A GENERIC RECONFIGURABLE NEURAL NETWORK ARCHITECTURE IMPLEMENTED AS A NETWORK ON CHIP

T. Theocharides, G. Link, N. Vijaykrishnan and M. J. Irwin

The Pennsylvania State University
{theochar, link, vijay, mji}@cse.psu.edu

V. Srikantam

Agilent Technologies
vamsi_srikantam@agilent.com

ABSTRACT

Neural Networks are widely used in pattern recognition, security applications and data manipulation. We propose a novel hardware architecture for a generic neural network, using Network on Chip (NoC) interconnect. The proposed architecture allows for expandability, mapping of more than one logical unit onto a single physical unit, and dynamic reconfiguration based on application-specific demands. Simulation results show that this architecture has significant performance benefits over existing architectures.

I. INTRODUCTION

Artificial neural networks (ANN) are used in a wide range of applications, including pattern and object recognition, statistics and data manipulation, security applications, data mining, machine learning applications, digital signal processing and many other applications. Due to the popularity of ANN applications, there has been significant research involving artificial neural network implementations, both in software and hardware. Software implementations have been researched and optimized heavily, yet still lack in performance. This makes software implementations not applicable to real time computation and emphasizes the requirement of a hardware implementation for real time systems [1 and 3]. This need is further stressed by an increasing demand for machine learning and artificial intelligence that exhibit both speed and portability in their implementation. As a result, hardware neural network implementations have been proposed for quite some time with a large amount of activity in the early 1990's. However, hardware architectures faced major implementation issues in terms of resource demand and interconnection. In the most common neural network topologies, every node in layer *n* must forward its result to every node in layer n+1, resulting in extremely complicated inter-layer interconnect for any reasonably large number of neurons per layer.

Recently, researchers have proposed novel architectures to solve on chip interconnection issues using Network on Chip (NoC) design. NoCs attempt to solve the interconnect problem by routing on-chip signals using a packet-based network. NoCs allow for mapping of one or more logical units into a single physical hardware unit, a method known as virtualization. This paper proposes the use of NoC architecture as a backbone for a generic reconfigurable and expandable neural network processor.

This work was supported in part by grants from NSF CAREER 0093085, NSF 0130143, NSF 0202007 and MARCO/DARPA GSRC-PAS

II. BACKGROUND AND RELATED WORK

The network operates in two steps – the first involves training to perform an operation, the second involves adapting to perform that operation based on the training. Neurons are used in multilayer configurations, where they are trained to perform a desired operation in a similar manner to the way the human brain operates [5]. A neuron takes a set of inputs and multiplies each input by a weight value. Weights are determined during the training process. The result of each multiplication is accumulated in the neuron until all inputs have been received. A threshold value (also determined in training) is then subtracted from the accumulated result, in order to determine whether the minimum expected sum was met. This result is then passed through an activation function to determine the final output of the neuron. The activation function used depends on the application. Typical activation functions include the sigmoid function, the hyperbolic tangent function and Gaussian and linear functions and their variations. This output is then propagated to a number of destination neurons in another layer, which perform the same general operation with the newly received set of inputs and their own weight/threshold values. The accuracy of the computation is determined by the bit widths of the inputs and the weights, as well as the accuracy of the activation function. Neurons can be connected in different configurations. The most commonly used configuration is the multilayer feed forward (FF) perceptron, although certain applications benefit from particular configurations such as Radial Basis Function networks [5].

A large number of hardware implementation architectures, including analog, digital, and hybrid have been proposed. Analog implementations were proposed by multiple researchers in the late 80's and early 90's [1] and dominated the early research. Technology scaling has made digital designs with their inherent scalability and accuracy feasible. A number of different digital architectures based on systolic arrays [1] and multiprocessor environments [2] have been proposed. However, they suffer from the following drawbacks: systolic array architectures do not allow reconfigurability [4] and multiprocessor platforms suffer from cost and size factors. Bit slice architecture is a proposed alternative; however its performance is poor as it limits the inherent parallelism that a neural operation allows [2]. Other proposals include the neurochip (i.e. an independent ANN

on a single chip) and the neurocomputer (multiple ANNs on multiple chips/boards). A good neurochip example is the Neuricam Digital NN processor, used for computer vision and pattern recognition algorithms. However, it is limited to a small range of applications due to its static configuration, and targets only a specific network topology [12].

