OPTIMUM MICROARCHITECTURES FOR NEUROMORPHIC ALGORITHMS

Thesis
Submitted to
The School of Engineering of the
UNIVERSITY OF DAYTON

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

By
Shu Wang
Dayton, Ohio
December, 2011
OPTIMUM MICROARCHITECTURES FOR NEUROMORPHIC ALGORITHMS

Name: Shu, Wang

APPROVED BY:

Tarek M. Taha, Ph.D.
Advisory Committee Chairman
Department of Electrical and Computer Engineering

Eric J. Balster, Ph.D.
Committee Member
Department of Electrical and Computer Engineering

Vijayan K. Asari, Ph.D.
Committee Member
Department of Electrical and Computer Engineering

John G. Weber, Ph.D.
Associate Dean
School of Engineering

Tony E. Saliba, Ph.D.
Dean, School of Engineering
& Wilke Distinguished Professor
ABSTRACT

OPTIMUM MICROARCHITECTURES FOR NEUROMORPHIC ALGORITHMS

Name: Shu, Wang
University of Dayton

Advisor: Dr. Tarek M. Taha

At present there is a strong interest in the research community to develop large scale implementations of neuromorphic algorithms. These systems consume significant amounts of power, area, and are very expensive to build. This thesis examines the design space of multicore processors for accelerating neuromorphic algorithms. A new multicore chip will enable more efficient design of large scale neuromorphic computing systems. The algorithms examined in this thesis are the HMAX and Izhikevich models. HMAX was developed recently at MIT to model the visual system of the human brain. The Izhikevich model was presented by Izhikevich as a biologically accurate spiking neuron model. This thesis also examines the parallelization of the HMAX model for studying multicore architectures. The results show the best single core architectures for HMAX and Izhikevich are almost same, though HMAX needs more cache. The multicore study shows that the off chip memory bus width and physical memory latency could improve the performance of the multicore system.
ACKNOWLEDGMENTS

My special thanks are in order to Dr. Tarek M. Taha, my advisor, for providing the time and equipment necessary for the work contained herein, and for directing this thesis and bringing it to its conclusion with patience and expertise.


**TABLE OF CONTENTS**

ABSTRACT ................................................................................................................................. iv
ACKNOWLEDGMENTS ............................................................................................................... v
LIST OF FIGURES .................................................................................................................... vii
LIST OF TABLES ...................................................................................................................... ix

CHAPITERS
I. INTRODUCTION .................................................................................................................. 1
II. RELATED WORK ................................................................................................................... 3
III. ALGORITHM ....................................................................................................................... 5
IV. PARALLELIZATION OF HMAX ........................................................................................ 11
V. SINGLE CORE ARCHITECTURE STUDY ........................................................................... 12
VI. MULTICORE ARCHITECTURE STUDY ............................................................................ 24
VII. CONCLUSION ................................................................................................................ 25

REFERENCES .......................................................................................................................... 31
LIST OF FIGURES

Figure 1  Network used to test Izhikevich algorithm ..................................................6
Figure 2  Image recognition with HMAX ......................................................................8
Figure 3  The process of HMAX before we parallelize the algorithm .........................12
Figure 4  The process of HMAX before we parallelized the algorithm .......................13
Figure 5  HMAX performance with different numbers of threads ..............................14
Figure 6  Izhikevich performance with different numbers of threads .......................14
Figure 7  Superscalar width variation .........................................................................17
Figure 8  BTB entries variation ..................................................................................18
Figure 9  ROB entries variation ..................................................................................19
Figure 10 Branch predictor variation with different global history table
          (Izhikevich) ........................................................................................................19
Figure 11 Branch predictor variation with different global history table
          (HMAX) ...........................................................................................................20
Figure 12 L1 instruction cache variation .....................................................................21
Figure 13 L1 data cache variation ...............................................................................21
Figure 14 L2 cache variation .....................................................................................22
Figure 15  Bus width variation (Izhikevich) .................................................................................. 24
Figure 16  Memory latency variation (Izhikevich) ...................................................................... 25
Figure 17  Performance degradation (Izhikevich) ...................................................................... 26
Figure 18  Core utilization (Izhikevich) ....................................................................................... 26
Figure 19  Memory latency variation (HMAX) ........................................................................... 27
Figure 20  Performance degradation (HMAX) ............................................................................ 27
Figure 21  Core utilization (HMAX) ............................................................................................ 28
LIST OF TABLES

Table 1  Architecture parameters.................................................................17
Table 2  Final single core design (Izhikevich model)........................................23
Table 3  Final single core design (HMAX model)................................................23
CHAPTER I.
INTRODUCTION

Current computing systems excel at a variety of applications such as scientific simulations. However, they perform poorly at cognitive tasks such as learning, vision, and language. Humans excel at these tasks. This is because the brain works on a different approach compared to traditional computing systems. The brain utilizes a large collection of neurons (about $10^{10}$) connected through a dense array of synapses (about $10^{14}$). There is a strong push currently to design computing systems that perform computations similar to the human brain.

