Circuits and Systems, 2016, 7, 2132-2141
Published Online July 2016 in SciRes. http://www.scirp.org/journal/cs
http://dx.doi.org/10.4236/cs.2016.79185

FPGA Implementation of a Scalable and Highly Parallel Architecture for Restricted Boltzmann Machines

Kodai Ueyoshi[1], Takao Marukame[2], Tetsuya Asai[1], Masato Motomura[1], Alexandre Schmid[2]

[1]Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan
[2]Swiss Federal Institute of Technology, Lausanne, Switzerland
Email: ueyoshi@lalsie.ist.hokudai.ac.jp, takao.marukame@epfl.ch, asai@ist.hokudai.ac.jp, motomura@ist.hokudai.ac.jp, alexandre.schmid@epfl.ch

Received 11 April 2016; accepted 5 May 2016; published 8 July 2016

Abstract

Restricted Boltzmann Machines (RBMs) are an effective model for machine learning; however, they require a significant amount of processing time. In this study, we propose a highly parallel, highly flexible architecture that combines small and completely parallel RBMs. This proposal addresses problems associated with calculation speed and exponential increases in circuit scale. We show that this architecture can optionally respond to the trade-offs between these two problems. Furthermore, our FPGA implementation performs at a 134 times processing speed up factor with respect to a conventional CPU.

Keywords

Deep Learning, Restricted Boltzmann Machines (RBMs), FPGA, Acceleration

1. Introduction

Restricted Boltzmann Machines (RBMs) are an important component of Deep Belief Networks (DBNs). Moreover, DBNs have achieved high-quality results in many pattern recognition applications [1]-[3]. Therefore, many researchers are actively studying them, and further development is expected. However, as RBM scale increases, the amount of required processing also exponentially increases. As a result, calculations on a conventional CPU require a significant amount of time, which contributes to limited efficiency. Following this observation, the processing speed can be increased using specific hardware circuits. To achieve this goal, various RBM architectures have been proposed. Kim *et al.* [4] have proposed an architecture supporting large-scale RBMs suitable for implementation on multiple field-programmable gate arrays (FPGAs), and Ly and Chow [5] proposed a reconfigurable architecture suitable for implementation on single or multiple FPGAs.

However, these architectures do not fully exploit the high parallelism of the neural networks, because each layer is processed sequentially. Higher parallelism requires a substantial circuit scaling, resulting in a trade-off between calculation speed and circuit scale. Therefore, we focus on high parallelism in this study. Furthermore, to address this trade-off, a high scalability is required, which must be optimally designed to satisfy the requirements of the designers.

In this study, we have devised an architecture that divides RBMs into blocks, to prevent exponential circuit scale increases and sequential calculations with respect to the number of blocks. Each RBM block can perform completely parallel calculations. The trade-off according to the circuit scale of each block has been verified in the following.

Section 2 describes the RBM algorithm. Section 3 explains the architecture proposed in this study. Section 4 studies the necessary bit precision of our architecture. Section 5 shows the register-transfer level (RTL) simulation results and FPGA implementation results. Section 6 provides our conclusions.

2. Restricted Boltzmann Machines

RBMs are a stochastic neural network model consisting of two layers (a visible layer and a hidden layer). As shown in **Figure 1**, it is an undirected graph model in which the neurons in one layer are fully connected to the neurons in a second layer. RBMs are configured from three parameters, *i.e*, the connection weights, visible biases, and hidden biases. RBMs learn in an unsupervised process, as they are updated.

Different pairwise potential functions [6]-[9] determine various types of RBMs. In this study, we use the binary RBMs which are the most fundamental and common type of RBMs [6] [9]. Hidden and visible nodes are binary. The joint distribution obeys the following model:

$$p(v,h) = \frac{1}{z} e^{-E(v,h)} \tag{1}$$

$$E(v,h) = -\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} h_i v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i \tag{2}$$

$$Z = \sum_{v}\sum_{h} e^{-E(v,h)} \tag{3}$$

where E is the energy function, w is the weight between the visible layer and hidden layer, b is the visible bias, and c is the hidden bias. Because the connection of each neuron is restricted to the neurons in the complementary layer, the probability of firing of each neuron is given by

$$p(h_i = 1 | v) = sigmoid\left(\sum_{j=1}^{m} w_{ij} v_j + c_i\right) \tag{4}$$

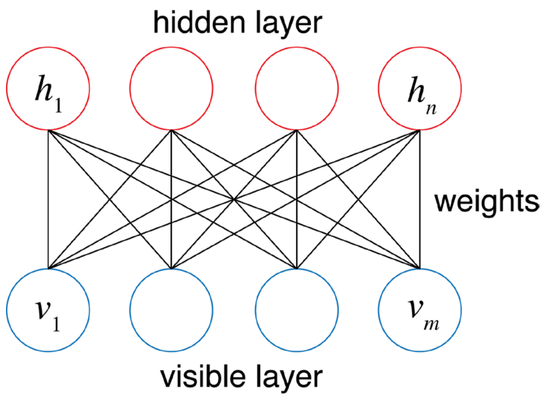


Figure 1. RBM model; in this case, $n = m = 4$.

