CICC

IEEE Custom Integrated Circuits Conference

9-2: An Energy-Efficient and Runtime-Reconfigurable FPGA-Based Accelerator for Robotic Localization Systems

Qiang Liu[1,], __Zishen Wan__[2,*], Bo Yu[3,*], Weizhuang Liu[1], Shaoshan Liu[3], Arijit Raychowdhury[2]*

** Equally Contributed Authors*

[1] Tianjin University, China [2] __Georgia Institute of Technology, USA__ [3] PerceptIn, USA

April 25, 2022

IEEE SOLID-STATE CIRCUITS SOCIETY™ | IEEE

Bio

Email: zishenwan@gatech.edu

Homepage: https://zishenwan.github.io

- **Speaker: Zishen Wan**
  - **PhD Student in Georgia Tech (20Fall-Now)**
    - **Advisor: Prof. Arijit Raychowdhury**
  - MS in Harvard University
    - Advisor: Prof. Vijay Janapa Reddi
  - BS in Harbin Institute of Technology

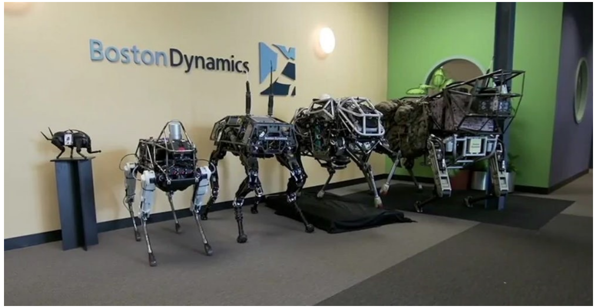
- **Research Interest**
  - VLSI, computer architecture, edge computing.
  - Efficient and resilient hardware and system design for autonomous machines.

Motivation: Autonomous Systems

Drones

Self-Driving Cars

Robots

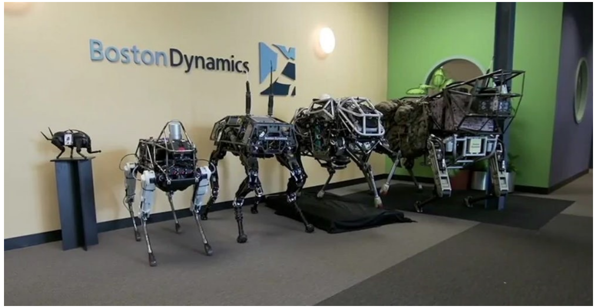


Motivation: Autonomous Systems

Drones

Self-Driving Cars

Robots

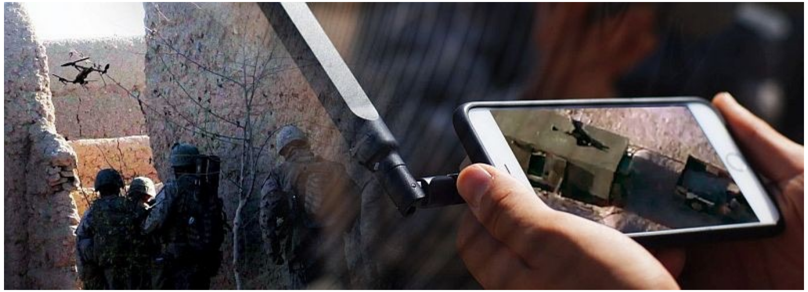


Applications

Search & Rescue

Package Delivery

Surveillance



[Source: V. Sze]

How Does Autonomous System Work?

Perception

Localization

Motion Planning

Control

How Does Autonomous System Work?

Perception

Localization

Motion Planning

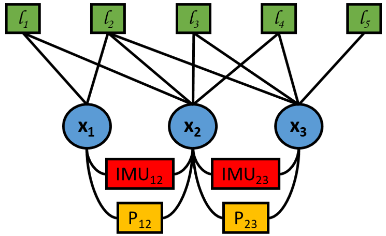
Control

Localization is computationally intensive

[Wan, Circuits and Systems Magazine 2021]
[Wan, Synthesis Lectures on Comp Arch 2021]

Challenges

Large Factor Graph:

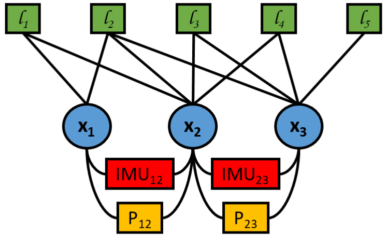


4000+ factors

[Wan, Circuits and Systems Magazine 2021]
[Wan, Synthesis Lectures on Comp Arch 2021]

