Probabilistic Ising Architectures for Combinatorial Optimization, Machine Learning and Neuromorphic Computing

Saavan Patel

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-37
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-37.html

May 1, 2024

Probabilistic Ising Architectures for Combinatorial Optimization, Machine Learning and Neuromorphic Computing

by

Saavan K Patel

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sayeef Salahuddin, Chair
Professor Jan Rabaey
Professor Bruno Olshausen

Summer 2023

Probabilistic Ising Architectures for Combinatorial Optimization, Machine Learning and Neuromorphic Computing

Abstract

Probabilistic Ising Architectures for Combinatorial Optimization, Machine Learning and Neuromorphic Computing

by

Saavan K Patel

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Sayeef Salahuddin, Chair

As the demand for big data increases and the speed of traditional CPUs cannot keep pace, new computing paradigms and architectures are needed to meet the demands for our data hungry world. To keep pace with this, Ising Computing and probabilistic computing have emerged as a method to solve NP-Hard optimization problems (such as logistics, place and route in circuits), perform Machine Learning training and inference, model decision making in animal brains, and much more.

This work centers around parallelized computing algorithms and hardware based on probabilistic formulations of the Ising Model, known as Boltzmann Machines. The algorithms demonstrated use techniques from machine learning, structured algorithmic mapping, and mixed approaches. Using these algorithms we demonstrate potential solutions to 16 bit Integer Factorization, 3-SAT, LDPC Codes and scalable techniques for solutions to a variety of problems.

We map these algorithms to a variety of hardware, from small and medium scale ($\approx$ 1000s of nodes) FPGA approaches to large and ultra large ($> 100,000$ nodes) scale on GPU and TPU instances. These accelerated instances demonstrate state of the art performance on the MAX-CUT benchmark problem.

We finally demonstrate the Parallel Asynchronous Stochastic Sampler (PASS), a neuromorphic, clock-free accelerator that mimics brain-like asynchronous computation using the Ising Model. This has the potential for orders of magnitude speed increase over traditional methods for solving these problems while being the first on-chip, fully CMOS, demonstration of such an architecture.

To my family

For their unwavering support, compassion, and understanding

and to Divya

for helping me through the hard times, and celebrating every small achievement

Contents

List of Figures

List of Tables

Acknowledgments

My biggest thanks goes to my advisor, Sayeef Salahuddin. He took me on as his student and gave me the space to explore and let my mind and ideas expand. Although my research was not in the group's expertise, I was allowed to try new things and form a Ph.D. that I truly can call my own work and be proud. I am perpetually thankful to him for the opportunities, advise and direction he has given me throughout this journey. Thank you to my thesis and qualifying exam committee, Bruno Olshausen, Jan Rabaey, and Alistair Sinclair for their advice and direction through this process.

The support of my research group has been amazing. Although many times my research has been orthogonal to theirs, they have provided help and insight at each step. The Ising Machine and unconventional computing subgroup of Pratik Brahma and Philip Canoza, along with the new additions of Chirag Garg and Michael Odouza have been amazing to bounce ideas and collaborate with. They have sewn the seeds for some great ideas, and helped me through the hard times. My deskmate Adi Jung has been a great companion, friend and amazing sounding board for wild ideas and theories, both scientific and non-scientific, even helping me source some amazing new projects and applications. The rest of the group, Jason Hsu, Ava Tan, Dominic Labanowski, Korok Chatterjee, Justin Wong, Steve Volkman, Suraj Cheema and many others have been instrumental in my success, providing advice, guidance and helping me along the way. A special thanks to the team who made the PASS chip, Philip Canoza, Adhiraj Datar and Steven Lu who worked tirelessly for months to get out an amazing device, bringing to fruition something I had proposed in my first months of graduate school. I had the pleasure of working with some fantastic masters students, Kaitlyn Chan, Adhiraj Datar, and Steven Lu have proposed and executed many amazing ideas. My undergraduate mentees, Angela Chen, Gavin Liu, George Zong, Jared Brown, Lili Chen, Madison Manley, Megan Zeng, Naveen Gopalan, Shreyas Agarwal, Troy Sheldon, Varun Menon, Vince Han, Wendy Hua and Yijin Wang, who have all provided great value to my thesis and thinking while experimenting with new (and sometimes wacky) ideas.

My family has shown unwavering support, compassion, and understanding through this wild rid of 6 years (and 10 total years at Berkeley!). My sister, Kavita, has always been there to celebrate every victory, pick me up after every failure, and make sure I was always well fed. My parents who have always supported my endeavors and pushed me to be the best version of myself at every point. They moved to this country to give me and my sister better opportunities, and I cannot thank them enough for the sacrifices they made to get us to the point we are at today. I would never have been able to make it to my PhD, let alone finish it, without the help of my family.

My friends have continued to show me that there is life outside my lab, and reminded me about the things that make me happy. My graduate school friends, Nate Lambert, Mauricio Bustamente, Ryan Kaveh, Sam Holladay, Chinmay Nirkhe, Mike Danielczuk, Brian Kilberg, Lydia Lee, and many others have made sure that I made it through school, as we struggled together through the trials and tribulations of graduate school. My Oakland friends, Eric Noordham, Anja Kong, Tej Modi, Suhani Chhatrapati, Vikram Sreekanti, Nikita Desai and

Aparajita Pande, Roshan Ramankutty and Neelima Goddagottu have made sure that I have a wonderful group of people who have given me a community and made the bay feel like home. My friends from undergrad at Berkeley who have given me excitement and supported me through everything, Sid Moghe, Karthik Gururangan, Aditya Pradhan, Ashvin Nair, Leah Dickstein, Beliz Gunel, Tavor Baharav, and so many more. We may have dispersed around the bay (and the country), but we have always been there for eachother. My friends high school and from further back, Ning Liu, Donald Feng, and George Zhang who have understood me and helped me grow for so many years. And of course, a special thanks to my friends who I have been together with for multiple stages of life, Ashvin Nair who has been my partner in school and life since we were in high school, and Vikram Sreekanti who has made sure I keep big goals while still enjoying life through both graduate school and undergraduate studies.

This would not be complete without thanking Divya, who has kept me together for the entirety of graduate school. We met at the beginning of the journey, and she has been immensely supportive through everything. Whether she was staying up with me while I was up all night working on my chip, or making sure I'm well fed during a paper deadline, she has been through all of it. When I was not able to see my own victories, she celebrated all of the small accomplishments along the way, reminding me that I was still making progress even if I couldn't see it myself. Without her support, I would not be graduating today.

They say it takes a village to raise a child, and I think it truly took a village for me to get to this point. There are so many people I want to thank for their help, but I don't have the space to thank all of them. You know who you are, and thank you for all of your help.

Chapter 1

Introduction, Ising Model Computation

1.1 Introduction

As the demand for big data increases and the speed of traditional CPUs cannot keep pace, a new need for domain specific accelerators and new computing paradigms has arisen. Problems in the class of NP-Hard and NP-Complete are particularly challenging as they scale exponentially with problem size, causing traditional CPUs and hardware to struggle to solve them. From scheduling and logistics to analysis of physical systems and machine learning, these hard problems exist all around us. This leads to an interest in novel algorithms, architectures, and systems to solve these problems. The Ising Model problem is an example of this type of problem, with foundations in statistical physics [1, 2, 3]. For this reason, the Ising Model has emerged as an efficient way of mapping these problems onto various physical accelerators [4, 5, 6, 7, 8]. In this work, we explore the usage of the Ising Model, with emphasis on hardware acceleration of Ising Model problems, to solve hard problems in a variety of different fields.

1.2 The Ising Model

Physical Systems

The Ising model was originally formulated by Ernst Ising and Wilhem Lenz as a model of interaction between electronic spins in a lattice almost 100 years ago in 1924. It was originally used to understand magnetic phase change and spontaneous magnetization in ferromagnetic materials [9, 10]. Notably the model only includes local interactions between spins, but is able to capture global phenomena in the material. The equation describing the Ising Model is listed below.

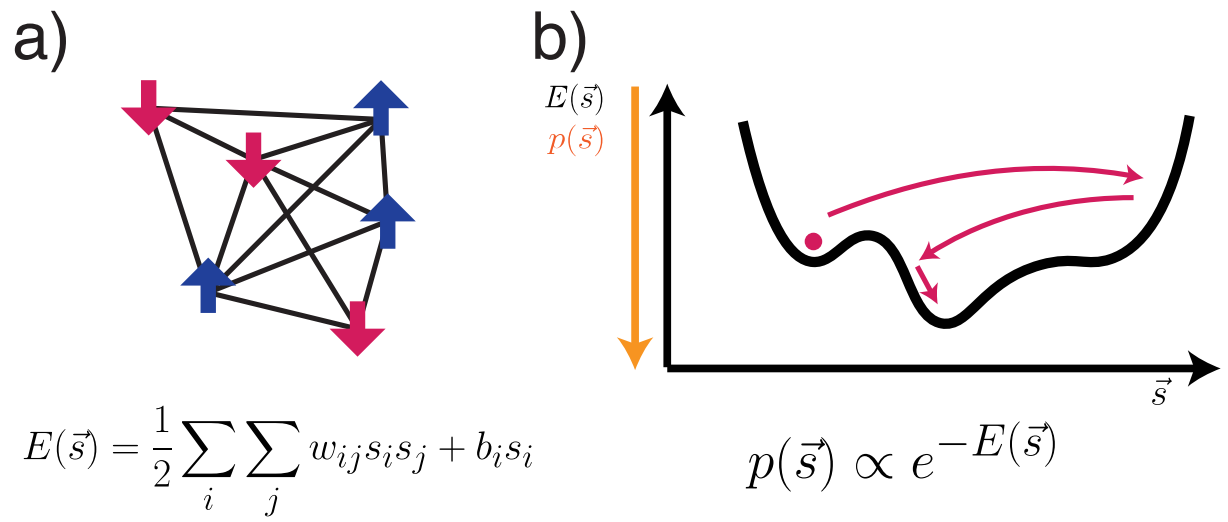

Figure 1.1: Left: The Ising Model as a lattice spin system. The Energy is modeled as the quadratic coupling between spins. Right: Finding the energy minimum of the Ising model problem is a non-convex optimization problem with many local minima. It can be modeled probabilistically by taking the exponential of the energy function (note that the normalizing constant is not shown)

$$E(s) = \frac{1}{2}\sum_{ij} J_{ij} s_i s_j + \sum_i b_i s_i \tag{1.1}$$

In the equation above, the Hamiltonian E describes the global energy of the spin system. Each spin has a scalar interaction between every other spin (i.e. $J_{ij} \in \mathbb{R}$), and spins are quadratically coupled (i.e. only pairwise $s_i s_j$ interactions are included rather than terms of higher order, like $s_i s_j s_k$). The spin values are usually constrained to being $s_i \in \{-1, 1\}$, representing up and down spin values. There are many extensions of the Ising Model to higher dimensions and including greater than binary spin states. These extensions have direct mappings onto the simplified binary spin, quadratic interaction Ising model shown below. Some of these will be discussed in later sections.

Neural Computation

A probabilistic extension of the Ising Model, known as the Boltzmann Machine, was first introduced in 1983, with an associated learning algorithm [11, 12]. The Boltzmann Machine was originally introduced as a constraint satisfaction network based on the Ising model problem, where the weights would encode some global constraints, and stochastic units were used to escape global minima. However, learning was very slow with this model due to the

difficulties with sampling and convergence, as well as the inability to exactly calculate the partition function.

A subset of the Boltzmann Machine, the Restricted Boltzmann Machine (RBM) was able to circumvent some of the issues with sampling and convergence using a simplified learning algorithm and a parallelized sampling algorithm which has the ability to be efficiently learned [13]. The RBM found favor through much of the early 2000s up until the rise of deep learning and generative algorithms. With the ability to map a variety of problems from various domains, it was a popular tool for Machine Learning before the large scale advent of Deep Learning algorithms.

Optimization Problems

Finding the ground state of solution for the Ising Model problem has been demonstrated to be NP-Hard, with the decision form of the problem being NP-Complete. [1, 2, 3, 14] This puts it in a class of exponential time scaling systems, and gives it strong connections to other NP-Hard optimization problems [15, 16].

In the context of Optimization problems the Ising Model has also been referred to as the Quadratic Unconstrained Binary Optimization (QUBO) problem. The problem formulation is the same, but the variables usually take values of $s_i \in \{0, 1\}$. This mapping has been variously applied to problems such as the MAX-CUT problem, Travelling Salesman Problem, Quadratic Assignment Problem, Set Assignment Problem, and many others. [3, 14]. So far, there has not been a successful application of Ising Model based optimization problems in large industrial scale problems. This has been mainly due to a need for more effective mappings as well as large, fast solvers that can solve problems at a scale that is relevant to real life problems.

In the optimization context, each variable in the QUBO model system usually corresponds to a decision variable, (i.e. x_i might correspond to whether node i is on on the right or the left side of the cut in a MAX-CUT problem). Depending on the problem, there many be a 1 to 1 mapping of QUBO variables to problem variables (such as the MAX-CUT problem) or there may be a many to 1 mapping of QUBO variables to problem variables (such as the Travelling Salesman Problem, where n cities correspond to n^2 QUBO variables in the naive implementation). Integers may be mapped onto the QUBO model as well by using n variables to represent an n-bit number. The QUBO model, in the form described here, cannot however model continuous variables unless the continuous variable is discretized. Constraints may be mapped via the use of a Lagrangian multiplier for the given constraint combined with the energy function of the minimization problem.

No Free Lunch Theorems

The no free lunch theorem (in basic terms) states that no algorithm will be able to perform uniformly better on all search and optimization problems than another. If one algorithm performs better than another algorithm on a given set of problems, it necessarily performs

worse than the other algorithm on a different set of problems. Without knowledge of prior structure of the problem, and choosing an algorithm that exploits the known structure of the problem, we will be unable to do better than simply random search [17].

From the Ising Model Optimization problem perspective this means that we need to design our algorithms and problems for the Ising model accelerators. One of the goals of this research was to understand which kinds of problems the Ising Model is well suited to solve, and which problems it is ill-suited to solve, and to design both algorithms and hardware to reflect the parts of Ising Computing that it is well suited for. Similarly, it is possible to make modifications to the underlying algorithms that are used to solve the Ising Model depending on the problem type being mapped to this. This would insinuate that just one accelerator is not necessary, but a family of accelerators to solve a family of problems.

1.3 Hardware Accelerators for Ising Model Problems

Recently the Ising Model has received attention for optimization problem due mainly to its connections to Adiabatic Quantum Computing and Quantum Annealing as put out by the DWave Computer [18]. This has led to a Renaissance of accelerators based on a variety of computing topologies. These are roughly divided into three categories "Digital Hardware", "Analog Hardware", and "New Physics" Based systems.

Digital Hardware Based Accelerators

A variety of digital hardware based accelerators have been created to solve Ising model problems as well. Most rely on an annealing mechanisms to slowly force each neuron into a 0 or 1 (-1 or +1) state. [19, 4].

The digital mixed signal systems will generally rely on updating each spin based on a probabilistic update rule, and accepting the change with some probability. This is similar to a Markov Chain Monte Carlo, or a Simulated Annealing based scheme, which is further discussed in Chapter 2.

If the accelerator used obeys traditional rules of sampling and simulated annealing, they can provide a guarantee of convergence that analog and physical accelerators may not be able to provide. The digital accelerator systems may not be able to take advantage of the physical nature of the Ising Model accelerator as a set of interacting spins, instead they chose to model the systems simply as a mathematical model of optimization. This has the advantage of reducing the system to a set of mathematical equations which can be easily modeled by hardware, but may leave some performance gains on the table by not tying the physical process directly to computation.

Analog Accelerators

Analog Accelerators based on the ising model rely on the dynamics of an analog circuit system. These systems use an analog dynamical system to evolve the set of Ising spins to a

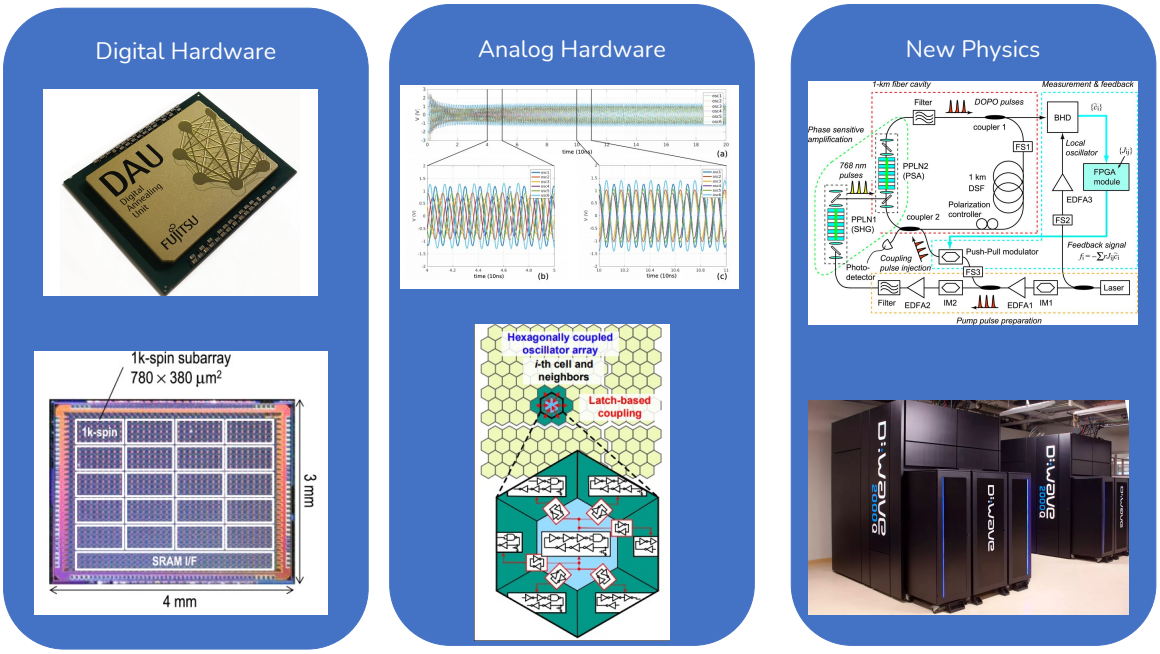

Figure 1.2: Left: Digital Accelerators Based on the Ising model, primarily using annealing mechanisms to reach the ground state [19, 4], Middle: Analog Accelerators based on coupled oscillators [20, 21] Right: Accelerators based on new physics. Top is the "Coherent Ising Machine" based on optical physics, bottom is the DWave Quantum Annealer [22, 18]

global solution [22, 21, 24].

A notable example of analog systems are those based on coupled oscillators which have gained favor recently. These systems use free running oscillators which are coupled together by electronic components. They are perturbed by a signal at twice the oscillation frequency which causes them to fall into the 0 deg or 180 deg phase, a phenomenon known as Sub-Harmonic Injection Locking (SHIL), demonstrated in 1.3. These coupled oscillators have been variously modeled through mathematics in digital systems [20], analog coupled oscillators based on ring oscillators [21], and physical oscillating devices [24].

Physical Accelerators

Physical accelerators rely on the application of physics equations to find ground state solutions of an Ising Model. Instead of using the dynamics of an analog electronics based system, they use the dynamics of a physical system to evolve the Ising model and find the ground state of the system. As the Ising Model itself is a model of interacting systems, this formulation can naturally arise from interacting devices in a system. There are three notable categories of physical accelerators, probabilistic accelerators based on stochastic interacting

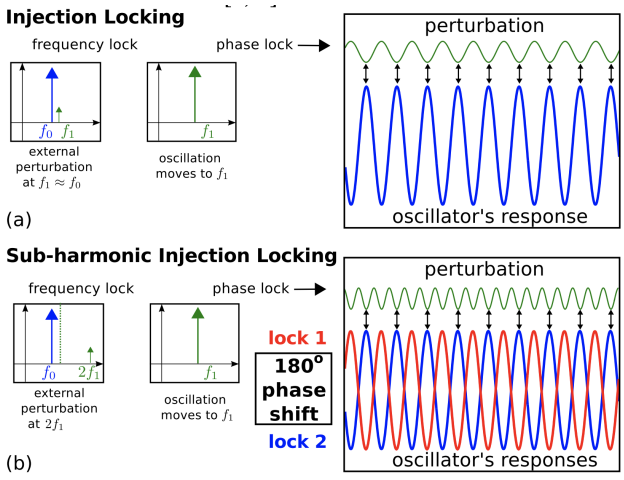

Figure 1.3: Pictoral description of sub-harmonic injection locking, the mechanism used in many oscillator based Ising Machines. The sub-harmonic injection locking allows for different oscillators to be out of phase from one another, representing a +1 and a -1 state. Figure from [23]

elements, optical accelerators using coupled light waves, and the quantum accelerators based on quantum annealing and quantum adiabatic computing.

Probabilistic accelerators usually rely on a stochastic element (such as an unstable magnetic tunnel junction [7]) that produces telegraphic noise that can be biased up or down based on a sigmoidal activation function. Couplings are created by combining the telegraphic outputs of each system into analog or digital inputs that modulate the stochastic input. These probabilistic accelerators are the ones that will be most examined in this thesis.

Organization of Thesis

This thesis explores a series of hardware accelerators and algorithms based on the Boltzmann Machine framework of the Ising Model. The hardware demonstrated ranges from traditional CPU algorithms, to specialized work on the TPU (Tensor Processing Unit) and GPU (Graphics Processing Unit), to lower level hardware development in the form of FPGA (Field Programmable Gate Array) and a specialized ASIC. The thesis is divided into two main parts, the first part explores algorithms based on the Ising Model framework, with an emphasis on methods that have the ability to apply to specific hardware accelerators. The second part of the thesis examines implementation of different accelerated hardware strategies on a field of different hardware accelerators.

Chapter 2

Stochastic Sampling Methods

2.1 Introduction

For most of this thesis, I have chosen to model the Ising Model problem using a probabilistic framework. By using this framework, we can have greater assurances of convergence of our algorithms, as well as the ability to use the rich mathematics and theory from statistical sampling methods. Probability frameworks also allow to mix in Machine Learning based methods, which rely on understanding of the underlying probability distribution. The probabilistic formulation of the Ising Model (known as the Boltzmann Machine [12]) is shown below in Equation 2.1.

$$p(x) = \frac{1}{Z} e^{-E(x)} \tag{2.1}$$

This transformation effectively maps low energy states $\vec{s}$ to high probability states $p(\vec{s})$. It additionally forces all states to have a positive probability. When sampling from the above distribution we can expect that we are more likely to sample low energy states than we are to sample from high energy states. This changes the problem of finding the ground state of the Ising Model to a problem of sampling from the Boltzmann Machine distribution. What makes this distribution hard to sample from absolutely is the presence of the normalizing "partition function" Z, whose definition is shown in equation 2.2.

$$Z = \sum_{x} e^{-E(x)} = \sum_{x_1} \sum_{x_2} ... \sum_{x_n} e^{-E(x_1, x_1, ...x_n)} \tag{2.2}$$

The partition function is especially challenge to calculate as you must sum over all 2^n possible states. This makes it exponentially hard to solve as the size of the state space increases causing it is computationally infeasible to calculate the exact probability of a state within the system, only the unnormalized probability. Because of this, we cannot analytically get samples from this distribution for generic values of the energy function. In general, there are only a few instances of the Ising Model and Boltzmann Machine where the

partition function can be tractably computed, and it is generally in the category of NP-Hard problems [25, 26].

2.2 Monte Carlo Methods for sampling

To take samples from the Boltzmann Machine distribution listed above, many algorithms have been developed. A first motivational algorithm would be simply to numerically sample points to estimate the partition function, and use this to estimate where maximal probability points are. This basic algorithm is outlined below.

Algorithm 1 Naive Sampling

Require: Distribution $p(x) = \frac{\hat{p(x)}}{Z_p}$ of dimension N Samples Desired N_s
  while $i \leq N_s$ **do**
    Generate $U_i \sim Binom(0.5)^N$
    Calculate $\hat{p}(U_i)$
  end while
  $\hat{Z}_p = \sum_{j=1}^{N_s} \hat{p}(U_i)$

By the law of large numbers, we will expect that the partition function $\hat{Z}_p$ will converge to the desired partition function Z (equivalent to saying that $\mathbb{E}[Z_p] = Z$ and as it does so we would be able to have a better estimate for probabilities of each sample. Once we have a $\hat{Z}_p$ that we are sufficiently happy with, then we have the ability to get normalized samples from the distribution of interest, using a variety of discrete sampling methods [27]. However, this overlooks a few issues:

- Although each U_i is easy to generate (as it is just a uniformly distributed random 0-1 vector), this doesn't represent the distribution we are interested well.
- It may take many draws of $p(U_i)$ to get the value of Z_p to converge
- For some distributions, most of the probability mass may be centered as a particular point or area in the probability space (i.e. $p(x) >> p(y) \forall y$. If you don't sample from this particular point, your estimates for Z_p will be incorrect.
- In high dimensional spaces (for $N >> 100$), the geometry of the space starts to make the probability mass further centered around certain points or areas. [28]

These points motivate us to develop alternate sampling schemes which try to address these problems. In particular the last 2 points are the ones that are most pertinent for Ising Model problems, as in many cases we are interested in finding the configurations with highest probability or lowest energy, which would require sampling the highest probability points.

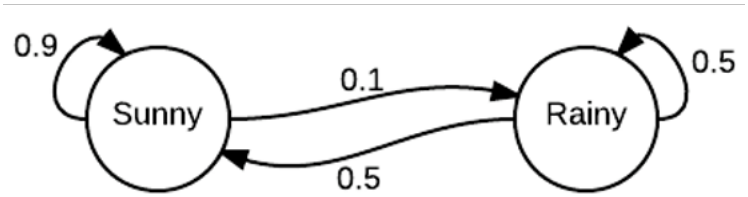

Figure 2.1: Simple Markov Chain illustrating a 2 state system

2.3 Markov Chains

Markov chains are defined as a stochastic process which are purely defined by state transitions. In particular, they are processes that are fully described by their "one step" transitions, which are the transition probability to go from one state to the next. This is mathematically shown below in equation 2.3

$$p(x_{n+1}|x_n, x_{n-1}, ...x_1) = p(x_{n+1}|x_n) \tag{2.3}$$

A simple example of a markov chain model in real life is shown in Figure 2.1, here each state has a probability of staying in it's current state, or transitioning into the other. In this case, if it is currently sunny outside, there is a 0.9 probability that the next day will be sunny and 0.1 chance of rain the next day. Similarly if it is rainy, there is a 0.5 probability that it will be rainy tomorrow, and a 0.5 probability it will be sunny. This 2 state chain can be understood by the transition probability matrix P given below it.

$$P = \begin{pmatrix} 0.1 & 0.9 \\ 0.5 & 0.5 \end{pmatrix} \tag{2.4}$$

The state transition matrix P can operate on an initial vector of probabilities x to generate the state probabilities for the next step. This is shown in the equation below.

$$x_{n+1} = x_n P \tag{2.5}$$

Certain classes of Markov Chains (specifically those which are aperiodic and irreducible [29])) admit what is known as a "stationary distribution". This can be understood as the long run state probabilities or limiting distribution if the transition matrix were applied to the initial distribution. The interesting piece about markov chains which admit stationary distributions is that it does not matter what the initial state is, with sufficient applications of the transition matrix the distribution will always approach the stationary one.

$$\pi_0 = \pi_0 P \tag{2.6}$$

$$xP^n = \pi_0 \quad \forall x, n \to \infty \tag{2.7}$$

These markov chains admit an interesting set of properties that can be used to help us sample from most distributions. Specifically, if we can design a Markov Chain whose stationary distribution is the distribution we want to sample from, then we would be able to pull samples from this distribution without full knowledge of normalizing constants and similar quantities.

Detailed Balance

For all states s_i and s_j in the state space S, the product of the probability of transitioning from s_i to s_j (i.e., P_{ij}) and the probability of being in s_i (i.e., the equilibrium distribution π_i) is equal to the product of the probability of transitioning from s_j to s_i (i.e., P_{ji}) and the probability of being in s_j (i.e., the equilibrium distribution π_j):

$$\pi_i \cdot P_{ij} = \pi_j \cdot P_{ji} \tag{2.8}$$

In other words, the rate at which the Markov chain transitions from state s_i to state s_j under the equilibrium distribution is equal to the rate at which it transitions from state s_j to state s_i.

The equilibrium distribution π is a stationary probability distribution, meaning that if the Markov chain starts in this distribution, it will remain in the same distribution as time progresses. The detailed balance condition ensures that the Markov chain reaches an equilibrium state where the probabilities of being in different states are constant over time.

Detailed balance is a sufficient (but stronger than necessary) condition for Markov Chain to admit a stationary distribution. By constructing a Markov Chain that obeys detailed balance, we can make assumptions about the presence of a stationary distributions, and design the stationary distribution to converge to the distribution we want to sample from.

2.4 Markov Chain Monte Carlo Sampling

Introduction

With the background of Markov Chains and sampling techniques, we can begin to understand Markov Chain Monte Carlo algorithms. Specifically, if we can use the stationary distribution of a Markov Chain to encode a probability distribution that we wish to sample from, and the Markov Transition Operator to progress towards that distribution, we can get samples from complicated distributions. Below we demonstrate two algorithms that would allow us to do this, both of which use the principle of detailed balance [29]. The first is the Metropolis-Hastings (MH) algorithm which is popular for simulation of the Boltzmann Machine distribution.

The MH algorithm was the first demonstration of a MCMC algorithm which uses the detailed balance conditions to sample from an arbitrary distribution [31]. It relies on using a combination of an acceptance probability and a proposal distribution to satisfy these

Algorithm 2 Metropolis-Hastings Algorithm

Require: Distribution $p(x) = \frac{\hat{p(x)}}{Z_p}$ of dimension N, Samples Desired N_s, proposal distribution $g(x)$
  Initialize State x_0
  for $i = 1$ to N_s **do**
    Generate $x' \sim g(x|x_i)$
    Calculate Acceptance Probability $A(x', x_i) = min(1, \frac{P(x')g(x_i|x')}{P(x_i)g(x'|x_i))}$
    Generate $U \sim Unif(0, 1)$
    if $A(x', x_i) > U$ **then**
      $x_{i+1} = x'$
    else
      $x_{i+1} = x_i$
    end if
  end for
  $\hat{Z}_p = \sum_{j=1}^{N_s} \hat{p}(x_i)$

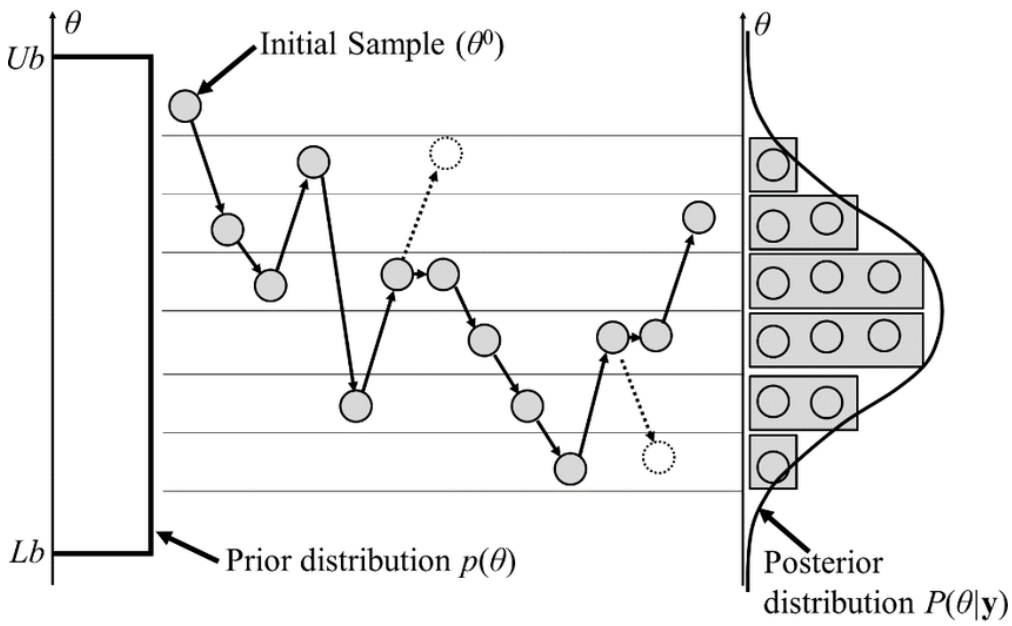

Figure 2.2: Simple figure to understand Metropolis Hastings Algorithm. A proposal distribution (here shown as prior distribution $p(\theta)$) is used to produce samples given previous states. These states produce a markov chain which is then used to sample from the more complicated distribution $P(\theta|y)$. Figure copied from [30]

detailed balance conditions as well. In general the MH algorithm can be tuned with a proposal distribution that allows for a high acceptance probability generally and that can move around the state distribution. In general, the usefulness of the algorithm, especially for more complicated distributions, hinges on a good choice for a proposal distribution. Especially for very high dimensional state spaces, it can be difficult to put together a good proposal distribution.

This algorithm also has a special use case for sampling of the Boltzmann Distribution, in that it gets around the partition function sampling updates. When we are proposing moves, we are calculating the ratio of the same probability distribution for different states $p(x')/p(x)$. This allows us to calculate the acceptance ratio without using the partition function, only using the unnormalized probability $\hat{p}(x)$ to calculate the next steps.

Algorithm 3 Gibbs Sampling Algorithm

Require: Distribution $p(x) = \frac{\hat{p(x)}}{Z_p}$ of dimension N, Samples Desired N_s, conditional distribution $p(x \mid y)$, Updates per sample k
  Initialize $x_1, x_2, \ldots, x_n$ with random initial values.
  for $t = 1$ to N_s **do** ▷ Number of iterations
    for $j = 1$ to k **do**
      Pick a random index $i = 1 \ldots n$
      Sample x^i from the conditional $p(x^i_{t+1} \mid x^1_{t+1}, x^2_{t+1}, \ldots, x^{i-1}_t, x^{i+1}_t, \ldots, x^n_t)$.
    end for
  end for

The Gibbs sampler uses a series of serial updates of each variable for the update of the full state of the system. It relies on the existence of closed form conditional probability for each of the variables in the distribution rather than a proposal distribution, and each conditional probability update can be seen as an individual Markov Chain that obeys detailed balance. Many practical implementations of the Gibbs sampler update each of the states sequentially x^1 to x^N, but the sequential form of the Gibbs sampler does not obey the detailed balance equations [29].

The conditional probability for a Gibbs Sampler of the Boltzmann Machine is also well defined, and can be used to efficiently calculate the update probability for a given state. This has been derived in many other works and will not be re-derived here (see, for example [32, 33]).

$$p(x^i_{t+1} = 1 \mid x^1_{t+1}, x^2_{t+1}, \ldots, x^{i-1}_t, x^{i+1}_t, \ldots, x^n_t) \tag{2.9}$$

$$= \frac{p(x^i)}{p(x^1_{t+1}, x^2_{t+1}, \ldots, x^{i-1}_t, x^{i+1}_t, \ldots, x^n_t)} \tag{2.10}$$

$$= \sigma(\sum_j J_{ij} x_j + b_i) \tag{2.11}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

The Gibbs sampler for Ising Model problems is very useful as it also is able to get around the problem of the partition as well, due to the cancellation of the partition function in the conditional probability calculation. This reduces the problem of sampling each conditional to a multiply and accumulate operation, followed by a sigmoid. The original Gibbs sampler and Boltzmann machine found favour as a method to solve various combinatorial optimization problems[15]. However, convergence of the sampling and training schemes for use on the Boltzmann machine has been slow[13].

Additionally, the Gibbs sampler requires mainly serial updates of the conditional probabilities, which can cause convergence to be extremely slow and many samples to be taken. Some of the other problems that come from Gibbs samplers will be discussed below and in later chapters.

Convergence of Sampling Algorithm

Markov Chain Samplers (and specifically the Gibbs Sampler mentioned above) have been proven to converge in a geometric rate with the number of samples. In addition, the distance in total variation between the sampled distribution and the model distribution strictly decreases with each Gibbs sampling step [34]. This implies that the quality of our solution should increase as the sampled distribution approaches equilibrium. This is especially useful for optimization problems where run time can be traded directly for better solutions.

Many combinatorial optimization problems are highly multimodal, which can lead to problems of getting stuck in local minima and spurious modes during MCMC. Once the MCMC algorithm finds a low energy mode it tends to stay around it. To leave the mode, the Markov Chain must do a traversal from one of these modes to another, which goes as p^n where p is the probability of transition between states and n is the number of transitions needed to go from one mode to another. As the distribution becomes increasingly multimodal, pseudo-convergence of the MCMC becomes a problem. This is demonstrated in Figure 2.3 in a toy form.

Of course, it is considered NP-Hard to solve these optimization tasks, so we cannot expect the sampling procedure to converge in sub-exponential time on problems which themselves are NP-Hard [35]. However, there are many algorithms that can help speed up convergence of the sampler to find the ground state of the system.

An interesting property of convergence of the Gibbs Sampler, specifically within the Boltzmann Machine framework is that it is directly related to the maximum energy difference (and thus relative size of couplings J_{ij}) between states. This means that distributions that are closer to uniform are easier to sample from (small weights and couplings), while distributions that have large weights and biases become increasingly difficult to sample from [34, 13].

