Neurocomputing 112 (2013) 189–199

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A reconfigurable neuroprocessor for self-organizing feature maps

J. Lachmair [a,*], E. Merényi [b], M. Porrmann [a], U. Rückert [a]

[a] Cognitronics and Sensor Systems, Bielefeld University, Universitätsstrasse 21-23, 33615 Bielefeld, Germany
[b] Department of Statistics, Rice University, MS 138, 6100 Main Street, Houston, TX, USA

ARTICLE INFO

Available online 4 March 2013

Keywords:
Self-organizing feature maps
FPGA
Hardware accelerator
Hyperspectral data

ABSTRACT

In this paper we compare a scalable FPGA-based hardware accelerator for the emulation of Self-Organizing Feature Maps (SOMs) with a multi-threaded software implementation on a state-of-the-art multi-core microprocessor. After discussing the mapping of SOMs to the reconfigurable digital hardware implementation, we present how the modular system architecture can be flexibly adapted to various application datasets as well as to variants of SOMs like Conscience SOM. Hyperspectral image processing is used as a benchmark scenario for the comparison of our FPGA-based hardware accelerator and state-of-the-art multi-core microprocessors. The hardware costs, power consumption, and scalability of the FPGA-based accelerator using Xilinx Virtex-4 FPGAs are discussed. For the real-world datasets used here, which require large SOMs, a speedup and energy reduction of one order of magnitude are achieved.

1. Introduction

Since SOMs were first proposed by Kohonen [1], they have become an important tool in analyzing high-dimensional data such as those in hyperspectral imaging tasks [2,3] or medical research [4]. Following the principle of operation of the human neocortex, a SOM is able to generate a spatially ordered map of similarity groups from input space in an unsupervised learning process. A major advantage of this topology-preserving learning is that it works well for high-dimensional data, and it provides unique visualization possibilities. The underlying SOM knowledge can facilitate detailed and precise extraction of clusters, either through interactive visualization or with automated approaches ([5] and references therein). SOM learning, however, is computationally expensive. Although parallelism is increasing in today's microprocessor architectures, the software-based simulation of large SOMs with high-dimensional weight vectors still suffers mainly from sequential processing.

When targeting the analysis of high-dimensional datasets in energy-restricted environments like satellites or mobile robots, dedicated yet flexible hardware implementations are necessary to achieve the required real-time performance. The hardware accelerator proposed in this paper is based on the single-FPGA neuroprocessor described in [6] and extends this architecture toward a flexible multi-FPGA-based SOM accelerator. In addition to an enhanced scalability, leading to a speed-up that increases almost linearly with the number of FPGAs, we extended the instruction set of our neuroprocessor to the simulation of Conscience SOMs (CSOMs) according to [7]. This is motivated by both the effectiveness of CSOMs for the analysis of large, high-dimensional datasets and their advantages for hardware implementation. In the following Section the CSOM algorithm and its realization in digital hardware will be introduced. Section 3 details the modular FPGA-based system architecture and its partitioning. Section 4 presents a large real-world application and the test environment for measuring performance in terms of execution time and power consumption. The scaling properties of our architecture in respect to the number of SOM neurons and data dimensionality will be discussed as well. Finally, the experimental results for the FPGA-based hardware accelerator are compared to a multi-threaded software implementation and selected hardware realizations developed by other researchers.

2. Conscience SOM implementation

The classical SOM algorithm causes a magnification of frequently presented input data in the map, which manifests in a disproportional number of SOM neurons allocated to their representation while the same magnification effect results in an under-representation of small clusters. Magnification, in general, is induced by vector quantization algorithms, which can be formally described as a power law relationship between the density of the trained SOM weights $D(\mathbf{w})$ and the density of the input data $P(\mathbf{x})$

* Corresponding author. Tel.: +49 521 106 67372.
E-mail addresses: jlachmair@cit-ec.uni-bielefeld.de (J. Lachmair), erzsebet@rice.edu (E. Merényi), mporrmann@cit-ec.uni-bielefeld.de (M. Porrmann), urueckert@cit-ec.uni-bielefeld.de (U. Rückert).

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2012.11.045

with $\mathbf{x} \in V \subset \mathbb{R}^d$ and $\mathbf{w} \in \mathbb{R}^d$: $D(\mathbf{w}) \propto P(\mathbf{x})^r$. In this relationship, r is the *magnification factor*, which is an inherent property of a given vector quantization algorithm [8,9]. To maximize the information representation in the trained SOM, maximum entropy mapping is desired, which equates to a magnification factor of 1. The classical (2-dimensional) SOM induces $r = 2/3$ [10]. The CSOM achieves a good approximation of a magnification factor of 1 by computationally inexpensive heuristics [7], and as shown in [11] this also works for higher dimensional data. The CSOM heuristics consist of adding a bias value B_i (the conscience) for calculation of the distance between input vectors $\mathbf{x}$ and $\mathbf{w}_i$ in the SOM winner selection (Eq. (1)), to discourage frequent winners from winning and encourage infrequent winners to win (thus eventually equalizing the winning frequencies, achieving maximum entropy mapping)

$$\|\mathbf{x}(t) - \mathbf{w}_c(t)\|_{p_V} - B_c(t) \le \|\mathbf{x}(t) - \mathbf{w}_i(t)\|_{p_V} - B_i(t) \; \forall i \ne c \tag{1}$$

$$B_i(t) = \gamma(t)\left(\frac{1}{N_N} - F_i(t)\right) \tag{2}$$

$$F_i(t+1) = F_i(t) + \beta(t)(y_i - F_i(t)) \tag{3}$$

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + h_{c,i}(t)[\mathbf{x}(t) - \mathbf{w}_i(t)] \tag{4}$$

$$h_{c,i}(t) = \begin{cases} \alpha(t) & \text{if } \|\mathbf{r}_c - \mathbf{r}_i\|_{p_{SOM}} \le \sqrt{p_{SOM}} \\ 0 & \text{if } \|\mathbf{r}_c - \mathbf{r}_i\|_{p_{SOM}} > \sqrt{p_{SOM}} \end{cases} \tag{5}$$

Here, $\mathbf{x}(t) \in V \subset \mathbb{R}^d$ is the vector from input space V presented to the SOM at time step t; d is the dimensionality of the input data. The parameter p_V determines what distance metric is used in the input space. Currently, the choices are Euclidean ($p_V = 2$) and Manhattan ($p_V = 1$). While the Manhattan distance is not a very sensible metric in the input space we provide it because it is implemented in the hardware for a choice in the computation of SOM lattice distances. The parameter c is the index of the winner neuron, N_N is the number of neurons in the SOM (static for classical SOM and CSOM) and F_i is the winning frequency of neuron i. The bias B_i (Eq. (2)) is calculated at each learning step. The bias depends on the user-specified parameter γ and the winning frequency F_i (Eq. (3)). The user-controlled parameters (α, β, γ) have to decrease during simulation in order to strengthen the already learned information. The parameter y_i is 1 for the winner and 0 for all other neurons. The distance metric applied in the map $M \subset \mathbb{R}^n$ is determined by the parameter $p_{SOM} = 1$ or 2 (Eq. (5)); n is the dimensionality of the map; $\mathbf{r} \in M$ is the neuron position in the map. In contrast to the classical SOM the CSOM only updates the immediate map neighbors. Thus, in the special case of CSOM, the distance metric in the map directly determines the number of updated neurons (i.e., in a rectangular SOM, for $n = 2$ and $p_{SOM} = 1$ the neighborhood is diamond-shaped (Manhattan distance) including 4 neighbors, while $p_{SOM} = 2$ defines a square neighborhood (Euclidean distance) including all 8 map neighbors).

An advantage when mapping the CSOM to our hardware is the similarity between the calculation of B_i and F_i to the weight update algorithm of the classical SOM. Hence, F_i and B_i can be easily calculated without the need for additional calculation units. During weight update the lattice distance between each neuron and the Best-Matching-Unit (BMU) in the map has to be calculated and the weights of the BMU ($\mathbf{w}_c$) as well as the weights of its neighbor neurons are adapted according to Eq. (4). Another advantage when using CSOM is the constant neighborhood function $h_{c,i}$ (Eq. (5)), which can be realized much easier in hardware than a Gaussian function, often used in classical SOMs. Hence, for the hardware implementation of the CSOM algorithm, the instruction set of our neuroprocessor only had to be extended to support bias and winning frequency calculation.

