HIERARCHICAL BAYESIAN CORTICAL MODELS: ANALYSIS AND ACCELERATION ON MULTICORE ARCHITECTURES

Pavan Yalamanchili
Clemson University, pyalama@clemson.edu

HIERARCHICAL BAYESIAN CORTICAL MODELS: ANALYSIS AND ACCELERATION ON MULTICORE ARCHITECTURES

A Thesis
Presented to
The Graduate School of
Clemson University

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

by
Pavan Kumar Yalamanchili
August 2009

Accepted by:
Tarek Taha, Committee Chair
Adam Hoover
Stanley Birchfield

ABSTRACT

There is a significant interest in the research community to develop large scale, high performance implementations of neuromorphic models. These have the potential to provide significantly stronger information processing capabilities than current computing algorithms. This thesis examines the parallelization of two recent biologically inspired hierarchical Bayesian cortical models onto recent multicore architectures. These models have been developed recently based on new insights from neuroscience and have several advantages over traditional neural networks. In particular, they need far fewer network nodes to simulate a large scale cortical model than traditional neural networks, making them computationally more efficient. This is the first study of the parallelization of this class of models onto multicore processors. Results indicate that the models can take advantage of parallelism present in the processors to provide significant speedups on multicore architectures. These models are further shown to scale well on a cluster of 336 PS3s available at the Air Force Research Lab which is shown to emulate between 10^8 to 10^{10} neurons. In particular, the results indicate that a cluster of Playstation 3s can provide an economical, yet powerful, platform for simulating large scale neuromorphic models.

ACKNOWLEDGMENTS

The work presented in this thesis would not have been possible without the help of my research group. I would like to thank Tarek Taha for his guidance, help and support. Next, I would like to thank Sumod Mohan, Rommel Jalasutram, Kenneth Rice and Mohammad Ashraf Bhuiyan for their help during this research.

I would also like to acknowledge National Science Foundation, the Air Force Research Lab, and the Air Force Office of Scientific Research, whose funding made this thesis possible. Finally I would like to thank Clemson University, the Georgia institute of Technology, and the Arctic Region Supercomputing Center whose resources helped make this research possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE
INTRODUCTION

There has been a strong interest amongst researchers to develop large parallel implementations of cortical models on the order of animal or human brains. At this scale, the models have the potential to provide much stronger inference capabilities than current generation computing algorithms [1]. A large domain of applications would benefit from the stronger inference capabilities including speech recognition, computer vision, textual and image content recognition, robotic control, and data mining. Recent scientific studies of the primate brain have led to new neuromorphic computational models [3][9][13][18][23][38] of the information processing taking place in the cortex. These cortical models provide insights into the workings of the brain and concur well with experimental results. The models differ significantly from traditional neural networks in that they are generally at a higher level of abstraction than neural networks and they consider several new biological details about the organization and processing in the cortex. Some of these newer cortical models [8][14] are based on hierarchical Bayesian networks and incorporate several of the recently suggested properties of the neocortex [23][29]. These include a hierarchical structure of uniform computational 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.

These new models utilize several recent finings from neuroanatomists. In particular, neuroanatomists have identified that a collection of about 80 to 100 neurons form into regular patterns of local cells running perpendicular to the cortical plane [17]. These collections of neurons are called mini-columns. Mountcastle [28] states that the basic unit of cortical operation is the mini-column and that a collection of mini-columns are grouped into a cortical column. He also states that the mini-columns within a cortical column are bound together by a common set of inputs and short-range horizontal connections.

Hierarchical Bayesian network based cortical models have a significant computational advantage over traditional neural networks. Each node in the former models a cortical mini-column or a cortical column, while in the latter each node models only a single neuron. Thus to model a large collection of neurons, a hierarchical Bayesian network based model would require far fewer nodes than a traditional neural network based model. Additionally, the number of node-to-node connections is greatly reduced in hierarchical Bayesian network based cortical models. Anatomical evidence suggests that most of neural connections in the cortex are within a column as opposed to being between columns.

The brain utilizes a large collection of slow neurons operating in parallel to achieve very powerful cognitive capabilities. There has been a strong interest amongst researchers to develop large parallel implementations of cortical models on the order of animal or human brains. At this scale, the models have the potential to provide much stronger inference capabilities than current generation computing algorithms [7]. Several research groups are examining large scale implementations of neuron based models [1][24] and cortical column based models [21][35]. Such large scale implementations require high performance resources to run the models at reasonable speeds. IBM is utilizing a 32,768 processor Blue Gene/L system to simulate a spiking network based model [1], while EPFL and IBM are utilizing a 8,192 processor Blue Gene/L system to simulate a sub-neuron based cortical model [24]. The PetaVision project announced recently at the Los Alamos National Laboratory in June 2008 is utilizing the Roadrunner supercomputer (currently ranked as the world's fastest supercomputer) to model the human visual cortex [31].

This thesis examines optimizations and parallel implementations on multicore architectures, both single machines as well as clusters using MPI, of the recognition phase of two recent hierarchical Bayesian network cortical models. The two models examined are Hierarchical Temporal Memories (HTM) [18] and Dean's Hierarchical Bayesian model (to be referred to as the Dean model in the rest of the thesis) [8]. At present there are no other hierarchical Bayesian cortical models (other than updates to these models and Lee and Mumford's work [23], on which Dean's model is based). This thesis examines the parallelization and optimizations of the recognition phase of these models. The training of the models is generally carried out in a longer offline process.

With the limited scaling in processor clock frequencies, multicore processors have become the standard industrial approach to improve processor performance. However there are no studies examining the parallelization or implementation of any hierarchical Bayesian cortical models onto multicore processors. Lansner and Johannson [21] have shown that mouse sized cortical models developed on a cluster of commodity computers are computationally bound rather than communication bound. Therefore the acceleration of the computations of these models on multicore architectures can provide significant performance gains to enable large scale implementations.

The main contributions of this work are:

1) A study of the parallelization of two hierarchical Bayesian cortical models. Both thread level parallelization and data level parallelization of the models are examined.
2) A study of different optimizations and parallelization strategies for the models.
3) An evaluation of the multicore implementations of the models. This thesis examines the performance of the models on three multicore processors on four platforms (a Sony Playstation 3, an IBM QS20 blade, a Sun Enterprise 5140 server, and a dual processor Intel Xeon blade). Several sizes of the model networks were implemented to examine the effect of scaling.
4) A preliminary study to identify the effect of scaling in multi-core clusters of two platforms for both models (the Palmetto cluster at Clemson University using Xeon blades, the ARSC Cell cluster using PS3s).
5) A comprehensive analysis of the HTM model on the AFRL cluster of 336 PS3s

Results indicate that optimized parallel implementations of the model can provide significant speedups on multicore architectures. Using all eight cores on of Cell processor on an IBM QS20 blade provided a speedup of 107 times for the Dean model and 93 times for the HTM model over a serial implementation on the Power Processor Unit of the Cell processor. The quadcore Intel Xeon processor provided a speedup of 36 times for the Dean model and 43 times for the HTM model. The Sun UltraSPARC T2+ processor provides a speedup of 17 times over the serial implementation for both the Dean and HTM models. The MPI versions were able to provide near linear speed ups. A cluster of eight Intel Xeon blades was able to provide a speed up of 7.7 times over a single blade for the HTM model and up of 5.9 times for the Dean model. The cluster of eight Playstation 3s provided speedups of 5.7 for the HTM model, and 4.3 for the Dean model over a single Playstation 3. It has been noticed that the speedups on all the processing platforms increased as the network size increased. These speedup numbers were for the largest networks tested.

The Air Force Research Laboratory at Rome, NY has set up a cluster of 336 IBM/Sony/Toshiba Cell multicore processor based Sony PlayStation 3's (PS3s) [43] primarily to examine the large scale implementations of neuromorphic models [40]. This cluster is capable of providing a performance of 51.5 TF and cost about $361K to setup (of which only 37% is the cost of the PS3s). This is significantly more cost effective than an equivalently performing cluster based on Intel Xeon processors [40].

The large clusters were able to model 10^{10} neurons on the cluster. In comparison a mouse cortex in comparison contains about 1.6×10^{7} neurons and 1.6×10^{11} synapses [21]. The number of neurons simulated in this thesis is comparable to a recent study [1] where a 32,768 processor IBM BlueGene supercomputer was able to simulate a rat scale cortex (55×106 neurons and 4.42×1011 synapses) at near real time. However the cost of our computing cluster was significantly lower than the one used in [1]. This indicates that the 336 node PS3 cluster provides a highly economical, yet powerful, platform for neuromorphic simulations.

CHAPTER TWO
RELATED WORK

Cortical models can be categorized based on the level of biological detail examined. There have been several studies examining the acceleration and large scale simulation of these different cortical models. The lowest level of modeling considers the chemical changes inside the soma, axon and dendrites of a neuron [19][30]. These models have significant complexity and provide insights into how neurons work. However they are too complex to develop large scale models of the brain as the computations for a single neuron can be computationally challenging. IBM/EFPL [24] is developing cortical simulations that utilize these models. They model neurons at very low levels of details and have been able to simulate a collection of 100,000 neurons on an IBM Blue Gene/L supercomputer with 8,192 processing cores.

The next level in the modeling hierarchy is based on modeling individual neurons using a set of differential equations. These artificial neural network models were first developed in 1952 [19]. Spiking neural networks fall under the third generation of these models and are currently in wide use [25]. Several large scale models based on spiking neural networks include [1][11][12][20][39]. At the IBM Almaden Research Center, Ananthanarayanan and Modha are using spiking neuron models [1] to simulate brain scale neural systems. They simulated 55 million randomly connected neurons (equivalent to a rat-scale cortical model) using a 32,768 processor IBM Blue Gene/L supercomputer. Djurfeldt et al. [12] simulated a randomly connected neuron level model with 22 million neurons and 11 billion synapses using an 8,192 processors IBM Blue Gene/L supercomputer. Izhikevich and Edelman [20] utilized a neuron level model to develop a brain simulation of a million neurons and about half a billion synapses on a 60 node Beowulf cluster.

Recently, several models of the neocortex have been proposed that are based on modeling mini-columns/columns [3][9][18][23][38]. The models by Dean [9], George and Hawkins [18], and Anderson [3] are based on hierarchical graphical networks and concur well with experimental results. They describe the brain as a hierarchical device that computes by performing sophisticated pattern matching and sequence prediction. Johansson and Lansner [21] utilized a cluster of 442 dual Xeon processors to simulate a randomly connected brain model utilizing recurrently connected neural networks grouped into cortical columns. Anderson et. al. [4] are examining the design and implementation of large scale cortical models based on the brain state in a box model [3].

Several studies have examined the acceleration of various models on multicore architectures. Wu et al. [35] are examining the acceleration of the Brain State in a Box model [2] on the Cell processor in a Playstation 3. They achieve about 70% of the theoretical peak of the processor. Felch et. al. [13] examined the acceleration of the Brain Derived Vision algorithm on the Cell processor. They achieved a speedup of 140 times using a cluster of three Sony Playstation 3 systems over a serial implementation on 2.13 GHz Intel Core processor. Xia and Prasanna examined the parallelization of the exact inference algorithm in junction trees [36] and examined its acceleration on a Sony Playstation 3 based Cell processor [37]. They achieved speedup of about four times a 3.0 GHz Intel Pentium 4 processor.

CHAPTER THREE
BAYESIAN CORTICAL MODELS EXAMINED

The neocortex is known to be organized hierarchically, with both feed-forward and feed-back connections between different regions. For example, it is thought that vision is processed by the following hierarchical regions in the cortex: V1, V2, V4, and IT. Higher level regions in the cortex receive input data from a variety of sources through feed-forward connections. The higher levels can then use this global view to adapt the inputs coming in from the lower levels by sending feedback connections. It has been shown that there are actually more feedback connections within the brain than feed-forward connections [32]. It has also been seen that the different regions within the neocortex have a remarkably similar structural organization. This has led researchers to suggest that the neocortex consists of a uniform computational fabric with the different parts utilizing a similar set of computations. As a result, the brain can adapt underutilized regions for new tasks that no healthy parts of the brain can work on [17].

Hawkins presented [14] a theoretical framework describing the working processes in the neocortex based on the above properties. He described the neocortex as a highly efficient pattern matching device [17] – as opposed to a computing engine. He stated that one of the most important properties of the brain is its ability to recognize patterns under various transformations (i.e. invariant pattern recognition). 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 he cited 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.