Cognitive models can be used with both high performance computer clusters or embedded single chips. There are many real applications used with these two platforms. At present several research groups [1][2] are developing brain scale implementations of neuromorphic algorithms. Given that the brain has a large collection of neurons, all operating in parallel, it is necessary for these large scale implementations to exploit hardware level parallelism. Thus several research groups [3][4][2][5] are developing specialized architectures for neuromorphic algorithms.

This thesis examines the design of single-chip computing systems that are efficient at running these new brain inspired compute algorithms. Multicore
architectures based on this single core design are then evaluated. This thesis examines architectures for two new brain inspired compute algorithms: the Izhikevich [4] and the HMAX [1] models. Our studies have shown that the HMAX model is more compute intensive.

The processing architectures for these two algorithms were evaluated using the gem5 [6] processor simulation software on a high performance compute cluster. For the HMAX algorithm, we utilized the MATLAB source code provided by MIT [7]. This code was hand converted to C and was then parallelized to use POSIX threads for the architecture study. A parallelized version of the Izhikevich code was provided by [2].

Chapter II of this thesis discusses related work including three new architectures that are used for spiking neuron models. Chapter III examines details of the Izhikevich and HMAX algorithms. Chapter IV presents the process of the parallelization of the HMAX algorithm and the result of parallel program performance. Chapter V introduces the simulation framework and testing platform along with its use to examine optimum single core processors for these algorithms. Chapter VI examines the design of optimum multicore architectures for the algorithms, while chapter VII concludes the thesis.
CHAPTER II.

RELATED WORK

At present most research in implementing brain inspired algorithms are focused on using large computing systems [3][8][9][10]. These systems consume significant amounts of power, area, and are very expensive to build. IBM is utilizing a 147,456 processor Blue Gene/P system to simulate a spiking network based model [8], while EPFL and IBM are utilizing an 8,192 processor Blue Gene/L system to simulate a sub-neuron based cortical model [3]. The PetaVision project at the Los Alamos National Laboratory is utilizing the Roadrunner supercomputer to model the human visual cortex [9]. Nageswaran [10] considered the implementation of spiking networks on a cluster of GPGPUs.

At the chip level, Rice et al [11] examined the use of FPGAs to model brain simulation algorithms. Although such systems are single chip, they lack the density and performance a specially designed single chip system might be able to provide. New efforts at developing memristor based single chip systems are underway [12]. These systems are however more long term solutions, as memristor devices themselves are quite novel. Current investigations are examining developing circuits for them and for integrating them into existing CMOS circuits (used in the design of existing computing
systems). SpiNNaker used two ARM cores, each clocked at 200MHz, to build parallel spiking neural network architectures successfully [4]. They are planning to build a bigger network, with 50,000-60,000 SpiNNaker chips, to achieve the goal of simulating 1% of the human brain [4]. The FACETS project successfully built a single wafer that contains up to 200,000 neuronal circuits, 50,000,000 plastic synapses, and includes a freely configurable communication fabric [13]. Recently MIT [5] presented a new computer chip which mimics how the brain accepts new information. This would help to understand activities within the brain.

Interest in the area of modeling neural algorithms is rapidly increasing in the computing research community, and we believe that examining the design of specialized processors using only existing technologies is important. Using existing technologies has the highest potential of generating a successful product, while single chip solutions (as opposed to large compute clusters) can be low cost and utilized in a variety of products.
CHAPTER III.

ALGORITHM

At present several research groups are developing algorithms inspired by the neural connections in the brain and the activity of each neuron. The models examined in this study are the Izhikevich and HMAX models.