$$p\left(v_j = 1 \middle| h\right) = sigmoid\left(\sum_{i=1}^{n} w_{ij} h_j + b_i\right) \tag{5}$$

$$sigmoid(x) = \frac{1}{1-e^{x}} \tag{6}$$

The RBM processing flow consists of two repeating steps. First, input data is added to the visible layer, and the hidden layer is calculated using the visible layer's values as input. At this point, all visible layer and hidden layer combinations are calculated. Second, the visible layer is calculated, using sampling results from the hidden layer as input. By repeating these steps, it is possible to approximate the update formula given by

$$\Delta w_{ij} = w_{ij} + \eta\left\{p\left(h_i = 1 \middle| v^{(0)}\right) v_j^{(0)} - p\left(h_i = 1 \middle| v^{(k)}\right) v_j^{(k)}\right\} \tag{7}$$

$$\Delta b_j = b_j + \eta\left\{v_j^{(0)} - v_j^{(k)}\right\} \tag{8}$$

$$\Delta c_i = c_i + \eta\left\{p\left(h_i = 1 \middle| v^{(0)}\right) - p\left(h_i = 1 \middle| v^{(k)}\right)\right\} \tag{9}$$

where η is learning rate and k is the sampling number. This update formula can be obtained by using contrastive divergence (CD) learning [6]. In this study, we use this algorithm to design the proposed architecture.

To construct the DBNs, a hidden layer that has completed learning is used as a visible layer for the next RBMs. Deep networks are constructed by stacking these RBMs. After using back-propagation to perform fine-tuning, the DBNs are completed [2].

3. Proposed Architecture

In this Section, we describe the overall data flow and specifications of the proposed architecture.

An overall diagram of the architecture is presented in **Figure 2**. First, input data to the RBM unit is obtained from the input buffer, and CD learning computation is repeated within this unit. Simultaneously, the learning update formula is calculated for the update unit, and is stored in the local memory of the update unit. When a learning process is completed, the learning data of the local memory of the update unit is added to the local memory of the RBM unit. These operations are controlled by a common finite-state machine (FSM), controller, and a linear feedback shift register (LFSR).

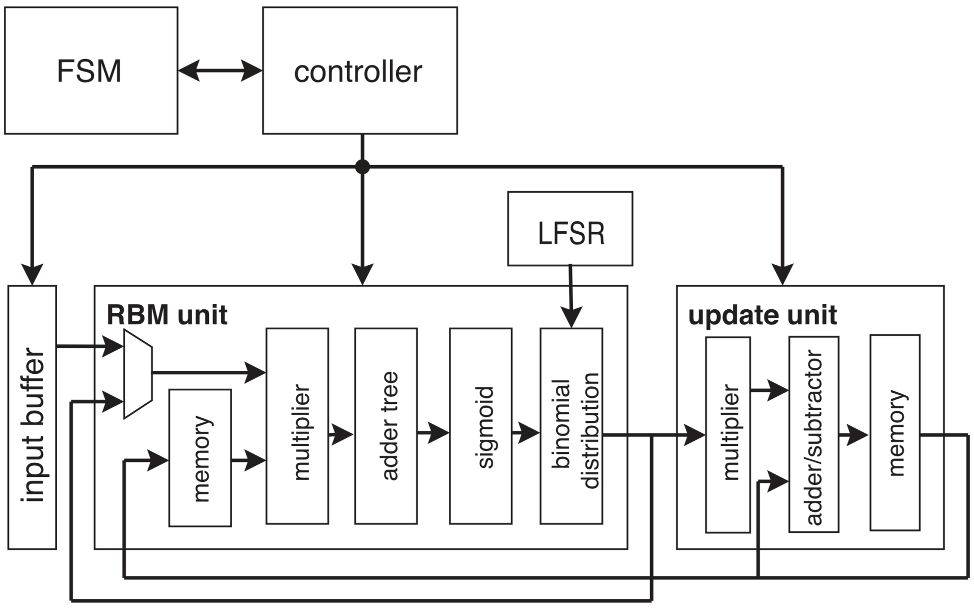


Figure 2. Overall data-flow.

Input data is assumed to consist of unsigned 8-bit fixed-point numbers representing continuous values from 0 through 1. The bit precision of other parameters is discussed in detail in Section 4. This proposed architecture implements the most basic binary model. We have adopted this model as a foundation, because this model, although it is more suitable for binary data, can be utilized with continuous data and other models after minor modifications. Using binary data as input, every multiplier can be replaced with an AND operation.

In this Section, we present an example that consists of four RBM blocks with four inputs and four outputs. A detail block-diagram using RBMs divided into 4 blocks is shown in **Figure 3**. Three phases occur, which are physically separated by a shift register. These registers store and propagate the value of each block, and also play the role of the pipeline registers. Phase 1 implements the sum-product operation, phase 2 the sampling operation, and phase 3 the parameter update operation. The operation of each phase is presented in the following.

