\mathbf{v} + \hat{I}p) &= 0, \\ \frac{\partial E}{\partial t} + \nabla\cdot((E+p)\mathbf{v}) &= 0, \end{aligned} \tag{1}$$

where t denotes time, ∇ is the nabla operator, ρ is the density, u, v are the x- and y-component of the velocity vector $\mathbf{v}$, respectively, p is the pressure of the fluid, $\hat{I}$ is the identity matrix, and E is the total energy density defined as

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho\mathbf{v}\cdot\mathbf{v}. \tag{2}$$

In (2), the value of the ratio of specific heats is taken to be $\gamma = 1.4$. For later use, we introduce the conservative state vector $\mathbf{U} = [\rho, \rho u, \rho v, E]^T$, the set of primitive variables $\mathbf{P} = [\rho, u, v, E]^T$, and the speed of sound $c = \sqrt{\gamma p/\rho}$. It is also convenient to merge (1) into hyperbolic conservation law form in terms of U and the flux tensor,

$$\mathbf{F} = \begin{pmatrix} \rho\mathbf{v} \\ \rho\mathbf{v}\mathbf{v} + Ip \\ (E+p)\mathbf{v} \end{pmatrix}, \tag{3}$$

as

$$\frac{\partial U}{\partial t} + \nabla\cdot\mathbf{F} = 0. \tag{4}$$

3. Discretization of the Governing Equations

Since logically structured arrangement of data is fundamental for the efficient operation of the FPGA-based implementations, we consider explicit finite volume discretization of the governing equations over structured grids employing a simple numerical flux function. Indeed, the corresponding rectangular arrangement of information and the choice of multilevel temporal integration strategy ensure the continuous flow of data through the CNN-UM architecture. In the followings, we recall the basic properties of the mesh geometry, and the details of the considered first- and second-order schemes.

3.1. The Geometry of the Mesh. For the sake of simplicity, in this paper, we only consider rectangular computational domains labeled by Ω. The sides of the rectangle are a and b units long. We divide Ω into $M \times N$ nonoverlapping rectangular finite volumes (cells) of equal sizes. The volume situated in the ith column and the jth row is indexed by (i, j). The resolution of the mesh in the x- and the y-directions coinciding with the length of the cells' edges are $\Delta x = a/M$ and $\Delta y = b/N$, thus the volume of the cell (i, j) is $V_{i,j}$. Following the finite volume methodology, we store all components of the volume-averaged state vector $U_{i,j}$ at the mass center of cell (i, j).

3.2. The Discretization Scheme. Application of the finite volume discretization method leads to the following semidiscrete form of governing equations (4)

$$\frac{dU_{i,j}}{dt} = -\frac{1}{V_{i,j}}\sum_f \mathbf{F}_f\cdot\mathbf{n}_f, \tag{5}$$

where the summation is meant for all four faces of cell (i, j), $\mathbf{F}_f$ is the flux tensor evaluated at face f and $\mathbf{n}_f$ is the outward pointing normal vector of face f scaled by the length of the face. Let us consider face f in a coordinate frame attached to the face, such that its x-axis is normal to f (see Figure 1). Face f separates cell L (left) and cell R (right). In this case, the $\mathbf{F}_f\cdot\mathbf{n}_f$ scalar product equals to the x-component of $\mathbf{F}(F_x)$ multiplied by the area of the face. In order to stabilize the solution procedure, artificial dissipation has to be introduced into the scheme. According to the standard procedure, this is achieved by replacing the physical flux tensor by the numerical flux function F^N containing the dissipative stabilization term. A finite volume scheme is characterized by the evaluation of F^N which is the function of both U_L and U_R. In this paper, we employ the simple and robust Lax-Friedrichs numerical flux function defined as

$$F^N = \frac{F_L + F_R}{2} - (|\overline{u}| + \overline{c})\frac{U_R - U_L}{2}. \tag{6}$$

In the last equation, $F_L = F_x(U_L)$ and $F_R = F_x(U_R)$ and notations $|\overline{u}|$ and $|\overline{c}|$ represent the average value of the u velocity component and the speed of sound at an interface, respectively. The temporal derivative is discretized by the first-order forward Euler method

$$\frac{dU_{i,j}}{dt} = \frac{U_{i,j}^{n+1} - U_{i,j}^{n}}{\Delta t}, \tag{7}$$

where $U_{i,j}^n$ is the known value of the state vector at time level n, $U_{i,j}^{n+1}$ is the unknown value of the state vector at time level $n+1$, and Δt is the time step.

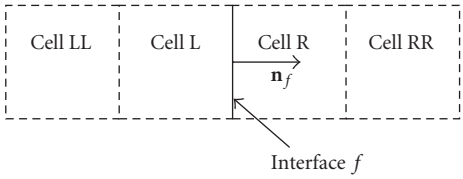

Figure 1: Interface with the normal vector and the cells required in the computation.

By working out the algebra described so far, it leads to the discrete form of the governing equations to compute the numerical flux term F and the dissipation term D,

$$
\begin{aligned}
F_i^{\rho,n} &= \frac{\rho u_C^n + \rho u_i^n}{2}, \quad i = E, W,\\
F_i^{\rho u,n} &= \frac{(\rho u^2 + p)_C^n + (\rho u^2 + p)_i^n}{2}, \quad i = E, W,\\
F_i^{\rho u,n} &= \frac{\rho u v_C^n + \rho u v_i^n}{2}, \quad i = N, S,\\
F_i^{\rho v,n} &= \frac{\rho u v_C^n + \rho u v_i^n}{2}, \quad i = E, W,\\
F_i^{\rho v,n} &= \frac{(\rho v^2 + p)_C^n + (\rho v^2 + p)_i^n}{2}, \quad i = N, S,\\
F_i^{E,n} &= \frac{(E + p)u_C^n + (E + p)u_i^n}{2}, \quad i = E, W,\\
F_i^{E,n} &= \frac{(E + p)v_C^n + (E + p)v_i^n}{2}, \quad i = N, S,\\
D_i^{\rho,n} &= (|\overline{u}| + \overline{c})\frac{\rho_i^n - \rho_C^n}{2}, \quad i = E, N,\\
D_i^{\rho,n} &= (|\overline{u}| + \overline{c})\frac{\rho_C^n - \rho_i^n}{2}, \quad i = W, S,\\
D_i^{\rho u,n} &= (|\overline{u}| + \overline{c})\frac{\rho u_i^n - \rho u_C^n}{2}, \quad i = E, N,\\
D_i^{\rho u,n} &= (|\overline{u}| + \overline{c})\frac{\rho u_C^n - \rho u_i^n}{2}, \quad i = W, S,\\
D_i^{\rho v,n} &= (|\overline{u}| + \overline{c})\frac{\rho v_i^n - \rho v_C^n}{2}, \quad i = E, N,\\
D_i^{\rho v,n} &= (|\overline{u}| + \overline{c})\frac{\rho v_C^n - \rho v_i^n}{2}, \quad i = W, S,\\
D_i^{E,n} &= (|\overline{u}| + \overline{c})\frac{E_i^n - E_C^n}{2}, \quad i = E, N,\\
D_i^{E,n} &= (|\overline{u}| + \overline{c})\frac{E_C^n - E_i^n}{2}, \quad i = W, S.
\end{aligned}
\tag{8}
$$

Complex terms in the equation were marked with only one super- and subscript for better understanding, for example, $(\rho u^2 + p)_C^n$ is equal to $\rho_C^n (u_C^n)^2 + p_C^n$. Additionally, in the subscripts E, W, N, and S denote the eastern, western, northern, and southern interfaces of the examined cell.

Finally, in (9), the update scheme for each layer can be seen based on (8),

$$
\begin{aligned}
\rho_C^{n+1} &= \rho_C^n - \frac{\Delta t}{\Delta x}(F_E^{\rho,n} - F_W^{\rho,n} + D_E^{\rho,n} - D_W^{\rho,n})\\
&\quad - \frac{\Delta t}{\Delta y}(F_N^{\rho,n} - F_S^{\rho,n} + D_N^{\rho,n} - D_S^{\rho,n}),\\
\rho u_C^{n+1} &= \rho u_C^n - \frac{\Delta t}{\Delta x}(F_E^{\rho u,n} - F_W^{\rho u,n} + D_E^{\rho u,n} - D_W^{\rho u,n})\\
&\quad - \frac{\Delta t}{\Delta y}(F_N^{\rho u,n} - F_S^{\rho u,n} + D_N^{\rho u,n} - D_S^{\rho u,n}),\\
\rho v_C^{n+1} &= \rho v_C^n - \frac{\Delta t}{\Delta x}(F_E^{\rho v,n} - F_W^{\rho v,n} + D_E^{\rho v,n} - D_W^{\rho v,n})\\
&\quad - \frac{\Delta t}{\Delta y}(F_N^{\rho v,n} - F_S^{\rho v,n} + D_N^{\rho v,n} - D_S^{\rho v,n}),\\
E_C^{n+1} &= E_C^n - \frac{\Delta t}{\Delta x}(F_E^{E,n} - F_W^{E,n} + D_E^{E,n} - D_W^{E,n})\\
&\quad - \frac{\Delta t}{\Delta y}(F_N^{E,n} - F_S^{E,n} + D_N^{E,n} - D_S^{E,n}).
\end{aligned}
\tag{9}
$$

The overall accuracy of the scheme can be raised to second order if the spatial and the temporal derivatives are calculated by a second-order approximation. One way to satisfy the latter requirement is to perform a piecewise linear extrapolation of the primitive variables P_{L} and P_{R} at the two sides of the interface in (6). This procedure requires the introduction of additional cells with respect to the interface, that is, cell LL (left to cell L) and cell RR (right to cell R) as shown in Figure 1. With these labels, the reconstructed primitive variables are

$$
\begin{aligned}
P_{\mathrm{L}} &= P_{\mathrm{L}} + \frac{g_{\mathrm{L}}(\delta P_{\mathrm{L}}, \delta P_C)}{2},\\
P_{\mathrm{R}} &= P_{\mathrm{R}} - \frac{g_{\mathrm{R}}(\delta P_C, \delta P_{\mathrm{R}})}{2},
\end{aligned}
\tag{10}
$$

with

$$
\begin{aligned}
\delta P_{\mathrm{L}} &= P_{\mathrm{L}} - P_{\mathrm{LL}},\\
\delta P_C &= P_{\mathrm{R}} - P_{\mathrm{L}},\\
\delta P_{\mathrm{R}} &= P_{\mathrm{RR}} - P_{\mathrm{R}}.
\end{aligned}
\tag{11}
$$

while g_{L} and g_{R} are the limiter functions. The scheme without limitation yields acceptable second-order time-accurate approximation of the solution, only if the variations in the flow field are smooth. However, the integral form of the governing equations admits discontinuous solutions as well, and in an important class of applications the solution contains shocks. In order to capture these discontinuities without spurious oscillations, in (10) we apply the minmod limiter function, also

$$
g_{\mathrm{L}}(\delta P_{\mathrm{L}}, \delta P_C) = \begin{cases} \delta P_{\mathrm{L}}, & \text{if } |\delta P_{\mathrm{L}}| < |\delta P_C|,\ \delta P_{\mathrm{L}} \delta P_C > 0,\\ \delta P_C, & \text{if } |\delta P_C| < |\delta P_{\mathrm{L}}|,\ \delta P_{\mathrm{L}} \delta P_C > 0,\\ 0, & \text{if } \delta P_{\mathrm{L}} \delta P_C \le 0. \end{cases}
\tag{12}
$$

The function $g_{\mathrm{R}}(\delta P_C, \delta P_{\mathrm{R}})$ can be defined analogously.

The temporal derivative is discretized by the standard two-stage Runge-Kutta method [8]. During the second-order update procedure, the primitive variables (ρ, u, v, and p) are computed from the conservative variables (ρ, ρu, ρv, and E) and extrapolated by using the limiter function. The resulting variables are used to compute the spatial derivatives (9) and time is advanced by half time step according to the second-order Runge-Kutta method. Finally, the whole procedure is repeated to compute the next timestep.

A vast amount of experience has shown that these equations provide a stable discretization of the governing equations if the time step obeys the following Courant-Friedrichs-Lewy (CFL) condition:

$$\Delta t \leq \min_{(i,j)\in([1,M]\times[1,N])} \frac{\min(\Delta x, \Delta y)}{|u_{i,j}| + c_{i,j}}. \tag{13}$$

4. Implementation on Falcon CNN-UM Architecture

The Falcon architecture [9] is an emulated digital implementation of CNN-UM array processor which uses the full signal range model. On this architecture, the flexibility of simulators and computational power of analog architectures are mixed. Not only the size of templates and the computational precision can be configured, but space-variant and nonlinear templates can also be used.

The Euler equations were solved by a modified Falcon processor array in which the arithmetic unit has been changed according to the discretized governing equations.

Since each CNN cell has only one real output value, four layers are required to represent the variables ρ, ρu, ρv, and E. In case of a simple first-order forward Euler temporal discretization, the nonlinear CNN templates acting on the ρu layer can easily be taken from the discretized equations. Equations (14) show templates in which cells of different layers at positions (k, l) are connected to the cell of layer ρu at position (i, j),

$$A_1^{\rho u} = \frac{1}{2\Delta x}\begin{bmatrix} 0 & 0 & 0 \\ \rho u^2 + p & 0 & -(\rho u^2 + p) \\ 0 & 0 & 0 \end{bmatrix},$$

$$A_2^{\rho u} = \frac{1}{2\Delta x}\begin{bmatrix} 0 & -\rho uv & 0 \\ 0 & 0 & 0 \\ 0 & \rho uv & 0 \end{bmatrix}, \tag{14}$$

$$A_3^{\rho u} = \frac{1}{2\Delta x}\begin{bmatrix} 0 & \rho v & 0 \\ \rho u & -2\rho u - 2\rho v & \rho u \\ 0 & \rho v & 0 \end{bmatrix}.$$

The template values for ρ, ρv, and E layers can be defined analogously.

In accordance with (9), we have designed four complex circuits. These are able to update the values of the conservative state vector of a cell in every clock cycle using emulated digital CNN-UM architecture. The arithmetic unit for the computation of the ρu layer is shown in Figure 2. The $\rho uu+p$, ρuv, ρu, and ρv terms can be reused during the computation of the neighboring cells and they should be computed only once in each iteration step. This solution requires additional memory elements but greatly reduces the area requirement of the arithmetic unit.

Other trick can be applied if we choose the ratio of Δt and Δx or Δy to be integer power of two because the multiplication with $\Delta t/\Delta x$ and $\Delta t/\Delta y$ can be done by shifts so we can eliminate several multipliers from the hardware and additionally the area requirements will be greatly reduced.

Unfortunately, in the second-order case, limiter function should be used on the primitive variables and the conservative variables are computed from these results. The limited values will be different for the four interfaces and cannot be reused in the computation of the neighboring cells. Therefore, this approach does not make it possible to derive CNN templates for the solution. However, a specialized arithmetic unit still can be designed to compute the second-order update scheme described in the previous section directly.

In accordance with the discretized governing equations, we have designed a complex circuit which is able to update the values of the conservative state vector of a cell in every clock cycle using emulated digital CNN-UM architecture. The main building blocks of the proposed unit are shown in Figure 3(a). From the blocks, two identical arithmetic cores can be built according to the two steps of the second-order Runge-Kutta method. In order to get the conservative state values at time level $n + 1$, the two identical units need to be applied successively. The arithmetic core computing ρu value after the first step can be seen in Figure 3(b). Two similar units (F_N and F_E) are required to compute the flux value at the North and South or East and West interfaces while four instances of the third unit (D_E) is required to compute the artificial diffusion term. Inputs of these units are connected to the output of the appropriate limiter units. In order to achieve the highest possible clock speed during the computation, pipelining technique and parallel working hardware units have been used.

5. Fixed-Point Arithmetic Unit

FPGA implementation of the previously described arithmetic unit using floating-point IP cores was reported in [5]. The results show that even computing with 32-bit single precision numbers, the currently available largest FPGAs are required for the implementation. Size of the arithmetic unit is greatly increased by the area requirements of the floating-point adders.

Some previous studies proved the effectiveness of fixed-point numbers during the solution of simple PDEs [10]. In case of simple PDEs, all bits computed during the evaluation of the derivative are kept and rounding is carried out at the last step when the state value is updated. Unfortunately, this method cannot be used in our case because the bit width of the partial results is growing quickly as shown in Figure 4(a). To reduce the bit width inside the arithmetic unit and reduce

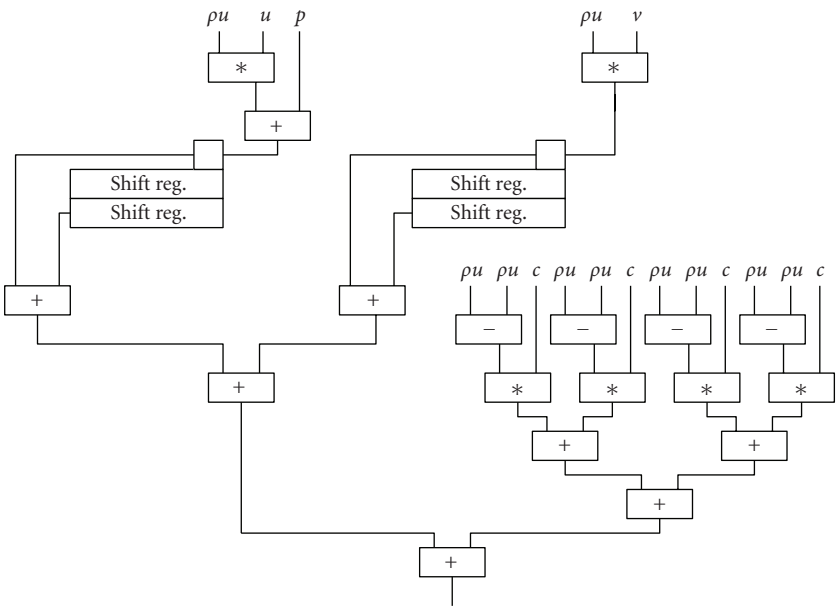

Figure 2: The proposed arithmetic unit to compute the derivative or ρu layer in the solution using first-order Lax-Friedrichs approximation method.

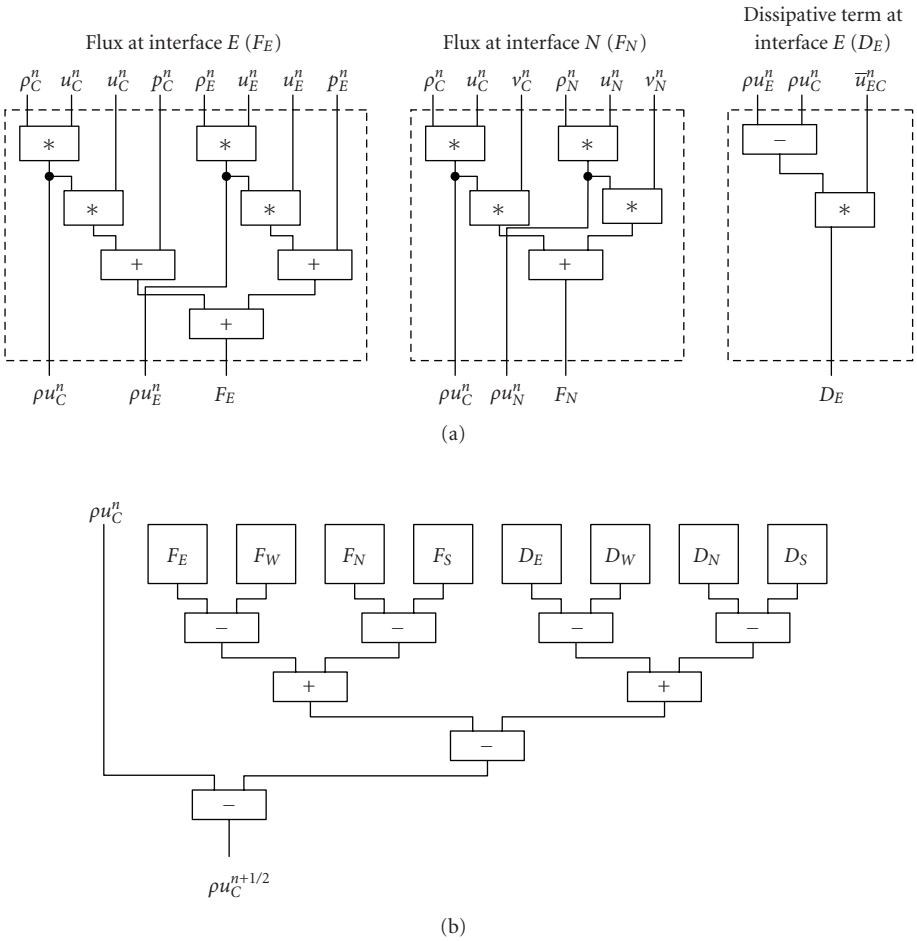

Figure 3: (a) The main building blocks of the proposed arithmetic unit, (b) the whole arithmetic unit built from the main blocks.

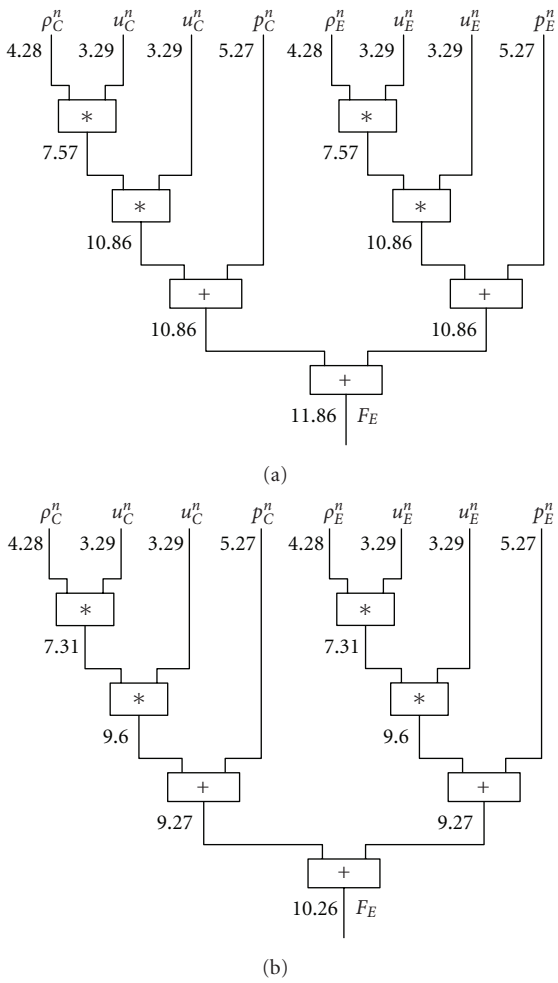

Figure 4: Bit width of the fixed-point arithmetic unit to compute F_E, (a) without optimization, (b) optimized by using interval arithmetic (bit width is denoted by (integer width) . (fractional width)).

area requirements, rounding is required. However, it should be carried out very carefully because important information required to accurately compute the derivative of a state value may be lost during improper rounding.

One possible solution to determine the number of fractional bits required during the computation is to use interval arithmetic [11] and compute the error of the operation along with the result. The basic arithmetic operations computed in interval arithmetic have the following form (m: computer representation of the number, ε: computer representation of the error):

$$m_1 \pm \varepsilon_1 + m_2 \pm \varepsilon_2 = (m_1 + m_2) \pm (\varepsilon_1 + \varepsilon_2), \tag{15a}$$

$$m_1 \pm \varepsilon_1 - m_2 \pm \varepsilon_2 = (m_1 - m_2) \pm (\varepsilon_1 + \varepsilon_2), \tag{15b}$$

$$m_1 \pm \varepsilon_1 \times m_2 \pm \varepsilon_2 = (m_1 \times m_2) \pm (\varepsilon_1 |m_2| + |m_1| \varepsilon_2 + \varepsilon_1 \varepsilon_2), \tag{15c}$$

$$m_1 \pm \varepsilon_1 \div m_2 \pm \varepsilon_2 = \left(\frac{m_1}{m_2}\right) \pm \left(\frac{\varepsilon_1 + |m_1/m_2| \varepsilon_2}{|m_2| - \varepsilon_2}\right). \tag{15d}$$

The error of the addition and subtraction is simply the sum of the error of the operands while in the case of multiplication and division, the error of the results also depends on the value of the operands.

In our case, we assume that a priori information is available about the maximum value of the input variables (this is usually true in engineering applications), which can be used to determine the number of integer and fractional bits. We also assume that the least significant bit (LSB) of the input values is erroneous, therefore, ε is set to $2^{-\mathrm{LSB}}$. Error of the additions and subtractions can be easily determined by using (15a)-(15b). However, to determine the error of the multiplication and division, the value of the operands are also required which is not known in advance. Therefore, a worst case analysis of the accuracy of the arithmetic unit should be carried out by computing the minimum and maximum values and the minimum and maximum errors of each partial result. The number of integer bits is computed from the maximal value while the number of fractional bits can be computed form the minimum error value by using the following equations:

$$\begin{aligned} \mathrm{int} &= \lceil \log_2(2 \cdot \max) \rceil, \\ \mathrm{frac} &= \lceil -\log_2(\varepsilon_{\min}) \rceil, \end{aligned} \tag{15}$$

where int is the number of integer bits, frac is the number of fractional bits, and max is the computed maximal value of the partial result, while its minimum error is denoted by $\varepsilon_{\min}$. The computed minimum error values represent the theoretically achievable accuracy of the computation. The LSB of the variable (and the smallest representable number $2^{-\mathrm{LSB}}$) should be set to be in the same range as the computed minimal error. If the number of fractional bits is smaller, valuable information is lost. On the other hand, using more fractional bits does not really improve the results. A small part of the arithmetic unit after the optimization (assuming $\rho_{\min} = 0.2$) is shown in Figure 4(b).

Without optimization, the results of the multiplications are stored on 64 and 96 bits and the output of the arithmetic unit (F_E) is 97-bit wide. If the results are used later during multiplications, the bit width is further increased and quickly hits an unpractical size. Using the previously described method, the width of the partial results can be significantly reduced. The width of the multiplications is decreased by 26 bits while the width of the final result is reduced to 36 bits from 97 bits. Area requirements of the arithmetic units are significantly decreased by using these optimizations while the operating frequency is improved.