For a new generation of neurochips to evolve, the architecture has to offer significant advantages over existing hardware. A new architecture should alleviate interconnect issues and support multiple target applications via dynamic reconfigurability, network topology independence, and network expandability. The architecture proposed in this paper adopts the NoC architecture allowing for design-time expandability and runtime reconfigurability (network topology adjusted by sending new data), virtualization (limited physical hardware used to implement multiple logical neurons) and dynamic adjustment of weights and activation functions. Our architecture targets applications where the neural processing stage is critical, allowing the slow and tedious, yet not necessarily real-time operation of training to be done off-chip using existing software training algorithms. As a result, the architecture proposed in this paper accepts training data derived from training done off-chip using software.

III. ANN ARCHITECTURE

Our proposed architecture can be reconfigured to the application's demand to implement popular ANN topologies, such as single and multi-layer perceptron, RBF networks, competitive networks, Kohonen's self-organizing maps and Hopfield networks [2]. However, as almost 80% of the existing neural network applications use the multilayer perceptron configuration [2], the on-chip routing algorithm and network topology were selected to optimize the feed forward multilayer perceptron. Allowing for slightly higher network congestion enables all other implementations as well. The proposed architecture consists of four types of units – a neuron unit, an aggregator and ingress and egress nodes. The architecture calls for two packet payloads, training and computation. Training packets carry the configuration values from the off-chip trainer, which are loaded into the units' memories. Computation packets carry fixed-point values between neurons to enable ANN application computations.

A. Ingress and Egress Nodes

The ingress and egress nodes are responsible for communicating with the external data source and sink. These two nodes are also responsible for accepting or rejecting data from the external data source for congestion control purposes. The ingress nodes inject data into the network, while the egress nodes collect results from the network and pass it to the external world. These units are relatively much smaller than the other units, and have I/O pads to communicate with the external data source.

B. Neuron Unit (NU)

Neuron hardware requires a high precision multiplier as well as sufficient memory to hold one weight value for each of its input connections. It also needs a large functional unit to compute the activation function value. This would result in a prohibitively large neuron; hence we decided to cluster partial neurons into groups of 4, each called a "neuron unit". The neuron units include the weight lookup tables as well as individual multiply-accumulate units. These four neuron units are clustered around an "aggregator", which implements the activation function, and is discussed in more detail later. Combined, four neuron units and an aggregator act as four entire neurons. This clustering of four does not limit the neural network topology, in that that for a single configuration, same layer neurons share the same activation function.

Each neuron unit consists of hardware to support computation corresponding to multiple separate layers, allowing it to be part of any layer computation. Each unit provides support for hardware virtualization, allowing the logical mapping of multiple logical neurons into one physical neuron unit. This requires additional input buffers for the neuron, but still only one multiply-accumulate unit is required. Virtualization is achieved by using a virtual address in the header of each packet. The neuron node identifies the virtual address of the neuron/layer that sent the data, and uses that address to generate the proper virtual and physical destination addresses. By virtualizing neurons, we increase the number of neurons available to an application without significant increase in the hardware resources. The unit consists of a decoder which decodes the neuron address and determines which neuron in the computation the data corresponds to. A control logic block receives the decoded header and data and sends a memory request for the appropriate weight. Weights are addressed using the layer and virtual source id as addressing bits, information contained in the header. Control logic is simple, consisting of counters and flag registers to keep track of to which layer and which neuron a given input value belongs. This information is used to construct the outgoing data returned to the aggregator.

Once all inputs for a given layer have been received and the computation for the layer is completed, the arbiter directs the proper accumulated sum to the aggregator.

