Abstract: Neuromorphic vision algorithms are biologically inspired models that follow the processing that takes place in the primate visual cortex. Despite their efficiency and robustness, the complexity of these algorithms results in reduced performance when executed on general purpose processors. This paper proposes an application-specific system for accelerating a neuromorphic vision system for object recognition. The system is based on HMAX, a biologically-inspired model of the visual cortex. The neuromorphic accelerators are validated on a multi-FPGA system. Results show that the neuromorphic accelerators are 13.8X (2.6X) more power efficient when compared to CPU (GPU) implementation.
Hardware Acceleration for Neuromorphic Vision Algorithms

Ahmed Al Maashri*, Matthew Cotter*, Nandhini Chandramoorthy*, Michael DeBole†, Chi-Li Yu‡, Vijaykrishnan Narayanan*, Chaitali Chakrabarti‡

*Microsystems Design Laboratory, The Pennsylvania State University
{maashri, mjcotter, nic5090, vijay}@cse.psu.edu
†IBM System and Technology Group
mvdebole@us.ibm.com
‡School of Electrical, Computer and Energy Engineering, Arizona State University
{chi-li.yu, chaitali}@asu.edu

Abstract

Neuromorphic vision algorithms are biologically inspired models that follow the processing that takes place in the primate visual cortex. Despite their efficiency and robustness, the complexity of these algorithms results in reduced performance when executed on general purpose processors. This paper proposes an application-specific system for accelerating a neuromorphic vision system for object recognition. The system is based on HMAX, a biologically-inspired model of the visual cortex. The neuromorphic accelerators are validated on a multi-FPGA system. Results show that the neuromorphic accelerators are 13.8X (2.6X) more power efficient when compared to CPU (GPU) implementation.

Keywords

Domain-Specific Acceleration, Power Efficiency, Neuromorphic Systems

1. Introduction

In the last few years, the world has witnessed a wide range employment of machine learning and artificial perception algorithms into many applications such as autonomous cars [1], surveillance and security [2], and space exploration [3]. While research continues in enhancing these algorithms, however, the visual processing of primates – in general – and humans – in particular– still outperforms these algorithms in terms of efficiency and capability. For example,
the human brain, a massively parallel processor, contains 100 billion neurons operating in parallel to deliver $10^{16}$ FLOPS, while consuming 20 Watts. While still an active research area, neuroscientists have proposed computational models that are believed to represent the processing that takes place in the mammal’s visual cortex. One such model is HMAX [4]—a feed forward hierarchical model that mimics the processing in the ventral path. While similar in structure to Convolutional Neural Networks (CNNs) [5] and other Artificial Neural Networks (ANN), HMAX originates from the current understanding of the mammalian ventral visual stream, a hierarchy of brain areas responsible for carrying out object recognition. In fact, HMAX has been shown to correctly predict the output of similar units in experimentally captured read-outs from the monkey IT cortex [4]. From a computer vision perspective, HMAX is a scale-, orientation-, and shift-invariant feature extractor used for object recognition, with classification accuracy that competes with state-of-the-art machine vision algorithms [6,7].

Previous attempts to implement HMAX on CPUs and GPUs have shown that these platforms may not be the most suitable platforms, due to power and computational inefficiencies [6,8]. On the other hand, the use of rigid domain-specific accelerators may hinder the flexibility needed to allow variation of the model’s parameters. This paper proposes a highly-configurable, accelerated neuromorphic system that combines both speed and power-efficiency, while maintaining the accuracy provided by the HMAX model. Towards this end, we experiment with the HMAX model on a multi-core CPU to investigate the computational complexity, and potential for exploitation of both data- and thread-level parallelism. The results from these profiling experiments help in determining the model’s hotspot(s), the data flow, memory access patterns and requirements, and allocation of parallel resources.

Next, we propose an accelerated neuromorphic system that exploits parallel resources to speed up the algorithm, while maintaining a relatively low power profile to improve the overall power efficiency of the system. Additionally, since there may be a number of variations to the HMAX model, the proposed neuromorphic system supports runtime re-configurability.

To evaluate the performance of these neuromorphic accelerators, a heterogeneous system composed of a CMP that is loosely coupled to a Multi-FPGA system was used as an emulation platform. The performance of the emulation platform was compared to a 16-core CPU and an Nvidia Tesla GPGPU. Results indicate that the proposed neuromorphic system is 13.8X (2.6X) more power efficient than the CPU (GPU) platform. Additionally, we tested the accuracy of the accelerated HMAX using two well-known datasets; namely, Caltech256 and PASCAL VOC2007.

The rest of this paper is organized as follows; Section 2 introduces the HMAX computational model and discusses the complexity of the algorithm. Section 3 proposes the neuromorphic accelerators that are implemented to speed up the HMAX model, followed by experimental evaluation of these accelerators in Section 4. Section 5 presents Related Work. Finally, Section 6 concludes the paper.

## 2. HMAX Computational Model

### 2.1 Model Description
HMAX is a model of the ventral visual pathway from the visual cortex to the inferotemporal cortex, IT [4,6,7]. This model attempts to provide space- and scale-invariant feature extraction by building complex features from a set of simple features in a hierarchical fashion. Figure 1 shows a computational template of HMAX. The model consists of a preprocessing stage, $S_0$, followed by two distinct types of computations, convolution and pooling (i.e. non-linear subsampling). The convolution and pooling stages correspond to the simple, $S$, and complex, $C$, cell types found in the visual cortex, respectively. Following the orientation-tuned $S_1$ layer, processing proceeds through alternating layers of complex pooling and simple template-matching. A number of HMAX implementations exist; where each implementation introduces variations in the layers of the model. This work uses an implementation derived from the extension developed by Mutch & Lowe [7]. This extension and its implementation [9] represent the most up-to-date representation of the HMAX model. The following text describes the details of each layer in the model.

