Message Scheduling for Performant, Many-Core Belief Propagation

Mark Van der Merwe, Vinu Joseph, Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, USA
mark.vandermerwe@utah.edu,{vinu,ganesh}@cs.utah.edu

***Abstract*—Belief Propagation (BP) is a message-passing algorithm for approximate inference over Probabilistic Graphical Models (PGMs), finding many applications such as computer vision, error-correcting codes, and protein-folding. While general, the convergence and speed of the algorithm has limited its practical use on difficult inference problems. As an algorithm that is highly amenable to parallelization, many-core Graphical Processing Units (GPUs) could significantly improve BP performance. Improving BP through many-core systems is non-trivial: the scheduling of messages in the algorithm strongly affects performance. We present a study of message scheduling for BP on GPUs. We demonstrate that BP exhibits a tradeoff between speed and convergence based on parallelism and show that existing message schedulings are not able to utilize this tradeoff. To this end, we present a novel randomized message scheduling approach, Randomized BP (RnBP), which outperforms existing methods on the GPU.**

***Index Terms*—General Purpose GPU Computing, Randomized Algorithms, Message-Passing Algorithms.**

I. Introduction

Probabilistic Graphical Models (PGMs) are powerful, general machine learning models that encode distributions over random variables. PGM Inference, in which we seek to compute some probabilistic beliefs within the system modeled by the PGM, is in general an intractable problem, leading to dependence on approximate algorithms. Belief Propagation (BP) is a widely employed approximate inference algorithms for PGMs [1]. BP has been successfully utilized in many areas, including computer vision [2], error-correcting codes [3], and protein-folding [4].

BP is a message-passing algorithm, in which messages are passed along edges of the PGM graph. While BP is exact on tree PGMs, it is approximate on general graphs containing loops, where iterative updates are applied until convergence. Like others [5], we break the performance of BP into two properties: convergence (for how many input graphs does it reach a convergent state) and speed (how long does it take to reach the convergent state). While BP has been shown to perform well on many graphs containing loops, there is no guarantee of convergence in most cases, and graphs of varying structure and parameterization can prevent BP from converging or can have slow speed for convergence [6].

General-Purpose GPU computing has begun recently exploring many-core parallelism for graph-based problems [7]. This, combined with the inherent parallelism available between message updates, suggests that many-core parallelism can be effectively applied to BP to yield good performance on the GPU (that is, good convergence and speed). In order to ensure good performance, one must be careful in the implementation such as to avoid the convergence and speed pitfalls inherently present in Belief Propagation.

In existing BP literature, there has been much interest in exploring the use of message schedulings for improving BP performance. The naive scheduling is known as Synchronous or Loopy BP (LBP), where all messages are updated in parallel [6]. Asynchronous approaches, where some amount of sequentiality is enforced during the message updates, for example via subgraph updates [8] or greedy message selection [5], [9], have been shown both empirically and theoretically to outperform LBP in single-core environments. The general intuition is that enforcing sequentialism in the scheduling encourages more direct propagation of information, thus converging faster. The contrast between LBP and Asynchronous BP introduces a parallelism vs. efficiency spectrum (also found in other graph problems such as SSSP [10]). LBP exposes high levels of parallelism but is work-inefficient. Asynchronous BP is efficient and convergent but exposes little parallelism. We hypothesize that there exists a tradeoff between the parallelism and sequentialism in Belief Propagation, and that GPUs can effectively harness that tradeoff to yield performant BP.

We start by presenting many-core frontier-based implementations for two greedy asynchronous message schedulings, Residual Belief Propagation [5] and Residual Splash [9]. We then benchmark the performance, varying parallelism to explore how parallelism affects the performance of BP. As expected, we find that as parallelism is increased, we see less convergence but obtain faster speed. As parallelism is decreased, we see more convergence but lower speeds. This is encouraging, as it means we can still get convergence boosts while exploiting parallelism, but we also see that existing approaches incur significant overheads, and performance is heavily tied to the choice of parallelism. To overcome these drawbacks, we propose a new message scheduling, called Randomized Belief Propagation (RnBP) which uses low-overhead, randomized scheduling, and outperforms existing approaches.

To summarize, our contributions are:

- Many-core, frontier-based implementations of two greedy asynchronous message schedulings, Residual Belief Prop-

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: The University of Utah. Downloaded on June 14,2020 at 22:27:25 UTC from IEEE Xplore. Restrictions apply.

agation [5] and Residual Splash [9].
- Demonstration of tradeoff between parallelism and sequentialism in terms of speed/convergence of BP.
- Demonstration that overheads prevent existing asynchronous message scheduling approaches from scaling to the GPU.
- A novel message-scheduling, Randomized BP (RnBP), that utilizes randomization to effectively handle the tradeoff BP demonstrates. We demonstrate speedups on Ising grid [5] and protein-folding [4] datasets.

