Optimized Priority Scheduling for Faster Scalable Belief Propagation

Abnash Bassi ABNASH@CMU.EDU
Carnegie Mellon University
Pittsburgh, USA

Gilead Posluns GIL.POSLUNS@MAIL.UTORONTO.CA
University of Toronto
Toronto, Canada

Mark C. Jeffrey MARK.JEFFREY@UTORONTO.CA
University of Toronto
Toronto, Canada

Editors: Gustau Camps-Valls, Manuele Leonelli and Gherardo Varando

Abstract

Belief propagation (BP) is a marginalization method using local message passing on probabilistic graphical models, seen in disciplines like Bayesian inference, statistical physics, and wireless communications. With the proliferation of manycore processors with 100s of cores, BP parallelization solves larger marginalization problems in reasonable time. On one hand, the classic synchronous algorithm easily maps to multithreaded execution with abundant parallelism per synchronization barrier. On the other hand, asynchronous algorithms converge on more graphs and reduce total work and sequential time to convergence, but are more difficult to parallelize due to priority-ordered scheduling. Recently proposed *relaxed* ordered parallel algorithms are plagued by the overheads of a decade-old scheduler. This paper unifies recent innovations in concurrent priority schedulers with relaxed parallel BP, using the Stealing MultiQueue (SMQ) with lazy priority updates. Prior work proposes different relaxed BP algorithms but fails to identify one BP algorithm that outperforms alternatives across inputs. We are the first to empirically show relaxed residual BP (RRBP) consistently outperforming all other baseline BP algorithms: at 384 threads our SMQ-based RRBP outperforms synchronous and other asynchronous BP algorithms by geomean 2.66× and >1.6×, respectively. Our code is available at github.com/mcj-group/rbp.

Keywords: Belief Propagation, Concurrent Priority Scheduling

1. Introduction

A decade of innovation in artificial intelligence (AI) was underpinned by rapidly growing parallelism offered by modern computer architectures. AI techniques that build on tensor algebra thrived because their regular data and control flow patterns easily map to highly parallel architectures like GPUs. Consequently, tensor-algebra-based algorithms like large language models have seen substantial run-time speedups that enabled ever-larger models. Unfortunately, entire classes of *irregular* AI algorithms have stagnated in performance because their data dependences and control flow do not map easily to modern architectures. New techniques are needed to improve performance and solve larger problems.

© 2026 A. Bassi, G. Posluns & M.C. Jeffrey.

Belief propagation (BP) is one such irregular AI algorithm. BP is a probabilistic inference algorithm to determine the probability that random variables take on specific values (Pearl, 1982, 1988; Frey and MacKay, 1997). BP operates on graphical models, passing messages between neighboring nodes. BP has applications in protein folding (Yanover and Weiss, 2002), error-correction in network communication (Shen et al., 2020), and statistical physics (Potts, 1952; Cipra, 1987), among other domains (Majewski et al., 2001). For example, determining the states of physical particles is useful to understand their interactions, but is computationally expensive. BP approximates the probability of specific particle states, such as solving the Ising model for ferromagnetism to model the two-state spin of particles in a grid. The Ising model (Ising, 1925; Cipra, 1987) remains highly relevant, with a recent collaboration between Cerebras and the U.S. Department of Energy demonstrating 88× speedup over NVidia GPUs (Van Essendelft et al., 2025). However, access to specialized wafer-scale systems like Cerebras' hardware remains expensive, so improving the run time of BP-based approaches democratizes access to large probabilistic inference problems.

Prior algorithms for BP trade off *parallelism*, *convergence rate*, and *convergence coverage*. *Synchronous* BP updates all messages in the graph during each bulk iteration. This is straightforward to parallelize by dividing messages across threads. However, it performs a large amount of work in each iteration, increasing the number of updates needed for inference to converge (convergence rate). Furthermore, on some cyclic graphs, synchronous BP struggles to converge at all (Jiang et al., 2017), reducing convergence coverage. *Asynchronous* BP algorithms (Elidan et al., 2006; Sutton and McCallum, 2007; Knoll et al., 2015) offer faster convergence rates (Goldberger and Kfir, 2008) and broader convergence coverage (Hemati and Banihashemi, 2006). For example, residual belief propagation (RBP) (Elidan et al., 2006) carefully selects only one message to update per iteration, using a priority heuristic for selection. Residual splash (Gonzalez et al., 2009) selects a neighborhood of messages to update around one priority-ordered message. Unfortunately, these asynchronous algorithms are conventionally difficult to parallelize: the priority queue that sorts messages must be accessed by many cores in parallel. Naively wrapping the queue in a global lock causes significant contention at high thread counts (Lenharth et al., 2015).

Aksenov et al. (2020) present *relaxed* priority-ordered parallel BP algorithms, demonstrating performance gains with 70 threads. Whereas sequential algorithms follow strict priority schedules that always select the highest priority message per iteration, relaxed versions permit message re-ordering that enables flexible priority scheduling to reduce lock contention. Specifically, the authors use the MultiQueue (MQ) (Rihani et al., 2015), a concurrent priority scheduler that returns one of the top k highest priority messages in expectation, where k is proportional to the number of threads. However, they show performance that is *input-dependent*: the winning BP algorithm varies across undirected graphs, known as Markov random fields (MRFs). This requires programmers to experiment with their specific use case and adapt the selected BP algorithm for a different input MRF. Additionally, although their system yields speedup, it does not scale well beyond 20 threads. This scalability gap remains to be addressed for relaxed ordered BP. Our work addresses both the input-dependency and scalability limitations of prior work on relaxed BP algorithms.

