# A Resource-Aware Residual-Based Gaussian Belief Propagation Accelerator Toolflow

Omar Sharif
*Department of Electrical and Electronic Engineering*
*Imperial College London*
London, United Kingdom
omar.sharif18@imperial.ac.uk

Christos-Savvas Bouganis
*Department of Electrical and Electronic Engineering*
*Imperial College London*
London, United Kingdom
christos-savvas.bouganis@imperial.ac.uk

***Abstract*—Gaussian Belief Propagation (GBP) is a graphical method of statistical inference that provides an approximate solution to the probability distribution of a system. In recent years, GBP has emerged as a powerful computational framework with numerous applications in domains such as SLAM and image processing. In pursuit of high performance efficiency (i.e., inference per watt), streaming-based reconfigurable hardware solutions have demonstrated significant performance gains compared to leading-edge processors and high-power, server-grade CPUs [1]. However, this class of architectures suffers from performance degradation at scale when on-chip memory is limited. This paper addresses this challenge by building on previous GBP architectural and algorithmic developments, introducing a novel hardware method that dynamically prioritizes node computations by monitoring information gain. By leveraging the inherent properties of the GBP algorithm, we demonstrate how convergence-driven optimizations can push the performance envelope of state-of-the-art reconfigurable accelerators despite on-chip memory constraints. The performance of our architecture is rigorously evaluated against this across both real-world and synthetic SLAM and image-denoising benchmarks. For equal resources, our work achieves a convergence rate improvement of up to 3.5× for large graphs, demonstrating its effectiveness for real-time inference tasks.**

## I. Introduction

Gaussian Belief Propagation (GBP) is a method of statistical inference that has garnered increased attention from the research community. It has been argued extensively in prior works that GBP has the necessary computational features to be a powerful tool to target noise-sensitive problems such as SLAM [2], [3] and image denoising [4], [5]. An attractive feature of GBP is that node updates depend exclusively on local node information, eliminating the need for any global synchronization [6], [7]. This, combined with GBP's support for asynchronous node updates, makes it particularly appealing for distributed systems and large-scale applications. GBP can be effectively applied to different problem domains by modeling systems graphically using factor graphs and applying iterative message-passing updates to converge on node marginal probabilities [7]. GBP offers distinct advantages over traditional machine learning (ML) techniques in that inference can be performed without prior training [8]. State-of-the-art acceleration of GBP using the high-power, 120W Graphcore Intelligence Processing Unit (IPU) optimized for ML [9] has motivated research into low-power reconfigurable hardware solutions, which achieve comparable acceleration with significantly lower power consumption [1]. However, in this paper (Section III-B), we will show that this class of FPGA-based streaming architectures achieves performance degradation when the memory constraints imposed by the size of the graph significantly exceed the available on-chip memory storage. Our research addresses this challenge by optimizing the sequence of node updates based on their anticipated effect on convergence through a novel scheduling algorithm centered around information gain across graph nodes. By tailoring these convergence-driven optimizations to ensure hardware compatibility, we demonstrate that the applicability of this class of accelerators can be extended to larger problem settings. This paper seeks to achieve this by making the following major contributions:

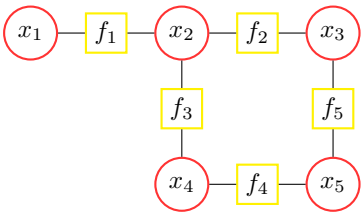


Fig. 1: Factor graph $G$ with joint probability distribution: $p(X) = f_1(x_1, x_2)f_2(x_2, x_3)f_3(x_2, x_4)f_4(x_4, x_5)f_5(x_3, x_5)$

- We present a novel hardware-tuned, residual-based node-update schedule to improve the convergence rate of GBP across inference tasks, demonstrating that the overhead to support this policy is outweighed by the increase in convergence rate afforded by the design.
- We also present an associated resource-aware toolflow for this method that enables the selection of optimal designs given the properties of the target device and prior statistics about the graph topology.

## II. Background

### A. Factor Graphs

Factor graphs (as shown in Figure 1) are a class of bipartite graphical models that can be used to represent the factorization of a function. Factor graphs can be used to model numerous applications by representing the joint probability distribution $P(x)$ across a set of variables. A factor graph $G = (X, F)$ consists of two types of nodes: variable nodes $X = \{x_i\}_{i=1}^{n}$ and factor nodes $\{f_j\}_{j=1}^{m}$. The edges in the factor graph connect variable nodes to factor nodes [8] and highlight conditional independence when a factor does not directly connect a pair of

978-3-9826741-0-0/DATE25/© 2025 EDAA

variables. The joint probability distribution can be expressed as the product of factors $p(X) = \prod_{i=1}^{m} f_i(X_i)$ where $X_i$ is the subset of variables that factor $f_i$ depends on.

