Accelerating Belief Propagation with Task-Based Hardware Parallelism

by

Balaji Venkatesh

Supervisor: Professor Mark C. Jeffrey

April 2024

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

Abstract

This project is the convergence of two directions, the first being innovations in residual belief propagation. Belief propagation is an algorithm used to compute statistical inferences on graphs called probabilistic graphical models, where random variables form nodes, and a probability mass function forms edges. Belief propagation takes these models and computes the marginal distributions of all the random variable nodes. Applications include stereo image depth estimation [1], workplace safety [2], [3], and healthcare patient experience [4]. Existing implementations of BP fail to achieve acceptable convergence coverage, convergence rate, and linear scaling with hardware resources [5]–[9]. The second direction is hardware support for priority-ordered algorithms through task-based parallelism [10]. Speculative hardware parallel execution of BP has been simulated and implemented with CPUs, but no cost-effective pure hardware implementation exists [11], [12]. This work implements belief propagation on an FPGA-based speculative parallel accelerator [13] and demonstrates the possibility of increased performance.

Acknowledgements

I would like to thank my supervisor, Professor Mark C. Jeffrey, for his guidance and support throughout the course of this project. I'm grateful for his patience and advice through the numerous challenges faced. I would also like to thank Gil Posluns and Leo for their help and guidance in understanding the Chronos architecture and their gracious feedback at every step along the way.

Finally, I would like to thank my family for their unwavering support and encouragement throughout my academic career and my friends for their encouragement throughout this project.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

This project is the convergence of two directions – innovations in relaxed-priority belief propagation (BP), and hardware support for priority-ordered irregular algorithms. BP is an algorithm used to compute statistical inferences on graphs called probabilistic graphical models (PGMs), where random variables form nodes, and a probability mass function forms edges. BP takes a PGM and computes the marginal distributions of all the random variable nodes. Applications include stereo image depth estimation [1], workplace safety [2], [3], and healthcare patient experience [4].

Given a set of random variables $X_1 \ldots X_n$, related to each other through a joint probability mass function $p(x‘)$, BP will find the marginal distributions of all the variables, $P(X_i)$. Messages containing ”beliefs” about random variable estimates are passed to neighboring nodes, allowing updates to propagate through the network until convergence. BP is approximate, as there is no analytic solution when there are loops [14].

$$P_{X_i}(x_i) = \sum_{x‘:x‘_i = x_i} p(x‘)$$

With compute-intensive bulk-synchronous techniques, convergence fails with respect to time and model size. However, ordered asynchronous residual propagation improves performance and convergence [7]. Relaxed ordering has lower priority queue overhead, resulting in greater parallelism while maintaining convergence [9]. This is the software optimization limit for BP on standard multi-core processors, as its graphical nature and on-the-fly prioritization makes prediction difficult. Swarm [10] and Chronos [13] have developed a highly parallel architecture for ordered irregular task-based workloads on reconfigurable computing fabric. They make use of ordering, space awareness, and speculation to parallelize tasks with lower overhead.

1.2 Research Gap

Various commercial BP accelerators exist for specific applications, such as computer vision [15] and onboard error correction [16], [17]. However, these are not easily accessible to the general public, and are not easily adaptable to new applications. BP acceleration has also been implemented on Hive, which is a general-purpose speculative parallel computing platform; however, Hive requires a custom hardware design, and as such has only been simulated [11]. These solutions require dedicated hardware implementations, making them inaccessible to most users, as hardware development costs range in the millions of dollars. As such, there is a need for a general-purpose BP accelerator that is easily accessible to the general public, and can be adapted to new applications. FPGAs present a good platform, as they are more easily accessible, with costs similar to that of standard servers.

1.3 Research Objectives and Methodology

This project is a continuation of a previous EngSci thesis [12]. The aim is an easily implementable efficient BP accelerator, bringing together relaxed-priority BP and speculative priority-ordered parallel hardware. Initial work involves the RISC-V soft processors used as processing units in Chronos, which will be updated with new hardened floating point arithmetic units. After successful use of soft processors, a hard processor will be developed and tested in order to achieve further acceleration. Adjustments need to be made to Chronos for effective use of relaxed-priority scheduling. Tuning and other changes then need to be made to Chronos to scale without overloading queues. Finally, the system needs to be optimized for performance and area effectiveness.

1.4 Significance

A successful project would result in an easily implementable FPGA-based BP accelerator, which would be able to compute much larger workloads than software techniques allow. Additionally, it would not require custom hardware and would rather use FPGAs, which are commercially available at similar costs to servers. FPGAs can already be accessed at low cost through Amazon Web Services (AWS), which is used for this project. This would result in accessible BP acceleration for new, more intensive applications.

As mentioned earlier, BP has applications in various fields. Prior work in software acceleration has been limited by compute time and model size, and prior work in hardware acceleration has been limited by cost and accessibility. This work would allow users to compute larger graphs, allowing them to take more factors into account and be more precise when designing intermediate factors. Current applications limit the size of the graph or simplify the model to a tree, which results in reflecting the problem less accurately [2].

In the example of workplace safety, a larger, more accurate model could be used to predict and therefore prevent workplace hazards [2], [3]. Similarly, it could be used to predict patient experience in healthcare [4]. In the example of stereo matching, a larger model could be used to improve the resolution of depth estimation [1]. This would allow for more accurate 3D reconstruction, which could be used in various applications, such as autonomous cars and unmanned arial vehicles [18]. Belief propagation is also used in insurance risk analysis, which uses enormous models with many factors [19]. This work would allow for more accurate risk analysis and insurance premiums.

BP continues to be a useful algorithm when compared to neural networks, as it uses named random variables that can be interpreted more easily. When there is already enough information to create a model, BP also uses much fewer resources for training and similar overhead. Combined BP graph neural networks are being developed to make best use of both worlds, meaning that efficient BP remains more important than ever before [20].

2 Literature Review

2.1 Introduction

Before delving into the specifics of the proposed project, it is essential to understand the current state of relaxed-priority belief propagation (BP) and its significance in accelerating graphical model computations. Belief propagation, as introduced in the project's introduction, is a widely utilized algorithm for computing statistical inferences on probabilistic graphical models (PGMs). However, the conventional bulk-synchronous techniques often face challenges in terms of convergence time and scalability concerning model size [14]. Software optimizations have pushed the limits of BP on standard multi-core processors, laying the groundwork for the proposed project's objectives. Concurrently, the literature reveals the importance of hardware support for priority-ordered irregular algorithms, a facet critical to the success of BP acceleration. This synthesis of software and hardware optimizations sets the stage for an easily implementable and efficient BP accelerator.

The purpose of this literature review is to provide background on the project, identify the research gap, and establish the research objectives. It begins with background on Bayesian statistics and probabilistic graphical models, followed by an explanation of belief propagation. Then it covers the two lines of research that are converging in this project, the first being algorithmic advancements in computing BP, and the second being the hardware acceleration of irregular algorithms. Then, efforts to combine the two are discussed. Finally, the gap in the literature is identified.

2.2 Background

This background on Bayesian statistics, probabilistic graphical models, and belief propagation is based on the textbooks INTRODUCTION TO BAYESIAN STATISTICS by Karl-Rudolf Koch [21] and INFORMATION, PHYSICS, AND COMPUTATION by Marc Mezard and Andrea Montanari [22].

2.2.1 Bayesian Statistics

Bayesian statistics is founded on Bayes' theorem, a fundamental principle in probability theory. At its core, it provides a framework for updating probabilities based on new evidence. The theorem is expressed as,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where $P(A|B)$ is the probability of hypothesis A given evidence B, $P(B|A)$ is the probability of evidence B given hypothesis A, $P(A)$ is the prior (previously known) probability of hypothesis A, and $P(B)$ is the probability of evidence B regardless of A.

An example of this inference method the diagnostic test problem, where a patient is tested for an infection. Our hypothesis is that the patient is infected, which has a prior probability of $P(D) = 1\%$. In other words, 1% of the population is infected. The probability that the test is positive when the patient is infected (also known as the likelihood) is $P(T|D) = 95\%$. In this case, it means that the test is 95% sensitive. The evidence is the test result, which is positive with probability $P(T) = P(T|D)P(D) + P(T|\neg D)P(\neg D)$. This is the probability that the test is positive, regardless of whether the patient is infected (also known as the marginal). We can compute this using the chances of false positives and chances of not being infected, which are the inverses of the probabilities given above. Finally, we are trying to find the probability that the patient is infected given that the test is positive, $P(D|T)$. For this, we can use Bayes' Theorem, given in Equation (2.2.1).

In this example. both the prior and likelihood were based on our knowledge of the population and the sensitivity of the test respectively. This subjectivity is true in general when using Bayesian statistics. In many problems, expert knowledge is used to determine these values.

