A VLSI ARCHITECTURE FOR A NEUROCOMPUTER USING HIGHER-ORDER PREDICATES

Ronnie Dee Geller
B.A., Reed College, 1979

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

May, 1987

The thesis "A VLSI Architecture for a Neurocomputer Using Higher-Order Predicates" by Ronnie Dee Geller has been examined and approved by the following Examination Committee:

Dan Hammerstrom
Associate Professor
Thesis Research Advisor

Richard B. Kieburtz
Professor

Richard Hamlet
Professor

Stuart Hawkinson
Floating Point Systems

TABLE OF CONTENTS

List of Figures iv
List of Tables v
Abstract vi
1. Introduction - A Biological View 1
2. A Proposal for a Specialized Digital Architecture 7
3. A Microarchitectural Implementation 21
4. Technological Feasibility 38
5. Performance Analysis 48
6. Initialization, Computing and Learning 65
7. Summary and Conclusion 72
References 77

LIST OF FIGURES

1. A Depiction of a Stereotyped Neuron Model 3
2. A Diagram of a Connection Node 8
3. A Likely Configuration Of a Connection Computer 18
4. A High Level Block Diagram of the PN with Control Signals 24
5. A Block Diagram of the Input Buffer and Routing Logic 25
6. A Broadcast Hierarchy Transmission Packet 26
7. A Block Diagram of the Update Products Logic 30
8. A Block Diagram of the Update Sum Logic 33
9. A Block Diagram of the Output Buffer and Routing Logic 34

LIST OF TABLES

1. The Number of Addressable CNs and PNs at each BH Level 14
2. An Itemized Summary of the PN's External Memory Requirements 39
3. Silicon Area Information of the Principal PN Logic Cells 45
4. Some Examples of Estimated PN Performance 60

ABSTRACT

A VLSI Architecture for a Neurocomputer
Using Higher-Order Predicates

Ronnie Dee Geller, M.S.
Oregon Graduate Center, 1987

Supervising Professor: Dan Hammerstrom

Some biological aspects of neural interactions are presented and used as a basis for a computational model in the development of a new type of computer architecture. A VLSI microarchitecture is proposed that efficiently implements the neural-based computing methods. An analysis of the microarchitecture is presented to show that it is feasible using currently available VLSI technology. The performance expectations of the proposed system are analyzed and compared to conventional computer systems executing similar algorithms. The proposed system is shown to have comparatively attractive performance and cost/performance ratio characteristics. Some discussion is given on system level characteristics including initialization and learning.

1. INTRODUCTION - A BIOLOGICAL VIEW

Modeling biological intelligence by mimicing neural interactions is a field of computer science that could yield significant practical results and scientific insights. A completed technology of this form could greatly enhance the pursuit of computer based artificial intelligence. It is also likely that efforts in this direction could provide information for neural scientists on how neurons group together to provide perception, motor functions and intelligent behavior. This thesis addresses these topics and proposes a computer architecture which is well suited to modeling large neural networks.

The work presented here actually represents only a portion of broader research interests being pursued at Oregon Graduate Center (OGC) [Ham86a]. The computer architecture presented in this thesis uses a communication structure which is being developed in a parallel effort at OGC [Bai85]. The communication portion of the proposed architecture is not considered in this work.

As this thesis is grossly based on a simulation of neurons and neural networks, it is important to explicitly define the limits of the accuracy and intent of this simulation. The ultimate goal of this research is to design a computer architecture that allows much more computational parallelism than is possible with other computer systems. An architecture of this sort would lend itself well to applications that are not efficiently solved using essentially sequential computing. Target applications

tend to have large numbers of "soft" constraints and these include image recognition and natural language understanding.

To accomplish these goals, a computer architecture will be presented which derives its computational attributes from biologically based neural systems. There are several reasons for the choice of neural networks as a basis. One is that neural networks are capable of supporting tremendous amounts of computational parallelism. Furthermore, life-forms are capable of performing the cognitive functions which are the goal of this research area. Several researchers have used computer simulations of neural networks to obtain promising results. These include pattern completion by Hopfield [Hop82] and generalized learning by Rumelhart and Hinton [RHW85]. Sejnowski and Rosenberg have designed and built a system that has learned to convert text to speech [SeR86]. Finally, previous research at OGC has validated the usefulness of the specific neuron-based model used here [1] [Ham86b]. Rumelhart and McClelland have covered this general research area carefully [RuM86]. A complete bibliography of research in this field has been prepared by Hammerstrom [Ham86c].

The following paragraphs present some biological information on the neural model used in this research. Also defined are some relevant biological terms which might otherwise be foreign to computer scientists. Every attempt has been made to present this information as accurately as possible without getting into unnecessary details. The references provide these details to the interested reader.

---

[1] As will be seen later, the proposed model differs from its biological counterpart in many substantial ways. In fact, it is best to regard the computational model as an oversimplified and stereotypical view of a real system that is otherwise much too complicated and diverse.

Although biological evidence indicates that there are many different kinds of neurons with a tremendous amount of differentiation between them, there are also similarities regardless of the neuron type or its specific biological function [KuN77]. In general terms, any neuron can be thought of as the composition of four distinct components - the *dendrites*, a *cell body*, an *axon* and the *presynaptic terminals* of the axon [KaS81]. The dendrites of the neuron act as its input sensors. The axon, coupled with its presynaptic terminals, provide the neuron's output capabilities [2]. *Signals* are passed from the axon of one neuron to a dendrite of another neuron across a *synapse*. Incoming signals are passed from the dendrites to the cell body where they are converted into an output signal. In biological terms this conversion is referred to as *integration*. Figure 1 depicts a neuron and illustrates the stereotypical features described above.

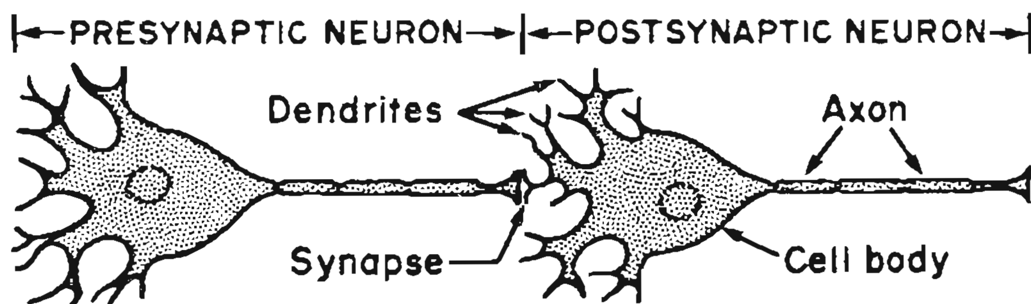

Figure 1 - A Depiction of a Stereotyped Neuron Model With Nomenclature Definitions.

[2] Although a differentiation between the axon proper and its presynaptic terminals may be important to the biological understanding of a neuron, referring to them separately in this thesis would cause needless confusion. Therefore, future references to the axon will include the presynaptic terminals unless differentiation is required and explicitly stated.

Although the communication scenario presented above is accurate in its simplicity, there are several important points where clarification is required. First, it must be noted that the biological mechanisms used to support communication between neurons is incredibly complex. The communication path actually involves several stages of transformations both within the neuron and at the synapse. These transformations typically involve electrical and chemical interactions that are still not completely understood. Further, there is much differentiation between the specific communication mechanisms used by different types of neurons. As a result, some neural scientists are skeptical of the value of any neural simulation that only allows one specific mechanism for neuron communication. Their claim is that the diversity in communication mechanisms is a necessary component of biologically based cognitive behavior. Arguments of this nature must be deferred until more results in this research area are available.

Although there are major differences between specific communication mechanisms used by different neurons, most neural scientists would agree that there are some characteristics that are easily stereotyped and readily understood [KaS81]. In this stereotypical view, there are several points worth noting for future reference. Interneuron communication occurs at a synapse between the axon of the *presynaptic* neuron and the dendrite of the *postsynaptic* neuron. There are several energy transitions that occur during this communication. These include generation of an *action potential* which is electrical in nature and is transmitted along the axon of the presynaptic neuron. An action potential causes a *secretory potential* by releasing chemical transmitters into the synaptic region. The dendrite of the postsynaptic

neuron reacts to these chemical transmitters by generating an electrically based *synaptic potential.* A synaptic potential may be either *excitatory* or *inhibitory* in its effect on the cell body. There is considerable controversy in the biological neural sciences over the importance of the dendrites role during complex interneuron communications [KPT82, Per83]. Nonetheless, it is clear that the dendrite's role as a functional input site could represent a significant portion of neural systems processing power.

Synaptic potentials from a neuron's dendrites are carried to the cell body where they are integrated, both spatially and temporally, to determine whether an action potential should be generated on its own axon. The integration performed by the cell body can be viewed as a weighted algebraic summation of its individual synaptic potentials. If the result of the integration is above a *threshold,* then the cell body will cause its axon to *fire,* i.e., generate an action potential. The exact details of the integration, including the assignment of weights to each of its inputs, depends greatly on the type of neuron, its biological function and its previous history.

The intensity of secretory and synaptic potentials depends on the magnitude, duration and timing of their stimulus. In this manner, their behavior may be accurately viewed as a time integration of analog input signals. However, biological evidence indicates that action potentials are of an all-or-none nature with the amplitude and duration roughly fixed for any individual neuron. As a result, stimulus intensity information is only conveyed by the number of action potentials generated and the time interval between the potentials.

Beyond the microscopic details of a neuron, there is some higher level biological information that requires examination. A human brain contains on the order of one trillion neurons (10^{12}) with about one thousand different neuron types [KaS81]. The function of a neuron depends on both its biological type and the functions of the neurons that are connected to it. A neuron may receive inputs from as many as 10,000 to 80,000 other neurons and its output may affect a similar number. The cycle time of a neuron, from receiving its input to generating its output, is in the order of 2 to 5 milliseconds [Pos78]. Certain interesting behavior that uses large subsystems of the nervous system, such as simple image recognition, requires only about one half a second to complete. Other more complicated cognitive tasks may take more time. Nevertheless, it is obvious that macroscopic response times of this magnitude are impressive when the basic response time of a neuron (about 10^6 slower than a transistor!) is considered along with the incredible number of neurons that are involved. These facts indicate that the neural systems utilize significant concurrency and little sequentiality in solving problems. In summary, response times of this sort represent performance efficiency that will be carried over to the neural model based computer architecture presented in later chapters.

2. A PROPOSAL FOR A SPECIALIZED DIGITAL ARCHITECTURE

As with all models, it is necessary to differentiate between important elements of the real system and complicating details that may be ignored. In this case, there must be defined a digital model of a neuron that contains enough of the neuron's fundamental properties that interesting, if not intelligent, behavior could be expected. Complicating features can be ignored if they are peculiar to the biological nature of neurons and are not critical to the macroscopic functions desired. Finally, there are some biological details that are important to mimic but are not suited to direct implementation using digital computer technologies. In these cases, functionally similar counterparts must be derived and substituted into the model.

The model used in this research defines the *Connection Node* (CN) as the logical component which corresponds to an individual neuron. A group of Connection Nodes are combined together into a network to form a system called a *Connection Computer* (CC). In some ways, a connection computer's capabilities may be closer to those of a nervous system than those of current computers. In this way, the connection computer could perform certain cognitive tasks that are not solvable with the computers of today. While keeping these goals in mind, the implementation details of the CN and CC are given below. As some of the attributes of the CN and CC are roughly based on biological neural networks, some comparisons of the two systems are given.

As with neurons, the CN is complicated enough that its description is more easily presented if it is segmented into separate functional units. One of the most fundamental of these is the communication facility which allows a CN to receive input and generate output. As depicted in Figure 2, a CN accepts digital values as input and transmits a digital output. As with a neuron, a CN accepts inputs from many sources and integrates these to generate an output value. Unlike neurons, the CN's signals are represented by discrete digital values rather than analog signals. It is likely that the digital implementation of the CN is acceptable as it satisfies the functional requirement of conveying intensity information. Another reason for using digital outputs is that efficient transmission across a single carrier is only possible with a digital representation of the data. An analog transmission scheme would not allow reasonably efficient time-multiplexed use of the transmission medium.

To extend this comparison, the CN uses the value of the digital signal as the *only* indication of intensity whereas neurons rely heavily on the inter-"fire" interval to denote intensity. From a more specific perspective, our CNs use eight bits of

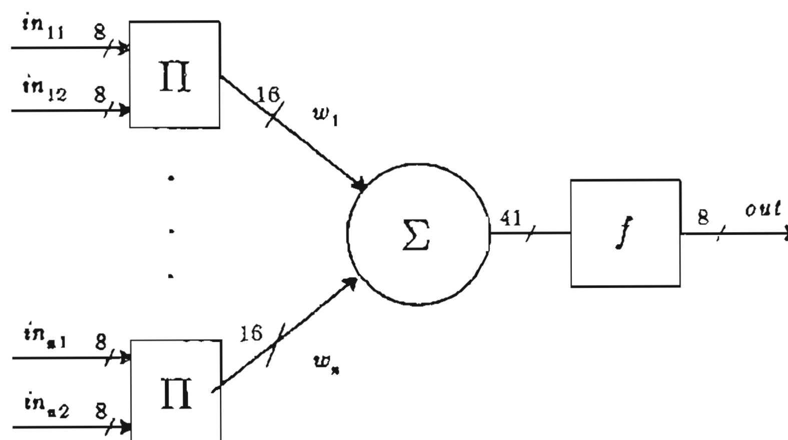

Figure 2 - A Diagram of a Connection Node (CN).

information for input and output values. The granularity of communication accuracy is probably greater than that of real neurons. Initial simulations of this specific model at OGC shows that this quantization is more than acceptable for the target applications [Ham86b].

The second major functional portion of the CN requiring explanation is its method of integration to generate an output value from its input values. The general function used is called a Sigma-Pi function. This name is derived from the fact that the function is a sum (Sigma) of intermediate products, Pi terms. Specifically, the output of a CN is derived from its inputs as given by Equation 1.

$$out = f\left(\sum_{k=1}^{n} w_k \left(\prod_{j=1}^{2} in_{kj} \right) \right) \qquad \textit{Equation 1}$$

In Equation 1, in_{kj}, and w_k are 3-tuple inputs which are used to calculate the output, *out*. Each in_{kj} represents an input signal into the CN and it is typically an output of some other CN. The product of the two inputs, in_{k1} and in_{k2}, is referred to as a *2-codon* and it is an intermediate value used during the computation. The 2-codon is a particular type of ***higher-order predicates*** [3] and both of these terms are explained below. Each 2-codon is multiplied by a 16-bit weight, w_k, before being submitted to the summation function shown in Equation 1. After the summation is complete, a firing function, *f*, is applied that converts the sum to the output value. Throughout these computational processes, full precision is retained by increasing the widths of the data paths as appropriate.

