SPINDLE: SPINtronic Deep Learning Engine for Large-scale Neuromorphic Computing*

Shankar Ganesh Ramasubramanian, Rangharajan Venkatesan, Mrigank Sharad, Kaushik Roy and Anand Raghunathan
School of Electrical and Computer Engineering, Purdue University
West Lafayette, IN, USA
{sramasub,rvenkate,msharad,kaushik,raghunathan}@purdue.edu

ABSTRACT

Deep Learning Networks (DLNs) are bio-inspired large-scale neural networks that are widely used in emerging vision, analytics, and search applications. The high computation and storage requirements of DLNs have led to the exploration of various avenues for their efficient realization. Concurrently, the ability of emerging post-CMOS devices to efficiently mimic neurons and synapses has led to great interest in their use for neuromorphic computing.

We describe SPINDLE, a programmable processor for deep learning based on spintronic devices. SPINDLE exploits the unique ability of spintronic devices to realize highly dense and energy-efficient neurons and memory, which form the fundamental building blocks of DLNs. SPINDLE consists of a three-tier hierarchy of processing elements to capture the nested parallelism present in DLNs, and a two-level memory hierarchy to facilitate data reuse. It can be programmed to execute DLNs with widely varying topologies for different applications. SPINDLE employs techniques to limit the overheads of spin-to-charge conversion, and utilizes output and weight quantization to enhance the efficiency of spin-neurons. We evaluate SPINDLE using a device-to-architecture modeling framework and a set of widely used DLN applications (handwriting recognition, face detection, and object recognition). Our results indicate that SPINDLE achieves 14.4X reduction in energy consumption and 20.4X reduction in EDP over the CMOS baseline under iso-area conditions.

Categories and Subject Descriptors

C.1.3 [**PROCESSOR ARCHITECTURES**]: Other Architecture Styles (Neural Nets)

Keywords

Spintronics; Emerging Devices; Nanoelectronics; Post-CMOS; Neural Networks; Neuromorphic Computing

1. INTRODUCTION

Neuromorphic algorithms, which mimic the functionality of the human brain, are used for a wide class of applications involving classification, recognition, search, and inference. While the roots of neuromorphic algorithms lie in simple artificial neural networks, contemporary networks have grown to be much larger in scale and complexity. In this work, we focus on deep learning networks (DLNs) [1–3], an important class of large-scale neural networks that have shown state-of-the-art results on a range of problems in text, image, and video analysis. DLNs are currently used in real-world applications such as Google+ image search, Apple Siri voice recognition, house number recognition for Google Maps, *etc.* [4, 5]. In order to achieve high accuracy and robustness to variations in inputs, DLNs utilize several layers with each layer consisting of a large number of neurons and varying interconnectivity patterns. For example, a DLN that recently won the Imagenet visual recognition challenge contains around 650,000 neurons and 60 million synapses, and requires compute power in the order of 2-4 GOPS per classification. The complexity of DLNs, together with the growth in the sizes of data sets that they process, places high computational demands on the platforms that execute them. Several research efforts have been devoted to realizing efficient implementations of DLNs using multi-core processors, graphics processing units, and hardware accelerators [6–8]. All these approaches are limited by one common fundamental bottleneck - in effect, they emulate neuromorphic systems using primitives (instructions, digital arithmetic units, and Boolean gates) that are inherently mismatched with the constructs that they are used to realize (neurons and synapses).

A concurrent trend that has catalyzed the field of neuromorphic computing is the emergence of post-CMOS devices. Although a clear replacement for Silicon and CMOS is yet to be found, many emerging devices have unique characteristics and strengths that are different from CMOS. Among them, spintronics, which uses electron spin rather than charge to represent and process information, has attracted great interest. Spintronic memories promise high density, non-volatility, and near-zero leakage, leading to extensive research, industry prototypes, and early commercial offerings in recent years [9]. Recently, it has been demonstrated that spintronic devices can also be used to directly mimic the computations performed in neurons and synapses while operating at very low voltages [10, 11], leading to greatly reduced area and power over digital and analog CMOS implementations. These advances raise the prospect of realizing large-scale neuromorphic systems such as DLNs in an energy-efficient manner using spintronics. However, several challenges need to be addressed to realize this vision.

First, a direct mapping of a DLN into a network of spintronic neurons and synapses, as envisioned by previous work, leads to a large, inefficient design, especially for DLNs of high complexity (*e.g.*, ~10^6 neurons and ~10^8 synapses). Second, for broader utility, it is desirable to have a programmable platform that can implement a wide range of networks of varying complexity and topology, as required by different applications. Third, the low spin diffusion length in most materials mandates the use of charge-based interconnects, imposing overheads for spin$\Longleftrightarrow$charge conversion. Finally, the energy consumed by spin-neurons increases drastically with the precision at which their weights are stored and their outputs are computed. Therefore, careful architectural design is required in order to preserve the intrinsic efficiency of spintronic devices for large-scale neuromorphic computing.