6. Results and Performance

6.1. Area Requirements. During the implementation of the first- and second-order method, customized precision fixed-point arithmetic cores from Xilinx [12] are used. Implementation and testing of the previously described arithmetic unit can be very time-consuming but using rapid prototyping techniques and high-level hardware description languages such as Handel-C from agility [13] make it possible to

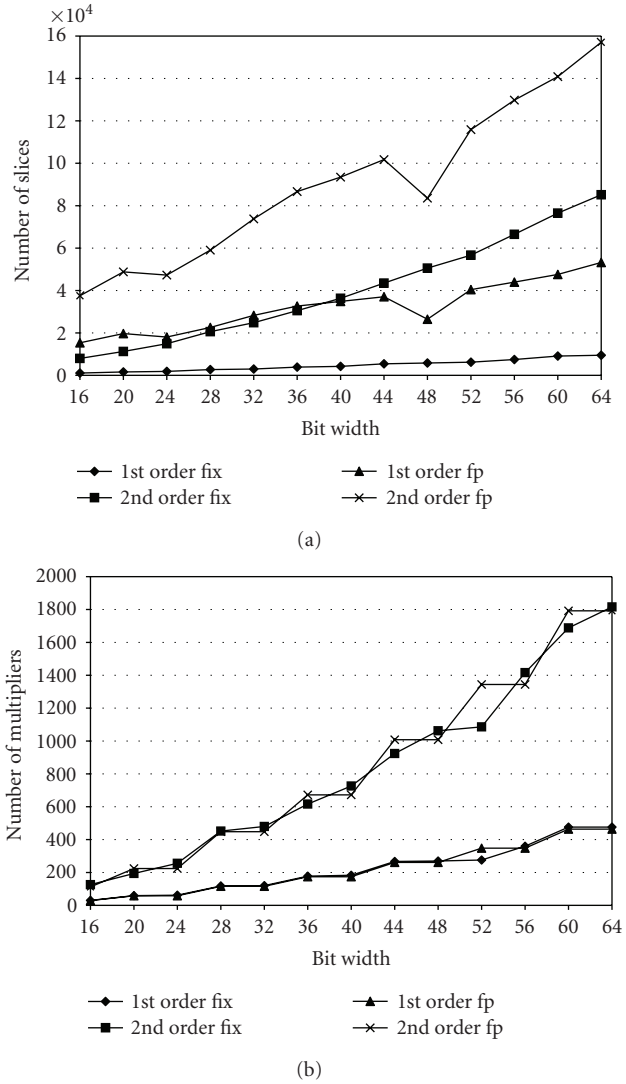

Figure 5: The area requirement of the fixed-point (fix) and floating-point (fp) arithmetic units using different precisions.

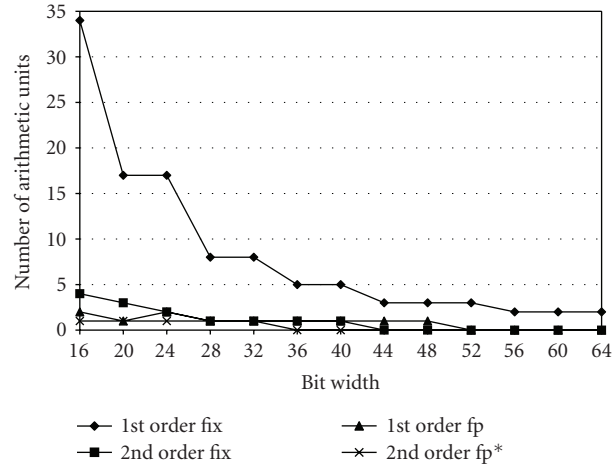

Figure 6: Number of implementable arithmetic units on Virtex-5 XC5VSX240T FPGA (*half arithmetic unit—two clock cycles per cell).

develop the optimized arithmetic unit much faster than using conventional VHDL-based approach.

Area requirement of the proposed fixed-point parallel arithmetic units along with the area requirements of the floating-point implementations [5] is shown in Figure 5 (in the following figures, bit width means the sum of the integer and fractional bits of the fixed-point numbers and the width of the mantissa bits in case of the floating-point numbers). Due to the large area requirements of the floating-point arithmetic units, especially the size of the floating-point adders, only the low precision configurations of the fully parallel first-order arithmetic unit can be realized even on the currently available largest FPGAs (Virtex-5 SX240T and LX330T). The fully parallel second-order arithmetic unit cannot be implemented on these devices when floating-point numbers are used. A possible solution could be for this problem if the two steps of the Runge-Kutta method are computed in two steps on the same arithmetic unit. In this case, area requirements can be halved but the computing performance is also halved.

Area requirements of the arithmetic unit can be significantly reduced, compared to the floating-point solution, by using fixed-point numbers and using the optimization method described in the previous section. The required number of dedicated multipliers is about to be equal in the case of fixed- and floating-point arithmetic. However, using fixed-point arithmetic 2–5 times fewer logic elements (slices) are required for the implementation of the first-order arithmetic unit. In the second-order case, the area is decreased more significantly by a factor of 5–15. The number of implementable arithmetic units on the DSP optimized Virtex-5 SX240T FPGA is summarized in Figure 6.

6.2. Test Setup. To show the efficiency of our solution, a complex test case was used, in which a Mach 3 flow over a forward facing step was computed. The simulated region is a two-dimensional cut of a pipe which has closed at the upper and lower boundaries, while the left and right boundaries are open. The direction of the flow is from left to right and the speed of the flow at the left boundary is 3-time the speed of sound constantly. The solution contains shock waves reflected from the closed boundaries. This problem was solved by using the Handel-C simulation of the previously described first- and second-order arithmetic units. In Figures 7 and 8, results of the computation using the derived methods after 0.4 second, 1.2 seconds, and 4 seconds of simulation time with 3.125 milliseconds (1/320 second) time step are shown. In these figures, the dissipative property of the first-order solution can be clearly recognized, while using the second-order method the boundary of the shock waves is sharp on the density distribution map. Because of the applied rectangular, regular grid system a mask was necessary to define the computational domain for the solution. The grid points under the step are masked out and do not take part in the solution resulting in dummy computing cycles. This problem can be eliminated from the system

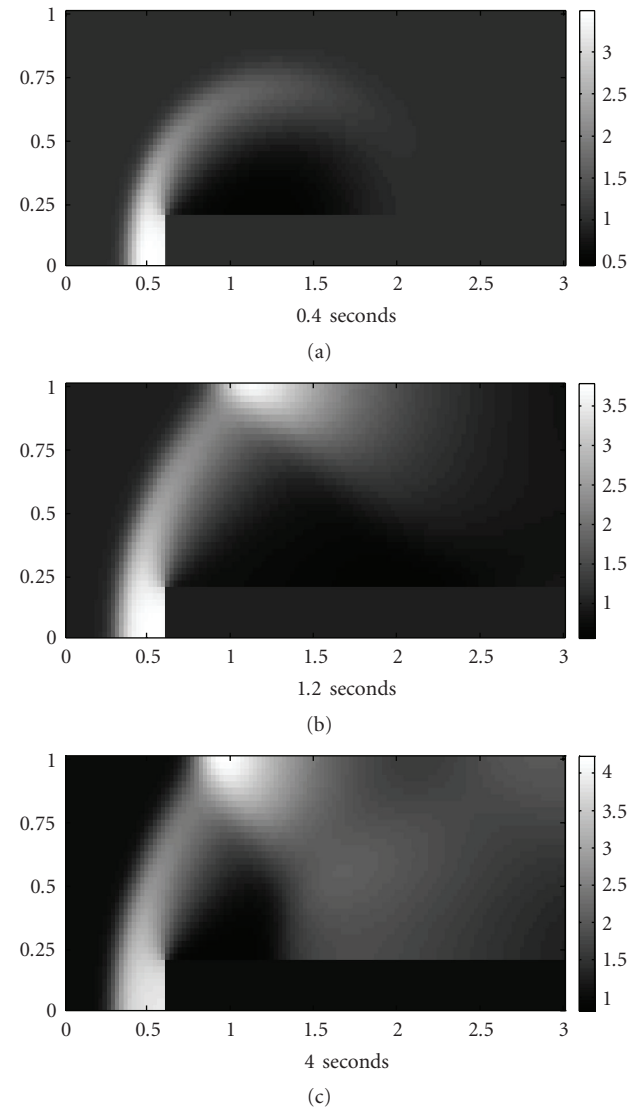

Figure 7: First-order solution of the Mach 3 flow on an 80×240 array after 0.4, 1.2, and 4 seconds of simulation time.

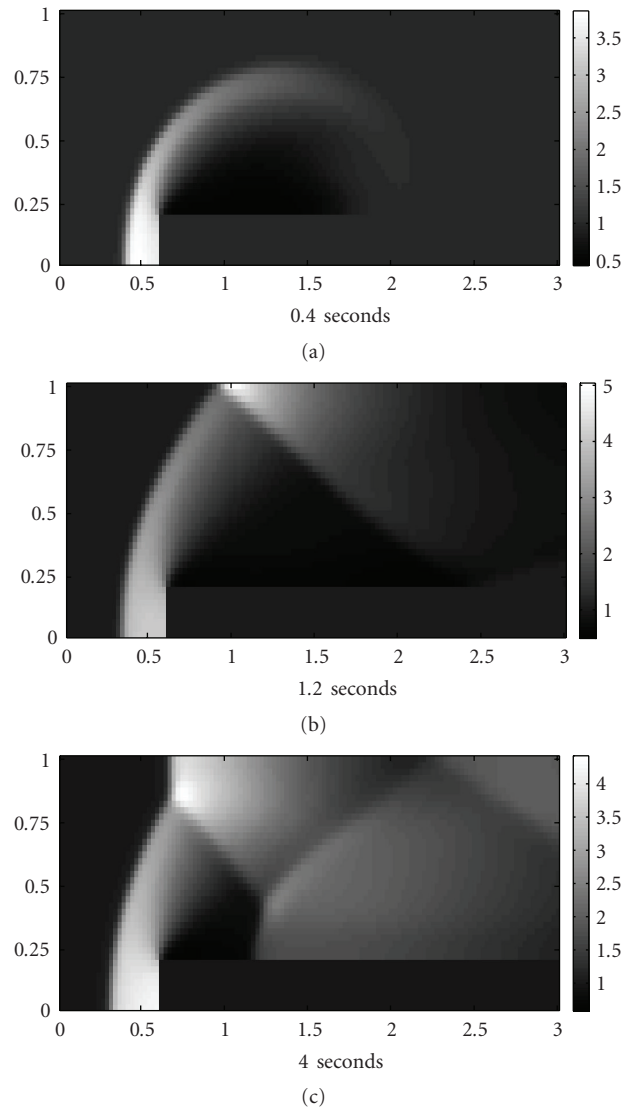

Figure 8: Second-order solution of the Mach 3 flow on an 80×240 array after 0.4, 1.2, and 4 seconds of simulation time.

with the implementation of the multiblock technique when the computational domain is divided into two parts at the forward face of the step.

Reference solution for the previous problem computed by the more accurate residual distribution upwind scheme can be found in [14].

6.3. Performance. Performance of the architecture is determined by the maximum clock frequency and the number of arithmetic units. The huge amount of possible configurations of the arithmetic unit does not enable to carry out postlayout simulations in each case. Therefore, performance data is provided by measuring the maximum performance of the individual functional units. According to the Xilinx data sheets, the floating-point arithmetic cores can run on 350 MHz clock frequency in the case of Virtex-5 FPGAs. Performance of the fixed-point arithmetic cores depends more on the width of the operands, and about 400–550 MHz clock frequency can be achieved. Actual clock frequency of a given configuration can be 0% to 20% smaller according to the utilization of the device and due to changes in placement and routing. Expected performance of the different arithmetic units compared to an Intel Core2Duo microprocessor running on 2 GHz clock frequency is summarized in Figure 9.

The computation of the Mach 3 problem lasts about 2419 seconds on the Core2Duo T7200 microprocessor using first-order approximation while 10591 seconds are required to compute the second-order result. This is equivalent to approximately 1.3 million cell update per second for the first-order method and 0.297 million cell update per second for the second-order approach.

Using 32-bit fixed- and floating-point numbers, all arithmetic units can be implemented on a Virtex-5 SX240T FPGA. On this device, the first-order computation lasts

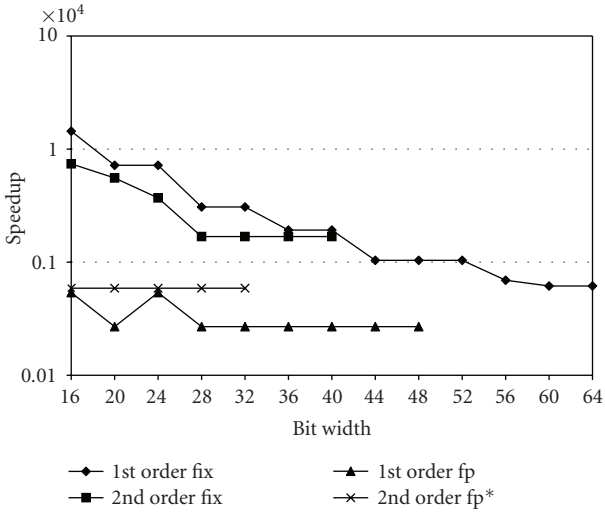

Figure 9: Speedup of the arithmetic unit implemented on Virtex-5 XC5VSX240T FPGA compared to a Core2Duo 2 GHz microprocessor (*half arithmetic unit—two clock cycles per cell).

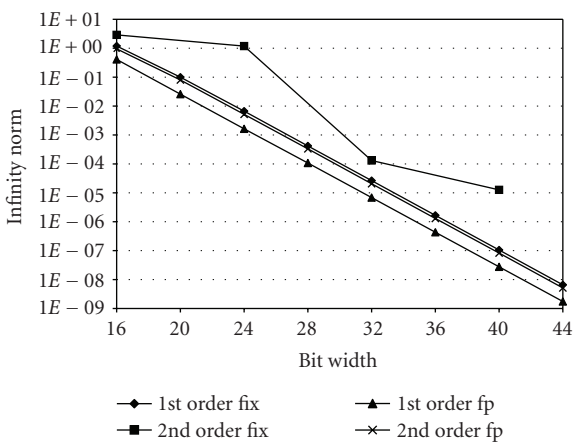

Figure 10: The infinity norm of the solutions.

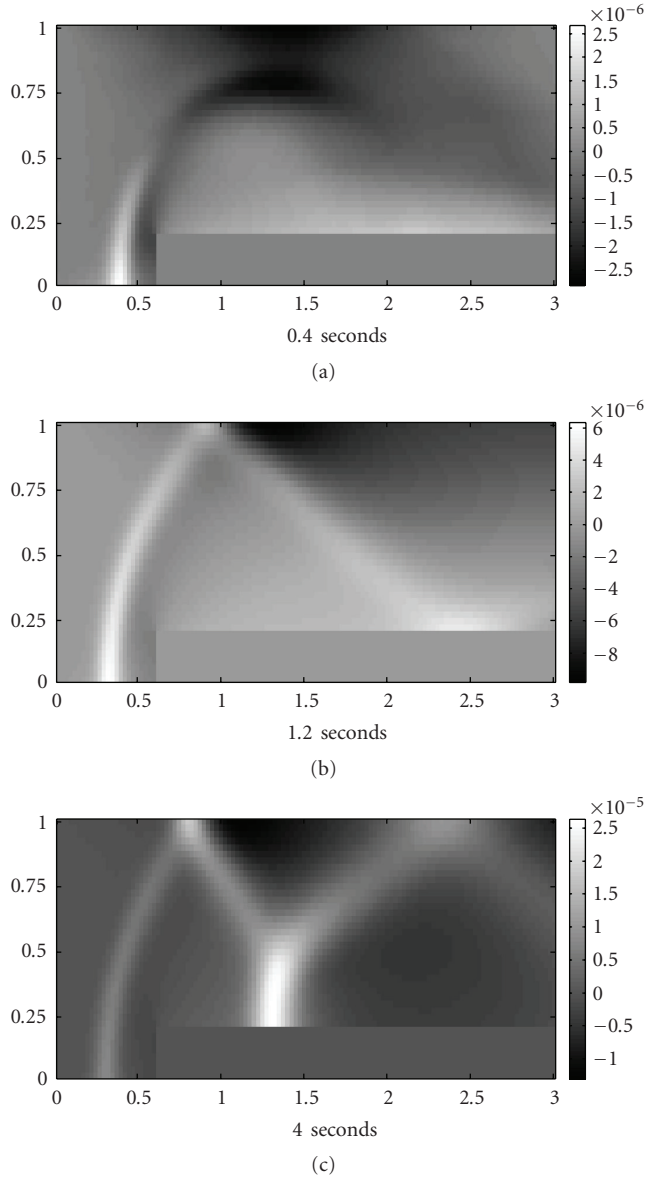

Figure 11: Error distribution of the first-order 32 bit fixed-point solution of the Mach 3 problem after 0.4, 1.2, and 4 seconds of simulation time.

approximately 0.78 second and 8.98 seconds in the fixed- and floating-point cases , respectively, while in the second-order case runtime is increased to 6.29 seconds and 17.97 seconds. The first-order fixed-point arithmetic unit is 11-time faster than its floating-point counterpart and more than 3000-time faster than the Core2Duo microprocessor. In the second-order case, the results are more balanced and the fixed-point arithmetic unit is about 3-time faster than the floating-point arithmetic but its performance is still superior compared to the Core2Duo microprocessor.

Additionally, we tried to use performance data reported in previous works, but fair comparison is hard because different CFD models and discretization schemes are used. Additionally different FPGA architectures are used during the implementations. Smith and Schnore [15] published an FPGA-based CFD solver, but they used 3D model and smaller neighborhood during the computation. Additionally, their architecture was implemented on several FPGAs. In the solution of the Euler equations, they reported 24.6 GFlops sustained performance on four Virtex-II 6000 FPGAs. Sano et al. [16] used 2D systolic array to solve 2D flow problems and reported 11.5 GFlops peak performance on an ALTERA Stratix II FPGA. Sustained performance of our solution using 32-bit fixed-point numbers is 416 and 141 billion fixed-point operations per second in the first- and second-order case, respectively.

6.4. Accuracy of the Solutions. As described in Section 6.1, area requirements of the arithmetic unit can be significantly reduced by decreasing the precision of the state values.

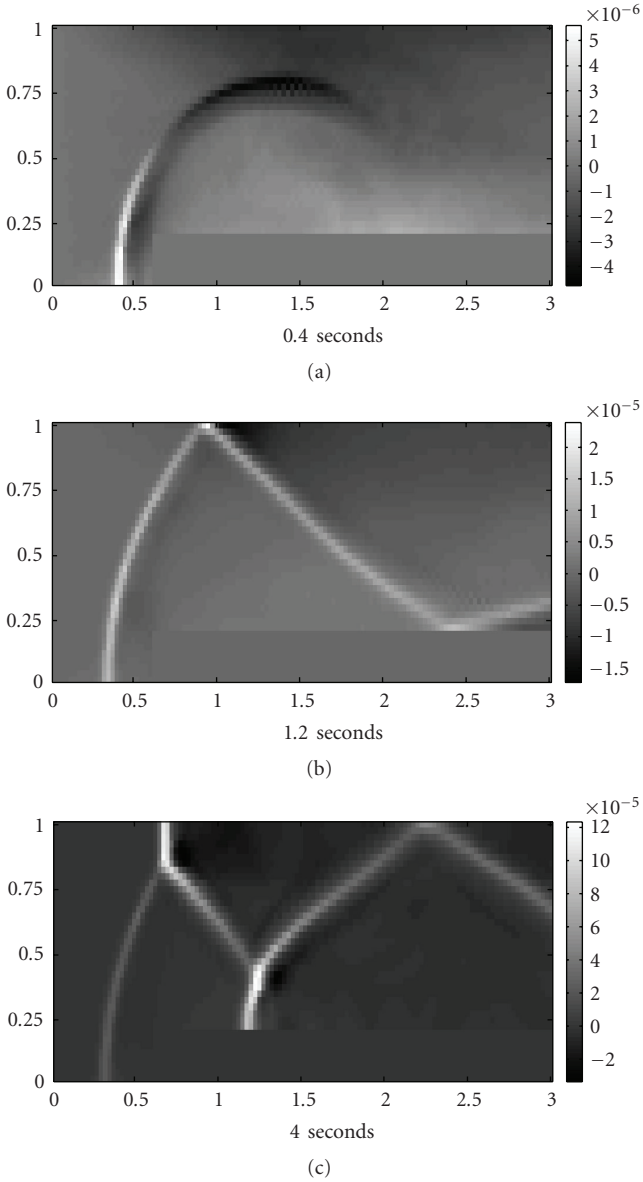

Figure 12: Error distribution of the second-order 32 bit fixed-point solution of the Mach 3 problem after 0.4, 1.2, and 4 seconds of simulation time.

However, smaller precision results in less accurate solution. Unfortunately, the exact solution of the Mach 3 problem does not exist, therefore, the fixed- and customized-precision floating-point results were compared to the 64-bit floating-point result. The accuracy of the solutions was measured by computing the infinity norm which is defined as

$$\|e\|_\infty = \max_i |u_i^A - u_i^E|, \tag{16}$$

where u_i^A is the exact (or in our case the 64-bit) solution, while u_i^E is the numerical approximation using the update scheme with different fixed- and floating-point numbers. The results of the comparison in the case of the Mach 3 problem are shown in Figure 10. Comparing the infinity norm of the solutions to the largest density value ($\rho_{\max}$) in the system, which was in this case about 10, a relative error can be defined as

$$r_{\text{err}} = \frac{\|e\|_\infty}{\rho_{\max}}. \tag{17}$$

The error of the first-order fixed-point solution follows the same trend as the error of the custom width floating-point solution, but the error value in this case is about 4 times higher. The larger error of the solution is balanced by the smaller size and faster operation of the fixed-point arithmetic unit, therefore, it is possible to slightly increase the bit width and compute the results more accurately without loss of the high computing performance.

In the second-order case, the error of the 32-bit fixed-point solution is one-order higher compared to the error of the 32-bit floating-point solution. Increasing the computing precision to 40 bits just slightly increases the accuracy of the solution, and the error compared to the 40-bit floating-point solution is two orders higher. Further investigation is required to find the roots of the different behaviors.

The results, which were calculated applying very low precision (less than 24 bits), are unusable in engineering applications, because the relative error is larger than 10^{-2} in each case. Increasing the precision to 26–36 bits, the relative error of our solution is in the range of 10^{-4}–10^{-6}. These results are accurate enough to use in common engineering applications. Accuracy of the solution can be further increased by using higher precision to represent the state values.

The distribution of the error of the 32-bit fixed-point solutions in the first- and second-order case is presented in Figures 11 and 12, respectively. As it can be seen in these figures in the first-order case the distribution of the error is quite smooth and has a maximum value near the shock waves. In the second-order case, the maximum value of the error is one-order larger and concentrated near the shock waves.

7. Conclusion

The governing equations of the two-dimensional compressible Newtonian flows were solved by using modified emulated digital CNN architecture. The second-order Lax-Friedrichs scheme was used during the solutions. The main advantage of this method over the forward Euler method which is used extensively in the computation of the CNN dynamics is that this approximation is more robust in the case of complex computational geometries and in the presence of shock waves in the solutions.

The arithmetic unit was designed by using both fixed- and floating-point number representations. Interval arithmetic is used to optimally set the precision of the partial results and to reduce the size of the fixed-point arithmetic unit while preserving the accuracy of the solution. The fixed- and floating-point solutions are compared in terms of implementation area, accuracy of the solution, and computing performance.

Implementation area of the arithmetic unit is significantly decreased by the application of fixed-point numbers. The proposed first-order fixed-point arithmetic unit can be implemented on midsized gate arrays. Area requirements of the second-order arithmetic unit are much higher and the currently available largest FPGAs are required for the implementation. The first-order solution using 32 bit fixed-point numbers can be computed 3000 times faster compared to a high-performance microprocessor, while its accuracy is acceptable in engineering applications. The second-order approximation, which models the physical phenomenon more accurately, can be solved 1600 times faster.

In the future, the designed arithmetic unit will be extended to three-dimensional flow problems and nonuniform computational grids could be possible.

References

[1] T. Roska and L.O. Chua, "The CNN universal machine: an analogic array computer," *IEEE Transactions on Circuits and Systems II*, vol. 40, no. 3, pp. 163–173, 1993.

[2] P. Szolgay, G. Vörös, and G. Erőss, "On the applications of the cellular neural network paradigm in mechanical vibrating systems," *IEEE Transactions on Circuits and Systems I*, vol. 40, no. 3, pp. 222–227, 1993.

[3] T. Roska, L. O. Chua, D. Wolf, T. Kozek, R. Tetzlaff, and F. Puffer, "Simulating nonlinear waves and partial differential equations via CNN—part I: basic techniques," *IEEE Transactions on Circuits and Systems I*, vol. 42, no. 10, pp. 807–815, 1995.

[4] Z. Nagy and P. Szolgay, "Numerical solution of a class of PDEs by using emulated digital CNN-UM on FPGAs," in *Proceedings of the 16th European Conference on Circuit Theory and Design (ECCTD '03)*, vol. 2, pp. 181–184, Cracow, Poland, September 2003.

[5] S. Kocsárdi, Z. Nagy, Á. Csík, and P. Szolgay, "Simulation of two-dimensional inviscid, adiabatic, compressible flows on emulated digital CNN-UM," *International Journal of Circuit Theory and Applications*, accepted.

[6] J. D. Anderson Jr., *Computational Fluid Dynamics: The Basics with Applications*, McGraw-Hill, New York, NY, USA, 1995.

[7] T. J. Chung, *Computational Fluid Dynamics*, Cambridge University Press, Cambridge, UK, 2002.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK, 2007.

[9] Z. Nagy and P. Szolgay, "Configurable multilayer CNN-UM emulator on FPGA," *IEEE Transactions on Circuits and Systems I*, vol. 50, no. 6, pp. 774–778, 2003.

[10] Z. Nagy, Z. Vörösházi, and P. Szolgay, "Emulated digital CNN-UM solution of partial differential equations," *International Journal of Circuit Theory and Applications*, vol. 34, no. 4, pp. 445–470, 2006.

[11] O. Aberth, *Introduction to Precise Numerical Methods*, Elsevier, Amsterdam, The Netherlands, 2007.

[12] Xilinx products, 2008, http://www.xilinx.com.

[13] Agility design solutions, 2008, http://www.agilityds.com.

[14] Á. Csík and H. Deconinck, "Space-time residual distribution schemes for hyperbolic conservation laws on unstructured linear finite elements," *International Journal for Numerical Methods in Fluids*, vol. 40, no. 3-4, pp. 573–581, 2002.

[15] W. D. Smith and A. R. Schnore, "Towards an RCC-based accelerator for computational dluid dynamics applications," *Journal of Supercomputing*, vol. 30, no. 3, pp. 239–261, 2004.

[16] K. Sano, T. Iizuka, and S. Yamamoto, "Systolic architecture for computational fluid dynamics on FPGAs," in *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '07)*, pp. 107–116, IEEE Computer Society, Los Alamitos, Calif, USA, April 2007.

Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 930619, 11 pages
doi:10.1155/2009/930619

Research Article

GPU Boosted CNN Simulator Library for Graphical Flow-Based Programmability

Balázs Gergely Soós,[1,2] Ádám Rák,[1] József Veres,[1] and György Cserey[3]

[1] *Faculty of Information Technology, Pázmány Péter Catholic University, Práter u. 50/A, 1083 Budapest, Hungary*
[2] *Computer and Automation Research Institute of the Hungarian Academy of Sciences, Kende u. 13-17, 1111 Budapest, Hungary*
[3] *Infobionic and Neurobiological Plasticity Research Group, Hungarian Academy of Sciences, Pázmány University and Semmelweis University, Práter u. 50/A, 1083 Budapest, Hungary*

Correspondence should be addressed to Balázs Gergely Soós, soos@digitus.itk.ppke.hu

Received 2 October 2008; Revised 13 January 2009; Accepted 12 March 2009

Recommended by Ronald Tetzlaff

A graphical environment for CNN algorithm development is presented. The new generation of graphical cards with many general purpose processing units introduces the massively parallel computing into PC environment. Universal Machine on Flows-(UMF) like notation, highlighting image flows and operations, is a useful tool to describe image processing algorithms. This documentation step can be turned into modeling using our framework backed with MATLAB Simulink and the power of a video card. This latter relatively cheap extension enables a convenient and fast analysis of CNN dynamics and complex algorithms. Comparison with other PC solutions is also presented. For single template execution, our approach yields run times 40x faster than that of the widely used Candy simulator. In the case of simpler algorithms, real-time execution is also possible.

Copyright © 2009 Balázs Gergely Soós et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

Nowadays there are many hardware implementations of the Cellular Neural Network Universal Machine (CNN-UM) [1] that can be divided into three major categories. The first and usually the most powerful is the mixed-mode (analog and digital) VLSI implementation like the Ace16K [2], the SCAMP chip [3], and the eye-RIS chip [4]. They are also referred to as focal plane processors since they have direct optical input. These architectures employ one-to-one mapping between pixels and processors. Vision Systems are created based on them like the Bi-i system [5] or the eye-RIS system [4] designed for industrial purposes. The second is the emulated digital class that splits into two subcategories. The first is the pipelined version such as the FALCON [6] architectures implemented on DSPs or FPGAs while the other is the coarse grained processor array (e.g., Xenon [7]) where n pixels are mapped to m processors (usually n being much higher than m). The third category is the optical implementations like POAC [8] which has the advantage of the massive parallelism. These implementations give researchers a handful of tools for processing two or three dimensional sensory data flows which are received usually from visual or tactile sources. Manipulations of data flows need a special programming environment. Describing these kinds of algorithms can be effectively done by using a high-level Universal Machine on Flows (UMF) [9] flow-chart model and its UMF diagrams.

The advantage of our simulator system is that it combines functionality with clarity of UMF diagrams for description, since these graphically represented data flow algorithms can be run directly without the need for further coding. Furthermore, these diagrams can easily be drawn only using a simple drag-and-drop technique. This feature is very helpful in algorithmic development by speeding up the drawing process. The modeling environment gives fast test results before optimizing the algorithm for any specific hardware.

Our simulator environment is embedded into MATLAB Simulink from MathWorks Inc. [10]. The aim was to exploit its simulation ability for continuous time dynamic systems and utilize its construction method of simple additional

Blocks (basic functional units). For referring to this concept, capital letter is used to distinguish it from the one appearing in Section 3. We have created a high level Blockset for the CNN-UM programming structure including easy access to the linear-type, one-layer subset of the CNN template library [13]. The first version of our Blockset used the built-in partial differential equation solver of Simulink. Despite of its relatively slow speed it can be used efficiently for educational purposes.

By deploying the execution of the most computation expensive part to a fast external hardware component, we obtain a powerful development environment that combines the relatively high simulation speed with the advantages of convenient UMF modeling. The mostly data parallel structure of CNN tasks can be exploited by using hardware accelerators, either multicored CPUs or the Graphical Processing Units (GPUs) of graphical cards. We have chosen the GPU for our purposes, because these extremely fast developing systems have high performance and it is easily incorporated into common PCs. There are other research groups working on the development of the GPU powered acceleration of CNN template running [11], which underlines its significance.

The structure of this paper is as follows: after the introduction, Section 2 describes our graphical programming interface, SimCNN in detail. Section 3 gives an overview of the architecture of the recent generation of NVIDIA graphics cards that clarifies the reasons for our choice. Section 4 describes the hardware mapping of the dynamic CNN equations. In Section 5 a complex algorithmic example is presented. Finally, performance comparison with other simulating platforms is given.

Since the first publication [12], our framework has been improved in two regards: (i) we have added a special function for uncoupled templates, (ii) and improved the interfacing Blocks for data transfer between the memory units of the PC and that of the GPU. Thus the possible iteration number that can be calculated for a single template with 25 frames per second has increased from 80 to 90 with the same hardware configuration.

2. SimCNN and Its Graphical Flow-Based Programmability

2.1. The Software Framework. The simulator is based on Simulink from MathWorks Inc., which is a widely used software that has support for nearly any operating systems. Graphical flow-based programmability has many advantages compared to the standard imperative command-based ways. Algorithms are mapped to transformations realized by *Blocks* and links between them. *Source Blocks* are emitting *signals* that are transferred by the links, processed by numerous transformations, and finally stored or displayed by *sinks*. Blocks can handle multiple inputs or outputs connected to their *ports*. Simulink has built-in Blocks for realizing specific data flow structures, like parallel or conditional execution and signal routing. Simulation can be done for models with continuous states using differential updates, integrator Blocks, and feedbacks. Alternatively, discrete time models can be designed with memory and delay elements based on larger fix time-steps. Built-in solvers exist for both approaches.

In the first version of our SimCNN blockset we used built-in continuous time solvers. In this way multilayer CNNs can be covered, and the dynamics of the system can be visualized easily. In order to increase simulation speed we decided to use hardware acceleration. The current model works as follows: discrete time simulation is used in the top level and all templates are executed atomically, while the dynamic of a single template is handled by our algorithm running on the video card. After the evolution is calculated, the result is passed onto the connecting transformations, and the Simulink scheduler selects the new Block to evaluate. This method is closer to real CNN-UM hardware realizations.

In the SimCNN Blockset, we defined necessary extensions for accessing the video card and to run templates on it (Figure 1). We also created a small user interface to template Blocks for browsing the linear one-layered subset of the template library. However, custom templates are also supported. The Blocks are implemented using the s-function extension interface. The *Video and Image Processing Blockset* covers common image processing tasks although in our case only I/O Blocks are used since algorithmic tasks are solved by CNN templates.

In all time instances, signals are scalars or matrices in the case of images. The memory space of the video card and the *host* computer are separate. The core algorithm of Simulink containing the scheduling and execution of Blocks and data transfer are running on the CPU. Template execution has a layer of code running on the CPU, invoking the layer running on the GPU. Once an image is uploaded to the memory of the device using the ≪Upload to GPU≫ Block, we can refer to it using a real value as ID, similar to pointers used in C language. Image data are handled on the video card, and only the IDs are transferred by the host from one template Block to another. The final results are loaded back to the CPU using the ≪Download from GPU≫ Block. For the first execution time instance, the structure is initialized using the parameters of the surrounding Blocks to determine image dimensions and memory requirements. After the initialization phase, template execution will be performed. Simulink calls event handler functions of each Block one by one in a good serialized order.

2.2. UMF Description. The UMF library [13] contains the basic image processing algorithms and numerous powerful ones like the CenterPointDetector or the GlobalConnectivityDetection. This library is an excellent starting point for developers to create their own programs. In the next paragraphs, mapping between UMF notations and Simulink elements marked with ≪symbol≫ are given for most important functionalities. Our aim was to implement the most important subset of the library to cover algorithms using one-layer linear CNN templates.

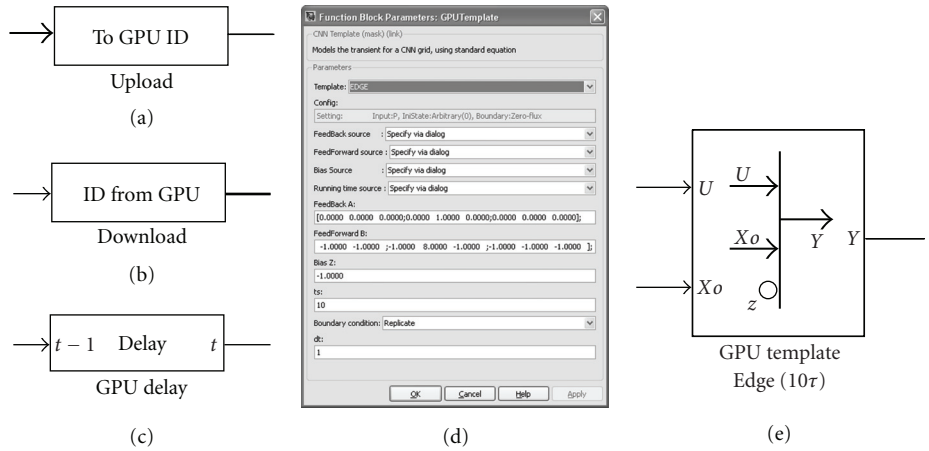

Figure 1: Basic Blockset of SimCNN. Interfacing Blocks (a, b) are required to transfer data to the graphical hardware. The dynamics of each template are calculated on the hardware in one atomic step. Template values, boundary condition, and simulation parameters like time step and running time can be set using a dialog box (d) for each template Block (e). A subset of the template library is also included and can be selected from a drop-down box. A special Block is also added with null transform to delay image flow with one step (c). The Blocks are implemented using C language.

(1) *Elementary instruction*: is the basic template execution that is realized by the Block ≪GPU Template≫. The parameters can be set using the user interface. The Block displays the execution time and also the name of the template if it is from the library otherwise labels it as "USER DEFINED."

(2) *Signals, variables*: simulink supports naming of signals directly. Variables can be created by the ≪GPU Delay≫ Block.

(3) *Boundary conditions*: can be set on the user interface of the corresponding template.

(4) *Continuous delay*: is out of scope since simulation is in discrete time.

(5) *Step delay*: can be realized with ≪GPU Delay≫, or they can be joined together to form longer delay line.

(6) *Parametric templates*: on the user interface of the templates external input can be selected for external parameters.

(7) *Algorithmic structures*: for creating the data path, basic Simulink knowledge is needed. Fortunately MATLAB Help is rather detailed and has good tutorials for learning. To realize "Triggers" ≪Unit Delay Enabled≫ Blocks can be applied. The input flow is blocked if the condition is not satisfied. Complex flow structures like conditional loops can be joined together using ≪MUX≫ (multiplexer) or ≪Switch≫ Blocks.

(8) *Decisions*: decisions on scalar values can be done using ≪Logic and Bit Operations≫, and scalars can be derived from images (like average or min, max) using the ≪ Math Operations≫ Blockset after downloading the image to the host.

(9) *Operators*: various operators are supported by Simulink but only for data in the host memory space. However, implementation of most important global operators can be expected in the near future.

(10) *Subroutines*: are element of the Simulink environment as well.

(11) *Triggers, Cycles*: ≪Enabled Subsystems≫ combined with ≪If≫ or ≪Switch≫ Blocks can be used for UMF-like branching, or ≪Iterator Subsystem≫ can be used for a slightly more readable notation.

In the following example the Center Point Detection algorithm from the UMF library is presented. This is an iterative method to detect mass center of objects in a binary image using series of erosions. Objects are shrinking around their perimeter one pixel in a specific direction in each step. Directions are repeated in clockwise order to slim patches symmetrically. The final result is a single pixel close to the original mass center. Figure 2 shows both UMF description and SimCNN model, and the evolution of the image. In Simulink a ≪Unit Delay≫ Block is needed to avoid direct loops. The ≪Switch≫ Block selects input image or the result from a previous iteration for further processing. For the used morphological templates 2 tau running time with 1 tau time-step is reliable for convergence (DT CNN model).

Figure 3 shows a second example. A basic algorithm was designed to help counting worms in blood samples. Video flows were captured by a camera mounted to a microscope. The specimens are not thin enough to fit into the limited depth of field yielding partially blurred images. Blood particles are relatively dark blobs, haemolyzed particles are giving salt-and-pepper like noise factor. Worms are elongated dark structures. With thresholding, a binary image can be extracted covering the whole worms but also the blood particles. The parts of the worms in focus are black thus a stronger threshold value can be used to select them. The remaining tiny particles can be removed by a Small Object Killer template. A Recall wave can now be started from the

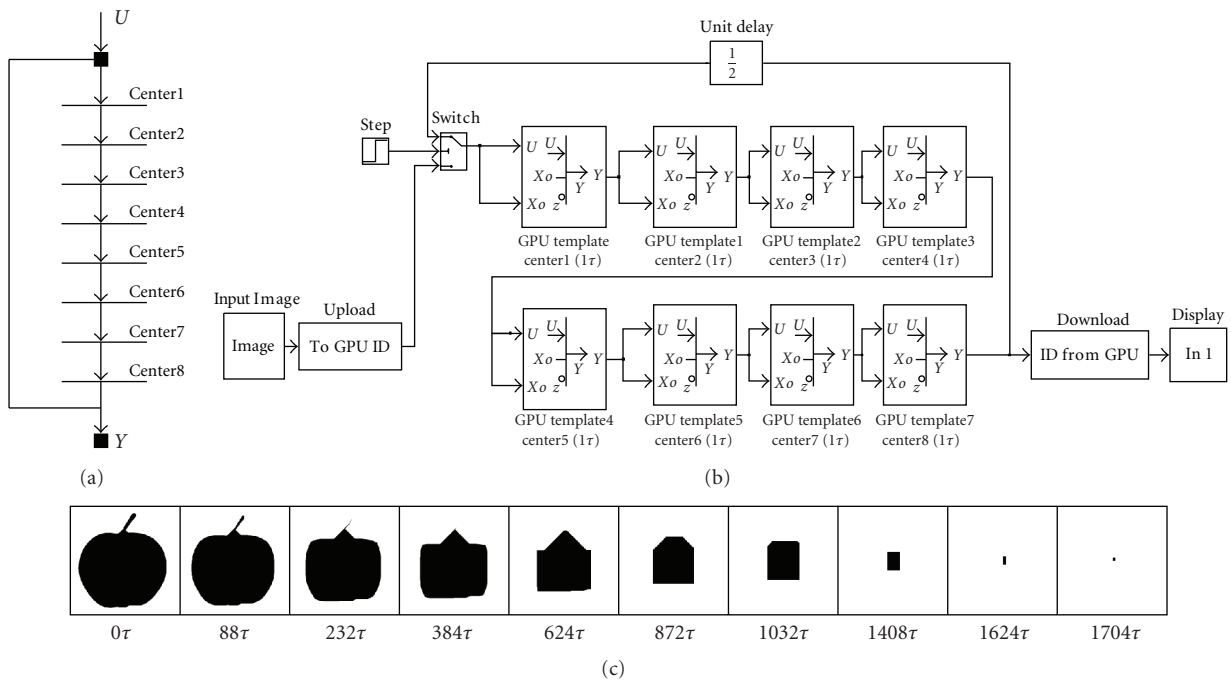

Figure 2: Center Point Detection algorithm. The UMF model (a) and the corresponding SimCNN representation (b) are given alongside the result for a 512 × 512512 × 512 image of an apple (c). The algorithm consists of iterating a sequence of eight erosion templates for all eight direction. The SimCNN model comprises the ≪InputImage≫ and ≪Display≫ Blocks for reading the image file and displaying the result, the ≪Upload≫ and ≪Download≫ Blocks to transfer images to and from the graphic card, and the eight ≪GPUTemplate≫ Blocks. To realize the feedback a ≪Switch≫ Block can be used to select from the original or the updated image. The ≪Step≫ Block provides control signal for the first time instances to send the input into the loop.

cleaned mask to reconstruct worm bodies. The algorithm run video real time.

2.3. Template Running: The Approximate Equation of the CNN. CNN dynamics are encapsulated in subroutines running mostly on the graphical hardware.

Let us consider the standard CNN configuration with space invariant templates, connection radius r, cloning templates A_{pq}, B_{pq} where $p, q \in \{-r, \ldots, r\}$, bias map z_{ij} and output characteristic function f [14]

$$\frac{dx_{ij}}{dt} = -x_{ij} + \hat{A}_{ij}(y) + \hat{B}_{ij}(u) + z_{ij}, \tag{1}$$

$$\hat{A}_{ij}(y) = \sum_{p,q\in\{-r,\ldots,r\}} A_{pq} * y_{i+p,j+q}(t), \tag{2}$$

$$\hat{B}_{ij}(u) = \sum_{p,q\in\{-r,\ldots,r\}} B_{pq} * u_{i+p,j+q}(t), \tag{3}$$

$$y_{ij} = f\left(x_{ij}\right), \tag{4}$$

$$f(a) = \frac{1}{2}(|a+1| - |a-1|). \tag{5}$$

The differential equation can be solved from a starting point x_{ij}^0 iteratively with Euler formula, (1) turns to be (6). More sophisticated solvers such as the Runge-Kutta method could also be used. Discussion will be presented in Section 4.2.

$$x_{ij}(t+dt) = x_{ij}(t) + dt * x'_{ij}. \tag{6}$$

As the input picture does not change over the template execution time, a cumulated bias map W_{ij} can be precalculated. $dt * A_{pq}$ can also be calculated and stored to speed up calculation of the feedback part V_{ij}:

$$x_{ij}(t+dt) = (1-dt) * x_{ij}(t) + V_{ij}(x) + W_{ij}, \tag{7}$$

$$V_{ij}(x) = \sum_{p,q\in\{-r,..,r\}} \left[dt * A_{pq}\right] * f\left[x_{i+p,j+q}(t)\right], \tag{8}$$

$$W_{ij} = \left[\hat{B}_{ij}(u) + z_{ij}\right] * dt. \tag{9}$$

Altogether two 2D data sets: state matrix x_{ij} and W_{ij} are needed together with the constant template values to calculate the state update for the next iteration. The template execution can be simulated with adequately small time step for n iterations to achieve steady state (in most cases) or some specific states for templates like heat-diffusion.

3. GPU as a Hardware Accelerator

3.1. CPU versus GPU Comparison. Nowadays more than a billion of transistors [15] could be found on a single digital chip. This is an outstanding opportunity for creating not only

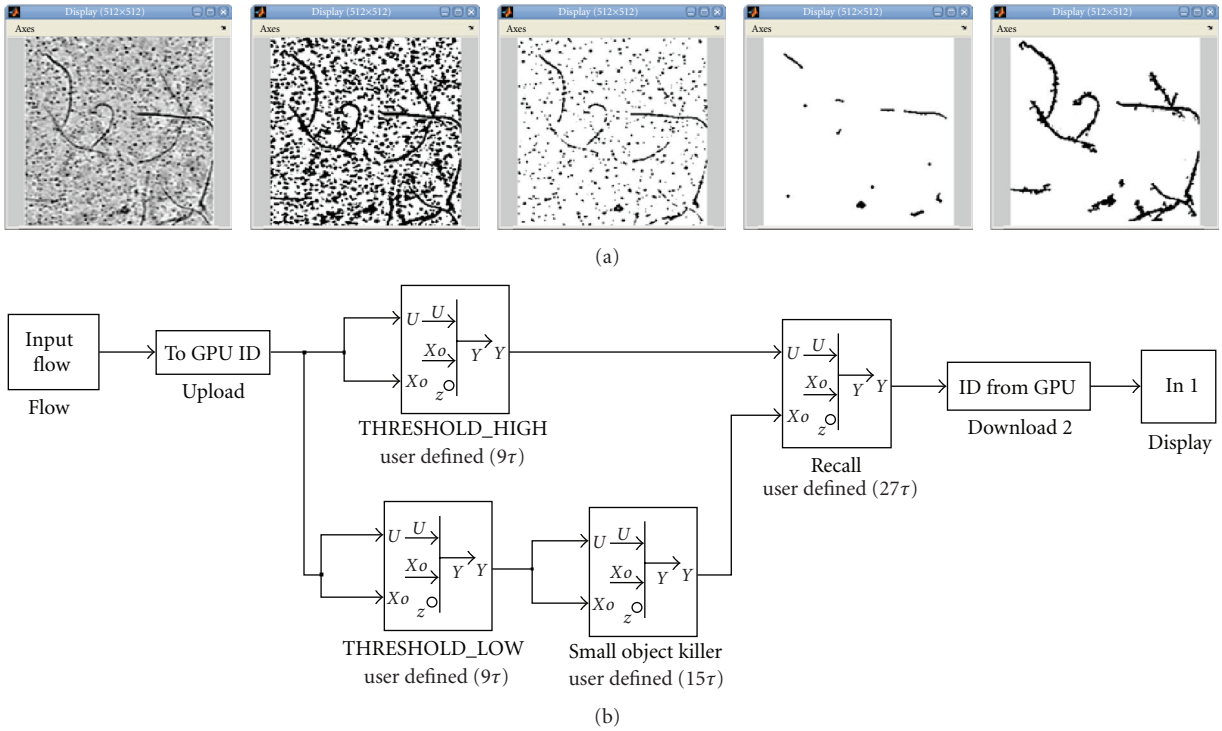

Figure 3: Worm detection algorithm, a realtime example. This figure shows the SimCNN model for the algorithm. It contains the UMF-like diagram flow chart of a sample algorithm (b) that can be run video real-time. This is a worm detection application, for extracting elongated structures from microscope images of blood samples. The upper first image on the upper panel (a) displays the input followed by the two thresholded intermediates, the one after running the Small Object Killer template and finally, the reconstructed image using Recall template that is the result of the algorithm. ≪Upload≫ and ≪Download≫ Blocks are used for interfacing the parts running on CPU and GPU, while templates are running mostly on GPU, represented by ≪GPUTemplate≫ Blocks.

single but dual, quad or many cores on a single die. This multicore tendency, however, had a higher impact on the Graphics Processing Units than on the Central Processing Units of PCs. This is confirmed by the fact that nowadays GPUs with 128 processors are widely available, but only dual- or quad-core CPUs are common on the market. Owens et al. show in their survey [16] that more than five times greater computational power can be achieved compared to nowadays' CPU used worldwide in personal computers. The main significant difference is the amount of transistors dedicated to the data processing elements versus the amount used in local cache circuits. The high percentage of the transistors in CPUs is responsible for flow control and for caching. The GPU needs more effort from the programmer to design efficient code with explicit care of data transfers but offers more computational power on the other hand.

If consider other cheap, compact solutions for scientific calculation than PCs, another possible solution is Sony PlayStation III based on Cell processor technology from the STI alliance (Sony, Toshiba, and IBM). This can be used with Linux operating system but the small amount of memory embarasses the use of graphical environments so this cannot replace a PC in everyday use. Cell is a very powerful digital processor but for some applications, GF8800GT overperforms it. While the Cell Broadband Engine Architecture with 8 Synergistic Processing Elements (SPEs) clocked at 3.2 Ghz is capable of theoretical 208.4 Gflops [17], the 8800 GTX with 128 stream processors clocked at 1.35 Ghz reaches 520 Gflops in peaks (above 300 Gflops in practice) [18] for nearly the same price.

Therefore, we can state that today's GPUs with their high level of parallelism are cost-efficient processors for performing the power resource extensive task of CNN simulator, namely, template running.

3.2. Compute Unified Device Architecture (CUDA): NVIDIA's Newest Architecture. In order to realize a nongraphic operation like template running on the graphics card, there must be a software and hardware solution to access the high computational power of the GPU. This need is fulfilled by NVIDIA's newest development, the Compute Unified Device Architecture (CUDA). This novel approach code can be built in C-like language without any knowledge about graphical programming like DirectX or OpenGL. It is available not only on the high-end category, but also on the whole NVIDIA's 8 series cards. The gaming market is huge, driving production in enormous quantities compared to any scientific equipment, making it available for a wide audience in the near future to benefit from its advantages for a low price. Before the technical implementation, we highlight the major hardware parts of the GPU's architecture.

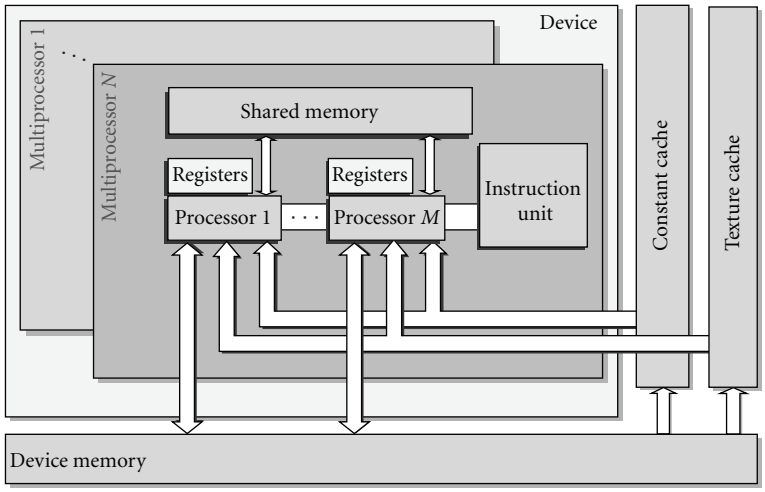

Figure 4: Hardware model of the GeForce 8 series cards [19]. It is a new unified hardware architecture with multiprocessors (MPs) that have dedicated shared memory accessed by a few scalar-based processors replacing the separate vertex and pixel shaders. Processors work with their own registers and are driven by a common Instruction Unit forming a single instruction multiple data architecture. Algorithms can run on multiple MPs, although communication between MPs via the Device Memory is relatively slow. Data can be loaded from the read-only Constant and Texture Cache as well.

In Figure 4 the hardware model of this architecture is described briefly. One can realize that dedicated pixel and vertex shaders were replaced with unified, scalar-based processors. Eight of these processors are grouped with an Instruction Unit to form a single-instruction-multiple-data (SIMD) multiprocessor (MP). The number of MPs varies from 1 to 16, yielding 8 to 128 parallel processors. Global memories are much faster than those in use on mainstream motherboards nevertheless, these would be still too slow for supplying the data for all processors. Therefore, three ways for caching were introduced. A 16 KB per MP high-speed universal purpose shared memory is available beside the constant (1D) and texture (2D) caching memory. I/O instructions accessing device memory through cache converges to 1-2 cycles while direct access costs 100–200 clock cycles.

CUDA supports implementing parallel algorithms with topological processing, running the same code segment on all nodes, which is similar to the CNN architecture. Since the number of the physical computing elements (processors) is typically less than the number of nodes for one-to-one mapping, an automatic serialization is applied. All processors run a thread at a given time, which is its execution context. The atomic algorithm part covering an execution of a subroutine on the nodes is called kernel. Groups of topologically associated threads called blocks (notation with small starting letter) are also arranged in a *grid* topology (Figure 5). The low-level cluster is especially important because all threads in a given block are executed on the same MP, so they have a common high-speed, low-latency (1-2 clock cycles) shared memory that can be used for creating local interconnection. It is tempting to group all nodes to a single block in order to form a fully connected network, a block, however, can encapsulate maximum 256 threads. Moreover, all threads need space for storing their data, and the amount of memory available for a block is only 8 Kbytes, therefore a hard limitation is encountered. Since there are only eight processing elements in a Multiprocessor unit, when large number of threads are associated, frequent context switching also slows down execution [19].

