Hardware Accelerator for Convolutional Restricted Boltzmann Machines

Junghoon Han

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2025-29
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-29.html

May 1, 2025

Acknowledgement

I would like to thank Professor Sayeef Salahuddin for his continued mentorship and generous sponsorship during my master's program. I thank Pratik Brahma, who pioneered this research topic, for his close guidance, ideas, and help on this project. Thanks to the rest of the Unconventional Computing group members, Chirag Garg, Saavan Patel, and Philip Canoza, in helping out with this project in various ways.

I am profoundly grateful for the exceptional support of my family throughout the years. And thanks to all my friends, especially my fellow Ra-On band members, who made my graduate program fruitful.

Hardware Accelerator for
Convolutional Restricted Boltzmann Machines

by Junghoon Han

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Sayeef Salahuddin
Research Advisor

5.07.2024

(Date)

* * * * * * *

Professor Sophia Shao
Second Reader

5/6/2024

(Date)

Hardware Accelerator for Convolutional Restricted Boltzmann Machines

by

Junghoon Han

Abstract

Restricted Boltzmann Machines (RBMs) have gained attention for their strength in aiding Monte Carlo simulations for Combinatorial Optimization, Quantum Applications, and Machine Learning problems. Convolutional RBM (CRBM), a variant of RBM, has sparked interest due to its lower parameter counts and efficient performance for translationally-symmetric problems. However, the stochastic nature of CRBM often makes it take long duration to reach the ground-state solution, demanding an approach to accelerate the computation process.

In this work, we demonstrate our hardware accelerator for CRBM, implemented in RTL and programmed on FPGA. Software applications can harness the accelerator by simply programming the weights, bias, and lattice sizes. We show that for solving frustrated classical Hamiltonians for Ising Shastry-Sutherland model, our hardware accelerates the reaching of ground-state solution by up to 5 orders of magnitude compared to GPUs.

Contents

Chapter 1

Introduction

1.1 Background

In the ever-evolving landscape of Ising models, the quest for efficient and robust models capable of processing complex data remains paramount. Among the myriad of techniques that have emerged for mapping Ising models, Restricted Boltzmann Machines (RBMs) stand out as a fundamental building block in the realm of unsupervised learning. With their ability to capture intricate patterns, parallelize gibbs sampling, and map relationships between different neurons, RBMs have garnered considerable attention and acclaim in the field of Combinatorial Optimization, Quantum problems, and classical Ising models.

Part of this attention is ascribed to Convolutional Restricted Boltzmann Machines (CRBMs). CRBMs harness the power of probabilistic inference to explore solution spaces more effectively, thereby enabling the discovery of optimal or near-optimal solutions in computationally challenging problems. CRBMs, a convolutional variant of RBMs, have lower parameter counts, thereby increasing the compute efficiency for training and inference. Recent work has sparked interests in its ability to optimally map translationally symmetric problems, in which convolution weights are repeated every stride.

The transformative potential of CRBMs has immense practical significance in addressing real-world challenges with profound implications. In materials science, the ability to explore vast solution spaces with probabilistic methodologies enables researchers to expedite the search for novel compounds and materials with desired properties. This paper will partially include demonstration of mapping a classical Ising Shastry-Sutherland model to CRBMs to accelerate the sampling computations to reach ground-state solution.

Due to their stochastic nature, CRBMs may require significant iterations of sampling to reach the desired ground-state solution. The required sampling count also increases with the number of neurons in the CRBM. Thus, to harness the power of CRBMs within a reasonable compute time, an efficient implementation is essential. This motivates our approach to designing hardware accelerators for CRBMs to improve compute time and energy efficiency.

1.2 Motivations and Previous work

Motivation for Hardware Acceleration

Mapping the mathematical logic directly into digital Register Transfer-level (RTL) logic, rather than encoding them to instructions for general purpose computers, can speed up the calculations by several orders of magnitude. This process can not only save computation time, but also reduce the energy required to compute a desired program.

Same logic follows for designing a custom digital hardware accelerator for Convolutional RBMs. Transistor logic can be customized and optimized to suit the specific requirements of CRBMs, such as optimizing memory access patterns, exploiting spatial parallelism at the hardware level, and implementing specific modules tailored for Gibbs sampling computations. The details of the hardware implementation are noted in Chapter 3.

Relevant Previous work

This research is part of Salahuddin Lab's Unconventional Computing subgroup, which has been using RBMs for NP-Hard combinatorial optimizations. Our team's former members have demonstrated the used of hardware accelerated RBMs for solving optimization problems such as MAX-CUT problem and Sherrington-Kirkpatrick spin glass. The FPGA-mapped RBM has demonstrated similar or better scaling performance compared to Quantum Computers such as DWave 200Q Quantum Adiabatic Computer [1]. Subsequent work has used the RBM Hardware accelerator for integer factorization of 16-bit numbers. This work showed a staggering runtime improvement of 10000x over CPUs and 1000x over GPUs. [2]

As previous research on hardware-accelerated RBMs have been meaningful, our group was motivated to design hardware accelerators to specific variants of RBMs, notably CRBMs. In this paper, we embark on a journey to explore the design, implementation, and evaluation of a hardware accelerator tailored specifically for Convolutional Restricted Boltzmann Machines for non-deterministic polynomial-time computing. Through RTL-level descriptions and FPGA mappings, we demonstrate the efficacy and versatility of hardware-accelerated CRBMs in solving combinatorial optimization problems.