II. Background

A. Belief Propagation

We focus our attention on the Sum-Product Belief Propagation algorithm over discrete pairwise Markov Random Fields (MRFs), though we expect the results to generalize to other variants of BP. Suppose we have the set of discrete random variables $X = \{X_1, ..., X_n\}$, each taking on a value $X_i \in A_i$, where A_i is a finite set. An MRF is an undirected graph $G = (V, E)$. Each vertex $i \in V$ represents a discrete random variable X_i. $\{\psi_i : A_i \to \mathbb{R}^+ | i \in V\}$ is set of unary potential functions for each random variable. Each edge $(i, j) \in E$ represents the probabilistic relationship between two variables. $\{\psi_{i,j} : A_i \times A_j \to \mathbb{R}^+ | (i, j) \in E\}$ is the set of binary potential functions for each edge. An MRF yields the following joint distribution over X:

$$P(x_1, ..., x_N) \propto \prod_{i \in V} \psi_i(x_i) \prod_{\{i,j\} \in E} \psi_{i,j}(x_i, x_j) \quad (1)$$

The goal of inference is to derive the vertices's marginal distributions $P(x_i)$. This is intractable in general, however BP can be used to find exact marginals (for trees) or approximate marginals (for graphs containing loops). This is done through the iterative passing of messages along the edges of the graph. Each edge $(i, j) \in E$ has two messages $m_{i \to j}, m_{j \to i}$ being passed along it, indicating each vertex's belief about the other's state. The message $m_{i \to j}$ is a distribution, updated as follows:

$$m_{i \to j}(x_j) \propto \sum_{x_i \in A_i} \psi_{i,j}(x_i, x_j) \psi_i(x_i) \prod_{k \in \Gamma_i \setminus j} m_{k \to i}(x_i) \quad (2)$$

where Γ_i indicates the neighbors of v_i. Each message is initialized to the uniform distribution and normalized between updates. Messages are iterated until ϵ convergence, at which point we calculate the beliefs at each vertex:

$$P(X_i = x_i) \approx b_i(x_i) \propto \psi_i(x_i) \prod_{k \in \Gamma_i} m_{k \to i}(x_i) \quad (3)$$

B. Message Scheduling for Belief Propagation

BP message schedulings differ by the messages that are updated each iteration. LBP simply updates every edge, every iteration in parallel. That is, all messages are updated using the previous iteration's messages. LBP performance has been examined both empirically [6] and theoretically [11].

Asynchronous approaches enforce sequentialism in message updates, updating each message using the most recent messages. That is, a single message is updated, and that update is immediately used to update other messages. If we assume LBP to be a max-norm contraction, ABP has at least as good convergence rate guarantees as LBP [5].

Both [5], [9] build on ABP using *greedy* update schemes. Residual Belief Propagation (RBP) [5] introduces the *residual*, simply defined as:

$$r(m_i^t) = ||f_{BP}(m^t)_i - m_i^t|| \quad (4)$$

RBP then selects the next message to update asynchronously based on the highest residual. Intuitively, the program focuses its computational effort to parts of the graph where it moves closer to a converged state.

Residual Splash (RS) [9] is an extension of RBP for multicore parallelization. They extend residuals to vertices, where the residual of a vertex is the maximum residual of incoming messages. Similar to RBP, vertices are selected greedily, however, in RS, a splash, or BFS search of depth h around the vertex, is performed with updates moving sequentially through the BFS tree. RS demonstrates linear speedup in the number of cores. In this paper we explore LBP, RBP, and RS because of their simplicity and good performance in existing work.

C. Related Work

BP has been implemented on the GPU for specific BP workloads, including stereo matching [12], [13] and error correcting codes [14]. Several works specifically explore memory usage, as the unique architecture of the GPU closely ties memory use and performance. Grauer et al. [15] explores using registers, shared memory, and local memory for Belief Propagation and their effect on GPU occupancy for the stereo matching problem. Liang et al. [16] shows a general approach for reducing memory usage for BP by storing only the messages along the edges of partitions of the graphs, allowing messages to be stored in faster shared memory. While we do not explore memory use, our message scheduling work combines naturally with the memory work of both of these approaches.