$S_0$ (Preprocessing) layer: This layer is used for preprocessing the input image to ensure the uniformity of inputs and provide scale invariance. First, the image is converted to grayscale and then pixel values are normalized to the range [0, 1]. A 12-scale pyramid is created by downscaling the input image to sizes of 256×256 and smaller. The interpolation method can vary, however no noticeable improvement was gained using more complex techniques (e.g. bicubic) over simpler ones (e.g. nearest-neighbor) that are more favorable for hardware implementations.

$S_1$ (Gabor filter) layer: The $S_1$ layer, corresponding to the V1 simple cells, is based on an accepted model of the simple receptive fields of the cortex, Gabor filters [6]. These filters produce outputs for each scale at all desired orientations. The Gabor filters are 11×11 in size and are described by equation (1):

$$G(x, y) = \exp \left(-\frac{(X^2 + Y^2)}{2\sigma^2}\right) \times \cos\left(\frac{2\pi}{\lambda} \cdot x\right)$$

where $X = x\cos(\theta) + y\sin(\theta)$ and $Y = -x\sin(\theta) + y\cos(\theta)$. The model follows [6] and varies $x$ and $y$ between -5 and 5, and $\theta$ between 0 and $\pi$, while the wavelength ($\lambda$), width ($\sigma$), and aspect ratio ($\gamma$) are 5.6, 4.6, and 0.3, respectively.

$C_1$ (local invariance) layer: The $C_1$ layer – modeled after the complex cells of the V1 – pools over the outputs of the $S_1$ layer. This maximum-value pooling over local windows of adjacent scales provides both local scale-invariance and reduces the processing units required in subsequent layers by sampling only a subset of the $S_1$ output.

$S_2$ (Tuned features) layer: The $S_2$ layer models V4 or posterior IT by matching a set of randomly sampled 4×4×m, 8×8×m, 12×12×m, and 16×16×m prototypes. The value of $m$ represents the number of orientations extracted from the image in the $S_1$ layer. These prototypes make up a dictionary of $k$ patches used as fuzzy templates for simple
position-and scale-invariant features. \( S_2 \) then computes the response of a \( C_1 \) layer output patch, \( X \), to a particular \( S_2 \) feature prototype, \( P \), of size \( n \times n \times m \) (typical \( n = \{4,8,12,16\} \)). The number of patches, \( k \), is determined through a learning phase, which randomly selects feature prototypes of varying sizes from a set of images that represent the categorization task. If a general model is desired, the training set should contain images not related to any specific categorization task. This work uses a 5120-prototype dictionary. The \( S_2 \) layer computes the final response using Normalized Dot Product, NDP, as shown in equation (2).

\[
R(X,P) = \frac{X \cdot P}{\sqrt{\sum x_i^2 - (\sum x_i)^2/n^2 \cdot m}}
\]

As shown in the numerator of equation (2), the \( S_2 \) layer accumulates the responses from all orientations within each scale. The normalization stage, as shown in the denominator of equation (2), is performed by computing the normalization patch, which is computed from a windowed average of the current scale. The variable \( x_i \) denotes a single pixel in the \( C_1 \) layer output patch. Then, pixel-wise division is performed by dividing the accumulated responses by the normalization patch.

**\( C_2 \) (Global invariance) layer:** The final layer provides global invariance by finding the per-prototype global maxima over all scales and positions, thus removing all position and scale information. The resulting complex feature set can then be used for classification to perform final object recognition.

### 2.2 The Computational Complexity of HMAX

The computational structure of the HMAX model exhibits a number of inefficiencies when executed on a general purpose processor; resulting in degradation of performance. This subsection presents a detailed empirical study of the HMAX algorithm’s performance when subjected to certain variations, on varying CPU configurations. This study helps in making adequate design decisions pertaining to the architecture of the hardware accelerators.

In order to study the performance of the HMAX model, we use a specific implementation of HMAX [9] that was further extended to allow thread-level parallelism. The extended version was executed on an Intel Xeon-based system, consisting of two 2.4 GHz quad-core processors, with 12 GB system memory. Intel’s HyperThreading was enabled on all cores, providing a total of sixteen logical processors. Furthermore, the HMAX implementation made use of the SSE2 instruction set extension to benefit from the Single Instruction Multiple Data, SIMD, resources available on the cores.