Chapter 2

Convolutional RBM (CRBM)

2.1 Restricted Boltzmann Machine (RBM)

The Restricted Boltzmann Machine (RBM) is a stochastic 2-layer graph neural network. The 2 layers are each called "visible" and "hidden" layers, which are all-to-all connected, containing the form of a bipartite graph. RBMs are used by block Gibbs sampling between the 2 layers repeatedly, then track the visible layer values every sample to derive the probability distribution of the resulting node (neuron) values. RBM is an energy-based model, which means that the objective of sampling is to minimize the energy value associated with the weight, bias, and node values. [3]

All node values are binary: 0 or 1. The next value of a node is determined by deriving a probability for it to be of value 1 and conducting random sampling according to the probability. The next set of values for each layers is sampled by the conditional probability dependent on the other layer. The values of all nodes in a single layer are sampled jointly; the next set of hidden nodes will be sampled by probability $p(h|v)$, and the visible nodes by probability $p(v|h)$. This form of simultaneous sampling is called *block Gibbs sampling*.

The nodes and edges of the RBM correspond to neurons and synaptic connections. Thus, when we map different problems to RBM, we can assign the visible nodes to represent physical variables (such as spins, direction, group assignment) and the hidden nodes to interactions between them (such as spin interactions).

2.2 Convolutional RBM (CRBM)

While RBMs are assumed to have fully-connected edges between the visible and hidden layers, CRBMs work with strides and convolution. CRBMs show translational invariance, where the pattern of weights are identical across different parts of the nodes. As the all-to-all connection of RBM can be memory-heavy and compute-heavy, CRBM helps relax the logic by using only a set of connections to fully represent the probabilities for block Gibbs sampling.

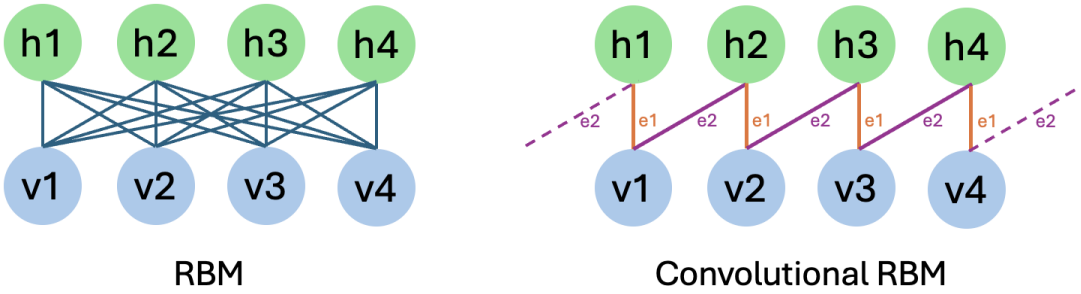

Figure 2.1: Pictorial representation of RBM and CRBM.

The 2.1 shows the structure of RBM and CRBM. As seen on the right of the figure, CRBMs have the same weights repeated every a stride (in this case, stride equal to 1). The figure also notes *periodicity*, which means when the stride goes out of bounds of the visible nodes, it wraps back to the first index of the hidden nodes (in this case, connecting v4 with h1). Periodicity can be turned on or off, depending on the problem formulation.

CRBMs can have multiple set of weights. For example, as per Figure 2.1, the first set of weights can be $w1 = (e1, e2) = (1, 2)$, while the second set of weights can be $w2 = (e1, e2) = (3, 4)$. Each set of weights will produce a group of hidden nodes. Another set of weights will produce a separate group of hidden nodes. Hereon, we will note them as convolution *groups*.

Energy and Probability formulation

The following formulas are derived by converting the general RBM energy and probability equations to reflect the convolutional nature of CRBM.

Here, the notations are: v_{ij} is the visible node at the i-th row and j-th column. k represents the convolutional group, which corresponds to the kth set of weights, also known as 'filters'. W^k is the k-th filter. $\tilde{W}^k$ is the k-th filter, flipped in both horizontal and vertical axes. ${h^k}_{ij}$ in turn represents the hidden node at group k, i-th row and j-th column. b is the hidden bias and c is the visible bias. $\bullet$ is the element-wise product followed by summation: $A \bullet B = trA^T B$. $*$ operator denotes convolution. σ denotes the sigmoid operator. [4]

$$E(v, h) = -\sum_{k=1}^{K} h^k \bullet (W^k * v) - \sum_{k=1}^{K} b_k \sum_{i,j} {h^k}_{i,j} - c \sum_{i,j} v_{ij} \tag{2.1}$$

$$P({h^k}_{ij} = 1|v) = \sigma((W^k * v)_{ij} + b) \tag{2.2}$$

$$P(v_{ij} = 1|h) = \sigma\left(\left(\sum_k (\tilde{W}^k * h_k)\right)_{ij} + c\right) \tag{2.3}$$

The objective of our CRBM is to sample repeatedly until the energy reaches the ground-state solution. (The ground-state solution is also the output with highest-likelihood). The probabilities are used to sample each of the visible and hidden node values. This probability is used to randomly sample the node value of 0 or 1, thereby determining the next value of the nodes.