The communication between the threads can be realized only through memory transfers. Synchronization is needed in order to preserve checkpoints to avoid inconsistencies in communication. We have to minimize data exchange between threads that are mapped to distinct blocks, because the only way to do that is accessing the high-latency (100–200 clock cycles) global memory. Since there is no support for the synchronization outside the blocks, the whole kernel should be finished to be sure that all the threads are ready. This means splitting algorithms into smaller kernels. Programs in the CUDA framework consist of kernels compiled into binary code, executed on the graphic card and controlling code running on the host computer.

4. Mapping

4.1. Input/Output Requirements. The actual topological problem is the CNN dynamics equation. CNN cells are arranged in topological grid. Similar configuration can be created from threads. Analysing the state equations, one could see that this problem is a memory intensive one. To calculate the state update for a pixel, the constants for the template values, the nine state variables from the neighbourhood, and the nine values from the cumulated bias map (x_{ij}, W_{ij}) are needed. The memory fetch from global memory can cost 100–200 machine cycles compared to 1-2 cycles needed for arithmetic calculations.

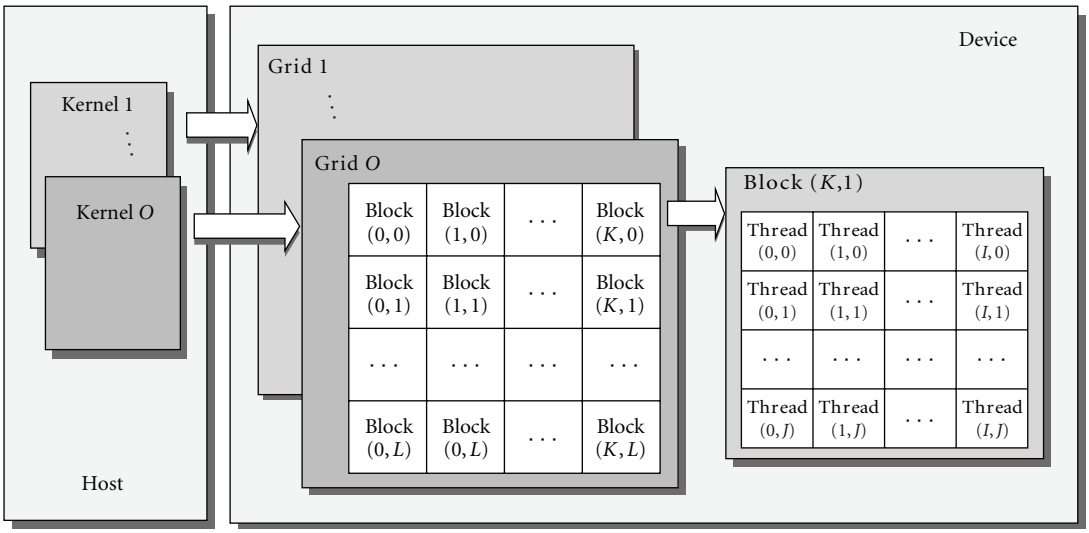

Figure 5: Software running in CUDA [19]. Programs are coordinated by the host computers. Functions that can be executed on GPUs are special data-parallel multithreaded segments called kernels. A 2-level topology (1D/2D/3D), or Grid configuration is assigned to threads. At low level threads are organized into blocks. This grouping is relevant because they mapped to a common MP. Threads run the same code but different data can be referred using index parameters.

If all pixels are accessing their nine neighbors separately, that will not be efficient. The 2D aware texture cache can be applied to assist joint prefetch, but the cache hit rate is less than 100%. Unnecessary redundant memory access can be eliminated with direct copying of all referred states for threads in a block into the shared memory. The small amount of constant values can be fit into the constant cache. After the reading commands, a checkpoint can be placed for synchronization before starting the calculation.

4.2. Tiling. Rectangular parts of the image can be assigned to a block and calculated by threads running on the same Multiprocessor, so they can synchronize with each other. After the state update, blocks should write results to global memory to be accessible for neighbours and for the host computer. To calculate derivatives, neighbouring values are also needed, thus overlapped tiling is necessary when covering the full image. The overlapping regions are read from both tiles but only the nonoverlapped regions are updated.

For the Euler method only the r direct neighbours are needed for a state update step, while for 4th-order Runge-Kutta (RK4) to calculate immediate substeps $4 \times r$ cells are needed to be read from x_{ij} image in each direction along the diameter. Mapping squared regions offer the best area perimeter ratio.

Considering 16×16 pixel blocks and a template with $r = 1$ neighborhood, for Euler neighbor constraint rises $(17 \times 17 - 16 \times 16) = 33$ read overhead compared to 256 updated cell (13 percent), while for RK4 $(20 \times 20 - 16 \times 16) = 144$ extraread is needed (56 percent).

The first version of our system relies on the built in solvers of Simulink so we could test both methods. The linear subset of the templates that can be used in present hardware implementations consists of robust ones that converge relatively fast even with large (0.5 tau, 0.6 tau) steps or even 1 tau for binary templates. The RK method gives more accurate solution, converges faster, and a larger time-step can be used [20]. Although considering the penalty for extra memory access, the advantage can be taken only for more complicated dynamics like polynomial or multilayered CNNs. The aim of this work was to give a tool for algorithmic testing with many templates rather then to write a new high precision solver. The Euler method and (7) will be used to estimate the dynamics.

4.3. Grid Configuration. The CUDA gives support for four element vectors beside the standard data types. If a thread is responsible for more pixels, the outputs can be packed using vector representations. On the other hand, simple algorithms—minimizing the branching—are needed for efficient parallel execution. Therefore, four pixels from a row are associated to a thread. To minimize read overhead around perimeters, squared image parts are required. Considering the maximal 256 threads allowed in a block, 4 threads in a row, and 16 rows configuration ($4 \times 16 = 64$ threads) is used for handling tiles with 16×16 pixel efficient regions.

4.4. Data Format. In Simulink the preferred data format is *double* (double precision floating point represented on 64 bit) especially for ≪Video Display≫ Block in $[0, 1]$ interval for images. To store data on the graphic card, *short* (16 bit integer) format was selected. The signal range of CNN $[-1, 1]$ was mapped into $[-2048 \cdots 2047]$, allowing $[-8 \cdots 8]$ value to be represented in a short value assigned to the state variable of a pixel, which is large enough to cover cell dynamics and also a compact form to keep memory consumption low. The arithmetic calculations are

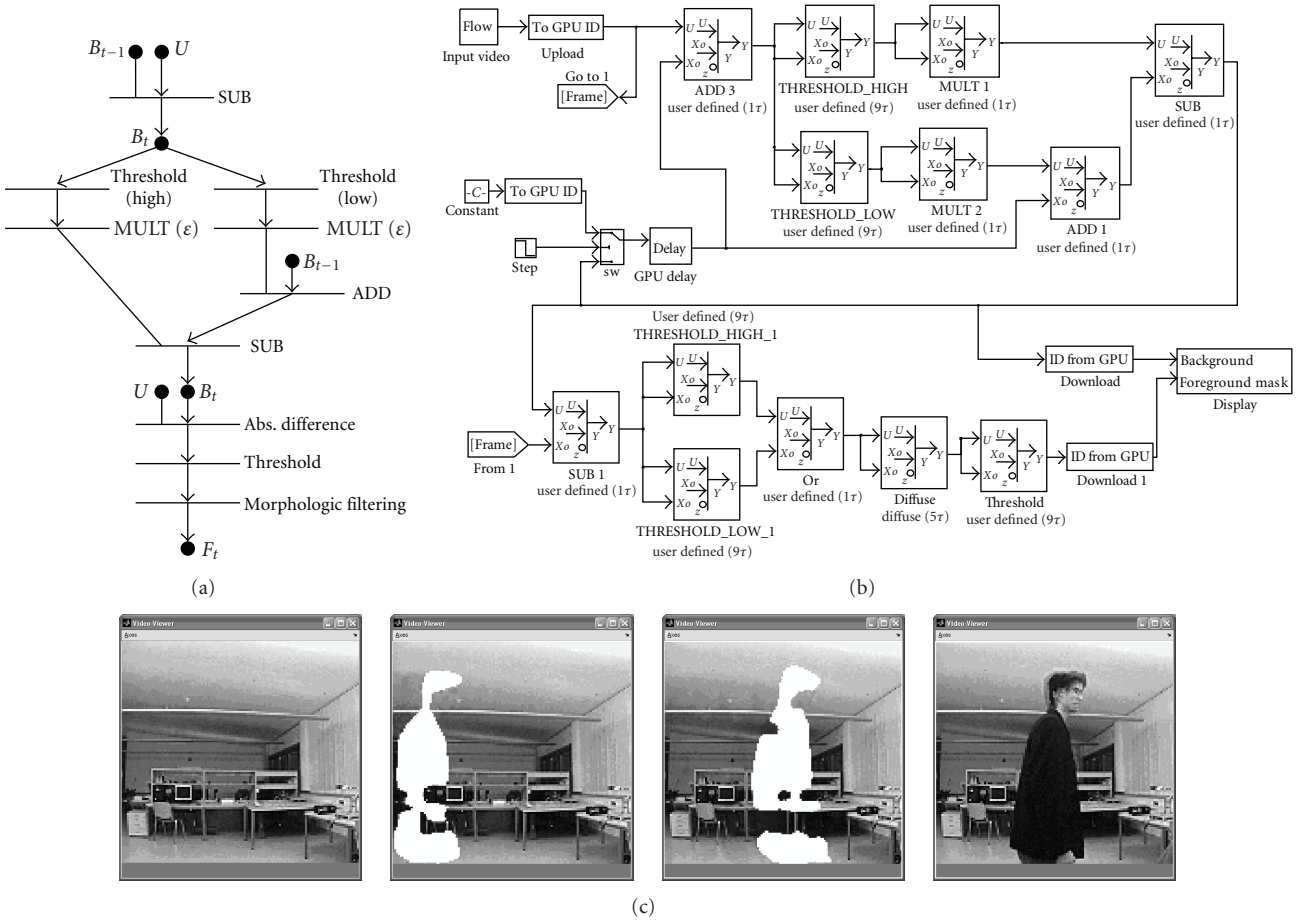

Figure 6: Implementation of a complex problem. UMF diagram of the Adaptive Background and Foreground Estimation subroutine (a) is shown with the corresponding SimCNN model (b). The algorithm can process input flow captured with a web camera on the fly. The static part of the input flow is estimated in a feedback loop (Background). Parts with significant difference in gray level are considered to be moving objects (Foreground). Abs. differences and Threshold are implemented using two Thresholds ≪GPUTemplates≫ and Morphological Filtering with a Diffusion and Threshold. During the experiment a person is coming into the screen from the left and stops in the middle. Characteristic screenshots are shown on the upper panel (c). Foreground parts are overlapped to the input flow.

done in *float* (single precision floating point represented on 32 bit) to fit hardware capabilities of the GPU. Format conversion is done automatically during memory access. Range conversion is handled by the interfacing Simulink Blocks implemented by kernels running on GPU.

The execution of a template has two stages: calculating the cumulated bias map (W_{ij}) and iterating the state update kernels. Between state updates handling of the boundary condition is also needed. Three appropriate kernels were created with a controlling host function. This function is invoked by the ≪GPUTemplate≫ block.

The main objective was to handle coupled templates. Uncoupled templates for thresholding, logic operations, and for addition are also frequently used in algorithms, so a special kernel with embedding host function was also designed. This kernel can calculate the whole dynamic of a cell since no synchronization is needed between blocks. Using this method gives moderate speed-up, but this is not extremely remarkable since templates from this class need only a small number of iterations to converge (about 10) with small execution time compared to the overhead of executing a Block.

5. Complex Algorithm

To demonstrate the possibility of handling complex algorithm in SimCNN the implementation of the Adaptive Background and Foreground Estimation subroutine from the UMF library is presented (Figure 6). The UMF diagram refers to two input variables, namely, U for the actual input image, and B_i for the actual background image. The background model is used from the previous time instant (B_{i-1}) that implies a feedback in the processing and a Delay element. Starting value is needed (e.g., 0) in the SimCNN model for initializing B_0. Image parts that are significantly brighter in the input than the model are increased, significantly darker regions are decreased in the model image. In case of static scene the model converges.

Table 1: Comparison of test results with other implementations.

	T7200	CELL	**8800GT**	Candy	XC3S5000
Frequency (MHz)	2000	3200	**1350**	2130	150
No. of Processors	2	8	**120**	1(2)	34
M cell iteration	48	3627	**590**	15	1700
Power (W)	34	86	**160**	65	10

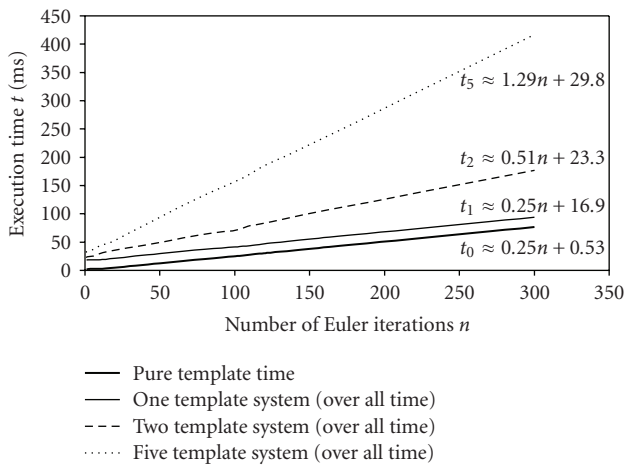

Figure 7: Execution speed of a series of template each running for n iterations. The measured value for a single template running influenced by all necessary interface Blocks is displayed with straight thin line (t_1) and its theoretical maximum without overhead with straight bold line (t_0). Results for two and five serially joined templates with overhead are indicated by striped (t_2) and dotted line (t_5) , respectively.

The updated model is compared to the image again in the detection part of the algorithm. Abs. Difference and Thresholding are implemented with multiple Threshold Blocks. Diffuse and a further Threshold Blocks are used to form connected foreground mask. This result is similar in functionality to a series of morphological operators but it is more effective in SimCNN due to the lover number of Blocks.

In the experiments the camera was inspecting our laboratory. Person entering the static scene is detected while he is moving. After he stops the background model is shortly updated. Small changes are not highlighted, like the slowly turning head. Sensitivity can be tuned in the ≪THRESHOLD_LOW_1≫ and ≪THRESHOLD_HIGH_1≫ modules, the learning speed for the model with the multiplication factors.

6. Experimental Results

The simulation platform for the SimCNN toolkit was an NVIDIA GeForce 8800 GT GPU connected to an Intel Core-2 2 × 2.13 GHz CPU (E6400) via PCI Express 16x. Our applications were tested both in Linux and Windows environments with NVIDIA driver version 169.07 (32bit) and CUDA Tool kit 1.1. The embedding MATLAB version was 2007 b with Simulink 7.0.

Experiments were conducted on 512 × 512 sized images. The simulation steps were as follows: image loading from hard drive, Upload to GPU, a series of template operation on GPU, Download from GPU, visualization, and frame rate estimation. The built-in profiler in Simulink reliably measures in the millisecond range, so for fast running Blocks it is quite unreliable, but the averaged measurement for large number of independent template executions can be used (1000 inputs). It converges to 0.25 milliseconds for a single Euler-iteration. This would allow for 4000 Euler-iterations per second (ips), but the execution of the interface Blocks, and the embedding layer to Simulink environment present significant additional load. In addition, Simulink also introduces small overhead for joining the Blocks (i.e., communication). The turn around speed for evaluating a single input image of this minialgorithm was also recorded (in frames per second-fps) with the ≪Frame Rate Display≫ Block in the function of Euler-iteration numbers. It is an overall speed indicator that includes the total overhead. For comparison we evaluated networks with a single template, two templates and five templates joined in a line (Figure 7).

Since the first report of our work small improvement has been achieved in the interface Blocks. As a result, real-time processing (25 input images in a second) can be achieved with up to 90 iterations for a frame together with displaying the outputs (590 mega cell iteration per second).

Furthermore, we have compared the results to other simulator systems. For educational purposes, one of the most common tools is Candy from Analogic Computers Inc. [21]. It also offers an algorithmic framework with the template iteration time of 17.5 milliseconds for a 512×512 sized image running on a 2.13 Ghz Intel Core2 Duo (E6400 with 2 MB L2 cache) computer with 14.98 million cell iteration per second. Since this software does not use multithread computing there was no significant effect of the multicore architecture. Measurement for 1.86 Ghz Intel Pentium 4 (M750 with 2 MB L2 cache) gave 12.48 million cell iteration per second. Taken everything together, almost 40x speed-up has been reached.

A comprehensive comparison can be found in [22] for the actual CNN simulators and emulators that we extended with the result of our measurements (Table 1). The first column shows characteristic values for Intel Core Duo (T7200 with 2 MB L2 cache) using the highly optimized Intel Performance Library [23] to implement convolution. The next column stands for the Cell Broadband Engine hosted by a PlayStation III. The last column shows results for an FPGA board with Xillinx Spartan3 chip (XC3S5000). Cell processor gives the best performance but integration into our desktop computer is not yet supported. In the Cell and FPGA setups a variation of the Falcon architecture was used based

on the Euler method. We can conclude that our solution is twelve times faster than that of the CPU of cutting-edge PCs.

The worm detection algorithm that was presented in Section 2.2 consists of four templates, namely, two Threshold templates, a Small object killer, and a Recall template. They converge in 9 tau, 15 tau, and 27 tau, respectively, with 1 tau step size. The complete algorithm consumes 60 iterations for each input image, thus with about 25 milliseconds of total overhead it runs with 25 fps.

The complex algorithmic example presented in Section 5 can run with 5 fps. The online capturing and the large number of connected templates give significant overhead. The processing speed is low but still enough to catch moving people in a live demonstration.

7. Conclusion

We have presented an algorithmic development framework for designing and testing CNN algorithms using UMF diagram-like notation. Simulink gives an environment for handling various data streams and for creating complex flow structures to build algorithms. With the additional Simulink Blocks of SimCNN we can get an efficient CNN simulator for an affordable price. We have stated our design considerations for optimal mapping between the CNN and the GPU architecture, resulting in a competitive speed that is twelve times faster than the implementation using high-level optimization on a cutting-edge CPU for a single template. To apply for this SimCNN software package, visit [24].

Acknowledgments

The Office of Naval Research (ONR), the Operational Program for Economic Competitiveness (GVOP KMA), the NI 61101 Infobionics, Nanoelectronics, Artificial Intelligence Project, and the National Office for Research and Technology (NKTH RET 2004) which supports the Multidisciplinary Doctoral School at the Faculty of Information Technology of the Pázmány Péter Catholic University are acknowledged. The work was sponsored by NVIDIA Corporation and Tateyama Laboratory Hungary Ltd. The authors are also grateful to Professor Tamás Roska, Kristóf Karacs, and the members of the Robotics Lab for the fruitful discussions and their suggestions. Special thanks go to Zsófia Cserey and Éva Bankó.

References

[1] T. Roska and L. O. Chua, "The CNN universal machine: an analogic array computer," *IEEE Transactions on Circuits and Systems II*, vol. 40, no. 3, pp. 163–173, 1993.

[2] A. 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.

[3] P. Dudek and P. J. Hicks, "A general-purpose processor-per-pixel analog SIMD vision chip," *IEEE Transactions on Circuits and Systems I*, vol. 52, no. 1, pp. 13–20, 2005.

[4] AnaFocus, "eye-RIS 1.1 leaflet of AnaFocus ltd.," 2007, http://www.anafocus.com.

[5] A. Zarándy and C. Rekeczky, "Bi-i: a standalone ultra high speed cellular vision system," *IEEE Circuits and Systems Magazine*, vol. 5, no. 2, pp. 36–45, 2005.

[6] Z. Nagy and P. Szolgay, "Configurable multilayer CNN-UM emulator on FPGA," *IEEE Transactions on Circuits and Systems I*, vol. 50, no. 6, pp. 774–778, 2003.

[7] P. Földesy, A. Zarándy, C. Rekeczky, and T. Roska, "Digital implementation of cellular sensor-computers," *International Journal of Circuit Theory and Applications*, vol. 34, no. 4, pp. 409–428, 2006.

[8] S. Tőkés, L. Orzó, C. Rekeczky, T. Roska, and A. Zarándy, "An optical CNN implementation with stored programmability," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '00)*, vol. 2, pp. 136–139, Geneva, Switzerland, May 2000.

[9] T. Roska, "Computational and computer complexity of analogic cellular wave computers," in *Proceedings of the 7th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '02)*, pp. 323–338, Frankfurt, Germany, July 2002.

[10] The MathWorks, "Simulink—Simulation and Model-Based Design," 2008, http://www.mathworks.com/products/simulink.

[11] A. Fernandez, R. S. Martin, E. Farguell, and G. E. Pazienza, "Cellular neural networks simulation on a parallel graphics processing unit," in *Proceedings of the 11th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '08)*, pp. 208–212, Santiago de Compostela, Spain, July 2008.

[12] B. G. Soós, Á. Rák, J. Veres, and G. Cserey, "GPU powered CNN simulator (SIMCNN) with graphical flow based programmability," in *Proceedings of the 11th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '08)*, pp. 163–168, Santiago de Compostela, Spain, July 2008.

[13] L. Kek, K. Karacs, and T. Roska, "Cellular wave computing library v2.1," Tech. Rep. CSW-1-2007, Cellular Sensory Wave Computers Laboratory, Computer and Automation Research Institute, Hungarian Academy of Science, Budapest, Hungary, 2007.

[14] L. O. Chua and T. Roska, *Cellular Neural Networks and Visual Computing*, Cambridge University Press, Cambridge, UK, 2002.

[15] Intel, "The world's first 2-Billion transistor microprocessor," 2008, http://www.intel.com/technology/architecture-silicon/2billion.htm?iid=homepage+news_itanium_platform.

[16] J. D. Owens, D. Luebke, N. Govindaraju, et al., "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[17] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, no. 4-5, pp. 589–604, 2005.

[18] NVIDIA, "GeForce 8800 GPU architecture overview," Technical Brief TB-02787-001_v1.0, NVIDIA, Santa Clara, Calif, USA, November 2006.

[19] NVIDIA, "CUDA compute unified device architecture—programming guide 1.1," November 2007, http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf.

[20] R. Kunz, R. Tetzlaff, and D. Wolf, "SCNN: a universal simulator for cellular neural networks," in *Proceedings of the*

4th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '96), pp. 255–259, Seville, Spain, June 1996.

[21] C. S. W. C. Laboratory, "Candy," 2000, http://lab.analogic.sztaki.hu/index.php?a=downloads&c=2.

[22] Z. Nagy, *Implementation of emulated digital CNN-UM architecture on programale logic devices and its applications*, Ph.D. thesis, Department of Image Processing and Neurocomputing, University of Pannonia, Veszprém, Hungary, 2007, http://twilight.vein.hu/phd_dolgozatok/ls.php?s_dir=nagyzoltan.

[23] "Intel performance libraries," 2008, http://www.intel.com/software/products/perflib.

[24] RobotLab, "SimCNN," 2008, http://robotlab.itk.ppke.hu/simcnn.

Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 751687, 9 pages
doi:10.1155/2009/751687

Research Article

APRON: A Cellular Processor Array Simulation and Hardware Design Tool

David R. W. Barr and Piotr Dudek

School of Electrical and Electronic Engineering, The University of Manchester, P. O. Box 88, Manchester M60 1QD, UK

Correspondence should be addressed to David R. W. Barr, d.barr@postgrad.manchester.ac.uk

Received 12 September 2008; Accepted 21 March 2009

Recommended by David Lopez Vilarino

We present a software environment for the efficient simulation of cellular processor arrays (CPAs). This software (APRON) is used to explore algorithms that are designed for massively parallel fine-grained processor arrays, topographic multilayer neural networks, vision chips with SIMD processor arrays, and related architectures. The software uses a highly optimised core combined with a flexible compiler to provide the user with tools for the design of new processor array hardware architectures and the emulation of existing devices. We present performance benchmarks for the software processor array implemented on standard commodity microprocessors. APRON can be configured to use additional processing hardware if necessary and can be used as a complete graphical user interface and development environment for new or existing CPA systems, allowing more users to develop algorithms for CPA systems.

Copyright © 2009 D. R. W. Barr and P. Dudek. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

Massively parallel processing systems have been the topic of high-performance computing research and design for many years [1–4], but in recent years single-chip implementations of such systems are becoming a reality. Cellular processor arrays (CPAs) such as the ones presented in [5–10] implement data processing at a fine-grain level of parallelism. Unlike sequential and coarse-grain parallel processing systems, such as multicore processors [11–13], which have large and complex instruction sets, CPAs are comprised of simpler processors, with a reduced instruction set. Typically they are used to perform lower-level data-parallel algorithms and operate in the Single Instruction Multiple Data (SIMD) mode, that is, every processor executes the same instruction but operates on their own local memories. The processors require much less silicon area due to their simplicity (compared to standard microprocessors) which means high numbers of them can be placed on a device, usually arranged in a 2D grid (Figure 1). The massively parallel nature of the device and spatial positioning of the processors give performance and efficiency advantages, in particular in data-intensive algorithms which require a high memory bandwidth, as each processor is able to access its own data and exchange data with its neighbours. A nonparallel system would have to perform many instructions, including memory sequential accesses to accomplish the same task. The low power consumption of a CPA, significantly lower than that of an equivalently performing sequential system (such as a desktop PC or workstation), is an appealing property for designers of low power embedded systems.

The highly parallel nature of CPAs has increased appeal to a variety of fields, where many simple computations must be performed on data arrays. The applications include embedded vision systems [6], high-speed image processing [5, 8], cellular nonlinear networks (CNNs) [7, 14], and computational neuroscience models [15–17]. These applications often require simple numerical operations and neighbour communication but use operations that are applied homogeneously to large data-sets, at high speeds. The fine-grain nature of many CPAs means that thousands of local memories can be operated on simultaneously, and the simplicity of the processors allows for fast execution of instructions.

The design of CPAs presents a set of challenges. It requires difficult trade-offs and design decisions, such as accuracy or speed, analogue or digital, and memory or array size. The drive to achieve a high density of processors in

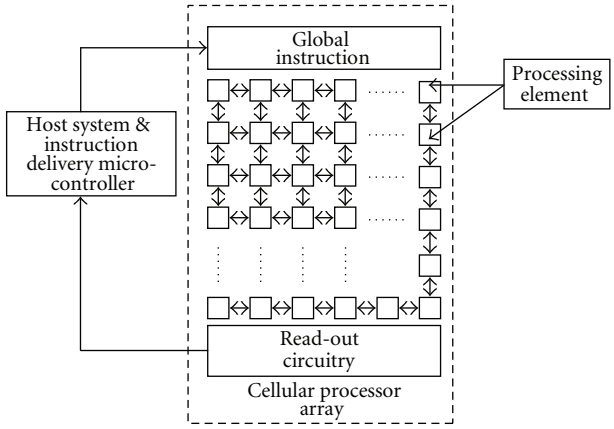

Figure 1: An overview of a simple cellular processor array system.

a single chip often makes memory a limited resource and reduces their computational ability. Also, the specialised nature of the hardware often makes it less accessible to potential application developers, while typically used short word lengths or analogue storage techniques limit their accuracy. Further to this, many CPAs require additional interfaces and control systems which are specifically built for the device, introducing additional design costs and possibly restricting the performance of the overall CPA system.