Figure 2 shows the overall execution time of the model under two variations related to the number of orientations, while executing at different levels of parallelism (i.e. 1 to 16 threads). The figure shows that HMAX with 12 orientations is 2.5X to 2.8X slower when compared to HMAX with 4 orientations. This observation is expected since increasing the number of orientations increases the computational complexity of the \( S_1 \), \( C_1 \), and \( S_2 \) layers. Also, the figure shows that doubling the number of threads is associated with a consistent improvement in speed up by a factor of approximately 2X. An exception to this is the 16-thread configuration, where a speedup of only 1.3X is observed when compared to the 8-thread configuration. To explain this behavior, we study the per-layer execution
time and its contribution to the overall execution time as demonstrated in Table 1. The table demonstrates how the contribution of each layer is influenced by the number of threads and the number of orientations. Note that the preprocessing stage, $S_0$, is not included in the table due to its insignificant contribution. The table shows that the $S_2$ layer dominates the total execution time with an average of 96.4% across all thread configurations for both 4 and 12 orientations. This explains the relatively low increase in speedup of the 16-thread configuration compared to the 8-thread configuration. Performing input-to-prototype correlation constitutes the critical path in the $S_2$ layer, as shown in Figure 3. Similarly, Figure 4 reveals that $S_2$ layer takes advantage of thread-level parallelism in gaining more speed up for both 4 and 12 orientations. However, the performance benefit associated with the increase in number of threads starts to decrease as number of threads exceeds a certain threshold (i.e. 8 threads in this case), as convolution-like operations dominate the computation. Similarly, Figure 5 shows the increase in power consumption as number of instantiated threads increase.

Table 1: Execution time for each layer in the HMAX model, in percentage, out of the total execution time

<table>
<thead>
<tr>
<th># Threads</th>
<th># Orientations</th>
<th>$S_1$</th>
<th>$C_1$</th>
<th>$S_2$</th>
<th>$C_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1.62</td>
<td>0.19</td>
<td>97.61</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>1.77</td>
<td>0.21</td>
<td>97.82</td>
<td>0.19</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1.69</td>
<td>0.22</td>
<td>97.45</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>1.79</td>
<td>0.22</td>
<td>97.78</td>
<td>0.19</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1.89</td>
<td>0.30</td>
<td>97.15</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>1.90</td>
<td>0.26</td>
<td>97.60</td>
<td>0.17</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>3.54</td>
<td>0.58</td>
<td>94.97</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>2.96</td>
<td>0.40</td>
<td>96.27</td>
<td>0.25</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>4.71</td>
<td>0.87</td>
<td>93.26</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>3.08</td>
<td>0.56</td>
<td>95.92</td>
<td>0.29</td>
</tr>
</tbody>
</table>
From the observations presented above, it is evident that the HMAX model would benefit from parallelism. However, to further speed up the algorithm, application-specific acceleration is required for non-linear operations, in general, and convolutions, in particular. Also, profiling the model shows that $S_2$ layer is the hotspot and therefore, accelerating this layer will improve the overall execution time the most.

3. The Architecture of the Neuromorphic Accelerators

This section discusses the design and implementation of the hardware accelerators for the HMAX model. These accelerators are the result of careful analysis of the HMAX computational structures and the specifics of the data flow across the layers.

To interconnect these accelerators, we use a high bandwidth, packet-switched Network-on-Chip, NoC, which supports runtime re-configurability [10]. The accelerators attached to this network can be either stream-based for on-the-fly per-pixel operations, or compute-based for iterative processing operations on non-contiguous data. Additionally, the network supports the notion of flow, where each packet is prefixed with a certain data flow that can be used to drive the operation currently being executed on stream-based accelerators. These flows are configured and tracked through a distributed routing scheme, which can be updated at runtime.

3.1 HMAX Accelerators

3.1.1 Downsampling (DS) Accelerator

The DS accelerator is a simple direct-subsampling module which samples from the input stream every $i^{th}$ input pixel, where $i$ is determined by a subsampling factor associated with a scale. The subsampling also occurs in the $y$-
dimension by dropping entire rows from the input. The subsampling factors for x and y are stored in registers, which allows the image to be subsampled to multiple scales.

### 3.1.2 $S_1$ Accelerator

The $S_1$ layer performs a streaming convolution on its input. The $S_1$ accelerator, illustrated in Figure 6, consists of a serial-to-parallel FIFO, a 2D streaming convolution engine that feeds an adder tree, and a bank of Gabor coefficients. The serial-to-parallel FIFO reconstructs the serially streamed image pixels into row-wise parallel output that is fed to the 2D filter. The convolution engine is a 2D 11x11 filter that feeds an adder tree. The accelerator utilizes a scratchpad memory (coefficients bank) to store the Gabor coefficients for all supported orientations. These coefficients are selectively accessed and loaded to the convolution engine based on the incoming flow associated with the current frame.

![Figure 6](image)

**Figure 6:** $S_1$ accelerator. (a) The accelerator consists of a serial-to-parallel FIFO to convert the streaming serial input into parallel output that feeds the 2D systolic filter. The filter supports reloadable coefficients. (b) The coefficients are stored in a multi-bank scratchpad memory.

### 3.1.3 $C_1$ Accelerator

![Figure 7](image)

**Figure 7:** $C_1$ accelerator. (a) The accelerator consists of units to keep track of the current x and y coordinates and processors that perform windowed pooling in the x and y positions. (b) The Horizontal-Window processor is a 1D window processor in the x direction.
3.1.3 \( C_1 \) Accelerator

The \( C_1 \) accelerator, illustrated in Figure 7, performs a windowed pooling on a stream of inputs. The accelerator contains several local memories used to store \( x \) and \( y \) position information for each window. This information is compared against the current incoming pixel’s location in order to determine to which window a current pixel belongs. Since windows can overlap in both \( x \) and \( y \) dimensions, multiple compare units exist, with each responsible for comparing a separate sub-window in \( x \) and \( y \) in a hierarchal fashion. At the lowest level, Horizontal-Window processors are responsible for a 1D horizontal window in \( x \), which tracks the current sub-windows assigned to it. Alternatively, a Vertical-Pixel router keeps track of which Horizontal-Window processors are currently active in order to handle overlaps in \( y \).