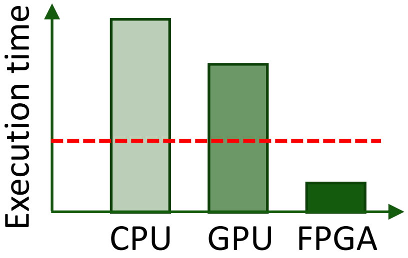
Challenges

Large Factor Graph:



4000+ factors

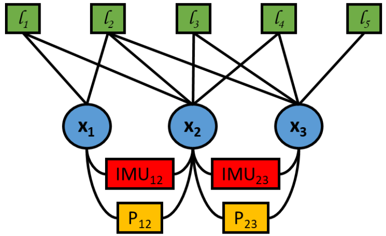
Real-Time Requirement:



[Wan, Circuits and Systems Magazine 2021]
[Wan, Synthesis Lectures on Comp Arch 2021]

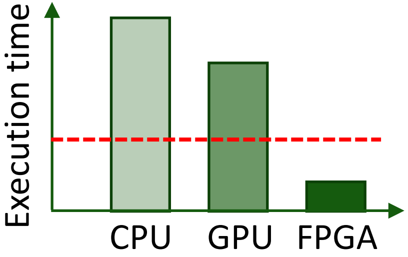
Challenges

Large Factor Graph:



4000+ factors

Real-Time Requirement:

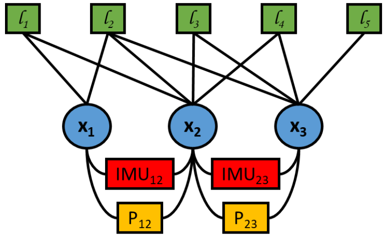


Low Power Budget:

Big battery
CPU/GPU: 10-100W

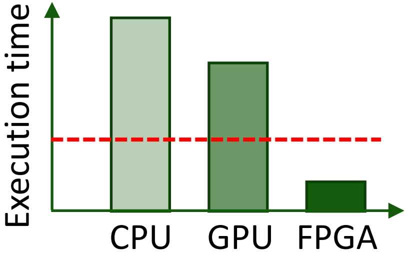
Challenges

Large Factor Graph:



4000+ factors

Real-Time Requirement:

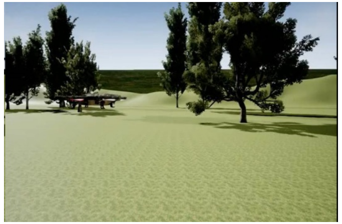


Low Power Budget:

Big battery
CPU/GPU: 10-100W

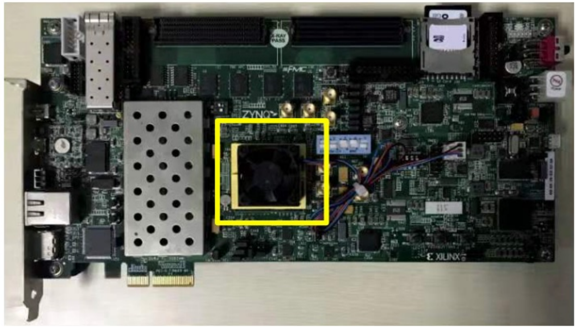
Dynamic Changing Environments:

Sparse



Dense

Energy-Efficient Localization and Mapping



FPGA Zynq-7000 SoC ZC706
with XC7Z045 FFG900-2

- Energy-efficient & real-time localization and mapping
- Dynamic reconfiguration at runtime
- Real-time performance of 61 fps at 3.45W (56mJ/frame)

Outline

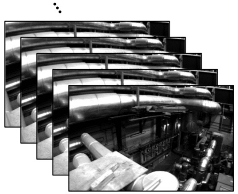
- SLAM: Simultaneously Localization & Mapping
- Hardware Architecture
- Main Contributions
- Evaluations and Comparisons
- Summary

Outline

- SLAM: Simultaneously Localization & Mapping
- Hardware Architecture
- Main Contributions
- Evaluations and Comparisons
- Summary

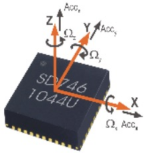
Localization and Mapping Using SLAM

Camera



Feature Tracks

IMU
(Inertial Measurement Unit)



Estimated States

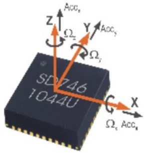
[Source: V. Sze]