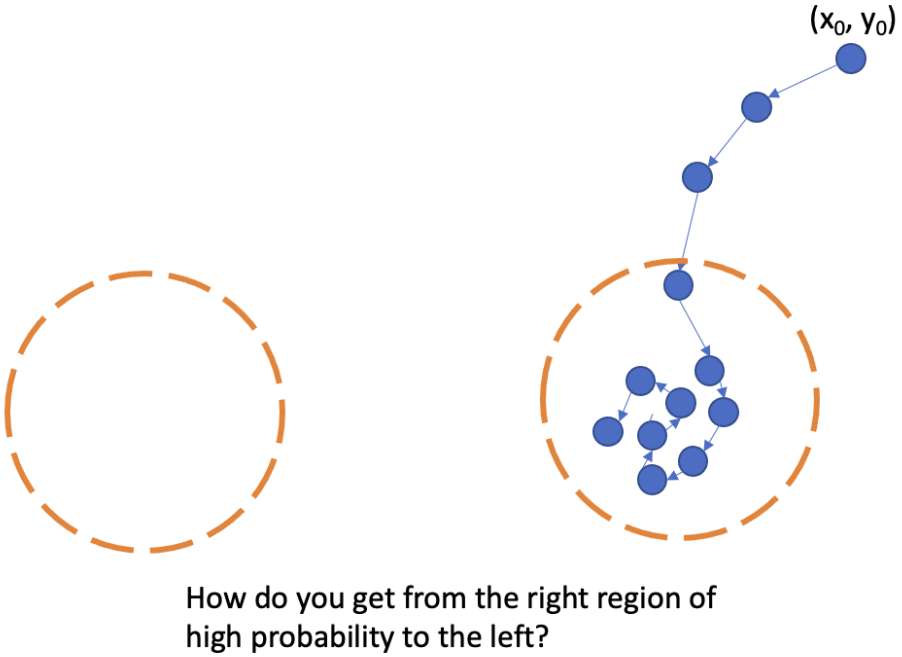

Figure 2.3: A demonstrative example of the difficulties in performing sampling in a bimodal distribution. The sampler starts in an area of low probability, and will get stuck in one of the modes. The sampler then has to traverse the areas of low probability between the two modes, which becomes increasingly difficult if the distance between the modes are large.

Hitting time vs. Mixing Time

The time domain sampler also shows a large difference between time taken to model the full distribution (the mixing time) and the time taken to sample the correct solution to the problem of interest once (the hitting time). We demonstrate this phenomenon in Figure 2.4 for the application of a factorization problem. In this instance, we are looking for the correct answers to a factorization problem, and see how long it takes for the system to hit the correct answer once. The difference in time between the mixing time and hitting time is shown directly in Figure 2.4 C), with the difference in times being close to 4x. By adding in a sample verification methodology, the time taken to identify correct factors can be drastically decreased. For NP-Hard problems, the solution can be verified in polynomial time but finding a given solution is done within exponential time, a feature which is also present in this factorization problem. This means the relative overhead of checking of factors is relatively low, and can be done on a regular basis to reduce the sampling time significantly. This method of adding heuristics to the sampler is present in many different problem types, and must be done differently for each type of problem. Here we demonstrate both that the sampler approaches the correct distribution, and that effective heuristics using the hitting

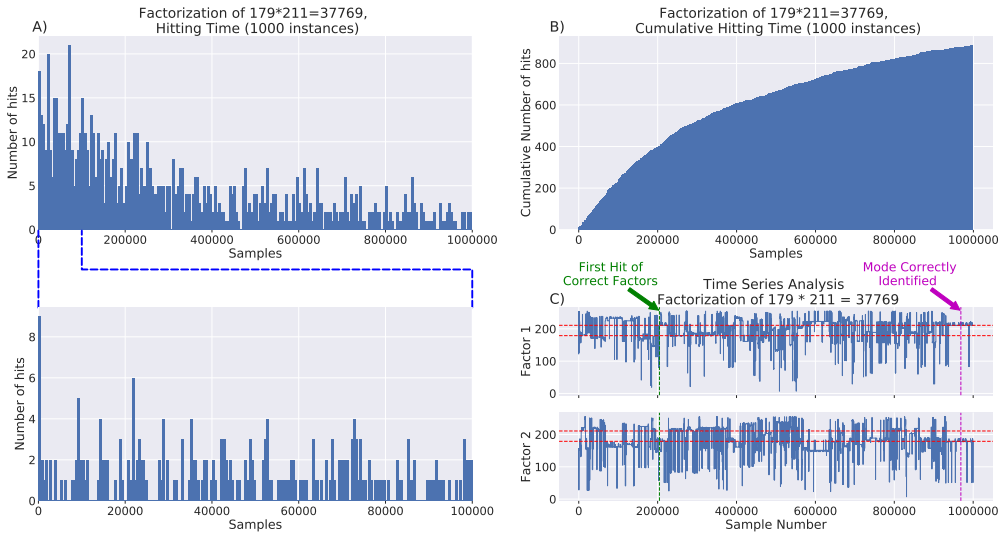

Figure 2.4: **Understanding the hitting time vs. the mixing time of a sampler**A demonstrative example of the difference between hitting and mixing time for a given sampler. Here we examine the task of using MCMC to factorize a number. **A)** The distribution of hitting time for the MCMC sampler to factorize the number. We see that it roughly follows a poisson type distribution. **B)** The cumulative distribution function for the hitting time follows a logarithmic curve **C)** Zoomed in to part A) demonstrating the distributional nature of the system. **D)** Time series analysis showing almost a 4x difference in the time for mixing time and hitting time.

time are available to reduce the time to solution.

To create a method to take advantage of the difference between mixing time and hitting time, we can need to design hardware and algorithms for quickly verifying the answers to the problem of instance. In the following chapters we explore how this is accomplished in two different types of systems, the FPGA system and the GPU system. By using the approximate probability of the sample taken, and taking the sample with highest probability, we can effectively say that the solution to the problem is the sample that possesses the highest probability.

Parallel Tempering (Replica Exchange Monte Carlo)

Parallel tempering, also known as replica exchange Monte Carlo, is a powerful Markov chain Monte Carlo (MCMC) method used to improve the sampling efficiency of complex probability distributions, especially in high-dimensional spaces or when dealing with multimodal distributions. [33, 36]

The main idea behind parallel tempering is to run multiple copies (replicas) of the MCMC algorithm in parallel, each at a different "temperature." The temperature parameter is a

scalar that controls the exploration behavior of the Markov chain. Higher temperatures correspond to more uniform exploration of the state space, while lower temperatures focus on fine-tuning the exploration around local modes. In the Boltzmann Machine framework, a different temperature is applied by scaling the Energy Function of the original Boltzmann Machine. In this context we are using the formulation of inverse temperature where $\beta = 1/T$.

$$p_\beta(x) = \frac{1}{Z_\beta} e^{-\beta E(x)} \tag{2.13}$$

Conceptually, for values of $\beta \to 0$ the distribution of the Ising model becomes closer to sampling from a uniform distribution, allowing for the markov chain to progress quickly through the state space. As $\beta \to 1$, where we are sampling from the exact distribution that we have constructed.

The algorithm proceeds as follows:

- Initialization: Create M replicas of the system, each starting from different initial states.
- Temperature Ladder: Assign a temperature to each replica, typically on a logarithmic scale, ranging from low to high temperatures.
- Simultaneous Evolution: Run each replica independently using an MCMC algorithm, such as Metropolis-Hastings or Gibbs sampling, but with the added modification of temperature-dependent proposal distributions.
- Replica Exchange: Periodically attempt exchanges of states between replicas with neighboring temperatures. Higher temperature replicas are more likely to accept moves from lower temperature replicas, allowing the replicas to explore different regions of the state space effectively.

Parallel tempering is particularly useful when the target distribution has rugged energy landscapes or multiple modes, where a single MCMC chain may get trapped in a local mode and struggle to explore the entire space efficiently. By using multiple replicas with different temperatures, parallel tempering encourages better exploration and enhances the chances of finding global modes. This alleviates the problem shown in Figure 2.3, as it allows many chains to proceed independently and explore different areas of the state space, while exchanging pieces to sample from the distribution of interest.

2.5 The Restricted Boltzmann Machine (RBM) for Parallelized Sampling

The Restricted Boltzmann Machine is a stochastic neural network that encodes an exponential family probability distribution over binary variables, taking the form given in Equation

Algorithm 4 Parallel Tempering

Initialize M replicas of the system with different initial states $x_{i,0}$.
for $t = 1$ to T **do** ▷ Number of iterations
  for $i = 1$ to M **do** ▷ Update each replica in parallel
    Sample $x_{i,t}$ using Preferred Method (Gibbs, Metropolis Hastings, etc.)
  end for
  for $i = 1$ to $M - 1$ **do** ▷ Replica exchange
    Generate a random number $u \sim \text{Uniform}(0, 1)$.
    Calculate $P_{swap} = \frac{p_{i+1}(x_{i,t})p_i(x_{i+1,t})}{p_{i+1}(x_{i+1,t})p_i(x_{i,t})}$
    if $u < \min(1, P_{swap})$ **then**
      Exchange the states of replicas i and $i + 1$: $x_{i,t} \leftrightarrow x_{i+1,t}$.
    end if
  end for
end forreturn Samples from all replicas: $x_{1,T}, x_{2,T}, \ldots, x_{M,T}$.

2.14. The energy function $E(v, h)$ is typically set to reflect the problem being solved, and can take a variety of possible forms [29]. Here we choose the energy function to reflect linear synaptic weights with two body interactions to allow for simplicity of hardware and algorithmic implementation.

$$p(v, h) = \frac{1}{Z} e^{-E(v,h)}, v \in \{0, 1\}^n, h \in \{0, 1\}^m \tag{2.14}$$

To sample from the RBM probability distribution, we perform block Gibbs sampling [13], a type of Markov Chain Monte Carlo, on the RBM nodes. Each neuron has a stochastic activation function where $p(v_i = 1|h) = \sigma(w_i^T h + b_i)$ and $\sigma(x) = (1 + e^{-x})^{-1}$, with w_i being the ith row vector of the weight matrix, and b_i being the bias associated with that neuron. The lack of intra-layer connections allows for each neuron in a layer to be sampled in parallel, creating a massively parallel sampling scheme. Each layer is sampled and the result is passed to the next layer to get the next sample.

The conditional nature of the Blocked Gibbs Sampling means that many neurons can be updated in parallel. On standard computing CPU platforms, this allows for vectorized updates using standard linear algebra libraries to perform the matrix multiplication in a parallel fashion (using baseline SIMD instructions and the like). On more parallel friendly platforms, such as FPGAs, GPUs, and TPUs, this parallelism can be exploited even further due to the parallel cores available for computation. This is especially useful on large and very-large problem scales, where many nodes can be updated in parallel. These ideas of exploiting parallelism will be explored further in the following chapters.

Chapter 3

Inverse Logic Algorithms

3.1 Introduction

In the previous sections, we have demonstrated that the Boltzmann Machine and Restricted Boltzmann Machine can be used to model complex probability distributions, and introduced methods for finding solutions to them. In recent years, the Restricted Boltzmann Machine (RBM) has experienced a resurgence as a generative model that is able to fully approximate a probability distribution over binary variables due to its ease of computation and training via the contrastive divergence method [13]. However, the success of the RBM as a generative model has been limited due to the difficulties in running the Markov chain Monte Carlo (MCMC) algorithm to convergence [37, 38].

Here, we use the RBM as generative model composed of multiple learned modules that is able to solve a larger problem than the individually trained parts. This allows us to circumvent the problem of training large modules (which is equivalent to solving the optimization problem in the first place, as we are simply providing the correct answers to the module as training data), thus minimizing training time. As RBMs have the ability to fill in partial values and solutions, this approach is very flexible to the broad class of combinatorial optimization problems which can be composed of individual atomic operations or parts. Most notably, we show that our approach of using "invertible boolean logic" is a method of solving the boolean satisfiability problem, which can be mapped to a large class of combinatorial optimization problems directly [39, 40]. The ability of RBMs to fully model a probability distribution ensures model convergence and gives ideas about the shape of the underlying distribution.

3.2 The Restricted Boltzmann Machine

An RBM is a bipartite graph model comprising a visible layer (input layer) and a hidden layer. The neurons within each layer are connected, but there are no connections within the same layer. The state of the visible layer is determined by the input data, while the hidden

layer captures important features and interactions between the visible units. The energy function of an RBM is given by:

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{ij} h_j \tag{3.1}$$

where v represents the visible layer state, h represents the hidden layer state, a_i and b_j are biases, and w_{ij} denotes the weight between visible unit i and hidden unit j.

3.3 Contrastive Divergence Learning

Contrastive Divergence is an iterative algorithm that aims to approximate the gradient of the log-likelihood of the training data with respect to the model parameters. The CD learning procedure is as follows:

Algorithm 5 Contrastive Divergence Learning on RBM

1: Initialize RBM parameters: a_i, b_j and w_{ij}.
2: Sample a training data instance v_{data}.
3: Perform Gibbs sampling to obtain the hidden layer sample h_{data}:
4: Sample h_{data} from $P(h|v_{\text{data}})$ using the sigmoid activation function.
5: **for** $t = 0$ to T **do**
6: Sample v_{recon} from $P(v|h_{\text{data}})$.
7: Sample h_{recon} from $P(h|v_{\text{recon}})$.
8: **end for**
9: Update gradients using data and reconstructions:
10: $\Delta w_{ij} = \epsilon \left(v_{\text{data}} h_{\text{data}}^T - v_{\text{recon}} h_{\text{recon}}^T\right)$
11: $\Delta a_i = \epsilon \left(v_{\text{data}} - v_{\text{recon}}\right)$
12: $\Delta b_j = \epsilon \left(h_{\text{data}} - h_{\text{recon}}\right)$
13: Update RBM parameters:
14: $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$
15: $a_i \leftarrow a_i + \Delta a_i$
16: $b_j \leftarrow b_j + \Delta b_j$
17: Repeat steps 2-9 for a number of training iterations.

In Algorithm 5, $P(h|v)$ and $P(v|h)$ denote the conditional probabilities of the hidden and visible units, respectively, given the states of the other layer. T represents the number of Gibbs sampling steps performed during CD.

3.4 Merging RBMs

The difficulty of training large RBMs means that new innovations are necessary at the training step. [37, 38, 41] In this work, we propose merging smaller models, that are already

trained, to form an initial condition for larger models as a way of improving the training of large RBMs. This methodology is inspired from the digital logic where larger functions can be constructed by combining small functional blocks [42]. Notably, all NP-hard problems can be formulated through the Boolean Satisfiability problem. Therefore, constructing the RBM in the aforementioned way provides a natural approach to solving hard optimization problems. An example of this is shown in Figure 3.1 F) where we solve a toy example of a 3SAT Boolean Satisfiability problem. This shows that this method has the representational power to solve a wide variety of NP-Hard problems. There has been many efforts to combine individual machine learning and statistical models through ensemble learning methods[43]. In a similar vain, RBMs have been combined to produce joint predictions across multiple problem domains [44]. There has also been work on using fully connected Boltzmann Machines to solve similar problems [42, 45] with some success. Our approach is unique in its usage of machine learning models and Restricted Boltzmann Machines which give it better performance than fully connected models, while offering a parallelized sampling scheme which can be exploited by an efficient hardware accelerator.

Merging is performed by combining RBMs along their common visible neuron connection. We show the mechanics of the merging process in Figure 3.1 C) and D) where we combine digital logic gates together in this manner. Merging across the visible neurons like this retains the bi-partite, *product of experts* nature of the RBM while giving the expected distribution if we were combining gates to perform logical synthesis. Using this as inspiration, we construct adders and multipliers as shown in Figure 3.1 E) to combine trained n-bit adder and multiplier units into 2n-bit multiplier units. Detailed information of this merging protocol has been provided in the Methods section.

After smaller models have been trained and merged, we retrain the larger models to fine-tune them. As shown in Equation 3.12, the models are good approximations for the correct distribution we are interested in, and provide a good initial conditions for training of the final model. Importantly, merging in this way retains the bi-directional nature of the network. The same RBM can be queried to solve what the output is for a given input (the "forward" direction), and queried what outputs caused a given input (the "reverse" direction). In Figure 3.2 A), B), and C) we see the consequence of the bi-directional nature where the same model can perform multiplication, division and factorization, as it learns the full joint distribution over variables. In these tasks, the model must get the exact solution as the mode of the sampled distribution. Performance is reported as $p_{correct}$ whi en number of samples. We sample for the given number of samples and check whether the mode of the sampled distribution corresponds to the correct answer to the problem of interest. We also validate our merging approach by showing in these figures that training of an RBM via Contrastive Divergence without merging catastrophically fails in these problems, while our merging procedure allows for successful training. The blue curve in Figures 3.2 A), B), C) represents the baseline of trying to train this machine on the 16-bit factorization via Contrastive Divergence, showing close to 0 success probability, even after 100 hours of training (see Table 3.1 for further details). Merging smaller units alone boosts performance to the green curve, while using these merged units as an initial condition for further fine-

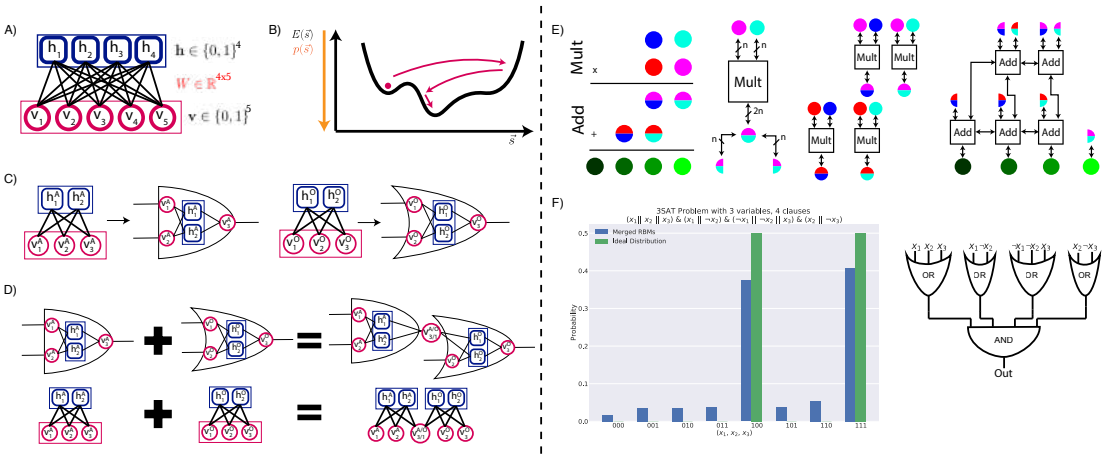

Figure 3.1: **Demonstration of RBM structure and sampling algorithm**
(A) Structure of the RBM neural network. The Restricted Boltzmann Machine is a binary neural network structured in a bipartite graph structure. **(B)** The RBM maps out the non-convex state space of a probability distribution. Low energy states map to high probability states which the network identifies through a Markov Chain Monte Carlo (MCMC) algorithm. **(C)** A graphical mapping of RBMs to gate level digital circuits. The visible nodes correspond to the inputs and outputs of the logic gate, and the hidden nodes are the internal representation of the logic gate. **(D)** Graphical Demonstration of the merging procedure, showing how two RBMs which represent an AND gate and an OR gate can be merged together to form a connection. **(E)** We can create arbitrary adders and multipliers by merging together smaller units to create the logical equivalent of larger units. The leftmost image shows how we create a 2n bit multiplier using n bit multiplications and n bit additions. The color coding shows how the partial products are broken apart amongst the adders and multipliers. To the right of that we show how we perform 4, n-bit input $2n$-bit output multiplications, and then accumulate the result. **(F)** Using this strategy of merging logical units to solve a simple 3SAT, Combinatorial Optimization problem.

tuning boosts performance further to the red curve. This result proceeds from the fact that the error in KL-divergence of the combined distribution is approximately the sum of the originals (see Section 3.5 for further details). Indeed, if there were no error in the models before merging, there would be no error in the final model after merging. The fine-tuning step is simply to reduce the error created by merging imperfect models together.

By merging together small RBMs using the principles of logic synthesis, many complex and NP-Hard problems can be solved using the bi-directional nature of RBMs. In Figure 3.1 D) we see the mechanics of this procedure, while Figure 3.1 E) shows how it can apply to integer factorization and Figure 3.1 F) shows how it can apply to 3SAT and boolean satisfiability. We note that the logic gates formed in part 3.1 C) can be operated in reverse to find solutions to the given satisfiability problem with 3SAT. This is the canonical example

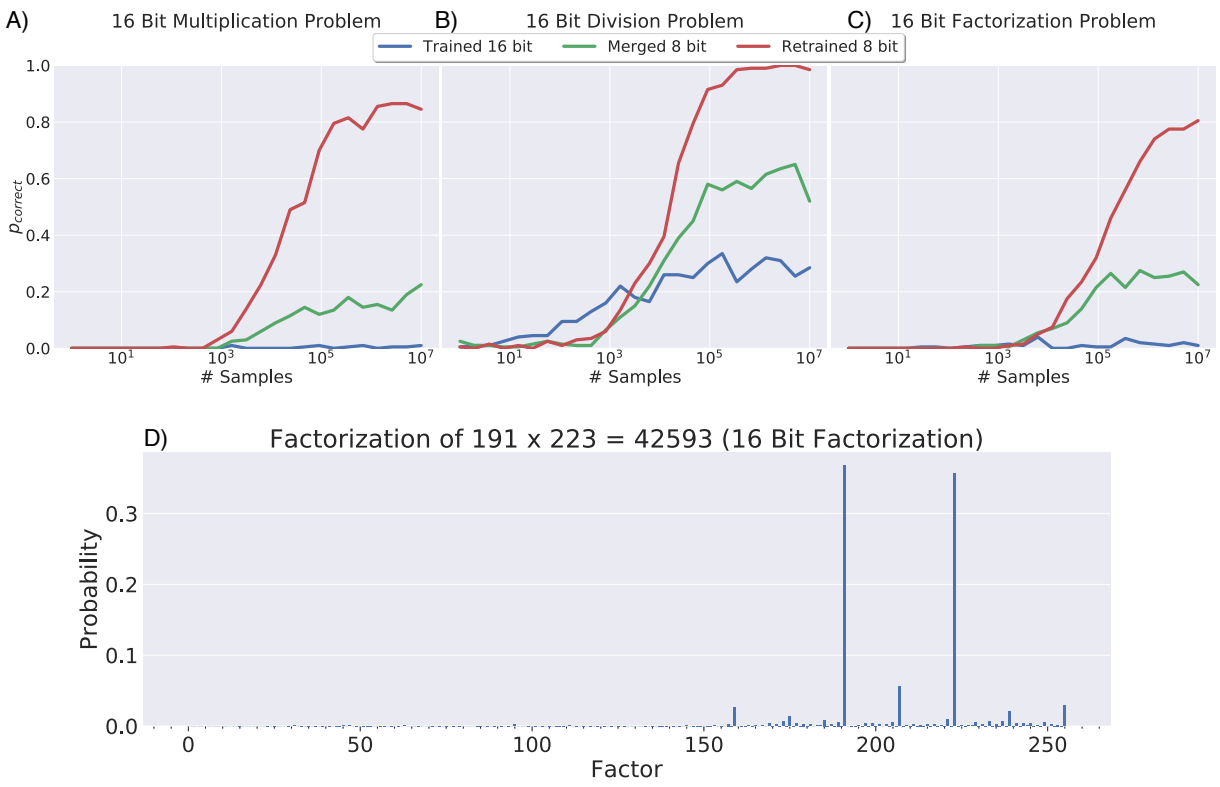

Figure 3.2: **Performance on 16 Bit Multiplication, Division and Factorization (A), (B), (C)** Showing the performance on multiplication, division, and factorization performed by directly training a 16 bit network (trained 16 bit), merging two 8 bit networks (merged 8 bit), or merging two 8 bit networks and retraining (retrained 8 bit). **(D)** An example of a factorized distribution after 10^7 samples showing factorization of a 16 bit number into its two prime factors. The sampled distribution shows clear peaks at the two correct answers to the factorization problem.

of an NP-Hard problem, which maps directly to all other NP-Hard problems, showing that these networks have the representational power to solve a variety of such problems. [39, 40]. These problems are at the heart of many computationally difficult problems. We additionally show how this type of training can outperform directly trained models. Figure 3.2 A), B) and C) show that merging smaller models and retraining them drastically outperforms directly training the model, with errors close to 3-10x less for problems of interest. In Figures 3.2 D) we show how these models can factor a semi-prime number into its two factors with high accuracy. The factorization of a semi-prime number into its two co-primes is at the heart of the RSA cryptography algorithm and is the basis of most of modern encryption systems.

3.5 Mathematical details of Merging RBMs

Given two RBMs we wish to merge along a common connection (see Figure 3.1) with the following parameters: $W_A \in \mathbb{R}^{n\times r}$ and $W_B \in \mathbb{R}^{m\times s}$, visible biases $b_A \in \mathbb{R}^n$ and $b_B \in \mathbb{R}^m$, and hidden biases $a_A \in \mathbb{R}^r$ and $a_B \in \mathbb{R}^s$. The energies and probabilities of these are as follows:

$$\begin{aligned} E_A(v,h) &= -v^T W_A h - a_A^T h - b_A^T v; \;\; p_A(v,h) = \frac{1}{Z_A} e^{-E_A(v,h)} \\ E_B(v,h) &= -v^T W_B h - a_B^T h - b_B^T v; \;\; p_B(v,h) = \frac{1}{Z_B} e^{-E_B(v,h)} \end{aligned} \tag{3.2}$$

We can write the weight matrices as a series of row vectors corresponding to one visible unit's connections to a set of hidden units.

$$W_A = \begin{bmatrix} —w_1^A— \\ \vdots \\ —w_n^A— \end{bmatrix}, \;\; W_B = \begin{bmatrix} —w_1^B— \\ \vdots \\ —w_m^B— \end{bmatrix}, \tag{3.3}$$

With this definition, the merge operation is shown below. If unit k of RBM A is merged with unit l of RBM B the associated weight matrix $W_{A+B} \in \mathbb{R}^{(n+m-1)\times(r+s)}$, visible bias $v_{A+B} \in \mathbb{R}^{n+m-1}$ and hidden bias $h_{A+B} \in \mathbb{R}^{r+s}$ dictate the probabilities and energies for the merged RBM. Merging multiple units between these two RBMs corresponds to moving multiple row vectors from W_B to W_A, which creates the associated decrease in dimensionality of W_{A+B} and b_{A+B} (where $W_{A+B} \in \mathbb{R}^{(n+m-d)\times(r+s)}$ and $v_{A+B} \in \mathbb{R}^{n+m-d}$ where d is the number of merged units.

$$W_{A+B} = \left[\begin{array}{c|c} W_A & \begin{matrix} 0 \\ —w_l^B— \\ 0 \end{matrix} \\ \hline 0 & W_{B\setminus l} \end{array}\right] = \left[\begin{array}{c|c} \begin{matrix} —w_1^A— \\ \vdots \\ —w_k^A— \\ \vdots \\ —w_n^A— \end{matrix} & \begin{matrix} 0 \\ —w_l^B— \\ \\ 0 \end{matrix} \\ \hline & \begin{matrix} —w_1^B— \\ \vdots \\ —w_{l-1}^B— \\ —w_{l+1}^B— \\ \vdots \\ —w_m^B— \end{matrix} \end{array}\right] \tag{3.4}$$

$$b_{A+B} = \begin{bmatrix} b_1^A \\ \vdots \\ b_k^A + b_l^B \\ \vdots \\ b_n^A \\ b_1^B \\ \vdots \\ b_{l-1}^B \\ b_{l+1}^B \\ \vdots \\ b_m^B \end{bmatrix} \quad a_{A+B} = \begin{bmatrix} a_1^A \\ \vdots \\ a_r^A \\ a_1^B \\ \vdots \\ \\ a_s^B \end{bmatrix} \tag{3.5}$$

Below, we show how this relates to the original energies and probabilities. The vectors v and h correspond to the visible vector put into the combined RBM, while v_A, v_B, h_A and h_B correspond to the equivalent state vectors that would be inputted into the single RBMs. Using these equations, we can see that the combined RBM energy factorizes into a sum of the original RBM energies and the probability is the product of the original probabilities.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_{n+m-1} \end{bmatrix} \quad h = \begin{bmatrix} h_1 \\ \vdots \\ h_{r+s} \end{bmatrix} \tag{3.6}$$

$$v_A = \begin{bmatrix} v_1 \\ \vdots \\ v_l \\ \vdots \\ v_n \end{bmatrix}, v_B = \begin{bmatrix} v_{n+1} \\ \vdots \\ v_{n+l-1} \\ v_l \\ v_{n+l} \\ \vdots \\ v_{n+m-1} \end{bmatrix}, \tag{3.7}$$

$$h_A = \begin{bmatrix} h_1 \\ \vdots \\ h_r \end{bmatrix} h_B = \begin{bmatrix} h_{r+1} \\ \vdots \\ h_{r+s} \end{bmatrix} \tag{3.8}$$

$$E_{A+B}(v,h) = E_A(v_A,h_A) + E_B(v_B,h_B), \tag{3.9}$$

$$p_{A+B}(v,h) = \frac{1}{Z_{A+B}} e^{-E_{A+B}(v,h)} \propto p_A(v_A,h_A) p_B(v_B,h_B) \tag{3.10}$$

Because of the probabilities approximately multiplying (Eq. 3.11), we can also say that if each of the distributions differed from the "ideal" distribution (denoted here by q), then we can expect the error (as measured by the KL divergence eqn. 3.12) to increase approximately

Model (minutes)	Visible Units	Hidden Units	Training Time
1 bit Adder	5	6	1
2 bit Adder	8	28	13.5
4 bit Adder	14	64	133
8 bit Adder	26	96	201
16 bit Adder	50	128	321
32 bit Adder	98	192	13000 (approx.)
4 bit Multiplier	8	16	46
6 bit Multiplier	12	48	79
8 bit Multiplier	16	64	655
10 bit Multiplier	20	144	4063
12 bit Multiplier (Merged)	60	352	512
16 bit Multiplier	32	512	3611
16 bit Multiplier (Merged)	78	576	5156

Table 3.1: **Model sizes and training times for various RBMs** As the size of the RBM grows, both the training time for model convergence and the number of hidden units needed to model the distribution both increase. We find that the largest multiplier model that can be fit on our version of the FPGA design is the 16 Bit Multiplier/Factorizer. Model analysis and training for models larger than the 16 bit multiplier took too long to train, and are not displayed here.

linearly with the number of distributions summed together. As contrastive divergence learning approximately follows the gradient of the KL divergence, the merged model represents a good initial condition for training of the larger model [46]. This means that only small corrections in CD training are needed on the merged model to create a good trained model for the larger network. Training the merged model is possible as intermediate nodes are represented as extra visible units, and can be calculated based on the input and outputs of the dataset. The data for the merged models can be calculated as if we were propagating values through a digital circuit and keeping track of intermediate values, which become the data for the merged model to be trained on.

$$p = p_A(v_A, h_A)p_B(v_B, h_B), \quad q = q_A(v_A, h_A)q_B(v_B, h_B) \tag{3.11}$$

$$D_{\mathrm{KL}}(p\|q) \approx D_{\mathrm{KL}}(p_A\|q_A) + D_{\mathrm{KL}}(p_B\|q_B); \tag{3.12}$$

3.6 Convergence Theorems of Merged RBMs

As sampling from the full distribution of an RBM is intractable due to the the partition function, a Markov chain Monte Carlo based technique is used to generate samples from

the model distribution during inference both while training and while solving. Due to the bipartite graph nature of RBMs, Gibbs sampling is computationally efficient and can be done in 2 steps. Gibbs sampling converges to the model distribution in a geometric number of steps [29]. Exact estimates of the convergence rate is intractable, as it involves calculating the SLEM (Second Largest Eigenvalue Modulus) of the gibbs transition matrix. However, a theoretical bound can be analyzed using Dobrushin's Ergodic Coefficient as shown below. A similar derivation is also shown in [29].

$$|\mu P^n - \pi| \leq \frac{1}{2}|\mu - \pi|(1 - e^{-2\Delta})^n \tag{3.13}$$
$$\Delta = \sup_{x,y \in \{0,1\}^N} \{|E(x_v, x_h) - E(y_v, y_h))|\}$$

A proof of this is shown in Section 3.6. In this equation, μ is the starting distribution (or starting point) P is the transition probability matrix (represented by applying the gibbs transition operator on all units), π is the stationary distribution and n is the number of transition steps taken. The convergence parameter Δ represents the maximum energy difference between two states in the RBM, where we denote the to states x and y, with visible and hidden states (x_v, x_h) and (y_v, y_h) respectively.

From our derivation above, we can estimate the effects of merging on the upper bound of convergence rate. If we combine two RBMs A and B, with parameters labeled as such the highest energy states correspond to "correct" answers (E_A^{max}, E_B^{max}) and the lowest energy states correspond to "incorrect" answers (E_A^{min}, E_B^{min}). Denote the maximum and minimum energy of the combined RBM as E_{comb}^{max} and E_{comb}^{min}, and Δ_{comb} as the convergence coefficient for the combined RBM.

$$E_{comb}^{max} \leq E_A^{max} + E_B^{max}; \quad E_{comb}^{min} \geq E_A^{min} + E_B^{min}$$
$$\Delta_{comb} \leq \Delta_A + \Delta_B$$
$$|\mu P^n - \pi| \leq \frac{1}{2}|\mu - \pi|(1 - e^{-2\Delta_{comb}})^n \leq \frac{1}{2}|\mu - \pi|(1 - e^{-2(\Delta_A + \Delta_B)})^n$$

This implies a considerable decrease in convergence rate over the original RBMs, amounting to an exponential increase in the upper bound of convergence constant as more RBMs are combined together. We note that this is meant to be a theoretical upper bound on convergence, and thus the bound is never tight. Empirical results show that in many cases, this bound tends to be very loose, even while merging. For this reason, an empirical study is necessary to understand the effects of merging RBMs. We additionally note that when we train the RBM after merging (as is demonstrated in fig 3.2, these convergence theorems are no longer valid as the model has changed. We see that the performance gets remarkably better than this bound with the post-merging training step, so this proof provides a loose bound on performance to help us gauge worst case behavior.

Proof of Merged RBM Convergence Equations

This proof follows a similar structure to [29], but has further simplifications and additions due to the conditional independence structure of the RBM.

Gibbs Transitions over the RBM can be be factored into two stages, a transition probability distribution over visible units, and a transition probability distribution over hidden units. The respective transition matrices are multiplied to yield the full state transition matrix. We will call $\mathbf{P}$ the full transition matrix with elements p_{xy}, and $\mathbf{P}_v$, $\mathbf{P}_h$ the visible and hidden transitions with elements p^v_{xy} and p^h_{xy} respectively.

$$\mathbf{P} = \mathbf{P}_v \mathbf{P}_h \tag{3.14}$$

$$p^v_{xy} = \frac{e^{-E(y_v, x_h)}}{\sum_v e^{-E(v, x_h)}}, \quad p^h_{xy} = \frac{e^{-E(x_v, y_h)}}{\sum_h e^{-E(x_v, h)}} \tag{3.15}$$

The states x and y are parameterized as $x = (x_v, x_h)$ with visible (x_v) and hidden (x_h) portions. The individual transition probabilities p^v_{xy} represent the probability of transitioning from state x to state y, when only changing the visible states of x. Thus, the entry p^v_{xy} in $\mathbf{P}_v$ is nonzero iff $x_h = y_h$.

Starting with the convergence inequality from [34].

$$|\mu P^n - \pi| \leq \frac{1}{2}|\mu - \pi|(\delta(\mathbf{P})^n$$

$$\delta(\mathbf{P}) = 1 - \inf_{x,y \in \{0,1\}^N} \sum_{k \in \{0,1\}^N} \min(p_{xk}, p_{yk})$$

From here, we can take further bounds on the ergodic coefficient:

$$\delta(\mathbf{P}) \leq 1 - 2^N \inf_{x,y} p_{xy}$$

Defining $m_v(h) = \inf_{v \in \{0,1\}^{N_v}}(E(v,h))$ and $m_h(v) = \inf_{h \in \{0,1\}^{N_h}}(E(v,h))$, these represent the states with lowest transition probability from a given input state. The variables N_v and N_h represents the number of visible and hidden units respectively.

$$p^v_{xy} = \frac{e^{-(E(y_v, x_h) - m_v(x))}}{\sum_v e^{-(E(v, x_h) - m_v(x))}} \geq \frac{e^{-\delta_v}}{2^{N_v}}$$

$$\delta_v = \sup_{h \in \{0,1\}^{N_h}} (|E(v', h) - E(v, h)|; v, v' \in \{0,1\}^{N_v})$$

$$p^h_{xy} = \frac{e^{-(E(x_v, y_h) - m_h(x))}}{\sum_h e^{-(E(x_v, h) - m_h(x))}} \geq \frac{e^{-\delta_h}}{2^{N_h}}$$

$$\delta_h = \sup_{v \in \{0,1\}^{N_v}} (|E(v, h') - E(v, h)|; h, h' \in \{0,1\}^{N_h})$$

In words, δ_v is the maximal energy difference between two states that have the same hidden states, and δ_h is the maximal energy difference between two states that have the same visible states. Using this definition, we can define

$$\inf_{x,y} p_{xy} \geq \frac{e^{-(\delta_v+\delta_h)}}{2^N} \geq \frac{e^{-2\Delta}}{2^N}$$

$$\Delta = \sup_{x,y\in\{0,1\}^N} \{|E(x_v, x_h) - E(y_v, y_h))|\}$$

And finally

$$\delta(\mathbf{P}) \leq 1 - 2^N \inf_{x,y} p_{xy} \leq 1 - e^{-2\Delta}$$

$$|\mu P^n - \pi| \leq \frac{1}{2}|\mu - \pi|(\delta(\mathbf{P}))^n \leq \frac{1}{2}|\mu - \pi|(1 - e^{-2\Delta})^n$$

3.7 Low Density Parity Check Codes and Communications Algorithms

Introduction

Low-density parity-check (LDPC) codes are a class of linear block error-correcting codes, which are used in communication systems and data storage devices to correct errors and ensure the integrity of transmitted or stored data. LDPC codes work based on the idea of "parity checks", which means each bit in the data is checked according to a certain pattern (the pattern is "sparse", or has few ones, hence the name "low-density"). If an error is detected, the bit is corrected using an LDPC decoding algorithm [47]. Many modern communications standards, such as WiFi and 5G, use LDPC codes to correct for transmission errors. LDPC codes are heavily used as they are very close to the Shannon Limit for coding, while also having algorithms which can exploit the low density of the codes to achieve good performance.

The usage of LDPC codes is divided into two steps, first a code word is generated on the sending side, which uses a "generator matrix" G to create a bit-string to send across the channel. This message is then passed through a noisy channel, where an error may or may not occur. On the receiving side, the received message will be passed into a decoding algorithm which will then extract the original message while attempting to correct for any errors. Traditionally, the belief propagation algorithm has been used for the decoding process, but this process can be computationally expensive and does not have guarantees of optimal decoding. With these in mind, there has been interest in alternate methods for performing LDPC decoding.