3.1.4 \( S_2/C_2 \) Accelerator

The \( S_2 \) layer performs template matching through correlation between the outputs of the \( C_1 \) layer and pre-stored prototypes. The outputs of this layer are pooled on by the \( C_2 \) layer to find the global maximum across all positions and scales for each prototype in the \( S_2 \) dictionary. Careful analysis of the data flow between \( S_2 \) and \( C_2 \) layers shows that combining these two layers into a single accelerator leads to several benefits. First, this allows for the pooling operation to occur immediately following the computation of the current \( S_2 \) output without a delay. Second, combining these two layers can effectively decrease the amount of data required to be sent across the network by

\[
\frac{\sum_{S=0}^{S-1}(X_S \times Y_S) \times N_{\text{proto}}[X_S, Y_S]}{(N_{\text{proto}} \times 2)}
\]

Here \( S \) is the number of input image scales at the \( S_2 \) layer, \( X_S \) (\( Y_S \)) is the dimension of scale \( S \) in the \( x \) (\( y \)) dimension, and \( N_{\text{proto}} \) is the number of prototypes. We use the notation \( N_{\text{proto}}[X_S, Y_S] \) to denote total number of prototypes that can be correlated with that scale. With a dictionary of 5120 prototypes, this results in a data transfer reduction of 4,154X when operating on 12 scales at 12 orientations.

Figure 8 illustrates the architecture of the \( S_2/C_2 \) accelerator. First, the input scale (i.e. \( C_1 \) output) is buffered in the Image Memory, which is a scratchpad memory with enough capacity to store all orientations of the largest scale. Since template matching is an iterative process across all prototypes in the \( S_2 \) dictionary, the local buffering of the input scale can dramatically reduce the amount of data sent across the network by:

\[
\frac{\sum_{S=0}^{S-1}(X_S \times Y_S) \times N_{\text{orient}} \times N_{\text{proto}}[X_S, Y_S]}{\sum_{S=0}^{S-1}(X_S \times Y_S) \times N_{\text{orient}}}
\]

Here, \( N_{\text{orient}} \) is number of orientations. Buffering the image reduces the amount of data sent across the network by approximately 5005X when using 12 scales, 12 orientations and 5120 prototypes.

The image buffer feeds an array of convolvers, allowing the accelerator to parallelize the template-matching process across these convolvers. Each one of these convolvers is a 2D, multi-tap systolic engine with reloadable coefficients, where coefficients are the fuzzy templates in the \( S_2 \) dictionary. Each convolver can support all four kernel sizes (i.e. 4, 8, 12 and 16), which can be configured at runtime by the accelerator’s controller (not shown in figure). The prototypes are stored in SRAM memory due to their relatively large size (24 MB). To hide SRAM memory latency, the accelerator overlaps memory reads and convolution, where small-size FIFOs are used to buffer prototypes from the memory.
The S₂ layer accumulates the correlation output across all orientations within the same scale for the current prototype. This is accomplished through the Accumulation pipeline stage, where the output of the convolution is processed in a streaming fashion. A temporary buffer, Accumulation Memory, stores the current output of each correlation. When the convolution engine produces the output of the next orientation, the hardware logic reads the corresponding accumulated output of the previous correlation from memory, updates it with the current output, and writes the result back to the Accumulation Memory. The logic was designed such that accumulation is done on-the-fly in order to sustain a throughput of one accumulation per cycle. Only after processing the last orientation, does the Accumulation stage stream the accumulated output to the Normalization stage. The Normalization stage normalizes the accumulated correlation output to the pre-computed normalization patch, which is the output of a windowed averaging function applied on the current input scale. This stage operates in streaming fashion, where in each cycle the logic reads the corresponding normalization pixel from the Normalization Patch Memory, performs normalization, and then transfers the results to the C₂ pooling stage.

The last stage in the accelerator performs the global pooling to find the maximum response across all positions and scales for each prototype. Upon receiving a new response from the previous stage, the hardware logic in the C₂ stage fetches the corresponding response from the C₂ Tables. If the current value is larger than the value stored in the table, then the logic will write back the current value. When scales and prototypes are fully processed, the user makes a read request to the S₂/C₂ accelerator, which transfers the feature vector stored in the C₂ tables by DMA.

Figure 8: S₂/C₂ accelerator. (a) Pipeline stages. (b) The Accumulation stage performs pixel-wise accumulation across correlation outputs for each orientation within the same scale. (c) The Image Memory buffers the C₁ outputs. (d) The Normalization stage normalizes the accumulated output. (e) The C₂ performs global max over all positions and scales for each prototype independently.
### 3.1.5 Network-Configurable Correlation Filter

This section proposes an enhanced architecture for the correlation filter, which can be effectively used in $S_2$ template-matching operation. As presented earlier, the $S_2$ layer performs correlation with multiple kernel sizes, namely, 4×4, 8×8, 12×12 and 16×16. A configurable correlation filter is proposed to implement these kernels that maximize the hardware utilization for three of the four kernel sizes.