Several works explore different message schedulings on the GPU for specific BP applications. Yang et al. [17] filters messages to be updated by removing any messages that have already converged. We employ the same filter as one of the filters in our final RnBP scheduling approach. Xiang et al. [18] changes BP on a grid-based stereo problem by using directional updates, that is, messages are updated along dimensions of the grid. Of course, this directional update is specific to grid-based models such as ones used in computer vision. Romero et al. [3] constructed an LDPC code structure in such a fashion that the updates could be partitioned so many could be completed in parallel while still maintaining sequentiality overall. In general, we cannot control the problem as in their case, and creating effective message partitions are problem-specific and non-trivial. Our work takes a general approach that can apply to *any* BP problem, and explore message schedulings that have not yet been explored on the GPU, to the authors's best knowledge.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: The University of Utah. Downloaded on June 14,2020 at 22:27:25 UTC from IEEE Xplore. Restrictions apply.

III. Frontier-Based Belief Propagation on the GPU

We present all algorithms examined as realizations of a frontier-based BP framework. In this section, we implement several existing schedulings and benchmark their performance. In the next section, we introduce our own GPU-centric scheduling approach, Randomized Belief Propagation.

To transfer the schedulings onto the synchronous, many-core architecture of the GPU, we utilize a data-centric, frontier-based parallelization framework [7], [19]. We consider the frontier to be the set of messages selected to be updated synchronously and in parallel each iteration. Message schedulings differ on selection of the frontier, but follow the same general structure presented in Algorithm 1.

Algorithm 1 Frontier-Based Belief Propagation

Input: pgm, ϵ
Output: $marginals$
1: $converged \leftarrow False$
2: **while** !$converged$ **do**
3: $\quad frontier \leftarrow$ GenerateFrontier(pgm)
4: $\quad$ Update$(frontier, pgm)$
5: $\quad converged \leftarrow$ IsConverged(pgm, ϵ)
6: **end while**
7: **return** Marginals(pgm)

A. Greedy Update Frontier Selection

We use this frontier-based approach to implement several existing schedulings on the GPU, specifically LBP, RBP, and RS. LBP is simple to implement in this framework: every iteration, all the messages are put in the frontier to be updated. RBP and RS rely on greedily selecting updates based on message residuals. In order to explore the trade-off between parallelism and greedy sequentialism, we will simply adjust the greedy approach to select multiple elements as a frontier per iteration as opposed to a single element. We can consider this to be the selection of the top-k values for update each iteration. Adjusting k allows us to adjust parallelism.

For single-core implementations, the primary data structure employed to perform these greedy updates is a Priority Queue. While concurrent priority queues have been developed, they rely on mutual exclusion, and thus are best suited for asynchronous environments, unlike the GPU. Other work in using GPUs for algorithms with Priority Queue based methods have turned to other approaches, involving sort-and-select, binning, or problem division [10], [20]. Several algorithms for direct top-k GPU selection exist [21], [22], but speedup only occurs for very large problem sizes. We choose to use a simple sort-and-select approach in order to select the top k elements.

We now present the high level approach for our bulk-parallel greedy update selection. We maintain the residuals of either the messages for RBP or the vertices for RS. Each iteration, we perform a key-value sort of the residuals with their corresponding vertices/edges. The top k elements after the sort form the update frontier. RBP updates this frontier directly, RS updates the splash around the selected nodes. A single update is visualized in Fig.1.

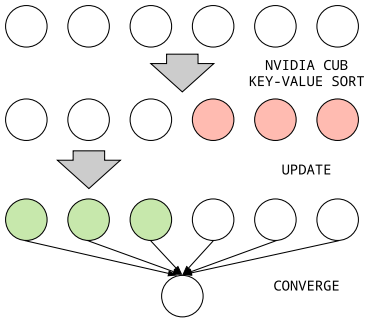


Fig. 1. A single bulk-parallel greedy update iteration based on a sort-and-select approach. Each circle represents a message (RBP) or vertex (RS): Green indicates that message/vertex was updated; red indicates that message/vertex has been pruned for this round. Each iteration sorts-and-selects based on the residual, then updates the top-k messages/vertices.

B. Implementation

We implement LBP, RBP, and RS using Nvidia's CUDA library [23]. We use a simple adjacency list format for storing graph structure and parameterization. Each edge and vertex is assigned IDs, and for parallel operations, each thread is assigned a subset of the IDs to update. We use the CUDA occupancy API for kernel launch settings and Nvidia's CUB library Radix Sort for the sort operation [24]. We implement serial RBP (SRBP) as a performance benchmark. We use the same adjacency list format and use the Boost library's Fibonacci heap for the Priority Queue.