*This work was supported in part by STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and in part by the Natonal Science Foundation under grant no. 1320808.

ISLPED'14, August 11–13, 2014, La Jolla, CA, USA.
Copyright 2014 ACM 978-1-4503-2975-0/14/08 ...$15.00.
http://dx.doi.org/10.1145/2627369.2627625 .

In this work, we propose SPINtronic Deep Learning Engine (SPINDLE), an energy-efficient programmable architecture for Deep Learning Networks (DLNs). We consider various key characteristics of DLNs and spintronic devices in the design of SPINDLE. Notably, DLNs exhibit multiple levels of nested parallelism and are comprised of a few recurring computation patterns. At the lowest level, they contain fine-grained data-parallel computations such as convolution and sub-sampling. These computations are in turn organized into layers, with task parallelism within each layer and producer-consumer parallelism across layers. DLNs exhibit significant data reuse across computations, and exploiting this reuse is critical to reducing off-chip memory accesses, as well as limiting the on-chip storage requirements. Considering these characteristics, we propose a hierarchical three-tiered architecture for SPINDLE, consisting of Spin Neuromorphic Arrays (SNAs), Spin Neuromorphic Cores (SNCs) and SNC Clusters. SPINDLE uses a two-level memory hierarchy, consisting of on-chip distributed scratchpad memories that are local to SNCs, and shared off-chip memory. We propose various techniques to enhance the efficiency of SPINDLE, including intra- and inter-layer data reuse, and neuron output and weight quantization. We evaluate SPINDLE using a hierarchical modeling framework, starting with physics-based device simulation, and constructing circuit and architectural macro-models of spin-based neurons and memories. We compare SPINDLE to a well-optimized CMOS baseline using three popular DLN applications - handwriting recognition, face detection, and object recognition. Our analysis shows that SPINDLE achieves 14.4X reduction in energy and 20.4X reduction in energy-delay product under iso-area, establishing the potential of spintronic devices for large-scale neuromorphic computing.

The rest of the paper is organized as follows. Section 2 provides the necessary background on DLNs. Section 3 presents the design of spin-based neurons and memory, which form the building blocks of SPINDLE. Section 4 describes the SPINDLE architecture and the techniques used to improve its energy efficiency. Section 5 details the experimental setup including our device-to-architecture modeling framework. Section 6 presents results comparing SPINDLE with a CMOS baseline. Section 7 provides an overview of prior efforts on hardware for neuromorphic computing, and Section 8 concludes the paper.

2. DEEP LEARNING NETWORKS

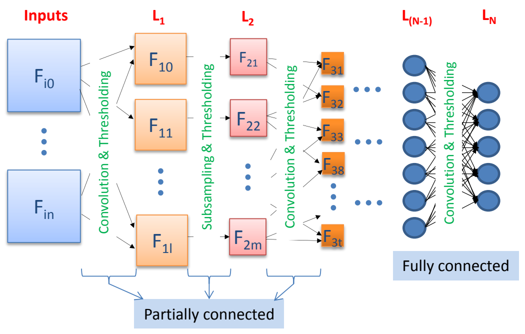

Inputs
L1
L2
L(N-1)
LN
Fi0
Fin
F10
F11
F1l
F21
F22
F2m
F31
F32
F33
F38
F3t
Convolution & Thresholding
Subsampling & Thresholding
Convolution & Thresholding
Convolution & Thresholding
Fully connected
Partially connected

Figure 1: Structure of a DLN

DLNs (Fig. 1) are feed-forward neural networks in which the neurons are organized into well-defined layers. Each layer is composed of multiple features that can be generated in parallel from features of the previous layer using one of two kinds of operations: (i) Convolution-and-Thresholding and (ii) Subsampling-and-Thresholding.

Convolution-and-Thresholding (C-T): A C-T operation takes i features from the input layer ($i \leq N$ where N is the number of features in the input layer), and produces one feature in the output layer. First, a kernel (matrix of weights) is convolved with each input feature (convolution involves computing dot-product of the kernel with regions of the input feature in a sliding window fashion), and the results are summed up to produce an intermediate output. A bias value is then added and a thresholding operation, also known as an activation function (typically tanh or sigmoid) is applied to produce a feature of the output layer. This process is repeated M times, where M is the number of features in the output layer.

Subsampling-and-Thresholding (S-T): Subsampling takes one input feature and produces one output feature in which each value of an output feature is produced by first computing the average of the neighboring values over a sliding window, adding a bias, and then performing a thresholding operation.