2.3 CRBM Computation Logic

The CRBM computation logic and sequence is illustrated in Figure 2.2. Note that the logic flows from visible nodes → hidden nodes → visible nodes, and repeats.

2.3.1. Visible nodes

The sampling starts with the initial state of visible nodes. In our setting, the visible layer is configured as a 2-dimensional array of binary nodes.

Figure 2.2 starts with visible nodes of size 3x3.

2.3.2. Wrapping

Wrapping is done to ensure periodicity is incorporated into the convolution logic. Assume that the filter size is MxM. If periodicity is 'on' in the column direction, the first M-1 columns is copied to the last column index. If periodicity is 'off' in the column direction, there will be M-1 columns of zeros inserted. The same logic holds for the row direction.

Figure 2.2 notes the wrapping logic for a 2x2 size filter and periodicity on in both column and row direction. The wrapped nodes are denoted in color orange.

2.3.3. Convolution - Forward

Forward convolution notes the convolution logic necessary for sampling hidden nodes from visible nodes (visible → hidden). Convolution here occurs as an element-wise matrix multiply with the filter and current position's visible nodes, followed by accumulation (mac). This operation is conducted repeatedly with a stride, which moves the filter to the next respective location. The stride occurs in both column and row direction, and the process is repeated until each direction's index is out of bounds.

The complete process mentioned above is identical for all different filters. The number of output groups will be equal to the number of different filters.

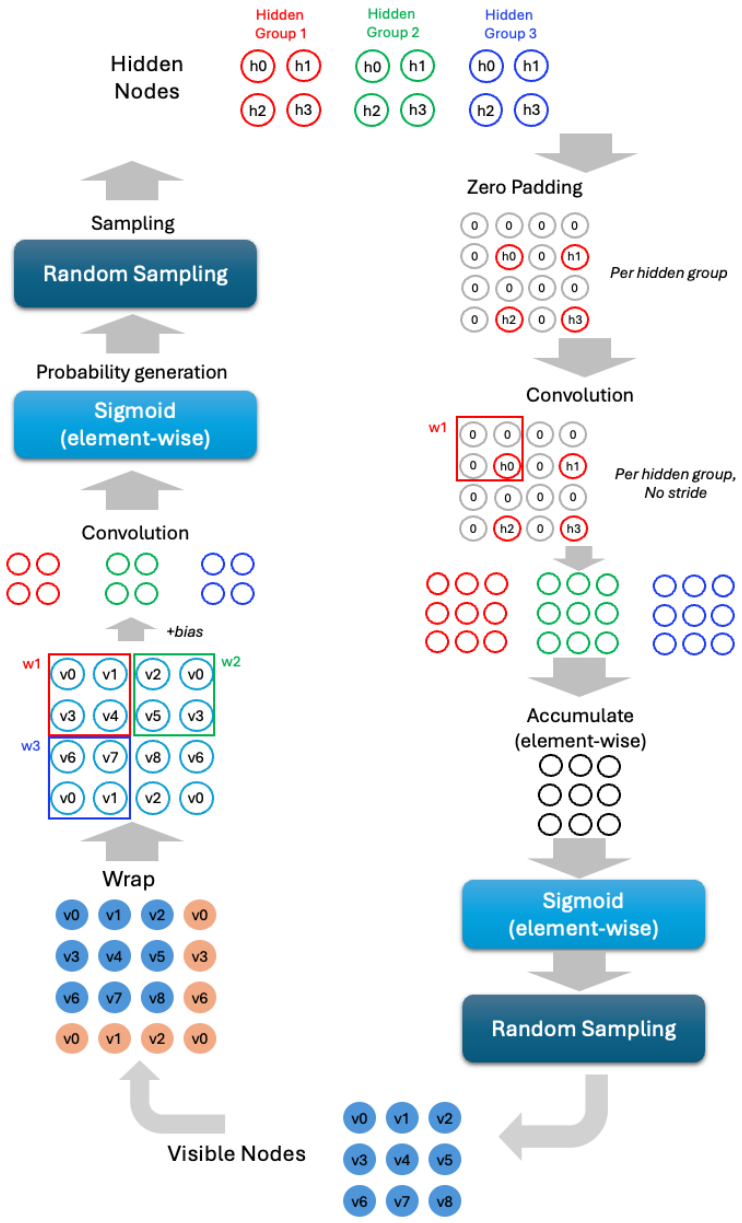

Figure 2.2: CRBM Computation logic

Figure 2.2 illustrates the convolution logic for 3 different 2x2 size filters with a stride of 2. For 4x4 visible nodes, this process creates a 2x2 result for each filter group.

2.3.4. Probability and Sampling - Forward

The convolution result is sent to a sigmoid operator to obtain the probability of $P(h|v)$. The sigmoid is applied element-wise to each of the outputs of the convolution.

Sigmoid will provide a probability value between 0 and 1, which is in-turn used for random sampling. The sampler will take the probability as the likelihood of result node being equal to 1. Then, the sampler's result, either 0 or 1, will be the next value of the hidden nodes. In practice, this process is done by generating a random floating point value between 0 and 1, comparing it to the sigmoid output, and setting the result value to 1 if the random number is less than the sigmoid output.