C. Benchmarks

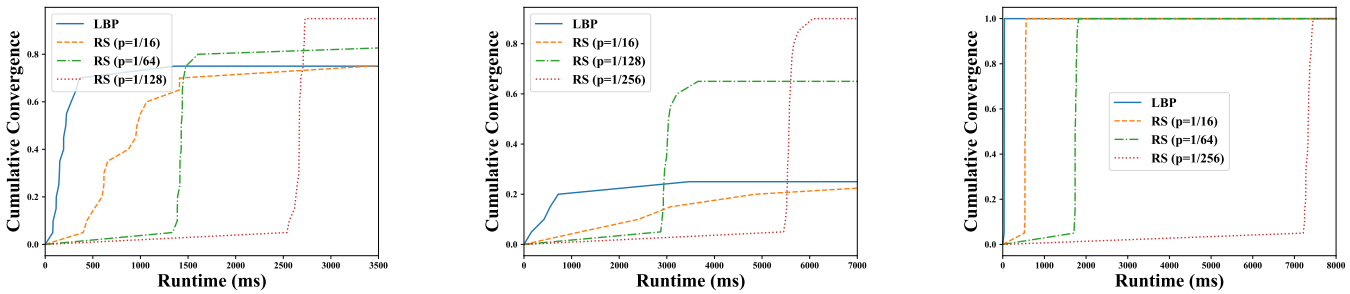
To accurately benchmark performance, we would like to be able to adjust the difficulty of the inference problem. A synthetic benchmark that gives us control over difficulty is the Ising dataset, a standard benchmark for message propagation algorithms [5]. Ising grids are $N \times N$ grids of binary variables. Univariate potentials ψ_i are randomly sampled from the [0,1] range. The pairwise potentials $\psi_{i,j}$ are set to $e^{\lambda C}$ when $x_i = x_j$ and $e^{-\lambda C}$ otherwise. λ is sampled from [-0.5,0.5] to make certain potentials favor agreement while others favor disagreement. Varying C changes the difficulty of the inference problem (higher C being more difficult). For RBP and RS, we test on Ising grids of size 100×100 and 200×200, with $C = 2.5$. We also run on simpler chain graphs, where N binary variables are formatted in a single long chain. Of course, when a graph is a chain, BP is guaranteed to converge. We sample ψ_i and $\psi_{i,j}$ in the same manner used for our Ising grids. For RBP and RS, we test on chain graphs of size 100000, with $C = 10$.

D. Performance

In order to examine parallelism's effect on performance, we introduce a multiplier p, where the frontier size each round is $p \cdot 2|E|$. Varying p thus varies the parallelism used. For RS,

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: The University of Utah. Downloaded on June 14,2020 at 22:27:25 UTC from IEEE Xplore. Restrictions apply.



(a) Ising 100×100, $C = 2.5$ (b) Ising 200×200, $C = 2.5$ (c) Chain 100000, $C = 10$

Fig. 2. GPU RS cumulative percentage of converged runs for our 2 Ising and 1 Chain dataset, compared to LBP. We see that lower p causes more convergence, at the cost of speed.

TABLE I
GPU RBP SPEEDUPS OVER SRBP

Dataset	Settings	SRBP Speedup
Ising 100×100, $C = 2.5$	$p = 1/256$	3.47x
Ising 200×200, $C = 2.5$	$p = 1/256$	> 4.54x
Chain 100000, $C = 10$	$p = 1/16$	> 72.31x

TABLE II
GPU RS SPEEDUPS OVER SRBP

Dataset	Settings	SRBP Speedup
Ising 100×100, $C = 2.5$	$p = 1/128$	25.85x
Ising 200×200, $C = 2.5$	$p = 1/256$	> 16.08x
Chain 100000, $C = 10$	$p = 1/16$	> 166.51x

we lock[1] splash depth to be $h = 2$. We time how long it takes the message updates to converge. Our GPU code is run on a single NVidia Tesla V100 and our CPU code is run on Intel Xeon Processors.

Fig.2 shows GPU RS performance on our three benchmarks as cumulative convergence graphs, indicating the cumulative percentage of the set of input graphs that have converged as a function of time. GPU RBP exhibits the same patterns on each dataset and thus is not shown for brevity.

Our results indicate that a tradeoff does indeed exist between parallelism and sequentialism. Specifically, we see that as we decrease p, that is, we reduce our parallelism, more graphs converge, but they take longer to do so. Thus, low parallelism encourages convergence, while high parallelism encourages speed. LBP, with full parallelism, demonstrates only partial convergence, while RS is able to extend convergence, given time, by reducing parallelism (Fig.2a,2b).