The two hierarchical Bayesian models examined in this study capture these properties of the neocortex: hierarchical structure, uniform computations, feedback connections, and invariant pattern recognition. The models consist of several layers of processing nodes arranged hierarchically with all the nodes performing the same set of computations. The nodes can be considered to be the functional equivalent of cortical columns. Input data is preprocessed before being fed to the bottom layer of nodes. There are several feed-forward and feedback passes through the network to allow a proper distribution of information throughout the network. The next two sections examine these two models in detail.

Hierarchical Temporal Memory model

George and Hawkins developed an initial mathematical model [14] of the neocortex based on the framework described by Hawkins in [17]. Their model utilizes a hierarchical collection of nodes that employ Pearl's Bayesian belief propagation algorithm [29]. As shown in Figure 1, each node has one parent and multiple children (hence there is no overlap in the input fields of any two nodes in a given layer). Input data is fed into the bottom layer of nodes (level 1) after undergoing some preprocessing. After a set of feed-forward and feedback belief propagations between nodes 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. The model is trained in a supervised manner by presenting a set of training data to the bottom layer of nodes multiple times.

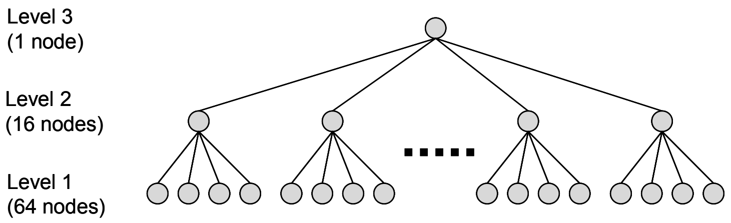


Figure 3.1. Network structure of model implemented.

The computational algorithm within each node of the model is identical and follows equations 1 through 6 below. Before a node starts computing, it receives belief vectors from its parent (π) and children (λ) as shown in Figure 3.2(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 with an internal probability matrix, P_{xu} (generated in an offline training phase), and the belief vector from the parent (see equation 2). The matrix 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 3.2(b).

$$\lambda_{product}[i] = \prod\nolimits_{child} \lambda_{in}[child][i] \qquad (1)$$

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

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

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

$$\lambda_{out}[j] = m_{row}[j] / \pi_{in}[j] \qquad (5)$$

$$\pi_{out}[child][k] = m_{col}[k] / \lambda_{in}[child][k] \qquad (6)$$

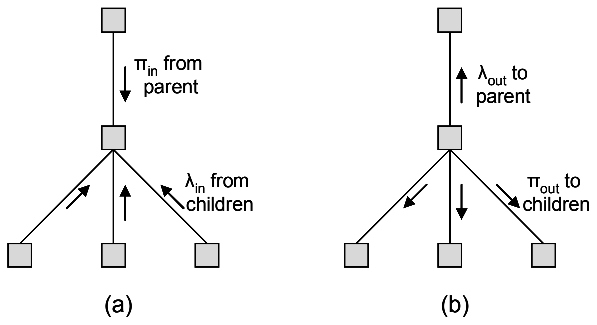


Figure 3.2. Belief transfer in the network (the squares represent computation nodes). (a) Gathering beliefs from parent and children nodes before node computation. (b) Distribution of beliefs to parents and children nodes after node computation.

Example Model

George and Hawkins demonstrate a three layer network implementation [14] based on equations 1 through 6 to model the image recognition process of the visual cortex. Their example utilized a set of 81 nodes with 16 nodes in the layer 2 and four layer 1 nodes under each layer 2 node (as shown in Figure 3.1). It was trained to recognize a set of 91 binary images that were 32×32 pixels in size. These images consisted of characters and several simple shapes (such as a bed, a rake, and a window). The model examines an input image and outputs a belief vector describing the degree of matching between the input and each image from a set of training images. Figure 3.3 shows a sample testing image and the four training images with the highest matches.

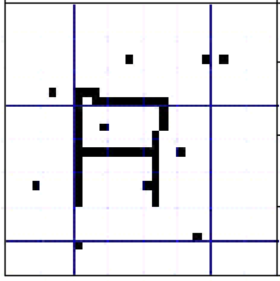
Input Image	Output Belief	
	1st match	
	2nd match	
	3rd match	
	4th match	

Figure 3.3. A sample testing image matched against the training images for the HTM model.

Dean model

Thomas Dean proposed a hierarchical Bayesian model [8] based on the work by Lee and Mumford [23] to model the invariant pattern recognition seen in the visual cortex. The example model examined in this study consists of hierarchy of nodes with each node connected to a set of lower level nodes. There is a degree of overlap in the receptive field of the nodes in some of the layers (such as layer 2 in Figure 3.4). Inputs to the layer 1 nodes are processed through a set of feed-forward and feedback processing steps through the network. A final inference based on this input is produced by the top layer node. The model is trained in a supervised manner by presenting a set of training data to the bottom layer of nodes multiple times.

The input image is preprocessed by a preprocessing layer before being fed to the layer 1 nodes. Each layer 1 node has a 4×4 patch of pixels from the input image corresponding to it. In the preprocessing layer, the 4×4 patch of pixels is transformed into a mixture of Gaussians and this mixture is matched against 16 predefined classes of mixtures of Gaussians. Thus each 4×4 pixel region is represented by a number between 1 and 16, with this number being fed to the corresponding layer 1 node by the preprocessing layer.

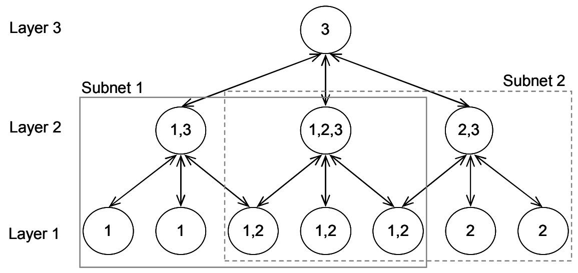


Figure 3.4. A simple example of Thomas Dean's hierarchical Bayesian network model. This example can be divided into three subnets as shown. The nodes are numbered with the subnets they belong to.

Dean examined several approaches to process the Bayesian network structure. The approach examined in this study is the one proposed by Dean where the network is decomposed in to a set of subnets, and each subnet is evaluated individually. A subnet can be defined as a node, its parents, and all the children of those parents in the same level as the original node [8], and as shown in Figure 3.4, a node can belong to multiple subnets. The subnets are identified during the training process and only the largest subnets (those that would not be a subset of another subnet) are utilized. All the subnets in a layer are processed before moving to the next layer of subnets. Each subnet produces evidence to send to the next layer of subnets.

Algorithm 1 shows the overall set of steps in processing the Dean model. The first step is to preprocess the input image (line 1). For any given input image, the network is processed through multiple bottom-to-top (line 3) and top-to-bottom (line 8) passes. In each pass, all the subnets for a certain layer are processed before moving to the next layer. The process has to be repeated at least twice to check for output convergence (line 2). The output is generated by the top level subnet during the upward pass. The processing inside a subnet is similar for both the upward and downward passes (described by lines 5-7 for the upward pass and lines 10-12 for the downward pass).

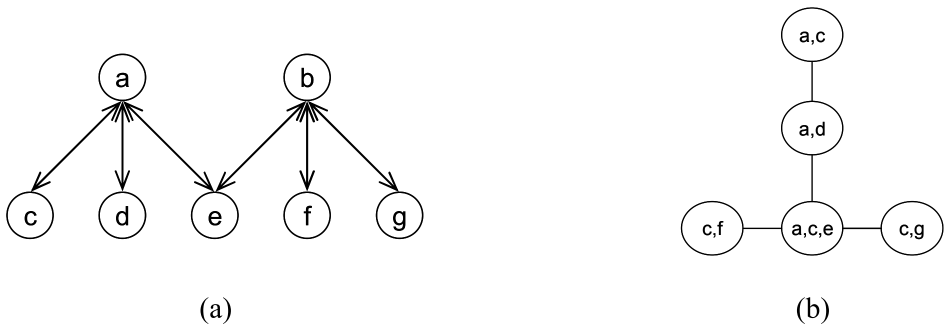


Figure 3.5. A simple example of a junction tree derived from one of the lower level subnets shown in Figure 1. Part (a) shows the subnet with the nodes number 1 through 7. Part (b) shows the junction tree equivalent to this subnet. Each clique in part (b) is numbered with the corresponding nodes from the subnet in part (a) that are used to build the clique.

Algorithm 1. Processing in the Dean model

```
1.  Preprocess inputs: find mixture of Gaussian for each 4×4 pixel patch

2.  Repeat till output convergence:
3.      Upward pass (from layers 1 to 3):
4.          For all subnets in a layer:
5.              Incorporate evidence (from below) and initialize junction tree
6.              Process junction tree (collecting and distributing evidence)
7.              Calculate evidence to send to upper layer of subnets (lambda values)
8.      Downward pass (from layers 3 to 1):
9.          For all subnets in a layer:
10.             Incorporate evidence (from above) and initialize junction tree
11.             Process junction tree (collecting and distributing evidence)
12.             Calculate evidence to send to lower layer of subnets (pi values)

13. Read output
```

In order to process a subnet, it is first converted to its equivalent junction-tree representation (see Figure 3.5). The subnet to junction tree mapping is carried out during training and does not have to be redone during inference (as the mapping is reused). A junction-tree consists of a set of nodes called cliques, where each clique is based on a collection of nodes in the original subnet. Each clique has a potential based on the conditional probability tables of the nodes in the subnet it is composed of. The connection between two cliques is called a separator and has a separator potential based on a reduced form of one of the clique potentials (with the clique chosen being the one sending information to the other). The operations in the junction tree processing consist primarily of element by element multi-dimensional matrix adds, multiplies, and divides. The processing inside a subnet takes place through the following three parts:

Part 1 (lines 5 and 10): Incorporate evidence and initialize junction tree. All the nodes in the network receive π or λ belief values (soft evidence) from the subnets above or below them respectively. All the λ values are initialized to one. Additionally, the bottom layer nodes see the preprocessed input image as hard evidence (E). The junction tree corresponding to a subnet has its clique potentials (ψ_C) initialized based on the subnet node potentials (P(Xi)) and the input evidence:

$$\psi_C = \prod_i (P(X_i|E) \times \lambda_i \times k_{C,i}), \text{ where } k_{C,i} = 0 \text{ or } 1 \qquad (7)$$

The values of $k_{C,i}$ are determined through the training process (here C represents the clique being examined). In the downward pass (line 10), the potential of upper nodes in each subnet get their potentials replaced by the π belief received from above ($P(X)= \pi_X$).

Part 2 (lines 6 and 11): Process junction tree. The Lauritzen and Spiegelhalter's junction-tree algorithm [22] is utilized for exact inference in the tree. The junction tree derived from a subnet is evaluated in a single bottom-to-top (collect evidence) and then top-to-bottom pass (distribute evidence).

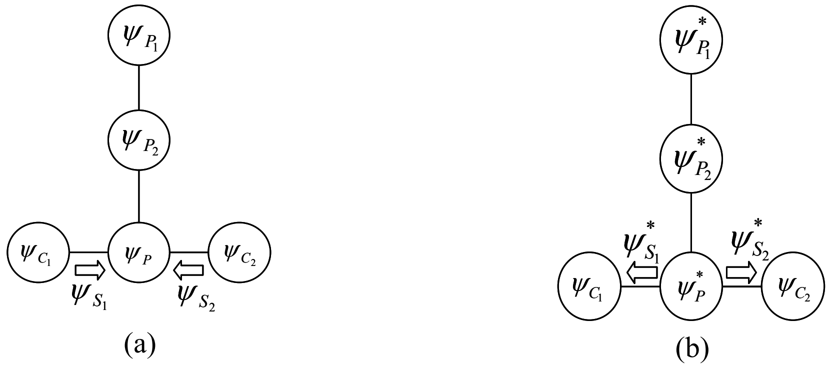


Figure 3.6. Updating evidence within a junction tree takes part in two steps while using the Lauritzen Spiegelhalter algorithm. (a) shows the evidence being collected from children to a parent node (b) shows the evidence being distributed back to the children after the parent has been updated.

