Factor Graph Accelerator for LiDAR-Inertial Odometry (Invited Paper)

Yuhui Hao*
Tianjin University
China

Bo Yu*
PerceptIn
U.S.A.

Qiang Liu†
Tianjin University
China

Shaoshan Liu
PerceptIn
U.S.A.

Yuhao Zhu
University of Rochester
U.S.A.

ABSTRACT

Factor graph is a graph representing the factorization of a probability distribution function, and has been utilized in many autonomous machine computing tasks, such as localization, tracking, planning and control etc. We are developing an architecture with the goal of using factor graph as a common abstraction for most, if not, all autonomous machine computing tasks. If successful, the architecture would provide a very simple interface of mapping autonomous machine functions to the underlying compute hardware. As a first step of such an attempt, this paper presents our most recent work of developing a factor graph accelerator for LiDAR-Inertial Odometry (LIO), an essential task in many autonomous machines, such as autonomous vehicles and mobile robots. By modeling LIO as a factor graph, the proposed accelerator not only supports multi-sensor fusion such as LiDAR, inertial measurement unit (IMU), GPS, etc., but solves the global optimization problem of robot navigation in batch or incremental modes. Our evaluation demonstrates that the proposed design significantly improves the real-time performance and energy efficiency of autonomous machine navigation systems. The initial success suggests the potential of generalizing the factor graph architecture as a common abstraction for autonomous machine computing, including tracking, planning, and control etc.

KEYWORDS

factor graph, autonomous machine computing, computer architecture, robotics

1 INTRODUCTION

Autonomous machine computing (AMC) is the next big trend in information technology in the coming decades, after personal computing, mobile computing, and cloud computing [1]. Specifically, AMC is the core technology stack that empowers various kinds of autonomous machines, including Mars or Lunar explorers, intelligent vehicles, autonomous drones, delivery robots, home service robots, agriculture robots, industry robots and many more that we have yet to imagine [2].

Similar to other information technology stacks, the AMC technology stack consists of hardware, systems software and application software. Sitting in the middle of this technology stack is computer architecture, which defines the core abstraction between hardware and software. This abstraction layer allows software developers to focus on optimizing the software and hardware developers to focus on developing faster, more affordable, more energy-efficient hardware that can unlock the imagination of software developers.

While there have been many recent proposals of computer architectures for AMC [3–7], this paper is the first to explore factor graph as a common abstraction, or architecture, for autonomous machine operations. A factor graph is a graph representing the factorization of a probability distribution function and has been utilized in many autonomous machine operations, such as localization, tracking, planning and control [8, 9]. As the initial step, we focus on developing a factor graph accelerator for LiDAR-inertial odometry, a key localization method in many autonomous machines.

The rest of this paper is organized as follows. Sec. 2 introduces the background of factor graph. Sec. 3 summarizes the traits of factor graph computing that motivates the design of the proposed accelerator. Sec. 4 delves into the hardware design of the proposed accelerator. Sec. 5 presents the results of performance evaluation, and we conclude in Sec. 6.

2 BACKGROUND ON FACTOR GRAPH

Factor graphs are probabilistic graphical models that can essentially model complex state estimation problems. For representing a state estimation problem, the factor graph is organized as a bipartite graph consisting of variables connected by factors, where variables represent unknown states and factors connected to states represent probabilistic relations between states [10]. According to the factor graph, the joint probability distribution of the entire states can be factorized into products of probabilistic functions; as a result, the state estimation is turned into the maximum a posterior (MAP) inference [8], of which solution is to maximize the products of probabilistic factors.

Many AMC problems, such as estimation, planning and optimal control, have an optimization problem at their core, as a result factor graphs have been well applied to represent, reason and solve those AMC operations. The factor graph abstraction of robotic optimization brings several benefits: First, it is an unified model that well suits for various forms of optimization problem, such as

* indicates equal contribution to the paper.
† indicates the corresponding author of the paper.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3561112

filtering [11], incremental smoothing [12] and batch optimization [13]. Second, it provides a concise and abstract programming interface to compose robotic optimization problems, through which all programmers need to program a robotic optimization application is to specify the variables, factor functions and their connections according to the graph. Third, its graph structure facilitates sparse data storage that can be exploited to optimize memory resources and performance.

Several software libraries [13–15] using factor graph for robotic optimization have been developed. However, hardware accelerators for factor graph are less studied, which however are highly required by AMCs that usually have stringent power and performance constraints [4]. To take the first step towards the hardware accelerator of factor graph, we use LiDAR-Inertial Odometry (LIO, a dominant localization approach for autonomous vehicles) [16] as an example to exploit characteristics of factor graph computing that facilitate hardware design.

