Memristive Boltzmann Machine: A Hardware Accelerator for Combinatorial Optimization and Deep Learning

Mahdi Nazm Bojnordi and Engin Ipek
University of Rochester, Rochester, NY 14627 USA
{bojnordi, ipek}@ece.rochester.edu

ABSTRACT

The Boltzmann machine is a massively parallel computational model capable of solving a broad class of combinatorial optimization problems. In recent years, it has been successfully applied to training deep machine learning models on massive datasets. High performance implementations of the Boltzmann machine using GPUs, MPI-based HPC clusters, and FPGAs have been proposed in the literature. Regrettably, the required all-to-all communication among the processing units limits the performance of these efforts.

This paper examines a new class of hardware accelerators for large-scale combinatorial optimization and deep learning based on memristive Boltzmann machines. A massively parallel, memory-centric hardware accelerator is proposed based on recently developed resistive RAM (RRAM) technology. The proposed accelerator exploits the electrical properties of RRAM to realize *in situ*, fine-grained parallel computation within memory arrays, thereby eliminating the need for exchanging data between the memory cells and the computational units. Two classical optimization problems, graph partitioning and boolean satisfiability, and a deep belief network application are mapped onto the proposed hardware. As compared to a multicore system, the proposed accelerator achieves 57× higher performance and 25× lower energy with virtually no loss in the quality of the solution to the optimization problems. The memristive accelerator is also compared against an RRAM based processing-in-memory (PIM) system, with respective performance and energy improvements of 6.89× and 5.2×.

1. INTRODUCTION

Combinatorial optimization is a branch of discrete mathematics that is concerned with finding the optimum element of a finite or countably infinite set. An enormous number of critical problems in science and engineering can be cast within the combinatorial optimization framework, including classical problems such as the traveling salesman, integer linear programming, knapsack, bin packing, and scheduling [1], as well as numerous optimization problems in machine learning and data mining [2]. Because many of these problems are NP-hard, heuristic algorithms commonly are used to find approximate solutions for even moderately sized problem instances.

Simulated annealing is one of the most commonly used optimization algorithms. On many types of NP-hard problems, simulated annealing achieves better results than other heuristics [3]; however, its convergence may be slow. This problem was first addressed by reformulating simulated annealing within the context of a massively parallel computational model called the *Boltzmann machine* [4]. The Boltzmann machine is amenable to a massively parallel implementation in either software or hardware; as a result, high performance implementations of the model using GPUs [5, 6], MPI-based HPC clusters [7, 8], and FPGAs [9, 10] have been proposed in recent years. With the growing interest in deep learning models that rely on Boltzmann machines for training (such as deep belief networks), the importance of high performance Boltzmann machine implementations is increasing. Regrettably, the required all-to-all communication among the processing units limits the performance of these recent efforts.

This paper proposes a massively parallel, memory-centric hardware accelerator for the Boltzmann machine based on recently developed resistive RAM (RRAM) technology. RRAM is a memristive, non-volatile memory technology that provides FLASH-like density and DRAM-like read speed. The accelerator exploits the electrical properties of the bitlines and wordlines in a conventional single level cell (SLC) RRAM array to realize *in situ*, fine-grained parallel computation, which eliminates the need for exchanging data among the memory arrays and the computational units. The proposed hardware platform connects to a general-purpose system via the DDRx interface and can be selectively integrated with systems that run optimization and machine learning tasks.

Two classical examples of combinatorial optimization, graph partitioning and boolean satisfiability, as well as a deep belief network application are mapped to the proposed hardware accelerator. As compared to a multicore system with eight out-of-order cores, the end-to-end execution time is improved by an average of 57× over a mix of 20 real-world optimization problems; the system energy is decreased by 25× on average. The respective speedup and energy savings for deep learning tasks are 68× and 63×. The proposed system is also compared against a processing-in-memory (PIM) based accelerator that integrates the processing units close to the memory arrays for efficient computation. The experiments show that the memristive Boltzmann machines outperforms PIM by more than 5× in terms of both performance and energy.

2. BACKGROUND AND MOTIVATION

The Boltzmann machine is a massively parallel computational model that implements simulated annealing—one of the most commonly used heuristic search algorithms for combinatorial optimization.

2.1 The Boltzmann Machine

The Boltzmann machine, proposed by Hinton *et al.* in 1983 [4], is a well-known example of a stochastic neural network capable of learning internal representations and solving combinatorial optimization problems. The Boltzmann machine is a fully connected network comprising two-state units, and employs simulated annealing for transitioning between the possible network states [11]. The units flip their states based on the current state of their neighbors and the corresponding edge weights to maximize a global consensus function, which is equivalent to minimizing the network energy.

Many combinatorial optimization problems, as well as machine learning tasks, can be mapped directly onto a Boltzmann machine by choosing the appropriate edge weights and the initial state of the units within the network. As a result of this mapping, (1) each possible state of the network represents a candidate solution to the optimization problem, and (2) minimizing the network energy becomes equivalent to solving the optimization problem. The energy minimization process is typically performed by either adjusting the edge weights (*learning*) or recomputing the unit states (*searching and classifying*). This process is repeated until convergence is reached. The solution to an optimization problem can be found by reading—and appropriately interpreting—the final state of the network.

2.1.1 Stochastic Dynamics of the Boltzmann Machine

A binary[1] Boltzmann machine minimizes an *energy function* specified by

$$E(\mathbf{x}) = -\frac{1}{2}\sum_{j}\sum_{i\neq j} x_j x_i w_{ji} - \sum_{j} x_j w_{jj} \tag{1}$$

where w_{ji} is the weight of the connection between the units i and j, x_i is the state of unit i, and $\mathbf{x}$ is a vector specifying the state of the entire machine. The state transition mechanism of the Boltzmann machine relies on a stochastic acceptance criterion, which allows the optimization procedure to escape from local minima. A change in the state of unit j results in a new state vector $\mathbf{x}^j$. Let x_i^j denote an element of this new vector, where

$$x_i^j = \begin{cases} x_i & \text{if } i \neq j \\ 1 - x_i & \text{if } i = j \end{cases} \tag{2}$$

In other words, only one of the units—unit j—has changed its state from x_j to $1-x_j$ in this example. (In reality, *all* of the units compute their states in parallel.) The corresponding change in energy is computed as follows:

$$\Delta E = (2x_j - 1)(\sum_{i\neq j} x_i w_{ji} + w_{jj}). \tag{3}$$

[1]Without loss of generality, we restrict the discussion to Boltzmann machines with binary states. Background on Boltzmann machines with bipolar states can be found in the literature [11].

Notably, the change in the energy is computed by considering only local information. State transitions occur probabilistically: unit j flips its state with probability

$$P(\mathbf{x}^j|\mathbf{x}) = \frac{1}{1+e^{\frac{\Delta E}{C}}} \tag{4}$$

where $\mathbf{x}$ represents the current state of the machine, $\mathbf{x}^j$ is the new machine state after unit j flips its state, and C is a control parameter analogous to the temperature parameter in simulated annealing. Conceptually, C influences the probability of accepting a sub-optimal state change: when C is large, the state transition probability is insensitive to small changes in energy (ΔE); in contrast, when C is small, a relatively small change in energy makes a big difference in the corresponding state transition probability.

2.1.2 Mapping Combinatorial Optimization Problems to the Boltzmann Machine

Numerous mapping algorithms have been proposed in the literature for formulating classic optimization problems within the Boltzmann machine framework [11, 12]. We review two examples, the Max-Cut and Max-SAT problems, to demonstrate representative mapping algorithms.

The Maximum Cut Problem. Max-Cut is a classic problem in graph theory [1]. Given an undirected graph $\mathbf{G}$ with N nodes whose connection weights (d_{ij}) are represented by a symmetric weight matrix, the maximum cut problem is to find a subset $S \subset \{1,...,N\}$ of the nodes that maximizes $\sum_{i,j} d_{ij}$, where $i \in S$ and $j \notin S$. To solve the problem on a Boltzmann machine, a one-to-one mapping is established between the graph $\mathbf{G}$ and a Boltzmann machine with N processing units. The Boltzmann machine is configured as $w_{jj} = \sum_i d_{ji}$ and $w_{ji} = -2d_{ji}$. Figure 1 depicts the mapping from an example graph with five vertices to a Boltzmann machine with five nodes. When the machine reaches its lowest energy ($E(\mathbf{x}) = -19$), the state variables represent the optimum solution, in which a value of **1** at unit i indicates that the corresponding graphical node belongs to S.

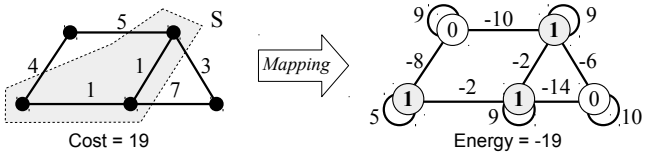


Figure 1: Mapping a Max-Cut problem to the Boltzmann machine model.