Localization and Mapping Using SLAM

Camera

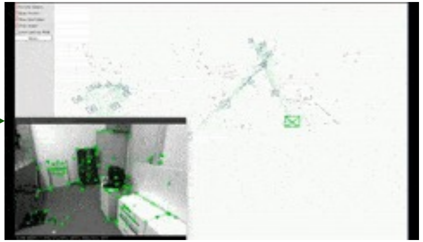
Feature Tracks

IMU
(Inertial Measurement Unit)

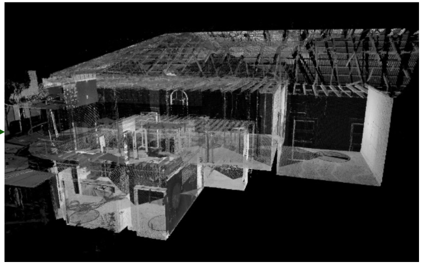


Estimated States

Simultaneous Localization and Mapping (SLAM)



Localization
(6 DoF poses)



Mapping
(3D coordinates)

[Source: V. Sze]

Localization and Mapping Using SLAM

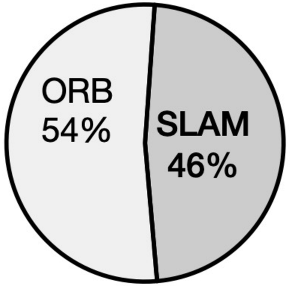
Camera

IMU
(Inertial Measure

SLAM is computationally intensive:

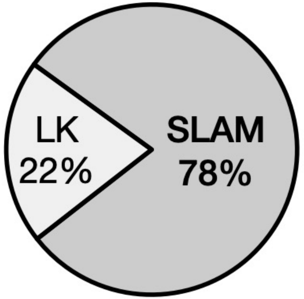
ORB-SLAM

FrontEnd: ORB
BackEnd: SLAM

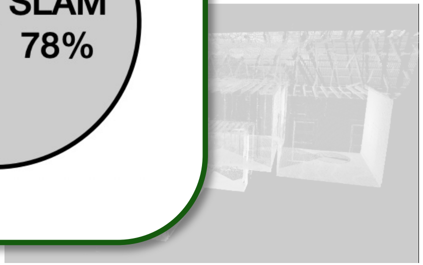


LK-SLAM

FrontEnd: LK
BackEnd: SLAM



ization
poses)



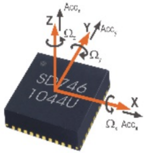
Mapping
(3D coordinates)

How Does SLAM Work?

Camera

Feature Tracks

IMU
(Inertial Measurement Unit)



Estimated States

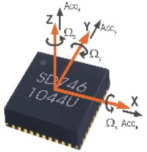
How Does SLAM Work?

Camera

Feature Tracks

IMU

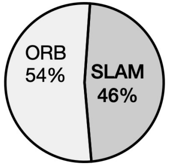
(Inertial Measurement Unit)



Estimated States

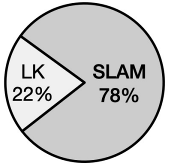
ORB-SLAM

FrontEnd: ORB
BackEnd: SLAM



LK-SLAM

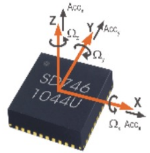
FrontEnd: LK
BackEnd: SLAM



How Does SLAM Work?

Camera

IMU
(Inertial Measurement Unit)



Feature Tracks

Estimated States

Maximum a Posteriori (MAP) Estimation

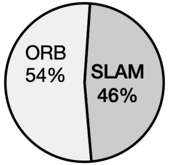
State

Nonlinear least squares (NLS) optimization problem:

$$\min_{\mathbf{p}}\{\sum_{i=1}^{N}||\mathbf{r}_p - \mathbf{H}_p\mathbf{p}||^2 + ||\mathbf{o}_i - \mathcal{P}_i(\mathbf{p})||^2_{C_i}\}$$

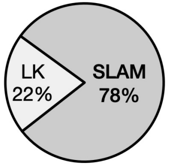
ORB-SLAM

FrontEnd: ORB
BackEnd: SLAM



LK-SLAM

FrontEnd: LK
BackEnd: SLAM



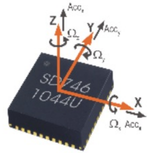
State:
6 DoF poses (location)
3D coordinates (mapping)

CICC

How Does SLAM Work?