3.1. Product-Sum Calculation

In Phase1, each RBM block multiplies all the inputs and connection weights in parallel, and calculates the respective outputs using an adder tree. This approach is possible because each RBM block is configured on a small scale. Each RBM block has its own local memory to save parameters, which are only used inside each block.

Each shift controller is responsible of shifting propagated inputs and outputs to calculate all combinations. This movement of data is different in the $\boldsymbol{v} \rightarrow \boldsymbol{h}$ operation and the $\boldsymbol{h} \rightarrow \boldsymbol{v}$ operation. **Figure 4** shows the operation diagram of Phase1, where (a) pertains to the $\boldsymbol{v} \rightarrow \boldsymbol{h}$ operation and (b) to the $\boldsymbol{h} \rightarrow \boldsymbol{v}$ operation. The bit width and the number of words in each local memory area are determined by the number of RBM blocks and the I/O of each block. During the $\boldsymbol{v} \rightarrow \boldsymbol{h}$ operation, the inputs are shifted between RBM blocks to match the local memory address. During the $\boldsymbol{h} \rightarrow \boldsymbol{v}$ operation, the accumulated outputs are shifted between RBM blocks. In this case,

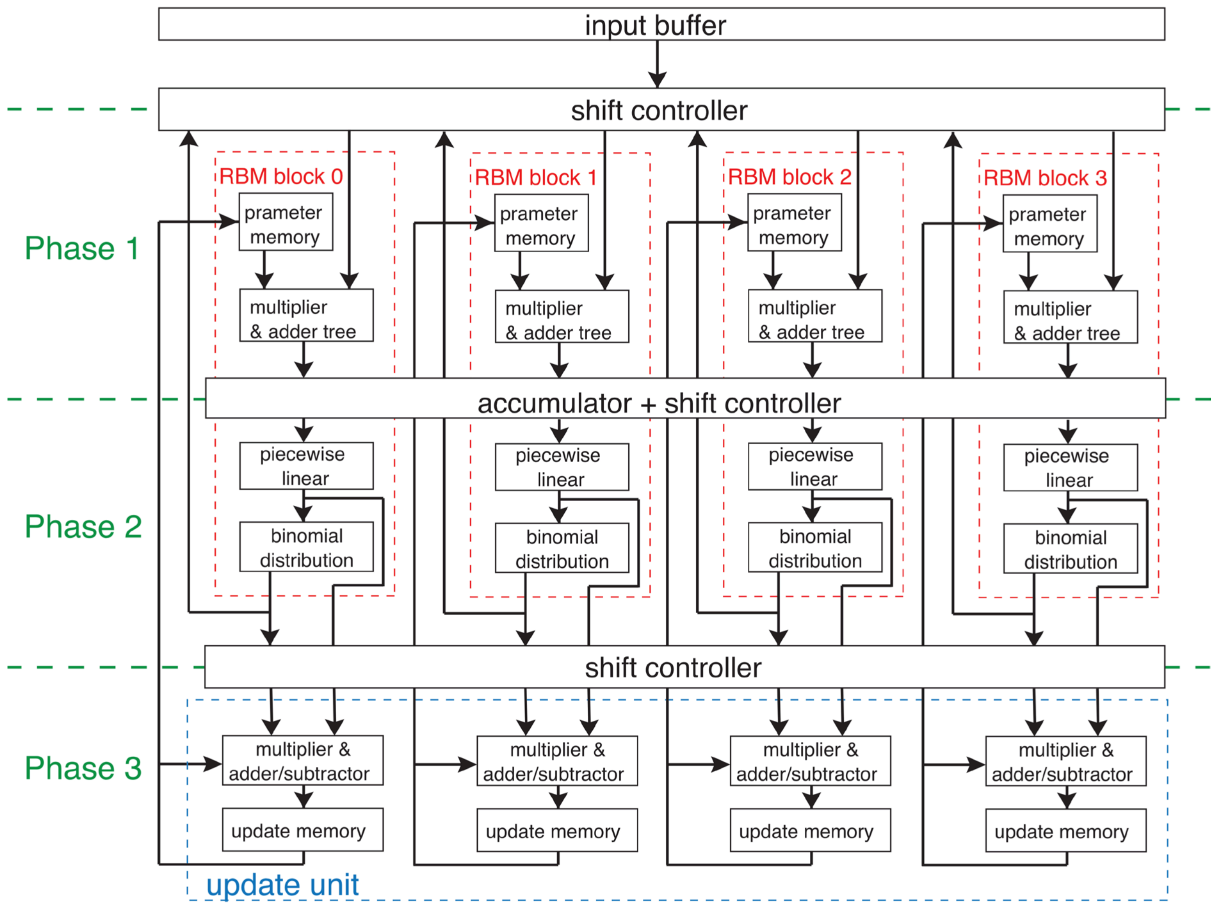
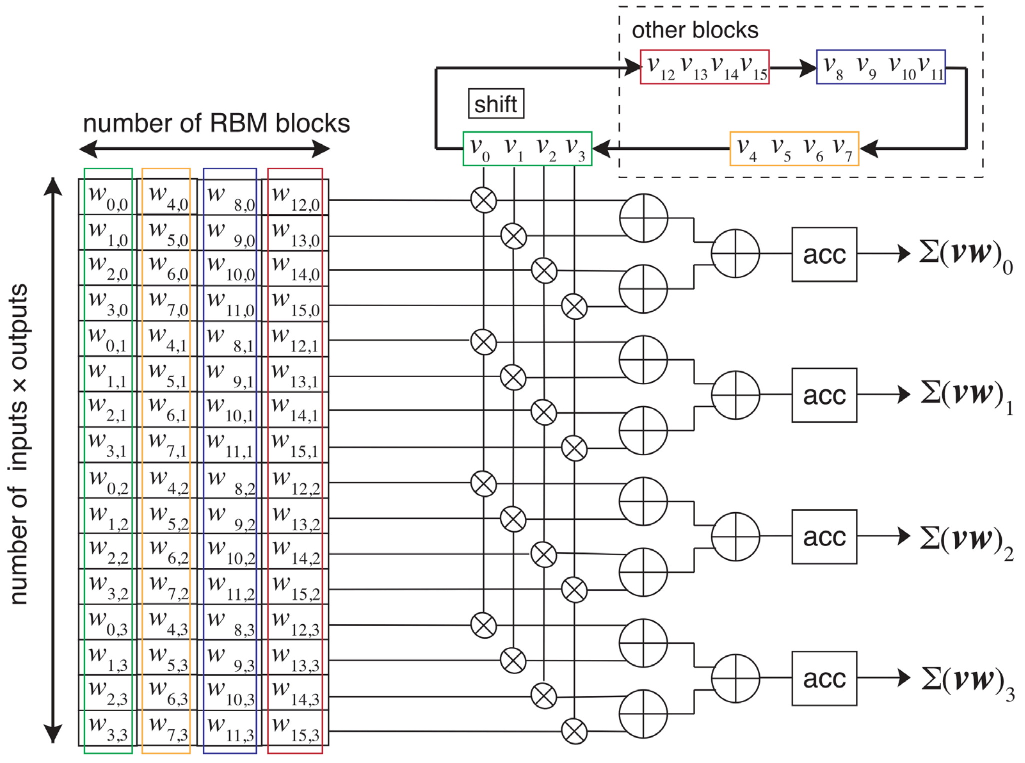
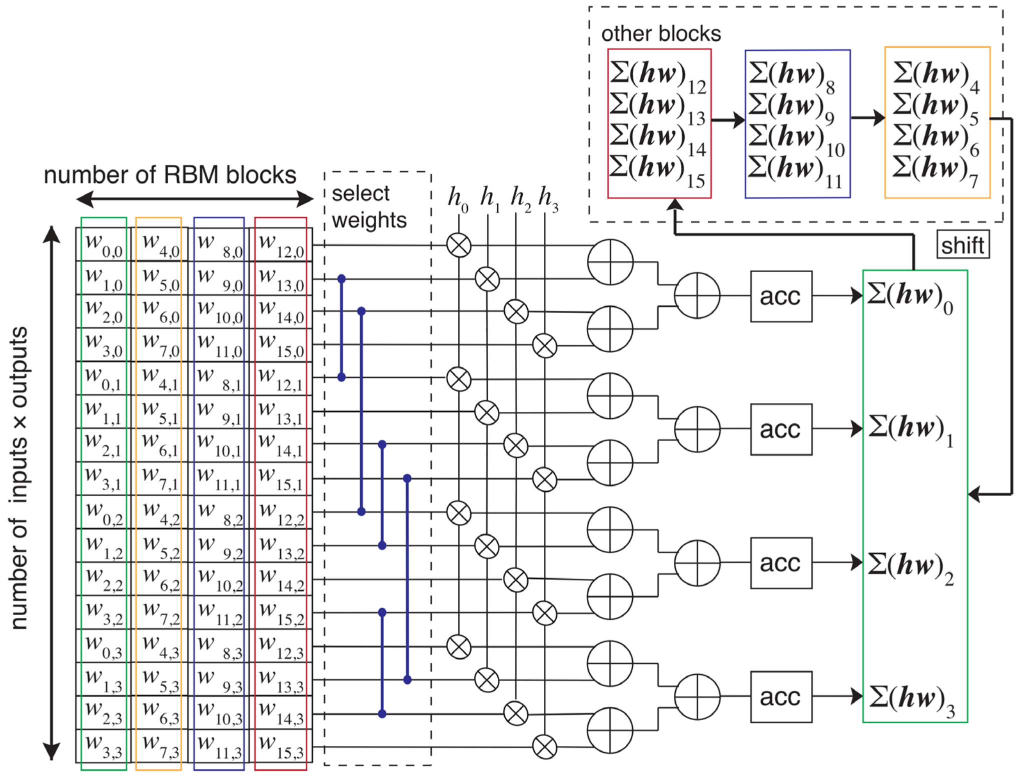


Figure 3. Detail block-diagram of the proposed architecture.