The basic building block is a 4×4 filter, which is used to build the larger filters (i.e. 8×8, 12×12 and 16×16). To this end, a total of sixteen 4×4 filters are arranged in a network-like structure, such that these blocks may be cascaded on demand to form the desired filter size. Figure 9 shows a configurable filter capable of supporting kernel sizes of 4×4 and 8×8. When only 4×4 kernel sizes are required, the outputs of 4×4 filters A, B, C and D are directly sent to the adder tree. When 8×8 kernel sizes are required, the outputs of convolvers A and B are fed to convolvers C and D.

The inputs of filters A and B are fed by the configurable FIFO array built with two cascaded 1-to-4 FIFOs. For the 8×8 correlation filter, the adder tree sums the eight outputs from convolvers C and D and produces the final result.

The multiplexers route the data according to the kernel size chosen by the user. Similarly, four 8×8 filters can be used to form a 16×16 filter, as depicted in Figure 10, and it can be configured as sixteen 4×4 or four 8×8 correlation filters. The 16×16 filter also supports 12×12 correlation operation, where only nine 4×4 filters are used while the other seven are disabled.

### 3.2 System-Level Operation

Figure 11 illustrates the HMAX accelerators mapped to a heterogeneous system. The CMP is mainly used for configuration, writing the input image, and reading out the results. First, a process running on the CMP schedules a DMA transfer of Gabor coefficients to the $S_1$ accelerator and $S_2$ dictionary prototypes to the $S_2/C_2$ accelerator. Then, the CMP performs the necessary configurations of the accelerators (e.g. orientation count, frame size, etc…).

Whenever a frame is received from an input source (e.g. camera), the process schedules a DMA transfer of the frame from its system memory to the $C_1$ accelerator. On its path to $C_1$, the frame traverses through the DS and $S_1$ accelerators. The $C_1$ pools over two adjacent scales and then transfers its output to the $S_2/C_2$. Then, the $C_1$ notifies $S_2/C_2$ to start the template-matching process followed by global max pooling. When all scales are processed, the CMP process issues a read request to the $S_2/C_2$ accelerator and schedules a DMA read transfer from the accelerator to its memory. Then, the process uses the results (i.e. feature vector) for classification.
4. Experimental Evaluation

This section presents an evaluation of the proposed accelerators in terms of classification accuracy, performance and power efficiency. A multi-FPGA system is used to validate the accelerators. In addition, CPU and GPU implementations of HMAX model are used to compare with the accelerated HMAX.

4.1 Experimental Setup

A CPU implementation of the HMAX model [9] is used to evaluate the performance of HMAX when executed on the CPU platform. The workload was further extended to allow thread-level parallelism across scales and orientations. The workload was compiled with the highest optimization and using the SSE2 instruction set extension. The workload was executed on an Intel Xeon machine, consisting of two 2.4 GHz quad-core processors. By enabling Intel’s HyperThreading, a total of 16 virtual processors are available to execute the workload. The CPU platform contains 12GB of system memory. This section uses CPU performance when all 16 threads are utilized for executing the workload.

We use [8] as an optimized CUDA implementation of the HMAX model. The workload is executed on an Nvidia Tesla M2090 board [11], which houses a 1.3 GHz Tesla T20A GPU, with a 1.34 GB global memory. A 3 GHz 12-core Xeon processor is used to compile the GPU, with a total of 49 GB system memory.

The proposed neuromorphic accelerators are mapped to a heterogeneous system [12] that loosely-couples a CMP with four FPGA devices. The system houses an Intel S7000FC4UR motherboard with a 3.2GHz quad-core Xeon processor, and a total of 24 GB system memory. The motherboard uses Front-Side Bus (FSB) to interface the CMP to the FPGAs. The four FPGAs are Virtex-5 SX-240T [13] operating at 100 MHz.

In addition, our experimental setup included power consumption measurements for all three platforms. For the GPU platform, the command tool “nvidia-smi -q” is used to probe the power consumption from a power sensor found on the GPU board. For the CPU and FPGA platforms, power consumption was measured using an accurate meter for

![Figure 11: Accelerated HMAX mapped to a heterogeneous system. CMP-Accelerator interactions are indicated with the red-dotted lines, while the blue-dotted lines indicate inter-accelerator data movement.](image)
power consumption measurements. The meter provides continuous and instantaneous reading of power drawn by the platform with 99.8% accuracy. The power consumption for all platforms is measured only after the platform reaches steady-state to obtain the baseline power measurement. Then, the workload is executed and peak power is measured throughout the duration of the workload execution.

### 4.2 Classification Accuracy

Two datasets are used to evaluate the classification accuracy of the accelerated HMAX, as shown in Table 2. The accelerated HMAX is used to extract features from the images contained within the datasets. These features are fed to a regularized least-square (RLS) classifier that is trained and tested with non-overlapping image sets.

<table>
<thead>
<tr>
<th>Dataset name</th>
<th># Categories</th>
<th># images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech 256 [14]</td>
<td>257 (256 categories + clutter)</td>
<td>30607</td>
</tr>
</tbody>
</table>

Figure 12 illustrates the classification accuracy for Caltech256 for both 4 and 12 orientations, where the number of scales is fixed to 12. The figure shows the influence of number of training images on the overall accuracy. For example, when using 40 images to train the classifier, the overall accuracy is 25.95% (23.23%) for 12- (4-) orientation configuration. Compared to other object recognition algorithms, we find that accuracy of HMAX is at most 12% less than Griffin et al. [14]. However, we are unable to determine exactly how much of that loss in accuracy is contributed to HMAX feature extractor, as we use a different classifier.