2.3.5. Hidden nodes

The sampled values will be the next hidden node values. With N different filters, there will be N groups of hidden nodes. All hidden node values are binary as well.

2.3.6. Zero Padding

We conduct a zero-padding technique to ensure that the resulting reverse sampling (hidden $\rightarrow$ visible) has the same dimension as the starting visible node dimension. That is, we insert zeros between the hidden nodes in all directions.

Similarly to the wrapping step, zero padding also includes copying the last columns and rows to the beginning column and row. If periodicity is on, we copy the hidden node values along with padded zerios. If periodicity is off, we simply zeros are added to the beginning column and row positions.

2.3.7. Convolution - Reverse

The convolution logic here is similar to that of the convolution in forward direction. The key difference here is that the filters applied are flipped in horizontal and vertical directions. Moreover, the stride value is always equal to 1 in the reverse direction.

2.3.8. Accumulation

For N different convolution groups, there will be N different convolution outputs. This step accumulates all the node values from the convolution output, element-wise. The dimension of the output from this step is equal to that of the visible nodes.

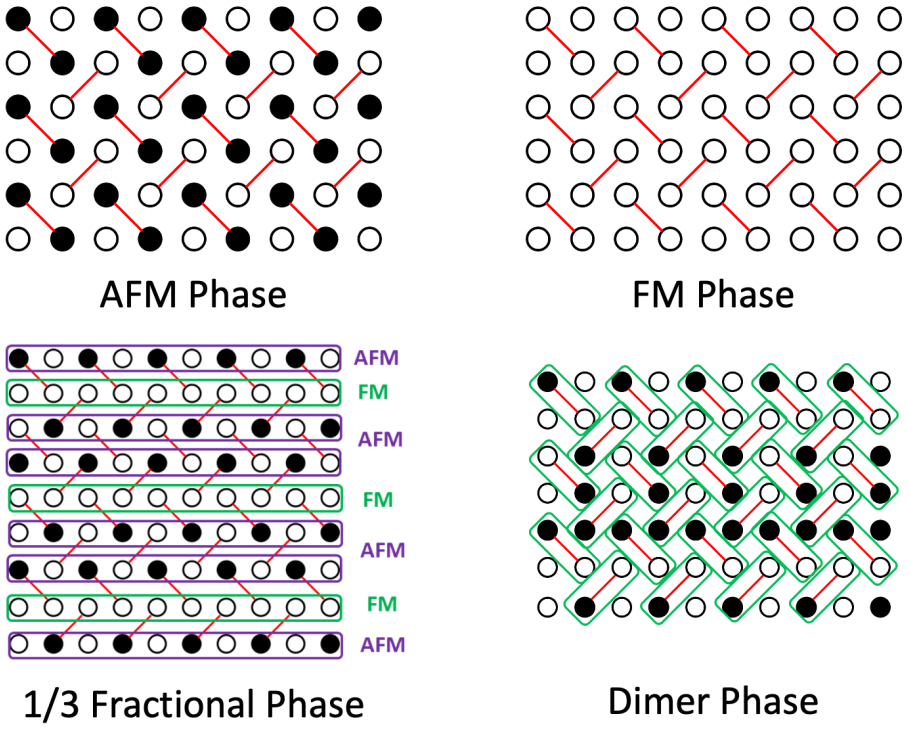

Figure 2.3: Shastry-Sutherland Magnetization Phases

2.3.10. Probability and Sampling - reverse

Similar to the forward direction process, the sigmoid is applied to produce the probability, which is used for randomly sampling the next set of visible nodes. This step produces the next set of visible node values, which completes the full cycle.

2.4 Shastry-Sutherland model mapping

In our work, we map the classical Ising Shastry-Sutherland model on the CRBM structure to solve frustrated classical Hamiltonian. Our results demonstrate that the CRBM can be used to simulate any kind of translationally-symmetric classical Hamiltonian. The Shastry-Sutherland Lattice has discrete translational symmetry, where certain set of spin interactions are repeated elsewhere on the lattice. The Shastry-Sutherland model can be mapped to CRBM in the following way: the visible nodes can represent phsyical variables, in this case the magnetic spins. The hidden nodes can represent interactions between the spins.

To map the Shastry-Sutherland Ising model to the CRBM framework, we equate the physical lattice's Boltzmann distribution to RBM's marginal distribution. The RBM weights are then mapped to be unique only upto the unit cell on the lattice, of size 3x3. Thus, the

filter sizes are 3x3. The Shastry-Sutherland contains unique 10 repeated interactions, leading to the formulation of 10 different filters.

We focused our experiment on 4 of the Shastry-Sutherland Magnetization phases, as noted in Figure 2.3. Each node, mapped to the visible nodes of CRBM, represent the magnetization spins. The empty circles are represented as 1, and filled circles are represented as 0. Different phase problems produce different filters and biases.

- AFM Phase: Anti-Ferromagnetic Phase. Every non-diagonal nodes have the opposite spins.
- FM Phase: Ferromagnetic Phase. All nodes have the same magnetization spins
- 1/3 Fractional Phase: the rows of the lattice show a pattern of FM phase row sandwiched between two AFM phase rows
- Dimer Phase: certain diagonal set of nodes are expected to be opposite spins of each other (marked in green boxes)

Detailed mapping result of the Shastry-Sutherland to CRBM will be illustrated in a coming paper from the Salahuddin Group, in a work pioneered by Pratik Brahma.