---

[3] The word ***predicate*** is used here for its meaning as a term designating a *relationship* and not for its formal logic meaning that is familiar to computer scientists.

Although this Sigma-Pi construct may appear to be rather specific in its application, it is actually quite general. In fact, some key features used by other neural models, such as threshold and residual state, can be easily handled with this general computational formulation.

The specific firing function used in this research is a simple binary division which is easily performed by just extracting the upper eight bits from the resultant sum. This is a special case of the more general non-linear *sigmoid* function that is most often used in this context [RuM86]. The microarchitecture presented in the following chapter relies on the choice of f as the firing function, but the more general architecture of the CN does not, as it uses the more general *sigmoid* function. Furthermore, if subsequent research shows a need for a more powerful firing function, this could be accomplished with only minimal changes to the proposed microarchitecture.

The computation method given in Equation 1 is a special case of r-codons as formulated by Marr [Mar70, MGL86a, MGL86b, MiP69]. This computational strategy is based on hypotheses of the biological mechanisms used in neural systems to encode and process information [4]. Specifically, there are several areas where the CN functions are analogous to those of neurons. The most obvious of these similarities is the use of a weighted sum method for performing the integration function. In this context, the mathematical summation of the CN precisely corresponds to a similar function performed by the cell body. Further, weighting the inputs to the summation is very similar to the function performed by the connection between the dendrites and

[4] As a result, a secondary benefit of this research area will be to test the validity of these biological theories.

the cell body. In the biological mechanism, the electrical resistance between the dendrite and the cell body depends on the physical dimensions of the dendrite [KaS81] and this resistance greatly determines the weighting factor associated with each dendrite. The weighting of inputs also may correspond in function with the temporal and spatial facilitation that occurs at a synapse.

The use of higher-order predicates is another case where the CN is similar to a neuron. The alternative to using higher-order predicates would be basing the CN's computations exclusively on the weighted sum of its inputs. In biological terms, this alternative would correspond to a neuron whose input processing consists of only simple signal integration at the cell body. This simplified neuron model would lack all input processing and correlation functions that are associated with the neuron's dendrites. As the dendrites role is critical to interneuron communication [KPT82, Per83], omitting it completely from the CN model would be unwise. In fact, Marr shows that higher-order predicates, in the form of r-codons, are consistent with biological theories of neuron based information processing [Mar70]. Maxwell et al. [MGL86b] and Minsky and Papert [MiP69] have discused the functional advantages of higher-order predicates from a computer science perspective.

The particular type of higher-order predicate used in the CN model is the 2-codon and it is a two input r-codon. The use of 2-codons, rather than 3-codons or even some different type of higher-order predicate, was chosen as a result of simulations at OGC of target application requirements [Ham86b].

Once the integration is complete, the firing function generates a digital output. In the neural system, the actual firing conveys little information as its intensity

is roughly fixed for any individual neuron. As a result, neurons must convey intensity information by the relative times at which they fire. The underlying mechanisms used by neurons rely heavily on analog processing and synchronization of time critical events. As the CN model does not support these capabilities, the digital gradation of outputs must suffice for expressing intensity information.

Now that the internal aspects of the CN have been presented, certain system issues of the CC must be considered. As with a biological nervous system, it is important that the CC have sufficient connections that efficient parallelism may be accomplished. On the other extreme, it is obviously impossible for each CN to be connected to every other CN in the CC. These two opposing factors suggest the need for a communication scheme that is a mixture of acceptable connection richness and technological viability. As mentioned earlier, parallel research at OGC is directed towards defining a global communication mechanism for the CC which will support rich connections and fast communications in very large systems [BaH86]. Bailey and Hammerstrom have shown that conventional networks based on such constructs as direct connections, nearest neighbor, hypercube, shared memory, etc., are unacceptable for the very large systems under consideration at OGC. Furthermore, they have proposed a communication scheme called the *Broadcast Hierarchy* (BH) that satisfies the requirements for the target systems [5]. This research uses their proposal of the BH and relies on their rationale. Therefore, this thesis provides only enough relevant information on the BH to establish its use and function within the CC. Beyond this, it will be shown specifically how the CN supports communication

[5] The BH construct as used in this thesis is derived from ongoing research at OGC. As the BH definition is evolving, related implementation details may slightly conflict with current and future definitions used by other

using the BH construct.

Before presenting the details of the BH, it is important to discuss a technological constraint on the CC design that affects its BH implementation. Because a CN is such a simple computational element and many of its functions relate directly to I/O requirements, it is useful to group together many CNs into a single physical entity. Combining several CNs together allows efficient utilization of VLSI technology and lowers the number of physical circuit board connections required. To accomplish this physical grouping, 64 CNs are combined together to form a *Physical Node* (PN). Within a PN, communication between its CNs is accomplished by methods that are consistent with standard intrachip VLSI methods and, therefore, the BH construct does not strictly apply at this internal level [6]. The exact details of the intrachip CN communication facilities are presented in the following chapter which defines the PN microarchitecture.

It should be remembered that the PN construct is just a technological necessity for grouping together many CNs. The existence of the PN as a physical entity forces frequent references to it even though the PN's computational properties are completely defined by its emulation of the *virtual* CNs.

The BH networking method is a mechanism of connecting many different PNs together while minimizing physical connections, communication latency and addressing overhead. This is accomplished by segmenting PNs into logical groups that communicate with each other on a Broadcast Hierarchy *level*. In the specific form of the

---

researchers at OGC.

[6] It could be argued that the internal communication facilities actually are the lowest level of the Broadcast Hierarchy. This argument has practical and aesthetic merit but is not pursued here.

BH used in this research, each PN is a member of three different BH levels and the PN supports complete parallel utilization of all three levels concurrently. Level 1 connects four PNs, level 2 connects 32 and BH level 3 connects 128 PNs to each other. These BH level sizes were chosen as a result of the previously cited initial simulations of this system by Bailey [BaH86]. The simulation results indicate that target applications require each CN to have direct communication access to approximately 1,000 other CNs. As can be seen from Table 1, the sizes chosen for the three BH levels are satisfactory in this regard.

The method of grouping PNs together could have significant affect on PN performance and its inherent fault tolerance. The exact details of the grouping are not pursued in this thesis as they are more intimately tied up with details of the communication theory of the CC. Nonetheless, some of the characteristics of the grouping are discussed here as they relate to the details of PN implementation. As lower levels would have less contention, communication latency would tend to be shorter at the lower levels than at the higher ones. As a result, CC algorithms would be able to specify that communications between any two PNs would be accomplished at

BH LEVEL	*NUMBER OF PNs*	*NUMBER OF CNs*	*REQUIRED BITS IN ADDRESS FIELD*
INTERNAL	*1*	*64*	*6*
1	*4*	*256*	*8*
2	*32*	*2048*	*11*
3	*128*	*8192*	*13*

Table 1 - The Number of Addressable CNs and PNs at each BH Level.

the lowest possible BH level. This choice could tend to keep concentrated communication traffic on the smaller, local levels and keep them off the more competitive higher levels. This preferred use of lower levels would minimize contention problems at higher BH levels that would otherwise become prohibitive in very large systems. As communication "locality" is typical in neural networks, the preferred use of lower BH levels is natural.

The *logical* communication mechanism used by the BH is level independent, but there could be variation in the *physical* connection method depending on the number of PNs connected. Logically, each BH level must have an address space within which each definable CN has a unique address. Table 1 lists the size of the different levels and the related addressing requirements. To simplify the implementation, it is useful to define the addresses of all of the CNs within a PN sequentially from a base address. In this way, the PN would only need to be given its base address at each BH level to define the addresses of each of its CNs.

Whenever a CN *fires*, the parent PN takes the eight bit output of the CN and appends the CNs unique address onto the data to create a BH communication packet for each of the three BH levels. The PN then simply *broadcasts* the appropriate packet onto each level using the physical communication facility provided at that BH level. All PNs connected on the specific BH level receive this packet and are responsible for looking at the CN address of the originator to determine if they have any CN which relies on the data message. If there is no reliance, the PN may simply discard the packet. If there *is* a reliance, the PN must retain the input data and then update the output of its own affected CNs. This protocol is referred to as

"come-from" addressing, as the address of the sender is specified and not the address of the recipient.

This logical protocol shows the elegance and simplicity of the BH construct when used in very large systems. In particular, the broadcasting of all messages onto a common carrier is an apt choice if it is likely that a packet will be used by multiple PNs on a BH level. This likelihood is consistent with biological evidence of neural based information processing. Besides, an application would be formulated for execution on the CC with this efficiency characteristic in mind.

Actually, broadcasting on all levels would be unnecessary and inefficient in a pure Broadcast Hierarchy. By broadcasting on multiple levels, the PN is capable of supporting the more general case where there is overlap between BH levels. A more efficient implementation (that yields the same flexibility) uses a programmable *Broadcast Control Field* to specify which BH levels are to be used for each CN output. The Broadcast Control Field is not supported in the definition of the PN given in this thesis, but it could be easily added in the future.

The choice of *physical* connection facilities used at each level could depend on the number of PNs connected, their proximity and other environmental factors. For example, it is possible that at the lower levels a fully arbitrated, parallel transmission, shared bus could be used. At the higher levels, some sort of serial transmission, non-arbitrated network would be used. As network protocol methods are subject to considerable controversy among specialists, this thesis will not propose a precise method for communication. Instead, it will be assumed that *some sort* of a serial transmission will be used and it will be the same choice for each BH level.

From a biological perspective, the Broadcast Hierarchy is attractive in several ways. One is that is supports fast local communications between PNs at the lower BH levels. This is similar to communication between neurons that are directly connected. Besides fast local communication, the BH provides a network with high fan-out and fan-in that allows many CNs to indirectly (but quickly) communicate, with even logically distant CNs. Therefore, a carefully formulated group of overlapping BH levels could provide connection richness similar to that of biological neural systems. If this richness were accomplished, it is likely that any two CNs could indirectly communicate with only a couple of intermediary CNs required to propagate a message. These simple indirect message passing abilities could easily be programmed into a CC application as shown in a later chapter. This relatively straightforward global communication capability provided by the BH closely correlates with similar biological functions.

We next show how the constructs defined above are combined to design a real computer system. Figure 3 shows a hypothetical configuration of a CC circuit board that contains 128 PNs. It is entirely possible that a realistic CC could be composed of 32 of these boards providing a system of over 250,000 CNs with approximately 43 million logical connections.

Although these numbers may be technologically aggressive, they do not represent overly optimistic expectations as integrated circuit packaging techniques are rapidly improving. Furthermore, the regularity and locality of the inter-chip communication requirements could make this proposed system easily realistic. Finally, it is anticipated that the research at OGC will ultimately evolve into a

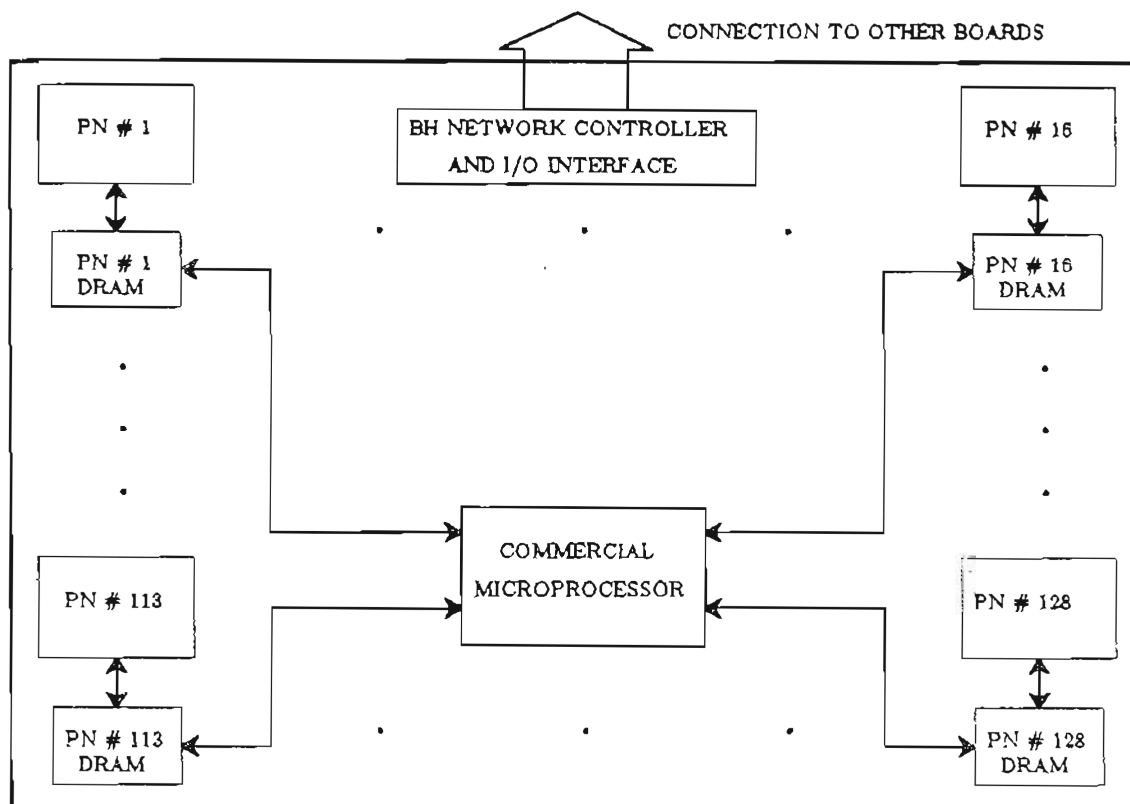

Figure 3 - A Likely Configuration of a Connection Computer. With 64 CNs per PN, this would allow 8192 CNs per board. The microprocessor is used during system initialization and to execute learning algorithms. A specialized Broadcast Hierarchy network controller and I/O interface may be used.

Wafer Scale Integration (WSI) implementation that would require some architectural changes from the CC presented in this thesis. As a result, the technological aggressiveness of this proposal does not introduce significant risk into future research. Further discussions of the technological, academic and economic viability of the proposed system will be presented in later chapters.

