7-2008

FPGA ACCELERATION OF A CORTICAL AND A MATCHED FILTER-BASED ALGORITHM

Kenneth Rice
Clemson University, krice@clemson.edu

FPGA ACCELERATION OF A CORTICAL AND A MATCHED FILTER-BASED ALGORITHM

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Kenneth Lee Rice
August 2008

Accepted by:
Tarek Taha Committee Chair
Robert Schalkoff
Adam Hoover

ABSTRACT

Digital image processing is a widely used and diverse field. It is used in a broad array of areas such as tracking and detection, object avoidance, computer vision, and numerous other applications. For many image processing tasks, the computations can become time consuming. Therefore, a means for accelerating the computations would be beneficial. Using that as motivation, this thesis examines the acceleration of two distinctly different image processing applications. The first image processing application examined is a recent neocortex inspired cognitive model geared towards pattern recognition as seen in the visual cortex. For this model, both software and reconfigurable logic based FPGA implementations of the model are examined on a Cray XD1. Results indicate that hardware-acceleration can provide average throughput gains of 75 times over software-only implementations of the networks examined when utilizing the full resources of the Cray XD1. The second image processing application examined is matched filter-based position detection. This approach is at the heart of the automatic alignment algorithm currently being tested in the National Ignition Faculty presently under construction at the Lawrence Livermore National Laboratory. To reduce the processing time of the matched filtering, a reconfigurable logic architecture was developed. Results show that the reconfigurable logic architecture provides a speedup of approximately 253 times over an optimized software implementation.

ACKNOWLEDGMENTS

I would like to acknowledge the summer student support at Lawrence Livermore Laboratory. I would like to also acknowledge grants from the Air Force Research Laboratory (including the AFRL Information Directorate) and a National Science Foundation CAREER award. This work was also supported in part by a grant of computer time from the DOD High Performance Computing Modernization Program at the Naval Research Laboratory. Most especially, I would like to acknowledge and thank Tarek Taha, Abdul Awwal, Christopher Vutsinas, Robert Schalkoff, and Adam Hoover for their help and support in this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued)

Figure Page

CHAPTER ONE

INTRODUCTION

Digital image processing is a widely used and diverse field. Techniques of digital image processing are used in a broad array of areas such as tracking and detection, object avoidance, computer vision, and numerous other applications. For a number of image processing tasks, the time required to complete the computations can be expensive. Therefore, a means for accelerating the computations would be beneficial (for instance to help meet real-time requirements).

Inherent within many digital image processing computations and algorithms is a degree of parallelism. The inherent parallelism of these computations points towards a hardware solution. Hardware solutions, in particular reconfigurable logic (such as field programmable gate arrays (FPGAs)), can be crafted in such a way that they would take advantage of the inherent parallelism within the image processing computations. Therefore, the motivating factor behind this research is the use of FPGAs to accelerate image processing applications.

This thesis examines two distinctly different image processing applications for acceleration. The first image processing application that is examined is a neocortex inspired cognitive model. The neocortex is the outer layer of the human/primate brain where cognition and learning take place. Recent research in this area has resulted in new models for information processing inspired by the neocortex [3,14,19,29]. Several of these models are based on hierarchical Bayesian networks and describe the brain as a

hierarchical device that computes by performing sophisticated pattern matching and sequence prediction. A Bayesian modeling framework incorporates several of the properties suggested for the neocortex [16]. These include a hierarchical structure of uniform processing elements, invariant representation and retrieval of patterns, auto associative recall, and sequence prediction through both feed-forward and feedback inference between layers in the hierarchy.

Large scale versions of these models have the potential for significantly stronger inference capabilities than current computing systems [13]. Given the simple computation within the nodes of the models, hardware-acceleration of these systems holds significant promise to speed up these models. This could enable real-time implementations of large scale versions of these models to solve interesting problems. This thesis examines the acceleration of a recent neocortex inspired model [14] geared towards image recognition. Results indicate that hardware-acceleration can provide average throughput gains of 75 times over software-only implementations of the networks examined when utilizing the full resources of the Cray XD1.

The second image processing application examined in this thesis is matched filter-based position detection. The matched filter-based position detection technique examined here is at the heart of the automatic alignment (AA) currently being tested in the National Ignition Faculty presently under construction at the Lawrence Livermore National Laboratory. The matched filtering being performed here is currently very expensive. To reduce the processing time, a hardware solution for the matched filtering would be profitable. Therefore, a reconfigurable logic architecture for the matched filtering was

developed. This resulted in a speedup of approximately 253 times over a software implementation.

Chapter 2 analyzes the performance scaling of both hardware and software implementations of a neocortex inspired cognitive model on a Cray XD1. The Cray XD1 provides an ideal platform for accelerating large scale versions of these models using hardware. This is because a Cray XD1 contains a large number of FPGAs and general purpose processors connected through a low latency, high bandwidth communication network. Chapter 3 discusses an approach to accelerate matched filter-based position detection computations using an FPGA. Chapter 4 concludes the thesis.

CHAPTER TWO

ACCELERATION OF A NEOCORTEX INSPIRED COGNITIVE MODEL

While conventional von Neumann architectures excel at logical applications such as spreadsheets, signal processing, and modeling scientific experiments, they generally perform quite poorly on cognitive applications. These include tasks such as speech recognition, computer vision, textual and image content recognition, robotic control, and making sense of massive quantities of data [13]. These cognitive applications are found in many domains including national security, medicine, transportation, industry, and science. Biological systems excel at these applications largely due to differences in the underlying architectures and algorithms utilized compared to traditional computers [4].

In the human brain, cognition and learning are primarily handled by the neocortex. The neocortex is the outer layer of the human/primate brain and consists of a fairly uniform structure. Research over the past decade has shed much light into the structures and functions of the neocortex [9]. Based on this understanding, several mathematical models of the neocortex have been proposed recently [3,14,19,29]. These models are significantly different from traditional artificial neural networks. They provide insights into the possible workings of the neocortex, and agree with many experimental results [9]. The newer models [14, 19] are based on hierarchical Bayesian networks and describe the brain as a hierarchical device that computes by performing sophisticated pattern matching and sequence prediction. A Bayesian modeling framework incorporates several of the properties suggested for the neocortex [16]. These include a

hierarchical structure of uniform processing elements, invariant representation and retrieval of patterns, auto associative recall, and sequence prediction through both feed-forward and feedback inference between layers in the hierarchy.

Large scale versions of these models have the potential for significantly stronger inference capabilities than current computing systems [13]. The hierarchical structure of uniform computations within the Bayesian models provides scope for large amounts of parallelism. The computations within each node of these models are generally quite simple. Given the large amounts of inherent parallelism and the simplicity of the nodes in these models, hardware implementations allow for a high density of node computations to take place in parallel. Therefore hardware-acceleration of these systems holds significant promise to speed up these models. This would enable real-time implementations of large scale versions of these models to solve interesting problems.

In this chapter, the performance scaling of both hardware and software implementations of a neocortex inspired cognitive model on a Cray XD1 is analyzed. The Cray XD1 provides an ideal platform for accelerating large scale versions of these models using hardware because it contains a large number of FPGAs and general purpose processors connected through a low latency, high bandwidth communication network. This enables the investigation of large scale hardware-accelerated implementations of neocortex inspired cognitive models, and also a comparison to large scale software implementations of the models. The Cray XD1 utilized had 432 dual core 2.0 GHz Opteron processors and 144 Virtex II Pro FPGAs (part XC2VP50). The main advantages of FPGA implementations are that several components of the cognitive models can be

evaluated in parallel, and that specialized hardware components can be designed to speed up computation as compared to software implementations.

The hierarchical Bayesian network model based on the neocortex developed by George and Hawkins [14] is accelerated. The model implements invariant pattern recognition as seen in the visual cortex using a collection of hierarchically connected nodes that perform similar computations. George and Hawkins [14] demonstrate that it performs well at recognizing a sequence of objects under various transforms. This model serves as the foundation for a commercial design currently being developed by Numenta [17] for a range of cognitive applications.

Several networks of varying complexity based on the George and Hawkins model are implemented. From an analysis of the performance of these networks, an estimation is made for the throughput of the Cray XD1 for larger networks that would utilize the full resources of the system. These larger networks could be network-of-networks, as described by Anderson and Sutton [5], where many simple cortical networks are linked together to model more complex functions. Results indicate that hardware-acceleration can provide average throughput gains of 75 times over software-only implementations of the networks examined when utilizing the full resources of the Cray XD1. Preliminary results of this study was published in [24].

Background

Hawkins' Framework

Hawkins presented [16] a theoretical framework describing the processes in the neocortex. This is the basis of the model implemented in this chapter. Hawkins describes the neocortex as a highly efficient pattern matching device [15] – as opposed to a computing engine. The brain learns by storing patterns and recognizes by matching incoming sensory data with learned patterns. It can recognize the same pattern under different conditions (invariance) much more efficiently than existing computer based systems. One example is the ability of the brain to recognize a face under different lighting conditions, or from different angles. Another example is the ability of a person to catch a ball thrown at him or her without much effort or thought. The brain can determine where to position the hand to catch a ball without complex calculations of velocity and wind direction. This is because it constantly matches (moment by moment) the ball's movements with observations from the past of other ball throws. The fact that it is able to match patterns even though this throw is unique from all previous throws (different ball and wind velocities) demonstrates invariant pattern matching.

It is well known that the neocortex is organized hierarchically and that it consists of a uniform computational fabric [10]. Hawkins states that activities in the bottom most layers of this hierarchy are performed without need for "conscious thought". This includes regular activities such as walking. Thus patterns that do match in the brain are

dealt with at this level. Unknown patterns are sent to higher levels to be dealt with by "conscious thought". An example of this may be walking on a slippery or unfamiliar surface.

After training is complete, most artificial neural networks make decisions without using feedback from previous inputs. Hawkins states that since the inputs to the brain vary with time, a model of the brain needs to account for this dynamic input stream. Thus a model of the neocortex needs to recognize patterns not only spatially but also temporally. This would enable the neocortex model to constantly predict the next input in a sequence based on current and past inputs. For instance, if the neocortex model sees the inputs A, B, and C, it will predict that the next input will be D. This prediction is performed by layers higher in the hierarchy than the input layer. The predictions from the higher layers are sent back to the input layer through a series of feedback connections. When the future inputs do match the predictions, upper levels of the cortical hierarchy are not involved in the pattern analysis (for instance when walking normally, each step follows what the brain predicts will happen). In case the prediction does not match the input (hence a new input pattern is seen), higher levels of the hierarchy are brought into play.

This idea of feedback and prediction is one of the fundamental components of Hawkins' model. The mathematical model presented in [14] captures many of the properties presented by Hawkins in [16]. This model is being enhanced further by Numenta Inc. to capture more of the ideas presented in [16]. The main differences between this model and traditional artificial neural networks are the role of feedback

during recognition (this allows the model to have a temporal aspect in addition to a spatial aspect), the hierarchical structure, and the invariant pattern recognition. Other studies of large scale cortical models include [1, 4, 18]. Johansson et. al. [18] and Ananthanarayanan et. al. [1] are developing large scale models of the neocortex on parallel clusters (without hardware-acceleration). Anderson et. al. [4] are designing the models for a large scale version of the neocortex based on a network-of-networks. They are using a BSB attractor [2] for the cortical model. It is important to note that the current understandings of the neocortex are far from complete. Thus the current models of the neocortex, even though they match many experimental results, are still highly speculative. These models however do perform well in many real world cognitive applications, and thus are worth accelerating.