The Maximum Satisfiability Problem. Given a Boolean formula in conjunctive normal form, the goal of the Max-SAT problem is to determine the maximum number of clauses that hold true when truth values are assigned to the Boolean variables. Let ε be a Boolean formula represented as $\varepsilon = \wedge_{j=1...M} C_j$, where M is the number of clauses, $C_j = \vee_{i=1...m_j} L_i$ is a clause in disjunctive form, m_j is the number of literals in clause C_j, and L_i is either a Boolean variable or its negation. The maximum satisfiability problem can be stated as the search for the maximum $\varepsilon^* \subseteq \varepsilon$ such that ε^* is satisfiable. To solve a Max-SAT problem with N Boolean variables, a Boltzmann machine comprising $2N$ units is required. Two units (i and j) are used to represent a Boolean variable (u)

and its negation ($\overline{u}$). The connections of the Boltzmann machine are then defined as *clauses* (w_{jk} where $k \neq i, j$), *biases* (w_{jj}), and *exclusion links* (w_{ji}), which are initialized according to a previously proposed algorithm [12].

Figure 2 illustrates the three-step mapping of the Boolean formula $\varepsilon = (x \vee y \vee z) \wedge (x' \vee y \vee z)$ to a Boltzmann machine with six processing units. (For simplicity, details of how the auxiliary edges are assigned are omitted from the discussion.) The units are first labeled by the true and complementary values of the Boolean variables and all of the network edge weights are initialized to zero. The Boolean clauses are then mapped onto the machine by decrementing the edge weights involved in connecting the literals of each clause (1). An edge weight may be adjusted multiple times during this process; for instance, the edge weight between the units y and z is decremented twice. The newly adjusted clause edges are then used to determine the unit biases by computing the sum of all of the edges connected to each unit (2). A large weight value is assigned to the exclusion links—the edges between the true and complementary values of each Boolean variable—to eliminate invalid solution candidates from the search space (3). At the end of the optimization process, where the network reaches its lowest energy, the final states of the units are used to evaluate the Boolean formula (ε) and to find the optimization outcome, which is the number of satisfied clauses.

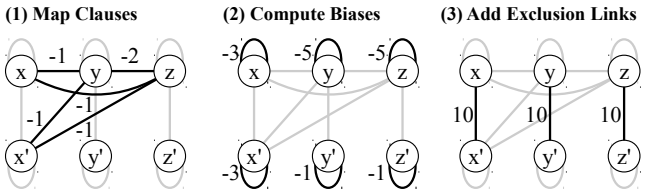


Figure 2: Three steps of mapping a Max-SAT problem to the Boltzmann machine model.

2.1.3 Mapping Deep Learning Problems to the Boltzmann Machine

Deep learning, one of the most successful supervised learning methods, relies on hierarchical feature learning in which higher level features are composed from lower level ones [13]. Boltzmann machines have shown potential for efficient feature extraction in deep machine learning; in particular, restricted Boltzmann machines (RBMs) are the fundamental building blocks of *deep belief networks* (Figure 3) [14, 15].

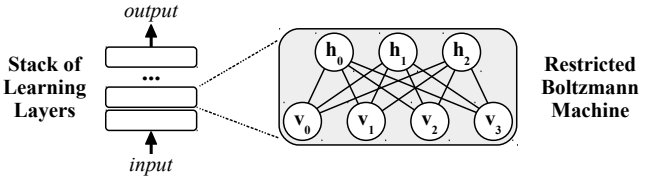


Figure 3: Deep learning with Boltzmann machines.

Restricted Boltzmann Machines. Restricted Boltzmann machines (RBMs) are a variant of the Boltzmann machine whose units are partitioned into *visible* and *hidden* units. Similarly to "unrestricted" Boltzmann machines, symmetric links are used to connect the visible and hidden units; however, hidden-to-hidden and visible-to-visible connections are not allowed. This restriction allows for more efficient training algorithms than those that are available for the general Boltzmann machine [16]. Traditional training algorithms for RBMs are time consuming due to their slow convergence rate [17]. However, they usually are trained using approximate training algorithms. One such recently proposed algorithm that has proven successful in practice is the *contrastive divergence* algorithm [18].

Contrastive Divergence Learning. This algorithm consists of multiple steps for updating the connection weights of the RBM. A single step of contrastive divergence (CD-1) comprises the following phases:

- *Positive phase*: Clamp the input sample $\mathbf{v}$ to the input layer, and propagate $\mathbf{v}$ to the hidden layer. The resulting hidden layer activations are represented by the vector $\mathbf{h}$.
- *Negative phase*: Propagate $\mathbf{h}$ back to the visible layer with result $\mathbf{v}'$. Propagate the new $\mathbf{v}'$ back to the hidden layer with new activation vector $\mathbf{h}'$.
- *Weight update*: Update the connection weights according to $\mathbf{W} = \mathbf{W} + \gamma(\mathbf{v}\mathbf{h}^T + \mathbf{v}'\mathbf{h}'^T)$

where $\mathbf{W}$ is the weight matrix, $\mathbf{v}$, $\mathbf{h}$, $\mathbf{v}'$, and $\mathbf{h}'$ are state vectors, and γ is a real number in the range $[0, 1]$.

2.1.4 Implementation Challenges

The massive parallelism of the Boltzmann machine, as well as its ability to solve optimization problems without requiring detailed knowledge of the problem structure, is highly attractive [11]. Numerous hardware solutions have been proposed in the literature to improve the computational speed of the Boltzmann machine. For example, a recent proposal introduces a hybrid hardware system using a DSP processor and customized function blocks on an FPGA [9]. Kim *et al.* propose a scalable, FPGA-based hardware environment for accelerating the Boltzmann machine [10]. Unlike these accelerators, the proposed memristive Boltzmann machine stores the state of the machine and performs *in situ* state updates directly within the memory arrays, which vastly surpasses earlier approaches to accelerating the Boltzmann machine.

2.2 Processing in Memory

Processing in memory (PIM) aims at reducing data movement by processing data directly on the memory chips. Early proposals on PIM involved random access memories in which the sense amplifiers were connected directly to single-instruction, multiple-data (SIMD) pipelines [19]. A configurable PIM chip was proposed that can operate as a conventional memory or as a SIMD processor for data processing [20]. Active Pages [21] proposes placing microprocessors and reconfigurable logic elements next to the DRAM subarrays for fast processing. Guo *et al.* propose DDR3 compatible DIMMs capable of performing content addressable searches [22] and associate computing [23] on resistive memory chips. Unlike these proposals, the proposed accelerator enables computation within conventional data arrays to achieve the energy-efficient and massively parallel processing required for the Boltzmann machine model.

In addition to digital accelerators, analog processors for specific application domains have been proposed in the literature; for example, Kerneltron [24] realizes a massively parallel mixed-signal VLSI processor suitable for kernel-based real-time video recognition. The chip relies on charge injection devices to perform integer vector-matrix multiplication [25]. In contrast, the proposed memristive accelerator is designed for optimization and learning tasks using the Boltzman machine. The key idea is to exploit the electrical properties of the conventional 1T-1R RRAM cell to perform a three-operand multiplication involving the state variables and the weights. These operations are then supplemented with efficient reduction techniques to update the weights and the machine state.

2.3 Resistive Switching

The resistive switching effect has been observed in a wide range of materials such as perovskite oxide (*e.g.*, $SrZrO_3$, $LiNbO_3$, $SrTiO_3$), binary metal oxide (*e.g.*, NiO, CuO_2, TiO_2, HfO_2), solid electrolytes (*e.g.*, AgGeS, CuSiO), and certain organic materials [26]. Resistive RAM (RRAM) is one of the most promising memristive devices under commercial development; RRAM exhibits excellent scalability (<10*nm*) [27, 28], high-speed switching (<1*ns*) [29, 30], low power consumption (<1*pJ*) [31, 32], a high endurance (>10^{12} writes) [33, 34], and a high dynamic resistance range ($\frac{R_{HI}}{R_{LO}}$>10^5) [35, 36]. Figure 4 illustrates an example RRAM cell comprising an access transistor and a resistive switching medium. The content of the switching medium is read or written by applying electrical signals through the two vertical bitlines (shown as $\overline{Bitline}$ and $Bitline$).

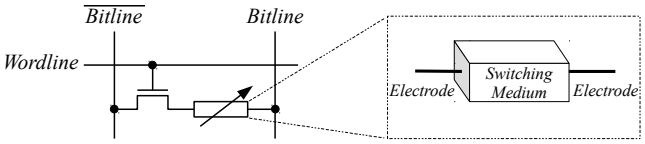


Figure 4: Illustration of an RRAM cell.

The proposed accelerator exploits the electrical properties of parallel RRAM cells connected to a single bitline to compute dot products within the memory arrays. In theory, this could be accomplished with any memory technology capable of switching between two resistive states (R_{HI} and R_{LO}). However, memory cells with a limited dynamic resistance range (*e.g.*, Magnetoresistive RAM [37]) would require complex sensing mechanisms and limit the functionality. In contrast, the proposed accelerator benefits from the large difference between R_{HI} and R_{LO} in RRAM cells to enable efficient *in situ* computation within the memory arrays.

2.4 Neuromorphic Networks

Neuromorphic computing, which leverages connectionist models inspired by the human brain [38], is ill suited to von Neumann architectures. As a result, energy efficient analog accelerators based on memristive circuits have been explored in the literature [39, 40]. In a typical memristive neural circuit, a memristor acts as a synapse whose weight (*i.e.*, conductance) can be changed by an electrical signal [41].