We can extend this example to multiple tests. For example, we can add a second test that is 99% sensitive, meaning that it is 99% likely to be negative when the patient is not infected. We can then compute the probability that the patient is infected given that both tests are positive, $P(D|T_1, T_2)$, using Bayes' Theorem. This is the joint probability distribution of the two tests. This allows us to compute the probability of the patient being infected given any combination of test results. As such, we can use Bayesian inference to improve the confidence of our predictions by using multiple sources of evidence.

2.2.2 Joint Probability Distributions and Conditional Independence

A joint probability distribution describes the simultaneous probabilities of two or more random variables taking specific values. For two random variables, X and Y, the joint probability distribution $P(A, B)$ provides the probabilities of all possible combinations of values for A and B. This distribution can be dependant on the probabilities of A, B, both, or even other variables.

Two random variables, A and B, are conditionally independent given a third variable C if the occurrence or value of C makes A and B independent from each other. Mathematically, this is expressed as $P(A, B|C) = P(A|C)P(B|C)$. This means that the joint probability distribution of A and B is equal to the product of the conditional probability distributions of A given C and B given C, allowing us to break down what would otherwise be a complicated joint distribution.

2.2.3 Probabilistic Graphical Models

A probabilistic graphical model (PGM) is a graph where nodes represent random variables and edges represent the conditional probability distributions between them. PGMs are used to model complex systems, such as the relationship between the weather and the likelihood of a flight being delayed based on numerous factors like the saturation of the airport. We can represent a given PGM as a large joint probability distribution. However, this is often intractable, as the number of variables and combinations of values grows exponentially.

A class of PGM is the Markov Random Field (MRF), where a node is conditionally independent of all other nodes given its neighbors. In the example in Section 2.2.2, we would therefore see a connection between A and C as well as B and C, but not between A and B, as the presence of C makes them independent.

This means that the joint probability distribution of all the nodes is equal to the product of the conditional probability distributions of each node given its neighbors. This is useful, as it allows us to break down the joint probability distribution of a large number of variables into smaller conditional probability distributions.

MRFs find uses in various areas. For instance, the Ising model is a MRF used to model ferromagnetism [23]. The Potts model is a generalization of the Ising model that can be used to model other phenomena, such as the spread of cancer [24]. MRFs are also used in computer vision, where they can be used to model the relationship between pixels in an image [14]. Another application was discussed in the introduction, where MRFs are used to model the relationship between workplace safety factors [2]. As seen in Figure 1, the graphical nature of MRFs makes them easy to interpret, which is useful in many applications.

2.2.4 Belief Propagation

Various inference algorithms exist to compute Bayesian inferences on MRFs. We focus on Belief Propagation (BP), which is an message passing algorithm that computes the marginal probability distributions of all the nodes. As previously mentioned, in the graphical representation of MRFs,

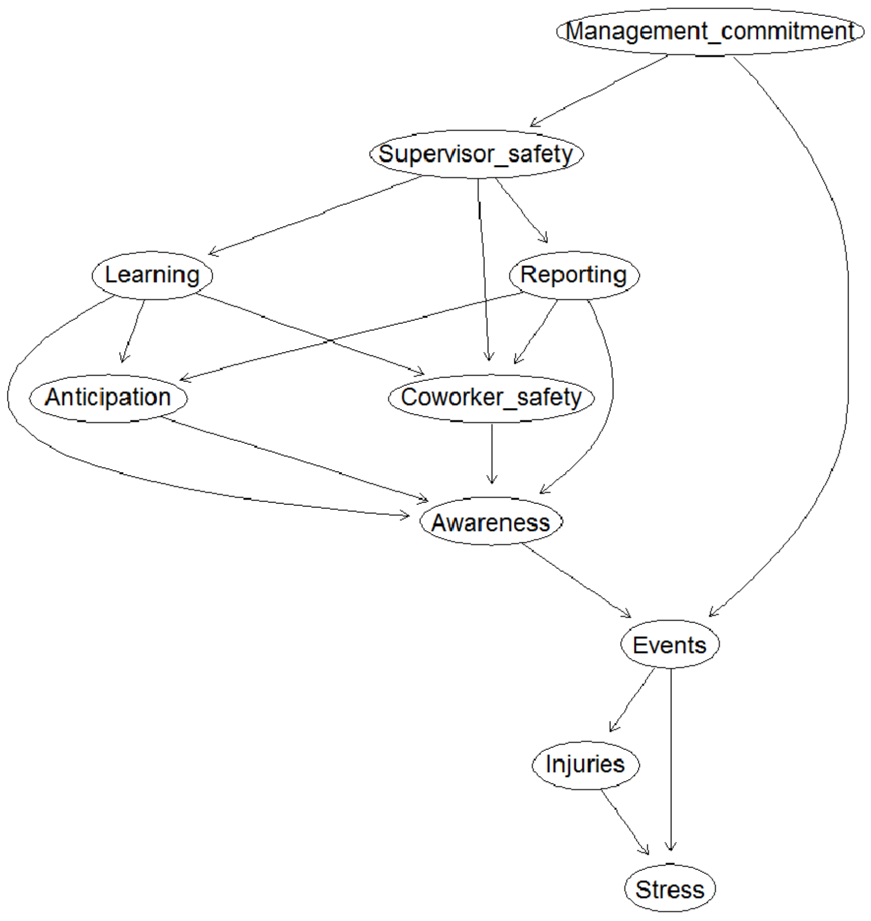
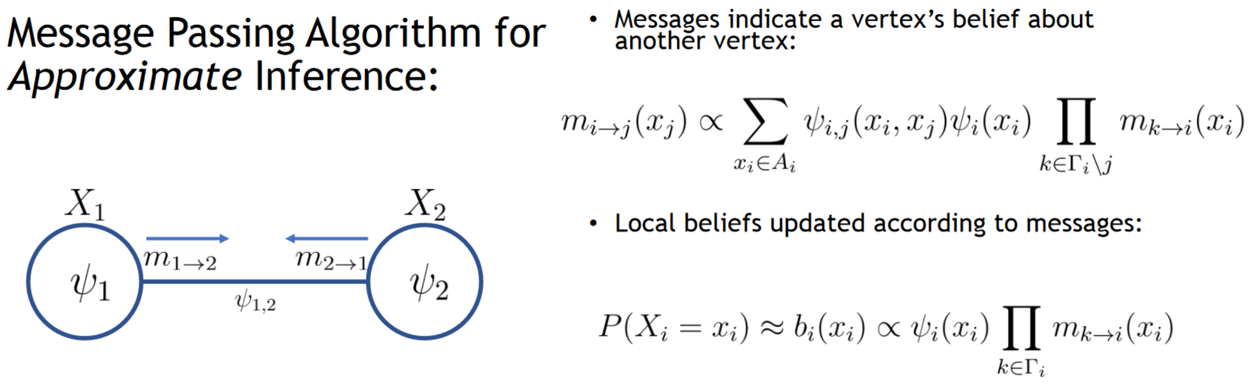


Figure 1: An example of a Markov Random Field being used for workplace safety [2]



Message Passing Algorithm for *Approximate* Inference:

- Messages indicate a vertex's belief about another vertex:

$$m_{i\to j}(x_j) \propto \sum_{x_i \in A_i} \psi_{i,j}(x_i, x_j)\psi_i(x_i) \prod_{k\in\Gamma_i\setminus j} m_{k\to i}(x_i)$$

- Local beliefs updated according to messages:

$$P(X_i = x_i) \approx b_i(x_i) \propto \psi_i(x_i) \prod_{k\in\Gamma_i} m_{k\to i}(x_i)$$

Figure 2: An example of the message passing during a cycle of the BP algorithm.[25]

nodes correspond to random variables, and edges denote conditional dependencies between these variables. The BP algorithm works by sending messages on each edge, as shown in Figure 2, where messages act as probability distributions conveying beliefs about neighboring variables. The iterative process unfolds through forward and backward passes:

1. Forward Pass (Message Calculation): Nodes exchange messages with their neighboring nodes based on their current beliefs and local information. The messages are calculated by taking into account the potential functions associated with the edges, encapsulating the conditional dependencies between variables.

2. Backward Pass (Message Update): Nodes update their beliefs using the received messages from neighboring nodes. This update incorporates both the incoming messages and the local potential functions associated with the node. Each node stores the marginal probability distribution of the corresponding random variable.

The iterations continue until a convergence criterion is met, indicating that the messages have stabilized. The final beliefs at each node represent the approximate marginal probabilities of the corresponding random variables. This information is valuable in understanding the likelihood of different variable states given the observed data and the graphical model's structure.

The computations of the marginals using BP is much less intensive than computing a full joint probability distribution. This is because only the local information from neighbors is used to compute the messages, rather than the entire graph. This is especially useful when the graph is sparse, as the number of neighbors is much smaller than the total number of nodes. This is the case in many real-world applications, such as the workplace safety example in Figure 1.