3 TRAITS OF FACTOR GRAPH COMPUTING

This section first introduces the general MAP inference formulation for factor graph (Sec. 3.1). Then the algorithm for solving the LIO based on factor graph is illustrated in Sec. 3.2. The structural characteristic of the LIO factor graph that facilitates parallel computing is discussed in Sec. 3.3.

3.1 MAP Inference for Factor Graph

For AMC state estimation tasks, the core function is to compute unknown states X, such as poses (position and orientation) in localization applications, given the noisy measurements, Z. Factor graphs can essentially model AMC estimation problems, in which variable nodes represent unknown states and factor nodes represent functions that only apply on variables connected with the factor node. With the topology definition, a factor graph defines the factorization of a global function $\Phi(X)$. In AMC state estimation, the global function is a joint probability distribution. The object of state estimation is to maximize the joint probability, which turns into a MAP inference,

$$X^* = \arg\max_X \Phi(X) = \arg\max_X \prod_i \phi_i(X_i) \tag{3.1}$$

Assuming that all factors are of the form as Equ. 3.2, which include both Gaussian priors and likelihood factors derived from measurements corrupted by zero-mean, normally distributed noise, where z_i is the i-th measurement or ground truth, $h_i(\cdot)$ is a measurement function that maps the state X_i to be estimated to its sensor's measurement space, Σ_i is the covariance matrix of the i-th measurement and $\|\cdot\|^2_{\Sigma_i}$ is the Mahalanobis norm that quantifies the error. Taking the negative log of Equ. 3.1 allows us to instead minimize a sum of nonlinear least-squares as Equ. 3.3.

$$\phi_i(X_i) \propto \exp \|h_i(X_i) - z_i\|^2_{\Sigma_i} \tag{3.2}$$

$$X^* = \arg\min_X \sum_i \|h_i(X_i) - z_i\|^2_{\Sigma_i} \tag{3.3}$$

Nonlinear least-squares problems can not be solved directly, but require an iterative solution starting from an initial estimate. Typical nonlinear solvers, such as the Gaussian-Newton method,

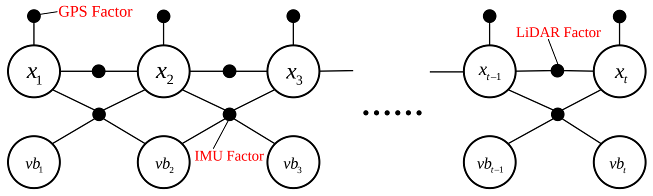

Fig. 3.1: Factor graph of tightly coupled LiDAR-Inertial Odometry. The white circles represent the pose, velocity and biases variables. The black dots denote the different factors.

have a similar compute process. They start from an initial X^0. In each iteration, an increment Δ is computed and applied to obtain the next estimate $X = X + \Delta$. This process stops when certain convergence criteria are reached, such as the change Δ falling below a small threshold. In each iteration, Δ is found by solving the linear least-squares problem as Equ. 3.4,

$$\Delta^* = \arg\min_\Delta \sum_i \|A_{bi}\Delta_i - \epsilon_{bi}\|^2_2 = \arg\min_\Delta \|A_b\Delta - \epsilon_b\|^2_2 \tag{3.4}$$

$$A_{bi} = \Sigma_i^{-\frac{1}{2}} \left.\frac{\partial h_i(X_i)}{\partial X_i}\right|_{X_i}, \quad \epsilon_{bi} = \Sigma_i^{-\frac{1}{2}}(z_i - h_i(X_i)) \tag{3.5}$$

where the Mahalanobis norm is transformed to 2-norm by Equ. 3.5. A_b and ϵ_b are obtained by collecting all Jacobian matrices A_{bi} and residuals ϵ_{bi} into a large matrix A_b and right-hand-side (RHS) vector ϵ_b, respectively.

In fact, the structure of factor graph is equivalent to the sparse pattern of A_b, i.e., each block row in A_b corresponds to a factor node; each block column in A_b corresponds to a variable node. Thus, the structure of factor graph directs the solution of Equ. 3.4. By traversing all variable nodes in factor graph, the adjacency factors of each variable are combined and then factorized, namely variable elimination [8]. After eliminating all variable nodes, Equ. 3.4 is transformed to Equ. 3.6, where R is an upper triangular matrix, also called the Bayes net in probabilistic graph theory [10], obtained by QR decomposition of A_b. Then Δ can be solved by back substitution in R.

$$\begin{aligned}\arg\min_\Delta \|A_b\Delta - \epsilon_b\|^2_2 &= \arg\min_\Delta \|Q^TA_b\Delta - Q^T\epsilon_b\|^2_2 \\ &= \arg\min_\Delta \|R\Delta - d\|^2_2\end{aligned} \tag{3.6}$$

However, data is obtained as a temporal sequence in many inference problems for AMC. The latency to solve grows over time to the point that real-time batch optimization is no longer feasible. In fact, the constraints formed by new measurements usually affect only a local part of factor graph and the remaining part is unchanged so that incremental smoothing requires solving only for partially affected factor graph [15]. In terms of linear algebra, only partial entries in R require to be updated [17]. Therefore, incremental smoothing can be regarded as the recalculation of a subgraph of the factor graph, of which process is consistent with the batch solution but the dimension is reduced significantly.

