Designs for a Cortically–Inspired Neurocomputer Architecture

Eric Means
B.S., Washington State University, 1984

A thesis submitted to the faculty of the
Oregon Graduate Institute of Science & Technology
in partial fulfillment of the
requirements for the degree
Master of Science
in
Computer Science and Engineering

December 1991

The thesis "Designs for a Cortically–Inspired Neurocomputer Architecture" by Eric Means has been examined and approved by the following Examination Committee:

Dan Hammerstrom
Associate Professor
Thesis Research Adviser

Todd Leen
Assistant Professor

William L. Bain
Adjunct Professor

Acknowledgements

Many individuals contributed to this work and deserve my thanks. They include: the members of my thesis committee (Todd Leen and Bill Bain) for their patience and helpful review comments; Tom Baker, for his patient software skills that helped me through numerous 'C' programming adventures; Jon Inouye for putting together a LaTeX package to use for this thesis; and my parents, Lamar and Darlene Means, for providing many years of emotional support.

My graduate studies and this thesis would not have even begun without the support of the Semiconductor Research Corporation, the Office of Naval Research, and the Department of Defense Advanced Research Projects Association. Their support is gratefully acknowledged.

The research agenda of my advisor, Dan Hammerstrom, originally attracted me to the Oregon Graduate Institute. His continued support despite my innumerable delays and broken deadlines has been nothing short of amazing. His technical and organizational skills have forged a substantial neural network research and development effort at and around OGI, and I consider myself fortunate to have been caught up in the whirlwind. For these and other reasons, I say thank you.

Finally, my wife Carol has been unbelievably supportive during my years of full and part-time graduate work. This thesis would have been impossible without her. More important, I often think that *I* would have been impossible without her. She deserves and gets my thanks, gratitude, and love.

Contents

Acknowledgements iii

Abstract vi

1 Introduction 1

2 The Piriform Model 4

2.1 The Complex Model – Model I 5

2.1.1 Network Organization 5

2.1.2 Biological Basis of the Model 10

2.1.3 Summary of Model I Operation 12

2.2 Model II 13

2.2.1 Network Organization 14

2.2.2 Patch Organization 14

2.2.3 Signal Processing Using Model II 16

2.3 Summary 20

3 Architectural Issues 21

3.1 Traditional Architectures 21

3.1.1 Summary of Traditional Architectures 24

3.2 Neurocomputer Architecture 25

3.3 Conclusion 26

4 Execution of the piriform model on the CNAPs neurocomputer 27

4.1 The CNAPS Architecture 27

4.1.1 Processor Organization 29

4.1.2 Mapping the piriform model onto CNAPS 31

4.1.3 The Piriform Algorithm on CNAPS 33

4.2 Model Performance 35

4.3 Conclusion 39

5 The SS Architecture **41**
5.1 Parallelism in the Piriform Model 41
5.2 The SS Architecture 42
5.2.1 Connectivity 42
5.2.2 Neural computation 43
5.2.3 Weight storage & modification 44
5.3 Summary 44

6 Analog Implementation of the SS Architecture **46**
6.1 The Analog Appeal 46
6.2 Analog Processor Designs 47
6.2.1 Analog EEPROM 52
6.2.2 Practical Drawbacks of Analog EEPROM 57
6.2.3 Inter–processor communication 60
6.3 Analysis of analog implementation 60
6.3.1 Analog performance 60
6.3.2 Analog cost 64
6.4 Summary 66

7 Digital Implementation of the SS Architecture **69**
7.1 An analog model in the digital domain 69
7.2 SSD 70
7.2.1 Overview 70
7.2.2 Communication 72
7.2.3 Weight storage and learning 74
7.2.4 Neural computation 76
7.3 Cost/performance estimates 78
7.3.1 Cost: silicon area estimate 78
7.3.2 Performance: input vector processing rate 83

8 Conclusion **87**
8.1 Future work 92

Bibliography **95**

Appendix A: Area Estimation Program **99**

Abstract

Designs for a
Cortically–Inspired
Neurocomputer Architecture

Eric Means, M.S.
Oregon Graduate Institute of Science & Technology, 1991

Supervising Professor: Dan Hammerstrom

Artificial neural networks constitute attempts to put computer systems on the road to natural intelligence. Promising applications await even limited progress in this direction, as challenging problems in perception, control, and reasoning resist traditional computational approaches. Unfortunately, the capabilities and organizations of today's artificial neural networks are far removed from those of their biological namesakes. This difference will narrow, however, as increasing understanding of nervous system operation is incorporated into artificial neural network design. The hardware performance required by such networks will not be met by conventional computers; new architectures will be needed that support the processing requirements of nervous system organization.

An investigation is described of neurocomputer architectures and circuit designs required to implement a neural network modeled directly from studies in mammalian cortex. Previous research in electronic neural network hardware has targeted neurons performing peripheral sensory roles, such as in the auditory and visual systems, or has viewed neural network operation as routine (albeit massively parallel) vector processing. The cortical model used here is the piriform model of olfactory/piriform cortex, developed by Richard Granger and Gary Lynch at the University of California, Irvine.

The "piriform model" is actually a spectrum of models ranging from biochemically detailed descriptions to abstract signal processing versions divorced from biology. Two points in this range will be studied. The first is a high-level model description that has been shown to perform a useful signal processing function known as vector quantization. An implementation of this model on a commercial neurocomputer will be described. The second version of the model, containing more biological detail, is the primary focus of the architectural and circuit design effort. A neurocomputer architecture called "Super Sniff" (SS) is proposed, and two possible implementations are studied. One features analog processing and direct connections between neurons, and the other utilizes a shared communication structure and digital processing. The cost/performance tradeoffs of each are compared.

Sparse temporal activity and connectivity, limited precision arithmetic, and a discrete-event style of operation are characteristics of the model that are compatible with VLSI processing. Mammalian olfactory cortex contains many millions of neurons; this thesis investigates networks containing more modest neuron counts numbering in the tens of thousands.

Lessons learned about the suitability of VLSI processes for implementing biologically-faithful cortical structures are discussed in the conclusion.

Chapter 1

Introduction

Neurocomputers are machines specialized for the rapid execution of neural network algorithms. Specialized hardware is essential for the practical application of neural networks because these algorithms are computationally intensive and impractically slow on general purpose platforms. The application domains where neural networks are expected to have their greatest practical application require real–time execution, meaning they must execute very quickly.

Research on neurocomputer hardware has tended to split into two camps. One focus has been on the design of electronic neurons; that is, circuits that electronically perform the basic operations of individual biological neurons. The work of Graf and Jackel [GJ89], Hollis and Paulos [HP90b], Meador [MWC+91], and Card [CM89b] is typical of this effort. Carver Mead at Cal Tech [Mea89] is particularly well known for his group's work in this area using sub–threshold CMOS processes as their electronic substrate. This research is generally aimed at developing specialized analog circuits, and in general has not emphasized the integration of these special–purpose circuits into entire systems that perform cortical–like associative processing. A good reason for this is the paucity of good cortical models; modeling the brain is an ambitious research undertaking.

The other neurocomputer research path has aimed at producing machines akin to traditional vector processors. Here there is no attempt to create electronic analogs of biological structures; instead, neural network algorithms are viewed as simply being highly pipelined or parallel operations on massive amounts of data. The neural network algorithms that are the focus of these architectures are only loosely modeled on biological principles. Largely for that reason, the architectures that implement these models are referred to here as *first-generation* neurocomputers.

Recently, some neural models have emerged that model a wider range of neural network operation, from individual to entire assemblies of neurons. A prominent example is the olfactory/piriform cortex model (the *piriform model*) developed by Richard Granger and Gary Lynch of the University of California at Irvine [GAIL89][GAISL89][AIGL90]. Characteristics of this class of models make them sufficiently different from the artificial neural networks in widespread use today (e.g., backpropagation) to describe them as *cortically-inspired* models. The distinguishing elements of these models are their close adherence to biological nervous system organization, and their functional descriptions of neural assemblies (as opposed to simply .the operations of individual neurons).

First-generation neurocomputers have performance problems with this class of neural models. This is largely a result of the latter's biological pedigree; these models feature the massive connectivity and computational characteristics of real biological nervous systems. These models will require new computer architectures if they are to be executed in critical real–time environments.

The purpose of this thesis is twofold: first, to propose a computer architecture optimized for the execution of a neural algorithm derived from a cortical model, and second, to establish an approach for translating biological neural networks into practical neurocomputers.

The piriform model will be the focus of examining how to integrate electronic neural components into a complete neurocomputer system. It offers a clean input/output interface; internal operations are well defined; and any implementation of it must address the salient points of neurocomputer architecture.

The way that neurocomputer architectures are analyzed is an important part of this thesis. Any architecture, implementing any neural model, can be evaluated on how well it balances three key implementation issues: (1) inter-neural communication; (2) weight representation and modification (i.e., learning); and (3) neural computation and precision. These three issues frame the discussions in the following analyses.

The thesis will begin with a detailed description of the piriform model. The model includes several novel biological features that combine for an interesting computation, but they complicate an understanding of the processing in traditional signal processing terms. A simplified version of the piriform model will then be described that performs *hierarchical vector quantization*. Simulations will show the operation of this analytically tractable model variant.

After these descriptions, an implementation of the simplified network model on a commercial neurocomputer will be described. An analysis of two proposed implementations of the full-scale piriform model will follow: an analog and a digital version. The strengths and weaknesses of these two approaches will be covered in some detail.

This thesis represents one example of how a biological neural model can be translated into the language of computer architecture. It is hoped that the conclusions and methodology will have application to the subsequent generations of models and architectures that are certain to follow as these models attain more widespread usage.

Chapter 2

The Piriform Model

The piriform model emerged from attempts to explain observed laboratory phenomena. Described here are successive stages in an ongoing evolution; early versions captured significantly more anatomical and biophysical detail than later ones, and with each successive generation, more of the fine points of operation were abstracted into higher–level behavioral rules. That evolution will be reflected in two versions of the model discussed here. The first, called model I, is faithful to significantly more biological detail than its successor, model II.

Many of the novel features of model I are artifacts of the biology, and it is postulated that they are only peripherally related to the primary computation. Therefore the model's developers introduced model II, which strips away much of the biological detail and introduces behavioral rules intended to capture the essential elements of the computation. In its resulting form model II, in combination with features of the surrounding cortical structures, can be recognized as a special case of a more general class of signal processing algorithms.

This chapter discusses both versions of the piriform model. Model I performs an interesting computation, but its analysis is complicated by the many biological features it retains. The simplifications introduced in model II suggest an evaluation in terms of a traditional signal processing algorithm. It remains to be seen if model I can also be viewed in this way, or if what model II dismisses as "biological detail" is essential to the biological computation. In this thesis, both possibilities will be covered. Following its description in this chapter, an implementation of model II on a commercially available neurocomputer will be described. The remainder of the thesis will focus on neurocomputer designs that implement the biologically explicit model I in its entirety.

2.1 The Complex Model – Model I

2.1.1 Network Organization

The piriform model is abstracted from piriform layer II, and constitutes one portion of the entire mammalian olfactory system. Figure 2.1 shows a schematic diagram of piriform layer II embedded within this system. The olfactory receptor sheet is the primary sensory surface for the olfactory system. While much of the nature of olfaction is not understood, it appears that neurons in the sheet respond with high-frequency oscillations when particular airborne molecules impinge on the sheet. Signals on the olfactory nerve are frequency-coded: stronger excitations elicit higher frequency pulse trains.

These signals undergo significant modification within the olfactory bulb. An automatic gain control operation on input activity keeps bulb output (signals on the Lateral Olfactory Tract, or LOT) largely constant regardless of the intensity of olfactory nerve input. In addition, the bulb scrambles the spatial relationship of signals from olfactory nerve to the LOT. While topography is preserved on the nerve, neighboring signals on the LOT bear no spatial relationship to one another. Finally, the frequency-coded information on the olfactory nerve is converted to time–locked, synchronous, binary (off/on) data on the LOT.

LOT synchronization results from the interplay of excitation in mitral tufts and mitral cells (those cells whose axons form the LOT), and feedback from inhibitory granule cells. Figure 2.2 shows the postulated mechanism of how olfactory nerve input is converted into periodic pulsed LOT signals [She79]. Granule cells suppress the mitral cell pulses that activated them. This inhibition lasts for some unknown period of time.

The LOT, which forms the inputs to layer II, consists of mitral cell axons originating in the olfactory bulb. These axons sparsely contact distal dendrites of layer II neurons. The neurons are arranged in a linear sequence of "patches", each composed of between

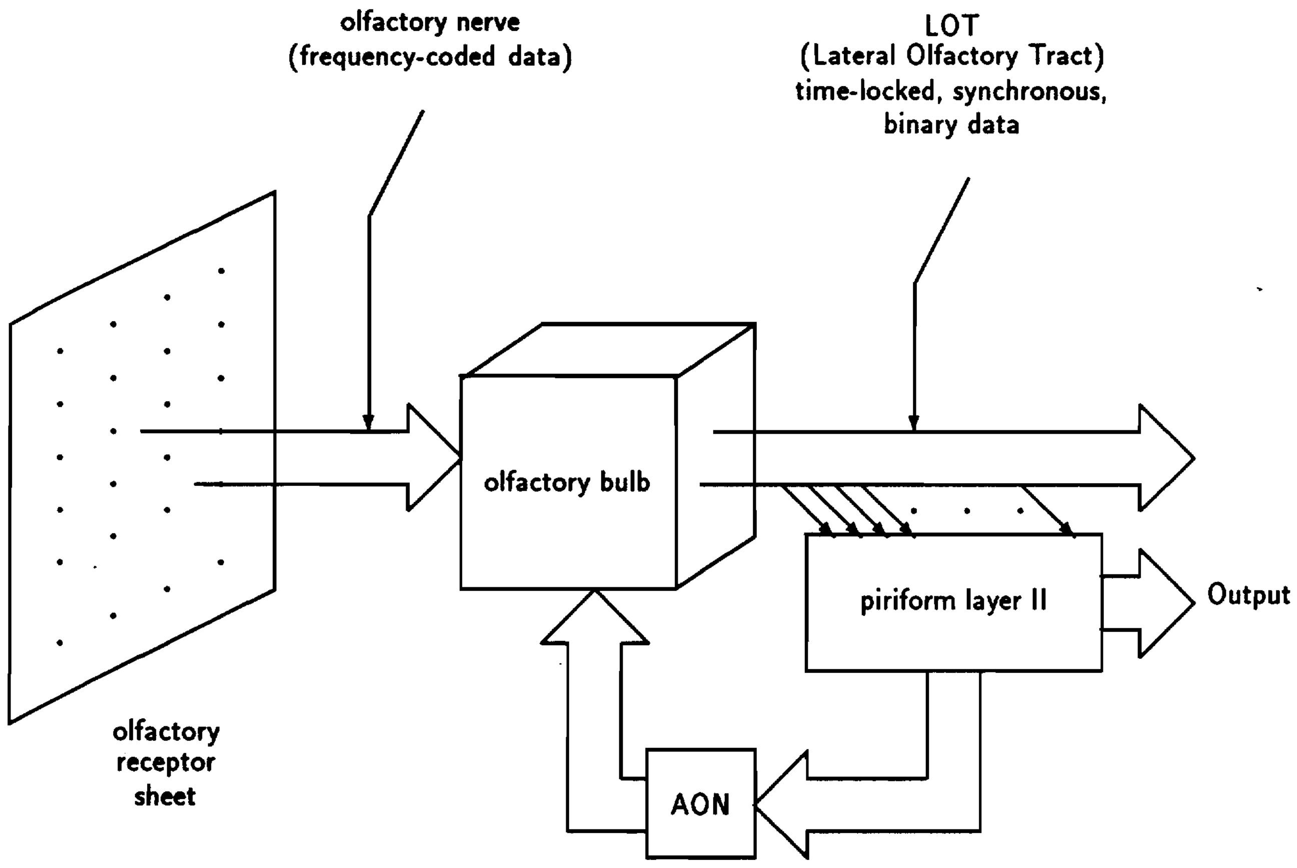


Figure 2.1: Schematic diagram of the mammalian olfactory system.

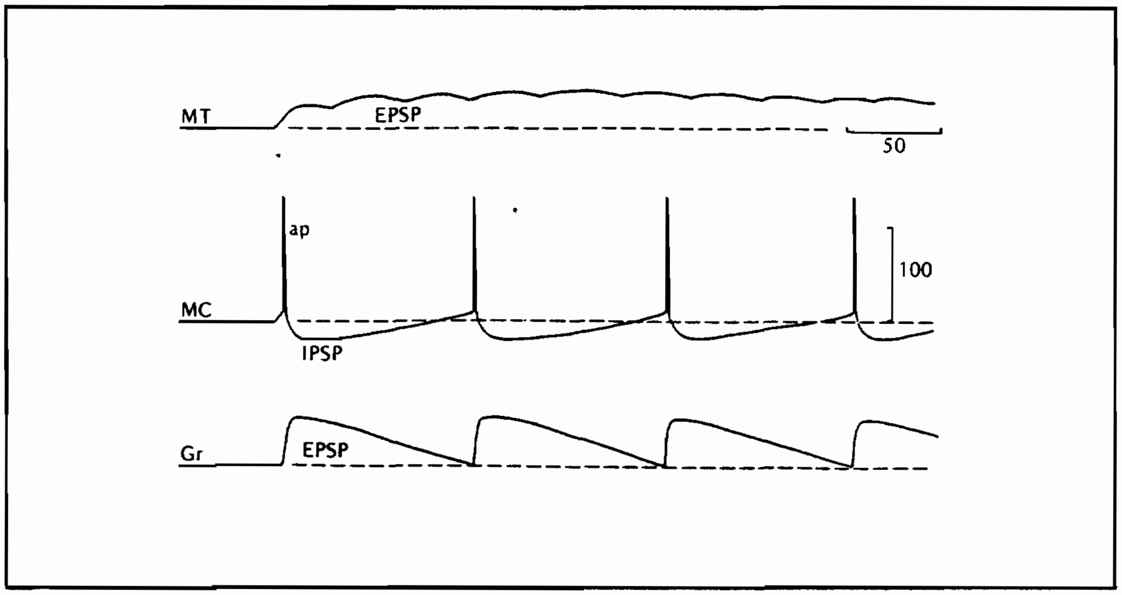


Figure 2.2: Olfactory input gives rise to a constant excitatory post-synaptic potential (EPSP) within mitral cell tufts (MT) in the bulb: the resultant excitation in the mitral cells (MC) energizes inhibitory granule cells (Gr), which temporarily suppress all mitral cell activity. The resulting synchronous pulses within the mitral cells (MC) reflect the recharging rate of the inhibitory granule cells. Mitral cell axons form the Lateral Olfactory Tract (LOT). The horizontal bar in the figure at top represents 50 milliseconds of time; the vertical bar in the middle figure measures 100 millivolts of depolarization. From Shepherd, *The Synaptic Organization of the Brain*, 1979, pg. 171.

20 to 100 neurons.

The axon from each layer II cell rejoins the LOT and propagates caudally, replacing a randomly chosen axon from a prior source. Figure 2.3 shows a small version of layer II (two patches, four cells each) and the nature of LOT to layer II connectivity. As recurrent collaterals randomly replace rostral (upstream) axons, LOT size remains constant during traversal of layer II. As a result LOT composition changes as it progresses from the rostral portions of the network to caudal (downstream) regions. The effect is that dendrites of caudally–located neurons receive more highly–processed signals than do those of rostrally–located neurons.

Processing in the piriform model proceeds in distinct cycles, or *sniffs*. Olfactory input appears on LOT axons in the form of periodic samples, or pulse trains, one per sniff. On every such sample, each neuron forms a scalar product of its own synaptic weight vector with a sparse subset of the LOT. Local lateral inhibition causes each patch to perform a winner–take–all function; only the most strongly activated neuron(s) exceeding a minimum threshold is allowed to fire. The remainder of the patch is quiescent.

Neurons that have won the competition within a patch on any sample respond with a pulse train on their own axonal output. In subsequent samples, however, a post–activation refractory period prevents those neurons from winning their patch again. Successive patch winners are selected from neurons whose dendrites sample a different region of the LOT input space. Model II describes enhancements that add an active feedback path from layer II to the olfactory bulb via the anterior olfactory nucleus [GAISL89]. The function of this path is cycle–by–cycle inhibition of dominant components of the LOT input, based on input from patch winners.

Feedback inhibition and post–activation refractory periods combine to cause the network to form different outputs on successive samples of a single LOT input. It is hypothesized that the resulting series of responses establishes a hierarchical categorization of inputs, corresponding to a hierarchical structuring of the data. Similar inputs initially elicit similar network outputs, but with each successive sniff the elements common to each input are removed and the network responds to progressively dissimilar components.

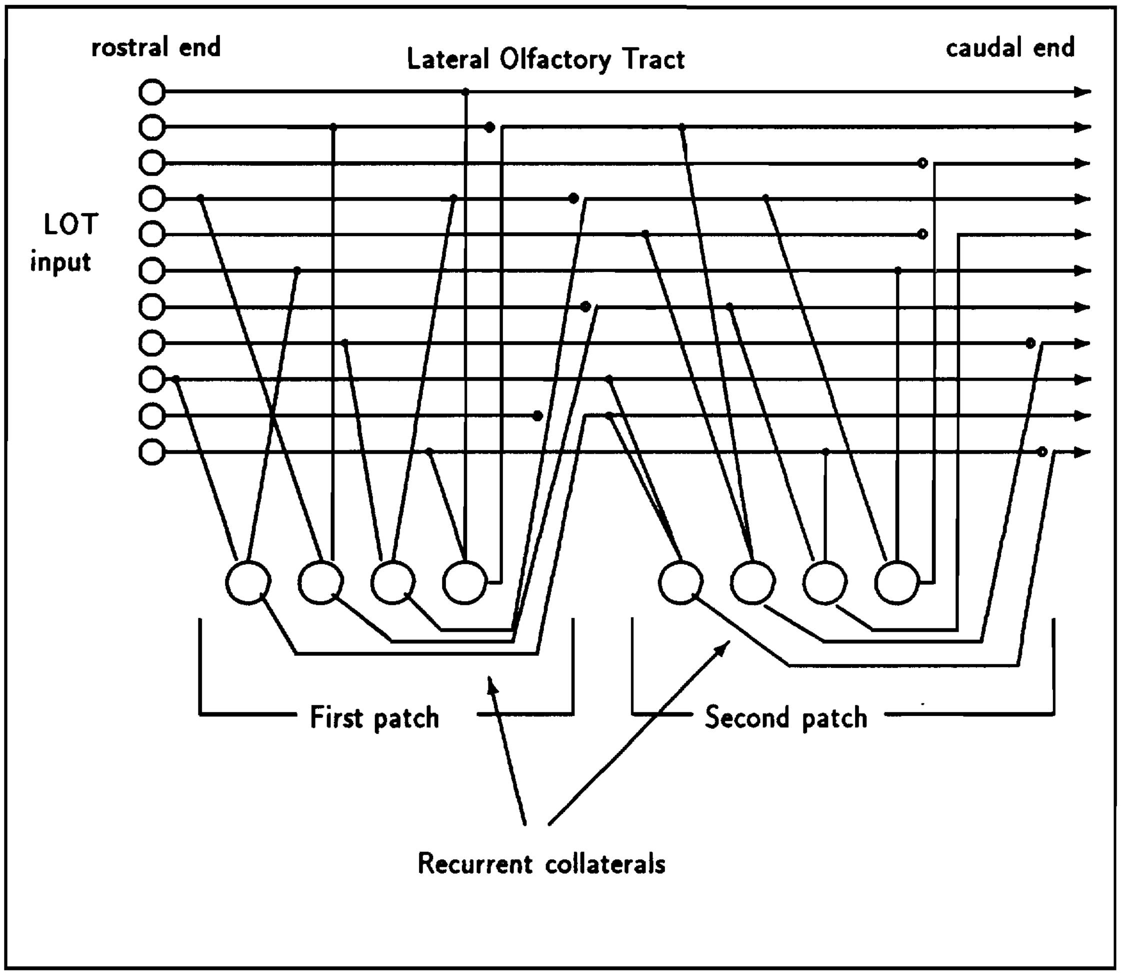


Figure 2.3: An illustration of the connectivity between layer II patches and the Lateral Olfactory Tract (LOT). Cell outputs rejoin the LOT, randomly replacing a LOT signal. Not shown in this figure are inhibitory interneurons within each patch which enforce winner–take–all behavior.

Hence, network outputs in later sniffs diverge.

Neurons in the model have two distinct operating regimes: learning mode and performance mode. Inputs cause similar network responses in both modes, but only in learning mode can synaptic weights be changed. Only in performance mode do neurons use their "learned" synaptic weight values: in learning mode itself, when neurons are competing to win their patch, all neurons use a constant value for synaptic weights. (Different biochemical mechanisms are postulated to be acting in the two modes.) This feature removes prior learning effects from subsequent training episodes.

2.1.2 Biological Basis of the Model

The network model can be understood without recourse to biology, but a brief description of the biological basis provides some justification for its characteristics.

The learning mechanism is based on models of long term potentiation (LTP). Increases in synaptic efficacy of approximately 50% have been brought on with remarkably little onset time, and have lasted for weeks in test cases. The increase commonly outlasts the lifespan of the monitoring equipment, and may be considered permanent [GAIL89]. The mechanism for LTP is presently unknown; some theories attribute it to a change in synaptic biochemistry, while others suggest changes in the geometry of the synapse. While there is growing evidence for the characteristics of LTP described here (incremental increase up to a ceiling value), there is at present little evidence for its supposed counterpart, long term depression [GAIL89]. Hence, in this model, synaptic weights can only increase with learning.

Laboratory evidence indicates that synaptic weights change only when inputs are coincident with a global low frequency oscillation in piriform cortex known as the "theta rhythm". This 4–7 Hz oscillation is apparently caused by signals arising from other cortical structures innervating piriform layer II. Theta rhythm onset typically accompanies a heightened awareness level in actively exploring animals, evidently permitting the synaptic modification that constitutes LTP learning. Theta rhythm oscillations combine with olfactory input to boost cellular activation beyond voltage depolarization thresholds.

In active animals, the difference between learning and performance modes is observed in the cyclical pattern of network activity. In performance mode, LOT inputs appear as brief low–frequency ($\approx$ 40 Hz) voltage spikes. In learning mode, the identical input causes a flurry of high–frequency ($\approx$ 100 Hz) pulses on the LOT. Synaptic efficacy increases by an incremental amount when: (1) the network is in learning mode, (2) the neuron wins its patch and exceeds the learning threshold, and (3) the synapse has not yet reached its ceiling value.