This paper is an extended version of work presented in [18] and presents APRON (Array Processing enviRONment), a high-speed, flexible, and virtual CPA implemented entirely in software. The software fulfils two objectives: firstly to provide an extensible, customisable implementation of a general purpose array processing engine, which is a basis for the design and "virtual prototyping" of new CPAs; secondly to provide a high-speed emulation of any CPA-based device, allowing the user to design pixel-parallel image processing algorithms and explore multilayer neural networks, cellular automata, and other phenomena related to CPAs, with a variety of tools for data analysis, performance evaluation, and algorithm development. APRON software is written to take advantage of the high-speeds of modern desktop computer systems to provide optimised array processing performance. To this end it is not intended to replace low-power embedded devices but serves as a fast, accurate substitute in environments where custom hardware is not available. APRON has been designed to provide user-friendly tools and achieve the maximum computational performance, with minimal overheads, without the need for custom low-level implementations.

This paper is organised as follows: Section 2 presents the APRON simulation core and the integrated development environment (IDE); Section 3 presents the potential applications/research fields where APRON can be used effectively and how the system can emulate several types of CPA device simultaneously; Section 4 highlights the flexible architecture of the APRON compiler and illustrates how it can be modified to emulate or prototype hardware devices; Section 5 presents a performance analysis and benchmarks of the APRON system. Throughout the paper, performance figures are presented, which are summarised in the conclusion.

2. APRON System Overview

2.1. The Simulation Core. The APRON simulation core is a platform-independent module that can be used in many applications for fast array processing in software. The core provides a virtual array processor, operating on three data types with a flexible instruction set and a virtual microcontroller. The microcontroller controls the program counter which permits nonlinear sequencing of algorithm instructions. APRON by default uses 32-bit single precision floating point numeric representations in all operations but it is possible to change this if necessary to lower bit-size integer types. The data types are registers, kernels, and linkmaps.

Registers are two-dimensional arrays of numeric elements called cells, which are 16-byte aligned in memory contiguously, to facilitate cache/page management and to take advantage of the extended x86 intrinsic architectures: SSE (Streaming SIMD Extensions) and SSE2 [19]. These architectures provide an SIMD vector processor in the silicon fabric of the CPU. This vector processor can process four single-precision floats in a minimal number of clock cycles, providing a calculation throughput much faster than that of the standard floating point unit (FPU). SSE also provides cache management facilities, meaning that registers can be pre-cached close to the processor, and even read/written straight from/to the main system memory, without polluting the cache if necessary. APRON registers adopt one of two forms. A data register contains numeric data, and a mask register contains binary data, which is used to enable and disable the write privileges to a data register, in effect permitting regional register operations (where global register operations are default). For many operations, the target register can also be a source register due to the sequencing of instructions sent to the CPU (e.g., giving $x = x + 1$ functionality). Figure 2 shows how data is sequenced, sent to the vector processor, and combined with a mask register to emulate the effect of disabling a processing element's "enabled flag", providing local autonomy. The contents of the target register should only be updated if the mask register's corresponding bit is set. This requires additional logic when working with a vector of four elements, of which only some of the corresponding mask register bits are set, to form a combined result vector, which is written to the target register. The mask register contains a 32-bit floating point NaN (0xFFFFFFFF), for each processing element that is enabled, and a 0 (0x00000000) for those that are disabled. Whilst at first this may seem like a gross overuse of memory, having all the bits set to 1 is useful for rapid logic functions; in this case a logic OR is computed between the data register's vector and the corresponding mask vector. This is an efficient way of implementing branching in the vector processor.

A kernel is an $N \times N$ matrix used in two-dimensional convolution/filtering operations and is applied to a register. Kernels come in two forms. A standard kernel is used in typical greyscale image filtering operations (convolutions), such as Gaussian blur and gradient-detect. A template kernel is a binary kernel used for binary pattern matching, comprising of hits, misses, and do not cares. A logic OR operation is performed within the domain of the kernel

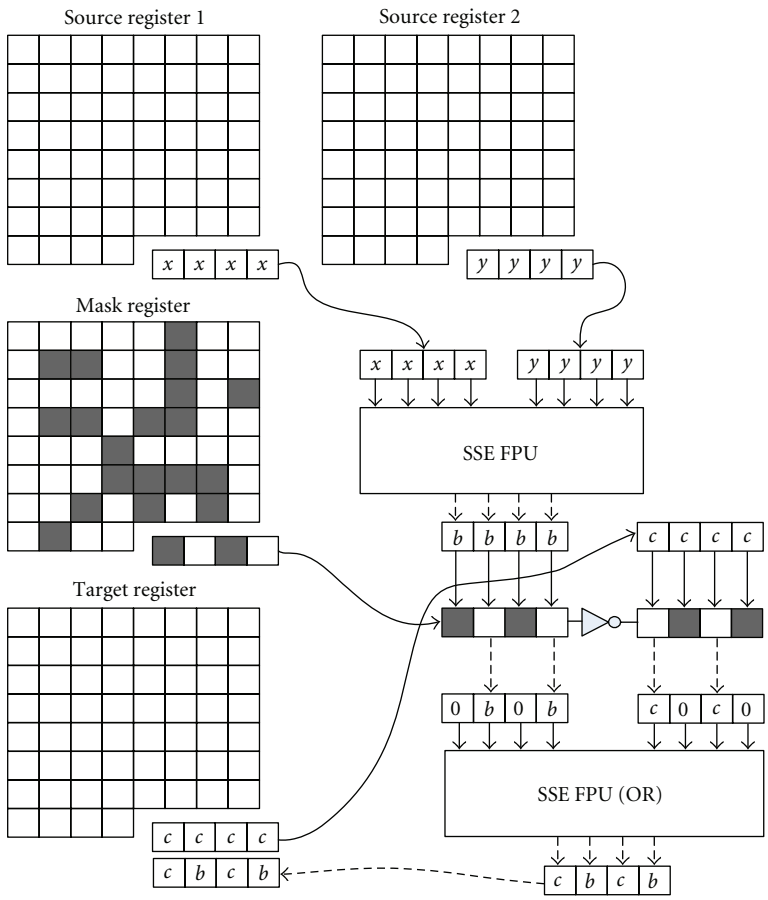

Figure 2: Disabling processor elements is achieved by restricting writes to the register memory by using the mask register in a sequence of logic operations. Dashed lines are memory writes, and the others are reads.

and the register, and the binary result is placed in a result register. Unfortunately, SSE2 does not offer a simple way of executing the kernel operation quickly, due to a lack of efficient sideways communication within the vector processor (however SSE3 does support this and is becoming standard on modern processors). Instead, APRON ensures all data required is in the highest level of cache, as close to the processor as possible, reducing cache misses (and eliminating paging), and the OS is stopped from swapping out the APRON process for the duration of the operation. The code is written in such a way as to be optimised greatly by the C++ compiler. The disassembly shows that 95 assembly level instructions are required for a fully filtered pixel using a 3×3 matrix.

Linkmaps are a structured list of all the connections from a single cell. APRON supports local neighbour connectivity by default, so registers can be shifted and rotated in the 4 standard compass directions, but by having no hardware connectivity limitations, it is possible to have any cell connected to any number of cells anywhere within the register. Linkmaps exist for each cell in the register, allowing unlimited connectivity possibilities. Each link is also accompanied by a multiplicative coefficient which is analogous to a synaptic weight (See Figure 3). When a linkmap is used, each source register value is multiplied by each of its connections defined by the linkmap. Each target register value is the result of an accumulation function, which is either a summation or maximisation of all the multiplicative connections that link to that value's location. The APRON IDE allows the algorithmic generation of linkmaps through a built-in Python interpreter and a set of parametric maps (registers pre-filled with data), which means the connectivity output from a register's cell can vary in a spatially controlled way. Linkmaps can become large, for a fully specified linkmap of a register size $N \times N$; N^4 connections are required. Often this is not the case, so connections below a specified value are assumed to be negligible and are omitted from the linkmap list.

A variation of the linkmap is the parametric linkmap, in which the connectivity is described parametrically; for example, all of the connections are based on the same Gaussian equation, but different coefficients can change the shape of that equation. Currently, this is computationally time consuming and is the subject of ongoing research, as the potential of using additional special-purpose hardware to perform this task could greatly reduce the memory bandwidth requirements of a fully connected linkmap. APRON is a vehicle for testing various connectivity acceleration strategies, whilst keeping the model structure intact, permitting fair benchmark comparisons between each technique.

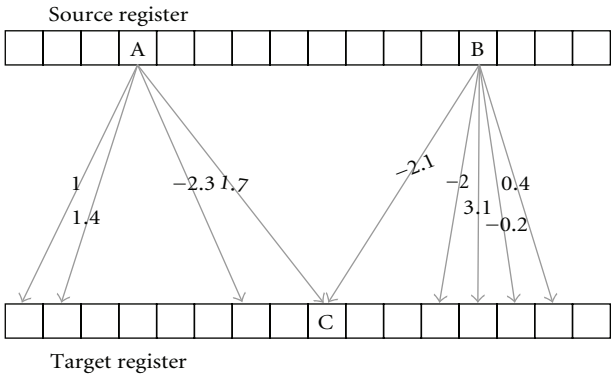

Figure 3: Linkmaps allow multiplicative connections between the processing cells. Each cell can be connected to any other cell. Here, different linkmaps are shown for cells A and B, where C = 1.7A−2.1B.

Linkmaps allow modelling of CPAs with elaborate interprocessor connectivity patterns. However, their main use is for implementing multilayer topographic neural networks and other models in the field of computational neuroscience. The use of APRON in this context will be elaborated in our future publications.

The simulation core has been designed to be extensible, so if necessary additional hardware (such as FPGAs or existing processor arrays) can be used to increase the computation speed. Currently the core supports 63 operations/instructions, including mathematical operators, register translation, filtering, linkmap training, and external data transfer. A simple plug-in interface permits users to build their own modules, which can be accessed at runtime during the simulation for transferring/computing register and other data. In effect, this allows any form of operation to be performed. Being a "virtual processor array" many techniques which are difficult to implement in hardware are relatively simple in APRON, an example being subwindowing and scaling of registers, where isolated regions of registers can be transferred to different locations in other registers.

2.2. Flexible Simulation Compiler and Simulation Environment. To accompany the simulation core, a flexible compiler has been developed that allows the user to develop algorithms, systems, and model devices in an intuitive way. Consequently, a special purpose scripting language called "APRON-Script" has been developed which is easy to use and serves as the underlying description mechanism for all simulations, prototypes, and algorithms. All APRON-Script code can be checked for syntax errors and validated automatically. The scripting language does not require variable declaration and operates on an assumption that one line of code corresponds to one hardware instruction. This makes looking for performance bottlenecks and optimisation easier. Multiple source-code files can be combined into a single project, which is subsequently compiled into an object code for the simulator. This promotes source-code reuse and eases the amalgamation and distribution of projects. The user can define environmental constraints such as the number and dimensions of registers, constants, and macros for a specific device they are emulating, which can be given to other users who wish to develop algorithms for that platform. By using a stand-alone compiler, a variety of popular source editing tools can be used, and user-friendly attributes such as on-line documentation and syntax highlighting are available, promoting more rapid development.

The simulator provides tools for executing and analysing algorithms. All registers are visually viewable as 2D images and can be numerically inspected. The user can step through code or enter a "run" mode, which executes the algorithms as fast as possible. For long simulations on slower computers, a low-priority run mode is provided. The simulator also provides dynamic input/output with some included plug-ins. Currently APRON provides a module to read and write data to files, a module to interface with an imaging device (e.g., a webcam) for capturing real-time visual input and a module to communicate with other APRON simulations over a network, providing a convenient way of implementing large-scale simulations. A module could also be an interface to external hardware devices, such as robotic arms, pan, tilt chassis, and so forth, in effect enabling APRON to be a development environment for intelligent systems or autonomous agents, with sensor input, efficient image and neural processing, and actuator output. An APRON script algorithm can also send interrupts to the simulator to perform tasks, such as update the display, reposition windows, display performance information, issue breakpoints, and display run-time error information. Simulations can be terminated at any time. Each instruction is timed and displayed, for identifying performance bottlenecks.

3. Applications

3.1. Generic Processor Array. APRON has been optimised for speed and accuracy. Using just the default APRON-Script instructions, it is possible to implement a wide range of array processing algorithms. Essentially, APRON is still a serial processing system, and so the smaller the registers, the better the performance. The benchmarks in Figure 7 show that for simple numeric operations, global register (128×128) operations occur in several microseconds, meaning tens of thousands of register operations can be performed per second. This would imply APRON could potentially be used in both practical and research applications. Algorithms are written in default APRON-Script, which in this scenario can be considered a high-level language. The optimised simulation core allows APRON to outperform popular alternatives such as MATLAB and regular (naïve) C/C++ implementations.

3.2. Vision Systems. The APRON simulator allows registers to be filled with data acquired from an imaging device (e.g., a webcam). Therefore live image data can be used during simulation. When operating in "run" mode, many algorithms execute fast enough to be interactive and responsive to the visual input. Taking Sobel Edge detection as an

$$G_x = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \times A \qquad G_y = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \times A$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Figure 4: The Sobel Operator equations, used in APRON benchmarks, where G is the result of a filtered pixel at location (x, y), and A is the unfiltered image.

example (see Figure 4), APRON can complete this algorithm for a 128 × 128 array in 0.5 ms. Imaging devices vary in capturing/transfer time, but webcams typically take between 10 and 100 ms to both capture and transfer an image into APRON. Therefore, a complete capture-process cycle can take 10.5 ms, giving a frame-rate of 95 fps. Higher quality cameras may bring this down to 2 ms, giving 400 fps. Vision systems implemented in the APRON virtual array processor can be considered similar to pixel-parallel vision chip systems [20]. For 3 × 3 filtering of a 128 × 128 array, APRON has a throughput of 81 MP/s (Mega-Pixels/Second) and for simple operations (add/multiply) 958 MP/s; for 512 × 512 this is 51 MP/s and 244 MP/s, respectively. For comparison, in [21], a 3 × 3 filter on a 3.0 Ghz Pentium4 processor gives 9.7 MP/s; a Xilinx Virtex-II Pro, 202 MP/s; an *n*Vidia 6800 Ultra GPU, 278 MP/s. These results do not take into consideration any other processing or communication requirements, unlike APRON. It may often be the case that actually transferring the data to/from these devices has a substantially negative effect on the performance of the system as a whole.

3.3. Cellular Nonlinear Networks. APRON is easily configured to behave as a Cellular Nonlinear Network (CNN), for example, implementing the classical Chua model [22]. As with the Vision System description previously, an imaging device can be used as input. The performance of the model varies with the array size and the type of algorithm executing (timing and iteration requirements vary). For example, APRON can perform the "greyscale feedback convolution" task in 180 μs, which could be compared with results presented in [7].

3.4. Topographic Neural Network Models. These models often represent abstractions of known neuroanatomy, where neurons are grouped into layers and can have complex interlayer connectivity, such as those described in [15–17]. Usually special purpose software is developed to simulate such models as in [23], but APRON can be configured to perform these simulations taking advantage of its optimised routines. The high numerical precision and specific instruction set of APRON enables the implementation of many types of neural network models, using array registers in a way similar to that in [24]. Unlike hardware processor arrays, APRON has fewer limitations on resources used, supporting over 32 000 registers (of a maximum 2048 × 2048 elements). Neurons are arranged into homogeneous layers, the size of a register; however, the neuron parameters need not be homogeneous. For example, a layer of Izhikevich [25] neurons can be implemented using 2 registers to store the states, 1 register to store the spike output, and several temporary registers for calculating the state update. Network connectivity is provided through the use of linkmaps described previously. On a 1.8 GHz Intel Core2 Duo processor, an Izhikevich neuron takes approximately 21 ns to update, potentially allowing over 45 000 neurons to be updated in real time with a 1 ms time step. Gaussian receptive fields can be implemented either through linkmaps or by using a Gaussian blurring filter on a register filled with the output of the layer [24]. A single 3 × 3 blurring filter applied to a 128 × 128 register takes approximately 170 μs.

Combining the "virtual" vision chip and processor array results in a powerful modelling tool suitable for use in retinal and visual-cortical simulations, often in a more controllable environment than that delivered by hardware-oriented systems. This flexibility allows the developer to create entire "system-on-virtual-chip" applications that may execute at speeds suitable for practical application use.

4. Hardware Prototyping/Emulation

One definite benefit a software CPA has over hardware is the relatively unlimited possibilities for processor element complexity, communication strategy, and memory resources. APRON provides many optimised operations (here considered low level) which can be combined to form higher-level more complex macros. This ensures that the CPA designer is using optimised code to implement the functionality of a cell's processing element. APRON provides at the very least the ability to read and write to each cell individually, which is sufficient to implement the most complex and diverse operations.

4.1. Customizing APRON for Device Emulation. Figure 5 shows the data flow through the APRON compiler and simulator. The APRON compiler has three levels of abstraction. The highest is the scripting language itself, which is human readable, and consists of a combination of APRON instructions and macros. A macro is similar to a function, in that it encapsulates many APRON instructions into what appears to be a single instruction, except it becomes explicitly unrolled into the instruction sequence rather than placed onto the stack when called. Macros have global bodies, that is, they can access any resource, but have local labels and variables as well, to maintain the encapsulation. For example, in the scenario where the same macro is called twice, it is important not to have one macro jump to instructions in the other macro. Macros are defined in external "ruleset" files (akin to "header files" programmed in APRON-Script syntax), which also define resource name changing, constant declaration, operator overloading, and some environmental properties such as Register dimensions. The simulator provides a facility to debug the macros verbosely, in order to find errors in the ruleset.

Using macros, a cellular processor array designer could specify the complete instruction set of a hardware device, knowing the APRON instructions will deliver repeatable results. This is useful for both the emulation of existing

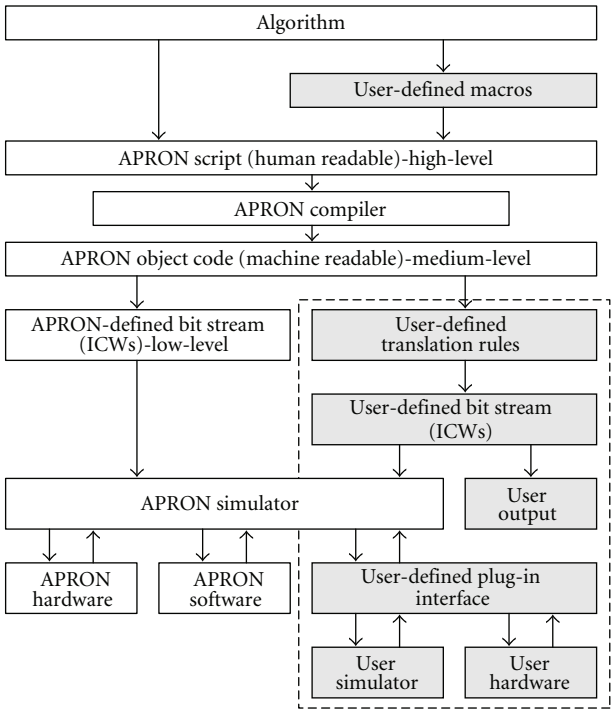

Figure 5: APRON Compilation/System Diagram. The shaded regions show user configurable components that customize APRON. The boxed region indicates components that need to be programmed and imported as a dynamic library.

hardware and the prototyping of new hardware. The designer could build several rulesets, describing the hardware at different levels of abstraction, from an accurate, slower model of the analogue current or bit transfers/operations in the individual cells to a faster, abstract behavioural model of the device. In particular, when designing analogue processor arrays, it is possible to model low-level circuit and device behaviour including spatial and temporal noise, by emulating such components. Although APRON does not simulate these effects intrinsically (i.e., there is no dedicated support for individual device characteristics), they can be sufficiently modelled using the constructs APRON provides to create accurate real-world behaviours, allowing algorithm designers to take phenomena such as nonlinearities, error propagation, and device mismatch into account [26]. The APRON IDE already provides the tools for the development and analysis of algorithms and a fast interactive simulation tool; therefore, the designer does not have to create any software himself. The "ruleset" can be distributed to any user that wishes to develop algorithms for the particular CPA. This approach has been applied successfully to the ASPA vision chip [5], and APRON is used for algorithm development for that device. APRON will also be used for the development of SCAMP devices [6].

The middle layer of abstraction is APRON object code, which is a machine readable structure defining each instruction in the APRON-Script algorithm, after all necessary preprocessing (such as macro unrolling) has been applied. This structure breaks down the script instruction into a function name and a variety of parameters. The object code by default is then turned into a binary Instruction Code Word (ICW) stream. A single object-code instruction can consist of many ICWs. The ICW stream is the lowest level of abstraction in the APRON system. This stream is delivered to the APRON Core for execution

The module responsible for the translation from object code to an ICW stream is hot swappable, with a defined interface, meaning it is possible for designers to create their own ICW streams. This has the following several benefits.

(1) The user can use the APRON environment as a code editor/compiler for a custom CPA.

(2) The ICWs can be delivered to other applications, and even hardware devices.

(3) The ICWs can be delivered to a custom simulation core, by passing the APRON Core entirely, providing a highly customised simulation, but utilising the development environment and tools APRON provides.

(4) It is similar to 3 but actually uses a hardware device and APRON as a software interface or graphical user interface to the hardware.

4.2. Modelling ASPA in APRON. The ASPA vision chip [5] is a digital cellular processor array, which contains a bit-serial/bit-parallel synchronous data path as well as asynchronous wave-propagation circuitry. This allows a wave edge to be propagated across the CPA in a single instruction, very quickly. Therefore it has many applications in more complex image processing tasks, such as skeletonisation and object identifying. The traditional approaches to building a complete CPA system involve building the device, then building an accompanying interface system, and then designing some compiler and simulation tools. APRON is being used for the latter, with a little customisation.

Firstly, the APSA instruction set was completely modelled in detail in APRON, by defining a ruleset, which is a set of macros built entirely in APRON-Script, one corresponding to each ASPA instruction. This limits the algorithm developer to only using these macros and therefore only using valid ASPA instructions. The resulting "ASPA-Script" can be compiled in two ways. The first is to generate the actual hardware ICWs and store them in a file, which can subsequently be used to program the ASPA device. (The hardware ICWs may differ from those that are created for the simulator and can be defined algorithmically as part of the macro definitions.) The second is to unroll the ASPA-Script macros back into APRON-Script and run this code in the simulator. The process of simulating when using a ruleset is invisible to the algorithm developer, as the simulator presents the ASPA-Script for debugging and analysis. The detail of the ruleset can be targeted for different types of simulation. The most detailed simulation involves modelling the bit-level transfers between the registers in ASPA, providing good debugging information at the expense of performance. A higher-level simulation may assume the user is using 8-bit integers and model the device at that level, providing an adequate simulation of the device at high speed.

```
!macro_def
        propagate(0)
!macro_begin
 r[P_FLAG] = 0
 r[AUX2] = mask(r[C_FLAG], <=, 0)
 maskon(r[AUX2])
 r[P_FLAG] = 255
 maskoff(0)
 :#START
        :glob_sum = 0
        r[AUX1] = r[P_FLAG]%kernel[OR_KERN]
        r[AUX2] = mask(r[AUX1], >, 0)
        maskon(r[AUX2])
                r[AUX1] = 255
        maskoff(0)
        r[AUX2] = mask(r[PFL], >=, 1)
        maskon(r[AUX2])
                r[AUX1] = 0
        maskoff(0)
                r[AUX3] = r[P_FLAG]−r[AUX1]
        r[AUX2] = mask(r[AUX3], !=, 0)
        maskon(r[AUX2])
                r[AUX3] = 1
        maskoff(0)
        :glob_sum = sum(r[AUX3])
        r[P_FLAG] = r[AUX1]
        jumpif(:glob_sum, >, 0.0, :#START)
!macro_end
```

APRON-SCRIPT MACRO

```
. . .
. . .
r[PFL] = 0
vr = 255
sub_glob(r[C], r[A], vr)
if(C)
        r[PFL] = 255
endif(0)
vr = 250
sub_glob(r[C], r[B], vr)
propagate(0)
r[C] = 0
. . .
. . .
```

ASPA-SCRIPT ALGORITHM

Figure 6: APRON Macro definition of the ASPA function "propagate()". On the right is a snippet of ASPA-Script assembled from macros. On the left is an expansion of the propagate macro written in APRON-Script.

Table 1: Descriptions of the benchmark tests performed.

Test	Description
ADD	Two registers were filled with random values. The two registers were added and written to a third separate register 10 000 times.
MULT	Same as ADD, but with a multiply operation.
CONDMULT	Same as MULT, but the two registers were multiplied, only where the contents of register 1 are greater than θ, using a mask register.
FILTER	One register was filled with random values. A 3×3 Gaussian Blur Kernel was used to filter the register and the result was written to a second separate register, repeated 10 000 times.

As an example, Figure 6 shows an implementation of the ASPA "propagate()" function, which has been implemented in APRON-Script. Even though this is one instruction on an ASPA device, it needs several looped APRON operations. However, when the instruction is simulated, it will behave as a single step. This process is transparent to the user and often fast enough due to the efficient operations provided by the APRON core.

The ASPA ruleset is a portable definition of the ASPA device, with which other developers can rapidly create algorithms for the chip, without having the hardware to test them on. This has been achieved without needing to build a custom simulator for the ASPA device. The plug-in feature of APRON can be exploited for additional functionality, including being used as an interface to an actual ASPA device. This makes APRON a complete design, test, and development suite for the ASPA. At present a hardware interface has been implemented successfully for the SCAMP device, where frames of preprocessed data from the SCAMP device can be operated on further by efficient APRON routines.