Contributions: This paper unifies recent orthogonal innovations in concurrent priority scheduling and belief propagation, demonstrating that relaxed ordered asynchronous BP

can be made faster and scalable to more threads: we *(i)* replace MultiQueues with Stealing MultiQueues (SMQs) (Postnikova et al., 2022) in relaxed residual and residual splash BP, and *(ii)* replace the expensive priority update operations in these algorithms with a low-overhead lazy updating protocol (Posluns et al., 2022) to further reduce queue overheads. We characterize the memory access overheads of MQ- and SMQ-based RBP to show why the SMQ outperforms the MQ. Using our `C++` implementations of BP algorithms and the optimized SMQ, we demonstrate performance scaling up to 384 threads with a geomean 124× speedup over the sequential RBP algorithm. Our SMQ-based relaxed residual BP outperforms other relaxed residual BP algorithms by geomean 1.68×, 1.63×, and 1.67×, and synchronous BP by geomean 2.66× across MRFs at 384 threads. We show that the SMQ outperforms the MQ by geomean 1.68× at 384 threads. To the best of our knowledge, this paper is the first to identify one BP algorithm that can consistently outperform synchronous BP and other relaxed BP algorithms across cyclic MRFs, including Ising and Potts models and LDPC decoding, and non-cyclic MRFs like trees. Based upon our characterization, this appears to be because the SMQ better leverages cache locality.

2. Background and Motivation

Belief propagation (BP) solves the marginalization problem, determining the probability that random variables take on particular values. In statistical physics, BP estimates the probability that one particle within a particle grid has a spin of -1 or +1 (the Ising model (Ising, 1925; Cipra, 1987)) or generalizes this to physical variables with > 2 states (the Potts model). In network communications, BP decodes low-density parity check codes (LDPC) which provide error correction for 5G (ETSI, 2025).

Specifically, BP solves inference on graphical models and we focus on Markov random fields (MRF). An MRF is an undirected graph $G = (V, E)$. Each node corresponds to a random variable X_i from domain D_i, with its node factor ψ_i representing prior information about X_i taking a specific value. An edge between nodes represents the dependence between random variables X_i and X_j using the edge factor ψ_{ij}.[1] Graph G is an MRF if the joint distribution $p(v_1, \ldots, v_n)$ satisfies the local Markov property: $\forall v \in V, p(v|V \setminus \{v\}) = p(v|N(v))$, where $N(v)$ denotes the neighbors of v. The marginal distributions for some subset of nodes can be computed by examining only their neighbors, rather than the entire graph.

2.1. Belief Propagation Algorithms

BP algorithms "pass messages" in both directions along the edges of the MRF until values converge, with varying levels of *work efficiency* depending on the strategy.[2] For every edge connecting dependent random variables (nodes) X_i and X_j, the MRF has a directed *message* $\mu_{i \to j}$, a vector with $|D_j|$ entries ($\mu_{j \to i}$ in the reverse direction). A BP algorithm passes a message along $\mu_{i \to j}$ by updating it based on the node and edge factors and the

1. We use notation from Aksenov et al. for a pairwise MRF.
2. A work-efficient algorithm's total work across threads asymptotically matches the best sequential algorithm for the problem (Blelloch and Maggs, 1996).

Algorithm 1: Seq. synchronous BP

```
do
    updated ← false
    foreach msg ∈ allMessages do
        oldMu ← μ(msg)
        newMu ← futureMu(msg)
        if oldMu - newMu > sensitivity
         then
            updateMessage(msg, newMu)
            updated ← true
        end
    end
    foreach node ∈ allNodes do
        updateNodeSum(node)
    end
while updated
```

Algorithm 2: Sequential residual BP

```
PriorityQueue pq(allMessages)
while !pq.empty() do
    priority, msg ← pq.top()
    if priority ≤ sensitivity then
        break
    end
    newMu ← futureMu(msg)
    updateMessage(msg, newMu)
    updateNodeSum(msg → j)
    pq.update(msg, priority=0)
    for affected ∈ msgsFrom(msg → j) do
        pq.update(affected, newPriority(affected))
    end
end
```

prior incoming messages to X_i:

$$\mu_{i\to j}(x_j) \propto \sum_{x_i \in D_i} \psi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k \in N(i)\setminus\{j\}} \mu_{k\to i}(x_i). \tag{1}$$

BP repeatedly updates MRF messages until convergence. Each random variable's marginal probability is estimated as the product of its node factor and its incoming messages.

Scheduling the message updates (Eq. 1) differentiates BP algorithms. Synchronous BP simultaneously updates all messages. Alg. 1 shows each outer-loop iteration looping through every message. Synchronous BP is easy to parallelize because the order of inner-loop iteration execution is arbitrary: independent message updates are trivial to split among threads using barriers for synchronization. Unfortunately, synchronous BP has a slow convergence rate: every iteration recomputes the outbound messages for all nodes, even those with unchanged inbound messages. Consequently, only a few updates contribute to improving the vertex belief while the rest are useless (Gonzalez et al., 2009).

Asynchronous algorithms such as Residual BP (RBP) converge faster by updating one message per iteration. RBP selects for updating the message that would change the most, or the one with the largest *residual*, defined as $|\mu_{i\to j}\prime - \mu_{i\to j}|$ (Elidan et al., 2006). By prioritizing based on residual, RBP converges faster than synchronous algorithms. Alg. 2 shows the selection of a message from a strict priority queue.

Although RBP converges quickly, it is not easily parallelizable. Aksenov et al. show the performance benefits of *relaxed* RBP with MultiQueues. This is a multi-threaded approach where each thread completes tasks, i.e., updates messages between graph nodes. In contrast to *strict* RBP that always selects the highest priority task, MQ-based relaxed RBP selects one of the top k highest priority tasks (in expectation) where k scales with thread count. The algorithm is similar to Alg. 2 but with synchronization and an MQ instead of a priority queue. Because the MQ provides a rank guarantee (Alistarh et al., 2017), relaxed RBP converges quickly, similarly to RBP. Aksenov et al. also propose relaxed smart splash, a variant of Gonzalez et al.'s Residual Splash. Rather than selecting one message to update,

smart splash chooses one vertex in the input graph to execute a Splash operation. A vertex's priority is the maximum residual of all incoming messages. The Splash operation uses the selected vertex as the root for breadth-first search (BFS) to a depth of `splashH`. Smart splash updates messages from the leaves to the root of the BFS tree and then from the root to the leaves. As a result, only messages within the constructed BFS tree are updated.

2.2. Convergence Metrics

We quantify the utility and performance of BP algorithms along three axes: *(i)* convergence coverage, *(ii)* convergence rate, and *(iii)* convergence time. Convergence coverage represents the ability to converge on arbitrary graphs (Elidan et al., 2006). In this paper, we compute it as the percentage of graph topologies of a given class (e.g., Ising models) on which each algorithm converges. Convergence rate measures the total number of message updates required to reach the equilibrium point per converging trial. Convergence time, on the other hand, refers to the total run time to convergence (Goldberger and Kfir, 2008).