Camera

IMU
(Inertial Measurement Unit)



Feature Tracks

Estimated States

Marginalization

Maximum a Posteriori (MAP) Estimation

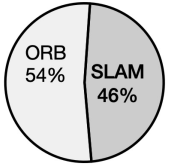
State

Nonlinear least squares (NLS) optimization problem:

$$\min_{\mathbf{p}}\{\sum_{i=1}^{N} ||\mathbf{r}_p - \mathbf{H}_p\mathbf{p}||^2 + ||\mathbf{o}_i - \mathcal{P}_i(\mathbf{p})||^2_{C_i}\}$$

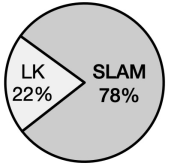
ORB-SLAM

FrontEnd: ORB
BackEnd: SLAM



LK-SLAM

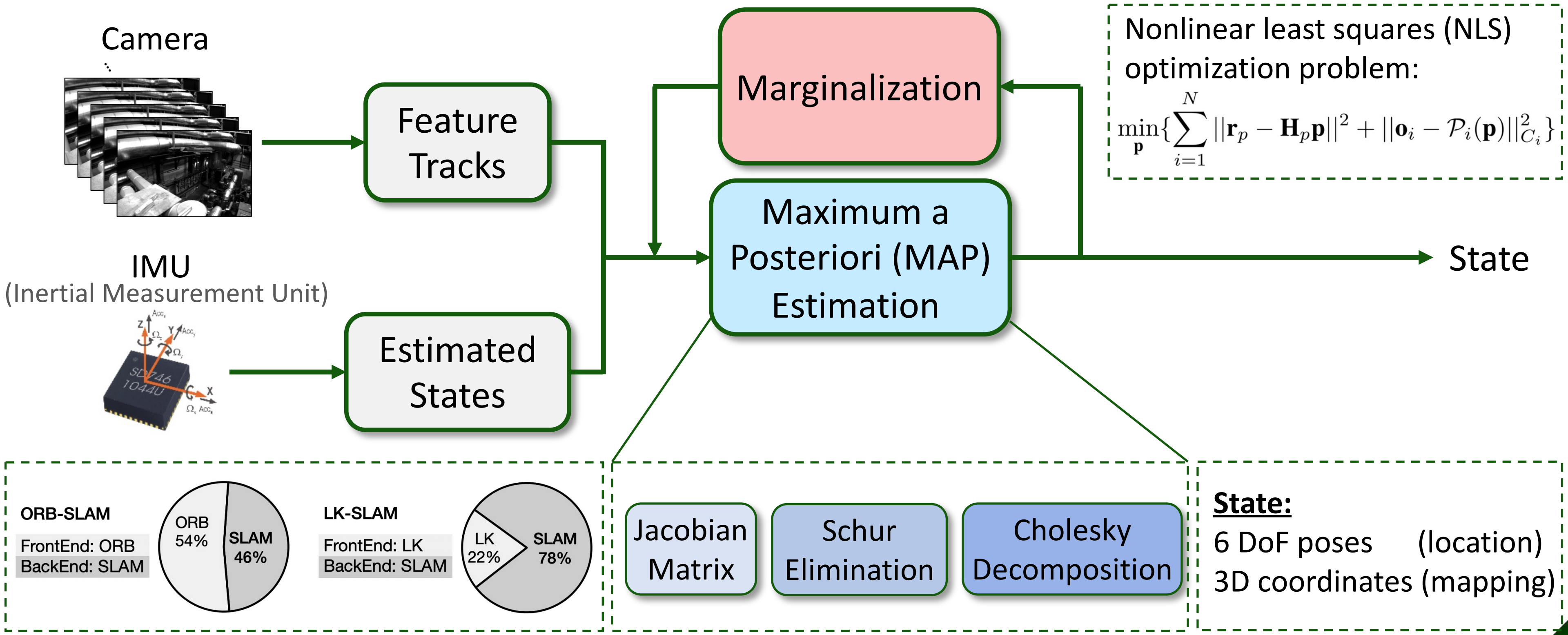
FrontEnd: LK
BackEnd: SLAM



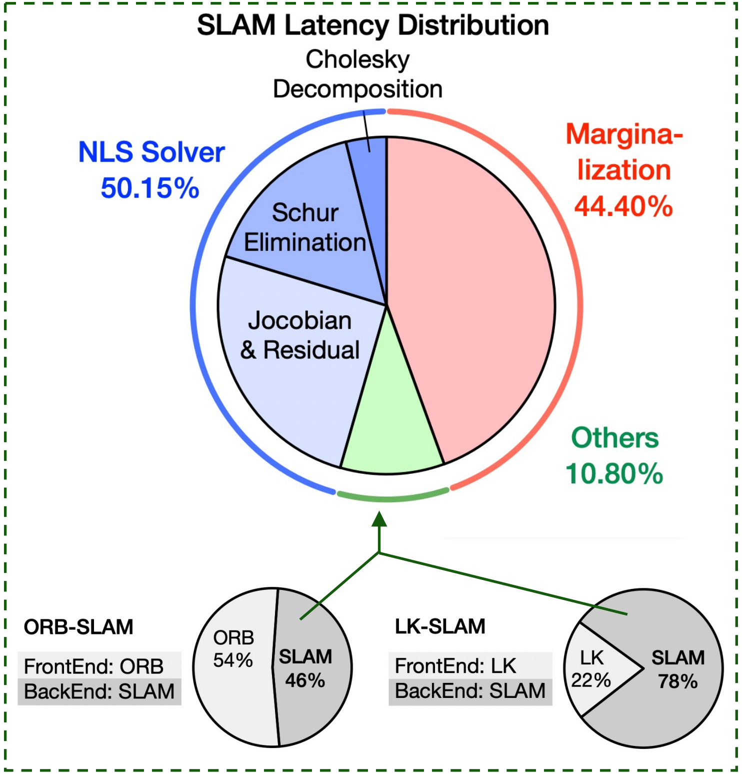
State:
6 DoF poses (location)
3D coordinates (mapping)

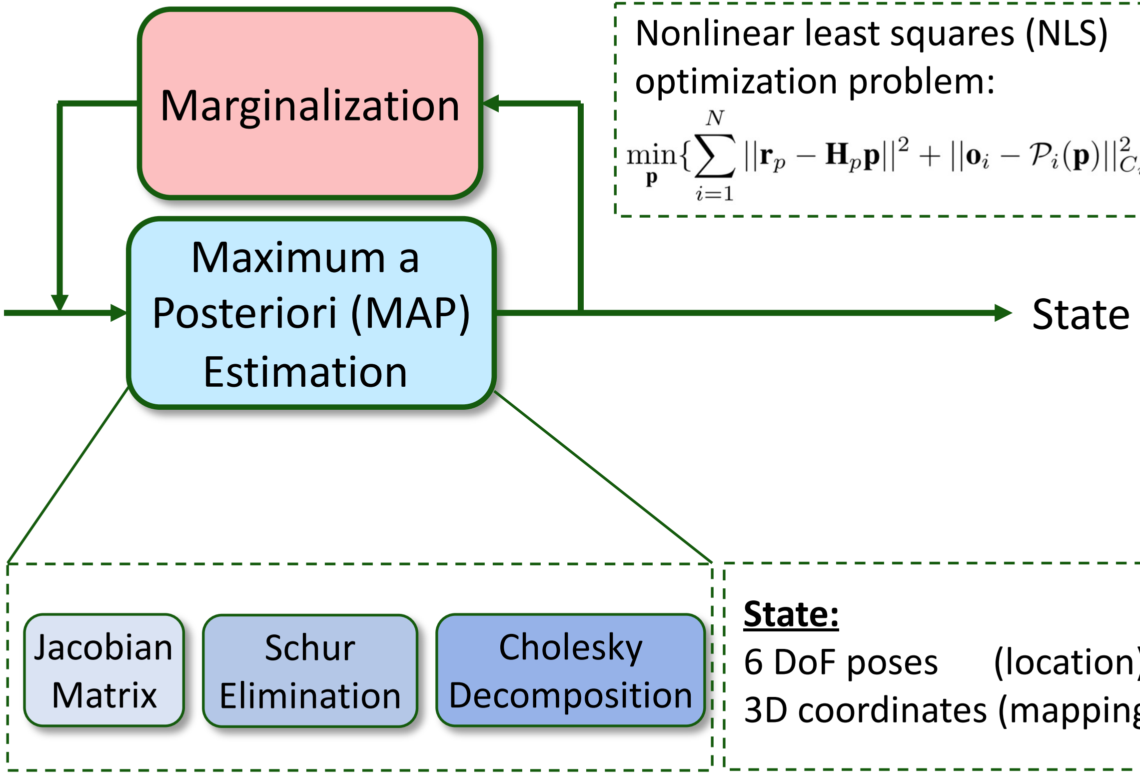
CICC

How Does SLAM Work?



How Does SLAM Work?





