World Academics Journal of Engineering Sciences **Survey Paper**
Vol.7, Issue.4, pp.47-52, December (2020) **E-ISSN:** 2348-635X

Digital VLSI for Neural Networks

Priyanjali Jain[1]*, Priyanshu Jain[2]

[1]Dept. of Electronics and Communication, Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar (Ahmedabad) Gujrat, India
[2]Dept. of AVIONICS, Indian Institute of Space Science and Technology, IIST,
[2]Dept. of Space, Govt. of India, Valiamala P.O., Thiruvananthapuram – 695547, Kerala, India

**Correspondence Author: 2710priyanjali@gmail.com,Mob no +91 9425436379*

Available online at: www.isroset.org

Received: 10/Oct/2020, Accepted: 05/Nov/2020, Online: 31/Dec/2020

Abstract- This chapter discusses digital electronic implementations of ANNs. First, we look at the differences between digital and analog design techniques with a focus on cost- performance trade-offs. Second, we consider the use of traditional processors in parallel configurations for ANN emulation. Third, to convey a sense of some of the issues involved in designing digital structures for ANN emulation, a custom digital ANN processor is discussed: the Adaptive Solutions CNAPS. Although this chip is no longer produced, it is still being used. It's simple architecture makes it a good vehicle to understand the trade-offs inherent in emulating neural structures digitally. And fourth, we look briefly at FPGA technology as a promising alternative for digital implementation of ANNs.

Keywords:- VSLI, Digital, Network ANN , CNAPS, FPGA

I. INTRODUCTION

The computational overhead required to simulate Artificial Neural Network (ANN) models, whether simplistic or realistically complex, is a key problem in the field because of the computational complexity of these models.

Network simulations are required both for research and for commercial products. Most researchers currently perform these simulations on standard computer technology, such as high-end workstations or personal computers. However, as the field progresses, researchers are moving to larger and ever more complex models that challenge even the fastest computers.

A reasonably realistic neural model could approach one million neurons and tens of billions of connections, where a "connection" is a data transfer path between two neurons. In addition to size, the models themselves are becoming more complex as we move from simple inner-products to spiking neurons with temporal time course that require a convolution to be performed at each synapse.

For these reasons, there has been much interest in developing custom hardware for ANNs. The inherent parallelism in ANN and connectionist models suggests an opportunity to speed up the simulations. Their simple, low precision computations also suggest an opportunity to employ simpler and cheaper, low-precision digital hardware implemented by full-custom silicon or by FPGAs (Field Programmable Gate Arrays).

II. WHY DIGITAL?

Cost-Performance

One commonly-held belief in the ANN research community is that analog computation, where signals are transmitted and manipulated as strengths, generally voltage or current, is inherently superior to digital computation, where signals are transmitted and manipulated as serial or parallel streams of 1s and 0s. But in fact, both technologies have advantages and disadvantages. The best choice depends on a variety of factors.

Why is analog appealing? An important reason is that it provides 10-100 times greater computational density than digital computation.

Computational density - the amount of computation per unit area of silicon - is important because the cost of a chip is generally proportional to its total area. In analog circuitry, complex, non-linear operations such as multiply, divide, and hyperbolic tangent can be performed by a handful of transistors. Digital Circuitry requires hundreds or even thousands of wires and transistors to perform the same operations. Analog computation also performs these operations using far less power per computation than digital computation.

If analog is so good, why are people still building

digital chips, and why are most commercial products digital? One important reason is familiarity. People know how to build digital circuits, and they can do it reliably, no matter the size and complexity of the system. This is partly the legacy of having thousands of digital designers all over the world constantly tweaking and improving design techniques and software. It is also easier to create a digital version of a computation, where a computer program represents the algorithm, than an analog version, where the circuit itself represents the algorithm. This is particularly true if you are trying to build a system that is robust and reliable over the wide temperature and voltage ranges needed in commercial products. Analog design is an uncommon capability. And it is becoming less common as people find they can do more with digital circuitry. For example, Digital Signal Processors (DSPs) now perform most of what was once the domain of analog circuitry. Another advantage to the digital emulation of neural networks is that it significantly eases the integration of the neural network portion of the design with the larger digital system that it connects to.