The three non-PN logic components shown in Figure 3 play crucial roles in the CC architecture. An explanation of their functions sheds some light on how the CC operates. Briefly, these are used primarily in system initialization, operational computing and the execution of learning algorithms. The initialization consists of the

microprocessor loading the DRAM of each PN with the numerical information required to define the problem to be solved. As will be seen in the following chapter, this information must include a definition of which 2-codons will be used and the specification of all required weight terms. Once the DRAMs are initialized, the PNs may be started and they will begin to generate outputs. For communication between PNs, on-board references are passed directly between PNs but off-board references must be passed through the I/O controller on each board. The I/O controller also performs the DRAM refresh. In many applications, the CC will "complete" its computation and the microprocessor may be called upon to execute a learning algorithm. In other applications, learning may be performed as a background job while the PN computes. A more detailed discussion of topics relating to initialization, computing and learning will be presented in a later chapter.

Although this proposed system will have approximately seven orders of magnitude less CNs than a human brain has neurons, it is still likely that the CC could perform useful functions that are not possible with current computers. For example, NETtalk uses only 300 nodes [SeR86]. The CC could also perform some functions that are already possible but at a vastly improved cost/performance ratio. Nevertheless, the brain provides a *very* general processing mechanism that is capable of such diverse functions as sensory analysis, memory, rational thought, emotional behavior, reflex actions and instinctual behavior such as propagation of the species. The expectations of the CC's abilities are well below those of human intelligence. In fact, the initial CC design would probably be considered successful if it were capable of just one fairly trivial function such as *learning* to reliably recognize subtle visual

patterns in different contexts. An alternative task could be the real-time recognition of several words spoken continuously. In general, any computational problem that could profit from significant parallelism would be a target application for the CC.

No proposal for a radically different computer would be complete without showing why the proposal is even required. In other words, it is important to establish why currently available computer architectures are not acceptable for the execution of the target applications. It is becoming increasingly clear that conventional von Neumann computers cannot support the computational parallelism required to solve complicated artificial intelligence problems. To circumvent these limitations, this thesis presents a computational architecture that is capable of tremendous algorithmic parallelism. A later chapter presents some calculated performance expectations for the CC and compares them with those expected from more conventional computers executing the same algorithm.

3. A MICROARCHITECTURAL IMPLEMENTATION

It is now possible to define a microarchitecture that implements the digital model presented above. This chapter presents a microarchitectural proposal for a VLSI based digital chip that implements an individual PN and provides a building block for the *Connection Computer.*

This thesis gives the initial proposal for a computer architecture that is totally different from any other computer system. Furthermore, brand new computational methods are being developed for this architecture that are *fundamentally* different from those of conventional computer systems. As a result of these two factors, it has proved impossible to completely resolve every technical question encountered during the architectural definition. On the other hand, this thesis explicitly points out the questions that require resolution before this architecture is finalized. In these situations, the microarchitecture is defined in such a way that resolution of the questions affects only design parameters and not the fundamental architectural structure. As a result, the structure presented here should be flexible enough to accommodate future research results.

Before describing the details of the microarchitecture, it is important to point out two important implementation decisions that cause significant repercussions. The first of these decisions is to use external Dynamic Random Access Memory (DRAM) to satisfy the high-capacity memory requirements of a PN. As will be

shown later, each PN (as defined in this chapter) requires on the order of two megabits of memory. As these requirements border on the capacity limits of commercially available DRAM, it is clear that it would be unlikely to include this amount of memory on-chip in the PN. Therefore, the decision to use external memory was made even though this solution is not optimal in many ways. In fact, several fundamental characteristics of the microarchitecture of the PN differ from those that would be expected if the memory were internal. In some cases, specific logic elements require actual trade-offs to accommodate the external memory decision and these are discussed below as appropriate. Some discussion is also provided on how the microarchitecture could be enhanced in the more ideal situation where internal memory is feasible.

The second topic that requires resolution before actually launching into the discussion of the microarchitecture, is the definition of some system level details of the PN as they relate to memory use. As described in the previous chapter, a PN implements 64 CNs and uses three levels of the Broadcast Hierarchy. The different BH levels support either 4, 32 or 128 PNs depending on the level. These system parameters imply that the PN must be capable of accepting input from 10,560 unique CNs. This number may be derived from Table 1 by adding the number of CNs possible at each BH Level. This total includes every input CN separately even though this results in replicated storage, because in true hierarchical structures inputs are duplicated on different BH levels. This replication of storage for each input CN supports the worst case grouping in which inputs on lower levels are not present in higher levels.

OREGON GRADUATE INSTITUTE LIBRARY

In the proposed design, the PN allocates data buffer memory for each of the defined CN inputs, even though it is likely that most applications would not use them all. Other memory allocation methods were considered that are more conservative of memory use in the situation where many CNs are unused. Unfortunately, none of these was feasible as the savings in memory use was easily offset by the increased complexity of addressing logic. This trade-off between regularity and memory requirements occurs throughout this design and is generally resolved in a similar manner.

Although the PN reserves storage for each CN input, it is not possible for it to reserve storage for each 2-codon. As a result, the *2-codon Products Table* and the *CN Weight Tables* shown in Figure 4 must be limited in size. If the sizes of these tables were not restricted by the implementation, it would be possible that the PN would require all possible 2-codons to be generated. Furthermore, in the extreme, it would be possible for each of the 64 CNs to have a Weight Table with an entry for each of the possible 2-codons. This scenario is clearly infeasible as the number of possible 2-codons is in the approximate order of the square of the number of input CNs. This would imply the need for over 100 *megabytes* of memory just to store the entire 2-codon Products Table. As a result, the 2-codon Products Table is limited in size to 8192 entries and the Weight Tables are limited to 512 entries per CN. Limiting table sizes is possible because neural networks exhibit significant locality in their computational references. Ongoing research at OGC indicates that the proposed limits are consistent with application requirements [Ham86b]. Furthermore, the actual table sizes are *not* critical to the microarchitecture of the PN and could be

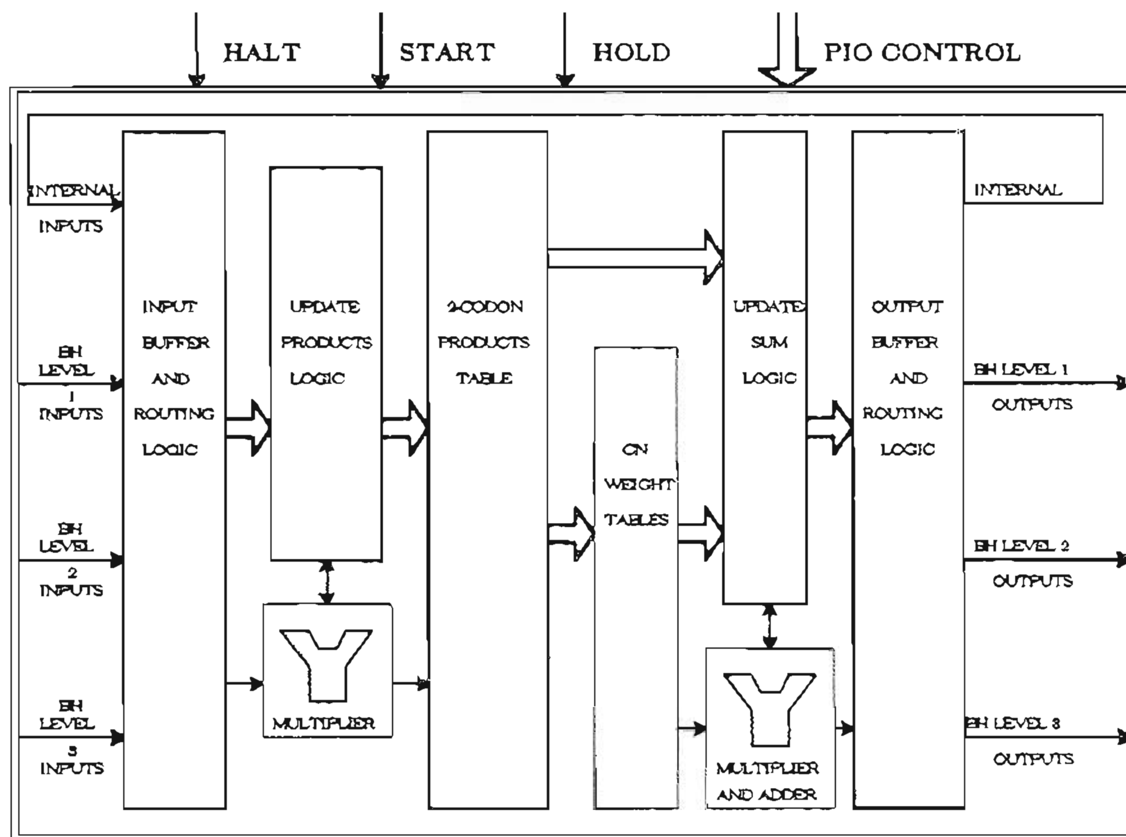

Figure 4 - A High Level Block Diagram of the PN Microarchitecture with Control Signals.

easily changed if required.

It is now possible to define the microarchitecture of the PN. As can be seen from an examination of Figure 4, the PN is segmented into five major functional units. The remainder of this chapter is devoted to defining the structure of these functional blocks in detail. The communication and synchronization between these blocks is also described.

Figure 5 shows a detailed block diagram of the INPUT BUFFER AND ROUTING LOGIC used by the PN. When a BH transmission packet is received by the PN, it is deserialized, buffered and converted to PN internal addresses as shown in

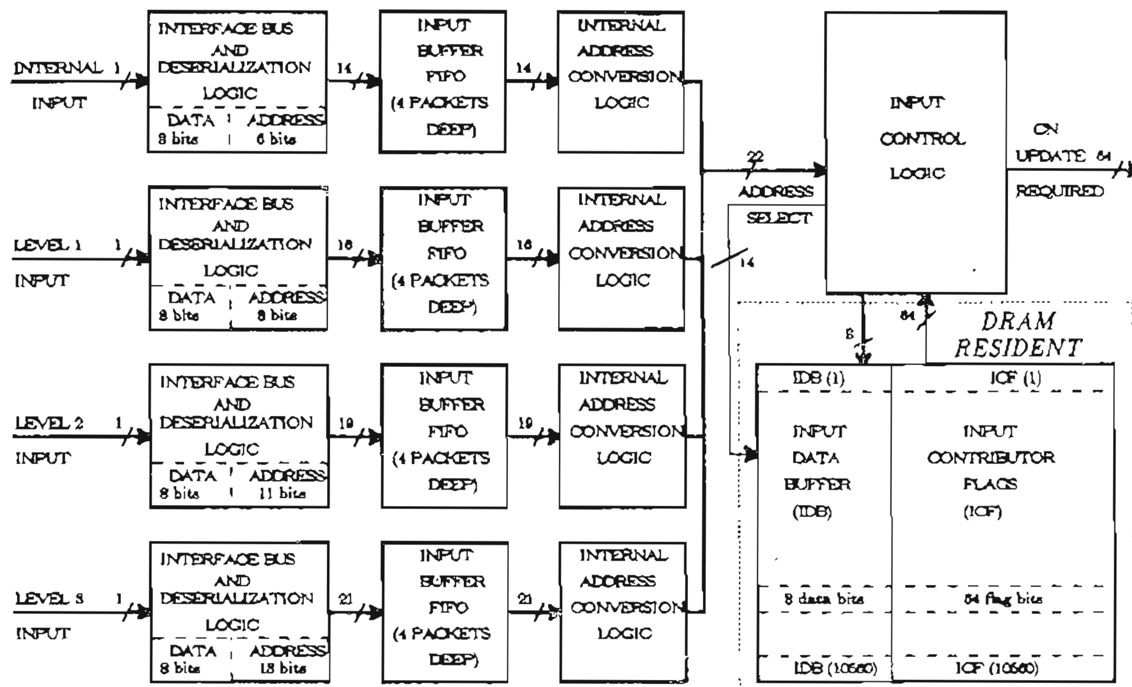

Figure 5 - A Block Diagram of the INPUT BUFFER AND ROUTING LOGIC. The Input Data Buffer is available to other functional units of the PN. The CN Update Required signals are passed to other functional units of the PN to initiate processing.

Figure 5. Once converted to internal format, the data in the received packets are written to the external memory where the Input Data Buffer (IDB) resides. These data are then accessible to other functional subsystems of the architecture. At this time, the input control logic uses the Input Contributor Flags (ICF) associated with the particular input to set the global CN Update Required bits. The ICF bits, coupled with related data structures, provide the control and synchronization mechanism that governs how and when CNs are updated. The use of these global mechanisms are defined below after first addressing some of the more fundamental issues of the input logic.

Parallel logic is provided to allow deserialization, buffering and address conversion to occur concurrently and independently on each of the BH levels. The reasons behind the use of this (mostly) duplicated logic is to isolate BH level specific considerations - both from the internal structure of the PN and from the other BH levels.

One particular dependency that can be masked using this sort of replicated BH specific logic is the slight difference in transmission packet formats. As shown in Figure 6, each BH level has a transmission packet that differs from other levels in the size of its address field. (Figure 5 also shows this feature as the input busses vary in size depending on the BH level.) The use of BH specific logic to deserialize transmission packets requires the implementation of a simple deserialization circuit that accepts a fixed number of serial bits and converts to a fixed size parallel word. The alternate method of using common logic would require some programmability of the deserialization circuitry to accommodate the different packet sizes. As this logic complication is not necessary, dedicated deserialization hardware is used for each BH level. The same type of arguments apply to the choice of using BH level specific dedicated circuitry for the data buffering and the address conversion logic.

As shown in Figure 5, the PN uses a four packet deep FIFO to buffer its inputs at each BH level. Buffering is required, because after the internal address is

DATA FIELD (8 bits)	ORIGINATOR ADDRESS FIELD (Bit width is BH level dependent)

Figure 6 - A Broadcast Hierarchy Transmission Packet. The size of the data field is fixed but the size of the address field is BH level dependent.

computed for a packet, the external memory must be accessed to write the data to the IDB and to fetch the ICFs. This accessing of memory could represent a bottleneck during times of high input frequency and as a result, it is necessary to slightly decouple the BH networks from the PN's processing of the packets. This is accomplished by the use of a FIFO buffer for each of the BH levels. It is not known whether a four deep buffer is acceptable in this architecture as the BH network load characteristics are not well defined at this time. Actually, buffer overflow is not a catastrophic occurrence in the CC and losing some small percentage of the packets could be tolerated. Nonetheless, this specific portion of the architecture will require significant future analysis before this question is completely resolved. As the bottleneck causing the problem is related to the use of external memory, it is possible that if the design were converted to use internal memory, then the problem could be eliminated.