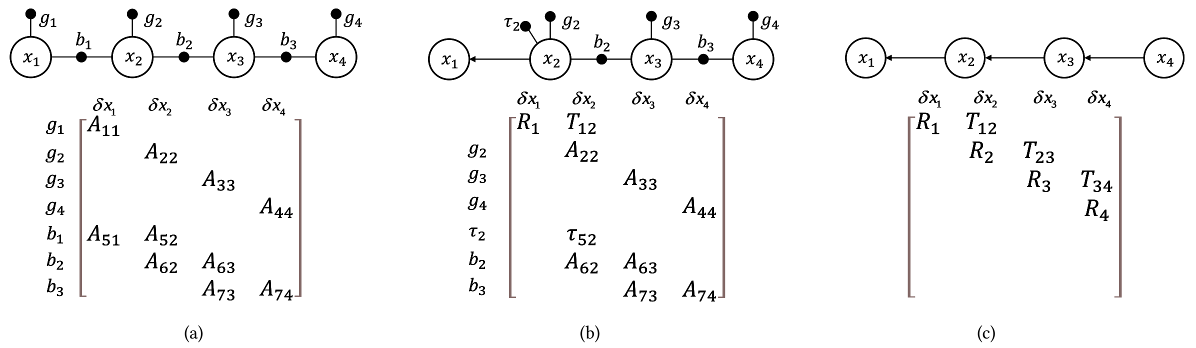
g_1 g_2 g_3 g_4 b_1 b_2 b_3 x_1 x_2 x_3 x_4
δx_1 δx_2 δx_3 δx_4
A_{11} A_{22} A_{33} A_{44} A_{51} A_{52} A_{62} A_{63} A_{73} A_{74}
(a)
τ_2 R_1 T_{12} τ_{52}
(b)
R_2 T_{23} R_3 T_{34} R_4
(c)

Fig. 3.2: Factor graph and corresponding matrix obtained using serial elimination. (a) Toy example of LIO factor graph with four variables and its corresponding Jacobian matrix A_b in serial elimination order. (b) Factor graph and the corresponding matrix after eliminating x_1. (c) The Bayes net obtained using serial elimination and the corresponding upper triangular matrix R.

Algorithm 3.1 Variable Elimination On LIO Factor Graph

```
1: function EliminationOnLio(LIO Factor Graph $\Phi_{1:n}$)
2:     for $i = 1$ to $n$ do
3:         $\overline{A_i} \leftarrow g_i, b_i, \tau_i$
4:         $p(x_i|x_{i+1}), \tau_{i+1} \leftarrow$ partial QR decomposition on $\overline{A_i}$
5:     return $p(x_1|x_2)p(x_2|x_3)\cdots p(x_n)$
```

Algorithm 3.2 Incremental Inference On LIO Factor Graph

```
1: function IncrementalLio($g_{j+1}$, $b_j$)
2:     Reconstruct the factor graph $\Phi_{j-1:j+1}$ of $x_{j-1}, x_j$ and $x_{j+1}$
3:     EliminationOnLio($\Phi_{j-1:j+1}$)
4:     return $p(x_{j-1}|x_j)p(x_j|x_{j+1})p(x_{j+1})$
```

3.2 Solving for LIO Factor Graph

LIO uses GPS, IMU and LiDAR as the main sensors. GPS measurements generate constraints on each keyframe; IMU and LiDAR measurements produce constraints on two adjacent keyframes. Thus, the LIO factor graph has a chain-like structure, as shown in Fig. 3.1. The factor nodes contain the constraints formed by measurements of three sensors, and the variable nodes include the pose, velocity and biases of each keyframe. It results in a factor graph that is rich in regularity when solving for the MAP solution. The chain-like property avoids the necessity of traversing the factor graph, saving time and space overhead.

