# Accelerating Belief Propagation with Task-Based Hardware Parallelism

Balaji Venkatesh
*University of Toronto*
Toronto, Canada
b.venkatesh@alumni.utoronto.ca

Leo Han
*Cornell Tech*
New York, USA
lxh4@cornell.edu

Mark Jeffrey
*University of Toronto*
Toronto, Canada
mcj@ece.utoronto.ca

***Abstract*—This project is the convergence of two directions, the first being innovations in belief propagation (BP), an inference algorithm for probabilistic graphical models. These are graphs where random variables form nodes, and a probability mass function forms edges. BP computes the marginal distributions of the nodes. Applications include workplace safety, cancer detection, and healthcare patient experience [1]–[3]. Existing implementations of BP fail to achieve acceptable convergence coverage, convergence rate, and linear scaling with hardware resources [4]–[7]. The second direction is hardware support for priority-ordered algorithms through task-based parallelism. Speculative hardware parallel execution of BP has been simulated and implemented in software, but no cost-effective pure hardware implementation exists [8]–[11]. This work implements BP on an FPGA-based speculative parallel accelerator and demonstrates the possibility of increased performance.**

***Index Terms*—Field programmable gate arrays, Hardware acceleration, Parallel processing, Belief propagation.**

## I. Introduction

Probabilistic graphical models (PGMs) are graphs where nodes are random variables and edges are the conditional probability distributions between them. As shown in Figure 1, they can model complex systems like the workplace safety relationships between injuries, training, and reporting. A PGM can be represented as a set of random variables $X_1 \dots X_n$ and a joint probability distribution $p(x')$. Markov Random Fields (MRFs) are a class of PGMs where each node is conditionally independent of all other nodes given its neighbors. This means that the joint probability distribution of all the nodes can be broken down into the product of the conditional probability distributions of each node given its neighbors.

$$P_{X_i}(x_i) = \sum_{x':x'_i = x_i} p(x') \tag{1}$$

Belief propagation (BP) finds the marginal distributions, $P(X_i)$, of the nodes in an MRF. As shown in Figure 2, BP works by sending messages on each edge containing probability distributions conveying beliefs about neighboring variables. The iterative process unfolds through forward and backward passes.

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

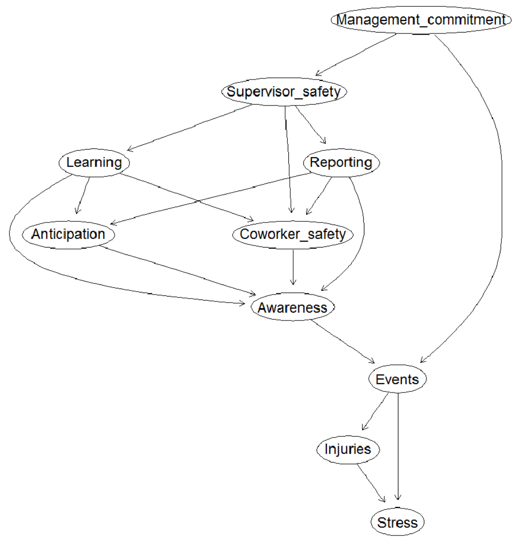

Fig. 1. An example of a Markov Random Field being used for workplace safety [1]

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

Iterations continue until messages stabilize and a convergence criterion is met. The final beliefs at each node are the marginal probabilities of the corresponding random variables. This is much less intensive than computing a full joint probability distribution, especially when the graph is sparse, as information is only needed from local neighbors to compute the messages. When there are loops in the model, BP has no analytic solution, becoming an approximate algorithm, and convergence fails as graph sizes increase [13]. Ordered asynchronous residual propagation improves performance and convergence [6], and relaxed ordering has lower queue overhead, resulting in greater parallelism [7]. This is the software optimization limit for BP on standard multi-core processors.

Various commercial hardware accelerators exist for specific BP applications [14], [15], but these are not easily accessible or adaptable to new applications. Simultaneous operation

