A Framework for Designing Scalable Gaussian Belief Propagation Accelerators for use in SLAM

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 an iterative method for factor graph inference that provides an approximate solution to the probability distribution of a system. It has been shown to be a powerful tool in numerous applications including SLAM, where the estimation of the robot's position and the map of the environment is required. State-of-the-art implementations suffer from scalability issues, or exhibit performance degradation when off-chip memory access is required. This paper addresses these challenges using a streaming architecture via a chain of parameterizable Processing Elements (PE) that can be tuned to the problem's characteristics through the use of an optimizer. This work overcomes the limitations of existing GBP implementations achieving 142x-168x performance improvements over an embedded CPU for large graphs.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) involves concurrently creating an environment map and estimating the robot's position relative to features of interest [1]. Traditional hardware sets with high-compute capabilities are power-intensive, which hinders the adoption of on-device SLAM at the edge. Low-power hardware accelerators, which sit on-device, are sought in order to allow robots to make decisions locally, garnering increased attention from the research community [2], [3]. Gaussian Belief Propagation (GBP) is a framework which can be effectively applied to SLAM problems by modelling systems graphically and employing iterative message passing to achieve probabilistic inference [4], [5]. This approach allows for concurrent statistical inference across graph localities, lending itself well to highly parallelizable devices.

II. Gaussian Belief Propagation (GBP)

GBP operates over factor graphs, which are a class of probabilistic graphical models that can be used to represent the joint probability distribution across a set of variables. Factor graphs contain two node types: variable nodes, which represent unknown quantities in a system; and factor nodes, which define the relationship between a set of variable nodes. Expressed formally, a factor graph $G = (X, F, E)$ has variables $X = \{x_i \mid i \in \{1, \ldots, n\}\}$ (where n is the number of variables), factor space $F = \{f_s \mid s \in \{1, \ldots, m\}\}$ (where m is the number of factors) and edges E [4]. The joint probability distribution across X is given as $p(X) = \prod_{s=1}^{M} f_s(X_s)$. Factor graph edges are always between variables and factors, and never between respective variables and factors. Fig. 1 shows an example factor graph with $|X| = 4$ (circles) and $|F| = 3$

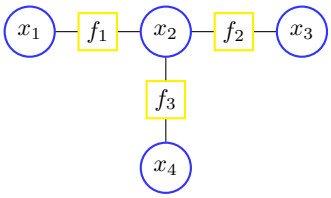

Fig. 1: Example factor graph G with joint probability distribution: $p(X) = f_1(x_1, x_2)f_2(x_2, x_3)f_3(x_2, x_4)$

(squares). Belief propagation builds upon factor graphs to compute the marginal distribution $p(x_i)$ for each variable in X from the joint distribution $p(X)$. Using the assumption of Gaussian distributed nodes, and expressing nodes in the canonical form $x \sim \mathcal{N}^{-1}(\eta, \Lambda)$, it is possible to fully express message passes between variable nodes as linear algebra operations in η and Λ. GBP may be applied to SLAM problems by initializing the joint probability X using sensory data and selecting a fixed measurement variance Σ_n for Gaussian noise [6]. GBP is beneficial in that it guarantees convergence to correct posterior means for arbitrary graphs with no loops via both synchronous and asynchronous node updates, which makes it a robust framework for solving SLAM problems.

III. System Architecture

A high-level diagram of the proposed architecture is shown in Fig. 2. Off-chip memory (such as DDR) is employed to store factor graph data which is transferred to the Programmable Logic (PL) via Direct Memory Access (DMA). The full factor graph is streamed through a chain of adjacent Processing

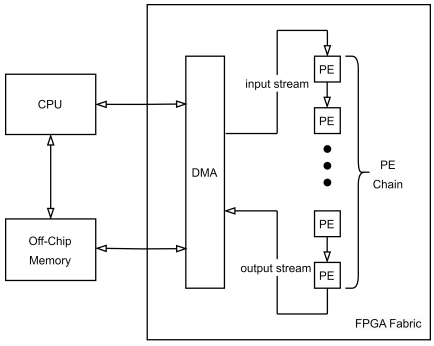

Fig. 2: A system architecture schematic illustrating the key components of the chained PE architecture.

979-8-3503-4859-0/DATE24/© 2024 EDAA

Elements (PE) via the AXI slave stream. In addition to this, associated meta-data is also streamed through PEs to aid in node updates. PEs are responsible for parsing stream contents and collecting factor graph data from it to perform asynchronous GBP node updates. Each PE is assigned a unique identifier at compile time, after which the CPU is responsible for allocating a unique set of variable nodes to each PE for update. PEs have been designed to consume inputs with an initiation interval $II = 1$, accepting data every cycle with a bus width equal to the data width. Data is streamed through the PEs in a chained manner, where PEs assume a specific stream structure for retrieving from and placing data into the stream. A streaming protocol is defined for this design with two distinct types:

- Stream 1: contains setup data for all PEs 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 X_s.