The connectivity between layers falls into two categories: (i) partially connected layers where each feature is connected to a subset of features from the previous layer (ii) fully connected layers where each feature is connected to all features in the previous layer. Note that DLNs contain three levels of parallelism - producer-consumer parallelism across layers, task parallelism within a layer, and fine-grained data parallelism within each C-T or S-T operation. Furthermore there exists significant data reuse across and within these operations.

The computational complexity of a DLN is determined by the number of layers, the number and sizes of network inputs and features in each layer, connectivity between layers, and kernel sizes. For example, CIFAR, an image classification DLN, takes an input of size 3x32x32, and has 685 features spread across 6 layers, requiring 3.77 million multiply-accumulate computations per classification. DLNs are also memory-intensive. For example, CIFAR requires memory accesses amounting to 7.5 MB per input.

3. SPINDLE BUILDING BLOCKS

The fundamental building blocks of SPINDLE are spintronic neurons and memory. In this section, we provide a brief description of their structure and operation.

3.1 Spin-Neurons

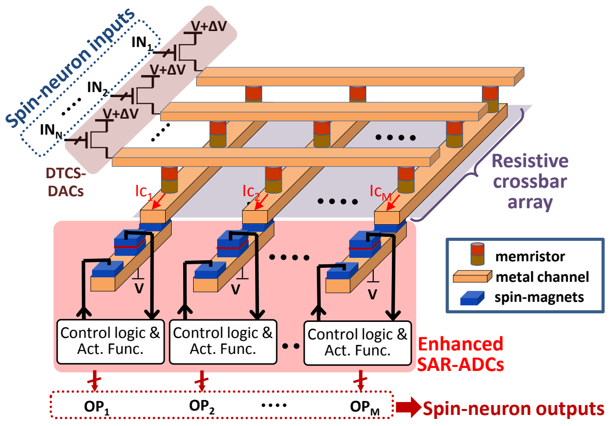


Figure 2: Array of spin-neurons

Fig. 2 shows an array of M spin-neurons that take N inputs each. The spin-neuron array consists of (i) Deep Triode Current Source Digital-to-Analog Converters (DTCS-DACs) that convert the N digital inputs ($IN_1 \ldots IN_N$) into analog currents, (ii) a resistive crossbar array (N x M) that is used to perform weighted summation of the neuron inputs, and (iii) enhanced Successive Approximation Register Analog-to-Digital Converters (SAR-ADCs) that evaluate the activation function and produce the M digital outputs. Although the proposed design utilizes transistors and memristive elements, the spintronic comparator unit in the SAR-ADC is key to the energy-efficiency of the entire neuron since it enables operation at very low voltages (~10s of mV).

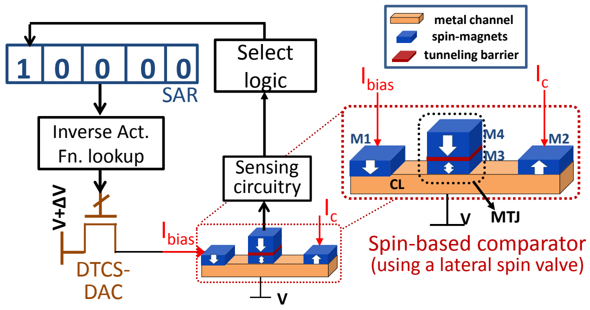


Figure 3: Enhanced spin-based SAR-ADC

The enhanced spin-based SAR-ADC is described in greater detail in Figure 3. The input current (I_C) is received from one of the columns of the resistive crossbar array. The SAR-ADC consists of a successive approximation register (SAR), a lookup table that stores the inverse of the desired neuron activation function, a DTCS-DAC, a spin-based comparator that is based on a lateral spin valve [12], and control logic. The SAR-ADC successively computes the quantized digital neuron output over i cycles where i is the desired precision. In each cycle, it generates a bias current (I_{bias}) using an inverse activation function lookup based on the current SAR value, compares I_{bias} against the input current I_C, and updates one bit of the SAR with the output of the comparator. The spin-based comparator consists of two fixed magnets (M1 and M2) of opposite spin-polarizations and a free magnet (M3) that are all connected to a metallic channel. The free magnet M3 is combined with a fixed magnet (M4) and a tunneling barrier to form a magnetic tunnel junction (MTJ). The comparison is performed by passing I_{bias} and I_C through magnets M1 and M2, respectively, resulting in the injection of spin currents of opposing polarity into the channel (CL). This spin-current injection results in the switching of M3 (the free layer of the MTJ) along a direction given by the greater of I_c or I_{bias}. The output of the comparison is determined by sensing the resistance of the MTJ, much like the read operation in STT-MRAM. Since the magnets and the channels are metallic, the comparator can be operated at very low voltages of the order of 10s of millivolts.

The resistive crossbar array stores weights in the memristors that are located at its cross-points. A memristor storing a k-bit weight needs to support 2^k resistance levels. The memristors can be programmed to different resistance values using current pulses. The number of current pulses required to program the memristor depends on the number of resistance levels. Hence, the programming energy varies exponentially with the precision of weights stored in the memristor.