Flexibility

Another factor working in favor of digital is that analog designs are generally algorithms wired into silicon. Such designs are inflexible. Though there is an interesting class of designs that are programmable analog. Perhaps the most powerful and widely studied is the CNN – Cellular Neural Network (Chua and Roska, 2001).

Digital designs can be either hardwired or programmable. Their flexibility is a major benefit, since it allows software control as well as an arbitrary level of precision (low to high, and fixed or floating point). The price of this flexibility is reduced performance /cost, but the result is a chip that can solve a larger part of a problem. It also leads to a device that has broader applicability and that can track incremental algorithm improvements by changing the software, not by redesigning the circuitry.

The role flexibility plays in system performance /cost can be understood more clearly by examining *Amdahl's law* (Hennessy and Patterson, 1991) that describes the execution time benefits of parallelizing a computation. Briefly, a computing task has portions or *subtasks* that often can be executed in parallel.

Other, sequential tasks, cannot begin until a previous task has completed, which forces a sequential ordering of these tasks.

Amdahl's law states that no matter how many processors are available to execute subtasks, the speed of a particular task is roughly proportional to those subtasks that cannot be executed in parallel. In other words, sequential computation dominates as parallelism increases. Amdahl quantifies the relationship:

$S = 1/(op_S + (op_P / p))$

Where S is the total speed-up, op_S the number of operations in the serial portion of the computation, op_P the number of operations in the parallel portion, and p the number of processors. As p gets large, S approaches $1/op_S$.

For example, suppose we have two chips to choose from. The first can perform 80% of the computation with a 20x speed up on that 80%. The second can perform only 20% of the computation, but executes that 20% with a 1000x total system speed up. Plugging into the equation, the first chip gives us a total speed up of over 4x, while the second - and "faster" - chip has only a 1.25x total system speed up. A programmable device that accelerates several phases of an application generally offers a much larger benefit than a dedicated device.

Below we discuss FPGAs (Field Programmable Gate Arrays), which are flexible to the point of allowing the arbitrary configuration of physical digital circuitry. They are a promising approach to efficiently implementing the inherent parallelism of neural-like structures.

Signal Intercommunication

One difference between silicon and biological networks is that for silicon internode communication is relatively more expensive than for biological systems. Although several levels of wire interconnect (8-10 today) available in most silicon processes, each level is restricted to two-dimensional interconnection because wires on the same level cannot pass over or touch one another.

Two-dimensional layout and large expensive wires require us to modify our biologically-derived computational models to more closely match the strengths and weaknesses of the implementation technology. To show the need for such modifications, Bailey and Hammerstrom (Bailey and Hammerstrom, 1988) modeled a hypothetical neural circuit.

The first calculation assumed a direct implementation - that is, one connection per wire. This billion-connection ANN required tens of square meters of silicon for dedicated communication pathways. Since silicon averages tens of dollars per square centimeter, such a system is too costly to be practical. These costs result from a theorem showing that the metal area required by direct communication is proportional to the *cube* of the fan-in or "convergence" at each node.

Their second calculation assumed a multiplexed interconnect structure – where several connections shared a metal wire. Wire multiplexing adds complexity at each end. Likewise an address must be

sent with each data packet to identify the sender, and some decoding must be performed on that address. Bailey and Hammerstrom showed that with the proper communication architecture, a 100x reduction in silicon area over the direct approach was possible with little impact on performance. Since for these large networks only a few nodes will be active in any given time interval, multiplexing interconnect makes even more sense.
Even analog designers of neuromorphic circuitry have recognized the need for multiplexed interconnect.

However, Analog voltages and currents are difficult to multiplex. One alternative is to represent analog values by using pulses. There are several ways that pulses can be used to represent information, including pulse rate, phase, and inter-pulse-interval. It is possible for different pulse streams to share a single wire by sending, at the time the pulse occurs, the address of the pulse stream. This approach is called Address Event Representation or AER (Boahen, 2000). Pulse or "spike" signal representation is also much more neurobiologically plausible.