Model

George and Hawkins developed an initial mathematical model of the visual cortex [14] based on the framework described above. Their model utilizes a hierarchical collection of nodes that employ Pearl's Bayesian belief propagation algorithm [21]. Each node has one parent and multiple children (see Figure 2.1). Image information is fed in as input data to the bottom layer of nodes, and after a set of feed-forward and feedback belief propagations in the network, a final belief is available at the top level node. This belief is a distribution that indicates the degree of similarity between the input and the different items the network has been trained to recognize.

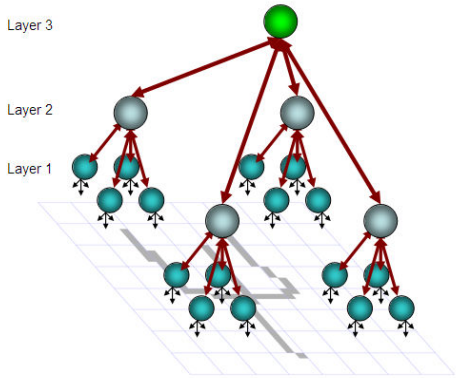

Figure 2.1. A simplified model of the Bayesian network in the George and Hawkins model. Input data is fed to the bottom layer of nodes.

The computational algorithm within each node of the model is identical and follows equations 1 to 6. Before a node starts computing, it receives belief vectors from its parent (π) and children (λ) as shown in Figure 2.2(a). The belief vectors from its children are all combined together as shown in equation 1. This combined belief vector from the children is then multiplied by an internal probability matrix, P_{xu} (generated in an offline training phase), and the belief vector from the parent (see equation 2). The multiplications are carried out element-by-element. A set of belief vectors are then generated for the parent and child nodes (equations 3 to 6). These output belief vectors are then transmitted to the parent and children of the node as shown in Figure 2.2(b). In this study, the implementation shown in [14] is utilized. This consists of three layers of nodes.

$$\lambda_{product}[i] = \prod_{child} \lambda_{in}[child][i] \tag{1}$$

$$F_{xu}[j][k] = \pi_{in}[j] \times P_{xu}[j][k] \times \lambda_{product}[k] \tag{2}$$

$$m_{row}[j] = max(m_{row}[j], F_{xu}[j][k]) \tag{3}$$

$$m_{col}[k] = max(m_{col}[k], F_{xu}[j][k]) \tag{4}$$

$$\lambda_{out}[j] = \frac{m_{row}[j]}{\pi_{in}[j]} \tag{5}$$

$$\pi_{out}[child][k] = \frac{m_{col}[k]}{\lambda_{in}[child][k]} \tag{6}$$

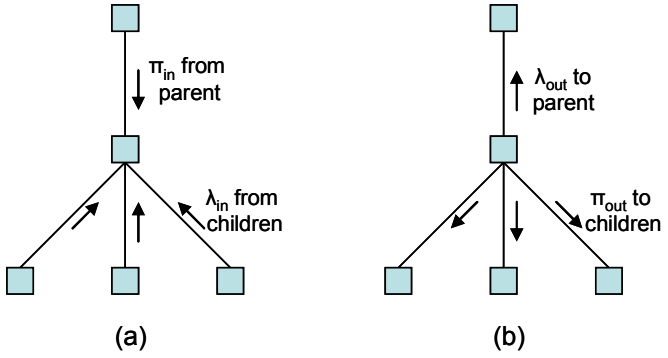

Figure 2.2. Belief transfer in the network (the squares represent computation nodes). (a) Gathering beliefs from parent and children nodes before node computation. (b) Distribution of beliefs to parents and children nodes after node computation.

Survey of Related Work

This chapter discusses the FPGA based hardware-acceleration of the neocortex inspired cognitive model presented in [14]. Although several computational models of the neocortex have been developed recently [3,14,19,29], the hardware-acceleration of such models is scarce. This study is the first hardware-accelerated implementation of the

George and Hawkins model as far as the authors know. FPGA acceleration of visual cortex models are seen in [8,12,27]. Torres-Huitzil et al. [27] implemented a bio-inspired model of visual perception of motion on an FPGA. Their model considered the interactions among the primary visual (V1), the middle temporal (MT), and the middle superior temporal (MST) areas of the brain. Their neuron based model with excitatory–inhibitory connectionist processing for 128x128 images achieved about a 100 times speedup over a software implementation on a Pentium 4 processor. Bouganis et. al. [8] examined the FPGA acceleration of a visual attention model in the primary visual cortex (V1). FPGA acceleration showed a 10 times speedup over a 3.2 GHz Pentium 4 processor. Furlong et. al. [12] implemented a visual brain circuit architecture (VBCA) on an Xilinx Virtex 4 FPGA and on a general purpose CPU. They demonstrated that the FPGA implementation gave a performance gain of 62 times over a general purpose CPU implementation.

Several authors have examined the acceleration of neural network processing through FPGAs [6,11,22], custom integrated circuits [7,13], and parallel computation [1,18]. The FPGA designs in [6,11,22] implemented feed-forward fully connected neural networks, while the design implemented in this thesis is a Bayesian network that operates through both feed-forward and feedback belief propagations. Atencia et al. [6] presented the implementation of Hopfield networks on an FPGA, where the number of inputs into a neuron and the number of neurons could be parameterized. Pearson et. al. [22] implemented a collection of neural processing elements on an FPGA to emulate leaky-integrate-and-fire neurons. Their system modeled a tactile sensory system for object

recognition and texture discrimination. Gao and Hammerstrom [13] proposed a simplified model of the neocortex based on spiking neurons and examined conceptual implementations of the model using future CMOS and CMOL technologies. Pournara et al. [23] presented an FPGA implementation of a Bayesian network for the reconstruction of gene regulatory networks. The objective of their system was to determine the connections between different nodes, as opposed to the training weights of each node in a pre-connected system. Starzyk et al. [26] proposed a biologically inspired classifier geared towards hardware-acceleration. Weinstein et al. [28] presented FPGA acceleration of detailed but small scale neural models.

Several researchers have explored the option of large scale implementations of neural networks. Boahen [7] proposed a collection of specialized mix-signal chips that would model a million neurons in the cortex at very low power consumption levels. Initial implementations of this work looking at the networking between components are presented in [20]. As an alternative approach for large scale modeling of neurons, Johansson et. al. [18] presented a parallel software-only implementation of spiking neural networks. They utilized a cluster of 442 dual Xeon processor based systems for their implementation and are able to model a mouse sized cortex. Ananthanarayanan and Modha [1] used a computationally efficient cortical simulator on a 32,765 processor BlueGene/L supercomputer to simulate a rat-scale cortical model based on spiking neural networks.

Implementation

Both parallel hardware-accelerated and software implementations on a Cray XD1 of the neocortex inspired cognitive model proposed by George and Hawkins [14] were developed. This platform consisted of 144 Xilinx Virtex II Pro FPGAs (part XC2VP50), 6 Xilinx Virtex 4 FPGAs, and 864 AMD Opteron 2.0 GHz cores (432 dual core processors). Each FPGA has access to high speed off-chip SRAM banks. On this system, a large network of nodes would be distributed across a set of processors and each processor would be able to take advantage of an FPGA for hardware-acceleration.

Data Optimizations

The data format and storage for the algorithm were optimized to accelerate the node computations. The computations listed in equations 1-6 are element-by-element matrix multiplications (as opposed to dot-products) and divisions. Hence there are no additions or subtractions needed. Thus to simplify the hardware implementation of the algorithm, a fixed point logarithm data representation is utilized instead of a floating point format. This enables the computations to be implemented using fast, highly efficient integer adders instead of slow, area consuming floating point multipliers and dividers. It also makes the nodes smaller and faster, thus allowing more nodes to be implemented in parallel and running at higher speeds on an FPGA. Utilizing smaller bit widths reduce the amount of logic necessary per node. As shown in [25], a bit width of

16 was shown to have sufficient accuracy for this model and is therefore utilized in this study.

The probability matrix (P_{xu}), utilized in equation 2, for the node computations is large and sparse. In the networks described in [14], these matrices had dimensions of 139×744 and 744×91 for the level 1 and 2 nodes respectively, with about 97% zeros. Given that the matrix operations in equations 1-6 are element-by-element, the P_{xu} matrix is accessed sequentially in the implementation of equation 2. Given also that the outcome of equation 2 is a zero when $P_{xu}[j][k]$ is zero, a significant fraction of computations can be avoided by encoding a string of zeros in this matrix. This implementation used a modified version of run-length encoding where only a string of consecutive zeros was encoded as a single zero followed by a number indicating the total number of consecutive zeros (as shown in Figure 2.3) to compress the matrix. This reduces memory storage requirements and speeds up the node computations.

Both the hardware-accelerated and software implementations utilized compressed P_{xu} matrices, while the fixed point logarithm data representation was used only in the hardware-accelerated implementation (primarily since floating point calculations are very expensive in FPGAs). The software implementation utilized a floating point data representation.

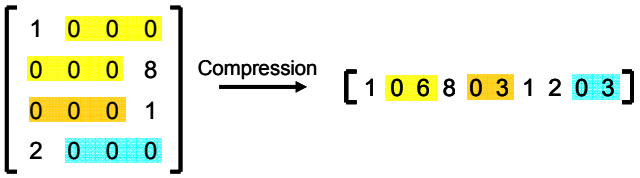

Figure 2.3. Data compression example.

Software Implementation

The software implementation of the model was developed in C and utilized MPI for communication between processors. The software implementation was accelerated by compressing each node's training matrix (P_{xu}). The models implemented have three levels of nodes (see Figure 2.4). Each level 2 node has four level 1 node children, while a level 3 node has all the level 2 nodes as children. In this design, the amount of communication between level 1 and level 2 nodes is significantly higher than between level 2 and level 3 nodes. This is because the π and λ belief vectors sent between the level 1 and level 2 nodes are larger, and there are more level 1 nodes than level 3 nodes connected to each level 2 node. In order to localize the heavier communication between nodes, each level 2 node and its four level 1 node children was grouped together (as shown by the boxed region in Figure 2.4) and partitioned the design based on these groupings.

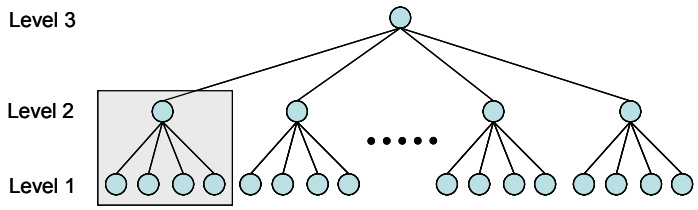

Figure 2.4. Network structure of model implemented. The box represents a grouping of nodes. The networks were partitioned based on this grouping.

Hardware Implementation

Each of the nodes in the network was implemented using a state machine. A similar grouping of nodes as in the software implementation was utilized – a level 2 node and its four level 1 children were grouped to form a processing element (PE). As shown in Figure 2.5, multiple processing elements can be placed on an FPGA, while a collection of FPGAs can operate in parallel. In this study, the level 3 node was implemented on a separate AMD processor (for the rest of this work this thesis refers to this as the root processor). Additional AMD processors (which this thesis calls host processors) were used as hosts to the FPGAs and acted as communications bridges between the nodes implemented on the FPGAs and the level 3 node implemented on the root processor. MPI was utilized for the communication between the root and host processors.