2.3. Concurrent Priority Scheduling

An ordered parallel BP algorithm uses a concurrent priority scheduler (CPS). A CPS dispatches tasks to threads based on their priority. The tasks are executed in an interleaved or parallel fashion. A single lock-protected priority queue is a naive CPS which would guarantee strict priority ordering, but Lenharth et al. (2015) have shown this to be a poor choice due to lock contention. Relaxed CPS designs do not guarantee strict ordering, but scale better with thread count. There are many relaxed CPS designs, some of which provide probabilistic rank guarantees. We focus on the MultiQueue (Rihani et al., 2015) and Stealing MultiQueue (Postnikova et al., 2022).

MultiQueues comprise k lock-protected sequential priority queues, all shared across all threads (Figure 1). Each priority queue contains tasks, i.e., messages to be updated, ordered from highest to lowest priority. A thread pushes an update to a random queue, and pops by randomly selecting two queues and popping from the one with the highest priority top. A key limitation of the MQ is that because it is shared among all threads, threads must acquire a lock to pop from or push to each priority queue. This adds overhead which may limit the performance of a multi-threaded implementation. The Aksenov et al. Java MQ implementation has an additional source of overhead: it supports a `changePriority()` function. The MQ design does not provide any way to access a node other than by popping. Aksenov et al. resolve this lack with an array that maps message IDs to specific queues and heap nodes. Supporting the node lookups necessary for priority updates adds significant overhead to the underlying heap data structure of a priority queue (Tavory et al., 2004).

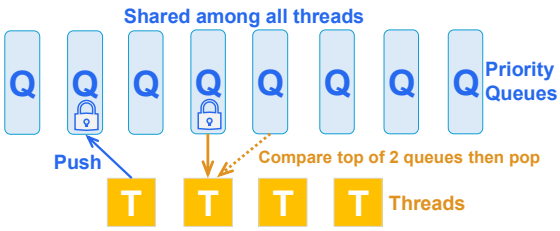


Figure 1: A MultiQueue.

Stealing MultiQueues enhance the MultiQueue with lock-free techniques and more thread-local memory accesses, while maintaining rank guarantees (Figure 2). Each thread has a thread-local priority queue, a stolen buffer, and a stealing buffer. The local queue and stolen buffer contain tasks that only that thread is able to pop. The stealing buffer holds tasks that a thread has made available for other threads to steal. With some probability, or when its local queue is empty, a thread steals tasks from the stealing buffer of a randomly selected thread and moves them into its local stolen buffer. Postnikova et al. show that the SMQ outperforms other leading schedulers, including the traditional MultiQueue, PMOD (Yesil et al., 2019), and OBIM (Nguyen et al., 2013). SMQ's performance gain is attributed to better rank guarantees than prior schedulers because these lead to less wasted work, as well as improved cache locality in its priority queues and reduced contention.

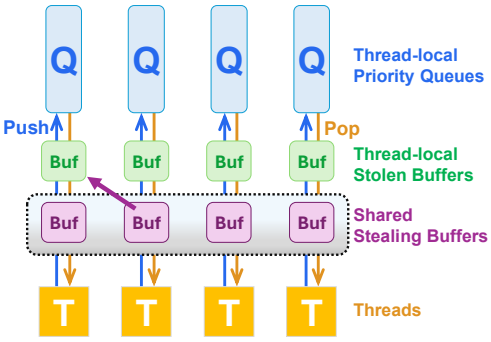
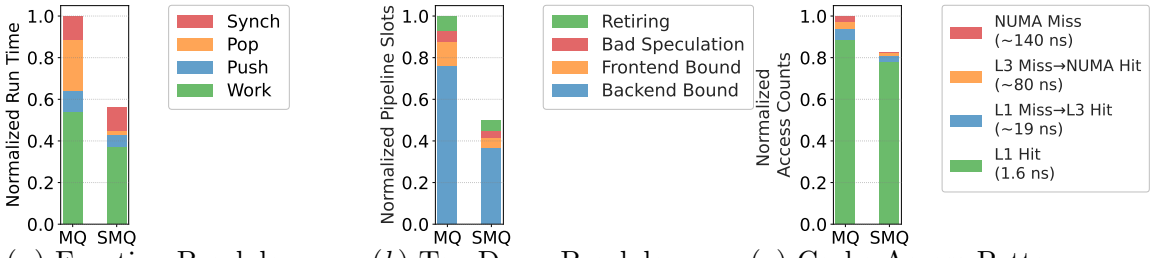


Figure 2: A Stealing MultiQueue.

2.4. Motivation

Despite the importance of BP, state-of-the-art implementations leave performance on the table and fail to identify one superior BP algorithm that outperforms other algorithms across a range of MRFs. This variability can be addressed by improving the scalability of relaxed RBP such that it outperforms other common BP algorithms at high thread counts. We motivate this work by observing that MQ-based relaxed RBP scales poorly with more than twenty threads (Aksenov et al., 2020), plateauing due to the following reasons.

MQ overheads consume BP convergence time: Ideally, threads would spend most of their time on the core work of the relaxed RBP algorithm, but a significant portion of their convergence time is spent on queue operations. Figure 3(a) shows how 384 threads spend their time on the Ising model (Section 4.1 details our methodology). The height of each bar is the convergence time normalized to MQ-based relaxed RBP. Threads spend



(a) Function Breakdown (b) Top-Down Breakdown (c) Cache Access Pattern

Figure 3: Run-time and memory behavior comparing MQ- and SMQ-based relaxed RBP for the Ising input at 384 threads. (a) Time spent in key functions, normalized to MQ run time. (b) Top-down microarchitecture breakdown of pipeline slots, normalized to MQ's total slots. (c) Breakdown of accesses on each level of cache or memory.

35% of the time running the pop and push operations, which extract the next message to be updated from the MQ, or push a new message to it, respectively. The MQ pop and push operations read and write data and acquire a lock for a randomly selected queue. Our SMQ-based relaxed RBP cuts this time dramatically as queue operations are more often local and avoid locking. The reduction in queue-related overhead for the SMQ versus the MQ motivates its viability as a scheduler for ordered asynchronous BP algorithms.