Fig. 3.2(a) shows the LIO factor graph for a toy example and its corresponding Jacobian matrix A_b, where the velocity and biases variables as well as the IMU factors are omitted for the sake of brevity and clarity. If the forward (serial) elimination order of $x_1 \to x_2 \to x_3 \to x_4$ is selected, when eliminating x_1, the adjacent factors $g_1(x_1)$ and $b_1(x_1, x_2)$ are multiplied into the product $\psi(x_1, x_2)$ and factorized into a conditional probability $p(x_1|x_2)$ (the first block row in Fig. 3.2(b)) and a new factor $\tau_2(x_2)$, that is

$$\psi(x_1, x_2) \leftarrow g_1(x_1)b_1(x_1, x_2) \tag{3.7}$$

$$p(x_1|x_2)\tau_2(x_2) \leftarrow \psi(x_1, x_2) \tag{3.8}$$

From the perspective of the matrix, Equ. 3.7 corresponds to the process of constructing a matrix $\overline{A}$; Equ. 3.8 represents the process of decomposing $\overline{A}$. When eliminating x_1, $\overline{A_1} = \begin{bmatrix} A_{11} & \\ A_{51} & A_{52} \end{bmatrix}$. Then perform partial QR decomposition on $\overline{A_1} = Q_1 \begin{bmatrix} R_1 & T_{12} \\ & \tau_{52} \end{bmatrix}$, where R_1 is an upper triangular matrix, as shown in Fig. 3.2(b). The above process is then iterated for subsequent variables. As the number of iterations increases, the dimension of τ grows linearly, thus the dimension of $\overline{A}$ is also growing linearly. The obtained Bayes net is shown in Fig. 3.2(c) after all variables are eliminated. When solving the MAP solution in back substitution, the reverse order of the elimination should be followed, i.e., $x_4 \to x_3 \to x_2 \to x_1$ should be solved in turn.

However, in each elimination, factors adjacent to x_j to be eliminated are fixed due to the chain-like nature of LIO factor graph, i.e., only a GPS factor $g_j(x_j)$, a LiDAR factor $b_j(x_j, x_{j+1})$, and a new factor $\tau_j(x_j)$ obtained by previous elimination are involved when eliminating x_j. Therefore, the matrix $\overline{A_j}$ can be directly constructed without traversing the factor graph. Besides, during the back substitution, each variable depends only on its neighboring variable so that this process can also be performed without traversing the graph. Algo. 3.1 shows pseudocode for eliminating a LIO factor graph.

When solving LIO factor graph incrementally, a factor graph of the three most recent variables need to be recalculated according to [15], while the other parts remain unchanged. Algo. 3.2 shows pseudocode for incremental inference on LIO factor graph.

3.3 Parallel Computation of LIO Factor Graph

The parallelism of elimination and back substitution is explored by considering the symmetric properties of LIO factor graph to improve the inference performance in hardware. Parallel elimination is feasible because only the elimination order is changed and the parts without data dependency are computed in parallel, while minimizing the filling-in (the number of non-zero entries) in the upper triangular matrix R [10].

Reviewing the toy example, when eliminating x_1, there is no common factor between x_1 and x_4. Therefore, the elimination of x_1 and x_4 is considered simultaneously. The computations involved are similar due to the symmetry of LIO factor graph, as shown in Fig. 3.3(a). Parallel elimination from both sides to the middle could

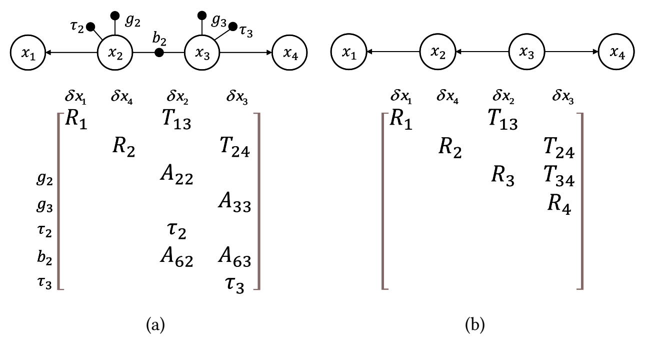

Fig. 3.3: Factor graph and corresponding matrix obtained using parallel elimination. (a) Factor graph and the corresponding matrix after eliminating x_1 and x_4 simultaneously. (b) The Bayes net obtained using parallel elimination and the corresponding upper triangular matrix R.

continue if the chain was longer, but not in this case because x_2 and x_3 have common factor b_2 here. Moreover, the maximum dimension of the matrix $\overline{A_j}$ to be decomposed for parallel elimination is smaller than that for serial elimination, as the number of iterations required for parallel elimination is half that of serial elimination. Parallel elimination also results in a different Bayes net, in which case the root is in the middle and the directed edges point to the variables on both sides, as shown in Fig. 3.3(b). This means that the root variable can be solved first, after which the children variables on both sides can be solved in parallel in turn, e.g. x_2 and x_4 can be solved simultaneously.