In the collect evidence pass, separator potentials (ψ_{S_i}) are first calculated based on the child clique potentials (ψ_{C_i}) as shown in equation 8. The parent clique potential is then updated (ψ_P) based on the separator potentials (ψ_{S_i}) as shown in equation 9. This process is illustrated in Figure 3.6 (a) and continues recursively until the root is updated.

$$\psi_{S_i} = \sum_{C_i \setminus S_i} \psi_{C_i} \qquad (8)$$

$$\psi^*_P = \psi_P \times \prod_i \psi_{S_i} \quad (9)$$

The distribute evidence pass starts once the root clique in the junction tree has been updated through the collect evidence phase. In this pass, new separator potentials ($\psi^*_{S_i}$) are calculated by reducing the potentials of the parent clique (as shown in equation 10). The children are then updated using both old and updated the separator potentials (ψ_{S_i} and $\psi^*_{S_i}$ respectively as shown in equation 11). The updated values are propagated downward recursively until all leaf cliques are updated. The process is illustrated in Figure 3.6 (b).

$$\psi^*_{S_i} = \sum_{P \setminus S_i} \psi^*_P \qquad (10)$$

$$\psi^*_{C_i} = \psi^*_{S_i} \times (\psi_{C_i} / \psi_{S_i}) \quad (11)$$

Part 3 (lines 7 and 12): Calculate evidence to be sent. Evidence to be sent to the next layer of subnets is calculated in this phase. While going through the upward pass (line 7), these are the λ values, and while going down (line 12) these are the π values. Lambda values are updated based on equation 12. Here P(I) is a prior distribution of the training classes generated during the training process, while P(E) is the sum of all the clique potentials in the junction tree. During the downward pass, π values are generated based on equation 13.

$$\text{if } k_{c,x} = 1,\ \lambda_X = (\,P(E) / P(I)\,) \times \sum_{X \setminus C} \psi_C \qquad (12)$$

$$\text{if } k_{c,x} = 1,\ \pi_X = \sum_{X \setminus C} \psi_C \qquad (13)$$

In the example implementation of the model presented by Dean in [8], the model performs recognition of hand written characters on 28×28 pixel images. This example network consists of three layers of nodes connected in a pyramidal form, with the bottom layer consisting of 49 nodes (in a 7×7 layout), the middle layer of 9 nodes (in a 3×3 layout), and the top layer of 1 node. Each layer 2 node has nine layer 1 children (arranged in a 3×3 layout) forming a pyramidal collection. The field of view of each layer 2 node overlaps with its neighbors' by an edge that is one node thick. Thus each layer one node can have up to four layer 2 parents.

Figure 3.7. Sample images from the MNIST library used for training and testing.

CHAPTER FOUR
MULTICORE ARCHITECTURES EXAMINED

Semiconductor constraints have taken the industry towards multi-core architectures. With their inherent parallel processing capabilities, these architectures prove to be both interesting and useful for scientific applications. In this thesis, three multi-core architectures are examined. Two of these exploit vectorization capability along with multiple cores.

The architectures examined include the Intel Xeon E5345, the Sun UltraSPARC T2+, and the STI Cell BE. The platforms it was tested on are explained in the experimental setup section. A brief comparison of the architectural capabilities is listed in the table 3.1. The following sections in this chapter give a brief description of the architectures examined in this thesis.

INTEL XEON E5345

The Intel Xeon E5345 processor examined in this thesis contains four Intel Core based processing cores clocked at 2.33 GHz. These processors contain a 256 KB level one cache per core and an 8 MB shared level two cache. The processor can execute vector instructions (with four floating point operations) using the SSE3 instruction set.

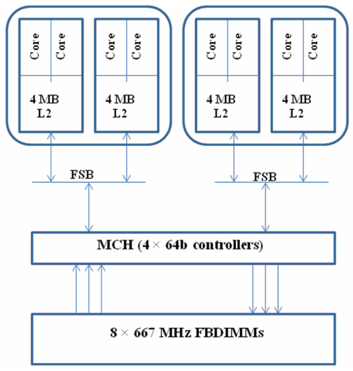


Figure 4.1. Dual-socket, quad-core Intel Xeon E5345 (Clovertown) processor architecture Sun UltraSPARC T2+

The Sun UltraSPARC T2+ processor [33] contains 8 cores running at 1.4 GHz. Each core can execute up to eight threads simultaneously, with up to two threads in each pipeline stage. Thus the entire processor can run a maximum of 64 threads concurrently. Each core contains 8 KB of data and 16 KB of instruction cache, and share a 4 MB level two cache.

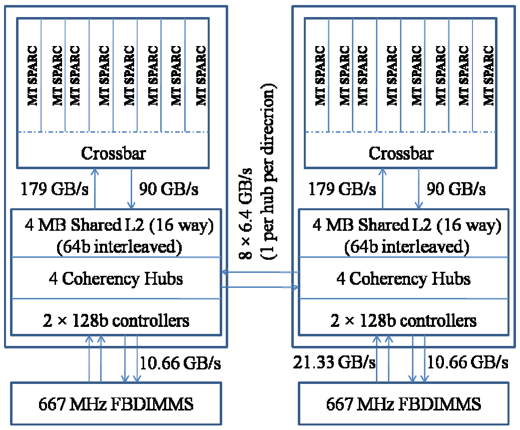


Figure 4.2. Dual-socket × eight-core Sun UltraSPARC T2+ T5140 processor architecture

STI Cell BE Processor

The Cell Broadband Engine developed by IBM, Sony, and Toshiba [16] is a multicore processor that heavily exploits vector parallelism. The current generation of the IBM Cell processor consists of nine processing cores: a PowerPC based Power Processor Unit (PPU) and eight Synergistic Processing Units (SPU). The processor operates at 3.2 GHz. Each SPU is capable of processing up to four instructions in parallel each cycle (eight, if considering fused multiply-add instructions). The processing cores in the Cell utilize in-order execution with no branch prediction. This simplified hardware design means that several software level optimizations are necessary to achieve high performance on the SPUs (these are generally not needed on traditional processors, such as the Intel Xeon). The optimizations include use of vectorization, reducing the frequency of branch instructions through loop unrolling and function in-lining, and explicit memory optimizations. Instead of a processor controlled data cache, each SPU contains a programmer controlled local store to explicitly optimize memory operations. This enables several memory level optimizations not possible on most high performance processors. Since high compute-to-I/O ratios are needed to achieve the full potential of the Cell processor [6], the programmer controlled memory stores are especially important.

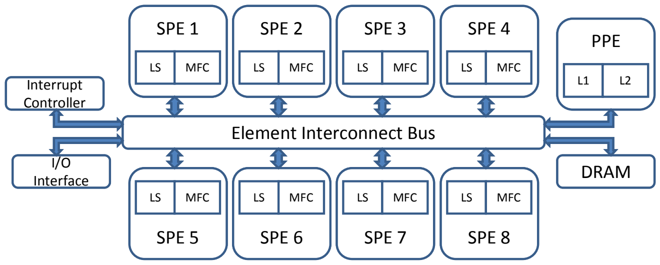


Figure 4.3. STI Cell processor architecture

Table 4.1: Comparative list of the capabilities of the architectures examined

Core Architecture	Intel Core2 (Xeon E5345)	Sun UltraSparc T2+	IBM PPE	SPE
Type	Superscalar out-of-order	in-order	MT dual issue	SIMD dual issue
Clock (GHz)	2.33	1.16	3.2	3.2
Local store	-	-	-	256 KB
L1 Data Cache per core	32 KB	8 KB	32 KB	-
L2 Cache per core	-	-	512 KB	-
# Sockets	2	2	1	
Cores per Socket	4	8	1	8
DRAM Capacity	16 GB	64 GB	256 MB	
Threading	Pthreads	Pthreads	Pthreads	
Compiler	gcc	Cc	Gcc	spu-gcc

CHAPTER FIVE
IMPLEMENTATION

To achieve optimum performance on the multiprocessors, a few optimization techniques had to be used. This chapter describes those techniques in addition to the steps taken in parallelizing the given models.

Optimization

HTM model

All the nodes in a particular layer are independent of each other and can therefore be evaluated in parallel. Therefore in this study, the HTM network was parallelized by assigning groups of nodes in a particular layer to separate processing cores. Nearly all computations in equations 1 through 6 are element-by-element matrix multiplies and divides (thus there are no addition operations needed). In order to accelerate the computations, the matrix values were converted into logarithmic form so that more expensive multiplies and divides could be replaced by less time consuming additions and subtractions. The comparisons involved in equations 3 and 4 could still be performed in logarithmic form and were thus unaffected by this change.

P_{xu} matrix compression and model vectorization

The P_{xu} matrix in equation 2 is large enough that it needs special consideration when examining the vectorization of the nodes. These matrices themselves are extremely sparse, being made up almost 90% zeroes. The computations in equations 1 through 6 are element-by-element rather than dot products. Compressing the P_{xu} matrices can significantly speed up the algorithm computation by skipping over strings of zeros. Thus any vectorization approach needs to consider the compression of the P_{xu} matrix. Two possible approaches to utilize vectorization for the George Hawkins model were examined. The first involves vectorizing the code to process a single image more efficiently. The second approach involves vectorizing the code to process multiple images simultaneously.

Single image vectorization: In this case, equations 1 through 6 need to be vectorized for a single image. Equations 1, 5, and 6 can be vectorized easily if the variables for the equations are padded to be multiples of the vector width. Equation 2, however, cannot be vectorized as easily, given that the P_{xu} matrix is sparse. The feasibility of block compression [34] of the P_{xu} matrix to vectorize the computations in equation 2 has been examined. In order to be efficient, there should be on high density of non zero elements in uncompressed blocks.

Two possible approaches for block compression are to compress along the rows or along the columns of the target matrix. Tables 5.1 and 5.2 show the density of non zero blocks for both row and column wise compression with block sizes of 4 and 8 respectively. Several network sizes (described in details in the experimental setup section) are examined. The results indicate that with a vectorization factor of four, the average P_{xu} uncompressed block contains less than two non-zero elements per block, while a vectorization factor of eight yields at most 2 elements per block on average. Thus vectorizing the equations for single images is not very efficient.

Table 5.1. Block compression of P_{xu} with a block size of 4.

	Compression along rows			Compression along columns		
Network Size	Percentage of non-zero blocks	Average non-zero elements in non-zero blocks	Percentage of non-zero blocks with more than two non-zero elements	Percentage of non-zero blocks	Average non-zero elements in non-zero blocks	Percentage of non-zero blocks with more than two non-zero elements
81	*3.84*	*1.2531*	*20.06*	*3.97*	*1.2605*	*17.72*
181	*5.17*	*1.3035*	*22.82*	*5.02*	*1.3891*	*24.89*
321	*6.23*	*1.3935*	*27.47*	*5.96*	*1.4940*	*28.79*
501	*7.41*	*1.4624*	*30.53*	*6.72*	*1.6483*	*34.48*
721	*7.82*	*1.4848*	*31.4*	*7.12*	*1.6659*	*35.02*

Table 5.2. Block compression of P_{xu} with a block size of 8.

	Compression along rows			Compression along columns		
Network Size	Percentage of non-zero blocks	Average non-zero elements in non-zero blocks	Percentage of non-zero blocks with more than two non-zero elements	Percentage of non-zero blocks	Average non-zero elements in non-zero blocks	Percentage of non-zero blocks with more than two non-zero elements
81	*6.44*	*1.4587*	*27.89*	*6.97*	*1.4374*	*22.74*
181	*8.32*	*1.5837*	*32.66*	*8.38*	*1.6506*	*29.77*
321	*9.75*	*1.7418*	*36.69*	*9.73*	*1.8270*	*33.35*
501	*11.31*	*1.8754*	*39.91*	*10.75*	*2.0571*	*38.98*
721	*11.88*	*1.9125*	*40.38*	*11.36*	*2.0851*	*39.45*

Multiple image vectorization: The computations for any input image are identical throughout the network because each node in the network processes any input given in exactly the same manner. Therefore multiple images can also be evaluated in parallel using vectorization. In this case any compression scheme can be adopted for the P_{xu} matrices. The matrix is compressed by providing a coordinate for each nonzero value in the P_{xu} matrix. Two approaches for dealing with this are to treat the P_{xu} matrix as a linear vector (see Figure 5.1(b)) or to treat it as a two dimensional matrix (see Figure 5.1(c)). In the former case, only one coordinate is needed per nonzero element, while in the latter case, two coordinate values are needed. The first approach results in a higher compression level and thus lower data transfer time. It however does require the generation of a two dimensional (x,y) coordinate for each linear coordinate (for equation 2). Our studies indicate that a two dimensional representation provides the lowest overall execution time. For example, for the 721 node HTM model examined, the single dimensional approach required 18.26 ms on a Playstation 3, while the two dimensional approach required 10.96 ms.