Izhikevich:

The Izhikevich model is a biologically accurate spiking neuron model. This model, proposed in 2003, is based on two differential equations (eq. 1 and 2).

\[
\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \quad (1)
\]

\[
\frac{du}{dt} = a(bv - u) \quad (2)
\]

If \( V \geq 30 \text{mV} \), then \( \begin{cases} V \leftarrow c \\ u \leftarrow u + d \end{cases} \)

The network utilized for this study consisted of two layers (see Figure 1), where the first layer is an input set of neurons and the second layer is an output set of neurons. The first layer of neurons utilizes pixels from an input image. Each input image pixel is fed to one neuron. The number of input neurons is equal to the number of pixels input images. The number of output neurons is equal to the number of training images. Each input neuron is connected to all the output neurons.
There is an input current in each neuron that is used to estimate the membrane potential. If the membrane potential is higher than a certain threshold in one cycle, we consider the neuron as fired. If the pixel in the input images is “off”, the input current of that pixel’s corresponding level one neuron is zero. If the pixel is “on”, the input current should be supplied. Each level two neuron’s total input current is the sum of all the currents that are received from each level one neuron. The level two neuron’s input current is described by:

\[ I_j = \sum_i w(i,j)f(j) \]

where \( w(i,j) \) is the input weight from level one neuron \( i \) to level two neuron \( j \), \( f \) is a vector where \( f(i) \) is zero if level one neuron \( i \) has not fired, and is one if the neuron has fired. Algorithm 1 shows the steps in processing this model.

Algorithm 1: The testing phase of the spiking neuron image recognition model.
**HMAX:**

The HMAX algorithm [1] was developed at MIT to model the human visual system. This algorithm can be applied to a variety of other areas. This algorithm is representative of other such algorithms developed recently [14][15]. A study using HMAX will likely apply to other such algorithms. Figure 2 shows the process of HMAX.

The following sections describe the layers shown in Figure 2.

Input layer: We read and convert bmp files to grayscale. Then we resize the image to different scales. Each scale is $2^{1/4}$ smaller than the last one. We can create an image pyramid of 11 scales. The largest scale's size is 256x256 pixels.
S1 Layer (Gabor filter): To center Gabor filters with 4 orientations over each scale and pixel position, the S1 layer is computed from the input layer. The S1 layer is a 4D structure of pixels and orientations. It is based on the C1 3D pyramid shape, with 4
orientations at each position and scale. The Gabor filters’ sizes are 11x11, and are given by equation below:

\[ G(x, y) = \exp\left(-\frac{(x^2 + y^2)}{2\sigma^2}\right) \cos\left(\frac{2\pi x}{\lambda}\right) \]  

(3)

where \(X = x\cos\theta - y\sin\theta\) and \(Y = x\sin\theta + y\cos\theta\). The range of \(X\) and \(Y\) is between -5 to 5, and the range of \(\theta\) is between 0 to \(\pi\). We set parameters \(\gamma\) (aspect ratio), \(\sigma\) (effective width), and \(\lambda\) (wavelength) to 0.3, 4.5, and 5.6. In theory, the long axis of Gabor filters should decrease to zero before the boundary of the Gabor filter is reached. In fact, experiments indicate that the classification performance won’t be improved by increasing the size of arrays; and larger arrays require more cost to compute. The response of a patch of pixels \(X\) to a special Gabor filter \(G\) is described by:

\[ R(X, G) = \left| \frac{\sum x_i g_i}{\sum x_i^2} \right| \]

(4)

C1 Layer (Max filter): The C1 layer is a 3D pyramid, similar to the S1 layer. This layer is sampled from S1 to reduce the numbers of units and have the same orientation as S1. The S1 pyramid is convolved with the max filter at each orientation. A C1 unit’s value is the maximum value of S1 units which falls in the max filter. The max filter is moved around the S1 pyramid to perform the subsampling.

S2 Layer (GRB filter): In the S2 layer, we compare the patch of C1 units centered at every position and scale in the C1 layer with each prototype patch. We randomly
sample the prototypes from the C1 layers of the training data in an initial feature learning period.