The two elimination patterns get the same minimum filling-in, three in this toy example, with different elimination orders. However, the competitiveness of parallel elimination is reflected in: (a) parallelization of the matrix decomposition, (b) dimension reduction of the matrix to be decomposed and (c) parallelization of the back substitution.

Parallel elimination can also be employed for incremental smoothing which recalculates the symmetric LIO factor graph containing the three most recent variable nodes. Therefore, incremental smoothing of LIO factor graph is also accelerated by parallel elimination.

4 HARDWARE ACCELERATOR DESIGN

This section first gives an overview of the hardware architecture (Sec. 4.1). Then, how the key blocks in the hardware architecture are designed by leveraging the data and computation patterns inherent to LIO factor graph is presented, respectively (Sec. 4.2 to Sec. 4.4).

4.1 Hardware Design Overview

Fig. 4.1 shows the hardware architecture of the accelerator, consisting of a series of optimized hardware blocks to accelerate the inference on LIO factor graph while minimizing the area and power consumption through circuit reuse. The main functional blocks of the accelerator include the factor block, the partial-QR decomposition blocks and the back-substitution blocks.

First, the state variables X, measurements Z and covariance matrices Σ are loaded from the input buffer, and then the residuals ϵ_{bi} and the Jacobians A_{bi} are calculated by the factor block. The sensors supported by this hardware architecture are GPS, IMU and LiDAR. A_b and ϵ_b used to construct the system of linear equations are stored in on-chip memory, which leverages the inherent sparsity in LIO factor graph to reduce memory size.

The hardware architecture uses two sets of partial-QR decomposition blocks and back-substitution blocks to accelerate the inference process. After multiple iterations of partial-QR decomposition and back substitution, the increment Δ of the state variables is calculated. In each iteration, Δ is added to X to obtain the updated state $X = X + \Delta$. If the convergence conditions are not met, X is written directly to the input buffer for the next iteration. Otherwise, $X^* = X$ is output as the final result.

4.2 Circuit Reuse

Computation similarities across algorithm modules are exploited to reuse circuits and thereby reduce the resource consumption. Circuit reuse here refers to two aspects: (1) Computation Results Reuse and (2) Functional Units Time-Multiplexing.

Computation Results Reuse Fig. 4.2 shows the architecture of the factor block. There is a significant overlap of general parts in the three factors, which is divided into two levels: between sensors, and within sensors. Although the IMU factor and LiDAR factor are produced by different sensors, they are both constraints on two adjacent keyframes, so there are a large number of same intermediate results in the computing process. Computation overlap also exists in the IMU factor. While calculating the ϵ_{bi} in IMU, some items in A_{bi} can be calculated at the same time. These common computations are finished by the General Calculation unit, avoiding repeated calculations.

Functional Units Time-Multiplexing The factor block has two operating modes, which perform the calculation of ϵ_b, A_b and the cost function, respectively. ϵ_b and A_b need to be constructed when solving Δ in each iteration. However, only ϵ_b^{new} is required to calculate the cost when evaluating whether accepting Δ or not. Therefore, the computations of ϵ_b, A_b and the cost are mapped onto the configurable basic computing units. In Fig. 4.2, solid lines and dashed lines denote the data flow for calculating ϵ_b, A_b and the cost, respectively.

4.3 Storage Optimization

Storage optimization can be divided into three steps. The first step is to store ϵ_b and A_b according to the sparse factor graph structure instead of the original matrix with substantial zero entries. Each factor needs to store its ϵ_{bi} and A_{bi}, in addition to its factor type (to know the dimension of ϵ_{bi} and A_{bi}) and corresponding variable indexes.

However, due to the chain structure of LIO factor graph, the sparsity pattern of the matrix $\overline{A}$ can be determined each time before variable elimination is performed. In other words, the factors and variables involved are fixed in each elimination so the factor types and corresponding variable indexes are always known. Therefore, in the second step, ϵ_{bi} and A_{bi} corresponding to each factor can be stored sequentially without any redundant information.

The third step is to skip the large number of zero and identity entries in A_{bi}, e.g. A_{bi} in IMU factor has a large number of identity matrices and zero entries in fixed positions. Besides, since the partial derivatives of ϵ_{bi} with respect to x_i and x_{i+1} are symmetric

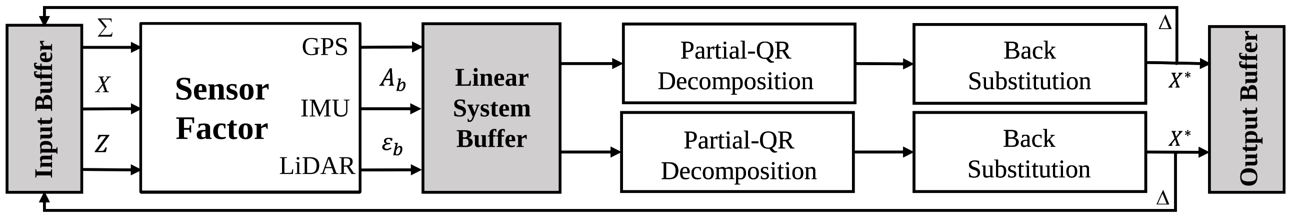

Fig. 4.1: Overall hardware architecture.