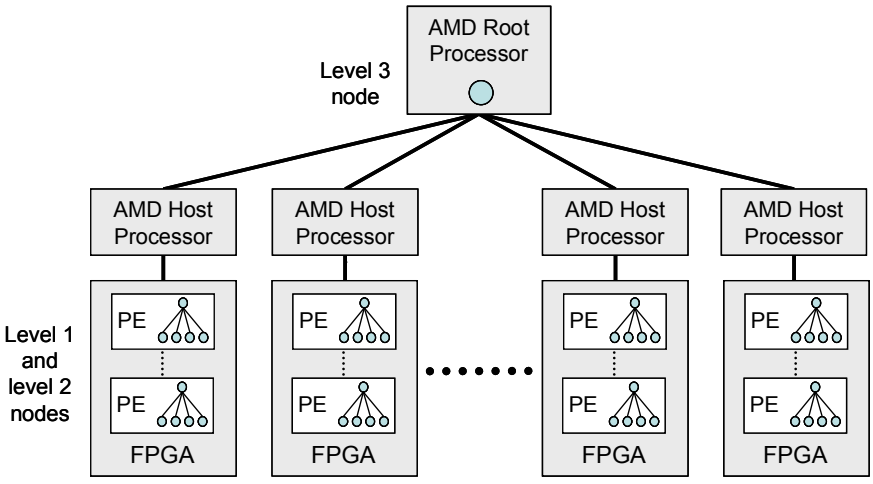

Figure 2.5. Hardware implementation of a network.

Figure 2.6 illustrates the connections between a collection of processing elements on an FPGA and shows the components within each processing element. Each processing element consists of a group of node state machines (four level 1 and one level 2 node) that are connected together through buffers. The buffers contain the λ and π belief vectors exchanged between nodes and are implemented using the FPGA's on-chip block RAM units. The internal data needed by the nodes (P_{xu}, m_{row}, and m_{col}) are also contained in block RAMs. The host processor sends the λ inputs to the level 1 nodes. It also sends the π inputs and receives λ outputs of the level 2 nodes.

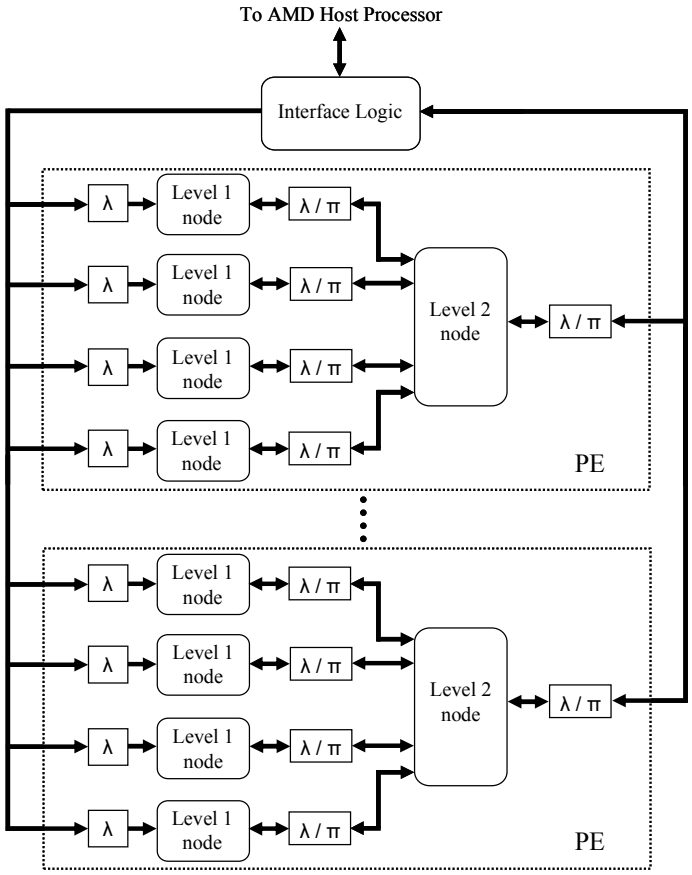

Figure 2.6. Processing elements on an FPGA.

The computations within each node as defined by equations 1 to 6, along with the optimizations listed in section 3.1, can be captured in a state machine. Given that nodes in different layers may have variations in the number of parent and children nodes, their state machines may also differ slightly. Figure 2.7 shows the state machine that describes the behavior of the level 1 nodes. The node state machines consist of three phases:

Phase 1: This is an initialization phase. The block RAMs holding the m_{row} and m_{col} outputs of equations 3 and 4 are initialized. These two arrays hold the maximum value of the rows and columns of the F_{xu} matrix respectively. Each level 2 node has multiple level 1 node children. Therefore, for this node, equation 1 is implemented, where the different λ_{in} beliefs are combined to form $\lambda_{product}$.

Phase 2: In the second phase, equations 2-4 are evaluated. This amounts to generating the F_{xu} matrix and storing the maximum of the rows and columns of this matrix in the m_{row} and m_{col} arrays. The entire F_{xu} matrix is never stored. Instead, as each value in the F_{xu} matrix is generated, it is compared against the appropriate element of the m_{row} and m_{col} arrays and is stored if the maximum value is encountered.

Phase 3: In the last phase, equations 5 and 6 are evaluated. The outgoing λ and π beliefs are calculated from the m_{col} and m_{row} arrays generated in the previous phase. Note that level 1 nodes do not need to compute π values as they do not have any children.

In Figure 2.7, equation 2 is evaluated in the "F_{XU} Computation" state, equation 3 in the "Write Max Row" state, and equation 5 in the "Write λ_{out}" state. In a level 2 state machine, additional states would be needed for equation 1 (in phase 1), equation 4 (in phase 2), and equation 6 (in phase 3).

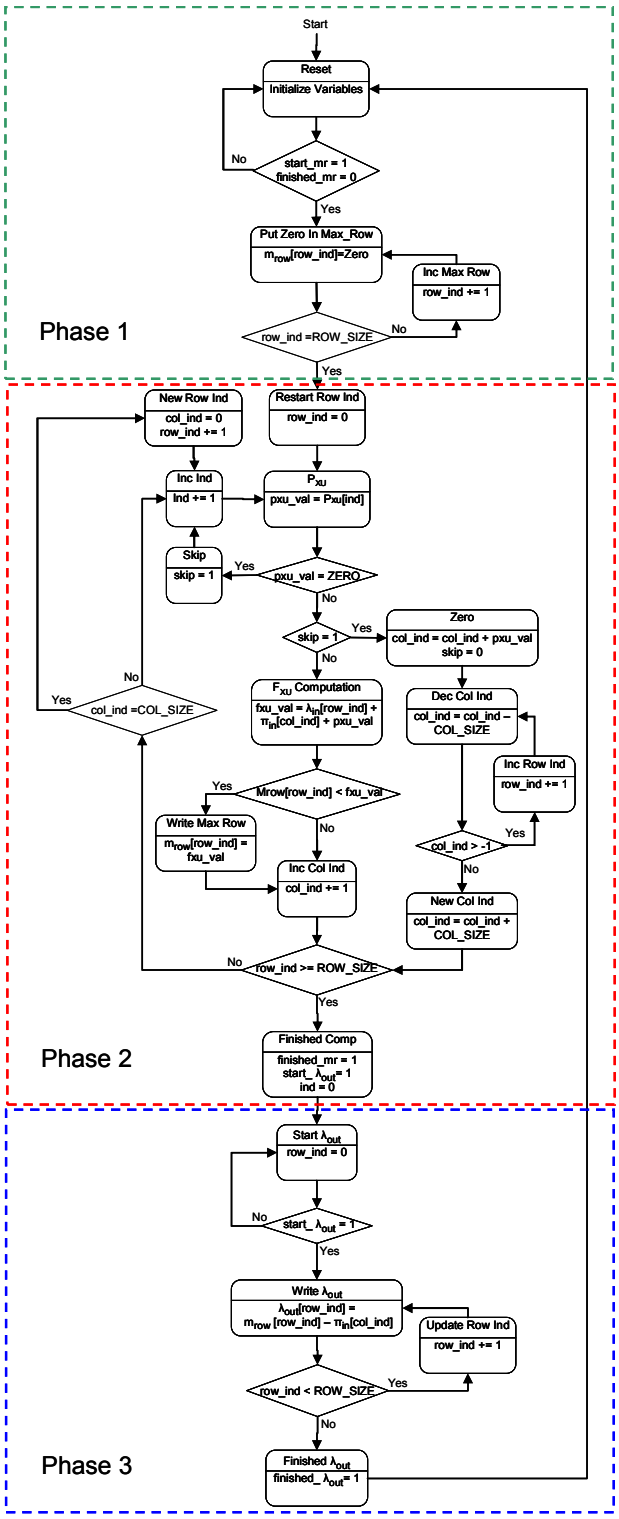

Figure 2.7. State machine diagram for a level 1 node. The diagram is separated into three phases.

In the model presented in [14], all values in the entire system are either zero or positive. In the hardware implementation, zeros are represented by the smallest number given the data bit width (in logarithmic form, this would be the negative number with the highest absolute value possible). To prevent error propagation from this zero representation, control statements were added to the state machine to check if any of the operands are the log of zero. If this is the case, the result is also set to the log of zero.

Experimental Setup

Network Configurations

In order to evaluate the parallel hardware and software implementations of the George and Hawkins neocortex inspired model, four networks of varying size and complexity were examined. The overall network structure was kept similar to the design in [14], with three layers of nodes per network and each level 2 node having four level 1 children. The level 1 and 2 nodes were arranged in a square matrix form (as shown in Figure 2.1). Table 2.1 lists details about each of the networks examined, including the number of nodes implemented in each network ($nodes_{NET}$) and the input image size.

Table 2.1. Networks implemented.

Network parameter	Nodes Implemented ($nodes_{NET}$)			
	181	321	501	721
Level 3 nodes	1	1	1	1
Level 2 nodes	36	64	100	144
Level 1 nodes	144	256	400	576
Input image size	48x48	64x64	80x80	96x96
Images per Category	4	6	6	6

The networks were trained using the images and training algorithm described in [14]. For the networks examined, the input image size (listed in Table 2.1) is determined based on each level 1 node using a 4x4 pixel patch from the input image. Table 2.1 also lists the number of images per image category for each network. Each image category represents a different object for the networks to classify. Within each category, several variations of the object, such as width and/or height, are used for training. 76 of the 91 binary image categories used in [14] was utilized. The George and Hawkins design contained two images in each image category. As the networks that were trained are larger than the ones in [14], larger images with more images per category were generated. This is to reflect the increased complexity that would be seen in larger networks. Figure 2.8 shows some example training images from 2 different image categories and examples of different image variations within each of the categories.

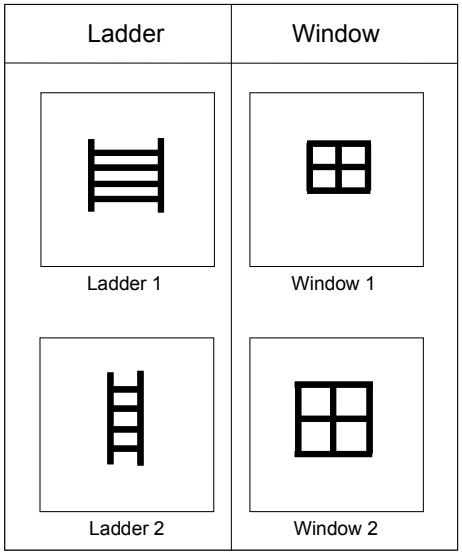

Figure 2.8. Example of image categories. Two image categories are shown (ladder and window) with two example images for each category.

The images for these networks were created by scaling the images from the 32x32 images in [14]. While scaling, the single pixel wide straight line characteristic seen in 14 was maintained. This enabled maximum network recognition capability. Increasing the number and size of images within an image category increases the complexity of the P_{xu} training matrix for the nodes in the networks. This also increases the computation time for each node. A collection of hand drawn images were used to test the systems. In order to allow the beliefs to stabilize, five passes through the network were used to analyze each input image.