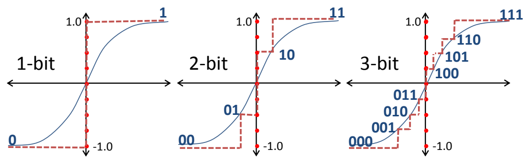


Figure 4: Staircase approximations of tanh()

The enhanced SAR-ADC described in Figure 3 can be used to perform a multi-bit staircase approximation of various activation functions. An example of successive approximation for tanh() is shown in Fig. 4 for different bit-precisions. The 1-bit approximation leads to the step function and as we increase the number of bits the staircase approximation more closely approximates the actual function. However, in order to get a n-bit approximation, we need to repeat the SAR-ADC computation 'n' times with an exponentially increasing bias current. This increases the energy and latency for high precision computations.

The cross-bar based spin-neuron performs a 120-input weighted summation ~117X and ~60X more energy efficiently as compared to state-of-the-art digital and analog CMOS (45 nm) implementations, respectively [10, 11]. The energy efficiency of the spin-neuron derives from two primary sources: (i) very low voltages compared to analog and digital CMOS implementations, and (ii) much lower circuit complexity (device count) compared to digital implementations that utilize adders and multipliers. Finally, we believe that the proposed design should be feasible from an integration perspective, since both spintronic and memristive devices have been shown to be compatible with current CMOS fabrication processes.

3.2 Spin Memory

Spintronic memories achieve high density and very low leakage, but require higher write energy compared to CMOS memories. In this work, we utilize a memory bit-cell based on Domain Wall Memory (DWM) [13] that preserves the density and leakage benefits of spintronic memories while significantly reducing the high write energy requirements. The schematic of the utilized bit-cell, which we call 1bitDWM, is shown in Fig. 5. It consists of a ferromagnetic wire, a magnetic tunneling junction (MTJ) and 2 access transistors. The ferromagnetic wire consists of 3 domains – two fixed and one free. When the magnetic orientation of the free domain is parallel (anti-parallel) to the fixed layer of the MTJ, it offers low (high) resistance, representing the '0' ('1') state. The read operation is performed by sensing this difference in resistance, similar to conventional STT-MRAM. However, the write operation uses a completely different mechanism based on the phenomenon of domain wall motion. The magnetization of the fixed domains in the left and right ends of the nanowire can be propagated to the free domain by applying a current along the nanowire [13]. The shift-based write mechanism is far more efficient than the MTJ-based writes used in STT-MRAM. The area of the 1bitDWM bit-cell is comparable to a 1T-1R STT-MRAM bit-cell, despite the former using two access transistors, due to the smaller transistor sizes required for 1bitDWM as a result of the lower write current requirement. In summary, 1bitDWM outperforms both SRAM and STT-MRAM [13]; therefore, we use it to design on-chip memories in SPINDLE.

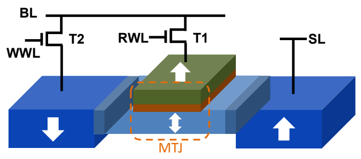


Figure 5: Schematic of spin memory bit-cell [13]

4. SPINDLE ARCHITECTURE

In this section we describe the SPINDLE architecture, which composes spin-neurons and memories to provide a programmable platform for the execution of DLNs.

The SPINDLE architecture, shown in Figure 6, employs a three-level hierarchy to match the nested parallelism present in DLNs. Spin Neuron Arrays (SNAs) constitute the lowest level of the hierarchy, and combine the spin-neurons described in Section 3.2 with peripheral circuitry to realize convolution-and-thresholding or subsampling-and-thresholding operations. SNCs, which represent the second level of the hierarchy, are composed of multiple SNAs, local scratchpad memory, and a dispatch unit. They are typically used to perform a collection of operations that share input features (hence, fostering data re-use from the scratchpad). SNC Clusters form the next level of the hierarchy and consist of multiple SNCs connected by a local bus, and further exploit intra-layer parallelism. SPINDLE is a collection of SNC clusters connected to a Global Control Unit through a common global bus. Inter-layer parallelism is exploited at the top level of the SPINDLE hierarchy.

When the degree of parallelism in the DLN exceeds that supported by SPINDLE, the DLN can be decomposed into partitions that are executed in a serial manner, with intermediate results stored in the memory hierarchy. This allows SPINDLE to execute a wide range of DLNs and makes the architecture scalable and programmable.

The components of SPINDLE are described in greater detail in the following subsections.