Chapter 3

CRBM Hardware Accelerator

3.1 Background

The objective of the CRBM hardware accelerator is to significantly reduce the runtime of reaching the ground-state solution of CRBM.

The hardware design is implemented in RTL (Register Transfer Level) and mapped to Field Programmable Gate Array (FPGA). We used the Virtex Ultrascale+ FPGA device (VCU118), a product of Xilinx - AMD. This FPGA represents a cutting-edge solution in the field of FPGAs with 14nm/16nm FinFET process technology, dynamic power management, and integrated Gen3 x16 PCIe blocks. We use this FPGA jointly with the experiment server with 11th Gen Intel Core i9-11900K @ 3.50GHz and 135GB RAM. To program the FPGA, we use Xilinx's Vivado tools.

3.2 Architecture

The hardware architecture, as denoted in Figure 3.1, maps each step of CRBM into respective hardware modules. Note that there are corresponding modules to the described steps in Figure 2.2.

The hardware is *pipelined* with 2 stages: forward and reverse. The forward stage contains logic of sampling from visible nodes → hidden nodes (stages 3.2.1 to 3.2.5). The reverse stage contains logic of sampling from hidden nodes → visible nodes (stages 3.2.5 to 3.2.9).

3.2.1. Visible Node Registers

The 2D visible node layer is represented in a single register. As the node values are binary, they take up a single bit in the register. This technique minimizes the LUT resource usage on the FPGA.

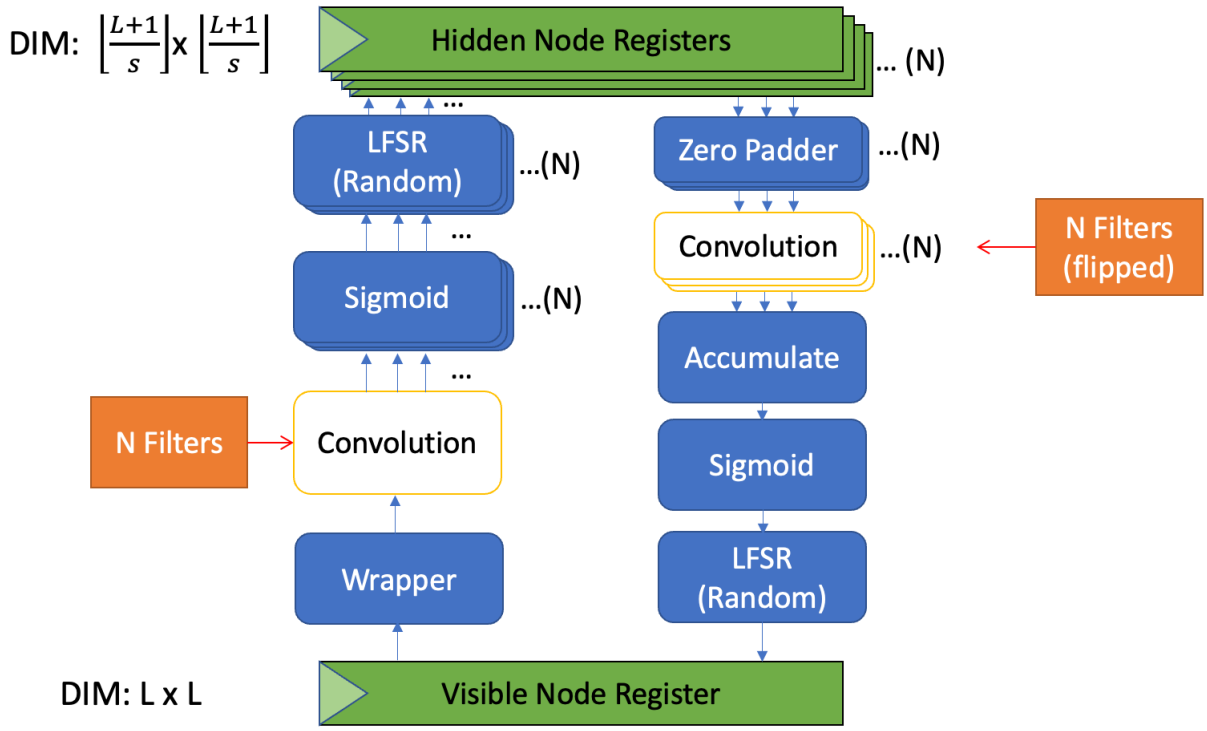

Figure 3.1: CRBM Hardware Accelerator Architecture

The dimension of the lattice size is noted as LxL, which notes L rows and L columns of visible nodes, making a total of LxL visible nodes. Thus, there are LxL bits in the visible node register.

3.2.2. Wrapper

The wrapper module follows the logic of wrapping technique noted in section 2.3.2. It takes in the visible node register and periodicity signal as inputs, and copies or zeros out the respective columns and rows accordingly.

3.2.3. Convoluter - Forward

The convoluter module takes in the wrapper and filters to conduct convolution logic as noted in section 2.3.3. The filter values are provided by the user's software via PCIe.