![Classification accuracy for Caltech256 dataset using 4 and 12 orientations with varying number of training images](image-url)
Figure 13: Classification accuracy for the PASCAL VOC2007 dataset compared with best Average Precision (AP) as reported by [16]. HMAX results are obtained using 12-scale and 12-orientation configuration.

Figure 13 shows the classification accuracy for the PASCAL VOC2007 dataset. A multiple-classifier voting scheme was used for generating the probabilities (i.e. confidence rates). A total of 10 independent classifiers were used, each trained using equal-sized disjoint subsets of the training data, with a total of 15662 objects, 1566 objects per subset, extracted from 5011 images. The classifications and generated probabilities are fed to the PASCAL evaluation tool in order to retrieve the Average Precision (AP) rate.

Additionally, we compare the classification accuracy of the accelerated HMAX to the CPU implementation. We find that the accelerated HMAX is 2% less accurate than the CPU implementation. This degradation is due to the truncation of the fixed-point representation, used by accelerated HMAX, during the multiply-accumulate operation. In contrast, both CPU and GPU platforms use floating-point representation to carry out the computations.

Figure 14: Speedup of neuromorphic accelerators over CPU and GPU for 4 and 12 orientations. Values are normalized to CPU.

Figure 15: Power efficiency of neuromorphic accelerators over CPU and GPU for 4 and 12 orientations. Values are normalized to CPU.
4.3 Speedup and Power Efficiency

Figure 14 demonstrates the speedup in frames-per-second (fps) gained by the accelerated HMAX compared to CPU and GPU implementations. For the 4-orientation configuration, the neuromorphic accelerators outperform the CPU (GPU) by 7.3X (1.1X). In contrast, for the 12-orientation configurations, the FPGA platform delivers 8X (1.24X) more speedup compared to the CPU (GPU) implementations.

There are a number of factors that played a role in speedup exhibited by the neuromorphic accelerators:

1) Fully pipelined and customized streaming architecture: These customized architectures allow for data reuse, hence avoiding unnecessary data fetching. For instance, the systolic correlation filter implemented within the $S_2/C_2$ accelerator exploits data reuse where pixels are propagated across multiple multiply-and-accumulate units.

2) Exploitation of parallelism: FPGAs offer high degree of parallelism that is often not available on other platforms. For example, the $S_2$ correlation filter performs 256 multiply-and-accumulate operations simultaneously, providing a 256X increase in performance over sequential operation. This high degree of parallelism is not achievable on general purpose CPU architectures. Even contemporary architectures with explicit vector-processing extensions lack the number of functional units and optimized memory infrastructure to exploit the immense data-level locality inherent in the many convolution operations of both the $S_1$ and $S_2/C_2$ accelerators.

3) Custom numerical representation: All accelerators use fixed-point representation with varying bit widths suitable for the current operation. For instance, the $S_1$ accelerator uses 18 bits to represent each Gabor coefficient, while 22 bits are used to represent each pixel in the input image. This ability to operate on varying bit width operands is unmatched by CPUs and GPUs.

4) Task-level parallelism: In order to improve the execution time of HMAX, four instances of the $S_2/C_2$ accelerator were mapped to the FPGAs. Each instance operates on a subset of the prototypes, and therefore, total execution time was reduced to approximately 25%.

5) Efficient use of memory hierarchies: The HMAX accelerators made use of multiple hierarchies of memory. For instance, the $S_2/C_2$ accelerator used SRAM memory that feeds a queue buffer in order to overlap data fetch and computation. This mechanism hides the latency of data fetch and hence improves overall performance. Similarly, the accelerator used a local buffer to store the input images in order to reduce communication latency by 5005X.

6) Reliable communication infrastructure: As stated earlier, the neuromorphic accelerators are interconnected using a NoC [10] that supports runtime re-configurability. Our measurements show that this network can achieve up to 1.6 GB/s (3.2 GB/s) bandwidth when clocked at 100 MHz (200 MHz), supporting high transfer rates across the network. Additionally, using flows to associate incoming packets with operations reduces the latency that would otherwise be incurred if accelerators were required to be reconfigured. For instance, the $S_1$ accelerator determines the current orientation to process by simply examining the flow of the current frame. Hence, the user is not required to pre-configure the accelerator for each orientation.
Additionally, using the network-configurable convolution engine presented in Section 3.1.5, the performance of the S2 accelerator improved by 5X.

Speedup is not the only metric used for evaluating the performance and efficiency of a particular design. Power efficiency, measured in fps-per-watt, is another metric that is used to assess the amount of performance that can be delivered within an available power budget. As shown in Figure 15, the FPGA platform exhibits 10.9X (2.3X) more power efficiency when compared to CPU (GPU) platforms when using 4-orientation configurations. Moreover, when using 12 orientations, the FPGA platform delivers even more power efficiency (~13.8X (2.6X) when compared to the CPU (GPU) platform.

HMAX accelerators implement customized architectures that deliver high throughput while operating at a low frequency. This is the main driver for lower power consumption, and consequently higher power efficiency.

These speedup and power efficiency benefits are achieved by mapping the neuromorphic accelerators to 65nm SRAM-based FPGA devices operating at 100 MHz clock. It is expected that implementing these accelerators in silicon will accentuate such benefits.