In the model, every neuron's learning mode synaptic weights differ from its performance mode synaptic weights. It is hypothesized that the higher–frequency learning–mode inputs utilize a different cellular synaptic transmitter than is used in lower–frequency performance mode. Experimental evidence suggests that the presence of this learning–mode transmitter acts to induce LTP by opening calcium channels in the neural membrane. The calcium influx alters the efficacy of performance–mode synaptic transmission without affecting learning–mode synaptic transmission (hence the bi-modal synaptic weight rule).

Local interneurons provide the inhibition that is responsible for the winner-take-all behavior of the patches. A neuron that is sufficiently excited beyond an activation threshold by afferent inputs in turn energizes an inhibitory neuron, and the latter suppresses all activity in the patch via strong shunting inhibition currents. The brief, high frequency output spiking of the excited neuron represents this patch's active output on one cycle, or one sniff. A lengthy refractory period follows the high frequency spiking, and during this period no further activity by that neuron can be observed. This process repeats for each successive sniff in performance mode.

Input to layer II pyramidal cells is delayed by LOT transmission time and the processing time of upstream layer II cells. Model I operation requires virtually coincidental arrival times of inputs, and thus some synchronization between these two delays. Anatomical evidence indicates that transmission times within piriform cortex may satisfy this requirement. Noted piriform researcher Lewis Haberly describes the anatomical situation as follows:

In the association fiber system there is an interesting trend in the organization of the projections to the anterior piriform cortex. Longitudinally oriented strips of cortex at increasing distance from the LOT get associational inputs from olfactory cortical areas at progressively more caudal locations. Since afferent volleys from the olfactory bulb propagate down the LOT, then spread medially and laterally via fine collaterals at a much slower rate, one possible hypothesis concerning this organization is that it keeps the time of arrival of afferent and associational inputs relatively constant. [Hab85]

The sparse connectivity of the model is derived from known connectivity of LOT axons onto dendrites of neurons in piriform layer II. Connectivity reportedly varies between 5% and 25%; that is, any LOT axon has a 5% to 25% probability of contacting a dendrite of any given layer II piriform neuron.

2.1.3 Summary of Model I Operation

Learning mode:

1. An input vector is presented to the network; this represents afferent signals on the Lateral Olfactory Tract (LOT).

2. Neurons in each patch compute their activations from the subset of active LOT signals that each contacts. The first neuron in each patch whose activation exceeds the winning threshold is the winner of that patch. Its output signal, through a ***recurrent collateral***, randomly replaces a previous LOT signal.

3. Each patch continues to compute activations. Moving caudally, the LOT content changes as additional recurrent collaterals replace original input. Rostral patches generally produce the initial patch winners because the LOT population they contact contains a higher proportion of active inputs. Caudal patches see more recurrent collaterals, whose activation is delayed by the processing time of rostral neurons. Most of the delay is due to neural response time (on the order of 10

milliseconds) with some additional delay caused by the slower propagation time of the unmyelinated (i.e., slower) recurrent collaterals. Hence, the output of caudal portions of the network are delayed in time.

4. If a winning neuron achieves an excitation level exceeding a learning threshold, that neuron increases the weight of each active synapse (an active synapse is one that had an active input on that cycle) by a standard increment.

Performance mode:

1. Repeat steps 1–2 above, this time using performance–mode synaptic weight values.
2. Every neuron's output is combined to form the entire network's "first–sniff" response.
3. Using the same LOT input, steps 1–2 above are repeated. This time, each previous winner is inactive, so every patch will produce a different winner than before.
4. Again, every neuron's output is combined to form the network's "second–sniff" response. This process continues for four to six sniffs, with each patch producing a different winner on every sniff.

2.2 Model II

The simplified version of the piriform model is completely described in [AIGL90]. Model II takes a step back from model I's detailed representation of piriform layer II and encodes many of the first model's details into a simpler set of rules. At the same time, model II attempts to incorporate portions of the olfactory system that lie outside the scope of the detailed model. In particular, the second version models feedback from layer II via the Anterior Olfactory Nucleus (AON) to the network input, the olfactory bulb (refer to figure 2.1 for illustration).

2.2.1 Network Organization

Model II changes many of the spatial and temporal characteristics of model I. In the latter, hierarchy is developed over time with successive responses from the entire network. The hierarchical outputs of model II are instead attributed to *subnets*, roughly analogous to model I *patches.* Each subnet is responsible for one level of hierarchy, and the LOT is presented to only one subnet at a time. Model II may be considered to be transforming model I's temporal representations into spatial representations; each subnet in model II corresponds to temporal response in layer I. Connectivity is altered in model II also. Previously, recurrent collaterals from each patch replaced random LOT elements in the propagating LOT bundle; in this way, each subsequent layer sees a slightly different LOT input. Instead of altering the LOT composition, model II adjusts the value of each LOT element between successive patches. This is done by subtracting the weight vector of each subnet's winning cell from the LOT vector prior to submitting the LOT vector to the next subnet (see figure 2.4).

Model II is further simplified by substituting complete LOT–to–patch connectivity for the sparse connectivity of model I. That is, in this version, each LOT element contacts every neuron in each layer.

2.2.2 Patch Organization

Subnets in model II behave similarly to patches in the detailed model, with some interesting differences. Each cell calculates the Euclidean distance between its weight vector and the afferent input vector instead of computing the inner product, as is done in model I. The Euclidean distance d between weight vector x and LOT vector y (each of length n) is defined as:

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

(The distance measure was substituted because it produced slightly superior performance than the scalar product in network simulations. This is a departure from the biological basis of model I [LRW91].) The cell with the smallest distance is the patch winner.

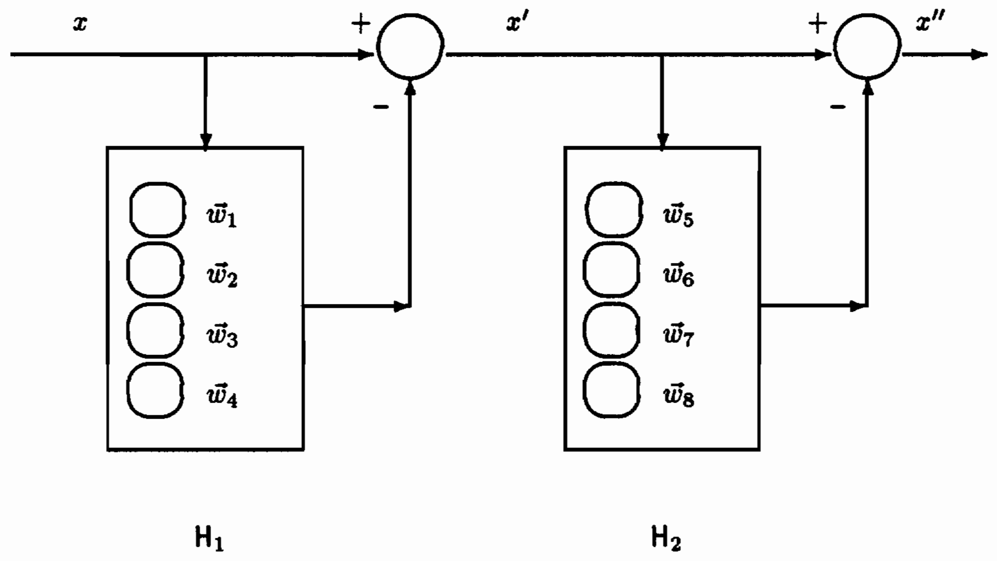


Figure 2.4: Schematic view of a two–layer piriform model II with four cells per layer. The network input vector x is modulated by feedback from each layer's winning cell prior to presentation to the next layer. H_1 and H_2 represent the first and second network layers, or subnets. $\vec{w}_1$ through $\vec{w}_8$ represent the weight vectors of the eight cells.

The winner's weight vector is then subtracted from the LOT vector prior to presenting the LOT to the next subnet. Learning in model II occurs only when the network is in learning mode, just as in model I. At those times, each subnet's winning cell adjusts its weights according to the following equation:

$$\vec{w}(t+1) \leftarrow \vec{w}(t) + \gamma(\vec{x} - \vec{w}) \tag{2.1}$$

The result is to move the weight vector $\vec{w}$ closer to the input vector $\vec{x}$ by an amount proportional to the learning rate γ.

In model I, the network forms a series of outputs as winning cells from *multiple* patches are simultaneously activated on each sniff. In contrast, in model II, an *individual* subnet is responsible for each output in the series. A three–subnet network will form a series of three outputs to each input vector, for example. The hierarchical response that previously resulted from the refractory periods of firing cells is, in the simple model, a result of a simplification: the LOT input vector is presented to a single subnet at a time, and each subnet's response forms a stage in the network's overall hierarchical response.

2.2.3 Signal Processing Using Model II

When viewing the operations of the simple model from a signal processing perspective, the piriform model can be considered to be a *multistage vector quantizer*. This is the statistical encoding of input vectors in order to quantize and compress them [AKCM90]. In this formulation, the piriform model resembles vector quantization algorithms described in [Gra84]. A LOT input vector is encoded as a *signature* composed of the unique identifiers of the cells in each subnet that "won" the competition for that input. For example, consider a three–layer network composed of three subnets, each with 5 cells. Cell number 2 (out of 5) wins the first subnet's competition: cell number 1 wins the second, and cell number 4 wins the third. The network encodes the input with a signature of 2-1-4. A decoder storing the weight vectors of every network cell eventually translates 2-1-4 into three weight vectors, which are summed to yield an estimate of the original LOT input vector.

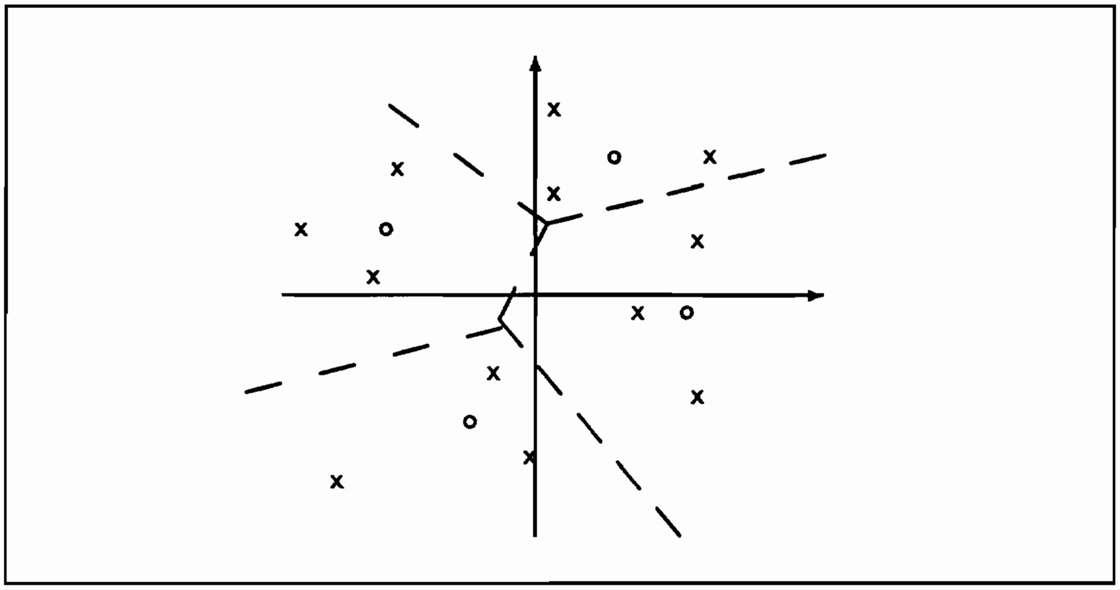


Figure 2.5: Optimal partitioning of a two–dimensional input space by an unsupervised classification network. Dotted lines represent partitions that reflect the clustering shown by a set of 11 input vectors, each of which is represented by an x. Each o is located at the centroid of a partition. From Gray, Vector Quantization, IEEE ASSP Magazine, 1984.

Vector quantization algorithms map continuous inputs onto a discrete space; they effectively *classify* inputs as belonging to one member of a non–overlapping set. The challenge is to develop a mapping that preserves the fidelity of the data within some error rate at an acceptable computational cost. The simplified piriform model resembles algorithms for optimal encodings [Gra84]; that is, minimum distortion encodings. The idea of an optimum encoding is to let the statistics of the input ensemble determine the partitions of the input space. Networks that perform this type of encoding are also known as ***unsupervised classification algorithms***. Figure 2.5 shows a near–optimal partitioning of a two–dimensional input space. Each input vector falling within the space of figure 2.5 would be *quantized* into one of four signatures that represent the four partitions. Optimal encodings adjust the partitions to represent the clustering of the input vector ensemble.

The simplified piriform model adds a hierarchical twist to this basic algorithm. Normally, each subnet forms its own division of the input space; a three–layer network would perform three consecutive partitions like the one shown in figure 2.5. As an example of the operation of the piriform model in a vector quantization role, figure 2.6 shows how a sample network partitioned 32–element input vectors representing vowel data from the phonemes *A, E, F, O*, and *R*. This is a three-dimensional figure, with each dimension representing one of the three principal components of the data. (These components contain most of the information in this 32-dimensional data set) The length of the sides of the cube are proportional to the magnitudes of the three principal components. Dots of four different sizes are shown in the figure. The smallest dots show the spread of LOT data. The other three dot sizes represent weight vectors; the largest is the weight vector of the single cell in the first layer; note that is has moved to the centroid of the input data. The four second-largest dots represent the weight vectors of the four second-layer cells. Each of these are surrounded by a four-cell constellation of weight vectors from the third layer. Although there are only four cells in the third layer, they move their weight vectors to different positions corresponding to different second-layer winners. Knowing the identity of each layer's winning cell and the weight vectors of each allows classification and estimation of the LOT input vector.

Performance of model II

Extensive classification experiments using spectral data from human speech were done in [LRW91] using piriform model II. The experiments were intended to measure the performance of several variations of the basic model. Spectral data from the phonemes *A, E, F, O*, and *R* was used. Each LOT input vector was generated by first digitizing an utterance at a 16 kHz sampling rate using 16-bit accuracy. Following a discrete Fourier transform, the spectral coefficients of the lowest 32 DFT coefficients were used as the LOT input.

The vowel phoneme data is naturally clustered, and for that reason makes a good target for an unsupervised classification algorithm like the piriform model. Results of simple experiments using small model II networks (typically three subnets, each composed of from one to five cells) showed classification accuracy ranging from 86% to 90.6%.

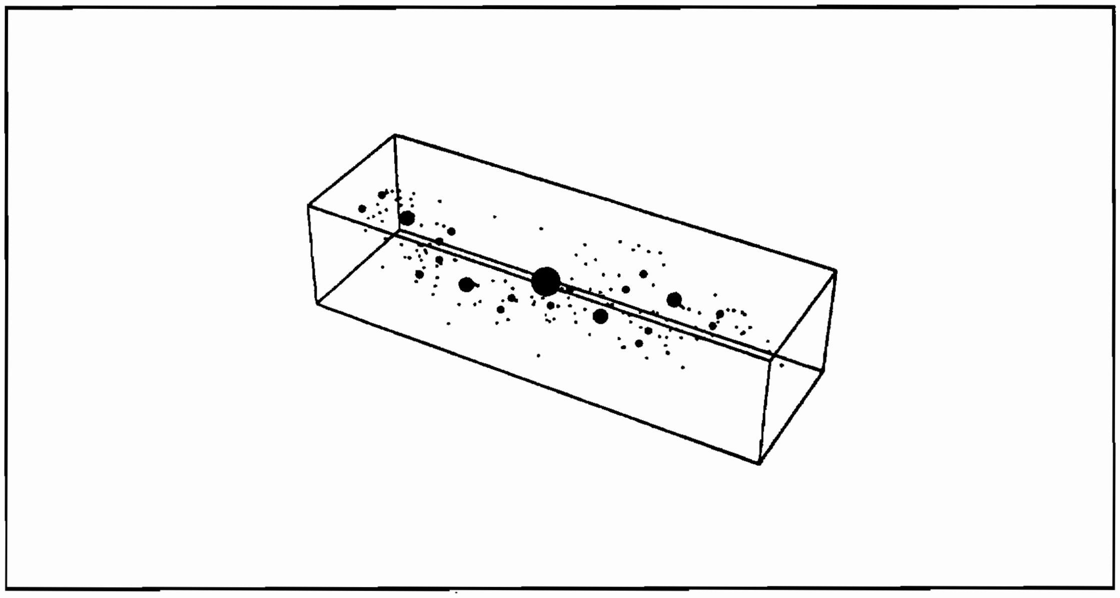


Figure 2.6: A plot of weight vectors as they track spectral components from human vowel utterances. The smallest dots represent speech data components plotted against the three principal components of the input data. The large dots represent the weight vectors of cells in a 1-4-4 model II piriform network. See text for additional explanation. From Webb, Hierarchical Competitive Learning, 1991.

This compares to the approximate 97% accuracy of supervised perceptron networks performing the same task. Unsupervised algorithms like model II classify strictly according to the statistics of the input ensemble. They do not require the global learning signals required by perceptron networks.

2.3 Summary

Model II emphasizes the hierarchical clustering characteristics of model I and discards most of the latter's biological detail. Features of model II that make it analytically tractable are primarily the complete connectivity between the LOT and each subnet, and the fact that only a single subnet at a time is exposed to the LOT. The resulting behavior of model II can be likened to well–known multistage vector quantization algorithms.

Model II discards many of the features that make model I unique. It remains to be seen if these features are crucial elements in the processing of piriform cortex. In any event, the purpose here is to propose a neurocomputer architecture for the execution of the piriform model. The existence of two versions of the model, one less exotic than the other, argues for discussing two separate architectures. That is the approach to be followed in the remainder of this thesis. Following an excursion into general neurocomputer principles, an implementation of model II (the simpler model) will be described on a commercial neurocomputer. I will refer to the neurocomputer as an example of a "first generation neurocomputer" because it is designed to support neural network algorithms loosely modeled on biology.

The commercial neurocomputer presents a good starting point for exploring implementations of an architecture to implement model I. It will be seen that particular features of model I such as precision requirements and connectivity have a crucial bearing on the design of the hardware. The purpose of this chapter was to introduce the piriform model in both its flavors, to provide some biological background, and to suggest the nature of the computation that it performs. The remainder of this thesis focuses on architectural and implementation issues of both piriform model versions.

Chapter 3

Architectural Issues

The piriform model dispenses with most of the transient complexity of real neurobiological systems, and involves mostly large numbers of simple scalar products. Scalar products are familiar elements in computational models of many physical processes, and computers that excel at their rapid computation have existed for many years. Before proposing a new architecture, it is worth examining why traditional computers are insufficient platforms for executing this model. This chapter begins with a brief discussion of the strengths and weaknesses of some common vector processing architectures. A summary of architectural attributes necessary for rapid execution of piriform–like models follows.

3.1 Traditional Architectures

Vector Supercomputers

Formally, the scalar product operation at the heart of the piriform model can be described as:

$$a_j = \sum_{i=0}^{N-1} (x_k) \times (w_{ji}) \tag{3.1}$$

where:

$a_j \doteq$ activation level a of neuron j, $0 < j < P$

$N \doteq$ number of connections that each neuron forms to the LOT

$x_k \doteq$ input value of LOT signal k

$w_{ji} \doteq$ synaptic weight of connection i, $0 < i < N$, on neuron j

Rapid scalar product computation requires a fast CPU for multiply–accumulate operations. The best vector processors achieve between one and three arithmetic operations in one machine cycle. In addition to a fast CPU, a large memory is required to store the state of an entire network of neurons and synaptic weights. The third and most critical architectural element required is a very high bandwidth from main memory to the CPU to support the many interconnections between neurons in a piriform network. This final requirement poses the most difficulty for traditional vector architectures, because the sparse, random pattern of connections in a network degrades most traditional approaches to achieving the requisite memory bandwidth.

To illustrate the need for a large memory, consider that piriform cortex in small mammals contains over 60,000 LOT axons (input signals) and layer II neurons numbering in the millions. A reasonable size estimate for artificial networks based on this model could have from 1000 to 100,000 neurons, and LOT signal populations numbering up to 10,000. The matrices describing connectivity between the LOT and patches in piriform layer II are largely empty; connectivity on the order of 5% [GAIL89] would imply that each neuron in a network of 10,000 neurons and 1,000 LOT lines would have 100 inputs, requiring a total of $10,000 \times (0.05 \times 1,000) = 500$ thousand synaptic weights to be stored in memory. Precision requirements would determine the size of each synaptic weight, but at a minimum of one byte per weight, two megabytes would be required to store the weights in such a network. Networks that approximate actual biological dimensions of piriform cortex could easily require a gigabyte of memory simply for synaptic weight storage.

A two megabyte memory capacity is not a problem for networks of the size discussed in this thesis, but CPU cycle time can be. Typical LOT inputs contain on the order of 10% active signal lines: a network of 10,000 neurons, each with 100 connections, would need to compute 10,000 $\times$ 100 $\times$ 0.1 = 100,000 connections on every sniff; given that four or more sniffs are required to process a single LOT input, approximately 500,000 multiply–accumulate operations would be needed for every LOT input vector. A vector supercomputer capable of a sustained processing rate of 100 megaflops could theoretically process 200 LOT vectors per second. While this rate would be sufficient for real–time processing of audio frequencies, the cost of such a machine would preclude its use in most useful real–time applications. In addition, signals in many application domains (video, for example) require significantly higher processing rates than even a supercomputer can currently provide.

Another serious concern is the memory bandwidth. Numerous techniques have been successfully used to provide high bandwidth to large memories, and these generally require subdividing the main memory into banks that are accessed in parallel. In this way, the high latency penalty of access to individual banks is paid only once in each parallel access. This technique requires that program memory references be consistent with the physical structure of memory; fortunately, most vector operations are highly regular, and map well to such a structure.

Connectivity from piriform layer II to the LOT is relatively sparse in the piriform model, varying between 1% and 20%. The connectivity matrix between the LOT and layer II will be correspondingly sparse. Sparse matrix operations are a traditional challenge to such memory organizations. To prevent the memory access time from degrading to that of a single scalar access, *scatter/gather* operations are used to compress a sparse vector into a dense one prior to processing. While not as fast as pure parallel memory accesses, scatter/gather mechanisms offer an improvement over the alternative of slow scalar access [HP90a]. Unfortunately, because of their cost, scatter/gather mechanisms are usually restricted to multi–million dollar supercomputers.

Killer Micros

The gap between execution times of programs running on supercomputers and desktop workstations (dubbed "killer micros" by their enthusiasts) has been closing rapidly in recent years. Memory organization is an important reason for this. Smaller, cheaper machines have been able to leverage the *spatial* and *temporal locality* characteristics of most programs to keep a portion of main memory in small, high–speed caches, effectively boosting bandwidth of the entire memory to that of the cache. Programs with a large degree of non–local memory references, however, are effective "cache–busters". A machine with a cycle time of 20 nanoseconds can have its performance degraded by a factor of six to ten if data accesses regularly require memory references beyond the limits of the small cache. Modern caches typically range from 32 Kbytes to 256 Kbytes. In the 10,000 neuron network described above, typically the state of only ten to twenty patches could be resident in a data cache at one time. Inputs to each neuron's 100 synapses could come from neurons in any of 200 to 400 random patches, a domain 10 to 40 times larger than the cache capacity. A low hit rate on cache accesses would result due to the unlikelihood that the cache would contain the required data.

Memory bandwidth would then be limited by the access time of a single main memory location. Modern DRAM access times are on the order of 100 nanoseconds, and this parameter has traditionally lagged by a substantial margin the improvement rate of CPU cycle times[1]. Architectures limited by memory access times of 100 nanoseconds would be limited to performances under 10 megaflops. A killer micro executing the piriform model on the 10,000 neuron network described above could process a single LOT input in 50 milliseconds, or perhaps 20 input vectors per second. This is orders of magnitude longer than the time required for real–time execution of the piriform model in an application such as speech recognition.

3.1.1 Summary of Traditional Architectures

CPU processing speed is the primary problem for traditional computer architectures; real–time processing of large networks requires significant compute power. Memory bandwidth considerations pose an additional problem for both vector supercomputer architectures and killer micros. The two techniques used to escape the bandwidth problem require certain program traits for their success: vector machines thrive on regular stride–length memory accesses, and killer micros rely on spatial and temporal locality of

[1]Since 1980, memory access time has improved by an average of 7% per year; CPU performance has improved between 50% and 100% annually since 1985 [HP90a].

memory references. The piriform model algorithm conforms to neither requirement.

3.2 Neurocomputer Architecture

The essense of the bandwidth problem is that as memories get larger, their access time increases. Another way to view each memory access is as a communication between two neurons. In both architectures described above, every inter–neuron communication must transit the same physical path over the main memory bus. Alternative architectures designed specifically for piriform–like neural network models focus primarily on alleviating this communication bottleneck. A straightforward solution is to distribute synaptic weight memory among multiple processors so that multiple communications can occur in parallel. Each processor would access a smaller, and thus a faster, memory. As in all distributed computation environments, inter–processor communication then becomes a primary architectural concern.

An additional architectural refinement concerns the *precision* of neural computation. Most machines today emphasize 32–bit floating point operations, yet it has been shown [BH89] [Bak90] [HB91] that most neural network computations using the backpropagation algorithm require significantly less precision. Research on model II has indicated that following training, classification performance can be reduced to between 4 and 8 bits before a measurable falloff is observed [Web]. Piriform–like networks that conform closer yet to biological models can be expected to require even less numerical precision than do current generation network algorithms. It is unlikely that noisy, faulty biological neurons are capable of significant precision; precise results instead emerge from the collective efforts of the entire network.

Neurocomputer architectures offer numerous opportunities to take advantage of reduced precision. Foremost among these are the near linear reductions possible in processor and synaptic weight memory storage area, increasing the number of processors that can be built in the same area. However, when precision requirements are sufficiently low, analog computation becomes a viable alternative to digital computation because analog arithmetic circuits can be packed more densely than equivalent digital circuits. Recent advances in the technology of analog memory may enable analog processors to interface directly with analog memories, obviating the need for area–expensive digital/analog conversion circuitry. Despite these hypothetical advantages, the integration of analog elements into a complete neurocomputer system offers challenging problems.

Neurocomputers must also provide support for the fundamental network operation of learning, or modification of synaptic weights. This requirement especially influences the design of the synaptic memory system. Some early generation neurocomputers provide little or no support for weight modification, but practical applications will require a rapid learning ability.