C. Aggregator

The aggregator performs two operations; as an activation function unit, it computes the activation function result for all four neuron nodes attached to it, and as a routing unit, routes packets in the network. The unit consists of the standard routing hardware for NoC's [6, 7 and 8] with the modification that each routing node now is connected to 4 neuron nodes (processing elements) instead of the traditional 1 processing element per routing node. To allow for optimal reconfigurability, the activation function is implemented via lookup table, programmed using configuration packets. The activation function values are

loaded into the lookup table memory on configuration, and remain unchanged during the computation unless the application makes any dynamic adjustments. As such, it supports popular activation functions such as the hyperbolic tangent function, the piecewise linear function, the sigmoid function and the Gaussian function [5].

When a neuron unit finishes its computation and sends the results back to the aggregating node, the aggregating node performs the activation function via the appropriate lookup table. It then combines the output of the four logical neurons into a single packet, which it forwards to the appropriate aggregator (in the next logical layer) or to an egress node. In a non-clustered network, each *n*-node layer sends n^2 packets to the next N-node layer. By clustering, we reduce this to only $(n/4)^2$ packets, significantly reducing network load.

D. System Overview

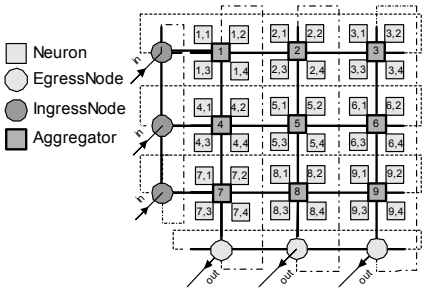

Figure 1: System Architecture

The system performs two types of operations; configuration and computation. Configuration involves initialization control registers and the network structure, allocation of physical and virtual addresses in order to achieve desired network topology, and initialization of the weights and activation function values for the on-chip memories. The network is fully reconfigurable, allowing the application to load desired weights and activation function during configuration, and to change these values whenever desired. During computation operation, the network performs the algorithm desired by the application. The architecture is expandable and can be minimally modified if desired to support more layers and more neurons by paying the extra hardware penalty.

Figure 1 shows the topology of the network with 9 aggregator nodes. The ingress, egress, and aggregator nodes are all connected using packet switches. Our architecture supports deadlock free X-Y routing, and uses wormhole/packet switched communication. Each packet is composed of a header and four data fields, as this allows each packet to carry data to four neurons simultaneously. The general network topology is a 2D torus, allowing packets leaving one 'edge' of the network to enter the opposite 'edge' of the network, significantly reducing congestion hotspots. If desired, the FF network can be mapped in a near-systolic manner, allowing for minimal network congestion. The heaviest congestion occurs near the input nodes, as the input layer of a neural network typically sees the highest amount of data, receiving a much larger number of inputs than any other layer. Internally, the network uses congestion signals to prevent the loss of any packets, as a single packet loss can disrupt the computation heavily.

IV. EXPERIMENTAL PLATFORM AND RESULTS

In order to evaluate our proposed architecture, we implemented the system in Verilog HDL, and used various system level simulations to obtain optimal network parameters. The system used for the remainder of the paper is configured as in the following paragraph.

All values are represented in fixed-point 32 bit notation. The network is as shown in Figure 1, with 9 aggregators in the network, for a total of 36 physical neurons. Each physical neuron can represent at most three logical neurons, resulting in a maximum of 108 neurons. Each neuron can store up to 512 weights (requiring three 2KB memories), and this allows up to 512 inputs per layer. Data is transmitted in 160-bit packets, using a flit size of 80 bits. A flit is defined as the smallest bufferable packet chunk. The connections between aggregators and the ingress and egress nodes are therefore 80 bits wide. The 160-bit packet consists of a 32-bit header, and 4 32-bit data words, one word for each neuron unit. Packets are split in a best-fit manner, with the 32 bit header and 48 bits of data in the first flit, and the remaining 80 bits of data in the second flit. Packets are assembled at the destination aggregator. The header contains control information, physical and virtual addresses and configuration and routing information. The 128-bit payload contains data values when the system performs computation, and source/destination configuration data, weight values and activation function values when a configuration packet is sent. We synthesized our proposed architecture in a commercial 160nm technology with a target 200Mhz clock rate. Overall, the experimental platform required 80 mm^2, with each neuron unit consuming roughly 2mm^2 and the aggregators occupying 1mm^2 each.