Other compression techniques were also tested; Most notably the Compressed Row Storage (or the Yale format). If the matrix is of size MxN and having D non-zero elements, then the size of the compressed matrix in the Yale format is (2xMxNxD) + M. While in the format mentioned earlier, the size of the compressed matrix turns out to be 3xMxNxD which is slightly bigger than the size attained by the Yale format. Although this format provides a slight advantage in these very sparse (D < 5%) matrices, further optimizations such as loop unrolling prove to be cumbersome and ineffective in general. Hence the position encoding was used for all the runs.

1	**0**	**2**	**0**	**0**
0	**4**	**0**	**0**	**0**
0	**6**	**0**	**0**	**0**
1	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**9**

(a)

1	*0*	**2**	*2*	**4**
6	**6**	*11*	**1**	*15*
9	*24*			

(b)

1	*0*	*0*	**2**	*0*
2	**4**	*1*	*1*	**6**
1	*2*	**1**	*3*	*0*
9	*4*	*4*		

(c)

Figure 5.1. Restructuring the P_{xu} matrix. (a) Original P_{xu} Matrix. (b) Single dimensional position representation, [p :value, *x* :coordinate]. (c) Two dimensional position representation, [p :value, *x,y* : coordinates]

Dean model

As shown in Chapter 3, the nodes in a network in the Dean model can be grouped into subnets and the network would be processed by evaluating subnets rather than individual nodes. Also, as shown in the same section, each subnet was evaluated by processing its junction-tree representation. Each node of the junction tree is called a clique and has a clique potential associated with it. This potential is derived by combining the conditional probability tables of each node in the subnet that forms the clique (these tables are multi-dimensional with a maximum of five dimensions in our study).

There are two possible approaches to parallelize the evaluation of the Dean model: the first is at the subnet granularity and the second is at the clique granularity. This latter approach will yield a higher level of parallelism as there are more cliques than subnet (given that a subnet can be decomposed into multiple cliques). Dependencies between the cliques may limit the number of cliques that can be evaluated in parallel at any given level within a junction tree. Both approaches were evaluated and found that for the networks examined, the clique based approach had a better utilization of the available processing cores. In both approaches the order in which the subnets or cliques will be evaluated is predetermined and does not vary with the network inputs.

Vectorization: As with the HTM model, there are at least two approaches to vectorization for this model: vectorizing the operations for a single image and vectorizing to evaluate multiple images simultaneously. In the former case, matrix operations would have to be vectorized as a large portion of the junction tree evaluations consist of multi-dimensional matrix operations. In the networks examined, these matrices had up to five dimensions with each dimension being up to 16 elements wide. The matrix operations included element-by-element matrix multiplies and divides. There were also matrix dimension reductions which essentially were summations along a given dimension of the matrix. Not all of these operations can be vectorized efficiently, particularly as the matrix dimensions were of small widths (that were not always multiples of the vectorization factor).

Since the model evaluates any input data in precisely the same way, multiple inputs can be evaluated in parallel through vectorization. In case of a vectorization factor of four, there will be four versions of each matrix (one for each image). The same set of operations will be carried out for all four versions of each matrix. In this case vectorization can be applied to almost 100% of all the operations.

Parallelization

The parallel implementation of the two models had a similar flow on the different multicore architectures examined (Cell, Intel Xeon, and Sun UltraSPARC T2+). In general one thread was created for each core at the start of the program. Since the UltraSPARC supports multiple threads per processor, it was tested it with multiple threads per core. Each thread was assigned tasks intermittently by the master thread and was terminated only at the end of the program. Creating threads only once and letting them run for the lifetime of the program significantly reduces the thread creation overhead; this issue has been discussed at great length in several recent papers including [5][16][37].

HTM Model

In case of the HTM model, each thread is given a set of nodes to evaluate. Nodes are assigned to threads in a round robin manner at runtime. Once nodes are assigned, each thread would need to read in the node's P_{xu} matrix and input beliefs. In the IBM Cell processor, this is carried out explicitly using a DMA transfer to bring values from the external memory to the corresponding SPU's local store. In most general purpose multicore architectures these values will be stored in global variables and thus be brought in through cache misses. The nodes are then evaluated to generate a set of output beliefs that have to be either transferred through DMA transfers in the Cell or through writes to global variables in other architectures. The node computations are evaluated primarily using a set of four loops: the first for equation 1, the second for equations 2, 3, and 4, and the third and fourth for equations 5 and 6 respectively. The second loop is the most time consuming (taking about 30% of the overall time). In this loop the value generated by equation 2 is processed immediately by equations 3 and 4 (thus the temporary variable F_{xu} is never stored). Once the nodes assigned to a thread have been evaluated, the thread will wait for the next set of nodes to process. This assignment of multiple nodes at a time reduces inter-thread communications (this is particularly significant for the mailbox communications overhead in the Cell processor). Nodes assigned to threads can be from any level in the HTM network.

For the MPI implementation, a similar procedure was used for parallelization. But instead of assigning the groups arbitrarily, they were grouped into families. i.e. a second layer node along with all its children were accounted for as a family. These families were assigned to different machines in a round robin manner. Every processor had a copy of the network parameters it needed. This allowed us to minimize the MPI communication time. MPI communication was thus needed only when the nodes at layer 2 have to communicate with the root node and to send the input image information to all the processors.

Dean Model

In case of the Dean model, a similar set of processing takes place. The threads are assigned a set of subnets or cliques depending on the level of parallelism implemented. Each thread then brings in the clique and node potentials needed along with any necessary subnet input beliefs. The outputs of a thread are the updated potentials and any subnet output beliefs generated. The three steps to process a subnet are:

1) incorporate evidence (lines 5/10 in Algorithm 1),

2) process junction-tree (lines 6/11), and

3) calculate evidence to send out (lines 7/12).

For the subnet based parallelization approach, these three steps are carried out serially for each subnet on each SPU. For the clique based parallelization approach, the three steps for a subnet are parallelized cross multiple SPUs. In both approaches the multiple subnets present in a layer are evaluated in parallel.

In the earlier approach, the equations7 through 15 (of a particular subnet) were evaluated on an individual processing core thereby decreasing the data transfer time. But the number of subnets available was limited and this would not use all the processing cores present on the machine. Therfore the latter approach was used, where Equations (8 through 15) were evaluated by considering each instance of these equations individually. Each instance of these equations corresponded to a unique set of cliques, and thereby could be evaluated in parallel. Although this increased the data flow between the processing cores, they were able to fully utilize the computational resources. Both approaches were tested and the results are presented in Chapter 7.

For the MPI implementation the same method of parallelization (using parallelization by cliques as opposed to subnets) was used irrespective of the number of machines available. This is because, any special grouping (as done in the HTM case) would only result in us using the subnets based parallelization, which it has already been shown to not use the complete resources. Although the initial network parameters are stored on each machine, the data communication takes place between the machines at every level and hence is much higher compared to the HTM model. This is due to the high level of connectivity present in this model.

Architecture specific optimizations

The Cell processor requires some specific code optimizations to achieve high performance. The code optimizations described in earlier sections (vectorization, minimizing thread creation, and reducing inter-thread communications) are essential to obtain high performance on the Cell architecture. Several other explicit code changes are needed as well, including branch elimination and double buffering. Since the SPUs in the Cell processor do not contain any branch prediction units, it is essential to reduce the number of branch instructions in the code run on the SPUs. This was accomplished by unrolling loops and in-lining most function calls. Given that data transfers to the SPUs on the Cell are explicitly programmer controlled, double buffering was needed to ensure that data transfers took place in parallel with data evaluation. In this process, once the data required for the first iteration of a loop has been transferred, the first iteration of the loop can be evaluated simultaneously with the DMA data transfer for the second iteration of the loop. For the Intel and Sun implementations of the models, the POSIX thread library was used for thread implementations. On the Intel platform SSE3 instructions were used for vectorizing operations.

CHAPTER SEVEN
MULTICORE IMPLEMENTATION

The experimental setup of the platforms that was used to implement the models is described in this chapter followed by the results of these implementations. The results show that the multicore processors provide significant performance gains for the Bayesian cortical models examined.

Experimental Setup

Four hardware platforms were utilized in this study, one was Intel Xeon based, two were STI Cell based, and one was Sun UltraSPARC T2+ based. The Intel Xeon platform utilized was a blade on the Palmetto Cluster at Clemson University. Each blade on the system contained two quadcore Intel Xeon processors running at 2.33 GHz (model E5345), had 12 GB of DRAM, and ran the CentOS 5 operating system. The STI Cell platforms utilized was a Sony Playstation 3 at Clemson University and an IBM QS20 cluster at Georgia Tech. The Playstation 3 has one Cell processor on which six of the eight SPUs are available for use and contains 256 MB of DRAM. This platform was running Fedora Core 6 with IBM Cell SDK 2.1. The QS20 blade utilized had two Cell processors, each with all eight cores available, 2 GB of DRAM, and also used IBM Cell SDK 2.1. The Sun UltraSPRAC T2+ platform utilized was a Sun SPARC Enterprise T5140 running Solaris 10. This system contained 2 Sun UltraSPARC T2+ processors and 64 GB of DRAM. All the programs were compiled with -O3 optimizations using *gcc*. On the UltraSPARC platform, one processor was used for running the operating system, while the other was used to run the hierarchical Bayesian models, with each thread of the model bound to a specific core to ensure optimum performance.

Five networks with varying input image sizes were developed to examine the acceleration of the HTM model on the multicore platforms. 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 grid. Table 7.1 lists details about each of the networks examined including the number of nodes implemented in each network and the input image size. The smallest network was identical to the example presented by George and Hawkins. In order to train the different sized networks, the training algorithm described in [14] was used to generate the internal P_{xu} matrices for the networks. A subset of 76 of the 91 binary image categories presented in [14] was utilized for the training of these networks since these were used in the training example provided by the authors of the model. The set of images chosen would affect the runtimes on all the processors similarly. All the nodes in each layer are processed in parallel. The model was optimized separately for the different architectures. A set of nodes to be implemented was assigned to each thread (an SPU in the case of the cell processor), and these set of nodes were implemented in serial by each thread. The set of nodes to be assigned to each thread was pre-assigned to optimize the load on each thread.

Table 7.1. HTM configurations evaluated

Network input size	**32×32**	**48×48**	**64×64**	**80×80**	**96×96**
Total Nodes	81	181	321	501	721
Layer 3 nodes	1	1	1	1	1
Layer 2 nodes	16	36	64	100	144
Layer 1 nodes	64	144	256	400	576

Four networks with varying input image sizes were developed to examine the acceleration of the Dean model on the multicore platforms. As shown in Table 7.2, all the networks had three layers. The smallest network was identical to the example presented by Dean in [8]. Dean utilized 10,000 images from the MNIST database [26] by Yann LeCun for training and testing of his model. These consist of one thousand versions of 10 objects (handwritten numerals 0 to 9 from the MNIST database), resulting in 10,000 images. The images are 28×28 pixels in dimension (Figure 2.3 shows some samples from this database). The smallest network was trained with the 28×28 images in the database, while the larger networks were trained with zero padded versions of these images.

Table 7.2. Dean model configurations evaluated

Network input size		**28×28**	**36×36**	**40×40**	**52×52**
Nodes	Total	59	98	110	186
	Layer 3	1	1	1	1
	Layer 2	9	16	9	16
	Layer 1	49	81	100	169
Subnets	Total	6	11	6	11
	Layer 3	1	1	1	1
	Layer 2	1	1	1	1
	Layer 1	4	9	4	9

Dean's implementation of the model was in Matlab and utilized Kevin Murphy's Bayesian Network Toolbox (also written in Matlab) [27]. A C implementation of the model along with relevant parts of the Bayesian Network Toolbox was developed. Although C++ Bayesian Network libraries are available, they would need significant modifications in order to be utilized in our study. These include, parallelizing to run on multiple cores, vectorization using Cell SPU SIMD intrinsics, and being able to handle the DMA data transfers needed for the explicit memory management of the SPU local stores. The model and the relevant Bayesian Network libraries were optimized separately for the different architectures. In the Cell version, the PPU assigned a set of subnets or cliques to be processed to each SPU.