3.3 Conclusion

These, then, are the three key elements of neurocomputer architecture:

- Inter–processor communication
- Neural computation and precision
- Synaptic weight storage and modification

The remaining chapters alternately focus on architectures to support piriform model II and model I. It begins with an analysis of model II executing on a commercial neurocomputer. This general purpose SIMD (Single Instruction, Multiple Data) machine has characteristics similar to the architecture that will be introduced to support execution of the detailed version of the piriform model. The evolution from the commercial neurocomputer to the architecture targeted at model I parallels the transition from first to second-generation neural models. Analyzing the execution of model II on the commercial machine will provide some insight into the execution of model I on the proposed design. Two different implementations of this latter design will be described, and each approach will be analyzed on how it addresses the three issues above.

Chapter 4

Execution of the piriform model on the CNAPs neurocomputer

4.1 The CNAPS Architecture

The CNAPS[1] architecture features a linear array of homogenous processors (figure 4.1) in an SIMD configuration. Each processor, or physical, node (*PN*) receives input data through an eight–bit input bus (IN bus) and provides output through another eight–bit output bus (OUT bus). The single IN bus and OUT bus are common to the entire processor array; broadcast communication over these is the primary mode of inter–processor communication. Each processor is controlled via a single 32–bit global *command* bus (PNCMD bus), sourced by an external sequencer. CNAPS is designed to process a single layer at a time in a feed–forward neural network, with each PN emulating a single cell in each layer. All PNs process IN bus data in parallel, then sequentially transmit this processed output over the OUT bus. OUT bus data can be routed back to the IN bus, where it is treated by the processor array as input to the next layer in the feed–forward network. Execution proceeds layer–by–layer in this fashion through the entire network. The architecture is flexible enough to execute virtually any network configuration.

In addition to feedforward broadcast communication, each PN can communicate with its immediate neighbors via two pairs of local of two–bit inter–PN busses (a total of four

[1]CNAPS stands for Connected Network of Adaptive ProcessorS and is a trademark of Adaptive Solutions, Inc.

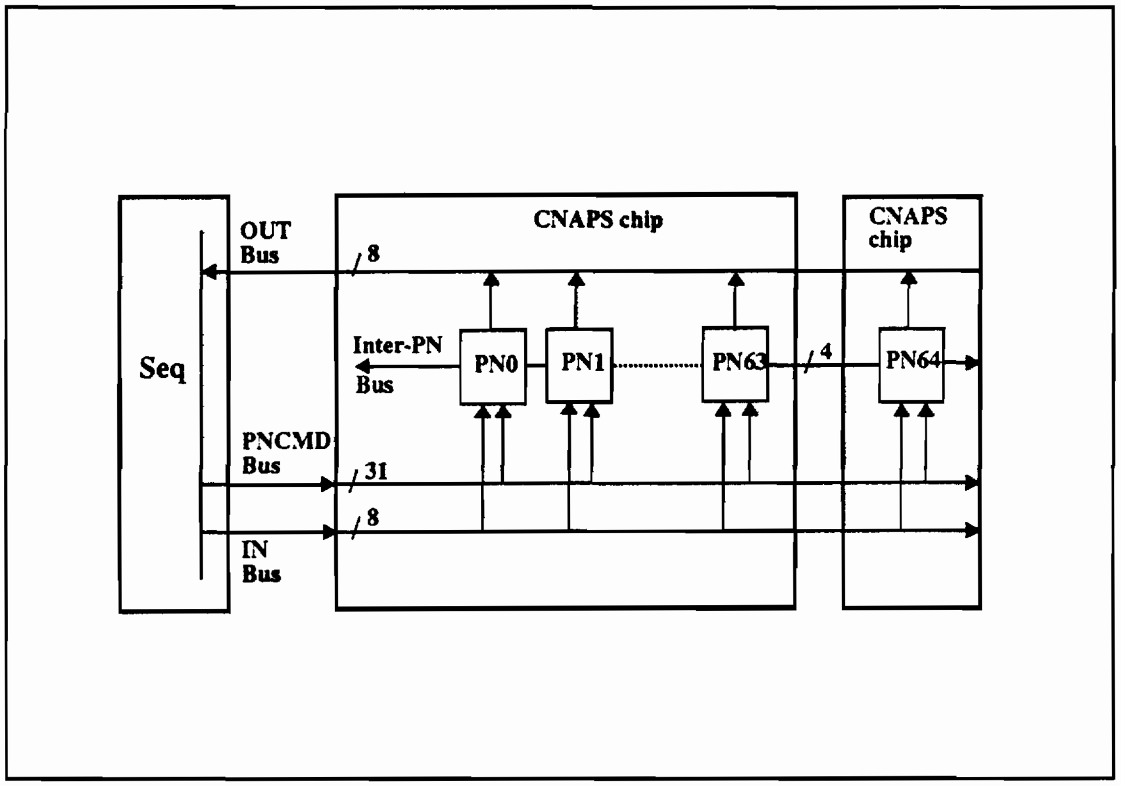


Figure 4.1: Top–level view of the CNAPS architecture. All processors are controlled by a single instruction stream on the 32-bit PNCMD bus (31 bits are shown here; one bit is reserved for future expansion). A single input and a single output bus serve the entire array. This figure shows portions of two chips containing 64 processors each, although chip-to-chip boundaries are transparent to the programmer.

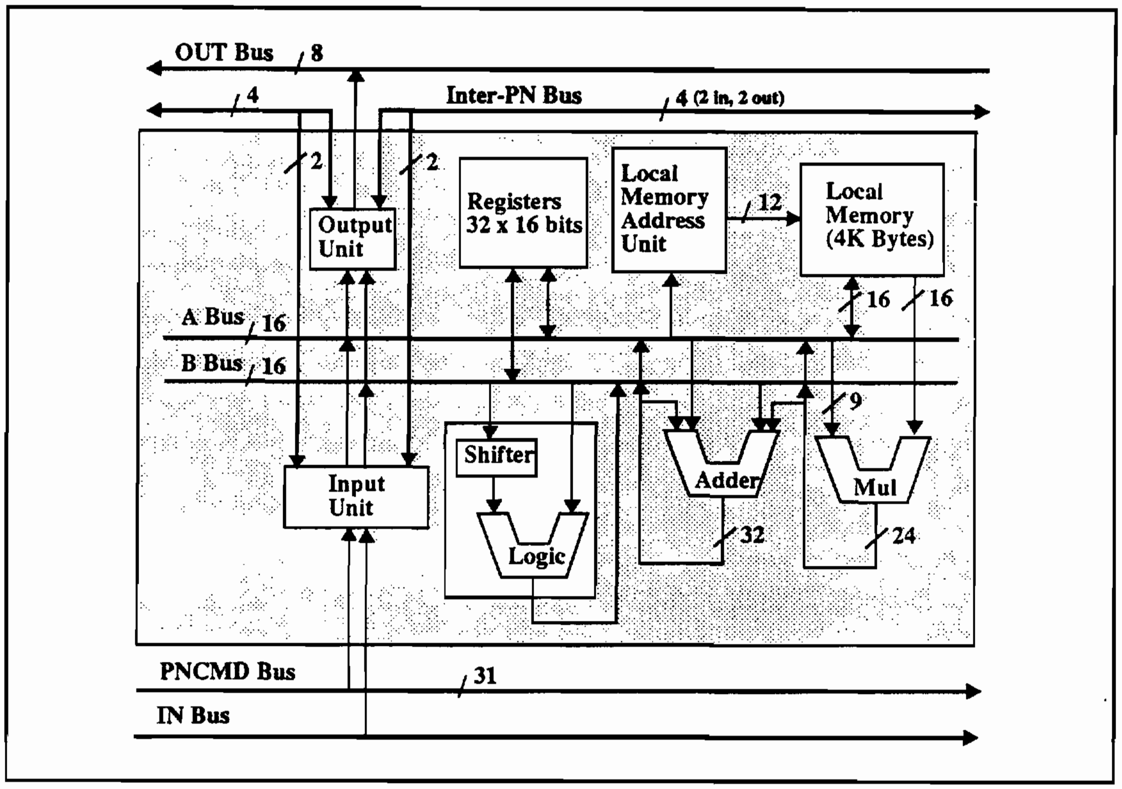


Figure 4.2: Organization of a single PN. A pair of 16–bit internal busses connect the eight functional units.

bits to each neighbor). This feature allows efficient implementation of nearest–neighbor communication and winner–take–all competition; the latter is a useful feature in the implementation of the piriform model.

4.1.1 Processor Organization

Every PN in the CNAPS array is a 16–bit fixed point signal processor. A PN is organized around eight functional units and two 16–bit internal busses (Abus, Bbus). The units are briefly described below. Refer to figure 4.2 for overall organization of a PN.

Input Unit – places data from IN bus and Inter PN bus onto the A and B busses.

Output Unit – places data onto OUT bus and Inter PN bus. 8–bit operands go onto the OUT bus in a single cycle; 16–bit operands require two cycles.

Shifter/Logic – performs shifting and logical operations on 16–bit data. Single–cycle operation.

Adder – adds two 16–bit operands in a single cycle, or two 32–bit operands in two cycles. Addition is two's complement.

Mul – performs 8x16, 9x16 (single cycle), or 16x16 (two cycle) two's complement multiplication. A dedicated bus directly connects multiplier output to an adder input, enabling fast pipelined multiply/adds.

Registers – single–ported file of 32 16–bit registers.

Local Memory – 4K bytes of weight memory.

Local Memory Address Unit – weight memory address unit: includes dedicated 12–bit adder for fast memory address computation.

The arithmetic units allow complex signal processing operations to execute in parallel across the entire array. Each processor's 4 kilobytes of local memory permit rapid access to a large number of synaptic weights. Broadcast communication over the OUT bus to the IN bus is the primary medium of inter–processor communication. Data placed on the OUT bus by one processor can be read by every other processor on the IN bus one clock cycle later. In this way, n^2 inter–processor connections can be completed in n clock cycles; this provides an efficient solution to the connectivity problem encountered in large neural networks.

Adaptive Solutions is building a neurocomputer system based on the CNAPS architecture that features a linear array of 256 processors, permitting rapid execution of networks with up to 256 cells per layer. CNAPS is an all–digital architecture that permits the programming of an unlimited number of network models. Programming is done using Adaptive Solutions' proprietary CNAPS Programming Language (CPL), a hybrid of assembly language and microprogramming. Like all low–level languages, programming at the CPL level offers superior performance at the cost of increased complexity; the programmer can explicitly control all processing units.

4.1.2 Mapping the piriform model onto CNAPS

In CNAPS, a single layer is processed by first distributing the layer's neurons (or *cells*) across the array of PNs. Each PN processes its cell in parallel with all other PNs in the array. Maximum performance is attained when every PN has a cell mapped to it for every network layer; this mapping is specified by the CNAPS programmer. When a cell is mapped to a PN, all of its synaptic weights are stored as a vector in the PN's weight memory. The number of cells that can be mapped to a single PN is limited by the size of PN memory, and this ultimately limits the number of network layers that CNAPS can support.

Layer dimensions in piriform model II can vary between layers, meaning that the number of PNs executing code will vary with the index of the current layer. Similarly, each PN's memory organization depends on the number of cells assigned to it. It is the programmer's responsibility to construct the data structures in each PN's weight memory and to control the range of the processors that execute for each layer. Figure 4.3 shows an example of mapping cells to processors. This two–dimensional portrait of memory organization across the array of PNs illustrates how successive layers map onto processors, and how each cells' synaptic weight vectors are mapped into each PN's weight memory. In the figure, the processor array index is shown on the horizontal (x) dimension, and the linear address of processor memory is shown on the vertical (y) dimension. The figure shows a six–layer network with a variable number of cells per layer. The only PNs that execute are those corresponding to the width of the bar representing each layer. For example, only PNs with indices from 0 to 119 execute during the processing of layer 1; PNs 120-255 are idle during this time. During layer 2 processing, only PNs numbering from 121 to 240 are active. Layer 3 through 6 are larger layers, and correspondingly more PNs execute during the successive processing of those layers. The degree of parallelism, and therefore overall performance, is determined by the number of active vs. idle processors in every layer.

Memory mapping is also illustrated in figure 4.3. A weight vector must be stored

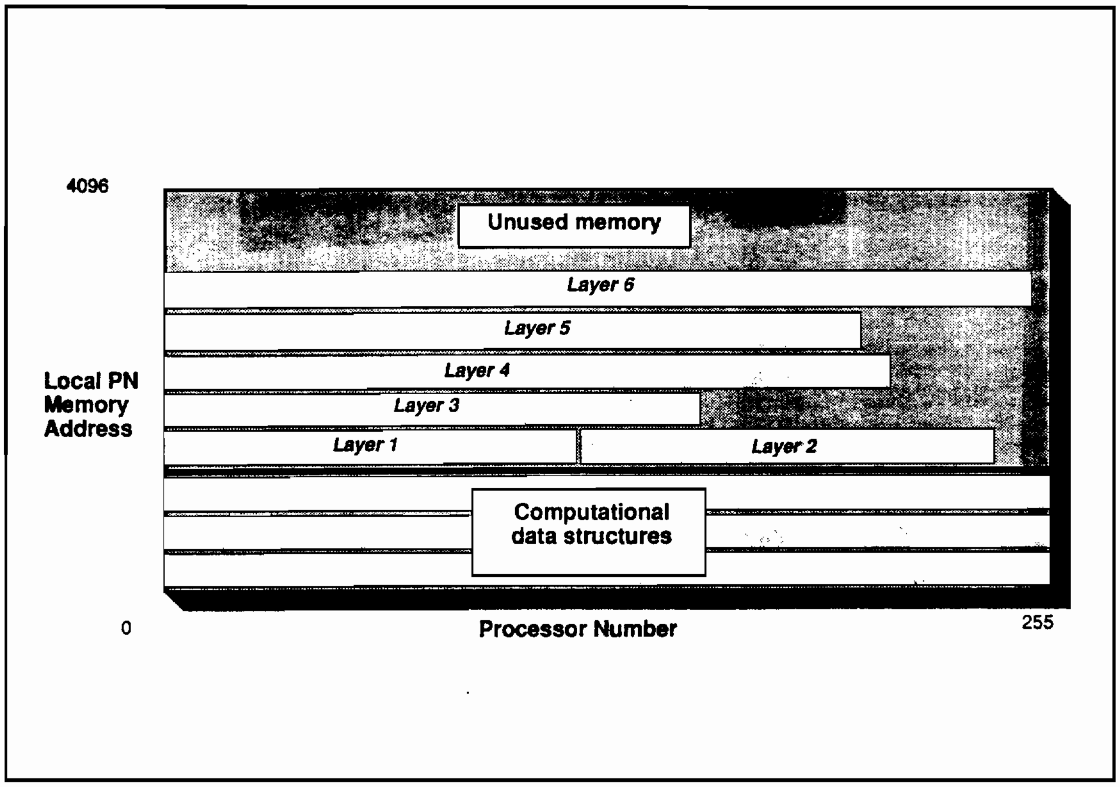


Figure 4.3: An example of mapping a piriform network onto the physical array of processors in the CNAPS architecture. The horizontal axis represents the index of the linear array of PNs, ranging from 0 to 255; the vertical axis represents each PN's memory array, from 0 to 4K bytes.

for every cell mapped to a PN. Note that because low–numbered PNs execute in almost every layer, more of their memory is utilized than high–numbered PNs. Note also that the total number of cells in layers 1 and 2 is less than the width of the processor array size of 256. In this case, weight memory storage is spread across a larger portion of the array.

The task of mapping each unique piriform network to the processor array is a burden to a human programmer, and is best performed automatically. A program named *Mapper* was written to accomplish this. Mapper has three functions: to assign each network layer to a subset of PNs, to separately configure each PN's weight memory, and to create data structures in each PN with pointers to its weight vectors and computational data structures. All PNs use these data structures in network computations. The input to Mapper is a file describing the number of layers in the network, the size of each layer, and the dimension of the input vector (the LOT). The program then performs a straightforward assignment of network cells to PNs. Three constraints define this mapping: (1) layers cannot exceed 256 cells in size; (2) layers must be mapped to an unbroken string of consecutive PNs (i.e., no wrap–arounds from PN255 to PN0); (3) no PN can be assigned cells that would force it to exceed its weight vector storage capacity. Mapper allows the CNAPS programmer to conceptualize network operation in terms of *layers*, and to freely vary network parameters without requiring manual program modification.

4.1.3 The Piriform Algorithm on CNAPS

Allocating cells and synaptic weight vectors to processors and memory is essentially a geometrical exercise. A different sort of challenge is that of coercing the serial elements of the piriform algorithm into parallel execution. The following sequence describes the inner loop operations that dominate execution time of the model:

```
for every input vector
  for every subnet
    (1) present input to subnet, calculate Euclidean distances
    (2) select subnet winner, identify winner to system
    (3) subtract winner's wt. vector from the modulated input vector
    (4) renormalize modulated input vector
```

When the network is in learning mode, the winner's synaptic weight vector is also modified in (3). The weight vector closest to the modulated input vector, determined by the Euclidean distance calculated in (1), will be chosen as the winner in (2). The Euclidean distance calculation is performed simultaneously by every PN. (2) is also performed in parallel using CNAPS' patented "max" operation, where every PN compares its computed distance to every other PN in a bit-serial manner, beginning with the most and ending with the least significant bit. Performance is high during these operations because of the parallel activity and because the synaptic weight data needed by each processor is locally stored in that processor's own weight memory. Each PN can access a 16-bit operand from memory on every cycle; across the entire array, memory bandwidth exceeds 12 Gbytes/sec.

Loop operation (3), however, requires that synaptic weight data stored locally on the winning PN be communicated across the entire processor array, because the modulated input vector target is globally distributed. Transmission over the 8-bit OUT bus requires $2n$ machine cycles to communicate n 16-bit data elements. Alternative implementations could greatly increase the parallelism of this operation, but at the cost of using proportionately greater weight memory. As it turned out this operation was not one of the primary consumers of execution cycle time (using approximately 10% of the time required per layer), so the effort required to speed it up was not justified.

Step (4), LOT renormalization, was added to the algorithm for those applications that required all of CNAPS' 16 bits of precision. Experiments using model II as a classifier of spectral speech required the full 16-bit dynamic range. Without renormalization, the feedback mechanism (a simple subtraction of the winning cell's weight vector from the modulated LOT vector) reduces the magnitude of the LOT vector such that greater than 16 bits are required for accurate resolution. N–element vectors were renormalized by mapping them onto the n-1 dimensional surface of a sphere of unit magnitude. The mapping function entailed the loss of one dimension, but for large dimensional vectors this loss was felt justified by the simplicity of the renormalization scheme. Modulated input vector elements x_i for layer $k+1$ were renormalized by:

$$x_i(k+1) = \frac{x_i(k)}{\sqrt{\sum_{i=1}^{n} x_i^2(k)}}$$

A serial successive–approximation technique was used for calculation of the square root. This proved to be a relatively expensive operation for large networks.

4.2 Model Performance

The test bed selected for model II classification experiments used spectral data from human speech. Speech classification was chosen because of the challenging nature of the task, and because of the proximity of available expertise and data (an active speech recognition research effort is underway at the Oregon Graduate Institute). Experiments for speech classification were conducted using relatively small networks for classifying small data sets [LRW91]. Large scale classifiers for real time, real speech applications will doubtless consist of larger networks, and the example networks studied here are scaled to sizes that attempt to approximate the requirements of industrial strength speech processing. The dimensions of input vectors in this study were taken from experiments using smaller networks: each input vector consists of spectral components taken from a discrete Fourier transform, computed every 3 milliseconds, using a 10 millisecond time window. The first example studied was a five–layer network with 120, 140, 160, 180, and 200 cells per layer, respectively. Total cell count is 800, and requires over 25,000 connections. Such a network can classify a 32–element input vector (representing 32 spectral coefficients) in approximately 9,500 machine cycles. At CNAPS' rated speed of 25 Mhz, 9,500 cycles require less than 400 microseconds. Classification thus would require only 12% of a typical 3 millisecond sampling period.

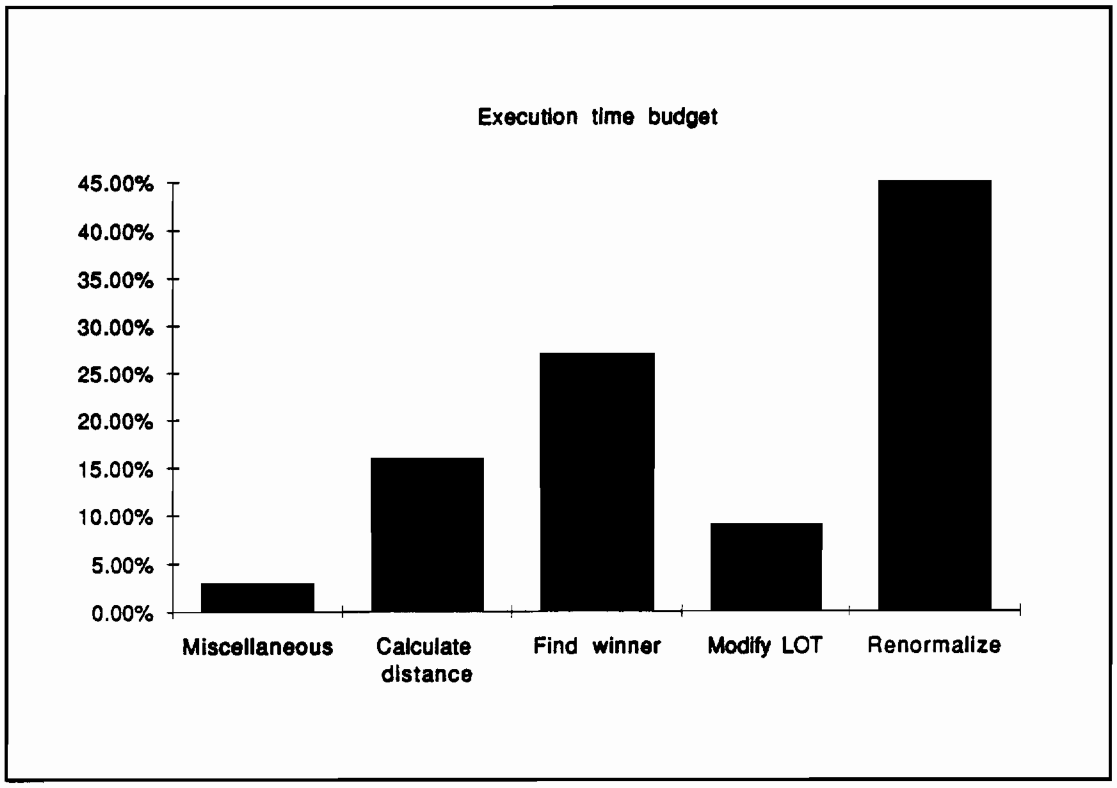


Figure 4.4: Single–layer execution time distribution. The example is for a 160–cell layer, with LOT dimension (N) of 32. The network spends 45% of its processing time performing renormalization of the LOT input vector.

Model execution largely consists of loops. Each input vector is processed by each subnet in sequence. The time for each subnet to process an input vector therefore forms the critical execution path. Figure 4.4 shows a breakdown of execution time during this phase.

Renormalization of the modified input vector is the most expensive operation, requiring 45% of the time required to process a single layer. The next three most lengthy operations in each layer are finding the winner (27%), calculating the distance from each cell's weight vector to the input (16%), and modifying the input vector (9%). Miscellaneous layer operations take up the remaining 3% of processing time. Different network dimensions will change these proportions to some degree. Execution time for distance calculation scales linearly with input dimension, but is independent of layer size. The time required to find the winner scales sub–linearly, and vector modification time scales linearly, with layer size. The renormalization algorithm requires a constant number of machine cycles, independent of vector length, so its relative share of execution time shrinks as network size grows.

One way to reveal the serial and parallel nature of the algorithm is to estimate the *speedup*, the relative reduction in execution time, of network execution on CNAPS compared to execution on an single idealized processor. The processor model used for comparative purposes was a single PN executing with unlimited amounts of weight memory. For speedup estimation purposes a PN was used because each PN is similar to commercial fixed–point signal processors. Figure 4.5 graphs the estimated speedup for five example networks using a varying–length input vector. Each example network consisted of five layers, with the same number of cells in each layer. The smallest network had 50 cells per layer, the largest 250. The LOT input dimension was varied from 4 elements to 128. These network dimensions are realistic estimates of the requirements of large–scale spectral classifiers.

Figure 4.5 clearly shows that speedup increases with layer size. This is not surprising, because larger layers permit correspondingly larger degrees of parallelism. Note, however, that speedup lags the increase in layer size; e.g., 100 processors buy a speedup of between 3 and 30, not 100. Coordinating and communicating between multiple processors requires processing overhead that a single–processor implementation can ignore. Broadcast communication and SIMD control are efficient mechanisms, but they do have a cost. This overhead is especially evident in networks with small input vector dimensions; note that when this dimension is 4, 250 processors only buy a speedup of 10 over the uniprocessor implementation. With larger input dimensions, the control and communication overhead is amortized over more processors and cycles.