Nonlinear least squares (NLS) optimization problem:

$$\min_{\mathbf{p}}\{\sum_{i=1}^{N}||\mathbf{r}_p - \mathbf{H}_p\mathbf{p}||^2 + ||\mathbf{o}_i - \mathcal{P}_i(\mathbf{p})||^2_{C_i}\}$$

State:

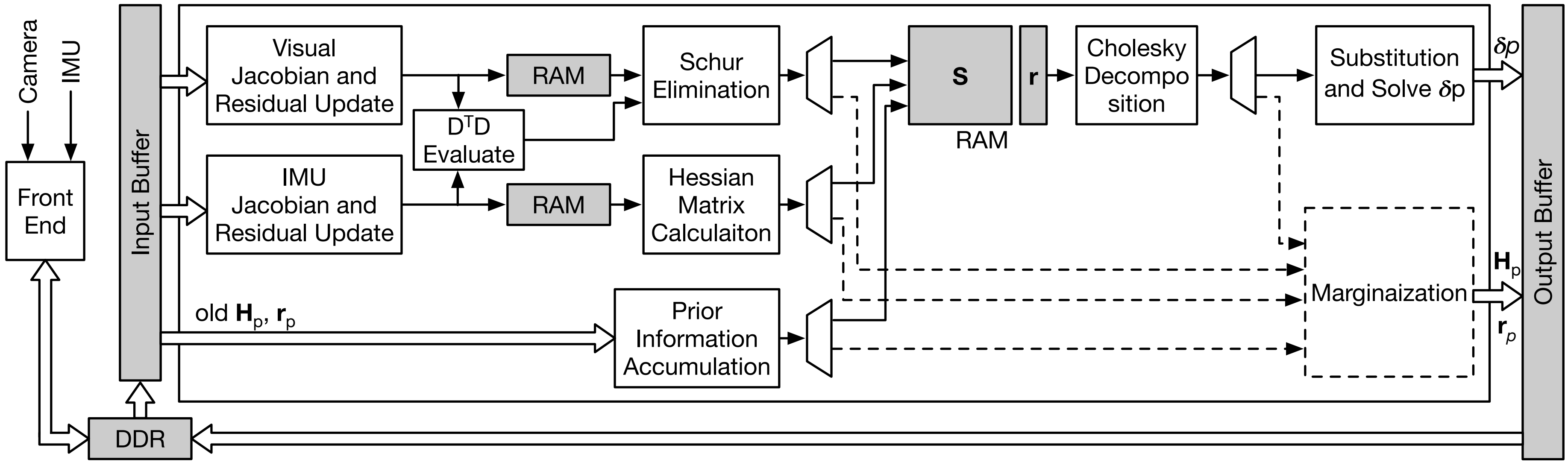
6 DoF poses (location)

3D coordinates (mapping)

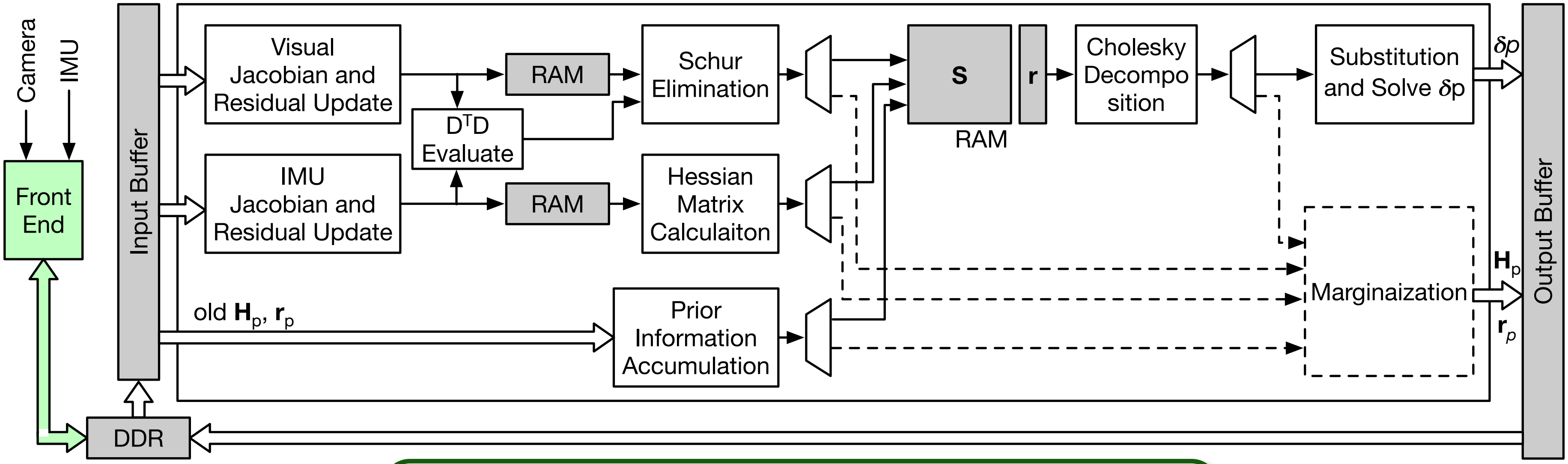
Outline

- SLAM: Simultaneously Localization & Mapping
- Hardware Architecture
- Main Contributions
- Evaluations and Comparisons
- Summary