For each platform, the performance of the models was examined using the maximum number of cores per processor, and for the Cell and Xeon architectures, with all the processors on a blade. Thus the following ten parallel configurations were tested:

1. Intel blade with 4 and 8 threads.
2. Playstation 3 with 6 SPU threads.
3. QS20 with 6, 8 and 16 SPU threads.
4. Sun UltraSPARC T2+ with 8, 16, 32, and 64 threads.

A six SPU thread implementation on the QS20 was examined to compare it against the Playstation 3 (PS3) performance. A serial version of the program was developed and tested on the Sony Playstation 3's Cell PPU. All the implementations (both serial and all parallel) utilized the data structure optimizations for the models listed in earlier sections (such as P_{xu} matrix compression in the HTM model).

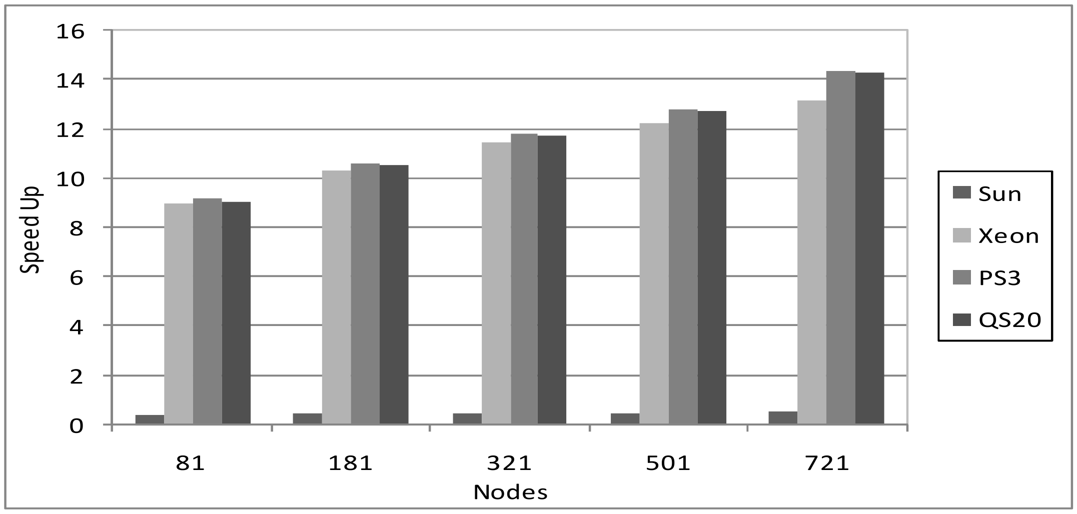
Results

Speedup

Figures 7.1 and 7.2 present the speedup of each of the parallel implementations over the serial PPU implementation. From these figures it is seen that the parallel implementations of the models provide a significant performance gain over their serial implementations. This is mainly due to the use of multiple cores and the use of vectorization on the Intel and Cell architectures.

There is sufficient parallelism in the models examined, so that for all of the platforms, use of more cores provided higher speedups. Our experiments showed that increasing the number of threads on the Intel Xeon blade beyond 8 provided no further improvement in performance. The Dean model produced a higher speedup than the HTM model for all the platforms examined. It is possible that the larger number of training categories in the HTM model produces larger potential tables, which translates to more data transfers, thus limiting its speedup over the Dean model.

For both models, it is seen that the Cell processor outperformed both the Intel Xeon and the Sun UltraSPARC T2+ processors. The Playstation 3 with 6 available SPU cores outperforms the Intel Xeon processor (with 4 cores) by about 1.9 times for the HTM model and by 2.4 times for the Dean model. As a result the Playstation 3 also outperformed the blade with two Intel Xeon processors. The speedup of the Cell processor on the QS20 with all 8 SPU cores available over a single Intel Xeon processor was about 2.3 times for the HTM model and about 3 times for the Dean model. Utilizing both Cell processors on the QS20 (16 threads) provides only a 11% performance gain for HTM model and a 22% performance gain for the Dean model over one Cell processor (8 threads). This is due to the memory accesses becoming a bottleneck as calculation times become close to data access times (as shown in Table 7.3). This effect is not seen on the Sun processor when going from 8 to 16 threads as the calculations take much longer on that system.



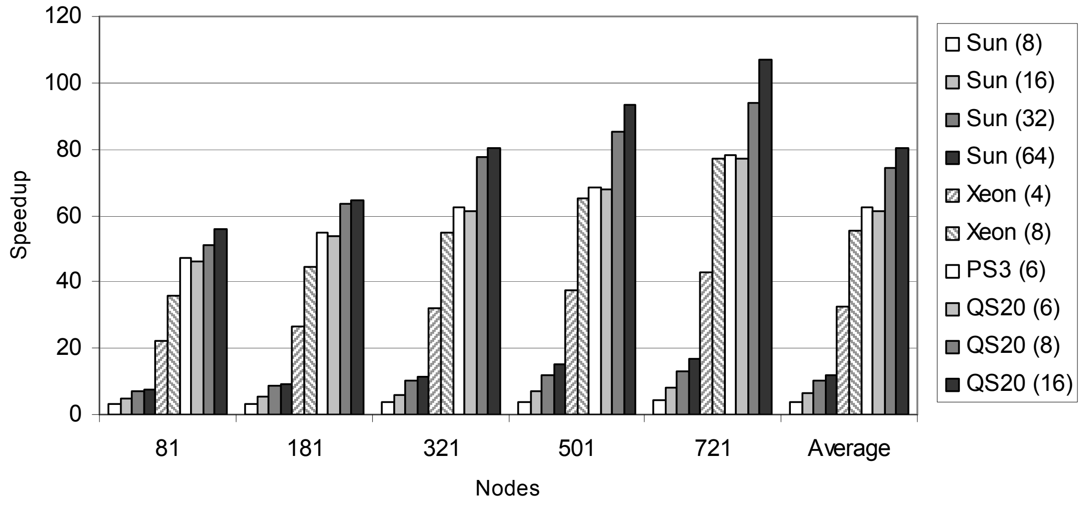
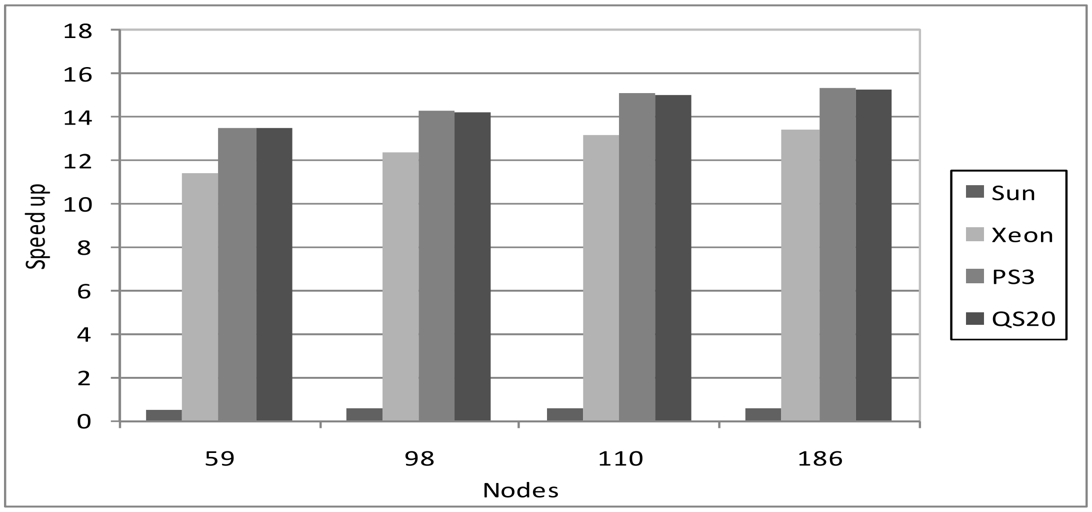
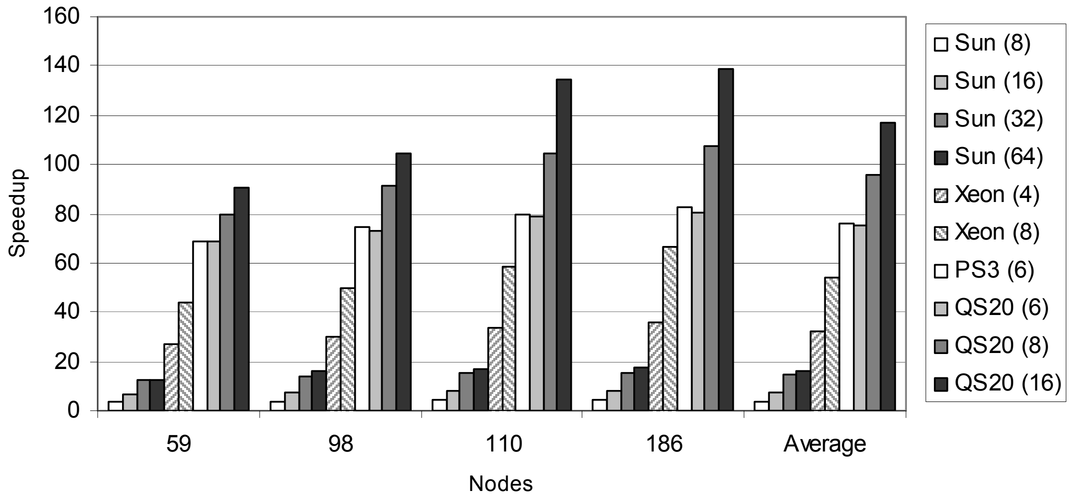


Figure 7.1. Speedup for the HTM model on different multicore architectures over the Cell PPU. (a) The speed up is shown for single thread implementations. (b) The speed ups are shown for multi thread implementations.The numbers in parenthesis in the legend represent the number of threads utilized on each platform.



(a)



(b)

Figure 7.2. Speedup for the Dean model on different multicore architectures over the Cell PPU. (a) The speed up is shown for single thread implementations. (b) The speed ups are shown for multi thread implementations.The numbers in parenthesis in the legend represent the number of threads utilized on each platform.

On the UltraSPARC processor, the Dean model provides speedups of about 2 when going from 8 to 16 threads and when going from 16 to 32 threads. The speedup from 32 to 64 threads is minor (about 1.1 times for the 186 node network). The HTM model provided lower speedups than the Dean model: 1.9 times for the largest model tested when going from 8 to 16 threads, 1.7 times when going from 16 to 32 threads, and 1.3 times when going from 32 to 64 threads.

The Sun processor provides a lower speedup than the Xeon and Cell processors because of a lower clock frequency and a lack of vector capabilities. If multiple images were not available to process simultaneously (such as if there were only one small camera source), then it would not be able to take advantage of the vectorization utilized. In this case the performance of the Intel and Cell architectures would be about one fourth of their current values. Since the Sun does not support vector operations, its performance would not be affected. In this situation, the Sun processor with 64 threads would actually be faster than the Xeon processor with 4 threads; about 2 times for the largest HTM model and 1.6 times for the largest Dean model.

Runtime breakdown of models

Figures 10 and 11 show the runtime breakdowns of the HTM and Dean models respectively on the Cell processor (on the Playstation 3) and the Intel Xeon processor (4 thread implementation). The runtime break downs are given for the smallest and the largest network sizes for both the models. This is done to compare the change in each part of the algorithm with the scaling of the model. The time for signaling between the different threads on all the platforms was insignificant due to the pre-assigning of nodes to different threads at the start of the program. Therefore this time is not listed separately in the timing breakdown.

For the Cell platform, the non-overlapped memory access time is calculated by taking the difference between the overall runtime of the application and the runtime with DMA data transfers commented out of the code. This is the part of the DMA accesses that could not be overlapped with computations (generally through double buffering). On the Intel Xeon platform, this time was calculated by taking the difference between the overall runtime and a version of the code with all global variables in the threads converted to local variables (synchronization barriers between threads were kept intact). The number of computations (array accesses and other operations) was kept the same in both cases.

