2015 28th International Conference on VLSI Design and 2015 14th International Conference on Embedded Systems

A Flexible Scalable Hardware Architecture for Radial Basis Function Neural Networks

Mahnaz Mohammadi*, Nitin Satpute*, Rohit Ronge*, Jayesh Ramesh Chandiramani*, S. K. Nandy*
Aamir Raihan†,Tanmay Verma† , Ranjani Narayan‡ and Sukumar Bhattacharya§
*Indian Institute of Science, Bangalore, India
Email: {mahnaz, nitin, rohit, jayesh}@cadl.iisc.ernet.in, nandy@serc.iisc.in
†Indian Institute of Technology, Varanasi, India
Email: {aamir.raihan.ece11, tanmay.verma.ece11}@itbhu.ac.in
‡Morphing Machines Pvt. Ltd. ,Bangalore, India
Email: ranjani.narayan@morphingmachines.com
§ Indian Institute of Technology, Mandi, Himachal Pradesh, India
Email:Sukumar.Bhattacharya@acm.org

***Abstract*—Radial Basis Function Neural Networks (RBFNN) are used in variety of applications such as pattern recognition, control and time series prediction and nonlinear identification. RBFNN with Gaussian Function as the basis function is considered for classification purpose. Training is done offline using K-means clustering method for center learning and Pseudo inverse for weight adjustments. Offline training is done since the objective function with any fixed set of weights can be computed and we can see whether we make any progress in training. Moreover, minimum of the objective function can be computed to any desired precision, while with online training none of these can be done and it is more difficult and unreliable. In this paper we provide the comparison of RBFNN implementation on FPGAs using soft core processor based multi-processor system versus a network of HyperCells [8], [13]. Next we propose three different partitioning structures (Linear, Tree and Hybrid) for the implementation of RBFNN of large dimensions. Our results show that implementation of RBFNN on a network of HyperCells using Hybrid Structure, has on average 26x clock cycle reduction and 105X improvement in the performance over that of multiprocessor system on FPGAs.**

Keywords. Multi Processor System on Chip, Pattern Recognition, Radial Basis Function Neural Network, Reconfigurable Architecture.

I. Introduction

Artificial Neural Networks (ANNs) [1] are computational simulation of biological neurons, made up of simple and highly interconnected processing elements, arranged in layers and capable of producing outputs through processing information by their dynamic state response to external outputs.

Radial Basis Function Neural Network (RBFNN) [2] is a special type of feedforward neural network with a simple topological structure and three learning phases [3], composed of three layers:

- Input Layer: Each node in the input layer represents a dimension of the input.
- Hidden Layer: This layer has a variable number of neurons (the optimal number is determined by the training process). Each neuron consists of a radial basis function centered on a point with as many dimensions as the input. The spread (radius) of the RBF function may be different for each dimension. The centers and spreads are determined by the training process. When presented with vector x of input values from the input layer, a hidden neuron computes the Euclidean distance of the test case from center point of the neurons and then applies the RBF kernel function to this distance using the spread values. The resulting value is passed to the the output layer.
- Output Layer: The value coming out of a neuron in the hidden layer is multiplied by a weight associated with the neuron and passed to the summation unit which adds up the weighted values and presents this sum as the output of the network.
  Figure 1 shows the architecture of RBFNN. While similar to back propagation in many respects, RBFNN has several advantages:
  - RBFNN architecture is simpler as there is only one hidden layer compared to other feed-forward neural networks with multi hidden layers.
  - RBFNN usually trains faster than back propagation networks.
  - RBFNN is less susceptible to problems with non-stationary inputs because of the behavior of the hidden units in RBFNN .

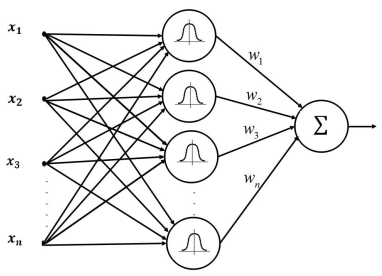

Fig. 1: Radial Basis Function Neural Network Architecture. ["$w_1, w_2, ..., w_n$" are corresponding weights between hidden layer and output layer.]