2.2.5 Belief Propagation Goals

Specific BP methods exist that are guaranteed to converge on chains and trees with an analytical result. This is because a chain or a tree allows a natural ordering of the nodes, which can be used to determine the order in which messages are passed. Figure 1 shows a problem that was modelled in this way [2]. However, this is not the case for general graphs, where loops can exist. In these cases, BP is not guaranteed to converge, and the result is not necessarily accurate. This is known as loopy BP, and it is an approximate algorithm [5].

In general, there are three metrics for a BP algorithm: convergence coverage, convergence rate, and scalability. Convergence coverage refers to the size of graph for which the algorithm converges. Convergence rate refers to the speed at which the algorithm converges. Scalability refers to the ability to increase the speed of the algorithm by increasing resources, such as the number of processors or program elements.

2.3 Progression of Belief Propagation Algorithms

2.3.1 Synchronous BP

Traditional BP algorithms are synchronous, meaning that all nodes update their beliefs at the same time. This is known as bulk-synchronous belief propagation. Bulk synchronous techniques have limited convergence coverage and rate, and scale poorly when parallelized.

Advanced bulk synchronous techniques involve parallelizing the synchronous update across numerous cores. Parallel BP (PBP) and Communication Efficient Parallel BP (CE-BPB) are examples of this. While numerous efforts are made to improve their scalability, they face diminishing returns as the number of cores increases past 8 and 16 cores respectively [6].

2.3.2 Residual BP and Residual Splash BP

Residual BP was developed to address the shortcomings of bulk-synchronous BP. Prior to its development, asynchronous BP was avoided since there was little guidance on which nodes should be updated first. Residual BP uses a residual to determine when a node should be updated. The residual is the difference between the current message and the message from the previous iteration. The messages with the highest residuals are received first. This allows the algorithm to be asynchronous, as nodes can be updated at different times. Residual BP has been shown to have better convergence coverage and rate than bulk synchronous BP [7].

The natural extension of residual BP is residual splash BP, which operates using node residuals, which are the sums of incoming message residuals. The node with the highest residual is updated first, and the residuals of its neighbors are updated accordingly, up to a certain distance. This is known as the splash radius. As such, residual splash BP can be parallelized using heuristics to evenly space splashes. Residual splash BP has been shown to have better convergence coverage and rate than residual BP [8]. However, it still scales poorly when parallelized. This is because the priority queue is a global construct, and therefore requires synchronization between processors. This is a problem, as synchronization is a bottleneck in parallel algorithms.

2.3.3 Relaxed Priority BP

In order to mitigate the synchronization overhead of the priority queue, relaxed priority BP was developed by Akensov et. al. This is a relaxed-priority scheduler for asynchronous BP algorithms like residual BP and residual splash BP. It uses a relaxed priority queue that is not strictly ordered, allowing for parallelization. The priority queue is relaxed by allowing a set of the top-priority nodes with to be updated in any order. The data structure used here is a multiqueue, which consists of many exact priority queues. When dequeuing tasks, two priority queues are picked uniformly at random and the higher priority task is chosen. In order to allow for greater parallelization, they use many independent priority queues and allow multiple cores to pick tasks using the comparative random method. This allows for much greater parallelization, as these tasks can be completed in parallel.

This method presented a significant improvement in scalability compared to residual and residual splash BP. While they designed Relaxed BP to target hardware parallelism, Aksenov et al. did not implement it on hardware. Their software implementation still does not scale linearly with many cores (32+), indicating significant remaining software overhead [9].

2.4 Task-Based Parallel Hardware

2.4.1 Why Task-Based Parallelism?

Due to the dynamic ordering of relaxed priority BP, it is difficult to scale using traditional parallelization techniques. This is because the order in which nodes are updated is not known in advance. As such, it is difficult to divide the work between processors. This is a problem, as parallelization is essential to scaling the algorithm.

Simultaneous operation devices like GPUs and vector processors have also been used to compute BP. However, these devices are not well suited to the irregular nature of BP. This is because they are designed for regular workloads, where the same operation is performed on many data elements. Merwe et al. found that, for generic BP, GPUs face a tradeoff between parallelism and convergence coverage [26]. However, they are effective for applications such as stereo matching which relate closely to the image processing capabilities of a GPU [27].

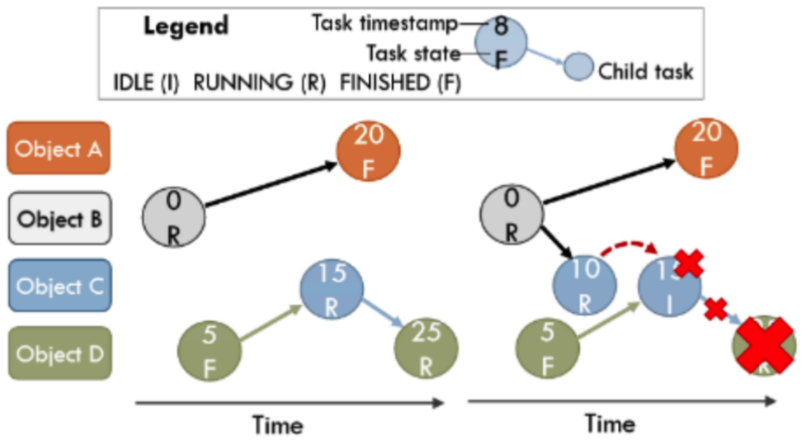
Task-based parallelism is a programming paradigm that focuses on breaking down a computation into smaller, independent tasks that can be executed concurrently to improve overall system performance. In task-based parallelism, rather than explicitly specifying the execution order of program components, developers identify tasks that can be executed concurrently and define dependencies between them. This approach allows for better utilization of available computational resources, as tasks can be dynamically scheduled and distributed across multiple processors or cores. Task-based parallelism is particularly beneficial in scenarios where the workload is dynamic or unpredictable, as the system can adapt to changes by efficiently allocating resources to tasks as they become available [28]. Given that the order of node updates in residual BP is not known in advance, task-based parallelism is a promising approach to accelerating the algorithm, but the issue of strict priority ordering remains.

2.4.2 Speculative Parallelism

For this reason, we explore speculative parallelism, which a technique used to improve the performance of irregular parallel algorithms by executing tasks out of order. This is done by speculating that future tasks will not be affected by the outcome of current tasks and executing them ahead of time. If a task changes something that a speculative task depends on, the speculative task is aborted and restarted. Figure 3 shows this process.

In order to do this, tasks are ordered by timestamp, maintaining correctness for sequential progreams. They are sent to cores, where they execute and then can enqueue other tasks to the task queue. They can also write to a cache that is located among a group of cores called a tile. Tiles allow for shared infrastucture between groups of cores without synchronization across the entire chip. When tasks finish executing, their results and state are stores in a structure called the commit queue, and tasks can only commit their results to memory if they are the oldest, ensuring correctness. Any speculative tasks can be aborted before committing, preventing any changes to memory that would cause incorrectness.

This speculative parallelism has been shown to be effective in improving the performance of irregular algorithms. Algorithms like RBP that have a strong sequential order can run correctly while also making use of parallel hardware resources [10].



(a) Tasks can run out of timestamp order

(b) Selective aborts: Task 10 aborts 15 and 25, but not 20

Figure 3: Speculation and Aborts [13]

2.4.3 Custom Hardware Implementations

Based on Swarm, Hive [11] is a custom-hardware highly parallel architecture for ordered irregular task-based workloads. It makes use of ordering, space awareness, and speculation to parallelize tasks with lower overhead.

Figure 4 shows the Hive architecture. It consists of a set of tiles connected through an on-chip network. Each tile contains processing cores, caches, and a task unit. The task unit is responsible for tracking, scheduling, executing, and aborting tasks on that tile. Swarm and Hive both show large speedups over traditional parallelization techniques. However, they require a custom hardware design, which is expensive and difficult to implement. Typical costs for development of application-specific integrated circuits range in the millions of dollars. However, the architecture shown in Figure 4 is physically similar to other accelerator architectures developed on FPGAs. As such, it is possible to implement the similar ideas on an FPGA.