3. Modular system architecture

The modular system architecture of our hardware accelerator, called gNBXe, is specified in VHDL and optimized for Xilinx FPGAs (Fig. 1). It consists of a central control FPGA including the global controller (GC) and several FPGAs implementing the processing elements (PE-FPGAs). The processing elements (PEs) are the hardware units performing the calculations for simulating the SOM neurons. The calculation precision (Ξ), the number of processing elements per FPGA (N_{PE}), and the local memory space of each processing element can be flexibly chosen at design time using VHDL generic parameters. The high flexibility of the architecture is further increased by enabling each PE to emulate not just one SOM neuron but to perform the calculations for several neurons, thus facilitating the simulation of large maps. Fig. 2 depicts the stages of the training process when simulating a SOM with our hardware accelerator. After the FPGAs have been configured, the training dataset is sent to the global controller. Once the simulation process is started, all PE-FPGAs initialize their neuron weights either randomly or with a custom weight set. While training the SOM, the global controller distributes the input vectors to the PE-FPGAs, which calculate the distances between the input vectors and the neuron weights sequentially

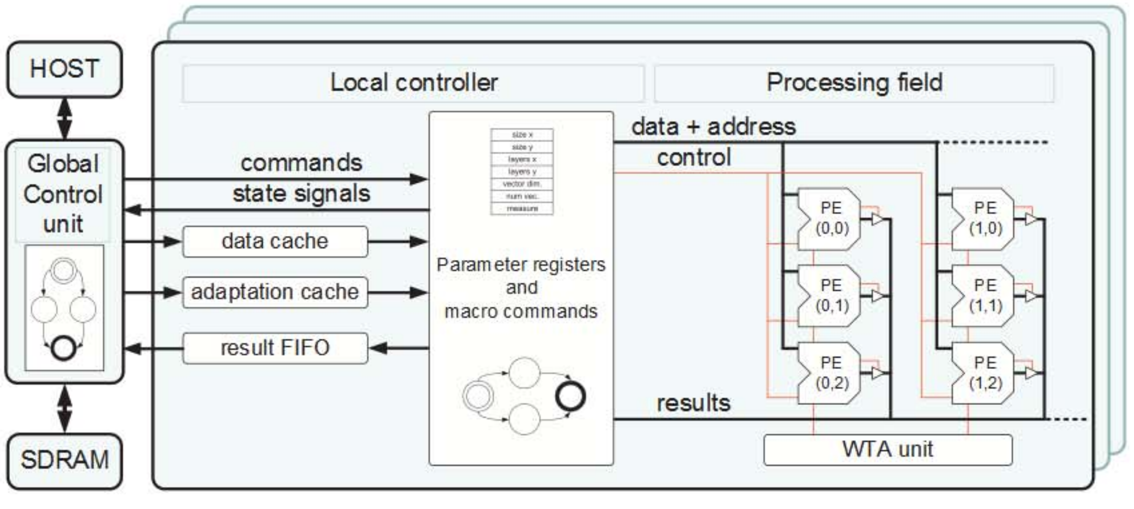

Fig. 1. (Color online) Principle of the modular system architecture.

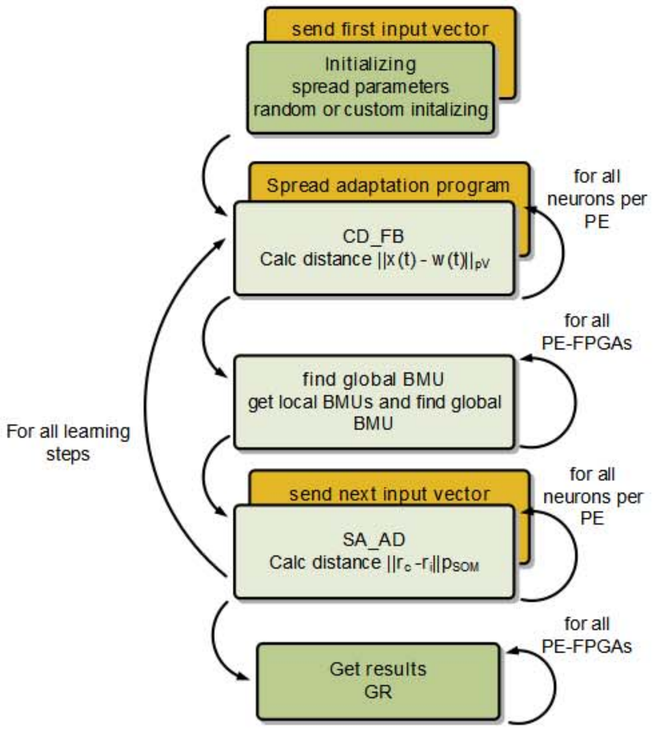

Fig. 2. (Color online) Stages of the training process.

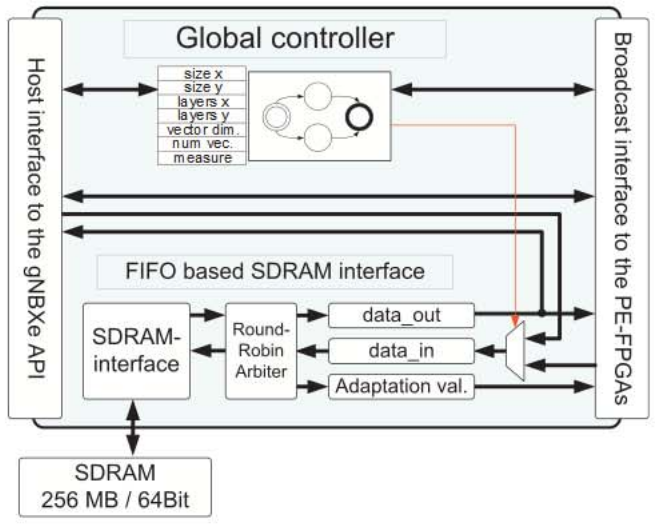

Fig. 3. (Color online) Architecture of the global controller.

for all components. To prevent unnecessary idle cycles of the PE-FPGAs, the first input vector is already sent to the PEs during initialization. When all required distances have been calculated, the best matching unit (BMU) is determined: all PE-FPGAs perform a local bit-serial search to identify their local BMU's and subsequently, the global BMU is determined by comparing the BMUs of the PE-FPGAs in the global controller. To adapt the neuron weights, the distance of the BMU to all neuron positions in the map is calculated and the weights of the neurons inside the neighborhood are adapted. When the number of pre-determined training iterations has been reached, the trained SOM is sent back to the global controller and can be fetched by the host system.

3.1. Global controller

The global controller (Fig. 3) connects the accelerator to the host system and manages the training and adaptation data. To ease the control logic inside the global controller, a memory management unit has been implemented, which is working independently from the control process itself. The training data is stored in the external SDRAM, directly attached to the global controller. In the current implementation, the entire dataset (used for SOM learning) is randomized in software before downloading it from the host PC to the SDRAM. Thus, the picking order of the data vectors is pre-determined according to a pseudo-random sequence. We do this in order to ensure that hardware and software processing performs the exact same computation except for the difference in precision, and so we can meaningfully compare the outcomes of the software vs. hardware learning. Once initialized, FIFOs organize the next needed data ($\mathbf{x}(t+1)$, $h_{c,i}(t+1)$, $\mathrm{b}(t+1)$, and $\mathrm{g}(t+1)$) and make it available to be sent to the PE-FPGAs via broadcast communication. In cases of a multi-FPGA system architecture, utilizing more than one PE-FPGA, the global controller also compares the local BMUs from each PE-FPGA, finds the global BMU, and sends the information back to the PEs for adaptation.

3.2. Processing elements

The PE-FPGAs consist of a local controller, parallel processing elements, and a winner takes all (WTA) unit determining the local BMU.

The local controller translates the macro commands from the global controller to control signals for the PEs. The availability of training and adaptation data is managed using additional cache structures. Fig. 4 depicts the architecture of the gNBXe processing elements. The data path of the gNBXe-PE is realized in a five-stage pipeline including additional pipeline bypasses for increased performance.

To efficiently realize the Euclidean norm, each PE integrates an embedded multiplier of the Xilinx FPGAs. Since there is no need to calculate the root function for comparing the distance values, the root function has not been implemented. We compare, as many software implementations do, squared Euclidean distances. As mentioned above, one design goal was high flexibility with respect to the number of available neurons. Therefore, each processing element is capable of performing the calculations for multiple neurons. The number of neurons that can be emulated by a single PE is only limited by the available memory. Each neuron is represented by an individual address (the position in the n-dimensional map) and its neuron weights. Both parts are stored in the local address space of a PE, requiring $d+\mathrm{n}$ addresses. For CSOM each neuron needs an additional address to store its winning frequency F_i. Therefore, for each PE with a local address space of k addresses, up to $N_{N_{PE}} = \lfloor k/(d+\mathrm{n}) \rfloor$ neurons can be stored and calculated sequentially for SOM simulation and $N_{N_{PE}} = \lfloor k/(d+\mathrm{n}+1) \rfloor$ neurons for CSOM. Thus, each PE represents $N_{N_{PE}}$ neurons, dividing the complete SOM into $N_{N_{PE}}$ sub-maps (in the following called virtual layers), which are calculated sequentially.