(a)



(b)

Figure 4. Shift I/O data and select connection weights flow; the color code suggests the operation order. (a). *v* to *h* processing. (b). *h* to *v* processing.

because the arrangement sequence of the connection weights from the local memory is not identical to the $\boldsymbol{v} \rightarrow \boldsymbol{h}$ case, the appropriate connection weights must be re-selected. To this aim, the Fredkin gate (**Figure 5**) is used to select the correct combinations, as shown in **Figure 4(b)** (as a $w_{ij} = w_{ji}$ in first column); hence, the Fredkin gate is placed at the location of every vertical connection line of the select weights array in **Figure 4(b)**. Considering n inputs and n outputs, the number of required Fredkin gates equals $(n^2-n)/2$. This approach enables calculating data propagation in both ways using one single circuit, and is possible because each RBM block is configured on a small scale.

3.2. Sampling Calculation

In Phase 2, the sigmoid calculation and binomial distribution calculation are performed. After completion of the sum-product operation, a sigmoid function is applied to the sum value. This result is processed by the sampling operation which yield the output of neurons as a probability of neuronal firing as expressed in Equations (4-6). The sigmoid function is approximated as a piecewise linear function to facilitate the implementation on hardware. The piecewise linear function is given by

$$f(x) = \begin{cases} 0 & (x \le -2) \\ \frac{1}{4}x + \frac{1}{2} & (-2 < x \le 2) \\ 1 & (2 < x) \end{cases} \tag{10}$$

where x is the input of the sigmoid function. In order to realize this equation, an arithmetic shifter [10] is used instead of a divider since 1/4 equals 2^{-2}. The corresponding circuit schematic is shown in **Figure 6**. A binomial distribution calculation is also obtained from a subsequent circuit that determines the output as 0 or 1 by comparing the output of the sigmoid function to random numbers generated by an LFSR. **Figure 7** shows the overall sampling operation circuit schematic.

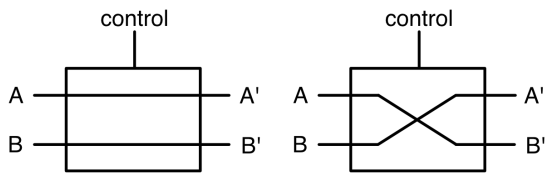


Figure 5. Fredkin gate.

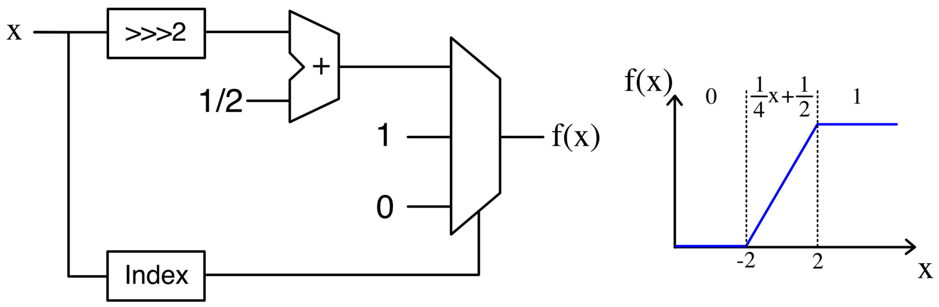


Figure 6. Circuit schematic of the sigmoid function as a piecewise linear function.

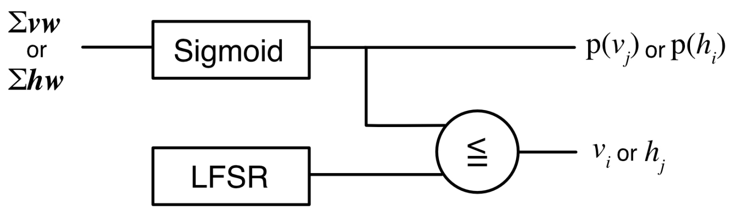


Figure 7. Circuit schematic of the sampling operation.

3.3. Parameter Update Calculation

In Phase 3, the parameter update calculation is performed. While repeating propagation between the visible layer and hidden layer during the learning process, the update unit calculates the update formula in Equations (7)-(9). **Figure 8** shows the circuit schematic of the update operation for weight parameters. Because the RBM blocks calculate in a completely parallel manner, the update unit must have a completely parallel architecture to receive and process the data. In addition, because the update unit must calculate all parameters, it resembles the structure of Phase 1. It contains a local memory to store the update data for the RBM unit; these values are constantly updated during the learning process. Multipliers can efficiently be replaced by AND operators when using binary input data. After all parameters are calculated, the data of the update memory is added to the data of the parameter memory. Multiplication with the learning rate is carried out prior to addition to the parameter memory. The learning rate is quantized to an integer power of 2 (like 1, 1/2, 1/4...), such that multiplication with the learning rate can be approximated to arithmetic shifts.