An important difference between model I and model II is that the latter dispenses with sparse connectivity and instead fully connects adjacent network layers. CNAPS is well–suited to such networks, because full connectivity permits the maximum utilization

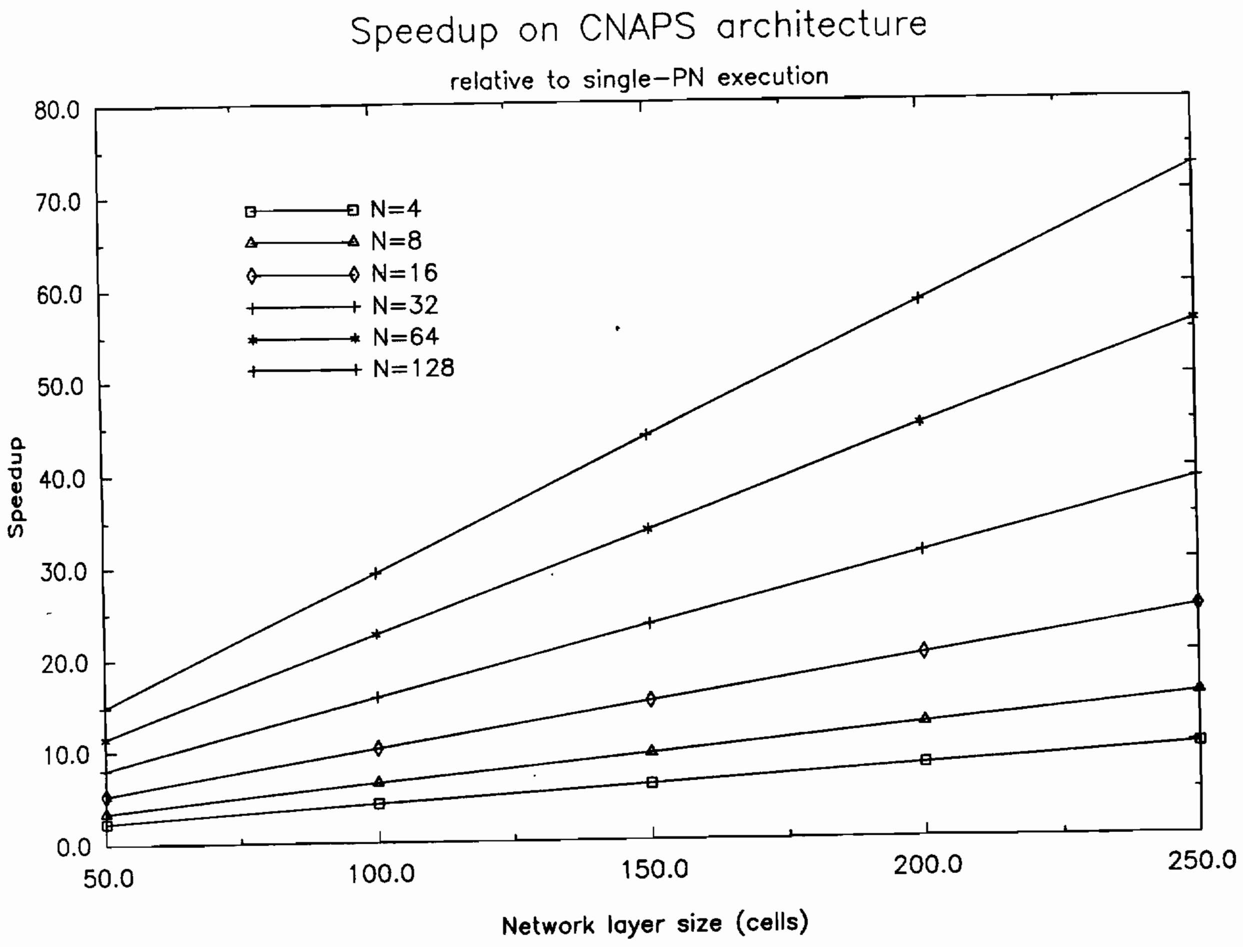


Figure 4.5: **Plots of CNAPS speedup over idealized single–processor execution. LOT input size (N) was varied from 4 to 128, and used as input to five different networks. Each network was five layers deep, and layer size varied from 50 to 250 cells per layer.**

of the processor array. More biologically realistic networks featuring sparse connectivity will pose problems for architectures like CNAPS. Architectures that handle the sparse connectivity of model I are addressed in the chapters that follow.

4.3 Conclusion

For networks with high–dimensional inputs and many large layers, the most computationally expensive elements of the simplified piriform algorithm are the calculation of each subnet's Euclidean distance and determination of a winner. CNAPS's 256 parallel processors, high local memory bandwidth, and hardware–assisted maximization technique significantly reduce the execution time required for these operations. However, those portions of the piriform algorithm that require serial execution or global communication of locally–stored data remain execution bottlenecks. This is reflected in overall network performance, because while large networks show significant speedup, small networks do not exhibit enough parallel activity to offset the performance penalty of the serial portions. This is simply an example of "Amdahl's Law" [HP90a] in action. Amdahl's Law effectively limits the benefits of parallelizing code because serial code (that resists parallel execution) eventually dominates overall execution time. In other words, the incremental benefits accrued from speeding up one portion of an algorithm lessen with successive improvements. On CNAPS, the traditional computational bottlenecks of vector quantization have been addressed via SIMD parallelism, but the serial tasks remain.

Renormalization was required in this study because piriform model II required 16 bits of precision in speech recognition applications. Successive feedback to the LOT input vector reduces its resolution to less than this. The renormalization algorithm described in section 4.2 executes serially on a single processor, which is why it is the slowest part of the computation. The basic successive–approximation algorithm can be parallelized, however, greatly reducing the computational penalty. The easiest change to make would be to compute blocks of bits simultaneously, using the otherwise idle processors. Each of 256 PNs could simultaneously compute a different eight-bit value, and compare it to a target. That PN matching the target would broadcast its value to the entire array, which would then use it as the starting point for computing the next eight-bit value. A renormalized 16-bit value could be computed in this way in approximately one-fifth the number of cycles required by a serial approach.

Renormalization is important in piriform model I also, but it is implemented differently. One of the many functions of the olfactory bulb is to perform automatic gain control (AGC) on olfactory input. AGC is a form of renormalization, and it is presumed to operate on every sniff [GAISL89]. The AGC function of the bulb is performed in other brain regions that operate primarily on sensory input, such as the retina and cochlea [Mea89]. Renormalization is only required in model II because this simpler model has no sensory pre–processor. Any complete system based on piriform model I will require a bulb-like pre-processor for implementing AGC; for that reason piriform model I has no renormalization mechanism.

The exercise of implementing piriform model II on CNAPS has illustrated the benefits of SIMD architectures for data parallel applications. The remainder of this thesis focuses on piriform model I and a proposed neurocomputer architecture targeted for it called Super Sniff (or *SS*). SS retains the SIMD flavor of CNAPS but is described in sufficiently high level terms to permit numerous alternative implementations of the architecture. Following a brief description of the SS architecture, two designs for its implementation are presented and compared on a cost/performance basis.

Chapter 5

The SS Architecture

This brief chapter begins with an analysis of the inherent parallel and serial characteristics of the piriform model and ends by proposing an architecture that borrows ideas from CNAPS, yet is optimized for the key features of networks like model I.

5.1 Parallelism in the Piriform Model

A key difference between the two versions of the piriform model is the broader scope of parallel activity in model I. Layers in model II must execute sequentially because the LOT is modified by feedback from each previous layer. This strict temporal segmentation of activity is an abstraction from the biological model (model I), and it prevents taking advantage of significant parallelism occurring in model I.

Shepherd reports that layer II is characterized by a *wave of activity* proceeding from the rostral to the caudal regions of the network [She79]. LOT transmission times from the olfactory bulb and rostrally-located layer II cells largely account for these delays; as stated in chapter 2, there is evidence that each layer II cell sees its afferent LOT input and associative recurrent collaterals simultaneously, albeit delayed in time from cells elsewhere in layer II. Inputs to rostral neurons are the first to arrive; delays in forming associative signals account for subsequent delays to caudal cells.

Model I can be described, then, by a combination of serial and parallel processing. Rostrally located patches process their inputs first. All cells in a given patch receive their inputs more or less simultaneously. Cells within each patch compete (in parallel) to be the first to cross an activation threshold. Active axons from winning cells rejoin and propagate down the LOT, and caudally located cells combine these signals with the original LOT input from the bulb in their own intra-patch competition. This LOT delay time can be used to advantage by an artificial piriform processor. Signals representing afferent input from the bulb can be computed by every patch simultaneously, upon arrival, and later combined with the delayed recurrent collaterals to compute overall patch activation.

5.2 The SS Architecture

The architecture proposed here is called SS, for *Super Sniff* and its key feature is a linear array of processors where entire *patches* rather than simple *cells* are mapped to individual processors. Patches are arranged in a manner analogous to the layout of piriform layer II: a linear array receives input from and broadcasts output to a single data bus, the LOT bus. Control of each processor is via a global command bus, akin to the PNCMD bus in CNAPS. Figure 5.1 provides a global view of the SS architecture. In the following sections the salient characteristics of the architecture are presented and briefly described.

5.2.1 Connectivity

The LOT bus emulates the lateral olfactory tract in the piriform model. All distal connectivity between neurons (i.e., processors) is made through this medium. Local connectivity is implicit in the mapping of patch cells to processors. The purpose of local connectivity in this model is to enforce the winner-take-all paradigm, and that function is implemented by the patch operating rules rather than explicitly through connections between neurons in the same patch (i.e., each processor permits only the most strongly activated cell to win the patch competition).

Connections in model I indicate only the presence or absence of activity in the source neuron, not the strength of that activity. Data on the LOT bus is therefore binary

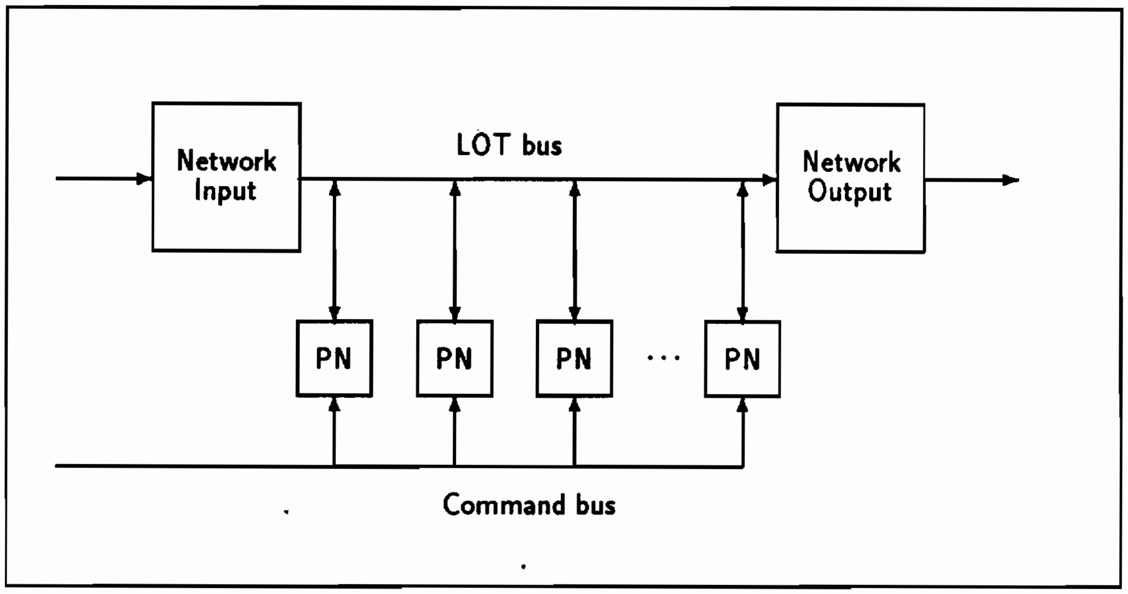


Figure 5.1: Overview of the linear array of processors in the SS architecture. Inter–processor communication is via the LOT bus, and all processor control is over the command bus.

valued. Greater input precision can be provided by using multiple LOT lines to encode a single analog–valued input via a "thermometer scale". In a thermometer scale, a single analog–valued input is encoded as the activation of binary–valued signals whose number is linearly proportional to the magnitude of the analog input.

The SS architecture does not specify how connectivity is encoded on the LOT bus. In the two implementations to be studied, one uses a physical wire for every connection in the network, while the other encodes this information as a binary address. The only requirement is that all connectivity information is completely contained within the LOT, so that a PN can monitor all network communication by watching the LOT bus.

5.2.2 Neural computation

Each processor, or PN, performs the arithmetic operations for all cells in a patch. The traditional scalar product computation reduces to simple repeated additions in model I due to the binary–valued character of LOT signals, so there is no need for the arithmetic operation of multiplication. Numerous cells in each patch can be simultaneously active, so each PN must be capable of supporting parallel internal operations on different cells. In addition, PNs must support the bi-modal operation of model I cells: they must be capable of operating in either learning or performance modes.

Each PN in the SS architecture must continually monitor all signals on the LOT bus. When active inputs from LOT lines connected to any cell in that patch are detected, the active input is communicated to the relevant patch cell(s) and cell activation is modified according to the strength of the synaptic connection. SS permits multiple encodings of connectivity information and different computation schemes, including both digital and analog. Each processor must determine which of its cells, if any, is the winner of the patch competition, and must communicate this information back to the LOT bus and to the rest of the network.

5.2.3 Weight storage & modification

A large part of each processor's task is to store the synaptic weights of every cell in the patch. All weights must be modifiable in learning mode, and control over modification must be controlled locally. That is, each PN must have independent control over its weight changes, and cannot require the intervention of a central controller. The SS architecture does not define the storage mechanism. In the two implementations of the architecture to be described, two radically different storage techniques are proposed.

5.3 Summary

The SS architecture owes much of its heritage to CNAPS: the single input data path, its SIMD organization, the linear array of processors. It is a natural successor to CNAPS because it supports parallel execution of neuron *assemblies*, not simply individual neurons. SS is consistent with the evolution of second–generation neural models, which describe a higher level of cortical organization than models in wide use today. This architecture is described in sufficiently abstract terms to consider widely different implementation options. Two such options are presented in the following chapters.

These two options exemplify the two different approaches to neural network hardware design described in chapter 1. The first implementation features analog processing in a manner reminiscent of biological neurons. The second does not attempt to emulate analog neurons but utilizes simple DSP–like processors to carry out their operations. Each method has its strengths and weaknesses, as the following chapters will illustrate.

Chapter 6

Analog Implementation of the SS Architecture

6.1 The Analog Appeal

Research in analog circuit design for neural networks has been motivated primarily because analog "neurons" can be fabricated using a few standard VLSI components. A full blown signal processor would be required to implement equivalent neural behavior in a digital circuit. There has also been in the research community a widespread assumption that because biological neural networks utilize analog hardware, their artificial counterparts should do the same. An important theme of this thesis, however, is that analog artificial neural networks have serious practical engineering problems to overcome, and because of this they will offer only peripheral utility for some time to come. In addition, analog computation has characteristics unsuitable to this computational model.

This chapter will begin with a brief review of proposed analog processor designs that are potential candidates for *SSA*, an analog implementation of the SS architecture. The focus will then be narrowed to a single approach, and an implementation will be proposed. This will be followed by a summary of the advantages and disadvantages of this and other analog neurocomputer systems.

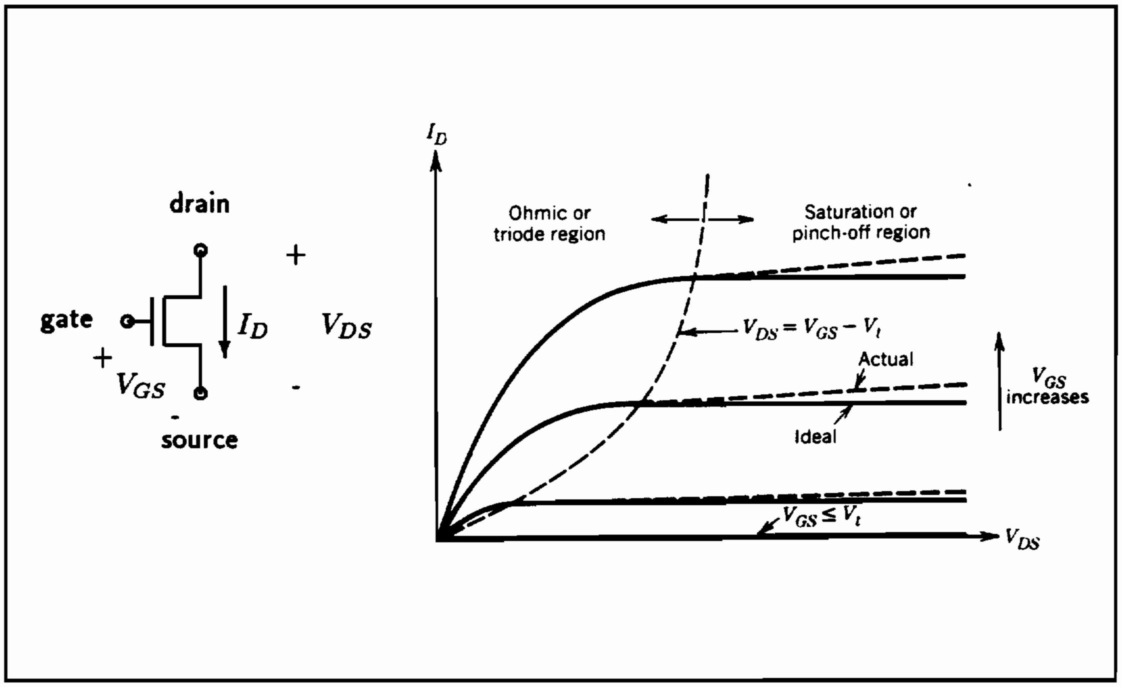


Figure 6.1: **Analog behavior of a CMOS n–channel transistor in strong inversion. (left)** V_{GS} **is applied to the gate, which modulates the current** I_D **through the device. (right) V–I curves for a typical device, shown for a variety of applied gate voltages. From Gray & Meyer,** *Analysis and Design of Analog Integrated Circuits,* **1984, pg. 67.**

6.2 Analog Processor Designs

The goal of analog neural network design is to build every synapse out of small analog circuits. Ideally, the synaptic circuits would be constructed from very few devices, perhaps even a single transistor. The analog behavior of a CMOS transistor is shown in figure 6.1. Current through the device I_D is modulated by the applied gate voltage V_{GS} and the voltage across the device V_{DS}. As figure 6.1 shows, for most of its operating regime the device current and the modulating input voltage have a decidedly non–linear relationship that bears little resemblance to models of synaptic operation proposed in the literature. The current–voltage relationship shown in fig 6.1 is for devices in strong inversion, where nearly all modern MOS circuits are designed to operate. Recently, researchers have designed MOS circuits that operate in the sub–threshold region, where currents are reduced by orders of magnitude and device response times approximate

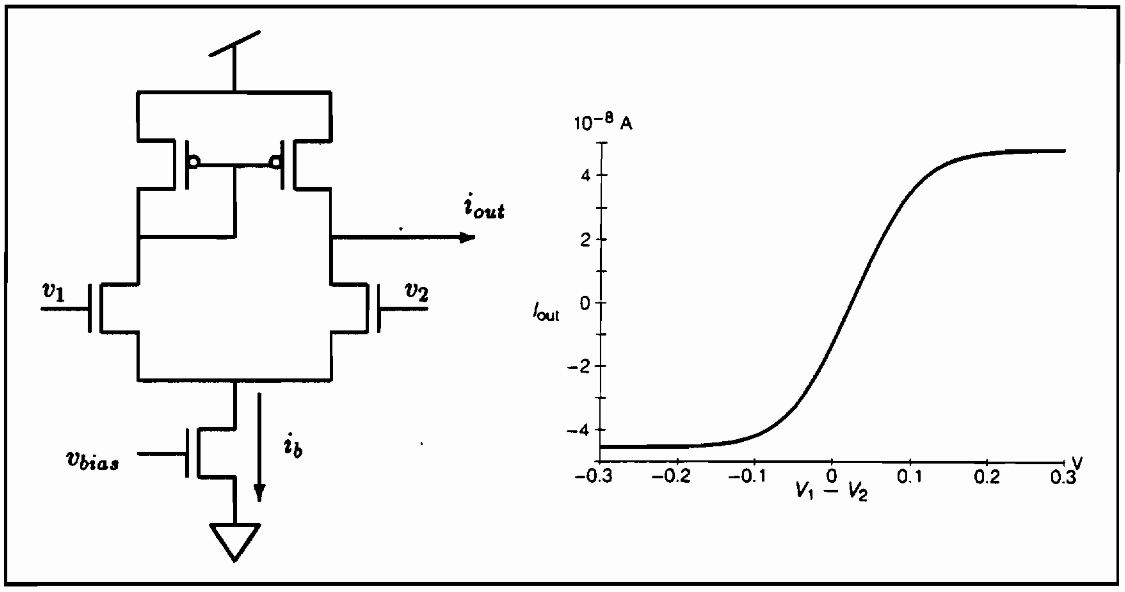


Figure 6.2: (left) Schematic diagram of the basic transconductance amplifier. (right) Typical output current of this circuit, which can be interpreted as performing a two–quadrant multiplication between the bias current i_b and the difference between two input voltages v_1 and v_2. From Mead, *Analog VLSI and Neural Systems,* 1989, pg. 70-71.

those of biological neurons [Mea89]. Such techniques have considerable potential, but until designers have a better handle on sub–threshold operation, their use will remain limited.

Analog circuits are routinely designed with a linear output, which is more suitable for most neural network models. Figure 6.2 shows a schematic diagram of a *transconductance amplifier*, which produces an output current proportional to the product of a bias current and the difference between two applied voltages. Numerous neural network research programs have produced analog processor designs containing derivatives of this basic circuit. While considerably larger than a single transistor, this circuit performs the desired operation of two–quadrant multiplication, so named for the quadrants of the Cartesian plane where it is defined. More sophisticated circuits employing multiple transconductance amplifiers achieve four–quadrant multiplication, where both inputs can take on positive and negative values [GM84] [Mea89]. Such circuits typically require three to four times as many devices as the simple transconductance amplifier shown in figure 6.2. An analog neurocomputer could require one such circuit at every synapse in the network.

Many neural models require linear multiplication over a wide range of input values. Simple analog multipliers such as this one restrict inputs to a narrow range of values. It also suffers from a systematic offset produced when both inputs are at zero, an artifact of the imprecise device matching that is inherent in CMOS processing. These and other deviations from ideal behavior combine to limit simple analog multipliers such as this one to approximately four to eight bits of precision, depending on the circuit complexity. [HTCB89].

Piriform model I differs from most neural models in that pure multiplication is not required, considerably simplifying the design of a synapse. LOT signals are binary rather than real valued, indicating whether each source is active or inactive. Rather than computing a scalar product between the LOT input and each weight vector, each processor need only sum the values of synaptic weights at sites with active inputs. *Addition* thus becomes the critical arithmetic operation. Analog addition is easier using currents rather than voltages, because multiple inputs can be summed using a simple wire. Figure 6.3 illustrates one such technique.

The biggest problem with the design shown in figure 6.3 is the accurate storage and generation of the synaptic weight values. *Charge* storage is the form most commonly employed in a CMOS circuit, either on a capacitive node or on the floating gate of an active device. The weight in its stored form must be converted to current prior to summation. Conversions are usually expensive in terms of silicon real estate and should be avoided whenever possible, but two common techniques for converting a stored charge into a current are shown in figure 6.4. The precision of the digital–to–analog converter (DAC) is, to a first approximation, limited by the number of bits used. Storage density declines with increased precision. Storage as a continuous range of voltages offers greater densities than digital storage, but precision is limited by the poor control of capacitance values attainable with current VLSI processes (capacitor values can vary up to 20% across a chip). Feedback using additional amplifiers can partially compensate for process deviations, but the extensive area required by the compensation circuits eclipses by a

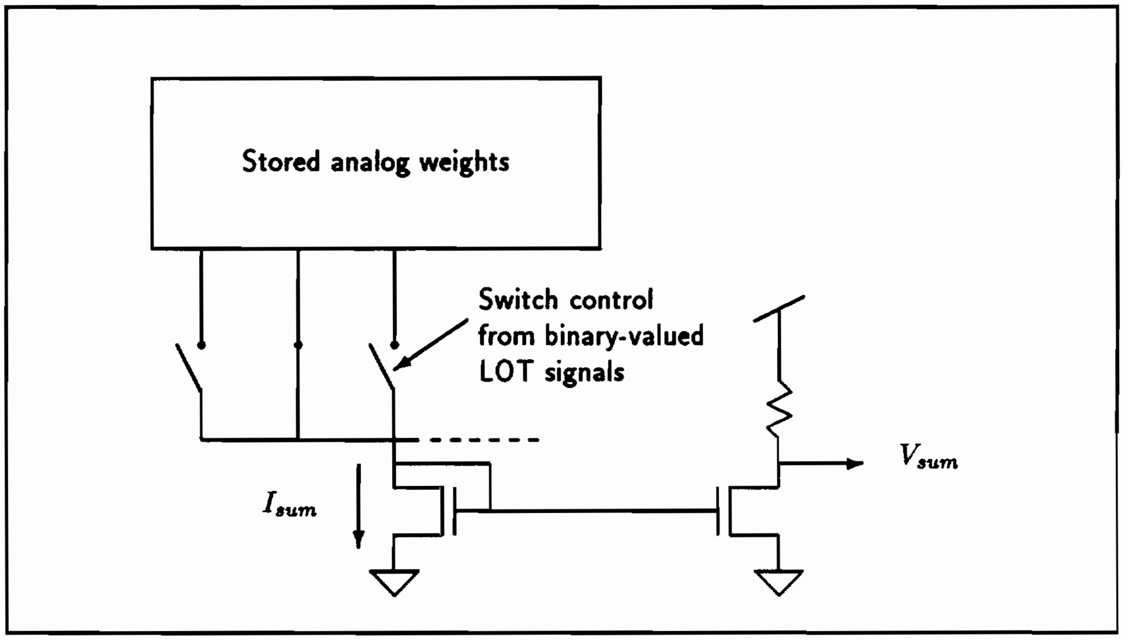


Figure 6.3: One method of performing analog current–mode addition. A single wire sums currents from multiple sources, each representing a synaptic weight. The summed current is mirrored to a second transistor and converted to a voltage. Storage of the analog valued weights presents the primary difficulty with this technique.

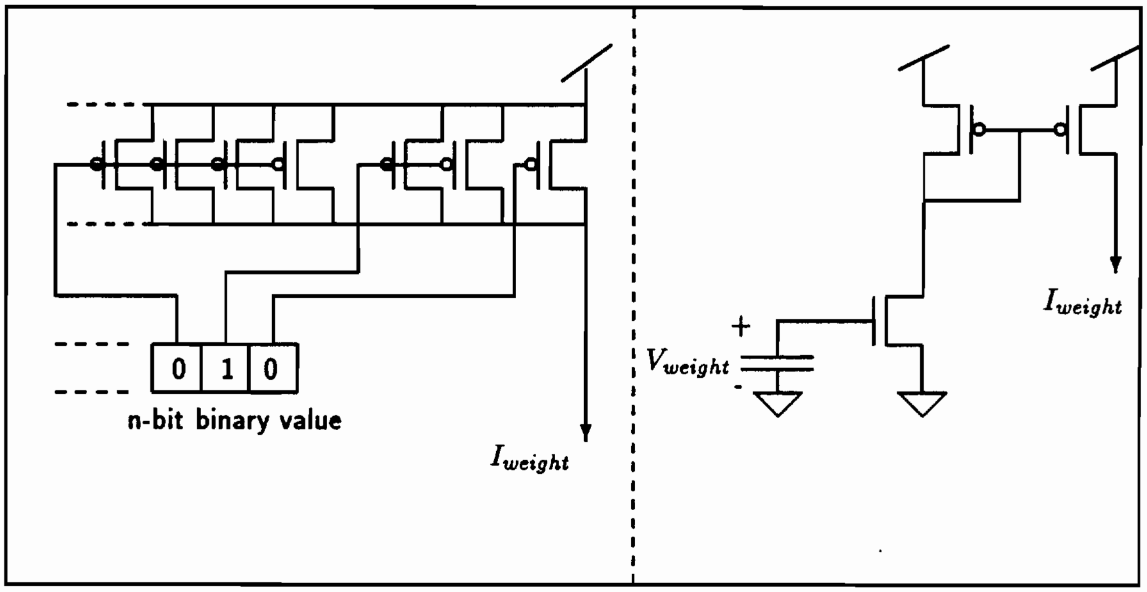


Figure 6.4: Two common techniques for converting a stored analog value for use as an input into a current–mode adder. (left) Digital storage: each stored bit (lsb on right) controls twice as many numbers of a unit current source, a single p-channel transistor, as its predecessor. (right) Analog storage: the stored capacitor voltage forces a current that is mirrored to the adder input.