3.2.1. Winner-takes-all-unit

To handle multiple neurons for one PE sharing the same data path, the distance calculation and the detection of the best matching unit are divided into two parts and supported by a central winner-takes-all-unit. First, the distance between an input vector and the neuron weights of the same virtual layer is calculated and locally stored in the PEs in $d+3$ clock cycles. Subsequently, the BMU of all neurons on the same virtual layer is determined by a bit-serial search in $\mathrm{E} \cdot p_{SOM} + \lceil \log_2(d) \rceil$ clock

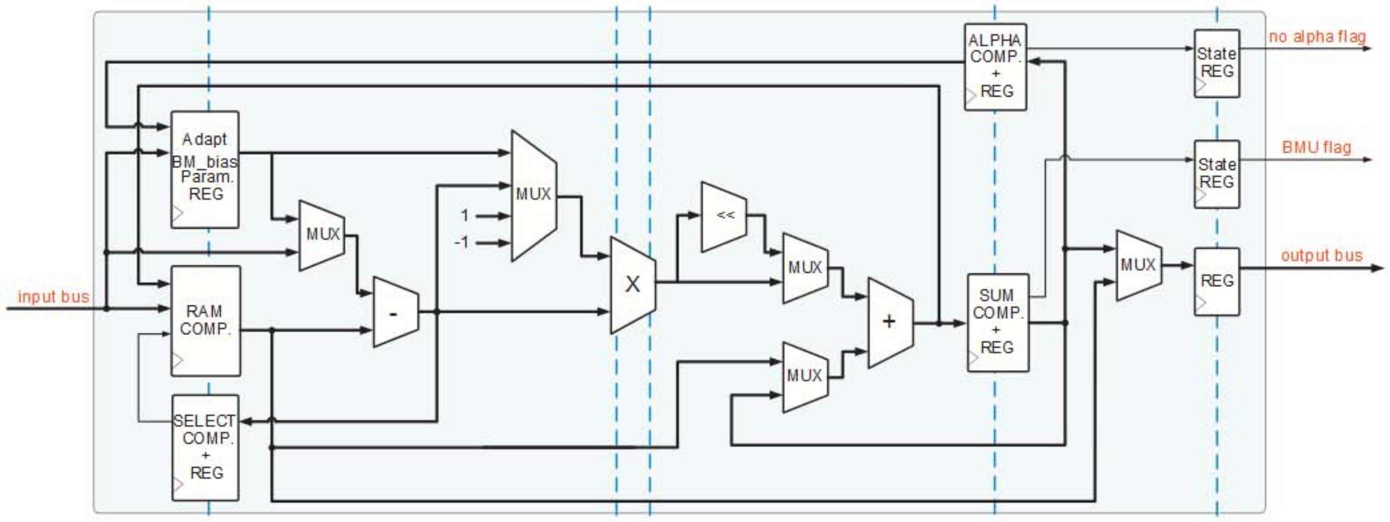

Fig. 4. (Color online) Architecture of the gNBXe processing elements.

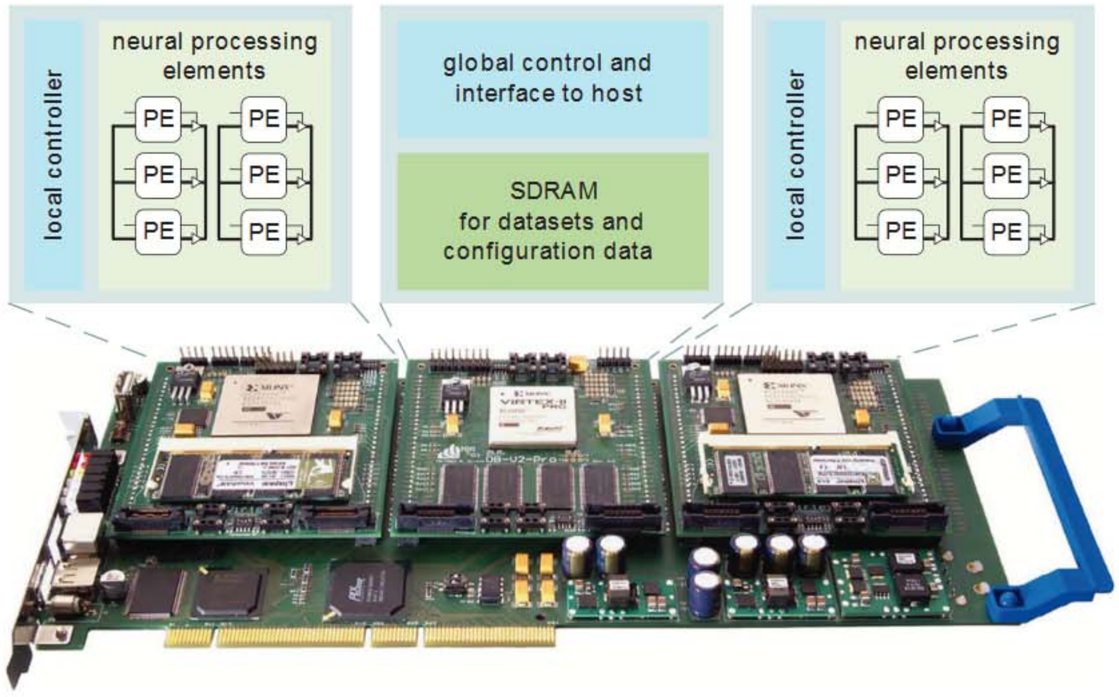
local controller
neural processing elements
PE
global control and interface to host
SDRAM for datasets and configuration data
local controller
neural processing elements
PE

Fig. 5. (Color online) The RAPTOR prototyping system.

cycles. While repeating the distance calculations and determining the best matching units for all neurons of each virtual layer, requiring 2 additional clock cycles for switching between the virtual layers, the local BMU of a PE-FPGA is detected by successively comparing the found BMUs in 9 clock cycles using the WTA unit. The overall number of clock cycles for the distance calculation and detection of the best matching unit adds up to $(d+\mathrm{E}\cdot p_{SOM}+\lceil\log_2(d)\rceil+14)\cdot N_{N_{PE}}$.

3.2.2. Distribution of adaptation parameters

During adaptation the neuron weights have to be updated using distance-specific user parameters (h_{cj}). To distribute the individual parameters, the distances between the map position of the global BMU (stored in the PEs) and the positions of all other neurons is calculated and stored in the PEs in $\mathrm{n}+3$ clock cycles. Subsequently, the local controller successively applies the individual parameters for all neurons within a specific adaptation radius to the PEs while the map distances are stepwise decreased. Once a map distance reaches zero, the individual adaptation parameter, applied by the controller in $p_{SOM}+1$ clock cycles, is locally stored in the PEs. After the individual parameters have been distributed, the adaptation of the neuron weights is performed in $d+3$ clock cycles taking winning frequency calculation into account. The overall number of clock cycles for distributing the adaptation values and for adapting the neuron weights adds up to $(p_{SOM}+d+\mathrm{n}+9)\cdot N_{N_{PE}}$.

4. Experimental setup

Based on the RAPTOR-X64 prototyping system [12] (Fig. 5) we have implemented a SOM accelerator with 16-bit precision and 2048 local addresses. A global controller clock frequency of

Table 1
Hardware costs, available processing elements (N_{PE}), and possible number of neurons ($N_{N_{max}}$) for Xilinx-Virtex-4 FPGAs using 16-bit precision, 2048 local addresses and varying number of neuron weights (d).

Hardware resources	Global ctrl.	PE-FPGAs	
	V2-4000	V4FX100	$5 \times$ V4FX100
N_{PE}		121	605
Slices	3173 (13%)	40,586 (96%)	202,930
RAMB16s	6 (60%)	252 (67%)	1260
MULT 18×18s		121 (75%)	605
$N_{N_{max}}$ (d=2000)		121	605
$N_{N_{max}}$ (d=200)		1210	6050
$N_{N_{max}}$ (d=100)		2299	11,495
$N_{N_{max}}$ (d=8)		22,506	112,530

33 MHz and a three times higher PE clock frequency (99 MHz) have been achieved using five PE-FPGAs (Xilinx Virtex-4 FX100) and a Xilinx Virtex2-4000 for the global controller. Table 1 shows the hardware cost and the number of available neurons for the gNBXe realization.

For comparing the analysis results and the performance of our accelerator, two software implementations are used. On one hand, a commercially available simulation environment is used (see Section 4.1). On the other hand, a parallelized software implementation of the CSOM running on a Core-i7 950 Quad Core, code-named *Bloomfield*, core clock 3.07 GHz, working at 3.20 GHz utilizing Intel® Turbo Boost Technology, has been realized. The reference system has 6 GB RAM and uses an ASUS P6T WS PRO mainboard. The software is written in C++ with parameterizable precision and multi-threading enabled using the OpenMP library. The hardware accelerator and the software reference implementation are identically parameterized and trained for real-world data using a Matlab front end.