5. Benchmarking

Throughout the development of APRON, a great effort has been made into optimizing the APRON Core, to make it as fast as possible. In order to assess its performance, a set of benchmarks were compiled that measure the execution time of commonly used operations and the performance of APRON, MATLAB (7.01), and unoptimised high-level

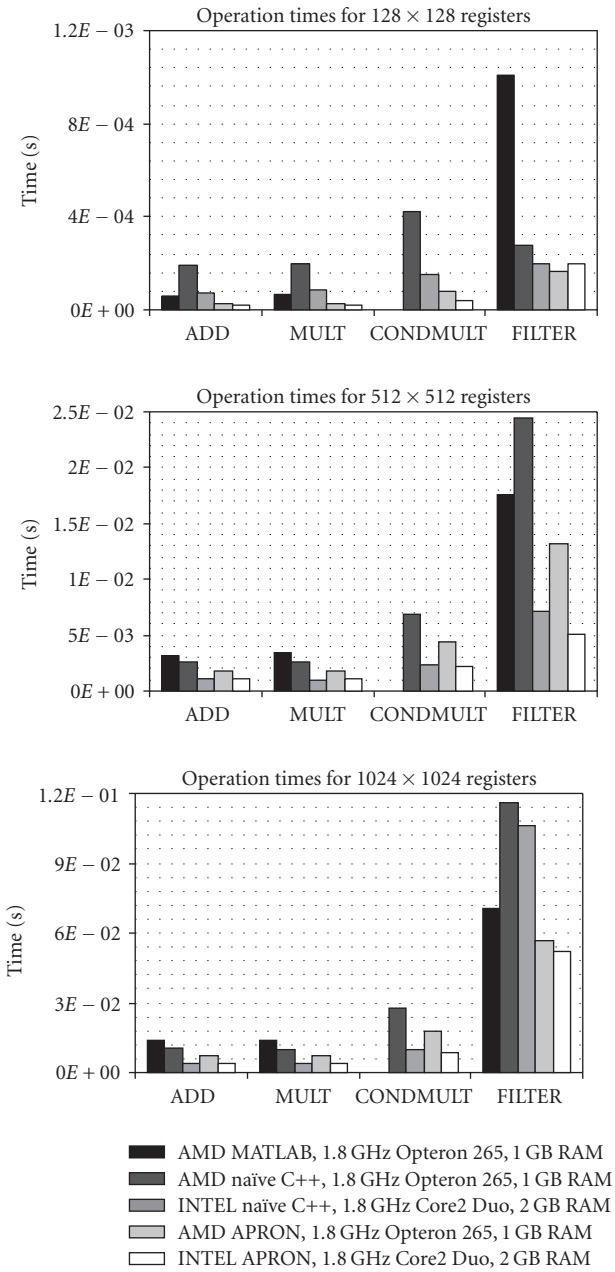

Figure 7: Benchmarks showing the performance of different array processing operations on different software platforms.

(naïve) C++ is compared. The tests performed are described in Table 1, and the results are shown in Figure 7.

Two test machines were used with similar specifications, a 1.8 GHz AMD Opteron 265, running Microsoft Windows XP, and a 1.8 GHz Intel Core2 Duo running in Windows Vista. It was not possible to benchmark MATLAB programs on the Intel platform at this time. The benchmark results show that APRON is faster than MATLAB at all tasks, often by a factor of 3, and faster than a naïve C++ approach in almost all cases. The results have shown interesting differences between Intel and AMD processors, with AMD being on the whole slower. This is most likely due to the nature of AMD's implementation of Intel's SSE technology. The operating system's memory/process management system also contributes to the performance, and this can be seen in the results of the larger register filtering tests, as SSE is not used for this operation.

APRON performs 128×128 array operations using single precision floats in tens of microseconds, which is suitable for a great range of experiments and applications. Execution times can largely be reduced four-fold if the change to using 8-bit integers is made.

6. Conclusion

This paper introduced an efficient software package used for the simulation and prototyping of cellular processor arrays. This software is useful throughout the life cycle of a cellular processor array-based device, from the initial design, modelling, and prototyping of the hardware, through to being used as an algorithm development platform, a simulator, and even a hardware interface. An example application of modelling a hardware CPA was presented, and the benefits that APRON provided to the user experience of working with the CPA were highlighted. APRON software proves to be faster than other software (and some hardware) environments for the tasks of emulating, prototyping, and simulating cellular processor arrays and can even be used as a stand-alone "array processing" system in many applications. Software tools like APRON can encourage the "massively-parallel" programming style and provide tools that ease the transition from software to massively parallel hardware implementations.

It is interesting to see that "virtual" massively-parallel devices created in software are beginning to compete with custom ICs and FPGAs in terms of computational performance, at comparatively lower cost. With commercial CPU manufacturers producing more parallel, lower power devices, perhaps software environments such as APRON will become more appealing to researchers interested in fine-grain processor arrays and their applications in areas such as computational neuroscience and image processing.

Acknowledgment

This work has been supported by the EPSRC, Grant no. EP/C516303.

References

[1] K. E. Batcher, "Design of a massively parallel processor," *IEEE Transactions on Computers*, vol. 29, no. 9, pp. 836–840, 1980.

[2] S. F. Reddaway, "The DAP approach," in *Infotech State of the Art Report on Supercomputers*, vol. 2, pp. 309–329, Infotech International, Maidenhead, UK, 1979.

[3] W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, Mass, USA, 1985.

[4] Y. I. Fet, *Parallel Processing in Cellular Arrays*, John Wiley & Sons, New York, NY, USA, 1995.

[5] A. Lopich and P. Dudek, "Global operations on SIMD cellular processor arrays: towards functional asynchronism," in *Proceedings of the International Workshop on Computer Architectures for Machine Perception and Sensing (CAMPS '06)*, pp. 18–23, Montreal, Canada, September 2006.

[6] P. Dudek, "Implementation of SIMD vision chip with 128 × 128 array of analogue processing elements," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 6, pp. 5806–5809, Kobe, Japan, May 2005.

[7] A. Zarandy, M. Foldesy, P. Szolgay, S. Tokes, C. Rekeczky, and T. Roska, "Various implementations of topographic, sensory, cellular wave computers," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 6, pp. 5802–5805, Kobe, Japan, May 2005.

[8] A. 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] M. Laiho, J. Poikonen, P. Virta, and A. Paasio, "A 64 × 64 cell mixed-mode array processor prototyping system," in *Proceedings of the 11th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '08)*, Santiago de Compostela, Spain, July 2008.

[10] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, no. 4-5, pp. 589–604, 2005.

[11] D. Wentzlaff, P. Griffin, H. Hoffmann, et al., "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[12] L. Seiler, D. Carmean, E. Sprangle, et al., "Larrabee: a many-core ×86 architecture for visual computing," *ACM Transactions on Graphics*, vol. 27, no. 3, article 18, pp. 1–15, 2008.

[13] W. J. Dally, U. J. Kapasi, B. Khailany, J. H. Ahn, and A. Das, "Stream processors: progammability and efficiency," *ACM Queue*, vol. 2, no. 1, pp. 52–62, 2004.

[14] D. Balya and T. Roska, "Face and eye detection by CNN algorithms," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 23, no. 2-3, pp. 497–511, 1999.

[15] B. E. Shi, "An eight layer cellular neural network for spatio-temporal image filtering," *International Journal of Circuit Theory and Applications*, vol. 34, no. 1, pp. 141–164, 2006.

[16] J. Sirosh and R. Miikkulainen, "A unified neural network model for the self-organization of topographic receptive fields and lateral interaction," in *Proceedings of the Joint Conference on Information Sciences (JCIS '94)*, pp. 282–285, Durham, NC, USA, November 1994.

[17] D. Balya and B. Roska, "Cellular wave computing the multi-channel mammalian retina model by the Bi-I camera-computer," in *Proceedings of the 10th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '06)*, pp. 1–12, Istanbul, Turkey, August 2006.

[18] D. R. W. Barr and P. Dudek, "A cellular processor array simulation and hardware prototyping tool," in *Proceedings of the 11th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '08)*, pp. 213–218, Santiago de Compostela, Spain, July 2008.

[19] Intel, AP-809, "Real and complex FIR filter using streaming SIMD extensions," version 2.1, January 1999.

[20] D. R. W. Barr, S. J. Carey, A. Lopich, and P. Dudek, "A control system for a cellular processor array," in *Proceedings of the 10th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '06)*, pp. 176–181, Istanbul, Turkey, August 2006.

[21] B. Cope, P. Y. K. Cheung, W. Luk, and S. Witt, "Have GPUs made FPGAs redundant in the field of video processing?" in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT '05)*, pp. 111–118, Singapore, December 2005.

[22] L. O. Chua and L. Yang, "Cellular neural networks: applications," *IEEE transactions on circuits and systems*, vol. 35, no. 10, pp. 1273–1290, 1988.

[23] J. A. Bednar, "Understanding neural maps with topographica," *Brains, Minds, and Media*, vol. 3, Article ID bmm1402, 2008.

[24] D. R. W. Barr, P. Dudek, J. M. Chambers, and K. Gurney, "Implementation of multi-layer leaky integrator networks on a cellular processor array," in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN '07)*, pp. 1560–1565, Orlando, Fla, USA, August 2007.

[25] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[26] P. Dudek, "Accuracy and efficiency of grey-level image filtering on VLSI cellular processor arrays," in *Proceedings of IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '04)*, pp. 123–128, Budapest, Hungary, July 2004.

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.

Copyright © 2009 S. Malki and L. Spaanenburg. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

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.

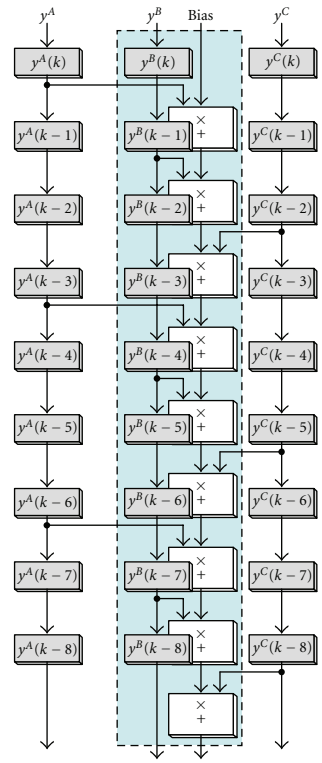

Figure 1: Data dependencies for a pipeline in naive architecture. Only the pipeline corresponding for the middle node is shown. White boxes represent functional blocks, consisting of a multiplier and an adder, while grey boxes represent registers. The middle node corresponds to a pixel sequence y^B.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

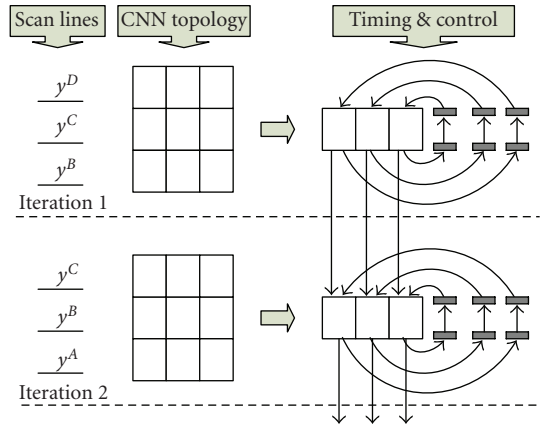

Figure 2: A pipelined CNN architecture with a pipeline three nodes.

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.

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.

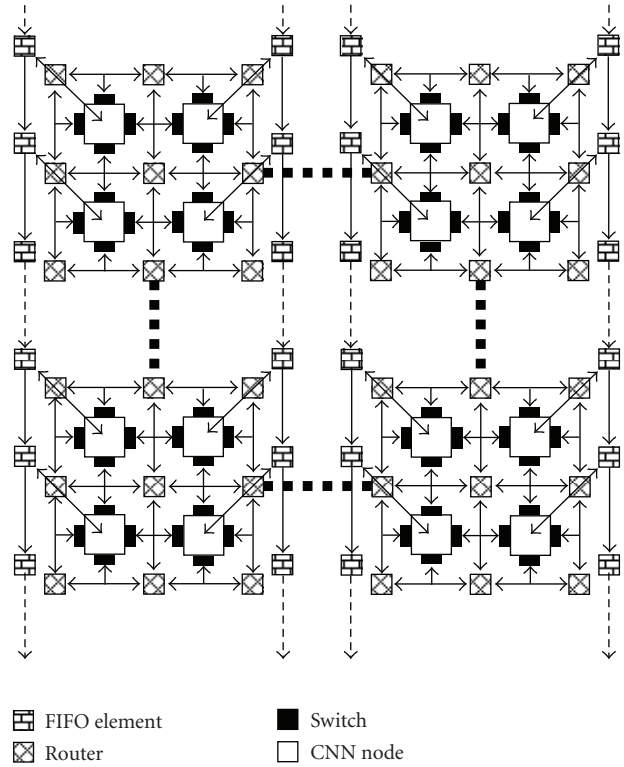

Figure 3: Caballero architecture uses a network-on-chip of CNN nodes, while the pixels are transported over a distributed FIFO.

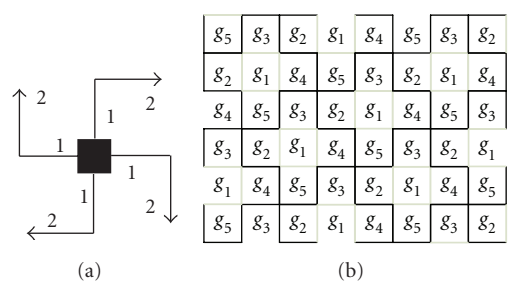

Figure 4: (a) Communication scheme and (b) activation groups in Caballero.

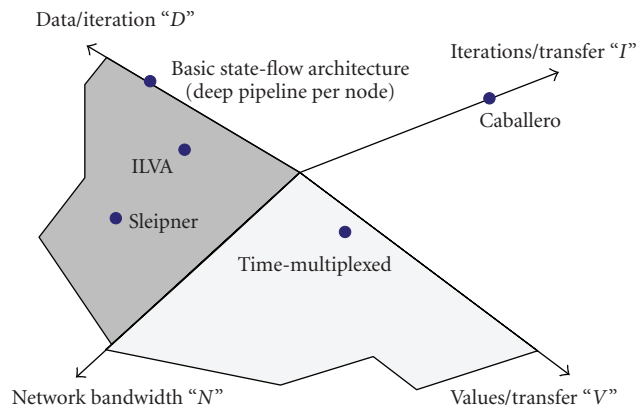

Figure 5: A 4-dimensional design space $\{V, I, N, D\}$ of CNN architectures.

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.

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

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 $n_{\text{slice}}^{\text{ILVA}}$:

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

(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.

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

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.

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

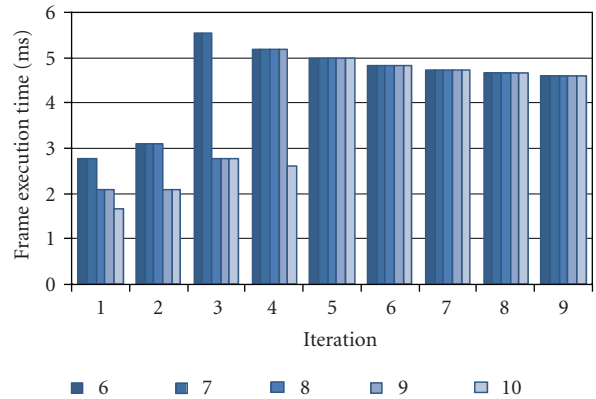

Figure 6: Frame execution time for ILVA with different CNN sizes, when slicing is required. The legends, 6 to 10, represent the number of pipelines, that is, the number of columns in the design.

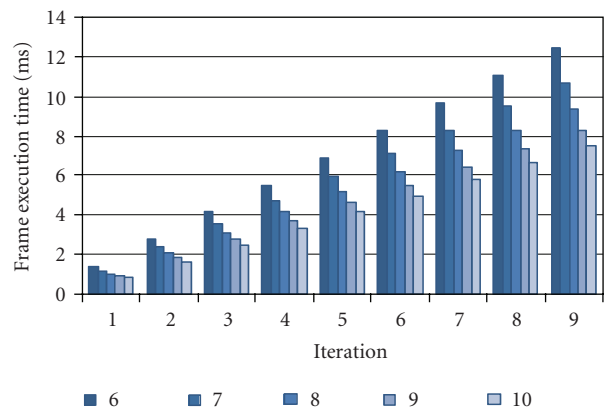

Figure 7: Frame execution time for Caballero with different CNN sizes, when slicing is required.

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.

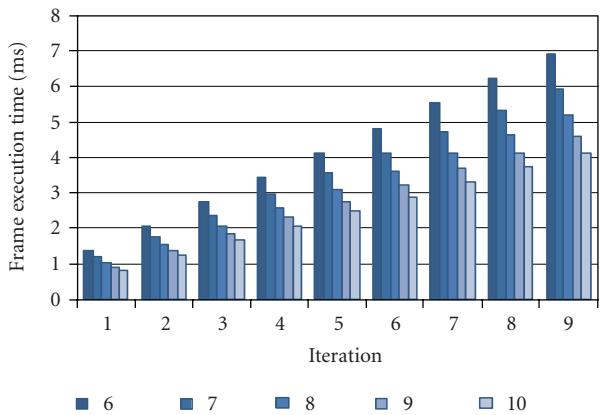

Figure 8: Frame execution time for Caballero is reduced when all the iterations are performed on a slice before next slice is brought in.

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,

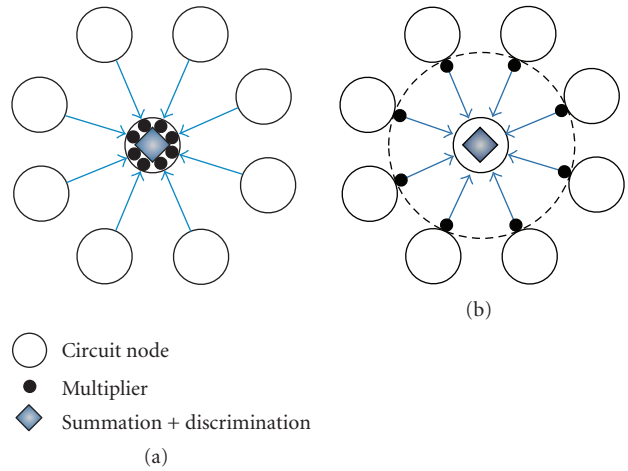

Figure 9: (a) Consumer and (b) producer cell to node mapping.

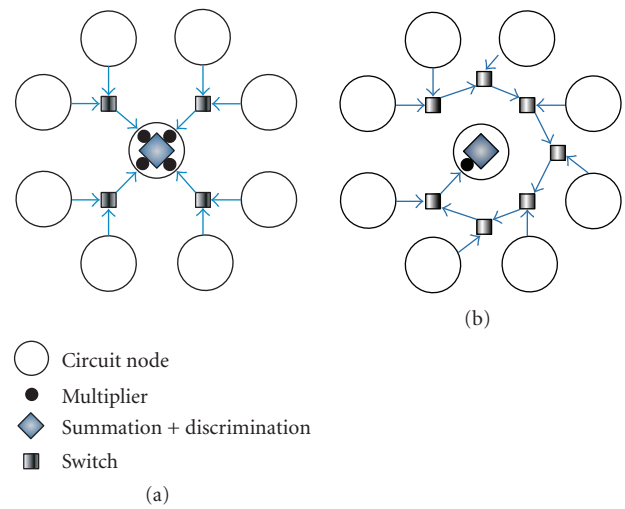

Figure 10: Value routing by multiplexing (a) in space and (b) in time.

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.

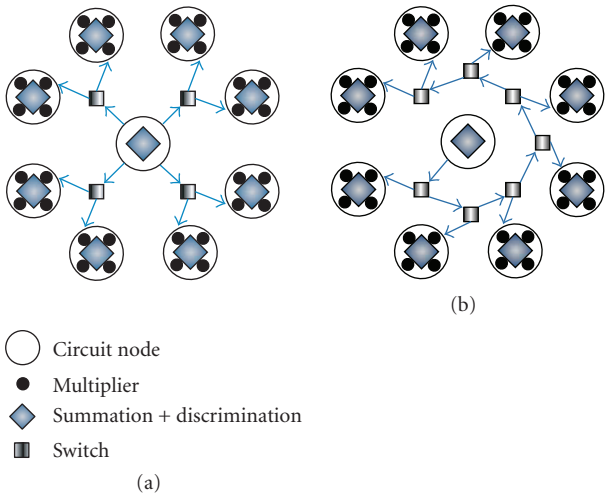

Figure 11: More value routing by multiplexing (a) in space and (b) in time.

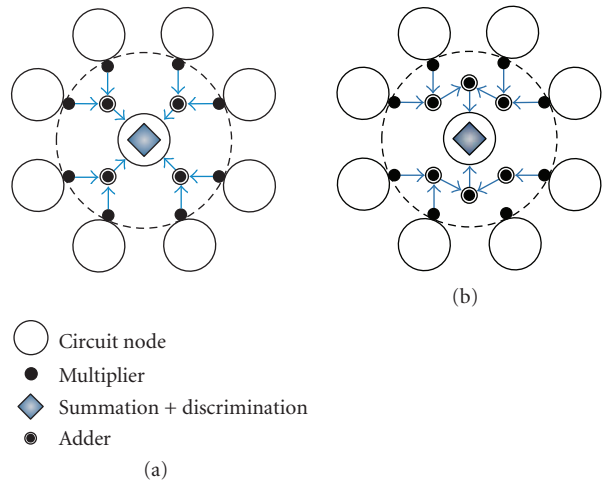

Figure 12: Adder trees combine the network in the producer architecture.

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

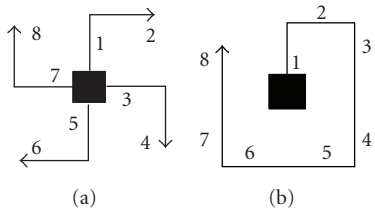

Figure 13: (a) Semiparallel and (b) serial switched broadcasting.

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

Base	u/y flag	Index	Address	
000000	0	XXXX	0 15	$B_1 + i_1$
000000	1	XXXX	16 31	A_1
000001	0	XXXX	32 47	$B_2 + i_2$
000001	1	XXXX	48 63	A_2

Figure 14: Address space of the nodal template memory.

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

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)). 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

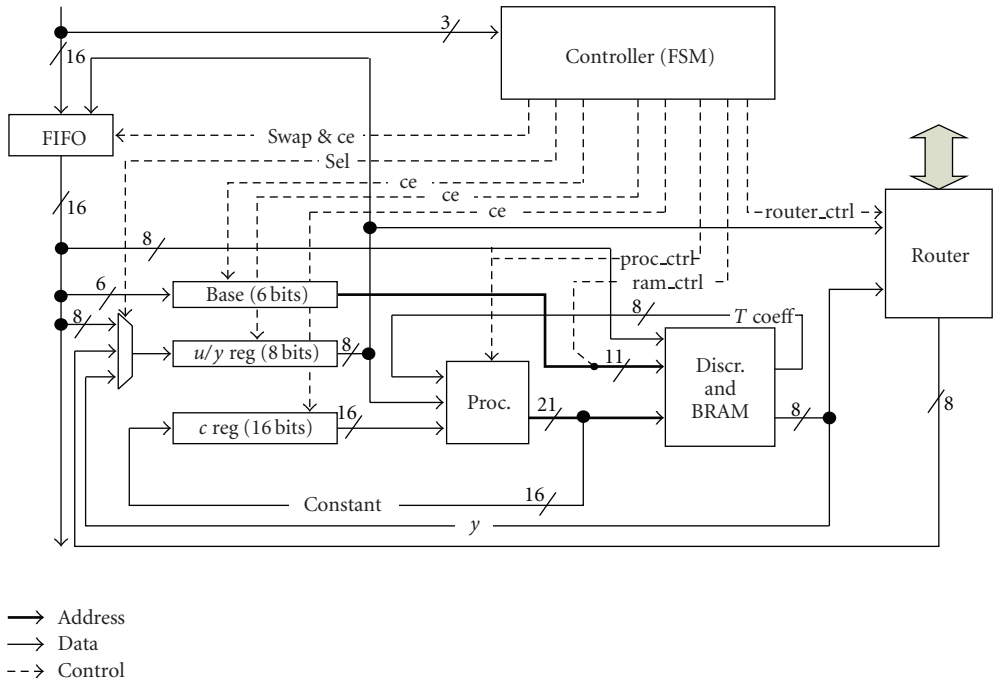

Figure 15: Schematic view of a serial CNN node.

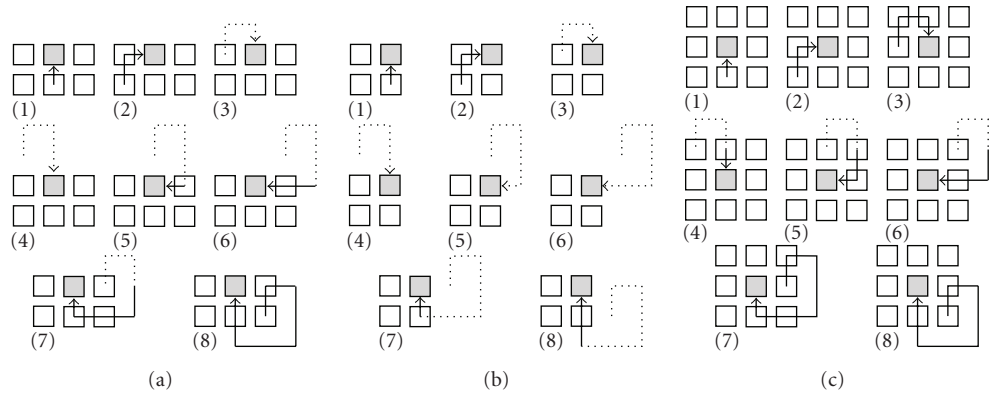

Figure 16: (a) Top edge, (b) right upper corner nodes, and (c) even close-to-boundary nodes have an incomplete communication cycle. Squares represent regular nodes; while the dotted lines show which part of the packet path is missing. The node of interest is shaded.

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

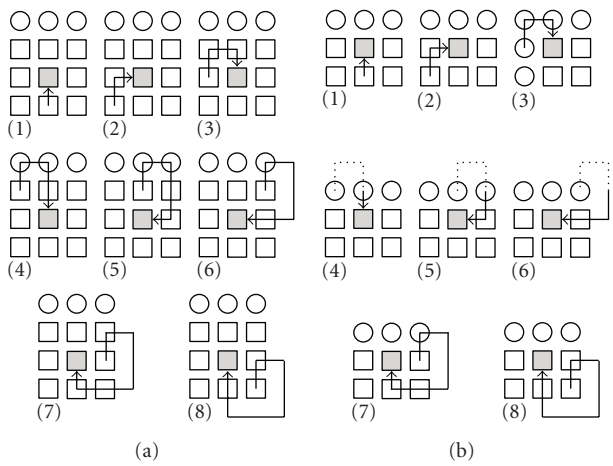

Figure 17: (a) The absence of neighbors is salvaged by adding a single layer of virtual nodes. (b) For boundary nodes more than one layer of virtual nodes is needed.

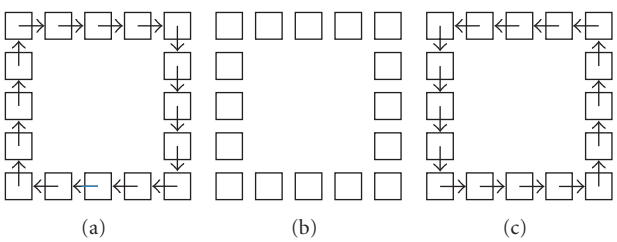

Figure 18: Swing broadcasting allows distributing of boundary conditions in (a) three steps clockwise and (c) anticlockwise. (b) For proper functionality on the duplex lines, a separating idle step is introduced.