4. Evaluation of the Necessary Bit Precision

The necessary bit precision of parameters which are weights, visible biases and hidden biases has been evaluated using a custom emulator written in a C code which can accurately simulate the proposed architecture in terms of data process and bit precision. The MNIST dataset [11] consisting of a popular hand-written digits benchmark is used in this study. The training set includes 60,000 images for training and 10,000 images for testing. We modified all images to binary values and to a size of 16 × 16 pixels to fit our architecture, and developed a 5 layer network of (256-256-256-256-10) neurons. These parameters were selected as a suitable example, without restriction of the generality of the proposed method. The bit-precision evaluation results are presented in **Figure 9**. The number of integer bits is fixed to 8-bit, and the number of fractional bits is parameterized from 8-bit to 20-bit. The vertical axis shows the accuracy of classification of hand-written digits from 0 to 9 using 10,000 test images. An excess of 12-bit fractional bit precision of each parameter is observed to be required, while applying fine-tuning improves the accuracy at lower fractional bit-precision.

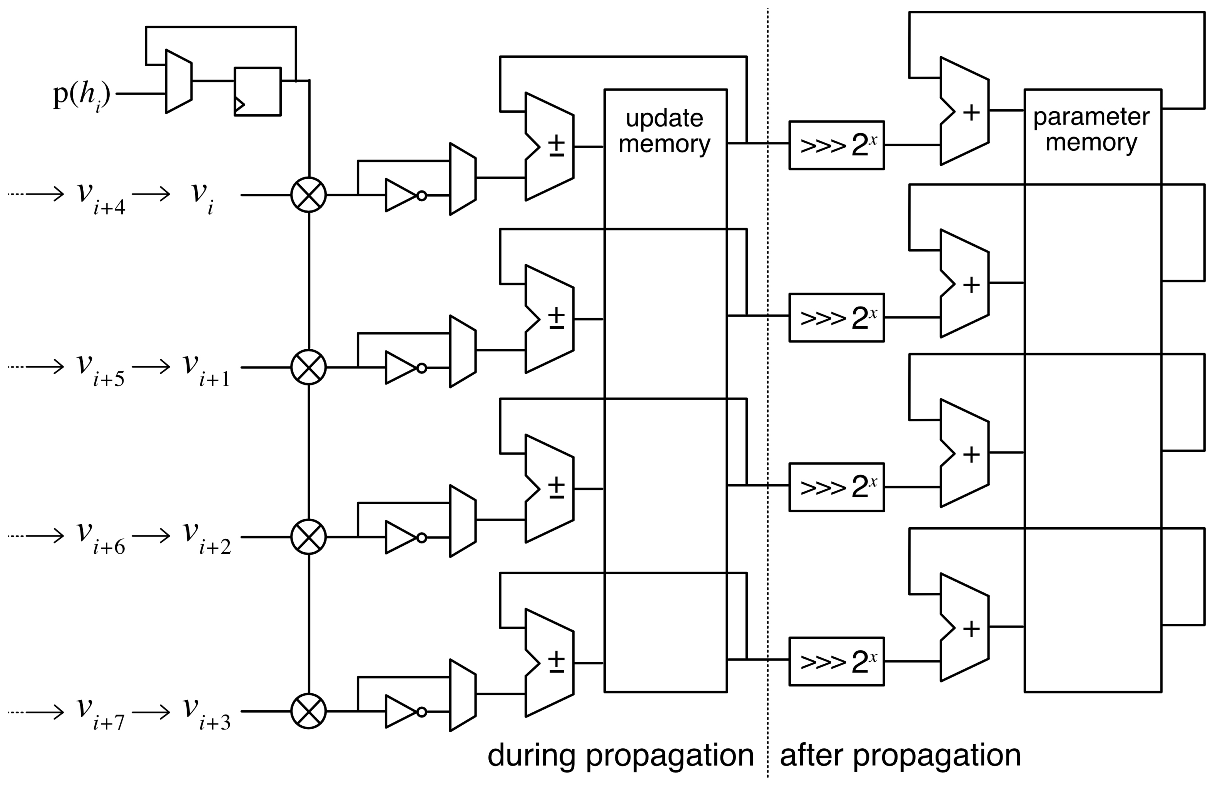


Figure 8. Circuit schematic of the update operation for weight parameters.