There are two fundamentally different alternatives for implementing neural networks: a software simulation in conventional computers and a special hardware solution capable of decreasing execution time. A software simulation is useful to develop and debug new algorithms, as well as to benchmark them using small data sets. In hardware implementation, hardware specification is the first step in selecting a hardware solution of ANNs and includes the type of the ANN, number of neurons, number of inputs and outputs, number of connections to each neuron, precision, speed of operation or performance and other characteristics that can be important depending on the application. The precision used should be an important parameter to take into account.

However, if large data sets are used, a software simulation is not fast enough. Hardware implementation of neural techniques has a significant number of advantages, mainly in the processing speed. For networks with large numbers of neurons and synapses, the conventional processors are not able to provide real time responses and training capabilities, while parallel processing of multiple simple procedures achieves a large increase in speed. Specialized hardware can offer very high computational power at limited price and thus can achieve several orders of speed-up, especially in the neural domain where parallelism and distributed computing are inherently involved [4], [5], [7].

Researchers developed implementations of ANN on Application-Specific Integrated Circuits (ASIC). ASICs are responsible for the evolution of computer systems from workstations to hand-held devices that need real-time performance within the budget for physical size and energy dissipation. However, these circuits are inflexible as any modification requires redesign and refabrication, which is both expensive and time consuming considering the complexity of recent embedded platforms. Therefore, reconfigurable architectures (RAs) that can be dynamically reconfigured and reused, is suggested and they are known to provide high performance in a wide range of applications.

RAs [8], [13] are devices that contain programmable functional blocks and programmable interconnects between functional blocks. Spatial distribution of functional blocks in conjunction with a flexibility of interconnect, allows exploiting various forms of parallelism inherent in the application. In comparison with the programmability provided by Instruction Extension architectures (IE) [8], [12], [13], the programmability provided by RAs allow substantial changes to the datapath itself. Hence, as with dedicated architectures, RAs can implement application-specific computing structures without sacrificing flexibility. Traditional RAs, such as FPGAs [6], provide interconnect structures and functional blocks that operate at bit-level. Hence, they are able to realize datapaths and controllers with arbitrary word lengths. For algorithms that are based on operands represented with multiple bits, bit-level reconfigurability results in large overhead in terms of area, delay, energy, and configuration time.

HyperCell (HC) [8], [13] is a reconfigurable architecture that consists of Compute Units (CUs) and switches. The computations of an IE [13] are assigned to CUs and the switches are configured to connect CUs as per communication requirements within the IE. The IE synthesis methodology of HC ensures maximal utilization of resources on the reconfigurable datapath. The methodology for realizing IEs through HCs permits overlapping of potentially all memory transactions with computations and this introduces significant improvement in performance for streaming applications over general purpose processor based solutions, by fully pipelining the datapath. Post-silicon realization of IEs on HC entails synthesis of multiple-input and multiple-output (MIMO) macro operations on the same hardware datapath which avoids redesigning hardware datapaths for each individual choice of IEs. In this paper we target the realization of classification using RBFNN on a network of HCs by synthesizing RBFNN specific MIMO operations on HCs. This involves mapping and scheduling of RBFNN onto multiple HCs. Proposing and evaluating three different partitioning structures (Linear, Tree, Hybrid) on HyperCell and comparing the results of Hybird Structure on MPSoC and HyperCell are our contributions in this paper.

Rest of the paper is organized as follows. In section II mapping RBFNN on HC and two different implementation methods for that are shown. In section III different partitioning structure on HCs are discussed. In section IV RBFNN on soft core processor is emulated. In section V results of implementing RBFNN on HC and MB are presented. In section VI we conclude with a summary of the contribution of the paper.

II. Mapping of RBFNN on HyperCell

RBFNN computations require the computation of Euclidean Distance between RBF centers and the input pattern. Utilizing fully pipelined architecture of HyperCell, this computation can be mapped on HC in two ways: Pipelined Implementation and Parallel Implementation. In Pipelined Implementation, the Euclidean Distance calculation from different RBF centers are pipelined and in Parallel Implementation, Euclidean Distance calculation is done in parallel.