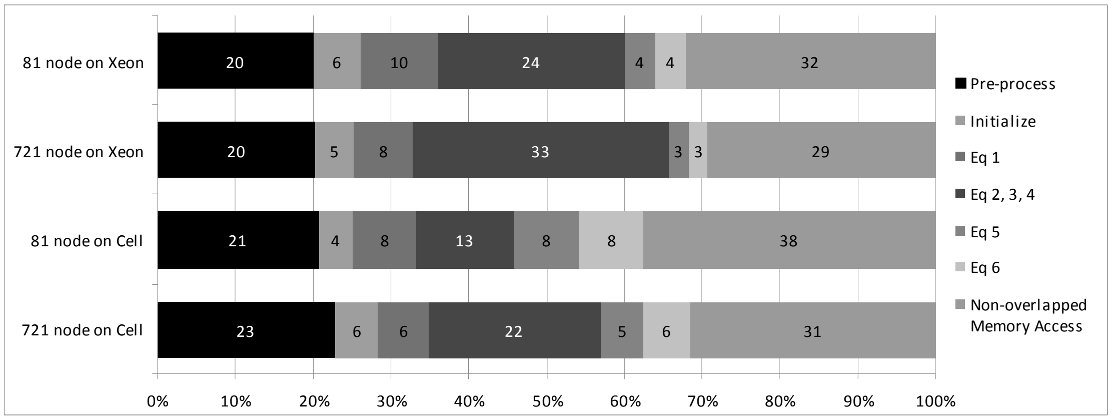


Figure 7.3. Runtime breakdowns for the HTM model on the PS3 and Xeon processors (a) Runtime breakdown for the 81 node network on the Playstation 3. (b) Runtime breakdown for the 721 node network on the Playstation 3. (c) Runtime breakdown for the 81 node network on the Xeon Processor. (d) Runtime breakdown for the 721 node network on the Xeon Processor.

The results show that on the Cell processor, DMA transfers that could not be overlapped can be a significant percentage of the overall runtime. However this fraction decreases as the network sizes increase since the nodes in the network become more complex and thus have more computations to be carried out per node. This is seen by the increase in the computation percentage for equations 2, 3, and 4 in the HTM model and in the percentage of time for getting evidence in the Dean model. Stalls due to global variable accesses on the Xeon processor (listed as non-overlapped memory access in Figures 7.3 and 7.4) showed similar trends as well.

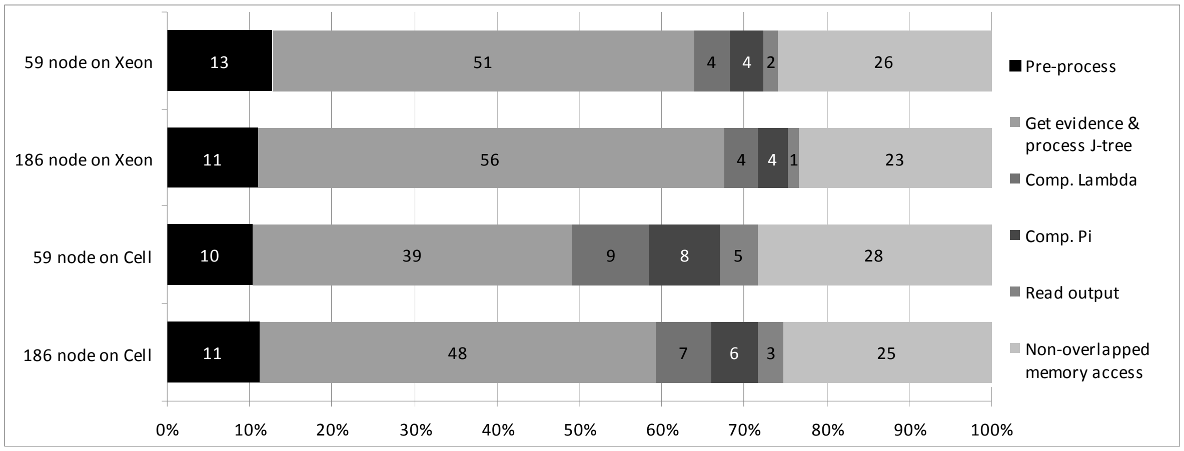


Figure 7.4. Runtime breakdowns for the Dean model on the PS3 and Xeon processors (a) Runtime breakdown for the 59 node network on the Playstation 3. (b) Runtime breakdown for the 186 node network on the Playstation 3. (c) Runtime breakdown for the 59 node network on the Xeon Processor. (d) Runtime breakdown for the 186 node network on the Xeon Processor.

The overall DMA is unlikely to change with the number of cores used on the Cell processor as these accesses go to a centralized memory system. However the overall computation time is likely to decrease with more cores due to increased parallelism. Hence, as the number of cores increase, the DMA time can exceed the computation time, thus limiting the speedup seen with increasing cores. This effect is seen in Figures 7.1 and 7.2: the speedup does not double when going from 8 to 16 cores. Although this could be due to the impact of off-chip memory buses, the results in Table 7. seem to indicate that it is due to a memory bottleneck. Table 7.3 shows the runtime breakdown of the largest HTM and Dean networks on the Cell processor platforms examined: Playstation 3 with 6 SPU, and QS20 with both 8 and 16 SPUs. While the computation time decreased with increasing numbers of cores, the non-overlapped DMA time increased slightly (since the computations started taking less time than data transfers). To alleviate this issue, a higher memory bandwidth would be needed. This would be seen by having each cell processor have access to its own dedicated memory.

Table 7.3. Run time break down of the largest HTM and Dean models on the QS20 with 6, 8, and 16 threads. All times are in ms.

	HTM			Dean		
SPUs	6	8	16	6	8	16
Computation Only (ms)	7.51	5.45	3.50	12.10	8.10	4.50
Non-Overlapped DMA (ms)	3.45	3.60	4.45	4.10	4.34	5.12
Total (ms)	10.96	9.05	7.95	16.20	12.44	9.62
% of DMA in runtime	31.47	39.77	55.97	25.30	34.88	53.22

Parallelization strategy for the Dean model

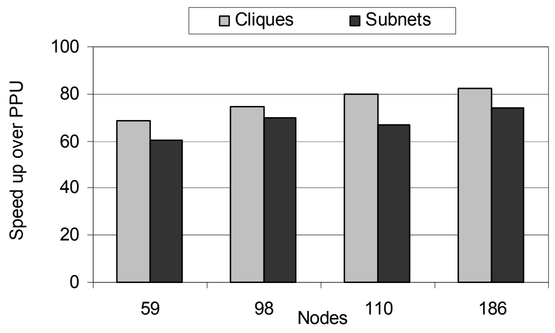


Figure 7.5. Parallelization of the Dean model by cliques vs. subnets. The 59 and 110 node networks are using only 4 SPUs because of the limited set of subnets on those networks. The other two are using six SPUs.

Figure 7.5 compares the two parallelization approaches examined for the Dean model: clique based and subnet based. All the subnets in a layer can be evaluated in parallel. The networks with 59 and 110 nodes had fewer subnets in level 1 than the 98 and 186 node networks. Thus the former set of networks provided lower speedups than the latter set when parallelized by subnets. For all the networks, there were more cliques that could be evaluated in parallel than subnets (since each subnet could be decomposed into multiple cliques). Thus the clique based parallelization approach provided higher speedups for all the network sizes evaluated.

CHAPTER EIGHT
LARGE SCALE IMPLEMENTATION

Given the performance gains achieved on the multicore architectures, the next step toward large scale capabilities of the cortex was to port these models onto computing clusters. MPI was used for communication between all the machines in each cluster described in this chapter. This chapter talks about the initial MPI implementation of the networks, described in earlier sections, on two clusters. One based from the ARSC (based on PS3s) and another from Clemson University (a part from the palmetto cluster based on Xeon E5345 blades). The results of these experiments are compared against the multicore performance to verify the scalability and parallelizability of these models across the two different multiccore platforms.

ARSC PS3 Cluster

The ARSC PS3 cluster consists of 9 PS3s with one of them acting as the head node. All of these are connected to each other through a high speed switch. All the nodes had Fedora 9 installed on them. IBM's Cell SDK 3.1 was utilized for building cell specific applications. Only 6 of the PS3s were available for the implementations described in this chapter, and the head node was not part of our implementations.

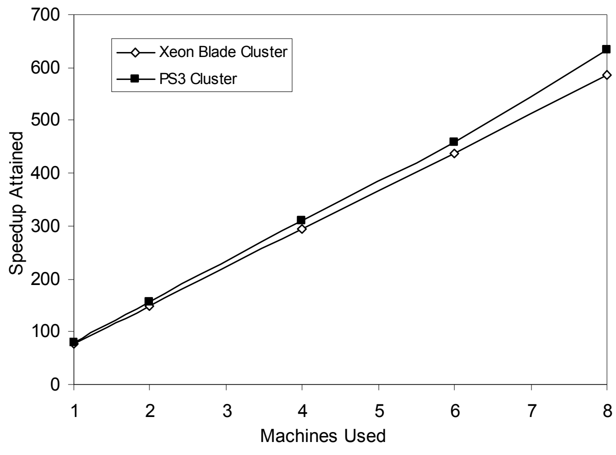
Palmetto Cluster

The palmetto cluster present at Clemson University is a cluster of 257 Intel Xeon blades. Each blade has two quad core (Intel Xeon E5345) processors. The operating system installed on all the systems in the cluster is CentOS 5. 8 of these blades has been used for the implementations described in this chapter. MPICH2 was used for implementing the MPI code on this cluster.

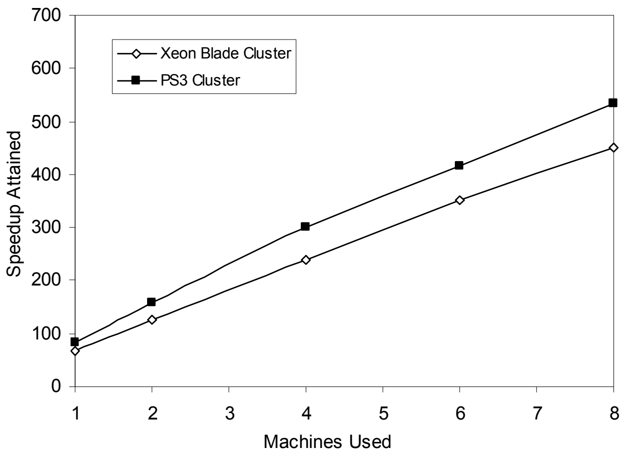
Network Configurations Tested

The largest network to be implemented on a single machine has been tested out for the performance gain in clusters. All the network parameters remain the same from earlier implementations. The clusters were tested with varying number of nodes and varying number of threads per node. This was done to accurately predict the effect of MPI on each model, with the increasing number of cluster nodes.

Results



(a) HTM

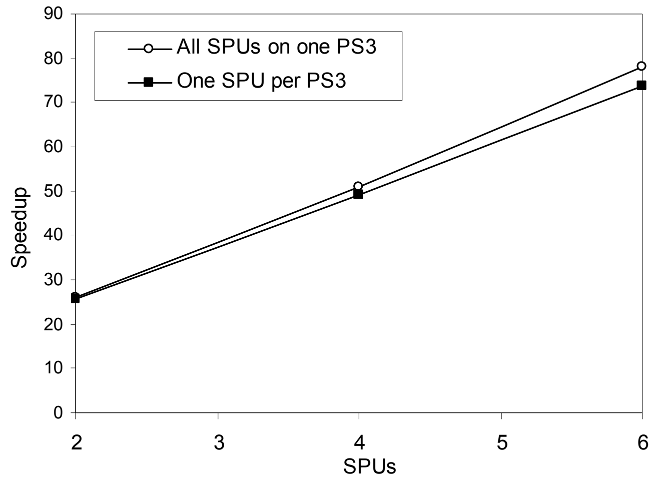


(b) Dean

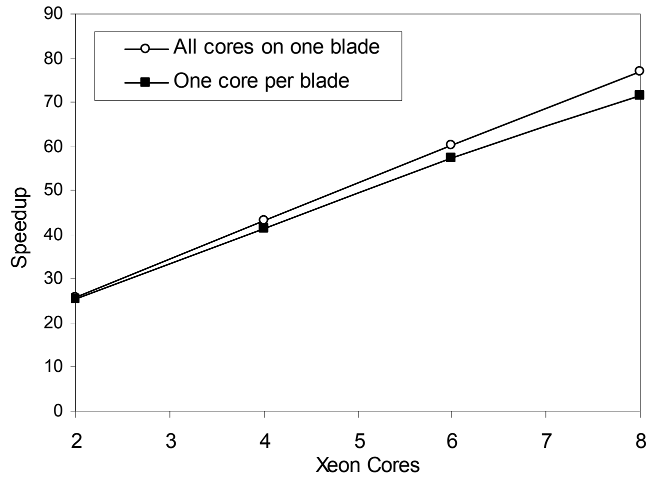
Figure 8.1. Performance scaling of the MPI based implementations of the largest models examined.