III. DIGITAL NEURAL NETWORKS: OFF-THE-SHELF PROCESSORS

One successful approach to high-speed ANN simulation has been to use arrays of commercial microprocessors. This approach works because desktop machines, thanks to Moore's law have achieved a tremendous level of performance/cost. Moore's law states that the number of transistors that can be manufactured economically on a single silicon die doubles every 24 months. Moore's law has held constant for roughly 32 doublings, which is truly impressive. There are not many industries that can claim exponential growth over such a long period.

In addition to raw clock speed, another effect of Moore's law is that more transistors are available to dedicate to specialized functionality. Today the latest microprocessors offer on-board SIMD (Single Instruction Multiple Data) parallel co-processors. For Intel these coprocessors have evolved from MMx to SSE (Pentium III), and now to SSE2 (Pentium IV) (Intel, 2001). The Motorola / IBM PowerPC has the similar AltiVec system. Although these coprocessors have been designed primarily for basic image processing, video codecs, and graphics, they can also be used to emulate certain ANN models.

A problem these machines have though is limited memory bandwidth. Most applications have a fair amount of referencing locality, where a collection of physically contiguous addresses are referenced multiple times. Reference locality allows the processor to use several layers of cache memory (the Pentium IV has 3).

However, neural network algorithms typically require that an entire network be accessed for each state update. Since this network can be very large, it generally does not fit in the caches. Consequently, there is a significant slowdown as the processor ends up waiting for data from memory.

Perhaps the best approach is to use a commercial multi-processor machine that hides the memory bandwidth problems by providing large numbers of processors. For example, the NASA Ames Research Center, has several large Silicon Graphics parallel machines (Shan, Singh et al., 2000). The largest currently has a 1024 processor machine. These systems use very high speed interconnect and are able to emulate large, complex neural network structures. Our research group at OGI has simulated simple association networks approaching one million nodes on this machine.

However, such computational power is typically not available to the average researcher. One popular alternative has been to build large computer clusters using relatively inexpensive PCs. Often known as Beowulf clusters (Reschke, Sterling et al., 1996), these systems connect large numbers of simple processors and are typically built from off-the-shelf hardware (PCs and LAN switches). The software is usually free. Programming is done using traditional languages and MPI (the Message Passing Interface) or PVM (Parallel Virtual Machine). Unfortunately, the inter-processor communication tends to be fairly slow relative to the computation, which compromises the total speed-up to some degree. However, they can be fairly efficient if complex models of the neuron are used that require more computation than intercommunication.

As neural network models become larger and more complex, the model connectivity issues become a major factor in the speed of emulation regardless of the hardware platform. Real neural structures demonstrate *sparseness*, small subset of all possible connections are actually made, and *locality*, there is a higher probability of connections to neurons that are physically near each other. However, ANN models have typically not exhibited significant sparseness or locality, which is another reason for researchers to study more biologically plausible systems, so that we can create structures that are computationally robust and have sparse, localized connections.

IV. DIGITAL NEURAL NETWORKS: FULL CUSTOM PROCESSORS: CNAPS

Designing specialized architectures customized for ANN simulation permits significant improvements in performance/cost, since the processors andtheir interconnection architecture are optimized for computations they perform. This section discusses the

Adaptive Solutions CNAPS architecture, which was, for a time, a successful commercial product, but is no longer produced. It represents the specialized functionality end of the design spectrum of digital chips.

The CNAPS architecture (Hammerstrom, 1995) had multiple Processor Nodes (PNs) connected in a one-dimensional structure, forming a Single Instruction Multiple Data (SIMD) array (Figure 1). SIMD architectures have one instruction storage and decode unit and many execution units, simplifying system design and reducing costs. Unlike a PC cluster, each CNAPS PN did not have program storage and sequencing hardware, and each executed the same instruction each clock. Node outputs were broadcast from each PN to all the others over a single broadcast bus.