4.4 Resource Utilization

The neuromorphic accelerators were validated on four Virtex-5 SX-240T FPGAs. Table 3 shows the amount of resources utilized by the HMAX accelerators when mapped to the FPGAs. The registers and LUTs were mainly utilized for implementing registers and control logic. The BlockRAM units are mainly dedicated for scratchpad memories and FIFO-like structures, while the DSP48 slices were mainly allocated for Multiply-And-Accumulate operations that took place in the convolution engines.

5. Related Work

There have been many approaches that attempt to mimic the hardware substrate of the visual cortex using spiking neural networks and artificial synapse implementations [17,18]. In contrast to specialized analog circuitry used in these efforts, we rely on digital CMOS substrate in this work and focus only on the algorithmic abstractions of the brain models. Unlike digital circuits, scaled technologies – with lower supply voltages and low power transistors – are not very beneficial for analog design [19]. An additional challenge is that converting sensed input to a format recognized by spiking neural networks is still an active research area. Consequently, our architecture is able to leverage technology scaling without additional design challenges.

<table>
<thead>
<tr>
<th>Registers</th>
<th>Lookup Tables (LUTs)</th>
<th>Block RAM (BRAM)</th>
<th>DSP48E</th>
</tr>
</thead>
<tbody>
<tr>
<td>316,794</td>
<td>133,611</td>
<td>623</td>
<td>2,206</td>
</tr>
<tr>
<td>(52.88%)</td>
<td>(22.3%)</td>
<td>(30.18%)</td>
<td>(52.23%)</td>
</tr>
</tbody>
</table>
This work is aligned with other efforts towards domain-specific computing [20,21], CPU specialization [22], and configurable accelerators [23,24].

This work is based on our previous effort in accelerating HMAX for object recognition [25]. The neuromorphic accelerators presented in this paper include a number of optimizations including enhanced runtime configurability, optimized memory access and hierarchy, and leveraging task-level parallelism to the user.

Since then, other works have proposed specialized accelerators to accelerate HMAX. For example, both [26] and [27] proposed an implementation customized for sparsified HMAX [7]. Our work is distinguished from these efforts in the following aspects:

1) An extensive set of experiments was conducted to profile the HMAX model. The findings from these experiments gave a better understanding of the computational and power efficiency shortcomings when executing the model on a multi-core processor.

2) Unlike [26] and [27], we proposed and implemented all stages of the HMAX model into custom accelerators that are either stream-based or compute-based, depending on the inherent computational structure of a particular stage.

3) Both [26] and [27] use Radial Basis Function (RBF) as the kernel for the $S_2$ layer. However, a more recent implementation of HMAX uses Normalized Dot Product\(^3\). This requires augmenting the layer with a normalization stage that introduces additional complexity to the design. Our $S_2/C_2$ accelerator implements an efficient streaming normalization block as described earlier.

4) Unlike previous efforts targeted towards accelerating HMAX, this work provides the classification accuracy of accelerators using two well-known datasets (i.e. Caltech256 and PASCAL). Furthermore, this work discusses the impact of using fixed-point representation on the overall accuracy, when compared to a floating-point representation.

In [28], we present a neuromorphic accelerator system for universal recognition. However, this work focuses more on analyzing the complexity of the HMAX specifically for object recognition applications. The HMAX for object recognition has unique data flow requirements compared to other applications. Moreover, the profiling was done on a more powerful machine (16- vs. 12-core CPU) and execution time and power consumption were measured for multiple variations of thread-level parallelism. Additionally, this paper discusses the neuromorphic accelerators in more depth. Finally, this work uses different workloads to measure the performance, power and accuracy.

6. Conclusions

This paper proposed an accelerated neuromorphic system for object recognition. First, we profiled HMAX to have a better understanding of the model’s computational structure. Then, we discussed the architectural details of the accelerators for each layer in the model. A multi-FPGA system was used to validate the proposed system. The accuracy of the feature extractor was tested against Caltech256 and PASCAL VOC2007 datasets. Furthermore, a comparison is made between the performance of the accelerators, CPU, and GPU implementations of the HMAX

---

\(^3\) Personal communication with Jim Mutch, CBCL, MIT
model. In terms of execution time, the accelerators outperform the CPU (GPU) platform by 8X (1.24X). Similarly, results reveal that the accelerators are 13.8X (2.6X) more power efficient when compared to CPU (GPU) platform.

**Acknowledgments**

The authors would like to thank the reviewers for their valuable comments and suggestions. Also, the authors would like to thank Jim Mutch, MIT for his help in providing the most up to date implementation of the HMAX model. This work was funded in part by DARPA’s NeoVision 2 program, and NSF Awards 1147388, 0916887, 0903432. Ahmed Al Maashri is sponsored by a scholarship from the Government of Oman.

**References**


RESPONSES TO QUESTIONS AND COMMENTS

The authors would like to thank the reviewers for their valuable comments and suggestions. In this document we address the issues raised by the reviewers.