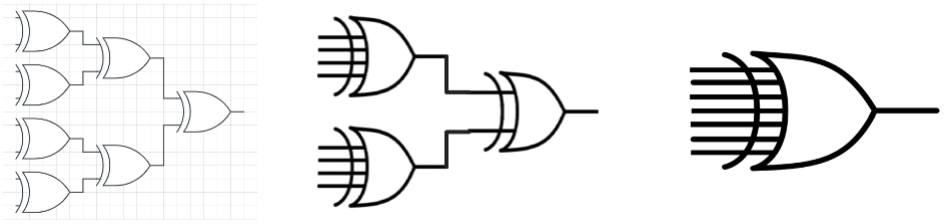

Figure 3.3: Depiction of multi-XOR (parity check) done 3 different ways. On the left is the XOR tree using a cascading tree of 2 input XOR gates. The middle is an XOR tree

Maximum Likelihood Decoding using RBMs

Maximum Likelihood Estimation for decoding of LDPC codes is a method of using probabilistic techniques to find the state which minimizes the error function of the input LDPC coded message. It is summarized by the system in Equation 3.16.

$$x^* = \arg\min_x [||r - x||^2 + \lambda \mod 2(Hx)] \tag{3.16}$$

In Equation 3.16, r is the received message, x is proposed method, H is the parity check matrix, and λ is a hyperparameter for optimization. This system can naturally be reformulated using the system of Inverse Logic and Restricted Boltzmann Machine that we have demonstrated in previous sections of this chapter. To map this to a probability problem, we set the visible nodes of the RBM to x, and then use hidden nodes to create correlations between these bits.

First, the H matrix is a sparse matrix (usually only having 4 to 16 values in each row which are non-zero), which means when we take the $\mod 2(Hx)$ product we only have to find the parity of the number of non-zero values in each rows. To take care of the $\mod 2$ operation of the input product Hx we use the formulation of $\mod 2$ as an exclusive OR (XOR) computation. To accomplish this we use an XOR tree and clamp the output of the tree to to ensure that parity is held at 0 for the full $\mod 2(Hx)$ product. This parity check mod 2 operation is shown using a multi-XOR operation in Figure 3.3 and is done by adding hidden units between the relevant nodes to create the XOR operation. Finally the received message r is added as a bias to the input nodes on x.

We have evaluated this decoding scheme against the standard belief propagation algorithm for 16, 32, and 64 length codes at various noise levels. In Figure 3.4 we show this performance difference. In that figure, BER represents Bit Error Rate (lower is better) and Eb/No is a measure of the amount of noise in the channel. Lower values of Eb/No represent higher noise. At smaller code lengths, the RBM based method is able to outperform belief propagation, but at larger code lengths, we find that the system does not keep pace in performance. We can attribute this to the vanilla gibbs sampling algorithm used here. We

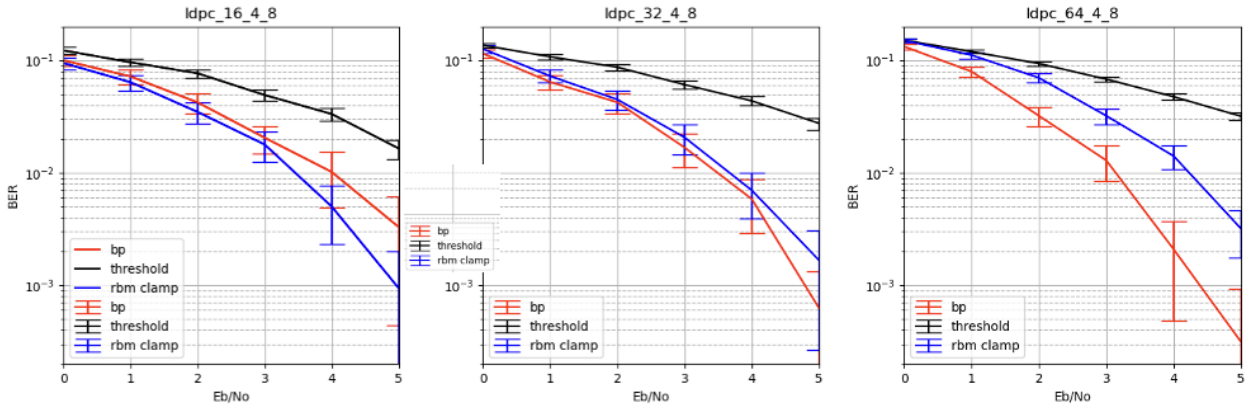

Figure 3.4: Performance evaluation of the RBM based LDPC decoder on code lengths of 3 sizes. Here "bp" represents the belief propagation algorithm, "rbm clamp" is the RBM formulation described here, and "threshold" is decoding by simply founding the input message. Left: RBM performance showing superior performance at all noise levels as compared to the belief propagation algorithm. Middle: For code lengths of 32, we see that the RBM performs on par with the belief propagation algorithm. Right: For longer code lengths, we see that the belief propagation has lower error rates than the RBM formulation.

believe that by using Parallel Tempering or something similar, performance on the LDPC decoding problem would increase.

Chapter 4

Direct Mapping Algorithms

4.1 Introduction

As many optimization problems have been formulated for the Ising Model [3], they must be transformed from the fully connected Ising model problem, to the bipartite graph structure in the RBM as demonstrated in Figures 4.1 A) and B). To do this, each logical node in the original graph is copied to create two physical nodes in the RBM, with one being in the visible layer and one being in the hidden layer. The connection between the two physical copies is referred to here as the "coupling coefficient" ($\mathcal{C}$) and forces both nodes to remain at the same value. Once the bipartite graph is formed, the RBM energy function is set to $E(v, h) = v^T W h$ with W being the bipartite graph's weight matrix. This encodes the lowest energy states in the original Ising Model problem as the highest probability state in the RBM probability distribution. From there we can use some of the sampling algorithms introduce in Chapter 2 to find the high probability states from the distribution of interest.

4.2 MaxCUT problems

An example of the sampling algorithm on a 150 Node MAX-CUT instance is shown in Figure 4.1 D). The raw samples produced stochastically fluctuate through the high probability states, with a higher probability of transitioning into and staying in the highest probability/lowest energy state. The samples can be analyzed to find the solution in one of two ways; either the samples can be collected and the mode of the sampled distribution is taken as an estimate for the highest probability state, or the unnormalized probability can be calculated for each sample and the highest probability state seen thus far can be taken as an estimate of the highest probability state for the underlying distribution. The first method is related to the "mixing time" of the Markov Chain and the second is related to the "hitting time" of the Markov Chain [29, 48]. The mixing time method requires less computation but more post-processing, while also allowing for analysis of the full distribution to find other potential high probability states as solutions to the problem. The hitting time method produces

only one output solution and converges to the solution faster than the mixing time method, but it requires an extra computation for every sample. In Figure 4.1E) we show that the hitting time method performs better for the same number of samples, with a cut distribution closer to the maximum. Further, Figure 4.1F) shows that the hitting time method has a constant factor of improvement in probability of reaching the ground state over the mixing time method. This result is expected from the theory of Markov chain samplers [49, 50].

4.3 Effect of Sampler Parameters on Algorithm Performance

The choice of parameters has a strong effect on algorithm performance and should be characterized to select optimal parameters for new problem instances. These parameters are found for each problem and set empirically based on trends seen in the data. The parameters for this sampler are the coupling ($\mathcal{C}$), the temperature (β), and the number of samples taken (N_s). We first optimize over the temperature parameter, as the other parameters have a strong dependence on the temperature chosen. The temperature parameter (β) refers to a scaling constant in the probability model, where $p(v,h) = \frac{1}{Z}e^{-\beta E(v,h)}$, and is equal to the inverse temperature seen in the physical Boltzmann distribution. As $\beta \to \infty$ the distribution becomes more sharply peaked with a larger probability difference between the ground state and first excited state. Conversely, as $\beta \to 0$, the distribution closer to uniform making it easier to sample from and converge to a solution. The results of the temperature analysis are shown in Figure 4.2 A), where $\beta \approx 0.25$ yields the best results. We note that we only tested in increments of 0.125, as the fixed precision available on our FPGA accelerator only allowed for increments of 2^{-2}. More precision is possible, but deemed unnecessary in comparison to the hardware costs.

The coupling coefficient, $\mathcal{C}$ is a soft constraint that forces the two copies of the logical node to be the same on the physical RBM implementation. For small values of this constraint, the two copies of the logical node are free from constraint, and an incorrect state will generally be chosen as the ground state. In the case of MAX-CUT, this generally leads to a state of 0 cut, where all nodes are the same value. In the case of the SK Problem, the ground state becomes a random state depending on the problem instance values. The coupling coefficient has a much stronger effect on the performance in the MAX-CUT problem than the SK problem, demonstrated by examining the performance for various values of $\mathcal{C}$. In Figure 4.2 B) we see the probability of finding the ground state solution strongly peaks around the optimal $\mathcal{C} \approx 12$ for $N_s = 70000, \beta = 0.25$ for the MAX-CUT problem. For the SK problem, we see less of a sharp peak, with a more gentle decline in performance (shown in Supplementary Figure 4.3 A)). In addition, the large values of $\mathcal{C}$ in the MAX-CUT problem cause a slower mixing time leading to worse performance on the MAX-CUT problem compared to the SK problem [32]. If we instead use the median cut outputted by the sampler as our metric, we see a slightly different picture where the performance is very poor below a certain threshold,

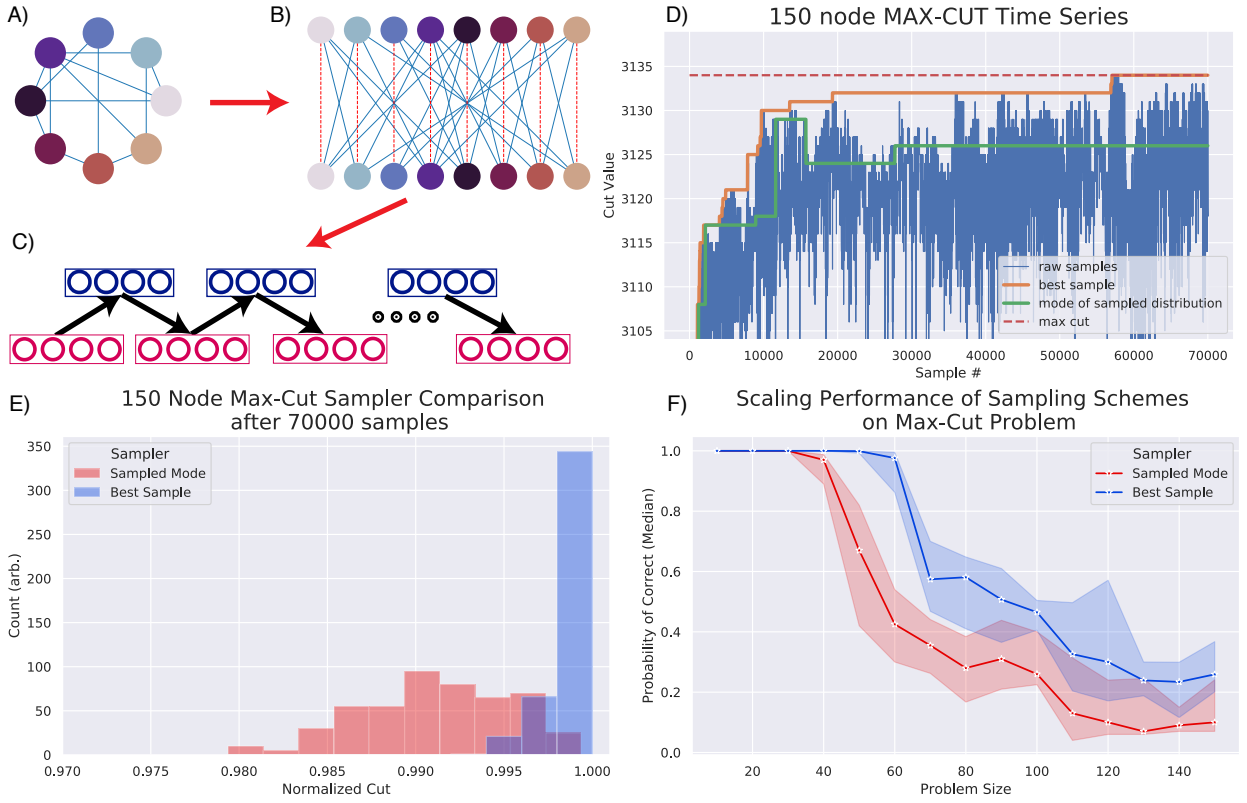

Figure 4.1: **Demonstration of RBM structure and sampling algorithm** **(A)** Structure of the input graph for an Ising Model type algorithm. The graph is fully connected, with no restrictions on the size or magnitude of the weight matrix. **(B)** The Ising Model is mapped to an RBM by making two copies of each graph node and edge and arranging them into a bipartite graph. One copy is in the "visible" layer neurons and one in the "hidden" layer neurons, with no intra-layer edges. Each physical copy of the neuron is connected by a "coupling" parameter ($\mathcal{C}$) which constrains the two copies to be the same value. **(C)** Due to the lack of intra-layer connections, the layers can be sampled in parallel. Each of the neurons in a layer is sampled in parallel and used to calculate the values of the opposite layer, creating a two-step sampling procedure. This sampling procedure proceeds until the output of the algorithm has reached the ground state, or until the algorithm output is of sufficient quality. **(D)** A demonstrative sampling run showing two different methods for interpreting the output samples from the RBM. **(E)** A histogram showing the output cuts after 1000 independent sampler iterations with $\mathcal{C} = 12, N_s = 70000, \beta = 0.25$ on a 150 Node Max-CUT problem, comparing the performance of the two sampling methods. **(F)** Analysis of the scaling of both of these sampler types. We see that both the sampled mode and best sample procedure perform well with the best sample method performing a constant factor above the sampled mode method.

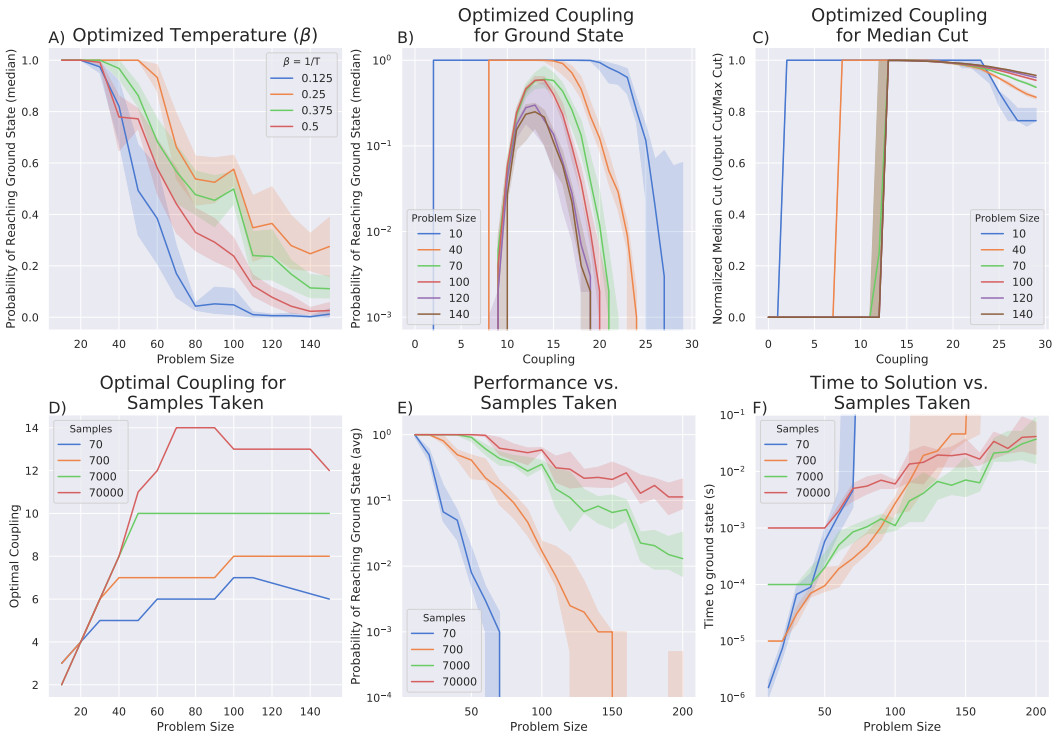

Figure 4.2: **Optimization of Algorithm Hyperparameters on the MAX-CUT problem**
(A) The parameter β scales the weights and biases by a constant factor to change the speed of convergence of the sampler. We settle on $\beta = 0.25$ as the optimal parameter, which is used for all experiments in this paper. **(B)** We show the performance on varying problem sizes for varying coupling parameters at a fixed $\beta = 0.25$ and $N_s = 70000$. This shows that for the MAX-CUT problem the $\mathcal{C}$ is generally optimal at $\mathcal{C} \approx 12$ for most problem sizes. **(C)** Although the probability of reaching the ground state is sensitive to the coupling parameter, the median cut outputted from the algorithm tends to be very close to the optimal value for a large range of coupling values. Below a certain value, the median value is very low, but it undergoes a sharp transition to its peak cut value, before slowly degrading. **(D)** The number of samples taken also increased the optimal coupling value for a given value. This is a consequence of the mixing time of the underlying distribution, where smaller coupling values mix faster but output statistics that are further from the ideal distribution for the problem. **(E)** With fewer samples taken, the probability of outputting the ground state decreases significantly. For a given number of samples, the probability of reaching the ground state decreases as $\mathcal{O}(e^{-bN})$ with different coefficients b in the exponent. **(F)** For a given number of samples taken, we can calculate the time to solution by evaluating Equation 4.1 along with the data from (E). The floor of this graph for each problem size is the optimal time to solution for the MAX-CUT problem using the hardware accelerated RBM. .

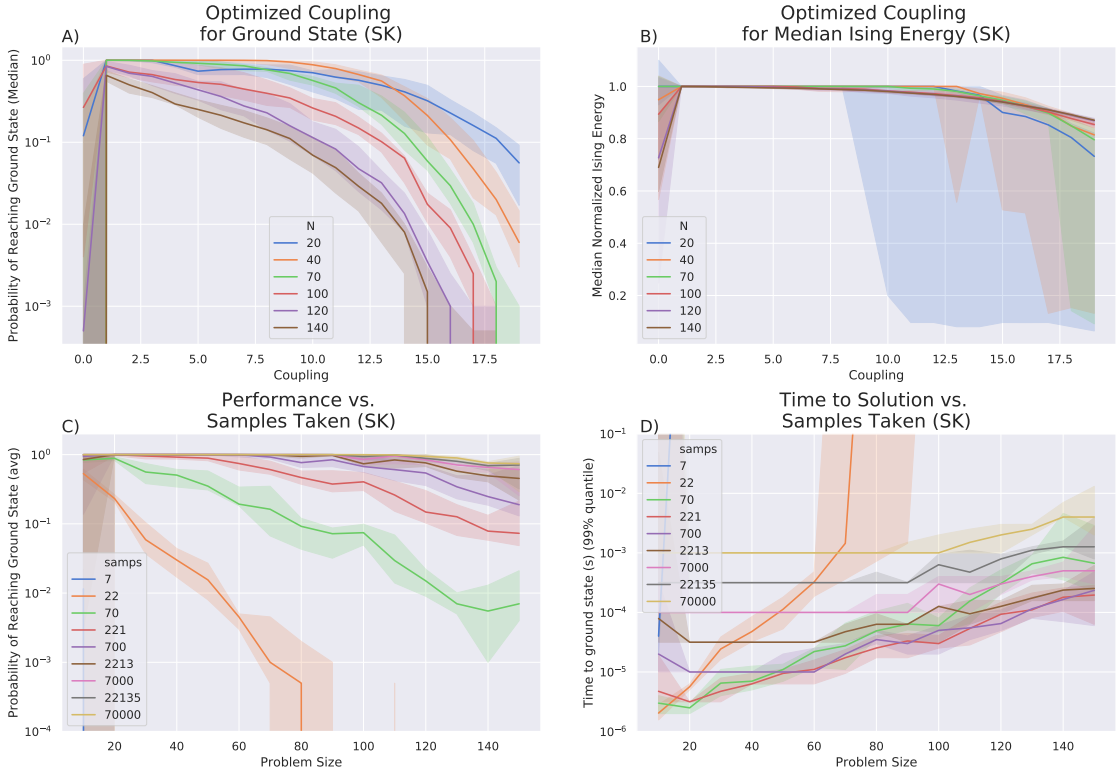

Figure 4.3: . **Hyperparameter analysis on the Sherrington-Kirkpatrick (SK) Problem**
(A) The coupling parameter for the SK problem is optimized at much lower values than the Max-Cut problem (closer to a coupling of 1 vs. a coupling of 12-13 on MAX-CUT). The SK problem has an inherent symmetry, as it has both +1 and -1 connections, causing the optimal coupling parameter to be lower for this type of problem. **(B)** The optimal coupling value for maximizing the median cut follows the optimal coupling to find the ground state. Similar to the MaxCut problem, the median cut remains high and is not as sensitive to the coupling parameter as the probability of reaching the ground state. **(C)** As above, in the MAX-CUT problem, the probability of reaching the ground state improves smoothly as more samples are taken. However, more samples takes more time, and we can optimize for the number of samples to take. The performance is significantly better on the SK problem, compared to the MAX-CUT problem when performed on the RBM. **(D)** Using the time to solution framework outlined in the Results section above we can use the sampling performance in part (C) and find the time to solution for a given sample number. Here, we see fewer than 2000 samples should be taken across all problem sizes, much lower than the MAX-CUT problem. The minimum across all sample numbers is taken to find the global time to solution for the RBM.

peaks at the optimal $\mathcal{C}$, and then slowly degrades with higher value, which is shown in Figure 4.2 C).

Additionally, the optimal $\mathcal{C}$ tends to change with the number of samples taken, demonstrated in Figure 4.2 D). This is because small values of the $\mathcal{C}$ allow for the system to approach the model distribution faster via a smaller mixing time and hitting time [13, 29] but this comes at the cost of having the highest probability state be a state of zero cut (See Figure 4.4 and accompanying discussion for further details). The result is that for small problem sizes, where enough samples are taken such that the sampled distribution is very close to the model distribution, we see a linear relation between the problem size and the optimal $\mathcal{C}$. This also corresponds to the region where the probability of reaching the ground state, $p_{gnd} \approx 1$ as shown by comparing Figure 4.2 D) and E). When problem sizes go past this point and the problem's sampled distribution is sufficiently far from the model distribution so it does not correctly identify the mode in all cases, the optimal coupling parameter is mostly flat.

As the number of samples taken increases, the sampled distribution approaches the model distribution, and the ground state can be correctly identified. This generally causes a smooth and monotonic increase in probability of reaching the ground state as shown in Figure 4.2 E). The stochastic hill climbing of the Gibbs sampling algorithm causes the mode to be correctly identified and found well before the full distribution is mapped. Additionally, as each trial is independent, many trials can be performed and a higher success probability can be reached by probability amplification [51]. We combine these properties with the Time to Solution framework used in other works [52, 6, 53, 54], and adapted for the RBM in equation 4.1, as the standard for evaluating probabilistic accelerators. This corresponds to the 99% quantile for reaching the ground state of a given problem.

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

In this equation N_s is the number of samples taken, f_{clk} the clock frequency (70Mhz for our FPGA implementation), and p_{gnd} the probability of reaching the ground state for that particular problem. We use this equation along with the data from Figure 4.2 E) to create Figure 4.2 F) which shows the time to solution for various N_s. We note that if multiple FPGAs are available we can parallelize this scheme and further reduce the T_{soln}. Here we use a sequential Time to Solution framework for fair algorithmic comparison to other accelerators. From the graph in Figure 4.2 E) we see that the optimal number of samples taken (in this framework) is generally lower than the number of samples needed to reach very high accuracy on an individual problem. We can take the lower bound on the data in Figure 4.2 F) and create a Pareto-optimal boundary for performance on these problems. The same parameter optimizations presented in Figure 4.2 for MAX-CUT are performed for the SK problem and shown in Figure 4.3.

4.4 Scaling and Connectivity

One of the biggest challenges in implementing Ising Machines is creating all-to-all connectivity between nodes. When mapping arbitrary graphs onto the Ising Machine, this is a necessary requirement to build a usable machine. The CIM supports this kind of all-to-all connectivity, while the DWave 2000Q, with its limited connectivity, uses $\approx N^2/\kappa$ where κ is the level of connectivity in the physical graph [52], leading to a large overhead in computation. A consequence of this is that the DWave annealer has scaling performance of $\mathcal{O}(e^{-N^2}))$ in probability of reaching the ground state, while both the CIM and the RBM based methods exhibit $\mathcal{O}(e^{-N})$ [52, 55]. The RBM based methodology, although mapping N logical nodes to $2N$ physical nodes, does not suffer from the same scaling problems as the DWave Machine. We believe this is because the RBM increases the number of physical nodes by a constant factor, rather than a factor that depends on the size of the input graph.

We expect similar asymptotic performance for both the SK problem and MAX-CUT as they both are NP-Hard problems, based on the Ising Model glassy spin system configuration, using the same embedding and underlying sampling algorithm. This is confirmed based on our experiments with both problems having an underlying $\mathcal{O}(e^{-N})$ for probability of reaching the ground state for a fixed number of samples (see Figures 4.2 E), and Figure 4.3 C)). Based on the same scaling behavior for the ground state probability metric, we see a scaling of $\mathcal{O}(e^{\sqrt{N}})$ for both the MAX-CUT and SK problems and fit curves to both problems (see Figures 4.2 F) and 4.3 D))). We note that as the SK problem requires such few samples for computation, the optimal sample point does not transition very much between different sample values. This causes the behavior to appear linear in the exponential at first glance, but we expect it to follow the $\mathcal{O}(e^{\sqrt{N}})$ behavior for larger problem sizes.

Although we empirically see a probability scaling of $\mathcal{O}(e^{-N})$ and a time to solution scaling of $\mathcal{O}(e^{\sqrt{N}})$ in the RBM annealing algorithm, we acknowledge that without a theoretical result, this scaling is not proven. More work is necessary to understand the sampling algorithm in greater detail. It should be noted, however, that the scaling principles seen in the CIM and DWave are also empirical [52, 56], thus only experimental data can be reasonably compared and care should be taken when extrapolations are taken of the data.

4.5 Effect of $\mathcal{C}$ and graph embedding on algorithm performance

The purpose of the coupling constraint ($\mathcal{C}$) is to enforce the two physical copies of the logical node to be the same value. When the constraint is violated, the two physical copies have different values and the state energy is no longer proportional to the Ising Energy of the problem Hamiltonian being solved. This would imply large values of $\mathcal{C}$ would improve performance, but that is not what is observed. When the value of $\mathcal{C}$ is too large, the Markov Chain does not mix quickly and settles in local minima [13, 32, 29].

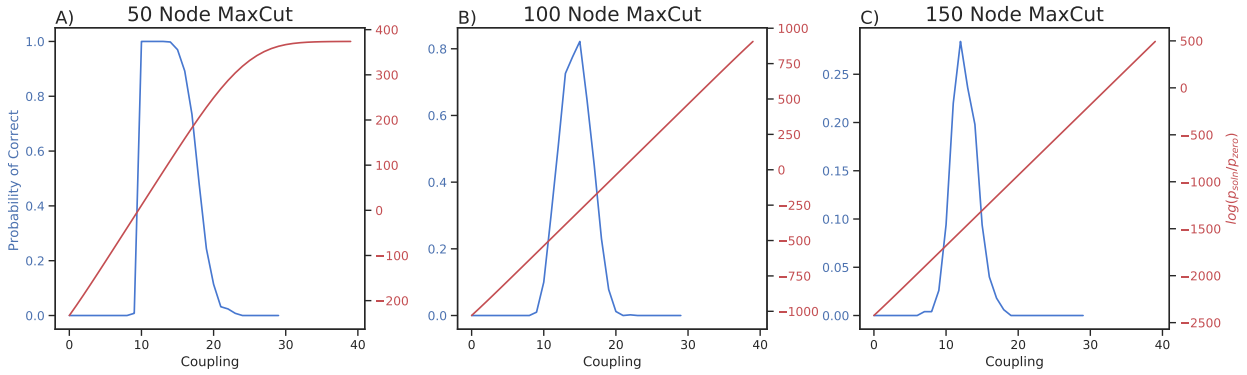

Figure 4.4: . **Effect of Coupling Coefficient on Sampled Performance vs. Limiting Performance**
This shows how the relative probability of the zero state (right axis) compares to the probability of reaching the ground state after 10000 iterations (left axis) for $N_s = 70000, \beta = 0.25$ on various problem sizes. **(A)** For $N = 50$, we can tell that the N_s is large enough such that the sampler has fully mixed, as the sampler performance peaks when $p_{soln} > p_{zero}$ and the probability of correct ≈ 1. **(B)** For $N = 100$, we see that the sampler is further from convergence as the peak performance no longer approaches 1, and the optimal coupling parameter is for a state where $p_{soln} < p_{zero}$. **(C)** As the problem size increases to $N = 150$ we see that the sampler is even further from convergence as $p_{soln} \ll p_{zero}$ at the optimal coupling value.

As shown by comparing Figures 4.2 and 4.3, we can see that the there is a significant difference in performance between the MAX-CUT and SK problems, even when comparing the performance difference to the other annealers. This is partially caused by the difference in optimal $\mathcal{C}$ required for the problems. While the SK problem uses $\mathcal{C} \approx 1$ for all problems, the MAX-CUT problem has an optimal value of $\mathcal{C} \approx 12$ for the same $N_s = 70000, \beta = 0.25$. In addition, the graph embedding we use leads the MAX-CUT problem to have a high probability for a state with a cut of 0, a state which is suppressed for high values of the coupling parameter. We would expect that remapping the MAX-CUT problem to the RBM via a different method that requires smaller $\mathcal{C}$ could result in a increase in performance due to lower mixing times.

For the MAX-CUT problem there is an intuitive explanation for the role of the coupling parameter. When $\mathcal{C} = 0$, the two physical copies of a node in the original graph are completely disconnected resulting in the two states not having any direct effect on each other. The maximum cut in this degenerate graph is the one which separates the hidden from the visible nodes and passes through all of the edges and where $v = \{0\}^N, h = \{1\}^N$ or $v = \{1\}^N, h = \{0\}^N$. We refer to this state as the "zero cut state" as it corresponds to a state that has zero cut in the original Ising Model graph. As $\mathcal{C}$ increases, the relative probability of this state decreases compared to the actual MAX-CUT state for the original Ising Model graph. However, large $\mathcal{C}$ causes slower mixing rates, which means that the performance of

the sampler tends to peak significantly before the solution state has a higher probability than the zero cut state. This also serves as a good proxy for how close the sampler is to the model distribution, as the probability of reaching the ground state should peak when the MAX-CUT state has higher probability than the zero cut state if the sampled distribution is close to the model distribution. In Figure 4.4 we show this phenomenon, where for $N_s = 70000$ we can see the regions of operation. For $N = 50$ we can see the sampled distribution is very close to the model distribution, as the probability of reaching the ground state peaks when the ground state probability is larger than the zero cut probability and we are able to reach the ground state in almost all instances. For the $N = 100$ and $N = 150$ instances we can see the sampler is further away from the the model distribution as the probability of reaching the ground state decreases and the sampler performance peaks for smaller coupling values where $p_{soln} \ll p_{zero}$.

Chapter 5

FPGA based RBM accelerators

5.1 Introduction

The massive parallelism present in the RBM algorithm makes it especially efficient on the FPGA. The RBM algorithm also doesn't contain any branches or explicit memory accesses while sampling, removing expensive branch misprediction cycles and DRAM fetch cycles. Furthermore, unlike other deep neural network accelerators, this algorithm is not memory bandwidth limited for mucho f its operation [57] as can be seen by the FPGA utilization, Table 5.1, further increasing the algorithmic performance on hardware. The bipartite nature means that many neurons can be sampled in parallel on the FPGA, allowing us to perform each neuron activation probability in parallel. There has been much work on accelerating RBM training through FPGA implementations [58, 59, 60], but by focusing on inference only, we reduce the necessary hardware requirements to the essential components, fully unlocking the inherent parallelism in the network architecture.

5.2 RBM Inference Accelerator for Inverse Logic

In the context of accelerating the solution for the Inverse Logic problems presented in Chapter 3, we need to design an accelerator that is designed to perform the block gibbs sampling algorithm effectively. This relies on running fast and parallel MCMC to take advantage of the FPGAs resources.

The computation speedup of the FPGA vs. the CPU/GPU we have accomplished is due to a series of hardware friendly modifications and model choices that we have made. The first is to develop a series of model quantization techniques specific to the RBM architecture and Contrastive Divergence algorithm, using inspiration from traditional quantization in deep neural networks [61, 62, 63]. The next technique is to utilize the binary activations to transform the matrix multiplication update in the RBM to a mask and accumulate operation. This removes the costliest portion of the update (the multiplication) greatly improving utilization on the FPGA and clock speed [64]. To further remove all multiplications in the

neuron update, we replace the sigmoid operation with a Look up Table (LUT) based approximation. This allows fast evaluation of the activation function without expensive hardware resources. The last major hardware based improvement that was possible is usage of the fast LFSR based pseudo random number generator. By using a 32 bit LFSR, we are able to achieve similar performance to the baseline PyTorch random number generation while using a significantly simpler algorithm.

Model Quantization

In addition to taking advantage of the inherent parallelism, to fit larger models on the FPGA we have performed model quantization to be able to lower precision during the inference. There has been much work on model quantization for deep neural networks, however most of them focus on Convolutional Neural Networks and Multi-Layer Perceptrons [61, 62, 63] specifically using algorithms which rely on the backpropagation algorithm for training. As the RBM uses Contrastive Divergence through MCMC sampling many of these methods are not directly useful, however we can use these methods for inspiration to quantize our models. Additionally, our goal of targeting an FPGA architecture requires the final weight values to fall into the regularly spaced quantization bins of the fixed point representation that we choose. For these reasons, we adopt a 2 step training procedure for quantization. The first is adding a constraint to the maximum value of the weight and retraining with this constraint. This makes sure that the weight magnitude cannot overflow the fixed point representation that we are using. As demonstrated in Figure 5.1 A), this does not change the overall accuracy of the model, as the retraining causes more weights of smaller magnitude to compensate for a single weight of large magnitude. The second retraining step adds an extra quantization loss term to the training step (see Eqn. 5.1). In Figure 5.1 B) we show that we can quantize from 32 bit floating point to 6 bit fixed point without large loss in error. Figure 5.1 D) shows the final weights distribution after these two quantization training steps, demonstrating how the retraining preferentially pushes weights towards their post-quantization values. The exact training steps are detailed below. In Figure 5.1 H) we see how this quantization effects the performance of a factorization task. This increase in error is accompanied by massive increases in speed and power efficiency.

Retraining for Quantization

To train for quantization, the loss function that is optimized for ($L(W, d)$) is modified so that in addition to having the regular Contrastive Divergence loss between the weights W and the data d denoted by $CD(W, d)$, we have a loss term that pushes weights to be closer to their quantized value. The hyperparameter λ is slowly increased during training to force the weights progressively closer to their quantized value. This method allows the contrastive divergence term to fix the errors created by quantizing the weights slowly while training. Although taking the exact gradient of this loss term is not possible (as $Q(W)$ is not a smooth function of W, see Eqn. 5.2), we find that by assuming the quantization gradient

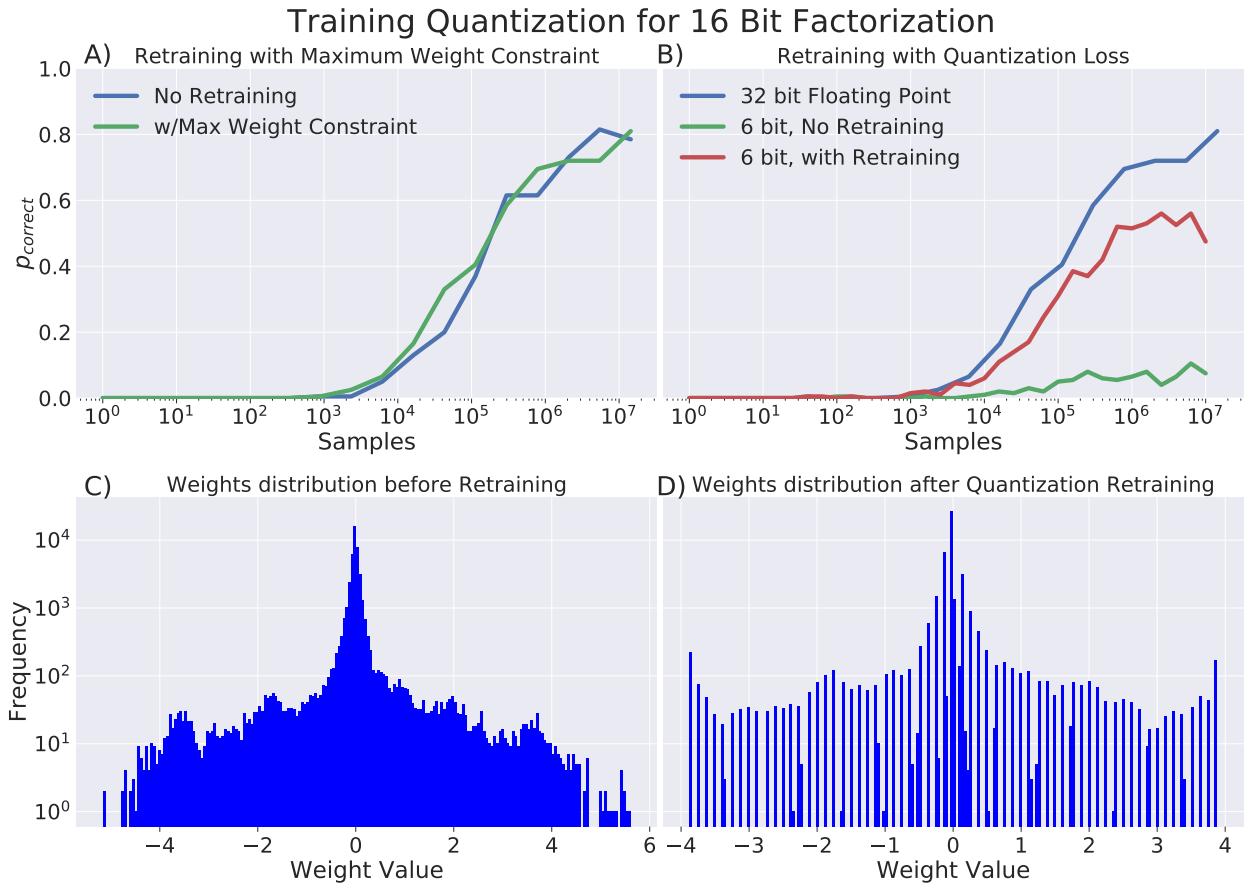

Figure 5.1: **(A)** Showing the effect of retraining with a maximum weight constraint. Here we see no performance degradation due to retraining the module by adding this extra constraint. **(B)** Retraining the network with added L_1 quantization loss. By retraining for 6 bit quantization, we see a large increase in performance compared to naive quantization. **(C)** A histogram of the weights before quantization retraining proceeds **(D)** A histogram of the RBM weights before and after retraining for quantization. We see the network is strongly clustered around the 6 bit values.