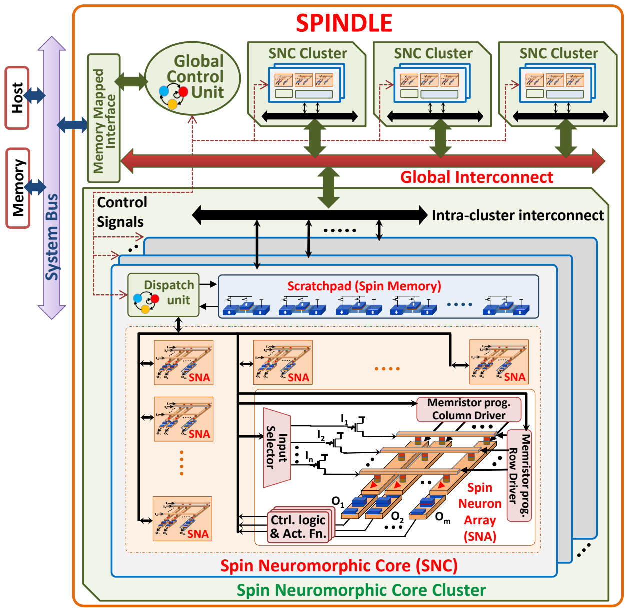


Figure 6: SPINDLE architecture

4.1 Spin Neuron Arrays

An SNA performs the smallest unit of computation, namely a convolution or subsampling followed by thresholding. It is fed with the required input features in a streaming manner and produces an output feature. As shown in Figure 6, each SNA consists of a set of spin-neurons (Fig. 2), an input selector unit and a memristor programming circuit.

The dispatch unit within each SNC streams the input features required by all SNAs. The input selector unit in each SNA selects the subset of inputs that SNA uses, and feeds this data to the spin-neurons. The spin-neurons accept the inputs offered by the input selector unit, and perform weighted summations with the kernel (weight matrix) stored in the memristors. Each spin-neuron (column of the crossbar) produces one element of the output feature. The complete output feature is obtained over multiple cycles. The memristor programming circuit consists of row and column drivers. The drivers select a memristor located at a particular row and column (analogous to the decode circuits in RAM), and drive the appropriate current to program the memristor.

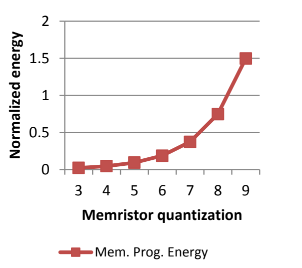
A key property of the SNA design is that all the neurons in the array share the same inputs and execute in parallel. This results in substantially lower control overheads, since these overheads are amortized across all spin-neurons. In addition, the computations that produce adjacent elements in an output feature share a large fraction of their inputs, and the crossbar structure is naturally suited for exploiting this fine-grained data reuse. However, each column requires only a subset of the inputs, and the memristor weights for the unused inputs are set to 0. Thus, increasing the data reuse by increasing the number of columns results in a larger fraction of the memristors being programmed to 0, leading to lower energy efficiency. We determine the number of columns in each SNA of SPINDLE by balancing the benefits of data reuse against the overheads of programming zero weights to memristors.

4.2 Spin Neuromorphic Cores

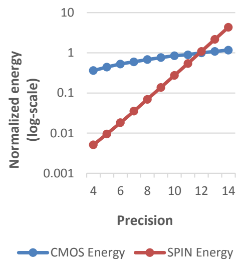
SNCs consist of a group of SNAs, local scratchpad memory, and a dispatch unit. Each SNC executes a set of convolution or sub-sampling operations from within a layer, while exploiting any available input feature re-use across these operations.

The scratchpad memory forms the highest level of the memory hierarchy and stores the input features used by the SNAs in the SNC as well as the output features that they generate. As described in Section 3.2, we utilize 1bitDWM for designing the scratchpad memory in SPINDLE. The dispatch unit performs three key functions. First, it reads all the input features stored in scratchpad memory and broadcasts them to all SNAs. The input selector in each SNA is programmed to select a subset of input features based on the connectivity of the layer being evaluated. This enables all SNAs in an SNC to perform their respective convolution or subsampling computations in a parallel lock-step fashion. This design choice amortizes control overheads across the SNAs, and enables data reuse across SNAs so that the read traffic to the scratchpad is minimized. Second, the dispatch unit controls the memristor programming circuitry, and supplies weights from the scratchpad memory to each SNA. Finally, the dispatch unit manages communication with the Global Control Unit (GCU).

As described in Section 3.1, the output precision and the number of memristor levels strongly influence the energy expended by SNAs. Therefore, each SNC has suitable tuning knobs to control the output precision by varying the number of successive approximations in the SAR-ADC, and the precision of memristor programming. Fig. 4.2(a) shows the increase in energy required to program a memristor at increasing levels of precision, and Fig. 4.2(b) compares the energy consumption of a spin-neuron to a digital CMOS neuron for various values of output precision. The figures show that it is highly desirable to operate spin-neurons at the lowest possible precision in order to achieve higher energy efficiency.