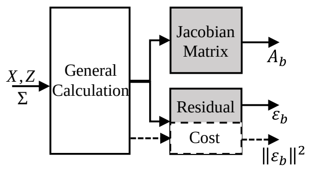

Fig. 4.2: The data flow of the factor block. Solid lines and dashed lines denote the data flow to compute ϵ_b, A_b and the cost, respectively.

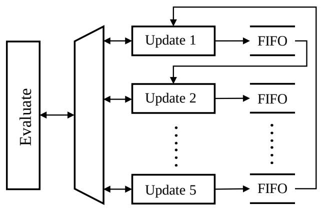

Fig. 4.3: To ensure a balanced pipeline between Evaluate and Update, the QR decomposition block uses one Evaluate unit and multiple (five here) time-multiplexed Update units.

in LiDAR factors, only half of them are stored. The Householder matrix constructed in the QR decomposition (Sec. 4.4) can also take advantage of its symmetry to further reduce memory size.

4.4 Optimization of QR Decomposition Block

The partial-QR decomposition starts from the first column of the $m\times n$ matrix $\overline{A}$, from which the Householder matrix H is constructed (the Evaluate phase). $H\overline{A}$ zeros the entries below the diagonal of the first column, and updates the second to n-th columns (the Update phase). Then the obtained $(m-1)\times(n-1)$ matrix in the lower right corner is the input for the second iteration. The iteration continues until the $\hat{n}$-th column, where $\hat{n}$ represents the dimension of x_i to be eliminated.

After each iteration, the entries below the diagonal are known to be zeros and they are not needed in subsequent calculations. Therefore, the calculations of these zeros entries can be omitted directly so as to improve performance and save resources.

Fig. 4.3 shows the architecture of the block, where each Evaluate-Update pair performs one iteration of the decomposition. The analysis of the fine-grained data dependencies shows that the Evaluate-Update phase can be pipelined. The Evaluate phase of next iteration and the Update phase of current iteration can start at the same time. However, the Update phase is usually more time-consuming than the Evaluate phase. Therefore, this block designs an Evaluate unit and n_u time-multiplexed Update units. The larger the n_u, the better the performance, but the larger the area.

5 PERFORMANCE EVALUATION

To evaluate the proposed accelerator, we conducted a series of experiments. We first introduce the experimental setup (Sec. 5.1). Then the results are presented in three parts. The first part shows the speed and energy of the proposed hardware accelerator compared to software (Sec. 5.2). The second part gives the localization accuracy of the accelerator while running the datasets (Sec. 5.3). The third part performs some comparisons before and after we optimize the hardware based on the data and computation patterns inherent to LIO factor graph (Sec. 5.4).

5.1 Experimental Setup

The accelerator is designed using Vitis-HLS, and synthesized and implemented onto the Xilinx Zynq-7000 SoC ZC706 FPGA using Vivado Design Suite 2021.1. The accelerator operates at a fixed frequency of 143 MHz. The FPGA power consumption is estimated by the Vivado power analysis tool using real workloads under test. All power and resource consumption data are obtained after the design passes the post-layout timing. The accelerator is evaluated on two common datasets: the Walking [18] and the Park [19]. The Walking dataset was collected using a custom-built handheld device on the MIT campus. The Park dataset was collected in a park covered by vegetation, using an unmanned ground vehicle.

A software implementation of localization is used as a baseline, which uses GTSAM [20] to implement factor graph optimization for sensor fusion in Robotics. The software is evaluated on two hardware platforms: the one on the 11th Intel processor that has 16 cores and operates at 2.5 GHz, and the other on the quad-core Arm Cortex-A57 processor on the Nvidia mobile Jetson TX1 platform [21] operating at 1.9 GHz. The Intel CPU power is measured through a power meter and the Arm core power is measured through the power sensing circuitry on TX1.

5.2 Performance Evaluation

We first evaluate the performance and energy efficiency of the accelerator. The proposed accelerator with FLP32 is compared to the Intel CPU implementation and the Arm implementation with FLP64.