Reviewer #1:

A. The paper by Al Maashri and colleagues entitled "Hardware Acceleration for Neuromorphic Vision Algorithms" describes the design and implementation of hardware accelerators for a popular computational model of object recognition in the cortex (HMAX). The approach is based on a detailed analysis of the model computational structures and the specifics of the data flow across the layers. Specifically the manuscript proposes to accelerate the S2 (and C2) layer of HMAX which accounts for more than 93% of total processing time. It is shown that proper customized architecture (FPGA) for data reuse speeds up the S2 processing and reduces data transfers. Another speed up strategy is proposed whereby the max pooling operations are computed on the fly during the S2 unit computations. The FPGA can also use customized data representations for each layer (18 bits for gabor and 22 for image pixel values). The approach is tested on two standard datasets (Pascal VOC and CalTech-256) and compared to state-of-the-art. Speed improvements with minimal decrease in accuracy (due to fixed point representation and truncated values compared to the CPU implementation of HMAX) are reported over the original model implementations (both CPU and GPU).

On the positive side, the manuscript makes a number of (small but novel) contributions. The architecture is properly tested for speed and accuracy. On the negative side, the improvements remain relatively modest (X13.8 compared to original CPU and 2.6X compared to original GPU implementations of HMAX).

Response: Thank you for your comment. We believe that the performance speedup quoted in the paper is due to the specific FPGA prototyping platform used in our experiments and not due to the architectural features. For instance, the FPGAs used in our experiments are based on 65nm node, while both the CPU and GPU are from lower technology nodes, namely, 45nm and Fermi 40nm, respectively. By scaling the process technology, it is expected that the speedup and power efficiency of the proposed architecture will only increase. Additionally, the SRAM-based FPGAs are clocked at only 100 MHz. If this architecture were fabricated in CMOS ASIC, then it could be clocked at a much higher frequency, boosting the overall speedup. Finally, Task-Level Parallelism (TLP) and Data-Level Parallelism (DLP) have proven to be very effective in reducing the S2/C2 execution time. However, due to resource limitation on the FPGAs, we were able to map only 4 instances of the accelerator, with each instance consisting of only two data processing paths (i.e. 2D filters, accumulation, normalization, etc ...). If the architecture was mapped to more recent FPGA device (e.g. Virtex-6 or Virtex-7), or even CMOS ASIC, many more of these accelerators could be supported resulting in significantly superior performance.

B. pp. 2: HMAX is actually not orientation invariant (and this agrees with the response properties of IT cells which are strongly tuned for the object orientation and also 3D pose).

Response: It is true that cells are strongly tuned to a certain orientation. This is why the HMAX model employs a bank of Gabor filters (S1) to extract the responses at different orientations. Then, these responses are aggregated in the accumulation stage within the S2 layer. The accumulated response produces a map that consists of regions of strong responses at edges of different orientations, and therefore shows some tolerance to changes in the rotation of the object (See [4] for more details on experiments performed by R&P on improving the rotation invariance). As the number of Gabor filters increases, the invariance to changes in rotation also increases. We have tested this in our experiments with 4 and 12 orientations. In both cases, the accelerators showed tolerance to rotations that conforms to the findings in [6, 7].
C. pp. 3: C1 is not just scale invariance but also translation invariance (via pooling over space). Is this missing in the current implementation?

Response: The C1 accelerator was designed to pool across two adjacent scales as well as to max-pool within the scale itself, such that the accelerator supports the scale- and translation-invariance properties of the HMAX model. Figure 7 was added to the revised manuscript to illustrate the architecture of the C1 accelerator.

D. pp.4 (eq 2): What is \( x_i \)? To my knowledge this is not in the original model. I am not sure what this does? Is \( ||X||^2=\text{Sum}_i x_i^2 \)? If so the denominator is always negative?

Response: That is true; the original HMAX model used Radial Basis Function, RBF, to compute the response at the S2 layer [6]. A more recent implementation of the model [9] replaces RBF with Normalized Dot Product, NDP. Empirically, NDP yields better results, as conveyed to us by the author of [7, 9] Jim Mutch, MIT. Having said that, and after revising the specifics of the implementation, we realized that there is a typo in the denominator of equation 2 that we have corrected in the revised manuscript (A square root was missing and the second term should be normalized to the patch size). Please note that our implementation is not affected by this since the typo was only found in the manuscript. In addition, we have updated the sub-section “S2 (Tuned features) layer” to elaborate on the specifics of the equation.

Reviewer #3:

This paper proposed an FPGA approach to neuromorphic vision systems that improved the power efficiency over CPUs (significantly) and GPUs (to a lesser degree).

The paper is well written and well organized. The "Related Work" section, unexpectedly placed after the results, which is unusual, actually worked well in this paper, since it was not essential for following the paper.

Formatting is well done, and figures are clear.

The results show the tradeoff is a minor degradation of accuracy.

A couple of potential improvements:

A. Page 3 line 23: "This work uses an implementation derived from the extension developed by Mutch & Lowe [7]" Why was that particular approach followed? A sentence or two about that would give the reader the motivation for that path followed.

Response: Thank you for your comment. This has been fixed in the revised manuscript.

B. Why was a 5120-prototype dictionary chosen? (Page 4 line 5)

Response: The dictionary size is determined during the HMAX learning phase and is governed by the current categorization task. Also, the increase in the size of the dictionary leads to slower execution time. We have tested our accelerators with three different dictionaries of sizes 4075, 5000, and 5120. All these dictionaries were obtained through personal communication with Jim Mutch, MIT. We opted to use the same dictionary used by the HMAX algorithm developers/keepers to give more credibility to the accuracy and performance gain against their reference results. We also want to stress that our accelerators are not tied to any particular dictionary size since the S2 accelerator is run-time reconfigurable.