The address conversion performed by the input oriented logic is straightforward. Within the PN, all input CNs are numbered sequentially from 1 to 10,560 and the corresponding memory locations store the latest values received for each CN. These addresses are computed from the BH level addresses by adding an offset that is BH level dependent. For internal levels, no offset is required as they both start at address one. For BH levels 1, 2 and 3 the offsets are 64, 320 (64 + 256) and 2368 (64 + 256 + 2048), respectively. The derivation of these offsets can be understood from an examination of the information given in Table 1. The computation of the internal addresses is therefore, quite simple. As a result, BH level specific dedicated translation hardware is an acceptable solution.

NAVAL POSTGRADUATE INSTITUTE LIBRARY

As mentioned earlier, the Input Control Logic accesses the external memory to write into the IDB and to fetch the ICFs. To accomplish these functions, it is necessary to perform an address mapping to obtain correct physical addresses in the DRAM. To avoid redundant logic, a centralized memory access facility is defined that all other portions of the microarchitecture use whenever they require access to external memory. This central facility would use a standard implementation of a base+offset memory controller that also has block transfer capabilities. The base address is derived from a lookup table by using a code passed to it to indicate which portion of the DRAM is being accessed. For example, a code of three might indicate that the request is for a word in the IDB portion in the DRAM. The offset and word count would also have to be passed as part of the requesting protocol to completely define the required transfer. This type of memory translation logic is commonplace in conventional computer systems and therefore does not require a detailed description here.

In addition to implementation ease, there are other reasons why a centralized memory access facility is desirable. One is that it limits the sphere of influence of the logical to physical mapping in such a way that changes to the mapping only affect the central facility. Also, memory bandwidth optimizations can be pursued without considerations of global features of the microarchitecture. The memory access facility also generates all required DRAM control signals, thus eliminating any need for "glue" logic between the PN and the DRAM. Finally, the use of this central facility allows maximum flexibility in adapting to technological changes in the interface characteristics of future memories.

OREGON GRADUATE INSTITUTE LIBRARY

The last portion of the input logic requiring discussion is the ICFs. These provide the cornerstone for the communication and synchronization mechanisms used by the PN. An ICF is a 64-bit wide, bit significant DRAM resident field that contains one bit for each of the PN's internal CNs. If the ICF bit for a CN is set for some input in the IDB, then the specified CN uses that input in the calculation of its output [7]. As a result, the ICFs provide a mechanism to define which CNs require recomputation of their output functions when an input changes. The role of the input oriented logic in this process is easily defined although its purpose and correctness will not become apparent until later. Whenever the Input Control Logic receives a new input value, it will fetch that input's ICFs, and for each ICF bit set, it will *pulse* the corresponding CN Update Required signal. Subsequent logic that uses the CN Update Required signals must latch the pulse on its rising edge. When other subsystems of the PN detect the rising edge of this pulse, it informs them that a recomputation of a particular CN's output is required. The details of the other blocks' use of this, and related synchronization structures, are described when their functional descriptions are given.

Figure 7 shows the UPDATE PRODUCTS LOGIC along with the previously defined Input Data Buffer. The data structures and computing elements shown in Figure 7 generate specific 2-codons that are required to compute CN outputs. When the Product Update Control Logic detects a pulse on a CN Update Required line, it asserts the corresponding CN Active bit to enable the sum update circuitry It then

---

[7] The loading of the ICFs occurs during system initialization when the Weight Tables and some other application specific memory are loaded. None of these values are modified during the normal computations of the PN. System initialization and dynamic learning are covered in a later chapter where some of the details of the loading are described.

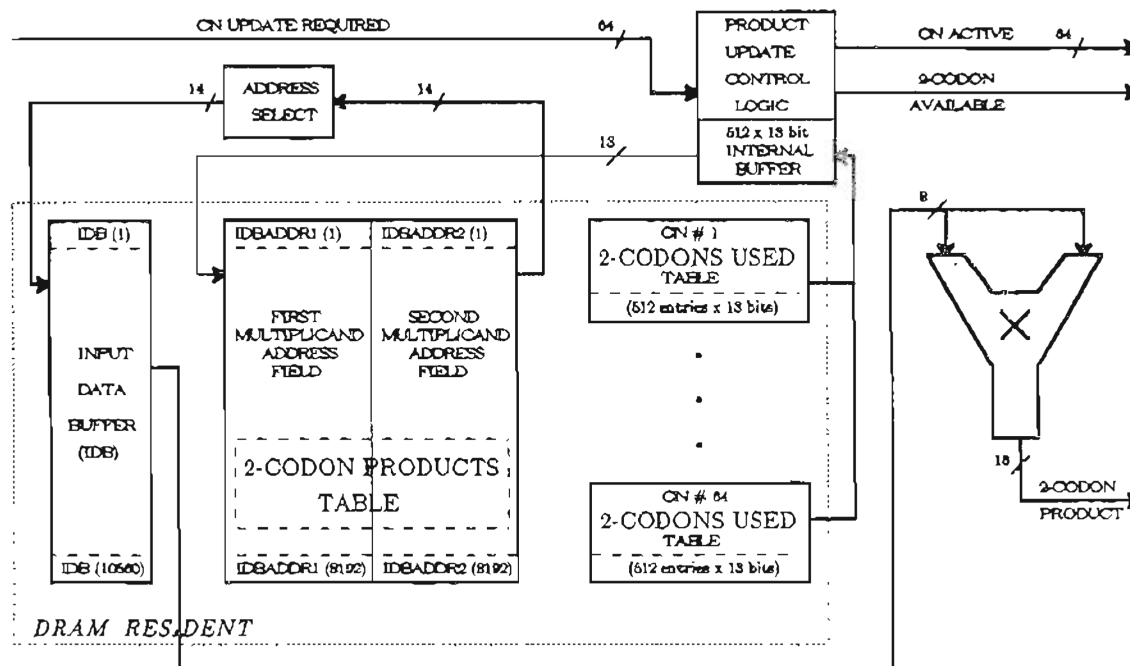

Figure 7 - A Block Diagram of the UPDATE PRODUCTS LOGIC. Whenever the CN Update Required bit is set for some CN, all 2-codons listed in the CN's 2-codons Used Table are computed. The products are passed directly to the Update Sum Logic.

reads the 2-codons Used Table for the corresponding CN. These 64 tables contain the addresses for each of the 2-codons that the particular CN uses to compute its output. As the number of 2-codons used by a CN is limited in this implementation to 512, these tables require 512 entries each. The PN contains on-chip memory for the specific purpose of storing an entire 2-codons Used Table. As a result, it is possible for the table to be read from the external memory into the internal buffer in a single memory transfer.

Once the table has been read into its internal buffer, the Product Update Control Logic reads each entry in the table to compute all 2-codons required by that CN. Specifically, it uses each entry as a pointer into the 2-codon Products Table where the IDB addresses of the two multiplicands are stored. After fetching these

IDB addresses, the specified multiplicand data is passed to the 8×8 multiplier. The 16-bit 2-codon result is then passed directly to the UPDATE SUM LOGIC and the 2-codon Available signal is strobed to cause the product to be latched. The Product Update Control Logic continues in this fashion until all 2-codons required by the given CN have been computed. The completion condition is detected when either the last of the 512 entries is exhausted or an address pointer of zero is detected. When complete, the control logic deasserts the CN Active bit for that CN to indicate that the last 2-codon product has been generated. At this time, the UPDATE PRODUCTS LOGIC either begins a similar process on a different CN, if required, or it goes into an idle mode.

In analyzing the performance implications of the proposed method of updating the 2-codons, two immediate concerns are raised. First, when a 2-codon is shared by multiple CNs, its 2-codon value is actually computed once for *every* CN that uses it. Although this may seem inefficient, implementations that eliminated multiple 2-codon computations required more complicated data structures and had significantly higher control overhead.

The second performance concern may actually become critical to PN performance if the inputs to the PN change too rapidly. This problem could become evident if *during* the computation of a CN's 2-codons, the CN Update Required signal for that CN is pulsed *again*. At this point, it will be necessary to discard all computations in progress and begin to recompute all 2-codon values even though it is possible that only one would change. In the ideal case, it would seem possible to store a designation of which input caused the midstream interruption and only recompute

OREGON GRADUATE INSTITUTE LIBRARY

2-codons that are based on it. Unfortunately, the mechanisms required to accomplish this selection were more complicated than their relative merit allowed. Future research in this area may find a method of allowing this type of midstream correction, if the applications that use the PN exhibit this sort of behavior routinely.

There is a final topic concerning the format of the 2-codon Products Table that requires explicit description. If one of the multiplicand addresses is zero, then the other input will be passed (unchanged) to the UPDATE SUM LOGIC as the 2-codon Product. Using a zero-value address in this manner provides a mechanism for the PN to use when a CN requires the simple weighting of a single input. This mechanism is efficient and straightforward in its implementation because this scenario could occur frequently.

The next major subsystem to be considered is the UPDATE SUM LOGIC shown in Figure 8. When the UPDATE SUM LOGIC detects a pulse on a CN Update Required line, it waits for a corresponding CN Active bit to be set by intermediary logic. When the latter bit gets set, it indicates that the UPDATE SUM LOGIC will start to receive 2-codon Products (for the specified CN) as an input to its 16-bit multiplier. As each 2-codon is strobed into the multiplier by the 2-codon Available signal, the control logic fetches the corresponding value from the appropriate Weight Table and passes it, as the other input, to the 16×16 multiplier. The 32-bit output products from the multiplier are repeatedly passed to the 32-bit by 41-bit full adder which recycles its previous result as one of its inputs [8].

[8] The 41-bit limit is derived from the maximum width that could be required when adding 512 (2^9) numbers that are 32 bits wide.

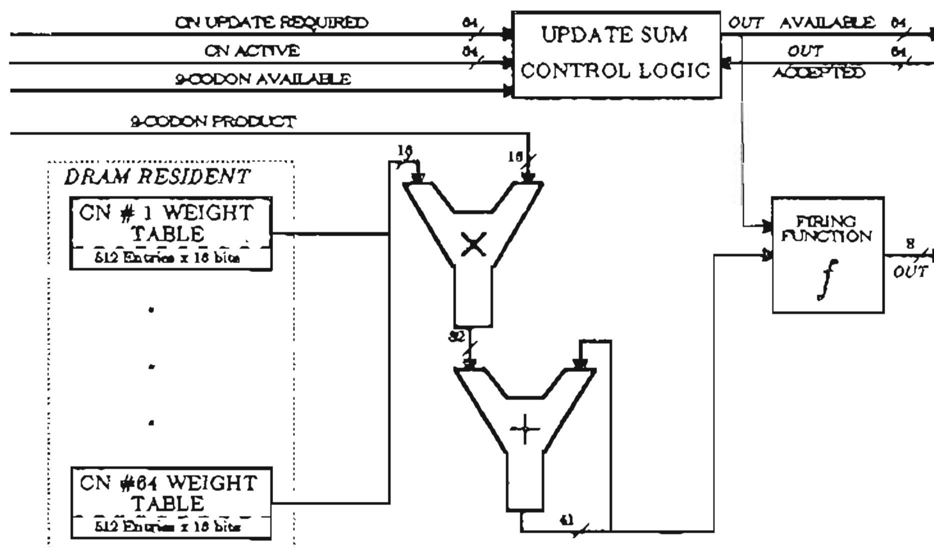

Figure 8 - A Block Diagram of the UPDATE SUM LOGIC with the Weight Tables for each CN. The 2-codon Available signal strobes the 2-codon product into the multiplier's input. The OUT Available and OUT Accepted lines provide full handshaking of the OUT value.

In this way, the summation defined in Equation 1 is performed and the result is passed to the Firing Function block. The assertion of the *OUT* Available bit instructs the Firing Function circuitry to accept the final result from the adder and then drive its own 8-bit result, *OUT,* towards the output oriented subsystem of the PN. The Firing Function block must continue to keep its output valid until the control logic deasserts the *OUT* Available bit. Because the control logic waits for the return of the *OUT* Accepted bit before it deasserts *OUT* Available, this provides a full handshake mechanism to insure that the output oriented logic receives the correct result.

Finally, Figure 9 shows the OUTPUT BUFFER AND ROUTING LOGIC used to send the newly calculated *OUT* values onto each of the BH levels. After detecting the assertion of a CN Update Required pulse, the output oriented control logic waits for the corresponding *OUT* Available bit to be set. After an implementation

OREGON GRADUATE INSTITUTE LIBRARY

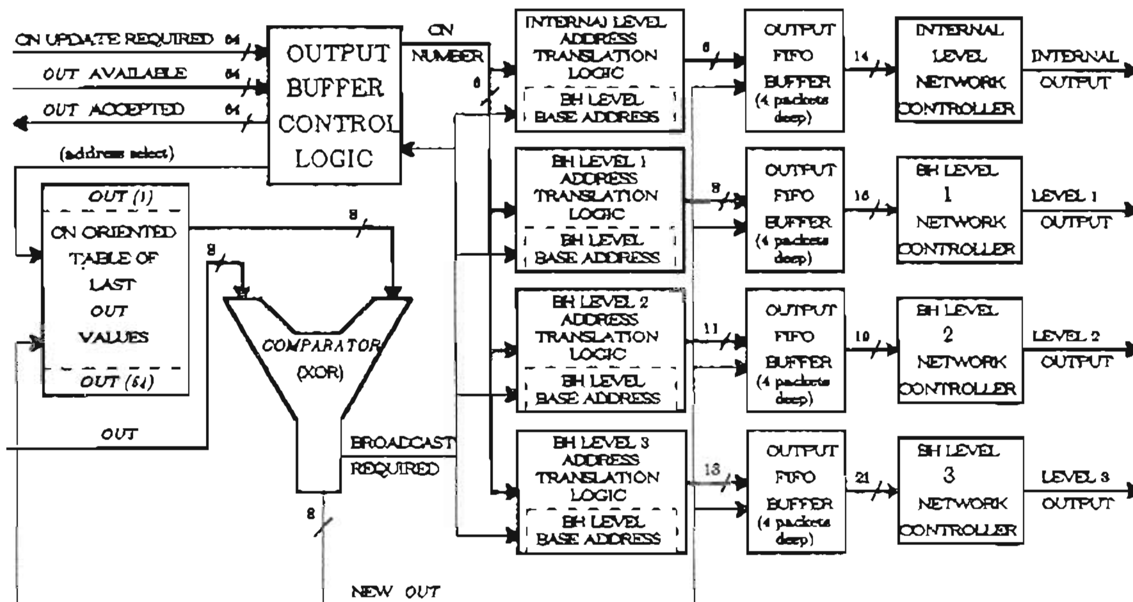

Figure 9 - A Block Diagram of the OUTPUT BUFFER AND ROUTING LOGIC. CN values are only broadcasted if they change from the last value broadcasted.

dependent deskew time, the 8-bit *OUT* value is latched as one input to the comparator shown in Figure 9, and the *OUT* Accepted bit for that CN is set. The Output Buffer Control Logic then causes the previously stored Last *OUT* Value, for the particular CN, to be passed as the second input to the comparator. If the comparator finds that the new *OUT* value and the last *OUT* value are equal, then no more processing is required as there has not been a net change in the state of the specified CN. In later implementations it will be desirable to loosen up the equality constraint and instead determine whether the current and last *OUT* values are equal to each other within some predefined threshold. Loosening this constraint will actually simplify the comparator logic as it will eliminate some of the low-order bits from the XOR circuitry.