$\frac{\partial Q(W)}{\partial W} \approx 0$, we obtain a sufficiently good performance.

$$L(W,d) = \epsilon CD(W,d) - \lambda||W - Q(W)||_1 \tag{5.1}$$

$$\frac{\partial L(W,d)}{\partial W} = \epsilon \frac{\partial CD(W,d)}{\partial W} - \lambda sign(W - Q(W))(1 - \frac{\partial Q(W)}{\partial W}) \tag{5.2}$$

Matrix Multiplication with Mask

The RBMs binary activations and fixed point weights allow for a very efficient matrix multiplication module. The binary activations convert multiplications into a binary mask, or a 2-to-1 mux using the activation value as the switch. This results in multiplications reducing to atomic operations on the FPGA, greatly reducing their area, power and latency. The usage of fixed point weights, instead of 32 bit floating point, is estimated to decrease the area of the accumulator circuits by 8x [64]. The usage of fixed point calculations also creates an incredibly large decrease in power consumption. A 32 bit floating point multiplication costs 187x the cost of an 8 bit addition, suggesting that by removing the multiplication steps, we have drastically reduced the power an energy consumption of the system as a whole. [57]. The smaller area of each component allows us to use a larger adder tree with less delay for accumulation as compared to 32 bit floating point operations, resulting in more computation which can be completed in one cycle.

Sigmoid Approximation

Exact calculation of the sigmoidal activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ is computationally expensive. To accomplish direct calculation, at least 3 extra hardware instructions are needed, exponentiation, addition, and division, which all incur a large hardware cost both in terms of latency and area. Instead, binary sigmoid values are precomputed and enumerated in a look up table (LUT) for use in the FPGA. This implementation allows for fast evaluation of the activation function without expensive hardware resources. After matrix multiplication and bias addition, the computed value is passed through the LUT based activation function to approximate the sigmoid.The Look Up Table values are hard-coded at synthesis time and thus do not use any LUT resources. Finally, the sigmoids are synthesized as 9 7-input Multiplexers (F7 MUX) within 6 Combinational Logic Blocks (CLBs) amounting to 1% of the total available F7 MUXs, and $< 1\%$ of the total available CLBs. This is a common technique used in many FPGA and ASIC based neural network accelerators, which is further adapted for our particular FPGA implementation. [65, 66, 67, 68] .

Pseudo-Random Number Generation

To generate high quality samples, uncorrelated random numbers need to be produced every cycle. To accomplish this we use a 32 bit length Linear Feedback Shift Register (LFSR) pseudo random number generator. The 32-bit LFSR size creates $2^{32} - 1$ random bits, or $2^{29} \approx 5 \times 10^8$ random 8 bit numbers. To determine the best LFSR size, we characterized performance for the sampling algorithm by varying the LFSR length and determining performance on the factorization problem. These results are presented in Figure 5.2. As, the total cost of these LFSR based random number generators amounts to just 5% of the design flip flop usage, and 2% of the lookup table usage we chose a longer than necessary LFSR chain to minimize accuracy and performance problems from this element of the design. Each

RBM Size (Vis x Hid)	LUT Usage (Absolute)	LUT Usage (%)	FF Usage (Absolute)	FF Usage (%)	Power (W)
8x32	14811	1.25	15437	0.65	5.2
16x64	33564	2.84	23717	1.00	5.2
32x128	117931	9.98	52582	2.22	5.5
64x256	418694	35.42	159428	6.74	5.6
64x512	736800	62.32	270182	11.43	5.9
80x600	990417	83.77	431544	18.25	6.2

Table 5.1: **FPGA Utilization: Utilization numbers for FPGA and various RBM sizes** All usage numbers reported are for 8 bit weights and biases . The usage shows that the FPGA is not memory limited for the problem sizes we are interested in, but compute limited, as the LUT usage goes up much faster than the FF usage as the problem size grows. All weights and biases fit in on chip SRAM, allowing for fast access and data reuse.

	Total Power (W)	Total Power (%)		Dynamic Power (W)	Dynamic Power (%)
Hard IP	0.134	2	Clocks	0.606	17
Dynamic	3.462	56	Signals	0.386	11
Static	2.580	42	Logic	0.318	9
			BRAM	0.164	5
			I/O	0.011	1
			GTY	1.976	57

Table 5.2: **Breakdown of Power usage for** 80×600 **RBM on FPGA** Power usage numbers reported for post implementation design by Xilinx Vivado Design Suite 2019. Numbers show that dynamic power is largest power draw, with the communication links (Gigabit Transceivers - GTY) being the highest power consumption within that component. This shows that the logic design is fundamentally low power and can be implemented very efficiently if done on a dedicated board.

neuron has its own LFSR and is seeded with a different value to minimize the possibility of correlation. Other PRNG algorithms can produce higher quality random numbers [69, 70], but they require greater hardware resources and are generally more complex. Based on our experiments, we found the LFSR random number generator to be the simplest, as well as most performative pseudo-random number generator available on the FPGA.

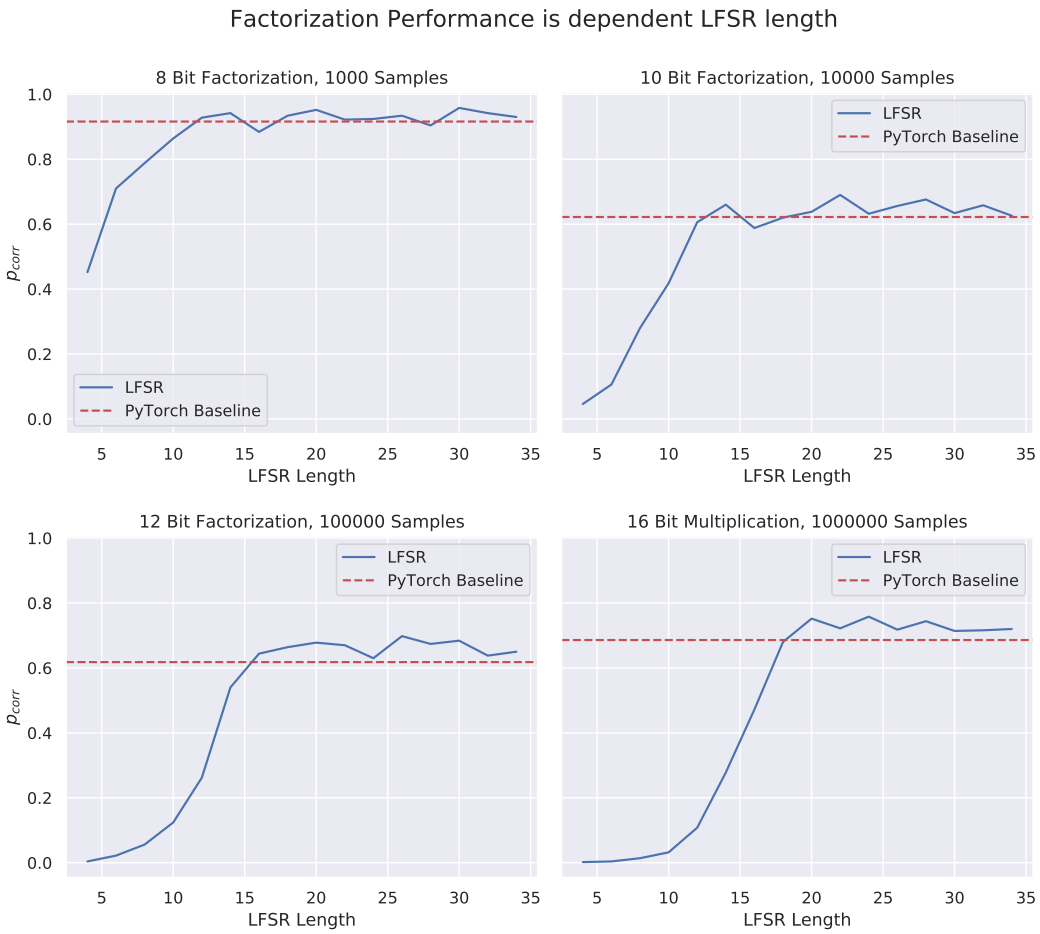

Figure 5.2: (Supplementary) **Effect of LFSR length on overall performance** For an 8 bit (top left panel) 10 bit (top right panel), 12 bit (bottom left panel) and 16 bit (bottom right panel) factorization problem, we analyze the performance by varying the length of the Linear Feedback Shift Register (LFSR). For the largest 16 bit factorization problems, we see that a 20-bit length LFSR is the minimum required to reach the same level as the PyTorch baseline level of accuracy. As hardware costs for longer LFSR are low, we chose to have a 32 bit LFSR for experiments as this ensured sufficient buffer for long sampling runs, and optimal performance for large factorization problems.

5.3 Performance Analysis

We have demonstrated scaling of the factorization algorithm up to 16 bit numbers. Markov Chain Monte Carlo based sampling methods for optimization problems fall into the class of "Stochastic Local Search" and are expected to have exponential scaling with problem size [71]. This exponential scaling dependence is shown in Figures 5.4 C) and D). Although this is the case, we see a 10^4 constant factor speed increase when the algorithm is implemented on the FPGA in Figure 5.4 B) in 16 bit factorization and Figure 5.4 E) across all problem instances. This massive speed increase across the whole spectrum of bit sizes has real world consequences, as it implies that other algorithms mapped onto this general framework can become very efficient in finding ground state solutions which would otherwise be difficult to obtain.

The FPGA implementation of our sampling algorithm has shown a 10^4 speed increase compared to a dual CPU system, and a 10^3 speed increase compared to a GPU. This comes with a 30x power decrease compared to the CPU (2×95W CPUs vs 6W FPGA) and 7x power decrease compared to the GPU (44W GPU vs 6W FPGA). This is compared to a dual CPU machine running the highly optimized industry standard PyTorch machine learning framework, and a hand optimized GPU algorithm using the CUDA and cuBLAS libraries. We note that the performance improvement of the GPU algorithm compared to the CPU is minimal due to the thread synchronization, limited cache sizes, and relatively small RBM size presented here. The GPU additionally cannot take advantage of the binary activations for efficient multiplications, or approximate sigmoid calculation. We hypothesize that these problems for the GPU would extend to other accelerators that were not purpose-built for the type of computation that our RBM based accelerator uses. Our specific GPU and CPU acceleration algorithms are further cited in the Methods section. [72, 73] Although our implementation takes up much of the resources of the FPGA, there are many possible areas where our design could be modified to scale for performance. For example, accelerator-level parallelization and better scaling become possible through the use of multi-FPGA designs and communication [74, 60, 58], time division multiplexing [75] more efficient pipeline stages [76] and utilization of different hardware paradigms such as stochastic computing primitives [77, 78]. With focused effort on the improvement of the hardware architecture [77, 79, 74] the speed and performance improvement is expected to get much larger. Our goal was not to create the most optimized hardware design, but to demonstrate that parallel hardware running our very hardware friendly algorithm had the potential for drastic improvement.

Although we demonstrate sampling speed increase for inference, using the same FPGA accelerated sampling algorithm can also work for decreasing training time. The major bottleneck in the training step is creating a series of uncorrelated samples, which takes a large number of samples for a highly correlated sampler. Using FPGA acceleration of the sampling algorithm could give a lower variance estimate of model probabilities in a much faster and energy efficient manner, than those provided by a CPU or GPU.

Importantly, these results show the factorization of the largest number (16 bit) that has been experimentally demonstrated on a hardware accelerator using an Ising Model based

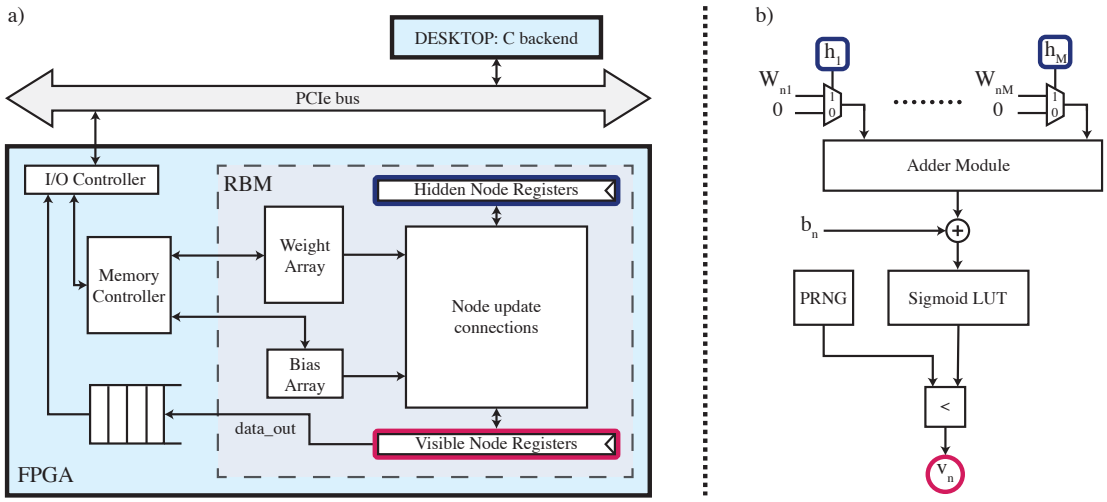

Figure 5.3: **FPGA Architecture (A)** Memory and compute hierarchy. The RBM consists of memory to hold the weight, bias, and clamp values, registers to hold the node values, and circuitry to perform the node updates, which take up the bulk of the resources. The output is buffered to the IO controller that communicates results to the PCIe bus. A C backend reads in the data stream from PCIe, and can program the weights and biases from the memory controller. **(B)** Example of a node update connection. Given M hidden nodes, the figure depicts the circuitry to update the n^{th} visible node. The hidden nodes binary mask the n^{th} row of the weight matrix. The results are accumulated in the adder module and added to the n^{th} visible bias. It is then passed through a sigmoid LUT and compared to the output of a PRNG to update the value of the visible node.

topology [7, 42, 80]. This is a direct result of our algorithm which lends itself well to scaling by breaking down a problem and building up via sub-parts via re-training. As Ising Model based systems are considered exciting candidates for next generation computing, this represents an important improvement on the state of the art. These results are tabulated in Table 5.3, where we show that the RBM and FPGA based algorithm shown here is able to perform better than similar Ising Model based accelerator and algorithms. All three of the accelerators perform algorithms which are optimized and designed for the hardware that they are running on, demonstrating how far each accelerator is able to solve their given algorithm. Not only is the RBM able to solve larger factorization tasks, it does it with lower power than the DWave computer, and less connectivity than the P-bit algorithm. Although the P-bit based algorithm is more efficient from a per-spin factorization basis, it does so with a non-scalable all to all connectivity pattern. Additionally the P-bit algorithm uses discrete components, operating at a much slower intrinsic frequency. Time to solution is not compared as the other accelerators do not report this figure of merit.

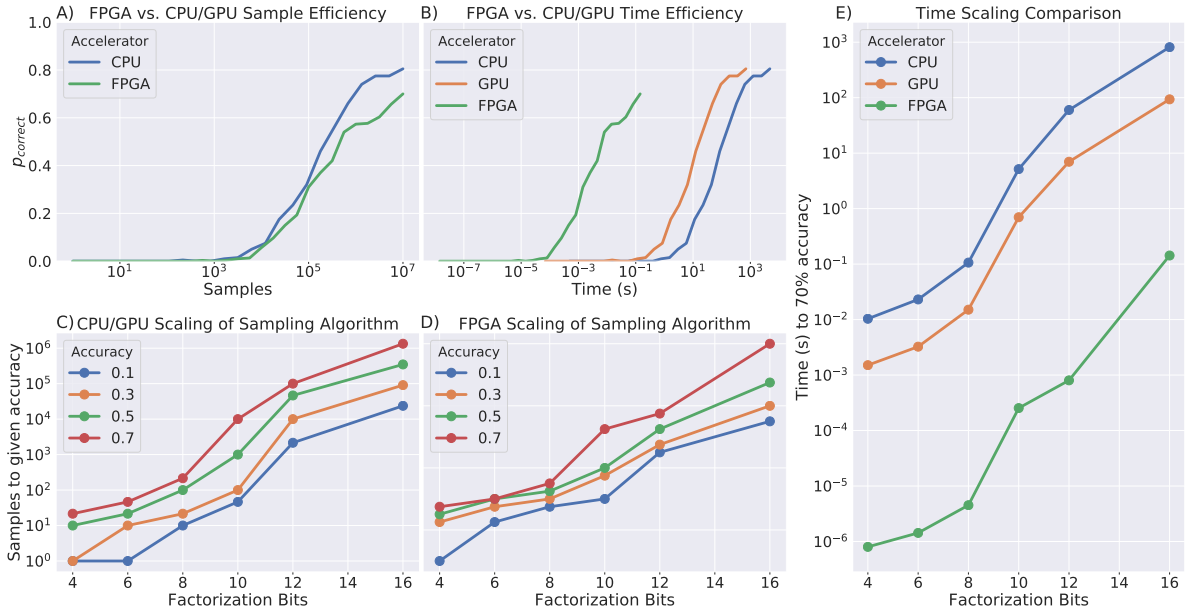

Figure 5.4: **Performance of the FPGA implementation vs the CPU and GPU implementations on factorization**
The sampling algorithm scales approximately exponentially with the bit size (and approximately linearly with the phase space). We see a 10^4 speed improvement across all model sizes compared to the CPU algorithm and 10^3 speed improvement compared to the GPU algorithm. **(A)** The sample efficiency of the FPGA implementation is similar to the CPU implementation, even after quantizing to 8 bit weights and biases and using the various approximation schemes detailed. **(B)** When the time taken to reach a solution is scaled for the FPGA vs. the CPU and GPU, the FPGA outperforms both by orders of magnitude., **(C)** The scaling of the algorithm when measured at various accuracy levels on the CPU. The RBM for each bit number is run until it hits the given accuracy on a set of random factorization problems. **(D)** Scaling of the sampling algorithm when run on the FPGA. The difference in sample number from part (C) is due to approximations necessary to efficiently port the model onto the FPGA. **(E)** Time scaling of the factorization problem measured at the 70% accuracy level. We see that the FPGA performs 4 orders of magnitude faster compared to the CPU and 3 orders of magnitude compared to the GPU across all bit counts for the outlined sampling algorithm.

5.4 RBM Accelerator for MaxCut and General Problems

The above architecture serves as a first step to demonstrate performance of the inference based algorithm. We further use the above described accelerator to map general Ising Model problems via the direct mapping approach described in Chapter 4. The accelerator supports 9 bit precision fixed-point weights and biases, which allows for solving Ising Hamiltonians for various problems other than the one included in here. We note that while the problems benchmarked in this work only use weights in $\{-1, 0, 1\}$, we wanted to maintain the generality of our accelerator to solve a variety of other problems within the RBM framework, such as machine learning inference [13], and other instances of NP-Hard Combinatorial Optimization problems [3, 48]. We note that if we were to restrict the weights our RBM implementation was able to support further we would expect the accelerator as designed to support larger problem instances.

As outlined previously, the two methods for finding the best solution to the optimization problem involves either using the mixing time method or the hitting time method. As the hitting time method has a theoretical advantage for solution quality, we prefer using this. Calculating the probability of each sample on the CPU is computationally expensive, so we implement a "hitting time engine" on the FPGA which calculates an approximate, unnormalized log-probability for each sample and keeps track of the state with the highest probability. By using partial computations already available when computing node activations, the hitting time engine has negligible hardware costs and is able to decrease the amount of FPGA to CPU communication to a minimum, freeing the CPU for other computational tasks.

Hitting Time Engine

The hitting time engine calculates the approximate log probability for each sample that the stochastic sampling algorithm outputs and keeps track of the highest probability state seen so far. This method works to offload the computation of calculating probabilities or aggregating results from the CPU to the FPGA. The FPGA can operate in raw sample output or hitting time engine output, where it either pushes the raw samples to the CPU or the highest probability state seen. The hitting time engine uses partial computations from each cycle to reduce computational overhead for the FPGA. To do this, we first look at the log probability for a given visible node state.

$$
\begin{aligned}
p(v,h) &= \frac{1}{Z}e^{-E(v,h)} \\
&= \frac{1}{Z}e^{\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}v_ih_j + \sum_{j=1}^{m} a_jh_j + \sum_{i=1}^{n} b_iv_i} \\
p(v) &= \sum_{h} p(v,h) \\
&= \frac{1}{Z}\prod_{i=1}^{n} e^{b_iv_i}\prod_{j=1}^{m}(1+e^{\sum_{i=1}^{n} a_j + w_{ij}v_i}) \\
log(p(v)) &= \sum_{i=1}^{n} b_iv_i + \sum_{j=1}^{m} log(1+e^{\sum_{i=1}^{n} a_j + w_{ij}v_i}) - log(Z)
\end{aligned}
$$

The $log(Z)$ term is a normalizing constant and can be ignored if we are only comparing probabilities between samples. Additionally, the $log(1+e^x)$ term is simplified as follows.

$$
log(1+e^x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}
$$

This simplification is valid for $x \gg 0$ and $x \ll 0$, but introduces errors when $x \approx 0$. These errors are not significant in the probability calculation, as the largest contributions to the probability mass are for $x \gg 0$. The $\sum_{i=1}^{n} a_j + w_{ij}v_i$ is calculated each cycle to update the hidden units and is thus recycled for calculation of the overall log probability of the given state. This means the only calculation the hitting time engine has left to do is accumulation of the visible biases and accumulation of the thresholded sums that have been pre-calculated by the hidden neurons. The hardware overhead for the hitting time engine is very small due to the efficient usage of these pre-calculated sums. The hardware usage translates to $< 1\%$ of additional FPGA LUT utilization and $< 1\%$ of additional FPGA flip flop usage (see Table 5.1).

The hitting time engine is split into two modules, each calculates the log probability for every other cycle. Each module is composed of an accumulator which takes the partial sums from the hidden nodes and accumulates half of them each cycle along with the visible biases. By splitting the calculation over two clock cycles we are able to meet the 70Mhz timing requirements set by the rest of the design. The hardware cost of these accumulators is approximately the same as an additional visible node.

Performance Analysis

In Figures 5.5 and 5.6 we show results of benchmarking performance on the MAX-CUT and SK problems. In Figure 5.5 A) we show how the probability of reaching the ground state scales for a fixed annealing schedule. We fix $N_s = 70000$ for the RBM, corresponding to

1000µs of time at 70Mhz and see that the performance outperforms the other annealers at all problem sizes when given less time to solution. Using the Time to Solution framework shown in Equation 4.1 we convert the optimal sampling solution from Figure 4.2 F) to compare the time to solution against the DWave 2000Q in Figure 5.5 B) and the Coherent Ising Machine instances in Figure 5.5 C). We see a particularly stark difference in scaling performance when comparing to the DWave 2000Q, where the performance on problem instances drops quickly to 0 after 50 Nodes in the MAX-CUT problem, while the RBM is still able to solve larger instances. When looking at time to solution, this accounts for a 10^6x difference in performance at 50 nodes. When comparing to the the Coherent Ising Machine, we see very similar scaling performance for time to solution. While the Stanford CIM and the NTT CIM perform very similarly, the RBM performs at a constant $\approx$ 150x advantage over all problem sizes. When comparing to the simulated curve, this constant advantage becomes even more apparent.

This difference in performance for a given problem size is more pronounced when examining the SK problem. First, we note that with much less computation time (10us compared to $\approx$ 1000us) the RBM is able to outperform both the DWave and CIM for the given problem instances shown in Figure 5.6 A). This difference becomes more apparent when looking at the Time to Solution metric, where we see a 10^5x improvement at 60 nodes compared to DWave in Figure 5.6 B) and a constant 10^3x improvement against the Coherent Ising Machines in Figure 5.6 C). As with the MAX-CUT problem, there appears to be a scaling difference between the RBM and the DWave 2000Q, while the difference between the RBM and Coherent Ising Machine seems to be a constant factor improvement. Final performance comparisons are summarized in Table 5.3 where we can see that the RBM outperforms all other accelerators in time to solution, while also maintaining a lower power budget and running on commodity hardware.

Performance against CPU benchmarks

Simulated Annealing is performed on a Xeon E5-2620 processor using the code from [81], while the Parallel Tempering results are copied from [52] using the NASA/TAMU Unified Framework for Optimization, running on a Xeon E5-1650 v2 processor. We benchmarked against a single algorithm run (no algorithm level parallelization) to fairly isolate algorithm performance differences from device level parallelization. The RBM performs better than both algorithms for small problem instances on the MAX-CUT problem (closer to 5x improvement), but converges for the larger instances presented in the dataset. Although parallel tempering narrowly outperforms the RBM on large MAX-CUT instances, the empirically seen $\mathcal{O}(e^{\sqrt{N}})$ scaling of the RBM is asymptotically favorable to the $\mathcal{O}(e^{N})$ scaling of the parallel tempering algorithm. The RBM is able to outperform both these algorithms on the SK problem across all problem instances, demonstrating a performance advantage for problems with full connectivity. The RBM performs competitively with these state of the art algorithms, but more work needs to be done to increase the performance relative to these baselines. Many of the optimizations used in the simulated annealing (pre-computing

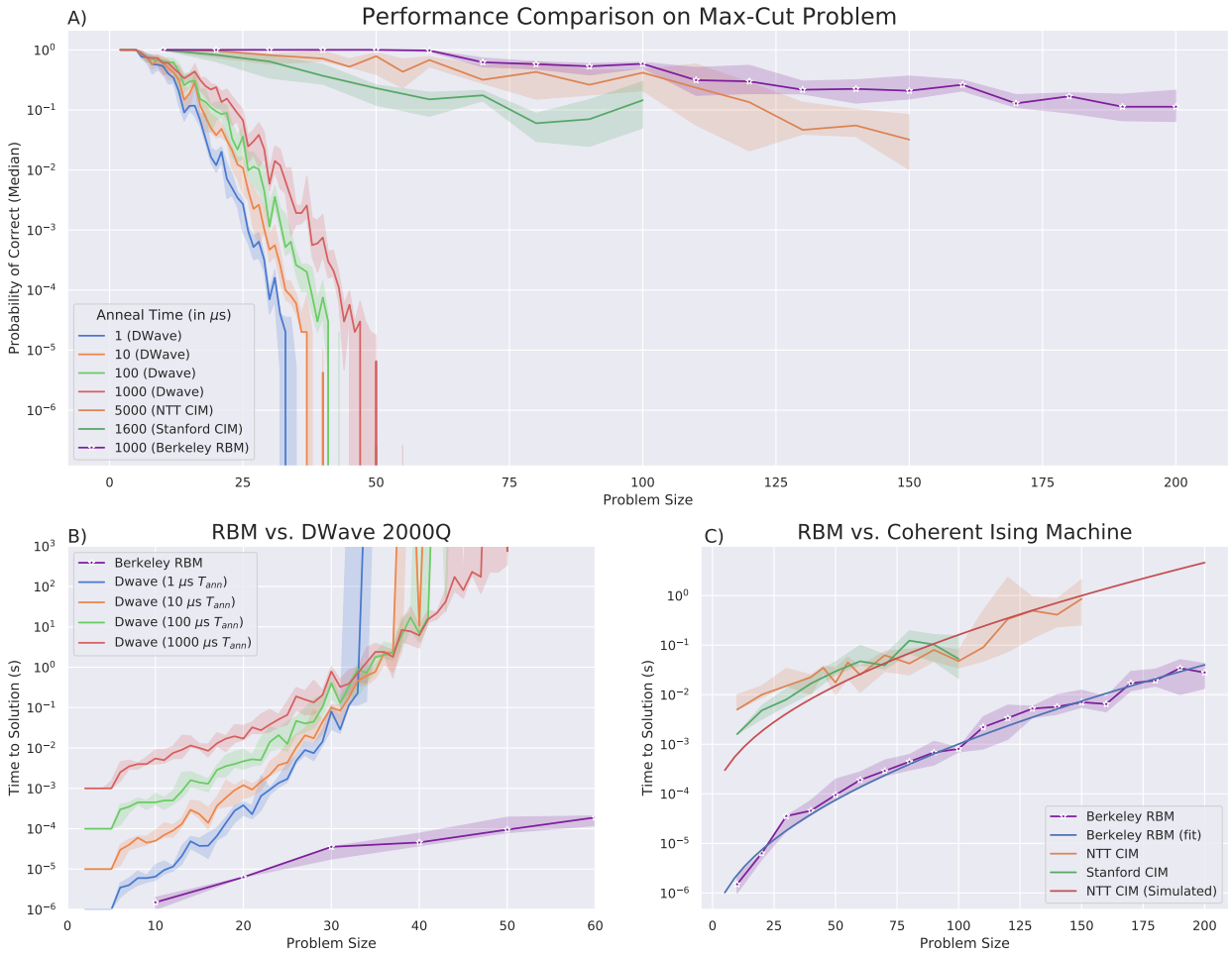

Figure 5.5: . **Benchmarking and Comparisons on the Dense MAX-CUT Problem** **(A)** A comparison of performance using the probability of reaching the ground state in various physical annealers as compared to the FPGA accelerated RBM. We see that for similar annealing times, the RBM achieves a best in class probability of reaching the ground state while maintaining a faster annealing time. **(B)** Using the time to solution framework described by 4.1 and above we compare the performance of Dwave 2000Q to the FPGA accelerated RBM. We see a 7 order of magnitude difference in time to solution for the largest problem instances that the DWave can fit. In addition, we show that the RBM has better scaling properties, with performance differences increasing dramatically with problem size. **(C)** Comparing the RBM to the Coherent Ising Machine created by NTT [5] and Stanford [6, 52] we see a constant factor performance improvement of $\approx$ 150x across all problem instances. The RBM shows similar asymptotic scaling to the CIM with both algorithms scaling as $\mathcal{O}(e^{\sqrt{N}})$.

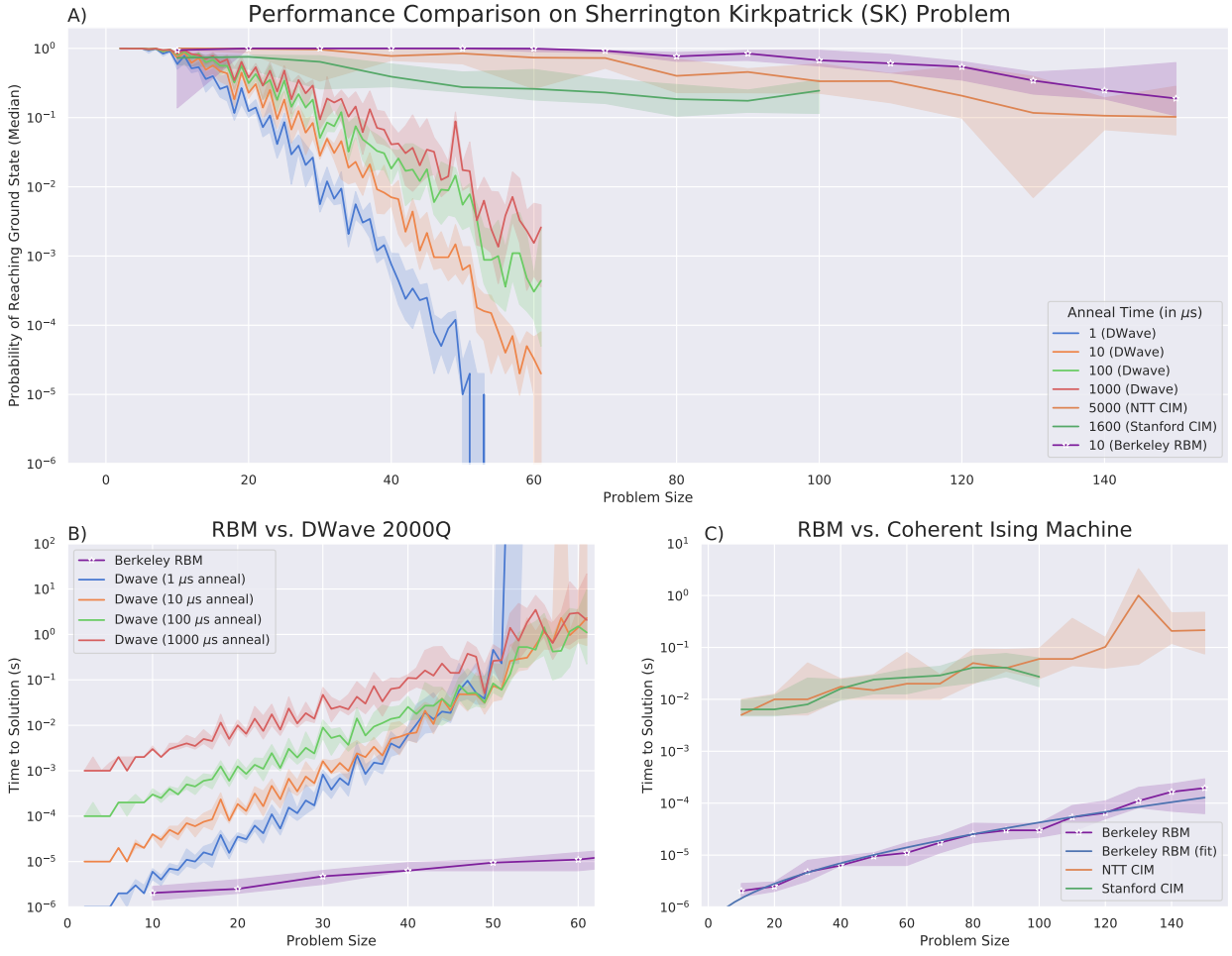

Figure 5.6: . **Benchmarking and Comparisons on the Sherrington-Kirkpatrick (SK) Problem**
(A) Similar to Figure 5.5 A), we compare the performance for a fixed Anneal Time on the Sherrington Kirkpatrick. Compared to the MAX-CUT problem, the RBM performs considerably better on this problem instance, only requiring 10 μs to get to the ground state with very high probability. This is compared to DWave and the CIM requiring 100x the anneal time to get close to this performance. **(B)** Compared to the DWave 2000Q, we see a performance increase of 10^5 on large problem instances with better asymptotic performance on the RBM in these problem instances. The lack of connectivity for the DWave annealer contributes to the drop in performance on these instances as many logical copies need to be made to accommodate the fully connected SK graph. **(C)** The RBM also compares very favorable to the two instance of the Coherent Ising Machine [5, 6], with a 1000x time to solution difference on the largest problem instances. The scaling performance of these two problems also suggests that the RBM will continue its constant factor performance increase for much larger instances of the SK problem.

	Berkeley RBM	NTT CIM [5]	DWave 2000Q [18]	CPU [81]	GPU [54]
Clock Frequency	70 Mhz	-	-	2.1 Ghz	1.2 Ghz
Power	5.8W	-	25kW	$\approx$ 20W	$\approx$ 50W
Time to Solution ($N = 150$, SK)	0.2 ms	215 ms	262 ms (N=50)	12 ms	445 ms
Time to Solution ($N = 150$, Max-Cut)	7 ms	855 ms	10^6 ms (N=50)	13 ms	590 ms
Scaling of Success Probability	e^{-N}	e^{-N}	e^{-N^2}	e^{-N}	e^{-N}

Table 5.3: **Single Threaded Performance Comparison Across Accelerators** While using less power and operating at a lower clock frequency, the RBM outperforms the other traditional accelerators, while also showing better performance than the novel accelerators.

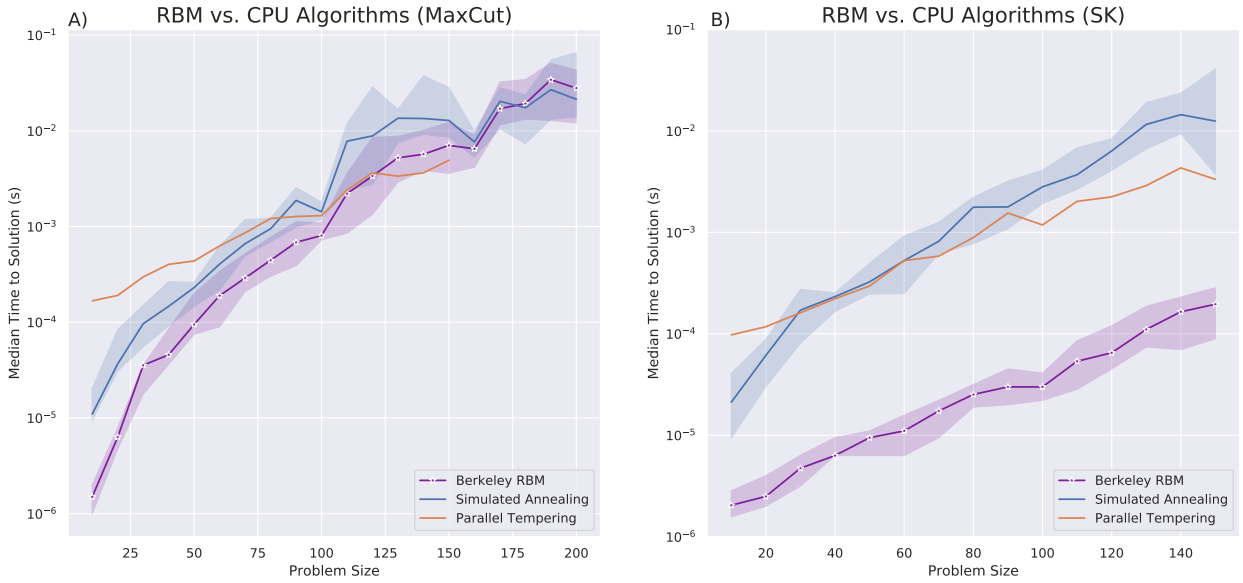

Figure 5.7: . **Benchmarking against CPU algorithms**
(A) The RBM performs competitively with two state of the art CPU algorithms for Ising Model problems, simulated annealing [81, 2] and Parallel Tempering [52, 82, 83]. **(B)** Comparison of the SK Problem against optimized CPU algorithms also yields constant factor speed improvement on the accelerated RBM. Across all problem instances we see a 10-20x speed improvement due to the hardware acceleration.