Setup

The hardware-accelerated implementation on the Cray XD1 utilized on-chip FPGA memory exclusively for the processing elements. The training P_{xu} matrix for each node was large even after compression, thus making the processing elements memory-constrained as opposed to logic-constrained. For instance, in the 321 node network implementation, each node consumed 40% of the block RAM (*BRAM*), but only 10% of the logic. The number of processing elements that can be placed on a single FPGA (PE_{FPGA}) is determined by the memory utilization of each processing element (as shown in equation 7). To exclude partial processing elements, equation 7 needs to be rounded down. The Xilinx Virtex II Pro FPGAs (part number XCVP50) on the Cray XD1 contained 232 18Kb block RAMs and 23,616 logic slices.

$$PE_{FPGA} = \left\lfloor \frac{BRAMs\ units\ per\ FPGA}{BRAMs\ units\ per\ PE} \right\rfloor \tag{7}$$

Table 2.2 shows the composition of the hardware-accelerated implementations of the networks. The number of processing elements needed to implement each network (*NPE*) is determined by the number of level 2 nodes. The number of AMD processing cores needed (*num_proc*) is determined by the number of host processors acting as communication bridges plus the root processor implementing the level 3 node. Given that each FPGA requires one host processor, the number of level 2 nodes divided by PE_{FPGA} plus the level 3 root processor gives *num_proc* (as shown in equation 8).

$$num\_proc = 1 + \left\lceil \frac{(Level\ 2\ nodes)}{PE_{FPGA}} \right\rceil \tag{8}$$

Table 2.2. Hardware components for the networks accelerated.

Hardware Component	Network Size ($nodes_{NET}$)			
	181	321	501	721
Number of processing elements (*NPE*)	36	64	100	144
Processing elements per FPGA (PE_{FPGA})	2	2	2	2
Number of processing cores (*num_proc*)	19	33	51	73

Large scale networks running on the full Cray XD1 would utilize a fixed number of processors. Therefore in the software implementation, the number of processing cores was restricted to five for all the networks examined. The level 3 node was executed on the root processor, while the lower level processors ran the network partitions shown in Figure 2.4. Each partition consisted of a level 2 node and its corresponding four level 1

children. The software implementation utilized the algorithm optimizations listed in section 3.1 (except for the fixed point logarithmic data format).

Results

The functionality of the parallel hardware implementation was verified against the software implementation using a variety of hand drawn input images. In order to test the invariance pattern recognition capability of the system, the input images were distorted by addition of noise, changes in size, horizontal and vertical translation, and shape variation (see Figure 2.9). In all cases, the hardware systems produced the same order of beliefs for the training images as the software systems.

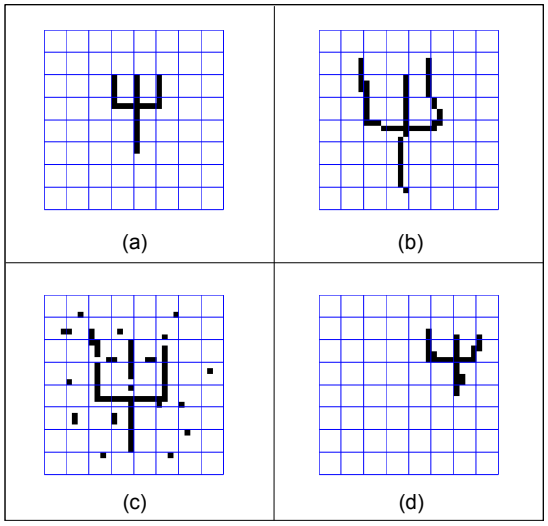

Figure 2.9. Example of hand drawn input images. Shows: (a) standard image, (b) size distortion, (c) noise distortion, and (d) translation.

To examine the worst case performance of hardware-accelerated systems, timing results for the slowest processing element for the networks was obtained. The system utilization of the FPGA with the largest processing elements in the networks is shown in Table 2.3 (these are generally also the slowest processing elements). The memory utilization of the larger networks increased because of the added complexity of the P_{xu} training matrices within the nodes of the networks. All the networks ran at 138 MHz.

Table 2.3. FPGA resource utilization.

Resource Utilization	Network Size ($nodes_{NET}$)			
	181	321	501	721
Logic	21%	22%	22%	22%
Memory	77%	85%	91%	91%

The performance of these networks was measured using a nodes per second throughput (τ) performance measure. To enable performance scaling analysis between the software and hardware implementations, comparisons between the throughput per processing core (τ_{CPU}) and the throughput per FPGA (τ_{FPGA}) are made. These terms are calculated in equations 9 and 10 respectively. These equations are based the number of nodes in the entire network implemented ($nodes_{NET}$), the time to complete one pass through a network (T_{SP}), and the number of processing cores (*num_proc*) or FPGAs (*num_FPGA*) used for implementing the networks. Figure 2.10 shows the throughput per processing core for the networks using the software implementation while Figure 2.11 shows the throughput per FPGA for the hardware implementation. In both of these figures, the throughput decreases as the complexity of the nodes increase for the larger

networks. This shows a throughput gain of approximately 123 on average per FPGAs over each processing core for the networks implemented.

$$\tau_{CPU} = \frac{nodes_{NET}}{T_{SP} \times num\_proc} \tag{9}$$

$$\tau_{FPGA} = \frac{nodes_{NET}}{T_{SP} \times num\_FPGA} \tag{10}$$

In order to simplify the analysis, the software portion of the network (the root processor running the level 3 node computation) is included in the FPGA throughput. This adds a very small increment to the FPGA throughput measure, given that there is only one level 3 node in the networks examined (this amounts to less than 1% of the overall nodes). This simplifies the analysis of scaling the FPGA implementation on the Cray XD1 since the system is reconfigurable logic-constrained.

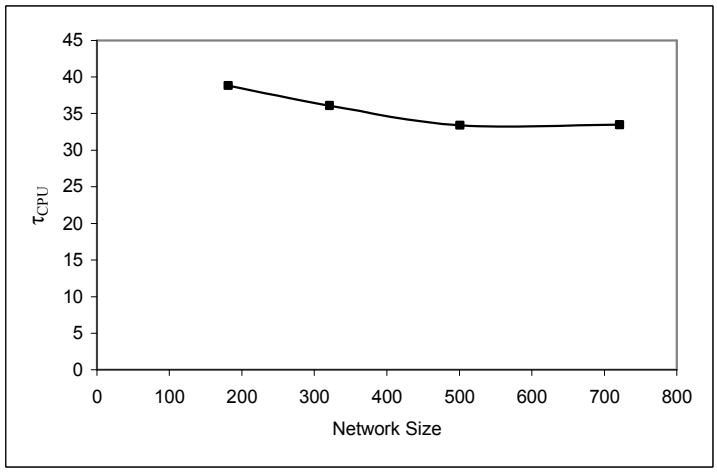

Figure 2.10. Average Nodes/(Second)(Core) throughput *(τ_{CPU})* for one network in the parallel software implementation.

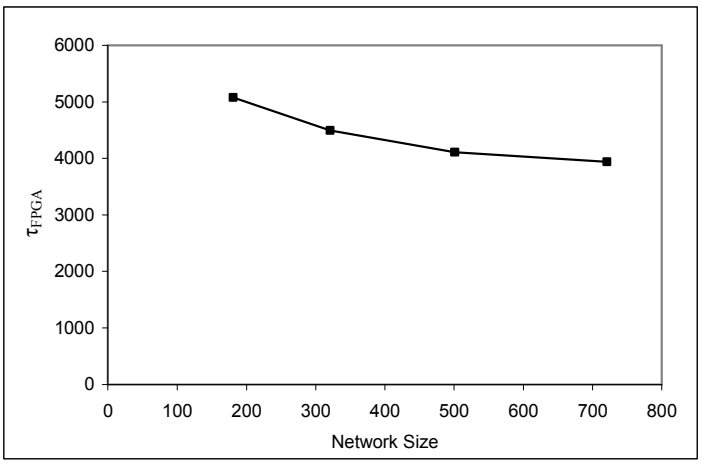

Figure 2.11. Average Nodes/(Second)(FPGA) throughput *(τ_{FPGA})* for one network in the parallel hardware implementation.

To predict the performance of other FPGA configurations for a scaling analysis, it is essential to examine the timing components in the FPGA designs implemented. Therefore, the time for one pass through a fully parallel hardware-accelerated network (T_{SP}) can be broken down into the following four parts:

T_{PE} – The time required for the level 1 nodes and connected level 2 node computation in one processing element on the FPGA.

D_{L1} - The average time per pass for the host processor to send the λ_{in} belief vectors to the level 1 nodes in a processing element implemented on an FPGA. These vectors need to be sent only once per image. Given that five passes are performed per image, the level 1 λ_{in} belief vectors are sent only once every five passes. Therefore D_{L1} is one fifth of the actual communication time to send the vectors.

D_{L2} – The average time for transferring the level 2 λ_{out} and π_{in} belief vectors between the FPGA and the host processor. These vectors have to be sent during each pass.

T_{MPI_ROOT} – This term accounts for: a) the MPI communication time between all the FPGA host processors and the root processor implementing the level 3 node during one pass through a network, and b) the time to compute the level 3 node belief on the root processor.

Table 2.4 lists these timing breakdowns as percentages for one entire pass through the different hardware-accelerated networks. The processing element computations can be performed in parallel, whereas the host processor to FPGA communications (the components contributing to D_{L1} and D_{L2}) cannot be parallelized for a single FPGA.

Table 2.4. Timing components for a single pass through the hardware implementation.

Timing Component	Network Size ($nodes_{NET}$)			
	181	321	501	721
D_{L1}	1.10%	1.03%	0.94%	0.93%
D_{L2}	10.46%	10.14%	9.42%	9.51%
T_{MPI_ROOT}	3.52%	5.04%	6.44%	8.74%
T_{PE}	84.92%	83.79%	83.20%	80.82%

Discussion

To solve interesting problems, larger networks than those in Table 2.1 would be needed. As described in [5], these larger networks could be developed as a network-of-networks built using the networks listed in Table 2.1. In such a system, all the networks would operate in parallel [5]. This section examines the potential performance of large networks utilizing the full resources of the Cray XD1 based on the results in section 5. An example of such a network may be a full-scale model of the human visual cortex. Examining the scaling of the model to large networks provides an indication of what

architecture options to pick for the larger networks. Two types of homogenous network-of-networks implementations are examined – a software-only implementation and a hardware-accelerated implementation. As shown in Table 2.3, the hardware implementations examined are memory bound. This limits the number of processing elements that can be placed in an FPGA. Utilizing the off-chip SRAM connected to each FPGA on the Cray XD1 would change the design to be logic bound and thus increase the number of processing elements that can be placed on an FPGA. Therefore for the hardware-accelerated implementation, two cases are considered: a) utilizing only the FPGAs' on-chip memory and b) utilizing the off-chip SRAM available to the FPGAs.

Software Scaling

The estimated nodes per second throughput ($\tau_{System,SW}$) of software implementations of large network-of-networks on all 864 AMD cores on the Cray XD1 is illustrated in Figure 2.12. The performances of four different networks are shown (based on the four networks examined). The values in Figure 2.12 were derived using equation 11. This is the throughput of a single processing core (τ_{CPU}, given by equation 9) multiplied by the number of total processing cores on the Cray XD1 (*cores_available*). The term τ_{CPU} includes the average I/O overhead per processor in the parallel implementation of a network. The larger networks in Table 2.1 have more complex P_{xu} matrices as they are trained with larger images. This causes the throughput of systems built around the larger networks to be lower than those built around the smaller networks.

$$\tau_{System,SW} = \tau_{CPU} \times cores\_available \quad (11)$$

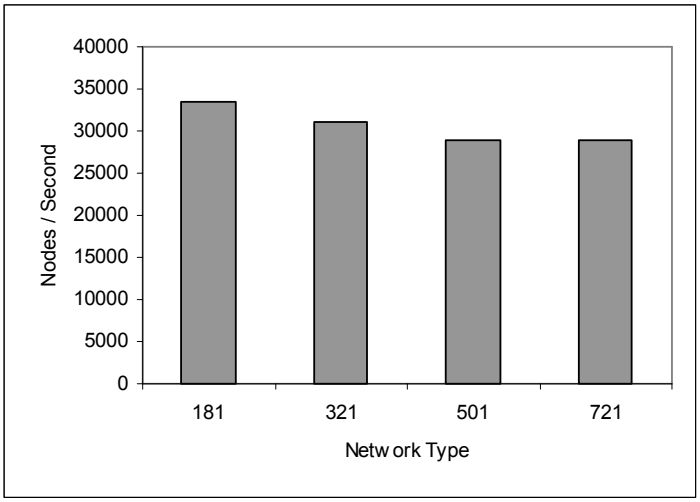

Figure 2.12. Nodes/Second throughput for a software implementation utilizing the full Cray XD1. The performance for four network types are shown based on the four networks listed in Table 2.1.