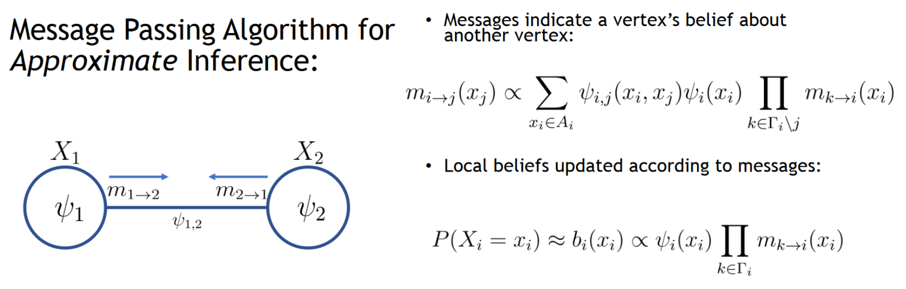
Message Passing Algorithm for *Approximate* Inference:
• Messages indicate a vertex's belief about another vertex:
$m_{i\to j}(x_j) \propto \sum_{x_i \in A_i} \psi_{i,j}(x_i, x_j)\psi_i(x_i) \prod_{k\in\Gamma_i\setminus j} m_{k\to i}(x_i)$
• Local beliefs updated according to messages:
$P(X_i = x_i) \approx b_i(x_i) \propto \psi_i(x_i) \prod_{k\in\Gamma_i} m_{k\to i}(x_i)$
$X_1$
$X_2$
$m_{1\to 2}$
$m_{2\to 1}$
$\psi_1$
$\psi_2$
$\psi_{1,2}$

Fig. 2. An example of the message passing during a cycle of the BP algorithm. [12]

devices like GPUs and vector processors have been used to accelerate BP, but these devices are not well suited to the irregular nature of BP [16]. General BP accelerators require incredibly expensive custom hardware designs, and as such have only been simulated [10]. Therefore, there is a need for a hardware BP accelerator that is cost effective and adaptable.

Swarm [8] and Chronos [9] developed an architecture for ordered irregular task-based parallel workloads on field-programmable gate arrays (FPGAs), making use of ordering, space awareness, and speculation to parallelize tasks with lower overhead. FPGAs present a good platform, as they are more easily accessible, with costs similar to that of standard servers. To summarize, the aim of this project is to create a general-purpose BP accelerator that makes use of task-based speculative parallelism via the Chronos FPGA architecture.

## II. Metrics

In models where loops exist, BP is not guaranteed to converge [4]. As such, there are three general 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.

## III. Residual Belief Propagation

Traditional BP algorithms are synchronous, meaning that all nodes update their beliefs at the same time. This is known as bulk-synchronous belief propagation. While numerous efforts are made to improve their scalability, these techniques have limited convergence coverage and rate, and scale poorly when parallelized [5].

Asynchronous BP was initially avoided since there was little guidance on which nodes should be updated first. As such, Residual Belief Propagation (RBP) was developed using a residual heuristic, and has better convergence coverage and rate than bulk synchronous BP [6]. The intuition 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 defines the residual as the magnitude of the difference between two consecutive values of a message.

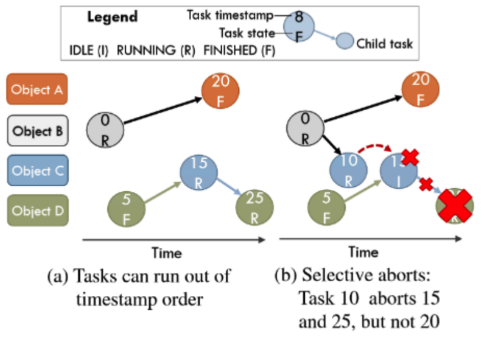
Legend
Task timestamp
Task state
8
F
Child task
IDLE (I) RUNNING (R) FINISHED (F)
Object A
Object B
Object C
Object D
Time
Time
(a) Tasks can run out of timestamp order
(b) Selective aborts: Task 10 aborts 15 and 25, but not 20

Fig. 3. Speculation and Aborts [9]

The message with the largest residual is sent first. The order is determined using a priority queue, which is a shared resource parallelization bottleneck. As such, we introduce a fourth RBP-specific metric: efficiency, which is how well the algorithm avoids the priority queue bottleneck.

```
Data: Input graph
Result: Output graph
Select random message;
while messages > convergence threshold do
    Update message;
    Compute residuals;
    Select message with largest residual;
end
```
**Algorithm 1:** Residual Belief Propagation

## IV. Task-Based Parallel Hardware

Task-based parallelism focuses on breaking down a computation into smaller, independent tasks that can be dynamically scheduled and executed concurrently, which is particularly beneficial in scenarios where the workload is dynamic or unpredictable. Since node update order in RBP is not known in advance, this is a promising approach, but the strict priority order poses an issue.

Swarm introduces speculative parallelism, which improves irregular algorithm performance by executing tasks out of order [8]. As shown in Figure 3, we order tasks by timestamp to maintain correctness and then speculatively execute future tasks, assuming they will not be affected by the outcome of current tasks, selectively aborting if dependencies change.