In this implementation, the convoluter takes advantage of spatial parallelism. It contains convolution logic of multiply and accumulate in place for corresponding positions. Same logic is copied to other positions that are separated in a distance equal to the stride value

in all directions. In summary, all convolution computation is contained in a single spatially parallelised combinational logic.

3.2.4. Sigmoid and LFSR - Forward

The sigmoid modules are synthesized with the input and output bit count parameters, which are used to determine the level of precision of the input and output. The input is the result of the convolution. The output is the corresponding sigmoid value. The sigmoid module internally contains a pre-coded LUT which is identical to a dictionary of key and value, input and output. The module selects the closest corresponding sigmoid value that was synthesized with the given precision parameters.

The Linear Feedback Shift Register (LFSR) module is synthesized according to a set seed value. The internal register in the module, initialized with the seed value, is shuffled every cycle to produce randomized bits. The LFSR output is converted to a value between 0 and 1.

For N filter groups, there are N sigmoid modules and N LFSR modules. The sigmoid value is compared with the output of the LFSR module. If the sigmoid value is greater, the corresponding hidden node will contain value 1. Otherwise, it will contain value 0. This logic completes the first pipeline stage.

The Sigmoid and LFSR hardware modules are pioneered by our former researchers, Saavan Patel and Philip Canoza.

3.2.5. Hidden Node Registers

For N filters, there are N hidden node groups produced. Each of them will have a dimension of $\lfloor (L+1)/stride \rfloor$x$\lfloor (L+1)/stride \rfloor$. Thus, the hidden node register will contain a total of N x $\lfloor (L+1)/stride \rfloor$x$\lfloor (L+1)/stride \rfloor$ bits.

3.2.6. Zero Padder

The zero padder module implements the logic noted in section 2.3.6. The hardware keeps an array of zeros with empty slots for positions that take in hidden node values. The hidden node values are inserted in a spatially parallel manner. For N filters, there are N zero padder modules.

3.2.7. Convoluter - Reverse

The convoluter module in the reverse direction is the same module used in the forward direction (section 3.2.3). The flipped weights are inputs to this module, which are provided by the user software via PCIe. For N filters, there are N reverse convoluter modules.

3.2.8. Accumulator

The output of the convoluter module is accumulated in this module. As the accumulation is done element-wise, it is simple to create a combinational logic that adds up the values for the same positions in N groups. Following the accumulator module, the N filter groups are aggregated to a single group.

Moreover, visible bias is applied in this module. The bias values are provided by the user software. We provide an option to use odd bias and even bias, which allows different bias values to be applied for odd columns and even columns.

3.2.9. Sigmoid and LFSR - Reverse

The sigmoid and LFSR modules used in the reverse direction are the same modules used in the forward direction (section 3.2.4). The output of these modules determine the next visible node values.

This module completes the second pipeline stage, and completes a full cycle of sampling.

3.3 Input and Output (I/O) and Programming Logic

The host machine and the FPGA communicates over the x16 PCIe. We implement the Input and Output (I/O) logic of the PCIe through an open source module named Xillybus. Xillybus provides both an FPGA IP core and a driver for the host PC's operating system. It provides customized bundles for different FPGA models.

Although our hardware need not communicate large data within each time steps, the host machine and FPGA run on different clock frequencies, producing a *clock domain crossing*. Thus, we use a First-In-First-Out (FIFO) module to enable sequential communication between the host and FPGA.

The Xillybus driver creates 2 devices files of the FPGA: one for writing and one for reading. The user software writes and reads the following values to and from the device files:

- Write (PC to FPGA): weights, flipped weights, biases, lattice sizes (visible layer dimensions), periodicity, and clear last hidden rows signal (some applications require clamping the last row of hidden nodes to zero)
- Read (from FPGA) : Visible node values of each cycle

After the host machine reads the visible node values, the user software calculates the energy of the nodes.

	Computational Complexity	Spatial Complexity
Visible Nodes	-	L x L
Convolution (x Number of Filters)	$\left\lfloor\frac{L+(L\%F)}{s}\right\rfloor^2$ convolutions Total: $\boldsymbol{F^2}\left\lfloor\frac{L+(L\%F)}{s}\right\rfloor^2$ **MACs**	(L+ ($L\%F$)) x (L + ($L\%F$))
Accumulation	- F^2 MACs + (H-1) Adds per node - L^2 operations Total: $\boldsymbol{L^2(F^2+H-1)}$ **MACs**	L x L
Hidden Nodes (x number of filters)	-	$\left\lfloor\frac{L+(L\%F)}{s}\right\rfloor$ x $\left\lfloor\frac{L+(L\%F)}{s}\right\rfloor$

Figure 3.2: CRBM Hardware Accelerator complexities

3.4 Testing

Each hardware module went through behavioral testing with individually created test benches. After each module's functionality was confirmed, all the modules were integrated and tested under the top-level test bench.

After behavioral testing, we programmed the hardware to the FPGA with integrated IO modules. Then, the hardware's correctness was tested once more with simple test samples of Shastry-Sutherland Ising model.

3.5 Analysis

Computational and Spatial Complexities

Computational and Spatial complexities are noted in Figure 3.5. Here, L is the Lattice size, which is the row and column count of the 2D visible nodes. F is the filter size, s is the stride, and H is the hidden layer size. MAC is multiply and accumulate.