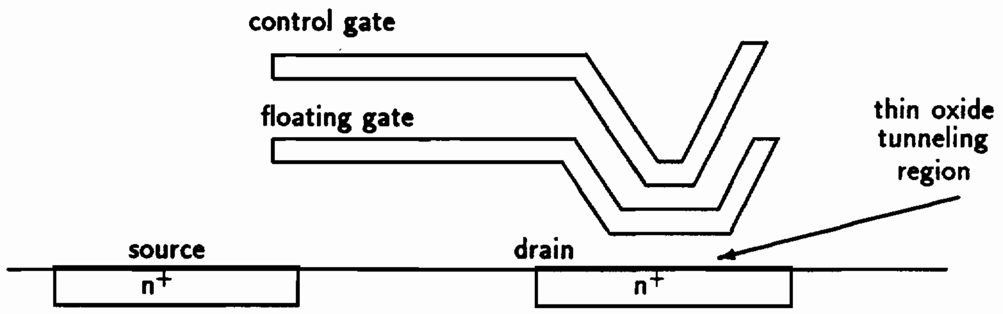


Figure 6.5: Cross–section of an EEPROM transistor. The second, or floating, gate captures charge from the drain (when programming) or releases charge (when erasing). High voltage is applied to the control gate when programming the device, or to the drain when erasing.

wide margin the area required for actual charge storage. Substrate leakage currents will degrade capacitive voltages significantly over time, so additional amplifiers are also required for periodic refresh of the stored values. Analog refresh requires large numbers of relatively large voltage comparator circuits, further adding to the overhead burden.

6.2.1 Analog EEPROM

Analog EEPROM (Electrically Eraseable Programmable Read Only Memory) has been proposed as an alternate form of analog storage particularly suitable for neural network applications [CM89a][HTCB89][OKH89]. In an EEPROM device, the single polysilicon gate of a conventional transistor is augmented with a second layer of polysilicon that is electrically isolated by surrounding layers of oxide. The transistor threshold voltage V_t can be altered by storing charge on this "floating" gate. Figure 6.5 shows a cross–section of a standard EEPROM device. Charge is stored on the floating gate by grounding the source and drain and applying a high voltage to the control gate. The resulting electric field is sufficient to induce electron tunneling from the drain to the floating gate across an extremely thin ($\sim$ 100Å) layer of oxide; given sufficient programming time, enough electrons are trapped on the floating gate to raise its V_t by several volts.

EEPROMs are typically used to store digital values in a non–volatile manner. The amount of charge on the floating gate can be varied to allow the storage of analog information. The amount of charge is a function of both the programming voltage V_{pp} and the programming time; typical values are 20 volts and 10 milliseconds. An analog EEPROM synaptic weight memory would be compatible with the analog adder circuit shown in figure 6.3 because the memory output and the adder input would both be in the form of current. Figure 6.6 shows how weights could be stored in such a scheme. Every LOT input line would function in a role similar to a "word" line in a conventional DRAM. There is one memory column in the array for every cell in a patch. Multiple LOT inputs arrive and cause a "read" of every synaptic site to which they are connected. Each addressed site sources an amount of current programmed by its V_t value onto its cell's summing wire, effectively performing analog addition.

EEPROM programming is usually a two–step process: first erase the old stored value, then store the new one. Erasure entails grounding the control gate while applying V_{pp} (the 20 volt programming voltage) to the drain (via the bit line). In an array composed of single–transistor storage cells as shown in figure 6.6, every storage cell in a column is erased simultaneously, a procedure known as *flash* erase. The control gate and drain voltages are then reversed when storing: the drain is grounded and V_{pp} is applied to the gate (via the LOT, or word, line). Table 6.1 describes the configuration of the EEPROM weight memory inputs during reading, programming, and erasing. Characteristics of piriform model I make a good fit with this programming model, and would simplify programming in normal circumstances.

Recall that piriform weight modification is based on the mechanics of LTP, Long Term Potentiation. Synaptic weights in LTP can only be *increased* in efficacy, meaning that the magnitude of each weight can only increase, up to a ceiling value. Programming each EEPROM synapse thus need not begin by erasing the previous value before storing the new value. A synapse has its weight changed by simply adding more charge onto the floating gate by applying V_{pp} for additional time and grounding the bit line. The all–or–none character of flash erasure (every device in a column is erased when V_{pp} is

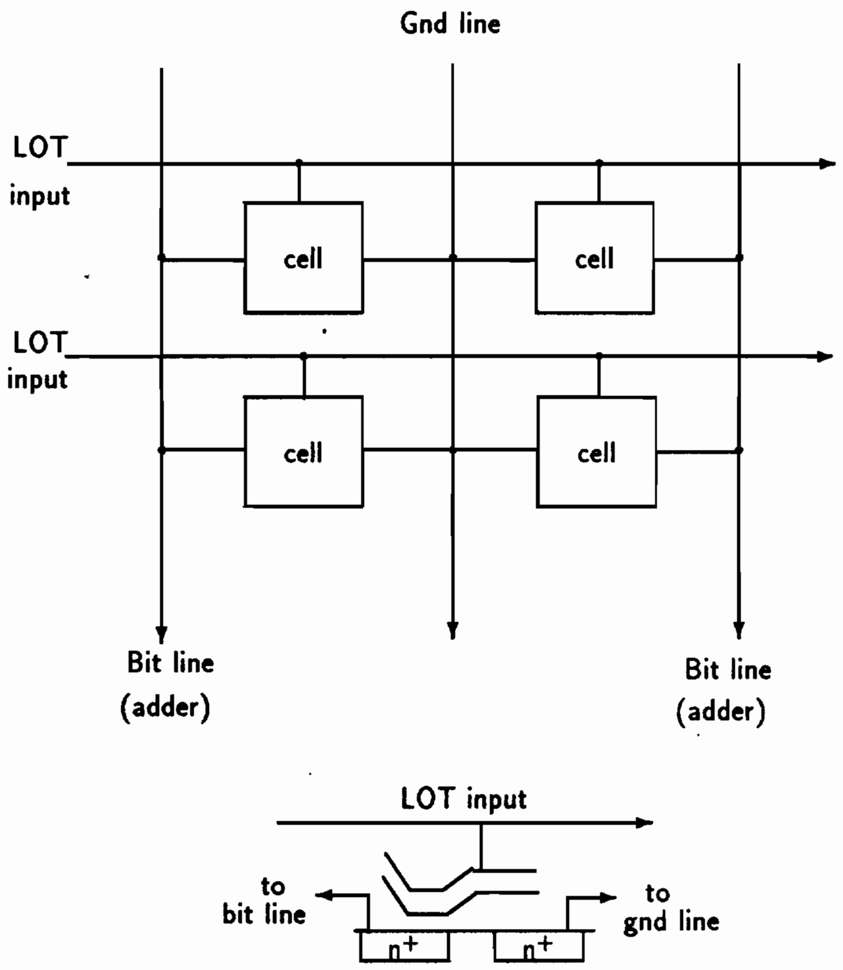


Figure 6.6: An array of EEPROM cells storing synaptic weight memory values; a single memory cell is shown below. Active LOT inputs cause each cell to source its programmed current onto a bit line, which sums the output of every active cell in the column.

memory mode	bit line	word line	gnd line
Programming	if winner V_{ss} else V_{pp}	V_{pp}	V_{ss}
Erasing	V_{pp}	V_{ss}	V_{ss}
Reading	to adder input	V_{read}	V_{dd}

Table 6.1: Configuration of the three terminals of the EEPROM memory array. V_{pp} is the 20 volt programming voltage, V_{dd} is the standard 5 volt power supply, V_{read} is the control gate read voltage, and V_{ss} is ground. V_{read} and V_{pp} are independently generated by each PN.

applied to the drain, or bit line) is therefore not a liability in normal operation, because weight cells are only erased when resetting the entire system.

Ideally, only the devices corresponding to the winning neuron would feel the effect of applying V_{pp} to word lines, because only the winner's weights should be adjusted. Columns corresponding to non–winning cells can be prevented from storing additional charge by also driving their bit lines with V_{pp} during programming. The target column would have its bit line grounded, as per normal programming operation. Programming would occur by varying the programming time rather than by varying the V_{pp} voltage level because incremental timing control is easier than modulating the output of the complex charge pump circuits that generate V_{pp}. Each synapse is modified independently of all others in a patch because the EEPROM programming time is a function of the target V_t level, and this will vary from synapse to synapse.

A proposed analog EEPROM–based implementation of a PN in the SS architecture is shown in figure 6.7. The LOT drives a large weight memory array composed of analog EEPROM devices. An analog adder is assigned to every weight memory column; each performs the additions for one patch cell. The voltage outputs of each adder feed an analog winner–take–all circuit similar to that described in [LRMM89], which selects the most strongly activated patch cell. LOT outputs of all but the winning cell are logical zeros, and the winner's LOT output is logic 1. If the system is in learning mode,

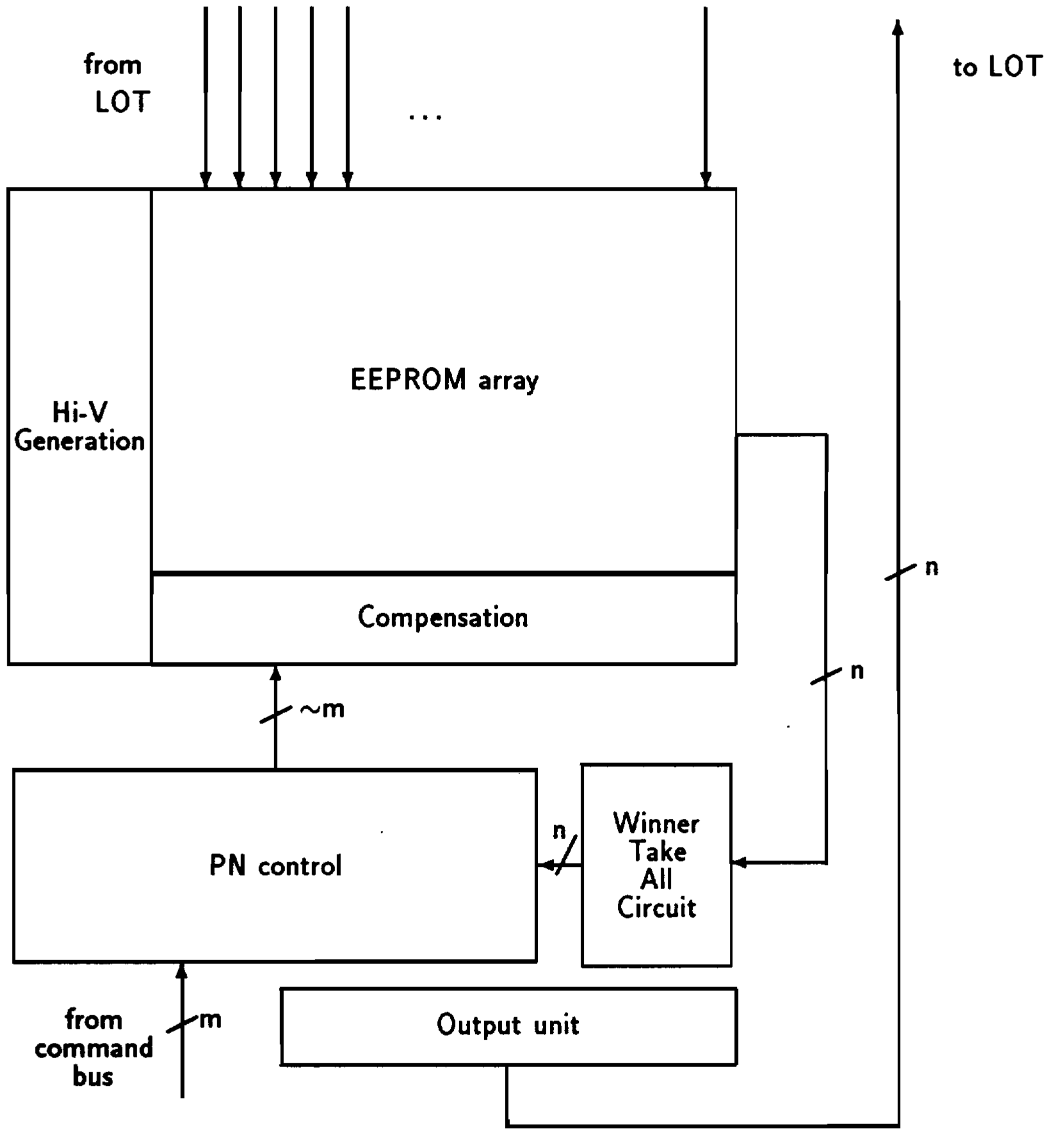


Figure 6.7: Architecture of a proposed analog EEPROM–based PN that processes n neurons. An m-bit-wide bus controls the PN.

the identity of the winning cell is stored and used to select high–voltage inputs in the subsequent programming of the winner's active synaptic weight sites.

6.2.2 Practical Drawbacks of Analog EEPROM

Despite numerous attractive aspects, serious problems prevent this proposed implementation from being a practical proposition. While the most serious deficiencies concern communication between PNs (more will be said on that later), the circuit design issues alone are sufficient to give pause. Some of these challenging issues are:

- Limited adder precision. The factors contributing to this are legion, but the key culprits are imperfectly matched V_t's and non–uniform resistance along each current summation path. The current from each device is also strongly dependent on temperature, so a programmed synaptic weight value may differ from the value that is later read if programming and reading are performed at different temperatures.

- Poor programming control. Each device will have a different "native" V_t, so each will require different amounts of charge to store a desired value. The proposed design contains extensive compensation circuitry to lessen, but not eliminate, this effect. Making many small, incremental changes to V_t is the only practical programming method.

- Non–linear relationship between current and programming time. The storage devices are saturated CMOS transistors, where the relationship between the applied gate voltage V_{gs} and device current I_{ds} is given, to a first order, by:

  $$I_{ds} = k\frac{W}{L}(V_{gs} - V_t)^2$$

  k, W, L are process parameters and device geometries. Adjustments in a device's V_t should result in a constant increase in I_{ds} to be consistent with model I's weight adjustment rule. Note that I_{ds} increases in proportion to the square root of V_t, so all V_t adjustments, and therefore the length of programming time, will vary depending on the target I_{ds} level. Short V_{pp} pulses are required at higher V_t targets.

On the other hand, EEPROM programming is self-limiting, because the charge accumulating on the floating gate counteracts the motive electric field seen by additional electrons, reducing their accumulation rate. It therefore takes longer to deposit the same increment of charge at higher V_t levels than it does at lower levels. These two effects work in opposite ways to simultaneously shorten and lengthen the incremental programming time; balancing their interaction requires a complex control structure in each PN. It is unlikely that such a powerful controller can be economically built within each PN. The single commercial product in existence today using analog EEPROM devices requires an off-chip computer for program control [HTCB89].

- Longevity. Tunneling electrons can be trapped in the oxide between the floating gate and the drain. After repeated programming episodes, enough electrons are trapped to induce a field that counteracts the field of the applied V_{pp}. Eventually the induced field effectively prohibits programming altogether. Extending the lifetime of programming episodes significantly beyond 10^4 to 10^6 programming events, the current limits of commercial EEPROMs [SSK+87], is an area of active research. This is a significant restriction in the lifetime of a neurocomputer that may have to undergo many millions of learning cycles.

Analog EEPROM offers a potential medium for dense analog storage, but the extensive compensation circuitry means that global control (in the best case, at the PN level) is required to achieve reasonable levels of programming precision. Without an effective analog storage mechanism, the promise of analog neural networks will be restricted to applications where continuous learning is not required (even then, analog storage remains a problem). Variance between individual transistors mandates that even in applications requiring only a single learning episode, networks must undergo repeated training cycles as the central learning algorithm "adapts" to the peculiarities of the network. Such requirements severely restrict the range of applications that have been envisioned for neural network technology.

The problem is that the circuits and devices that have historically pushed the development of VLSI technology are inappropriate for analog storage. This has been previously noted by many researchers in the field, among them industry pioneer John Hopfield, who summarizes here the dilemma:

> If there is a single place where the clever exploitation of analog electronic device physics is likely to have a high pay–off, it is in making connections which are continuously adjustable, capable of learning with a local learning rule (perhaps involving a global enable signal), and *not* requiring many transistors for implementation and control. Neurobiology solves this problem by using chemical modifications of ionic conductances, constructing a synapse which is a complex device with history–dependent properties. Equivalent cleverness in the device physics of artificial neural network connections could have great impact on the comparative effectiveness of analog and digital implementations of neural networks [Hop90].

Additional active circuitry in the form of feedback amplifiers can partially compensate for the many precision problems, but these efforts drastically increase the silicon area and complexity required for each analog function.

Recently another approach has been taken to reduce the precision error of uncompensated analog circuits without resorting to additional circuitry. Kerns reports [KTSL91] using unfocused ultraviolet light to store relatively precise levels of charge on floating gate devices, without requiring high voltage programming. Their primary objective was to compensate for amplifier input offset voltage, but as the authors point out, similar techniques could also be used to store analog–valued weights. This technology is new and unproven outside laboratory conditions, but if it can be made workable its potential is considerable, both for improving analog precision and analog storage.

Barring significant advances in UV–modulated analog storage, the conclusion is that a dense array of analog processors is only possible in CMOS processes if precision requirements are relaxed to the level of four or five bits (and even then, only if program control is possible locally at the PN level). Beyond that, the area requirements of the additional compensation circuitry reduce the density advantage that analog enjoys over digital implementations.

6.2.3 Inter–processor communication

Most research in the design of analog neural processors has focused on the computation. This chapter has addressed some of the synaptic weight storage issues. Yet the problem of *interconnecting* the thousands of neurons that practical systems would require is the most serious issue in extending neurocomputer architectures to emulating very large networks.

It has been shown that, for a general computational model, the required chip area increases *cubically* with node fan–in, or connectivity [BH88]. Piriform model I has a bus–based connectivity model, limiting the area scaling factor to the square of the connection count; nevertheless, as the analysis below will show, the area devoted to routing the metal interconnections between PNs would be prohibitive.

6.3 Analysis of analog implementation

It is worthwhile to examine the performance potential of the proposed design and the cost in silicon area that it would require. It will be assumed in this analysis that six- to eight–bit storage and computational precision is attainable using methods described earlier in this chapter.

6.3.1 Analog performance

The attraction of analog neural networks is that simple multiplication and current summation for addition allow massive parallel computation per synapse. Given nearly coincident LOT arrival times, each column in an EEPROM array can compute its activation concurrently. A reasonably–sized piriform network containing 10,000 neurons would typically have a 1000–element LOT, 10% connectivity between the LOT and patch neurons, and a patch size of 32. (This will be the baseline network used for performance analysis.) Each of the 32 neurons in a patch would have 1000 LOT lines $\times$ 10% connectivity = 100 synapses. Add time for each column would be comparable to the access time of a conventional DRAM chip, on the order of 50 nanoseconds. The WTA circuit could resolve a single winner in approximately another 30 nanoseconds (ns). Approximately six to eight levels of logic propagation delay within the EEPROM controller would be required before the LOT outputs are driven to their proper levels; this could take an additional 20 ns. Thus in approximately 100 ns, a PN could process 3200 inputs, compute the activation of every cell and select a winner. Except for the inherent rostral–caudal delay of recurrent collateral LOT signals, every patch would be performing these computations in parallel.

The "wave of activation" in piriform layer II is due to the delay of input signals arriving at layer II cells, in turn delaying the activation of each cell. Primary and recurrent collateral LOT inputs arrive at each cell nearly simultaneously. LOT signals are gradually replaced by recurrent collaterals, so the total network delay can be estimated from the number of times each LOT signal is replaced. This delay is easily estimated from the network dimensions given above.

Each signal from a LOT of dimension L has a probability p/L of being replaced by a recurrent collateral from a patch of dimension p. The total number of patches P in a network of N neurons is $P = N/p$. The probability of a LOT line encountering n delays is

$$\begin{pmatrix} P \\ p \end{pmatrix} (p/L)^n(1 - p/L)^{P-n}$$

As **figure** 6.8 shows, the delays are statistically distributed according to the binomial distribution.

Figure 6.8 indicates that for the 10,000 neuron network described above, 99% of the LOT lines will encounter less than 20 delays while passing through the network. At 100 ns per delay, total network delay would equal approximately 2 microseconds (μs). This is the estimated time that an analog implementation of piriform model I would require

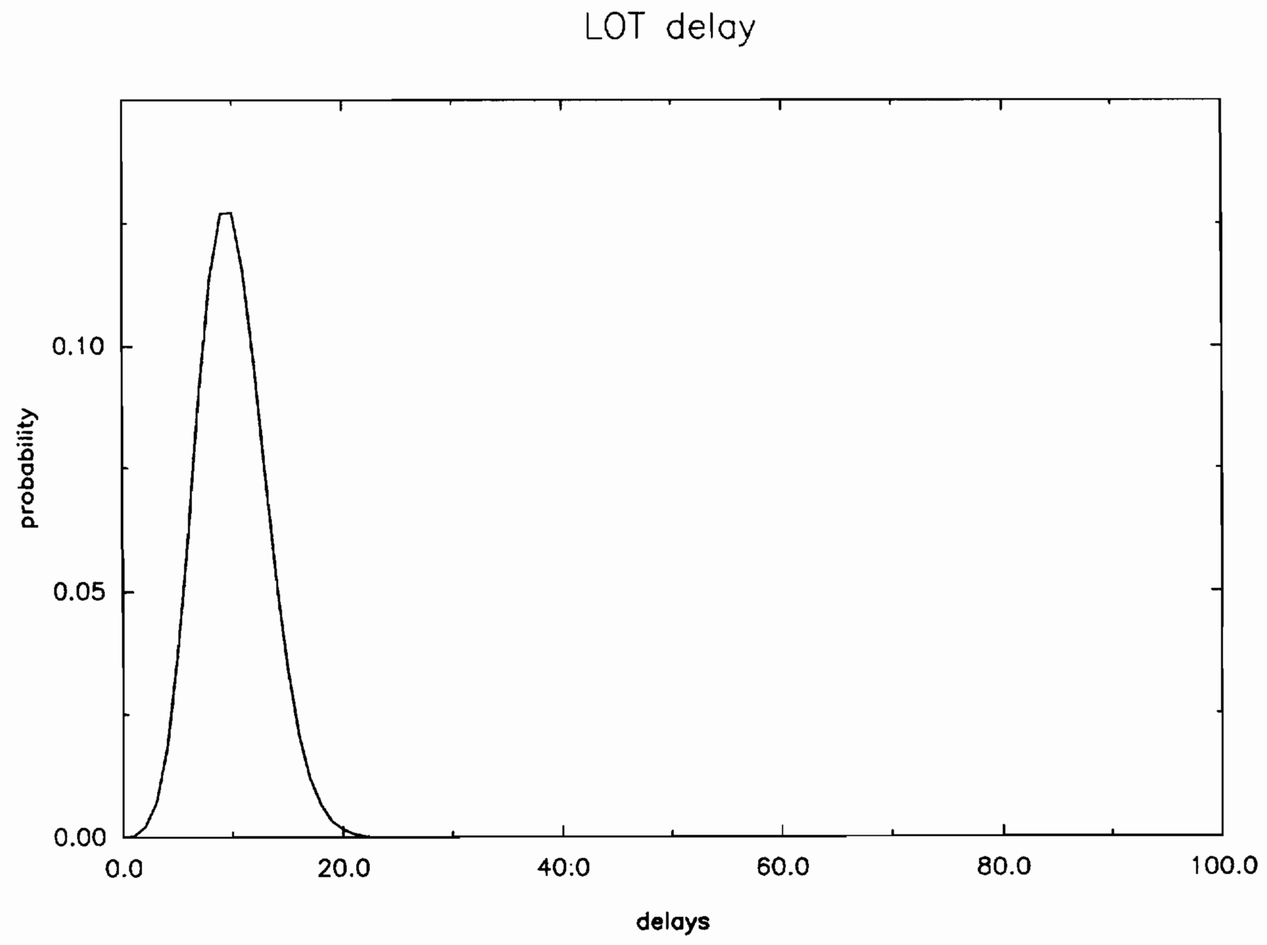


Figure 6.8: Probability distribution of LOT delays in piriform model I.

to process one "sniff", i.e. one sample, of an input. Recall that multiple (approximately four to six) such samples are required for the network to fully resolve each LOT input vector. At this rate the network could resolve a new LOT input vector in approximately 10 μs. Thus inputs could be presented to the network at a rate of roughly 100,000 every second.

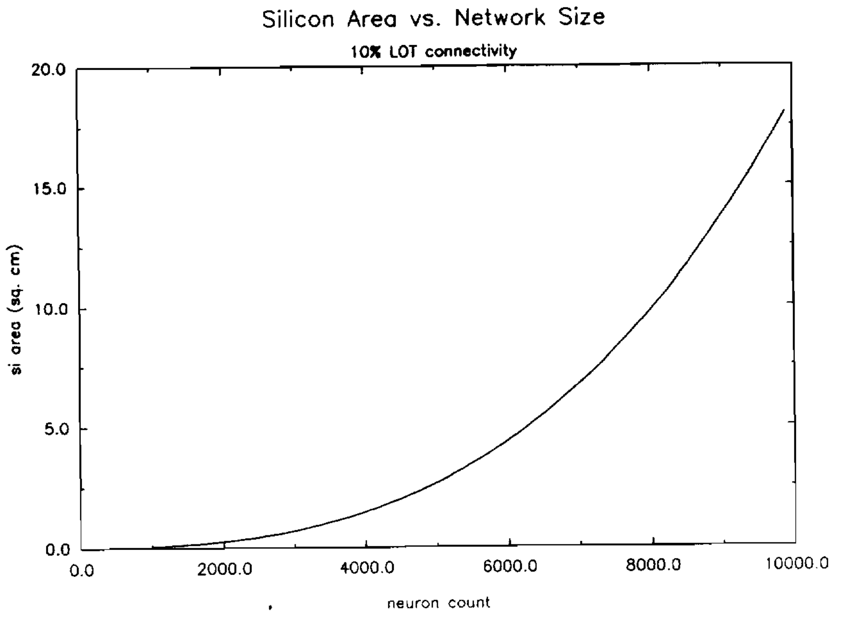
This performance rate is not strongly dependent on the network dimensions. For example, reducing the network size 50% from 10,000 to 5000 neurons would reduce the probabilistic delays by only 10%. The per–PN delay is mostly due to the EEPROM access time, and this delay will increase slowly as patch size grows. And a significant increase in network size need not equate to an equivalent increase in patch size; the number of patches, rather than their size, is more likely to increase in larger networks.