A. Pipelined Implementation

The Data Flow Graph (DFG) of RBFNN with N input nodes and C output nodes considering a center at a time is divided into two parts: The first part, which we call "Basis Generator", includes the operations needed for calculating Euclidean Distance squared between input and center and the radial basis function at the hidden node (Gaussian kernel). These operations are: N subtractions, N Squares, N-1 additions, 1 division and 1 exponentiation. The second part, called "Accumulator", includes the operation needed for output calculation (C multiplications and C additions). Therefore a total number of $3N + 1 + 2C$ computing nodes are needed to map the above mentioned operations on HC. Increase in the dimension of input will only affect the Basis Generator, and similarly, increase in the number of outputs (classes) will only affect the Accumulator. Therefore, mapping the Basis Generator and Accumulator separately will provide modularity.

In Pipelined Implementation a single Basis Generator is implemented on single HC or multiple HCs. This Basis Generator is used to calculate the Euclidean Distance from the input pattern to all RBF Centers in a pipelined way. The results for Pipelined Implementation are shown in section V.

B. Parallel Implementation

In this method we exploit the parallelism across centers. Different centers are mapped on different HCs and inputs are processed simultaneously. Implementation of Basis Generator on HC/HCs is/are similar to the Pipelined Implementation; instead of implementing one Basis Generator, multiple of them are implemented in parallel. Each Basis Generator generates the radial basis functions of different centers. Implementation of accumulator portion of DFG of RBFNN in this method is different from the Pipelined Implementation. As all the radial basis functions of different centers are available in parallel, we multiply and add in parallel. Though this results in substantial improvement in performance, it is very resource hungry. Additionally, due to constraints of available software tools for mapping and scheduling, we do not pursue this choice for implementing on HC (in this paper).

Figure 2 shows the DFG of four dimensional RBFNN. Operations performed in this figure are: Subtraction (SUB), Square (SQR), Addition (ADD), Division (DIV), Exponentiation (EXP) and Multiplication (MUL).

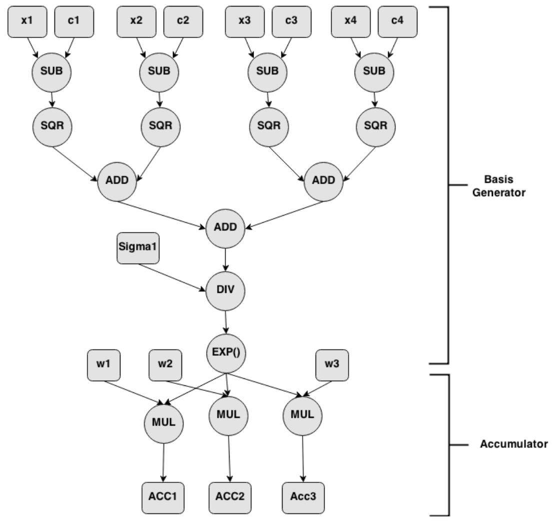

Fig. 2: DFG of 4 - dimensional RBFNN.
["$x_1, x_2, x3, x_4$" are input dimensions; "c_1, c_2, c_3, c_4" are center dimensions; "w_1, w_2, w_3" are weights between hidden layer and output layer; "$sigma1 = -2 * radius^2$" and "Acc_1, Acc_2, Acc_3" are outputs.]

III. Partitioning RBFNN on HyperCells

The amount of compute resources on a single HC is limited and hence only up to a certain dimension of inputs and outputs can be mapped on a HC. For mapping higher dimensions we have to partition the DFG of the RBFNN onto multiple HCs. Partitioning onto multiple HCs should be done in such a way that it satisfies the following conditions:

- Number of inter-HyperCell communications should be minimum.
- HyperCell resources should be utilized to the maximum.
- The partitioned DFG should not have more nodes having external inputs than the peripheral switches on the HC (Figure 3) as it would increase the latency due to switch traversal.