Figure 8.1 shows performance scaling of the MPI implementation of each model. Two clusters consisting of PS3s and Xeon blades were utilized with all the cores on each machine being used. The largest network for each model was implemented (721 nodes for HTM and 186 nodes for Dean). The performance of both models scaled with the increase in the number of cores used. The Dean model however has a lower performance gain than the HTM model. This is because 1) the HTM model has more computation units (576 nodes in layer 1 of HTM vs. 225 cliques in layer 1 of Dean), and 2) the Dean model has higher connectivity between its cliques than the

HTM model has between its nodes. This leads to lower parallelism and higher communication in the Dean model compared to the HTM model.

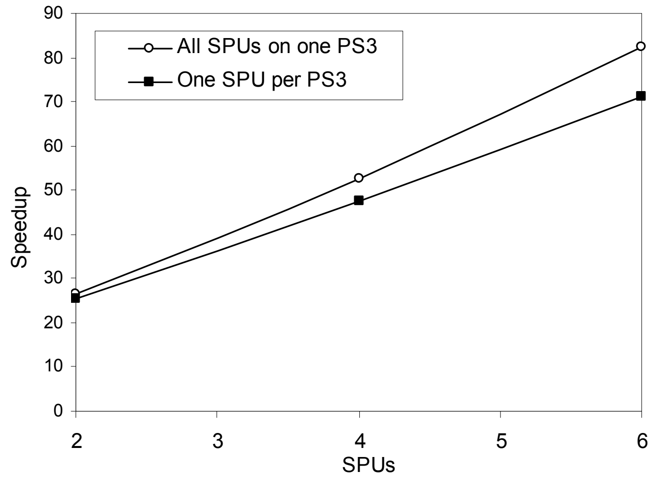


(a)

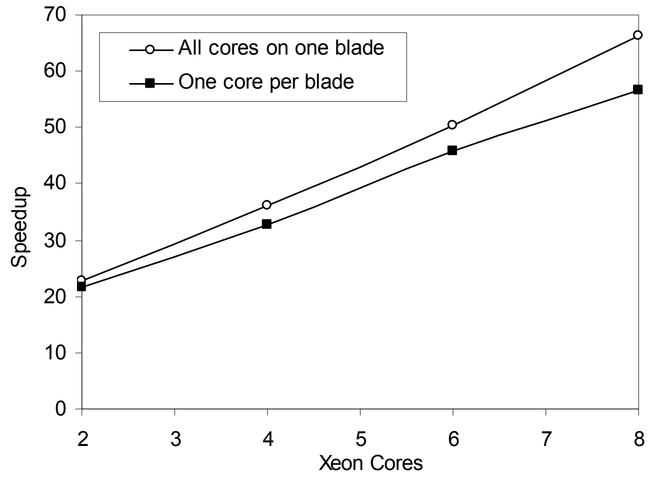


(b)

Figure 8.2. Speedup for the HTM model using the varying number of cores on (a) ARSC Cluster (b) Palmetto Cluster



(a)



(b)

Figure 8.3. Speedup for the Dean model using the varying number of cores on (a) ARSC Cluster (b) Palmetto Cluster

Figures 8.2 and 8.3 examine the impact of a purely MPI based parallelization scheme over a purely multicore parallelization approach. The MPI based parallelization scheme used only one compute node per machine, with the machines communicating through MPI. The multicore approach utilized only the cores available on a single machine. The results show that in all cases, the MPI based approach had a lower performance than a multicore based approach. This is primarily due to the higher communication cost of MPI (as it has to go through multiple NICs and the network connecting the different machines).

In the Dean model, the MPI approach had a higher performance loss over the multicore approach than the HTM model. This is because the Dean model has a higher connectivity between its cliques than connections between HTM nodes. The higher connectivity in the Dean model leads to greater MPI communications as the number of cores utilized is increased. As shown in Table 8.1, the fraction of overall communications going through MPI remains constant in the HTM model, while the fraction increases in the Dean model. In the HTM model the number of level 2 nodes is much higher than the number of cores examined (144 nodes vs. up to 8 cores), leading to a constant

Table 8.1. Data Transfer requests on various implementations of (a) HTM model (b) Dean model

Machines	1	2	4	6	8
Local	3605	3317	3317	3317	3317
MPI	0	288	288	288	288
Total	3605	3605	3605	3605	3605
%MPI	0	7.9889043	7.988904	7.9889043	7.9889043

(a)

Machines	1	2	4	6	8
Local	1112	976	932	902	888
MPI	0	136	180	210	224
Total	1112	1112	1112	1112	1112
%MPI	0	12.230216	16.18705	18.884892	20.143885

(b)

Towards Biological Scale

From the MPI implementation described earlier, it has been shown that the HTM model scales well (near linear) with the increasing number of nodes, while the Dean model does not provide such performance gains. To scale these models to have biological relevance (i.e. to simulate a model that has comparable number of nodes which can represent a mammalian brain), the HTM model proves to be the best option. Though this model scales linearly on both Intel Xeon and STI cell platforms, the performance gain on a single PS3 proves to be higher than that of a Xeon blade.

Hence to simulate a biological scale Bayesian network based cortical model, large scale versions of HTM were developed and were tested on the AFRL cluster with 300 PS3s. Though these models generated did not perform actual recognition, care was taken to preserve the complexity and structure of the models described in earlier sections. The following sections describe the setup used for the implementations and the results that were generated.

AFRL Cluster Setup

On the AFRL cluster utilized, approximately 300 out of the 336 PS3s were available for use. Our studies utilized only the PS3s on the cluster and did not run any code on the Xeon head nodes. In our runs, any impact of using PS3 from different clusters on the overall runtime had not been noticed. This indicates that the MPI overhead for using PS3s in different subclusters and within one subcluster were similar.

Several network structures with varying numbers of nodes, layers and complexities were simulated to examine their performance and the scalability of the model. Unless specified, the results in the following section are based on the 3 layer network parameters listed in Table 8.1. In all the studies each middle layer (layer 2) node had four bottom layer (layer 3) children. Each of the bottom layer nodes always looks at a 4x4 patch of the input image.

Table 8.2. Sample network structure used on the cluster

Layers	3
L1 States	100
L1 Density	100
L1 Children	1600
L2 States	500
L2 P_{xu} Density	3%
L2 Children per Node	4
L3 States	150
L3 Density	3%
SPUs	6
PS3s	1

The P_{xu} matrix of each node (used in equation 2) is typically developed through a training phase. Since the complexity of these matrices was varied in this study, the networks were utilizing randomly generated P_{xu} matrices. As a result the HTM model examined in this study were not performing actual recognition, and thus the input images were chosen to be random combinations of ones and zeros. Since the model performs the same set of computations for any input image, the actual content of the input image did not impact the results. The densities of the P_{xu} matrices listed in Table 8.1 are based on a network that was actually trained to recognize 76 image categories given by the original HTM network.

AFRL Cluster Results

Scalability Analysis

The first set of results show the scalability of the HTM model on the cluster of PS3s. Several three layer networks were tested with a varying number of nodes and PS3s. All the nodes within each layer had the same complexity (P_{xu} dimensions and density).

Figure 8.4 shows the performance of the cluster with a fixed set of nodes assigned to each PS3. Thus varying the number of PS3s would proportionately change the overall number of nodes in the networks modeled. Based on the limits of the virtual memory system of the PS3 and the complexity of the nodes utilized, up to 8000 nodes were modeled on each PS3. The results show that the nodes per second throughput for the cluster scaled up linearly with the number of PS3s. This indicates that compute to communication ratio for the model is quite high. With 500 nodes per PS3, the nodes per second throughput was slightly lower because of an increased fraction of time being spent on communication.

Figure 8.5 shows the runtime per pass for different network sizes on 324 PS3s. The results indicate that the runtime per pass increases almost linearly with the number of nodes evaluated by the cluster. Thus the nodes per second throughput of the cluster does not vary significantly with the network size for a constant node complexity. The maximum throughput observed with 324 PS3s was 51.2 million nodes per second (which equates to 158k nodes/s per PS3).

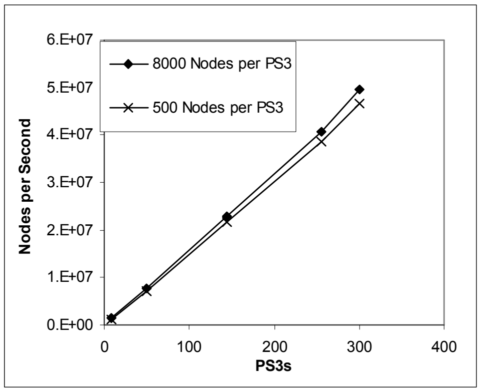


Figure 8.4. Time taken to implement a network when each PS3 was given equal load.

Figure 8.6 shows the change in the runtime of different networks with variations in the numbers of PS3s. The results indicate that overall runtimes decrease hyperbolically with the number of PS3s. As expected, smaller networks reach a limiting point (knee of curve) with fewer

PS3s, while larger networks show a significant decrease in runtime with larger numbers of PS3s. The performance limit represents the point at which the MPI communication and DMA transfer times become dominant.

Figure 8.7 shows the change in runtimes, with variations in the number of SPUs (the number of nodes and PS3s were kept constant). It can be seen that with a larger number of nodes, the runtimes scaled down proportionately with the number of SPUs.

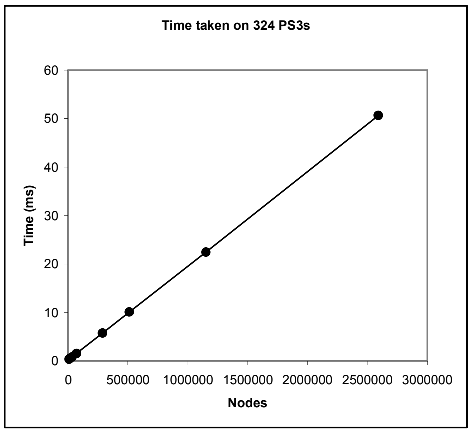


Figure 8.5. Runtimes on 324 PS3s for varying number of nodes

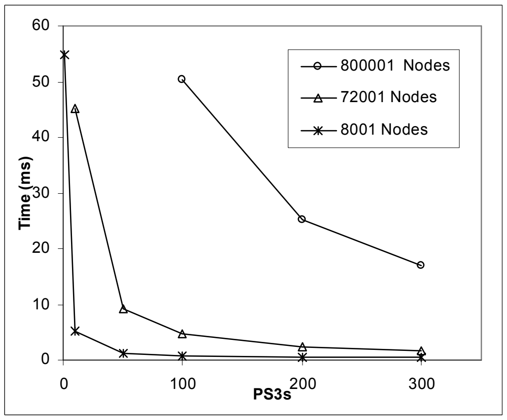


Figure 8.6. Time taken to implement constant size networks on varying number of PS3s

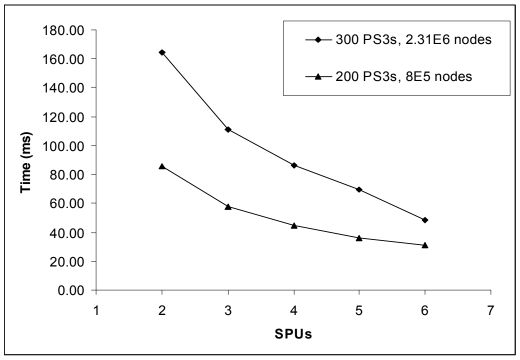


Figure 8.7. Time taken to implement constant size networks on number of PS3s, by varying the number of SPUs

Complexity Analysis

Figure 8.8 shows the change in runtime for networks with different number of layers with variations in the number of PS3s. All the networks had the same number nodes in their bottommost layer. An increase in the number of layers also increases the computations (due to more nodes being present), memory transfer, and MPI communication times.