The software implementation focuses on speed and inherent parallelism and it incorporates all optimizations that have been used for the hardware realization (e.g., avoidance of square root calculation). The Core-i7 processor provides four cores and up to eight threads (two threads per core).

Organization of the parallel calculation steps is done by block cyclic scheduling. The pseudocode in Algorithm 1 details the block cyclic scheduling for the distance calculation $J\mathbf{x} \ \mathbf{w}J_{p_V}$ between an input vector and all neuron weights.

Algorithm 1. Distance calculation.

```
function find_global_bmu (*x, g, num_threads)
  Num_Parallel_threads '  static (num_threads)
  x, w, thread_BMU '  shared across all threads
  local_BMUs, cur_dist '  private to each thread
  local_min_dist '  DIST_MAX
  local_BMU_ '  0
  for all i A  num_neurons do                    x  in parallel
    cur_dist '    g*(1=N  w_i:F)
    for all d A dim(x) do
      if p_V = = 1 then
        cur_dist + = (x(d)  w(d))
      else
        cur_dist + = (x(d)  w(d))^2
      end if
    end for
    if cur_dist ○ local_min_dist then
      local_min_dist '  cur_dist
      local_BMU '  w_i:position
    end if
    id '  thread_id
    thread_BMU[id]:BMU '  local_BMU
    thread_BMU[id]:distance '  local_min_distance
  end for                          x  end in parallel
  global_BMU '  thread_BMU[0]:BMU
  global_distance '  thread_BMU[0]:distance
  for all c A  num_threads do
    if thread_BMU[c]:distance ○ global_distance then
      global_distance '  thread_BMU[c]:distance
      global_BMU '  thread_BMU[c]:BMU
    end if
  end for
  return global_BMU
end function
```

4.1. *Processing large, real data*

4.1.1. *Data and test questions*

To gauge both the algorithmic correctness and the learning speed of the gNBXe implementation of CSOMs for relevant use-case scenarios, we reproduce clustering of a large real dataset, which was done in earlier data analysis studies using research software [2]. Initial testing of the gNBXe has been reported in [13]. The dataset used here is a spectral image, which represents a wide class of important large and complex datasets. In a spectral image each pixel is a d-dimensional vector called *spectrum*, where every vector element is a light intensity (reflectance, radiance, or emission) measured through a different *spectral band* centered at a different wavelength. As materials interact with light preferentially at different wavelengths, the spectrum at a given pixel location characterizes the chemical composition of the surface material in that pixel footprint. In *hyperspectral images* the spectra can contain measurements in hundreds of narrow adjacent spectral bands, resulting in unique fingerprints of the surface materials. The rich information content in these data can yield extremely detailed knowledge for monitoring environmental conditions, resources, medical diagnosis, quality control in food and drug industry, and more. Extraction of the relevant details is non-trivial, however. A number of unsupervised methods – other than SOMs – have been used by various researchers to exploit hyperspectral data for discovery and mapping of material clusters. Traditional methods, e.g., PCA, k-means, (fuzzy) c-means, ISODATA, and (Gaussian or other) mixture models are most frequently applied (either by themselves or in support of subsequent supervised classification) or compared with proposed new approaches [14–18]. These methods often provide robust segmentations but find less of the significant clusters than those exist in the data, or the accuracy of the clusters is not very high. One reason is that many use a significantly reduced number of spectral bands, another is that many are less than ideally suited to deal with the highly irregular cluster structure typically present in hyperspectral data. As an alternative approach, smart learning algorithms such as SOMs have been demonstrated to achieve excellent results, but in order to accomplish the same in reasonable time for large tasks (real-time classification on-board a spacecraft, searching in huge archives) fast hardware implementation such as the gNBXe is needed. In the experiment that we present for hardware testing here, we recreate the SOM from the clustering study by [2,19], where all spectral bands of a hyperspectral image (see below) were used, and from the resulting SOM 23 geologically significant clusters of widely varying sizes were extracted, verified by domain expert, and subsequently also verified by highly accurate supervised classification based on the clusters discovered through the SOM [20].

The dataset we use is a hyperspectral image of the Lunar Crater Volcanic Field in Nevada, taken by NASA's AVIRIS sensor (e.g., [21]) in 1994 with a spatial resolution of 17 m/pixel. It comprises 614 × 420 pixels in 194 spectral bands, amassing 190.84 Mbytes when each data item is a floating-point number, which is the case during processing. The distribution of these data is in 2-byte short integer form, appropriately scaled to preserve precision. Clustering of this scene with software CSOMs along with geologic interpretation of 23 extracted clusters is presented in [2]. This image has been thoroughly analyzed through multiple approaches (see references to previous studies) and the clusters produced by CSOMs were interpreted and verified against field knowledge by domain experts. Thus we have confidence that it provides a solid baseline for assessing the quality of the gNBXe learning as relevant for real applications.

We choose the LCVF dataset not only because it is fairly large and the input vectors are high-dimensional (in the hyperspectral image) but also because it has *complex cluster structure* which poses non-trivial challenges for clustering algorithms [3]. One aspect of this is that the CSOMs learned with this data are quite complex themselves, which makes comparison of software and hardware results   from an application point of view   an intricate exercise. Noise inherent in real imagery adds to this challenge.

Gaining speed of processing and saving energy are obvious advantages of hardware SOMs. But from a data analysis point of view, before we can rely on hardware CSOMs, we need satisfactory answers to several questions. Is there a degradation of cluster distinctions, in comparison to CSOMs produced by software, due to possible effects of lesser precision? If there is, how does it depend on the hardware precision? Does it get worse with longer learning? How does it depend on the data dimensionality, and on the combination of increasing dimensionality and increasing number of learning steps? While answering all these questions in an exhaustive manner is outside the scope of this paper, below we demonstrate excellent match between results from software CSOM and from its gNBXe implementation, for a specific set of parameters.

4.1.2. Software vs. gNBXe analyses

Data analyses with software CSOMs have been performed at the Neural Machine Learning Group, Rice University, using research modules developed in-house. These modules are built on top of library functions from Khoros [22] and NeuralWorks Professional [23], for development of new and advanced functionalities. For perfect synchronization between software and hardware, we deploy an initial state of a CSOM from the software as a set of files which describes every detail of the CSOM including the SOM weights and their winning frequencies at the given learning step and the pseudorandom sequence of indexes which the software uses for the "random" picking of input vectors. Both software and hardware perform a pre-set number of learn steps starting from this initial state, using the same order of input vectors and the same schedule to decrease the learning parameters a(t), b(t), g(t) with time t. The resulting hardware CSOM is loaded back into the software and a detailed comparison is made with the software counterpart.

The CSOM has 40 × 40 neurons, and we use diamond-shaped neighborhood in this experiment. The research software is run on a Quad Core Intel Xeon W3550 3.0 GHz, 12 Gbytes DDR3 SDRAM (Dell Precision T3500n), in Linux Red Hat operating system, using a single CPU. Despite the 64-bit architecture, we use a 32-bit system because of compatibility issues with some of the underlying 3rd-party components that the software is built on. The gNBXe implementation utilizes four PE-FPGAs with a precision of 16 bits.

The variable parameters of the analysis experiment are shown in Table 2. The learning parameters for this particular case do not change because we start the runs from a fairly mature stage where the aggressive decrease of these parameters has already occurred. We should add that   unlike in Section 4.2 where tests are focused on specific performance aspects, and the software implementation only contains the "bare" CSOM algorithms,   the research software in these experiments computes about a dozen different types of SOM products, in which the gNBXe does not compute. Therefore, we cannot claim a fair comparison of CPU times, since we have no way of estimating how long it would take the gNBXe to compute all of the same products, but   given that the bulk of the time is spent on SOM learning   we still think it is valuable to show the user time (data load and start-up operations, CPU time+writing products once at the end) for the software and the same for the gNBXe.

All else being equal, in a hardware vs. software experiment we are concerned with the differences in the resulting weights, winning frequencies, and the mapping of the input vectors to the CSOM neurons.

The pertinent numbers, including the (information-theoretical) entropies of the maps, are presented in Table 3. The entropies are normalized by the number of SOM neurons to facilitate direct comparison across different-sized SOMs. The maximum possible scaled entropy value is 1. Table 3 indicates a high degree of match between hardware and software processing.

For a visual illustration, Figs. 6 and 7 show a representation of the software and hardware CSOMs, respectively. Both learned for 1,031,520 steps (4 × the number of pixels in the LCVF image) starting from a state at 325,328 steps, formerly produced by software. Each grid cell represents an SOM neuron. Here, two properties are visualized: the density (number of data points mapped to the SOM neuron), indicated by the relative intensity of the red monochrome color, and the distance of the neuron's weight vector to the weight vectors of all eight neighbor neurons. The weight distances are expressed by the modified U-matrix

Table 2
Parameters of the LCVF data analysis experiment comparing results from software CSOM and gNBXe implementation.