MQ does not leverage cache locality: Where the MQ opts for random queue operations, the SMQ more often opts for local operations, and this affects the time spent accessing variables in memory. Server computer architectures often contain a three-level cache hierarchy, where frequently used data is held in an L1 cache that is fast to access, and less frequently used data is held in larger but slower caches (L2, L3) or dynamic random access memory (DRAM) (Balasubramonian et al., 2011). The more memory accesses that are served by lower-level caches, the faster the convergence time. In MQ-based relaxed RBP, threads access a pool of shared queues. Although the randomness of queue selection reduces lock contention, it increases cache line invalidation, as threads take turns reading/writing queues. Figures 3(*b*) and 3(*c*) provide further insight into this phenomenon, using hardware performance counters and the Top-Down method (Yasin, 2014). Figure 3(*b*) shows the time that threads spend retiring instructions (desired) or waiting on *(i)* the frontend of the processor (e.g., fetching instructions), *(ii)* branch mis-speculation, or *(iii)* the backend of the CPU (e.g., memory accesses). The SMQ cuts the time spent waiting on the processor backend by over half. Figure 3(*c*) shows the breakdown of how threads access data, across the L1 and L3 caches,[3] and memory. Latency estimates are derived from cycle counts reported for older Intel microarchitectures (NASA, 2025), scaled to the clock frequency of our experimental machine (Section 4.1). The SMQ spends relatively more of its memory accesses hitting in the faster L1 cache and less in the slower L3 cache and memory, compared to the MQ. The SMQ also sees fewer Non-Uniform Memory Access (NUMA) misses, indicating that the memory local to the processor's socket more often contains the requested data. This aligns with the SMQ's expected better memory locality relative to the MQ.

3. Implementation

The RBP algorithm includes a priority update operation (Vuillemin, 1978) that dynamically alters the scheduling of message updates (inner-loop body of Alg. 2). Aksenov et al. implement an MQ with updates by wrapping sequential heaps that support arbitrary priority updates. Unfortunately, priority updates are expensive (Tavory et al., 2004; Zhang et al., 2024), requiring an array of pointers to heap nodes that must be updated on each heap operation. High-performance CPS designs do not support priority updates in their data structure, but instead store object priorities in an array and apply updates lazily (Dhulipala et al., 2017; Posluns et al., 2022; Zhang et al., 2024). We detail the implementation of our MQ/SMQ and our approach to lazily apply updates.

RBP with a Stealing MultiQueue: We use the open-source relaxed RBP implementation from Zhang et al. as our baseline. We use the same MultiQueue implementation and an optimized SMQ implementation that outperforms the Postnikova et al. implementation.

3. Our experimental hardware does not report L2 cache hits.

Lazy Priority Updates: The MQ and SMQ do not support priority updates. A naive way to emulate priority updates with a push-pop queue is to replace all updates with inserts and discard stale pops. However, this significantly increases queue pressure, as updates in RBP outnumber pops (Posluns et al., 2022). To reduce queue pressure, we implement a variant of the Posluns et al. `Postpone` pattern. Algorithm 3 shows how we adapt this pattern. Current node priorities are stored in an array. When updating a neighbor (inner-loop body), the algorithm always updates the priorities array, but only pushes into the queue if the priority has increased. On a stale pop, if the current priority from the priorities array is less than the popped priority, the algorithm pushes the node back into the queue with its current priority; otherwise it drops it. This approach keeps queue sizes down by deferring the push(es) when a node is deprioritized.

Algorithm 3: Sequential RBP with a push-pop queue

```
PriorityQueue pq(allMessages)
double prios[M]
while !pq.empty() do
    prio, m ← pq.top()
    if prio ≤ sensitivity then
        break
    end
    pq.pop()
    if prio ≠ prios[m] then
        if prio > prios[m] and
          prios[m] > sensitivity then
            pq.push(m, prios[m])
        end
    end
    else
        prios[m] ← 0.0
        newMu ← getFutureMessage(m)
        updateMessage(m, newMu)
        updateNodeSum(m → j)
        for affctd ∈ msgsFrom(m → j) do
            oldPrio ← prios[affctd]
            prios[affctd] ←
              newPriority(affctd)
            if oldPrio < prios[affctd] then
                pq.push(affctd, prios[affctd])
            end
        end
    end
end
```

4. Evaluation

We evaluate our optimized scheduling techniques for BP across four input MRF types, two relaxed priority-ordered BP algorithms, and the two CPSs—MQ and SMQ. We find that SMQ outperforms MQ by geomean 1.68× at 384 threads. With SMQ at 384 threads, one algorithm outperforms all other BP algorithms on all inputs. We also validate the convergence coverage relative to prior work.

4.1. Experimental Methodology

Hardware: We use a dual-socket Intel Granite Rapids (Intel Xeon 6975P-C) machine on AWS with 96 cores per socket and hyper-threading enabled, for a total 384 threads. The shared L3 cache is 480 MB per socket.

Experiments: We test four algorithm-queue variants: Residual Belief Propagation (RBP) and Relaxed Smart Splash (RSS), each with MQ and SMQ. We also measure the performance of synchronous BP. Aksenov et al. found that other parallel BP algorithms were strictly worse than at least one of RBP, SS, or synchronous BP, so we do not test them. We measure performance on four input MRFs: Ising, Potts, LDPC, and tree. The Potts and Ising graphs have an input size of 1000 by 1000. The LDPC input decodes 3×10^5

bits. The tree input is a randomized binary tree of size 1×10^7. The Potts and Ising inputs model physical states. Ising represents interactions between particles in a lattice to predict the likelihood of phase transitions and is a foundational approach to modeling ferromagnetism (McCoy and Wu, 1973). Potts extends the Ising model to more possible states (Potts, 1952). Our LDPC implementation targets only regular LDPC codes, specifically the (3,6) Gallager code (GL), which is one of the most common LDPC schemes and can achieve near Shannon capacity performance (MacKay and Neal, 1996). Each variable node is guaranteed to be connected to three check nodes, and each check node must be bonded with six variable nodes. The variable nodes represent a received transmission while the check nodes perform parity checks. Running BP on the LDPC bipartite factor graph (Kschischang et al., 2001) corrects transmission errors. By construction, the Ising, Potts, and LDPC inputs contain cycles.