In Tables I and II, we show the speedup results comparing GPU RBP and RS to SRBP. We compare with the fastest setting in our test runs that converges on all or most of the graphs, indicated for each dataset. For cases where SRBP convergence did not occur (i.e., SRBP failed to converge on all but the Ising 100×100, $C = 2.5$ dataset), we provide a conservative lower-bound on speedup based on how long we gave SRBP to run (90 seconds). We see that RS outperforms RBP and both outperform SRBP.

There are two primary shortcomings to RBP and RS. First, performance relies heavily upon p, and effective p selection is non-trivial. Second, the sort-and-select approach incurs significant overhead. This is best demonstrated by the easy chain dataset (Fig.2c) where RS takes significantly longer than LBP, which converges very quickly. Profiling indicates that on many graphs, both RBP and RS spend more than 90% of runtime during the sort-and-select step, up to 98% for certain runs.

[1]Exploration of different splash depths could be interesting, though we change our focus to randomized updates, and thus do not pursue this further.

IV. RANDOMIZED BELIEF PROPAGATION

To overcome the shortcomings of existing approaches on the GPU and exploit the tradeoff we have demonstrated, we present our novel, low-overhead, randomized message scheduling technique for Belief Propagation on the GPU, Randomized Belief Propagation (RnBP).

A. Algorithm

We hypothesize that varying the parallelism affects performance more than the specific selection of messages each round when in a many-core environment. We thus perform *random* k selection as opposed to exact top-k selection.

In creating our message frontier, we employ two filters. In order to encourage selection to be similar to the top-k, we only choose the messages to update from those whose residual is above the ϵ thresholds. Thus, our first filter prunes all messages whose next update will move them less than ϵ.

The second filter is our randomized filter. We randomly select some percentage p of the remaining messages to be updated. Adjusting p thus allows us to adjust the parallelism for that round. A single update is visualized in Fig.3.

Finally, we dynamically range p based on the convergence of the run. Throughout the run, we can track how many of the edges have not converged. The ratio between the number of edges not converged between each iteration becomes an indicator of runtime convergence performance: $EdgeRatio = NewEdgeCount/OldEdgeCount$. If $EdgeRatio$ is low, it is indicative of good convergence, if $EdgeRatio$ is high, it is indicative of bad convergence. We introduce two p settings,

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: The University of Utah. Downloaded on June 14,2020 at 22:27:25 UTC from IEEE Xplore. Restrictions apply.

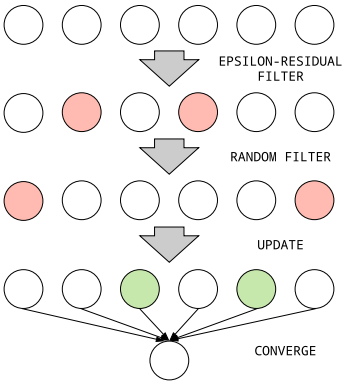


Fig. 3. A single RnBP update iteration. Each circle represents a message: green indicates the message was updated; red indicates the message has been pruned for this round. We apply two filters to select the frontier: prune all messages with residual $< \epsilon$ and then we randomly select p of the remaining messages.

one high and one low. We know from our results in Section III that low parallelism encourages convergence and high parallelism encourages speed. Thus, if $EdgeRatio > 0.9$, we use the lower parallelism setting, thus encouraging *convergence*. Otherwise, we use the higher parallelism setting, thus encouraging *speed*. We note, overhead prevented similar dynamic p selection from aiding GPU RBP/RS.

B. Implementation

We implement RnBP in CUDA, using the same data structures, occupancy approach, and parallel operation strategy as described in Section III-B. For the counting of messages above ϵ, we use Nvidia's CUB library reductions [24]. We use Nvidia's cuRAND for randomized update selection [25].

C. Benchmarks

We use the same chain and Ising grid benchmarks described in III-C. We test with Ising grids of size 100×100 with $C = 2, 2.5, 3$ and of size 200×200 with $C = 2.5$. For chain graphs, we test with size 100000 with $C = 10$.

D. Performance

Again, our GPU code is run on a single NVidia Tesla V100 and our CPU code is run on Intel Xeon Processors. We continue to compare to LBP and SRBP.

As for RBP and RS, we can vary our high and low parallelism settings to get different parallelisms during run time. We found that for our synthetic dataset the high parallelism setting mattered less than the low parallelism setting. As such, we locked our high parallelism to be a full update, thus whenever $EdgeRatio < 0.9$, we update the full message frontier update. We show performance on all datasets with low parallelism ($LowP$) being set to 0.7, 0.4, and 0.1.