Sheri *et. al.* propose a spiking neural network based on memristive synapses that implements a single step contrastive divergence algorithm for machine learning [42]. Each synapse comprises two memristors representing limited-precision positive weights. Prezioso *et al.* report the fabrication of a memristive single-level perceptron with ten inputs and three outputs, which can be used to classify 3×3 black and white images [43]. They optimize the fabrication process of an existing RRAM cell to reduce device variability, thereby eliminating the need for access transistors. Although they demonstrated the feasibility of fully *in situ* computation for 3×3 bit image classification, it is not clear how the proposed technique would scale to larger problems. Unlike prior work, the proposed memristive Boltzmann machine is a scalable framework that can be used for solving a wide range of combinatorial optimization and deep learning problems.

3. THE PROPOSED MEMRISTIVE BOLTZMANN MACHINE

The proposed memristive Boltzmann machine is an RRAM based hardware platform capable of accelerating combinatorial optimization and neural computation tasks. The key idea is to exploit the electrical properties of the storage cells and the interconnections among those cells to compute the dot product—the fundamental building block of the Boltzmann machine—*in situ* within the memory arrays. This novel capability of the proposed memristive arrays eliminates unnecessary latency, bandwidth, and energy overheads associated with streaming the data out of the memory arrays during the computation process. A high-level system overview and the fundamental operating principles of the proposed accelerator are discussed herein.

3.1 System Overview

The proposed accelerator resides on the memory bus, and interfaces to a general-purpose computer system via the DDRx interface (Figure 5). This modular organization permits selectively integrating the accelerator in systems that execute combinatorial optimization and machine learning workloads.

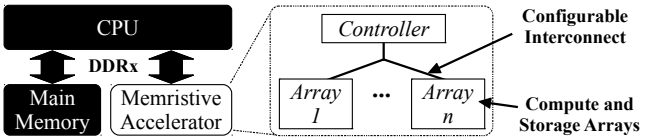


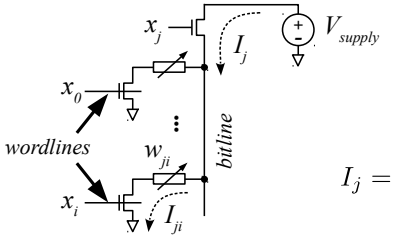
Figure 5: System overview.

The memristive Boltzmann machine comprises a hierarchy of data arrays connected via a configurable interconnection network. A controller implements the interface between the accelerator and the processor. The data arrays are capable of storing the connection weights (w_{ji}) and the state variables (x_i); it is possible to compute the product of a weight and two state variables ($x_j x_i w_{ji}$) *in situ* within the data arrays. The interconnection network permits retrieving and summing these partial products to compute the energy change ΔE associated with a potential state update, and ultimately sends the ΔE results to the controller. Given the energy change that results from a potential update, the controller probabilistically decides whether to accept that update based on the Boltzmann distribution.

3.2 In Situ Computation

The critical computation that is performed by the Boltzmann machine consists of multiplying a weight matrix **W** by

a state vector **x**. Every entry of the symmetric matrix **W** (w_{ji}) records the weight between two units (units j and i); every entry of the vector **x** (x_i) stores the state of a single unit (unit i). Figure 6 depicts the fundamental concept behind the design of the memristive Boltzmann machine. In the figure, the weights and the state variables are respectively represented using memristors and transistors. A constant voltage supply (V_{supply}) is connected to parallel memristors through a shared vertical bitline. The total current pulled from the supply voltage represents the result of the computation. This current (I_j) is set to zero when x_j is OFF; otherwise, the current is equal to the sum of the currents pulled by the individual cells connected to the bitline. Due to the constant voltage applied across all of the parallel branches, the current pulled by each cell is determined by V_{supply}, the state of the transistor x_i, and the conductance (*i.e.*, $\frac{1}{resistance}$) of the memristive element w_{ji}. For simplicity, we assume $V_{supply} = 1V$; therefore, the magnitude of this current represents the product $x_j x_i w_{ji}$, which is the same as the link energy of the Boltzmann machine (Equation 1).[2]



$$I_j = \sum_{i=0} I_{ji} = \sum_{i=0} x_j x_i w_{ji}$$

Figure 6: The key concept.

4. FUNDAMENTAL BUILDING BLOCKS

The fundamental building blocks of the proposed memristive Boltzmann machine are (1) storage elements, (2) a current summation circuit, (3) a reduction unit, and (4) a consensus unit. The design of these hardware primitives must strike a careful balance among multiple goals: high memory density, low energy consumption, and *in situ*, fine-grained parallel computation.

4.1 The Storage Elements

As mentioned in Section 2.1, every instance of the Boltzmann machine can be represented by a matrix **W**, comprising the connection weights, and a vector **x** consisting of the current binary states of the processing units. The weight matrix is iteratively accessed to update the state variables until convergence is reached. This iterative process requires all-to-all communication among the processing units, which results in excessive memory traffic and significantly limits the overall performance. These data movement overheads become even more pronounced in large scale Boltzmann machines.

To alleviate the energy and performance overheads of the data movement, this paper (1) decreases the distance over which the data are moved by employing dense memory structures, and (2) reduces the amount of data transferred among the storage cells and the processing units by enabling *in situ* computation within the memory arrays.

[2]SPICE simulations are conducted to accurately model the behavior of the transistors, memristive elements, and parasitic resistances of the bitlines (Section 6.2).

A conventional 1-transistor, 1-memristor (1T-1R) array is employed to store the connection weights (the matrix **W**), while the relevant state variables (the vector **x**) are kept close to the data arrays holding the weights (Figure 7). The memristive 1T-1R array is used for both storing the weights, and for computing the dot product between these weights and the state variables. During the dot product computation, the state variables are used to enable the corresponding wordlines and bitlines.

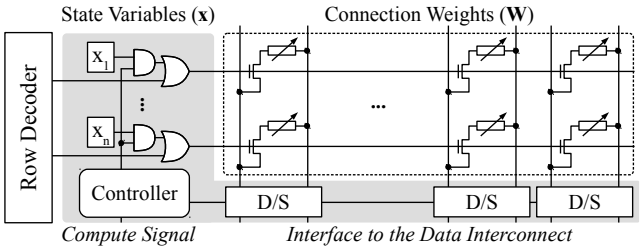


Figure 7: The proposed array structure.

4.1.1 Computing within the Data Array

During a dot product computation, the wordlines and the bitlines of the memristive array are selectively activated according to the vector **x**. The x_js are used to enable the bitline drivers, while the wordlines are controlled by the x_is.[3] As a result of this organization, the content of a memristive cell—representing a single bit of the connection weight w_{ji}—is accessed only if both x_j and x_i are set to one. This results in a primitive *bit product* operation, $x_j \cdot x_i \cdot w_{ji}$, which is supplemented with column summation to compute the machine energy (Equation 1).

4.1.2 Updating the State Variables

At every step of the energy optimization (Equation 1), each processing unit employs a probabilistic model to update its own state based on the states of its neighbors. As a result, updating the state variables is crucial to the system energy and performance. Moreover, high quality solutions can be found within a limited time only if one of the units connected to each active link[4] updates its state [11]. Selectively updating the state variables, however, generates extra memory traffic, which limits the performance and energy efficiency.

To minimize the overhead of data movement due to state updates, static CMOS latches are used to store the state variables at the periphery of the memristive data arrays. In addition to *in situ* dot product computation, this physical organization is employed to obtain all of the state variables that may flip simultaneously. A data array is used to represent an incidence matrix **B** corresponding to **W**, where b_{ji} is set to 1 if $w_{ji} \neq 0$, and to 0 otherwise. Due to the computational capability of the data array, reading row i from the array results in computing $x_i \cdot x_j \cdot b_{ij}$, which determines all of the active rows connected to unit i. This capability is employed to

[3]The sensing units typically are time multiplexed among multiple bitlines to amortize their high energy and area costs; without loss of generality, the degree of multiplexing is assumed one here.

[4]An active link is a non-zero edge between two units i and j, where $x_i = x_j = 1$.

speedup state updates in optimization problems and weight updates in deep learning tasks.

4.1.3 Storing the Connection Weights

Reading and writing the connection weights involves activating a single wordline and sensing the voltage on the corresponding bitlines. All of the vector control circuits (*i.e.*, the gray area of Figure 7) need to be bypassed during a read or write access. This is accomplished using a control signal (*compute*) from the controller that indicates whether a dot product computation or an ordinary data access is requested. Unlike the binary state variables, the connection weights are represented in fixed point, two's complement format (Section 5.1.2).

4.2 The Current Summation Circuit

The result of a dot product computation can be obtained by measuring the aggregate current pulled by the memory cells connected to a common bitline. Computing the sum of the bit products requires measuring the total amount of current per column and merging the partial results into a single sum of products. This is accomplished by local column sense amplifiers and a bit summation tree at the periphery of the data arrays.

4.2.1 The Column Sense Amplifier

The column sense amplifier quantizes the total amount of current pulled through a column into a multi-bit digital value. This is equivalent to counting the number of ones within a selected column, and is accomplished by a successive approximation mechanism [44] using parallel sample and hold (S/H) units (Figure 8).