2.5 Existing BP Hardware Acceleration Efforts

Various commercial BP accelerators exist for specific applications, such as computer vision [15] and onboard error correction [16], [17]. However, these are not easily accessible to the general public, and are not easily adaptable to new applications. BP acceleration has also been implemented on Hive; however, as mentioned before, it requires a custom hardware design, and as such has only

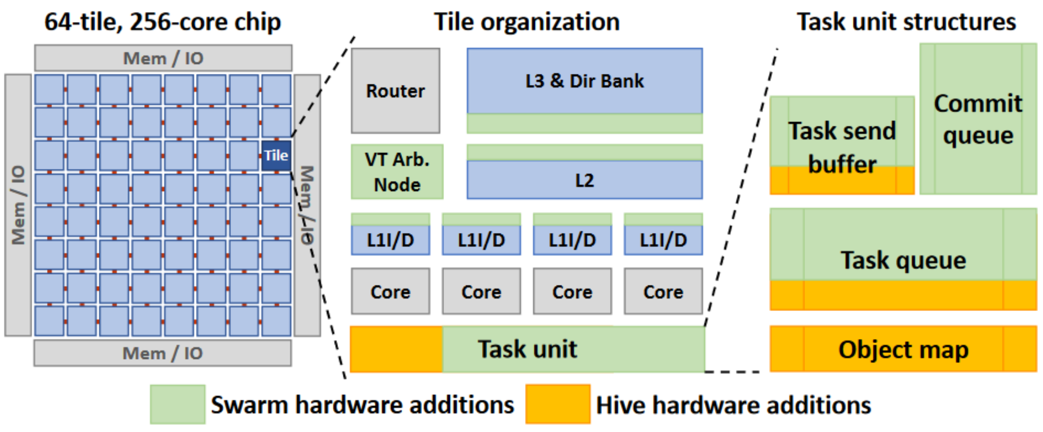


Figure 4: The Hive hardware architecture [11]

been simulated [11]. These solutions require dedicated hardware implementations, making them inaccessible to most users, as hardware development costs range in the millions of dollars.

2.6 Research Gap

The research gap is clear. There is a need for a hardware implementation of residual BP that is accessible to the general public. This is because the current software implementations do not scale well with many cores, and the hardware implementations are inaccessible. Implementing the Residual Belief Propagation algorithm on an FPGA will bridge this gap. The overarching objective is a BP accelerator running on an FPGA that performs better than current options.

2.7 Conclusion

Before delving into the specifics of the proposed project, this literature review provided background on Bayesian statistics and probabilistic graphical models, followed by an explanation of belief propagation. Then it covered the two lines of research that are converging in this project, the first being algorithmic advancements in computing BP, and the second being the hardware acceleration of irregular algorithms. Then, efforts to combine the two were discussed.

Belief propagation, as introduced in the project's introduction, is a widely utilized algorithm for computing statistical inferences on probabilistic graphical models (PGMs). However, the conventional bulk-synchronous techniques often face challenges in terms of convergence time and scalability concerning model size [14]. Software optimizations have pushed the limits of BP on standard multi-core processors, laying the groundwork for the proposed project's objectives. Concurrently, the literature reveals the importance of hardware support for priority-ordered irregular algorithms, a facet critical to the success of BP acceleration. This synthesis of software and hardware optimizations sets the stage for an easily implementable and efficient BP accelerator.

The research gap was identified as the need for a hardware implementation of relaxed priority BP that is accessible to the general public. This is because the current software implementations do not scale well with many cores, and the hardware implementations are inaccessible due to cost and difficult customization. Implementing the Relaxed Belief Propagation algorithm on an FPGA will bridge this gap. The overarching objective is a BP accelerator running on an FPGA that performs better than current options, resulting in a performant and accessible BP accelerator. The significance of this result is presented in Section 1.4.

With this literature review as a backdrop, the subsequent sections will delve into the design overview and methodology, aiming to bridge the identified research gap and contribute to the field of graphical model computations.

3 Design Overview

3.1 Residual Belief Propagation

This section provides an closer overview of residual belief propagation (RBP). We discuss the algorithm's strengths and weaknesses, and how it can be improved by making use of its parallelism opportunities.

The intuition behind RBP is that some messages are more useful than others for convergence. Sending a message that has a similar value to its previous message is redundant, whereas sending a message that has a large difference is more likely to pass information through the network. RBP uses a residual to determine the difference between the current and previous message, and it is defined as the magnitude of the difference between two consecutive values of that message. The message with the largest residual is the most useful for convergence, and is sent first, functioning as a greedy algorithm. Algorithm 1 outlines this process in pseudocode.

Algorithm 1: Residual Belief Propagation (RBP) Algorithm

```
Data: Input graph
Result: Output graph
1 Select random message;
2 while messages > convergence threshold do
3     Update message;
4     Compute residuals;
5     Select message with largest residual;
```

RBP is shown by Elidan et al. to both converge faster and converge more often than traditional BP [7], but it is not without drawbacks. The residual system is computationally expensive, as it uses a priority queue data structure to determine which message to send next, and while the authors claim that this is not an issue, it rears its head when the algorithm is parallelized. The priority queue is a shared resource, and as such, it is a bottleneck when multiple threads try to access it at the same time.

Prior work attempts to reduce the impact of the priority queue by relaxing the priority [9] or localizing updates [8]; however, Hive shows that a task-based parallelism approach provides higher scalability [11]. As such, we introduce a fourth metric to evaluate the performance of our implementation: efficiency, which is how well the algorithm avoids the priority queue bottleneck. This metric only applies to the residual variants of RBP, as only they use a priority queue. Table 1 shows the different variants of RBP and their respective strengths and weaknesses. This comparison jus-

Variant	Coverage	Rate	Scalability	Efficiency
Bulk Synchronous BP	Poor	Poor	None	N/A
Parallel	Poor	Poor	Linear	N/A
Residual	Good	Good	None	Poor
Residual Splash	Good	Good	Sub-linear	Poor
Relaxed Priority	Okay	Okay	Sub-linear	Okay
Speculative Parallel Residual	Good	Good	Linear	Good

Table 1: Comparison of RBP Variants

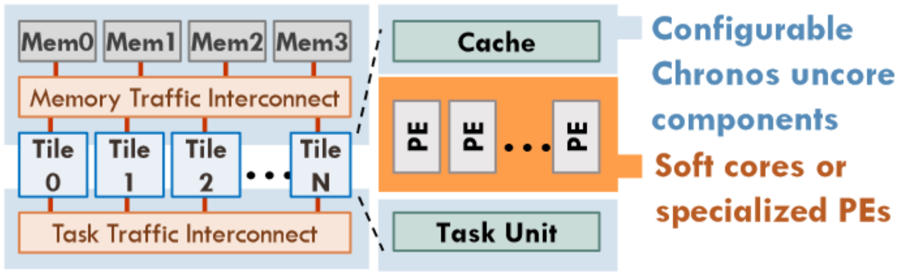

Mem0
Mem1
Mem2
Mem3
Memory Traffic Interconnect
Tile 0
Tile 1
Tile 2
Tile N
Task Traffic Interconnect
Cache
PE
PE
PE
Task Unit
Configurable Chronos uncore components
Soft cores or specialized PEs

Figure 5: The Chronos system overview [13]

tifies the choice of speculative parallelism for our implementation, as it is the most scalable and efficient variant, and maintains good coverage and rate.

3.2 Intro to Chronos and SLOT

Chronos is an accelerator platform based on Swarm that runs on an FPGA [13]. It makes use of a similar system of tiles, with cores called processing elements (PEs), and it also has the same system of task units. Instead of a grid of tiles, they are arranged along a memory traffic and task traffic interconnect. Overall, the complex custom hardware is simplified to run on an FPGA. Figure 5 shows an overview.

Chronos also extends the task-based parallelism paradigm with the Spatially Located Ordered Tasks (SLOT) model. SLOT restricts tasks to single read-write objects, introducing atomicity among computations, allowing for much simpler conflict tracking. The systems tries to schedule tasks to the same tile as their dependencies, reducing the need for communication between tlies.

Figure 6 shows the SLOT model in action. The tasks are ordered by their timestamps, and in turn their dependencies, and as such, we can implement atomic accesses and other parallel programming constructs. In this example, we want to transfer data from one object to another, and we can do so by enqueuing tasks to read from the source object and write to the destination

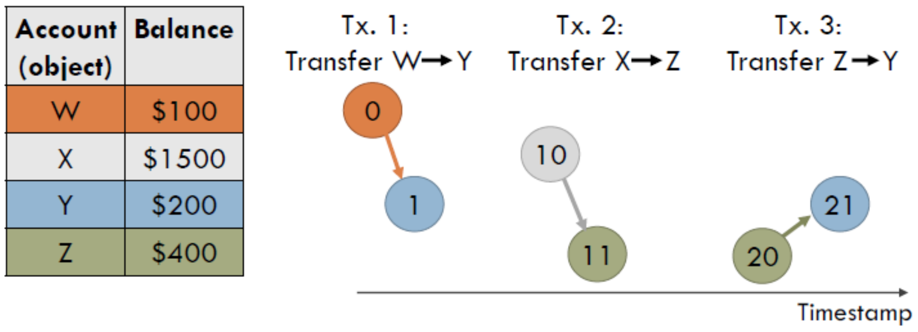


Figure 6: The SLOT model [13]

object. Timestamps ensure that no other tasks are scheduled between the read and write, and the scheduling system ensures that the tasks are sent to tiles where their objects are located. The read task enqueues the write task with the data to be written, and so data transfer between tiles happens without any explicit cache coherence protocol.

3.3 SLOT Task Breakdown for RBP

Based on the parallelism opportunities we identified in the previous section, we can further break down the RBP algorithm into SLOT-compliant tasks, which are shown in Table 2.