In the initial feature learning period, the sampling process is to center a patch of 4x4, 8x8, 12x12, or 16x16 at a random position and scale in the C1 layer of a random training data. All the values of the C1 units in the patch are read out and stored as a prototype. For example, if the patch size is 8x8, it means 64 different positions. However, there are 4 orientations over each position. Thus there are 256 (8x8x4) C1 unit values in an 8x8 patch. The response of a path of C1 units X to a S2 prototype is described by:

\[ R(X, P) = \exp\left(-\frac{\|X - P\|^2}{2\sigma^2\alpha}\right) \]

P is an \( n \times n \) array with \( n \in \{4,8,12,16\} \). Both \( X \) and \( P \) have dimensions of \( n \times n \times 4 \). The standard deviation \( \sigma \) is set to 1. The parameter \( \alpha \) is a normalizing factor that affects the weight of the dimensions. If the patches size is large (\( n \in \{8,12,16\} \)), we will compute distances in a high dimensional space. Smaller distances will cause dimensions that have to match. We use \( \alpha \) to reduce the weight of these additional dimensions. As in [1], the parameter \( \alpha = (n/4)^2 \) is the ratio of the dimension of \( P \) to the dimension of the smallest path size.
CHAPTER IV.
PARALLELIZATION OF HMAX

HMAX is currently available as MATLAB code [7]. This cannot be utilized with the gem5 multicore simulator utilized in this study. Thus we developed a C implementation of the algorithm and then parallelized the C implementation to run on multiple processors (the Izhikevich model was already parallelized in [2]). Figure 3 shows a flowchart of the functions within HMAX.

The original HMAX MATLAB code read and converted an input image to grayscale and created an image pyramid. We converted the MATLAB code to a C code which had the same functions. In our new code, we could read any picture. The new program converted all the images at once and stored the new image scales in separate arrays. There were several functions that utilized both C and MATLAB code together. These functions couldn’t be utilized with gem5 in their existing form. Therefore, C implementations of the functions were carefully developed.

Figure 2 illustrates that the S1, C1, and S2 layers are 3D pyramidal shape. Since these three layers’ sizes are much larger than the C2 layer, we just parallelized the three larger layers. Figure 4 shows the flow chart of the parallel HMAX algorithm. In the
parallel layers, each layer is parallelized into n parts, and then each part is computed by one thread. The parallel computing layer computes these components at the same time.

Figure 3. The process of HMAX before we parallelize the algorithm.
We separated the compute layer into $n$ threads and combined the $n$ outputs together after computing. This data was then sent to the next compute layer and computed in parallel again.
In theory, when running a parallel program with more threads, the run time should be shorter. However, experiments showed HMAX’s performance did not improve beyond 8 threads and would get more expensive to compute. The parallelized results can be seen in figure 5.

Figure 5. HMAX performance with different numbers of threads.

Figure 6. Izhikevich performance with different numbers of threads.
Figures 5 and 6 show that the parallelization worked. From 1 thread to 8 threads the runtime of HAMX simulation reduced about 82%. The runtime of the Izhikevich simulation reduced about 85%. The run time declined about 50% when the number of threads doubled.
CHAPTER V.

SINGLE CORE ARCHITECTURE STUDY

In this study we limited ourselves to develop a single core architecture that can efficiently run the Izhikevich and HMAX algorithms. We tried to find out the simplest single core architecture for the two algorithms. We utilized the gem5 simulation software and the parallel compute cluster to examine these two models.

Gem5 is a modular platform for computer system architecture research, consisting of the best part the M5 [16] and the best part of the GEMS [17] architecture simulators. For processor architecture simulation, gem5 provides a new detailed CPU model, O3CPU, which is an out of order CPU model loosely based on the Alpha 21264. We used O3CPU to find the best single core architecture for the Izhikevich and HMAX models. Gem5 provides two different memory system models, Classic and Ruby, that are used to test the cache and the topology of interconnection networks.

The IDCAST HPC Cluster is one of the few HPC clusters with FPGA, GPGPU, and Intel processing cores all on the same mother board. This cluster contains four Intel Intel Xeon 5650 chips (each consisting of six cores). Each core can run up to two threads, thus allowing 48 threads to run in parallel. The cluster also has 384GB DDR3 ECC memory, 2TB solid state drives that connect to 200TB capable storage cluster, four NVIDIA Tesla
GPUs, and two Gidel PROCStar IV card with 8 Altera 530L FPGA’s. This cluster enabled us to run several simulations simultaneously.

In this study we used the O3CPU to examine the Izhikevich and HMAX models with different options. The options, we examined are shown in table 1.

Table 1. Architecture parameters.