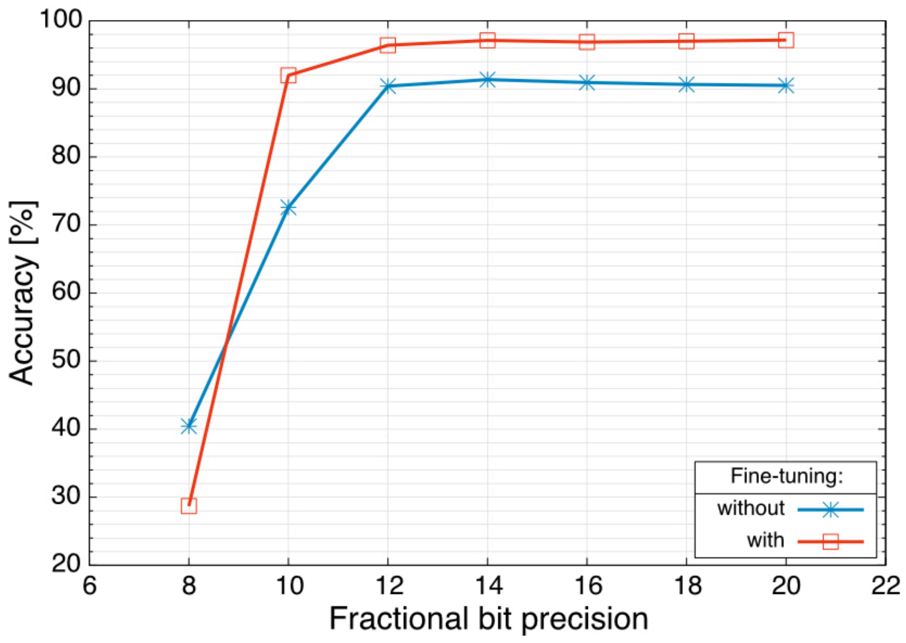


Figure 9. Accuracy of the classification with respect to the fractional bit precision, also considering the fine-tuning procedure.

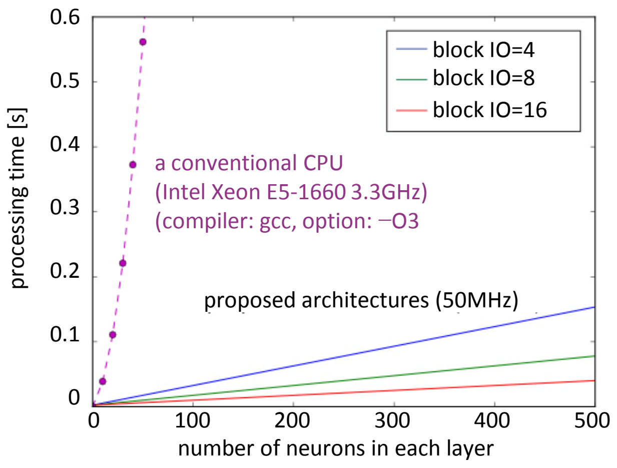
5. Results

We simulated the RTL model (coded in Verilog HDL) of the proposed architecture using ModelSim. The simulation conditions were set as follows. The size ratio of the visible and hidden layers of the RBMs is equal to 1:1. We confirmed the model's speed and scale characteristics by changing the data size (=the number of neurons in each layer). Four types of models were considered, in which the number of inputs and outputs for each RBM block were 4, 8, 16, and finally in which the size of the data that was not divided into blocks. **Figure 10(a)** shows the processing time of the dataset in which the number of training data batches is 100, the number of learning iterationsis 100, and the number of CD learning repetitions is limited to 1 with respect to the data size (speed characteristics). For each developed model, time is linear with respect to the data size, whereas the conventional CPU operation of an Intel Xeon E5-1660 3.3 GHz has an exponential characteristic. **Figure 10(b)** shows the required number of multipliers with respect to the data sizes (scale/resource characteristics). Because multipliers utilize most of the circuit area in this architecture, the number of multipliers directly affects the circuit scale. The models partitioned into blocks exhibit relatively small linear increases, whereas the non-partitioned model exponentially increases.

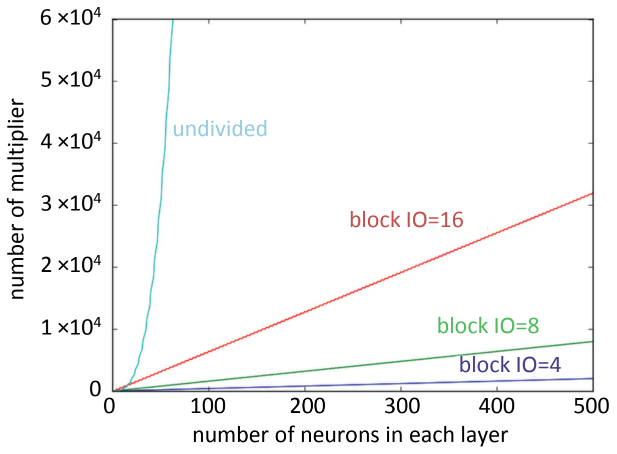
The proposed architecture was implemented on a commercial FPGA board (DE3 with an Altera Stratix III FPGA). The performance of the proposed architecture is compared with other FPGA implementations (**Table 1**). Our hardware shows fast speed in terms of CUPS (connection updates per seconds) even while using a slower FPGA clock rate than other developments. Moreover, because of the linear scalability, the speed can be adapted to unit RBM configurations. In addition, we compared our implementation results with a conventional CPU (**Table 2**). The software implementations are realized in 1 thread using gcc with –O3 optimization options. The modified MNIST benchmark detailed in Section 4 is used. Twenty learning times and 100 batch size of data were simulated, showing that our implementation provides a 134× speed up factor with respect to the CPU software implementation.