Learning mode performance is a different story. Learning entails individually programming every synapse in each patch's winning cell. Each PN would contain its own high–voltage programming and control circuitry, so all patches could be programmed in parallel. In a commercial digital EEPROM circuit, where the values to be stored are either fully "on" or "off", programming pulses vary between 0.1 and 10 milliseconds. Programming smaller, incremental analog values requires pulses of shorter duration. Eight bits of dynamic range would probably be the minimum for weight precisions of six bits. Assuming a full–scale pulse length of 1 millisecond (ms), each pulse increment would be 1/256th of this, on the order of 4 μs. Accurate programming would probably require multiple iterations of program, sense, program, sense, etc (as is the practice in the only such commercial device, Intel's ETANN [HTCB89]). For that reason, each programming episode would require on the order of 20 μs. All of the winner's active synapses require updating; in the baseline network described above, updating 100 synapses could easily require 2 ms. Four or more successive passes through each input vector are required for learning, just as for categorization. Therefore a reasonable estimate for the time required for this network to learn a new input vector is 10 ms, or a presentation rate of approximately 100 vectors per second. This is approximately 1000 times longer than the time required to classify the same vector. Because of the need for each synapse to be updated independently, learning time is linearly dependent on the number of synapses to be modified. Larger networks require proportionately more connections on each cell, and therefore more synapses needing modification for each learning episode. Therefore, learning time will increase in direct proportion to any change in network size.

6.3.2 Analog cost

The best determinant of VLSI circuit cost is silicon area (there are many additional indirect determinants, e.g. layout density and process complexity). An accurate area analysis requires a fairly low–level description of the required circuit components. Appendix A describes the area assumptions and algorithm used here. The key assumptions concern the density of EEPROM weight memory elements and the pitch between adjacent metal lines; the former largely determines the size of a PN, and the latter limits the number of interconnections that can be placed on a circuit. For this analysis, an EEPROM density of 15 μm^2/cell was assumed. This is comparable to current state–of–the–art commercial EEPROM devices. A metal pitch of 1.4μm was used, which is representative of metal pitch in sub–micron geometry CMOS processes. Area estimates for common analog components such as ADC, DAC, and comparator circuits are more problematical. These circuit areas can vary greatly depending on their required performance, and only performance estimates can be made at this level of analysis. Fortunately, they are relatively minor players in the architecture of this PN, so precise area estimates are not critical.

Figure 6.9 illustrates the dominant role of interconnect in a direct analog implementation. It effectively prevents the fabrication of networks containing more than a few thousand neurons. The key reason for the interconnect problem is the nature of connectivity between each piriform patch and the LOT; although individual neurons sparsely contact LOT lines, even if just one neuron in a patch contacts a LOT line, that LOT wire must be routed to the PN's weight memory. There is a high probability that even small patches (~32 neurons) will manage to contact most of the LOT at least once. Thus most of the LOT must be routed to each patch. The result is that interconnect area



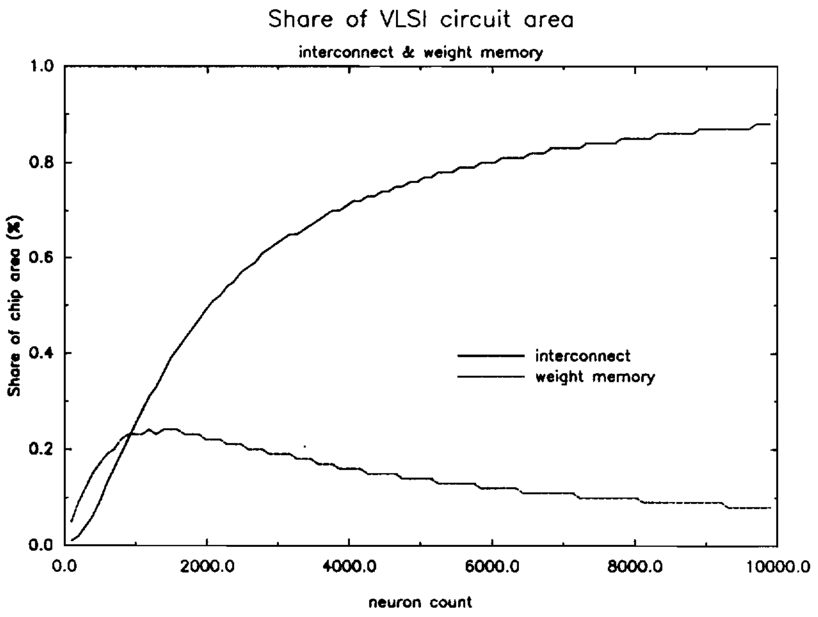


Figure 6.9: TOP: estimated silicon area, in cm^2, required for networks consisting of neurons numbering on the X-axis. (For purposes of comparison, a large commercial microprocessor such as the Intel 80486 requires approximately 1.6 cm^2). The near–full connectivity from each patch to the LOT is responsible for the square–law relationship. BOTTOM: the percentage of chip area devoted to interconnect and weight memory.

scales roughly with the square of the number of neurons.

Manufacturing defects occur in the fabrication of integrated circuits, so VLSI circuits beyond a certain die size cannot be built economically without including extensive fault–tolerance features in their design. An analog neural network such as described here, however, would naturally possess extensive fault tolerance. This is partially due to the distributed nature of neural network processing and because the analog design minimizes the number of potential point–source catastrophic failures. That is, the network could conceivably operate even with the loss of individual LOT lines or patches, albeit with degraded performance. This feature could permit the construction of analog VLSI circuits far larger than traditional digital die sizes. Fault tolerance is a complex issue, however. While ever larger die sizes may tolerate occasional functional faults, large die are increasingly vulnerable to gross electrical faults that cause catastrophic failure (dead shorts between power and ground, for example).

6.4 Summary

Analog neurocomputation is attractive because it appears to be a way to achieve the massive parallel processing that artificial neural networks will require. As it is currently practiced, analog VLSI does have some valuable attributes. Some computational circuits have clear application: obvious candidates would be derivatives of analog multipliers and current–mode adders. However, computation is only one of the three key requirements that any implementation must deal with. Connectivity and the need for modifiable synaptic memory are two areas where analog processing is weak. This chapter has focused on these latter two areas in the context of the piriform model to explore some of the practical aspects of building an analog neurocomputer.

A reliable technology for building an analog weight memory is elusive. EEPROM technologies have been proposed, and this chapter examined the prospect in some detail. Certainly analog EEPROM memories can be (and have been) built; at issue is their workability in this application. A neurocomputer to execute piriform model I will have to support rapid, repeated weight modification. Assuming that EEPROM devices of sufficient longevity can be made, the real challenge is how to accurately program them in continually varying operating environments. At present there is no alternative to extensive central control over the programming of every synaptic weight. Until a superior method is developed, analog EEPROM will be limited to applications that do not require real–time adaptation. This chapter assumed that programming control could be decentralized down to the level of individual PNs. In fact this assumption is quite a technological leap of faith, and practically speaking it may be unjustified. The crux of the problem is that, as John Hopfield pointed out, silicon VLSI has produced nothing remotely equivalent to the biological synapse in its programmable aspects. Perhaps the potential of widespread application in neurocomputation will help spur solid-state research in this area.

Connectivity has traditionally been the weak link in VLSI processing, and that shortcoming would be exaggerated in any proposed analog neurocomputer. This chapter has focused on the problems of routing neural connections within a single integrated circuit, but the problems do not end at the package pins. It is also a critical problem where the analog system has to interface with the external world. The LOT in piriform model I may contain thousands of signals, and each must be accessible to the surrounding system. Alan Murray of the University of Edinburgh, another early researcher in analog neurocomputation, puts the problem in this light:

> Real applications of neural networks require larger arrays of neurons and synapses than can be integrated on a single chip. It is therefore crucial to the effective realization of large, truly concurrent VLSI neural systems that the problem of interchip communication be tackled and solved now. This problem has not yet been addressed directly by any group working in analog VLSI. Earlier work has demonstrated that over 10,000 synapses can be included on a single chip ... However, this implies that large numbers of interneural signals have to cross chip boundaries, and allocating one or more pins to each neuron is grossly impractical. Constraints of chip pin–out (even

> with exotic packaging technology) and wiring, both on- and off- chip, render interneural communication the main impediment to the implementation of large artificial neural systems. [MCT91]

This chapter has addressed some but certainly not all of the practical issues involved in constructing an analog neurocomputer. Perhaps the most important point to be taken away from this is that innovative neural computation or weight storage designs are of marginal utility unless they also ease interconnect requirements, for the cost of additional interconnect will negate the benefits of ever–denser synapse designs. Neural processors and weight memories should therefore be designed first and foremost for their interconnect-ability. Direct routing of a metal line for every network connection is an inefficient way to construct large networks; fortunately, numerous alternatives are possible. The most obvious one is commonly employed in digital systems: that is, to share, or *multiplex*, several virtual connections on a single physical interconnection medium. An implementation of the SS architecture featuring this technique is described in the following chapter.

Chapter 7

Digital Implementation of the SS Architecture

This chapter describes in some detail a digital version of the SS architecture, known as *SSD*. The most significant advantage that a digital implementation has over analog is its realizability: in most cases, a digital circuit is easier to design, build, and test than its analog equivalent. In addition, some aspects of the piriform model naturally suggest a digital approach. This chapter begins with a description of these characteristics. The key components of SSD are then covered in detail, followed by a cost/performance analysis of the proposed design. The chapter concludes with a summary of the strengths and weaknesses of a digital implementation of the piriform model.

7.1 An analog model in the digital domain

The key problems with SSA in particular, and with any analog neural network implementation in general, are:

- Expensive and inefficient interconnections between neurons. Interconnect dominates silicon area, yet the winner–take–all behavior of patches suppresses most neural activity. Very few cells simultaneously utilize the available interconnection bandwidth.

- Precise, reliable integration of analog weight storage and computation are unsolved problems in current VLSI technologies.

The piriform model's sparsely activated, physically sparse interconnect argues for a shared communication resource. Also, because the neurons that contact a LOT line may be dispersed widely throughout the network, the communication mechanism must ensure that when a neuron sends a pulse down the LOT, it is heard by the widest possible audience of listening neurons. *Broadcast communication* over a central bus is the simple, multiplexed communication technique used to address these requirements in SSD. LOT signals are encoded and sent *sequentially* over a single shared LOT bus rather than via parallel transmission over signal lines dedicated to each unique LOT line.

Imposing a sequential order on LOT transmissions preserves the model's critical temporal information. All that matters temporally is the strict rostral → caudal sequence of neural activation within a single sniff. This serial ordering guarantees that every neuron receives its relevant inputs before forming its output. SSD takes advantage of sequential processing to achieve significant economies in silicon area. While the resulting performance penalty is substantial, the cost/performance ratio of large–scale systems using this approach is favorable.

7.2 SSD

7.2.1 Overview

From a high level, SSD is identical to SSA. As shown in figure 5.1, a linear array of processors communicate via a global interconnection medium, the LOT bus. Each processor, or PN, stores all the synaptic weights and performs all neural computation for neurons in a single patch of piriform layer II. In this case, the LOT bus is truly global: it is a single communication medium, visible to all PNs at all times. The LOT bus is no longer a bundle of individual connections between neurons but rather consists of a small number of signal lines. These lines contain the binary address of a single transmitting neuron, and this address is broadcast to the entire network.

A diagram of a single PN is shown in figure 7.1. The IO area contains the interface circuits necessary for communication with the bidirectional LOT (and input commands

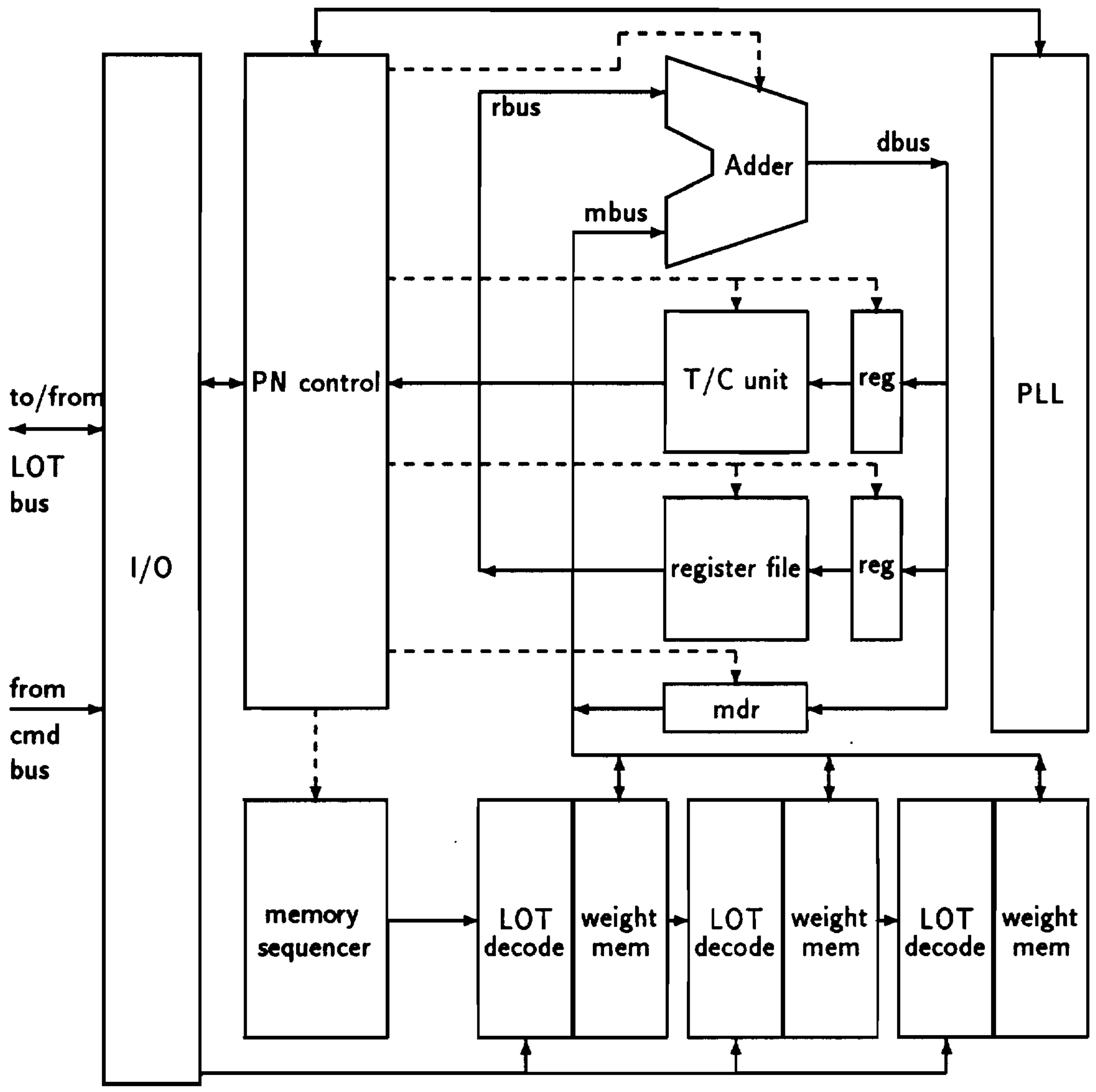


Figure 7.1: Block diagram of a single PN in SSD. Internal buses interconnect weight memory and the three datapath elements: the adder, register file, and the threshold/compare unit. Weight memory blocks are accessed from decoded LOT inputs or through the memory sequencer. I/O is via the binary encoded LOT bus.

from the command bus). The four element data path consists of an adder, a threshold/compare unit for evaluating cell activation levels, a register file containing intermediate cell activations, and a memory data register for communicating with the synaptic weight memory. This single data path is shared by all cells in one patch. Data path control and PN sequencing is accomplished in the PN control block. Finally, each PN contains one LOT decoder and weight memory for every cell in the patch.

A PN operates on a broadcast LOT address by presenting the address to each decoder in parallel. Those cells programmed with a connection to the sending cell respond with an active signal on a decoder output. This signal acts as a word line into each cells' unique weight memory. The addressed memory word is the current value of that synaptic connection. Each PN emulates the activity of many cells, and decoding and memory access for each cell is performed concurrently within each PN. Multiple cells may have active connections to the broadcast address. After the parallel memory access, parallel activity ceases, and each PN processes each active connection in series.

Following the processing of each connection a single cell may cross a pre-set activation threshold; if more than one do, the first is arbitrarily declared the patch "winner". The PN then arbitrates with other PNs for access to the LOT so it can broadcast the unique binary address of the winning cell. The arbitration mechanism is a simple daisy–chain, where upstream (rostral) PNs always have transmission priority over downstream (caudal) PNs. The transmitted address is decoded and processed by PNs elsewhere in the network.

Details of the main functional blocks of the PN are provided below.

7.2.2 Communication

The network's response to a single input is to send a flurry of messages over the LOT bus. Each message consists of a sending neuron's unique binary address, announcing itself as an active output (and therefore a patch winner). To permit each neuron to uniquely identify itself in a single transmission, the LOT bus must be at least $\lceil \log_2 n \rceil$ lines wide, where n equals the total number of neurons in the network.

Encoding LOT activity in this manner requires a fraction of the physical interconnect area required by encoding each connection directly in metal. Of course, *decoding* the LOT transmissions in the receiving neurons entails additional circuit complexity, and therefore, some cost in additional area.

LOT decoding in SSD is accomplished by dense programmed logic arrays (PLAs). Every PN contains multiple PLA decoders, one for every cell in the PN. Each decoder is programmed with the binary addresses of every sending neuron in the network to which that cell is connected. All decoders in each PN simultaneously decode every LOT transmission. When a transmission is recognized by a receiving cell, the decoded address selects an address in the PN's local weight memory corresponding to the synaptic weight of that cell's connection.

Thousands of PLA decoders occupy significant silicon area. Yet they are an economic alternative to the cost of encoding network connectivity information directly in aluminum wire. The decoder area grows linearly with the number of input (LOT) lines to be decoded, which in turn increases as the $\log_2$ of the number of neurons in the network. By contrast, as pointed out in the previous chapter, the area devoted to direct metal interconnect increases with the square of the network size. Another advantage of PLA–encoded interconnection is the relative ease of changing connectivity. Simple PLA reprogramming (usually a single mask change) suffices to change connectivity for the entire network.

The primary drawback in using encoded LOT communication is that interneural intercommunication is now *sequential*; only a single neuron can transmit on the LOT bus at a time. Two characteristics of the piriform model partially alleviate the performance hit that this would otherwise imply.

In the biological model, the wave of activation from rostral to caudal cells is due to the sequential activity generated in recurrent collateral axons from the earlier–activated rostral cells. By strictly ordering SSD communication such that rostral neurons are always the first to transmit on the LOT bus, arbitration in the neurocomputer is straightforward and consistent with this biological model. Output arbitration is accomplished via commands over the command bus. Raw LOT input has first priority to transmit, followed by winning cells from the most rostrally-located PNs. Subsequent transmissions from progressively more caudally-located patches follow in sequence, until all patches in the network have transmitted once on the LOT bus. This constitutes a single cycle, or "sniff", in the piriform model.

The second key characteristic of the model is the fact that winner-take-all behavior suppresses most network activity; only a single cell in each patch reaches sufficient activition to transmit. The patch sizes vary under current model simulations from 20 to 100 cells, meaning that only 1% to 5% of the network's cells will use the LOT on each sniff. Systems with dedicated hardware communication paths between individual neurons that emulate winner-take-all networks (a widespread characteristic of biological nervous systems) will under utilize these expensive communication resources. In SSD, minimum hardware is used to send only the messages that are actually needed on each cycle.

The piriform model describes a hybrid mixture of serial and parallel processing in biological nervous systems. SSD supports these operations. Coincident with the temporal sequence of LOT transmissions described above is parallel processing of each LOT message by all the neurons in the network downstream. Every PN's decoders are simultaneously decoding and operating on the received message.

7.2.3 Weight storage and learning

Dense PLAs (one for every cell in the network) decode the address on the LOT bus and select a memory location at the address transmitted. When a decoder produces a "hit" in response to a LOT transmission, it selects a word line at an address in weight memory. A portion of each PN's weight memory is assigned to each cell, where it stores the value of all the synaptic weights between the LOT and each neuron's receiving dendrites. In SSD, weight memory is implemented as static random access memory (SRAM).

The data at the selected weight memory address is written to a register for access by the neural computation system, where it is used to alter the activation level of the receiving cell.

Modification to weight memory is the essence of learning, and in SSD each PN modifies the weights of its constituent cells independently. Individual PNs must be able to distinguish each sniff's active weight memory addresses from inactive ones, necessitating the inclusion of an additional bit in every weight memory location that serves as an *activity* flag, set upon selection from the decoder and cleared at the end of every sniff.

After determining the patch winner and broadcasting its identity over the LOT, each PN must identify every active synaptic address in the the winning cell's weight memory. The value of each is increased by the weight increment value, and each sum is written back to the same memory location. A straightforward (but slow) way to accomplish this is for each PN to sequentially read every address in the winning cell's weight memory, searching for those addresses flagged as "active" on that sniff. Faster methods exist at the cost of additional circuit complexity. This lengthy sequential procedure is only performed when the network is in learning mode. Each patch performs this weight update operation independently, permitting the network of PNs to update their weights essentially in parallel. The speed of the network when operating in learning mode is consequently slower than its speed in performance mode.

For a neuron to learn in the piriform model, it must not only win its patch but also navigate through a complex set of activation thresholds. In a biological system, the mechanisms that change synaptic efficacy (i.e., the value of the synaptic weight) are hypothesized to be sensitive to voltage depolarization across the neuron membrane. Only strongly–activated neurons can exceed these thresholds and are modified. Different versions of the piriform model show great variability in the value of the different activation thresholds, requiring the flexibility of a programmable threshold mechanism. The learning system in SSD supports this function using registers in the threshold/compare (T/C) unit. These registers are programmed with the values of the various activation thresholds by which the activity of winning cells are measured. Programming is controlled by instructions on the command bus, broadcast to and executed by each PN in parallel during an initialization period. Upon evaluation of each patch's winning cell, circuits in the T/C unit indicate if the winner's activation level surpasses the target threshold. If the network is in learning mode, and if the winning cell exceeds the learning threshold, then the weight–update procedure is executed.

7.2.4 Neural computation

Arithmetic is performed by adding each cell's selected connection weight to the contents of accumulators containing the current activation levels of each cell. The accumulators (one for every cell, constituting the PN's register file) are then updated with the resulting sums. The weight memory access and accumulation for each cell must complete before the next LOT transmission. In most cases a decoded LOT address will select weight memory sites from only a few cells within the same PN. Nevertheless, *every* cell must be checked for activity. Even if only a single PN has active weight memory sites across every cell, the entire network must await the completion of its processing.

Incremental accumulation of activity is consistent with SSD's serial communication design. Simultaneous messages cannot be sent over the LOT, so individual cells have no use for an ability to compute massive inputs in parallel. The serial computation bottleneck slows the speed of communication, however. The rate that signals can appear on the LOT is dependent on the speed of digital addition.

The fastest way to perform digital addition is to use multiple hardware adders for those cases when weight memory sites from several different cells in a PN are addressed simultaneously by a single LOT transmission. Analysis of the area requirements of digital adders indicated that only a single hardware adder can be accomodated per PN, however. Therefore each PN contains only one adder, and access to it is sequential.

The time to process each cell's activation level using a single adder can be reduced by pipelining the datapath operations. The four–stage PN pipeline is shown in table 7.1. Three internal buses (*mbus* for memory operands, *rbus* for register file operands, and *dbus* for adder output) are required to prevent resource conflicts during pipeline operations. These are shown in figure 7.1. Table 7.2 is a pipeline reservation table showing pipe status during the processing of four cell memory addresses from a single

LOT decode	read wt. mem	read reg, add	write reg, threshold compare

Table 7.1: The four stages of the PN processor pipeline in SSD.

	t_0	t_1	t_2	t_3	t_4	t_5
LOT decode	a	b	c	d		
read weight memory		a	b	c	d	
read reg file, add			a	b	c	d
write reg file, perform T/C				a	b	c

Table 7.2: A reservation table showing the status of the PN processor pipeline during the processing of four weight memory addresses. In the table, each horizontal step is equivalent to consecutive time steps; a vertical slice shows coincident events. Note that simultaneous read/write access to the register file is required beginning at time step t_3.

LOT transmission. While serial access slows the processing of each LOT input, it should be possible for the adder and accumulator portion of each PN to run at cycle times considerably shorter than the LOT cycle time. The latter is determined primarily by the time required to drive the long, highly capacitive LOT bus, and by the need to have enough "dead time" to ensure proper synchronization between widely–dispersed PNs. Local adders and register files within each PN will have inherently less capacitive loading and clock skew, and can operate at higher frequencies. A phase–locked loop clock generator is included as a part of each PN for the generation of the local high–frequency clocks and to ensure synchronization across an entire wafer. The sparse connectivity between each cell and the LOT means that numerous LOT transmissions will appear between consecutive accumulations for any particular cell.

A key feature of the piriform model affecting computational density is its inherently low precision. Low precision makes massively parallel analog computation practical; it also reduces the cost of digital computation. (The area required by all of the processor blocks in each SSD PN increases linearly with increased precision.) The adder, register file, and T/C unit are designed to be twice as wide as the width of weight memory to allow sufficient overhead in the course of repeated additions (i.e. an eight bit adder to handle four bit weights).

7.3 Cost/performance estimates

7.3.1 Cost: silicon area estimate

Appendix A contains the source code and input files of an area estimation program written for this analysis. Estimates of layout densities were based on known commercial integrated circuit layouts.

The ideal area estimation technique would begin with a precise description of each PN component in terms of logic gates and transistor arrays. Integrated circuit layouts from several sources could then be used as references for estimating the required layout area. This proved to be a difficult task. Synthesizing a gate–level logical model of a PN would constitute an entire thesis in itself, even if the PN had been precisely defined beforehand. Given the wide scope of this architectural investigation, it was not possible to generate a PN model for each implementation accurate down to the gate level. Had it been possible, it would have been unwise: ongoing changes in the model definition itself would have negated much of any such design effort. Instead, PN components were defined at the functional level in terms of conventional logic blocks whose area could be roughly estimated from schematics and layouts of similar circuits. While some blocks were defined down to the gate level, most areas were estimated by extrapolating from empirical area models drawn from the reference circuits.