If the comparator finds the new and last *OUT* values to be different, then it is necessary to broadcast the new value to all other CNs in the CC. Under this circumstance, the comparator drives the new *OUT* value as its output and generates the Broadcast Required control signal. When the control circuitry detects the assertion of the Broadcast Required signal, it passes the CN number that is being updated to the Address Translation Logic blocks for each BH level [9]. At this time, the Output Buffer Control Logic causes the new *OUT* value to be written to the Table of Last *OUT* Values for future comparisons.

When the Address Translation Logic blocks receive the Broadcast Required signal, they compute the BH level specific address of the CN by adding the internal CN number to its own BH level specific base address. The BH level base address for each level is stored in a dedicated internal register that is loaded during system initialization as described in a later chapter. The computed BH addresses are passed to the BH level specific FIFO buffer where they are united with the data output from the comparator and buffered.

Once the data is in the FIFO buffer, it is available to the BH level Network Controller which is responsible for performing the required parallel to serial conversion. Each Network Controller arbitrates for use of its own BH physical network and then broadcasts the new *OUT* value to all other PNs that are connected to it. Thus the process begins anew on any PNs that have a CN that relies on the new value.

---

[9] Recent research at OGC [BaH86] has shown the need for programmable control over whether a CN's output gets transmitted on each separate BH level as defined by the *Broadcast Control Field*. Although this feature is not supported in this implementation, the Output Control Logic could be easily modified to accommodate it.

Most of the architectural features of the output oriented logic are simple inversions of the operations performed by the input oriented logic of the PN. Therefore, rationalization of the repetition of similar logic on each of the BH levels is not required as the arguments are the same as those given above for the input section. Nonetheless, it would seem possible to omit the use of any FIFO buffering in the output logic as the time required to compute a total CN update is probably quite large compared with the time required to transmit the transmission packet onto the BH network. It was decided that the choice of *not* using FIFO buffering was imprudent for two reasons. The first is that a CN input could cause an output to be generated quite quickly if there is only a small number of entries in the affected CN's 2-codons Used Table. The second is that a BH network could have bursts of high communications traffic and therefore significant network delays could occur. In either case, a FIFO buffer allows for these circumstances and insures that the PN is capable of responding to them. On the other hand, it is not entirely clear whether the four deep FIFO is optimal or even acceptable. Future work must be done to discover application requirements in this respect.

Now that the microarchitectural structure has been presented, it is important to describe the external signals that are used to control the PN. Figure 4 shows the major PN control signals. These are the START, HALT, HOLD and PIO (Programmed I/O) signals. Not surprisingly, assertion of the START signal causes the PN to begin execution and HALT causes it to cease execution. Assertion of the HOLD signal temporarily inhibits the PN from accessing its DRAM. This signal is used if the I/O controller or microprocessor require exclusive access to the DRAM

This function is required if learning is performed as a background task. The PIO control signals are used to cause the PN to enter a mode where it allows external access to its internal memory locations. This access is required during PN initialization and may also be useful in determining the status of an individual CN after execution has been halted. The actual implementation of the PIO logic is straightforward because all that is required is selective access to internal registers. This is a standard technical requirement in VLSI circuits and therefore does not require a more detailed explanation in this thesis. On the other hand, a brief discussion of the complexity of the circuitry used to implement the control signals is given in the next chapter. Furthermore, a later chapter on initialization and learning will show how these signals are used during these operations.

The microarchitectural definition of the PN has now been provided in sufficient detail to allow continued research to refine and validate the proposal. The next chapter covers some of the specific technological and implementation specific topics and shows the viability of this proposal. Furthermore, a performance analysis is given in a later chapter to round out the definition of this PN implementation proposal and establish its desirability.

4. TECHNOLOGICAL FEASIBILITY

This chapter presents an analysis of the PN that defines and quantifies its requirements. In particular, it examines the requirements of the PN in terms of memory use, transistor count, silicon area and external I/O pin count. The purpose of this analysis is to show that the PN is a feasible VLSI circuit. As a further result of this resource quantification, information is derived that can be used to estimate the "cost" of manufacturing the PN. This data will be critical to the cost/performance analysis presented in the next chapter.

It is important to remember while reading this chapter that this thesis presents a fairly high-level architectural description of the PN and it is not a detailed design of the circuitry. As a result, there are several areas in the following analysis where gross assumptions are made about the actual details of the required circuitry. These assumptions are acceptable because the major goal of this chapter is to show that this *architectural* proposal is feasible. Establishing this as fact, insures that any following research effort will be well spent.

The first task in analyzing the PN's resource requirements is estimating its use of memory. This analysis is actually best broken down into two separate topics. These are the PN's use of *external* DRAM and its use of *on-chip* memory. The first of these to be considered is the PN's requirements for external memory. Table 2 shows the memory capacity requirements of the PN's DRAM based data structures

that were defined in Chapter 3. It also lists the architectural parameters of the PN that ultimately dictate the memory size of each specific data structure. These architectural parameters include the number of input CNs, the number of internal CNs, data widths and various table sizes. Some of these parameters indirectly affect PN memory requirements by defining address field sizes or the replication of a data structure. The itemization of this information eases verification of the conclusions presented here.

As can be seen from an examination of Table 2, the PN requires approximately 2 M-bits of external memory. This could be accommodated with commercially available DRAM by using either two 1 M-bit ICs or by using a single 4 M-bit part. Although the higher capacity memories are not readily available today, their

MEMORY TYPE	DATA SIZE	ADDRESS OR CONTROL FIELD SIZE	REPLICATION	REQUIRED MEMORY IN K-BITS
INPUT DATA BUFFER	8 bits	64 bits (1 per internal CN)	10560 (1 per input CN)	760
2-CODON PRODUCTS TABLE	0	28 bits (2 times the 14 bits required to address 10560 input CNs)	8192 (1 per 2-CODON)	229
2-CODONS USED TABLES	0	13 bits (required to address 8192 2-CODON Table entries)	64 x 512 (1 per internal CN times the entries per Weight Table)	426
WEIGHT TABLES	16 bits	0	64 x 512 (1 per internal CN times the entries per Weight Table)	524

TOTAL = 1939 K-BITS

Table 2 - An Itemized Summary of the PN's External Memory Requirements.

use would be highly desirable in the CC application to lower the required number of ICs. Furthermore, the excess capacity of the 4 M-bit memory would be consumed if some of the architectural parameters of the PN are increased in future designs. As a result of these benefits, the rest of this thesis will assume the use of the 4 M-bit DRAM. This choice introduces an element of design risk that substantially depends on the time frame of the production of the CC.

The analysis of on-chip memory requirements is also straightforward. The information shown in Figure 5 can be used to compute the internal memory requirements of the Input Buffer and Routing Logic. As defined, this circuitry uses 414 memory bits that are arranged in 21 separately addressable registers. Five data buffers are used for each of the four BH levels and one 64-bit register is used to buffer one input CN's ICFs.

The Update Products Logic (Figure 7) only requires on-chip memory to buffer a single 2-codons Used Table. Each of these tables consists of 512 registers that are 13-bits wide to yield 6656 bits per table. The Update Sum Logic does not use any on-chip memory.

The Output Buffer and Routing Logic (Figure 9) uses 876 bits of internal memory in 88 separate registers. As with the input oriented logic, twenty of these registers are used for data and address buffering. But in this case, four registers are used to store the BH level base addresses and 64 registers are used to store the last *OUT* value for each internal CN.

Coupled together, this data implies that the PN needs to contain 7946 internal memory bits in 621 separately addressable registers. When more of the details of

this design are determined, it is likely that more on-chip memory would be required for unanticipated data transfer and control requirements. Therefore, this memory estimate actually gives only a lower bound on the PN's on-chip memory resources. Nevertheless, this capacity figure is a decent approximation and is possible with existing VLSI technology.

The next step in the technological analysis of the PN is to estimate the transistor count and silicon area that it will occupy. The total transistor count estimate will be derived from detailed estimates of each of the *major* computational and control circuits of the PN. These major circuits are the adder, the two integer multipliers, the external memory controller (EMC) and the PN global control and synchronization circuitry. This gross method of estimating the total transistor count of the PN omits consideration of *minor* computational circuits and miscellaneous control and synchronization logic that will surely be required. To adjust for this shortcoming, a generous allowance will be added to the sum of the estimates to obtain what should be a conservative upper bound on the transistor count of the PN.

It is assumed that the PN will be implemented in a CMOS design. This choice is consistent with current technological trends. It also allows the use of the comprehensive CMOS design reference by Weste and Eshraghian [WeE85]. This reference contains much of the information that was used to estimate the transistor count and silicon area of the fundamental computational circuitry as defined below.

The first circuits to be considered are the adder and the multipliers. A reasonable implementation of a CMOS full adder, i.e., a transmission gate adder, requires approximately 25 transistors per bit. Using this value, the 41-bit adder

shown in Figure 8 would be composed of 1025 transistors. In reality, there is significant variation in the complexity of adder designs with substantial trade-offs between performance and circuit size. Performance concerns tend to dominate as the data size increases, and the transmission gate adder may not be fast enough in a 41-bit application. As a result, it may be necessary to use a more complicated, higher performance, adder circuit than the one proposed.

The performance of the multipliers is critical to the computational throughput of the PN. Therefore, it is important to implement a high-speed, fully parallel, multiplication circuit. Multipliers in this class are constructed from full adders, such as the transmission gate adders considered above. Specifically, an $n \times n$ multiplier requires on the order of n^2 full adders to perform fully parallel multiplications. Therefore, the 8×8 multiplier would be composed of approximately 1600 (25×64) transistors. The 16×16 multiplier requires 6400 (25×256) transistors. This yields a total transistor count of 9025 for the adder and the multipliers.

The *external memory controller* (EMC) must be kept computationally simple to prevent it from becoming a performance bottleneck. A high performance implementation of the base+offset logic could be obtained from integrating a special purpose barrel-shifter that converts the *logical* offset, passed to the EMC, into the physical memory offset. This value is merged onto the lower-order bits of the address bus with the specified base address to obtain the desired physical memory address. All together, the required base+offset logic is composed of approximately 200 transistors. It is not necessary to list the intimate details used in estimating the transistor count for this trivial circuitry.

To implement the block transfer capabilities of the EMC, it is necessary to include automatic word count decrementing logic and automatic address incrementing circuitry. This could be done effectively by using a special-function adder like the one described above. Assuming byte addressability, 19-bit adders will be required to support complete access of the 4 M-bit DRAM. Therefore, the address and counting logic would each require approximately 475 (25×19) transistors. This yields a total sum of 1150 ($= 475 + 475 + 200$) transistors required to implement the entire EMC logic. Although this figure could be subject to substantial reconsideration as the exact circuitry evolves, the magnitude of the estimate will not change dramatically unless significant functionality is added.

The next task in estimating the PN's transistor count is analyzing its global control and synchronization circuitry. It is desirable to use a PLA as the major control structure for the PN. Although this method is conceptually appealing, it is not feasible in this application because the vast majority of the control signals must be replicated for each CN. More specifically, this application would require a control structure with approximately 640 inputs and 640 outputs. A PLA implementation of this logic was considered but was rejected because this application has only sparse connections and this would result in inefficient use of silicon area.

To circumvent this concern, the PN's control flow is designed in such a manner that it can be implemented by replicating circuitry that is both conceptually straightforward and physically dense. There are several fundamental qualities of the PN architecture that tend to support these goals. The first point worth noting is that the majority of the synchronization and control within the PN is accomplished

by independent circuitry that is replicated for each of the internal CNs. This replication provides a physical regularity of cells that minimizes circuit layout problems. Regularity can be crucial to successful VLSI designs.

From a more microscopic perspective, the structured "control flow" of the PN is a major factor in accommodating the design of compact and efficient control circuitry. Specifically, CN Update Required is the only major control signal that has a global effect. The other four major control signals for each CN are locally derived and have only local effects. They are structured in such a way that control "flows" in an orderly pattern from the input of the PN to its output. As a result of these two features, it is easy to visualize a rectangular cell that contains all of the control logic for a CN, but is still small and compact, and has few external connections. As a further benefit of this systematic control flow, the few external connections that are required may enter the logic at fixed intervals and therefore, the physical interference between them is minimized.

Given all of these arguments, it is reasonable to consider a fundamental control structure that uses 30 transistors per CN. These transistors could probably be arranged in a regular rectangular grid with roughly four conductor rows and ten conductor columns. Therefore, approximately 1920 (30×64) transistors would be used in the PN's global control and synchronization mechanisms. Physical size estimates of the control logic are more important and they are presented below.

Lastly, we need to determine the total transistor count of the PN's on-chip memory. A six transistor per bit SRAM cell can be used. Therefore, 47,676 transistors are required to implement the 7946 bits of on-chip memory.