For Mapping RBFNN of N input dimensions and C output classes we require at least X HCs such that the total number of compute resource available on these X HCs is greater or equal to the number of compute nodes on the DFG of RBFNN considering a centre at a time.(The size of a HC is defined as the total number of CUs in it. In this exposition, we consider HCs of size 25, i.e a HC of 5x5 CUs).

$$(X-1) * sizeofHC <= 3 * N + 1 + 2 * C < X * sizeofHC$$

Mapping RBFNN of six input dimensions and three output classes on HC is shown in Figure 3.

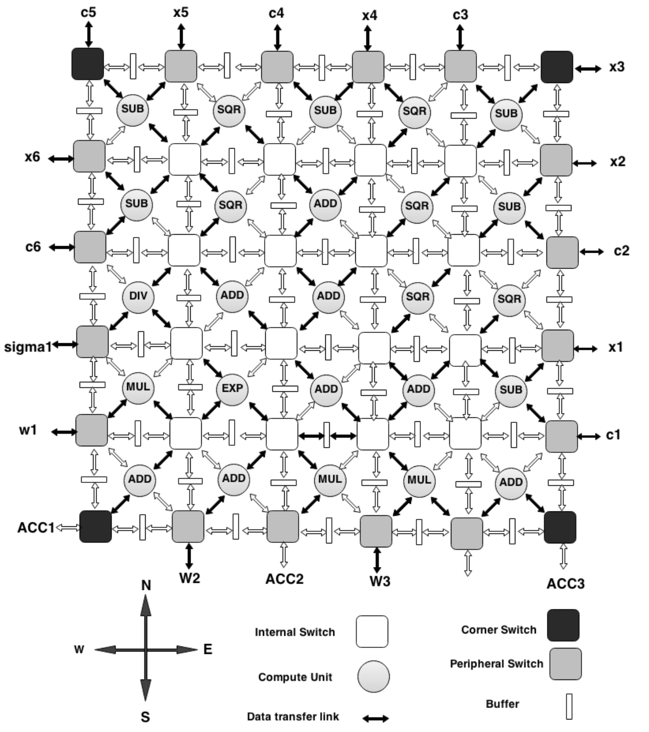

Fig. 3: Mapping RBFNN of six input dimensions and three outputs on HyperCell.
["x_1,..., x_8" are input dimensions; "$c_1, ..., c_8$" are center dimensions.]

A. Partitioning DFG of Basis Generator of N dimensional RBFNN on HyperCells

The procedure for partitioning DFG of Basis Generator part on HCs is carried out in 2 steps:

- Identifying Subgraphs optimal for mapping on HCs.
- Defining Communication links between these HCs.

Base Structure is defined as DFG of maximum possible dimension whose Euclidean Distance Squared can be calculated on a single HC. Let N_s be that maximum possible dimension. N_s satisfies the following equation.

$$3 * N_s - 1 < sizeofHC < 3 * N_s + 2$$

For example on a HC of size (5x5) the maximum possible dimension whose Euclidean Distance Squared can be calculated on a single HC is eight.

The DFG of the Base Structure is shown in Figure 4. In this figure Operations needed for Euclidean Distance calculation between input and center are: Subtraction (SUB), Square (SQR) and Addition (ADD).

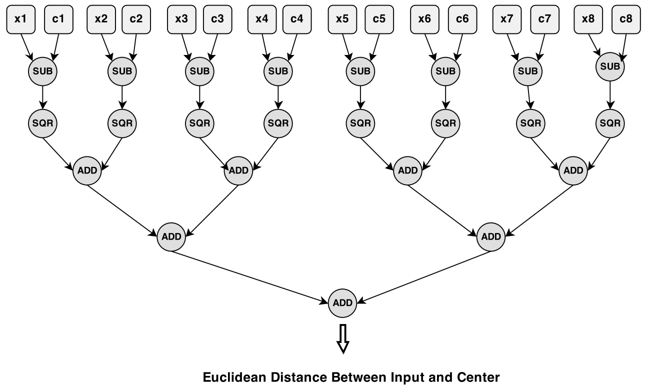

Fig. 4: Base Structure on HyperCell of size 5X5.