We can see that the tasks are far finer than the original RBP algorithm. We need to do this to stay compliant with the SLOT model, since each task can only have write to a single object. This means that a computation like reading inputs to the node and outputting to other nodes is broken up into reading, calculating, and outputting, where the values between stages are passed in the task data.

3.4 Chronos Architecture

The tasks outlined above form a very linear control flow. In a normal software implementation, this would be a problem, as the tasks would be executed one after the other, with no parallelism, but Chronos is designed to handle this and extract paralleism through speculation. We'll cover the achitecture in this section and then explore where it caused problems in the next section.

As shown in Figure 7, the Chronos system is made up of a number of tiles, each with a number of processing elements (PEs) and other tile infrastructure. The processing elements make up the custom logic that runs each app, and the tile infrastructure manages task scheduling, speculation,

Task	Object	Children	Description
Read Reverse Message	Message	Calculate Lookahead	Reads the value of the reverse message
Calculate Lookahead	Source Node	Calculate Priority	Calculates the lookahead
Calculate Priority	Message	Write Priority	Calculates the residual, to be used as task priority
Write Priority	Message	Update Message	Writes the residual, enqueuing a task with the previously calculated priority
Update Message	Message	Update Message Value	Check if the latest priority matches the enqueued priority (wait to reach head of queue)
Update Message Value	Message	Update Node	Updates the message value based on the lookahead
Update Note	Destination Node	Read Reverse Message (one per affected node)	Updates the node log(product in) value

Table 2: RBP Tasks

and communication between tiles. Please note that PEs and cores are synonymous in the context of this project.

Figure 8 zooms into a single tile to show the major tile infrastructure, consisting of a task queue, a commit queue, and a task send buffer. Additionally, there is a cache, which holds the state of the tile's objects and a task interconnect, which connects to other tiles.

The task queue receives and buffers tasks from other tiles and the host. Before a task can be sent to a PE, it needs to reserve a slot on the commit queue, which holds the state of running tasks and commits their changes in timestamp order. It does this by remembering the changes they make and undos them if aborted. The task send buffer holds tasks that are ready to be sent to other tiles.

In the next section, we'll explore how RBP was implemented on Chronos and the problems that arose.

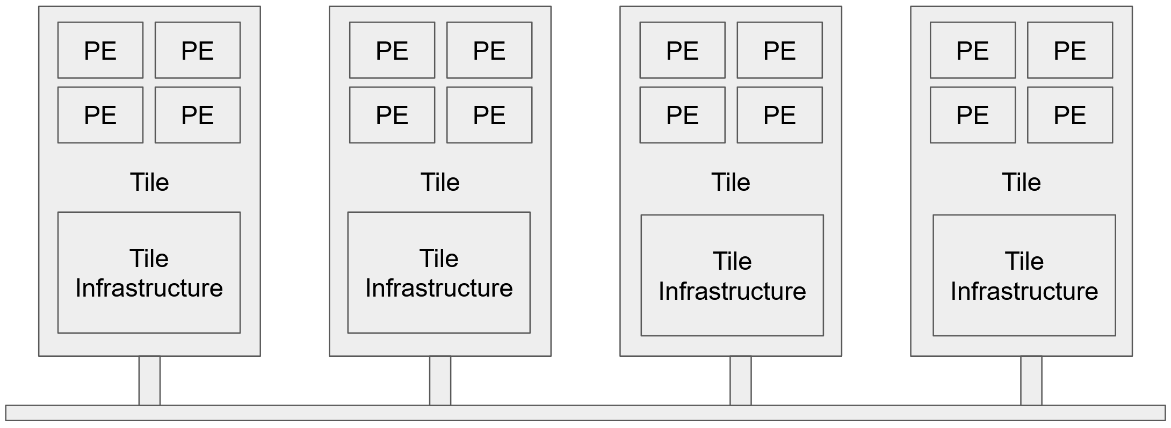


Figure 7: High level Chronos overview [13]

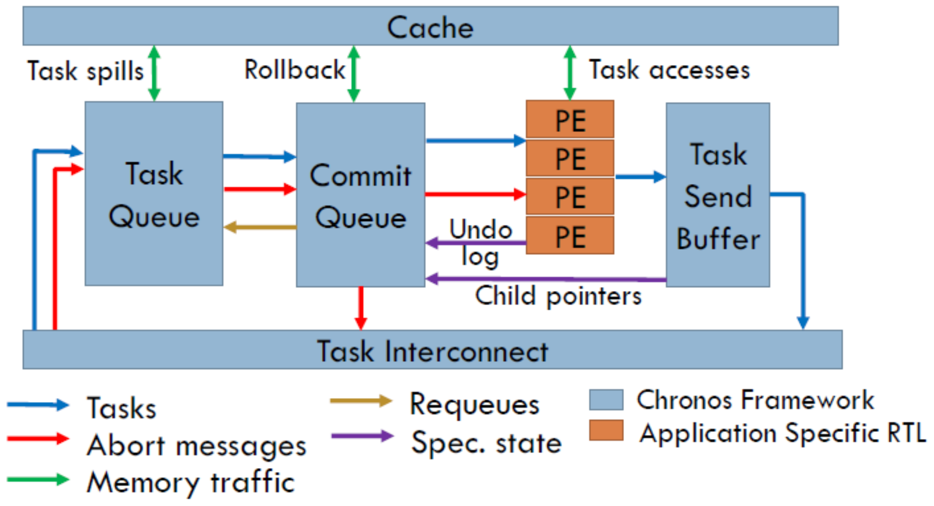


Figure 8: Tile structure [13]

4 Methodology

4.1 High Level Synthesis

Xilinx Vivado High Level Synthesis (HLS) was used to generate the RBP processing element from C++ code. The C++ code is attached in Section 7. This HLS project was then packaged as an IP core and placed in a Xilinx Vivado IP Integrator block diagram. Upon generation of output products, the IP core was instantied in the Chronos architecture, and the IP archive was included in the compile path. This process is described in the Xilinx Vivado Design Suite User Guide [29]. Section 9 shows the full process of generating the IP core and integrating it into the Chronos architecture and how it fits into the overall project flow.

4.2 Deadlocks

4.2.1 Deadlock Avoidance

Chronos has a number of queues, buffers, and other points of contention. A system of arbitration is used to manage these resources, based on the Swarm concept of Global Virtual Time (GVT) [10]. Each module computes its Local Virtual Time (LVT), which is its oldest timestamp. All the local times pass to an arbiter, which computes the GVT. The GVT is then broadcast to all modules, which will always treat the GVT with priority. For instance, in the commit queue, only the GVT can be committed to ensure in-order correctness.

This method improves system performance since the GVT corresponds to the oldest task in the system. Since this task is the oldest, it has no dependencies, only dependencts, and as such, we want to ensure it completes as soon as possible and is never blocked. Otherwise, the accelerator would not be guaranteed to make progress. We can liken the task corresponding to the GVT to a VIP who should be served first, and as such, they should skip all the queues and we can never kick them out if there isn't enough space.

Figure 9 shows the arbitration system managing which PEs are able to send finished tasks first. In the context of a tile, the figure shows a few cores as well as the tasks that they have finished, which are the coloured circles. All the cores wish to send the tasks to the task send buffer, but only one can do this at a time. As such, the cores provide their local virtual times to the arbiter, which computes the global virtual time. The arbiter then sends the global virtual time to the cores, which only send their tasks if their local virtual time is the global virtual time. This ensures that the tasks are sent in order of their timestamps. In this case, we see that the orange task is selected first.