6. Conclusions

From these results, we conclude that the proposed architecture could reduce circuit scale more effectively than simple parallelism. Furthermore, it could achieve both speed and circuit scaling in a linear manner with respect to the network size in terms of numbers of neurons. Consequently, designers may select the trade-off between speed and circuit scaling which is suitable to their requirements. In addition, the proposed architecture has been



(a)



(b)

Figure 10. Architectural simulation of processing speed and resource requirements. (a) Speed characteristics. (b) Resource characteristics.

Table 1. Performance summary of RBMs implemented on FPGAs.

	This work (2016)	Kim *et al.* (2009) [12]	Ly & Chow (2010) [5]		Kim *et al.* (2014) [13]
Platform	EP3SL340	EP3SL340	XC2VP70		XC6VSX760
# of chips	1	1	1	4	1
Network	256 × 256	256 × 256	128 × 128	256 × 256	256 × 256
Clock [MHz]	50	200	100		80
GCUPS[a]	16.6 (4 × 4)[b] 66.4 (16 × 16)[b]	N/A	1.6	3.1	59.6

a. CUPS = the number of weight/time for a learning step. b. Network size of a RBM block (16 × 16 is assumed for estimation).

Table 2. Performance comparison with a CPU.

	Intel Xeon 3.3GHz 1 thread –O3	Our FPGA implementation (block IO is 4 × 4)
Time [ms]	627,500	4753
Speed up	1×	134×

implemented on FPGA showing a 134× speed up factor with respect to a conventional CPU, and much faster operation compared to existing FPGA developments.

References

[1] Hinton, G. and Salakhutdinov, R.R. (2006) Reducing the Dimensionality of Data with Neural Networks. *Science*, **313**, 504-507. http://dx.doi.org/10.1126/science.1127647

[2] Hinton, G., Osindero, S. and Teh, Y. (2006) A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, **18**, 1527-1554. http://dx.doi.org/10.1162/neco.2006.18.7.1527

[3] LeCun, Y., Bengio, Y. and Hinton, H. (2015) Deep Learning. *Nature*, **521**, 436–444. http://dx.doi.org/10.1038/nature14539

[4] Kim, S., McMahon, P. and Olukotun, K. (2010) A Large-Scale Architecture for Restricted Boltzmann Machines. *Proceedings of the* 18*th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines* (*FCCM*), Charlotte, NC, 2-4 May 2010, 201-208. http://dx.doi.org/10.1109/fccm.2010.38

[5] Ly, D. and Chow, P. (2010) High-Performance Reconfigurable Hardware Architecture for Restricted Boltzmann Machines. *IEEE Transactions on Neural Networks*, **21**, 1780-1792. http://dx.doi.org/10.1109/TNN.2010.2073481

[6] Hinton, H. (2002) Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, **14**, 1771-1800. http://dx.doi.org/10.1162/089976602760128018

[7] Welling, M. and Sutton, C. (2005) Learning in Markov Random Fields with Contrastive Free Energies. *Proceedings of the* 10*th International Workshop on Artificial Intelligence and Statistics* (*AISTATS*), Barbados, 6-8 January 2005, 397-404.

[8] Salakhutdinov, R.R., Mnih, A. and Hinton, G. (2007) Restricted Boltzmann Machines for Collaborative Filtering. *Proceedings of the International Conference on Machine Learning*, **24**, 791-798. http://dx.doi.org/10.1145/1273496.1273596

[9] Fischer, A. and Igel, C. (2014) Training Restricted Boltzmann Machines: An Introduction. *Pattern Recognition*, **47**, 25-39. http://dx.doi.org/10.1016/j.patcog.2013.05.025

[10] Nitta, Y. and Nakamura, K. (1989) US Patent 4,890,251. Washington, DC: U.S.

[11] LeCun, Y., Bottou, L.,Bengio, Y. and Haffner. P. (1998) Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, **86**, 2278-2324. http://dx.doi.org/10.1109/5.726791

[12] Kim, S.K., MacAfee, L.C., McMahon, P.L. and Olukotun, K. (2009) A Highly Scalable Restricted Boltzmann Machine FPGA Implementation. *Proceedings the International Conference on Field Programmable Logic and Applications*, Prague, 31 August 2009-2 September 2009, 367-372. http://dx.doi.org/10.1109/fpl.2009.5272262

[13] Kim, L.W., Asaad, S. and Linsker, R. (2014) A Fully Pipelined FPGA Architecture of a Factored Restricted Boltzmann Machine Artificial Neural Network. *ACM Transactions on Reconfigurable Technology and Systems*, **7**, 1-23. http://dx.doi.org/10.1145/2539125