### B. Gaussian Belief Propagation

Gaussian Belief Propagation (GBP) builds upon factor graphs to perform inference on Gaussian graphical models. GBP is a special case of Loopy Belief Propagation (LBP), a method of statistical inference that uses iterative message-passing updates to converge on node marginal probabilities [10]. GBP leverages the properties of Gaussian distributions $\mathcal{N}(\mu, \Sigma)$ to update and pass messages with robust convergence behavior, allowing us to converge to correct posterior means for arbitrary graphs [11]. A powerful feature of GBP is its flexibility in achieving convergence through both synchronous and asynchronous message update strategies. For both of these strategies, a node can receive either multiple messages or just one. Research indicates that asynchronous schedules generally lead to faster convergence compared to synchronous methods [12]. GBP message passing can be summarised as follows:

1. **Factor to Variable Message**: The message from a factor node $f_i$ to a variable node $X_j$ is computed by integrating out other variables $X_{-j}$ in the factor. Using the canonical form of the Gaussian distribution, the energy function is given by $E(x) = -\frac{1}{2}x^T\Lambda x + \eta^T x$ where $\Lambda$ is the precision matrix (inverse of the covariance matrix) and $\eta$ is the information vector. The message can be computed as:

$$m_{f_i \to X_j}(X_j) \propto \int f_i(X_s) \prod_{X_k \in X_s \setminus X_j} m_{X_k \to f_i}(X_k)\, dX_{-j}$$

2. **Variable to Factor Message**: The message from a variable node $X_j$ to a factor node $f_i$ is the product of incoming messages from other factor nodes:

$$m_{X_j \to f_i}(X_j) = \prod_{f_k \in \text{ne}(X_j) \setminus f_i} m_{f_k \to X_j}(X_j)$$

3. **Belief Update**: The belief at each variable $X_j$ is updated by combining incoming messages from neighboring factors:

$$b(X_j) \propto \prod_{f_i \in \text{ne}(X_j)} m_{f_i \to X_j}(X_j)$$

### C. Residual Belief Propagation

$$r_{ij} = \|m_{ij}^{\text{new}}(x_j) - m_{ij}^{\text{old}}(x_j)\| \tag{1}$$

Residual belief propagation (RBP) is an extension of BP that dynamically prioritizes message updates along high-entropy edges to enhance convergence efficiency. RBP utilizes message residuals, which assess the change in a specific message. For a message $m_{ij}(x_j)$ from node $i$ to node $j$, the residual is defined as in (1) where $\|\cdot\|$ typically denotes the $L_2$ norm [12]. The algorithm updates messages by prioritizing those with the highest residuals, ensuring that the most impactful changes propagate through the network first. This approach achieves faster convergence than traditional methods that update messages in a fixed or random order [12], [13]. In [14], an updated scheduling strategy called residual BP with lookahead zero (RBP0L) is introduced resulting in up to a 5x improvement in performance compared to the standard algorithm [14].

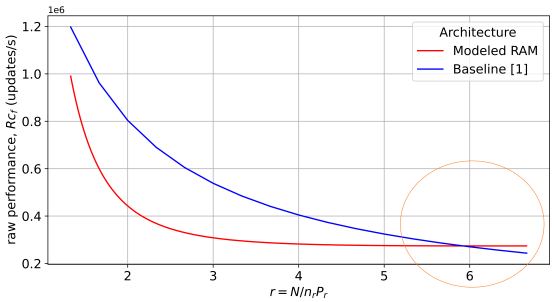


Fig. 2: A graph showing how raw performance (node updates per second $Rc_f$) scales with the ratio of the graph size ($N$) and the number of nodes the hardware can update per iteration ($n_r P_r$) (refer to Table I for taxonomy) for the baseline [1] and the modeled architecture introduced in Section III-B for a 1D Line Fitting problem (with $N = 1000$) targeting a PYNQ Z1.