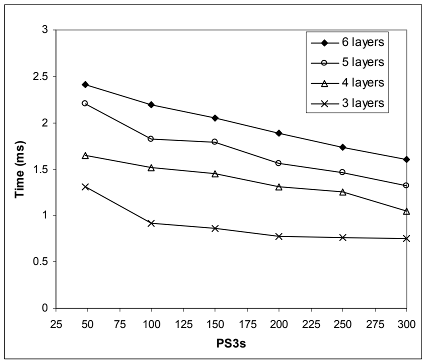


Figure 8.8. Runtimes for implementing networks with input size of 256 x 256, by varying number of layers.

Figures 8.9 and 8.10 show the change in runtime for different node complexities and variations in the number of PS3s. Only the configuration of the bottom most layer of nodes were varied. In Figure 8.9, the P_{xu} matrix dimensions of bottom most layer of nodes was varied and resulted in an almost linear increase in runtime for the networks. In the networks tested, 80% of the nodes were in the bottommost layer, and so changes to this layer affected the overall network runtimes significantly. Figure 8.10 shows the variation in runtime for changes to the bottommost layer node P_{xu} matrix densities (on 100 PS3s). Increasing P_{xu} matrix densities increased the overall computations and thus the runtimes.

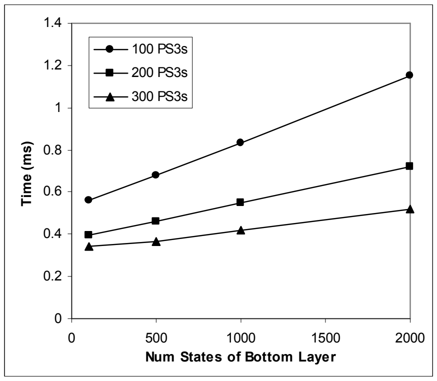


Figure 8.9. Runtimes for implementing nodes with increasing number of states.

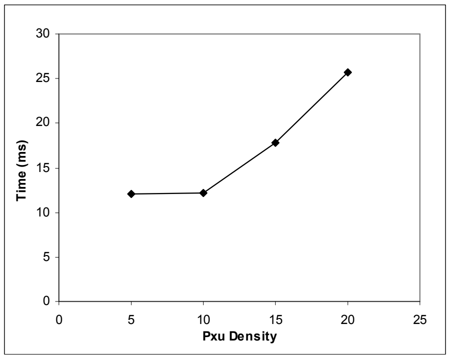


Figure 8.10. Runtimes for implementing a 200,001 node network on 100 PS3s with varying P_{xu} Density

CHAPTER NINE
CONCLUSION

There is a significant interest in the research community to develop large scale, high performance implementations of cortical models. These have the potential to provide significantly stronger information processing capabilities than current computing algorithms. Hierarchical Bayesian cortical models are a relatively new class of models that make it easier to develop larger scale simulations of the cortex than traditional neural networks. A collection of neurons forms a cortical column and are thought to be the basic units of computation in the brain. Since Hierarchical Bayesian cortical models are based on cortical columns as opposed to individual neurons, they have a significant computational advantage over the latter. Fewer nodes need to be modeled along with fewer node connections. Given that large scale cortical models can offer strong information processing capabilities, hierarchical Bayesian models are an attractive candidate for scaling. At present, multicore processors are the standard approach for achieving high performance. Thus a study of the parallelization of hierarchical Bayesian models and their implementations on multicore architectures is important.

This thesis examined the parallelization and implementation on multicore architectures of two hierarchical Bayesian models: the Hierarchical Temporal Memory model and the Dean's Hierarchical Bayesian model. Three multicore processors were examined for implementation: the IBM Cell processor, the Intel Xeon processor, and the Sun UltraSPARC T2+. Both the models and their relevant libraries were implemented in C, parallelized, and vectorized. This is the first study of the acceleration of this class of models on multicore architectures.

It has been shown that the hierarchical Bayesian cortical models can be parallelized onto multicore architectures to provide significant speedups over serial implementations of the models. The speedups come primarily from the use of multiple processing cores and vector operations. The speedups increase as the models are scaled and as the number of processing cores is increased. The highest performance gain was seen from the Cell processor, with speedups of 107 times for the Dean model and 93 times for the HTM model over a serial implementation on the Power Processor Unit of the Cell processor. The Dean model can be parallelized based on the subnets that it contains, or based on the cliques contained in the junction-trees that the subnets can be converted into. Our results indicate that the latter approach provides slightly higher speedups as there is more parallelism exposed. The vectorization of the two models and showed that it is easier to vectorize by processing multiple inputs simultaneously.

The cluster implementation shows that the 336 PS3 cluster provides a highly economical, yet powerful, platform for neuromorphic simulations. The system is capable of producing up to 50 TFlops. Five neuromorphic algorithms were scaled up on a cluster of 336 PS3s at the AFRL. Our results indicate that the models were fully scalable across the cluster. Additionally, three of the five models were scalable across the six SPUs available on each Cell processor in the cluster.

The largest HTM model that could be implemented had 2.4×10^{6} nodes (equivalent to 2×10^{10} neurons). Given that the human brain contains about 10^{11} neurons, this is a large number of components that the cluster was capable of modeling. As a simplistic comparison, an image recognition (for the largest image size tested) required about 55ms on the HTM model.

In a recent study [2], a 32,768 processor IBM BlueGene supercomputer was able to simulate a rat scale cortex (55×10^6 neurons and 4.42×10^{11} synapses) at near real time. This model did implement learning and was more biologically accurate than the models implemented in our study. However the cost of the BlueGene system is significantly higher (approximately 2-3 orders more) than the system utilized. The AFRL cluster cost \$337k, of which the PS3s cost about \$133k. Since a similar scale cortical system was modeled (although our model was much simpler) it indicates that a cluster of PS3s can be an economical platform for simulating large scale neuromorphic models.

It is important to note that the large cluster implementations are extremely simplistic. Future work should examine implementations should explore the applications of these models on large clusters. Given the capabilities of these machines as described in this thesis, the eventual goal has to be to develop networks with near human cognitive capabilities.

REFERENCES

[1] R. Ananthanarayan, D. Modha, “Anatomy of a cortical simulator”, Proceedings of ACM/IEEE conference on Supercomputing, 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] D.A. Bader, V. Agarwal, K. Madduri, and S. Kang, “High Performance Combinatorial Algorithm Design on the Cell Broadband Engine Processor,” *Parallel Computing*, *Elsevier,* 33(10-11):720-740, 2007.

[6] A. Buttari, J. Dongarra, and J. Kurzak, “Limitations of the Playstation 3 for High Performance Cluster Computing,” University of Tennessee Computer Science Technical Report, CS-07-594, May 2007.

[7] T. Dean, “A Computational Model of the Cerebral Cortex,” *Proceedings of the Twentieth National Conference on Artificial Intelligence* (AAAI-05): 938-943, 2005.

[8] T. Dean. “Scalable inference in hierarchical generative models,” Ninth International Symposium on Artificial Intelligence and Mathematics, 2006.

[9] T. Dean, “Learning invariant features using inertial priors,” *Annals of Mathematics and Artificial Intelligence*, 47(3-4), 223–250, Aug. 2006.

[10] T. Dean, G. Carroll, and R. Washington, “On the prospects for building a working model of the visual cortex,” *Proceedings the Twenty-Second Conference on Artificial Intelligence* (AAAI-07): 1597-1600, 2007.

[11] A. Delorme and S. J. Thorpe, “SpikeNET: an event-driven simulation package for modelling large networks of spiking neurons,” *Network-computation in neural systems*, 14(4), 613–627, Nov. 2003.

[12] M. Djurfeldt, M. Lundqvist, C. Johansson, M. Rehn, O. Ekeberg, and A. Lansner, “Brain-scale simulation of the neocortex on the IBM Blue Gene/L supercomputer,” *IBM Journal of Research and Development*, 52(1-2), 31–41, Jan.-Mar. 2008.

[13] A. Felch, J. Moorkanikara-Nageswaran, A. Chandrashekar, J. Furlong, N. Dutt, A. Nicolau, A. Veidenbaum, R. H. Granger. “Accelerating Brain Circuit Simulations of Object Recognition with a Sony PlayStation 3,” *Proceedings of the International Workshop on Innovative Architectures*, 2007.

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

[15] D. George, “How the brain might work: A hierarchical and temporal model for learning and recognition,” Doctoral Dissertation, Dept. of Electrical Engineering, Stanford University, 2008.

[16] M. 16, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, T. Yamazaki, “Synergistic Processing in Cell’s Multicore Architecture,” *IEEE Micro*, 26(2), 10–24, Mar. 2006.

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

[18] J. Hawkins, D. George, “Hierarchical Temporal Memory: Concepts, Theory and Terminology”, Numenta, http://www.numenta.com/Numenta_HTM_Concepts.pdf , 2006.

[19] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and application to conduction and excitation in nerve,” *Journal of Physiology*, 117, 500–544, 1952.

[20] E. Izhikevich and G. Edelman, “Large-Scale Model of Mammalian Thalamocortical Systems,” *Proceedings of the National Academy of Sciences*, 105(9), 3593–3598, Mar. 2008.

[21] C. Johansson, A. Lansner, “Towards Cortex Sized Artificial Neural Systems,” *Neural Networks*, 20(1): 48-61, 2007.

[22] S. Lauritzen, D. Spiegelhalter, “Local computations with probabilities on graphical structures and their application to expert systems,” *Journal of the Royal Statistical Society*, 50(2):157–194, 1988.

[23] 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.

[24] H. Markram, “The Blue Brain Project,” *Nature Reviews Neuroscience*, 7, 153–160, 2006.

[25] W. Maas, “Networks of spiking neurons: the third generation of neural network models,” *Transactions of the Society for Computer Simulation International*, 14(4), 1659–1671, Dec. 1997.

[26] Y. LeCun and C. Cortes, “The MNIST Database of handwritten images,” http://yann.lecun.com/exdb/mnist/.

[27] K. Murphy, “The Bayes Net Toolbox for Matlab.” *Computer Science and Statistics* 33(2): 1024-1034, 2001.

[28] V. Mountcastle, “Introduction to the special issue on computation in cortical columns,” *Cerebral Cortex*, 13(1):2–4, 2003.

[29] J. Pearl, “Probabilistic Reasoning in Intelligent Systems,” *Networks of Plausible Inference*, Morgan Kaufmann, San Francisco, CA, 1988.

[30] W. Rall, “Branching dendritic trees and motoneuron membrane resistivity,” *Experimental Neurology*, 1, 503–532, 1959.

[31] J. Rickman, “Roadrunner supercomputer puts research at a new scale,” Jun. 2008, http://www.lanl.gov/news/index.php/fuseaction/home.story/story_id/13602.

[32] P. Salin, J. Bullier, “Corticocortical connections in the visual system: structure and function”, *Physiological Reviews*, 75(1):107-54, 1995.

[33] Sun Microsystems, “UltraSPARC T2™ Supplement to the UltraSPARC Architecture 2007”, http://opensparc-t2.sunsource.net/specs/UST2-UASuppl-current-draft-HP-EXT.pdf, 2007.

[34] S. Vassiliadis, S. Cotofana, P. Stathis “Block based compression storage expected performance”, in the proceedings of HPCS2000, 2000.

[35] Q. Wu, P. Mukre, R. Linderman, T. Renz, D. Burns, M. Moore and Qinru Qiu, “Performance Optimization for Pattern Recognition using Associative Neural Memory,” Proceedings of the 2008 IEEE International Conference on Multimedia & Expo, June 2008.

[36] Y. Xia and V. Prasanna, “Junction Tree Decomposition for Parallel Exact Inference,” IEEE International Parallel & Distributed Processing Symposium (IPDPS'08), April 2008.

[37] Y. Xia and V. Prasanna, “Parallel Exact Infe rence on the Cell Broadband engine processor,” *Proceedings of the 2007 ACM/IEEE conference on Supercomputing,* 2008.

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

[39] S. Zhu and D. Hammerstrom, “Simulation of associative neural networks,” *Proceedings of the 9th International Conference on Neural Information Processing*, 1639- 1643, Nov. 2002.

[40] Richard Linderman, “Early experiences with algorithm optimizations on clusters of playstation 3’s,” *DoD HPCMP Users Group Conference*, Jul. 2008.