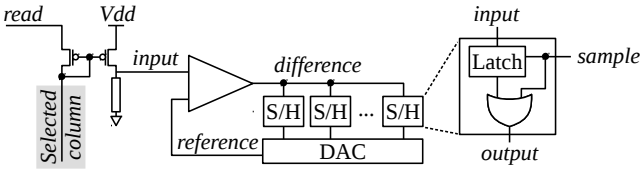


Figure 8: Column sensing circuitry.

Each S/H unit comprises a latch for holding the data, and an OR gate for sampling. A current mirror produces an amplified copy of the current pulled by the column, which is then converted to an input voltage using a pull-down load, and subsequently is fed into a differential amplifier [45]. The differential amplifier employs a reference voltage to output a one-bit value indicating whether the sensed voltage is greater than the reference voltage. As a result, a single bit of the final quantized result is obtained on every comparison. The reference voltage is generated by a digital-to-analog converter (DAC) [46]. The proposed summation circuit is used either to compute the sum of the bit products, or to read a single cell. When used for computing the sum of the bit products, the number of cells attached to each column determines the precision required for the summation circuit.[5] We therefore limit the number of rows in each data array, and explore a hierarchical summation technique based on local sense amplifiers and a novel reduction tree.

[5] A detailed discussion on the impact of precision on the fidelity of the optimization results is provided in Section 5.1.

4.2.2 The Bit Summation Tree

A bit summation unit merges the partial sums generated by the column sense amplifiers (Figure 9). The sum is serially transmitted upstream through the data interconnect as it is produced. Multiple bit summation trees process the columns of the array in parallel. For example, each row of a 512×512 data array can contain 16 words, each of which represents a 32-bit connection weight (w_{ji}). Every group of 32 bitlines forms a *word column* and connects to the connection weights across all of the rows. The goal is to compute the sum of the products for each word column. All of the bitlines within a word column are concurrently quantized into 16 ($512/32$) partial sums, which are then merged to produce a single sum of products.

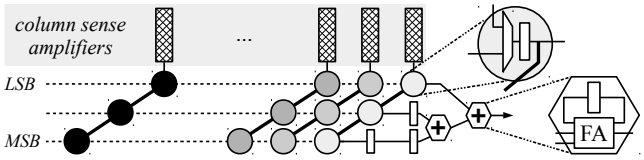


Figure 9: The proposed bit summation circuit.

Design Challenges. One challenge in designing the column sensing circuit is the precision of the current summation, which is affected by variability, noise, and parasitics. Although the stochastic nature of the Boltzmann machine goes a long way toward tolerating inaccuracy, a large machine would still require more efficient techniques to become viable. This paper proposes a hierarchical approach to computing the dot product of very large matrices and state vectors.

4.3 The Reduction Unit

To enable processing large matrices using multiple data arrays, an efficient data reduction unit is employed. The reduction units are used to build a *reduction network*, which sums the partial results as they are transferred from the data arrays to the controller. Large matrix columns are partitioned and stored in multiple data arrays, where the partial sums are individually computed. The reduction network merges the partial results into a single sum. Multiple such networks are used to process the weight columns in parallel. The reduction tree comprises a hierarchy of bit-serial adders to strike a balance between throughput and area efficiency.

Figure 10 shows the proposed reduction. The column is partitioned into four segments, each of which is processed separately to produce a total of four partial results. The partial results are collected by a reduction network comprising three bi-modal reduction elements. Each element is configured using a local latch that operates in one of two modes: *forwarding*, and *reduction*. A full adder is employed by each reduction unit to compute the sum of the two inputs when operating in the reduction mode. In the forwarding mode, the unit is used for transferring the content of one input upstream to the root. This reduction unit is used to implement efficient bank-level H-trees (Section 5.2).

4.4 The Consensus Unit

The next state of the processing units is determined by a set of *consensus units* based on the final energy change computed by the reduction tree. Recall from Section 2.1

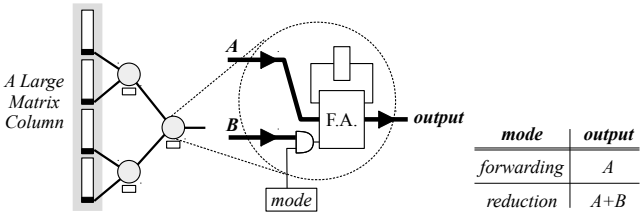


Figure 10: Illustration of the reduction element.

that the Boltzmann machine relies on a sigmoidal activation function, which plays a key role in both the optimization and the machine learning applications of the model. A precise implementation of the sigmoid function, however, would introduce unnecessary energy and performance overheads. As shown in prior work [47, 48], reduced complexity hardware—relying on subsampling and linear or superlinear approximation—can meet high performance and energy efficiency requirements at the cost of a negligible loss in precision. The proposed memristive accelerator employs an approximation unit using logic gates and lookup tables to implement the consensus function (Figure 11).

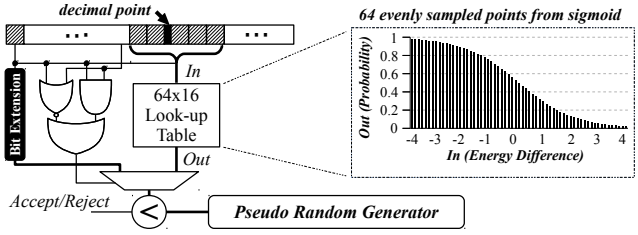


Figure 11: The proposed unit for the activation function.

The table contains 64 precomputed sample points of the sigmoid function $f(x) = \frac{1}{1+e^x}$, where x varies between -4 and 4. The samples are evenly distributed on the x axis. Six bits of a given fixed point value are used to index the lookup table and retrieve a sample value. The most significant bits of the input data are ANDed and NORed to decide whether the input value is outside the domain $[-4,4]$; if so, the sign bit is extended to implement $f(x) = 0$ or $f(x) = 1$; otherwise, the retrieved sample is chosen as the outcome.

5. SYSTEM ARCHITECTURE

Figure 12 shows the hierarchical organization of the memristive Boltzmann machine, which comprises multiple banks and a controller. The banks operate independently, and serve memory and computation requests in parallel. For example, column 0 can be multiplied by the vector **x** at bank 0 while a particular address of bank 1 is read. Within each bank, a set of subbanks is connected to a shared interconnection tree. The bank interconnect is equipped with reduction units to contribute to the dot product computation. In the reduction mode, all subbanks actively produce the partial results, while the reduction tree selectively merges the results from a subset of the subbanks. This capability is useful for computing the large matrix columns partitioned across multiple subbanks. Each subbank consists of multiple mats, each of which is composed of a controller and multiple data arrays. The subbank tree transfers the data bits between the mats and the bank tree in a bit-parallel fashion, thereby increasing the parallelism.

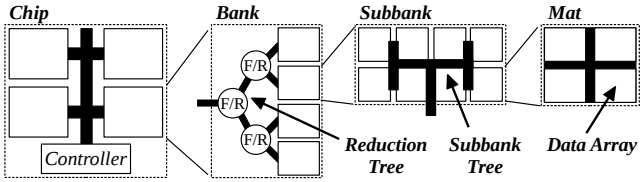


Figure 12: Hierarchical organization of a chip.

5.1 Array Organization

The data array organization is crucial to the energy and performance of the memristive accelerator.

5.1.1 Data Organization

To amortize the cost of the peripheral circuitry, the columns and the rows of the data array are time shared. Each sense amplifier is shared by four bitlines. The array is vertically partitioned along the bitlines into 16 stripes, multiples of which can be enabled per array computation. This allows the software to keep a balance between the accuracy of the computation and the performance for a given application by quantizing more bit products into a fixed number of bits.

5.1.2 Data Representation

In theory, the Boltzmann Machine requires performing computation on binary states and real-valued weights. Prior work, however, has shown that the Boltzmann machine can still solve a broad range of optimization and machine learning problems with a negligible loss in solution quality when the weights are represented in a fixed-point, multi-bit format [49, 50]. Nevertheless, we expect that storing a large number of bits within each memristive storage element will prove difficult [51, 37].

One solution to improve the accuracy is to store only a single bit in each RRAM device, and to spread out a single scalar multiplication over multiple one-bit multiplications (Section 4.1). The weights are represented in two's complement format. Each compute operation results in a partial sum, which is serially transferred over a single data wire. If $x_j = 0$ (Equation 3), the partial sums are multiplied by -1 using a serial bit negator comprising a full adder and an XOR gate.

5.2 Bank Organization

Each bank is able to compute the dot products on its own data and update the corresponding state variables independently.[6] This is accomplished by a consensus unit at each bank. To equalize the access latency to the subbanks within each bank, the bank interconnect is organized as an H-tree. A fixed subset of the H-tree output wires is equipped with the reduction units to form a reduction tree (Section 4.3). At every node of the reduction H-tree, a one-bit flag is used to determine the operational mode. These flags form a programmable reduction chain for each bank. Prior to solving an optimization or machine learning problem, a fixed length *reduction pattern* is generated by the software and serially loaded into the chain. For example, a reduction tree connected to 1024 subbanks would require 1023 cycles to pro-

[6] A minimal data exchange among the banks is coordinated by the chip controller to perform the necessary state updates.

gram all of the flags.[7] The same reduction pattern is applied to all of the banks in parallel.