## III. Related Works and Motivation

### A. Fixed Hardware Accelerators

The acceleration of GBP using fixed hardware sets has proven fruitful, with state-of-the-art advancements such as [9], which targets the problem of bundle adjustment by utilizing 1,415 IPU cores to achieve a 26x performance improvement over the Ceres CPU library. However, this approach is limited in its applicability to embedded systems due to its high power footprint. Moreover, [9] assumes that all factor graph data can fit within the IPU's on-chip memory, which means it lacks an off-chip memory structure and is not readily scalable to large, dynamically changing factor graphs. Additionally, server-grade CPUs have been explored for GBP acceleration, with FPGAs demonstrating up to a 10.5x performance improvement [1]. These challenges motivated previous work [1], which presents an architecture capable of processing arbitrarily large and evolving graphs with low power consumption.

### B. Streaming-Based FPGA Accelerator

This paper presents a resource-aware, residual-based GBP accelerator toolflow, building on the state-of-the-art work in [1], which targets reconfigurable hardware. In [1], a streaming architecture is introduced to address off-chip memory bandwidth bottlenecks associated with frequent dynamic off-chip memory accesses. To achieve better inference per watt than the IPU in [9], the full factor graph is streamed sequentially through adjacent compute units (CUs) in a dataflow manner. The architecture maintains a shared data stream across all CUs, ensuring that they do not block inputs from adjacent CUs, thereby facilitating continuous data processing. To validate this approach, we consider an ideal design with multiple CUs, where each CU updates multiple nodes, shares off-chip memory bandwidth, and no inter-CU communication exists. This method uses pointer operations to traverse the graph and the design is assumed to be memory-bound. In this modeled design, each CU accesses only local memory, with all on-chip memory

**Algorithm 1** NRBP0L Algorithm

$x \leftarrow$ uniform variable node array
$T \leftarrow$ uniform residuals array initialized to $\infty$
$q \leftarrow$ initialized random priority queue
**repeat**
  $x_i \leftarrow$ DEQUEUE($q$)
  Perform node update $x_i$ using all neighbors
  Compute residual $r = r(x_i)$
  $T(i) \leftarrow 0$
  **for all** $j$ in neighborhood of $i$ **do**
    $T(j) \leftarrow T(j) + r$
  **end for**
  **for all** $j$ in neighborhood of $i$ **do**
    Remove any pending update for $j$ from $q$
    Add $x_j$ to $q$ with priority $v$
  **end for**
**until** marginals converged

allocated for factor graph data. Our findings (Figure 2) have shown that the proposed streaming architecture outperforms the modeled design when the number of nodes brought on-chip does not significantly exceed the graph size. The node updates per second ($Rc_f$) for both architectures is plotted as the ratio of graph size to the subgraph brought on-chip ($\hat{r}$) is increased from 1 upwards. Despite the high performance potential of our baseline, it suffers degradation as the graph size exceeds the available on-chip memory by multiplicative factors. This finding motivates our current research to address this performance degradation at scale by exploring residual methods to prioritize nodes for update based on their anticipated effect on convergence, thus pushing the performance envelope of the design to larger problem settings. Furthermore, despite [1] introducing an off-chip memory structure that facilitates real-time inference, it does not leverage information from the previously converged state of the graph, limiting its applicability to real-time inference problems. This limitation motivates our approach of using residual methods that track node entropy after updates, allowing the reuse of information in evolving graph conditions to achieve global inference more efficiently.

## IV. Residual Algorithmic Augmentation

$$r_{ij} = \|x_i^{\text{new}} - x_i^{\text{old}}\| \tag{2}$$

In this paper, we present a new algorithm, Node-based Residual Belief Propagation (NRBP0L), which is a hardware-oriented approximation of the standard RBP0L algorithm. Unlike RBP0L, where node update decisions are made by tracking the entropy of directional messages along edges, NRBP0L focuses on node-level residuals, allowing for more efficient utilization of hardware resources. In RBP0L, updates are based solely on information from the broadcasting node, leading to increased off-chip memory accesses as data must be brought on-chip to update a node through a single edge. NRBP0L addresses this limitation by allowing updates to be informed by all neighboring nodes, which reduces memory access overhead and enhances parallelism. A key feature of

| Symbols | Definitions |
|---|---|
| $G = (X_s, F_s)$ | the factor graph stored in off-chip memory |
| $N = \lvert X_s \rvert$ | number of variable nodes in factor graph $G$ |
| $n$ | number of variables $x_i$ to be assigned to a CU |
| $P$ | the number of CUs instantiated |
| $k = 2\frac{\lvert F_s \rvert}{\lvert X_s \rvert}$ | the average connectivity of variables $x_i$ in $G$ |

TABLE I: A taxonomy of key terms used to describe the streaming architecture presented in Section V

NRBP0L is the measure of entropy of node marginals, denoted as $T(i)$, which is the store of the information gain due to node updates in the neighborhood of variable node $x_i$. By being agnostic to message direction and leveraging residuals on the node level, NRBP0L is specifically designed to integrate smoothly into hardware implementations, making it a more practical choice for real-time applications compared to the original RBP0L. This algorithm retains the core principles of RBP0L while optimizing for performance and scalability in hardware environments.