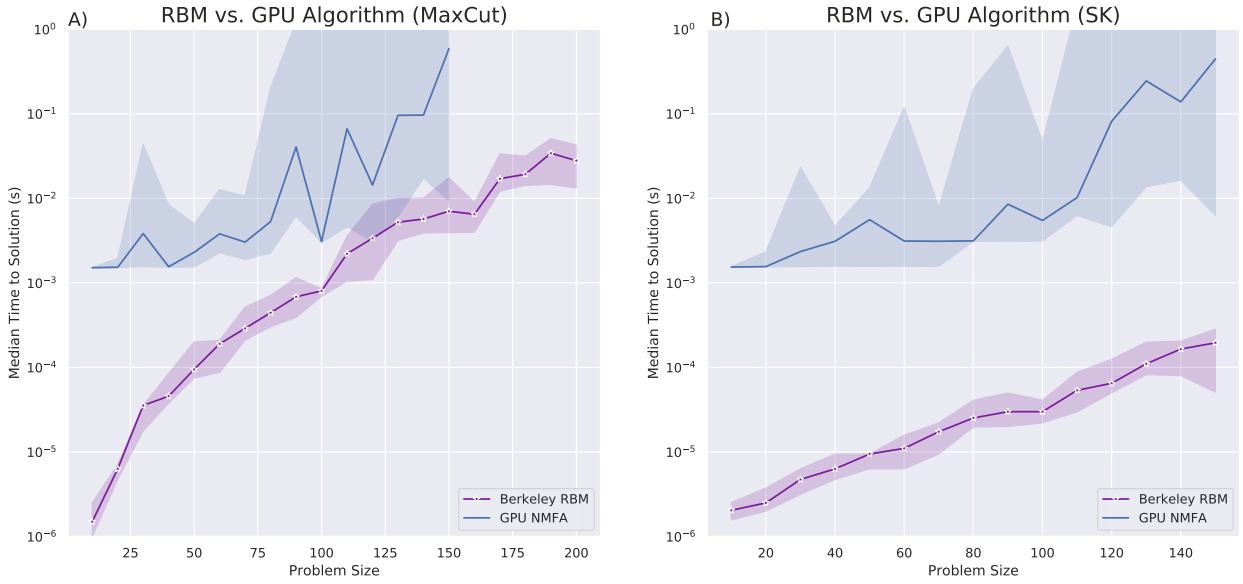

Figure 5.8: . **Benchmarking against GPU algorithms**
(A) Our FPGA accelerated RBM performs well against a GPU accelerated Noisy Mean Field Algorithm [54]. For larger problem sizes, we see a 10-20x speed increase, with a large decrease in variability. **(B)** The SK problem shows further speedup on these problems, with ¿100x improvement through all problem sizes.

energies, using weight sparsity, efficient random number generation) can be implemented with the RBM as well to increase its performance. Parallel tempering can also be added to the RBM sampling algorithm to yield improved sampling, which we expect to improve the overall algorithm scaling and performance. [36, 84]

Performance against GPU benchmarks

The noisy mean field annealing algorithm from [54] is run on a Nvidia Titan V GPU to benchmark against another parallel accelerator based algorithm. We use the same methodology presented in Ref [54], with identical simulation parameters and temperatures. We benchmarked against a single algorithm run (no algorithm level parallelization) to fairly compare algorithm performance differences. The GPU algorithm utilizes parallel neuron level updates to divide computation up amongst different threads. The fine grained updates of the RBM algorithm on our accelerator were able to outperform the calculations in the comparatively faster GPU. This can be explained by our usage of fixed point accumulates rather than explicit multiplications, fine grained clock level synchronization, and a distributed memory system. This shows that our RBM implementation performs favorably with state of the art algorithms across many accelerator types for these problems.

Node Count	Clock Freq	LUT %	FF%	URAM%	DSP%
4096	150 Mhz	40.49	17.3	53.33	14.97
5120	100 Mhz	60.99	24.96	66.67	18.71

Table 5.4: Scaled FPGA utilization table, all synthesis and implementation done for Xilinx VCU118 development board

5.5 Scaled RBM on FPGA Accelerator

While the accelerator shown in the last two sections has demonstrated state of the art performance on small problems, it does not have the scale to solve real life problems, which may use 1000s if not 10,000+ nodes. The solver architecture shown in Figure 5.3, which uses in-place transpose operations, and exclusively registers to store all of the weights and biases, cannot scale passed a few hundred nodes due to limits of memory in the FPGA. In this section, we will explore an alternate architecture which is able to efficiently use a weight streaming architecture to solve larger scale problems (up to 1000s of nodes).

The architecture features 3 main parts; a memory management engine, a stochastic sampling engine, and a probability estimator. The memory management engine streams rows of the weights matrix to the sampling engine utilizing as many RAM blocks as is available on the FPGA system. It also interfaces with the host system to receive the problem to be solved. The sampling engine performs parallelized updates of the visible/hidden states using the Gibbs Sampling update. The probability estimator takes the visible/hidden states being updated and keeps track of the best state seen so far and streams this back to the host to indicate problem completion. An image of this weight streaming architecture is provided below (Figure 5.9).

Our work relies on a few new innovations in the architecture over our previous published works. To avoid storing both the matrix and its transpose, we utilize a dual architecture for hidden updates and visible updates, allowing for simultaneous update of both visible and hidden nodes while maintaining the same weight streaming architecture. Additionally, the binary activations allow for multiplier-free system for RBM implementations, yielding further hardware efficiency. Based on our studies we believe that this will present state of the art performance on large problem sizes.

Weight Streaming and Memory Architecture

To effectively use the memory blocks available on the Ultrascale+ architecture, we chose to use the UltraRAM (URAM) blocks available. These are 72 bit x 4096 line blocks which can operate at close to 500-600 Mhz. The memory hierarchy is set up so that each URAM block holds 72 columns of the weight matrix, with each row of the weight matrix corresponding to a row of memory. This allows for each row of the weight matrix to be split across multiple

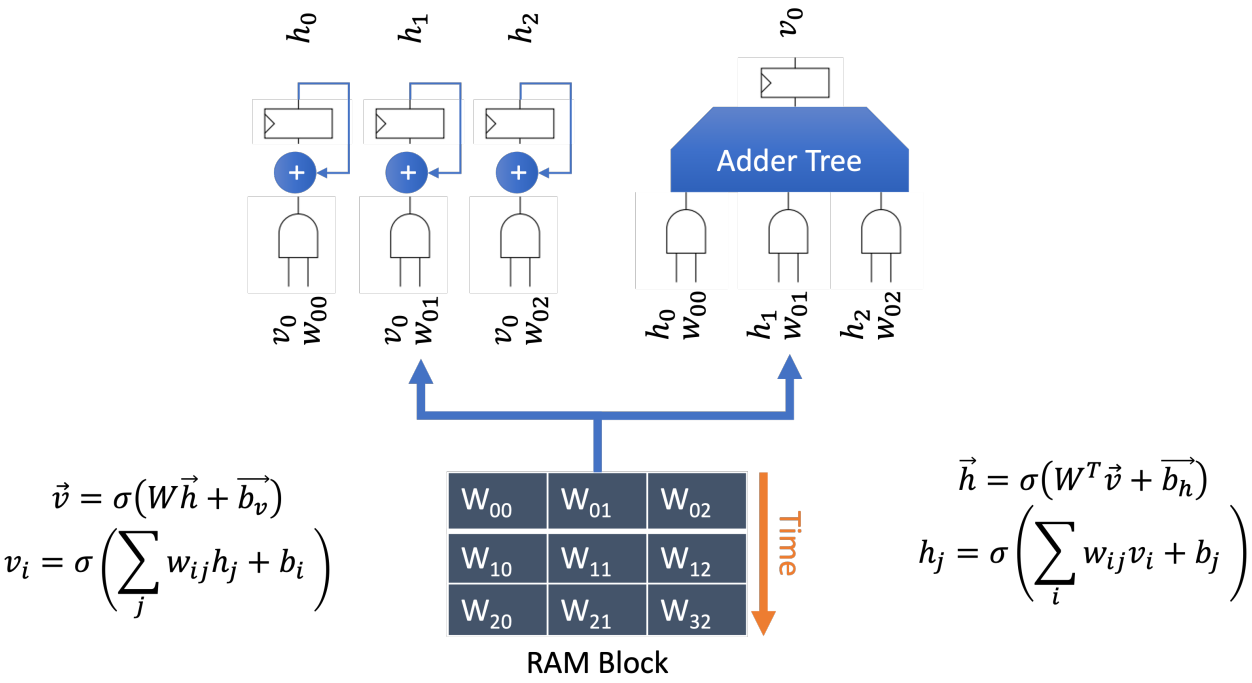

Figure 5.9: A description of the weight streaming architecture which solves the transpose problem of the RBM by having a dual architecture by updating both the visible portions (right) and the hidden portions (left) simultaneously. Rows of the weight matrix are streamed through time

URAM blocks and is thus fetched every cycle. Each cycle the URAM row is then fed into the visible or hidden update engine that is described below.

Sampling Engine

The sampling engine is composed of two parts, the visible update mechanism and the hidden update mechanism. Both of these update mechanisms take each row from the weight matrix and consumes it within one cycle of latency with a 3 stage pipeline structure to line up completion of computation. The visible and hidden updates have alternate architectures which allows the system to compute on the matrix and its transpose at the same time, circumventing complex memory structures. [59] One full sample update is produced every n cycles, where n is the number of rows in the weights matrix.

Visible Update Mechanism

The visible updates functions on an input stationary format. This is where the hidden activations (which mask the weights streaming into the array) are held constant while the

	LUT %	FF%	URAM%	DSP%
Total	40.49	17.3	53.33	14.97
Memory Management	12.04	1.01	53.33	0
Hitting Time Engine	6.20	1.53	0	0
Visible Update	1.83	1.88	0	0
Hidden Update	18.48	4.18	0	14.97

Table 5.5: Breakdown of FPGA utilization in Scaled system by specific sub-component usage. This shows that the vast majority of logic resources is consumed by the hidden update module.

rows of weights from the URAM stream across and are accumulated to form the output [85]. Each row of the visible update is passed into a pipelined adder tree which passes the value into the sigmoid look up table, and finally compares to a random number generator from the LFSR described in previous sections. Every cycle, this new visible activation updated and stored inside a register array until computation is complete after n cycles.

Hidden Update Mechanism

The hidden update, unlike the visible updates, functions on an output stationary format. Each hidden node must accumulate the value of a column of the weight matrix, which is updated cycle by cycle. For this reason, an accumulator is instantiated for each hidden neuron (for a total of n accumulators instantiated for the system). At the end of n cycles, the accumulators pass their values to the sigmoid look up table, and then compare against a LFSR generated random number to get the final node activation value.

To efficiently use FPGA resources, the mask and accumulate operation is done entirely within the DSP blocks. Each of the DSP48 blocks in the Ultrascale+ architecture can perform 4 SIMD accumulate operations, greatly saving on fabric resources. However, even with this additional optimization, the hidden update mechanism takes the majority of the utilization as it requires us to instantiate n different accumulators and sigmoid look up tables to achieve a reasonable latency of computation.

Performance and Analysis

The RBM accelerator on FPGA demonstrated here is shown to be the largest node count for a single FPGA design of a fully connected Ising Model system. The results are shown in Table 5.5 as compared to other systems. We can attribute the ability to tackle such large problems to the efficient usage of compute resources via the multiplier-free architecture, with the usage of pipelining and time based multiplexing in this design. Additionally, the

mapping onto even larger problems was stopped by on-chip memory size limitations (rather than computation resources) which suggests that by using off-chip DRAM we can compute even larger problems.

Performance for the scaled RBM accelerator has been validated on a series of MaxCUT problems. Here we are showing the performance of the FPGA accelerator on problem sizes of 800-2000 nodes, with mixed results. The problems explored here are from the Gset problem instances, and the K2000 problems both standard problems solved in literature by other accelerators [86, 87]. The accelerator is able to find approximate solutions to the problem quickly, but struggles to reach the exact ground state of these benchmark problems. We can attribute this to the standard Gibbs Sampling algorithm struggling in high dimensional spaces, and the need to traverse through large areas of the state space with low probability to find other areas of high probability. This has led us to look for further algorithmic improvements in future iterations of the FPGA architecture, such as Parallel Tempering and Annealed Sampling. The prototypes of these algorithmic changes are discussed further in the following chapter.

In Figure 5.10 we demonstrate that the sampling based FPGA system can reach the ground state of the 800 node G6 problem. The figure demonstrates analysis with respect to a fixed coupling parameter. The G6 Problem is a fully connected problem with random $\{+1, -1\}$ weights. Although we are able to solve the problem after ¿500 trials, the 0.53 seconds taken to solve the problem once this case is much slower than the state of the art for solvers such as the Simulated Bifurcation Machine developed by Toshiba on a similar FPGA, which is able to find the ground state of the problem in 0.012 seconds. This means that we could expect this FPGA solution to be 50-100x slower than the state of the art, suggesting that the scaled RBM on FPGA solver should be redesigned for higher speeds, or a more sophisticated solution method.

Additionally, we investigate other Gset problems of size 800, like the G6 to G13 shown in Figure 5.11. The G6, G7, G8, G9 and G10 problems are all random $\{1, -1\}$ weights, which is why we see that their performance is similar to eachother. However, the G11, G12, and G13 are random toroidal graphs perform worse than the other problems, suggesting that they may need a more advanced mapping to the RBM accelerator. Toroidal graphs have a more direct mapping to the RBM architecture, as they are bipartite by nature, but this property was not exploited when mapping to the FPGA accelerator in this instance.

Finally, we examine the performance of the K2000 problem as shown in Figure 5.12, which is a problem analyzed by many accelerators in this space. Similar to the G6 problem shown above, we find that the performance on this problem was not as high as expected. We can attribute this to the vanilla Gibbs sampling algorithm struggling at even higher node counts, as the state space increases. We also note that with this architecture, to run the K2000 problem for 100,000 sample steps (as tested here) would take 1.33 seconds to solve the problem a single time. This is much slower than the state of the art, where the Simulated Bifurcation Machine has shown the ability to reach the exact best known state within 1.3 seconds. Further discussion of algorithmic improvements using parallel tempering is done in the following chapter.

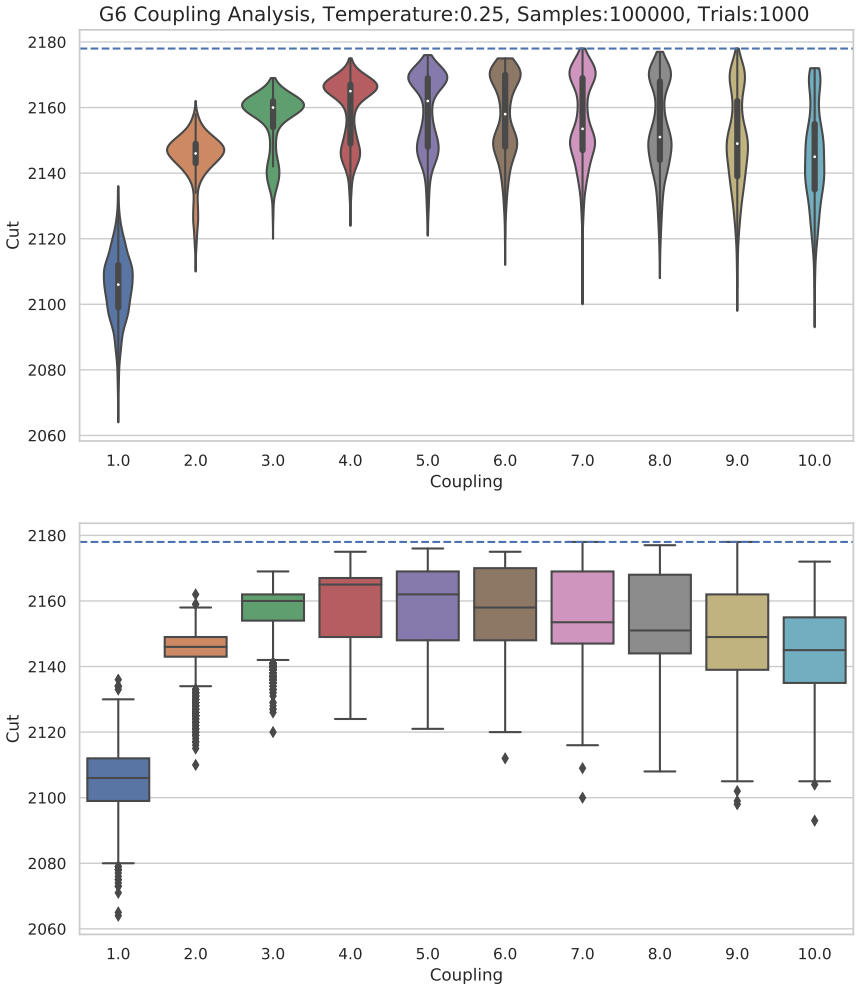

Figure 5.10: **Coupling and distributional analysis of the G6 problem Top**: Violin plot of problem distribution showing that for these problems the optimal coupling value is between 7 and 9. We note that although the larger coupling values tend to get to the ground state more often, smaller coupling values have a higher average cut. **Bottom**: Box plot of the same distribution showing the median cut is higher for lower coupling values, but the higher coupling values are able to reach the ground state.

Name	Operating Principle	Year Published	Spins	Hardware Platform	Connectivity	Precision
Toshiba SBM	Simulated Oscillator Bifurcation	2019, 2021,	4096 (Gen1) 2048 (Gen2)	FPGA	All-to-all	2 bit (Gen1) 16bit (Gen2)
Coherent Ising Machine	Optical Parametric Oscillator	2016	2048	Optical Fiber + 2 FPGA	All-to-all	2 bit
Coherent Ising Machine	Optical Parametric Oscillator	2022	100,000	Optical Fiber + 56 FPGA	All-to-all	2 bit
Fujitsu Digital Annealer	Parallel Tempering	2019 (Gen1) 2020 (Gen2)	1024 (Gen1) 8192 (Gen2)	ASIC	All-toall	16 bit (Gen1) 64 bit (Gen2)
RBM on FPGA (Gen 1)	Stochastic Sampling	2020	200	FPGA	All-to-all	8 bit +
RBM on FPGA (Gen 2)	Stochastic Sampling	-	5020+	FPGA	All-to-all	8 bit +

Table 5.6: Comparison of various physical annealers which have all-to-all connections, in single chip format. We are not including in this list software systems which create all-to-all connections using a CPU/GPU algorithm

5.6 Conclusion

In this chapter we have outlined two generations of FPGA based RBM accelerator. The first generation of RBM based FPGA accelerator continues to show state of the art improvements on small problem sizes (up to 200 nodes), as benchmarked by other researchers [88]. This accelerator was used first to demonstrate speed improvement on integer factorization and then extended for general combinatorial optimization tasks where it showed state of the art performance on small ($<$ 200 node) problems. Finally, we scaled up this accelerator to a 5000+ node system, but encountered algorithmic and hardware challenges of scaling up to large size. To counter these scaling issues, we suggest using Parallel Tempering (as outlined in Section 2.4), which we explore in the context of GPU systems in the next chapter.

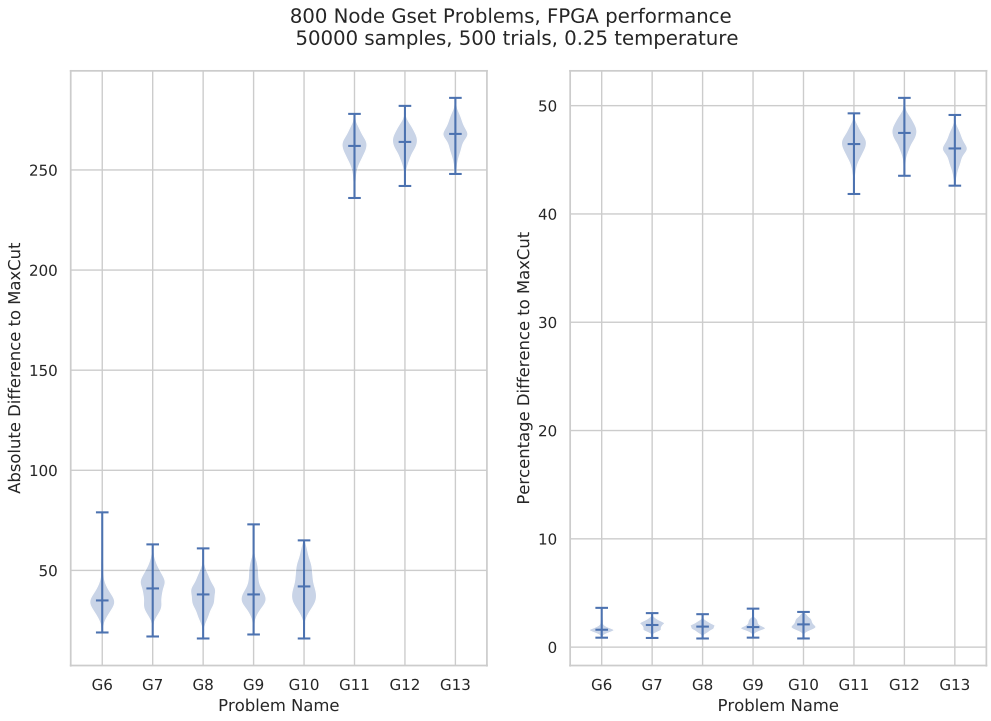

Figure 5.11: **Distributional analysis of 800 node Gset problems with 50,000 samples Left**:We see that the fully connected random instances (G6, G7, G8, G9) get closest to the ground state energy, with most of them performing similarly. The Toroidal Instances (G11, G12, G13) perform worse due to an inefficient mapping to the RBM structure. **Right**:When normalizing to the optimal MaxCUT value, we see that the random instances perform well, while the toroidal instances perform very poorly.

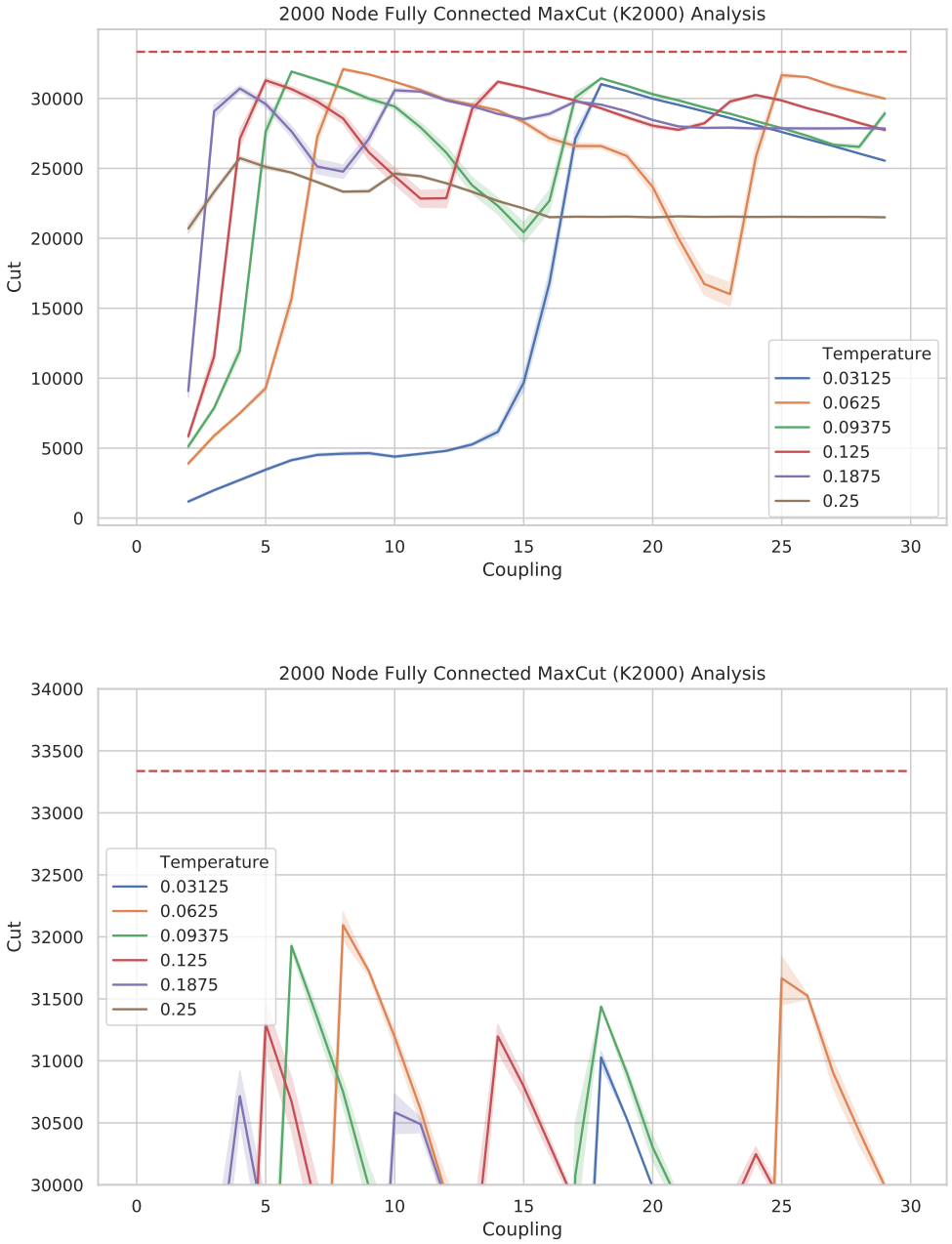

Figure 5.12: Analysis of performance of the K2000 MaxCUT problem. We analyze performance based on temperature and coupling parameter to find the optimal parameter set for our instance. For these problems, a $\beta = 0.0625$ mixed with a coupling of 9 seems to perform best. However, even after 100,000 samples we are not able to reach the ground state of the system, suggesting we need more advanced sampling techniques.

Chapter 6

TPU and GPU based RBM Accelerators

6.1 Introduction and Motivation

While the FPGA based systems presented in the previous chapter exhibit fine-grained parallelism and the ability to exploit various hardware and algorithm specific efficiencies, FPGA systems also have their downsides. In this chapter we will explore the use case of general-purpose parallel hardware to accelerate optimization problems with in the RBM framework, using the MAX-CUT problem as a problem of interest.

Comparison of FPGA to traditional parallel Compute Platforms

While FPGA systems have the ability to emulate a large variety of systems, they aren't specifically designed for any particular type of computation, meaning that the FPGA will not be nearly as efficient in compute usage as a dedicated ASIC designed for that particular application. The downsides of using an FPGA are listed below. Consequently FPGAs excel when there are complicated kernels which have a short latency budget, and which regular CPUs do not perform well at.

Unlike the CPU (Central Processing Unit), which is designed for general-purpose computing and sequential processing, a Graphics Processing Unit (GPU) excels at parallel processing tasks. It consists of thousands of smaller cores, grouped together into streaming multiprocessors (SMs) or compute units, which work together to execute tasks in parallel. This architecture enables GPUs to process a large number of data elements simultaneously, making them exceptionally efficient for tasks that can be divided into smaller, independent computations, such as graphics rendering, machine learning, scientific simulations, and more.

Using a GPU for large-scale linear algebra operations offers significant advantages due to the GPU's parallel processing power and efficient handling of matrix computations. In linear algebra, many operations involve performing numerous matrix multiplications, additions, and other transformations, which can be highly parallelizable. GPUs excel in handling these

tasks by leveraging their thousands of cores to process data simultaneously. As a result, complex calculations and computations involving large matrices can be performed at high speeds, significantly reducing the overall processing time.

A Tensor Processing Unit (TPU) [57] is a specialized application-specific integrated circuit (ASIC) developed by Google specifically for accelerating machine learning workloads, particularly those involving neural networks and tensor operations. TPUs are designed to provide high-performance, energy-efficient processing of tensor data, which is a fundamental mathematical construct used in deep learning algorithms.

The architecture of a TPU is optimized for matrix multiplications, which are prevalent in neural network computations. It consists of a large number of processing units, including multipliers and accumulators, capable of performing tensor operations in parallel. This massively parallel design allows TPUs to process vast amounts of data simultaneously, significantly speeding up the execution of machine learning models and reducing the training time.

Below we discuss the tradeoffs between these three architectures, and why using TPUs and GPUs at larger scales may be beneficial as compared to FPGA based systems.

- **Off-chip memory bandwidth** Many FPGA systems and boards will have lower off-chip memory bandwidth than a similar CPU or GPU type system. This means that solving memory bound systems will be stuck in memory bottlenecks. For example, an Nvidia A100 chip has bandwidth up to 2TB/sec, the TPUv4 has 1.2 TB/ssec, while the Xilinx Ultrascale+ HBM chip only has 460 GB/s (a 4x difference) in bandwidth

- **Off-chip memory sizes** FPGA systems are not generally optimized for large amounts of high bandwidth memory, the largest size FPGA off-chip HBM memory is 16GB (for Xilinx chips), while Nvidia offers up to 80GB of HBM memory and the TPUv4 offers 32GB of memory

- **Ease of programmability** Algorithm change and experimentation is extremely slow on the FPGA systems, meaning that the speed and efficacy of new algorithms can't be readily explored

- **Hardened Linear Algebra Kernels** New generations of GPU systems are being specifically designed for large scale linear algebra operations, with hardened kernels that are able to do Multiply Accumulate (MAC) operations in a few clock cycles. Additionally, the prevalence of tensor specific hardware in GPUs means that they can solve large matrix multiplication operations very efficiently.

- **Large Parallel Streaming Processors** With the hardened linear algebra kernels, the new generations of TPU and GPU processors are designed with large linear algebra in mind. This means that their performance benefits start to become apparent in large problem size and large batch size applications (many parallel chains).

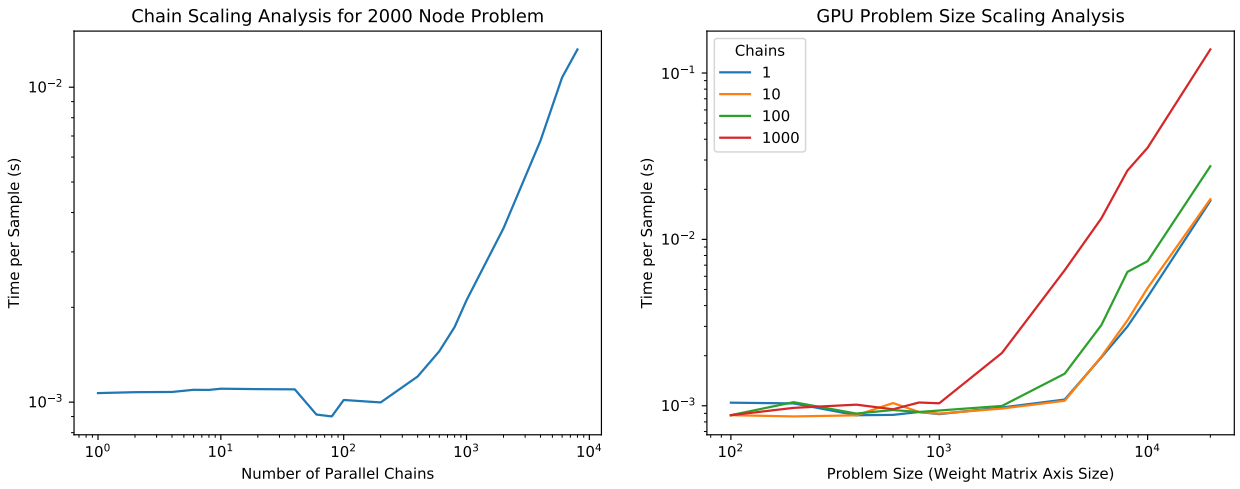

Figure 6.1: **Understanding parallelism in the GPU** Left: For a 2000 node problem, we see that as we add many parallel chains to the system, the time to take a single sample is constant, until we reach the point where the GPU is no longer memory limited, but compute limited. We can take more parallel Monte Carlo steps "for free" on the GPU. Right: For various problems, we see that the point of memory bandwidth limitation starts earlier with higher batch sizes. Again, the time per sample is constant until the point of memory bandwidth limitation. Experiments conducted on an Nvidia Titan V system.

- **Clock Speed** As the GPU is a specialized ASIC for linear algebra operations, it reaches clock speeds of 1.4Ghz, the TPUv4 reaching 1Ghz while our FPGA implementations went at a maximum of 200-300Mhz.

With these tradeoffs in mind, it becomes clear that for larger systems where we can design our model to use the GPUs parallel architecture. To motivate the usage of the GPU system, we first examine the use of many parallel MCMC samplers and the benefits it can provide.

6.2 Parallelized MCMC

The GPU and TPU systems are most efficient with high levels of parallelization and data reuse. For the GPU, the shared memory cache that is near each streaming mulitprocessor (SM) is usually very small compared to the L1 cache in a CPU or compared to the on-chip memory in an FPGA. The biggest piece of memory that is transferred from DRAM to the GPU/TPU is the weights matrix, so this memory must be maximally reused.

To accomplish reuse of the weights matrix, we choose to perform parallelized MCMC methods. This means we initialize multiple random MCMC runs (what we call "chains") which progress independently. This changes the matrix-vector multiplication of Wv to a

matrix-matrix multiplication, effectively reusing elements of the W matrix k times for k chains. We can then take the maximum value achieved from the ensemble of chains as our "answer" with a hitting time engine for each chain.

For a small number of chains, the computation in the GPU is memory-bandwidth bound, and the maximum time is spent retreiving elements of the weight matrix for each computation in the SMs. This effect is seen in Figure 6.1 where we see the time to process a sample for a 2000 node problem is constant for a small number of chains, until about 100 chains. After this we see the expected linear growth in processing time for each individual chain added. Similarly, the time per sample is constant for problem sizes up to 1000×1000 matrices, and then begins to increase for larger problems. This can be attributed to fixed launch time for the GPU kernels, and the overhead of the GPU scheduler and cores.

Parallel Tempering

To effectively take advantage of the parallel architecture of the GPU and the use of multiple chains, we adopted the parallel tempering algorithm, as described in Section 2.4 [36, 84]. The parallel tempering algorithm builds upon the usage of the parallel chains described in the previous chapter and is able to provide improved performance compared to simply running multiple chains and taking the best value taken amongst the chains.

As each parallel tempering swap step involves swapping the activations between adjacent temperatures, it is a memory intensive operation which can become a bottleneck. To combat this, swap steps are performed every 10 MCMC steps which also has the added benefit of allowing each chain to equillibriate with the swapped sample. The approximate probability is taken from the hitting time calculation described in Section 5.4, and used to calculate the swap probabilities from Section 2.4. This method allows for maximal reuse of calculations from the visible and hidden activations and introduces minimal overhead.

6.3 Programming Paradigms

When programming the GPU and TPU, we followed the paradigm of using the simplest methods that would provide reasonable performance. When doing this, we had the option of using more high-level interpreted languages with accelerated packages (like Python using Pytorch) vs. using more low-level languages that might provide better performance (such as C/C++ with CUDA). As our goal was to show proof of concept results, and not the most performant systems, we chose to use high level interpreted languages for most computation.

JAX and JIT compiled languages

When our goal was to increase performance as compared to the base PyTorch implementation, we chose to use the JAX framework for acceleration. This provided a few benefits. Firstly, it allowed our code to be portable between the TPU and GPU systems. The same

working code on the TPU could then be transferred to many GPUs with little modification. We could even expect the XLA compiler to take care of different GPU systems (such as using the Nvidia Titan V from 2017, and the Nvidia A100 from 2022), while remaining performant for both. The next main benefit for using JAX is the ability to Just-In-Time (JIT) compile functions, which means that we have the benefits of a high level language (JAX uses Python as the base language), while getting most of the speed benefits of a lower level language.

In our experience when transferring code between PyTorch and JAX, we were able to achieve a speed up of 2-4x without much additional tuning. This can be attributed to JAX being able to compile linear algebra kernels together to save on memory requirements while merging compute kernels to achieve better performance.

6.4 Results and Discussion

Using parallelized GPU based algorithms along with Parallel Tempering allows the system to find ground states of larger problems than the previous vanilla Gibbs Sampling method. This can be seen in Figure 6.2, where the "nominal" Gibbs Sampling algorithm is unable to find the ground state of the system after 40,000 samples. Indeed, even when sampling proceeds for many more samples the nominal Gibbs Sampler is unable to find the ground state of the system. However, by adding Parallel Tempering to the algorithm we see that the time per sample is slower but the system is able to find the ground state solution of the system.

These results can be compared to the FPGA based system as described in the previous chapter, where after many sample steps the FPGA could not reach the solution for the K2000 problem or to many of the Gset problems. In the below Table 6.1, after 40,000 samples of parallel tempering with 1000 temperatures, the system is able to find the ground state for problems up to 2000 nodes, while being within 1% of the ground state of problems up to 7000 nodes. As noted in the previous chapter, the RBM based sampler performs worse on toroidal graph problems than fully connected graph problems due to the poor mapping of the toroidal graph onto the RBM.

Similarly when analyzing the performance of the K2000 problem in Figure 6.2, we find that the RBM sampler reaches the ground state solution, within 20,000 samples. From these experiments, and rough performance benchmarks, we can estimate the Time to Solution (TTS) for the K2000 problem as $\approx$ 120seconds. This comes from using 40,000 samples, 500 temperatures and a 90% probability of reaching the ground state. We note that this is running un-optimized JAX code on a GPU, and only 1 batch system. By optimizing the code and moving to an FPGA we would expect to get within at least 10x of the best known ground state solution of the Toshiba Simulated Bifurcation Machine on an FPGA [86]. There have also been suggestions that other sampling techniques may be able to improve up on the performance of the Parallel Tempering and sampling systems we have shown here, such as Adaptive Parallel Tempering, Adaptive Monte Carlo, and others.