Estimation of the register file area offers an example of this technique. Transistor–level schematics and a detailed layout of a commercial VLSI product containing a similar register file formed the starting point; the next step was to decide how much the known and desired circuits had in common, and how much they differed. In this case, the reference circuit was a single ported register file while the PN requirement was for a dual ported design. It was assumed that this extra functionality could be gained at the cost of a 25% increase in the number of transistors from the reference design. For the known circuit, fabricated in a dual–leval metal, 0.8 micron CMOS process, the measured register file device density was just over 93 square microns per transistor. The density measure used in the PN's register file was the same, but 25% more transistors were required per cell than the 14 used in the reference design. The area of a single register was thus estimated to be:

$$(93\mu^2/\text{transistor}) \times 18 \text{ (transistors/bit)} \times m \text{ bits} \times 2$$

where m = weight memory width, a model variable.

Several factors complicated this estimation procedure. First of all, the degree of functional difference between the reference and target circuits can be significant. Adders, for example, come in many varieties, and the area of a carry–look–ahead adder can be considerably larger than, and therefore a misleading guide to, the area of a conditional–sum–adder layout. Each known layout had to be scrutinized for small functional differences. The technology used in each reference layout is a major factor, because automated place–and–route programs can require three to four times as much area as a human- or machine–generated datapath layout. Additional variables complicating the task are the details of the fabrication processes: reference layouts could be drawn using either one, two, or three levels of metal interconnect, for example. Some circuitry (e. g., PLA, SRAM) is inherently regular, and can be reasonably well estimated from the area of individual cells; other circuits like adders and multiplexors are random, irregular, and prone to estimation error. Finally, layouts can be optimized for either speed or area, producing overly large or small estimates. These factors required that a conservative approach to area estimation be taken.

Eventually, area estimates of individual blocks were combined and used to answer the following questions:

- How much silicon would be required to contain a network consisting of approximately 10,000 neurons?

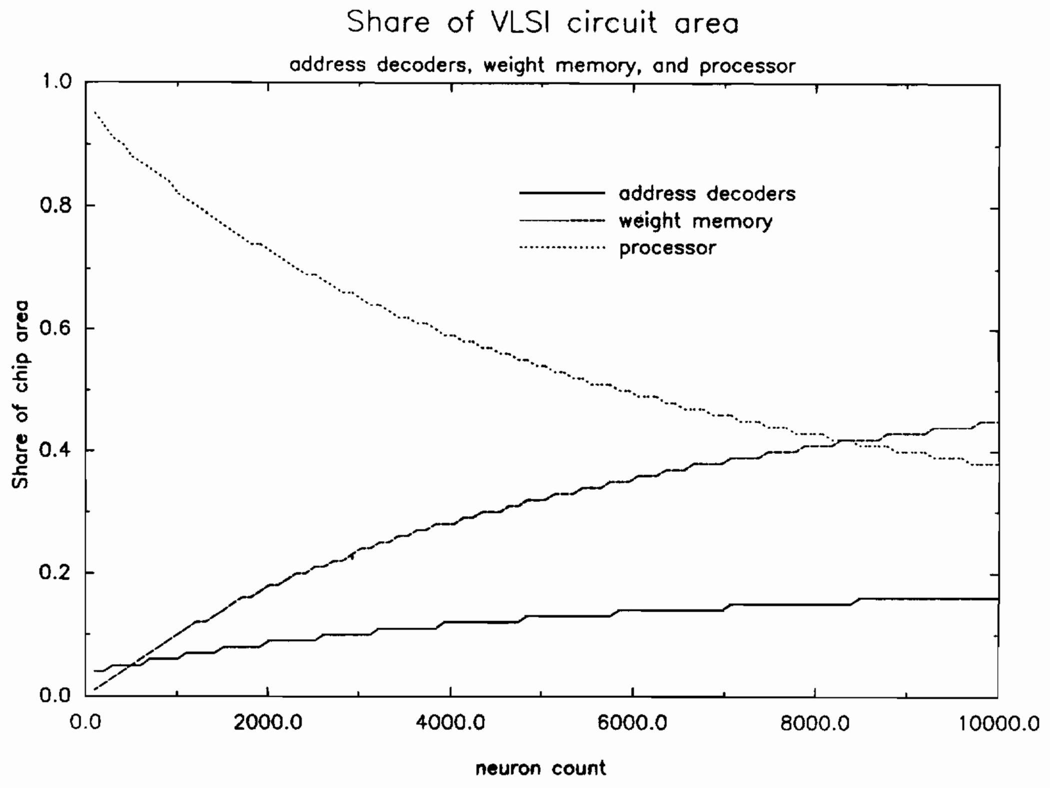


Figure 7.2: **Relative share of chip area consumed by the three primary components of the SSD implementation: the processor, synaptic weight memory, and LOT address decoders.**

- What is the maximum size network that could be fit onto a single 6-inch wafer?

Details will follow, but analysis indicates that a 10,000 node piriform network executing on SSD using six-bit synaptic weights would require approximately 11.2 cm^2 of silicon. Each of the 10,000 neurons would be sparsely contacting a 1000–element–wide LOT, making a combined total of approximately 10^6 connection weights stored on the chip. A scaled–up SSD executing the piriform model on a single 6–inch silicon wafer could support a 35,000 neuron network, containing more than 12 million six–bit synaptic weights.

Figure 7.2 divides total circuit area into three components (weight memory, decoders, and processor area) and shows the relative area of each as the network dimension is varied. Small networks consisting of less than 2000 neurons devote most of their area to the processing functions, and relatively little to weight memory and interconnect (i.e., address decoders). Processor area increases in absolute terms as network size grows,

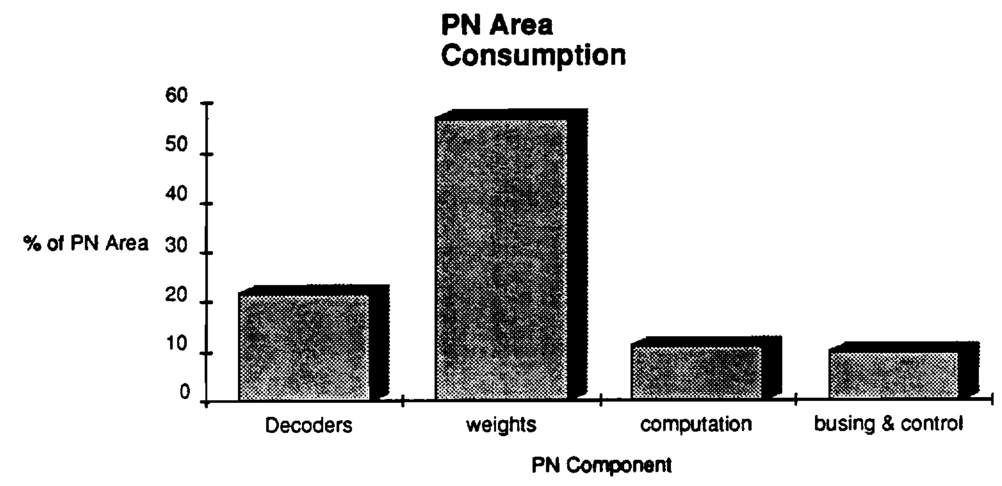


Figure 7.3: Relative share of chip area consumed by the primary components in the SSD implementation of a 35,000 neuron network. Synaptic weight memory and LOT address decoding together occupy more than 85% of the area of a single 6 inch wafer.

but decreases relative to the area required by weight memory. Figure 7.2 indicates the beginning of this trend, and figure 7.3 shows its culmination in a plot of the relative share of processor area for a large network consisting of approximately 35,000 neurons. A network of this size would occupy most of a six-inch wafer, of which 15% is devoted to processor area. Note that unlike SSA, interconnect costs (in the form of address decoders) do not dominate, and only become significant in networks with more than several thousand neurons. The large size required for the basic processor blocks becomes overshadowed by the ever-increasing need for more memory with very large networks. In SSD, interconnect costs do not get out of hand and effectively limit the ultimate size of the network, as they do in SSA.

Details of the area estimation procedure can be seen in appendix A, but a few comments about the assumptions that were used are in order. Each PN's register file estimation has been described previously. The adder area was based on a conditional–sum adder design that provides a good compromise between performance demands and area efficiency. Its area estimate resulted from a gate–level design and precise layout references. The phase–locked loop was based on several commercial layouts and designs described in [JH88] and [JBHK87]. When functional blocks could be decomposed into gate–level descriptions, a standard device density of 125 square microns per transistor was used as the basis of area estimations. This density is consistent with a current generation, sub–micron, double–metal CMOS layout.

The two large arrayed circuit elements, the PLA decoder and SRAM weight memory, were derived from two different area models. The area of the SRAM memory was estimated from known commercial designs: the number of bits required in each PN is a simple function of the patch size, LOT dimension, and connectivity. Memory cell density was assumed to be 42 square microns per bit; while slightly beyond the density of most commercial 1 megabit SRAM designs, this is well within the scope of next generation devices. Memory support circuitry such as sense amplifiers were assumed to consume additional area equal to 50% of the area of the SRAM cell array [Dil88].

Decoder area calculation is similar, but is based on a specific PLA layout model [WE85]. PLAs, like ROM and RAM, are dense arrays of transistors, but the sparse connectivity to the LOT gives PLA decoders a significant area advantage over a ROM–based decoder. The "and" plane of a PLA encodes only a subset of the possible minterms in a logical expression. This subset corresponds to the limited set of LOT addresses to which a particular cell is connected. There is no "or" plane in the decoder because all "and" terms select active weight memory addresses. Decoder size will scale linearly with the degree of network connectivity. A ROM decoder, on the other hand, would encode all possible logical combinations of the LOT address inputs, requiring a considerable increase in area. PLA area in this model is a function of network connectivity, cell size, metal pitch, and the number of inputs (encoded LOT width). The decoder arrays can get quite large, so area is allocated for periodic signal buffers to drive the line decoder inputs.

7.3.2 Performance: input vector processing rate

Summarizing the sequence of actions that occurs on each sniff in the piriform model:

1. sequential broadcast of each LOT message
2. each PN decodes source address, evaluates connectivity of its cells
3. each PN performs incremental accumulation of activity of its cells

Two principal delays mark this process: driving the LOT bus, and accumulating the cells' activity levels.

The single LOT bus tying together the entire network presents a considerable capacitive load to each patch's output drive circuits. Bus drivers with periodic signal repeaters should be able to drive the entire LOT bus in approximately 40 nanoseconds, establishing the period of a single LOT cycle.

Each stage in the PN processor pipeline can be completed in approximately 10 ns. A PN clock cycle is thus one–fourth as long as a LOT cycle. The time for a PN to process n cells in response to a single LOT input is linearly proportional to n; that is, $\Delta t(p + n - 1)$. In this case, p (pipeline depth) is four, and Δt (the PN clock period) is 10 ns. The time to process a patch containing $n = 32$ cells is approximately 350 ns.

Each PN could thus process a single LOT input in less than 400 ns. In a 10,000 neuron network with a LOT dimension of 1000, approximately 10% of the LOT lines would be active on each sniff. Added to this raw input are the LOT broadcasts of the winners from each patch. A network of 10,000 neurons organized into 40–neuron patches would have 250 patches, each of which would also broadcast one message onto the LOT, for 350 LOT broadcasts in one sniff. In addition, multiple sniffs (e.g., 4) are required to process a single LOT input vector, so processing time for a network of this size is estimated to be:

$$(350\text{ns/cycle}) \times (350 \text{ cycles/sniff}) \times (4 \text{ sniffs/vector}) \simeq 500\ \mu s/\text{vector}.$$

This is approximately equal to 2,000 LOT input vectors per second, each containing 1,000 input elements.

When the system is actively learning, an additional weight modification step follows each sniff. Every activity bit in the winning cell's weight memory must be examined along with the weight itself to see if that synapse was active. If so, the corresponding weight must be incremented and written back to the same address. Thus one or two weight memory accesses are required for each of the winner's synapses. Assuming 10% connectivity on a LOT composed of 2000 lines, with 5% activity, each winning cell would have 50 synaptic sites. On average it can be assumed that one–half of each winner's synapses will be active; each single read/modify/write cycle requires 30 ns, and a simple read/evaluate will take 20 ns. The entire weight update procedure would then add

$$(25) \times (30 \text{ ns} + 20 \text{ ns}) = 1.25\ \mu\text{s}$$

to each sniff.

This is only a small fraction of the time required to process a single vector in performance mode. It can be concluded that network performance during learning mode would differ little from its operation in performance mode.

Comparison to SSA

The 11.2 cm of silicon area required for a 10,000 neuron network executing on SSD would require approximately 60% of the area required for an SSA architecture of equivalent size. The digital implementation's area advantage relative to an analog implementation improves with larger networks. While an SSD architecture consisting of 35,000 neurons could fit onto a single 6–inch wafer, such a wafer could hold only 19,000 neurons in an SSA implementation.

SSD's area advantage is directly related to the treatment of interconnect. Recall that the bus–based architecture of Super Sniff requires an increase in area proportional to the square of the network size; even though an individual SSA PN is smaller than an SSD

PN, the area required to route the entire LOT to ever more PNs easily compensates for it. In the hypothetical 19,000 neuron SSA network that would completely fill a six-inch wafer, 94% of the wafer area would be devoted to routing wire.

Balancing the area economy of SSD is the potential in SSA for far higher performance. It is estimated that SSA could process a new input vector every 10 μs. Due to numerous circuit design uncertainties, it would be wise to assume a factor of two or three padding on this figure; assume one input vector every 25 μs. Even so, this is approximately 20 times the performance of SSD in non-learning mode. SSA's performance advantage is not related to the multiplexing of interconnect, however. The reason for the discrepancy is the processing bottleneck in SSD due to the need to multiplex a single adder within each PN. Digital adders are large, and PNs would be prohibitively expensive if each one contained multiple adders.

Learning, or weight modification, would be more than 100 times faster in the digital implementation than in SSA. The iterative high-voltage EEPROM programming required to modify analog weights in SSA would be a significantly slower process than the sequential memory read/write required in SSD.

On the basis of these figures, an initial cost/performance analysis of SSA vs. SSD would come down in favor of SSA. A 20x speedup would justify the 40% cost penalty. However, other issues enter to throw doubt on this conclusion. Most important is the scalability of the two architectural approaches. The largest SSA network envisioned here would contain 19,000 neurons. Doubling the available silicon area (by fabricating a single network on one 8-inch wafer) would permit only a 25% increase in network size, to 24,000 neurons. Successive increases would be even harder to attain, as interconnect requirements gobble up ever more circuit area. Combining multiple wafers to achieve larger networks exacerbates the problem, for inter-PN communication is vastly more difficult off-chip than on-chip. An SSD implementation cannot linearly increase network size with increasing circuit area, but it approaches that goal better than SSA would. For example, a similar doubling of silicon area would permit the construction of a network of 56,000 neurons on an 8-inch wafer, a 60% increase.

The importance of scalability centers around another question: how large must a network be for a particular application? Piriform model I is itself a new network model, and applications for it are currently nonexistent. The only available guide is biology itself. Piriform cortex in the mammalian brain contains millions of neurons, and LOT lines numbering on the order of 10^5. If these proportions are representative of real requirements, then scalability to large networks is indeed important. In that case, direct interconnect schemes like in SSA will not be practical. However, if networks containing neurons numbering in the low tens of thousands are sufficient, then SSA would most likely be the alternative of choice.

If rapid, real–time learning were crucial in a particular application, then there is no alternative to the digital SSD implementation: SSD would have a nearly 100–to–1 advantage in learning speed over SSA.

Chapter 8

Conclusion

The purpose of this thesis was not merely to describe another neural network model but to explore ways that a biologically–faithful model could be implemented in electronic hardware. In addition, this thesis pointed out the three architectural issues that any neurocomputer architecture, implementing any neural model, must address. Focusing on these issues is as important as the architectures themselves. It is worthwhile to review the results achieved.

As a prelude to custom architecture investigations, a modified version of the piriform model was implemented on CNAPS, a commercial neurocomputer. The results illustrated the suitability of the SIMD approach to neural computation. Speedup over uniprocessor implementations could be significant if the individual network layers were large enough. However, certain portions of the algorithm require reformulation from serial to parallel execution in order to gain maximum benefit from the available parallel hardware (for example, distributing the serial square root calculation among all processors).

Focusing on the implementations of piriform model I, this thesis explored two quite different hardware strategies, SSA and SSD. While each version was analyzed in terms of cost/performance tradeoffs, it would be somewhat unfair to compare them purely on the merits of these results. This is because many of the core features of SSA, such as analog EEPROM weight storage and programmability, are unproven techniques in VLSI. It is entirely possible that attainment of the precision levels contemplated here would require additional compensation circuitry extensive enough to wipe out the area advantage that an analog PN enjoys over a digital PN. It would not be inaccurate to say that the analog implementation was presented here in its best possible light.

This was done so as not to obscure the real problem with an analog implementation: the high cost of interconnect. The silicon area required for directly interconnecting thousands of analog neurons makes such a network prohibitively expensive. This does not mean that analog processing has no place in neurocomputation; it does suggest that analog is better suited for sensory, rather than associative processing, where connectivity is local in extent. VLSI processes routinely squeeze ever-greater numbers of components onto a given silicon area, but progress in interconnecting these components comes much more slowly. This trend can be expected to continue.

Measured by its treatment of interconnect, SSD is a success, because interconnect structures in SSD (including address decoders) do not dominate silicon area even for networks containing millions of connections. SSD also utilizes digital weight storage and computation, however, and the latter has area penalties of its own. Digital processing requires too much silicon area to allow multiple adders on each PN. The activation of each cell in a PN must be computed serially. This creates a processing bottleneck, which is the reason that throughput of SSD is one–twentieth that of SSA.

A second drawback of SSD is a result of the large die sizes required for large networks. It is unlikely that VLSI manufacturing can produce defect–free chips of the sizes contemplated here; the circuits will have to have fault–tolerant properties built in. Fortunately, the modular nature of the SS architecture permits the use of fault tolerant mechanisms with relative ease. Redundant PNs could be fabricated and substituted (either electrically or via software) for PNs that fail under test. This technique is used today in large signal processing circuits (and in implementations of the CNAPS architecture described in chapter 4). One key to a fault tolerant architecture is to minimize the number of single–point failure modes; that is, nodes in the circuit where a failure would kill the entire network. The LOT and command buses are two such potential points. Periodic error–correcting circuitry located along each bus is typical of one technique that would minimize the effect of such failures.

A third drawback of SSD is that multiplexed communication does a poor job of representing the analog time domain. Fortunately, the piriform model is sensitive only to the rostral–caudal ordering of neural transmissions, and SSD preserves only this skeletal ordering information. Time is not a factor in the computations themselves. Other biologically plausible neural models may require more precise continuous time representations to be effective: if so, the complexity of multiplexed communication would have to be increased significantly to accomodate more temporal information. Partly as a result of this, a common sentiment is that only analog computation can represent the analog temporal character of real nervous systems. However, the widespread commercial success of digital signal processing in continous, real–time applications argues strongly for the merits of discrete–time digital architectures even in this domain. Unlike SSA, no exotic circuit techniques were proposed for SSD. All of the functional blocks described are routinely realized in commercial designs today. Nor does SSD depend on revolutionary advances in process technologies (the large die sizes notwithstanding). For realization of the largest networks discussed, containing some 35,000 neurons and several million connections, the analysis assumes only slight increases in memory density, a technology which has not yet approached its practical limits. The phase–locked loop clock circuits and pipelined processor architecture featured in each PN are common elements in today's state of the art microprocessors.

Figure 7.2 in chapter 7 illustrates the dominant role that memory circuits will have in the largest versions of SSD. Processor area occupies less than 15% of the silicon, while SRAM consumes more than 60%. This observation reflects a trend evident in today's commercial RISC processors: to get top performance, ever–increasing portions of valuable chip area must be devoted to fast local memory. The front line of processor development today is defined by the integration of large amounts of local memory with a minimum amount of fast logic circuits. The key advantage of SSD is that it pushes the network capacity problem into this familiar arena, where progress has been rapid, and away from expensive interconnection solutions, where progress has historically been slow.

Figure 8.1 superimposes the area estimation curves shown earlier in chapters 6 and 7. It directly compares the cost (i.e., chip area) of the two approaches. Two noteworthy facts can be gleaned from this figure. First is the enormous cost of implementing large analog networks directly; individual circuits may be arguably larger or smaller than what I have estimated here, but for large networks, all that matters is the interconnect cost. The second key point is the magnitude of the networks contemplated. Figure 8.1 stops at 10,000 neurons, where the two approaches are roughly compatible in terms of area (beyond this, SSA's area requirements dwarf those of SSD). The largest network considered here contained 35,000 neurons and over 12 million synaptic connections. A network of that size would occupy most of a six-inch silicon wafer. Even the rapid process improvements that have traditionally accompanied VLSI technology are unlikely to produce the tremendous density increases that will have to occur to drastically change the maximum network dimensions. 35,000 neurons is a sizeable network, but a far cry from the millions of neurons and billions of connections in the piriform cortex of simple mammals. The complexity of a single digital PN, shown in figure 7.1, argues for slow change at best in these numbers in the near future.

Given the relative strengths and weaknesses of SSA and SSD, one obvious suggestion is to combine the strength of the two approaches in a hybrid design. The efficiency of digital communication and storage combined with analog computation would potentially offer an efficient compromise. But as mentioned in chapter 6, the problem with a hybrid system is the need to convert between digital and analog representations. DACs and ADCs are large, expensive circuits. Most current circuit research in the area of digital/analog conversion is aimed at ever-higher precision or speed, and ignores the realm of area efficient, low-precision conversions. It is likely that research in this area could produce faster and cheaper converter designs than are currently available. Without significant improvements, large hybrid designs may require more conversion overhead than would justify their cost.

How much is the piriform model itself responsible for these results, and how might these results change given a different model? The piriform model is a *cortical* model, and

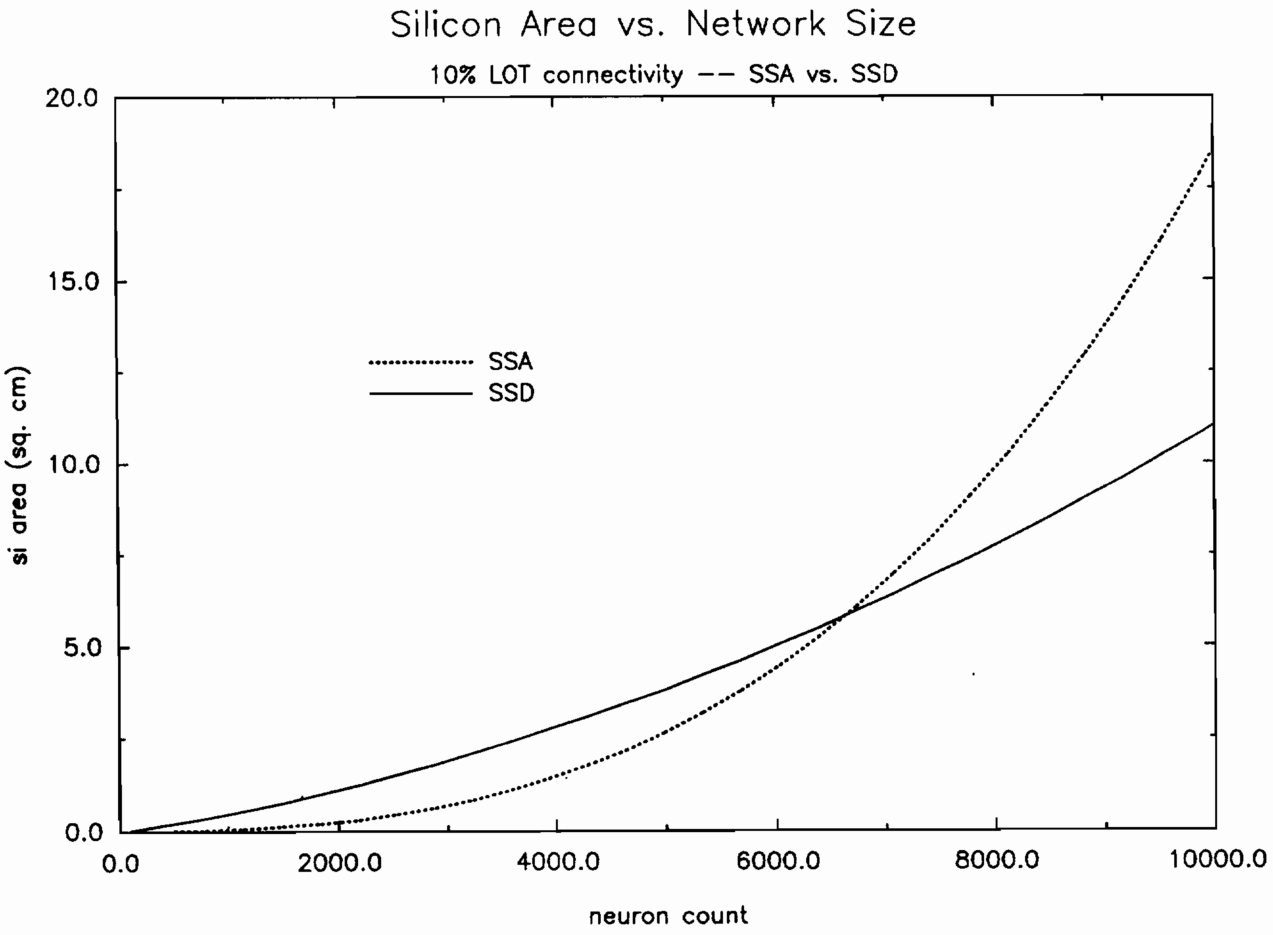


Figure 8.1: **Comparing the estimated silicon area, in cm^2, required for implementing SSA and SSD. Network size (number of neurons) is shown on the X-axis.**

shares significant connectivity and processing characteristics with higher–level cortical structures. In those respects, the results found here should not be expected to change. It must be stressed that sensory structures such as artificial retinas and cochleas are different circuits, however. These typically exhibit local connectivity and neural activity characterized by continuously–valued (analog) outputs. An analog vs. digital comparison for such a sensory circuit would have drastically different results than what is described here.

The second goal of this thesis was to introduce a methodology to be used when designing neurocomputers. The idea is to boil neurocomputer analysis down to variations in three component areas: connectivity, neural computation, and weight storage and modification. This technique was followed in the analysis of CNAPS, SSA, and SSD. It is a useful approach because it provides a common language in which different architectures can be compared. It is important because it reveals the ways in which neurocomputer components will have to interface with one another. For example, analog addition is an attractive implementation technique because it offers unparalleled layout density; a single wire can conceivably replace a digital adder circuit requiring hundreds of transistors. Unfortunately, analog addition requires current–mode inputs, which means that synaptic weights must be stored as currents, or in a manner that affords economic conversion to currents. As the discussion in chapter 6 emphasized, such designs for analog storage systems are only in their infancy. Practical neurocomputer systems must have efficient interfaces between the three critical systems.