## V. Residual-Based Streaming Toolflow

This section presents our residual-based GBP streaming architecture. Our architecture assumes no implicit graph structure. As such, this architecture is applicable to any problem setting, including problems where an implicit graph structure does exist, such as image-denoising, which generates 2D lattice factor graphs. The architecture assumes pairwise factors between variables, where every factor is connected to exactly 2 variable nodes (such as in Figure 1). A taxonomy is provided in Table I which defines key terms that will be used in the analysis of the presented architecture. The presented work provides a faithful implementation of GBP algorithm with respect to numerical precision utilizing single-precision arithmetic and thus yields no impact on the quality of computed marginals.

### A. Architectural Overview

The proposed architecture, illustrated in Figure 3, employs off-chip memory to store factor graph data, which is transferred to the Programmable Logic (PL) via Direct Memory Access (DMA) allowing it to target graphs that require storage beyond the on-chip memory resources. The entire factor graph is streamed through a chain of adjacent Compute Units (CUs) and the Residual Schedule Manager (RSM) via an off-chip slave memory stream. A key feature of the design is the

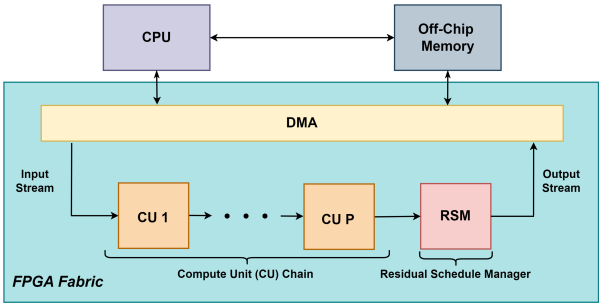


Fig. 3: A system architecture schematic illustrating the key components of the residual-based streaming toolflow.

configuration of CUs and RSM to consume inputs with a throughput of 1, allowing them to accept data every cycle. CUs parse stream contents and collect variable and factor data to perform asynchronous GBP node updates. Unlike similar dataflow architectures, CUs do not block subsequent CUs and RSM from receiving streaming contents while computing node updates, maintaining a non-blocking data flow throughout. The RSM monitors the stream and selects nodes to be updated for the next streaming cycle. By padding the end of the stream with metadata, the CPU is able to assign nodes to CUs for update at the beginning of the subsequent streaming iteration.

### B. Stream Configuration

Data is streamed through the dataflow in a chained manner where CUs and RSM assume a specific structure for retrieving from and placing data to the stream. A streaming protocol is defined for this design with two distinct stream types:

- Stream 1: contains setup data for all CUs followed by each factor index pair $(i, j)$ and accompanying precision matrix $f_{ij}$ for all factors in $F_s$.
- Stream 2: contains each variable index $i$ and marginal $x_i$ for all variables in Xs, as well as an additional packet $T_i$ which is injected to support Algorithm 1.

A streaming cycle consists of one pass of Stream 1 followed by one pass of Stream 2, with CUs assuming specific stream structures. During this, CUs gather marginals for the next iteration of node updates while also updating and returning previously computed marginals to the stream.

### C. Off-Chip Memory Organization

To abstract away the cost of accessing factor data from off-chip memory (DDR), we store off-chip data contiguously to facilitate low-overhead streaming. Three virtual spaces are maintained and streamed sequentially: block 1 for CU setup data, block 2 for factors $F_s$, and block 3 for variable nodes $X_s$. The latency for streaming a virtual space, assuming $II = 1$, is given by $t_s = C_m + n_{\text{elements}} c_f$, where $C_m$ is the cost of a memory access from the PL, $n_{\text{elements}}$ is the number of stream elements, and $c_f$ is the clock frequency. Since $C_m$ is in the order of tens of nanoseconds, $C_m$ becomes negligible. An additional benefit of this off-chip memory organization is that the architecture supports $O(1)$ time complexity for writing new nodes, as CUs and RSM are agnostic to the order of node data within streams. This eliminates the need for pointer-based graph traversal commonly found in graph processing, enabling

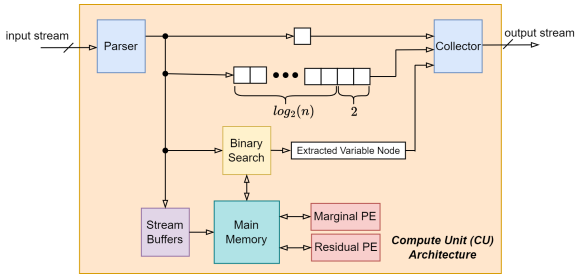


Fig. 4: A schematic illustrating the internal architecture of the CU highlighting the flow of data between internal units.