Experiment parameters	LCVF
Input dimension	194
# Data vectors	257,880
# Learn steps	1,031,520
a start/end	0.02/0.02
b start/end	0.001/0.001
g start/end	0.1/0.1

Table 3
Results of LCVF data analysis experiments comparing statistics from software CSOM and gNBXe implementation. The mean densities and entropies are shown for active neurons (neurons that have data points mapped to them).

Analysis results, LCVF data	Software	gNBXe	Difference
User time (s)	4130.42	29.83	4100.59
User time per 1 M learn steps (s)	4004.21	28.92	3975.29
Elapsed time (s)	4151.83	62.08	4089.74
# active neurons	1565	1569	−4
Mean weight value	−0.2923	−0.2939	0.0016
Mean density	164.79	164.36	0.43
Scaled entropy	0.9931	0.9919	0.0013

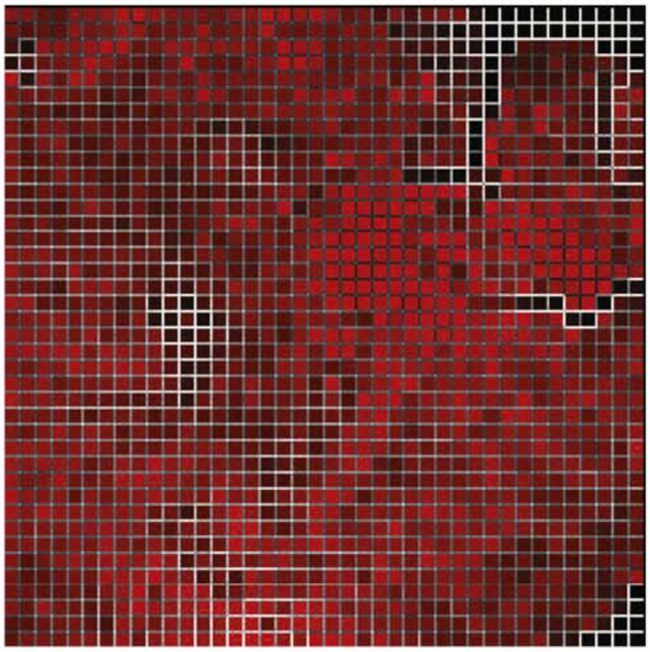

Fig. 6. (Color online) SOM resulting from software training with the LCVF data. Densities and mU-matrix are overlain on the SOM.

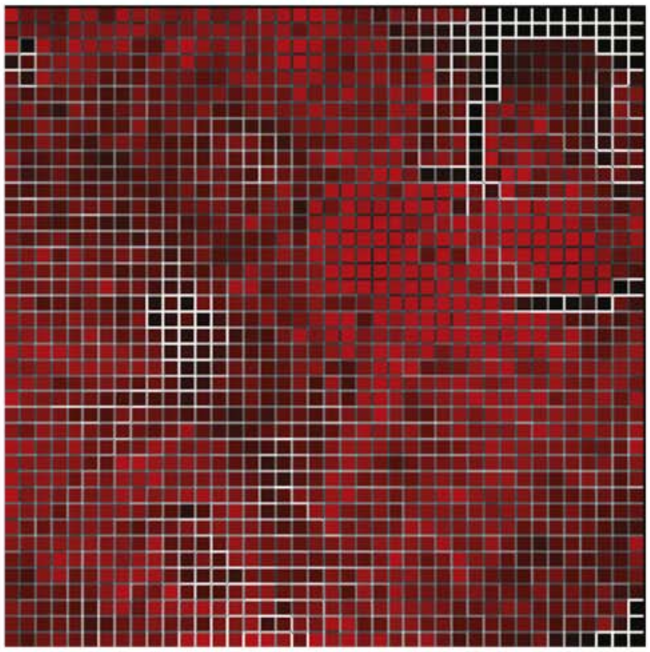

Fig. 7. (Color online) SOM resulting from gNBXe training with the LCVF data. Densities and mU-matrix are overlain on the SOM.

(mU-matrix, [11]) appearing as gray scale "fences" between grid cells. White fence means large distance (large dissimilarity), darker fence means smaller distance. Visually, the two SOMs are hard to distinguish.

From this real data analysis case, we can conclude that the SOM produced by the gNBXe is remarkably close to its software counterpart. This is not only supported by the statistics of the weight values and winning frequencies (densities), but even more strongly evidenced by the spatial distribution of the weights and densities in the SOM. In Figs. 6 and 7 we see the same spatial appearance of the weight distances (fences) and densities, with very slight differences. The match is sufficient to identify the same clusters in the data as from the gNBXe SOM and as from the software SOM, which is the most important quality measure.

4.2. *Performance measurement*

After the "high-level" tests in the previous Section, relevant from a data analyst's point of view, now we present detailed hardware characterization tests. In addition to the high-dimensional LCVF dataset introduced in Section 4.1.1, we also use a low-dimensional real dataset for hardware characterization tests. The low-dimensional dataset is a spatial and spectral subset of a multi-spectral image of Ocean City, Maryland, comprising 512×512 pixels in 8 spectral bands [24], altogether 8.39 Mbytes. Analysis of this urban image with software CSOMs are described in [25]. In future work we plan to compare hardware and software clusterings of this dataset as well. By including it here, our goal is to broaden the variety of real data characteristics for the performance measurements.

For hardware characterization, the execution time and power consumption are compared for the proposed hardware accelerator and the Core-i7 implementation, described at the beginning of Section 4. To measure the execution time of both architectures, high-resolution timers are added in the C++ functions of the software CSOM as well as in the C++ interface library for the gNBXe. Additional hardware timers are added to the gNBXe VHDL implementation, capturing detailed information on a cycle by cycle basis. General timing analysis is done using a high-level model of the neuroprocessor. The deviation between measured and modeled execution time is less than 0.4% for the core algorithm including initialization of the neuron map as well as reading the map back to the SDRAM of the GC.

The power consumption of both the systems is determined in several steps of granularity. The power for the core region of the Core-i7 processor is measured on the 12 V supply rail. The power for the gNBXe is measured on the 12 V supply rail of the prototyping system RAPTOR and additionally analyzed using Xilinx XPower Analyzer. Current measurement is done with the current probe amplifier AM503B from Tektronix and recorded with an oscilloscope. Additionally, the primary power of the complete system environment is measured on the 230 V side of the experimental setup.

4.2.1. *Test cases*

As explained in Section 3, the high flexibility of the neuroprocessor is achieved by the possibility to map multiple neurons to a single PE. Additionally, it is possible to add further PE-FPGAs and thus increase the available number of parallel PEs. For an in-depth analysis of the scalability of the architecture, two test cases have been defined, which are discussed in the following.

The first test case (case A) determines the power consumption and execution time needed to reach the maximum parallelism for a specific CSOM. Therefore, the map size is kept constant while the number of PE-FPGAs is increased. As depicted in Table 1, $N_{N_{PE}} = 121$ PEs can be realized on a single Virtex-4 FX100 FPGA. The maximum parallelism is achieved with a one-to-one mapping of neurons to PEs. With a maximum number of five PE-FPGAs ($N_{\mathrm{PE\ FPGA}}$), the largest map that can be completely calculated in parallel has $121 \times 5 = 605$ neurons.

Test case A:

- Train a CSOM with 605 neurons using one PE-FPGA (5 neurons per PE).
- Increase the number of PE-FPGAs in order to increase the parallelism until each neuron is mapped to exactly one PE (maximum parallelism).

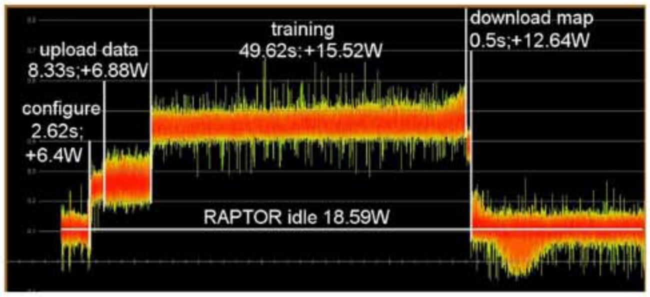

Fig. 8. (Color online) Execution time and power for training the CSOM with the LCVF dataset in 1,031,520 learning steps, utilizing two PE-FPGA with $N_{N_{PE}} = 10$.