Regardless of the problem type, the reduction pattern only depends on the number of units and connection weights in the Boltzmann machine. Every reduction pattern is specifically generated for an input problem based on (1) the maximum number of partial sums that can be merged by the bank reduction tree (Λ), and (2) the problem size in terms of the number of required partial sums to be merged per each computation (Γ). Figure 13 depicts how the reduction pattern is generated when Λ is eight and Γ is five. A binary tree is used where each leaf represents a partial sum. Each leaf is marked with one if its partial sum contributes to the aggregate sum, and with a zero otherwise. These values are propagated to the root by applying a logical AND at each intermediate node: a node is set to one if at least one of the right children *and* one of the left children are set to one. Note that the reduction pattern generation is a one time process performed by the software prior to configuring the accelerator.

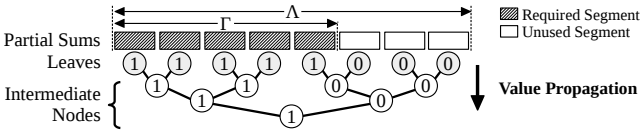


Figure 13: Generating an example reduction pattern for $\Lambda = 8$ and $\Gamma = 5$.

5.3 On-chip Control

The proposed hardware is capable of accelerating optimization and deep learning tasks by appropriately configuring the on-chip controller. The controller (1) configures the reduction trees, (2) maps the data to the internal resources, (3) orchestrates the data movement among the banks, (4) performs annealing or training tasks, and (5) interfaces to the external bus.

Configuring the Reduction Tree. Software generates the reduction pattern and writes to specific registers in the accelerator; the chip controller then loads the data bits into the flag chains.

Address Remapping. The key to efficient computation with the proposed accelerator is the ability to merge a large fraction (if not all) of the partial sums for a single column of the weight matrix. This is made possible by a flexible address mapping unit that is programmed based on the problem size. For an $m \times n$ weight matrix, the software has to stream the weights into the chip in a column major format.[8] When initializing the accelerator chip with the weight matrix, an internal counter keeps track of the number of transferred blocks, which is used to find the destination row and column within an internal data array. The least significant bits of the counter are used to determine the subbank and row IDs, while the rest of the bits identify the mat, stripe, column, and bank IDs. (Zero padding is applied to the half full stripes within each data array.) As a result of this internal address remapping, an external stream of writes is evenly distributed among the subbanks regardless of the original addresses.

Synchronizing the States. Due to the internal address remapping, weights and states are stored at predefined locations, and the control process is significantly simplified. Each bank controller—comprising logic gates and counters—synchronizes computation within the subbanks, collects the results, and updates the state variables. During an optimization process, compute commands are periodically sent to the subbanks such that the gap between consecutive commands guarantees the absence of conflicts on the output bus. The arrays and the reduction trees produce and send the results to the bank controller. A consensus unit is employed to compute the next states as the results arrive. Each 1-bit state variable is then transferred to the other banks. In a 64-bank chip, a bank controller receives up 63 state bits from the other bank controllers. The state variables are then updated via the input wires of the subbanks.

Annealing and Training. To perform an annealing (for optimization) or a training (for learning) task, iterative update mechanisms are implemented at the chip controller. When training the accelerator, the state variables are transferred among the banks at every training epoch; subsequently, the weights are computed and written to the arrays. At every iteration of the annealing schedule, a new temperature is sent to all of the banks. An internal register stores the current temperature, which is set to the initial temperature (α) at the beginning of an optimization task. A user-defined annealing factor (β) is applied to the current temperature using an integer multiplier and an arithmetic shifter.

Interfacing. The interface between CPU and the accelerator is required for (1) configuring the chip, (2) writing new weights or states, and (3) reading the outcome. Prior to a data transfer, software must configure the device by selectively writing to a set of *control registers*. All of the data transfers take place through a set of *data buffers*. Along the lines of prior work by Guo *et al*. [22, 23], both configuration and data transfer accesses are performed by ordinary DDRx reads and writes. This is made possible because (1) direct external accesses to the memory arrays are not allowed, and (2) all accesses to the accelerator are marked as strong-uncacheable [52, 53] and processed in-order. When writing the weights and states, the internal address remapping unit guarantees a uniform distribution of write accesses among the subbanks. As a result, consecutive external writes to a subbank are separated by at least 64 writes. This is a sufficiently wide gap that allows an ongoing write to complete before the next access. After transferring the weights, the accelerator starts computing. The completion of the process is signaled to the software by setting a *ready* flag. The outcome of the computation is read from specific locations on the accelerator by the software.

5.4 DIMM Organization

To solve large-scale optimization and machine learning problems whose state space does not fit within a single chip, it is possible to interconnect multiple accelerators on a DIMM [54]. Each DIMM is equipped with control registers, data buffers, and a controller. This controller receives DDRx commands, data, and address bits from the external interface, and or-

[7] The programming cost of the flags is modeled in all of the performance and energy evaluations (Section 7).

[8] This data transfer is accurately modeled in the evaluation.

chestrates computation among all of the chips on the DIMM. Software initiates the computation by writing the configuration parameters to the control registers.

5.5 Software Support

To make the proposed accelerator visible to software, its address range is memory mapped to a portion of the physical address space. A small fraction of the address space within every chip is mapped to an internal RAM array, and is used for implementing the data buffers and the configuration parameters. Software configures the on-chip data layout and initiates the optimization by writing to a memory mapped control register. To maintain ordering, accesses to the accelerator are made uncacheable by the processor [52, 53].

6. EXPERIMENTAL SETUP

Circuit, architecture, and application level simulations were conducted to quantify the area, energy, and performance of the proposed accelerator.

6.1 Architecture

We modify the SESC simulator [55] to model a baseline eight-core out-of-order processor. The memristive Boltzmann machine is interfaced to a single-core system via a single DDR3-1600 channel. Table 1 shows the simulation parameters.

Core Type		4-issue cores, 3.2 GHz, 176 ROB entries
Cache	**Instruction L1**	32KB, direct-mapped, 64B block, hit/miss: 2/2
	Data L1	32KB, 4-way, LRU, 64B block, hit/miss: 2/2, MESI
	Shared L2	8MB, 16-way, LRU, 64B block, hit/miss: 15/12
DRAM	**Memory Configuration**	8KB row buffer, 8Gb DDR3-1600 chips, Channels/Ranks/Banks: 4/2/8
	Timing (DRAM cycles)	^{t}RCD: 11, ^{t}CL: 11, ^{t}WL: 5, ^{t}CCD: 4, ^{t}WR: 12, ^{t}RP: 11, ^{t}RC: 39, ^{t}WTR: 6, ^{t}RTP: 6, ^{t}RRD: 5, ^{t}RAS: 28, ^{t}BURST: 4, ^{t}FAW: 32
Memristive Boltzmann Machine		Channels/Chips/Banks/Subbanks: 1/8/64/64, 1Gb DDR3-1600 compatible chips, ^{t}Read: 4.4ns, ^{t}Write: 52.2ns, ^{t}Update: 3.6ns, ^{v}Read: 0.8V, ^{v}Write: 1.3V, ^{v}Update: 0.8V

Table 1: Simulation parameters.

We develop an RRAM based PIM baseline. The weights are stored within data arrays that are equipped with integer and binary multipliers to perform the dot products. The proposed consensus units, optimization and training controllers, and mapping algorithms are employed to accelerate the annealing and training processes. When compared to existing computer systems and GPU-based accelerators, the PIM baseline can achieve significantly higher performance and energy efficiency because it 1) eliminates the unnecessary data movement on the memory bus, 2) exploits data parallelism throughout the chip, and 3) transfers the data across the chip using energy efficient reduction trees. The PIM baseline is optimized so that it occupies the same area as that of the memristive accelerator.

6.2 Circuits

We model the data array, sensing circuits, drivers, local array controller, and interconnect elements using SPICE predictive technology models [56] of NMOS and PMOS transistors at 22nm. Circuit simulations are conducted using Cadence (SPECTRE) [57] to estimate the area, timing, dynamic energy, and leakage power. (The parasitic resistance and capacitance of the wordlines and bitlines are modeled based on the interconnect projections from ITRS [37]). We use NVSim [58] with resistive memory parameters ($R_{LO} = 315K$ and $R_{HI} = 1.1G$) based on prior work [35] to evaluate the area, delay, and energy of the data arrays. The full adders, latches, and the control logic are synthesized using the Cadence Encounter RTL Compiler [59] with FreePDK [60] at 45nm. The results are first scaled to 22nm using scaling parameters reported in prior work [61], and are then scaled using the FO4 parameters for ITRS LSTP devices to model the impact of using a memory process on peripheral and global circuitry [62, 63]. The current summation circuit is modeled following a previously proposed methodology [64, 58] and is optimized for quantizing 32 rows per stripe when 10% resistance variation is considered for the memory cells. Since the RRAM cells require a write voltage higher than the core Vdd, we modeled the design of a charge pump circuit [65, 66] at the 22nm technology node to obtain the relevant area, power and delay parameters used in NVSim. All SRAM units for the lookup tables and data buffers are evaluated using CACTI 6.5 [67]. We use McPAT [68] to estimate the processor power.