At this point, all of the major logical components of the PN have been considered and the sum of these estimates is close to 60,000 transistors [10]. It may now be concluded that the PN architecture defined in Chapter 3 is feasible from a purely transistor count perspective. This conclusion remains true even if a 100 % transistor count allowance is reserved for the miscellaneous computational and control circuits that were omitted in the analysis presented above. As a matter of fact, commercial ICs composed of greater than 120,000 transistors are commonplace.

As an additional check of implementability, the silicon area requirements of the logical components discussed above are estimated. This is the final step in showing that the PN is technologically feasible using currently available CMOS processing techniques. Table 3 shows the approximate cell sizes of the most significant of the major logical elements discussed above. Although these values are only gross

CELL TYPE	CELL SIZE (in μm^2)	REPLICATION	SILICON AREA REQUIRED (in μm^2)
ADDER	7,744	399	3,089,856
SRAM	4,800	7,946	38,140,800
GLOBAL CONTROL CELL	5,760	64	368,640
			TOTAL = 41,599,296 μm^2

Table 3 - Silicon Area Information of the Principal PN Logic Cells. These values assume a minimum feature size of 2 μm.

[10] The vast majority of these transistors are in area-efficient SRAM cells.

estimates, they are based on 2 μm minimum feature size ($\lambda = 1\ \mu m$), two level metalization and standard CMOS design rules [WeE85]. Specifically, the adder cell is estimated as a square with 176 μm (88 λ) sides. The SRAM is a 80 μm (40 λ) by 60 μm (30 λ) rectangular cell. The control logic cell size was estimated by assuming a 4×10 conductor grid with a 12 μm (6 λ) grid spacing. Table 3 also gives the replication of each cell type and the total silicon area that they require.

As can be seen from examining Table 3, the major logical portions of the PN will require a silicon area of around 40 million μm^2. This could be easily accommodated in a somewhat typical 1 cm^2 (which is 100 million μm^2) VLSI chip.

No technological analysis of a VLSI architecture would be complete without performing a quick examination of its external I/O pin-count and some related parameters. In this regard, the PN is not only feasible but it has a low enough pin-count that ECB signal routing is simple. Specifically, the PN requires a total of only 56 external connections. Three of these are used for the START, HALT and HOLD signals shown in Figure 4. Assuming that each external BH level uses a serial interface that requires four connections, a total of twelve pins are required for the three external levels. The PIO interface shown in Figure 4 requires thirteen connections. Ten of these are required to specify addressing of the 621 different registers, one specifies the access direction (read/write) and two connections provide a handshake mechanism. Similarly, assuming a byte-wide DRAM, approximately eighteen pins are used for the connection between it and the PN. These are broken down into eight data lines, six address lines, CAS, RAS, chip select and write enable. Finally, approximately ten connections are required for power, ground and multiple clock

phases. Although these are only rough estimates, a pin-count of 56 is quite small and is easily accommodated with conventional VLSI packaging techniques. This low pin count is also desirable as it reduces board costs and increases system reliability.

The pin-count estimate given above includes four pins each for power and ground. This number is required to support the estimated 60K transistors of the PN. Furthermore, the raw power consumption of the PN is not of significant concern as the vast majority of the PN's transistors are used in SRAM cells and these are power efficient in CMOS technology. As a result, no major power related concerns are expected in either supplying raw power to the PN or in cooling it.

5. PERFORMANCE ANALYSIS

This chapter presents an evaluation of the computational performance of the PN microarchitecture proposed in Chapter 3. Unfortunately, due to the scope of this design, a detailed simulation of the proposed PN architecture was considered to be beyond the scope of this thesis. Also, simulation is not critical in this case, since reasonably accurate performance analysis is possible, and a strong argument is made that the performance of the PN is limited by the memory bandwidth of the DRAM in any case.

After establishing the PN's performance as memory access limited, a discussion is given that explains why memory limited performance is acceptable in light of other system considerations. A sensitivity analysis is then presented that shows the performance effects expected from varying the memory bandwidth. Next, an algebraic expression is derived that allows the quick computation of the PN response time given a specific stimulus. Several different stimuli will be explicitly considered that represent the PN operating under different computational loads. Although this approach does not validate the correctness of the proposed microarchitecture, it will allow the calculation of quantitative information to predict the performance of the PN based Connection Computer. Finally, cost/performance results will be derived and compared with more conventional computer systems performing the same application. As will be seen, the PN compares favorably in this regard.

Before presenting the detailed performance analysis, it is useful to present the reasons why a simulation of the PN is not critical to the performance analysis portion of this research. There are several reasons why a simple simulation of an individual PN would actually have only limited value. One reason is that the PN is designed to be used in groups (i.e. a CC) and the performance of the group is *not* directly related to the performance of any individual PN. Rather, the performance of the CC is derived from characteristics of both the individual PN's performance and communication delays between connected PNs. Therefore, to obtain accurate estimates of CC performance, a statistical treatment of both the PN response time and related communication overhead must be performed. This analysis must examine the effect of different computational loads and communication delays on the net performance of the CC. Currently, anticipated workload specifications within the CC are poorly defined so even statistical analysis would be of little concrete value. As a result of these complications and limitations, the following analysis is restricted to *a priori* methods rather than more detailed simulation methods. When the workloads within the CC become more defined, system simulation of PNs will become necessary.

The first step towards estimating the performance of the PN is showing that its DRAM accesses are the limiting factor. To establish this relationship, it is first necessary to compare the DRAM access times to the directly related computation times. If the time required to access memory is significantly greater than a reasonable performance estimate of the related circuitry, then it can be concluded that the memory access time is a reasonable approximation of the computation time.

To begin the detailed comparison, consider the logic shown in Figure 7 that computes the 2-codon products. The basic computation performed here is the repeated multiplication of two 8-bit values fetched from the DRAM to obtain the 16-bit results that are passed directly to the multiplier of the Update Sum Logic. Therefore, each multiplication requires the access of two *data* bytes, but Figure 7 shows that an additional 41-bits of address and control information must be read from the DRAM to obtain the IDB addresses of the multiplicands. Thirteen of these bits are the entry in the 2-codons Used Table that provides a pointer into the 2-codon Products Table. Using this pointer, 28 address bits are fetched that define the actual IDB addresses of the two target multiplicands. Altogether, this scenario results in the access of approximately seven bytes of DRAM just to perform the multiplication. It is useful to consider whether the total time required to access the DRAM is significantly greater than the time required to actually perform the 8-bit multiplication.

To answer this question, it is necessary to make some performance oriented assumptions about both the PN and the DRAM. It is reasonable to assume that the PN could operate at a clock rate of 10 M-Hz and that it could perform simple operations in a single clock cycle (100 nS). Specifically, it may be assumed that the 8×8 multiplier could be designed in such a way that it takes just one clock cycle per multiplication. On the other side of the comparison, consider a DRAM that is *very* aggressive in terms of performance. Assume that the DRAM used in the CC will have a data path that is 8-bits wide and an effective cycle time of 100 nS [11]. Given

[11] Currently available DRAMs tend to be no larger that 4-bits wide with cycle times closer to 200 nS. DRAM performance figures will probably not improve dramatically for high-capacity DRAMs in the near future.

these fundamental performance estimates and the DRAM reference characteristics explained above, eight separate byte-wide accesses would be required to yield a total memory access time of 800 nS. This is eight times the amount of time required to actually perform the multiplication. Furthermore, the multiplication time could be entirely hidden by performing the fetch of the next multiplicand address *during* the multiplication cycle. Simplistically speaking, these results show that the time it takes to compute a 2-codon is directly dependent on the related DRAM access time.

Before moving on to the other major circuit elements of the PN, it is important to consider the control overhead associated with the multiplications discussed above. In particular, one would wonder if the proposed microarchitecture contains some control path that would implicitly limit its performance to below what is possible given its DRAM references. Although this type of limit is always possible if circuitry is designed that is incorrect or inefficient, it is unlikely that this would be a great risk in the PN design because its control structure is simple. This is exemplified in the Update Products Logic discussed above and can be seen from a careful examination of its control flow. After detecting the assertion of the CN Update Required signal for a CN, the Update Products Logic activates the Update Sum Logic by asserting the appropriate CN Active bit for the given CN. At this point, control is transferred to the logic block that actually causes the multiplications to be performed. This logic *sequentially* fetches six pointer bytes that define the address of the two data bytes and then directly fetches the data and passes it to the multiplier. When the multiplication is complete, it passes the result to the multiplier in the Update Sum Logic and strobes the 2-codon Available signal to cause

the product to be latched as one input to the 16×16 multiplier. This *exact* process is repeated until every required 2-codon has been computed.

Thus, the only significantly complicated function performed by the Update Products Logic *directly* involves DRAM accesses. Therefore, the only significant performance bottleneck would have to result from inefficiencies in this part of the circuitry. This is not a large risk because efficient traversal of indirect memory references is a common function in many VLSI designs and good solutions to potential design problems are prevalent. For instance, general purpose commercial microprocessors must solve this problem in a general way to support a wide array of addressing modes. On the other hand, this is an area of the PN design where work towards optimization could have linear benefits.

Now that the 2-codon multiplication has been fully characterized, it is possible to consider the performance of the other components of the PN. In some cases below, performance arguments are similar to those made above and so they are only explained briefly. For instance, the description of the performance of the Update Sum Logic (Figure 8) closely follows from the previous discussion of the Update Products Logic. In particular, only two bytes of data are accessed from the DRAM for each 16-bit multiplication as the other two bytes are passed directly to the multiplier from the Update Products Logic. Accessing these two additional data bytes implies the need for an additional memory access time of 200 nS for every entry in the Weight Table. As the timing between 2-codon Products was derived from dedicated access to the DRAM, the two additional data bytes could not be fetched during the 800 nS required to compute each 2-codon. Therefore, the multiplier in the

Update Sum Logic can only be cycled every 1000 nS as limited by the ten bytes of data that are ultimately accessed in each iteration. Assuming that the 16-bit multiplier could also be designed to perform one multiplication every 100 nS clock cycle and that the multiplication could be overlapped with subsequent data accesses, its performance is clearly limited by DRAM references. Furthermore, the 41-bit addition, that follows every multiplication, could also be performed in one clock cycle. This addition requires no DRAM references so it could be performed in parallel with the computation of the next product value. This scheme yields an efficient computational pipeline where each iteration takes only 1000 nS.

In contrast to the addition, even though the Firing Function logic (also in Figure 7) is simple, it cannot be pipelined as it is only computed once per CN. Therefore, the Firing Function adds one additional clock cycle to the total time required by the Update Sum Logic.

The Input Buffer and Routing Logic (Figure 5) accesses nine DRAM bytes every time a new input value is received. After input deserialization, it is unlikely that any of its simple translation oriented circuits would require more than the 900 nS used by these DRAM references. Therefore, this value represents an acceptable approximation of its response time. Of course, this analysis assumes that the input logic's FIFO buffers are empty so that they do not contribute to the delay. Obtaining performance estimates associated with non-empty buffers would require substantial additional effort and is not done here.

Finally, the Output Buffer and Routing Logic (Figure 9) does not access any external memory. Therefore, the type of arguments given above do not apply. On

the other hand, the output logic is sequential and straightforward so an accurate estimate can be made by just examining its circuitry. Again assuming that the output FIFOs are empty, it would take approximately six clock cycles from the time that *OUT* Available is asserted until the deserialization and transmission can begin. As with the analysis of the input logic given above, it is assumed that network delays do not occur. The effects of network delays will have to be considered in later efforts.

Before deriving an algebraic expression that predicts the PN's macroscopic performance characteristics, it is interesting to explicitly consider topics that relate to system performance trade-offs, sensitivity analysis, global parallelism and resource contention. These topics are discussed briefly to assure the reader that they have been satisfactorily considered.

One might wonder why an architecture is proposed that is memory bandwidth limited when the use of high-capacity, high-speed memory components is extremely economical. It would be easily possible to increase the bandwidth by using either faster memories (such as SRAMs) or parallel access to multiple memory components. Although these solutions are feasible and would provide higher performance, they are inconsistent with a more fundamental system goal. As discussed in Chapter 2, an overriding goal in the CC design is to support computations using a *very* large number of PNs. To maximize the number of PNs possible in a CC, it is crucial to minimize the ECB real-estate and IC count associated with each PN. Therefore, the use of only a single DRAM provides an almost optimal solution in this regard and its performance appears to be acceptable for target applications. The PN based CC

also provides a high memory density per ECB and this is a major factor in this memory intensive application. As a result, other memory configurations were rejected.

In examining the sensitivity of the PN performance on DRAM bandwidth, if the bandwidth of the DRAM arbitrarily decreases, the effect on the performance of the PN will be a linearly related decrease. On the other hand, if the bandwidth increases, the performance benefit will only be linear until a threshold is reached when the computational circuitry becomes the limiting factor. It is not known when this performance threshold would be reached, but from a raw component perspective it would be roughly in the 20 nS memory cycle time range. Similar thresholds would be reached if different memory configurations were used that yield wider effective data accesses. Future efforts will be necessary to refine this estimate if significantly higher performance memories are considered.

Another general topic worth consideration is the PN's support of global parallelism. Global parallelism within the PN could be defined as occurring when different computational elements are performing operations at the same time. In fact, the computational efficiency of the PN is largely derived from its use of computational parallelism. In particular, the two multiplications and the addition (described above) are all computed concurrently using pipeline methods. Further, these arithmetic operations can occur in parallel with activities in both the input and output logic portions of the PN. This use of global parallelism yields net performance that is similar to much higher cost designs, as will be shown in the end of this chapter.

There is one key area where computational parallelism is required. Whenever the input logic receives new data, it must be able to fetch its associated ICFs, even if some other computational element is actively accessing external memory. This is required because the fetched ICFs could cause the computations that are already in progress to be terminated and restarted, which would be the case if the internal CN being computed has a data dependency on the *new* input value received. If the ICFs show no dependency, the computation could resume where interrupted. Clearly in this case, prioritized global parallelism is critical to the PN's performance as it minimizes unnecessary computations.

A concern that is often raised when assessing the value of performance estimates is the effect of resource contention. There are three distinct areas in the proposed architecture where resource contention could be an issue. The first results from the use of DRAM for the storage of many data and control structures. The effect of contention in this case was implicitly considered in the performance discussion given above. The second shared resource subject to contention is the external memory controller. This logic will have to be designed to insure that its access for one purpose does not adversely impact its efficiency for other purposes. Finally, the BH networks have contention problems that result in output value transmission delays. This scenario is not considered in this thesis.