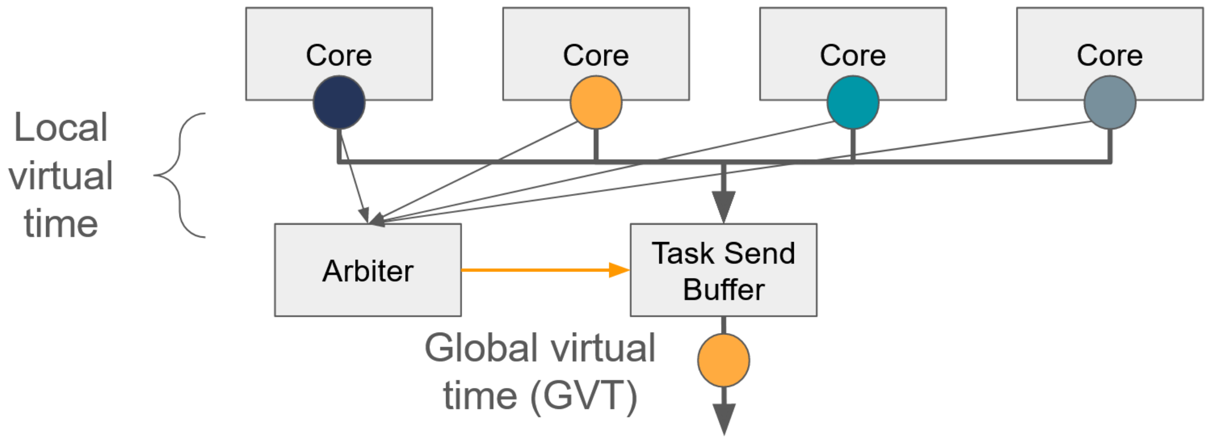


Figure 9: Arbitration After PEs Finish Tasks

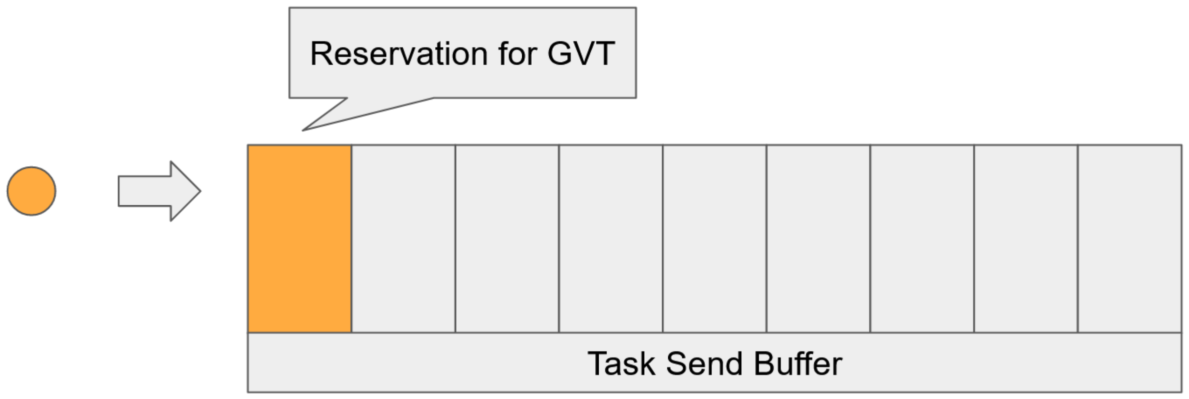


Figure 10: Reservation System in Task Send Buffer

The GVT is also used in maintaining progress through queues and buffers. There are two methods used in this context. Figure 10 shows the reservation system, which is used in the task send buffer. In this system, a slot of the queue is perpetually reserved for the GVT. Since there is only one GVT task in the system at any given time, there can never be contention for this slot. This was typically used for queues that had an order-based enqueuing logic.

The second method is known as a resource abort, shown in Figure 11. Instead of reserving a slot for the GVT, the GVT is allowed to kick out any other task in the queue. This method is often used when the queue doesn't have a particular order. It requires verification that aborting the task or request doesn't affect the functional correctness of the accelerator. In both cases, it is essential to ensure that backpressure is maintained. When the GVT is skipping queues or kicking out tasks, it's important to ensure that the signals that hold back other tasks in contention aren't affected.

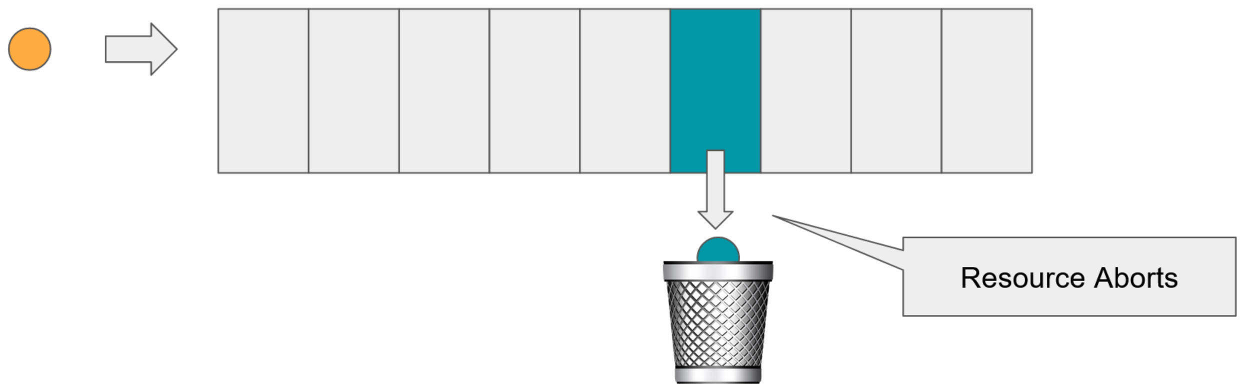


Figure 11: Resource Abort System

4.2.2 Deadlock Detection

Residual belief propagation is one of the most, if not the most intensive algorithm to be run on Chronos. It is a highly parallel algorithm that requires a large number of tasks to be enqueued, dequeued, and aborted. The algorithm is also iterative without the clear limits found in algorithms like Single Source Shortest Path. As such, the limits of the system are tested when running RBP, and a number of issues were found with regards to the deadlock avoidance system. At a high level, these issues are edge cases where the GVT is not correctly prioritized, and as a result, the accelerator is unable to make progress.

Finding the deadlocks is not an easy feat as Chronos is a very large project with many thousands of lines of code. The process to do so involved running the accelerator in simulation on increasingly bigger graphs. When the accelerator was unable to make progress, it was stopped and the waveforms were analyzed to figure out where the system was stuck. Typically this involved looking at the timestamps of the tasks in the system and comparing them to the GVT. Once the GVT was found, the signals holding it back were followed to find the root cause of the deadlock. Simulation at larger sizes took days, so the process was slow and tedious, and multiple simulations were run in parallel to speed up the process. Section 9 shows how the simulation process fit into the overall project flow.

4.2.3 Deadlock Resolution

Once deadlocks were found, they were resolved by adding in necessary arbitration logic, reservation slots, and resource aborts, depending on what was most effective in the specific scenario. The process was iterative, and each change required the simulator to be run on smaller graphs to ensure that correctness was maintained. Then, the simulator was run on larger graphs to ensure that the deadlock was resolved. This process was repeated until the accelerator was able to run on the largest graphs that were tested. At this point, there is no guarantee that there are no more deadlocks, but the simulator cannot run any further due compute and memory limitations. The following sections will detail some of the major deadlocks that were found and how they were resolved. Others were resolved using similar techniques.

4.2.4 Deadlock due to Task Send Buffer Untied Space

Another concept introduced by Swarm [10] is tied and untied tasks. Tied tasks are the children of parents who have not yet committed. They can still be aborted if the parent is aborted. Untied tasks are the children of parents who have committed or the GVT, and so they will not be aborted by their parent task, simplifying tracking within the system. The task send buffer has space for tasks, some of which is reserved for untied tasks. In the original Chronos implementation, the untied space was very small, and RBP can generate a large number of untied tasks when the GVT is the Update Node In Product task, which generates tasks for every affected node. Since overall the algorithm also generates many tied tasks, this caused the task send buffer to run out of space for the GVT's children, deadlocking the accelerator. The solution was to increase the untied space in the task send buffer, which allowed the accelerator to run on larger graphs.

4.2.5 Deadlock due to Child Manager Preemptive Enqueue Attempt

This distinction between tied and untied tasks also lead to issues within the the child manager, which serves as the arbiter between PEs sending their children and the Task Send Buffer. The child manager selects a PE to send its children, and the PE sends the children through the child manager to the task send buffer. It has logic that decides which of the PEs' children is the most important. The untied tasks are usually from the GVT, so they are the most important, and the child manager will pick the PE with untied tasks to send. However, if it has tried to send tied tasks and failed, which occurs when the Task Send Buffer is full of tied tasks, it is unable to switch to sending untied tasks. This caused the accelerator to deadlock. The solution was modifying the child manager's state machine to allow it to change the task it was sending if it found one of a higher priority. This allowed the accelerator to make progress.

4.2.6 Deadlock due to Child Manager Request Overflow

The child manager also has a request system to handle aborts and other requests from PEs and elsewhere. This request system itself is its own queue, even though it does not contain tasks. There were cases when the GVT would issue requests to the child manager, and the child manager would not be able to handle them, causing the accelerator to deadlock. The solution was to add resource aborts to the child manager request system, which allowed the GVT to kick out other requests and ensure that it was always able to make progress.

4.2.7 Deadlock due to Commit Queue Overflow

As mentioned earlier, the commit queue holds state for all running and completed tasks, and ensures that changes are made in an ordered manner using an undo log. The commit queue has a limited size, and did not have any mechanism to handle the GVT being enqueued during an overflow. As such, the GVT could be blocked by the commit queue, causing the accelerator to deadlock. The solution was to add a reserved slot for the GVT in the commit queue, which allowed the GVT to always be enqueued and ensure that the accelerator was able to make progress. An added bonus was that the GVT was in the first slot, meaning it would be served quickly.