Another major simplification of the CNAPS architecture, which is found in other digital ANN chips, was the use of limited-precision, fixed-point arithmetic. Many researchers have shown that floating point and high precision are unnecessary in ANN simulation (Fahlman and Hoehfeld,1992). CNAPS supported 1-, 8-, and 16-bit precision in hardware. Consequently, the PNs were smaller and cheaper. This reduced precision was more than adequate for the applications implemented on CNAPS.

The CNAPS architecture had 64 PNs per chip. At the then frequency of 25 MHz, each chip executed at a rate of 1.6 billion connections computed per second. A single chip executed back-propagation learning at a rate of 300 million connection updates per second-each update consists of reading the weight associated with the connection, modifying it, and then writing it back. Each PN (Figure 2) had 4096 bytes of on-chip local memory, used to store synaptic weight data and other local values. Hence a 64 PN chip could store up to 256 KB of information.

Multiple chips could be combined to create larger, more powerful systems. The general programmability of the device allowed it to execute a large range of functions, including many non-ANN algorithms such as the discrete Fourier transform, nearest neighbor classification, image processing, and dynamic time warping.

Figure 3 shows a simple two layer network mapped to a CNAPS array. The network nodes are labeled CN0-CN7; the processor PN0-PN3. Multiple network nodes map to a single processor node - in this example, one node from each layer is mapped to a single PN. For feed forward calculation, assume that the outputs of nodes CN0-CN3 have been computed. To compute the inner product of nodes CN4-CN7, the output value of node CN0 is broadcast on the bus to all PNs in the first clock. Each PN then multiplies the CN0 output with the corresponding weight element, which is different for each PN. On the next clock, CN1's output is broadcast, etc. After four clocks, all sixteen products have been computed - $O(n^2)$ connections in $O(n)$ time.

V. DIGITAL NEURAL NETWORKS: FPGA (Field Programmable Gate Arrays)

Perhaps the most promising approach to emulating neural models digitally is the FPGA, Field Programmable Gate Array (Sharma, 1998). Briefly, an FPGA is a device with a large number of generic logic blocks and generic interconnect between those blocks. The functions the logic blocks implement and how these blocks are connected to one another is determined by configuration bits that are loaded into the chip as one would load a program into a computer's memory. Because of Moore's law, it is now possible to buy FPGAs that are capable emulating millions of logic gates at frequencies approaching several hundred megahertz. There are very sophisticated design tools that allow logic to be expressed in a high-level hardware description language and then be converted to FPGA configuration bits by an automated synthesis process. These devices can implement large neural structures in parallel, see, for example, Hatano (Hatano and al., 1999).

Although FPGAs are appearing with larger on-chip memory, they still cannot approach the density of commercial DRAM. So for emulating very large networks, off- chip memory needs to be used to store the various parameters and state information associated with each neuron. However, unlike traditional processors, FPGAs are capable of supporting the access to several high-speed memory structures at once. Consequently, a board with several FPGAs could emulate networks at much higher speeds than a high speed desktop PC. In addition, the inherent parallelism in each FPGA would allow parallel implementation of the various structures within the neuron, such as sophisticated spike based computation.

VI. DISCUSSION

It is difficult to predict technology trends, but speculation is always possible. Today most ANNs are used for pattern recognition. The final stage of most pattern recognition algorithms involves checking a series of classification results to see if they fit in the larger context of the domain in question. Including this contextual knowledge can be as simple as spell checking; or it can be as complex as accessing high order rules or schemas that reflect complex syntactical and semantic relationships. Since classification is imperfect, contextual processing,

which makes knowledge of such higher order relationships available to the classification process, is essential to guarantee the accuracy of the final result.

Although the results are still speculative, research (Ambros-Ingerson, Granger et al., 1990; Braitenberg and Schüz, 1998) has shown that scaling to large contexts requires networks with relatively sparse interconnect and sparse activation, where only a few nodes are actively firing at a time. Based on research into VLSI connectivity (Bailey, 1993), digital-based systems can handle such networks more efficiently than analog. Therefore at some point in the processing, the data will probably need to be converted from analog to digital representation. Today the conversion is done at or just after the input transducer. Based on the state of analog technologies, systems of the future will probably take advantage of the computational density of analog VLSI to perform the feature extraction and some preliminary classification at the front end, with conversion to digital form for contextual processing and final classification by "higher level brain regions."