(a) Weight Precision



(b) Output precision

Figure 7: Effect of precision on energy

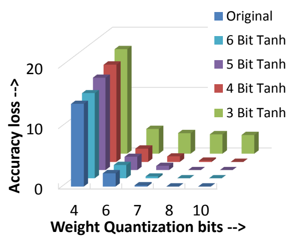


Figure 8: Quantization *vs.* application accuracy

However, excessively reducing the precision of weights and outputs can lead to a reduction in output quality for the network. Therefore, it is necessary to determine the lowest weight and neuron output precisions that lead to acceptable output quality. For example, Fig. 8 shows the impact of weight and output quantization on the classification accuracy loss for the object detection benchmark (CIFAR). For this benchmark, we choose a weight precision of 7 bits and output precision of 4 bits, since they result in negligible impact on the output accuracy.

4.3 SNC Cluster and Global Control Unit

The SNC Cluster consists of a group of SNCs that are connected through a high bandwidth and low-latency local bus. SNC clusters also exploit intra-layer parallelism by generating different features of the same layer in parallel. The hierarchical bus architecture, if combined with a locality-aware

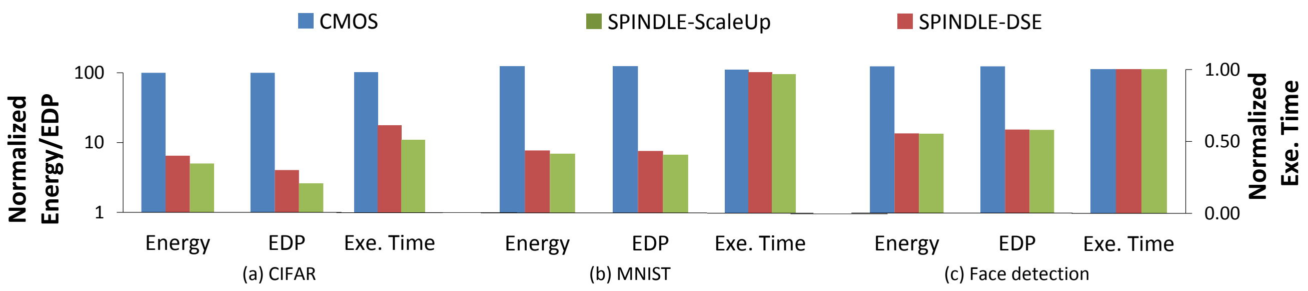


Figure 9: Comparison of benefits for individual benchmarks

mapping of the network to SPINDLE, has a potential to reduce traffic on the high-energy global bus and off-chip memory accesses. The Global Control Unit (GCU) orchestrates the overall execution of the DLN by triggering the execution of parts of the network on each SNC cluster, and the transfer of data between SNC Clusters, and to/from off-chip memory.

5. EXPERIMENTAL METHODOLOGY

In this section, we provide a description of the experimental methodology used to evaluate SPINDLE.

5.1 Modeling Framework

SPINDLE varies from a traditional CMOS architecture in that it is composed of disparate technologies namely spin-neurons, spin memory and CMOS. We model each of these technologies across multiple levels of abstraction using the framework shown in Fig. 10.

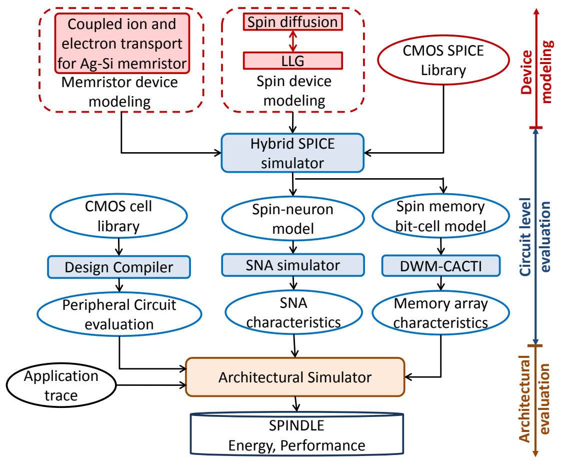


Figure 10: Modeling framework

At the device level, we model spin devices using a physics-based modeling framework [11] that incorporates the effects of variations in order to extract key energy and timing parameters. We model the dynamics of current-induced magnetic switching including the variations caused by thermal noise using Stochastic Landau-Lifshitz-Gilbert equation (LLG), while the transport of spin-current across magnetic devices is modeled using diffusive spin-transport. Parameter variations amounting to 3σ of 15% were considered in critical magnet parameters like saturation magnetization and damping coefficient, and also in the DTCS transistor threshold voltage. We note that the use of hard-axis switching significantly reduces the dependence of switching current on magnet parameters, as a small amount of positive or negative current (greater than thermal noise) can ensure correct switching operation.