Hardware Scaling

The throughput of a network-of-networks utilizing hardware-acceleration will depend upon the number of processing elements that can be placed onto an FPGA and also the time for one pass through each network. In a network-of-networks all the networks are assumed to operate in parallel.

FPGA Timing Analysis

There are two possibilities for placing the data required by the processing elements in the hardware-accelerated system. These are to use either the on-chip memory

on the FPGA or to use the high speed off-chip SRAM available to each FPGA on the Cray XD1. In the former case, the design is constrained by the memory utilization. As shown in section 4.2, only two processing elements can be placed on the FPGA when using only the on-chip memory (for example, in the 721 node design, over 90% of the on-chip memory and only 22% of the logic is used). If the off-chip memory is utilized the design becomes logic-constrained, with the number of processing elements that can be placed (PE_{FPGA}) given by equation 12. Based on this equation, PE_{FPGA} comes out as 8 for the Xilinx Virtex II Pro FPGAs utilized.

$$PE_{FPGA} = \left\lfloor \frac{Slices\ available\ on\ FPGA}{Slices\ per\ processing\ element} \right\rfloor \tag{12}$$

As the number of processing elements per FPGA increases, the number of FPGAs needed to implement a full network in parallel decreases. The time for one pass through such a network is based on the four timing components listed in section 5. Equation 13 is based on these four timing components and shows how the single pass time (T_{SP}) for a fully parallel hardware implementation changes with the number of processing elements per FPGA (PE_{FPGA}). Of the four timing components, the processing element computation time (T_{PE}) does not change since all the processing elements are computing in parallel. Studies indicate that the off-chip SRAM bandwidth is high enough that it would pose an insignificant impact on T_{PE}. Since the amount of information flowing into the root processor remains the same, the sum of the MPI communication time to the root processor and the root node computation time (T_{MPI_Root}) does not change. However the communication time (D_{L1} and D_{L2}) between each FPGA and it's host processor increases

because more data is exchanged over each communication channel. The level 1 input data communication time (D_{L1}) increases proportionally with the number of processing elements. In this design, a collection of level 2 nodes (denoted as *L2_par_io*) communicate with the host processor in parallel, therefore the level 2 communication time (D_{L2}) increases with multiples of *L2_par_io* processing elements (for this design *L2_par_io* is four).

$$T_{SP} = \left(D_{L1} \times PE_{FPGA}\right) + \left(\left\lceil \frac{PE_{FPGA}}{L2\_par\_io} \right\rceil \times D_{L2}\right) + T_{PE} + T_{MPI\_ROOT} \tag{13}$$

Throughput of full Cray using hardware acceleration

In a full Cray XD1 implementation of a network-of-networks utilizing hardware-acceleration, the throughput depends on the number of networks that can operate in parallel and the time to complete one pass through a network. The number of networks that can be implemented using all the FPGAs on the Cray XD1 ($networks_{System}$) depends on the number of processing elements that can operate in parallel (PE_{System}) and the number of processing elements needed to implement one network (*NPE*). These terms are derived in equations 14 and 15.

$$PE_{System} = FPGAs\_available \times PE_{FPGA} \tag{14}$$

$$networks_{System} = \frac{PE_{System}}{NPE} \tag{15}$$

Here *FPGAs_available* is the total number of FPGAs on the Cray XD1 (144). The nodes per second throughput for the full Cray XD1 implementation ($\tau_{System,HW}$) is given by

the number of nodes that can operate in parallel and the time for one pass through a network (T_{SP}). This is shown in equation 16. Based on this equation, the estimated nodes per second throughput of the full Cray XD1 implementing network-of-networks using hardware-acceleration is shown in Figure 2.13. The performances for four types of networks are shown (based on the four networks examined). The throughput for both the on-chip memory and off-chip memory utilization cases are presented.

$$\tau_{System,HW} = \frac{networks_{System} \times nodes_{NET}}{Time\ for\ 1\ pass} = \frac{PE_{System} \times nodes_{NET}}{NPE \times T_{SP}} \tag{16}$$

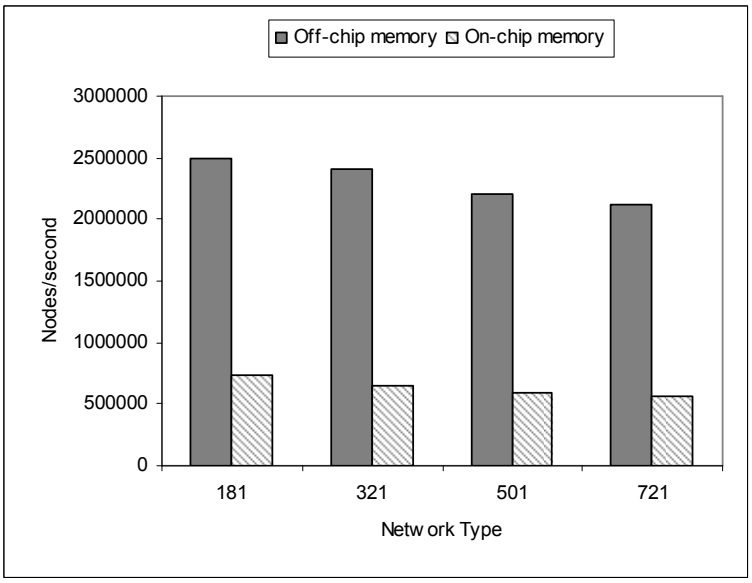

Figure 2.13. Nodes/Second throughput of the two hardware implementation cases when utilizing the full Cray XD1. These cases are the use of on-chip FPGA memory and the use of off-chip SRAM. The performance for four network types are shown based on the four networks listed in Table 2.1.

Comparison

Table 2.5 compares the potential throughput gain of the hardware-accelerated designs over the software design when utilizing the full resources of the Cray XD1 (based on Figures 12 and 13). The average throughput gains of the on-chip and off-chip memory utilization implementation over the software design are 20 and 75 respectively (see Table 2.5). Given that there are four times as many processing elements for the off-chip versus on-chip implementations, a maximum throughput gain of four times can be expected. However due to the serialization in I/O between each FPGA and it's host processor, a slightly lower throughput gain is seen (as shown in Table 2.5).

Table 2.5. Throughput gain of FPGA accelerated designs over a software-only design utilizing full Cray XD1 resources.

Throughput gain	Network Type ($nodes_{NET}$)			
	181	321	501	721
Using off-chip memory	74.56	77.40	76.61	73.28
Using on-chip memory	21.79	20.78	20.49	19.61
Ratio of off-chip/on-chip design	3.42	3.72	3.74	3.74

To accurately predict the throughput gains for larger networks, it would be necessary to implement the larger networks. This study was limited to smaller networks because the training of larger networks is very time consuming. The throughput gains of the hardware-accelerated over the software-only implementations were similar for all networks examined. Therefore one can speculate that for larger networks, similar throughput gains may be seen.

Summary

Biological systems have traditionally excelled over general purpose computing systems at cognitive applications. Better insights into the workings of primate brains have led to the development of new cognitive algorithms. Large scale implementations of these algorithms provide the potential to solve problems not currently possible with conventional computing systems. Given the large amounts of parallelism inherent in these models, hardware-acceleration can provide the potential to enable real-time implementations of these models.

This chapter presented an implementation of the George and Hawkins cognitive model based on the neocortex [14] on the Cray XD1 architecture. Software and reconfigurable hardware implementations were compared for networks of varying sizes and complexity. Based on these implementation results, the potential performance of larger networks that would utilize the full resources of the Cray XD1 was estimated. Since the hardware design was constrained by the amount of on-chip memory available on the FPGAs, the potential performance of a hardware implementation that would use the off-chip SRAM available to each FPGA on the Cray XD1 was examined. Results indicate that hardware-acceleration can provide average throughput gains of 75 times over software implementations of the networks examined when utilizing the full resources of the Cray XD1 and the off-chip memory on the FPGAs.

Although the implementations in this chapter were geared towards the Cray XD1, the results are applicable to other reconfigurable logic platforms. This is primarily since

the models implemented utilize a sequential stream of data from their large P_{xu} training matrices. If these matrices were to be placed on FPGA platforms with long latency memory, the memory accesses to the nodes can be started ahead of time to bring in the data as needed.

CHAPTER THREE

ACCELERATION OF MATCHED FILTER-BASED POSITION DETECTION

The National Ignition Facility, currently under construction at the Lawrence Livermore National Laboratory, is a stadium-sized facility containing a 192-beam, 1.8-MJ, 500-TW, ultraviolet laser system for the study of inertial confinement fusion and the physics of matter at extreme temperatures and pressures [45]. Automatic alignment (AA) based on computer analysis of video images adjusts the laser beams quickly and accurately enough to meet stringent system requirements in less than 30 minutes. The AA system directs all 192 laser beams along the 300-m optical path to focus on a 50 μm spot at the target chamber center [46]. At the heart of this alignment technique is the image processing algorithm that determines the position of beam features that are embedded in images recorded along the beam path. Varieties of alignment fiducials incorporated in the optical system designate various beam types, such as reference beams and main beams. Many beam images have well-defined spot profiles (e.g., Gaussian beams) for which centroiding is an acceptable technique to determine positions within the required accuracy of one half pixel. However, laser beam images often exhibit intensity variation or other distortions for which the centroid-based approach may result in high position uncertainty. In these cases, matched filtering provides an excellent and stable position measurement [33, 35], albeit at the expense of extra processing time required for each beam image. This chapter discusses an approach to speed up these computations using

field programmable logic array (FPGA). A performance improvement of 253 was achieved using the FPGA.

Background of Matched Filter-Based Position Detection

The matched filtering technique utilizes a given object as a template, whose position is known, to find the position of a second object by detecting the template's matching position in the correlation domain. The *classical matched filter* (CMF) [44] and its variation *phase only filter* (POF) [38] has gained popularity due to its ability of detecting an object with high discrimination to the presence of strong noise and background distortions. In the CMF, the complex amplitude and phase of the reference pattern is used, whereas POF only uses the phase of the reference pattern to perform the correlation [38]. The *amplitude modulated phase only filter* (AMPOF) [32, 40] was designed to further enhance filtering performance by modulating the POF by an inverse type of amplitude.