<table>
<thead>
<tr>
<th>Options</th>
<th>Test value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar width</td>
<td>1, 2, 3, 4, 8</td>
</tr>
<tr>
<td>Number of BTB entries</td>
<td>32, 64, 128, 256, 512, 1024</td>
</tr>
<tr>
<td>Number of ROB entries</td>
<td>12, 24, 48, 96, 192, 384</td>
</tr>
<tr>
<td>Global predictor and choice predictor</td>
<td>16, 32, 64, 128, 256, 512, 1024, 2048</td>
</tr>
<tr>
<td>Function Units</td>
<td>Int_ALU, Int_MultiDiv, FP_ALU, FP_MultiDiv, RdWrPort, lprPort</td>
</tr>
<tr>
<td>L1d_cahce and l1i_cache</td>
<td>1kB, 8kB, 16B, 32kB</td>
</tr>
<tr>
<td>L2 cache</td>
<td>8kB, 16kB, 64kB, 128kB, 256kB, 1024kB</td>
</tr>
</tbody>
</table>

![Figure 7. Superscalar width variation.](image)

A superscalar CPU can execute more than one instruction in one CPU cycle. So the superscalar width described the number of instructions executed in one CPU cycle.

Figure 7 is the result of the superscalar width variation on run time. The run time declines while the superscalar width increases. This indicates that increasing the
superscalar width will improve the performance. However, another question is that processors with larger superscalar width are more expensive than smaller ones. Figure 7 illustrates that when increasing the superscalar width from 2 to 4, the run time reduced about 29% for the HMAX algorithm and 16% for the Izhikevich algorithm. A processor with a superscalar width of four will cost much more than a processor with a superscalar width of two. Therefore a superscalar width of 2 was selected as the best choice.

![Figure 8. BTB entries variation.](image)

The BTB (branch target buffer) can dynamically predict branch instruction target addresses, thereby reducing the cost of the pipeline. Figure 8 illustrates that the BTB entries fail to influence the performance of these two algorithms. Thus in this thesis, we chose a general minimum number of 256 BTB entries.
The ROB (reorder buffer) is used in out of order CPU architecture to make instructions to be in-order after they are executed. Figure 9 shows that increasing the ROB entries beyond 48 fail to improve the performance of these two algorithms.

Figure 9. ROB entries variation.

Figure 10. Branch predictor variation with different global history table (Izhikevich).
Figure 11. Branch predictor variation with different global history table (HMAX).

Branch predictors are used in microprocessors to guess ahead of a branch instruction’s execution what direction the branch will go. The purpose of branch predictors is to improve the performance of the instruction pipeline. Figures 10 and 11 are the results of varying the global and choice predictor size on runtime. We examined these options with different global history size. If the history size is n, the predictor size should be the $n^{th}$ power of 2. These two figures indicate that the branch predictor size did not significantly impact performance. Therefore we chose the minimum global history size of 6. The predictor size would then be the $6^{th}$ power of 2 (that is 64 entries).
For the Izhikevich model, changes in the L1 instruction cache size failed to improve the algorithm’s performance. We can use a very small L1 instruction cache with one core. For the HMAX model, a small L1 instruction cache reduced the algorithm’s performance. Both Izhikevich and HMAX ran faster with larger L1 data caches. Therefore we chose 32kB to be the best L1 data cache size.
We examined a number of parameters for different function units (see Table 2). We found that one Int_MultiDiv and one FP_ALU leads to minimum cost and optimum performance.

![Figure 14. L2 cache variation.](image)

Figure 14 illustrates that the L2 cache size didn’t improve the performance of the Izhikevich model. However, a small L2 cache size could not fulfill the high performance needs of the HMAX model.

Based on these parameters, a very simple architecture that had high performance for the Izhikevich and HMAX algorithms is shown in Tables 2 and 3. The best single core architectures for the Izhikevich and HMAX models are almost the same. However, the HMAX algorithm needed more cache.
Table 2. Final single core design (Izhikevich model).

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar width</td>
<td>2</td>
<td>Number of BTB entries</td>
<td>256</td>
</tr>
<tr>
<td>Number of ROB entries</td>
<td>48</td>
<td>Global predictor and Choice predictor</td>
<td>64</td>
</tr>
<tr>
<td>Global history table</td>
<td>6</td>
<td>L1i_cache</td>
<td>1kB</td>
</tr>
<tr>
<td>L1d_cache</td>
<td>32kB</td>
<td>L2 cache</td>
<td>8kB</td>
</tr>
<tr>
<td>Function units</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int_ALU</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int_MultiDiv</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP_ALU</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP_MultiDiv</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RdWr Port</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ipr Port</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Final single core design (HMAX model).