Hardware Architecture - Overview



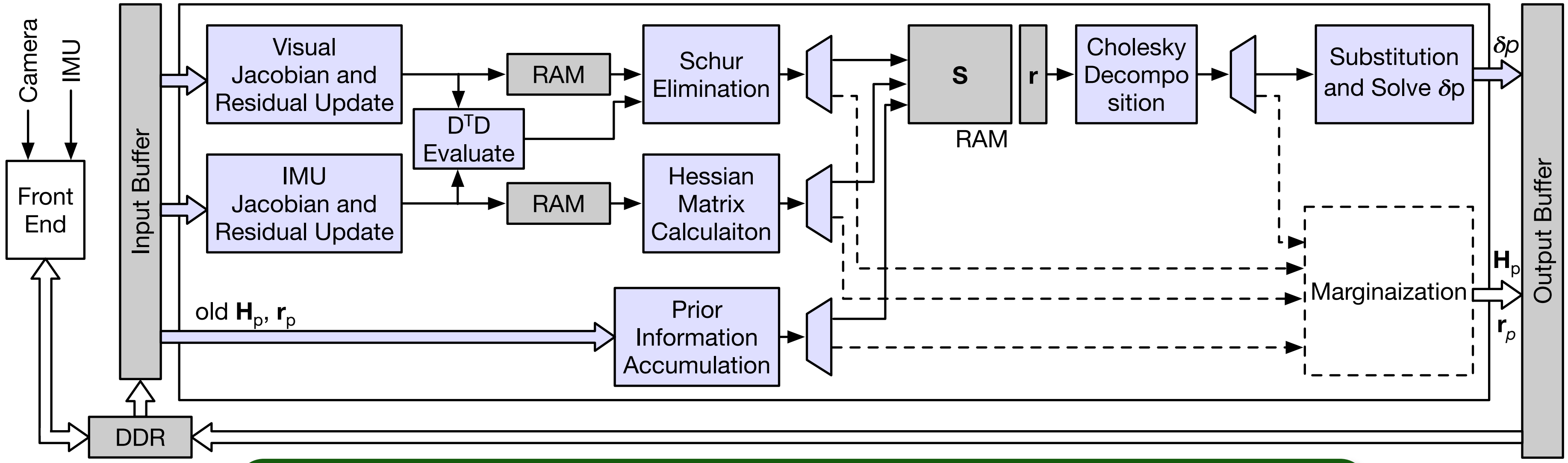
Hardware Architecture – Perception



Sensor Input:

Camera + IMU, process in host

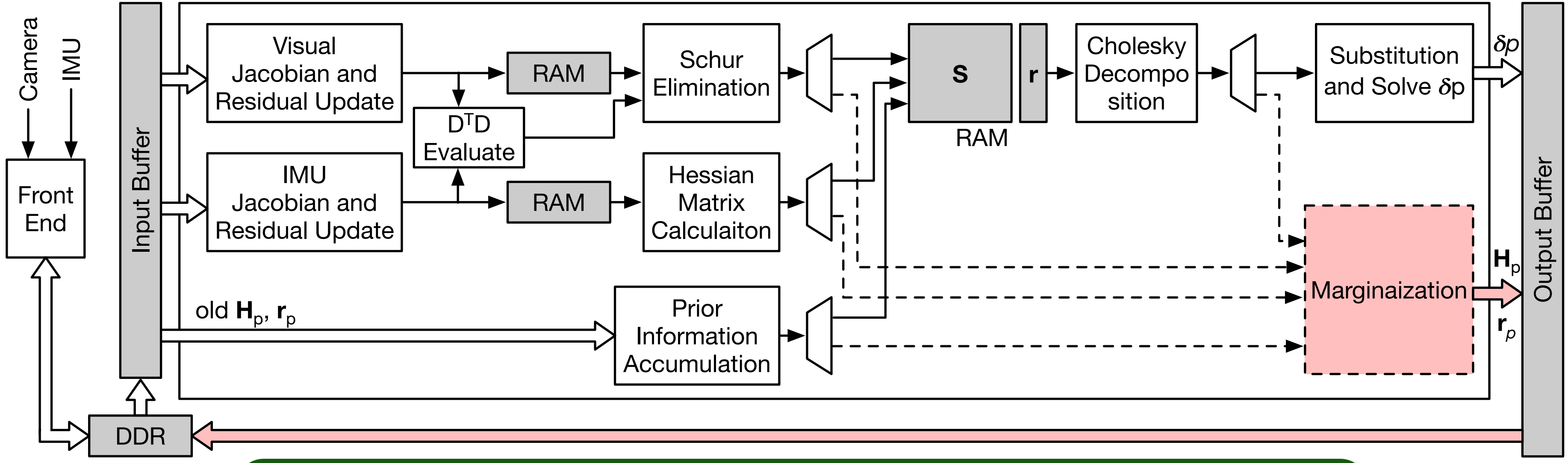
Hardware Architecture – SLAM (NLS Optimization)