We partition the DFG of RBFNN to create maximum number of Base Structures. Then we define communication links between these subgraphs. Depending on communication links, three partitioning methods are introduced:

1) Linear Structure: In this method for partitioning N dimension RBFNN considering one center at a time, $\lfloor N/N_s \rfloor$ Base Structures are required. If N is not an integral multiple of N_s, then the remaining dimensions is mapped on an additional HC. The number of HCs required, $a = \lceil N/N_s \rceil$. The communication between these HCs is done in a linear manner. (i.e. The output of each HC is fed as an input to the next HC). Linear Structure is modular, as increasing the dimension of RBFNN will affect only the number of the Base Structures, not the mapping of the Base Structures on HCs. The number of clock cycles required in this architecture is linearly proportional to the number of HCs used for mapping, therefore, it is not suitable for mapping RBFNN of large dimensions. Figure 5 shows the Linear Structure for partitioning RBFNN on a HC.

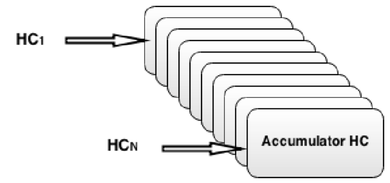

Fig. 5: Linear Structure.

["HC_1 to HC_N" are used for implementing Base Structures and Accumulator HC contains the operations needed for output calculations. The number of Accumulator HC may be more than one based on the number of the output classes in RBFNN.]

2) Tree Structure: In this method the DFG of RBFNN, considering one center at a time is partitioned in different stages :

First Stage : This stage is very similar to Linear Structure (i.e. $\lceil N/N_s \rceil$ Base Structures are required), but instead of passing the output/outputs of each HC to the next one, they pass to the HC/HCs of the second stage.

Second Stage : The HCs of this stage collect outputs of the HCs of first stage and add them to generate the final Euclidean Distance between the input and the center. The number of HCs in this stage depends on the number of outputs passed by the HCs in the first stage and the number of peripheral compute units on HCs. Let P be the number of peripheral compute units on a HC, the number of HCs required in this structure will be : $a + \lceil a/2P \rceil + \lceil a/(2P)^2 \rceil + ... + \lceil a/(2P)^{m-1} \rceil + 1$, where m is the number of substages (hierarchy of HCs) in second stage.

The clock cycles required for processing one center is proportional to m in this method. For very large dimensions of RBFNN, there will be considerable improvement in performance, compared to Linear Structure. The Tree Structure of HCs realizing the RBFNN is shown in Figure 6.

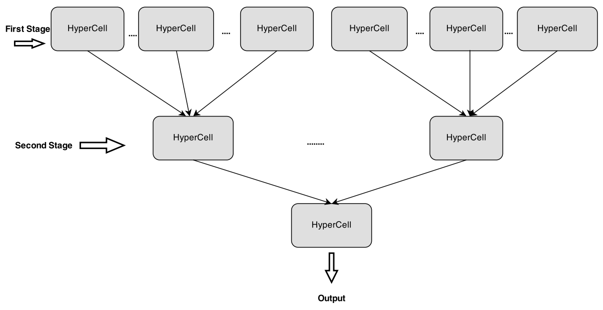

Fig. 6: Tree Structure.

3) Hybrid Structure: The second stage of Tree Structure will be complex if the total number of the outputs passed from the first stage becomes too large. To minimize this complexity we try to decrease the number of outputs passed from first stage to the second stage by combining the HCs in the first stage in a linear way by grouping them in different "Linear Blocks" and pass the outputs generated by each linear block to the HC/HCs of the second stage. Besides being as modular as Linear Structure, performance of Hybrid Structure is comparable to that of Tree Structure. In Hybrid Structure (L,M), L is defined as the number of HCs in each linear block in the first stage and M is the number of the substages in the second stage. For (L , 0) (i.e. zero substage in the second stage), only one HC will be present in the second stage, therefore to map a N dimensional RBFNN on HCs with P number of peripheral switches, where each HC can have N_s dimension mapped on it, the number of HCs in each linear block will be $L = N/(2 * P * N_s)$. Total Number of HCs required in this method for $M = 0$ will be $\lceil N/N_s \rceil + 1$ and the clock cycles required will be proportional to L. Thus there will be considerable reduction in the clock cycles required by this method compared to Linear method. By increasing the number of sub-stage in second stage (i.e. M), the clock cycles can be reduced further, as it is proportional to $L = N/(2 * P)^{M+1} * N_s$. Hybrid Structure, is shown in Figure 7.