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar width</td>
<td>2</td>
<td>Number of BTB entries</td>
<td>256</td>
</tr>
<tr>
<td>Number of ROB entries</td>
<td>48</td>
<td>Global predictor and Choice predictor</td>
<td>64</td>
</tr>
<tr>
<td>Global history table</td>
<td>6</td>
<td>L1i_cache</td>
<td>8kB</td>
</tr>
<tr>
<td>L1d_cache</td>
<td>32kB</td>
<td>L2 cache</td>
<td>64kB</td>
</tr>
<tr>
<td>Function units</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int_ALU</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int_MultiDiv</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP_ALU</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP_MultiDiv</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RdWr Port</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ipr Port</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER VI.
MULTICORE ARCHITECTURE STUDY

In this study we limited ourselves to develop a multicore architecture that can efficiently run the Izhikevich and HMAX algorithms. We believe that when the number of processing cores is very large, the off-chip memory bandwidth will have a larger influence on the efficiency of Izhikevich/HMAX algorithm implementation. Therefore we examined two options, off-chip bus width and physical memory latency, with different number of cores.

Figure 15. Bus width variation (Izhikevich).
Figure 15 shows that when the bus width increases, the total IPC will increase also. However, the improvement will be minimal, when the bus width crosses a certain threshold. We believe this is due to the cache block size not being not large enough compared to the off-chip memory bus width. Thus the performance is affected by the cache block size too.

![Figure 16. Memory latency variation (Izhikevich).](image)

Figure 16 illustrates the impact on IPC with variations in memory latency and number of cores. The total IPC declines with an increase of memory latency. This indicates that the memory latency will influence the performance of multicore architectures. The total IPC of a system with a 200ns memory latency and 16 cores is almost same as the total IPC of a system with a 40ns memory latency and 8 cores. This indicates that the memory latency is a very important option for multicore architectures.
Figure 17. Performance degradation (Izhikevich).

Figure 17 shows the performance degradation of the system (compared to a memory latency of 10ns) with increase in the physical memory latency. We found that the performance degradation will increase when the memory latency increases.

Figure 18. Core utilization (Izhikevich).

Figure 18 shows the utilization of each processing core with variation in memory latency and number of cores. The core utilization declines with increasing memory latency.
Figure 19. Memory latency variation (HMAX).

Figure 19 shows the impact on IPC with varying the memory latency and number of cores for the HMAX algorithm. Compared to the results seen for the Izhikevich model (Figure 16), the HMAX model does not get impacted significantly by changes in the memory latency. Changes in the number of cores does impact performance.

Figure 20. Performance degradation (HMAX).

Figure 20 shows the performance degradation with variation in the memory latency. We found the degradation will increase with higher memory latencies.
Figure 21 illustrates the utilization of each core with various memory latencies. These results are much better than for the Izhikevich model. The core utilization will decline with the increases in the number of cores. When examining same instructions, the core utilization will also decline with the increase of memory latency.
CHAPTER VII.

CONCLUSION

There is currently a strong interest in the research community to develop brain scale implementations of neuromorphic algorithms. Given that the brain has a large collection of neurons, all operating in parallel, it is necessary for these large scale implementations to exploit hardware level parallelism.

This thesis examined the design of single chip processing cores for these algorithms. Two neuromorphic algorithms, the Izhikevich and the HMAX models, were examined in this thesis. We developed a C implementation of the HMAX algorithm from a MATLAB source code provided by MIT. For the single core study, we limited ourselves to develop a single core architecture that would enable a more efficient design for these two algorithms. The results illustrate that large superscalar widths provide diminishing improvements on the performance. Changes in the BTB entries and ROB entries fail to impact the performance. With different global history table sizes, neither global predictor nor choice predictor improves the performance significantly. By testing a large number of function unit groups, we found the best combination of function units. The results indicate that the best basic architecture for Izhikevich and HMAX models are almost same, although the HMAX model needs a larger cache.
For the multicore study, we limited ourselves to find out the relationship between the physical memory’s bandwidth and the number of cores. The performance of the multicore system is improved by a larger off chip memory bus width. On the other hand, if the latency of the physical memory is lowered, the performance of multicore system improved also.

As future work, I plan to test the two algorithms with more core options. My final object is to examine the Izhikevich and HMAX with a mass of processors on one chip. Additionally, I plan to examine architectural requirements of other brain inspired compute algorithms on the single core and multicore systems.
REFERENCES


[7] HMAX algorithm
http://riesenhuberlab.neuro.georgetown.edu/hmax.html


[13] The FACETS Project
http://facets.kip.uni-heidelberg.de/index.html