By changing the number of Evaluate-Update units in the QR decomposition blocks, we obtain multiple sets of circuits with different performance. Fig. 5.1 shows the speedup of the accelerator with different configurations including n_u and elimination mode over Intel and Arm. The results show that the best performance design achieves 9.3× speedup over Intel and 47.6× speedup over Arm. Performing parallel elimination can improve performance by an average of 1.6× compared to serial elimination. Fig. 5.2 demonstrates the energy efficiency. It shows that the design with the highest performance has an energy reduction of 50.3× over Intel and 16.8× over Arm. Fig. 5.3 shows the latency of the accelerator compared to Intel while running the Park dataset. Batch optimization of LIO

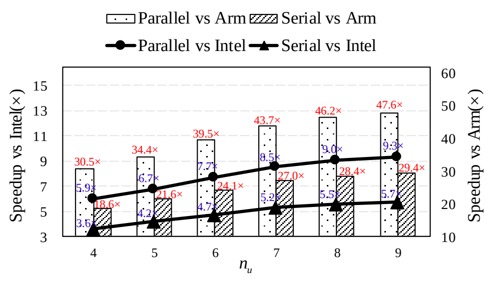

Fig. 5.1: With the increase of n_u, the speedup of the accelerator working in parallel elimination and serial elimination modes compared with Intel and Arm.

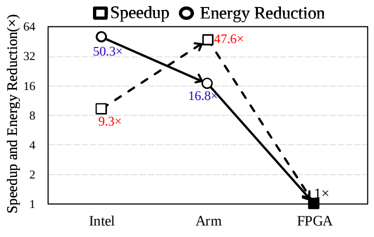

Fig. 5.2: Energy reduction compared to Intel and Arm when the accelerator works at its best performance.

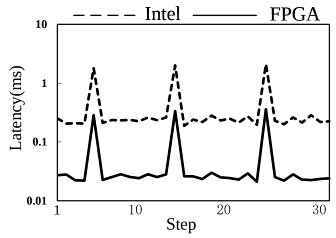

Fig. 5.3: Latency comparison between the accelerator and Intel for factor graph optimization when running the Park dataset.

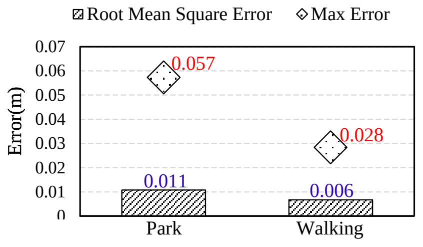

Fig. 5.4: localization accuracy measured by Maximum Error and Root Mean Square Error.

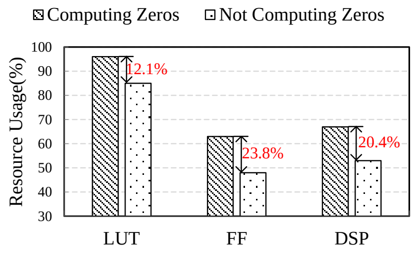

Fig. 5.5: Resource savings from omitting the computation of zeros.

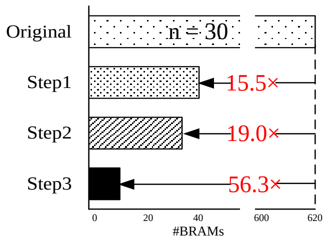

Fig. 5.6: Matrix A_b Storage optimization.

Table 5.1: FPGA resource consumption (utilization percentages and absolute numbers) for parallel and serial elimination modes.

Mode	LUT	FF	BRAM	DSP
Parallel	85% (184990)	48% (211895)	15% (163)	53% (473)
Serial	46% (99733)	28% (120416)	8% (88)	28% (255)

factor graph is performed at steps 5, 15, and 25, and incremental smoothing is performed for the rest.

5.3 Localization Accuracy Evaluation

With the improvement in performance and energy reduction, we are also concerned about the localization accuracy of the accelerator, as illustrated in Fig. 5.4. The localization accuracy is evaluated in terms of two Relative Pose Errors (RPEs): Root Mean Square Error (RMSE) and Maximum Error (ME). The result shows that our accelerator achieves the high localization accuracy compared to the ground truth. On the Walking dataset, its RMSE and ME are 0.6cm and 2.8cm, respectively. On the Park dataset, its RMSE and ME are 1.1cm and 5.7cm, respectively.

5.4 Resource Usage Evaluation

The resource consumption of the best performance design with two elimination modes is shown in Tbl. 5.1. The results show that our design is memory-friendly, but consumes a lot of FFs and LUTs. This is because we store part of the intermediate data on FFs and LUTs instead of BRAMs to improve the performance.

The resources that can be saved by omitting the zeros computation in partial-QR decomposition blocks are shown in Fig. 5.5, and the results indicate that it saves 12.1% LUTs, 23.8% FFs, and 20.4% DSPs in the best performance design.