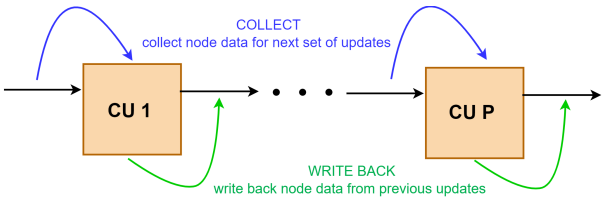


Fig. 5: An illustration of CU stream collection and writeback over an active shared-channel streaming variable node marginal (as in Stream 2 V-B).

real-time processing. The transformation of the graph into a set of arrays is only possible because, in common applications of GBP - such as image processing and SLAM - graph deletions are typically not performed. This enables sequential access to data within each block, eliminating the need for inter-block random accesses.

### D. Compute Unit (CU) Internal Architecture

Compute Units (CU) as seen in Figure 4 facilitate variable node marginal updates via: 1. parsing stream packets and storing factor graph data internally, 2. computing variable node updates, and finally, 3. writing computed marginals back to the stream which in turn updates the graph.

1. **Passing Stream Data**: CUs include a buffering network to extract nodes in the neighbor of the targeted nodes from Stream 2 using metadata and factor data from Stream 1. A key feature of CUs is that they are configured to parse stream data without stalling the inputs for later CUs (shown in blue in Figure 5).
2. **Computing Variable Node Updates**: CUs include Processing Elements (PEs) in order to facilitate Alg. 1. This includes a pipelined Marginal PE that handles node marginal updates. In addition to this, we provide CUs with a Residual PE which handles the computation of node residuals and the calculation of associated $T$ values.
3. **Writing Marginals to the Stream**: CUs write back updated node marginals to Stream 2 (shown in green in Figure 5). In order to not stall nodes we construct a binary search method to minimize latency and funnel the stream through a delay line to manage stream inputs and overwrite packets in cases where updated node marginals are found.

### E. Compute Unit (CU) Control Logic

In our architecture, we leverage one of the key properties of GBP, which is it allows for asynchronous updates using potentially outdated information from neighboring nodes. We investigated the trade-offs associated with the convergence rate when stalling the pipeline into CUs to ensure that node updates are based on the most current data from their neighbors. Our findings reveal that utilizing out-of-date information enables us to achieve high performance in processing rates. To facilitate this, CUs are equipped with control logic that writes back collected node marginals while simultaneously gathering data for upcoming updates. This approach eliminates the need for global synchronization, significantly reducing latency when passing stream inputs. As a result, CUs retain computed node marginals from the previous cycle, which have not yet been

written to off-chip memory, leading to a global stream that may contain outdated information.

To mitigate any negative effects on convergence, CUs implement internal caching to reuse these marginals in subsequent updates. Furthermore, since CUs share a global stream, later CUs benefit from more complete graph information, as earlier CUs write back node marginals to the stream. This enhances the ability to collect up-to-date information (see Figure 5). Finally, CUs incorporate control logic that allows for prefetching factor data for the next set of node updates while computing the current updates. This strategy effectively reduces the latency associated with computing information gain across node updates since it is parallelized with data fetching.

### F. Residual Schedule Manager (RSM)

To support residual-based node selection within this architecture, we introduce the Residual Schedule Manager (RSM), a module designed to prioritize nodes with the highest predicted impact on convergence. The RSM achieves this by maintaining a dynamic memory that tracks the largest $np$ information gains $T$ observed across the graph, paired with their respective node indices. During each streaming cycle, the RSM continuously monitors the incoming stream for information gain values associated with node indices. When a new gain exceeds the smallest value in the memory, the corresponding node replaces the lowest-ranked entry, ensuring that the memory dynamically reflects the most impactful nodes for convergence. At the end of each streaming cycle, residual nodes with high information gains are padded to the end of the stream. The CPU subsequently processes these nodes to assign nodes to CUs for the next streaming cycle. The local memory of the RSM is then flushed, resetting it for subsequent cycles. This design integrates seamlessly with the baseline streaming architecture proposed in [1]. By maintaining a prioritization of high-impact nodes across streaming cycles, our design ensures the highest entropy nodes are prioritized for update.