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.

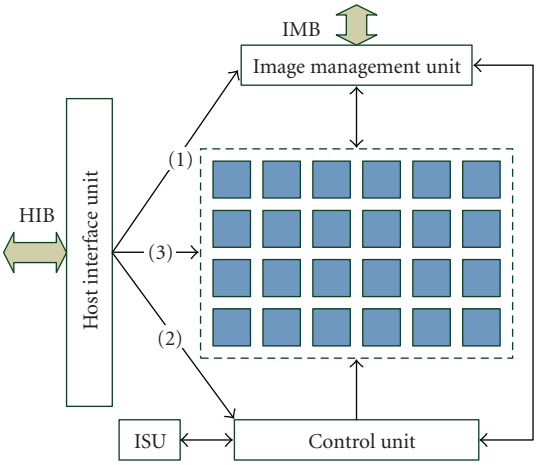

Figure 19: External view of the CNN architecture.

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

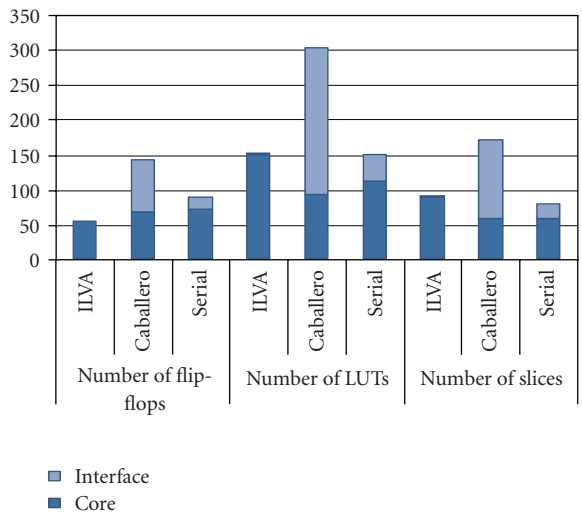

Figure 20: Area utilization per node compared to ILVA and Caballero architectures shows that nodal interface is kept at minimum which improves the overall logic utilization. The vertical axis reflects the number of flip-flops, LUTs, or slices of the three different architectures.

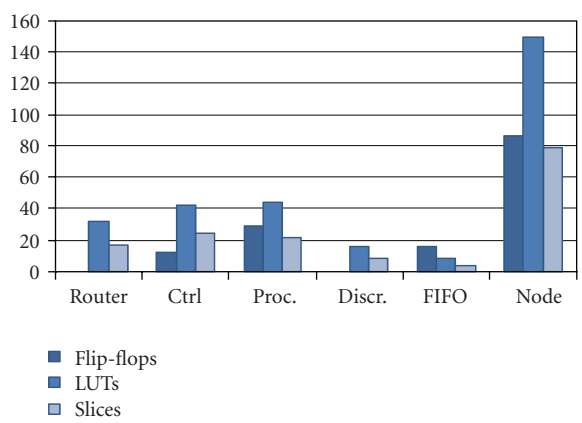

Figure 21: Area utilization of the different components with serial broadcasting scheme.

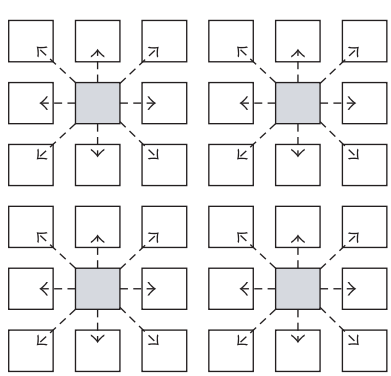

Figure 22: Semiglobal control with one controller per group of nodes.

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.

Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 127630, 11 pages
doi:10.1155/2009/127630

Research Article

An Analog Processor Array Implementing Interconnect-Efficient Reference Data Shift and SAD/SSD Extraction for Motion Estimation

Jonne Poikonen,[1] Mika Laiho,[1] Ari Paasio,[1] Lauri Koskinen,[2] and Kari Halonen[2]

[1] *Department of Information Technology, University of Turku, 20014 Turku, Finland*
[2] *Electronic Circuit Design Laboratory, Helsinki University of Technology, P.O. Box 300, 02015 Espoo, Finland*

Correspondence should be addressed to Jonne Poikonen, jokapo@utu.fi

Received 25 September 2008; Accepted 30 January 2009

Recommended by Diego Cabello Ferrer

A cellular analog processor array for use in variable block-size motion estimation with a new simple method for shifting reference image data is presented. The new shift method leads to a greatly reduced number of neighborhood connections for each cell of the array, and allows for all shifts within the [8,8] search area to be performed in a single step, with simple digital controls. The new shift circuitry, together with some other cell and system level optimizations , reduces silicon area and array layout complexity, enabling faster and more efficient parallel full search motion estimation hardware. A 32×32 cell parallel analog test array for reference-shift with a maximum block-size of 16×16, as well as absolute value/quadratic processing for variable block-size analog motion estimation (AME) has been designed in a 0.13 μm CMOS technology.

Copyright © 2009 Jonne Poikonen et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

Cameras with (multi-)megapixel sensors have become ubiquitous in even relatively low-end mobile phones. While this makes good quality still imaging possible, the limited amount of memory and processing power in such a battery-powered mobile platform often prohibits the use of the best available image quality for capturing video streams; typically a considerably poorer video capture resolution is used. The strong overall trend of memory technology scaling enables the integration of increasing amounts of memory within mobile phones. However, the increase of processing power which is required for real-time processing of the video stream is considerable.

An integral part of all video standards is motion estimation (ME), which can take up to 80% of the power consumption of a video encoder. For small frame sizes, the ME power consumption can be reduced through algorithmic methods, however, for megapixel resolutions these solutions are not sufficient. Without new optimized circuit techniques, the power consumption due to the motion estimation process will grow beyond the capabilities of small battery-powered platforms.

The currently applied video standards for mobile terminals (e.g., H.264) employ Block-Based Motion Estimation (BBME), and preferably variable block-size motion estimation. The most fundamental operation required for BBME is the shift of the reference-block data, to which the current frame data is compared, after which the best-matching new block position is determined with relatively simple processing. A performance advantage has been sought from performing the motion estimation operation in the analog domain and by employing a CNN-type [1] parallel processor array [2–8].

This paper describes the implementation of parallel processing hardware for an analog motion estimation (AME) array, with a focus on the implementation of a new reference data shift method. The proposed shift implementation leads to a significant reduction in the required cell interconnections, enabling a [8,8] cell search range to be implemented with a simpler array-level wiring than in previous implementations and with simple controllability.

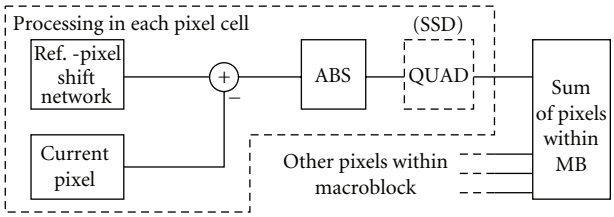

Figure 1: Cell-level functionality required for BBME.

This paper extends the original paper proposing the new reference-shift method [9], by also describing in detail the implementation of other circuitry for the array cell as well as presenting the implementation of a 32×32 cell AME test chip that has been designed and submitted for manufacturing.

The paper is divided into seven Sections. Section 2 discusses some implementation issues relating to a motion estimation array realization, Section 3 describes the new reference shift method in detail, and Section 4 examines the other circuitry in the array cell. In section 5 some important implementation issues are discussed, Section 6 describes the designed test array and examines the performance of other proposed motion estimation processors, and finally some conclusions are drawn in Section 7.

2. Analog Motion Estimation Array

Variable block-size motion estimation is based on comparing a macroblock of pixels, typically from 4 × 4 to 16 × 16 pixels, in the current image frame (C-frame) to blocks of the same size within the search area of a reference frame (R-frame). The position where the best matching of the macroblocks in the different frames is achieved represents the estimate for the motion in the image, that is, the motion vector. The matching at each position is evaluated by using a matching criterion which is typically either the Sum of Absolute Differences (SAD) or the Sum of Squared Differences (SSD) between the individual macroblock pixel values in the current and reference frames. The optimal selection of the method depends on the type of hardware implementation, SAD is more typically used in digital implementations because the required calculations are much simpler. A fair approximation of SSD can be easily implemented with current-mode analog circuitry, however, the accuracy compared to an actual squaring operation is limited by the nonideal characteristics of transistors, especially in modern deep-submicron technologies and with low power supply voltages. Figure 1 demonstrates the cell operations required for an analog motion estimation array. The different circuit blocks will be discussed in detail in the following chapters.

In principle, the optimal implementation of analog motion estimation would be to integrate the motion estimation circuitry together with each pixel in the photosensor array. By not having to convert the analog pixel values into digital form before motion estimation, considerable power savings could be achieved and the processing could be performed for the whole frame in a fully parallel manner. In reality this is not feasible for a megapixel sensor array, due to the resulting excessive silicon area required by the processing circuitry per pixel. Also, without A/D conversion, the input frames would have to be stored in analog memories, which creates many implementation and performance difficulties, especially with advanced CMOS technology.

Because of these reasons, a more realistic alternative is to separate the imager array and the analog motion estimation processor. Even in this case the processor array cannot be practically designed with the same spatial resolution as a very large sensor. There are different ways to overcome this problem. The processor array can, that is, be implemented with the same number of columns than the image sensor, however, with only a limited number of rows. Another possibility is to implement the processor as a significantly smaller but symmetrical array, which is applied to the larger image frame in a windowed manner. Making a single processing cell as simple and small as possible is still crucial, since it enables the implementation of a larger processing window, reducing the number of required iterations for a large image size, and thus increasing the possible processing speed.

The actual motion estimation process performed by the array processor should be fast enough not to limit the achievable frame rate or frame size. The efficiency of the implementation is also heavily dependent on the speed of data transfer between the imager and the motion estimation processor, which means that the communication scheme should be carefully designed. The first requirement can be fairly easily achieved by using efficient analog current-mode signal processing. Because the analog image data from a sensor is always converted into digital form for further handling and storage, also the data communication with an external motion estimation core should be digital. This enables high-speed I/O operations and makes a separate analog motion estimation processor compatible with a system environment which is otherwise fully digital.

In a motion estimation processor with digital input, each cell of the AME array has to include two (typically 8-bit) digital to analog converters for providing data for the two frames to be compared, and the corresponding in-cell digital memory elements. The digital I/O for the processor is heavily asymmetrical; the only output required from the AME processor is digital motion vector data, that is, the identification of the shift location which results in the smallest block difference. The actual image data does not have to be read out of the processor array. The motion estimation circuitry does not have to have, nor should have, any direct effect on the image data itself. This will prevent additional image errors due to inevitable inaccuracies in analog operation.

3. Shifting of Reference Data

In principle, the switching operation could be performed by moving the pixel values step-by-step through only first neighborhoods connections. However, this would require current memories for intermediate storage, if implemented

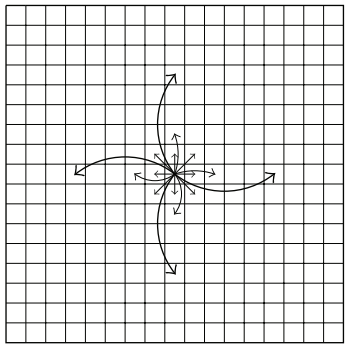

Figure 2: Cell neighborhood connections. Because the connections are bidirectional, the number of actual wires per cell is only 8.

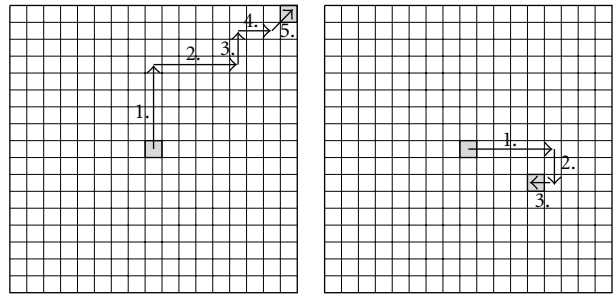

Figure 3: (a) Cell neighborhood connections. Example of (b) [8,8] shift, (c) [4,−2] shift.

in an analog fashion. The large number of sequential current memory read/write operations may slow down the shift operation and cause additional inaccuracy, and can potentially lead to higher power consumption from increased control signal activity. The proposed shift method also allows the efficient use of possible optimized search patterns, in addition to an exhaustive full search. Shifting the values cell-by-cell would make this much more inefficient.

The shift operation in a massively parallel array could also be performed in a fully digital manner, solving the problems of interconnect complexity and analog inaccuracies. On the other hand, this may lead to many new design challenges, that is, in terms of circuit complexity, power consumption and the implementation of the actual in-cell processing. However, the prospect is a very interesting direction for future research.

Figure 2 shows the neighborhood connections available in the network. Each connection between cells operates bidirectionally and is shared between two cells; the actual number of physical wires per cell is only half of the number of direct neighborhood connections. The choice between input and output operations for each direction is implemented with switches and logic inside the cell.

Because neighborhood connections to the 2nd and 5th neighbors are only available in the cardinal directions (N, E, S, and W), the shifts to the diagonal directions are implemented by using the same neighborhood twice in the same shift operation. The principle of the *double shift* is that first a connection to either N or S is used, after which the input to the cell is fed directly to the E or W connection of the same neighborhood. After this, the signal can either be taken into the target cell or to a lower neighborhood, from 5th to 2nd or from 2nd to the 1st neighborhood. By combining effectively 8-connected 5th and 2nd neighborhoods with an actually 8-connected local neighborhood, all cells within an 8-cell search area can be accessed (from 1 to 5 + 2 + 1). Because the output directions in the different neighborhoods can be controlled individually, that is, a shift with a length of 4 can be implemented simply by moving into the opposite direction in the lower neighborhood: East(4) = East(5) − West(1). Figure 3 shows two examples of shift operations with the proposed connectivity.

The approach proposed here significantly simplifies the connectivity in the array, compared to the previously proposed methods [5, 6]. The number of neighborhood connections per cell is now only 16, as opposed to 30 [5]. Although the number of separate cell connections required for some shifts is now 5 compared to a previous maximum of 3, all shifts are still implemented directly in one step, without having to store any pixel values in intermediate cells. The hierarchically implemented shift procedure is very straightforward and simple to control, requiring roughly 20 global control signals, which could be reduced by including in-cell control signal decoding. All controls could also be generated in-cell with a dedicated state-machine, however, in that case the cell complexity and area would be greatly increased. The metal pitch in current CMOS technologies is very small, which means that the number of global wires required for the proposed circuitry can be easily routed even over a fairly small cell size.

The layout design complexity is also greatly reduced with the proposed shift network, because of the fewer intercell connections and a fully symmetrical wiring arrangement; in [5] the connections were asymmetrical, which makes the layout design very complicated. In this case, since all connections are bidirectional, the number of individual neighborhood wires that have to be implemented for each cell is only 8 and the rest of the connections are realized automatically through symmetry.

3.1. Shift Configuration. The switch configuration for a single cell, used for the shift operation, is shown in Figure 4. The local input to the cell is provided by a current-mode Current-frame DAC (C-DAC) whereas the Reference-frame DAC (R-DAC) provides the output value of the cell which is shifted through the network. The local C-DAC current value is subtracted from the shifted R-DAC output, propagated from the source cell of the shift, and the current difference is applied to the ABS + QUAD block, which is implemented with very simple analog current-mode circuitry.

During the shift operation, the output current of the R-DAC is lead directly through a series of simple NMOS-transistor switches to the target cell. The simplified control signal configuration for the shift is as follows.

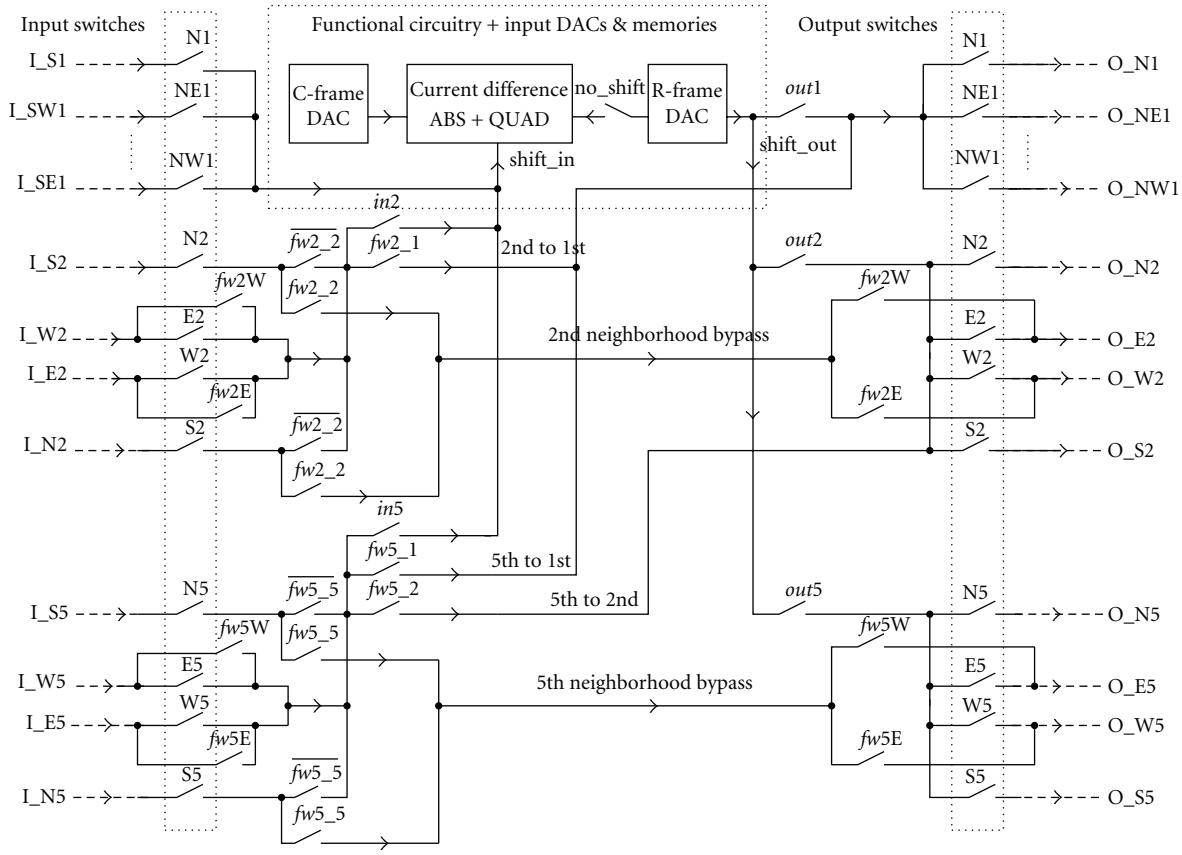

Figure 4: Cell switch configuration for the reference shift. All switches are implemented with NMOS transistors.

3.1.1. Selection of the Correct Output Neighborhood and Direction. Global control signals *out*1, *out*2, and *out*5 are used to select the neighborhood to which the R-DAC output current is propagated. The direction controls are implemented with 3 bits for the first neighborhood and with 2 bits each for the 2nd and 5th neighborhoods; a single output/input switch in the simplified schematic of Figure 4 is actually implemented either as 3 or 2 NMOS transistor switches in series. The control signal *noshift* is used to implement a [0,0] shift.

3.1.2. Selection of Propagation to a Lower Neighborhood. Global signals $fw5_1$, $fw5_2$, and $fw2_1$ are used for moving hierarchically to a lower (closer) cell neighborhood, in order to implement all necessary propagation paths. From the 5th neighborhood the signal can be propagated either to the 2nd or 1st neighborhoods and the 8-connected 1st neighborhood can be reached from the 2nd neighborhood.

3.1.3. Selection of the Direction of Secondary Propagation. In the 2nd or 5th neighborhoods, two propagation directions can be used at the same time. When the signal is propagated either to North or South, another wire in the same neighborhood can be used, either to East or West. The local control signals $fw2_2$ and $fw5_5$ in the cell are implemented as $OR(fwx_E, fwx_W)$. This means that if neither secondary direction (E/W) is globally enabled, the secondary connection will not be used (e.g., $fw2_2 = LOW$), and the first 2nd or 5th neighborhood connection (N/S) has to be directed either to the input of the target cell or to a lower neighborhood.

3.1.4. Selection of the Input Neighborhood. The neighborhood which provides the input to the target cell is selected with the global signals *in*2 and *in*5. Input switches are not required for the 1st neighborhood, because if a signal is applied to any 1st neighborhood output wire, it is always taken directly to the input of the neighboring target cell; propagation to an upper hierarchy level is not possible. Separate input and output direction switches are still required because the cell interconnect wires are used bidirectionally. The controls for the input direction switches are hardwired opposite to the output switches, so that each neighborhood wire can only be accessed by a cell in one direction at a time.

3.2. Shift Network Complexity. The cell circuitry required for the shifting consists of approximately 130 transistors, of which roughly 100 are NMOS-type switches, while the others account for additional inverters and logic within the cell. The complexity of the cell circuitry is reduced, for example, by implementing the shift-direction decoding directly with the switches used for the shift operation itself,

instead of using separate decoder circuitry. The realized shift circuitry is rather compact, however in future research and implementations some additional optimization may still be possible.

The complexity of the cell circuitry could be further reduced by separating the output and input wires used for the shift. If each neighborhood connection wire was made one-directional (input/output), input direction selection would not be required and a part of the switches could be omitted. This would reduce circuit area and the resistive effects discussed later, however, the neighborhood wiring complexity would be greatly increased because the number of physical wires would be doubled. In this case, simple interconnect wiring was targeted. Also, the area requirements of additional wiring may counteract some of the area savings from a reduced number of transistors.

A compromise between the number of switches and interconnect layout complexity could be reached by implementing only the first neighborhood connections with separate input/output wiring. This would reduce the number of transistors but would not require doubling the number of long interconnects, which have to pass over other cells, thus limiting the additional layout design more complexity.

4. Other Cell Circuitry

In addition to the shift network, each cell of the array includes the C-frame and R-frame DACs, which are NMOS-type 8-bit current mode binary-weighted converters, 16 static digital memory elements for storing the DAC input codes and the actual analog processing circuitry. The processing circuitry consists of a current-mode absolute value circuit followed by a current squarer circuit. This processing circuitry is effectively the same as in an earlier proposed AME designs [7, 8], however, the cell circuitry has been optimized for the new array design, which does not include current memories and in-cell current averaging. After the fairly simple in-cell analog processing, the summing of the cell outputs, within variable-sized macroblocks, has to be performed, and the sums for different macroblock locations have to be compared to find the best matching shift vector. In the current test chip design this is done with separate processing outside the chip.

4.1. Reference Source DAC Swapping. During the motion estimation procedure for a continuous stream of frames, after the motion vector for a frame has been determined, the reference frame will typically become the current frame for the next motion estimation step. In the AME array, where the input images are provided by in-cell DACs, the input data for the next C-frame is already stored in the cell as the R-frame for the previous operation. It is therefore desirable to use that frame data instead of writing both C-frame and R-frame data into each cell of the array for every frame of the image stream.

Because both the C-frame and the R-frame are stored in static digital memory registers inside the cell, the R-frame register could be simply written into the C-frame register. However, it is easier and more power-efficient to simply swap the outputs of the two in-cell DACs in every successive motion estimation step. Because current-output DACs and current-mode processing are used, the output current of a DAC can be simply redirected through a switch either to the local difference block (used as C-DAC) or to the shift network (used as R-DAC). This way only the reference frame data has to be written into the cells in each motion estimation step, and power and time is saved during the read-in phase of the processor array.

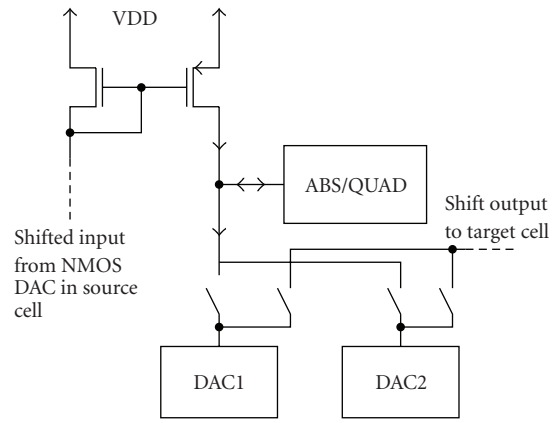

Figure 5: Input configuration for current shift with DAC swapping.

The benefit of the DAC swapping is only truly realized if a full image-sized processor array is implemented, that is, the whole image is processed at the same time. However, it maybe also be somewhat useful for I/O optimization in a windowed operation with a relatively large window (array) size, compared to the complete image, so that the swapping can be used for the last processed window of the image, to begin a new sweep with the existing DAC data. The swapping of the DACs and the cell input configuration are illustrated in Figure 5.

4.2. Absolute Value and Quadrature Operation. Figure 6 shows the actual processing circuitry in each cell of the array, along with the transistor sizes. The circuitry consists of a current-mode absolute value circuit and a current squarer. Some additional switches have been added to make the cell operation more flexible, so that either absolute value or quadrature output can be used for the cells.

The difference between the shifted reference frame pixel value and the local current-frame pixel value is realized at the input of the absolute value (ABS) block as a simple current subtraction. The absolute value circuit is implemented as proposed in [10]. Depending on whether the input current to the circuit is positive or negative (towards or away from the input node), the input voltage will be driven either higher or lower. The input voltage swing is amplified by the inverter, which is connected between the input node and the gates of the NMOS and PMOS transistors at the input of the ABS-clock. The inverter helps to efficiently close the unwanted and open the correct current path (direction) through the rectifier. This results in reduced voltage variation at the input node of the rectifier and thus improved performance,

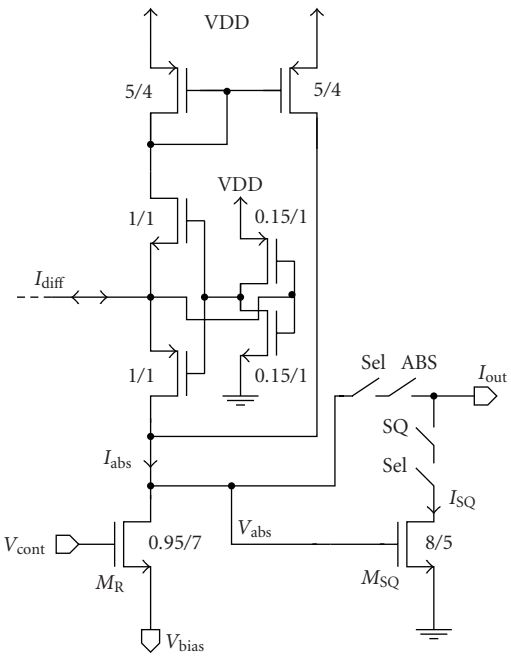

Figure 6: Absolute value and quadrature circuitry. The applied transistor dimensions (W/L) are in micrometers.