The Fourier domain treatment of the matched filter is described next. Let the Fourier transform of the to-be-detected object (template) function *f(x, y)* be denoted by

$$F(U_x, U_y) = \left|F(U_x, U_y)\right| \exp(j\Phi(U_x, U_y)) \tag{17}$$

and that of the input scene *g(x, y)* containing the desired object to be represented by

$$G(U_x, U_y) = \left|G(U_x, U_y)\right| \exp(j\Psi(U_x, U_y)) \tag{18}$$

A classical matched filter (CMF) corresponding to this function *f*(x, y) is expected to produce its autocorrelation. From the Fourier transform theory of correlation, the CMF

is given by the complex conjugate of the input Fourier spectrum as denoted by equation 19.

$$H_{CMF}(U_x, U_y) = F^*(U_x, U_y) = \left|F(U_x, U_y)\right| \exp(-j\Phi(U_x, U_y)) \tag{19}$$

The inverse Fourier transformation of the product of $F(U_x, U_y)$ and $H_{CMF}(U_x, U_y)$ results in the convolution of $f(x, y)$ and $f(-x, -y)$, which is the equivalent of the autocorrelation of $f(x, y)$. Moreover, when $|F(U_x, U_y)|$ is set to unity, H_{CMF} becomes a *phase only filter* (POF):

$$H_{POF}(U_x, U_y) = \exp(-j\Phi(U_x, U_y)) \tag{20}$$

The correlation of input image with the template is simply:

$$C_{CMF}(x, y) = F^{-1}\left\{ G\left(U_x, U_y\right) H_{CMF}\left(U_x, U_y\right) \right\} \tag{21}$$

The position of the object can be found from the position of the cross-correlation, autocorrelation, and the position of the template using equations 22-23.

$$x_{pos} = x_{cross} - x_{auto} + x_c \tag{22}$$

$$y_{pos} = y_{cross} - y_{auto} + y_c \tag{23}$$

Here (x_{pos}, y_{pos}) is the to-be-determined position of the pattern in the image plane, (x_{auto}, y_{auto}) is the position of the template autocorrelation peaks (where the template is correlated with itself and the peak is determined off-line). The position of the cross-correlation peak is (x_{cross}, y_{cross}), where the object appears in the actual scene and has to be determined from the real image. The position of the cross-correlation peak was estimated using a polynomial fit to the points surrounding the correlation peak. The

center of the template (x_c, y_c) and (x_{auto}, y_{auto}) are normally constant and may be calculated off-line, while the cross-correlation peaks move with changes in the object.

Related Works

In the computational demanding world of image processing, many different research groups have ventured into the realm of reconfigurable computing [37, 39, 42, 43, 47]. This was done in an effort to accelerate the image processing calculations to such a degree that would meet the requirements for real-time computation. The parallel architecture of FPGAs and the inherent parallel nature of image processing computations make this endeavor beneficial.

Yamaoka et al. [47] discuss a novel algorithm for object tracking in video pictures based on image segmentation and pattern matching. Because of the expensive calculations required of this algorithm, Yamaoka et al. developed an FPGA/ASIC architecture for their algorithm. This enabled them to perform the object tracking in real-time.

Lindoso and Entrena [43] compare the implementation of Zero-Mean Normalized Cross-Correlation in the spatial and spectral domains implemented on FPGAs. They proposed an FPGA based reconfigurable architecture where they achieved speedups of at least two orders of magnitudes over 3.0 GHz Pentium 4 systems. They show that real-time processing can be achieved from using this architecture by applying their design to a correlation-based fingerprint-matching algorithm.

Hezel et al. [39] presented a high-performance FPGA implementation for generic shape-based object detection in images. Here, Hezel et al. discussed their design of a pipelined template matcher on a FPGA. They correlate the distance transform of an observed image and the template for the actual match. At best, they achieved a speedup of about 200 times when comparing their 82 MHz FPGA system to a (not optimized) 500 MHz Pentium III PC system.

Guase et al. [37] described three reconfigurable systems implementing Shape-Adaptive Template Matching (SA-TM) to retrieve arbitrarily shaped objects within images or video frames. The three systems were a static system (static circuit configured to use off-chip memory only), a partially dynamic system (static circuit configured to use different on-chip memories), and a dynamic system (completely adapts to computation in terms of size and area of object template). They showed that their dynamic SA-TM design in a 50 MHz FPGA resulted in a speedup of almost 7,000 over a 1.4 GHz Pentium 4 PC when processing a 100x100 template on 300 consecutive HDTV format video frames.

Taking a different approach, Liang et al. [42] presented a generalized scheme to aid in mapping generalized template matching (GTM) operations to reconfigurable computers. As Liang et al. explain, GTM operations are image processing algorithms for two-dimensional digital filtering, morphologic operations, motion estimation, and template matching and others. Here, Liang et al. are focused on finding a balance between the host computer and the coprocessor FPGA. Reconfigurable design can be a very time consuming process, but by relaxing several constraints Liang et al. described a

systematic approach to automate GTM designs for FPGAs being used in reconfigurable systems.

Automatic Alignment Algorithms

The alignment system in each NIF beam line contains 26 control loops that analyze high-resolution beam and reference images. A number of beam image types require matched filtering to determine the object positions. One such set of corner-cube reflected pinhole images is shown in Figure 3.1. Here, the image processing algorithm exploits a template correlation to determine the pinhole centers ($x_{pos,}$ y_{pos}) as indicated by the cross in the center. The right side cross indicates the extent of the radius of the beam image.

A variety of distortions can challenge position finding algorithms. Examples in Figure 3.1 exhibit a wide variety of distortions such as illumination, shade, shape, and size. A weighted, or even a binary centroid, measurement [41] will be severely affected by beam non-uniformity, intensity gradient, beam elongation or diffraction effects. The purpose of the template shown in Figure 3.2 is to find the center by matching the edge of the beam. Since the beam size varies, the algorithm must search over a range of radii to determine the best-matched circle [30]. The center of the circle that yields the highest correlation is chosen as the position of the pinhole image. While this template works for the majority of the beam images, a more accurate template was recently determined to represent beam images [34] that have minimal distortion.

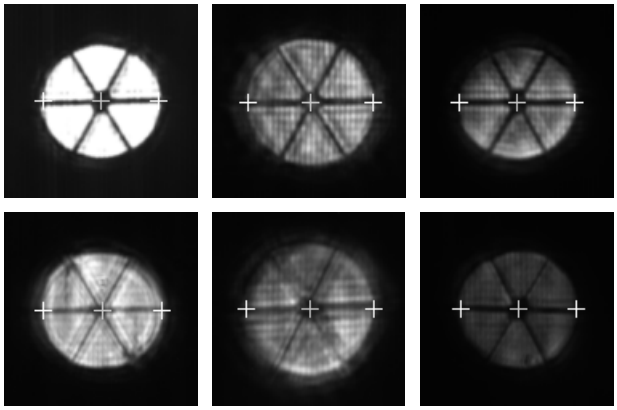

Figure 3.1. A set of corner-cube reflected pinhole images of various image qualities

In the example, the radius of the edge-template shown in Figure 3.2 was varied from 33 to 42 pixels. The correlation peak at various radii is plotted as shown in Figure 3.3. The peak reaches its maximum between a radius of 35 and 37, where 37 is the nominal radius of the image in Figure 3.1. A second order polynomial fit [33] through the correlation plane provides the x and y position of the correlation peak, from which equations 22 and 23 are used to find the center location.

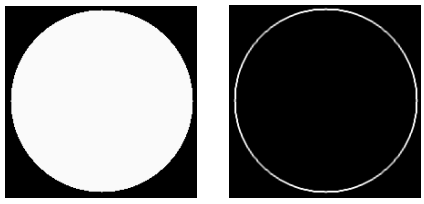

Figure 3.2. Image of the template used for pinhole images.

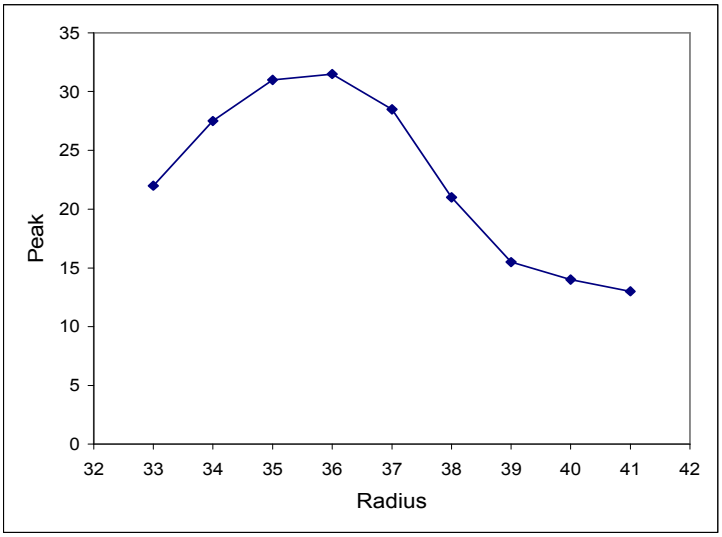

Figure 3.3. Correlation peak versus template radius

In another application, where the same template is used, the pinhole images are shown in Figure 3.4. Whereas the small pinholes vary from 32 to 45 pixels, these pinhole radii vary from 60 to 250 pixels. In order to reduce the processing time, instead of searching the whole range from 65 to 250 pixels, a measurement process is carried out to estimate the range to a smaller interval of 10 pixels [30].

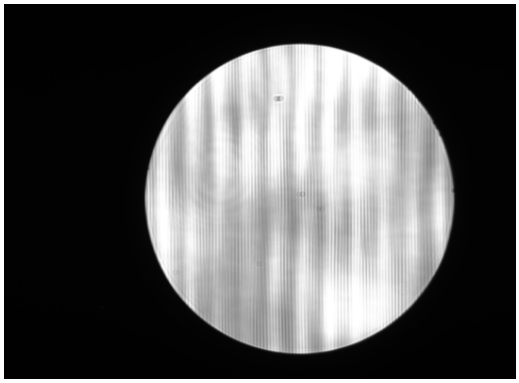

Figure 3.4. Image with 160 pixel radius

In some alignment beam images, two types of fiducials (circles and squares) [31] are used to indicate the beam position and the alignment reference location. The diameter of the circle is similar to the side of the square resulting in correlation peak values that may be hard to discriminate (Figure 3.5).

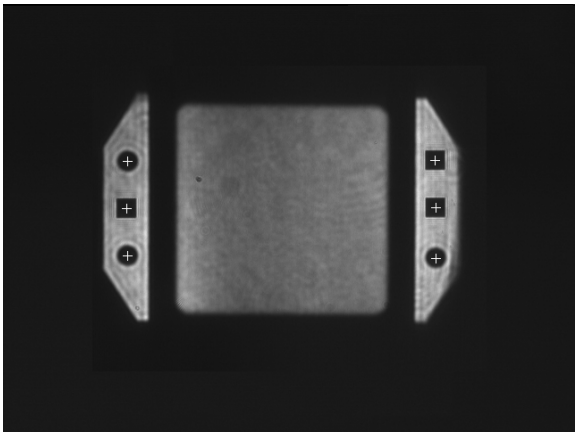

Figure 3.5. Two classes of fiducial patterns with positions identified

To enhance the discrimination, and hence the detection accuracy, of the to-be-detected objects, features such as object edges are used as shown in Figure 3.6. Instead of using circle templates, the circle edge is used for the filters. The resulting correlation cross-section from the right side of the wings is shown in Figure 3.7. Note from Figure 3.7 that the circle autocorrelation is higher than cross-correlation with the squares exhibiting a 2:1 discrimination between the two. Based on the normalized autocorrelation value, a dynamic threshold (as a percentage of the maximum peak) can be selected to reject the non-circles correlations. After selecting the circles, the image is correlated with a second template consisting of a square mask. Now using equations 22 and 23, the

position of the objects can be found from the position of the cross-correlation peak, the autocorrelation peak, and the template. In all these applications, the basic operation performed is a matched filtering via equation 21.