## VI. Design Space Explorer (DSE)

The architecture is supported by a design space explorer (DSE), which enables the selection of optimal configurations based on the target FPGA's properties and the graph topology statistics (refer to Table I). The proposed system is optimized at compile time in terms of the number of compute units (CUs), denoted as $P$, the number of node updates per CU, $n$, and the parallelization factors, $u$, for the processing units, which includes components such as the matrix inverter, multiplier, adder, and subtractor. The foundation of the DSE lies in performance and resource models developed for our residual-based toolflow, building upon prior work [1], which employed a random policy for graph node updates. These models allow us to estimate the accelerator's raw performance, $Rc_f$, defined as node updates per second, and select optimal configurations based on the available FPGA resources.

| Architecture | LUT% | FF% | BRAM% | DSP% | $\mathbf{Rc_f}$% |
|---|---|---|---|---|---|
| Residual | 7.44 | 6.79 | 3.13 | 0.32 | 7.21 |
| Baseline [1] | 6.58 | 5.87 | 2.53 | 0.10 | 6.15 |

TABLE II: A table presenting the accuracy of the residual and baseline [1] performance and resource models (measured as percentage delta against the place and route design).

| Benchmark | $N$ | $\lvert F\rvert$ | $k$ | $\frac{np_{\text{baseline}}}{N}$ | $\frac{np_{\text{residual}}}{N}$ | Real |
|---|---|---|---|---|---|---|
| Line 1000 | 1000 | 999 | 2.00 | 0.824 | 0.618 | ✗ |
| Lattice 1000 | 992 | 1921 | 3.87 | 0.5323 | 0.4234 | ✗ |
| Line 10000 | 10000 | 9999 | 2.00 | 0.1024 | 0.096 | ✗ |
| Lattice 10000 | 10000 | 19800 | 3.96 | 0.0795 | 0.0579 | ✗ |
| MITb [15] | 808 | 827 | 2.05 | 1.0 | 0.7574 | ✓ |
| Intel [15] | 1728 | 2512 | 2.91 | 0.192 | 0.144 | ✓ |
| M3500 [16] | 3500 | 5453 | 3.12 | 0.3796 | 0.2795 | ✓ |
| Ais2klinik [17] | 15115 | 16727 | 2.21 | 0.0611 | 0.0589 | ✓ |

TABLE III: A table summarizing the real and synthetic graph structures evaluated in Section VII targeting a PYNQ Z1.

## VII. Evaluation

### A. Performance and Resource Models

Accompanying this work are performance and resource models that accurately predict the latency of stream passes and node computations, as well as the resources required to support both the proposed residual policy and the baseline architecture [1]. Table II presents the average percentage differences in performance (stream and computation latencies) and resource utilization (FPGA base resources) between the models and the routed designs for the two architectures. A total of 18 designs were sampled, with inputs of 100 to 10,000 nodes, average connectivity $k$ of 2 to 4, 1 to 5 compute units (CUs), and a random configuration of processing element unroll factors $u$.

### B. Design Selection

GBP convergence is sensitive to factor graph properties such as connectivity, node magnitudes, and the order of node updates. As a result, predicting the convergence rate for a specific design configuration is challenging, since GBP typically exhibits high variance in the number of iterations required to converge [18]. To select optimal designs, we found that the raw performance of our accelerators $Rc_f$ offers an effective relative ranking between designs, serving as a reliable proxy for identifying optimal configurations based on the available FPGA resources. Figure 6 illustrates the strength of $Rc_f$ as a design selector for a selection of design configurations for

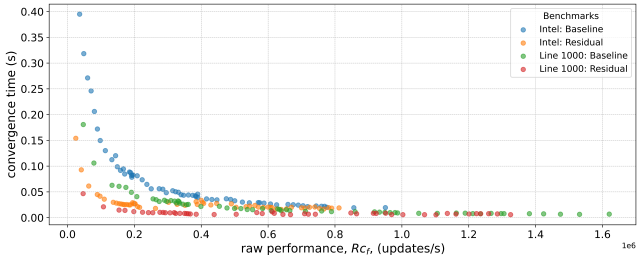


Fig. 6: A graph illustrating the strong correlation between convergence time and raw accelerator performance for the residual and baseline architecture (for selected benchmarks).