FIGURE CAPTIONS

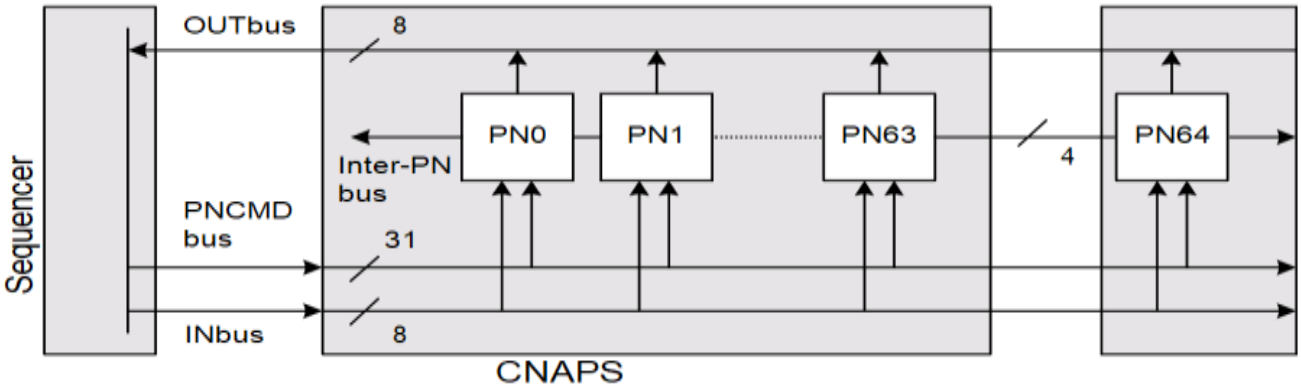

Figure 1. CNAPS Architecture. This is a Single Instruction Multiple Data (SIMD) architecture, where all processor nodes (PNs) execute the same instruction on each clock. There is a single broadcast data bus that allows efficient one to many and many to many communication.