Setup: We implement all code in `C++20` and compile with `g++` 12.3.1 using the `-O3`, `-g`, and `-ggdb` flags. For each input and algorithm, we run five trials, and average run times and update counts. Error bars indicate standard deviation. Results are normalized relative to sequential RBP. Our optimized sequential RBP caches the message update values used to calculate residuals in an array to reuse the computation when the update is applied. We verify correctness for Ising and Potts by comparing the error at each node against the sequential residual algorithm. Our correctness check for LDPC counts the number of successfully decoded bits, which matches the total number of bits transmitted.

We filter out residuals with a sensitivity of 1×10^{-5} and generate random numbers with a seed of 2 for Ising, LDPC, and Tree and a seed of 5 for Potts, to ensure fairness of inputs. We select these seeds to ensure convergence across algorithms, inputs, and thread counts. The SMQ probability of stealing is 1/8. We use the optimized MQ implementation from (Zhang et al., 2024) with `std::priority_queue` as the internal queue. We use 4 queues per thread and set the push-pop batch size to 1. For relaxed smart splash, we use a splash height (`splashH`) of 2 to match the best value identified by Aksenov et al.

4.2. Performance

The top row of Figure 4 compares the performance of each algorithm/queue as thread count grows from 1 to 384 threads. All speedups are normalized to sequential residual BP run times shown in Table 1. Figure 4 shows that our SMQ relaxed RBP outperforms the MQ-based version by 1.8× at 384 threads for Ising, 2.1× for Potts, 1.9× for LDPC, and 1.1× for tree. Using the relaxed RBP algorithm, the geomean speedup of SMQ is 124.5× at 384 threads across all inputs as opposed to 73.9× for MQ. When using the relaxed smart splash algorithm, this gap is reduced to a 76.6× geomean speedup for SMQ and 74.6× for MQ. Our results corroborate prior relative performance advantages and disadvantages of MQ-based relaxed RBP and synchronous BP across inputs (Aksenov et al., 2020). Whereas Aksenov et al. do not identify a singular best algorithm class

Table 1: Average run time in seconds and number of algorithm message updates for sequential RBP across inputs.

	Run time (seconds)	**Updates (millions)**
Ising	77.3	25.3
Potts	92.8	30.0
LDPC	166.1	18.6
Tree	147.5	49.0

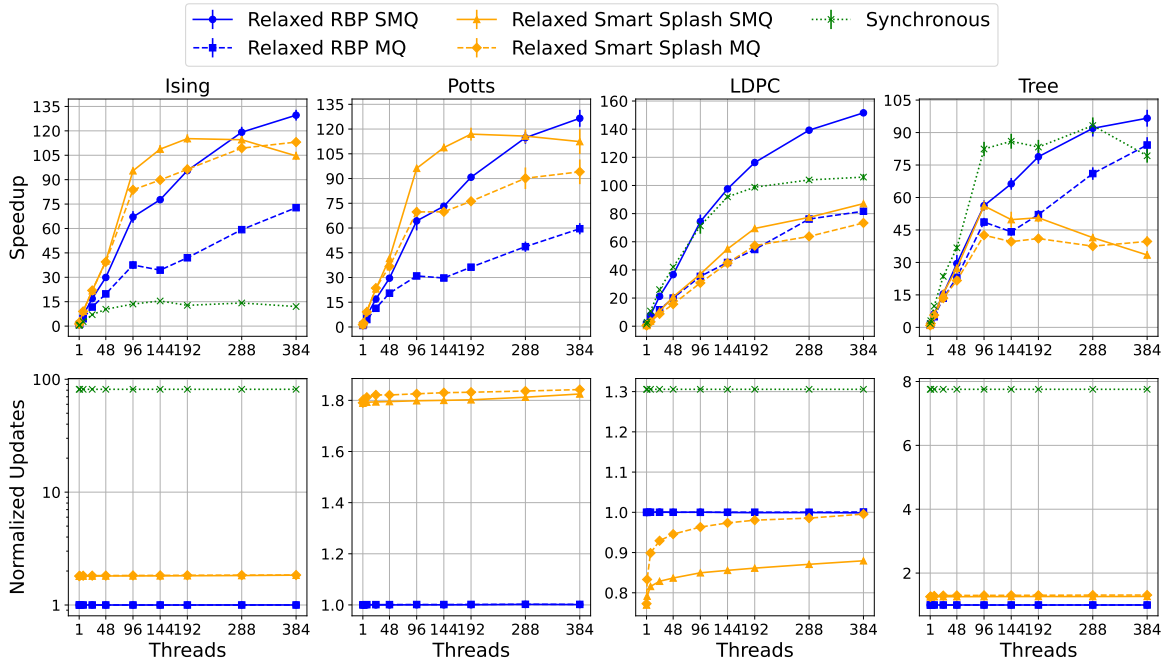


Figure 4: Average run-time speedup and number of updates, both normalized to sequential RBP across inputs with increasing thread count. Higher speedup and lower updates are better. Synchronous does not converge for Potts at any thread count.

across inputs, between synchronous and asynchronous algorithms, our SMQ relaxed RBP outperforms all measured alternatives on all input MRFs at 384 threads. By introducing the SMQ, we achieve top performance using relaxed RBP across all inputs.

The bottom row of Figure 4 illustrates work efficiency of the algorithms. We measure work efficiency, also known as the convergence rate, as the number of message updates, normalized to sequential RBP. Closer to one is better. Relaxed RBP algorithms show comparable update counts relative to sequential RBP. Our SMQ versions have slightly fewer additional updates than the MQ versions. RBP has fewer updates than RSS on all MRFs except LDPC, as LDPC's bipartite graph structure appears to benefit from the splash operation of RSS. RBP thus improves work efficiency, where up to 81× fewer updates are needed for convergence compared to synchronous BP, which is one contributing factor towards the performance advantage of the asynchronous BP over synchronous. Aksenov et al. observed similar update counts. These results demonstrate that our SMQ relaxed BP significantly reduces convergence time over MQ and scales well up to 384 threads.