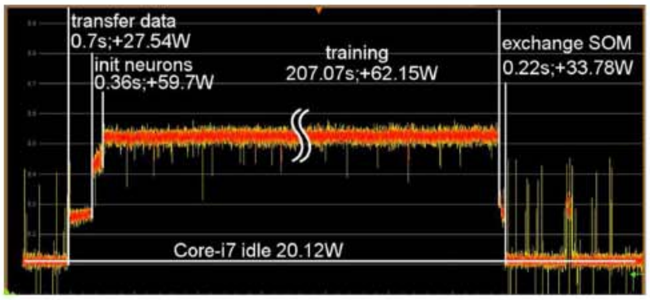

Fig. 9. (Color online) Execution time and power for training the CSOM with the LCVF dataset in 1,031,520 learning steps, utilizing four cores of the Core-i7.

The second test case (case B) determines the power consumption and execution time needed for the maximum number of neurons that can be mapped to a specific number of PE-FPGAs. As explained in Section 4, each Virtex-4 FPGA includes 2048 × 16 bit of local memory. For the LCVF dataset with 194 spectral bands and a two-dimensional neuron map, the maximum number of neurons per PE ($N_{N_{PE}}$) is $\lfloor 2048=(194+3)\rfloor = 10$. For the Ocean City dataset with 8 spectral bands the maximum number of neurons per PE is 186.

Test case B:

- Train a CSOM with the maximum number of neurons for a given dataset using one PE-FPGA.
- Increase the number of PE-FPGAs in order to increase the number of neurons.

4.2.2. *Experimental results*

Fig. 8 depicts the current measurement for the neuroprocessor training of LCVF dataset with $N_N = 2420$ neurons, $N_{\mathrm{PE\ FPGA}} = 2$, and $N_{N_{PE}} = 10$ in 1,031,520 learning steps. If the RAPTOR system is idle, i.e., no FPGA is configured, it requires an idle current of 1.549 A, corresponding to a power consumption of 18.59 W. The total power of the complete system environment (including the host PC) aggregates to 161 W. Configuring the FPGAs by uploading the bitstreams and waiting until the digital clock managers (DCM) are locked, takes about 2.62 s with 6.4 W of additional power. Because FPGA configuration is done once while powering up the test system, the configuration time does not have to be taken into account for the performance evaluation.

After the FPGAs are configured and the DCMs are locked, the initial parameterization is performed, including 119 ms for setting the startup configuration of the selected CSOM algorithm (e.g., the map size and the distance metrics). Additionally, the input data is loaded to the global controller during initialization. For the LCVF dataset (95.42 Mbytes when stored in 2-byte integer form) the input data is uploaded in about 8.3 s requiring about 400 mW for the global controller and 3.24 W for each PE-FPGA staying in idle state. Uploading the dataset to the hardware is required only once for each dataset. Nevertheless, this time is taken into account for the comparisons in this paper. Training the CSOM in 1,031,520 learning steps for $N_N = 2420$ neurons requires 49.62 s and 7.32 W per PE-FPGA. Additional 510 ms are needed to download the trained SOM to the host system.

Fig. 9 depicts the current measurement for the Core-i7 training $N_N = 2420$ neurons with the LCVF dataset. Four threads are used since further increasing the number of threads did not result in a significant speedup of the CSOM simulation (less than 2%) but required an additional power consumption of about 12 W. The cores of the microprocessor require a current of 1.68 A corresponding to 20.12 W for just running the operating system, without the CSOM algorithm. The resulting system power is about 142 W. The CSOM training starts with transferring the input and configuration data from the Matlab front end to the C++ CSOM function using a matlab executable (mex) file. The execution time for the data transfer is 0.7 s, requiring an additional power of about 27.54 W. This time is taken into account for both the CSOM software and the hardware implementation. After the C++ CSOM function is called, the neuron initialization takes about 360 ms with 59.7 W. Training the CSOM in 1,031,520 learning steps for $N_N = 2420$ neurons is done in about 207.07 s with 62.15 W. Finally, exporting the trained SOM from the C++ function to the mex interface function is done in about 0.22 s with 33.78 W.

Table 4 gives a detailed view of the experimental results for both test cases using the LCVF dataset with 194-dimensional input vectors. A similarly designed measurement is done for the Ocean City dataset and discussed in Section 4.3. In the top of Table 4 the energy consumption and execution time for a fixed number of neurons utilizing a different number of PE-FPGAs is measured. While a minimum number of $N_{N_{PE}}$ neurons per PE is needed to simulate the desired CSOM, a maximum number of $N_{N_{max}}$ can be calculated in the same setup without additional cost. The upload time stays constant for the training dataset and takes the upload of the input data into account. The execution time for training the CSOM decreases with the number of FPGAs used. In the current implementation, the global controller limits the achievable performance due to its low clock frequency. This will be optimized in future work. While simulating the CSOM using five PE-FPGAs in parallel requires more than twice the energy of a single-FPGA implementation, the total energy of the test system is reduced by 33.7% due to the decreased simulation time.

In the middle of Table 4, the results for simulating the maximum number of neurons for a fixed number of PE-FPGAs are given. Similarly to the first test scenario, the time needed for uploading the training dataset stays constant. In this scenario, also the simulation time stays nearly constant while the number of simulated neurons increases. Thus, performance (in terms of emulated neurons per time) increases linearly with the number of used FPGAs. Due to the added FPGAs, energy consumption increases linearly with the number of PEs as well.

In the lower part of Table 4 four examples from the Core-i7 measurements simulating a CSOM with 605, 1210, 2420, and 6050 neurons are shown. The execution time increases nearly linearly with the number of neurons. Only the memory requirements for the largest map with 6050 neurons exceed the level 3 cache and thus result in a significantly reduced performance.

4.3. *Performance evaluation summary*

Table 5 depicts the speedup and the percentage of saved energy using the FPGA-based neuroprocessor compared to the software implementation utilizing all four cores of the Core-i7 in parallel. The setup used for comparison utilizes five Virtex-4 PE-FPGAs each one identically parameterized as in the configuration

Table 4
Execution time and energy for test cases A and B for the LCVF dataset, utilizing several numbers of PE-FPGAs ($N_{PE\text{-}FPGA}$) and 4 cores of the Core-i7.

gNBXe-Execution time and energy for training a CSOM with 605 neurons for the LCVF dataset in 1,031,520 iterations									
$N_{N_{max}}$;$N_{N_{PE}}$	$N_{PE\text{-}FPGA}$	N_{PE}	Execution time (s)		Energy (Ws)				
			Upload	CSOM	CSOM	Upload	RAPTOR	System	Total
605; 5	1	121	8.3	25.34	195.62	30.21	625.36	5435	6286
726; 3	2	242	8.3	16.13	242.59	57.1	454.15	3952	4706
726; 2	3	363	8.3	13.63	304.76	83.9	407.67	3549	4346
968; 2	4	484	8.3	13.95	414.03	110.88	413.62	3601	4539
605; 1	5	605	8.3	11.64	430.68	137.78	370.68	3229	4168
gNBXe-Execution time and energy for training a CSOM with $N_{N_{max}}$ for the LCVF dataset in 1,031,520 iterations									
$N_{N_{max}}$;$N_{N_{PE}}$	$N_{PE\text{-}FPGA}$	N_{PE}	Execution time (s)		Energy (Ws)				
			Upload	CSOM	CSOM	Upload	RAPTOR	System	Total
1210; 10	1	121	8.3	49.15	379.4	30.21	1067	9268	10,744
2420; 10	2	242	8.3	49.62	744.3	57.1	1076	9344	11,221
3630; 10	3	363	8.3	49.8	1113	83	1080	9373	11,649
4840; 10	4	484	8.3	50.13	1487	110.8	1086	9426	12,110
6050; 10	5	605	8.3	50.46	1867	137.78	1092	9479	12,576
Core-i7-execution time and energy for training a CSOM with 605, 1210, 2420, and 6050 neurons for the LCVF dataset in 1,031,520 iterations (4 threads)									
N_N	Execution time (s)	Energy (Ws)			N_N	Execution time (s)	Energy (Ws)		
		CSOM	System	Total			CSOM	System	Total
605	51.87	3395	7384	10,779	2420	208.35	12,898	29,604	42,503
1210	93.92	6147	13,355	19,502	6050	743.42	48,658	105,584	154,243

Table 5
Speedup and percentage of energy savings when performing CSOM simulation with the gNBXe instead of a Core-i7.