Problem	Connectivity	Nodes	Edges	BKS	RBM PT	Diff
G17	random	800	19176	2006	2006	0
G18	random	800	19176	2005	2005	0
G19	random	800	19176	2054	2054	0
G10	random	800	19176	2000	2000	0
G11	toroidal	800	1600	564	564	0
G12	toroidal	800	1600	556	552	-4
G13	toroidal	800	1600	582	579	-4
G27	random	2000	19990	3341	3341	0
G28	random	2000	19990	3298	3298	0
G29	random	2000	19990	3405	3405	0
G30	random	2000	19990	3413	3414	0
G31	random	2000	19990	3310	3310	0
G56	random	5000	12498	4017	3983	-34
G57	toroidal	5000	12498	3494	3419	-75
G61	random	7000	17148	5796	5754	-42
G67	toroidal	10000	20000	6906	6761	-145

Table 6.1: Performance of Gset problems with RBM + Parallel Tempering algorithm. All experiments done with 40,000 samples and 1000 temperatures spaced geometrically between 1 and 100.

Solver	Hardware	Problem	Time to Solution (TTS)
Toshiba SBM	FPGA	K2000	1.3s
RBM + PT	GPU	K2000	120s

Table 6.2: Performance comparison for Time to Solution on the K2000 problem. The RBM system is running JAX code on the Nvidia A100 GPU, with 500 temperatures and 40,0000 samples. Time to solution is estimated from Equation 4.1.

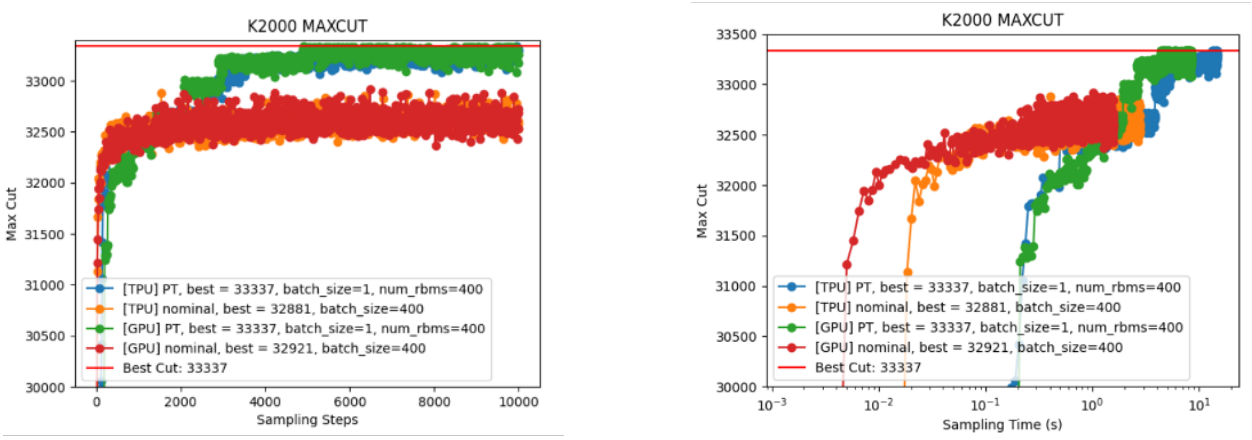

Figure 6.2: **Comparing Parallel Tempering to Sampling on K2000 problem** Left: Comparing sample for sample, we see that Parallel Tempering is able to outperform nominal parallelized sampling. Performance on the GPU and TPU is the same, as they are running the same algorithm. Right: When looking at time taken to reach a solution, we note that Parallel Tempering takes longer (per sample) but is able to reach the best known solution to the problem, while the regular sampling solution is unable to do so. Between these two systems, we see that the GPU is slightly faster per sample than the TPU. We can attibute this to the usage of the TPUv2 (released in 2017) vs. the Nvidia A100 (released in 2020) which are from different process nodes.

Scaling to 100,000 nodes

With the scaled RBM sampler, we have made strides in scaling to large and ultra-large problem instances. The largest all-to-all connected Max-CUT problems solved on existing hardware has been 100,000 node problems, demonstrated in 2021 on a cluster of 16 GPUs [86]. In an effort to compare performance we have mapped the RBM sampler onto problems of this size, demonstrated in Figure 6.3.

At problems of this size, we start to have non-intuitive results from the previous problems we have solved. To start, we see that as the number of parallel copies increases, the performance starts to degrade significantly (see left side of Figure 6.3). We can attribute this to the extra computation involved in the extra copies, and the GPU becoming severely compute limited. Additionally, when we look at the right panel of Figure 6.3 we find that vanilla sampling starts to outperform Parallel Tempering at the large problem scales, due to the significantly larger overhead of doing sample swaps.

Due to the difficulty in implementing these large problems (problem generation alone for the M100000 problem took hours on its own), substantial hyper parameter tuning became difficult. As such, the problem instances demonstrated here don't represent optimal hyperparameter tuning or problem tuning. Additionally, we note that the SB (Simulated Bifurcation) and SA (Simulated Annealing) both used a cluster of 16 GPUs and 16 CPUs

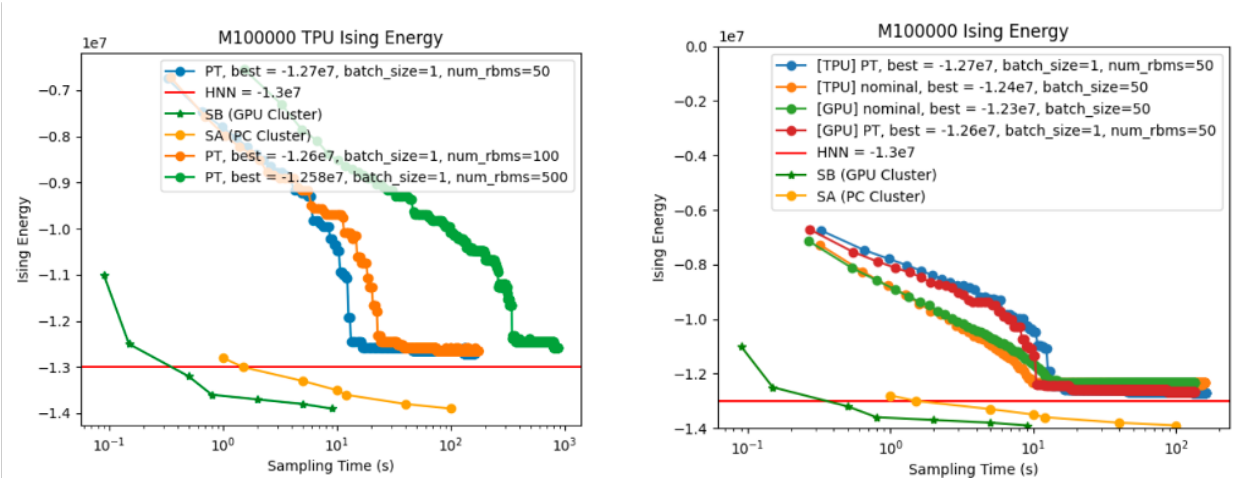

Figure 6.3: **RBM + Parallel Tempering Performance on Ultra-Large Problems** Left: Performance Analysis of RBM sampler vs. number of Parallel Tempering copies. Right: Performance Analysis of RBM sampler on GPU vs. TPU, and comparing sampling vs. parallel tempering. The SB (Simualted Bifurcation) and SA (Simulated Annealing) [86] are shown on a cluster of 16 GPUs and 16 CPUs respectively, while the RBM + Parallel Tempering is run on a single Nvidia A100 GPU.

respectively, while the RBM + Parallel Tempering implementation demonstrated here only used a single GPU or a single 8-core TPU board. With this in mind we believe there is substantial improvement possible with these implementations.

6.5 Conclusion

In this chapter we have demonstrated using the GPU and TPU systems to integrate a more complex sampling algorithm (the Parallel Tempering algorithm) along with the RBM system. By augmenting the traditional Gibbs Sampling method with this parallel swap method, we find the system is able to find ground states to very large problems. By integrating the GPU into the system, we also demonstrate that this is only the second system to demonstrate the ability to have a Time to Solution (TTS) for the K2000 problem that is under 5 minutes. We also demonstrate possible solutions to 100,000 node MAX-CUT problems, which are the largest problems that have been solved by any machines. This is a major step forward in solutions to difficult combinatorial optimization problems, and shows how the RBM sampling algorithms can be implemented on traditional parallel hardware.

Chapter 7

PASS: The Parallel Asynchronous Stochastic Sampler

7.1 Introduction and Problem Statement

We demonstrate usage of the Parallel Asynchronous Stochastic Sampler (PASS) as a stochastic neural network capable of solving NP-Hard Combinatorial Optimization and Machine Learning problems in a time and energy efficient manner. The PASS accelerator takes advantage of the intrinsic fine-grained, neuron-level parallelism present in the Ising Model and utilizes a multiplier-free, clock-free, probabilistic architecture. This work presents the first fully on-chip, CMOS demonstration of an Asynchronous Stochastic Sampling system, presenting a $\sim 10^6$ speed increase at similar power levels compared to previous MTJ based approaches. Additionally, scaling studies show that our system can solve the MAX-CUT problem 2x faster than any previously demonstrated architecture.

As the demand for big data increases and the speed of traditional CPUs cannot keep pace, a new processing paradigm is needed to tackle computing's most difficult problems. In particular, NP-Hard optimization problems are ones that both scale exponentially and have relevant practical applications in a variety of fields, including vehicle routing, place and route in circuits, scheduling, and many others. In the past, these problems have been solved by complex heuristics, using standard hardware. More recently, accelerators based on the Ising Model have arisen as a method to tackle some of these challenges, but few take advantage of its unique structure. We unlock the intrinsic fine-grained, neuron-level parallelism present in the Ising Model with a novel neural accelerator, the Parallel Asynchronous Stochastic Sampler (PASS), that has the potential for orders of magnitude speed increase over traditional methods for solving optimization problems.

The proposed accelerator offers a novel computational approach based on a clock-free asynchronous architecture. It is based on the Boltzmann Machine binary neural network, which mimics how the brain computes in a massively parallel, distributed, and asynchronous manner. Our previous digital realizations, while demonstrating significant improvements

over quantum computers, traditional digital computers and other accelerators [89], were limited by mappings to traditional computational architectures. PASS follows a sampling-based approach in which each neuron updates asynchronously based on its neighbors following a Poisson clock generated by inherent shot-noise present in advanced CMOS nodes. This method of computing offers a theoretical scaling advantage over traditional digital computing, where neurons are updated synchronously and in order, while also freeing the design to map more arbitrary graphical models.

Our main innovations in development of the PASSO accelerator are: (i) development of a new high-speed stochastic neuron based on shot noise (ii) combining many neurons through a weight-stationary digital fabric to create a fully asynchronous neural network (iii) programming and integrating this system with traditional computers to solve useful NP-hard optimization problems. This work will revolutionize future neural network accelerator architectures through asynchronous and stochastic techniques to meet the demands of a data-driven industry.

7.2 Relation to State of the Art

In recent years, Ising Model based computing has become an exciting architecture for the next generation of computing. A variety of accelerators are being developed utilizing hardware platforms such as specialized ASICs [86, 4], Quantum Annealing [18], Optical Pumping [22], and many others. Our work is distinct from these accelerators for 3 reasons; (i) the majority of accelerators rely on an annealing mechanism, while we support a sampling-based architecture (ii) other accelerators rely on an ad-hoc integration of noise into their system, where noise is a fundamental part of our algorithm and (iii) our work is limited by local interaction speeds rather than set by a global clock as with these other systems, unlocking a fully parallel update scheme.

These 3 points combine to give our system a strong promise for success. Firstly, the use of a sampling-based architecture allows for an efficient hardware design, along with a theoretical advantage. PASSO uses a low precision, fixed point, weight-stationary architecture, causing all calculations to be extremely hardware efficient, reducing the area, power and time to solution. From a theoretical perspective, a sampling-based system is guaranteed to converge to the correct answer, which is not the case for annealing systems. Recent work has also begun theoretically to show that sampling-based algorithms can outperform annealing-based algorithms on certain optimization tasks [90].

Secondly, a fundamental part of the PASSO sampler is the integration of noise to escape local minima. The ad-hoc integration of noise which many accelerators rely on limits how closely the hardware resembles the algorithm which they calculate on. For example, [4] uses noise created by faulty SRAM cells, [22] uses equipment based noise in coherent spin systems. By using noise as a fundamental part of our hardware, we are able to control and characterize it, allowing our hardware to closely follow the algorithm for which it is based

on. This guarantees that our algorithm will find the best possible answer to the optimization problem presented, even for very large problem instances.

Lastly, PASSO relies on a fundamentally localized form of interaction, creating a system which is fully parallel in its computation, with each neuron updating simultaneously. This is accomplished by using asynchronous updates, where each neuron probabilistically updates its state whenever the value of its neighbors changes. This scheme yields theoretically faster convergence (a factor of n times higher, where n is the number of neurons) compared to a deterministic, clock-driven scheme. For instance, when comparing a synchronous accelerator operating at the same effective clock speed as PASSO, we see in simulation that there is a 200x time to solution increase by using a PASS-like scheme as demonstrated in Figure 7.14. This type of update scheme mimics the way our brain computes, and demonstrates a neuromorphic approach to computing within the Ising Model.

7.3 Design of the PASS Accelerator

Neuron Circuit Design

Introduction

We use the inherent shot noise present in advanced CMOS nodes to create a binary, stochastic neuron circuit, which we couple with adjacent neurons through a connection circuit comprising of a digital fabric in an analog-mixed signal architecture, shown in detail in Figure 7.1. The neuron is composed of two pieces, a synapse and a stochastic neuron circuit. The synapse design which is composed of a binary dot product engine, which takes the activations of adjacent neurons to mask and accumulate them, and a Digital to Analog Converter (DAC) which passes the accumulated values into the neuron. The neuron circuit has three major components; a noise source and noise amplifier which generates a stochastic signal for every neuron, a sigmoidal activation function which combines the stochastic signal from the neuron with the output of the synapse circuit, and the output buffer which digitizes the output and drives the signal to other parts of the chip.

Each individual neuron amplifies shot noise to create a stochastic clock based on a Poisson process. This update scheme creates an asynchronous neuron that continuously and stochastically computes on the synapse output voltage. As there is no inherent clock for computation, the neuron speed is characterized by the autocorrelation of the Poisson process that governs it; the faster the autocorrelation decays to 0, the faster the neuron computes. The 150Mhz autocorrelation for the neuron is demonstrated in Figure 7.11 g), where we see the autocorrelation and the exponential fit. Ultimately, this speed is determined by the bandwidth of the amplifiers and size of the noise signal, both of which improve with smaller features, process nodes, and improved design. For this reason, we expect this type of circuit to improve rapidly as technology nodes scale. The Neuron Circuit and Connection Circuit design is described in detail below.

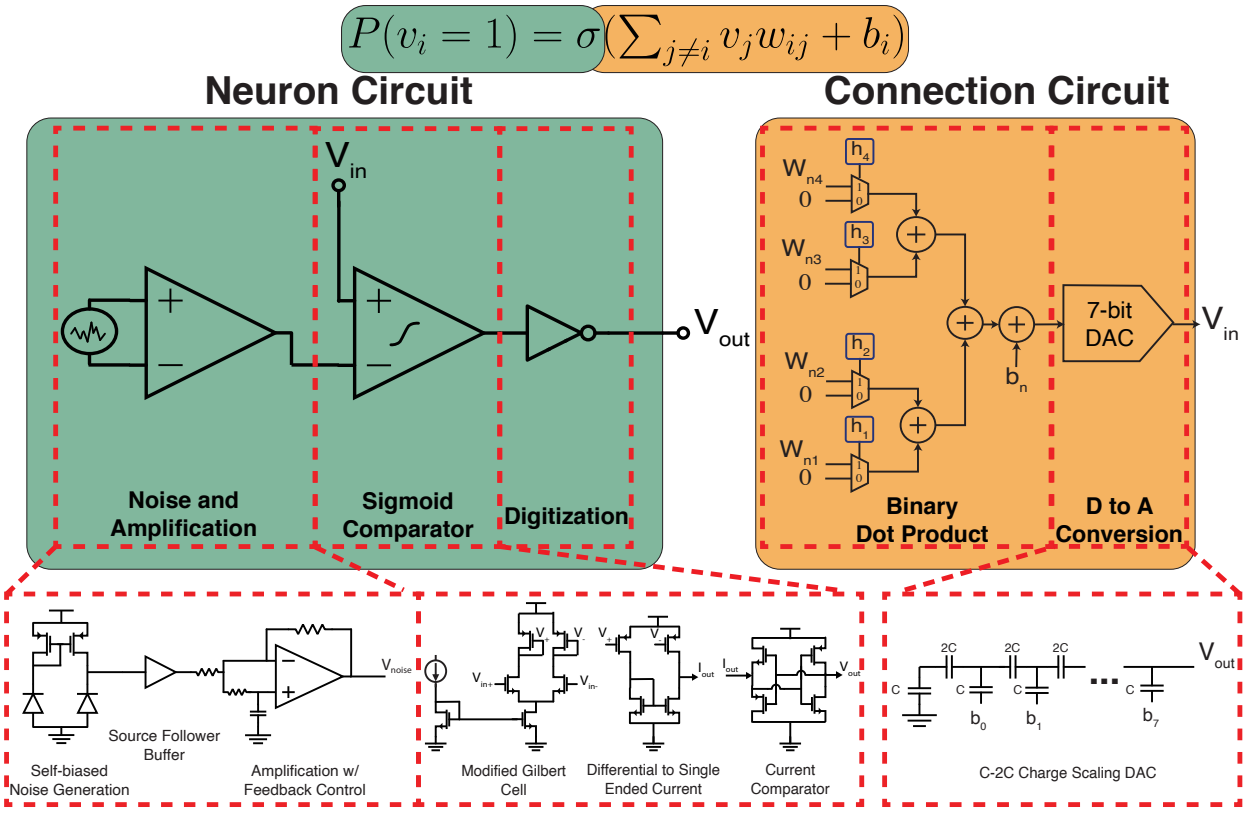

Figure 7.1: CMOS implementation of a PASS neuron. The neuron produces a random bit stream biased by the input voltage. Connections between neurons are implemented digitally

Noise Source

The noise source had to have the following characteristics:

- Independence between neurons (uncorrelated noise sources)
- Output Common Mode 500mV < Vcm < 700mV, allowing for efficient NMOS input stage
- Output Peak to Peak >5mV
- White noise spectrum
- Easy layout given process constraints

The independent neuron constraint was accomplished by having a self biased configuration, where a current mirror load was given to a tie down diode which created the noise

Parameter	Value
Output Common Mode Vout	643 mV
Peak to Peak Vout	8.9 mV
RMS Vout	1.32 mV

Table 7.1: Performance Parameters for self-biased noise source

source. To maximize the noise output, the noise diode sees a high impedance load through the drain of the PMOS transistor. The combination of small devices and high impedances (low overall current) meant that this system produces sufficient noise to be amplified by the amplification scheme.

The small size (minimal area diode) allowed for a large noise generation and minimal filtering capacitance. The one issue with this configuration was that the reverse biased diode did not provide exactly a white noise spectrum, which changes the autocorrelation of the final system. This can be attributed to the presence of both a transistor and diode, while the diode may provide a shot noise spectrum, the transistors noise spectrum is a pink noise source.

Ultimately the biggest constraint on the noise generator design was anxiety over latchup and ease of layout. We were worried about having a p-sub that was floating, which meant we reverse biased the tie down diode, so that the substrate was universally grounded. The transistors were 2 fingered instead of 1 so that I could lay them out in a rational way, even though a 1 finger transistor would have had higher noise.

One of the major things that increased the noise in simulation was grounding the n-well for the PMOS transistors. This forward biases a lot of the junctions in the PMOS transistors, which may cause substrate leakage problems. Based on my analysis, We wouldn't have latchup issues as long as the substrate voltage is stable.

Original iterations of the design had a differential noise source (two copied versions of the above circuit), however, the variation from these was so high that they would saturate a high gain amplifier due to common mode differences between them.

Final characteristics for the noise source are in Table 7.1

Buffer Stage

The buffer is meant to provide a low input capacitance stage (to prevent filtering of the input noise) while also supplying the necessary low output impedance to drive the gain stage. The biggest issue with the buffer stage was meeting a low enough output impedance, while not reducing the gain to the point of being ineffective. The low output impedance would allow the buffer to not load the subsequent gain stage which relied on resistive feedback to create signal gain. To accomplish the specification of low output impedance, we chose to use a "super source follower" configuration shown below in Figure 7.2.

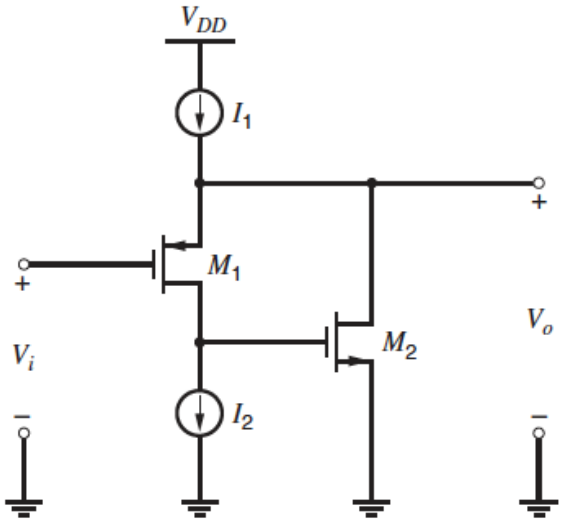

Figure 7.2: PMOS input Super Source Follower configuration copied from [91]

Parameter	Value
Input Capacitance	2.9 fF
Common Mode Vout	478 mV
RMS Vout	2.00 mV
Low Frequency R_{out}	155 Ω
Low Frequency Gain	-0.16 dB

Table 7.2: Performance Parameters for Super Source Follower Buffer Stage

Noise Amplifier

The noise amplifier is a 2 stage, single-ended amplifier with internal feedback whose goal is to provide a reasonable gain with low enough variation, such that the array of 256 neurons has switching behavior in all devices. We also needed a relatively high speed to demonstrate fast switching. The intended gain should be approximately 200, meaning the open loop amplifier gain should be ¿500 to reduce variation due to amplifier gain.

Due to the intricacies of the finfet process, the minimum channel length transistors have poor analog performance (specifically poor g_m and r_{out} compared to current consumption), so we were unable to achieve the gain specifications we were after. As we need to minimize variation in our noise amplifier as well, we chose to go with a slightly longer channel length transistor. The final performance characteristics of the amplifier are listed in Table 7.3 and

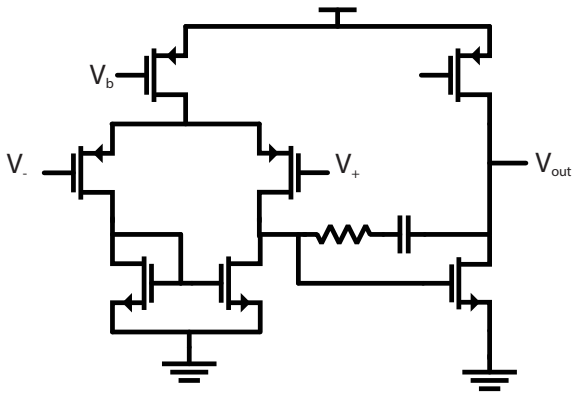

Figure 7.3: PMOS Input Operational Transconductance Amplifier (OTA) used for noise amplification after the buffer stage

Parameter	Value
Low Frequency Gain	251 (48 dB)
Corner Frequency	77 Mhz
Output Common Mode	416 mV

Table 7.3: Performance Parameters for OTA Noise Amplifier

the design is shown in Figure 7.3.

The main gain comes from the second stage due to bias conditions being difficult for the first stage. Specifically, it was hard to bias both the first stage gain transistors and the first stage bias transistors into deep saturation such that they had a large gain (first stage gain maxed out at 50). Similarly, to save on the power budget, the second stage transistors are (slightly) larger to achieve better gain performance. Overall the small size of the transistors meant that secondary techniques were necessary to get the amplifier process variation to 3 sigma performance levels and to have a high enough yield to be usable in an array format. These techniques are described below in Section 7.3.

Sigmoid Generation and Output digitization

After the amplified noise signal is generated by the OTA stage, the signal is sent into the sigmoid comparator. The sigmoid comparator is responsible for comparing the input noise signal to the signal coming from the synapses and DAC. It can be understood as doing the

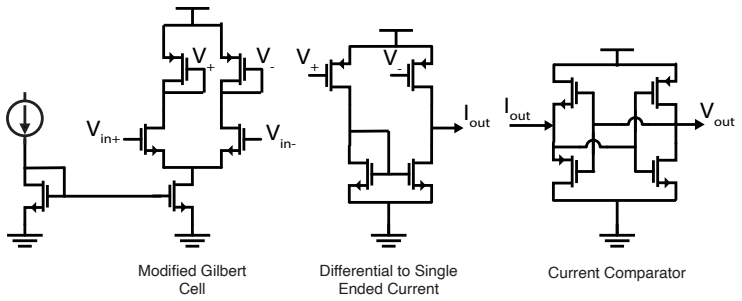

Figure 7.4: Diagram of the sigmoid design, divided into 3 parts, a modified gilbert cell, a differential to single ended current current source and a current comparator.

probabilistic activation function which can be seen as $\sigma(x) > R$ where R is the random signal coming from the synapse. The sigmoid circuit is divided into 3 parts shown in Figure 7.4.

The first part is a modified gilbert cell which formulates the activation function shape [91]. We note that a gilbert cells usually would either use BJT input transistors or have the input MOSFET transistors biased in subthreshold. In our case, our process flow did not offer BJT transistors with enough current per area to be area efficient, and biasing in sub-threshold meant that we would sacrifice too much speed for the sigmoid. We sacrificed a perfect sigmoid shape to have the speed needed for operation. In the end, with proper tuning, we found that the activation shape was sufficiently close to sigmoidal for our applications.

The second and third part of the circuit take the differential current output from the gilbert cell and convert it to a single binary output voltage. We use a current comparator composed of a bistable inverter first presented in [92], which gives a sufficiently fast output circuit. The final voltage output is passed through an inverter to ensure proper drive strength and to force the output to be purely binary.

Synapse Design

The neuron synapse is a multiply-accumulate operation, which is simplified by the use of binary activations with a MUX to perform multiplication. The neuron weights are stored in a distributed memory system addressed by a shift register running through the core. This allows the architecture to overcome problems arising from the von-Neumann bottleneck, as weights are stationary throughout computation. The synapse is digitally synthesized to mux and accumulate signals from adjacent neurons and pass the 7-bit answer into the DAC for conversion to an analog value. The synapse is a fully combinational logic cell, to ensure all computation is done in a clock-free manner.

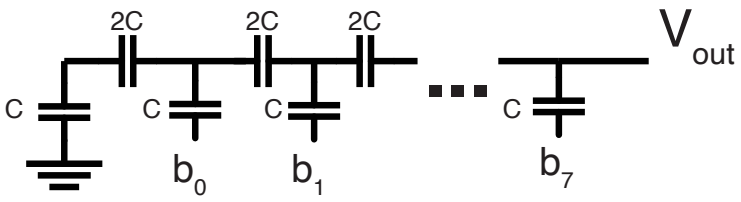

Figure 7.5: Diagram of the 7-bit C-2C DAC used in digital to analog voltage output

Digital to Analog Converter (DAC)

The Digital to Analog Converter (DAC) takes the digital accumulation from the synapse cell and converts it to an analog voltage which can then be used by the analog neuron cell and the sigmoid comparator. The DAC uses a C-2C topology which allows for area efficient conversion of digital to analog output values. The only problem with these C-2C topologies is charge leakage off of the output node as time goes on. We find that this means that there is a time window for computation to occur before it is invalid.

An additional problem with C-2C topologies is that parasitics induced by layout can cause the output to be non-monotonic when switching from LSB to MSB (i.e. 0111 to 1000 codes) configuration. To combat this, we were forced to increase the base capacitor size, and many of the area gains for the C-2C topology was washed out by this size increase.

The C-2C topology, and other capacitor based DAC topologies, have the added benefit of being very power efficient, with no leakage current, and power only used to charge the capacitors during switching. Along with this, the speed of the DAC is governed by the inverters driving the bit lines of the DAC, with switching speeds under < 10ps.

Controlling Process Variation

As with most analog designs, process variation is a major challenge when implementing an array of a large number of neurons. Even if each neuron has a 99% chance of working, with an array of 256 neurons the chance that all of the neurons are working is $\approx 7\%$. With an experimental design such as this even a 99% chance of working would be considered a stretch goal. Indeed, we would expect to just have a subset of the neurons working for any given chip run. Process Variation is more pronounced with analog sub-systems as we expect the output to be a precise value rather than a binary output. To combat design variation, we employed 5 major methods at different levels of the design hierarchy: Over-sized amplifiers for reduced transistor variation, Auto-Zeroing for offset cancellation, digital trimming of amplifier and neuron current, post-fabrication offset correction during programming, and modification of the supply voltage.

As is generally understood in analog systems, larger transistors have less pronounced

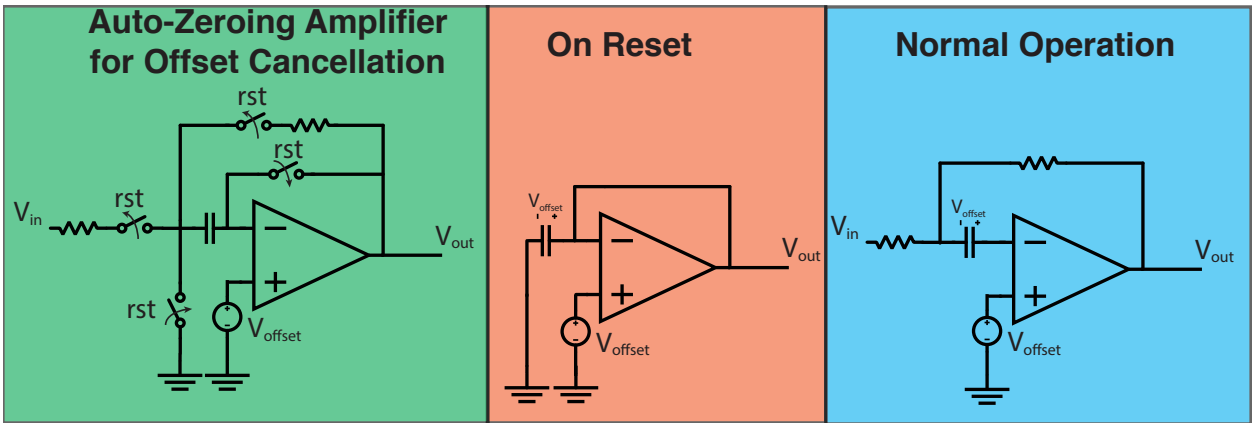

Figure 7.6: Autozeroing circuitry for the Noise Amplifier Circuit. Left: the full neuron circuitry, including switches which are flipped open or closed on reset. Middle: When the amplifier is reset, it is put into unity-gain to cancel out the effect of the input offset voltage. Right: On normal operation, the reset circuitry does not effect regular operation, and it is able to operate in regular negative feedback configuration.

transistor to transistor variation (and less variation across multiple dies) [91]. Defects present in larger transistors are less likely to destroy the entirety of the transistor performance and can be better managed. In the analog components whose variation needs to be well controlled, and is not controlled by other methods (such as the Noise Buffer and sigmoid), we have made efforts to make the system as large as possible to reduce neuron to neuron variation.

The performance parameters of the Noise Amplifier have the greatest effect on the overall neuron performance. The speed of the noise amplifier sets the speed of the overall neuron (along with the spectrum of the noise source), the amplitude of the noise amplifier sets the width and shape of the activation function, and the common mode output voltage of the amplifier sets the center of the sigmoidal activation. For this we have gone through great lengths to decrease the variation of this system. To offset any input offset voltage, we have implemented an Auto-Zeroing circuit [91, 93]. Before computation is conducted, a reset signal is broadcast to every neuron, which cancels any drift in the amplifier. The possible length of computation is partially set by the leakage off of the internal capacitor node of the auto-zeroing circuit, which causes some drift in the common mode output of the amplifier over the course of $\approx$ 1ms. The topology of this circuit is shown below in Figure 7.6.

To further reduce the effect of the Noise Amplifier variation, we have introduced a digital trimming circuit for both the noise amplifier and the the sigmoidal activation function. The digital trimming circuit is part of the configuration chain and modifies the input current to groups of 16 neurons together. The effect of the digital trim of current on the noise amplifier is to change the amplifier gain (which is itself controlled by the feedback resistors of the amplifier), and to control the speed of the amplifier. If variation of the amplifier causes its

gain to become too low, or the neuron to switch too slowly or too quickly, we are able to trim it using this method.

To reduce variation after fabrication we are able to extract activation data from each individual neuron and do a correction step before programming the neuron array with the problem of interest. This is done by a linear correction term, described in Equation 7.1 below. The sigmoid is individually characterized by incrementing the bias input code from -127 to +127 (the smallest and largest codes for the neuron). After this, a sigmoid with two linear correction parameters is applied to each individual neuron, which allows each neuron to have a different linear offset and slope of activation. The effect of implementing this correction is demonstrated in Figure 7.7, where we see that the average sigmoid variation is drastically decreased. Note that even with the correction, we see some neurons still behaving incorrectly (seen as the neurons which do not saturate to a probability of +1 on largest input). These neurons are "dead" and we should avoid programming them.

$$\sigma(x) = \frac{1}{1 + e^{-(a(x-b))}} \tag{7.1}$$

The last method of reducing variation in the chip is to modify the supply voltage of the analog core. This is somewhat of a non-intuitive result, but proves to be an important way of forcing the neurons to behave in a predictable way. When we decrease the supply voltage of the neuron, the noise amplifier is effected most by the change in behavior, this means that the gain and speed parameters of the amplifier start to degrade. When the gain degrades, the output sigmoid will become "narrower" due to it becoming easier for the output of the DAC to be larger than the output of the noise amplifier. In the case of this PASS system, the noise amplifier produces more noise than originally specified at nominal supply voltage of 0.8V,causing many of the neurons to never fully saturate to the +1 state. Thus, although the system is slower at 0.6V, all experiments for complex problems are conducted at this reduced supply voltage to maintain accuracy and predictability. These results are visually shown in Figure 7.8.

Sampler and I/O Design

The network is laid out in a regular grid with nearest-neighbor connections. To derive network statistics, we sample each neuron's binary output at a fixed clock frequency and stream out data off chip. We use a flip-flop synchronizer to ensure low probability of metastable states as we move from the asynchronous domain to the sampling clock domain.

The sampler and I/O is designed to be as simple and robust as possible to focus on design of the analog neuron core and to ensure the greatest chance of success of the system. The design of the system is composed of 3 components, as described below:

- Neuron State Sampler: This circuit samples the neuron's binary output at a rate of $>$ 100Mhz and shifts the output down the sample column to hold for eventual readout.

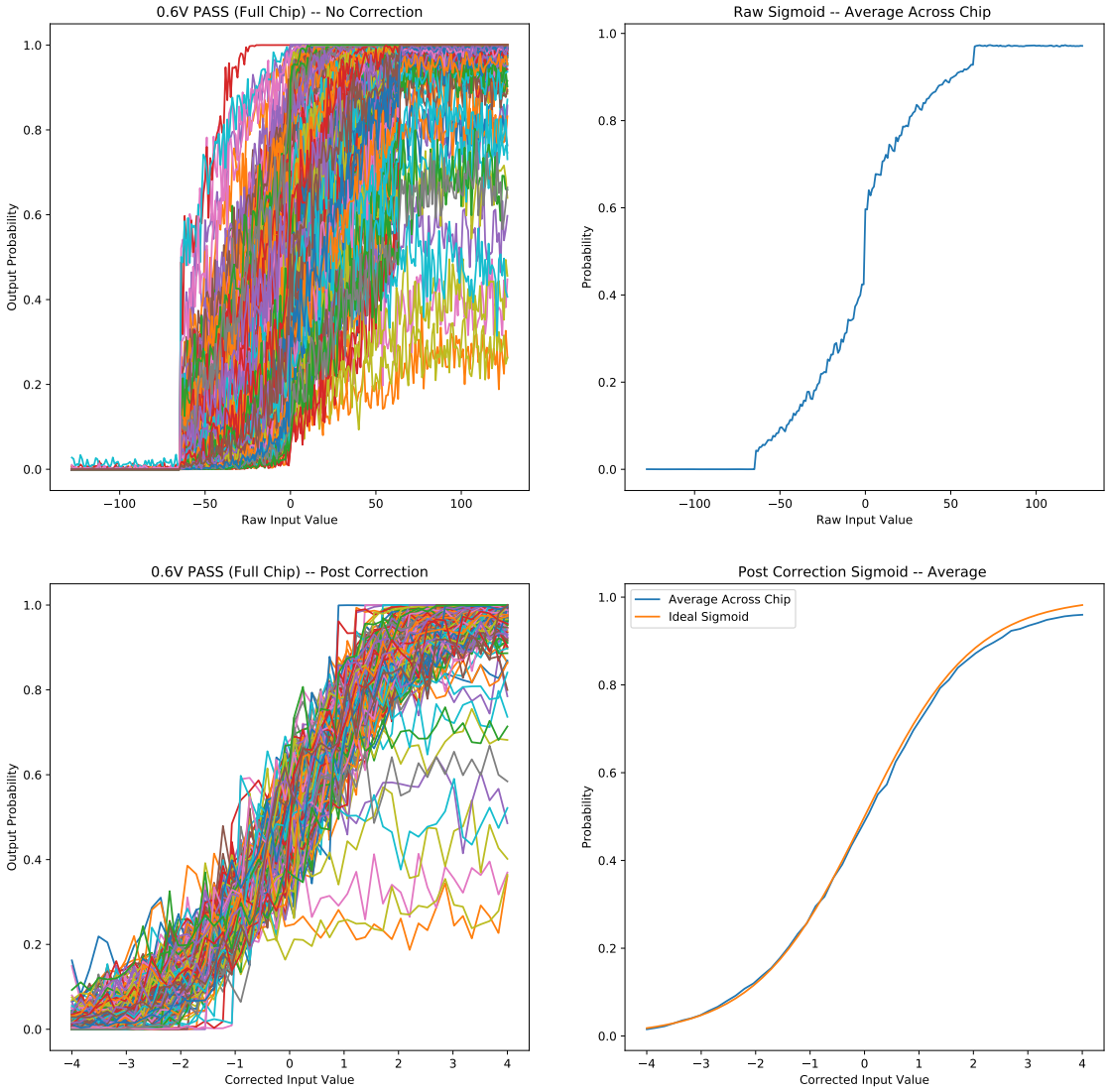

Figure 7.7: Correction of the Sigmoidal activation post-fabrication by fitting sigmoids to each neuron individually. Top Left: A plot of all neuron activations across a chip plotted on top of eachother. There is a lot of variation in activation between neurons. Top Right: The extracted average activation function across the chip. This activation is close to sigmoidal, but still requires fitting. Bottom Left: After doing a linear fit of each neuron activation, we see that the neurons have much less variation across the chip, and generally have a closer to ideal activation. Bottom Right: When averaged across the chip, the average activation function becomes almost exactly the ideal sigmoidal activation, which is plotted on top of it for additional information.