6.3 Applications

We develop a software kernel that provides the primitives for building Boltzmann machines. We use geometric annealing schedules with $\alpha = max\{\sum_j |w_{ij}|\}$ and $\beta = 0.95$ for Max-Cut, and $\alpha = \frac{\sum_{i,j}|w_{ij}|}{2N}$ and $\beta = 0.97$ for Max-SAT. We set the annealing process to terminate when the temperature reaches zero and no further energy changes are accepted [12, 69]. The kernel supports both single and multi-threaded execution.

6.4 Data Sets

We select ten matrices used for graph optimization from the University of Florida collection [70] to solve the Max-Cut problem. We use ten instances of the satisfiability problem in circuit and fault analysis [71, 72] to evaluate Max-SAT. On deep learning applications, a set of 400 grayscale images of size 64×64—from the Olivetti database at ATT [73]—are used to train a four-layer deep belief net (similar to [14]).[9] Table 2 shows the specifications of the workloads.

Max-Cut	MC-1: bp_0(822×3275)[§] MC-2: cage(366×2562) MC-3: can_838(838×4586) MC-4: cegb2802(2802×137334) MC-5: celegans_metabolic(453×2025) MC-6: dwt_992(992×7876) MC-7: G50(3000×6000) MC-8: netscience(1589×2742) MC-9: str_0(363×2452) MC-10: uk(4824×6837)
Max-SAT	MS-1: ssa0432-003(435×1027)[†] MS-2: f600(600×2550) MS-3: ssa2670-141(986×2315) MS-4: f1000(1000×4250) MS-5: ssa7552-160(1391×3126) MS-6: bf2670-001(1393×3434) MS-7: ssa7552-038(1501×3575) MS-8: f2000(2000×8500) MS-9: bf1355-638(2177×4768) MS-10: bf1355-075(2180×6778)
ML	DBN-1: (1024×256×64×16)[‡] DBN-2: (2048×512×128×32) DBN-3: (4096×1024×256×64) DBN-4: (8192×2048×512×128)

[§] (nodes×edges); [†] (variables×clauses); [‡] the number of hidden units (layer1×layer2×layer3×layer4)

Table 2: Workloads and input datasets.

6.5 Baseline Systems

We choose state of the art software approximation algorithms for benchmarking. We use a semi-definite programming (SDP) solver [75] for solving the Max-Cut problem. We

[9] We assume mini-batches of size 10 for training [74].

also use MaxWalkSat [76], a non-parametric stochastic optimization framework, as the baseline for the maximum satisfiability problem. These baseline algorithms are used for evaluating the quality of the solutions found by the proposed accelerator.

7. EVALUATION

This section presents the area, delay, power, and performance characteristics of the proposed system.

7.1 Area, Delay, and Power Breakdown

Figure 14 shows a breakdown of the compute energy, leakage power, compute latency, and the die area among different hardware components. The sense amplifiers and interconnects are the major contributors to the dynamic energy (41% and 36%, respectively). The leakage is mainly caused by the current summation circuits (40%) and other logic (59%), which includes the charge pumps, write drivers, and controllers. The computation latency, however, is mainly due to the interconnects (49%), the wordlines, and the bitlines (32%). Notably, only a fraction of the memory arrays need to be active during a compute operation. A subset of the mats within each bank perform current sensing of the bitlines; the partial results are then serially streamed to the controller on the interconnect wires. The experiments indicate that a fully utilized accelerator chip consumes 1.3W, which is below the peak power rating of a standard DDR3 chip (1.4W [77, 78]).[10]

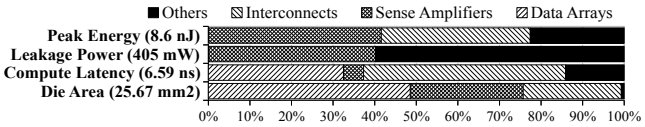


Figure 14: Area, delay, and power breakdown.

7.2 Performance

Figure 15 shows the performance on the proposed accelerator, the PIM architecture, the multicore system running the multi-threaded kernel, and the single core system running the SDP and MaxWalkSAT kernels. The results are normalized to the single-threaded kernel running on a single core. The results indicate that the single-threaded kernel (Boltzmann machine) is faster than the baselines (SDP and MaxWalkSAT heuristics) by an average of 38%. The average performance gain for the multi-threaded kernel is limited to 6% due to significant state update overheads (Section 4.1.2). PIM outperforms the single-threaded kernel by 9.31×. The memristive accelerator outperforms all of the baselines (57.75× speedup over the single-threaded kernel, and 6.19× over PIM). Moreover, the proposed accelerator performs the deep learning tasks 68.79× faster than the single-threaded kernel and 6.89× faster than PIM (Figure 16).

7.3 Energy

Figure 17 shows the energy savings as compared to PIM, the multi-threaded kernel, SDP, and MaxWalkSAT. On average, energy is reduced by 25× as compared to the single-threaded kernel implementation, which is 5.2× better than

[10]If necessary, power capping mechanisms may be employed on the chip to further limit the peak power consumption.

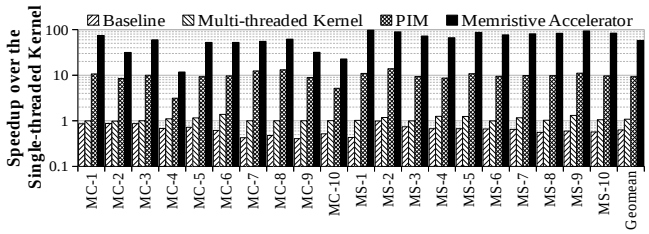


Figure 15: Performance on optimization.

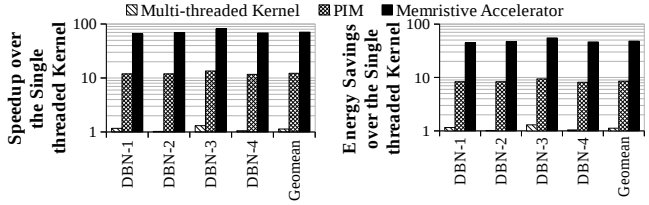


Figure 16: Performance on deep learning.

PIM. For the deep learning tasks, the system energy is improved by 63×, which is 5.3× better than the energy consumption of PIM.

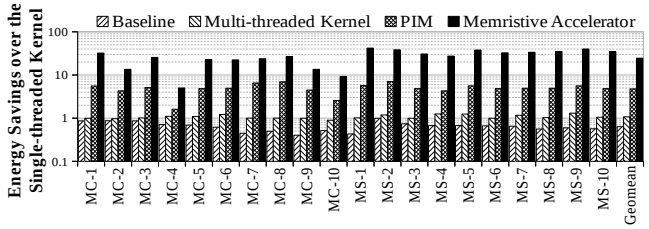


Figure 17: Energy savings on optimization.

7.4 Solution Quality

We evaluate the quality of the solutions and analyze the impact of various causes of imprecision.

7.4.1 Quality of the Optimization

The objective function used in evaluating the quality of a solution is specific to each optimization problem. For Max-Cut, the objective function is the maximum partitioning cost found by the algorithm; in contrast, Max-SAT searches for the largest number of satisfiable clauses. We evaluate the quality of the optimization procedures run on different hardware/software platforms by normalizing the outcomes to that of the corresponding baseline heuristic (Figure 18). The average quality figures of 1.31× and 0.96× are achieved, respectively, for Max-Cut and Max-SAT when running on the proposed accelerator. Therefore, the overall quality of the optimization is 1.11×.

7.4.2 Limited Numerical Precision

One limitation of the proposed accelerator is the reduced precision due to the fixed point representation. This limitation, however, does not impact the solution quality significantly. We observed that a 32-bit fixed point representation causes a negligible degradation (<1%) in the outcome of the optimization process and the accuracy of the learning tasks. This result confirms similar observations reported in prior work [49, 50].

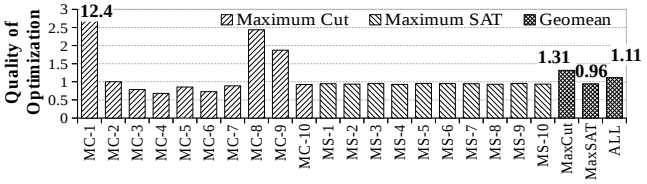


Figure 18: Outcome quality.

7.4.3 *Sensitivity to Process Variations*

Memristor parameters may deviate from their nominal values due to process variations caused by line edge roughness, oxide thicknes fluctuation, and random discrete doping [79]. These parameter deviations result in *cycle-to-cycle* and *device-to-device* variabilities. We evaluate the impact of cycle-to-cycle variation on the outcome of the computation by considering a bit error rate of 10^{-5} in all of the simulations, along the lines of the analysis provided in prior work [80, 81]. The proposed accelerator successfully tolerates such errors, with less than 1% change in the outcome as compared to a perfect software implementation.