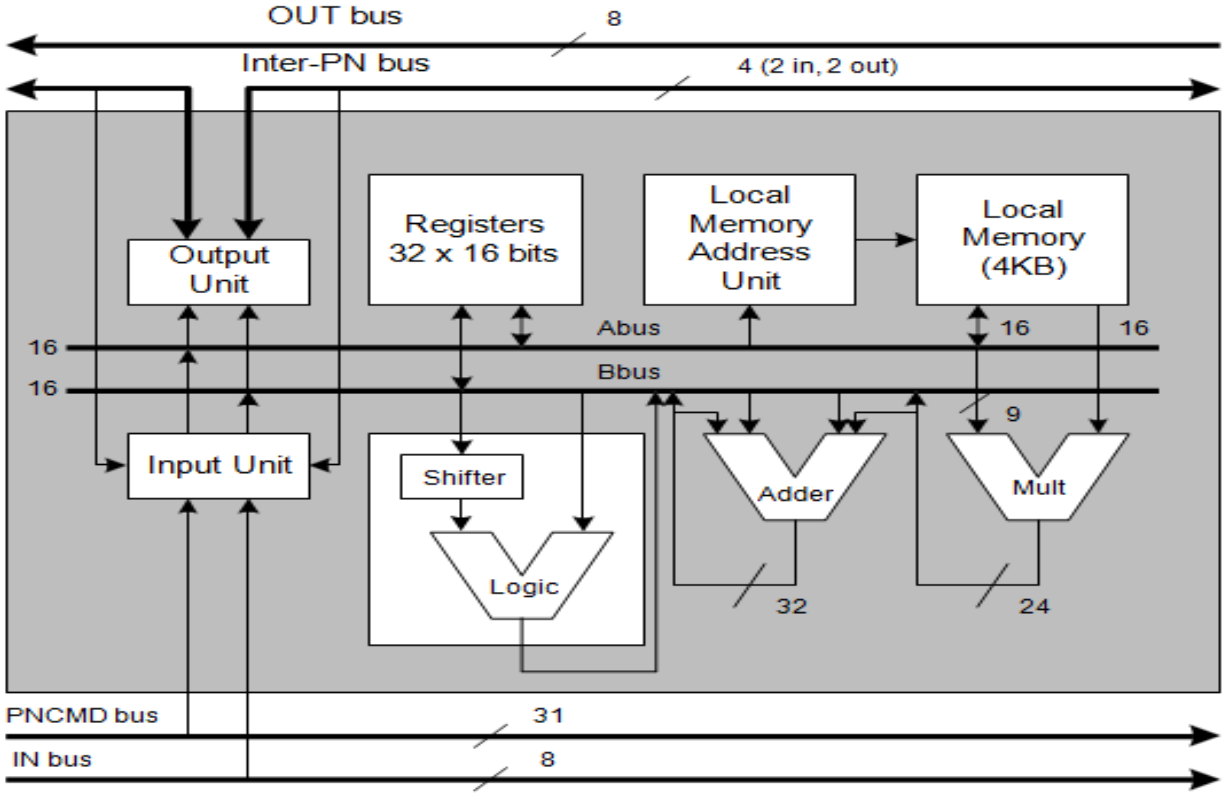

Figure 2. CNAPS PN Architecture. A single PN has a multiplier, accumulator, logic /shifter unit, register file, and separate memory address adder. Each PN also has its own memory for storing weights, lookup tables, and other data. Each PN generates its own unique address to memory.