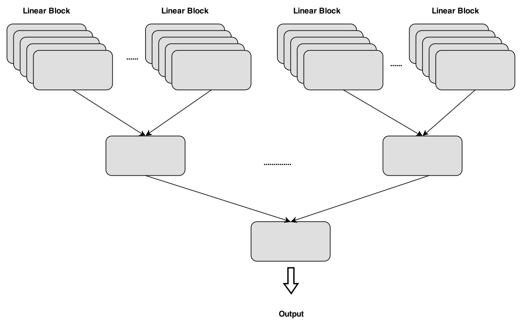

Fig. 7: Hybrid Structure.

IV. Emulation Of HyperCell Partitioning on a Multi-Processor System

In order to verify the functional correctness of the Pipelined Implementation scheme, a Multi-Processor System-on-Chip (MPSoC) configuration is used. The Hybrid Structure partitions are run as applications on interconnected soft core processors implemented on the FPGA. In addition, in order to keep all the processors busy (to maximize the performance), we implement the Parallel Implementation scheme (mentioned in section II.B).

A. MicroBlaze Base Multi Processor System

The MicroBlaze (MB) soft processor core from Xilinx is a 32-bit Reduced Instruction Set Computer (RISC) based on Harvard architecture[10] with a rich instruction set optimized for embedded applications. With the MB soft processor solution, we get complete flexibility to select the combination of peripheral, memory and interface features. This flexibility of the soft-core makes it an ideal processor to create a MPSoC on FPGA. MB v8.00b has been used in our experiment. The architecture of MB is shown in Figure8.

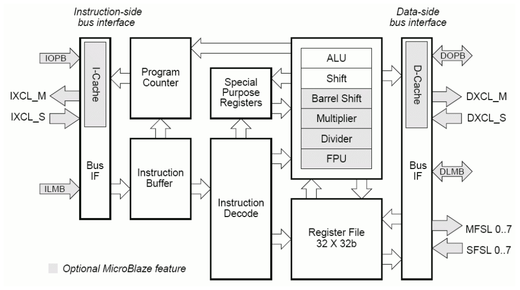

Fig. 8: MicroBlaze Block Diagram.

The MPSoC architecture [9] consists of multiple MBs each with its own private BRAM connected to its Local Memory Bus (LMB). Each MB is connected to the common Processor Local Bus (PLB) and has a full-duplex connection to every other MB via a pair of Fast Simplex Links (FSL). The MB Debug Module (MDM) and Timer are also present in the design. The MDM is connected to each processor to enable debugging. The MicroBlazes hardware FPU performs all computations (ie. addition, subtraction, multiplication and division) in the application program, but does not support hardware exponentiation. Hence a basic exponentiation hardware unit (consuming 39 cycles not including communication cost to and from the processor) is implemented on the FPGA and paired with each processor via FSL. The complete design is implemented on the Xilinx ML510 Evaluation Platform.

In this MPSoC emulation setup, partitioned applications (as per Pipelined and Parallel Implementations) are executed on the bare metal MBs (without a real-time operating system). The data is provided to MPSoC by the host machine via universal asynchronous receiver/transmitter (UART) . Each processor receives input data from the UART and sends computed data to the other processors (as per the partitioning scheme) via FSL. A shared memory (BRAM/DDR) could be used, but that would add to the latency in data access. So copies of the input data required by each MB is stored in its local BRAM. Once this initialization is done, the timer is started and a ready signal is passed to each processor via FSL to begin execution. The communication of the intermediate data is done via FSL. The proper synchronization and coherency among the processors in the execution is maintained by the blocking nature of the functions getfsl() and putfsl().