4.3 Hardware Implementation

When implementing the simulated design in hardware, another issue was found. There is an Advanced Extensible Interface (AXI) between the Chronos tile and the PEs designed by the user. Typically, this interface is developed with the Verilog ”reg” type, which forces flip-flops to buffer the signal. However, in both the Chronos and HLS implementations, the control signals were implemented with the ”wire” type and combinatorial logic. This would be fine on one side, but since both sides were combinatorial, there was a logic loop between the two modules, preventing implementation. The solution was to modify the logic to use a buffered signal, which allowed the design to be implemented in hardware.

5 Results

5.1 Overview

There were a number of challenges and missed turns throughout this project, resulting in slowed progress than hoped. Clearing the deadlocks in the Chronos accelerator turned out to be a much more involved process than expected. Currently, the accelerator is able to run the RBP algorithm on a small graph, but it is not yet scalable, as there are issues that arise in the hardware implementation related to the inital queueing of tasks. These issues and other next steps are described in Section 5.3. In the next section, we'll review the four metrics to evaluate current progress. Section 8 details other contributions and work that was done on this project that weren't included in the final product.

5.2 Metrics

To evaluate the current progress of the Chronos accelerator, we consider the following four metrics established earlier.

5.2.1 Convergence Coverage

How large is the maximum graph size that reaches convergence?
The accelerator is able to run the RBP algorithm on a small graph, up to around 7x7. Simulation shows a larger convergence than the 2x2 shown by the previous RISC-V implementation, but this is not enough to compete with the standard implementation of Residual Belief Propagation, or its successors, such as relaxed belief propagation, which operates on a 1000x1000 graph.

5.2.2 Convergence Rate

How much time does it take for the accelerator to reach convergence?
Convergence rate is not yet a metric that can be evaluated, as the accelerator is not computing graphs that are comparable to the standard. Rate has been improved in the simulation by increasing the number of cores, and the rate is greater than that of the soft CPU version.

5.2.3 Scalability

How much does performance improve with increased hardware resources?
The accelerator does benefit from reduced cycle count with increased cores, showing that scalabil-

ity is possible; however hardware verification would be needed.

5.2.4 Efficiency

How well does the accelerator deal with priority queue overhead?
The performance gains shown so far can be attributed to the paralleism gained from speculative execution. As such, this system shows promise for mitigating the effects of the priority queue.

As expected, the lack of numeric results means that it is difficult to evaluate the current progress of the accelerator. However, the accelerator is able to run the RBP algorithm on a small graph, and the simulation shows that it is able to converge on larger graphs than the previous RISC-V implementation.

5.3 Next Steps

Currently the accelerator runs correctly for larger graphs than before in simulation; however, when implemented in hardware, it does not produce a result. Debugging has shown that this has to do with the way the initial set of tasks is queued. With small numbers of initial tasks, Chronos can be initalized by directly enqueuing tasks. The RBP accelerator current uses this method and enqueues Update Priority tasks for all of the messages. However, when the number of initial exceeds the buffer capacity, the accelerator cannot initialize. A general solution for this problem is an enqueue tree, which is a tree of tasks that enqueue other tasks. This tree can be used to enqueue a large number of tasks in a scalable way. The tree is built by enqueuing a task that enqueues tasks that enqueues more tasks, and so on. There is a balancing act here, where we need to enqueue at a rate where the machine is kept busy but is not overwhelmed.

Another next step is configuration of the number of tiles and cores per tile that will produce the best accelerator. This is largely a matter of experimentation, but the goal is to find the best balance, as communication between tiles is very slow, but additional cores on a tile increases intra-tile latency and as a result, maximum operating frequency.

Additionally, while many deadlocks have been resolved, there is no guarantee that there are no more. The software simulation cannot run further due to compute and memory limitations, so a new method of testing must be found. Currently, the hardware implementation does not offer signal-level debugging, so some new method of debugging must be found to proceed further with deadlock detection and resolution.

6 Conclusion

To conclude on a positive note, this is the first pure hardware FPGA implementation of a general belief propagation accelerator! This project has shown that the pure hardware implementation unlocks performance that a soft CPU version could not access. Additionally, with many deadlock bugs in Chronos out of the way, further development could focus on improving the performance of the accelerator with respect to the four metrics previously defined. This development could lead this project to serve as a scalable, performant, and cost effective solution for belief propagation on large graphs. Section 5.3 details some of the next steps. Work on this project has also led to fixes for both Chronos and AWS FPGA, which can be useful for the wide range of applications that use these tools.

As previously mentioned, BP has various applications that have been hampered by the limits of software acceleration. With some continued work, this hardware accelerator has the potential to unlock these applications, allowing for more detailed and precise predictions in fields such as computer vision[15], healthcare patient experience[3], [4], workplace safety[2], cancer detection[24], and more.

7 Appendix: Code Sources

The orginal Chronos accelerator is available at `https://github.com/SwarmArch/chronos`. The RBP accelerator is available at `https://github.com/balaji-venkatesh/chronos_rbp_hls`.

8 Appendix: Other Contributions

8.1 RISC-V Implementation

Chronos supports two different types of processing elements. In the final product, the full hardware implementation of the RBP algorithm was done in Vivado HLS, and the processing element was implemented in the FPGA fabric. However, it is possible to implement the processing elements as RISC-V soft processors, which was implemented by Leo when he previously worked on this project. At the time, the RISC-V processor used, VexRiscv [30], did not correctly support floating point operations. When I started on this project, I began with updating the VexRiscv implementation as we thought that floating point operations had been upgraded. Unfortunately, this was not the case, and the RISC-V implementation was not used in the final product.

8.2 AWS-FPGA Improvements for Windows

The Chronos project was originally developed on AWS-FPGA instances, which are Linux-based, and in the end, I developed on a Linux virtual machine to work with Chronos. However, before this, I attempted to work on Windows and I made a number of improvements to the AWS-FPGA development environment to make it easier for Windows users to develop using the Vivado IP Integrator. This included updating a number of scripts to better automate the process of setting up the development environment.

9 Appendix: Project Flow

Figure 12 shows the full project flow for the Chronos project. The project began with the development of the RBP algorithm in C++ and the generation of the processing element in Vivado HLS. The IP core was then integrated into the Chronos architecture, and the full system was synthesized and implemented using AWS-FPGA scripts. A design checkpoint is uploaded to AWS for bitstream generation. The final image was then tested on AWS-FPGA instances. The project was developed in a Linux virtual machine, which is where simulation and waveform viewing occured.

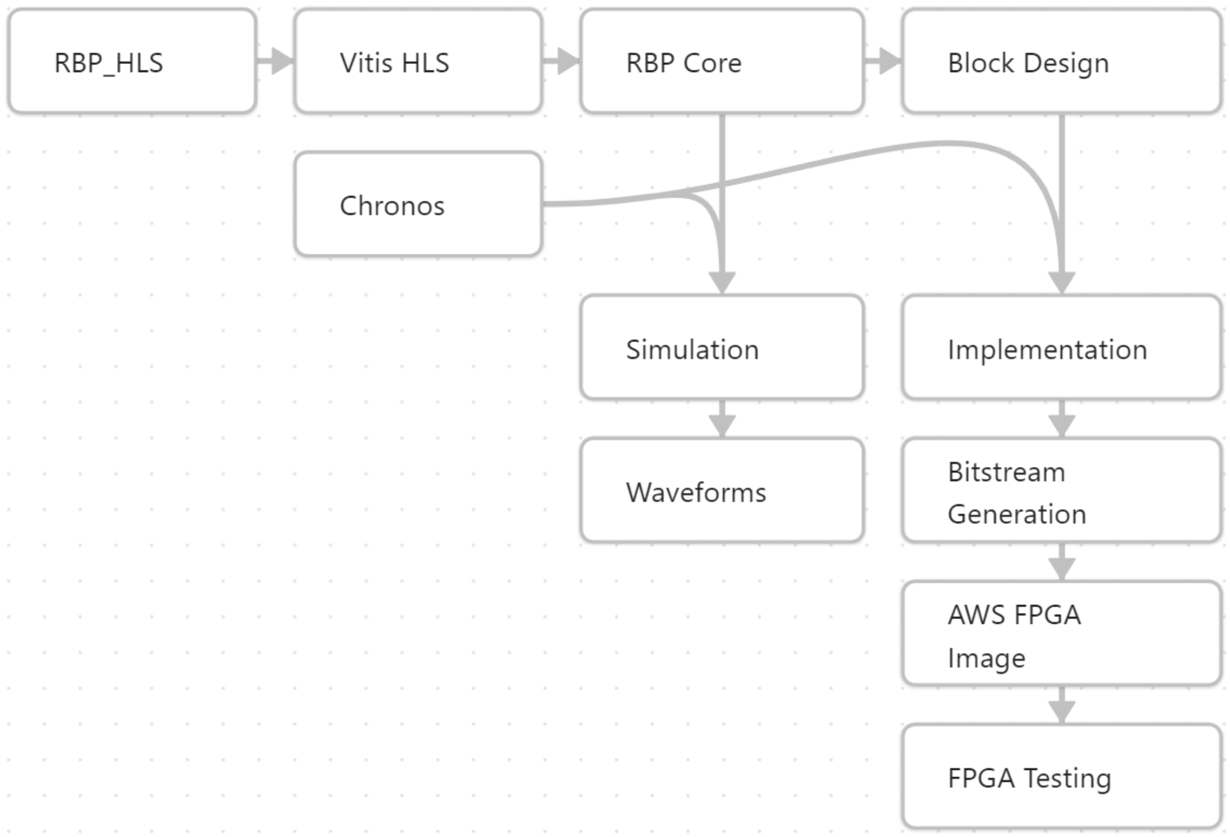


Figure 12: Chronos Project Flow

References

[1] T. Yan, X. Yang, G. Yang, and Q. Zhao, "Hierarchical Belief Propagation on Image Segmentation Pyramid," *IEEE Transactions on Image Processing*, 2023. DOI: `10.1109/TIP.2023.3299192`.