Another question that could be raised is whether a caching method could be used to lessen the PN's total memory access time requirements. The PN's references within the 2-codon Products Table and its references of CN inputs are nonsequential and could be widely scattered in DRAM memory. There is also no reason to believe

that there would be a high likelihood of reuse of these data elements. Furthermore, as the PN's memory *capacity* requirements are quite large, it is infeasible to cache the entire data sets. Given these three factors, caching could not be used effectively to obtain a performance benefit. If future research shows high reuse within either of these data structures, then a caching mechanism should be considered.

To develop an algebraic expression for the response time of the PN, it is necessary to define the related variables [12]. For any input CN, define n as the number of bits set in the input CN's ICF entry. This value gives the number of internal CNs that will have to be computed as a result of the input change. Next let l_k, where $1 \leq k \leq n$, be the number of entries in the Weight Table for each of the n affected CNs. Given that just one input CN changes when the PN is otherwise idle, the response time, R, is defined in Equation 2 [13].

$$R = 900 + \left(\sum_{k=1}^{n} l_k \times 1000 \right) + \left(n \times 700 \right) \; nS \qquad \textit{Equation 2}$$

In Equation 2, R is the time between the receipt of the input value and the transmission of the last output value. The specific numeric information is derived from the appropriate circuit performance estimates given above.

Although Equation 2 gives an accurate estimate of the response time of the PN, it requires specification of the size of each Weight Table. This requirement

[12] Throughout the following discussion, lower case letters denote variables that are defined by properties of a single CN. When upper case letters are used, they are derived from an ensemble of CNs and have similar characteristics as their lower case roots.

[13] All of the following specific performance estimates rely on the aggressive DRAM performance assumptions that were previously described. As a result, these estimates may have to be readjusted to reflect actual DRAM product availability when the CC is produced.

makes it too application specific for macroscopic estimates but this problem can be solved without compromising the precision of the estimate. To do so, let L be the average number of entries in the Weight Tables of the affected internal CNs. Given this definition, the equivalent expression for the response time is given in Equation 3.

$$R = 900 + (n \times L \times 1000) + (n \times 700) \ \ nS \qquad \textit{Equation 3}$$

This yields an accurate estimate of the response time of the PN given two application parameters that can be easily obtained. Note that R has the *exact* same meaning in Equation 3 as it does in Equation 2, but is just defined in terms of different variables.

Before calculating the PN's performance expectations, it is necessary to further generalize the function that predicts its response time. In particular, both Equations 2 and 3 assume that only one input CN changes at any one time. In reality, many input CNs may change at the same time so it is desirable to include this behavior in the performance model used. To include this situation, define I as the number of inputs that change at the *same* time. Also, let N represent the number of bits set in the logical OR of the ICF bits associated with each of the I input CNs that change at one time. Although N is derived differently than n, its significance is the same as it denotes the number of internal CNs that have to be updated when new inputs are received. Given these definitions, the response time of the PN, when several inputs change concurrently, is defined in Equation 4.

$$R = (I \times 900) + (N \times L \times 1000) + (N \times 700) \ \ nS \qquad \textit{Equation 4}$$

Equation 4 gives the definition of a new PN response time, $\bar{R}$, that will be used below. It is interesting to note that when I is set to a value of one, Equation 4 is equivalent to Equation 3, as would be expected. Also, increasing the value of I has only a relatively small direct effect on the PN response time. Its indirect effect of converting the small value n to the larger value N is likely to have a more substantial net effect.

There is one limitation of the performance model used to derive Equation 4 that needs to be explicitly stated and discussed. Specifically, the derivation of Equation 4 assumes that all of the changes to the input CNs occur at the exact same time. In fact, this restriction can be somewhat loosened to allow all cases where the inputs change at any time during the PN's computations as long as new inputs do **not** force recomputation of any kind within the PN. It is unlikely that this ideal case would occur often in most applications. Deviation from this ideal adversely impacts the response time of the PN. The magnitude of the impact is extremely application dependent. In the worst case, the PN would never be able to generate an output because it is continually having its computations interrupted. In fact, there will be an input frequency window within which substantial inefficiency will result. The actual parameters that define the width of this window are application dependent. Nonetheless, it is interesting to note that these performance problems are self-limiting because if CN results cannot be computed then they cannot be transmitted. Given these qualifications, the response time defined by Equation 4 is an approximate lower bound.

Finally, it is possible to present some interesting estimates of the performance of the PN. Table 4 shows predicted PN response times under different computational loads. There are several interesting insights that can be obtained from examining Table 4. One is that the minimum response time of 2.6 μS is fast, when one considers that even this simple case exercises all of the PN's logical circuitry. The second point is that the maximum response time of 42.3 mS is a promising value because it indicates that the PN can be used in real-time applications. Although both the minimum and maximum are promising, it is unlikely that either of these performance extremes would result from any real application.

SYSTEM LOAD	NUMBER OF CN INPUTS CHANGED (I)	NUMBER OF INTERNAL CNs AFFECTED (N)	AVERAGE NUMBER OF ENTRIES IN WEIGHT TABLES (L)	ESTIMATED RESPONSE TIME ($\bar{R}$)
MINIMUM	1	1	1	2.6 μS
LIGHT	5	5	20	108 μS
	5	10	20	212 μS
MEDIUM	15	32	256	8.2 mS
	400	32	256	8.6 mS
HEAVY	4000	50	200	13.6 mS
	4000	50	400	23.6 mS
MAXIMUM	10560	64	512	42.3 mS

Table 4 - Some Examples of Estimated PN Performance. The exact parameter values are representative examples reflecting the specified loads.

In comparing the times shown in Table 4 with prior expectations, one must remember that these estimates are based on computations using higher-order predicates. It is dangerous to compare these estimates with those of systems that do not use higher-order predicates as they provide substantially less computing power than the PN proposed here. In fact, a major accomplishment of this thesis is the development of an efficient computational vehicle using higher-order predicates.

One can also observe from Table 4 the performance impact of varying the three main application parameters. As can be seen from the Medium load data, varying I does not have a substantial impact on the response time. On the other hand, as can be seen from the Heavy load data, varying the average number of entries in the Weight Tables has a nearly linear effect on $\bar{R}$. Similarly, increasing the number of affected CNs also has a large effect on the response time. None of these insights are particularly surprising given the form of Equation 4. Nevertheless, these results dramatize these conclusions in a practical way.

There are other interesting generalizations that can be obtained from examining the data shown in Table 4 or from direct calculations of Equation 4 for different loads. To draw statistically based PN performance conclusions, it will be necessary to generate a simulation program for the proposed microarchitecture, but first an understanding of specific application requirements must be obtained. This simulation program will provide several benefits including microarchitectural verification, architectural optimization, BH network traffic analysis and application algorithm evaluation. In fact, one of the first goals of any future efforts in this research area should be the development of a suitable simulation program.

The final task in analyzing the PN's performance is comparing it to more conventional computers. Two different performance measures have to be considered to show the value of the PN. One is raw performance and the other is the cost/performance ratio. The following discussion examines two distinctly different conventional computer types and compares both to the PN based CC. One of these computers is microprocessor based and is designed to occupy minimum ECB real-estate. The second is composed of state-of-the-art Digital Signal Processing (DSP) components and represents a high performance, highly optimized, alternative to the PN. Explicit consideration of both of these extremes is necessary as each has been proposed as a viable alternative to the more specialized PN [CaG86, CrT85, HeG]. In fact, the DSP approach is being actively pursued by several commercial firms including IBM, Texas Instruments and Hecht-Nielsen Neurocomputer.

Throughout the following comparisons, only the inner loop of the PN's computation is considered. Additional comparisons would almost certainly require detailed simulations of the proposed systems. The inner loop of the PN is composed of the 8-bit multiplication, the 16-bit multiplication and the 41-bit accumulating addition. In the PN, all of these operations are pipelined and so the required time for an iteration is 1000 nS as shown in Equation 4.

The first computer architecture to be compared to the PN is simply a 68020 microprocessor connected to a single DRAM. This is an unlikely combination because a small amount of "glue" logic would almost certainly be required. Neglecting this complication, its "cost" is easily compared to that of the proposed PN based system and it is easily replicated. The 68020 performs a 16×16 integer multiplica-

tion in 28 clock cycles (worst case) and performs a 32-bit addition in 6 cycles (worst case). Beyond this, there are several assumptions that must be made to estimate the 68020 execution time for the inner loop. First, assume that all program memory references can be performed in parallel with the computing. On the other hand, data operand fetches cannot be performed in parallel so their related data access times must be added to computational times to determine total time estimates. Next, assume that the address computation for any *data* operand requires the 68020 to fetch a 16-bit address offset from the DRAM and perform one single integer addition [14]. Finally, assume use of a 12 M-Hz 68020 with the 100 nS cycle time, 8-bit wide DRAM that was proposed earlier.

Given these assumptions, fetching the three data operands requires a total of 1000 nS memory access time plus 500 nS to perform the three address additions. The two multiplications and the double-precision accumulating addition take an additional 5667 nS. This yields a total time 8167 nS required per inner loop iteration. As the PN performs this operation in 1000 nS, it has a factor of eight performance benefit over the 68020 in this application. The 8-to-1 advantage is actually a lower bound, as there are many application specific, efficiency related, features in the PN that are not available with the 68020. Using IC count as a gross measure of cost, a similar factor is obtained for the cost/performance comparison.

The next conventional architecture to be considered uses DSP logic that is tuned for high-performance in the PN application. The following analysis shows that the PN compares favorably against computers on this extreme of the performance

[14] This assumption is very generous as it is likely that significantly higher overhead will be required for data address calculations. This results in a decrease of the predicted PN performance advantage over the 68020.

spectrum. The specific system to be considered uses the Weitek ACCEL 8000, 32-bit wide, integer processor chip set that has a 100 nS cycle time. This chip set is composed of an *Integer Processing Unit* and a separate *Program Sequencing Unit*. Together, they perform 32-bit integer arithmetic operations in just a single clock cycle. To use these components effectively, data and program memory must be physically separated and both of the memories should have cycle times in the 50 nS range. The program portion of the memory is composed of four 8K $\times$ 8-bit SRAMS which are currently commodity items. This memory configuration provides the 32-bit program word width used by the Weitek chips. The data memory requires a total capacity of 4 M-bits and it is composed of sixteen 32K $\times$ 8-bit (state of the art) SRAMs. This yields a total IC count of 22 for the DSP solution. Using IC count as a rough measure of cost, this implies a factor of eleven cost advantage of the PN based system over the DSP solution.

The performance estimate for the DSP system is derived similarly to the 68020 performance estimate. In fact, all the same assumptions apply. Program memory fetches are completely hidden. Data fetches are assumed to require the same *address offset fetch - integer addition - data fetch* sequence. Furthermore, each of these operations takes one separate DSP clock cycle. Therefore, fetching the three data operands requires 900 nS. Performing the two multiplications and one double-precision addition takes four clock cycles (400 nS). This means that the DSP system completes each inner loop iteration in 1300 nS and it is slower than the (less costly) PN based system. As a specific result, the PN based system has an approximate cost/performance advantage of 14-to-1 over the DSP solution.

6. INITIALIZATION, COMPUTING AND LEARNING

There are three distinct operational phases used by the PN based CC in solving application problems. They are initialization, operational computing and learning. Although the PN is the cornerstone during all of these processes, the CC's on-board microprocessor and I/O controller (shown in Figure 3) also play crucial roles. This chapter describes how these three major components cooperate together to allow efficient execution in each of the CC's computational phases. Explicitly describing these interactions further validates the proposed architecture by establishing its suitability to the set of target applications described in Chapter 1.

Throughout this chapter, the roles and functions of the on-board microprocessor are frequently discussed. In fact, the microprocessor could be replaced by some other *general purpose computational agent* that is capable of the same functions. Although this thesis assumes the use of a microprocessor for this computational agent, the actual choice may be made at a later date without compromising the validity of the following discussion. Furthermore, no lack of generality results from this assumption.

Like conventional computers, before the CC can actually begin operational computing, it must first be initialized with the problem set to be solved. In conventional computers, this initialization includes the loading of the program to be executed as well as the specific data set for the current problem. Analogous functions

are required within the CC, although the "program" construct is markedly different. In conventional computers, the program defines the sequence of instructions that the computer executes. In the case of the CC, the "program" is composed of memory images of all of the data structures, the connections and weights, for each PN in the system.

Specifically, the CC program is segmented between data structures internal to the PN and those data structures that are located in each PN's external memory. The external data structures that require initialization are the Input Data Buffer, the 2-codons Used Tables and the Weight Tables. Internal PN program initialization consists of setting the BH level base addresses and setting the Table of Last *OUT* values. By initializing the Input Data Buffer and Table of Last *OUT* Values, correct generation of new *OUT* values is possible, even if only a single new input is received. Furthermore, by setting the BH level base addresses at run-time, there is greater flexibility in binding CC program portions to PNs that are actually physically available. In comparison to fixed BH level base addresses, this run-time binding supports fault tolerant computing within the CC.

Given these functional requirements, it is useful to consider the mechanics of the initialization process. The on-board microprocessor directs and controls the initialization sequence and uses the I/O controller to perform any required low-level I/O accesses [15]. (The I/O controller is also responsible for generating the control signals required to refresh the DRAM components.) From a high level perspective, the microprocessor must have access to a "program image" that includes each PN within

[15] A likely configuration of a CC would use a *host computer* for the highest level of control and I/O sequencing. The role of the host and its specific interactions with the CC's logic boards are not covered in this thesis.

the CC. This program image is stored on a bulk-storage device, such as a disk drive. As a result, the microprocessor must behave in its conventional role as a file system manager [16]. More specifically, after it first informs the I/O controller which PN is being initialized, the microprocessor starts the transfer of a portion of the program image from the disk drive to the I/O controller. The I/O controller is then directly responsible for passing the data to the appropriate memory structure for the target PN. Access to the PN's internal registers is accomplished via the PIO mechanisms defined earlier. To allow efficient access to memory external to the PN, each DRAM is directly connected to the I/O controller.

Given this initialization mechanism, it is interesting to consider initialization performance. As derived in Chapter 4, each PN requires initialization of approximately 2 M-Bits to define all of its memory structures. Assuming the use of a disk drive with a 1 M-Byte/second transfer rate, it follows that the complete initialization of a single PN will require approximately 250 mS. Using this result and assuming just a single I/O path between the disk drive and the I/O controller, initialization of the 128-PN board proposed in Figure 3 will require 32 seconds. It is likely that initialization times of this magnitude would be a problem in many applications but it would be more serious if multiple CC logic boards were to use the same disk drive. Therefore, system level I/O solutions would be required to lessen the initialization time.