TABLE I
COMPARISON OF BP VARIANTS

| Variant | Coverage | Rate | Scalability | Efficiency |
|---|---|---|---|---|
| Bulk Synchronous | 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 | Good | Good | Linear | Good |

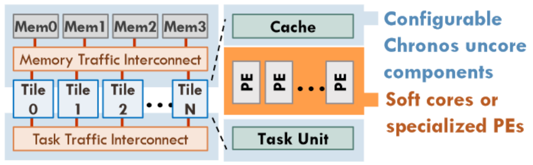
Mem0
Mem1
Mem2
Mem3
Memory Traffic Interconnect
Tile 0
Tile 1
Tile 2
Tile N
Task Traffic Interconnect
Cache
PE
PE
PE
Task Unit
Configurable Chronos uncore components
Soft cores or specialized PEs

Fig. 4. The Chronos system overview [9]

Algorithms like RBP that have a strong sequential order can run correctly while also making use of parallel hardware resources. Table I shows the different variants of BP and their respective strengths and weaknesses. This comparison justifies the choice of speculative parallelism for our implementation, as it is the most scalable and efficient variant, and maintains good coverage and rate.

Chronos is an accelerator platform based on Swarm that runs on an FPGA [9], outlined in Figure 4. Tasks run on cores called processing elements (PEs) which are grouped onto tiles that allow for shared infrastucture without synchronization across the entire chip. Figure 5 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.

When tasks finish executing, their results and state are stored in 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. 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.

## V. SPATIALLY LOCATED ORDERED TASKS (SLOT)

Chronos extends the task-based parallelism paradigm with SLOT, restricting tasks to single read-write objects and 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 tiles.

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

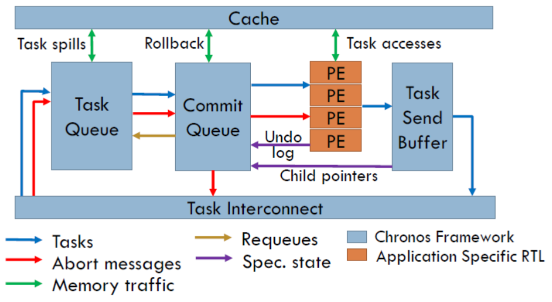

Fig. 5. Tile structure [9]

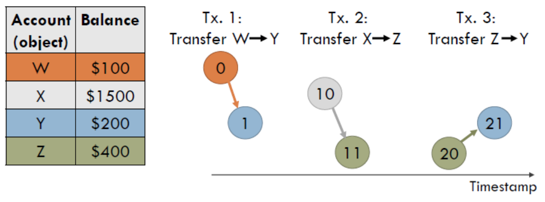

Fig. 6. The SLOT model [9]

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 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.

We can break down RBP into SLOT-compliant tasks, which are shown in Table II. The tasks are far finer than the original RBP algorithm, since each task can only write to a single object. As such, 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.

## VI. IMPLEMENTATION

Xilinx Vivado High Level Synthesis (HLS) was used to generate the RBP processing element from C++ code available at https://github.com/balaji-venkatesh/chronos_rbp_hls. It was packaged as an IP core and placed in a Xilinx Vivado IP Integrator block diagram. Upon generation of output products, the IP core was instantiated in Chronos, and the IP archive was included in the compile path. This process is described in the Xilinx Vivado Design Suite User Guide [17].

In order to implement the HLS cores in hardware, the Advanced eXtensible Interface between the PEs and Chronos was modified to avoid a logical loop.

### A. Global Virtual Time

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) [8]. Each module outputs its lowest timestamp as its Local Virtual Time (LVT) to an arbiter, which computes the

TABLE II
RBP TASKS

| 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 |

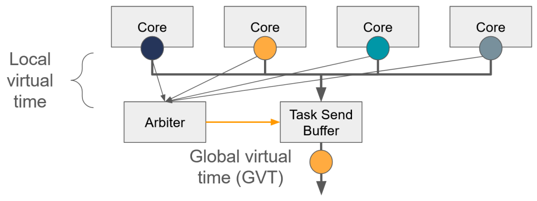

Fig. 7. Arbitration After PEs Finish Tasks

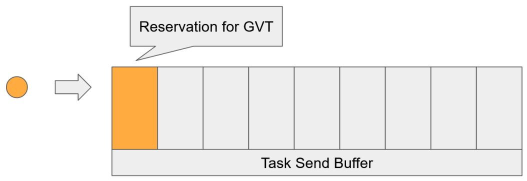

Fig. 8. Reservation System in Task Send Buffer

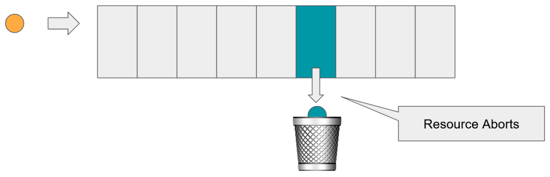

Fig. 9. Resource Abort System

GVT. The GVT is then broadcast back to all modules, which prioritise the corresponding task. For instance, Figure 7 shows the arbitration system managing which PEs are able to send finished tasks first. All the cores wish to send tasks (shown as coloured circles) to the task send buffer, but only one can do this at a time. As such, the cores only send their tasks if their LVT equals the GVT, ensuring that the tasks are sent in order of their timestamps. The GVT corresponds to the oldest task in the system, so 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.