8.1 Future work

There is no shortage of opportunities for future research in neurocomputer architecture. Topping the list is no doubt the most ambitious item: gaining a deeper, broader understanding of the fundamental mechanisms of nervous system operation. The piriform network model in all of its variations is typical of the type of work required. This model is significant for many reasons, not least of which is that it makes a legitimate attempt to translate the myriad, messy details of neurobiology into an information processing model that can be simulated and analyzed. Additional efforts to link neurobiology and computer science are needed.

As this thesis has illustrated, precision requirements are a central issue in the design of neurocomputer systems. Large hardware arrays are possible only if the precision of individual processors can be relaxed. Most researchers suspect that biological neural networks achieve their prodigious processing power in *spite* of, not *because* of, the precision of individual neurons. The high precision requirements of today's gradient descent algorithms most likely are due to our limited understanding of the true mechanisms of biological learning [Hop90].

If these requirements are relaxed sufficiently, significantly denser digital computational designs are conceivable. For example, a logical AND gate can perform a stochastic multiplication function, possibly substituting for an entire digital multiplier circuit [MCT91]. Such a substitution can only be made if the neural model can tolerate the occasional inaccurate neural response. It underscores the close relationship that must be maintained between research in biological neural networks and neurocomputer implementations.

Research in this area has largely been concerned with the arithmetical precision of neural algorithms and has ignored temporal precision. Some neural models, however, have been proposed [ERAD90] that stress the phase relationship between the firings of neural clusters. It may be that most of the information in a network is to be found in the temporal relationships between neural firings, not in the relative strength of neuron activations. Certainly the piriform model goes some distance down this road in the learning mode synchronization between LOT inputs and the low frequency oscillations knows as the alpha rhythm.

Research in computer hardware for neural networks also has several fertile options. Analog memory designs are one such area. EEPROM is the only viable candidate now, but the problems with programming analog EEPROM cells pointed out in chapter 7 may resist practical solutions. Solid state materials research may someday produce an engineering material that approximates the learning capabilities of biological membranes, but this cannot be expected in the near term.

Finally, the critical interconnection problem can certainly profit from further research. It is unlikely that VLSI processes will ever attain the ability to interconnect millions of widely separated neural elements. However, making large numbers of distant interconnections is a problem for biology as well [Mea89]. Biological structures have evolved that attain sufficient processing power using primarily local connectivity; artificial networks will doubtless require similar techniques eventually. Broadcast communication is used in the CNAPS and SS architectures for distal communication, but it may find its greatest application as a method for local communication in very large networks.

Bibliography

[AIGL90] Jóse Ambros-Ingerson, Richard Granger, and Gary Lynch. Simulation of paleocortex performs hierarchical clustering. *Science*, 247:1344–1348, 1990.

[AKCM90] Stanley C. Ahalt, Ashok K. Krishnamurthy, Prakoon Chen, and Douglas E. Melton. Competitive learning algorithms for vector quantization. *Neural Networks*, 3:277–290, 1990.

[Bak90] Thomas E. Baker. Implementation limits for artificial neural networks. Master's thesis, Oregon Graduate Institute of Science and Technology, May 1990. OGI Dept. of Computer Science and Engineering Tech. Report No. CS/E 90-008.

[BH88] Jim Bailey and Dan Hammerstrom. Why VLSI implementations of associative VLCNs require connection multiplexing. In *Proceedings of the International Conference on Neural Networks*, volume 2, pages 173–180, June 1988.

[BH89] Tom Baker and Dan Hammerstrom. Characteristics of artificial neural network algorithms. In *1989 IEEE International Symposium on Circuits and Systems*, volume 1, pages 78–81, 1989.

[CM89a] H. C. Card and W. R. Moore. EEPROM synapses exhibiting pseudo-Hebbian plasticity. *Electronics Letters*, 25(12):805–806, 1989.

[CM89b] H. C. Card and W. R. Moore. VLSI devices and circuits for neural networks. *International Journal of Neural Systems*, 1(2):149–165, 1989.

[Dil88] Thomas E. Dillinger. *VLSI Engineering.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[ERAD90] R. Eckhorn, H. J. Reitboeck, M. Arndt, and P. Dicke. Feature linking via synchronization among distributed assemblies: Simulations of results from cat visual cortex. *Neural Computation*, 2:293–307, 1990.

[GAIL89] Richard Granger, Jóse Ambros-Ingerson, and Gary Lynch. Derivation of encoding characteristics of layer II cerebral cortex. *Journal of Cognitive Neuroscience*, 1(1):61–87, 1989.

[GAISL89] Richard Granger, Jóse Ambros-Ingerson, Ursula Staubli, and Gary Lynch. Memorial operation of multiple, interacting simulated brain structures. In M. Gluck and D. Rumelhart, editors, *Neuroscience and Connectionist Models*, pages 95–129. Erlbaum Associates, 1989.

[GJ89] Hans P. Graf and Lawrence D. Jackel. Analog electronic neural network circuits. *IEEE Circuits and Devices Magazine*, 5(1):44–49, July 1989.

[GM84] Paul R. Gray and Robert G. Meyer. *Analysis and Design of Analog Integrated Circuits.* John Wiley and Sons, New York, 1984.

[Gra84] Robert M. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, April 1984.

[Hab85] Lewis B. Haberly. Neuronal circuitry in olfactory cortex: anatomy and functional implications. *Chemical Senses*, 10(2):219–238, 1985.

[HB91] Jordan L. Holt and Thomas E. Baker. Back propagation simulations using limited precision calculations. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 121–126, 1991.

[Hop90] J. J. Hopfield. The effectiveness of analogue 'neural network' hardware. *Network*, 1:27–40, 1990.

[HP90a] John Hennessy and David Patterson. *Computer Architecture: a Quantitative Approach.* Morgan Kaufmann, San Mateo, California, 1990.

[HP90b] Paul W. Hollis and John J. Paulos. Artificial neural networks using MOS analog multipliers. *IEEE Journal of Solid-State Circuits*, 25(3):849–855, June 1990.

[HTCB89] Mark Holler, Simon Tam, Hernan Castro, and Ronald Benson. An electrically trainable artificial neural network (ETANN) with 10,240 floating gate synapses. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 191–196, 1989.

[JBHK87] Deog-Kyoon Jeong, Gaetano Borriello, David A. Hodges, and Randy H. Katz. Design of PLL–based clock generation circuits. *IEEE Journal of Solid-State Networks*, 22(2):255–261, April 1987.

[JH88] Mark G. Johnson and Edwin L. Hudson. A variable delay line PLL for CPU–coprocessor synchronization. *IEEE Journal of Solid-State Networks*, 23(5):1218–1223, October 1988.

[KTSL91] Douglas A. Kerns, John E. Tanner, Massimo A. Sivolotti, and Jin Luo. CMOS UV-writable non-volatile analog storage. In *Advanced Research in VLSI: International Conference 1991*, 1991.

[LRMM89] J. Lazzaro, S. Ryckebusch, M.A. Mahowald, and C.A. Mead. Winner-take-all networks of O(n) complexity. In *Proceedings of the Neural Information Processing Symposium*, pages 703–711, 1989.

[LRW91] Todd Leen, Steve Rehfuss, and Max Webb. Encoding and classification in a model of olfactory cortex. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 553–559, July 1991.

[MCT91] Alan F. Murray, Dante Del Corso, and Lionel Tarassenko. Pulse–stream VLSI neural networks mixing analog and digital techniques. *IEEE Transactions on Neural Networks*, 2(2):193–203, March 1991.

[Mea89] Carver Mead. *Analog VLSI and Neural Systems*. Addison Wesley, Reading, Massachusetts, 1989.

[MWC+91] Jack L. Meador, Angus Wu, Clint Cole, Novat Nintunze, and Pichet Chintrakulchai. Programmable impulse neural circuits. *IEEE Transactions on Neural Networks*, 2(1):101–109, January 1991.

[OKH89] Tong-Chern Ong, Ping K. Ko, and Chenming Hu. The EEPROM as an analog memory device. *IEEE Transactions on Electron Devices*, 36(9):1840–1841, 1989.

[She79] Gordon M. Shepherd. *The Synaptic Organization of the Brain*. Oxford University Press, New York, 1979.

[SSK+87] Gheorghe Samachisa, Chien-Sheng Su, Yu-Sheng Kao, George Smarandoiu, Cheng-Yuan Michael Wang, Ting Wong, and Chenming Hu. A 128k flash EEPROM using double–polysilicon technology. *IEEE Journal of Solid-State Circuits*, SC-22(5):676–683, 1987.

[WE85] Neil Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design.* Addison–Wesley, Reading, Massachusetts, 1985.

[Web] Max Webb. Hierarchical competitive learning. Oregon Graduate Institute internal report, 1991.

Appendix A

Area Estimation Program

```
/*********************************************************************
 *                                                                   *
 * STANDARD area parameters for implementation of the piriform model. *
 *                                                                   *
 *********************************************************************/


#define  CHIP  1e8    /* area (in sq microns) of a chip 1 cm per side */
#define  WAFER 12.67e9 /* area (in sq. microns) of the use-able 5 inches
                              of a 6-inch wafer  */
#define  metalPitch 1.4 /* separation (microns) of two adjacent metal
   wires*/
#define  IOCell 2000 /* area of an interface cell to a bi-directional IO
bus signal  */
#define  WTA  150   /* area (per input) of a winner-take-all circuit */
#define  DAC  3500  /* area of a 4-bit current-mode DAC  */
#define  comparator 750 /* area of an analog comparator circuit */
#define  cnState  4000  /* area of latches, gates, and drivers to enable
   a single cn  */
#define  decoder 200  /* area per input of address decoding logic  */
#define  DRAMcell 10  /* area (in sq. microns) of a single DRAM cell  */
#define  SRAMcell 42  /* area (in sq. microns) of a single SRAM cell  */
#define  EEPROMcell 15  /* area of a single EEPROM synapse  */
#define  EEPROMcontroller 20000  /*  area of EEPROM program control
     circuitry. */
#define  twoBitcsa 64  /* transistor count of a 2-bit conditional-sum
```

```
  adder */
#define  DFF  20     /* transistor count of a master-slave flip-flop */
#define  reg  14    /* transistor count of a register in the reg file */
#define  mux  10     /* transistor count of a 2x1 multiplexor  */
#define  LATCH 10    /* transistor count of a D-latch  */
#define  repeater 1500  /* area of one repeater for a digital signal */
#define  obCellArea 2000  /* area of one digital output cell */
#define  deviceDensity 125 /* area (sq. u) per random device */
#define  PLL 2000 /* area (sq. u) of a pll clock generator */



#include <stdio.h>
#include <math.h>


float ssa(), ssd(), PLA(), addrDecoder();
float power();
int   logz();

int   lotSize, patchSize, cellCount, cellSynapseCount, bits, lotBus;
int   adderRegion, patchCount, initLotSize, initCnCount, adderCount;
int   lotStubCount, csaMuxCount, wtMemWidth, adderWidth;
float PNArea, thresholdCompare, connectivity, hiVcktArea, miscRegArea;
float memArrayWidth, memArrayHeight, wtMemArea, PNoutput, kbits;
float compensationArea, wtaArea, repeaterArea, patchLength, lotLength;
float SUBWAFER, IOArea, patchBusArea, lotWidth, lotArea, areaUsed;
float patchBusWidth, PNcontrolArea, xConnectArea, ADC, ADCarea, PLLarea;
float regFileArea, adderArea, dataPathArea, decoderArea;
float memSequencerArea;
FILE  *fp1, *fp2, *fp3, *fopen();

main()

/*  Program to determine the dimensions (# of LOT lines, # of neurons)
 *  of a wafer-scale-implementation of the piriform model.  */
```

```
{
/*  Most of the program's variables are external because both the
    main program and the principal subroutine must have access to them;
    there are too many variables to pass via parameters in function
    calls.  */

  float  area, multiplier, ceiling, floor, slop;
  int    archModel, initLotSize, initCnCount;
  char   areaOK;

/*    Open input and output files    */

  if ((fp1 = fopen("area.infile", "r")) == NULL) {
    printf("  file open failure on area.infile\n");
    exit();
  }

  if ((fp2 = fopen("area.outfile", "w")) == NULL) {
    printf("  file open failure on area.outfile\n");
    exit();
  }

  if ((fp3 = fopen("debugfile", "w")) == NULL) {
    printf("  file open failure on debugfile\n");
    exit();
  }

/*********************************************************************
 *                                                                   *
 *             READ NETWORK PARAMETERS                               *
 *                                                                   *
 *********************************************************************/

  fscanf(fp1, "%*s %*s %d", &archModel);
  fscanf(fp1, "%*s %*s %*s %d", &initLotSize);
  fscanf(fp1, "%*s %*s %*s %d", &initCnCount);
  fscanf(fp1, "%*s %*s %*s %d", &patchSize);
  fscanf(fp1, "%*s %*s %f", &connectivity);
  fscanf(fp1, "%*s %*s %*s %d", &adderRegion);
  fscanf(fp1, "%*s %*s %*s %d", &bits);
  fscanf(fp1, "%*s %*s %f", &SUBWAFER);
  fscanf(fp1, "%*s %*s %*s %f", &slop);

/*     Set up the control parameters for the iterative loop  */

  ceiling = 1.0;
  floor = 0.0;
  multiplier = (ceiling + floor) / 2;
  areaOK = 'n';

/*     Iteratively calculate the size of the largest network that can
       be fit on a wafer.   */

  while (areaOK == 'n') {
    lotSize = initLotSize * multiplier;
    cellCount = initCnCount * multiplier;
    if (archModel == 1)
      area = ssa();
    else if (archModel == 2)
      area = ssd();

printf("another pass ...\n");
    if (area > SUBWAFER) {
      ceiling = multiplier;
      if ((multiplier / 2) <= floor)
multiplier = (multiplier + floor) / 2;
      else
multiplier = multiplier / 2;
    }
    else
      if (area < (SUBWAFER * slop)) {
```

```
floor = multiplier;
if ((multiplier * 2) >= ceiling)
  multiplier = (multiplier + ceiling) / 2;
else
  multiplier = multiplier * 2;
      }
    else
      areaOK = 'y';
  }

/*********************************************************************
 *                                                                   *
 *            PROGRAM OUTPUT                                         *
 *                                                                   *
 *********************************************************************/


  fprintf(fp2,"\nNumber of LOT lines = %4d\n", lotSize);
  fprintf(fp2,"Number of neurons = %4d\n", cellCount);
  fprintf(fp2,"Number of PNs = %4d\n", patchCount);
  fprintf(fp2,"Number of connections = %4d\n",
(int)(lotSize * cellCount * connectivity));

  if (archModel == 1) {

/*   Write summary to area.outfile  */

    fprintf(fp2,"\n          SSA implementation\n");
    fprintf(fp2,"-----------------------------------------\n");
    fprintf(fp2,"LOT stub count = %4d\n", lotStubCount);
    fprintf(fp2,"Memory array width = %4.2f\n", memArrayWidth);
    fprintf(fp2,"Memory array height = %4.2f\n", memArrayHeight);
    fprintf(fp2,"Interconnect area as pct. of total = %4.2f%%\n",
    (lotArea / areaUsed) * 100);
    fprintf(fp2,"Repeater area as a pct. of total = %4.2f%%\n",
    (repeaterArea / areaUsed) * 100);
    fprintf(fp2,"Area of a single patch = %4.2e sq. microns\n",
    PNArea);
    fprintf(fp2,"\nPct. of patch area occupied by subsystems:\n");
    fprintf(fp2,"----------------------------------------\n");
    fprintf(fp2,"   - weight memory: %4.2f%%\n",
    wtMemArea * 100 / PNArea);
    fprintf(fp2,"   - Compensation circuitry: %4.2f%%\n",
    compensationArea * 100 / PNArea);
    fprintf(fp2,"   - EEPROM controller: %4.2f%%\n",
    EEPROMcontroller * 100 / PNArea);
    fprintf(fp2,"   - WTA circuit: %4.2f%%\n",
    wtaArea * 100 / PNArea);
    fprintf(fp2,"   - PN output: %4.2f%%\n",
    PNoutput * 100 / PNArea);
  }

  if (archModel == 2) {

/*   Write summary to area.outfile  */

    fprintf(fp2,"\n      SSD implementation\n");
    fprintf(fp2,"----------------------------------------\n");
    fprintf(fp2,"PN area as pct. of total = %4.2f%%\n",
    (patchCount * PNArea / areaUsed) * 100);
    fprintf(fp2,"Interconnect area as pct. of total = %4.2f%%\n",
    (xConnectArea / areaUsed) * 100);
    fprintf(fp2,"Side dimension of a single square PN = %4.2f
    millimeters\n", PNArea / 1e6);
    fprintf(fp2,"Width of encoded LOT bus: %3d bits\n", lotBus);
    fprintf(fp2,"\nPct. of PN area occupied by subsystems:\n");
    fprintf(fp2,"----------------------------------------\n");
    kbits = cellSynapseCount * patchSize / 1024.0;
    fprintf(fp2," synapses: %4.2f%% (%4.2f kbits)\n",
    wtMemArea * 100 / PNArea, kbits);
    fprintf(fp2,"   - address decoders: %4.2f%%\n",
    decoderArea * 100 / PNArea);
    fprintf(fp2,"   - memory sequencer: %4.2f%%\n",
    memSequencerArea * 100 / PNArea);
    fprintf(fp2,"   - register file: %4.2f%%\n",
    regFileArea * 100 / PNArea);
    fprintf(fp2,"   - adders: %4.2f%%\n",
    adderArea * adderCount * 100 / PNArea);
    fprintf(fp2,"   - LOT bus I/O area: %4.2f%%\n",
    IOArea * 100 / PNArea);
    fprintf(fp2,"   - threshold/compare unit: %4.2f%%\n",
    thresholdCompare * 100 / PNArea);
    fprintf(fp2,"   - PLL: %4.2f%%\n",
    PLLarea * 100 / PNArea);
    fprintf(fp2,"   - PN control: %4.2f%%\n",
    PNcontrolArea * 100 / PNArea);
    fprintf(fp2," Patch Bus: %4.2f%%\n",
    patchBusArea * 100 / PNArea);
  }
  fprintf(fp2,"\nPercentage of SUBWAFER occupied = %4.2f%%\n",
  (areaUsed / SUBWAFER) * 100);
}



float ssa()

/*  Function to calculate and return the area required for the ssa
    architecture (i.e., a fully analog implementation of the SS
    architecture, based on analog EEPROM weight memory and analog
    computation).

    Last revision: 5 April 1991  */

{
  patchCount = cellCount / patchSize;

/* Calculate area of each of the blocks that compose an analog PN:
```

```
- EEPROM weight memory array, including hi-voltage generation
  for programming and erasing
- analog array compensation
- WTA circuit
- PN control, learning, PN state
- PN output unit */

/* The width of each PN is set by the width of the memory array.
The inter-cell pitch set to be equal to the width of a single
cell (which is equal to the square root of the area of a single
cell).  The memory array is very sparse, so most of this area
is empty.  */

/* "lotStubCount" is the number of LOT lines that are contacted by
the cells in the patch.  This is a function of the patch
connectivity and the number of cells in the patch.  */

  lotStubCount =
    lotSize * (1.0 - power((1.0 - connectivity), (float)patchSize));
  memArrayWidth = lotStubCount * sqrt((double)EEPROMcell);
  memArrayHeight = patchSize * sqrt((double)EEPROMcell);

/* Calculate area of each PN's EEPROM weight memory and hi-V generation
   circuit. Hi-voltage circuit area model (20% of synapse area) is based
   on die photographs of commercial flash EEPROM chips.  */

  hiVcktArea = lotSize * connectivity * patchSize * EEPROMcell * 0.2;
  wtMemArea = memArrayWidth * memArrayHeight + hiVcktArea;

/*  The compensation circuitry required by each patch is assumed to be
    the following: a dummy synapse column, an ADC, and a DAC.
    The sum of their areas constitutes the compensation area.  */

  compensationArea = memArrayWidth * sqrt(EEPROMcell) + DAC + ADC;
  wtaArea = WTA * patchSize;          /* winner-take-all circuit  */
  PNoutput = obCellArea * patchSize;
  PNArea = wtMemArea + compensationArea + wtaArea + EEPROMcontroller
+ PNoutput;

/*    Chip interconnect area is equal to the product of LOT length and
      width.  The length is the product of the number of patches (every
      patch must be reached by the LOT) and the width of each patch.
      Width is simply the product of LOT size and metal pitch.  */

  lotLength = patchCount * memArrayWidth;
  lotWidth = lotSize * metalPitch;
  lotArea = lotLength * lotWidth;

/* A signal repeater is used every 10 patches to boost the LOT.  */

  repeaterArea = patchCount / 10 * lotSize * repeater;

  areaUsed = lotArea + repeaterArea + (PNArea * patchCount);
  return(areaUsed);
}



float ssd()

/* Function to calculate and return the area required for implementing
   the SSD architecture (digital implementation of the SS architecture).
*/
{
/*  A PN is composed of:
    - SRAM weight memory & support circuitry (sense amps, buffers, etc.)
    - LOT address decoders (big PLAs)
    - Memory sequencer
    - Digital adder(s)
    - Register file (an accumulator for every cell)
    - Threshold compare unit
    - Miscellaneous registers in the data path
    - PN controller (includes learning state machine)
    - PLL clock generator

    First, compute network--wide information: the size of the LOT, the
    number of PNs, the number of adders required per PN, and the number
    of weight memory sites (synapses) per cell.  */

  patchCount = cellCount / patchSize;
  lotBus = logz(2.0, (float)cellCount);   /* Width of the encoded LOT */
  adderCount = patchSize / adderRegion;   /* adders per PN */
  cellSynapseCount = lotSize * connectivity;

/*  Calculate the area of weight memory, LOT decoders, and
    support circuits  */

  wtMemWidth = bits + 1;
  wtMemArea = cellSynapseCount * wtMemWidth * SRAMcell * patchSize
      * 1.5;
  decoderArea = patchSize * addrDecoder(logz(2.0, (float)lotBus),
cellSynapseCount, 0.8);

/*  The memory sequencer is composed of a shift register for
    sequentially addressing each memory location.  The shift
    register's length is equal to the number of connections per
    cell (cellSynapseCount).  Device density for the registers
    is slightly higher than for other circuits.  */

  memSequencerArea = cellSynapseCount * DFF * 0.75 * deviceDensity;

/*  Calculate the area of the processor OUTSIDE the weight
    memory */

  IOArea = IOCell * lotBus;    /* One I/O cell per LOT line  */

/*  Assume each adder is twice as wide as a synaptic wt. */
```

```
  adderWidth = 2 * bits;
  csaMuxCount = 3.5 * bits;
  adderArea = ((twoBitcsa * adderWidth / 2) + (csaMuxCount * mux) +
       (2 * DFF * adderWidth)) * deviceDensity;

/*  The register file contains a register for every cell in the
    PN.  Dual-ported register file, which wipes out the density
    advantage of the very-regular registers. */

  regFileArea = (wtMemWidth * 2) * reg * deviceDensity * patchSize;
  miscRegArea = (wtMemWidth * 2) * reg * deviceDensity * 3;

/*  The threshold compare unit contains logic to compare a
    binary value against various threshold values, and to
    compute the winner-take-all selection. Assume it is as large
    as a single adder.  */

  thresholdCompare = adderArea;
  dataPathArea = IOArea + (adderCount * adderArea) + thresholdCompare
    + regFileArea + miscRegArea;

/*  Each phase--locked loop (one per PN) requires approximately
    220 transistors to implement.  */

  PLLarea = 220 * deviceDensity;

/*  Assume that the area of the PN controller is equal to 1/8th
    the area of the PN's data path. */

  PNcontrolArea = dataPathArea / 8.0;
  PNArea = wtMemArea + decoderArea + memSequencerArea + dataPathArea
   + PNcontrolArea + PLLarea;

  xConnectArea = lotBus * metalPitch * sqrt(PNArea) * patchCount;
  areaUsed = (PNArea * patchCount) + xConnectArea;
  return(areaUsed);
}



float PLA(inputs, outputs, width)

/*  Return an estimate of silicon area (in square microns) of a PLA  */

int    inputs, outputs, width;
{
  float    area;

  area = (45 + (8 * width) * (inputs) + (5 * width) * (outputs)) *
    ((58 * width) + (5.3 * width) * inputs) * 0.85;
  return(area);
}



float addrDecoder(LOTinputs, synapsesPerCn, lineWidth)

/*  Procedure to estimate the area of a PLA-like address decoder  */

int   LOTinputs, synapsesPerCn;
float lineWidth;
{
  float  height, width, area, bufferSize, perimeter;

  perimeter = 10.0 * lineWidth;
  bufferSize = 27.0 * lineWidth;
  height = perimeter + bufferSize + synapsesPerCn * 4 * lineWidth;
  width = perimeter + bufferSize + LOTinputs * 7.0 * lineWidth;
  area = height * width;
  return(area);
```

```
}


int logz(z, x)

/*  Return the (ceiling) value of the logarithm to the base z of x  */

float z, x;
{
  double  y;

  y = log(x) / log(z);
  if ((y - (int)y) < 0.0001)      /*  i.e., if y is an integer  */
    ;
  else
    y += 1.0;
  return((int)y);
}



float power(x, y)

     /*  Raise x to the power y;  x and y must be floating point
 numbers.  */

float x, y;
{
  float  z;

  z =  exp(log(x) * y);
  return(z);
}
```

Biographical Note

The author was born on March 19, 1960, in Spokane, Washington. He attended schools in the states of Washington, Idaho, and Montana, eventually graduating from Monroe High School in Monroe, Washington, in 1978. He then entered Washington State University in Pullman, Washington where he received the Bachelor of Science degree in electrical engineering in June 1984. His university studies included one year abroad attending the University of Stirling in Stirling, Scotland.

In July 1984 the author began working for Gould AMI in Pocatello, Idaho as a hardware engineer designing custom CMOS integrated circuits. He worked for Gould AMI from the period July 1984 through December 1987.

The author then began study at the Oregon Graduate Institute where he completed the requirements of the Master's Degree in Computer Science and Engineering in September 1991. His last two years at OGI were spent as a part time student while working full time for Adaptive Solutions, Inc. of Beaverton, Oregon.

The author has been married for three years to Carol Ungate.

He is currently working for Adaptive Solutions as a hardware engineer.