The resistance of RRAM cells may fluctuate because of the device-to-device variation, which can impact the outcome of a column summation—*i.e.*, a partial dot product. We use the geometric model of memristance variation proposed by Hu *et al.* [82, 83] to conduct Monte Carlo simulations for 1 Million columns, each comprising 32 cells. The experiment yields normal distributions for R_{LO} and R_{HI} samples with respective standard deviations of 2.16% and 2.94%. We then find a bit pattern that results in the largest summation error for each column. Figure 19 shows the distribution of conductance values for the ideal and sample columns, as well as the cumulative distribution (CDF) of the conductance deviation. We observe up to 2.6×10^{-6} deviation in the column conductance, which may result in up to 1 bit error per summation. Subsequent simulation results indicate that the accelerator can tolerate this error, with less than 2% change in the outcome quality.

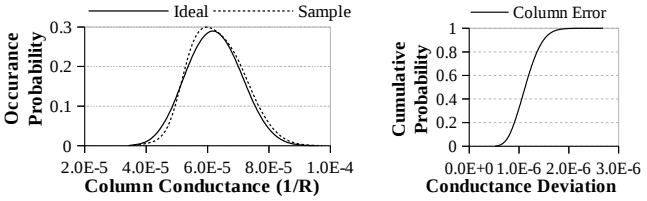


Figure 19: Process variation.

7.4.4 *Finite Switching Endurance*

RRAM cells exhibit finite switching endurance ranging from 10^6 to 10^{12} writes [35, 36, 34]. We evaluate the impact of finite endurance on the lifetime of an accelerator module. Since wear is induced only by the updating of the weights stored in memristors, we track the number of times that each weight is written. The edge weights are written once in optimization problems, and multiple times in deep learning workloads. (Updating the state variables, stored in static CMOS latches, does not induce wear on RRAM.) We track the total number of updates per second to estimate the lifetime of an eight-chip DIMM. Assuming endurance parameters of 10^6 and 10^8 writes [35], the respective module lifetimes are 3.7 and 376 years for optimization, and 1.5 and 151 years for deep learning.

7.5 Discussion

This section explains several practical constraints when using the proposed accelerator.

Problem Size. The proposed accelerator is capable of processing Boltzmann machines with at least two units, although not all problems can be solved efficiently. Figure 20 shows the speedups achieved over the multi-threaded kernel by PIM and the proposed accelerator as the number of units varies from two to 256. The proposed accelerator outperforms PIM for all problem sizes; however, due to the excessive initialization time, the multi-threaded kernel achieves higher optimization speed on small problems (<20 units).

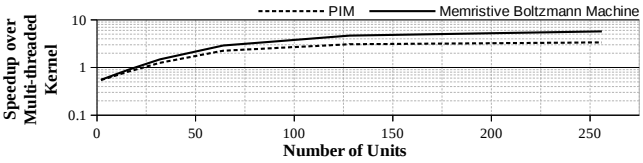


Figure 20: Sensitivity to the problem size.

Interfacing to the CPU. A host CPU emits control commands to the accelerator through an API on behalf of a user application. First, memory is allocated on the accelerator and the required data is transferred from main memory. The accelerator is then configured for the problem and the optimization begins. Finally, the outcome of the optimization is read by the processor from the local buffers of the accelerator. The communication interface between the accelerator and the CPU consumes 5% of the execution time.

8. CONCLUSIONS

The Boltzmann machine is an important type of model used for solving hard optimization and learning problems. It demands massively parallel computation at a very fine granularity. Unlike existing solutions, the proposed accelerator enables *in situ* computation within conventional RRAM arrays by exploiting the natural electrical properties of the RRAM cells. Novel control techniques and configurable interconnects eliminate unnecessary latency, bandwidth, and energy overheads associated with streaming the data out of the memory arrays during the computation process. We conclude that the proposed system exhibits significant potential for improving the performance and energy efficiency of large scale combinatorial optimization and deep learning tasks.

9. ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for useful feedback. This work was supported in part by NSF grant CCF-1533762.

10. REFERENCES

[1] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[2] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1 ed., 1997.

[3] I. Wegener, "Simulated annealing beats metropolis in combinatorial optimization," *Electronic Colloquium on Computational Complexity*, 2004.

[4] S. E. Fahlman, G. E. Hinton, and T. J. Sejnowski, "Massively parallel architectures for AI: NETL, Thistle, and boltzmann machines," in *Proceedings of Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 109–113, 1983.

[5] D. L. Ly, V. Paprotski, and D. Yen, "Neural networks on gpus: Restricted boltzmann machines," *see http://www. eecg. toronto. edu/˜ moshovos/CUDA08/doku. php*, 2008.

[6] Y. Zhu, Y. Zhang, and Y. Pan, "Large-scale restricted boltzmann machines on single gpu," in *Big Data, 2013 IEEE International Conference on*, pp. 169–174, Oct 2013.

[7] D. L. Ly and P. Chow, "High-performance reconfigurable hardware architecture for restricted boltzmann machines.," *IEEE Transactions on Neural Networks*, vol. 21, no. 11, pp. 1780–1792, 2010.

[8] C. Lo and P. Chow, "Building a multi-fpga virtualized restricted boltzmann machine architecture using embedded mpi," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 189–198, 2011.

[9] S. K. Kim, L. McAfee, P. McMahon, and K. Olukotun, "A highly scalable restricted boltzmann machine fpga implementation," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pp. 367–372, Aug 2009.

[10] L.-W. Kim, S. Asaad, and R. Linsker, "A fully pipelined fpga architecture of a factored restricted boltzmann machine artificial neural network," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, pp. 5:1–5:23, Feb. 2014.

[11] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York, NY, USA: John Wiley & Sons, Inc., 1989.

[12] A. d'Anjou, M. Grana, F. Torrealdea, and M. Hernandez, "Solving satisfiability via boltzmann machines," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 5, pp. 514–521, 1993.

[13] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.*, vol. 2, pp. 1–127, Jan. 2009.

[14] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[15] G. E. Hinton, "Learning multiple layers of representation," *Trends in cognitive sciences*, vol. 11, no. 10, pp. 428–434, 2007.

[16] A. Fischer and C. Igel, "An introduction to restricted boltzmann machines," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 14–36, Springer, 2012.

[17] M. Welling and G. E. Hinton, "A new learning algorithm for mean field boltzmann machines," in *Proceedings of the International Conference on Artificial Neural Networks*, ICANN '02, (London, UK, UK), pp. 351–357, Springer-Verlag, 2002.

[18] M. A. Carreira-Perpinan and G. E. Hinton, "On contrastive divergence learning," in *Proceedings of the tenth international workshop on artificial intelligence and statistics*, pp. 33–40, 2005.

[19] D. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie, "Computational ram: Implementing processors in memory," *IEEE Des. Test*, vol. 16, pp. 32–41, Jan. 1999.

[20] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: the terasys massively parallel pim array," *Computer*, vol. 28, pp. 23–31, Apr 1995.

[21] M. Oskin, F. T. Chong, and T. Sherwood, "Active pages: A computation model for intelligent memory," *SIGARCH Comput. Archit. News*, vol. 26, pp. 192–203, Apr. 1998.

[22] Q. Guo, X. Guo, Y. Bai, and E. Ipek, "A resistive tcam accelerator for data-intensive computing," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 339–350, 2011.

[23] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "Ac-dimm: associative computing with stt-mram," in *ACM SIGARCH Computer Architecture News*, pp. 189–200, 2013.

[24] R. Genov and G. Cauwenberghs, "Kerneltron: Support vector 'machine' in silicon.," in *SVM* (S.-W. Lee and A. Verri, eds.), vol. 2388 of *Lecture Notes in Computer Science*, pp. 120–134, Springer, 2002.

[25] R. Genov and G. Cauwenberghs, "Charge-mode parallel architecture for vector-matrix multiplication," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 48, pp. 930–936, Oct 2001.

[26] F. Pan, S. Gao, C. Chen, C. Song, and F. Zeng, "Recent progress in resistive random access memories: materials, switching mechanisms, and performance," *Materials Science and Engineering: R: Reports*, vol. 83, pp. 1–59, 2014.

[27] C. Ho, C.-L. Hsu, C.-C. Chen, J.-T. Liu, C.-S. Wu, C.-C. Huang, C. Hu, and F.-L. Yang, "9nm half-pitch functional resistive memory cell with $<1\mu$a programming current using thermally oxidized sub-stoichiometric wo x film," in *Electron Devices Meeting (IEDM), 2010 IEEE International*, pp. 19–1, IEEE, 2010.

[28] B. Govoreanu, G. Kar, Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. Radu, L. Goux, S. Clima, R. Degraeve, *et al.*, "10× 10nm 2 hf/hfo x crossbar resistive ram with excellent performance, reliability and low-energy operation," in *Electron Devices Meeting (IEDM), 2011 IEEE International*, pp. 31–6, 2011.

[29] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, no. 48, p. 485203, 2011.

[30] B. J. Choi, A. C. Torrezan, K. J. Norris, F. Miao, J. P. Strachan, M.-X. Zhang, D. A. Ohlberg, N. P. Kobayashi, J. J. Yang, and R. S. Williams, "Electrical performance and scalability of pt dispersed sio2 nanometallic resistance switch," *Nano letters*, vol. 13, no. 7, pp. 3213–3217, 2013.