[2] Y. Chen, B. McCabe, and D. Hyatt, *A Belief Network to Predict Safety Performance of Construction Workers*. Vancouver, Canada: University of Toronto, Jun. 2017. [Online]. Available: `https://csce.ca/elf/apps/CONFERENCEVIEWER/conferences/2017/pdfs/CONSPEC/FinalPaper_145.pdf`.

[3] M. C. E. Simsekler and A. Qazi, "Adoption of a Data-Driven Bayesian Belief Network Investigating Organizational Factors that Influence Patient Safety," en, *Risk Analysis*, vol. 42, no. 6, pp. 1277–1293, Jun. 2022. DOI: `10.1111/risa.13610`.

[4] A. Al Nuairi, M. C. E. Simsekler, A. Qazi, and A. Sleptchenko, "A data-driven Bayesian belief network model for exploring patient experience drivers in healthcare sector," en, *Annals of Operations Research*, Jun. 2023. DOI: `10.1007/s10479-023-05437-9`.

[5] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference* (The Morgan Kaufmann series in representation and reasoning), eng, Rev. 2. ed., transferred to digital printing. San Francisco, Calif: Morgan Kaufmann, 2009, ISBN: 978-1-55860-479-7.

[6] J.-F. Yan, J. Zeng, Y. Gao, and Z.-Q. Liu, "Communication-efficient algorithms for parallel latent Dirichlet allocation," en, *Soft Computing*, vol. 19, no. 1, pp. 3–11, Jan. 2015. DOI: `10.1007/s00500-014-1376-8`.

[7] G. Elidan, I. McGraw, and D. Koller, "Residual Belief Propagation: Informed Scheduling for Asynchronous Message Passing," 2006. [Online]. Available: `http://www.robotics.stanford.edu/~galel/papers/ElidanRBP.pdf`.

[8] J. Gonzalez, Y. Low, and C. Guestrin, "Residual Splash for Optimally Parallelizing Belief Propagation," en, in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, ISSN: 1938-7228, PMLR, Apr. 2009, pp. 177–184. [Online]. Available: `https://proceedings.mlr.press/v5/gonzalez09a.html`.

[9] V. Aksenov, D. Alistarh, and J. H. Korhonen, "Relaxed Scheduling for Scalable Belief Propagation," Dec. 2020. [Online]. Available: `https://proceedings.neurips.cc/paper/2020/file/fdb2c3bab9d0701c4a050a4d8d782c7f-Paper.pdf`.

[10] M. Jeffrey, "A Hardware and Software Architecture for Pervasive Parallelism," PhD Thesis, Massachusetts Institute of Technology, Oct. 2019. [Online]. Available: `https://dspace.mit.edu/bitstream/handle/1721.1/128566/1220833661-MIT.pdf`.

[11] G. Posluns, Y. Zhu, G. Zhang, and M. C. Jeffrey, "A scalable architecture for reprioritizing ordered parallelism," en, in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, New York New York: ACM, Jun. 2022, pp. 437–453, ISBN: 978-1-4503-8610-4. DOI: `10.1145/3470496.3527387`.

[12] L. Han, “Accelerating Belief Propagation with Hardware Speculative Parallelism,” Undergrad Thesis, University of Toronto, Toronto, Canada, Apr. 2023.

[13] M. Abeydeera and D. Sanchez, “Chronos: Efficient Speculative Parallelism for Accelerators,” Mar. 2020. DOI: 10.1145/3373376.3378454.

[14] C. M. Bishop, *Pattern recognition and machine learning* (Information science and statistics). New York: Springer, 2006, ISBN: 978-0-387-31073-2.

[15] J. Choi and R. A. Rutenbar, “Configurable and scalable belief propagation accelerator for computer vision,” in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Lausanne, Switzerland: IEEE, Aug. 2016, pp. 1–4, ISBN: 978-2-8399-1844-2. DOI: 10.1109/FPL.2016.7577316.

[16] Y. Sun, Y. Shen, W. Song, Z. Gong, X. You, and C. Zhang, “LSTM Network-Assisted Belief Propagation Flip Polar Decoder,” in *2020 54th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA: IEEE, Nov. 2020, pp. 979–983, ISBN: 978-0-7381-3126-9. DOI: 10.1109/IEEECONF51394.2020.9443504.

[17] W. Phakphisut, P. Supnithi, T. Sopon, and L. M. M. Myint, “Serial Belief Propagation for The High-Rate LDPC Decoders and Performances in The Bit Patterned Media Systems With Media Noise,” *IEEE Transactions on Magnetics*, vol. 47, no. 10, pp. 3562–3565, Oct. 2011. DOI: 10.1109/TMAG.2011.2155049.

[18] R. Fan, S. Guo, and M. J. Bocus, Eds., *Autonomous driving perception: fundamentals and applications*, eng. Singapore: Springer, 2023, OCLC: 1402041977, ISBN: 978-981-9942-87-9.

[19] L. Mkrtchyan, U. Straub, M. Giachino, T. Kocher, and G. Sansavini, “Insurability risk assessment of oil refineries using Bayesian Belief Networks,” en, *Journal of Loss Prevention in the Process Industries*, vol. 74, p. 104 673, Jan. 2022. DOI: 10.1016/j.jlp.2021.104673.

[20] J. Jia, C. Baykal, V. K. Potluru, and A. R. Benson, “Graph Belief Propagation Networks,” 2021. DOI: 10.48550/ARXIV.2106.03033.

[21] K.-R. Koch, *Introduction to Bayesian Statistics*, en. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ISBN: 978-3-540-72723-1. DOI: 10.1007/978-3-540-72726-2.

[22] M. Mézard and A. Montanari, *Information, Physics, and Computation*, en, 1st ed. Oxford University PressOxford, Jan. 2009, ISBN: 978-0-19-857083-7. DOI: 10.1093/acprof:oso/9780198570837.001.0001.

[23] E. Ising, “Beitrag zur Theorie des Ferromagnetismus,” de, *Zeitschrift für Physik*, vol. 31, no. 1, pp. 253–258, Feb. 1925. DOI: 10.1007/BF02980577.

[24] A. Szabó and R. M. H. Merks, “Cellular Potts Modeling of Tumor Growth, Tumor Invasion, and Tumor Evolution,” *Frontiers in Oncology*, vol. 3, 2013. DOI: 10.3389/fonc.2013.00087.

[25] M. V. D. Merwe, G. Gopalakrishnan, and V. Joseph, *Message Scheduling for Performant Many-Core Belief Propagation*, University of Utah. [Online]. Available: https://www.ieee-hpec.org/2019/2019Program/program_htm_files/04-MessageScheduling-hpec_slides.pdf.

[26] M. V. D. Merwe, V. Joseph, and G. Gopalakrishnan, "Message Scheduling for Performant, Many-Core Belief Propagation," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA: IEEE, Sep. 2019, pp. 1–7. DOI: 10.1109/HPEC.2019.8916366.

[27] S. Grauer-Gray and J. Cavazos, "Optimizing and Auto-tuning Belief Propagation on the GPU," in *Languages and Compilers for Parallel Computing*, K. Cooper, J. Mellor-Crummey, and V. Sarkar, Eds., vol. 6548, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 121–135. DOI: 10.1007/978-3-642-19595-2_9.

[28] M. Myllykoski, *Task-based parallelism in scientific computing*, May 2021. [Online]. Available: https://hpc2n.github.io/Task-based-parallelism/branch/spring2021/.

[29] AMD, *Instantiating IP Into the Design • System-Level Design Entry (UG895)*. [Online]. Available: https://docs.amd.com/r/2021.2-English/ug895-vivado-system-level-design-entry/Instantiating-IP-Into-the-Design (visited on 04/15/2024).

[30] C. Papon, *VexRiscv*, 2023. [Online]. Available: https://github.com/SpinalHDL/VexRiscv.