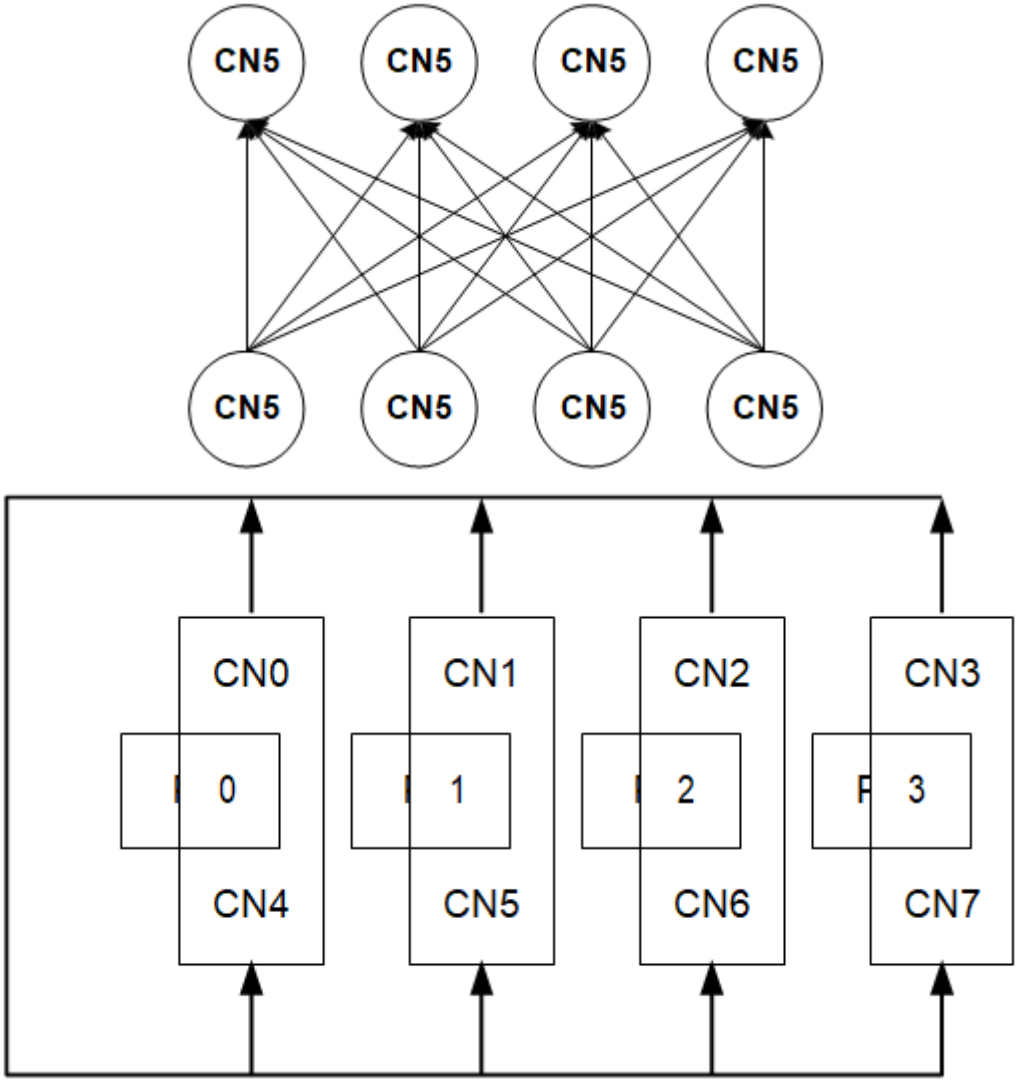

Figure 3. Mapping of a Simple Two-Layer Feedforward Network to the CNAPS Array. When emulating a feedforward network, each layer is spread across the PN array. The neuron outputs of one layer are broadcast sequentially to all PNs while they compute the multiply-accumulations for the next layer of neurons.

REFERENCES

[1] Ambros-Ingerson, J., R. Granger, et al., 1990, Simulation of Paleocortex Performs Hierarchical Clustering, in Science **247**, pp.1344-1348.

[2] Bailey, J., 1993, A VLSI Interconnect Strategy for Biologically Inspired Artificial Neural Networks, PhD Thesis, Department of Computer Science / Engineering, Beaverton, OR, Oregon Graduate Institute.

[3] Bailey, J. and D. Hammerstrom, 1988, Why VLSI Implementations of Associative VLCNs Require Connection Multiplexing, 1988 International Conference on Neural Networks, pp.173-180.

[4] Boahen, K. A., 2000, Point-to-Point Connectivity Between Neuromorphic Chips Using Address Events, IEEE Transactions on Circuits and Systems II - Analog and Digital Signal Processing , **47**(5), pp.416-434.

[5] Braitenberg, V. and A. Schüz, 1998, Cortex: Statistics and Geometry of Neuronal Connectivity , Springer-Verlag.

[6] Chua, L. and T. Roska, 2001, Cellular Neural Networks and Visual Computing , Cambridge University Press.

[7] Fahlman, S. E. and M. Hoehfeld, 1992, Learning with Limited Numerical Precision Using the Cascade-Correlation Algorithm, IEEE Transactions on Neural Networks , **3**(4), pp. 602-611.

[8] Hammerstrom, D., 1995, A Digital VLSI Architecture for Real-World Applications, An Introduction to Neural and Electronic Networks , S. F. Zornetzer, J. L. Davis, C. Lau and T. McKenna, San Diego, CA, Academic Press, pp.335-358.

[9] Hatano, F. and et al., 1999, Implementation of Cell Array Neuro-Processor by Using FPGA, International Joint Conference on Artificial Neural Networks, Washington DC, IEEE.

[10] Hennessy, J. L. and D. A. Patterson, 1991, Computer Architecture: A Quantitative Approach , Palo Alto, CA, Morgan Kaufmann.

[11] Intel, 2001, IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture, Intel, **2001**.

[12] Reschke, C., T. Sterling, et al., 1996, A Design Study of Alternative Network Topologies for the Beowulf Parallel Workstation , High Performance and Distributed Computing.

[13] Shan, H., J. P. Singh, et al., 2000, A Comparison of Three Programming Models for Adaptive Applications on the Origin2000 , Proceedings of SC2000, Dallas, TX.

[14] Sharma, A. K., 1998, Programmable Logic Handbook - PLDs, CPLDs & FPGAs , McGraw-Hill Handbooks.