A streaming cycle is defined as one pass of Stream 1 followed by one pass of Stream 2, which necessitates different processing requirements at each pass. During the Stream 2 pass, each PE collects the marginals for the next iteration of node updates, whilst updating previously computed marginals by placing them back into the stream. In order to write back the computed marginals nodes to the stream without back-pressuring inputs, a pipelined binary search has been devised with a fixed latency by using a binary tree structure.

IV. Processing Element (PE) Architecture

PEs include a compute unit which handles node marginal updates, local memory units to store collected factor graph data, as well as control logic to parse and place stream contents. A key feature of the PE architecture is the stream parser and stream collector which buffers stream data and controls the transmission of packets to the adjacent PE (Fig. 3). During Stream 1, the parser writes the PE input directly to the output via a register; whilst in Stream 2, the parser writes the PE input to a shift register which allows for the aforementioned pipelined binary search of previously computed node marginals.

V. Evaluation

The proposed system is compile-time parameterisable with respect to the number of PEs, the number of node updates per PE, and the parallelization factors for the compute unit. The

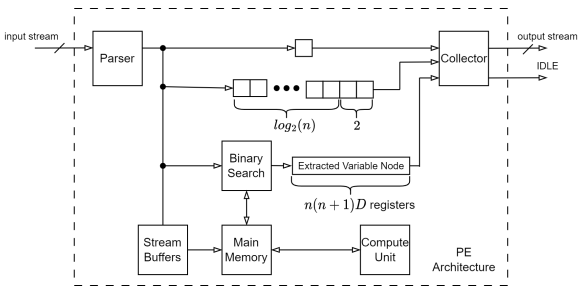

Fig. 3: A schematic illustrating the internal architecture of the PE incl. stream buffering, the compute unit and the flow of data between internal units (note, D is the data dimensionality).

	$(1000, 1500)$	$(10000, 15000)$	$(1000, 4000)$	$(10000, 40000)$
ARM	1.60×10^4	1.57×10^4	6.15×10^3	6.11×10^3
Intel	3.01×10^5	2.51×10^5	1.12×10^5	1.05×10^5
PYNQ Z1	8.03×10^5	2.14×10^5	2.62×10^5	7.27×10^4
Ultrascale+	5.65×10^6	2.64×10^6	2.72×10^6	8.66×10^5

TABLE I: The node updates per second (Rc_f) for the presented architecture using the toolflow (Section V) compared to CPU implementations for a variety of graphs $G = (X, F)$.

	CPU (ARM)	CPU (Intel)	FPGA (PYNQ)	FPGA (Ultrascale+)	IPU (Graphcore)
Rc_f	2.32×10^4	4.68×10^5	1.18×10^6	7.20×10^6	2.86×10^7
Power (W)	1.00×10^0	2.00×10^2	1.73×10^0	6.10×10^0	1.20×10^2
Normalized	2.32×10^4	2.34×10^3	6.82×10^5	1.18×10^6	2.38×10^5

TABLE II: The node updates per second (Rc_f) and power consumption for 2 toolflow-generated FPGA designs, compared to 2 CPU implementations, and the IPU architecture in [7].

modelled design is ported to the PYNQ Z1 and the Ultrascale+ MPSoC ZCU104 and is compared to a C++ implementation run on a high-power CPU (Intel Xeon Gold 6154 CPU, 3.00GHz), and a low-power processor (ARM Dual-Core Cortex-A9 Processor, 650MHz) to assess its performance in an embedded system. The performance is profiled to real 3D SLAM problems using randomly generated factor graph data. The proposed architecture yields a 141.7x-168.2x performance improvement against the ARM processor and 8.25x-10.5x against the Intel CPU for large sparse and dense graph configurations (Table I). The presented architecture is also compared to the IPU architecture in [7] which applies GBP to bundle adjustment. [7] assumes all data can be stored on-chip, which is an unrealistic assumption for real problems. The paper also assumes non-linear factors connecting variables and factors can connect more than 2 variables. It is possible to estimate its raw performance by subtracting the additional latency to relinearize factors for the provided case of $G = (1216, 929)$. This is compared to the presented architecture with $|F| = 1216$ (Table II). The Ultrascale+ design outperforms the IPU in Rc_f per watt drawn by a factor of $\sim 4.96\times$, which further illustrates the usefulness of this framework for tackling SLAM systems at the edge.

References

[1] C. C. et al., "Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, p. 1309–1332, 2016.

[2] K. Boikos and C.-S. Bouganis, "Semi-dense slam on an fpga soc," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–4.

[3] Q. Gautier, A. Althoff, and R. Kastner, "Fpga architectures for real-time dense slam," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2019, pp. 83–90.

[4] 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.

[5] C. Rhodes, C. Liu, and W.-H. Chen, "Scalable probabilistic gas distribution mapping using gaussian belief propagation," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 9459–9466.

[6] A. J. Davison and J. Ortiz, "Futuremapping 2: Gaussian belief propagation for spatial ai," *arXiv preprint arXiv:1910.14139*, 2019.

[7] J. Ortiz, M. Pupilli, S. Leutenegger, and A. J. Davison, "Bundle adjustment on a graph processor," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, jun 2020, pp. 2413–2422.