Pipelining

As the hardware is pipelined with 2 stages (forward and reverse), it takes 2 cycles to sample a single set of visible nodes. At the same time, it allows 2 different independent samplings to occur in parallel. At every cycle, the visible node register and hidden node registers contain values for different chain of sampling. These registers rotate between the 2 chains, thereby

enabling 2 different independent samplings for odd cycles and even cycles. This process boosts our expected *throughput* by a factor of 2.

Dimension and Precision

The maximum dimension that fits on our FPGA is 18x18, total of 324 visible nodes, followed with 810 hidden nodes. We implemented the hardware with filter size of 3x3, 10 filters, and 15 bits of floating point precision for sigmoid and LFSR. We use 10 precision bits for convolution calculations, which are allocated to 1 sign bit, 5 integer bits, and 4 floating-point bits.

Clock Frequency and Critical Path

The hardware was synthesized on the FPGA with the clock frequency of 30MHz. For each sampling chain, the visible node produces new values every 2 cycles. Thus, 1 chain completes a sampling sequence in approximately 66.66 nanoseconds. As there are 2 chains, there is 1 unique sample produced every 33.33 nanoseconds.

The clock frequency is bottle-necked by the critical path, which is the forward pipeline stage (visible to hidden). Significant portion of the area is allocated to the sigmoid and LFSR modules in this stage. Reducing the precision bits of these modules relaxes the critical path to a certain extent, but in turn reduces accuracy or raises the required sampling counts to reach ground state solution. Our current choice of precision bits was optimized between these two factors.

Chapter 4

Results

4.1 Time to Solution

As RBM and CRBM demand a Time to Solution framework that accommodates for its stochastic nature. Our lab's previous work on RBM acceleration have identified the Time to Solution framework as follows [1]:

$$TTS_{99} = \frac{N_s}{f_{clk}} \frac{log(0.01)}{log(1 - p_{gnd})} \tag{4.1}$$

TTS_{99} refers to the time to reach a solution with 99% confidence. N_s is the number of samples taken, f_{clk} is the clock frequency, and p_{gnd} is the probability of the sampling sequence hitting the ground state. However, this work focused on sampling enough cycles so that the mode of the sampling is equal to the ground state.

In our study of CRBM for Shastry-Sutherland Ising model, we focus on the first ever cycle that hits the ground state solution. Thus, we modify the equation to the following:

$$TTS_{99} = T_{avg} \frac{log(0.01)}{log(1 - p_{gnd})} \tag{4.2}$$

T_{avg} refers to the average time taken to reach the ground state sample. This is calculated by multiplying the hardware's clock frequency and number of cycles taken to reach the ground state, averaged across the batch size. p_{gnd} is also empirically estimated by calculating fraction of total experiments that reached the ground solution.

4.2 Runtime Results

The comparative results of GPU and the hardware accelerator are noted in Figure 4.2. Each phases are labaled in the same color. Both axes are in log scale.

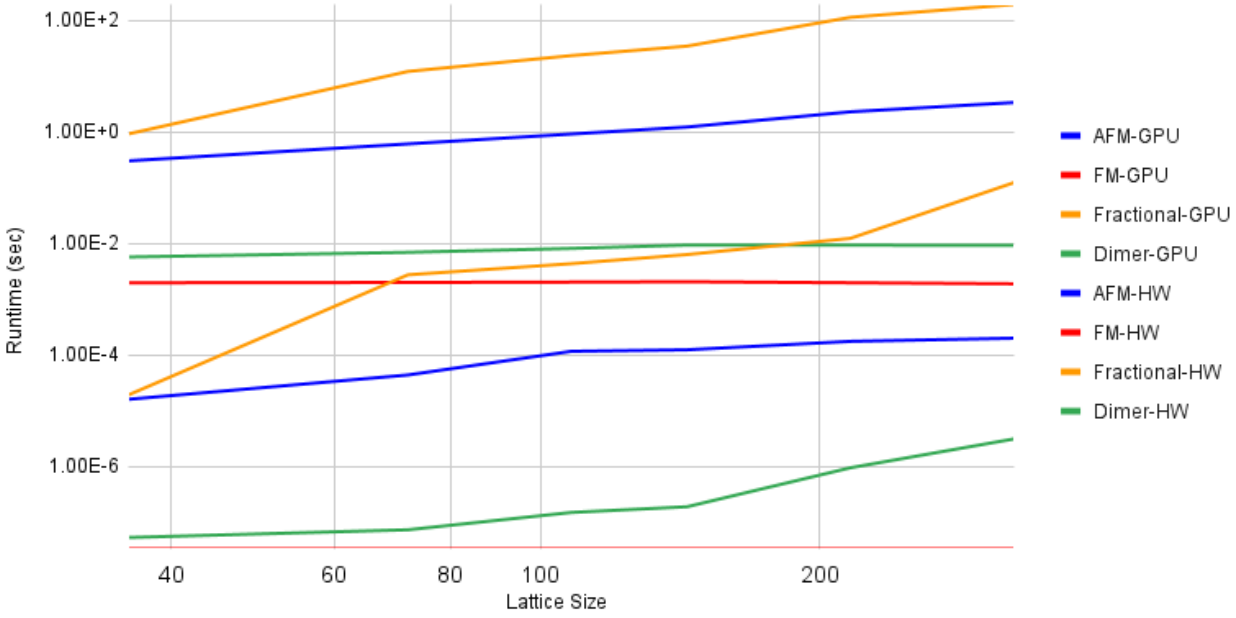

Figure 4.1: Runtime results of GPU vs CRBM HW Accelerator

GPU results

The GPU implementation, pioneered by Pratik Brahma, is written in PyTorch and CUDA. As GPU was used as a parallel machine, it is also used to test larger lattice size than the hardware accelerator can fit in. The GPU was tested in 36, 144, and 324 visible nodes.