Fig.4a-4e shows GPU RnBP performance on our benchmarks as cumulative convergence graphs. For easy graph datasets, where LBP converges for most or all, we notice that RnBP with higher parallelism settings (i.e., $LowP = 0.7, 0.4$) nearly matches LBP performance (see Fig.4a,4d). This shows the value in RnBP's lack of overhead. As the graphs become more difficult, where LBP only converges on some, we see that RnBP continues to converge quickly on *all* graphs (see Fig.4b, 4e). RnBP converges with much higher parallelism than that required for RS and RBP. Using the higher parallelism settings allows speed paired with convergence. We see that this allows for RnBP to actually provide speedups *over* GPU LBP runtimes (Fig.4b,4e), averaging 9x speedups on the Ising 200×200, C=2.5 dataset.

Notice, LBP fails to converge on any graphs for the difficult 100×100, C=3 dataset. We see that we can effectively drop parallelism in RnBP, however, to encourage convergence (see Fig.4c). We do so without significant overheads yielding dramatic slow downs. This convergence behavior applies to larger and more difficult graphs than the ones RBP and RS could handle. RnBP thus *extends the classes of Belief Propagation problems for which GPU speedups can be applied*. We note that for the difficult dataset, RnBP can still be sensitive to the selected parallelism. However, on all our other datasets, RnBP is fairly robust to parallelism selection. Thus, while not completely solved, RnBP is a considerable improvement to existing approaches.

We characterize the speedup of RnBP over SRBP in Table III. Again, we compare with the fastest setting in our test runs that converges on all or most of the graphs, indicated for each dataset, and present conservative lower bounds when SRBP failed to converge (given 90 seconds).

TABLE III
GPU RNBP SPEEDUPS OVER SRBP

Dataset	Settings	SRBP Speedup
Ising 100×100, $C = 2$	$LowP = 0.7$	2203.58x
Ising 100×100, $C = 2.5$	$LowP = 0.7$	1135.05x
Ising 100×100, $C = 3$	$LowP = 0.1$	61.28x
Ising 200×200, $C = 2.5$	$LowP = 0.7$	$>$ 529.997x
Chain 100000, $C = 10$	$LowP = 0.7$	$>$ 1676.92x

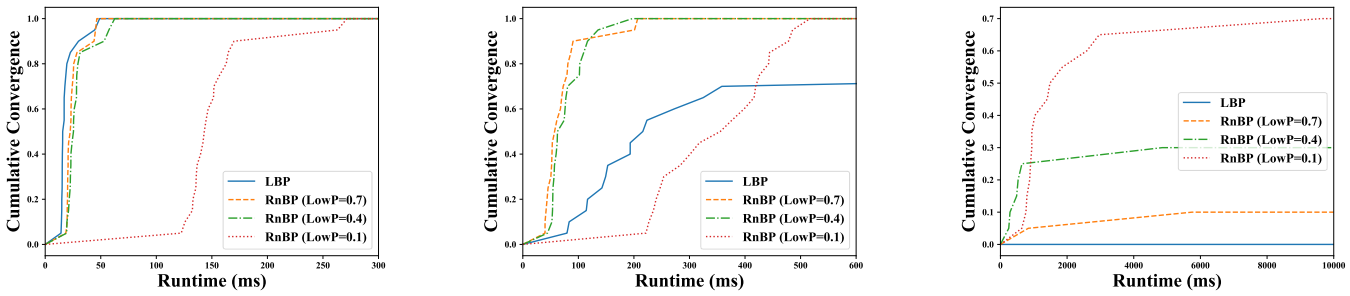
E. Additional Tests

As RnBP is a novel message scheduling, we provide several additional tests to examine performance. To test correctness, we created a smaller Ising dataset, size 10×10, $C = 2$, for which exact inference is tractable. We use Variable Elimination to find the exact marginal values, then determine the KL-divergence between the exact results and the results of both SRBP and RnBP (run with $LowP = 0.7$). These are shown in figure 5. We see that RnBP achieves the same quality of result as compared to SRBP.