At the circuit level, the spintronic devices are abstracted using behaviorally equivalent SPICE models [14]. Similarly, a circuit model for memristors is obtained using the Ag-Si memristor device parameters from [15]. We combine the SPICE models of spin devices, memristors and CMOS transistors to form a circuit model for spin-neurons. We use this circuit model to characterize SNAs.

In order to model spin memories, we evaluate the bit-cells using SPICE models and use a variant of CACTI [16] called DWM-CACTI [13,17] to evaluate memory array characteristics. We use CACTI [16] to model the CMOS memories in the baseline design. The energy and timing for the control logic in SPINDLE as well as the entire CMOS baseline are obtained by synthesizing RTL implementations using Synopsys Design Compiler to the 45nm NangateOpenCell FreePDK library.

The DLN networks for various benchmarks are built and trained on the Torch framework [18]. We train the application using the training input set and evaluate the application level accuracy using an independent testing input set. We then quantize the weights and activation function to a lower precision and retrain the application until quality bounds ($<$ 1.5% loss in accuracy compared to the original network) are met. The quantized DLNs are executed on the SPINDLE architectural simulator to obtain their execution traces. These traces are analyzed with the appropriate energy and performance models for the components of SPINDLE to compute the application level energy and Energy-Delay-Product (EDP).

5.2 DLN Benchmarks

To evaluate the benefits of SPINDLE, we use 3 representative DLN benchmarks: (i) handwriting recognition on the MNIST database [1], (ii) face detection, and (iii) object classification on the CIFAR-10 dataset [19]. Relevant details of these three benchmarks are given in Table 1.

Table 1: Benchmark characteristics

Benchmark	#Layers	#Features	#MACs (million)	Data Transfers (KB)
MNIST	6	165	0.3	1456
CIFAR	6	685	3.77	7540
Face Detection	4	83	0.14	176

6. EXPERIMENTAL RESULTS

In this section we first present a summary of the results demonstrating the benefits of the SPINDLE architecture compared to the CMOS baseline. We then perform a design space exploration to study the sensitivity of these benefits to various architectural parameters.

6.1 Result Summary

We evaluate the benefits provided by SPINDLE against the CMOS baseline described in Section 5.1 under iso-area conditions. Note that, for the same number of processing elements, SPINDLE achieves considerable area improvements over the CMOS baseline. In order to perform an iso-area comparison, we consider two different designs: (i) SPINDLE-ScaleUp, in which we reinvest the area benefits by increasing the number of SNC Clusters, and (ii) SPINDLE-DSE, in which we perform a design space exploration to determine the best configuration of SPINDLE that uses the same area as the CMOS baseline.

The energy benefits for each of our benchmarks are shown in Fig. 9. Our results show that SPINDLE-ScaleUp can achieve 12.6X lower energy and 15.5X lower EDP compared to the

CMOS baseline. This demonstrates that the SPINDLE architecture largely preserves the intrinsic benefits of spintronic neurons and memory. SPINDLE-DSE further improves the benefits and achieves 14.4X energy and 20.4X EDP improvements, underscoring the value of architectural design space exploration.

6.2 Design Space Exploration

In this section, we study the impact of varying the architectural parameters of SPINDLE on its energy and performance.
Impact of local memory per SNC: Varying the local memory per SNC results in an initial sharp decrease in overall system energy followed by a gradual increase as shown in Fig. 11. Initially, the increase in local memory enables us to exploit the data reuse inherent in DLNs, resulting in fewer off-chip memory transactions, and leading to significant energy savings. Further, this also reduces the idle time of the SNAs resulting in improved performance. However, once all of the input features fit in the local memory, any further increase only results in costlier reads and writes to local memory, and higher leakage energy without reducing the execution time. This results in the gradual increase in total energy.

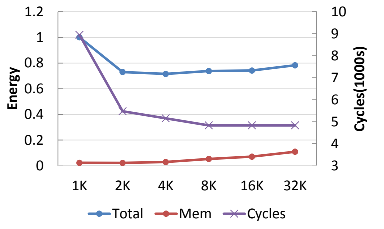


Figure 11: Effect of varying the local memory (MNIST)

Impact of SNAs per SNC: Increasing the number of SNAs per SNC exploits greater inter-feature parallelism resulting in more output features being computed in parallel in each SNC. However, it can also potentially result in memory starvation, leading to longer execution times, since each new SNA may require additional features as inputs. Thus, an increase in the number of SNAs/SNC results in faster execution time up to a point, beyond which the system slows down as shown in Fig. 12.