Figure 9 shows the MPSoC emulation setup.

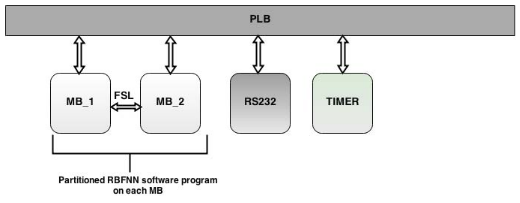

Fig. 9: MPSoC emulation setup.

B. Functional Verification

We emulated Pipelined Implementation scheme using one, two and five MB processors and checked for functional correctness, exactly mimicking the same partitioning done on HC, so that we can realistically measure their relative performance. Due to MicroBlazes slow mode of communication, a large number of cycles are expended, resulting in lower performance.

C. Partitioning schemes for obtaining better performance

Along with requiring a greater number of data transfers among processors (an overhead that gets magnified in the case of the MPSoC setup), the Pipeline Implementation also allots asymmetric execution load over the processors, wasting compute resources. By partitioning the RBFNN across the centers (i.e. using the Parallel Implementation), communication is minimized and all processors are kept busy until the end of execution.

V. Results

Three different data sets, ”IRIS”, ”WINE” and ”SPECTF HEART”, from UCI Machine Learning Repository [11] were used for implementing RBFNN using Hybrid Structure, on HC as a reconfigurable architecture and on MB as MPSoC architecture. We used Bluespec code of HC [8], [13] as a platform with RBFNN as an application on it and simulated the code to calculate the clock cycles. In case of MB, we calculated the number of clock cycles by running software programs of RBFNN on MB.

Table I shows the number of operations needed for RBFNN computation of the data sets. In case of MB implementation, both Pipelined Implementation and Parallel Implementation methods are used. The number of the clock cycles on both HC and MB, is shown in Table II. Speed up of HyperCell based implementation over MPSoC implementation is shown in Table III. The network of HCs were clocked at 500 MHz and MPSoC was clocked at 125 MHz. Figure10.A shows the comparison of clock cycles on MicroBlaze and HyperCell based on Pipelined Implementation for IRIS, WINE and SPECTF HEART data sets and figure 10.B shows the comparison of clock cycles on MicroBlaze based on PipeLined and Parallel Implementation for WINE and SPECTF HEART data sets (IRIS data set can not be implemented using Parallel Implementation method as there is need of only one processor for implementation). As the figure shows the Parallel Implementation scheme is the optimized method and gives better performance than Pipelined Implementation method for MB.

TABLE I: Operations needed for RBFNN calculations on IRIS (denoted by A), WINE (denoted by B) and SPECTF HEART (denoted by C).

Data Set	No. of Input Nodes	No. of Hidden Nodes	No. of output Nodes	Operations Needed for RBFNN Computation					
				SUB	SQR	ADD	DIV	EXP	MUL
A	4	8	3	4	4	6	1	1	3
B	13	26	3	13	13	15	1	1	3
C	22	44	2	22	22	23	1	1	2

TABLE II: Comparison of total clock cycles on HperCell and MicroBlaze. IRIS (denoted by A), WINE (denoted by B) and SPECTF HEART (denoted by C).

Data Set	No. of HyperCells used for mapping	No. of processors on MicroBlaze	Total clock cycles on HyperCell	Total Clock Cycles on MicroBlaze	
				Pipelined Implem entation	Parallel Implem entation
A	1	1	230	2088	—
B	2	2	439	16732	13183
C	5	5	584	18636	14350

TABLE III: Speed up of HC based implementation over MPSoC.