N_N	Speedup		Energy savings (%)			
	LCVF	Ocean city	LCVF		Ocean city	
			CSOM	Total	CSOM	Total
605	2.6	5.81	67.3	55.1	81.1	80.5
1210	4.16	7.02	82.3	64.5	84.3	83.8
2420	6.88	13.94	88.3	84.9	92	91.8
3630	9.12	17.96	91.4	88.7	93.8	93.6
4840	9.58	21.74	91.7	89.2	94.8	94.7
6050	12.65	25.41	93.6	91.8	95.6	95.5
60500	–	20.97	–	–	94.6	94.6
112,530	–	16.56	–	–	93.1	93.1

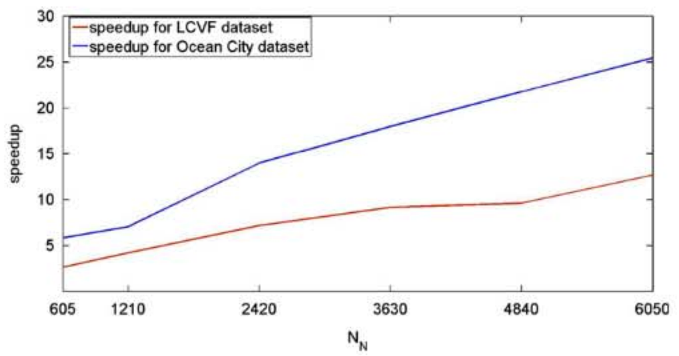

Fig. 10. (Color online) Speedup for CSOM learning of more than 1 million vectors with 194 and 8 dimensions (LCVF and Ocean City data, respectively), comparing the FPGA-based simulation with a multi-core based simulation.

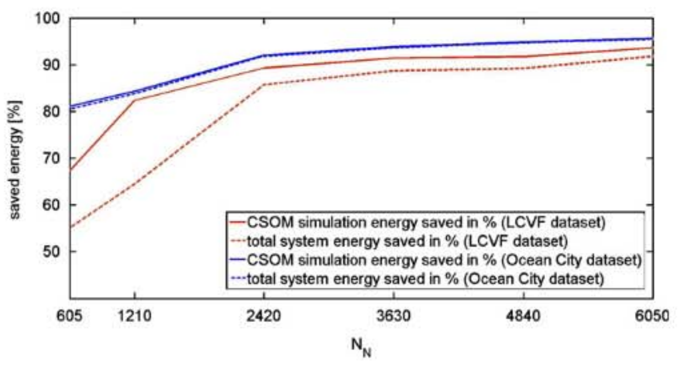

Fig. 11. (Color online) Percentage of energy savings for CSOM learning of more than 1 million vectors with 194 and 8 dimensions (LCVF and Ocean City data, respectively), comparing the FPGA-based simulation with a multi-core based simulation.

that has been described in Section 4.2.1. The speedup is the ratio of the execution time of the Core-i7 to the execution time of the hardware accelerator, both including the time needed to upload the input data (8.3 s for LCVF, and 0.436 s for Ocean City). The energy savings in the "CSOM" columns are taking into account only the hardware accelerator and the processor. The total energy of the two systems is compared in the columns "total".

As can be seen in Fig. 10, the neuroprocessor outperforms the Core-i7 by a factor of two to six even for small and low-dimensional CSOMs. When simulating large maps utilizing five PE-FPGAs, our hardware accelerator achieves a more than tenfold speedup compared to the Core-i7. The energy consumption decreases with the increasing speedup of the hardware accelerator even when taking the total energy consumption of the test scenario into account (Fig. 11).

4.4. Comparison to related work

A variety of approaches for hardware-implementations of self-organizing feature maps has been proposed in the past. Although some architectures are targeting ASICs (application-specific integrated circuits), today, the majority of digital hardware implementations is based on FPGAs. In the following, recent hardware realizations are discussed and compared to the architecture discussed in this paper. For a rough performance comparison we use the number of possible weight updates ($P_{C,I}$) in units of million per second, known as MCUPS (Million Connection Updates Per Second), as given in Eq. (6). Here, (T_l) is the time in microseconds for one learning step, i.e., for processing one input vector.

$$P_{C,I} = \frac{N_N \cdot d}{T_l} \tag{6}$$

Porrmann et al. presented a SOM neuroprocessor consisting of a SIMD array of processing units controlled by an external controller in [26]. In addition to the acceleration of SOM simulation, hardware implemented pre- and post-processing for component maps, pattern position maps, and U-Matrix visualization was added. The design was implemented in a 0.8 mm standard cell technology. Since the gNBXe presented here is a successor of this ASIC implementation, both architectures share the same basic concepts. For an implementation of 128 neurons with 128 weights and 8-bit precision, a performance of 1318.00 MCUPS is achieved at a clock frequency of 45 MHz.

In [27] Hikawa proposed an approach utilizing digital phase locked loops as computation elements for SOM simulation. To avoid multiplications, several simplifications to the original SOM algorithm were made, e.g., Manhattan distance is used and only the next neighbors of the best matching neuron are updated during adaptation. In an FPGA implementation of 25 neurons with two weights and 10-bit precision, 4.89 MCUPS are achieved using a Xilinx XC2V250 FPGA operating at a clock frequency of 200 MHz.

The architecture proposed by Ramirez-Agundis et al. in [28] is capable of performing batch update training, i.e., the neuron weights are updated after the entire training set has been applied, to speed up simulation. Again, simplifications of the original SOM algorithm include the use of Manhattan distance instead of Euclidean distance calculation and the restriction of the adaptation neighborhood to one (diamond shaped neighborhood) or zero (only the best matching unit is updated). For an implementation of 256 neurons with 16 weights (8 bits), a clock speed of 70 MHz has been achieved for a Xilinx XC2V6000 FPGA. Since updating the neuron weights requires 45 clock cycles, a performance of 6372.00 MCUPS can be achieved.

A parameterizable IP core has been presented by Hendry et al. in [29]. In this approach, the number of neurons, the dimension of the input vectors, and the precision can be easily modified at design time. Similar to the hardware accelerators described above, the distance metric is limited to Manhattan distance. Additionally, the adaptation parameters are limited to negative powers of two. For an implementation of 256 neurons, 16 neuron weights and a precision of 8 bits, synthesized for a 0:65 mm technology, a performance of 660 MCUPS is achieved at a clock frequency of 50 MHz.

Table 6 summarizes the characteristics of the hardware accelerator presented in this paper compared to related work. The parameter *Norm* describes which distance norm has been implemented in the referenced designs (column 1) and the column "Adapt." specifies whether the adaptation of the neuron weights is performed by means of multiplications or by shift operations, which limit the adaptation values to negative powers of two.

Table 6
Comparison of the proposed hardware accelerator to previously published implementations.

Ref.	prec: (bits)	N_N	d	Norm	Adapt.	$P_{C,I}$ [MCUPS]
[27]	10	25	2	1	N/A	4.89
[29]	8	256	16	1	Shift	660.00
[26]	8	128	128	1	Shift	1,318.00
[28]	8	256	3	1	Shift	6,372.00
Core-i7	16	6050	194	1, 2	Mult.	1,628.00
gNBXe	16	6050	194	1, 2	Mult.	20,604.00

5. Conclusion

We have shown that the proposed gNBXe-based multi-FPGA hardware accelerator for CSOMs significantly speeds up simulation of high-dimensional datasets compared to state-of-the-art multi-core PCs while the energy consumption decreases at the same rate. We have shown this also in the context of a real data analysis example, where the gNBXe hardware CSOM is envisioned as a fast "clone" of the software equivalent, performing exactly the same functions so that the SOM can be interchanged between hardware and software. While we presented remarkable agreement between hardware and software processing results for a limited set of circumstances, there remain finer differences that are of concern. In follow-up work we plan to address these finer differences, characterize their sources, improve our understanding of how to decrease them, and what the trade-offs are. This will require finer grained studies on one hand (involving both synthetic and real data), and extension of our experiments (with the LCVF and Ocean City datasets) to a wider range of parameters, on the other hand. Beyond that, much more work will be needed to complete systematic studies to answer the more general questions we posed in Section 4.1.1.

We want to add that as another future step. We also aim to extend the reconfigurable accelerator towards the emulation of additional artificial neural networks. Furthermore, the number of PE-FPGAs will be increased and high-speed serial IOs will be used for communication between the GC and the PEs to further increase the scalability of the system.

Acknowledgments

This research is partially supported by the Center of Excellence Cognitive Interaction Technology (CITEC).

We thank Professor Bea Csathó, Department of Geology, SUNY Buffalo, for the Ocean City dataset. Former data analyses and development of software tools by the Neural Machine Learning Group, Rice University, which are used for this study, were partially funded by NASA Grants NNG06GE95G, NNG05GA94G (and predecessor grants) and by DARPA through AFRL Contract FA8650-09-C-7915 with Rice University FA8650-09-C-7915.