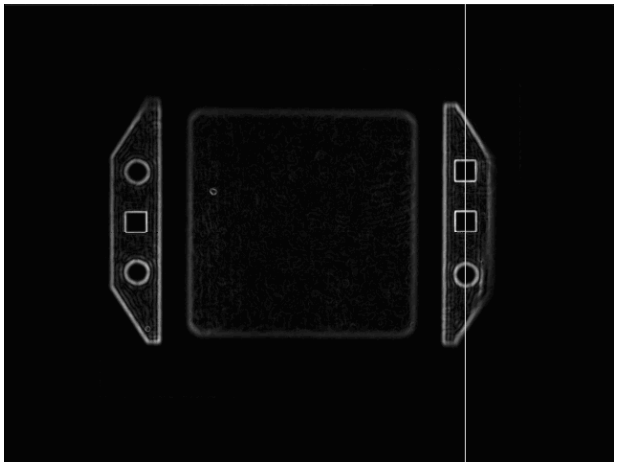

Figure 3.6. The edge of the image in Figure 3.5

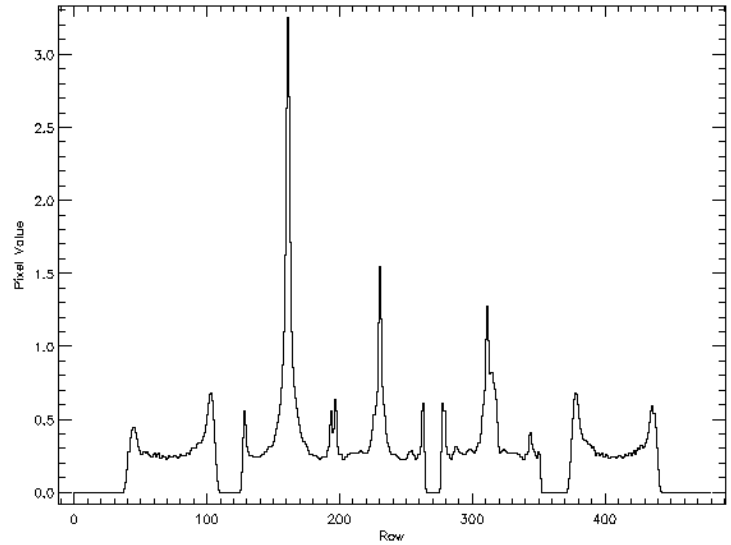

Figure 3.7. The correlation with circle of the image in Figure 3.5 (the cross-section through the right wing shown)

FPGA Acceleration of Image Correlation

The most computationally intensive portion of the image processing is the two-dimensional image correlation. Thus to shorten the alignment time, one can reduce the image processing time. For continuous high performance alignment operation such as may be required in a laser fusion power plant, faster methods of beam alignment will be necessary. One advantage of these computations is a significant amount of parallelism, thus enabling hardware acceleration.

Here, the potential of hardware acceleration by implementing the correlation computations on an FPGA was evaluated. The test system utilized was a Cray XD1 with 864 2.0 GHz AMD Opteron cores and 144 Xilinx Virtex II Pro FPGAs. In this system, only one FPGA and AMD core was utilized for the testing. The AMD core sends the images to be processed to the FPGA and receives back the location and peak value in the correlation output. A more practical approach for FPGA acceleration would be to utilize an FPGA accelerator card in a desktop computing system (such cards average about $2500 per FPGA at present).

Hardware design

Figure 3.8 presents a system overview of the FPGA implementation. Input data and intermediate values are stored in buffers (shown as the shaded boxes). These are on-chip memories on the FPGA. The inputs to the system, *f(x, y)* and *g(x, y)*, represent the

template and source image in equations 17 and 18, respectively. Up to 32 templates can be loaded into the FPGA (in the buffers labeled f0 to f31) and applied to each source image (in the buffers labeled g0 and g1). The two-dimensional Fast Fourier Tranforms (FFTs) in equations 17 and 18 are performed using two consecutive one-dimensional FFTs. Similarly, the inverse FFT in equation 21 is implemented with two one-dimensional forward FFTs.

The FFT units were built using Xilinx-supplied library components. To enable high-throughput computation, the system is pipelined into a pre-phase and four phases as shown in Figure 3.8. In this design, the amount of time required for the sobel filter computation is the same as the amount of time to complete approximately four phases. Therefore, the pre-phase is designed to occur independently of the phase computations. By alternating the two source image buffers (g0 and g1) between being used as input to the sobel filter unit and as a memory buffer to hold the incoming source image data, the system allows for the sobel filter computation to overlap the image-template computations.

Each phase works on a particular image–template combination. Since the same set of templates is used for each image, the templates are preloaded in on-chip buffers. This allows high-speed access to the templates that accelerated the system performance. Note the time to load each template onto the FPGA is longer than the pipeline phase computation time. Since each phase requires multiple cycles to compute, two buffers are needed between consecutive phases. For example, in Figure 3.8, the upper buffer (mb0) between Phases 1 and 2 holds the output being generated by Phase 1. The lower buffer

(mb1) holds the completed output previously generated by Phase 1, for use in Phase 2. Switches pipe data to the appropriate buffers.

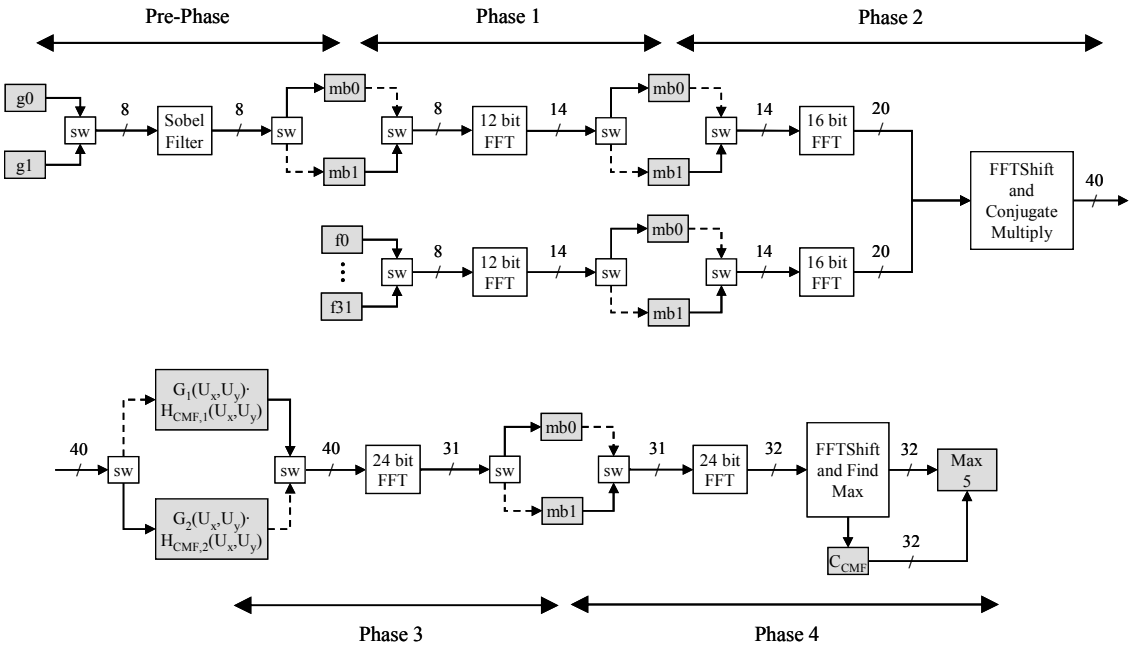

Figure 3.8. The block diagram of the FPGA operations. The boxes labeled "sw" are switches.

The pre-phase and the four phases in the architecture perform the following functions:

Pre-Phase: The pre-phase consists of applying a sobel filter to the input image *g(x, y)* to detect the edges. The time for this stage is only seen once because it is overlapped with the computation of the input image *g(x, y)* with the various template images.

Phase 1: The first one-dimensional FFT for Complex Fourier transform represented by equations 17 and 18 is computed. These two computations can be carried

out in parallel. The inputs to this phase are unsigned 8 bit values. Since an 8 bit FFT unit would treat the inputs as signed values, a larger bit width FFT unit is needed. Therefore a 12 bit FFT unit is used in the first phase. The first phase 12 bit FFT outputs are stored in buffers labeled mb0 and mb1 exiting Phase 1.

Phase 2: The second one-dimensional FFT to complete the Complex Fourier transform represented by equations 17 and 18 is computed. As the maximum output value for Phase 1 is 14 bit, a 16 bit FFT unit is used for the second phase. Also part of equation 21 is evaluated. Here the output of equation 17 is conjugated and multiplied by the output of equation 18. An FFT shift operation is executed in parallel with the multiplication in order to center the image. The 40 bit output is stored in a buffer.

Phase 3: The first one-dimensional FFT for the inverse FFT in equation 21 is evaluated. Since the inverse FFT is implemented with two 24-bit forward FFT units, they use only the most significant 24 bits of the inputs. This introduces round-off error as the computations take place in the integer domain.

Phase 4: The second one-dimensional FFT for the inverse FFT is equation 21 is evaluated here. Pipelined computation of the location of the peak in the output of equation 21 (C_{CMF}) is also determined. The absolute value of each location is computed and then compared against previously generated values to determine the peak location. The coordinates and amplitude of the peak along with the amplitude of the four surrounding locations are stored and returned to the processor. The index of the template image where the maximum has occurred among the submitted template images to the FPGA is also returned to the processor.

Hardware Performance

The system above was implemented on a Xilinx Virtex II Pro FPGA (part number XCVP50) on a Cray XD1. The FPGA synthesized system ran at 160 MHz. The logic utilization was 69% while the block RAM utilization was 75%. The algorithm was also ported to a 2.0 GHz AMD Opteron core using C. The optimized FFT library developed by Stefan Gustavson [36] was utilized by the C software implementation.

Both systems were tested with 64x64 images and 32 template images per image. The overall runtime of the FPGA system to process a source image through 32 template images was about 0.427 ms, while the C system required 108.406 ms. This is equivalent to the FPGA system giving a speedup of approximately 253 times over the C system. Newer generation FPGAs with larger resources and higher clock speeds would allow multiple pipelines to analyze more images in parallel, thus resulting in greater speedups. Table 3.1 compares the output of the C and the FPGA implementations for both auto and cross-correlation examples. The absolute value of the peak and its surrounding four locations are shown along with the error for each location. The average absolute error for these values is 2.20%.

Table 3.1. Output comparison between C and FPGA implementations for the peak and surrounding four locations (output values are to be multiplied by 10^9).

Pixel Locations	C	FPGA	Error (%)
Top Center	5.62	5.61	0.18
Right Center	5.62	5.91	-5.15
Center	6.67	6.28	5.74
Left Center	5.63	5.89	-4.67
Bottom Center	5.63	5.63	-0.11

(i) Cross correlation between a circle and a square

Pixel Locations	C	FPGA	Error (%)
Top Center	12.11	12.05	0.50
Right Center	12.11	12.41	-2.49
Center	16.11	15.83	1.74
Left Center	12.11	12.42	-2.58
Bottom Center	12.11	12.05	0.50

(ii) Auto-correlation of a square

Pixel Locations	C	FPGA	Error (%)
Top Center	9.35	9.32	0.40
Right Center	9.35	9.60	-2.59
Center	12.34	11.96	3.06
Left Center	9.35	9.60	-2.65
Bottom Center	9.35	9.30	0.60

(iii) Auto-correlation of a circle

Summary

Automatic alignment of the NIF laser is dependent on computationally intensive image processing. One important component of the image processing is matched filtering. This chapter describes an approach to speed up this computation using low cost parallel computing hardware. The results indicate a speedup of approximately 253 using an FPGA over a 2.0 GHz AMD Opteron processing core.