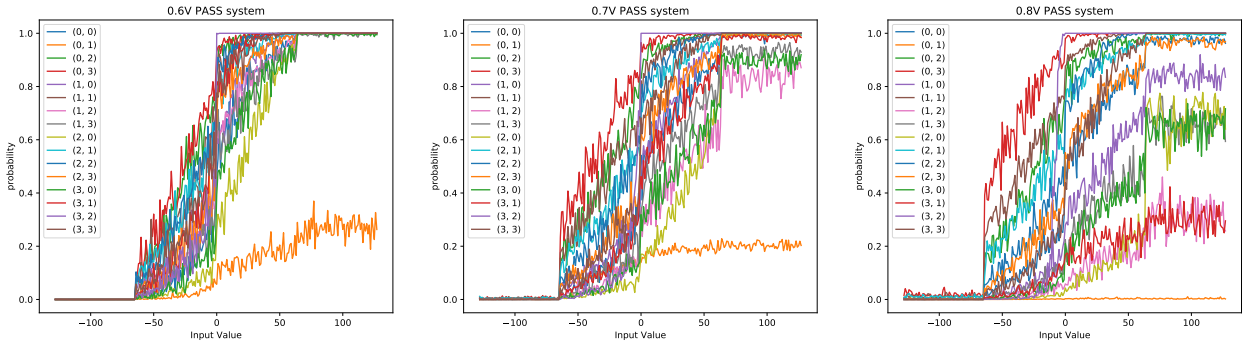

Figure 7.8: Analyzing neuron behavior as a function of supply voltage. Looking at the same neurons going from 0.6V to 0.7V to 0.8V we can see that as voltage increases, the variation amongst neurons increases as well. We can attribute this to the amplifier circuitry overpowering the sigmoid generation circuitry causing the neuron to never fully saturate to the +1 state.

- Chip Configuration: The configuration chain is responsible for setting the weights, biases, digital trims, and sampler settings for the full chip. It is implemented as a long shift chain.
- Data Readout: The readout circuitry takes the neuron states that are held in an SRAM buffer and sends them out at an I/O friendly speed of $\approx$ 10Mhz.

Neuron State Sampler

As the neuron states are binary output, no complex ADC circuitry is necessary to sample the output state of the system. Instead, a 3 register flip-flip synchronizer is used to ensure the circuitry does not enter a metastable state and to move from the asynchronous domain to the sampled clock domain. A 300Mhz clock is sent to each neuron, with the neuron state being sampled once every k cycles, where k is the number of rows we wish to sample at a given time. This means that each neuron would be sampled at a rate of $\frac{300}{k}$ Mhz, allowing us to sample a few rows at a very fast rate to get more accurate statistics, or to sample the full chip to get less granular statistics but over a larger chip area. This design decision helped relax timing requirements for movement of the sampled chip data off the chip. This k parameter is set in the configuration chain at set up time. Once the sample is taken, it is shifted along the sample column and finally held in the SRAM buffer.

The SRAM buffer for the Neuron State sampler is a 136kBit SRAM cell with 17 bit lines, which is able to hold 8192 row samples at a given time. The extra bit per row is meant to hold a "fingerprint" bit which allows us to synchronize the origin of each sample after samples are streamed to the computer for analysis. If we are sampling the full internal

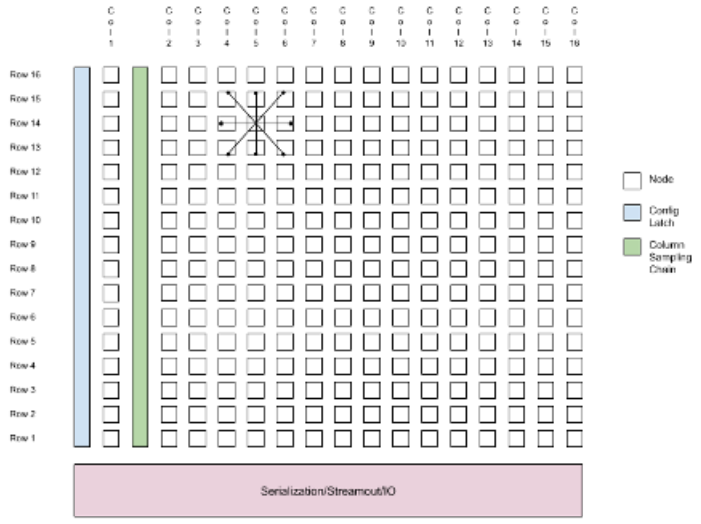

Figure 7.9: Layout of the chip in a 16×16 grid with sample and configuration columns in between each column of Neurons. Each Neuron is connected to its nearest neighbors and diagonals in a "King's Move" pattern

cluster, this means we are getting 512 samples from each individual neuron, or ≈ 440 us of data sampling the full cluster with a 300 Mhz clock.

Configuration Chain

The configuration chain holds all of the weights and biases, as well as sampling and trim information for the neuron core. Each neuron has 74 configuration bits, with 8 bit weights for each of the 8 neighboring neurons, 8 bits for the bias, and 2 bits to clamp the output to either 1 or to 0. Along with this, the configuration chain holds the 7 bit configurations for 16 current trimming circuits for the amplifiers, and 16 current trimming circuits for sigmoidal activation circuitry. The last piece of the configuration chain is a 3 bit sampling configuration register, which corresponds to how many rows of the chip core we wish to sample at a given time (the settings are 1 row, 2 rows, 4 rows, 8 rows, or 16 rows to sample at a given time).

The configuration chain is updated on a much slower 1 Mhz clock. The configuration chain is intentionally designed with a slower clock as the configuration data must snack across the entirety of the chip. This also allows for sufficient timing slack to optimize for other pieces of the system design.

Data Readout

Data from the SRAM buffer is read out at a slower I/O clock rate (20 Mhz) and shifted out in a Parallel-In-Serial-Out (PISO) fashion through a single General Purpose I/O (GPIO) pin. This design decision was done to minimize use of I/O pins when necessary, as it was deemed that I/O speed was not necessary for this proof of concept system. Additionally fewer I/O pins mean that more chip outputs can be dedicated to power, ground, and test outputs.

System integration

The system is finally integrated in a mixed-signal fashion to produce the final 16×16 array of neurons. An image of the post-layout chip is shown in Figure 7.10 where the integrated system is shown at a neuron level, a small scale integration level, and at the full chip level. We further are interested in the power, performance and area of the full chip system.

The 256 neuron core takes up a 1mm $\times$ 1mm in the center of the core, with the majority of the rest of the chip being fill area, and a small area devoted to I/O and the sampling circuitry. When analyzing the area of the chip, we find that the DAC has the highest area contribution within the cluster, while the binary dot product engine and analog neuron are smaller components. This is due to the necessity for large capacitors to combat layout parasitics.

When analyzing the power of the full system, the analog neuron takes the majority of the power, specifically the Noise Amplifier and Noise Buffer, both of which take $\approx$ 30uA of current when the full neuron taking a total of $\approx$ 80uA of current. We find that the digital accumulation of the system takes a much lower average current amount, as it does not run continuously, unlike the analog sub-systems.

As discussed previously, the speed of switching in the system is largely governed by the performance of the analog Noise Amplifier and noise generation system. To achieve faster performance, a higher bandwidth and higher gain amplifier would have to be designed, which in turn would cause a larger power draw. If we are willing to compromise on power of the system, we can decrease the bandwidth of the noise amplifier (by operating it in sub-threshold regime, for example) for next generations of design.

7.4 Results and Applications

The PASS chip is capable of solving a variety of problems that are in the NP-Hard and NP-Complete space. While many other Ising accelerators have demonstrated problem solutions in the optimization space, we choose to highlight the unique attributes of the PASS accelerator to solve problems in a larger variety of spaces than previously demonstrated accelerators. As discussed above, the PASS accelerator follows a sampling approach which gives full distributional information about the underlying distribution which allows us to understand the problem given at a deeper level. Below we highlight applications, results, and

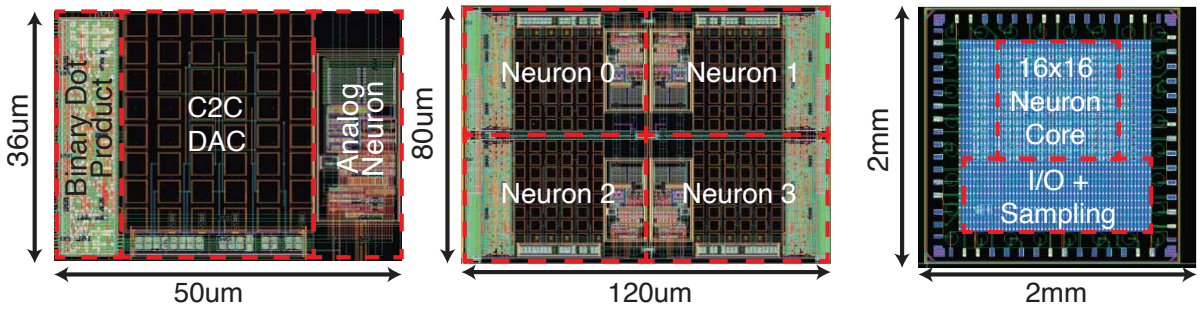

Figure 7.10: Post Layout images of PASS system. Left: Single Neuron showing the binary dot product, C2C DAC and analog neuron pieces. Middle: Image of four neurons connected together showing tiling of neurons. Right: Full chip micrograph showing core seperated from I/O and Sampling.

future work from four different areas, Combinatorial Optimization, Quantum Simulation, Neural Modeling, and Machine Learning.

Single Neuron and Small Cluster

Analysis of the system at small scale lets us understand the inner workings of the chip. In Figure 7.11 we demonstrate single neuron performance. In Figure 7.11 c) we can see that the activation function when averaged across many samples matches closely to that of a sigmoidal activation function. When looking at the time series data from this, we can see exactly how this works. When biased at low voltages (Figure 7.11 f) is at 0.1V) we can see that the neuron is uniformly at a low voltage. At middle voltages, shown at 0.45V in Fig 7.11 e), we can see the neuron oscillates stochastically between the 0 and 1 state, with an average activation of 0.4. Finally when biased at the upper end of the rail, at 0.55V in Figure 7.11 d), the system has a high average voltage, but we still get oscillations between the 1 and 0 state sporadically. When we analyze the switching behavior of the isolated neuron and fit an exponential decay function, we find that the autocorrelation of the isolated neuron at Vdd=0.8V can reach speeds of 150Mhz while pulling about $98\mu W$ of power. When compared to other stochastic architectures, we see that this system is the first with both high speeds and the ability to incorporate fully on-chip connections as shown in Table 7.4.

To understand performance in small clusters, we have implemented a 4 neuron test cluster which has visibility into the analog voltage outputs of the system (bypassing the digital sampling architecture). This allows us to see the raw voltage waveforms of switching performance and coupling between neurons. An illustrative example of running a Max-CUT problem on the four neuron cluster is shown in Figure 7.12. In Figure 7.12 a) we see the sampled probability distribution of the cluster favors the two maximum cut states, finding the solution to the underlying problem. When we analyze the time domain performance of

	PASS	[94]	[7]
Clock Period	6.6 ns	0.5 s	10 ms
Technology	14 nm CMOS	sMTJ	sMTJ
Connection Type	On Chip	Off Chip	Off Chip
Neuron Power	94 μW	$\approx$ 25 μW	$\approx$ 20μW

Table 7.4: Comparison of PASS chip with other stochastic neuron architectures.

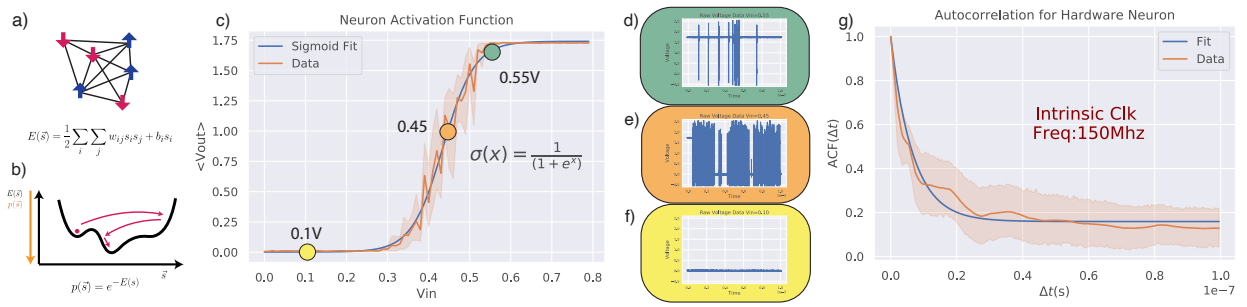

Figure 7.11: a) Ising Model Hamiltonians Energy function over binary variables. b) Probabilistic Sampling of PASS system c) Sigmoidal activation function from hardware neuron follows theoretical value. d), e), f) Time series for Vin=0.55V, Vin=0.45V, and Vin=0.1V, g) Autocorrelation demonstrating intrinsic clock of 150Mhz

the system in Figure 7.12 b), c), d) and e) we see that the system immediately identifies the two possible maximum cut states and stochastically oscillates between these two solutions on the order of us. This demonstrates how we expect the system to perform at a larger scale as well.

Combinatorial Optimization

Ising solvers have found many applications in the combinatorial optimization space, and it has remained an active area of research. In previous chapters we have shown how our FPGA, GPU, and TPU accelerators have the ability to solve Combinatorial Optimization problems, here we demonstrate that the PASS accelerator and its variants have the same ability to solve hard problems with potential for additional acceleration in time to solution and power consumption.

A demonstrative example of a difficult problem is shown in Figure 7.13 where we solve a MaxCUT problem, with the ground state solution spelling out the word CAL. This MaxCUT problem is constructed by having ferromagnetic (positive) coupling between regions of the same color and having opposing coupling in the transition regions between the letter and the

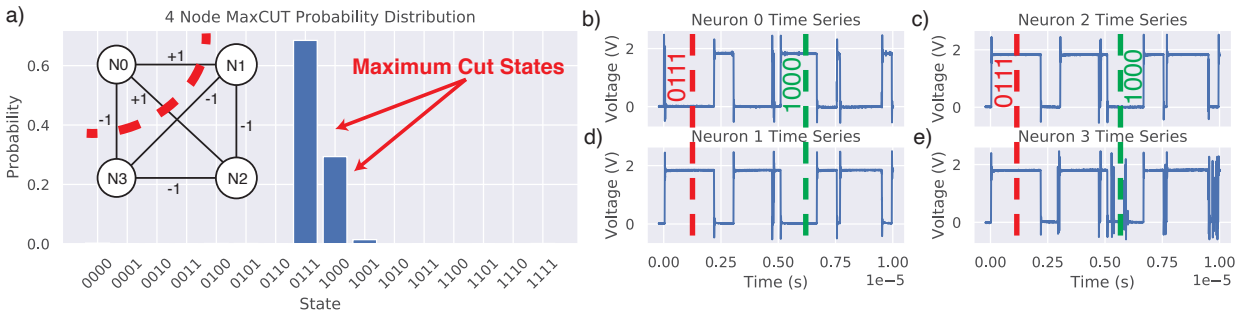

Figure 7.12: a) Probability Distribution of MaxCUT problem, sampled from Time Series b),c),d),e) Time Series showing fluctuations between correct states in µs timescale

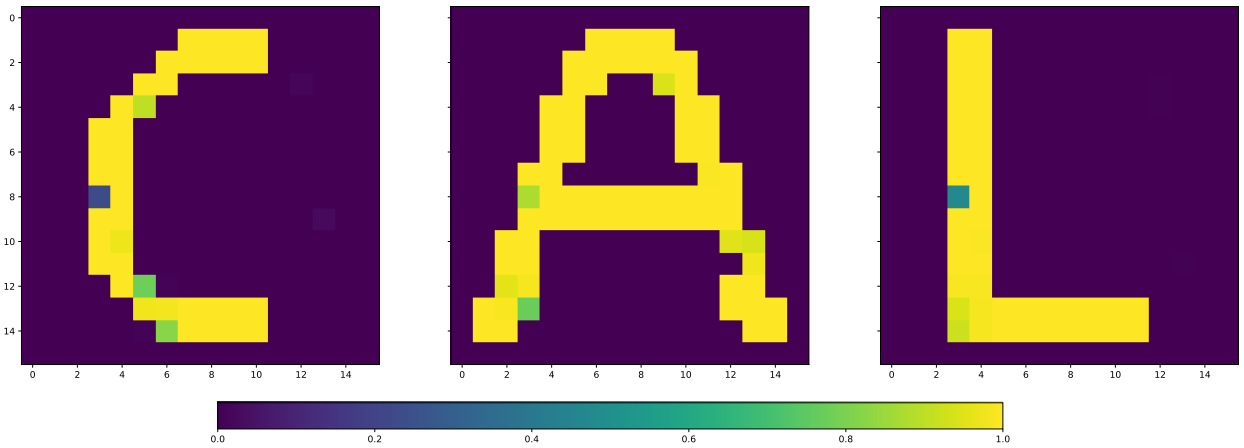

Figure 7.13: Demonstration of the PASS chip solving a Max-CUT problem whose ground state is to spell out the letters C, A, and L. This demonstrates the ability for the full 16×16 array to be connected together and to be programmed to solve such hard problems.

background. This demonstrates the ability of the full neuron core to act together to solve a problem, and exhibits how local couplings between neurons can interact together to create a global solution to a difficult problem at hand.

To understand how the PASS architecture compares to other state of the art architectures for optimization on these Ising Model architectures, we use a simulation model to project how the system would work on larger problem sets. We first extract activations and fit exponential decay curves to the autocorrelation curves from the hardware and then fit them to the Continuous Time Poisson Process model [26]. This models the neurons as each having an individual poisson clock, which updates at random times that are modulated by the activation probabilities. This is a similar system model to the Gaussian Machines

introduced in [95].

With the simulation model, we compare a discrete time process with the same effective clock frequency (the same average update time) to the stochastic time process presented with the PASS architecture. When we analyze this system, we see a scaling advantage emerge on the MaxCUT problem as demonstrated in 7.14, where a 200× improvement emerges at sizes of up to 150 nodes. We also see that the gap increases with larger problem sizes. From a theoretical perspective, [26] has proven Equation 7.2 below demonstrating a linear speed increase of mixing time with problem size when comparing discrete time mixing τ_{mix} with continuous time mixing τ^{C}_{mix}. From an intuitive perspective this can be understood as the spins being able to update completely independently in parallel, increasing the average update rate by a factor of n where n is the number of neurons.

$$\tau_{mix} \geq \frac{n\tau^{C}_{mix}}{6} \tag{7.2}$$

The connectivity of the current generation of the PASS chip is not sufficient to put these large fully connected problems onto the chip, forcing us to resort to these simulation frameworks. When we compare the simulated MaxCut performance to existing solutions to the MaxCUT problem we find that a stochastic solver with the same performance parameters as the PASS system outperforms all known solvers. Our solution would be able to solve the 100 node MaxCUT problem within 16us which is approximately 2× as fast as the existing solutions to solve the problem, such as the RBM on FPGA sampling solution presented in Chapter 5. This motivates further study into these stochastic systems to increase connectivity as well as to increase the speed of computation to realize these simulated performance gains.

Primitive Decision Making in Animal Brains

The Ising model and Hopfield Network have been tools in neurobiology for many years to model the behavior of firing neurons within the brain [96]. More recently an Ising type model was used to model animal decision making in primitive animal brains [97]. In this system, flies and other primitive animals were placed in a virtual reality environment, and given a set of targets. It was found that the animals would move to the average of the targets, and then spontaneously bifurcate and make a "decision" about the target they would proceed to.

The Ising model demonstrated ring attractor behavior that resembled the behavior of these animals when they were placed in virtual reality environments. With its underpinnings in neurobiology with relation to the Hopfield neural network [96, 12] the Ising Model was a good candidate for a potential system to model these decisions. The spin system represents neural activity within the brain, with the spin flips roughly representing firing of individual neurons. The spin model is repeated below in Equation 7.3

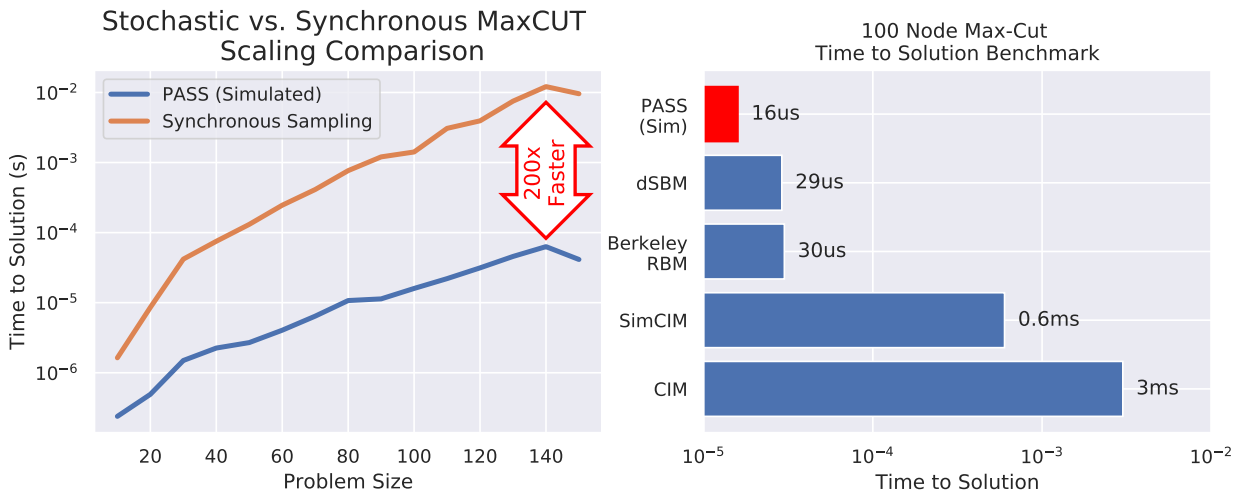

Figure 7.14: a) Scaling simulations of PASS system vs. Synchronous System showing 200x improvement, b) Performance Comparison of simulated PASS system (data taken from [[86], [89]])

$$H(s) = \frac{-k}{N} \sum_{i \neq j} J_{ij} s_i s_j \tag{7.3}$$

Here k is the number of options each individual has, N is the number of neurons, and J_{ij} is the interaction strength between neurons. The coupling parameters are set using a cosine geometry shown below in Equation 7.4.

$$J_{ij} = cos(\pi(\frac{|\theta_{ij}|}{\pi})^{\eta}) \tag{7.4}$$

In Equation 7.4, the θ_{ij} represents the angle between the goal vectors of spin i and spin j. The η parameter sets the "shape" that the animal uses to encode its space. Finally the velocity that the animal moves at is shown below in Equation 7.5. In this equation $\hat{\pi}_i$ represents the goal vector for each individual spin that points from the current position to that specific neurons target.

$$\vec{V} = \frac{v_0}{N} \sum \hat{p}_i s_i \tag{7.5}$$

At each step, the system performs "neuron updates" which are encoded as sampling updates within the chip. After a set of samples are taken, and the system settles into a local minima, the system moves forward with velocity $\vec{V}$ updating the goal vectors $\hat{p}_i$, the angles θ_{ij}, and the couplings J_{ij}. This update state then serves as the seed for the next set

of neuron updates. In this way the system can settle into a local attractor as it continuously samples from the state space after each update until it finally picks a direction to travel in.

The original neural models calls for the state at the end of each "neuron update" to serve as the seed state for the next update. However the PASS system does not have memory of previous system states between sampling runs. To combat this, we modify the Hamiltonian from above to include memory of the previous time step as shown in equation 7.6, where we add a bias term from the previous time step. Note that the value of s_i^{t-1} is a constant representing the value of that particular state in the previous sampling run, and is not a variable in this situation.

$$H(s^t) = \frac{-k}{N} \sum_{i \neq j} J_{ij} s_i^t s_j^t + \alpha \sum_i s_i^{t-1} s_i^t \tag{7.6}$$

With these modifications, we find that the PASS system is able to reproduce the geometry of decision making in fly and locust populations. In Figure 7.15 we demonstrate how the neural tuning parameter η moves the decision making point for the fly, and effects the individual geometry for that fly. On the right side of the figure, we overlay the $\eta = 1.0$ system on top of a heatmap generated by placing a fly in a virtual reality environment. We find that we are able to accurately reproduce the bifurcation point in decision making of actual flies. This demonstrates the ability of the PASS chip to model decision making in primitive animal populations, with the possibility of expanded connectivity in the chip to be able to model more complex decisions in animal brains.

Quantum Spin Systems

The basic building block of quantum computing systems is the q-bit, with complex interactions between them allowing for quantum computation to occur. The q-bit works by relying on complex phase interactions between q-bits which cannot be efficiently represented by classical systems of bits which only have values of 0 or 1. The probabilistic systems demonstrated here in the PASS system provide a bridge between the discrete systems of 0 or 1 and the complex phase space of the q-bit, where the PASS system is able to use probabilistic activations to represent states between 0 or 1. This suggests that there may be an efficient mapping for quantum spin systems onto the PASS architecture.

For classes of Hamiltonians which are known as "stoquastic" that are sign problem free, this has shown to be the case [98]. Specifically, the Suzuki-Trotter decomposition allows for decomposition of a d dimensional quantum hamiltonian to a $d+1$ dimensional classical spin hamiltonian. The PASS system generates statistics for the $d+1$ dimensional classical hamiltonian that can emulate the d dimensional hamiltonian.

For a proof of concept of these kinds of systems, we map the 1-dimensional Quantum Transverse Ising Problem onto the PASS system, creating a 2-dimensional Classical Hamiltonian which we can efficiently map onto our system. This is diagramatically shown in Figure 7.16 below, where the replicas are created in the second dimension for the 1 dimensional

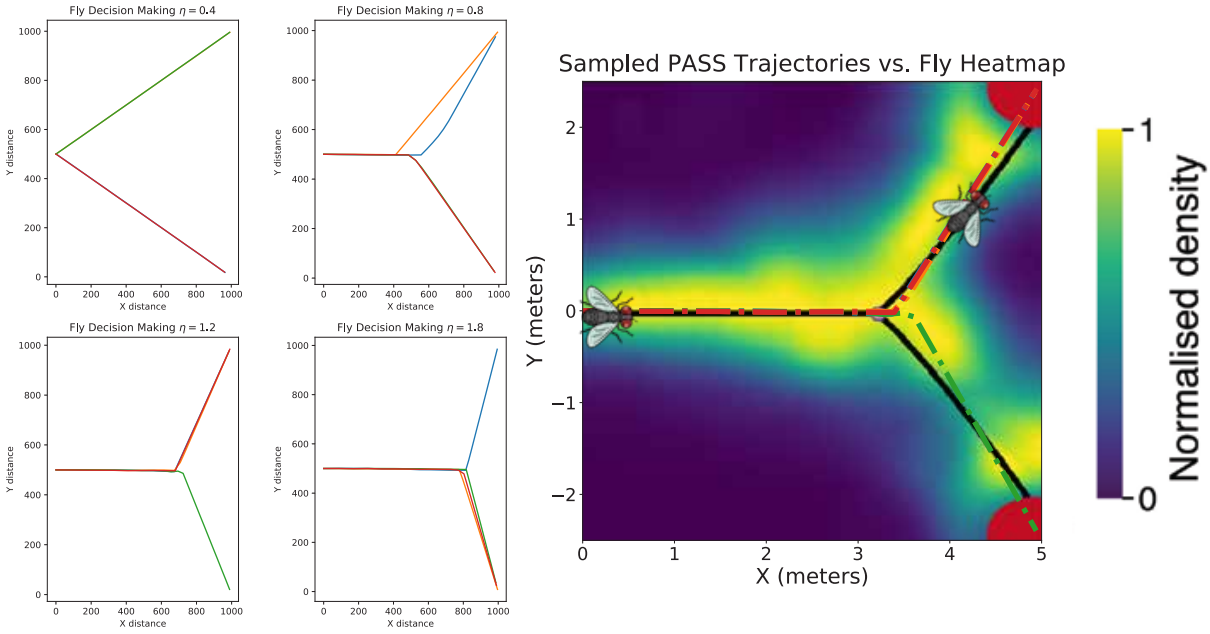

Figure 7.15: Neural Decision Making using the PASS system. Left: Showing how the neural tuning parameter η effects the geometry of the space that the fly operates in. As *eta* increases, the fly makes decisions closer to the targets. Targets are placed at {0, 1000} and {1000, 1000}. Right: When choosing the tuning parameter of $\eta = 1.0$ we see that the sampled trajectories from the PASS chip (the colored dotted lines) match closely with actual trajectories from flies placed into a virtual reality environment. The heatmap shows density for actual fly trajectories placed into a virtual reality environment with these two targets.

input problem. The Hamiltonian for the quantum system is shown in Equation 7.7 while the respective classical system is shown in Equation 7.8.

$$H_Q = -(\sum_i J_{i,i+1}\sigma_i^z\sigma_{i+1}^z + \Gamma_x \sum_i \sigma_i^x) \tag{7.7}$$

$$H_C = -(\lim_{n\to\infty} \sum_{k=1}^{n}\sum_{i=1}^{M}(J_{||})_{i,i+1}s_{i,k}s_{i+1,k} + J_\perp s_{i,k}s_{i,k+1}) \tag{7.8}$$

In the equations shown here $(J_{||})_{i,i+1} = J_{i,j}/n$, and $J_\perp = -\frac{1}{2\beta}\log\tanh(\beta\Gamma_x/n)$ with the states operating as bipolar spins $s_i \in \{-1, 1\}$ [99]. The parameter n represents how many replicas are being used to emulate the quantum spin system. One of the major features of this equation is the necessity for many replicas to accurately reproduce the behavior of the quantum spin system, with fully accurate representation only occuring for very large values

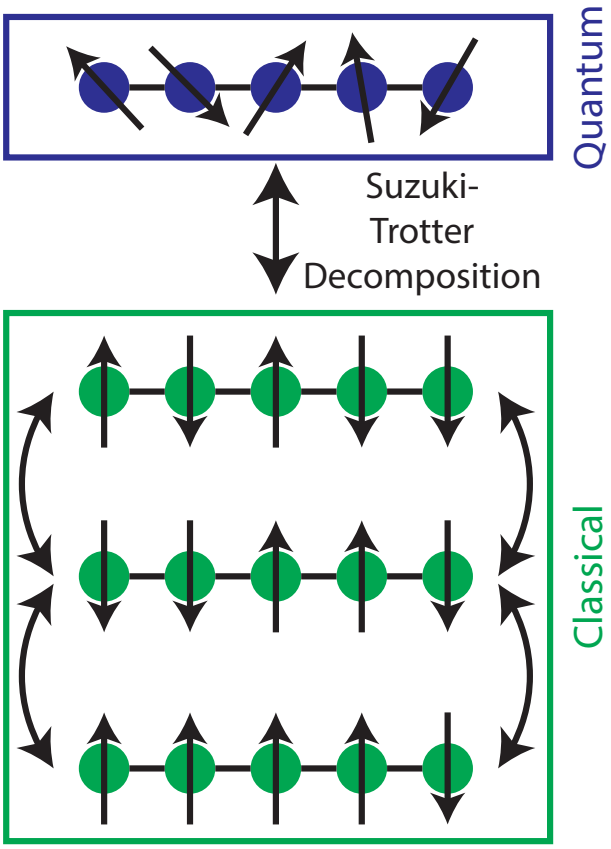

Figure 7.16: Suzuki-Trotter decomposition of a quantum spin system into parallel replicas of a classical spin system

of n. In experimental observations, we have found that for the spin chain shown here, we can accurately reproduce the magnetization curve with ≈ 10 spin chains.

To demonstrate the usability of the PASS system, we show curves for the 1-d Quantum Transverse Ising Problem, and reproduce the magnetization as a function of the $\Gamma_x = h_x$ parameter. This is shown in the left panel of Figure 7.17. We also demonstrate that the sampled wavefunction matches the wavefunction for the underlying system in the right panels of Figure 7.17.

Although the emulated quantum system here are relatively simple, greater connectivity in future PASS systems would allow for emulation of more complex quantum spin systems, such as the 2-dimensional lattices which have become a benchmark problem for both classical spin computers and quantum spin computers [100, 101]. Additionally more intelligent mappings would also have the ability to use the current PASS system.

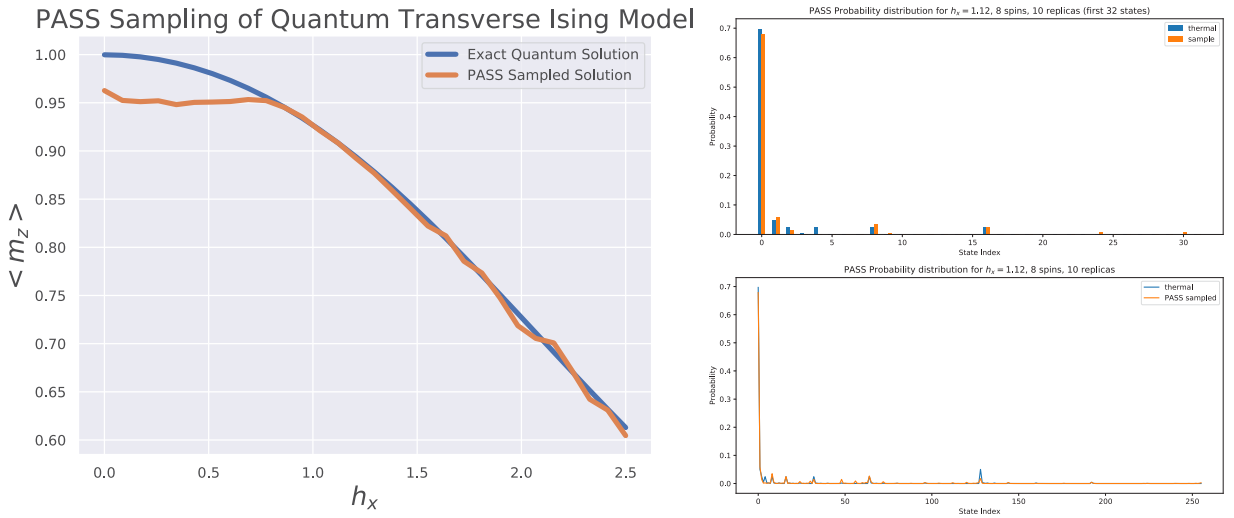

Figure 7.17: Demonstration of the PASS system to emulate the Quantum Transverse Ising Model. Left: Matching the average magnetization vs. transverse field curve. Right: For a particular point along the curve we show that we can emulate the exact thermal ground state with the wavefunction generated by the PASS sampled system.

Multiplier-Free Generative Machine Learning

The final application space for the PASS accelerator we would like to highlight is the usage for PASS, and PASS-type systems to perform multiplier-free generative machine learning. The design of the PASS system takes advantage of the binary activations of neurons to not implement any true multiplication circuitry, as a multiplication by a binary number is just a MUX or AND gate operation. This means that for running inference and sampling operations in the PASS systems, there is no usage of multiplication operations.

The basis for the PASS system is the Boltzmann Machine framework, as explored and explained in previous chapters. The Boltzmann Machine framework is a generative model, where the full probability distrubition of both input and outputs is used to understand problems. One of the goals of the PASS system is to accelerate sampling from difficult, complex, and correlated distributions, which would have a profound impact on the ability to perform machine learning tasks. By connecting the PASS system to an FPGA which would be able to calculate efficient gradients from the sampling distributions and the input weights, we would expect the PASS system to be used as a training accelerator too, using similar contrastive divergence algorithms explained previously.

An interesting piece of the contrastive divergence algorithm used here is that the contrastive divergence algorithm simply relies on averages, additions and subtractions shown below in Equation 7.9. Specifically, the $s_i s_j$ terms are binary AND gates (as both are binary activations), and the averaging across the expectation can be accomplished simply by bit-

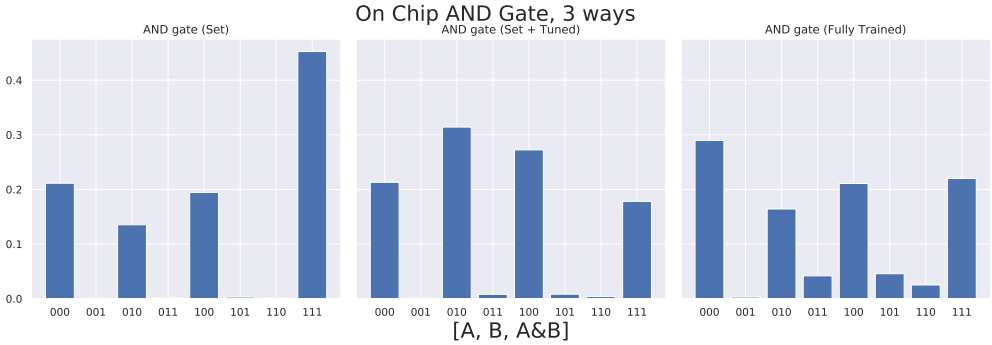

Figure 7.18: Probability distribution for an AND gate, created by a) direct setting of weights b) Setting weights and tuning using an ML algorithm, c) Fully trained distribution using ML

shifting and adding. The conclusion of this means that it is possible to develop the training and inference on the PASS system entirely without using any true multiplier blocks.

$$\Delta w_i j = \alpha(< s_i s_j >_{data} - < s_i s_j >_{model}) \tag{7.9}$$

The benefits of removing multiplications in future generations of Machine Learning algorithms and systems would be great. The majority of compute area and power for many machine learning systems is devoted to multiplying gradients for complicated back-propagation through many deep learning layers and for forward inference as well [57, 102]. The energy cost of performing a multiplication is approximately 10× higher than the cost of doing an addition (for Int8 precision) suggesting, that removing the multiplication step would have a large impact on the energy to train and do inference. For edge devices, which are significantly area and power constrained, this kind of multiplier-free training and inference could allow for continuous learning devices on the edge without draining of power resources.