---

[16] Throughout this chapter, the assumption is made that the application problem has already been translated to a form that the PN uses during execution. In fact, a compilation process, functionally similar to that used in conventional computers, would be required to translate the problem from a human specification to the PN usable form. This compilation process is a major concern in this general research area, but it is not covered here.

From a software perspective, mechanisms supporting learning capabilities can be used to minimize initialization requirements as explained below. Other solutions might include the use of faster bulk-storage devices and multiple bulk storage devices operating concurrently. Although either the faster disks or multiple parallel disks would provide benefits, the multiple disk solution is preferable because it is more extensible. It is also more consistent with the parallel computation paradigm of the CC. Incidentally, this sort of parallel concurrent access to different disk drives is similar to the "file striping" function that has recently become available on high-speed conventional computers. Nevertheless, there is one complication when multiple disk drives are controlled and accessed by separate processors. This architectural feature would directly imply the need for a distributed file system of some kind. This portion of the I/O design will require significant future effort as potential design problems in this area are numerous and nontrivial.

After initialization is complete, the CC begins execution by setting the START signal of each PN in the system. Preceding chapters describe in detail how the PN operates during this phase, but the roles of the microprocessor and I/O controller need to be considered. The function of the I/O controller is intuitively obvious as it simply routes any non-local communications to and from other portions of the system. The methods that the I/O controller uses to accommodate these communications are not covered in this thesis. Significant analysis of many communication related parameters will have to be performed before substantial progress can be expected in this area.

The role of the microprocessor during operational computing is obvious. It is responsible for determining when the application has completed. It then halts the PNs and extracts the "answer" from appropriate PN data structures. The microprocessor may use several criteria to determine completion. One could be monitoring traffic frequency on selected BH networks. When the frequency drops below some threshold for a specified amount of time, then it could infer that the application has "settled" to a solution. Similarly it could monitor the message contents on a selected BH network to look for a token that could be used to signal completion. Finally, it could be signaled by some other logic element in the CC. The general-purpose programmable capabilities of the microprocessor makes it well suited in this case as completion conditions will be application specific.

Extracting the answer from the PN is another area where the flexibility and computational power of the microprocessor is useful during the CC's operational computing phase. The application "answer" will probably be derived, after the PN has been stopped, from information contained in the Table of Last *OUT* Values. The PIO capabilities of the PN are used to read the *OUT* values associated with any given CN. After fetching any required *OUT* values, the microprocessor executes an arbitrarily complex algorithm to convert the answer to the desired form. As the algorithmic translation is extremely application specific, this is also an ideal use for the microprocessor in the CC's computational hierarchy.

The final purpose of this chapter is to show how the CC architecture supports learning algorithms and how these capabilities may be used to minimize initialization requirements. Before describing the actual mechanisms that support CC based

learning, it is necessary to examine some conceptual details of the generalized process that will be employed. As discussed in Chapter 1, significant effort is being expended to develop automatic learning algorithms for neural network based computer systems and there is a wealth of related literature that is not covered here. In the purest form, automated learning procedures rely exclusively on trial and error methods. As with humans, when the CC computes an answer during the learning process, it must be compared with an expected answer for the given input. In human learning, discrepancies between expected and actual results are explained and then the subject repeats the process. In a similar manner, the CC uses some algorithmic method to selectively modify the connections between, and inputs to, specified CNs. The specific algorithms used during this phase are still in the early stages of development, so flexibility has to be retained in this regard [17].

Given this general formulation for CC based learning, the functional requirements of the different major components become clear. As with the two earlier phases of CC operation, the microprocessor plays a key role in the learning process. Specifically, it is responsible for comparing the actual "answer", as obtained above, with the known expected answer. The required examination can be performed when the PN is halted or during PN execution if the PN's HOLD signal is used. In either case, the microprocessor must then execute an algorithm to determine exactly which elements of the PN data structures need to be modified and how much they need to be changed. Depending on the locality of the learning algorithm, with respect to

---

[17] There are a significant number of major considerations that must be addressed to support learning algorithms on the CC. Unfortunately, this thesis presents only a broad overview of *how* the PN could be used in these applications. As a result, differentiation between types of learning algorithms and analysis of their resource requirements is not possible, though the mechanism proposed here for learning should cover the important areas

microprocessors and related PNs, one or many of the CC's microprocessors could participate in the learning oriented calculations.

Once it has been determined which specific elements of a PN's data structure require modification, the microprocessor would selectively write only those memory elements. In this way, complete memory initialization overhead is not incurred between separate trial and error attempts.

The logical mechanism that supports learning uses selective update of PN data structures. As was implied earlier, this selected update mechanism could also be of significant use in minimizing I/O overhead associated with switching between problem sets. If separate problems that are executed sequentially can be formulated in such a manner that they share PN data structures, then the microprocessor can perform partial update of these memory structures. To facilitate this function, it may be useful to develop a concise logical protocol to specify exact update actions. The development and use of this type of protocol could prove to be beneficial when applications problems are similar to each other and a minimization of I/O overhead is critical.

7. SUMMARY AND CONCLUSION

The basic goal of this thesis is to define a computer architecture and its surrounding system that is capable of solving certain problems more cost-effectively than conventional computers can. Target applications for the proposed system are characterized by the ability to take advantage of massive computational parallelism. These include artificial intelligence applications such as image recognition and natural language understanding. To achieve cost-effectiveness, a radically different computer architecture was proposed. The computational model used by the new architecture was based on a derivation of a biologically accurate model of neural systems. The neural model was refined with the intent of reducing the interactions to a set that could be mimicked using digital computer techniques.

Given the proposed neuron based model, a generalized computer architecture was defined that derived many properties from its biological counterparts. This portion of the thesis was directly based on prior and ongoing research at OGC and, to some extent, is just a directed collection of other researchers' work in this area. The defined computer system is called a *Connection Computer* and its biological counterpart is the nervous system including the brain. The Connection Computer is composed of many *Connection Nodes* which are analogous to individual neurons. Connection nodes use the *Broadcast Hierarchy* as their fundamental communication mechanism and this provides facilities that have similar characteristics to biological

systems. The computational model uses higher-order predicates called 2-codons. This choice substantially increases the computational power and circuit complexity of the proposed system.

After developing and summarizing the specific computational model, a detailed microarchitecture was proposed that implements the specified functions using digital computer technology. Designing an efficient microarchitecture required the definition of the *Physical Node* as a group of intimately related Connection Nodes. The microarchitectural definition portion of this thesis represents a significant contribution to this research area as this is the first effort at detailed design of this type of computer system. As such, this practical viewpoint exposes complications and weaknesses that are not visible when considered from a more theoretical perspective and this represents a major contribution of this thesis.

The proposed microarchitecture is admittedly non-optimal in some respects, as would probably be expected given that this is the first detailed proposal. One weakness of the proposal is that it may have fixed some architectural parameters in silicon that are better left as application variables. These might include the number of BH levels, the size of the different BH levels, DRAM bandwidth and even the number of CNs per PN. The choice of fixing these variables was deliberate and was made to put physical bounds on the design, so that accurate silicon resource and performance predictions could be made. It may be desirable in future efforts to consider leaving these parameters as application variables.

Although the proposed microarchitecture is non-optimal in some respects, it is superior to any other microarchitecture considered. In fact, several alternatives

were considered at both the computational architecture level and at the microarchitectural implementation level. Many options were considered that initially seemed advantageous but after detailed analysis were found to lead to serious implementation problems. One recent proposal was considered that only computes 2-codons that are affected by a given input change. Although the proposed method could provide higher performance in some load situations, a fully general implementation (with the capabilities of the PN proposed here) would result in the need for over 90 M-bits of memory per PN. By severely restricting the number of 2-codons that an input could affect, the required memory capacity could be lowered to a realistic value. This restriction is contrary to an original supposition of the PN's communication and computational capabilities. Furthermore, it it not known whether computational loads (eventually encountered by the PN) will favor the PN as defined or as supposed. The moral here is - beware of proposed alternatives until their exact design and system level effects are detailed.

To establish the viability and value of the proposed microarchitecture, careful analysis was presented that addressed the technological viability and performance of the proposed system. It was shown that the PN-based microarchitecture, exactly as defined, could be designed and fabricated using currently available digital manufacturing techniques. This result, in and of itself, is of great value to others in this research area as it shows that an efficient execution vehicle could be available to them in the near future. This analysis also provides rough cost/performance estimates of such a vehicle.

To further their anticipation, performance analysis was presented that shows the expected performance available from the proposed system. As part of this analysis, an algebraic expression was derived that yields the response time of the PN, given specific input stimuli. This algebraic function will be of significant use to other researchers who are examining and developing applications for the CC. More specifically, it is not known whether the estimated performance of the PN is acceptable in all target applications or if substantial performance improvements will be required. If higher performance is necessary, then a reconsideration of some of the fundamental PN design decisions must be made. Specifically, as the external memory bandwidth was the limiting factor in the performance estimates, some way of minimizing PN external references could be required. In this respect, the choice that would yield most performance benefit would be to bring all of the memory on-chip. This decision would radically improve the expected performance and should be possible as technology improvements allow higher levels of logic integration per silicon area. If the memory is moved completely into the PN, many facets of the proposed design may require modification as it is currently optimized for relatively slow memory references. In particular, content-addressable memory could take the place of more memory capacity intensive pointer structures.

The performance and cost/performance ratio of the PN was compared to that of more conventional computers. Using assumptions that were generous to the conventional computers, the PN was shown to be desirable from both perspectives. Specifically, its performance was shown to be above that of a significantly more costly, high performance, DSP-based computer system. The PN's cost performance

advantage was conservatively estimated to be between a factor of eight and fourteen over that of conventional solutions.

Finally, some discussion was presented that showed how the proposed system would actually operate from a high-level system perspective. This information was presented to show that there are no major holes in the CC system that would prohibit its effectiveness. This discussion could also act as a tutorial to show how the CC performs the operations that are expected from more conventional computers.

References

[Bai85] Bailey, J., *Mapping Virtual Networks to Physical Networks*, Dept. of Computer Science, Oregon Graduate Center, December 1985. Unpublished Manuscript.

[BaH86] Bailey, J. and Hammerstrom, D., "How to Make a Billion Connections," Tech. Report CS/E-06-007, Dept. of Computer Science/Engineering, Oregon Graduate Center, Beaverton, Oregon, July 1986.

[CaG86] Carpenter, G. and Grossberg, S., "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," in *Computer Vision, Graphics, and Image Processing*, 1986. In press.

[CrT85] Cruz-Young, C. A. and Tam, J. Y., "NEP: An Emulation-Assist Processor for Parallel Associative Networks," IBM Palo Alto Scientific Center, June 1985.

[Ham86a] Hammerstrom, D. W., *Connectionist VLSI Architectures*, Oregon Graduate Center, Beaverton, OR, April 1986.

[Ham86b] Hammerstrom, D., "A Connectivity Analysis of Recursive, Auto-Associative Connection Networks," Tech. Report CS/E-86-009, Dept. of Computer Science/Engineering, Oregon Graduate Center, Beaverton, Oregon, August 1986.

[Ham86c] Hammerstrom, D., "A Connectionist/Neural Network Bibliography," Tech. Report CS/E-86-010, Dept. of Computer Science/Engineering, Oregon Graduate Center, Beaverton, Oregon, August 1986.

[HeG] Hecht-Nielsen, R. and Gutschow, T., Sensor Processing using Artificial Neural Systems.

[Hop82] Hopfield, J. J., "Neural networks and physical systems with emergant collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79(April 1982), pp. 2554-2558.

[KaS81] Kandel, E. R. and Schwartz, J. H., in *Principles of Neural Science*, Elsevier/North-Holland, New York, 1981.

[KPT82] Koch, C., Poggio, T. and Torre, V., "Micronetworks in Nerve Cells," in *Competition and Cooperation in Neural Nets*, vol. 45, S. Amari and M. Arbib (ed.), Springer-Verlag, Berlin, 1982, pp. 105-110. Chapter 6.

[KuN77] Kuffler, S. W. and Nicholls, J. G., *From Neuron to Brain*, Sinauer Associates, Inc., Sunderland, Massachusetts, 1977.

[Mar70] Marr, D., "A Theory for Cerebral Neocortex," *Proc. Roy. Soc. London*, vol. 176(1970), pp. 161-234.

[MGL86a] Maxwell, T., Giles, C. L., Lee, Y. C. and Chen, H. H., "Transformation

Invariance Using High Order Correlations in Neural Net Architectures," *Proceedings International Conf. on Systems, Man, and Cybernetics*, 1986.

[MGL86b] Maxwell, T., Giles, C. L., Lee, Y. C. and Chen, H. H., "Nonlinear Dynamics of Artificial Neural Systems," in *Neural Networks for Computing*, American Institute of Physics, 1986.

[MiP69] Minsky, M. and Papert, S., *Perceptrons*, The MIT Press, Cambridge, MA, 1969.

[Per83] Perkel, D. H., "Functional role of dendritic spines," *J. Physio., Paris*, vol. 78(1983), pp. 695-699.

[Pos78] Posner, M. I., *Chronometric Explorations of Mind*, L.E. Erlbaum Associates, Hillsdale, NJ, 1978.

[RHW85] Rumelhart, D. E., Hinton, G. E. and Williams, R. J., "Learning Internal Representations by Error Propagation," ICS Report 8506, Institute for Cognitive Science, La Jolla, CA, September 1985.

[RuM86] D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1 and 2, Bradford Books/MIT Press, Cambridge, MA, 1986.

[SeR86] Sejnowski, T. J. and Rosenberg, C. R., "NETtalk: A Parallel Network that Learns to Read Aloud," JHU/EECS-86/01, The Johns Hopkins Univ. Elec. Eng. and Comp. Sci. Tech. Rpt, 1986.

[WeE85] Weste, N. and Eshraghian, K., *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley, 1985.

BIOGRAPHICAL NOTE

The author was born the 26th of September, 1956, in Evanston, Wyoming. He moved with his family to the San Fernando Valley where he spent the rest of his childhood. He graduated from Monroe High School in 1974 and then moved to Portland, Oregon to attend Reed College. He received a B.A. in Chemistry from Reed College in 1979. He began part-time work on his Masters at OGC in 1981.