Figure 5 provides further insight into the measured benefits of SMQ-based relaxed RBP by revealing how threads spend their time at 384 threads. We measure run time per thread across inputs, normalized to the slowest asynchronous algorithm for a given input. We break down convergence time into four categories: *(i)* message synchronization (acquiring and releasing locks per message to ensure atomic message updates), *(ii)* popping from queues, *(iii)* pushing to queues, *(iv)* and the core computational work of the BP algorithm. SMQ relaxed RBP cuts the time spent on queue pop operations by geomean 5× compared to MQ relaxed RBP. Time saved on queue push operations also contributes to the speedup

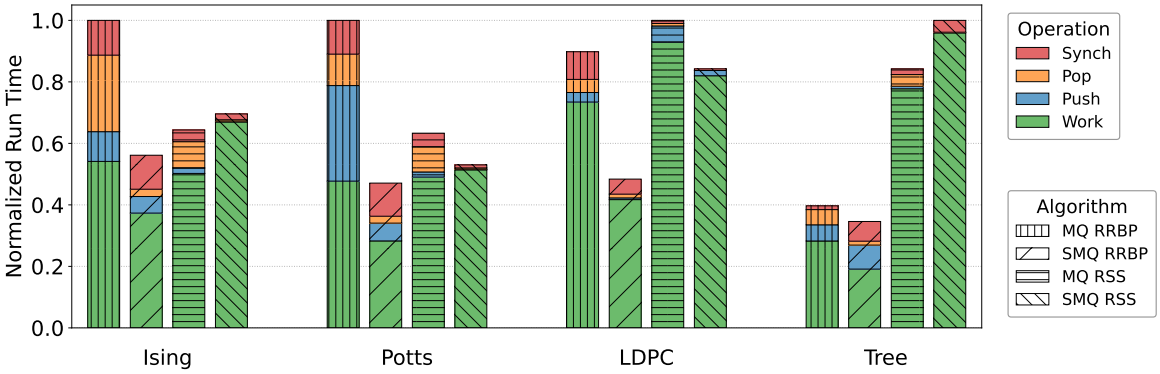


Figure 5: Breakdown of time spent per task thread across inputs for MQ- versus SMQ-based relaxed RBP and relaxed smart splash at 384 threads. Lower run time is better.

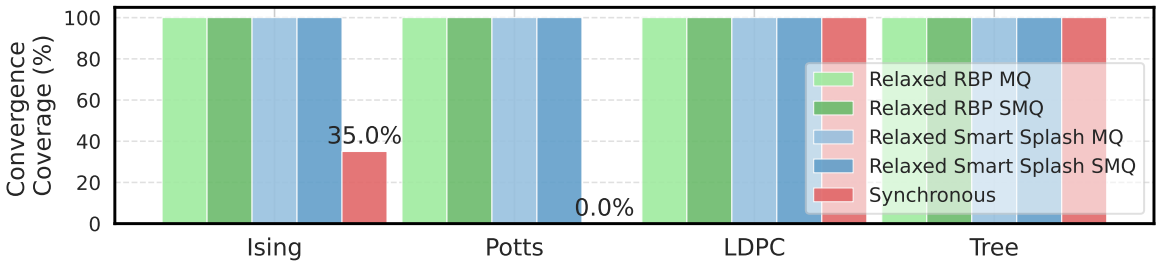


Figure 6: Convergence coverage for relaxed algorithms across MRFs. Synchronous BP does not converge for the majority of Ising MRFs and does not converge on any Potts MRFs.

but is less significant. SMQ is able to save time on queue operations because the queues are thread-local, which means that they do not require synchronization and that the upper levels of their heaps fit in the processor caches.

We also observe two key differences between RBP and RSS. RBP does significantly less work than RSS, but spends more time on message synchronization. RSS work includes a BFS rooted at the selected node, which does not require synchronization. This BFS contributes extra work to RSS that RBP does not do, but also implies that RSS threads spend a smaller portion of their time in critical sections, resulting in less contended locks and faster synchronization compared to RBP.

4.3. Convergence Analysis

Alongside the performance scaling benefits of introducing the SMQ into relaxed BP algorithms, we also confirm that there are no negative impacts of this priority scheduler upon the convergence coverage. For each BP algorithm, we generate varying input MRFs by sweeping seeds 0-99 and track which seeds lead to convergence. Figure 6 shows the percentage of seeds by algorithm and input MRF. Prior work finds that the synchronous algorithm fails to converge on the Potts model (Aksenov et al., 2020). Our results match this finding. We also find that synchronous BP does not converge on all seeds for the Ising model,

whereas all of the relaxed algorithms do converge. All the graph inputs that converge under residual belief propagation using the MQ still converge with the SMQ.

5. Related Work

Prior work has explored several approaches to parallelize BP. Residual splash BP (Gonzalez et al., 2009) combines the splash operation with residual BP, which achieves a speedup over residual BP by spreading data beyond neighboring nodes of the graph. However, Aksenov et al. show that the performance and work efficiency of residual splash BP is not competitive with other algorithms. They propose smart splash, which reduces the number of updates compared to splash, but still performs more work than relaxed residual BP. Prioritized Block Scheduling (Yin and Gao, 2014) reduces queue overheads by selecting entire blocks of nodes to be updated, but this approach targets clusters and Aksenov et al. show that relaxed RBP outperforms it on a single machine.

Although relaxed RBP is relatively fast and work efficient, it can perform poorly on certain graph inputs. A key limitation is that RBP may not converge when message values oscillate. Weight-Decay RBP (Knoll et al., 2015) reduces residuals by a damping factor to improve convergence. RBP may also fail to converge or converge incorrectly for graphs with loops. Kirkley et al. (2021) modify the message-passing equations and use neighborhoods of nodes to handle loops.

Recent BP implementations for GPUs have been developed to create algorithms that allow for more parallelism (Hou et al., 2020; Shan and Fang, 2020). Tile-based belief propagation (Liang et al., 2011) reduces BP's memory capacity and bandwidth costs for GPUs and ASICs by organizing the MRF into tiles and only persisting cross-tile messages. While the MQ and SMQ are not directly executable on GPUs, asynchronous priority schedulers have been implemented for GPUs (Wang et al., 2021; Al Khansa et al., 2026; Li, 2026; Hu et al., 2026) and could be applicable for RBP.