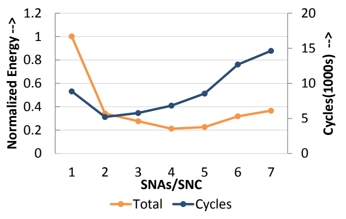


Figure 12: Effect of varying the SNAs/SNC (Face detection)

In summary, our results show that it is possible to achieve order-of-magnitude improvements in energy efficiency for large-scale neuromorphic computing with spintronic devices. The SPINDLE architecture balances efficiency and flexibility, by preserving the intrinsic benefits of spin devices to a large extent, while allowing for programmability across networks of varying sizes and topologies.

7. RELATED WORK

There have been several efforts to realize neuromorphic algorithms using custom accelerators [7,8] and graphics processors [6]. In addition, there have been efforts to more faithfully mimic biological neurons in CMOS circuits to achieve greater computational capability and efficiency [20,21]. The analog nature of biological neurons has also prompted efforts to realize neuromorphic algorithms using CMOS analog circuits [22]. Despite these efforts, the size and power consumption of CMOS implementations remains a major challenge.

In recent years there have been efforts to use emerging technologies such as PCRAM [23], memristors [24], and spintronics [10] to realize neurons and synapses. Preliminary investigations of these technologies at the device level have shown that they are highly promising for realizing the fundamental building blocks of neuromorphic computing. In particular, PCRAM and memristors enable dense, crossbar memory designs that can store the weights compactly. Spin-based devices can match the basic computation patterns in neuromorphic algorithms while enabling very low-voltage operation [10].

While the above efforts have shown that emerging devices are promising for neuromorphic computing, they are primarily at the device level and form the motivation for our work, which focuses on the design of a large-scale, programmable hardware architecture based on these building blocks.

8. CONCLUSION

Spintronic devices have emerged as a promising technology that can realize the building blocks of neuromorphic computing platforms. To investigate their potential for large-scale neuromorphic systems, we propose SPINDLE, a programmable spintronic processor for deep learning networks. The design of SPINDLE was driven by considering application and device characteristics, and aimed to balance the objectives of energy efficiency and programmability. Our evaluations using a device-to-architecture modeling framework demonstrate over an order-of-magnitude energy benefits over a CMOS baseline.

9. REFERENCES

[1] Y. LeCun et al. Gradient-based learning applied to document recognition. In *Proc. IEEE*, 1998.
[2] G. Hinton et al. A fast learning algorithm for deep belief nets. *Trans. Neural Computation*, 2006.
[3] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2009.
[4] George Rosenberg. Improving photo search: A step across the semantic gap http://googleresearch.blogspot.com/2013/06/improving-photo-search-step-across.html. June 2009.
[5] Y. Netzer et al. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
[6] J. Bergstra. Theano: Deep learning on GPUs with Python. In *Big Learn workshop NIPS, 2011*.
[7] M. Sankaradas et al. A massively parallel coprocessor for convolutional neural networks. In *Proc. ASAP*, 2009.
[8] C. Farabet et al. Hardware accelerated convolutional neural networks for synthetic vision systems. In *Proc. ISCAS*, 2010.
[9] E. Chen et al. Advances and future prospects of spin-transfer torque random access memory. *IEEE Trans. Magnetics*, 46(6):1873–1878, June 2010.
[10] M. Sharad et al. Boolean and non-boolean computation with spin devices. In *Proc. IEDM*, 2012.
[11] M. Sharad et al. Spin neuron for ultra low power computational hardware. In *Proc. DRC*, 2012.
[12] T. Kimura et al. Switching magnetization of a nanoscale ferromagnetic particle using nonlocal spin injection. *Phys. Rev. Lett.*, 2006.
[13] R. Venkatesan et al. DWM-TAPESTRI - An energy efficient all-spin cache using domain wall shift based writes. In *Proc. DATE*, 2013.
[14] G.D. Panagopoulos et al. Physics-Based SPICE-Compatible Compact Model for Simulating Hybrid MTJ/CMOS Circuits. *Trans. TED*, 2013.
[15] L Gao et al. Analog-input analog-weight dot-product operation with Ag/a-Si/Pt memristive devices. In *Proc. VLSI-SoC*, 2012.
[16] CACTI. http://www.hpl.hp.com/research/cacti/.
[17] R. Venkatesan et al. TapeCache: A High Density, Energy Efficient Cache Based on Domain Wall Memory. In *Proc. ISLPED*, 2012.
[18] R. Collobert et al. Torch7: A Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.
[19] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.
[20] E. Painkras et al. Spinnaker: A multi-core system-on-chip for massively-parallel neural net simulation. In *Proc. CICC*, 2012.
[21] J. Seo et al. A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *Proc. CICC*, 2011.
[22] B. Rajendran et al. Specifications of nanoscale devices and circuits for neuromorphic computational systems. *Trans. TED*, 2013.
[23] D. Kuzum et al. Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano Letters*, 2012.
[24] S. H. Jo et al. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Letters*, 2010.