Data Set	Speed up of HC based implementation over MP-SoC
IRIS	36.313
WINE	152.455
SPECTF HEART	127.643

VI. Conclusion

Radial Basis Function Neural Network with Gaussian function as the basis kernel and offline training was considered for classification purpose. We implemented RBFNN on HyperCell (using Pipelined Implementation method) and on MicroBlaze (using Pipelined and Parallel Implementation methods) and compared the obtained results in case of Pipelined Implementation scheme. For implementing RBFNN of large dimensions on HyperCells, three different partitioning schemes were proposed. Comparison of obtained results, demonstrates (as expected), HyperCells take much less number of clock cycles for the execution of RBFNN than the corresponding Multi-Processor-System-On-Chip (MPSoC). Results show that implementation of RBFNN using Hybrid Structure on HyperCell has on average 26x clock cycle reduction and 105X improvement in the performance over that of multi-processor system on FPGAs.

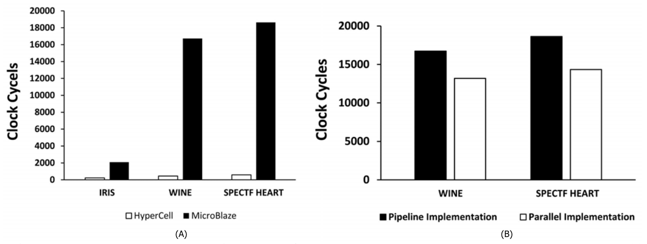

Fig. 10: A. Comparison of the clock cycles on HyperCell and MicroBlaze, based on Pipelined Implementation method. B. Comparison of the clock cycles on MicroBlaze, based on Pipelined and Parallel Implementation methods.

References

[1] Haykin, Simon S., et al. "Neural networks and learning machines." Vol. 3. Upper Saddle River: Pearson Education, 2009.

[2] M. D. BUHMANN , "Radial Basis Functions: Theory and Implementations.";First ed.; published by the press syndicate of the university of cambridge, Cambridge, United Kingdom, 2003.

[3] Schwenker, Friedhelm, Hans A. Kestler, and Gnther Palm. "Three learning phases for radial-basis-function networks." Neural networks 14.4 (2001): 439-458.

[4] Dias, Fernando Morgado, Ana Antunes, and Alexandre Manuel Mota. "Artificial neural networks: a review of commercial hardware." Engineering Applications of Artificial Intelligence 17.8 (2004): 945-952.

[5] Misra, Janardan, and Indranil Saha. "Artificial neural networks in hardware: A survey of two decades of progress." Neurocomputing 74.1 (2010): 239-255.

[6] Youssef, Ayman, Karim Mohammed, and Amin Nasar. "A Reconfigurable, Generic and Programmable Feed Forward Neural Network Implementation in FPGA." Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on. IEEE, 2012.

[7] Liu, Jihong, and Deqin Liang. "A survey of FPGA-based hardware implementation of ANNs." Neural Networks and Brain, 2005. ICNN&B'05. International Conference on. Vol. 2. IEEE, 2005.

[8] Das Saptarsi, Kavitha Madhu, Madhav Krishna, Farhad Merchant, Ipsita Biswas, Adithya Pulli, S. K. Nandy, and Ranjani Narayan. "A Framework for Post-Silicon Realization of Arbitrary Instruction Extensions on Reconfigurable Data-paths." Journal of Systems Architecture (2014).

[9] Huerta, P., et al. "Multi microblaze system for parallel computing." Proceedings of the 9th International Conference on Circuits, str. 2005.

[10] MicroBlaze Processor Reference Guide, http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf, 2008.

[11] UCI Machine Learning Repository, http://archive.ics.uci.edu/ml.

[12] Vassiliadis, Stamatis, Stephan Wong, Georgi Gaydadjiev, Koen Bertels, Georgi Kuzmanov, and Elena Moscu Panainte. "The molen polymorphic processor." Computers, IEEE Transactions on 53, no. 11 (2004): 1363-1375.

[13] Kavitha Madhu, Saptarsi Das, Madhava Krishna, Nalesh S, S K Nandy and Ranjani Narayan,"Synthesis of Instruction Extensions on HyperCell, a Reconfigurable Datapath", International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation,SAMOS Island, Greece, July 2014.