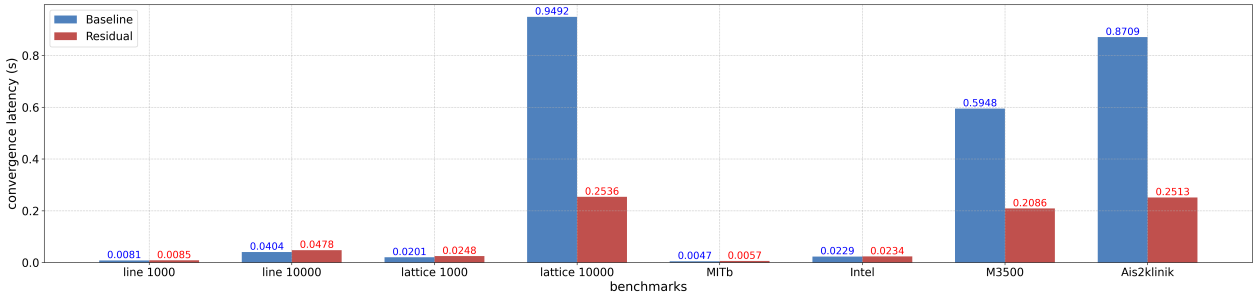


Fig. 7: A graph showing the convergence time across the benchmarks in Table III. Benchmark graphs are seeded with 3D random data ensuring a starting displacement of $MAE = 10$ up to a $MAE = 10^{-5}$ convergence.

the residual architecture and the baseline [1] across selected benchmarks (Section III).

### C. Convergence Evaluation

The performance of the proposed architecture is evaluated using a combination of real and synthetic SLAM and image-denoising graph configurations, seeded with randomly generated data. For these experiments, four real datasets (Intel [15], MITb [15], M3500 [16], Ais2klinik [17]) are utilized and, for comparison, four synthetic datasets are also provided. The real datasets were generated by processing raw measurements from wheel odometry and laser range finders, while the synthetic datasets correspond to line graphs (1D fitting) and lattices (image denoising) of varying sizes (refer to Table III).

A bar chart comparing the performance of the presented residual design against the baseline design [1] across the benchmarks (Table III) is shown in Figure 7. A PYNQ Z1 is targeted and a Design Space Explorer is utilized (refer to Section VI) to select the optimal design configuration for each architecture. Our results demonstrate that the proposed design achieves up to 3.7× (Lattice 10000) improvement on large synthetic graphs and 3.5× (Ais2klinik) on large real graphs, significantly reducing convergence latency through the residual-based toolflow. Moreover, the convergence rate improvement is accompanied by minimal performance degradation in the case of small graphs. In the extreme case, we see a degradation of 21.2% where the whole graph can fit on chip, with other benchmarks exhibiting more comparable performance between the baseline and residual methods as the node update ratio for stream iterations $np/N$ approaches 1. In these scenarios, the importance of node selection diminishes since all nodes in the graph are frequently updated using the random scheduling utilized in [1]. This makes the resource trade-off for residual-based prioritization less advantageous. Despite the resource overhead required to support the residual policy, which reduces the number and memory resources assigned to each CU, our approach effectively scales performance for larger graphs. This capability enables our toolflow to achieve real-time inference in larger, and more complex, environments and applications.

| | Residual (Ultrascale+) | Baseline (Ultrascale+) | IPU (Graphcore) |
|---|---|---|---|
| $Rc_f$ (updates/s) | $8.40 \times 10^6$ | $7.20 \times 10^6$ | $2.86 \times 10^7$ |
| Power (W) | $6.10 \times 10^0$ | $6.10 \times 10^0$ | $1.20 \times 10^2$ |
| $Rc_f$ per watt (updates/s/W) | $1.37 \times 10^6$ | $1.18 \times 10^6$ | $2.38 \times 10^5$ |

TABLE IV: The node updates per second $Rc_f$ per watt for the presented residual architecture, baseline [1] and IPU [9].

### D. Comparison to Other Works

For completeness, the presented architecture is compared to the state-of-the-art IPU architecture in [9] ported to the Xilinx Ultrascale+ MPSoC ZCU104. Table IV shows the $Rc_f$ performance for the IPU with $G$ = (1216, 929). In contrast, the presented architecture is evaluated with $G$ = (1216, 1216) since it assumes linear factor relationships. As a result, The latency associated with relinearizing weights in [9] is excluded from this comparison to estimate $Rc_f$. In the case of the Ultrascale+, our residual toolflow optimizes for a design with a $Rc_f$ improvement of 16.7% against the baseline [1]. This comes despite additional resources provided to the system to support residual-based prioritization, showing the significance of pre-fetching across stream iterations to increase the raw performance (refer to Section V-E). This allows us to achieve a performance per watt improvement against the IPU of 5.76×. (for comparison, the baseline achieves a 4.96× improvement). These update rate per watt improvements come in addition to the fact our design exploits residual information to improve the convergence rate unlike [1] and [9]. Furthermore, although the IPU achieves higher raw performance than the FPGA (which is to be expected since it stores all factor graph data on-chip), the high power consumption limits its suitability for real on-device SLAM applications at the edge. These results confirm that our architecture is the state-of-the-art for achieving real-time inference at low power profiles.