We simulated our architecture using a NoC simulator [11] to obtain network latency, congestion results and effective throughput. To evaluate our architecture, we chose 5 applications which we used as benchmark applications. We implemented a face rotation detection network (FRDN) [9], a face detection network (FDN) [9], a letter recognition algorithm (LR) [10], a FFT algorithm [10] and a PCA classifier [10].The objective of our tests was to compare our architecture against existing hardware ANN architectures as each algorithm uses a different network configuration and different set (and number) of inputs. Table 1a shows the network performance using the 5 listed applications. One can conclude that the performance is affected by the ANN topology, particularly the inputs to the first layer neurons.

	Parameters	Face Rotation Detection	Frontal Face Detection	Letter Recognition	PCA Classification	Fast Fourier Transform
Configuration	ANN Topology	15-15-36	20-3-1	12/4/2004	16-20-1	15-15-15 RBF
	# of inputs	400 x 8 bits	100 x 8 bits	144 x 8 bits	512 x 32 bits	512 x 32 bits
Architecture and Network Performance	Average Packet Latency	3.675 hops 18.375ns	2.777 hops 13.885ns	2.144 hops, 10.72ns	3.788 hops 18.94ns	3.934 hops 19.67ns
	Time per pattern	34.7µs	8.016µs	11.29µs	186.18µs	204.8µs
	Effective Throughput	92Mbps	98Mbps	102 Mbps	88 Mbps	80 Mbps

Table 1a: System Performance

	Face Rotation Detection	Frontal Face Detection	Letter Recognition	PCA Classification	Fast Fourier Transform
IBM ZISC 036	2812	11250	7813	549	549
Nestor/ Intel	450	1800	1250	88	88
Philips Lneuro1	30	117	813	6	6
Siemens MA-16	450	1800	1250	88	88
RC Neuromatrix	1350	5400	3750	3750	3750
NoC Neural Network	**28750**	**115000**	**88542**	**5371**	**4883**

Table 1b: CPS Comparison

Name	Architecture / Topology	Precision	Neurons	Synapses	Learning
IBM ZISC 036	RBF	8b x 8b	36	64	off-chip
Nestor/ Intel Ni 1000	RBF	5b x5b	1024 (16 per board)	256	off-chip
Philips Lneuro1	Multilayer FF	1-16b x 16b	16	16x16	programmable
Siemens MA16	Systolic Array	16b x 16b	16	64	off-chip
RC Neuromatrix	Multilayer FF	64b x 64b	1 to 64	1 to 64	programmable
NoC Neural Network	*Network on Chip*	*32b x 32b*	*36 physical, 108 virtual*	*400, 100, 512, 144, 512*	*off-chip*

Table 1c: Details of compared Architectures

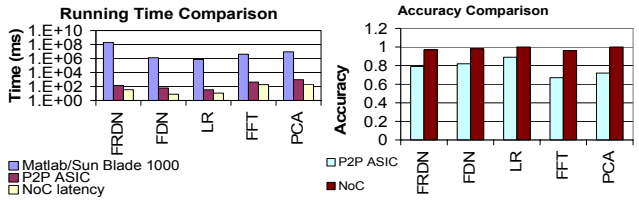

Figure 2: Timing and Accuracy Comparison

As we have found no standard metric which defines the performance of a hardware neural network implementation, we consider a number of different comparison techniques. Parameters such as the number of connections per second (CPS) – defined as the rate of MAC operations per second, the number of neurons, precision of the multiply-accumulate operation and application performance are more widely used than clock frequency. In an attempt to quantify the performance of our proposed architecture therefore, we compared it against some existing hardware architectures [1]. The major problem in doing so has been the stall of hardware progress in the mid-90's due to the previously noted interconnect and chip area bottlenecks.