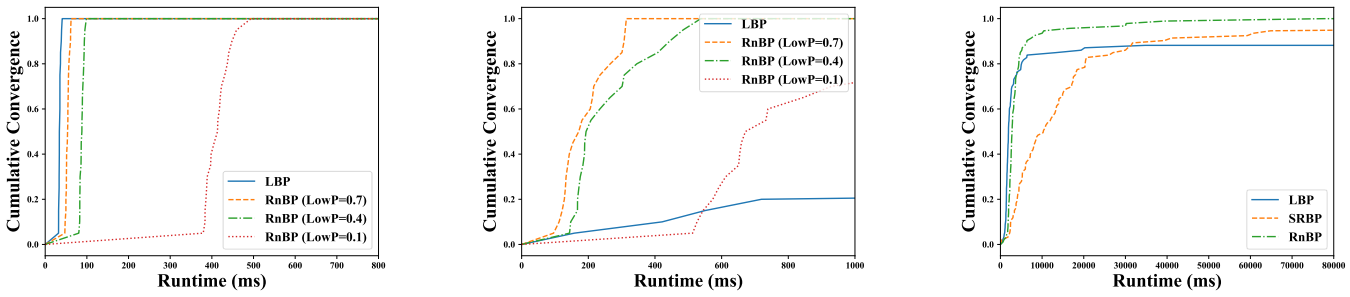
We tested RnBP on a real-world dataset, specifically a protein-folding dataset [4]. This dataset contains graphs with vertices representing amino acid units and the setting at each vertex representing the side-chain configuration. The possible settings at each vertex ranges from 2 to 81 and the graph structure is highly irregular. The cumulative convergence is shown in Fig.4f (We run RnBP with $LowP = 0.4, HighP = 0.9$). Despite the different structure as compared to our synthetic

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: The University of Utah. Downloaded on June 14,2020 at 22:27:25 UTC from IEEE Xplore. Restrictions apply.



(a) Ising 100×100, $C = 2$ (b) Ising 100×100, $C = 2.5$ (c) Ising 100×100, $C = 3$



(d) Chain 100000, $C = 10$ (e) Ising 200×200, $C = 2.5$ (f) Protein-Folding Dataset [4]

Fig. 4. GPU RnBP cumulative percentage of converged runs for 5 Ising, 1 Chain, and 1 Protein-Folding datasets, compared to GPU LBP.

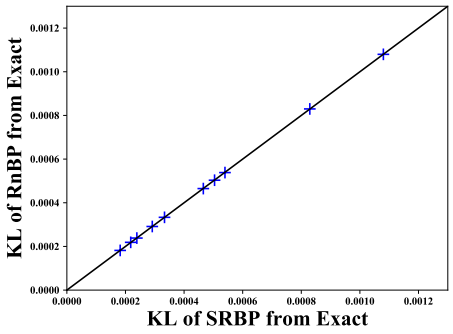


Fig. 5. Correctness of the converged marginals for GPU RnBP and SRBP as compared to the exact marginals for Ising 10×10, $C = 2$ dataset.

dataset and without any finetuning to handle load-imbalanced message updates, we see that RnBP yields fast, convergent performance. Given 3 minutes per graph, RnBP was the only approach to converge on all graphs and yielded an average of 4.4x speedup over SRBP when SRBP converged.

V. Conclusions and Future Work

In this work, we presented a study of message scheduling approaches for BP on many-core GPU systems (summarized in Table IV). We hypothesized the existence of a tradeoff between parallelism and sequentialism for BP speed and convergence, and that GPUs could be used to exploit that tradeoff for performant BP. We presented many-core, frontier-based implementations for two asynchronous message schedulings, RBP [5] and RS [9], and showed empirically that indeed a tradeoff exists. Specifically, lower parallelism encourages convergence, while higher parallelism encourages speed. We also show that these approaches incur significant overhead, suggesting that a new GPU-centric approach is needed. In this direction, we presented a novel message scheduling we call Randomized Belief Propagation (RnBP), which utilizes randomization to select frontiers for updating. We demonstrate that this approach yields higher convergence while maintaining speed, providing speedups over serial and existing GPU methods on both synthetic and real-world datasets. Our implementation is available online[2].

TABLE IV
Algorithms Explored (Bold indicates contribution)

Algorithm	Frontier Selection	Many-Core
GPU LBP	All Messages	✓
Serial RBP/RS	Priority Queue	X
GPU RBP/RS	Sort-and-Select	✓
GPU RnBP	Randomized	✓

Our approach is a general solution for BP, meaning it can be integrated naturally with many variants of BP [1], [2] as well as with GPU memory improvements [15], [16], to further extend performance.

We think that this work could be of interest to other algorithms that utilize iterative, convergent graph updates, such as the Generalized Distributive Law [26], of which BP is a subclass, and Graph Neural Networks [27].

[2]https://github.com/mvandermerwe/BP-GPU-Message-Scheduling

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: The University of Utah. Downloaded on June 14,2020 at 22:27:25 UTC from IEEE Xplore. Restrictions apply.

ACKNOWLEDGMENT

This work was supported in part by NSF awards 1704715 and 1817073.

REFERENCES

[1] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Generalized belief propagation," in *Advances in Neural Information Processing Systems*, 2001, pp. 689–695.

[2] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *International Journal of Computer Vision*, vol. 70, no. 1, pp. 41–54, 2006.