The GVT is also used in two ways to maintain progress through queues and buffers. First is the reservation system, shown in Figure 8. A slot of the queue is perpetually reserved for the GVT, and since there is only one GVT task any given time, there can never be contention for this slot. This is typically used for queues with ordered enqueuing. The second method is the resource abort, shown in Figure 9. Instead of reserving a slot, the GVT task instead kicks out any other task in the queue. This method is often used when the queue doesn't have a particular order and requires verification that aborting the task or request does not affect the functional correctness. In both cases, it is essential to ensure that backpressure is maintained by ensuring that signals that hold back other tasks in contention aren't affected.

## B. Deadlocks

RBP is a highly parallel algorithm that requires a large number of tasks to be enqueued, dequeued, and aborted. As such, the limits of the Chronos were tested when running RBP, and a number of deadlocks were found. At a high level, these are edge cases where the GVT is not correctly prioritized, and as a result, the accelerator is unable to make progress. Detection involved increasing graph size until deadlock whereupon 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.

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. One iteration is described in detail here, and a few others are listed.

Deadlock due to:

1) Task Send Buffer Untied Overflow
2) Child Manager Preemptive Enqueue Attempt
3) Child Manager Request Overflow
4) Commit Queue Overflow: The commit queue (CQ) holds state for all running and completed tasks and ensures that changes are made in an ordered manner using an undo log. The CQ has a limited size and no mechanism to handle enqueuing the GVT during overflow, causing deadlocks. The solution was a reserved slot for the GVT in the CQ, allowing it to always enqueue and make progress. An added bonus was that the GVT was in the first slot, giving it priority.

## VII. Results

Considering the four metrics established earlier:

1) Convergence Coverage: 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 RBP or its successors.
2) Convergence Rate  cannot yet be evaluated, as the accelerator is not computing graphs that are comparable to the standard. However, rate has improved in simulation to be greater than that of the soft CPU version.
3) Scalability: The accelerator demonstrates scalability with reduced cycle count when the number of cores is increased.
4) Efficiency: The performance gains shown so far can be attributed to improved paralleism from speculative execution, showing promise for mitigating the priority queue bottleneck.

## VIII. Conclusion

In conclusion, this work resulted in a speculative parallel general BP accelerator that demonstrates performance which a soft CPU version cannot access. It does not require custom hardware, instead making use of FPGAs, which are commercially available at similar costs to servers. Further development could result in a scalable, performant, and cost effective solution for belief propagation on large graphs. Various BP applications are hampered by compute time and model size limits, often reducing graph size or simplifying to a tree, modeling their environments less accurately [1]. This work could allow including more factors, allowing for more detailed and precise predictions in fields such as workplace safety, cancer detection, and healthcare patient experience [1]–[3]. Compared to neural networks, BP uses less training resources and named random variables that can be interpreted more easily. 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 [18].

## References

[1] 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

[2] A. Szabó and R. M. H. Merks, "Cellular Potts Modeling of Tumor Growth, Tumor Invasion, and Tumor Evolution," *Frontiers in Oncology*, vol. 3, 2013.

[3] 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," *Annals of Operations Research*, Jun. 2023.

[4] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*, rev. 2. ed., transferred to digital printing ed., ser. The Morgan Kaufmann series in representation and reasoning. San Francisco, Calif: Morgan Kaufmann, 2009.

[5] J.-F. Yan, J. Zeng, Y. Gao, and Z.-Q. Liu, "Communication-efficient algorithms for parallel latent Dirichlet allocation," *Soft Computing*, vol. 19, no. 1, pp. 3–11, Jan. 2015.

[6] 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

[7] 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

[8] 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

[9] M. Abeydeera and D. Sanchez, "Chronos: Efficient Speculative Parallelism for Accelerators," Mar. 2020.

[10] G. Posluns, Y. Zhu, G. Zhang, and M. C. Jeffrey, "A scalable architecture for reprioritizing ordered parallelism," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. New York New York: ACM, Jun. 2022, pp. 437–453.

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

[12] 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

[13] C. M. Bishop, *Pattern recognition and machine learning*, ser. Information science and statistics. New York: Springer, 2006.

[14] 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.

[15] 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.

[16] 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.

[17] 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

[18] J. Jia, C. Baykal, V. K. Potluru, and A. R. Benson, "Graph Belief Propagation Networks," 2021.