In Sec. 4.3 we optimize the storage according to the chain structure of LIO factor graph. Fig. 5.6 presents the results of memory saving. It shows that the memory size drops by 15.5×, 19.0× and 56.3× after three steps of optimization with 30 keyframes, respectively.

6 CONCLUSION

Rise of the autonomous machines demands an effective and efficient computer architecture to provide a concise and precise abstraction of the underlying autonomous machine operations, so as to unlock the imagination of AMC application developers. We believe a great candidate for AMC computer architecture is factor graph, which is a graph representing the factorization of a probability distribution function, and has been utilized in many autonomous machine computing functions, such as localization, tracking, planning and control etc.

This paper presents the first work exploring factor graph architecture for autonomous machine computing, starting with LiDAR-Inertial Odometry, a key method in autonomous machine localization. Through exploiting the traits of factor graph computing, we have achieved up to 9× acceleration of autonomous machine localization compared to an advanced Intel CPU, along with 50× improvement of energy efficiency. Based on this initial success, we are going to generalize the factor graph architecture to provide computing support for other autonomous machine functions, including tracking, planning, and control etc.

ACKNOWLEDGMENTS

The authors would like to thank the support of the National Natural Science Foundation of China under Grant U21B2031.

REFERENCES

[1] Shaoshan Liu and Jean-Luc Gaudiot. Rise of the autonomous machines. *Computer*, 55, 2022.

[2] Shaoshan Liu, Zishen Wan, Bo Yu, and Yu Wang. Robotic computing on fpgas. *Synthesis Lectures on Computer Architecture*, 16(1):1–218, 2021.

[3] Shaoshan Liu, Yuhao Zhu, Bo Yu, Jean-Luc Gaudiot, and Guang R Gao. Dataflow accelerator architecture for autonomous machine computing. *arXiv preprint arXiv:2109.07047*, 2021.

[4] Bo Yu, Wei Hu, Leimeng Xu, Jie Tang, Shaoshan Liu, and Yuhao Zhu. Building the computing system for autonomous micromobility vehicles: Design constraints and architectural optimizations. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1067–1081. IEEE, 2020.

[5] Shaoshan Liu, Jie Tang, Zhe Zhang, and Jean-Luc Gaudiot. Computer architectures for autonomous driving. *Computer*, 50(8):18–25, 2017.

[6] Weizhuang Liu, Bo Yu, Yiming Gan, Qiang Liu, Jie Tang, Shaoshan Liu, and Yuhao Zhu. Archytas: A framework for synthesizing and dynamically optimizing accelerators for robotic localization. In *2021 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2021.

[7] Yiming Gan, Yu Bo, Boyuan Tian, Leimeng Xu, Wei Hu, Shaoshan Liu, Qiang Liu, Yanjun Zhang, Jie Tang, and Yuhao Zhu. Eudoxus: Characterizing and accelerating localization in autonomous machines industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 827–840. IEEE, 2021.

[8] Frank Dellaert, Michael Kaess, et al. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017.

[9] Jing Dong, Mustafa Mukadam, Frank Dellaert, and Byron Boots. Motion planning as probabilistic inference using gaussian processes and factor graphs. In *Robotics: Science and Systems*, volume 12, 2016.

[10] Frank Dellaert. Factor graphs: Exploiting structure in robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:141–166, 2021.

[11] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

[12] Vadim Indelman, Stephen Williams, Michael Kaess, and Frank Dellaert. Factor graph based incremental smoothing in inertial navigation systems. In *2012 15th International Conference on Information Fusion*, pages 2154–2161. IEEE, 2012.

[13] Han-Pang Chiu, Stephen Williams, Frank Dellaert, Supun Samarasekera, and Rakesh Kumar. Robust vision-aided navigation using sliding-window factor graphs. In *2013 IEEE International Conference on Robotics and Automation*, pages 46–53. IEEE, 2013.

[14] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011.

[15] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[16] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 5135–5142. IEEE, 2020.

[17] Frank Dellaert Michael Kaess, Ananth Ranganathan. iSAM: Incremental Smoothing and Mapping. *IEEE Transactions on Robotics*, 24:1365 – 1378, 2008.

[18] Shan, Tixiao. the Walking dataset. https://github.com/TixiaoShan/LIO-SAM. Accessed: 2022-07-21.

[19] Shan, Tixiao. the Park dataset. https://github.com/TixiaoShan/LIO-SAM. Accessed: 2022-07-21.

[20] Georgia Institute of Technology. GTSAM. https://github.com/borglab/gtsam. Accessed: 2022-07-21.

[21] NVIDIA. TX1 datasheet. http://images.nvidia.com/content/tegra/embedded-systems/pdf/JTX1-Module-Product-sheet.pdf. Accessed: 2022-07-21.