[3] D. L. Romero and N. B. Chang, "Sequential decoding of non-binary ldpc codes on graphics processing units," in *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*. IEEE, 2012, pp. 1267–1271.

[4] C. Yanover and Y. Weiss, "Approximate inference and protein-folding," in *Advances in Neural Information Processing Systems*, 2003, pp. 1481–1488.

[5] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," in *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2006, pp. 165–173.

[6] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proceedings of the Fifteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1999, pp. 467–475.

[7] Y. Wang, Y. Pan, A. Davidson, Y. Wu, C. Yang, L. Wang, M. Osama, C. Yuan, W. Liu, A. T. Riffel *et al.*, "Gunrock: Gpu graph analytics," *ACM Transactions on Parallel Computing (TOPC)*, vol. 4, no. 1, p. 3, 2017.

[8] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "Tree-based reparameterization framework for analysis of sum-product and related algorithms," *IEEE Transactions on Information Theory*, vol. 49, no. 5, pp. 1120–1146, 2003.

[9] J. Gonzalez, Y. Low, and C. Guestrin, "Residual splash for optimally parallelizing belief propagation," in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, D. van Dyk and M. Welling, Eds., vol. 5. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, 16–18 Apr 2009, pp. 177–184. [Online]. Available: http://proceedings.mlr.press/v5/gonzalez09a.html

[10] A. Davidson, S. Baxter, M. Garland, and J. D. Owens, "Work-efficient parallel gpu methods for single-source shortest paths," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 349–359.

[11] J. M. Mooij and H. J. Kappen, "Sufficient conditions for convergence of the sum–product algorithm," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4422–4437, 2007.

[12] S. Grauer-Gray, C. Kambhamettu, and K. Palaniappan, "Gpu implementation of belief propagation using cuda for cloud tracking and reconstruction," in *Pattern Recognition in Remote Sensing (PRRS 2008), 2008 IAPR Workshop on*. IEEE, 2008, pp. 1–4.

[13] A. Brunton, C. Shu, and G. Roth, "Belief propagation on the gpu for stereo vision," in *Computer and Robot Vision, 2006. The 3rd Canadian Conference on*. IEEE, 2006, pp. 76–76.

[14] N. Chandrachoodan *et al.*, "A gpu implementation of belief propagation decoder for polar codes," in *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*. IEEE, 2012, pp. 1272–1276.

[15] S. Grauer-Gray and J. Cavazos, "Optimizing and auto-tuning belief propagation on the gpu," in *International Workshop on Languages and Compilers for Parallel Computing*. Springer, 2010, pp. 121–135.

[16] C.-K. Liang, C.-C. Cheng, Y.-C. Lai, L.-G. Chen, and H. H. Chen, "Hardware-efficient belief propagation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 5, pp. 525–537, 2011.

[17] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister, "Real-time global stereo matching using hierarchical belief propagation." in *BMVC*, vol. 6, 2006, pp. 989–998.

[18] X. Xiang, M. Zhang, G. Li, Y. He, and Z. Pan, "Real-time stereo matching based on fast belief propagation," *Machine Vision and Applications*, vol. 23, no. 6, pp. 1219–1227, 2012.

[19] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, M. Méndez-Lojo *et al.*, "The tao of parallelism in algorithms," in *ACM Sigplan Notices*, vol. 46, no. 6. ACM, 2011, pp. 12–25.

[20] H. Park and P. A. Fishwick, "A gpu-based application framework supporting fast discrete-event simulation," *Simulation*, vol. 86, no. 10, pp. 613–628, 2010.

[21] T. Alabi, J. D. Blanchard, B. Gordon, and R. Steinbach, "Fast k-selection algorithms for graphics processing units," *Journal of Experimental Algorithmics (JEA)*, vol. 17, pp. 4–2, 2012.

[22] L. Monroe, J. Wendelberger, and S. Michalak, "Randomized selection on the gpu," in *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. ACM, 2011, pp. 89–98.

[23] Nvidia, "Cuda programming guide," 2010.

[24] D. Merrill and Nvidia-Labs, "Cuda unbound (cub) library," 2015.

[25] Nvidia, "Curand library," 2010.

[26] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 325–343, 2000.

[27] R. Liao, M. Brockschmidt, D. Tarlow, A. L. Gaunt, R. Urtasun, and R. Zemel, "Graph partition neural networks for semi-supervised classification," *arXiv preprint arXiv:1803.06272*, 2018.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: The University of Utah. Downloaded on June 14,2020 at 22:27:25 UTC from IEEE Xplore. Restrictions apply.