[31] S. Lai, "Current status of the phase change memory and its future," in *Electron Devices Meeting, 2003. IEDM'03 Technical Digest. IEEE International*, pp. 10–1, IEEE, 2003.

[32] C. Cheng, C. Tsai, A. Chin, and F. Yeh, "High performance ultra-low energy rram with good retention and endurance," in *Electron Devices Meeting (IEDM), 2010 IEEE International*, pp. 19–4, IEEE, 2010.

[33] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, *et al.*, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta_2o_{5-x}/tao_{2-x} bilayer structures," *Nature materials*, vol. 10, no. 8, pp. 625–630, 2011.

[34] C.-W. Hsu, I.-T. Wang, C.-L. Lo, M.-C. Chiang, W.-Y. Jang, C.-H. Lin, and T.-H. Hou, "Self-rectifying bipolar tao x/tio 2 rram with superior endurance over 10 12 cycles for 3d high-density storage-class memory," in *VLSI Technology (VLSIT), 2013 Symposium on*, pp. T166–T167, IEEE, 2013.

[35] C. Cheng, A. Chin, and F. Yeh, "Novel ultra-low power rram with good endurance and retention," in *VLSI Technology (VLSIT), 2010 Symposium on*, pp. 85–86, June 2010.

[36] H. Akinaga and H. Shima, "Resistive random access memory (reram) based on metal oxides," *Proceedings of the IEEE*, vol. 98, pp. 2237–2251, Dec 2010.

[37] ITRS, *International Technology Roadmap for Semiconductors: 2013 Edition*. `http://www.itrs.net/Links/2013ITRS/Home2013.htm`.

[38] R. Kozma, R. E. Pino, and G. E. Pazienza, *Advances in Neuromorphic Memristor Science and Applications*. Springer Publishing Company, Incorporated, 2012.

[39] J. Wang, Y. Tim, W. Wong, and H. H. Li, "A practical low-power memristor-based analog neural branch predictor," in *International Symposium on Low Power Electronics and Design (ISLPED), Beijing, China, September 4-6, 2013*, pp. 175–180, 2013.

[40] C. Yakopcic, R. Hasan, T. Taha, M. McLean, and D. Palmer, "Memristor-based neuron circuit and method for applying learning algorithm in spice?," *Electronics Letters*, vol. 50, pp. 492–494, March 2014.

[41] M. D. Pickett, G. Medeiros-Ribeiro, and R. S. Williams, "A scalable neuristor built with Mott memristors," *Nature materials*, 2012.

[42] A. M. Sheri, A. Rafique, W. Pedrycz, and M. Jeon, "Contrastive divergence for memristor-based restricted boltzmann machine," *Engineering Applications of Artificial Intelligence*, vol. 37, pp. 336 – 342, 2015.

[43] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, pp. 61–64, 2015.

[44] B. Razavi, *Principles of data conversion system design*. New York, NY, USA: Wiley-IEEE Press, 1995.

[45] N. S. R.L. Geiger, P.E. Allen, *VLSI design Techniques for Analog and Digital Circuits*. New York, NY, USA: McGraw-Hill Publishing Company, 1990.

[46] W. Kester and I. Analog Devices, "Data conversion handbook."

[47] M. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," *Computers and Digital Techniques, IEE Proceedings* -, vol. 150, pp. 403–411, Nov 2003.

[48] D. Larkin, A. Kinane, V. Muresan, and N. E. O'Connor, "An efficient hardware architecture for a neural network activation function generator.," in *ISNN (2)* (J. Wang, Z. Y. 0001, J. M. Zurada, B.-L. Lu, and H. Yin, eds.), vol. 3973 of *Lecture Notes in Computer Science*, pp. 1319–1327, Springer, 2006.

[49] M. Skubiszewski, "An exact hardware implementation of the boltzmann machine," in *Parallel and Distributed Processing, 1992. Proceedings of the Fourth IEEE Symposium on*, pp. 107–110, Dec 1992.

[50] P. Wawrzynski and B. Papis, "Fixed point method for autonomous on-line neural network training," *Neurocomputing*, vol. 74, no. 17, pp. 2893 – 2905, 2011.

[51] S. Duan, X. Hu, L. Wang, and C. Li, "Analog memristive memory with applications in audio signal processing," *Science China Information Sciences*, pp. 1–15, 2013.

[52] Intel Corporation., *IA-32 Intel Architecture Optimization Reference Manual*, 2003.

[53] Advanced Micro Devices, Inc., *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, 2010.

[54] Micron Technology, Inc., http://www.micron.com//document_download/?documentId=4297, *TN-41-08: Design Guide for Two DDR3-1066 UDIMM Systems Introduction*, 2009.

[55] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC simulator," January 2005. http://sesc.sourceforge.net.

[56] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration," in *International Symposium on Quality Electronic Design*, 2006.

[57] "Spectre circuit simulator." `http://www.cadence.com/products/cic/spectre_circuit/pages/default.aspx`.

[58] X. Dong, C. Xu, Y. Xie, and N. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, pp. 994–1007, July 2012.

[59] "Encounter RTL compiler." `http://www.cadence.com/products/ld/rtl_compiler/`.

[60] "Free PDK 45nm open-access based PDK for the 45nm technology node." `http://www.eda.ncsu.edu/wiki/FreePDK`.

[61] M. N. Bojnordi and E. Ipek, "Pardis: A programmable memory controller for the ddrx interfacing standards," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pp. 13–24, IEEE, 2012.

[62] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "Fabscalar: composing synthesizable rtl designs of arbitrary cores within a canonical superscalar template," in *Proceeding of the 38th annual international symposium on Computer architecture*, pp. 11–22, 2011.

[63] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pp. 51–62, 2008.

[64] M. Zangeneh and A. Joshi, "Design and optimization of nonvolatile multibit 1t1r resistive ram," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, pp. 1815–1828, Aug 2014.

[65] M.-D. Ker, S.-L. Chen, and C.-S. Tsai, "Design of charge pump circuit with consideration of gate-oxide reliability in low-voltage cmos processes," *Solid-State Circuits, IEEE Journal of*, vol. 41, pp. 1100–1107, May 2006.

[66] G. Palumbo and D. Pappalardo, "Charge pump circuits: An overview on design strategies and topologies," *Circuits and Systems Magazine, IEEE*, vol. 10, pp. 31–45, First 2010.

[67] S. Wilton and N. Jouppi, "CACTI: An enhanced cache access and cycle time model," vol. 31, pp. 677–688, May 1996.

[68] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *International Symposium on Computer Architecture*, 2009.

[69] H. Suzuki, J. ichi Imura, Y. Horio, and K. Aihara, "Chaotic boltzmann machines," *Scientific Reports*, vol. 3, pp. 1–5, 2013.

[70] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, Dec. 2011.

[71] T. Larrabee, "Test pattern generation using boolean satisfiability," *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 4–15, 1992.

[72] J. Ferguson and T. Larrabee, "Test pattern generation for realistic bridge faults in cmos ics," in *In Proceedings of International Test Conference*, pp. 492–499, IEEE, 1991.

[73] "Olivetti-att-orl." `http://www.cs.nyu.edu/~roweis/data.html`.

[74] G. E. Hinton, "A practical guide to training restricted boltzmann machines," Technical Report 2010-003, Department of Computer Science, University of Toronto, 2010.

[75] R. O'Donnell and Y. Wu, "An optimal sdp algorithm for max-cut, and equally optimal long code tests," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, (New York, NY, USA), pp. 335–344, ACM, 2008.

[76] H. Kautz, B. Selman, and Y. Jiang, "A general stochastic approach to solving problems with hard and soft constraints," in *The Satisfiability Problem: Theory and Applications*, pp. 573–586, American Mathematical Society, 1996.

[77] Micron Technology, Inc., http://www.micron.com//get-document/?documentId=416, *8Gb DDR3 SDRAM*, 2009.

[78] Micron, *Technical Note TN-41-01: Calculating Memory System Power for DDR3*, June 2009. `https://www.micron.com/~/media/Documents/Products/Technical%20Note/DRAM/TN41_01DDR3_Power.pdf`.

[79] A. Asenov, S. Kaya, and A. R. Brown, "Intrinsic parameter fluctuations in decananometer mosfets introduced by gate line edge roughness," *Electron Devices, IEEE Transactions on*, vol. 50, no. 5, pp. 1254–1260, 2003.

[80] D. Niu, Y. Chen, C. Xu, and Y. Xie, "Impact of process variations on emerging memristor," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pp. 877–882, IEEE, 2010.

[81] D. Niu, Y. Xiao, and Y. Xie, "Low power memristor-based reram design with error correcting code," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pp. 79–84, Jan 2012.

[82] M. Hu, H. Li, Y. Chen, X. Wang, and R. E. Pino, "Geometry variations analysis of tio 2 thin-film and spintronic memristors," in *Proceedings of the 16th Asia and South Pacific design automation conference*, pp. 25–30, IEEE Press, 2011.

[83] M. Hu, H. Li, and R. E. Pino, "Fast statistical model of tio 2 thin-film memristor and design implication," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 345–352, 2011.