6. Conclusion

BP is among a class of irregular probabilistic algorithms that are challenging to parallelize for speedup but are highly relevant to AI and physics. We show that by combining developments in concurrent priority schedulers with improved BP algorithms, BP can be made faster while work efficient. Future work may extend our experiments to other architectures, such as GPUs or other CPUs, and compare against other distributed BP approaches.

To the best of our knowledge, this paper is the first to identify one relaxed BP algorithm (Stealing MultiQueue-based relaxed Residual BP) that outperforms other relaxed asynchronous and non-relaxed synchronous BP algorithms, across both cyclic and non-cyclic MRFs. We demonstrate that using an SMQ instead of an MQ along with a lazy updating strategy for relaxed algorithms leads to faster run times by up to 2.1× for relaxed RBP and up to 1.2× for relaxed smart splash compared to the same algorithms implemented with the MQ at 384 threads. Our SMQ is up to 81× and up to 44× more work efficient using relaxed RBP and relaxed smart splash respectively, compared to synchronous BP on certain inputs.

Acknowledgements

We thank Yue Fei for her help with implementing Synchronous BP. We thank Steven Hill, Aster Plotnik, Angus Wu, Guozheng Zhang, and the anonymous reviewers for their helpful feedback. Leo Han introduced us to the terms "convergence coverage" and "convergence rate." This work was supported in part by the Digital Research Alliance of Canada, the University of Toronto Connaught Fund, NSERC, and the Ontario Graduate Scholarship program.

References

Vitaly Aksenov, Dan Alistarh, and Janne H. Korhonen. Relaxed scheduling for scalable belief propagation. In *Proc. International Conference on Neural Information Processing Systems*, NeurIPS, 2020.

Hadi Al Khansa, Juan Gómez Luna, Amer Mouawad, and Izzat El Hajj. Parallel bidirectional A* search for GPU-accelerated pathfinding. In *Proc. International Conference on Supercomputing*, ICS, page 382–394. ACM, 2026. doi: 10.1145/3797905.3805620.

Dan Alistarh, Justin Kopinsky, Jerry Li, and Giorgi Nadiradze. The power of choice in priority scheduling. In *Proc. Symposium on Principles of Distributed Computing*, PODC, page 283–292. ACM, 2017. doi: 10.1145/3087801.3087810.

Rajeev Balasubramonian, Norman P. Jouppi, and Naveen Muralimanohar. *Multi-Core Cache Hierarchies*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2011. doi: 10.2200/S00365ED1V01Y201105CAC017.

Guy E Blelloch and Bruce M Maggs. Parallel algorithms. *ACM Computing Surveys (CSUR)*, 28(1):51–54, 1996. doi: 10.1145/234313.234339.

Barry A Cipra. An introduction to the Ising model. *The American Mathematical Monthly*, 94(10):937–959, 1987. doi: 10.1080/00029890.1987.12000742.

Laxman Dhulipala, Guy Blelloch, and Julian Shun. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *Proc. Symposium on Parallelism in Algorithms and Architectures*, SPAA, pages 293–304. ACM, 2017. doi: 10.1145/3087556.3087580.

Gal Elidan, Ian McGraw, and Daphne Koller. Residual belief propagation: informed scheduling for asynchronous message passing. In *Proc. Conference on Uncertainty in Artificial Intelligence*, UAI, page 165–173, 2006.

ETSI. 5G; NR; Multiplexing and channel coding. Technical Report TS 38.212, European Telecommunications Standards Institute, 04 2025.

Brendan J Frey and David MacKay. A revolution: Belief propagation in graphs with cycles. In *Proc. International Conference on Neural Information Processing Systems*, volume 10, 1997. URL https://proceedings.neurips.cc/paper_files/paper/1997/file/0245952ecff55018e2a459517fdb40e3-Paper.pdf.

J. Goldberger and H. Kfir. Serial schedules for belief-propagation: Analysis of convergence time. *IEEE Transactions on Information Theory*, 54(3):1316–1319, 2008. doi: 10.1109/TIT.2007.915702.

Joseph Gonzalez, Yucheng Low, and Carlos Guestrin. Residual splash for optimally parallelizing belief propagation. In *Proc. International Conference on Artificial Intelligence and Statistics*, AISTATS, pages 177–184. PMLR, 2009. URL https://proceedings.mlr.press/v5/gonzalez09a.html.

Saied Hemati and Amir H. Banihashemi. Dynamics and performance analysis of analog iterative decoding for low-density parity-check (LDPC) codes. *IEEE Transactions on Communications*, 54(1):61–70, 2006. doi: 10.1109/TCOMM.2005.861668.

Junteng Hou, Chengxiang Si, Shupeng Wang, Guangjun Wu, and Lei Zhang. Parallel belief propagation optimized by coloring on GPUs. In *Proc. International Conference on Algorithms and Architectures for Parallel Processing*, ICA3PP, page 645–660, 2020. doi: 10.1007/978-3-030-60245-1_44.

Zhengding Hu, Jingwen Sun, Le Jiang, Yuhao Wang, Junqing Lin, Yi Zong, and Guangzhong Sun. Beyond a single queue: Multi-level-multi-queue as an effective design for SSSP problems on GPUs. *arXiv preprint*, 2026. doi: 10.48550/arXiv.2602.10080.

Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31:253–258, February 1925. doi: 10.1007/BF02980577.

Anxiao Andrew Jiang, Pulakesh Upadhyaya, Ying Wang, Krishna R. Narayanan, Hongchao Zhou, Jin Sima, and Jehoshua Bruck. Stopping set elimination for LDPC codes. In *Proc. Allerton Conference on Communication, Control, and Computing*, pages 700–707, 2017. doi: 10.1109/ALLERTON.2017.8262806.

Alec Kirkley, George T. Cantwell, and M. E. J. Newman. Belief propagation for networks with loops. *Science Advances*, 7(17), 2021. doi: 10.1126/sciadv.abf1211.

Christian Knoll, Michael Rath, Sebastian Tschiatschek, and Franz Pernkopf. Message scheduling methods for belief propagation. In *Proc. Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, ECML PKDD, 2015. doi: 10.1007/978-3-319-23525-7_18.