To demonstrate small applications of this multiplifer-free version of this machine learning algorithm, we do training of an AND gate distribution in Figure 7.18. The left side of the figure shows setting the AND gate distribution, the middle shows setting the distribution and then training it to tune the distribution, and the right panel shows fully training the AND gate distribution without using any multiplication steps. This serves as a small proof of concept of this structure.

To show a possible application of such systems, we have implemented a factorization problem using the inverse logic approaches expanded upon in Chapter 3 in SPICE. The system is able to find factors of 143 in under 10us using SPICE simulations in the slower and less efficient 90nm process. We hope to expand upon this previous result to the PASS system to show the ability to do complex machine learning problems within the 14nm PASS chip.

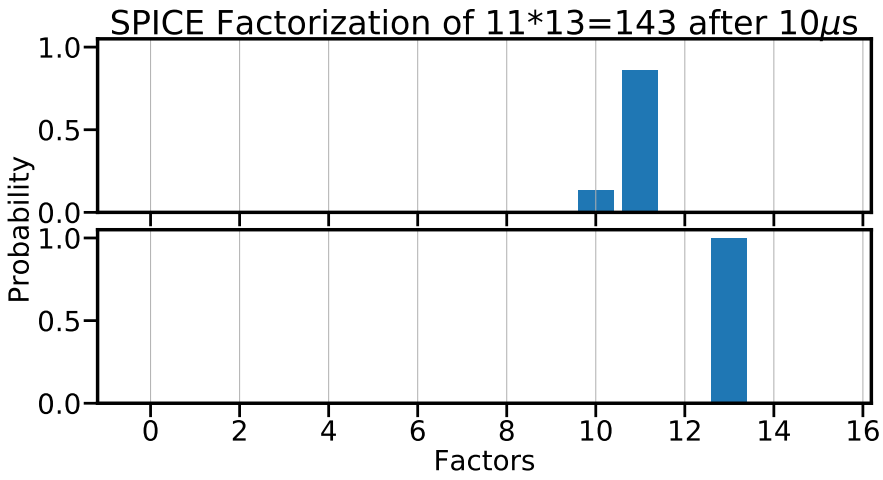

Figure 7.19: Factorization of an 8 bit number into two 4 bit factors through SPICE simulation of a 90nm type PASS system. As the multiplier calculates the joint probability of factors, the top panel shows the first factor chosen (correctly as 11), and the bottom panel shows the second factor to match the first factor (correctly chosen as 13). Figure shown after sampling for 10us

7.5 Conclusion

In this chapter we have demonstrated the theory, implementation, and applications of the PASS accelerator architecture. The demonstrated chip is the first of stochastic chip with fully on-chip connections, and the first stochastic chip that is able to scale to a usable number of nodes. In the implementation and development of this chip, we have innovated a series of methods to integrate stochastic chips with traditional digital hardware, as well as methods of characterizing and correcting for errors within the chip. Finally through this work, we have shown a series of applications that this chip and architecture would be able to provide value in. These applications are unique to the PASS stochastic architecture, and provide extra functionality over previous Ising Accelerators.

Chapter 8

Conclusion and Future Work

8.1 Chapter Summaries and Takeaways

In Chapter 1 we explained the historical context for Ising Model Computing, and what the current needs are for accelerators in the space of Optimization, Brain-Inspired Computing, and Machine Learning. Along with this we provided a taxonomy of various accelerators that have been built for these systems, positioning our probabilistic accelerators in reference to the current state of the art. We argued that probabilistic accelerators would have the ability to gain further information about the distributions of interest and use the rich mathematics of probability theory and statistics to achieve better solutions than previously demonstrated.

In Chapter 2, we introduced the theory of sampling from complex distributions with a particular application in sampling from the Ising Model distribution. We introduced the theory of Markov Chain Monte Carlo, as well as a set of MCMC algorithms which we would expand upon in later chapters. We explored the pros and cons of various MCMC methods, and explained our choices of using certain methods (such as the Blocked Gibbs Sampling and Parallel Tempering) over other methods. In this chapter we introduced the Restricted Boltzmann Machine and explained why we chose to use it as the basis for our accelerator architectures in the following chapters.

In Chapter 3, we identified the first new method of computation on these parallel MCMC systems in an inverse logic framework. We combined machine learning with logic design to show the possibility of factoring numbers and performing SAT problems as a basis for solving a variety of other Combinatorial Optimization Problems. We also expanded upon the theory of these methods, proving correctness of the distribution as well as convergence properties of our merging procedures.

In Chapter 4, the direct mapping approach of Ising Model problems onto the Restricted Boltzmann Machine architecture was explored. This method allowed general Ising Model problems to be mapped onto the RBM accelerator. We also explored the effect of various parameter choices on sampler performance, such as temperature, coupling, and number of samples taken. This provides a framework for mapping many further Ising Model problems

onto these architectures.

Through Chapter 5, we demonstrated our FPGA hardware platform for accelerated solutions of Ising Model problems. We first demonstrated a small FPGA accelerator which shows state of the art performance on small problem instances and is able to solve inference problems in the Machine Learning space as well optimization problems from the Ising Model space. The second generation of the RBM on FPGA accelerator was able to scale up to the largest problem sizes for a single FPGA accelerator at 5000+ nodes. After further investigation, we found that the larger RBM on FPGA accelerator requires better algorithmic performance to reach the ground state for problems of 1000s of nodes.

Building on the hardware in the previous chapter, Chapter 6 demonstrates GPU and TPU based exploration into parallel hardware. We first explore why GPU and TPU algorithms are necessary, and how to exploit the parallelism that these platforms provide. We then implement Parallel Tempering on the GPU and TPU to show that we are able to reach the ground state of increasingly large and difficult problems on the order of 1000s of nodes. We then show the true scalability of our algorithms and methods by scaling all the way up to 100,000 nodes on a single GPU instance, showing both the largest fully connected Ising Machine built so far, while also using the fewest number of compute resources to accomplish this.

Finally Chapter 7 demonstrates our Parallel Asynchronous Stochastic Sampler architecture. This architecture is a next generation computation platform with the ability to solve problems in a variety of spaces. We tackle problems of designing stochastic neurons, integrating them into a full system, and interfacing that compute core with the outside world. This architecture is the first stochastic computing architecture that is able to solve all of these problems together and create a full system on chip. We demonstrate a series of new applications on this platform which have not been shown on previous Ising Computing platforms.

8.2 Future Work

In the previous sections we have described all the work that has gone into building accelerated systems for the solution of Ising Model problems. However there is a lot of future work to be done to make these systems usable for real world problems. Here I try to explain what next steps may be necessary to advance Ising Computing further.

Improved Large Scale FPGA Architecture

The results from our FPGA architecture show that large Ising Model systems can be feasibly mapped onto the RBM based FPGA architecture. However, these systems do not take advantage of advanced sampling methods, such as the Parallel Tempering approaches demonstrated in Chapter 5. Because the FPGA was only able to use vanilla Gibbs sampling,

we found that it was unable to find the ground state to many larger problems (such as those above 1000 nodes).

To have an FPGA based accelerator that is capable of reaching the ground state of difficult problems, we propose re-architecting the FPGA accelerator to support an advanced sampling algorithm such as Parallel Tempering, Adaptive Parallel Tempering, or Non-Equillibrium Monte Carlo [103, 104]. The basis of this would involve the usage of parallel chains which have the ability to exchange information and states between different replicas. By including these advanced sampling techniques we would expect the FPGA system to achieve ground state solutions, similar to what was shown in the Parallel Tempering implementation demonstrated in Chapter 6, but with a faster time to solution than has been previously achieved by these architectures.

Efficient Algorithms to Map Real World Problems

Many of the problems demonstrated in this thesis are purely academic, and do not include the complex constraints necessary to model real world problems. The Max-CUT problem has applications in many fields, but needs further constraints to be able to map problems in other fields. Our understanding is that to go from the academic instance of solving a generic graph problem to a real world instance is to begin to incorporate real-world constraints into the system parameters. This is one of the biggest obstacles to bringing Ising Computing to the real world.

Building off of this, we have found that many of the algorithms developed for Ising Computing (for example many of those listed in [3]) can map onto the Ising Solver, but can be inefficient to do so. The Travelling Salesman Problem is a good example of this, where many heuristic solutions exist that can solve this problem much more efficiently than the standard Ising formulation is able to do. With this in mind, further work needs to be done to break down these problems into parts that can be efficiently consumed by an Ising Solver. This may be through inverse logic approaches to map certain subproblems, or through direct mapping approaches where we mathematically break the problem down into efficient sub parts.

Multiplier-Free Machine Learning

Both the RBM architecture and PASS architecture have shown the ability to perform inference on complex machine learning systems without the usage of multiplication steps. This greatly reduces the hardware cost of these architectures, especially for edge platforms where power and area are at a premium. We have demonstrated that these architectures may be useful for edge intelligence, solving complex routing problems as well as running machine learning inference.

An extension of these systems is to explore the use of these devices for multiplier-free training in machine learning algorithms. This would allow edge devices to have an added degree of intelligence, and the ability to train on new data in a power and area efficient

manner. A small proof of concept of these ideas was shown in Chapter 7 with the PASS architecture, but this should be expanded upon to show more complex use cases and the solution of more difficult problems.

Scaling up of Stochastic Hardware

The first PASS system demonstrated in this thesis serves as a proof of concept system for probabilistic and stochastic hardware. We have developed a series of methods to reduce the variability (both from chip to chip as well as across a given chip), which can be extended to create larger chips with more intricate interactions. The amplifiers and neuron design also consume a lot of power, while the DAC takes up a large amount of area. These systems can be optimized to produce a much lower power or a much higher performance second generation architecture for varying applications. Additionally this study showed us that the digital fabric is very extendable, suggesting that implementing even more connections per neuron (to next nearest neighbor, for example) should be possible without much greater effort.

The PASS system also relies on a fully clock-less computation, which may not be necessary to get good performance on problems of interest. There has been work recently showing that it can be possible to have a mixture of stochastic system and digital circuitry to achieve similar performance to the fully stochastic system here without using as much power, performance or area [100, 94]. These next generation designs can take lessons from the proof of concept systems generated here to create continued increases in performance.

8.3 Final Thoughts

When I started this work in 2017, stochastic hardware and Ising Machines was not nearly as mature as it is today. Since then, many groups have entered the space and a variety of hardware methods, algorithms, and knowledge has been created within this field. I believe that Ising Computing and in general accelerated optimization problems can have a profound impact in solving some of the hardest problems we have in the Post-Moore world. However, the next step which has still yet to be done, is to find a real world application where Ising Computing exhibits best in class performance. This “killer application” would truly be able to spark interest and development in this field. I hope that next generations of students and researchers can help expand the field and bring this research to the real world.

Bibliography

[1] F. Barahona. "On the computational complexity of ising spin glass models". In: *Journal of Physics A: Mathematical and General* 15.10 (1982), pp. 3241–3253. ISSN: 13616447. DOI: 10.1088/0305-4470/15/10/028.

[2] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by simulated annealing". In: *Science* 220.4598 (May 1983), pp. 671–680. ISSN: 00368075. DOI: 10.1126/science.220.4598.671. URL: https://www.sciencemag.org/lookup/doi/10.1126/science.220.4598.671%20http://science.sciencemag.org/content/220/4598/671.

[3] Andrew Lucas. "Ising formulations of many NP problems". English. In: *Frontiers in Physics* 2 (Feb. 2014), pp. 1–14. ISSN: 2296424X. DOI: 10.3389/fphy.2014.00005.

[4] Masanao Yamaoka et al. "A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing". In: *IEEE Journal of Solid-State Circuits* 51.1 (Jan. 2016), pp. 303–309. ISSN: 00189200. DOI: 10.1109/JSSC.2015.2498601.

[5] Takahiro Inagaki et al. "A coherent Ising machine for 2000-node optimization problems". In: *Science* 354.6312 (Nov. 2016), pp. 603–606. ISSN: 10959203. DOI: 10.1126/science.aah4243. URL: https://science.sciencemag.org/content/354/6312/603%20https://science.sciencemag.org/content/354/6312/603.abstract.

[6] Peter L. McMahon et al. "A fully programmable 100-spin coherent Ising machine with all-to-all connections". In: *Science* 354.6312 (Nov. 2016), pp. 614–617. DOI: 10.1126/science.aah5178. URL: http://www.ncbi.nlm.nih.gov/pubmed/27811274.

[7] William A. Borders et al. "Integer factorization using stochastic magnetic tunnel junctions". In: *Nature* 573.7774 (Sept. 2019), pp. 390–393. ISSN: 14764687. DOI: 10.1038/s41586-019-1557-9. URL: http://www.nature.com/articles/s41586-019-1557-9.

[8] M. W. Johnson et al. "Quantum annealing with manufactured spins". en. In: *Nature* 473.7346 (May 2011), pp. 194–198. ISSN: 00280836. DOI: 10.1038/nature10012. URL: https://www.nature.com/nature/journal/v473/n7346/full/nature10012.html.

[9] Satya Pal Singh. "The Ising Model: Brief Introduction and Its Application". In: *Solid State Physics* (Feb. 2020). DOI: 10.5772/INTECHOPEN.90875. URL: https://www.intechopen.com/chapters/71210%20undefined/chapters/71210.

[10] Lars Onsager. "Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition". In: *Physical Review* 65.3-4 (Feb. 1944), p. 117. ISSN: 0031899X. DOI: 10.1103/PhysRev.65.117. URL: https://journals.aps.org/pr/abstract/10.1103/PhysRev.65.117.

[11] GE Hinton, TJ Sejnowski - Proceedings of the IEEE conference on, and undefined 1983. "Optimal perceptual inference". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* 448 (1983), pp. 448–453. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b89e9f0cef5ace08946a7c07bf7

[12] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. "A learning algorithm for boltzmann machines". In: *Cognitive Science* 9.1 (Jan. 1985), pp. 147–169. ISSN: 03640213. DOI: 10.1016/S0364-0213(85)80012-4.

[13] Geoffrey E. Hinton. "Training products of experts by minimizing contrastive divergence". en. In: *Neural Computation* 14.8 (Aug. 2002), pp. 1771–1800. ISSN: 08997667. DOI: 10.1162/089976602760128018. URL: https://www.mitpressjournals.org/doi/abs/10.1162/089976602760128018.

[14] Fred Glover, Gary Kochenberger, and Yu Du. "A Tutorial on Formulating and Using QUBO Models". In: (Nov. 2018). URL: https://arxiv.org/abs/1811.11538v6.

[15] Jan H.M. Korst and Emile H.L. Aarts. "Combinatorial optimization on a Boltzmann machine". In: *Journal of Parallel and Distributed Computing* 6.2 (Apr. 1989), pp. 331–357. ISSN: 07437315. DOI: 10.1016/0743-7315(89)90064-6.

[16] Emile H.L. Aarts and Jan H.M. Korst. "Boltzmann machines and their applications". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 258 LNCS. Springer Verlag, 1987, pp. 34–50. ISBN: 9783540179436. DOI: 10.1007/3-540-17943-7{_}119.

[17] D.H. Wolpert and W.G. Macready. "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 67–82. ISSN: 1089778X. DOI: 10.1109/4235.585893.

[18] Alexandre Blais and Alexandre M. Zagoskin. "Operation of universal gates in a solid-state quantum computer based on clean Josephson junctions between d-wave superconductors". In: *Physical Review A - Atomic, Molecular, and Optical Physics* 61.4 (2000), p. 4. ISSN: 10941622. DOI: 10.1103/PhysRevA.61.042308.

[19] Maliheh Aramon et al. "Physics-inspired optimization for quadratic unconstrained problems using a digital Annealer". In: *Frontiers in Physics* 7.APR (2019). ISSN: 2296424X. DOI: 10.3389/fphy.2019.00048.

[20] Tianshi Wang and Jaijeet Roychowdhury. "Oscillator-based Ising Machine". In: (Sept. 2017). URL: https://arxiv.org/abs/1709.08102v2.

[21] Ibrahim Ahmed, Po-Wei Chiu, and Chris H. Kim. "A Probabilistic Self-Annealing Compute Fabric Based on 560 Hexagonally Coupled Ring Oscillators for Solving Combinatorial Optimization Problems". In: *2020 IEEE Symposium on VLSI Circuits*. IEEE, June 2020, pp. 1–2. ISBN: 978-1-7281-9942-9. DOI: 10.1109/VLSICircuits18222.2020.9162869.

[22] Zhe Wang et al. "Coherent Ising machine based on degenerate optical parametric oscillators". In: *Physical Review A - Atomic, Molecular, and Optical Physics* 88.6 (Dec. 2013), p. 063853. ISSN: 10502947. DOI: 10.1103/PhysRevA.88.063853.

[23] Tianshi Wang. "Sub-harmonic Injection Locking in Metronomes". In: *arXiv* (Sept. 2017).

[24] S Dutta et al. "An Ising Hamiltonian solver based on coupled stochastic phase-transition nano-oscillators". In: *Nature Electronics* 4.7 (2021), pp. 502–512. DOI: 10.1038/s41928-021-00616-7. URL: https://doi.org/10.1038/s41928-021-00616-7.

[25] Mark Jerrum and Alistair Sinclair. "Polynomial-Time Approximation Algorithms for the Ising Model". In: *SIAM Journal on Computing* 22.5 (Oct. 1993), pp. 1087–1116. ISSN: 0097-5397. DOI: 10.1137/0222066.

[26] Thomas P Hayes and Alistair Sinclair. "A general lower bound for mixing of single-site dynamics on graphs". In: *The Annals of Applied Probability* 17.3 (2007), pp. 931–952. DOI: 10.1214/105051607000000104.

[27] Karl Bringmann and Konstantinos Panagiotou. "Efficient Sampling Methods for Discrete Distributions". In: *Algorithmica* 79.2 (Oct. 2017), pp. 484–508. ISSN: 0178-4617. DOI: 10.1007/s00453-016-0205-0.

[28] Alexander N. Gorban, Valery A. Makarov, and Ivan Y. Tyukin. "High-Dimensional Brain in a High-Dimensional World: Blessing of Dimensionality". In: *Entropy* 22.1 (Jan. 2020), p. 82. ISSN: 1099-4300. DOI: 10.3390/e22010082.

[29] Pierre. Brémaud. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. 2. Springer New York, 1999, pp. 207–221. ISBN: 9781441931313.

[30] Jaewook Lee, Woosuk Sung, and Joo-Ho Choi. "Metamodel for Efficient Estimation of Capacity-Fade Uncertainty in Li-Ion Batteries for Electric Vehicles". In: *Energies* 8.6 (June 2015), pp. 5538–5554. ISSN: 1996-1073. DOI: 10.3390/en8065538.

[31] G. O. Roberts and A. F.M. Smith. "Simple conditions for the convergence of the Gibbs sampler and Metropolis-Hastings algorithms". In: *Stochastic Processes and their Applications* 49.2 (Feb. 1994), pp. 207–216. ISSN: 03044149. DOI: 10.1016/0304-4149(94)90134-1. URL: http://www.sciencedirect.com/science/article/pii/0304414994901341.

[32] Asja Fischer. "Training Restricted Boltzmann Machines". PhD thesis. University of Copenhagen, Denmark, 2014, pp. 441–444. ISBN: 1321801503712. DOI: 10.1007/s13218-015-0371-2. URL: https://www.researchgate.net/publication/273119734.

[33] Asja Fischer and Christian Igel. "An Introduction to Restricted Boltzmann Machines". In: 2012, pp. 14–36. DOI: 10.1007/978-3-642-33275-3{_}2.

[34] Pierre Brémaud. "Gibbs Fields and Monte Carlo Simulation". In: *Markov Chains*. New York, NY: Springer New York, 1999, pp. 253–322. DOI: 10.1007/978-1-4757-3124-8{_}7. URL: http://link.springer.com/10.1007/978-1-4757-3124-8_7.

[35] Abraham P. Punnen. "The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications". In: *The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications* (Jan. 2022), pp. 1–319. DOI: 10.1007/978-3-031-04520-2/COVER.

[36] Guillaume Desjardins et al. "Parallel tempering for training of restricted Boltzmann Machines". In: *Journal of Machine Learning Research* 9 (2010), pp. 145–152. ISSN: 15324435.

[37] Tijmen Tieleman. "Training restricted boltzmann machines using approximations to the likelihood gradient". In: *Proceedings of the 25th International Conference on Machine Learning*. ACM, 2008, pp. 1064–1071. ISBN: 9781605582054. DOI: 10.1145/1390156.1390290.

[38] Tijmen Tieleman and Geoffrey Hinton. "Using fast weights to improve persistent contrastive divergence". en. In: *ACM International Conference Proceeding Series*. Vol. 382. ACM Press, 2009, pp. 1033–1040. ISBN: 9781605585161. DOI: 10.1145/1553374.1553506. URL: https://dl.acm.org/citation.cfm?id=1553506%20http://portal.acm.org/citation.cfm?doid=1553374.1553506.

[39] Stephen A. Cook and Stephen A. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. New York, New York, USA: ACM Press, 1971, pp. 151–158. DOI: 10.1145/800157.805047. URL: http://portal.acm.org/citation.cfm?doid=800157.805047.

[40] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. Springer US, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2{_}9.

[41] Ilya Sutskever and Tijmen Tieleman. "On the convergence properties of Contrastive Divergence". en. In: *Journal of Machine Learning Research* 9 (2010), pp. 789–795. ISSN: 15324435.

[42] Kerem Yunus Camsari et al. "Stochastic p -Bits for Invertible Logic". In: *Physical Review X* 7.3 (July 2017), p. 031014. ISSN: 2160-3308. DOI: 10.1103/PhysRevX.7.031014. URL: http://link.aps.org/doi/10.1103/PhysRevX.7.031014.

[43] Omer Sagi and Lior Rokach. "Ensemble learning: A survey". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.4 (July 2018). ISSN: 1942-4787. DOI: 10.1002/widm.1249. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1249.

[44] Nitish Srivastava and Ruslan Salakhutdinov. "Multimodal learning with Deep Boltzmann Machines". In: *Advances in Neural Information Processing Systems* 3 (2012), pp. 2222–2230. ISSN: 10495258. URL: http://papers.nips.cc/paper/4683-multimodal-learning-with-deep-boltzmann-machines.

[45] Brian Sutton et al. "Intrinsic optimization using stochastic nanomagnets". In: *Scientific Reports* 7 (Mar. 2017). ISSN: 20452322. DOI: 10.1038/srep44370. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5353626/.

[46] Miguel Á Carreira-Perpiñán and Geoffrey E. Hinton. *On contrastive divergence learning*. Tech. rep. University of Toronto, 2005, pp. 33–40.

[47] T. Richardson and R. Urbanke. "The renaissance of gallager's low-density parity-check codes". In: *IEEE Communications Magazine* 41.8 (Aug. 2003), pp. 126–131. ISSN: 0163-6804. DOI: 10.1109/MCOM.2003.1222728.

[48] Saavan Patel, Philip Canoza, and Sayeef Salahuddin. "Logically Synthesized, Hardware-Accelerated, Restricted Boltzmann Machines for Combinatorial Optimization and Integer Factorization". In: *arXiv preprint arXiv:2007.13489* (June 2020). URL: http://arxiv.org/abs/2007.13489.

[49] Roberto Imbuzeiro Oliveira. "Mixing and hitting times for finite Markov chains". In: *Electronic Journal of Probability* 17 (Aug. 2011). DOI: 10.1214/EJP.v17-2274. URL: http://arxiv.org/abs/1108.1708%20http://dx.doi.org/10.1214/EJP.v17-2274.

[50] Yuval Peres and Perla Sousi. "Mixing Times are Hitting Times of Large Sets". In: *Journal of Theoretical Probability* 28.2 (July 2015), pp. 488–519. ISSN: 15729230. DOI: 10.1007/s10959-013-0497-9. URL: http://arxiv.org/abs/1108.0133.

[51] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. 1st ed. Cambridge University Press, 1995. ISBN: 0521474655.

[52] Ryan Hamerly et al. "Experimental investigation of performance differences between coherent Ising machines and a quantum annealer". In: *Science Advances* 5.5 (May 2019), eaau0823. ISSN: 23752548. DOI: 10.1126/sciadv.aau0823. URL: https://advances.sciencemag.org/content/5/5/eaau0823%20https://advances.sciencemag.org/content/5/5/eaau0823.abstract.

[53] Fuxi Cai et al. "Power-efficient combinatorial optimization using intrinsic noise in memristor Hopfield neural networks". In: *Nature Electronics* (July 2020), pp. 1–10. ISSN: 2520-1131. DOI: 10.1038/s41928-020-0436-6. URL: http://www.nature.com/articles/s41928-020-0436-6.

[54] Andrew D. King et al. "Emulating the coherent Ising machine with a mean-field algorithm". In: *ArXiv* (June 2018). URL: http://arxiv.org/abs/1806.08422.

[55] Troels F. Rønnow et al. "Defining and detecting quantum speedup". In: *Science* 345.6195 (July 2014), pp. 420–424. ISSN: 10959203. DOI: 10.1126/science.1252319. URL: http://science.sciencemag.org/.

[56] Tameem Albash and Daniel A. Lidar. "Demonstration of a Scaling Advantage for a Quantum Annealer over Simulated Annealing". In: *Physical Review X* 8.3 (2018). ISSN: 21603308. DOI: 10.1103/PhysRevX.8.031016.

[57] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit". In: *44th International Symposium on Computer Architecture (ISCA),* (Apr. 2017). URL: http://arxiv.org/abs/1704.04760.

[58] Daniel L. Ly and Paul Chow. "A high-performance FPGA architecture for Restricted Boltzmann Machines". In: *Proceedings of the 7th ACM SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'09*. 2009, pp. 73–82. ISBN: 9781605584102. DOI: 10.1145/1508128.1508140.

[59] Sang Kyun Kim et al. "A highly scalable restricted Boltzmann machine FPGA implementation". In: *International Conference on Field Programmable Logic and Applications*. 2009, pp. 367–372. ISBN: 9781424438921. DOI: 10.1109/FPL.2009.5272262.

[60] Sang Kyun Kim, Peter L. McMahon, and Kunle Olukotun. "A large-scale architecture for restricted Boltzmann machines". In: *Proceedings - IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2010*. 2010, pp. 201–208. ISBN: 9780769540566. DOI: 10.1109/FCCM.2010.38.

[61] Song Han, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (Oct. 2016). URL: http://arxiv.org/abs/1510.00149.

[62] Karen Ullrich, Max Welling, and Edward Meeds. "Soft weight-sharing for neural network compression". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2019.

[63] Wenlin Chen et al. "Compressing neural networks with the hashing trick". In: *32nd International Conference on Machine Learning, ICML 2015*. Vol. 3. 2015, pp. 2275–2284. ISBN: 9781510810587.

[64] William Dally. "High-Performance Hardware for Machine Learning". In: *Nips 2015*. 2015.

[65] Mahdi Nazm Bojnordi and Engin Ipek. "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning". In: *IEEE International Symposium on High Performance Computer Architecture*. Mar. 2016, pp. 1–13. DOI: 10.1109/HPCA.2016.7446049. URL: https://ieeexplore.ieee.org/abstract/document/7446049/.

[66] Chang Hung Tsai et al. "A Hardware-Efficient Sigmoid Function with Adjustable Precision for a Neural Network System". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 62.11 (2015), pp. 1073–1077. ISSN: 15497747. DOI: 10.1109/TCSII.2015.2456531.

[67] Ahmed Zeeshan Pervaiz et al. "Weighted p-bits for FPGA implementation of probabilistic circuits". In: *arXiv:1712.04166 [cs]* (Dec. 2017). URL: http://arxiv.org/abs/1712.04166.

[68] M T Tommiska. "Efficient digital implementation of the sigmoid function for reprogrammable logic". In: *IEE Proceedings: Computers and Digital Techniques* 150.6 (2003), pp. 403–411. ISSN: 13502387. DOI: 10.1049/ip-cdt:20030965.

[69] George Marsaglia. "Xorshift RNGs". In: *Journal of Statistical Software* 8 (2003), pp. 1–6. ISSN: 15487660. DOI: 10.18637/jss.v008.i14. URL: https://www.jstatsoft.org/article/view/v008i14/xorshift.pdf.

[70] Makoto Matsumoto and Takuji Nishimura. "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator". In: *ACM Transactions on Modeling and Computer Simulation*. Vol. 8. 1. Jan. 1998, pp. 3–30. DOI: 10.1145/272991.272995. URL: https://dl.acm.org/doi/10.1145/272991.272995.

[71] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search*. Elsevier, 2005. ISBN: 9781558608726. DOI: 10.1016/B978-1-55860-872-6.X5016-1.

[72] D.L. Ly, Volodymyr Paprotski, and Danny Yen. *Neural Networks on GPUs: Restricted Boltzmann Machines*. Tech. rep. 994068682. 2008. URL: http://www.eecg.toronto.edu/~moshovos/CUDA08/arx/NeuralNet_report.pdf.

[73] Song Han et al. "EIE: Efficient Inference Engine on Compressed Deep Neural Network". In: *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*. Institute of Electrical and Electronics Engineers Inc., Aug. 2016, pp. 243–254. ISBN: 9781467389471. DOI: 10.1109/ISCA.2016.30.

[74] Charles Lo and Paul Chow. "Building a multi-FPGA virtualized Restricted Boltzmann Machine architecture using embedded MPI". In: *ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA*. 2011, pp. 189–198. ISBN: 9781450305549. DOI: 10.1145/1950413.1950452.

[75] Kasho Yamamoto et al. "A Time-Division Multiplexing Ising Machine on FPGAs". In: *Proceedings of HEART 2017*. Vol. 6. Association for Computing Machinery, June 2017. ISBN: 9781450353168. DOI: 10.1145/3120895.3120905. URL: https://doi.org/10.1145/3120895.3120905.

[76] Lok Won Kim, Sameh Asaad, and Ralph Linsker. "A fully pipelined FPGA architecture of a factored restricted Boltzmann machine artificial neural network". In: *ACM Transactions on Reconfigurable Technology and Systems* 7.1 (Feb. 2014). ISSN: 19367406. DOI: 10.1145/2539125.

[77] Bingzhe Li, M. Hassan Najafi, and David J. Lilja. "An FPGA implementation of a Restricted Boltzmann Machine classifier using stochastic bit streams". In: *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*. Vol. 2015-Septe. 2015, pp. 68–69. ISBN: 9781479919246. DOI: 10.1109/ASAP.2015.7245709. URL: https://ieeexplore.ieee.org/abstract/document/7245709/.

[78] Bingzhe Li, M. Hassan Najafi, and David J. Lilja. "Using stochastic computing to reduce the hardware requirements for a restricted boltzmann machine classifier". In: *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, New York, USA, Feb. 2016, pp. 36–41. ISBN: 9781450338561. DOI: 10.1145/2847263.2847340. URL: http://dl.acm.org/citation.cfm?doid=2847263.2847340.

[79] Daniel L. Ly and Paul Chow. "A multi-FPGA architecture for stochastic Restricted Boltzmann Machines". In: *FPL 09: 19th International Conference on Field Programmable Logic and Applications* (2009), pp. 168–173. DOI: 10.1109/FPL.2009.5272516. URL: https://ieeexplore.ieee.org/abstract/document/5272516/.

[80] Shuxian Jiang et al. "Quantum Annealing for Prime Factorization". In: *Scientific Reports* 8.1 (Dec. 2018), pp. 1–9. ISSN: 20452322. DOI: 10.1038/s41598-018-36058-z. URL: www.nature.com/scientificreports.

[81] S. V. Isakov et al. "Optimized simulated annealing for Ising spin glasses". In: *Computer Physics Communications* 192 (Jan. 2014), pp. 265–271. DOI: 10.1016/j.cpc.2015.02.015. URL: http://arxiv.org/abs/1401.1084%20http://dx.doi.org/10.1016/j.cpc.2015.02.015.

[82] Salvatore Mandrà, Helmut G. Katzgraber, and Creighton Thomas. "The pitfalls of planar spin-glass benchmarks: Raising the bar for quantum annealers (again)". In: *Quantum Science and Technology* 2.3 (2017), p. 038501. ISSN: 20589565. DOI: 10.1088/2058-9565/aa7877.

[83] Salvatore Mandra and Helmut G. Katzgraber. "A deceptive step towards quantum speedup detection". In: *Quantum Science and Technology* 3.4 (2018), 04LT01. ISSN: 20589565. DOI: 10.1088/2058-9565/aac8b2.

[84] Kyunghyun Cho, Tapani Raiko, and Alexander Ilin. "Parallel tempering is efficient for learning restricted Boltzmann machines". In: *Proceedings of the International Joint Conference on Neural Networks*. July 2010, pp. 1–8. ISBN: 9781424469178. DOI: 10.1109/IJCNN.2010.5596837.

[85] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. "Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks". In: *ACM SIGARCH Computer Architecture News* 44.3 (Oct. 2016), pp. 367–379. ISSN: 0163-5964. DOI: 10.1145/3007787.3001177.

[86] Hayato Goto et al. "High-performance combinatorial optimization based on classical mechanics". In: *Science Advances* 7.6 (Feb. 2021). ISSN: 2375-2548. DOI: 10.1126/sciadv.abe7953.

[87] Takahiro Inagaki et al. "Large-scale Ising spin network based on degenerate optical parametric oscillators". In: *Nature Photonics* 10.6 (2016), pp. 415–419. ISSN: 17494893. DOI: 10.1038/nphoton.2016.68.

[88] Naeimeh Mohseni, Peter L. McMahon, and Tim Byrnes. "Ising machines as hardware solvers of combinatorial optimization problems". In: *Nature Reviews Physics* 4.6 (May 2022), pp. 363–379. ISSN: 2522-5820. DOI: 10.1038/s42254-022-00440-8.

[89] Saavan Patel et al. "Ising Model Optimization Problems on a FPGA Accelerated Restricted Boltzmann Machine". In: (Aug. 2020).

[90] Yi An Ma et al. "Sampling can be faster than optimization". In: *Proceedings of the National Academy of Sciences of the United States of America* 116.42 (2019), pp. 20881–20885. ISSN: 10916490. DOI: 10.1073/pnas.1820003116.

[91] Paul R. Gray et al. *Analysis and design of analog integrated circuits*. 5th ed. John Wiley & Sons, 2009.

[92] D.A. Freitas and K.W. Current. "CMOS current comparator circuit". In: *Electronics Letters* 19.17 (1983), p. 695. ISSN: 00135194. DOI: 10.1049/el:19830474.

[93] Christian C. Enz and Gabor C. Temes. "Circuit techniques for reducing the effects of Op-Amp imperfections: Autozeroing, correlated double sampling, and chopper stabilization". In: *Proceedings of the IEEE* 84.11 (1996), pp. 1584–1614. ISSN: 00189219. DOI: 10.1109/5.542410.

[94] Andrea Grimaldi et al. "Experimental evaluation of simulated quantum annealing with MTJ-augmented p-bits". In: *2022 International Electron Devices Meeting (IEDM)*. IEEE, Dec. 2022, pp. 1–22. ISBN: 978-1-6654-8959-1. DOI: 10.1109/IEDM45625.2022.10019530.

[95] Yutaka Akiyama et al. "Combinatorial optimization with Gaussian machines". In: Publ by IEEE, 1989, pp. 533–540. DOI: 10.1109/ijcnn.1989.118630.

[96] J J Hopfield. "Neural networks and physical systems with emergent collective computational abilities." In: *Proceedings of the National Academy of Sciences* 79.8 (Apr. 1982), pp. 2554–2558. ISSN: 0027-8424. DOI: 10.1073/pnas.79.8.2554.

[97] Vivek H Sridhar et al. "The geometry of decision-making in individuals and collectives". In: *ECOLOGY BIOPHYSICS AND COMPUTATIONAL BIOLOGY* (2021). DOI: 10.1073/pnas.2102157118/-/DCSupplemental.

[98] Matthias Troyer and Uwe-Jens Wiese. "Computational Complexity and Fundamental Limitations to Fermionic Quantum Monte Carlo Simulations". In: *Physical Review Letters* 94.17 (May 2005), p. 170201. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.94.170201.

[99] Kerem Y. Camsari, Shuvro Chowdhury, and Supriyo Datta. "Scalable Emulation of Sign-Problem–Free Hamiltonians with Room-Temperature ¡math display="inline" overflow="scroll"¿ ¡mi¿p¡/mi¿ ¡/math¿ -bits". In: *Physical Review Applied* 12.3 (Sept. 2019), p. 034061. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.12.034061.

[100] Shuvro Chowdhury, Kerem Y. Camsari, and Supriyo Datta. "Accelerated quantum Monte Carlo with probabilistic computers". In: *Communications Physics* 6.1 (Apr. 2023), p. 85. ISSN: 2399-3650. DOI: 10.1038/s42005-023-01202-3.

[101] Andrew D. King et al. "Scaling advantage over path-integral Monte Carlo in quantum simulation of geometrically frustrated magnets". In: *Nature Communications* 12.1 (Feb. 2021), p. 1113. ISSN: 2041-1723. DOI: 10.1038/s41467-021-20901-5.

[102] Norman P. Jouppi et al. "Ten lessons from three generations shaped Google's TPUv4i: Industrial product". In: *Proceedings - International Symposium on Computer Architecture* 2021-June (2021), pp. 1–14. ISSN: 10636897. DOI: 10.1109/ISCA52012.2021.00010.

[103] Navid Anjum Aadit, Masoud Mohseni, and Kerem Y. Camsari. "Accelerating Adaptive Parallel Tempering with FPGA-based p-bits". In: *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, June 2023, pp. 1–2. ISBN: 978-4-86348-806-9. DOI: 10.23919/VLSITechnologyandCir57934.2023.10185207.

[104] Masoud Mohseni et al. "Nonequilibrium Monte Carlo for unfreezing variables in hard combinatorial optimization". In: *arXiv* (Nov. 2021).