Hardware Accelerator Results

The hardware accelerator boasts significant performance increase compared to the GPU results. The performance improvement was at least 636x compared to the GPU. These results confirm that custom digital logic can improve runtime performance by several orders of magnitude.

The hardware accelerator was tested in 36, 72, 108, 144, 216, 324 visible nodes.

4.3 Evaluation

Table 4.1 shows the hardware accelerator's runtime performance improvement compared to GPU. The table includes improvements under overlapping experiment settings.

The lowest improvement is for 1/3 Fractional Phase with Lattice size 324. The greatest improvement is for Dimer Phase with Lattice size 36.

Lattice	AFM	FM	Fractional	Dimer
36	18867x	59165x	19459x	107236x
144	9876x	61731x	2222x	49276x
324	16839x	56807x	636x	2974x

Table 4.1: Runtime improvement compared to GPU

Chapter 5

Conclusion

5.1 Future Steps

Although the current hardware accelerator design provides promising results, there is always room for more experimentation and improvement. The objective of the hardware design was not to show the most optimal design possible, but to demonstrate the potential and versatility of hardware-accelerating the CRBM.

Other Applications

CRBMs can be used for solving any Ising models with translational symmetry. The Kagome lattice model is an ideal candidate, due to its repeated bond interactions across its lattice, which can be mapped to filter size of 4x4. [5]. Same goes for quantum lattice models such as transverse Ising and Heisenberg models, as they show a similar checkerboard-style lattice as certain Shastry-Sutherland formations. [6] CRBMs can be an interesting solution to solve these problems for its parallelizable Gibbs sampling, along with the demonstrated hardware acceleration.

Area efficiency techniques

- Correlated Noise: the current critical path involves the multiple copies of sigmoid and LFSR modules in the forward sampling stage. Instead of using 1 LFSR module per index and per group, we can experiment on using 1 LFSR module per index and shared across groups. Assuming correctness holds, this technique can reduce the LFSR LUT resource usage.

- Pipelining: the hardware currently takes in N filters and generate N filter groups, which is spatially parallelized. Instead, we can have a single set of convolution, sigmoid, and LFSR logic so that we sample one group per clock cycle. This technique can also significantly reduce area usage but will take more cycles per sampling.

- SLR Floor plans: Xilinx provides timing closure techniques in which one can clamp certain modules to super logic regions (SLR). If the floor planning is properly constrained, it is possible to efficiently use the resources to increase the clock frequency.

FPGA Accelerator Cards

Instead of trying to fit in all hardware logic into a single FPGA, a potential solution is to use multiple FPGAs to emulate a single hardware design. Our lab is already in the stage of experimenting on on FPGA Accelerator Cards: Alveo UL3524 FPGA accelerators. If hardware logic can be parallelized across multiple FPGA Accelerator cards, it can open the door for numerous hardware techniques under significant freedom from area constraints.

5.2 Conclusion

Convolutional Restricted Boltzmann Machine (CRBM), a variant of RBM, has sparked interest due to its lower parameter counts and optimal performance for translationally-symmetric problems. As its stochastic nature can render it to take long durations to reach ground-state solution, we designed a hardware accelerator to speed up the sampling process.

In this work, we demonstrated our hardware accelerator for CRBM, implemented in RTL and programmed on FPGA. The hardware is harnessed by software that writes in problem configurations and reads out visible node values. We demonstrated the hardware's performance in solving classical Hamiltonians for Ising Shastry-Sutherland model, in which it showed performance improvement of up to 5 orders of magnitude compared to GPUs.

We believe fulfilled our objective of demonstrating the potential and versatility of implementing a hardware accelerator for stochastic models such as CRBM.

Bibliography

[1] Saavan Patel et al. “Ising Model Optimization Problems on a FPGA Accelerated Restricted Boltzmann Machine”. In: *arXiv:2008.04436v2* (2020), pp. 1–25.

[2] Saavan Patel, Philip Canoza, and Sayeef Salahuddin. “Logically synthesized and hardware-accelerated restricted Boltzmann machines for combinatorial optimization and integer factorization”. In: *nature electronics* 5 (2022), pp. 92–101.

[3] Asja Fischer and Christian Igel. “An Introduction to Restricted Boltzmann Machines”. In: *CIARP* (2012).

[4] Honglak Lee et al. “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations”. In: *ICML* (2009).

[5] Tomonari Mizoguchi, L.D.C. Jaubert, and Masafumi Udagawa. “Clustering of Topological Charges in a Kagome Classical Spin Liquid”. In: *PRL 119* (2017).

[6] Louis-Paul Henry et al. “Spin-wave analysis of the transverse-field Ising model on the checkerboard lattice”. In: *Physical Review B 85* (2012).