F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001. doi: 10.1109/18.910572.

Andrew Lenharth, Donald Nguyen, and Keshav Pingali. Priority queues are not good concurrent priority schedulers. In *Proc. European Conference on Parallel Processing*, Euro-Par, pages 209–221, 2015. doi: 10.1007/978-3-662-48096-0_17.

ZhongYang Li. A multi-queue design for high-performance GPU priority queues. In *Proc. of the International Parallel and Distributed Processing Symposium Workshops*, IPDPSW, pages 828–836. IEEE, 2026. doi: 10.1109/IPDPSW71298.2026.00138.

Chia-Kai Liang, Chao-Chung Cheng, Yen-Chieh Lai, Liang-Gee Chen, and Homer H. Chen. Hardware-efficient belief propagation. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(5):525–537, 2011. doi: 10.1109/TCSVT.2011.2125570.

David J. C. MacKay and Radford M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 32:1645–1646, 1996. doi: 10.1049/el:19961141.

Jacek Majewski, Hao Li, and Jurg Ott. The ising model in physics and statistical genetics. *American journal of human genetics*, 69:853–862, 2001. doi: doi.org/10.1086/323419.

Barry M. McCoy and Tai Tsun Wu. *The Two-Dimensional Ising Model*. Harvard University Press, 1973. doi: 10.4159/harvard.9780674180758.

NASA. Cascade Lake Processors, 2025. URL https://www.nas.nasa.gov/hecc/support/kb/cascade-lake-processors_579.html.

Donald Nguyen, Andrew Lenharth, and Keshav Pingali. A lightweight infrastructure for graph analytics. In *Proc. Symposium on Operating Systems Principles*, SOSP, pages 456–471. ACM, 2013. doi: 10.1145/2517349.2522739.

Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *Proc. of the Second National Conference on Artificial Intelligence*, AAAI, pages 133–136, 1982. URL https://cdn.aaai.org/AAAI/1982/AAAI82-032.pdf.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., 1988. ISBN 1558604790.

Gilead Posluns, Yan Zhu, Guowei Zhang, and Mark C. Jeffrey. A scalable architecture for reprioritizing ordered parallelism. In *Proc. International Symposium on Computer Architecture*, ISCA, page 437–453. ACM, 2022. doi: 10.1145/3470496.3527387.

Anastasiia Postnikova, Nikita Koval, Giorgi Nadiradze, and Dan Alistarh. Multi-queues can be state-of-the-art priority schedulers. In *Proc. Symposium on Principles and Practice of Parallel Programming*, PPoPP, pages 353–367. ACM, 2022. doi: 10.1145/3503221.3508432.

R. B. Potts. Some generalized order-disorder transformations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(1):106–109, 1952. doi: 10.1017/S0305004100027419.

Hamza Rihani, Peter Sanders, and Roman Dementiev. MultiQueues: Simple relaxed concurrent priority queues. In *Proc. Symposium on Parallelism in Algorithms and Architectures*, SPAA, page 80–82. ACM, 2015. doi: 10.1145/2755573.2755616.

Bowei Shan and Yong Fang. GPU accelerated parallel algorithm of sliding-window belief propagation for LDPC codes. *International Journal of Parallel Programming*, 48(3):566–579, 2020. doi: 10.1007/s10766-019-00632-3.

Yifei Shen, Wenqing Song, Yuqing Ren, Houren Ji, Xiaohu You, and Chuan Zhang. Enhanced belief propagation decoder for 5G polar codes with bit-flipping. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(5):901–905, 2020. doi: 10.1109/TCSII.2020.2984536.

Charles Sutton and Andrew McCallum. Improved dynamic schedules for belief propagation. In *Proc. Conference on Uncertainty in Artificial Intelligence*, UAI, pages 376–383, 2007.

Ami Tavory, Vladimir Dreizin, and Benjamin Kosnik. Priority-queue performance tests. In *Policy-Based Data Structures*, chapter 5.2.2. GNU Operating System, 2 2004. URL https://web.archive.org/web/20250725145230/https://gcc.gnu.org/onlinedocs/libstdc++/ext/pb_ds/pq_performance_tests.html.

Dirk Van Essendelft, Hayl Almolyki, Wei Shi, Terry Jordan, Mei-Yu Wang, and Wissam A. Saidi. Record acceleration of the two-dimensional Ising model using a high-performance wafer-scale engine. *Computer Physics Communications*, 315, October 2025. doi: 10.1016/j.cpc.2025.109734.

Jean Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21(4):309–315, 1978. doi: 10.1145/359460.359478.

Kai Wang, Don Fussell, and Calvin Lin. A fast work-efficient SSSP algorithm for GPUs. In *Proc. Symposium on Principles and Practice of Parallel Programming*, PPoPP, page 133–146. ACM, 2021. doi: 10.1145/3437801.3441605.

Chen Yanover and Yair Weiss. Approximate inference and protein-folding. In *Proc. International Conference on Neural Information Processing Systems*, NeurIPS, 2002. URL https://proceedings.neurips.cc/paper_files/paper/2002/file/100d9f30ca54b18d14821dc88fea0631-Paper.pdf.

Ahmad Yasin. A top-down method for performance analysis and counters architecture. In *Proc. International Symposium on Performance Analysis of Systems and Software*, ISPASS, pages 35–44. IEEE, 2014. doi: 10.1109/ISPASS.2014.6844459.

Serif Yesil, Azin Heidarshenas, Adam Morrison, and Josep Torrellas. Understanding priority-based scheduling of graph algorithms on a shared-memory platform. In *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*, SC. ACM, 2019. doi: 10.1145/3295500.3356160.

Jiangtao Yin and Lixin Gao. Scalable distributed belief propagation with prioritized block updates. In *Proc. International Conference on Conference on Information and Knowledge Management*, CIKM, pages 1209–1218. ACM, 2014. doi: 10.1145/2661829.2662081.

Guozheng Zhang, Gilead Posluns, and Mark C. Jeffrey. Multi bucket queues: Efficient concurrent priority scheduling. In *Proc. Symposium on Parallelism in Algorithms and Architectures*, SPAA, page 113–124. ACM, 2024. doi: 10.1145/3626183.3659962.