CHAPTER FOUR

CONCLUSION

This thesis shows the utilization of hardware-acceleration for two very different image processing applications. The first image processing application, a neocortex inspired cognitive model, is presented in Chapter 2 while the second image processing application, matched filter-based position detection, is presented in Chapter 3.

This thesis presented an implementation of the George and Hawkins cognitive model based on the neocortex [14] on the Cray XD1 platform. Software and reconfigurable hardware implementations were compared for networks of varying sizes and complexity. Based on these implementation results, the potential performance of larger networks that would utilize the full resources of the Cray XD1 were estimated. Since the hardware design was constrained by the amount of on-chip memory available on the FPGAs, the potential performance of a hardware implementation that would use the off-chip SRAM available to each FPGA on the Cray XD1 was examined. Results indicate that hardware-acceleration can provide average throughput gains of 75 times over software implementations of the networks examined when utilizing the full resources of the Cray XD1 and the off-chip memory on the FPGAs.

Although the cognitive model implementations in this research were geared toward the Cray XD1, the results are applicable to other reconfigurable logic platforms. This is primarily since the models implemented utilize a sequential stream of data from their large P_{xu} training matrices. If these matrices were to be placed on FPGA platforms

with long latency memory, the memory accesses to the nodes can be started ahead of time to bring in the data as needed. One area of future work is to evaluate larger networks, the training of these networks, and other cognitive models on the Cray XD1.

The other image processing application examined in this thesis was matched filter-based position detection. An approach to accelerate the NIF automatic alignment matched filtering computations using reconfigurable hardware was described. The results indicate a speedup of approximately 253 times using an FPGA over a 2.0 GHz AMD Opteron processing core. Other applications that can benefit from the speed enhancement include associative recall of extremely large databases of medical or other images using a partial queue. Future work is to evaluate variants of the matched filter-based position detection geared towards other image processing applications. One such example would be fingerprint recognition.

REFERENCES

[1] R. Ananthanarayanan and D. Modha, “Antomy of a Cortical Simulator,” *Proceedings of the ACM/IEEE Conference on Supercomputing*, November 2007.

[2] J.A. Anderson, “The BSB Network,” In: Hassoun, M.H. (ed.): *Associative Neural Networks*. Oxford University Press, New York, 77-103, 1993.

[3] J. A. Anderson. “Arithmetic on a parallel computer: Perception versus logic,” *Brain and Mind*, 4:169–188, 2003.

[4] J. A. Anderson, P. Allopenna, G. S. Guralnik, D. Scheinberg, J. A. Santini, S. Dimitriadis, B. B. Machta, and B. T. Merritt, "Programming a Parallel Computer: The Ersatz Brain Project," In W. Duch, J. Mandziuk, and J.M. Zurada (Eds.), *Challenges to Computational Intelligence*, Springer: Berlin, 2006.

[5] J. A. Anderson and J. P. Sutton, “If we compute faster, do we understand better?,” *Behavior Research Methods, Instruments, and Computers*, 29:67–77, 1997.

[6] M. Atencia, H. Boumeridja, G. Joya, F. García-Lagos, and F. Sandoval, "FPGA implementation of a systems identification module based upon Hopfield networks," *Neurocomputing*, 70:2828–2835, October 2007.

[7] K. Boahen, "Neurogrid: Emulating a Million Neurons in the Cortex," in *IEEE International Conference of the Engineering in Medicine and Biology Society*, 2006.

[8] C.-S. Bouganis, P. Y. K. Cheung, and L. Zhaoping, "FPGA-Accelerated Pre-Attentive Segmentation In Primary Visual Cortex," in International Conference on Field Programmable Logic and Applications, 2006.

[9] T. Dean. “A Computational Model of the Cerebral Cortex,” *Proceedings of the Twentieth National Conference on Artificial Intelligence* (AAAI-05), pages 938-943, Cambridge, Massachusetts, 2005. AAAI, MIT Press.

[10] D. J. Felleman and D. C. Van Essen, “Distributed hierarchical processing in the primate cerebral cortex,” *Cereb. Cortex1*, 1–47 (1991).

[11] D. Ferrer, R. Gonz´alez, R. Fleitas, J. P. Acle, R. Canetti, “NeuroFPGA - Implementing Artificial Neural Networks on Programmable Logic Devices,” in *Design, Automation and Test in Europe Conference and Exhibition Designers' Forum*, 2004.

[12] J. Furlong, A. Felch, J. M. Nageswaran, N. Dutt, A. Nicolau, A. Veidenbaum, A. Chandrashekar, and R. Granger. "Novel Brain-Derived Algorithms Scale Linearly with Number of Processing Elements," *Proceedings of ParaFPGA Conference: Parallel Computing with FPGA's*, 2007.

[13] C. Gao and D. Hammerstrom. "Cortical Models Onto CMOL and CMOS—Architectures and Performance/Price," *IEEE Transactions on Circuits and Systems I*, 54(11):2502-2515, Nov 2007.

[14] D. George and J. Hawkins. “A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex,” in *International Joint Conference on Neural Networks*, 2005.

[15] J. Hawkins. "Why Can't a Computer be more Like a Brain?" *IEEE Spectrum*, April 2007.

[16] J. Hawkins and S. Blakeslee. “On Intelligence,” Times Books, Henry Holt and Company, New York, NY 10011, Sept 2004.

[17] J. Hawkins and D. George, “Hierarchical Temporal Memory – Concepts, Theory, and Terminology,” Whitepaper, Numenta Inc, May 2006.

[18] C. Johansson and A. Lansner. "Towards Cortex Sized Artificial Neural Systems," *Neural Networks*, 20(1):48-61, January 2007.

[19] T. S. Lee and D. Mumford. “Hierarchical bayesian inference in the visual cortex,” *Journal of the Optical Society of America A*, 2(7):1434-1448, 2003.

[20] P Merolla, J Arthur, B E Shi and K Boahen, "Expandable Networks for Neuromorphic Chips," *IEEE Transactions on Circuits and Systems I*, 54(2):301-311, February 2007.

[21] J. Pearl. “Probabilistic Reasoning in Intelligent Systems,” Morgan Kaufman Publishers, San Francisco, California, 1988.

[22] M. Pearson, M. Nibouche, A.G. Pipe, C. Melhuish and I.Gilhespy, B. Mitchison, K. Gurney, T. Prescott, and P. Redgrave, "A Biologically Inspired FPGA Based Implementation of a Tactile Sensory System for Object Recognition and Texture Discrimination," in *International Conference on Field Programmable Logic and Applications*, 2006.

[23] I. Pournara, C. S. Bouganis, and G. A. Constantinides. "Fpga-Accelerated Bayesian Learning For Reconstruction of Gene Regulatory Networks," in *International Conference on Field Programmable Logic and Applications*, 2005.

[24] K. L. Rice, C. N. Vutsinas, and T. M. Taha. "A Preliminary Investigation of a Neocortex Model Implementation on the Cray XD1," *Proceedings of the ACM/IEEE Conference on Supercomputing*, November 2007.

[25] K. L. Rice, T. M. Taha, and C. N. Vutsinas. "Hardware Acceleration of Image Recognition through a Visual Cortex Model," *Optics and Laser Technology*, 40(6):795-802, 2008.

[26] J. A. Starzyk, Z. Zhu, and T.-H. Liu. "Self-Organizing Learning Array," *IEEE Transactions on Neural Networks*, 16(2):355-363, March 2005.

[27] C. Torres-Huitzil, B. Girau, and C. Castellanos-Sanchez. "On-chip Visual Perception of Motion: A Bio-inspired Connectionist Model on FPGA," *Neural Networks Journal,* 18(5-6):557–565, 2005.

[28] R. K. Weinstein and R. H. Lee, "Architecture for high-performance FPGA implementations of neural models," *Journal of Neural Engineering*, 3, 21-34, 2006.

[29] R. Zemel. "Cortical belief networks," in Hecht-Neilsen, R., ed., *Theories of the Cerebral Cortex*. New York, NY: Springer-Verlag, 2000.

[30] A. A. S. Awwal, "Automatic identification of templates in matched filtering," in *Photonic Devices and Algorithms for Computing VI*, edited by K. Iftekharuddin and A. A. S. Awwal, *Proceedings of SPIE,* 2004.

[31] A. A. S. Awwal, "Multi-object feature detection and error correction for NIF automatic optical alignment" in *Photonic Devices and Algorithms for Computing VIII*, edited by K. Iftekharuddin and A. A. S. Awwal, *Proceedings of SPIE*, 2006.

[32] A.A.S. Awwal, M.A. Karim, and S.R. Jahan. "Improved Correlation Discrimination Using an Amplitude-modulated Phase-only Filter," *Applied Optics*, 29, 233-236, 1990.

[33] A. A. S. Awwal, W. A. McClay, W. S. Ferguson, J. V. Candy, T. Salmon, and P Wegner. "Detection and Tracking of the Back-Reflection of KDP Images in the presence or absence of a Phase mask," *Applied Optics*, 45(13):3038-3048, 2006.

[34] A. A. S. Awwal, K. Rice, R. Leach and T. Taha, "Higher accuracy template for corner cube reflected image," to be presented at the *Optics and Photonics for Information processing(II)*, San Diego, August CA 2008.

[35] J. V. Candy, W. A. McClay, A. A. S. Awwal, and S. W. Ferguson. "Optimal position estimation for the automatic alignment of a high-energy laser," *Journal of Optical Society of America A*, 22(7):1348-1356, 2005.

[36] FFT code and related material, <http://www.jjj.de/fft/fftpage.html>.

[37] J. Gause, K.Y.P. Cheung, and W. Luk. "Reconfigurable Shape-Adaptive Template Matching Architectures," *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002.

[38] J. L. Horner and J. Leger. "Pattern recognition with binary phase-only filters," *Applied Optics*, 24(5):609-611, 1985.

[39] S. Hezel, A. Kugel, R. Männer, and D.M. Gavrila. "FPGA-based Template Matching using Distance Transforms," *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002.

[40] M. A. Karim and A. A. S. Awwal. *Optical Computing: An Introduction*, John Wiley, New York, NY, 1992.

[41] W. A. McClay III, A. A. S. Awwal, H. E. Jones, K. C. Wilhelmsen, W. Ferguson, M. McGee, and M. G. Miller. "Evaluation of laser-based alignment algorithms under additive random and diffraction noise," *Photonic Devices and Algorithms for Computing VI*, edited by K. Iftekharuddin and A. A. S. Awwal, *Proceedings of SPIE*, 2004.

[42] X. Liang and J. S.-N. Jean. "Mapping of Generalized Template Matching Onto Reconfigurable Computers," *IEEE Transactions on Very Large Scale Integration Systems*, 11(3):485-498, June 2003.

[43] A. Lindoso and L. Entrena. “High Performance FPGA-based Image Correlation,” *Journal of Real-Time Image Processing*, 2(4):223-233, 2007.

[44] A.V. Lugt.“Signal Detection by Complex Spatial Filtering,” *IEEE Transactions on Information Theory*, 10(2):139-145, 1964.

[45] E. Moses, C. Wuest. “The National Ignition Facility: Status and Plans for Laser Fusion and High-Energy-Density Experimental Studies”, *Fusion Science and Technology*, Vol. 43, p. 420, May 2003.

[46] K.C. Wilhelmsen, A. A. S. Awwal, S. W. Ferguson, B. Horowitz, V. J. Miller Kamm, and C. A. Reynolds. “Automatic Alignment System for the National Ignition Facility” *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems*, 2007.

[47] K. Yamaoka, T. Morimoto, H. Adachi, T. Koide, and H.J. Mattausch. “Image Segementation and Pattern Matching Based FPGA/ASIC Implementation Architecture of Real-Time Object Tracking,” *Proceedings of the 2006 Conference on Asia South Pacific Design Automation*, 2006.