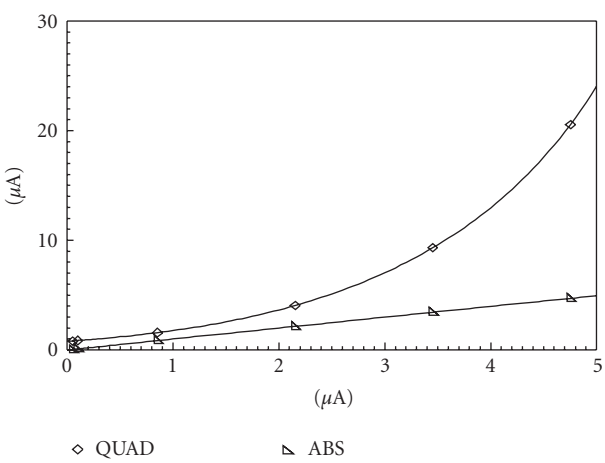

Figure 7: Simulated absolute value and quadratic responses of the cell.

compared to simpler rectifier circuits [11]. The addition of the inverter adds some extra complexity and power consumption, however, in this case the overall circuitry is so simple that the performance advantage is more significant.

When the current to the absolute value circuit is zero, the input voltage is somewhere in the middle of the power rails and a race situation is created in the inverter, leading to static current consumption. The magnitude of this current is limited in the cell to <1 μA by making the inverter transistors long and very narrow as can be seen from Figure 6.

The output of the absolute value circuit is taken to an NMOS transistor M_R which is effectively operating as a resistor. In normal operation, the gate voltage of transistor M_R is set to VDD and the voltage over M_R is relatively small. Therefore, the transistor is operating in the triode region as an approximately linear resistor. The source bias voltage of M_R can be adjusted in order to set the correct input range for the subsequent squaring transistor M_{SQ}.

The squaring transistor M_{SQ} takes as its input the approximately linear output voltage V_{abs}. The transistor is biased so that it is operating in saturation and thus provides an output current which is approximately quadratic with respect to the input voltage:

$$I_{SQ} \approx \frac{\beta}{2}\left(V_{abs} - V_{TN}\right)^2. \tag{1}$$

It has to be noted that the squaring is only approximate and the accuracy is also affected by the inevitable nonlinearity of the ABS output. Also, the transistor M_{SQ} is, for layout reasons, actually implemented as two $W/L = 8/2.5\,\mu$m NMOS devices in series. This has only a very small effect on the quadratic output response of the cell, as opposed to using a single transistor. It has been shown that an exact quadrature operation is not really necessary, nor even the most optimal solution [12]. Figure 7 shows the simulated responses of the absolute value circuit and the quadrature transistor when the input current was swept from 0 to 5 μA, which can be considered to be a suitable signal range for the circuitry.

4.3. SAD/SSD Operation. The cell circuitry shown in Figure 6 can provide two different output values. When V_{cont} = VDD, switch ABS is turned off and SQ is conducting, the output of the cell will be the squared response to the input difference. If $V_{cont} = 0$ and ABS is conducting with SQ turned off, the output current of the absolute value circuit will be directed to the cell output. This means that either the absolute value or quadratic output current can be read out from the same cell output node and either SAD or SSD operation can be selected and tested. The motion estimation process can be described as the minimization of the equation

$$D_N(dx, dy) = \sum_{i=x,\, j=y}^{x+N_v,\, y+N_h} \left|\mathrm{ref}\,(i + dx, j + dy) - \mathrm{cur}\,(i, j)\right|^p, \tag{2}$$

where $N_v x N_h$ is the macroblock size, (dx, dy) the applied shift vector for the reference frame, (x, y) the top-left pixel of the macroblock and p depends on whether SAD ($p = 1$) or SSD ($p = 2$) is applied.

The summing of the cell currents within the macroblock required for SAD/SSD is simply realized by connecting all of the output nodes of selected cells to a global output wire. Each cell receives a row and a column signal which are used to create the cell select (*Sel*) signal. The cells to be summed together, that is, ones belonging to the same macroblock, are selected with peripheral row/column decoder circuitry, which is capable of addressing several rows/columns at a time in a programmable fashion. The sum currents from different macroblocks can then be evaluated, for example, with an ADC, and compared, either one block at a time in a sequential manner or with fully parallel comparison circuitry.

The implementation of the comparison circuitry has to be carefully optimized in order to reach the best possible performance. The integration of the evaluation circuitry on the same chip as the array is a subject of further research; in the designed 32×32 test chip, the sum current is routed off-chip for measurement. However, because the SAD/SSD comparison, which is not a pixel-parallel operation, is not performed within the array itself, the cell circuitry can be kept very simple, allowing for a larger array size. Also, the measurement and evaluation circuitry can be optimized independently of the cell array, making the design more flexible and efficient.

Because the two outputs (ABS/SQ) result from different types of sources (PMOS/NMOS, resp.) also the cell selection switches (*Sel*) were implemented separately for both current paths. This allowed the use of the correct type of devices, in order to minimize the effect of the switches on the output current value. The different current polarities naturally also have to be accounted for in the measurement circuitry. In the implemented chip the SAD/SSD evaluations will be performed off-chip, however, the circuitry could also be integrated in the periphery of the AME array.

5. Implementation Issues

The circuit operation was simulated at the transistor level with a 9×9 cell array. A 0.13 μm standard CMOS technology was used with VDD = 1.2 V. A potentially difficult design issue in the proposed method is that the shifting of the reference value as a current signal makes the implementation vulnerable to resistive drops in the large number of series-connected switches used for the [8,8] neighborhood connectivity. The resistive effects can lead to deterministic shift-dependent offsets, which may cause errors in the motion estimation process.

5.1. Resistive Effects. The analog ABS/QUAD circuitry receives as input the difference between two current-mode DAC outputs. The applied current-mode signaling may lead to resistive distance-dependent offset in the difference extraction. A large current value causes a significant resistive voltage drop over the shift switches, which are in series between the R-DAC in the source cell and the diode-connected input transistor of the PMOS current mirror in the target cell. This lowers the output voltage of the R-DAC, which leads to current variations due to channel length modulation in the DAC transistors, in the worst case the DAC transistors may even start to come out of saturation. The channel length modulation can be mitigated to some extent by using long-channel transistors in the DAC, in this case $L = 4.88\,\mu$m was used for the unit current source in the DACs.

In order to simplify the design, the two DACs should be of the same type (NMOS/PMOS), which means that one of the inputs has to be mirrored in order to perform the subtraction. The type of the switches used (NMOS/PMOS) is dependent on the DAC-type and on the current mirror configuration. Simulations showed that PMOS switch transistors of a reasonable size exhibited considerable resistive voltage drops due to their low conductance. Because this would lead to large offset errors which depend on the shift distance, NMOS transistor switches were selected. This leads to a cell configuration where the output current of an NMOS-type R-DAC is propagated through (NMOS) switches and mirrored with a PMOS current mirror in the target cell as shown in Figure 5. The larger conductance of the NMOS switch transistors resulted in a much reduced distance effect, compared to PMOS switches.

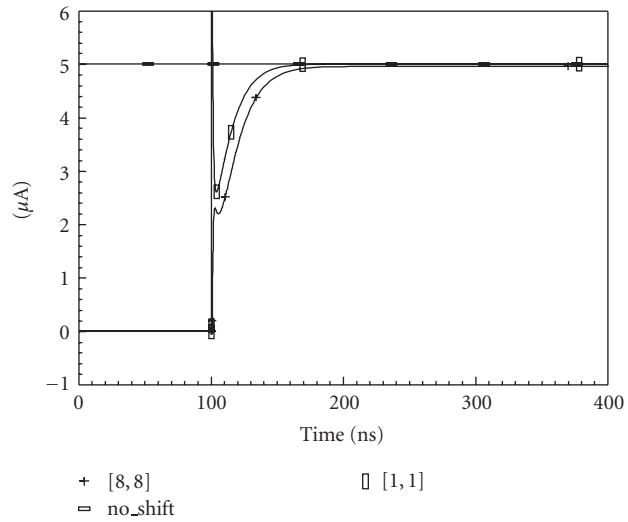

Figure 8: Transient simulations of [0,0], [1,1], and [8,8] shifts, with high-speed transistor switches.

The selection of the correct transistor type within the applied CMOS process can lead to considerable benefits in terms of both performance and cell area. Using lower-threshold and higher conductivity high-speed (HS) transistors available in the CMOS technology, enables the use of smaller switches, leading to more compact cell circuitry. The larger leakage current in HS transistors is not a serious problem in this case because of the many switches in series in the shift network. Also, the voltage differences over nonconducting switches between cells in the array are relatively small, because the output current of each DAC in the array is always taken into the same resistive load (i.e., diode-connected transistor).

Figure 8 shows a transient simulation comparing three different shift operations: $[dx, dy]$ = [8,8], [1,1], and [0,0] (no shift), the applied current magnitude was $5\,\mu$A which is specified to be the maximum input current for the cell circuitry. The output of the R-DAC in the source cell was constant and the switch to the output neighborhood was turned on at 100 nanoseconds. These cases correspond to different numbers of switches present in the current path, that is, different resistive voltage drops. The shift network was implemented with high-speed (HS) NMOS transistors, with $W = 0.5\,\mu$m and $L = 0.13\,\mu$m. The two extrema are [0,0] with only 2 switches and [8,8] with more than 20 switches in the current propagation path. The simulated difference in the DAC output current between theses cases

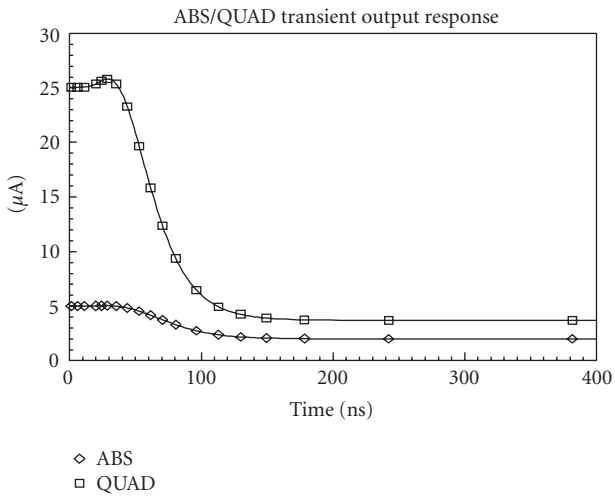

Figure 9: Transient settling of the cell output for ABS and QUAD operations.

is approximately 40 nA, with low-leakage transistors of the same size the output difference would be more than doubled.

The resistive loss in the switches should not be a limiting design issue for the proposed circuitry, since even with the maximum input value, the difference in the shifted currents is small. However, extensive simulations and testing with natural image streams on the manufactured array is necessary to verify if further optimization of the shift operation is necessary. The distance effects could be mitigated, for example, by applying cascode techniques to the DACs, to increase output resistance, or by taking the shift distance into account in the SAD/SSD evaluation phase. Other common-mode nonlinearity effects caused by the analog processing circuitry, which are not dependent on shift distance, should not be critical to the AME operation if they do not affect the ordering in the following macroblock comparison.

5.2. Mismatch Effects. Mismatch between the analog transistors in the processor cell is the most significant source of errors in the proposed motion estimation architecture. The achieved accuracy is proportional to the area of a device [13], therefore the circuitry should be made as simple as possible and, for example, unnecessary current mirroring operations should be avoided. In the shift operation, the current is only mirrored once through the local PMOS mirror. Another current mirror is required for the absolute value circuit. Because of the small number of devices, the mirror transistors can be relatively large, in the realized test chip devices with $W/L = 5/4\,\mu$m were used.

The mismatch variation in the different parts of the cell circuitry was simulated with a Monte Carlo simulator, with mismatch models provided by the manufacturer. The simulated output standard deviation of the input DAC at the full signal output of $5\,\mu$A was approximately 0.5%. The simulated relative output standard deviation of the absolute value current, without input mismatch, with the maximum input was approximately 0.6%. The squaring operation introduces additional mismatch variation. The simulated standard deviation in the quadrature output, which also includes the absolute value variation, but not input signal mismatch, was approximately 2.3%.

Fixed-pattern noise in the input image, due to the input DACs could, at least to some extent, be corrected by adjusting (predistorting) the digital input codes for different cells. The subsequent current summing operation within the whole macroblock, which results in the actual evaluated signal from the array for a given shift operation, leads to averaging of the individual cell output errors. The averaging is naturally more prominent for a larger block size. The total effect of the mismatch variation on the complete motion estimation operation is a very complex issue, since the actual realized inaccuracy is totally signal- and image-dependent. System level simulations and measurements with realistic image sizes and a real-world video streams are required to accurately characterize the performance of the AME array. This is not within the scope of this paper, but will be addressed in further research, with the help of the designed test array.

Because the analog circuitry inside each cell is very simple, the transistors can be made fairly large. However, minimal transistor area should still be targeted in order to maximize the spatial resolution or to minimize the area of the processor array. Also, because the whole image cannot be practically processed with the array at the same time, also the speed of operation should be maximized by targeting the smallest possible capacitive loads, that is, smallest possible transistors. In this respect the application of area-efficient mismatch compensation techniques in the AME cell, such as the one discussed in [14], should be considered.

5.3. Speed and Other Performance Issues. The delay in the shift operation can be observed from Figure 8. It can be noticed that the maximum delay, in the [8,8] shift, is approximately 80 nanoseconds. This delay is defined by the resistance of the shift switches and the capacitance from the current mirror in the target cell, which had a transistor size of $5/4\,\mu$m in the final design. The settling time of the analog absolute value and squaring circuitry is shown in Figure 9, where the input difference was changed from $5\,\mu$A to $2\,\mu$A at time zero. It can be noticed that the outputs settle to their new values in less than 200 nanoseconds. In practical operation also the delay of the evaluation circuitry and I/O has to be considered, however, it can be seen that the switched-current cell operation is relatively fast. The effects of device mismatch on the delay are negligible.

The inclusion of in-cell DACs for providing the input frames also leads to additional benefits related to the analog circuit implementation. Because the input currents are always provided by active DACs and no dynamic current memories are used, effects such as charge injection and memory retention problems due to leakage are not an issue, since the input values are static and robust. An important issue to be considered in further research, in terms of performance, is the optimal implementation of the array I/O and especially the writing of the in-cell DACs, so that a large image frame can be processed fast enough and with sufficiently low power consumption.

6. AME Test Array

A 32 × 32 cell test array was designed in the 0.13 μm 6 M digital CMOS technology with the high-speed transistor option, for evaluating the performance of the proposed motion estimation approach in practice. The chip enables the evaluation of the motion estimation operations within a [8,8] search range and a 16 × 16 maximum block size. The size of the actual active array is 16 × 16 cells, an 8-cell wide boundary is required for providing input values to the 16×16 cell active area, with a maximum shift distance of 8 cells. In practice the analog processing circuitry in the boundary cells is unnecessary, however, in the test chip layout design it was simpler to just use basically the same cell for the boundary, although with some wires and controls disconnected. Also, because the DACs (+memories) and the shift network take up most of the cell area, as can be seen from Figure 11, leaving out the processing circuitry would not have lead to significant area savings.

The layout of the array is shown in Figure 10. The layout of a single array cell is shown in Figure 11, with the different functional sections of the cell highlighted. The size of the chip is approximately 1.5 × 1.7 mm^2 and the size of a single cell is approximately 30 × 35 μm^2. The periphery of the array includes row-wise buffers for the global control signals and address decoders which enable the simultaneous selection of multiple rows/columns for different block-size summation operations. The sum current to be evaluated is available from a global wire, which is only connected to the outputs of the active 16 × 16 cell array and taken to a chip output pad for external evaluation.

The digital control for the array as well as the evaluation and comparison of the SAD/SSD results will be initially realized with an additional FPGA chip and an off-chip ADC. For a complete motion estimation processor realization also these operations have to be optimized and implemented with dedicated on-chip circuitry for optimal performance. However, at this stage a more thorough examination of the analog cell array through chip measurements is required to validate the feasibility of the approach, for example, in terms of speed and accuracy, and to derive more exact specifications for the remaining hardware and the whole system. Many design choices, such as the practical array size are still open to optimization.

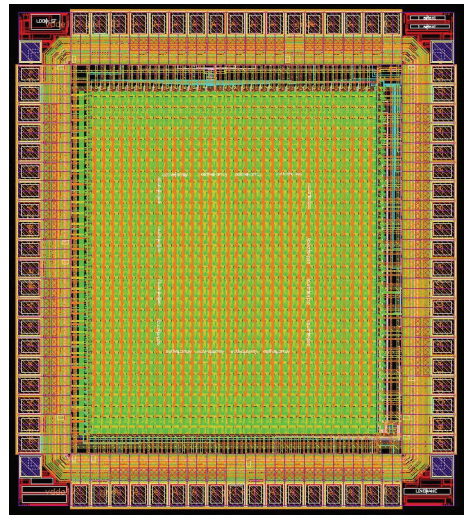

Figure 10: Layout of the 32 × 32 AME processor array. The active processing area of 16 × 16 cells is highlighted in the middle or the array.

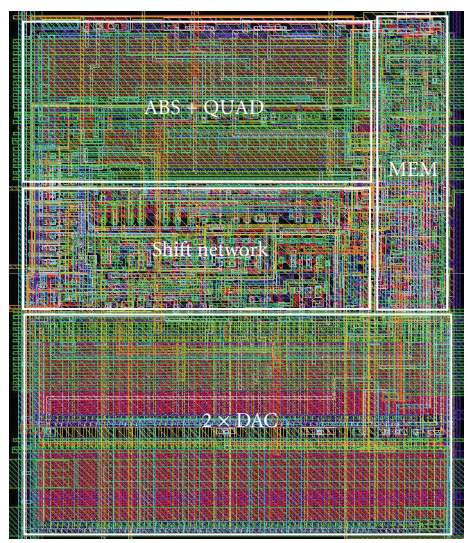

Figure 11: Layout of a single array cell with different functional sections highlighted.

6.1. Related Work. Since the complete motion estimation system has not been realized at this time, it is difficult to directly compare the performance of the proposed circuitry to other implementations. Also, the total system performance is a compromise between multiple factors, such as picture quality, bitrate, power consumption, and cost (i.e., silicon area). The performance of the underlying analog processing hardware has to be first evaluated in detail with measurements before making quantitative comparisons to other implementations. For example, the practical accuracy and robustness of the proposed analog processing, which is difficult to examine comprehensively with simulations, also affects the choice of the ME algorithm. The choice between implementing a full search operation and for example, a gradient descent search (GDS) algorithm, has a very large effect on the number of required pixel operations, that is, also on the power consumption. However, the parameters reported from ME implementations in the field are briefly reviewed to estimate the performance that should be targeted and improved upon.

Few separate ME realizations have been presented. Current ME realizations are typically embedded within full audio-video codecs. Comparing the presented work to such implementations is difficult due to the fact that little specific information (such as power consumption) on the ME part is available. The most comprehensively reported motion estimation implementations have been digital chips. In [15], a 0.4 mW (QCIF@15 fps, 0.85 MHz)/2.5 mW

(CIF@30 fps, 6.75 MHz) motion estimation IC, in 0.18 μm technology with a 1.0 V power supply, using a gradient descent search algorithm was presented. As the design does not incorporate frame memory the stated power consumption figures do not include the data transfer between the frame memory and local search area memories. In [15], it is also estimated that the power consumption of a Full Search QCIF@15 fps ME IC would be in the range of 20 mW. In [16], a 16.2 mW QCIF@15 fps motion estimation IC using pixel wordlength truncation was presented. The chip has a 20 MHz clock frequency (the operating voltage was not stated) and is implemented with 0.18 μm technology. The design incorporates frame memories whose portion of the power consumption is 11.7 mW. In [17], a 1920 $\times$ 1080 HDTV@30 fps ME core was presented. The ME core is designed for a power supply of 1.0 V and a clock frequency of 81 MHz and is implemented with 0.13 μm technology. The estimated power consumption is 65 mW without frame memories.

For H.264 [18] presents a 720$\times$480 SVGA@30 fps systolic array design that incorporates Full Search for the seven different block sizes of H.264. With 0.35 μm technology and a clock frequency of 67 MHz (the operating voltage is not stated) the design has a simulated power consumption of 737 mW without frame memories. In [19], a QCIF@15 fps implementation using variable block-size Full Search is presented. With 0.13 μm technology, a clock frequency of 6.7 MHz, and an operating voltage of 1.2 V the design has a simulated power consumption of 9.1 mW without frame memories. Both of these variable block-size ME implementations operate by computing the distortion measure values for the smallest block size and then combining these values to form the distortion measures for the larger block sizes. Also, neither of these designs comments on the choice of the optimal block size.

In other proposed block-based analog motion estimation approaches [2, 3], although the computation, memory, and data transfer are analog, the architectures resemble conventional digital ME processors. This is in contrast to the proposed work which proposes an interconnected analog parallel processor architecture. In both [2, 3], only the picture quality results are presented which, without presenting the effect on bit-rate, is not fully meaningful and makes comparison to other implementations difficult. For the CNN-based image stabilization architecture presented in [4] no power consumption or processing figures were given. Additionally, the effect of error in the ME distortion measure has been studied in [20, 21].

At the time of writing the designed 32 $\times$ 32 cell chip is being manufactured and measurement results, further analysis and comparison to the other implementations, based on experimental results, will be presented in future publications.

7. Conclusion

This paper presented an analog motion estimation array with a new cell neighborhood configuration and the required circuitry for reference image shifting. In the otherwise very simple AME cell architecture, the shift network is clearly the most complex aspect of the implementation. Compared to a previously proposed method, considerable savings in array level interconnect complexity are achieved. The new shift method thus allows for a more efficient implementation of the analog motion estimation array. Transistor level simulations show that the method and the related analog processing circuitry can be applied for high-speed operation and with sufficient accuracy. A realistic performance comparison with other proposed motion estimation circuit architectures can be achieved in future work, based on measurements of the implemented 32 $\times$ 32 test array.

Acknowledgment

This work has been supported by the Academy of Finland projects 107645 and 123354.

References

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

[2] A. Tomasini, M. Brattoli, E. Chioffi, G. Colli, D. Gerna, and M. Pasotti, "B/W adaptive image grabber with analog motion vector estimator at 0.3 GOPS," in *Proceedings of the 42nd IEEE International Solid-State Circuits Conference (ISSCC '96)*, pp. 94–95, San Francisco, Calif, USA, February 1996.

[3] M. Panovic and A. Demosthenous, "A low-power analog motion estimation processor for digital video coding," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 673–683, 2006.

[4] Y.-C. Cheng, J.-F. Chung, C.-T. Lin, and S.-C. Hsu, "Local motion estimation based on cellular neural network technology for image stabilization processing," in *Proceedings of the 9th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '05)*, pp. 286–289, Hsinchu, Tawian, May 2005.

[5] L. Koskinen, J. Marku, A. Paasio, and K. Halonen, "Architecture for analog variable block-size motion estimation," in *Proceedings of the 14th IEEE International Conference on Image Processing (ICIP '07)*, vol. 2, pp. 493–496, San Antonio, Tex, USA, September 2007.

[6] L. Koskinen, K. Halonen, and A. Paasio, "Efficient shift of reference data in analog motion estimation," in *Proceedings of the 9th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '05)*, pp. 130–133, Hsinchu, Tawian, May 2005.

[7] L. Koskinen, A. Paasio, and K. Halonen, "3-neighborhood motion estimation in CNN silicon architectures," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '04)*, vol. 5, pp. 708–711, Vancouver, Canada, May 2004.

[8] J. Marku, L. Koskinen, and A. Paasio, "A 130 nm implementation of analog variable block-size motion estimation cell," in *Proceedings of the International Symposium on Integrated Circuits (ISIC '07)*, pp. 57–60, Singapore, September 2007.

[9] J. Poikonen, M. Laiho, A. Paasio, L. Koskinen, and K. Halonen, "Interconnect-efficient reference data shift for optimized analog motion estimation," in *Proceedings of the 11th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '08)*, pp. 102–107, Santiago de Compostela, Spain, July 2008.

[10] A. Rodríguez-Vázquez, R. Domínguez-Castro, F. Medeiro, and M. Delgado-Restituto, "High resolution CMOS current comparators: design and applications to current-mode function generation," *Analog Integrated Circuits and Signal Processing*, vol. 7, no. 2, pp. 149–165, 1995.

[11] J. Poikonen and A. Paasio, "An area-efficient full-wave current rectifier for analog array processing," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '03)*, vol. 5, pp. 757–760, Bangkok, Thailand, May 2003.

[12] L. Koskinen, M. Laiho, A. Paasio, and K. Halonen, "Motion estimation matching criterion computation with analog circuits," in *Proceedings of the 8th International Workshop on Cellular Neural Networks and Their Applications (CNNA '04)*, pp. 216–221, Budapest, Hungary, July 2004.

[13] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers, "Matching properties of MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1439, 1989.

[14] J. Marku, K. Virtanen, J. Maunu, J. Poikonen, and A. Paasio, "Current mismatch and nonlinearity compensation in mixed-mode array processors," in *Proceedings of the 11th IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA '08)*, pp. 75–80, Santiago de Compostela, Spain, July 2008.

[15] M. Miyama, J. Miyakoshi, Y. Kuroda, K. Imamura, H. Hashimoto, and M. Yoshimoto, "A sub-mW MPEG-4 motion estimation processor core for mobile video application," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 9, pp. 1562–1570, 2004.

[16] C.-W. Yoon and H.-J. Yoo, "Low power motion estimation and motion compensation block IPs in MPEG-4 video codec hardware for portable applications," *IEICE Transactions on Electronics*, vol. E86-C, no. 4, pp. 553–560, 2003.

[17] M. Miyama, O. Tooyama, N. Takamatsu, et al., "An ultra low power motion estimation processor for MEPG2 HDTV resolution video," *IEICE Transactions on Electronics*, vol. E86-C, no. 4, pp. 561–569, 2003.

[18] Y.-W. Huang, T.-C. Wang, B.-Y. Hsieh, and L.-G. Chen, "Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '03)*, vol. 2, pp. 796–799, Bangkok, Thailand, May 2003.

[19] S. Y. Yap and J. V. Mccanny, "A VLSI architecture for advanced video coding motion estimation," in *Proceedings of the 14th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP '03)*, pp. 293–301, The Hague, The Netherlands, June 2003.

[20] M. Tartagni, A. Leone, A. Pirani, and R. Guerrieri, "A block-matching module for video compression," in *Proceedings of IEEE Symposium on Low Power Electronics*, pp. 24–25, San Diego, Calif, USA, October 1994.

[21] M. Panovic and A. Demosthenous, "Architectures for analog motion estimation processors: a comparison," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 5, pp. 4566–4569, Kobe, Japan, May 2005.