## VIII. Conclusion

This paper presents a novel resource-aware residual-based toolflow for accelerating Gaussian Belief Propagation (GBP) using FPGAs. Our results demonstrate exceptional performance in real-time inference for large graphs compared to the state-of-the-art streaming-based reconfigurable hardware solutions. Our work addresses the performance degradation observed when off-chip memory resources are limited, importantly showing that residual methods tailored to hardware can significantly improve the convergence rate in such scenarios. This paper underscores the exciting potential of this toolflow to enhance the efficiency and scalability of GBP processing, making real-time inference achievable across a wide range of applications.

# REFERENCES


[1] O. Sharif and C. Bouganis, "A Framework for Designing Scalable Gaussian Belief Propagation Accelerators for use in SLAM," in *Conference  Exhibition on Design, Automation & Test in Europe*, 2024.

[2] A. J. Davison and J. Ortiz, "Futuremapping 2: Gaussian Belief Propagation for Spatial AI," *arXiv preprint arXiv:1910.14139*, 2019.

[3] C. L. González, E. U. Molina, and J. Martínez, "Distributed Simultaneous Localization and Extrinsic Calibration for Multi-Robot Systems Using Gaussian Belief Propagation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3680–3687, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/10008583

[4] S. Patel, M. R. Shankar, and D. P. Mandal, "A Novel Approach to Multi-Robot Cooperative Localization Using Belief Propagation," *IEEE Transactions on Robotics*, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10387679

[5] Y.-S. Feng, J.-Y. Chen, H.-C. Wang, C.-W. Huang, and J.-L. Chern, "Learning-Based Gaussian Belief Propagation for Bundle Adjustment in Visual SLAM," in *2022 IEEE Globecom Workshops*, 2022, pp. 166–171.

[6] D. Bickson, "Gaussian Belief Propagation: Theory and Application," *arXiv preprint arXiv:0811.2518*, 2008.

[7] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[8] J. Ortiz, T. Evans, and A. J. Davison, "A Visual Introduction to Gaussian Belief Propagation," *arXiv preprint arXiv:2107.02308*, 2021.

[9] J. Ortiz, M. Pupilli, S. Leutenegger, and A. J. Davison, "Bundle Adjustment on a Graph Processor," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[10] A. T. Ihler, J W. Fisher III, and A. S. Willsky, "Loopy Belief Propagation: Convergence and Effects of Message Errors," in *Machine Learning Research, vol. 6, no. 31, pp. 905–936, 2005*.

[11] Y. Weiss and W. T. Freeman, "Correctness of Belief Propagation in Gaussian Graphical Models of Arbitrary Topology," in *Neural Information Processing Systems (NIPS)*, vol. 19, no. 4, 2000, pp. 673–679.

[12] 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 (UAI 2006)*, 2006, pp. 165–173. [Online]. Available: https://arxiv.org/abs/1206.6837

[13] J. E. Gonzalez, Y. Low, C. Guestrin, A. M. Dai, and W. Gu, "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. 246–253.

[14] C. Sutton and A. McCallum, "Improved Dynamic Schedules for Belief Propagation," in *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI Press, 2007, pp. 376–383.

[15] Institut für Pflanzenwissenschaften, Universität Bonn, "Datasets," n.d., accessed: 07/09/2024. [Online]. Available: https://www.ipb.uni-bonn.de/datasets/

[16] E. Olson, J. J. Leonard, and S. J. Teller, "Fast Iterative Alignment of Pose Graphs with Poor Initial Estimates," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2006, pp. 2262–2269.

[17] Universität Freiburg, "Datasets," n.d., accessed: 07/09/2024. [Online]. Available: http://europa.informatik.uni-freiburg.de/datasets.html

[18] F. Kamper, S. J. Steel, and J. A. du Preez, "On the Convergence of Gaussian Belief Propagation with Nodes of Arbitrary Size," *Journal of Machine Learning Research*, vol. 20, no. 165, pp. 1–37, 2019.