References

[1] T. Kohonen, Automatic formation of topological maps of patterns in a self-organizing system, in: E. Oja, O. Simula (Eds.), Proceedings of 2SCIA, Scand, Conference on Image Analysis, Helsinki, Finland, 1981, pp. 214–220.
[2] E. Merényi, Precision mining of high-dimensional patterns with self-organizing maps: interpretation of hyperspectral images, in: P. Sincak, J. Vascak (Eds.), Quo Vadis Computational Intelligence: New Trends and Approaches in Computational Intelligence, Studies in Fuzziness and Soft Computing, vol. 54, Physica Verlag, 2000.
[3] E. Merényi, K. Tasdemir, L. Zhang, Learning highly structured manifolds: harnessing the power of SOMs, in: M. Biehl, B. Hammer, M. Verleysen, T. Villmann (Eds.), Similarity Based Clustering, Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence, vol. 5400, Springer-Verlag, 2009, pp. 138–168.
[4] D.G. Covell, A. Wallqvist, A.A. Rabow, N. Thanki, Molecular classification of cancer: unsupervised self-organizing map analysis of gene expression microarray data, Mol. Cancer Ther. 2 (March) (2003) 317–332.
[5] K. Tasdemir, E. Merényi, Exploiting data topology in visualization and clustering of self-organizing maps, IEEE Trans. Neural Networks 20 (4) (2009) 549–562.
[6] C. Pohl, M. Franzmeier, M. Porrmann, U. Rückert, gNBX – reconfigurable hardware acceleration of self-organizing maps, in: Proceedings of the IEEE Int Field-Programmable Technology Conference, 2004, pp. 97–104.

[7] D. DeSieno, Adding a conscience to competitive learning, in: Proceedings of the IEEE International Neural Networks Conference, 1988, pp. 117–124.
[8] P.L. Zador, Asymptotic quantization error of continuous signals and the quantization dimension, IEEE Trans. Inf. Theory 28 (March (2)) (1982) 139–149.
[9] H.-U. Bauer, R. Der, M. Herrmann, Controlling the magnification factor of self-organizing feature maps, Neural Comput. 8 (4) (1996) 757–771.
[10] H. Ritter, K. Schulten, On the stationary state of Kohonen's self-organizing sensory mapping, Biol. Cybern. 54 (1986) 99–106.
[11] E. Merényi, A. Jain, T. Villmann, Explicit magnification control of self-organizing maps for "forbidden" data, IEEE Trans. Neural Networks 18 (May (3)) (2007) 786–797.
[12] M. Porrmann, J. Hagemeyer, C. Pohl, J. Romoth, M. Strugholtz, RAPTOR a scalable platform for rapid prototyping and FPGA-based cluster computing, in: Proceedings of the International Conference on Parallel Computing, ParCo2009, Symposium on Parallel Computing with FPGAs, Lyon, France, 1–4 September 2009.
[13] J. Lachmair, E. Merényi, M. Porrmann, U. Rückert, gNBXe – a reconfigurable neuroprocessor for various types of self-organizing-maps, in: Proceedings of the 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2012, pp. 645–650.
[14] O. Duran, M. Petrou, A time-efficient method for anomaly detection in hyperspectral images, IEEE Trans. Geosci. Remote Sensing 45 (12) (2007) 3894–3904.
[15] Y. Tarabalka, J.A. Benediktsson, J. Chanussot, Spectral-spatial classification of hyperspectral imagery based on partitional clustering techniques, IEEE Trans. Geosci. Remote Sensing 47 (8) (2009) 2973–2987.
[16] A. Paoli, F. Melgani, E. Pasolli, Clustering of hyperspectral images based on multiobjective particle swarm optimization, IEEE Trans. Geosci. Remote Sensing 47 (12) (2009) 4175–4188.
[17] T. Veracini, S. Matteoli, M. Diani, G. Corsini, Robust hyperspectral image segmentation based on a non-gaussian model, in: 2nd International Workshop on Cognitive Information Processing (CIP), June 2010, pp. 192–197.
[18] D. Lunga, O. Ersoy, Kent Mixture Model for Hyperspectral Clustering via Cosine Pixel Coordinates on Spherical Manifolds, vol. 407, ECE Technical Reports, Purdue University, 2011, pp. 1–18.
[19] T. Villmann, E. Merényi, B. Hammer, Neural maps in remote sensing image analysis, Neural Networks 16 (3–4) (2003) 389–403.
[20] E. Merényi, W.H. Farrand, J.V. Taranik, T.B. Minor, Classification of hyperspectral imagery with neural networks: comparison to conventional tools, in: T. Villmann, F.-M. Schleif (Eds.), Machine Learning Reports, vol. 5, 2011, pp. 1–15, ISSN:1865-3960. on-line / http://www.techfak.uni-bielefeld.de/~schleif/mlr/mlr_04_2011.pdfS. Also submitted to EURASIP Journal on Advances in Signal Processing.
[21] R.O. Green, Summaries of the 6th Annual JPL Airborne Geoscience Workshop, 1. AVIRIS Workshop, Pasadena, CA, 4–6 March 1996.
[22] J. Rasure, M. Young, An open environment for image processing software development, in: Proceedings of the SPIE/IS&T Symposium in Electronic Imaging, vol. 1659, Pasadena, CA, 14 February 1992.
[23] NeuralWare, Neural Computing, NeuralWorks Professional II/PLUS, 2003.
[24] B.M. Csathó, W.B. Krabill, J. Lucas, T. Schenk, A multisensor data set of an urban and coastal scene, in: International Archives of Photogrammetry and Remote Sensing, vol. 32, 1998, pp. 26–31.
[25] E. Merényi, B. Csató, K. Tasdemir, Knowledge discovery in urban environments from fused multi-dimensional imagery, in: P. Gamba, M. Crawford (Eds.), Proceedings of the IEEE GRSS/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas (URBAN 2007), Paris, France, 11–13 April 2007, pp. 1–13.
[26] M. Porrmann, U. Witkowski, U. Rückert, A massively parallel architecture for self-organizing feature maps, IEEE Trans. Neural Networks 14 (5) (2003) 1110–1121.
[27] H. Hikawa, FPGA implementation of self organizing map with digital phase locked loops, Neural Networks 18 (5) (2005) 514–522, IJCNN 2005.
[28] A. Ramirez-Agundis, R. Gadea-Girones, R. Colom-Palero, A hardware design of a massive-parallel, modular NN-based vector quantizer for real-time video coding, Microprocess. Microsyst. 32 (1) (2008) 33–44.
[29] D.C. Hendry, A.A. Duncan, N. Lightowler, IP core implementation of a self-organizing neural network, IEEE Trans. Neural Networks 14 (September (5)) (2003) 1085–1096.

J. Lachmair graduated as "Diplom-Ingenieur" in Electrical Engineering at the University of Paderborn, Germany, in 2011. Currently, he is a member of the research group Cognitronics and Sensor Systems, Center of Excellence Cognitive Interaction Technology, Bielefeld University. Jan Lachmair's main scientific interests are in resource-efficient massive-parallel hardware architectures for artificial neural networks and their applications to mobile systems.

E. Merényi received her M.Sc. in Mathematics (1975) and Ph.D. in Computational Science (1980) at Szeged (Attila József) University, Hungary. From 2000, she holds a joint appointment as a research professor in the Departments of Statistics, and Electrical and Computer Engineering, Rice University, Houston, Texas, USA, where she teaches neural computing, statistics, probability, and remote sensing courses. Her current interests focus on artificial neural networks, self-organized learning, manifold learning, segmentation and classification of high-dimensional patterns, data fusion, data mining, knowledge discovery, and application to information extraction from multi- and hyperspectral remote sensing imagery from Earth and other planets, hyperspectral imagery of biological tissues, and from multi-variate medical data.

M. Porrmann is an Academic Director in the research group Cognitronics and Sensor Systems, Center of Excellence Cognitive Interaction Technology, Bielefeld University. He is graduated as "Diplom-Ingenieur" in Electrical Engineering at the University of Dortmund, Germany, in 1994. In 2001 he received a Ph.D. in Electrical Engineering from the University of Paderborn, Germany, for his work on performance evaluation of embedded neurocomputers. From 2001 to 2009 he was "Akademischer Oberrat" and from 2010 to March 2012 he was an Acting Professor of the research group System and Circuit Technology at the Heinz Nixdorf Institute, University of Paderborn. Mario Porrmann's main scientific interests are in on-chip multiprocessor systems, dynamically reconfigurable hardware and resource-efficient computer architectures.

U. Rückert received the Diploma degree in Computer Science and a Dr.-Ing. degree in Electrical Engineering from the University of Dortmund, Germany, in 1984 and 1989, respectively. From 1985 to 1994 he worked on microelectronic implementation of neural networks at the Faculty of Electrical Engineering (University of Dortmund) and at the Technical University of Hamburg-Harburg, Germany. In 1995 he joined as a Full Professor at the Heinz Nixdorf Institute, University of Paderborn, Germany, heading the research group "System and Circuit Technology" and working on microelectronic systems for massive-parallel and resource-efficient information processing. Since 2009 he is a Professor at Bielefeld University, Germany, heading the "Cognitronics and Sensor Systems" group of the "Cluster of Excellence – Cognitive Interaction Technology". His main research interests are now bio-inspired architectures for nanotechnologies and cognitive robotics.