Table 1b shows the CPS comparison for each application tested between existing hardware architectures and our proposed architecture. As seen in table 1b, the CPS for our proposed system is much higher than the CPS achieved by previous implementations. Even though the compared implementations use older technology, they are still limited by the interconnect architecture, in that each neuron has to receive all inputs in order to complete its operation, therefore the throughput is limited. Details of the compared architectures are shown in Table 1c.

To obtain a solid performance comparison, we used the five algorithms mentioned, and compared their performance using our architecture and using existing hardware architectures. We also run the same benchmark algorithms on a point-to-point based ASIC architecture designed in the same technology and of roughly the same area [9]. Figure 2 shows the timing and accuracy results of our implementation for each application compared to the ASIC implementation. The figure also shows the corresponding accuracy performance for a software implementation for each of the algorithms. As seen in the two graphs, our architecture is more accurate than a custom ASIC chip and is almost as accurate as a double precision software implementation. Our architecture allows a far more efficient use of chip area for performance. Virtualization, use of partial neurons, and more efficient interconnect allow for more neurons per chip. In addition, one chip, properly programmed, can perform all the applications listed above without any design time changes. The high precision we obtain is also significantly better than that of other hardware implementations because we combine both a large number of neurons and high precision MAC units.

V. CONCLUSION

The paper presented a novel, reconfigurable architecture for a generic neural network processor, using NoC. NoC architecture is highly beneficial for ANNs as it allows for reconfigurability, mapping of more than one logical unit onto a physical unit, scalability of the net, and a higher number of CPS, opening new horizons for researchers in the ANN hardware domain. The architecture allows a large number of connections between neurons, and provides excellent performance when compared to alternative implementations. We believe that NoC implementations of ANNs can alleviate the wiring congestion problem of past designs, and offer an economically sound production platform due to ease of reconfiguration and high performance.

REFERENCES

[1] C. Lindsey and T. Lindblad, "Review of Neural Network Hardware: A user's perspective", IEEE *Third Workshop on Neural Networks: From Biology to High Energy Physics*, Marciana Marina, Isola d'Elba, Italy, Sept. 26-30, 1994

[2] M. Glesner and W. Pöchmüller, *Neurocomputers: An Overview of Neural Networks in VLSI*, Chapman &Hall, London, 1994

[3] Heemskerk, J. N. H., Overview of Neural Hardware. In: Neurocomputers for Brain-Style Processing. Design, Implementation and Application, *PhD Thesis*, Unit of Experimental and Theoretical Psychology Leiden University, The Netherlands, 1995.

[4] T. Axelrod, E. Vittoz, I. Saxena et al, "Neural Network Hardware Implementations", *Handbook of Neural Computation*, IOP Publishing Ltd and Oxford University Press, London, 1997.

[5] K. Jain, J. Mao, K M. Mohiuddin, "Artificial Neural Networks: A Tutorial", *IEEE Computer*, Volume 29, pp. 31—44, March 1996.

[6] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm", *IEEE Computer,* Volume 35, pp. 70--78, January 2002

[7] W. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. *In Proceedings of 38th Design Automation Conference,* Jun, 2001.

[8] S. Kumar, A. Jantsch, M. Millberg, J. Berg and J.P. Soininen and M. Forsell and K. Tiensyrj and A. Hemani. A Network on Chip Architecture and Design Methodology. *In Proceedings of IEEE Computer Society Annual Symposium on VLSI*, Apr, 2002.

[9] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Irwin and W. Wolf, "Embedded Hardware Face Detection", *In the Proceedings of the VLSI Design Conference, Mumbai, India*, January 2004.

[10] B. Krőse and P. Van Der Smagt, "An Introduction to Neural Networks, 8th Edition", *Internal Report*, The University of Amsterdam, The Netherlands, November 1996.

[11] D. Whelihan and H. Schmit, "NOCSim Simulator", http://www.ece.cmu.edu/~djw2/NOCsim/, September 2003.

[12] Neuricam, "NC3003", http://www.neuricam.com/