SLAM Nonlinear Least Squares (NLS) Optimization:

Jacobian, Schur elimination, Cholesky Decomposition, etc

Hardware Architecture – SLAM Marginalization



SLAM Marginalization:

Jacobian, Schur elimination, Cholesky Decomposition, etc

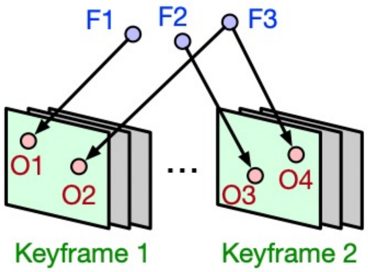
Outline

- SLAM: Simultaneously Localization & Mapping
- Hardware Architecture
- Main Contributions
- Evaluations and Comparisons
- Summary

Method 1

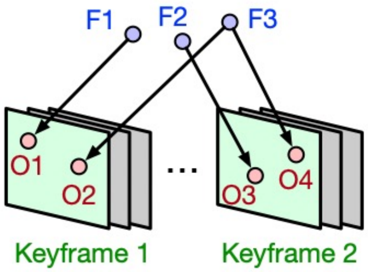
Data Reuse

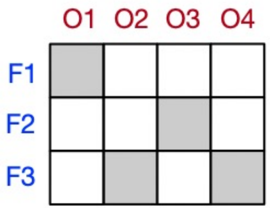
Data Reuse & Design Hierarchy



2 **K**eyframes
3 **F**eature Points (F1~F3)
4 **O**bservations (O1~O4)

Data Reuse & Design Hierarchy



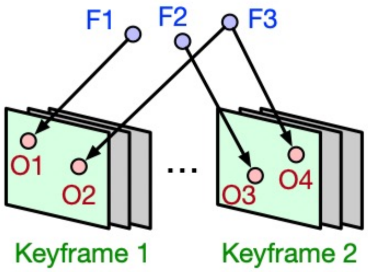


Jacobian Matrix

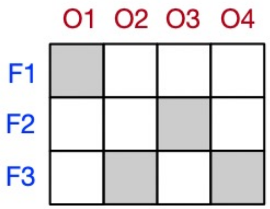
2 **K**eyframes
3 **F**eature Points (F1~F3)
4 **O**bservations (O1~O4)

<feature point, observation>
pairs have non-zero values

Data Reuse & Design Hierarchy

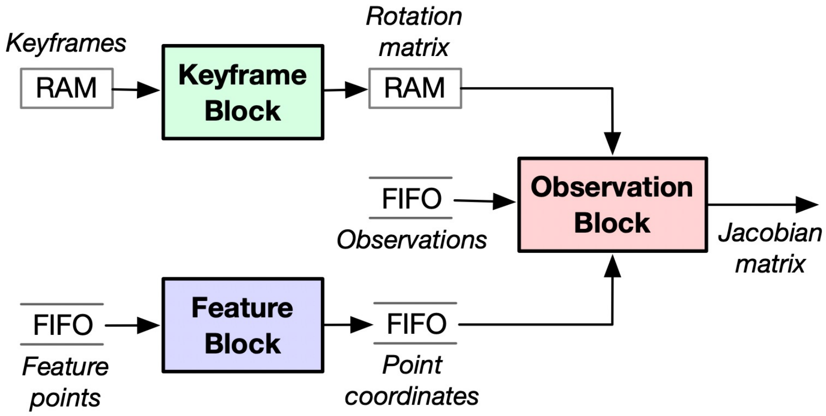


2 **K**eyframes
3 **F**eature Points (F1~F3)
4 **O**bservations (O1~O4)

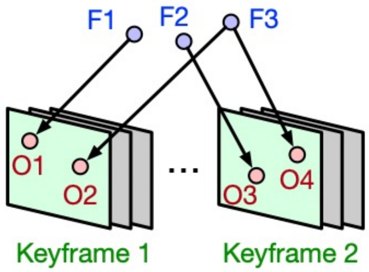


Jacobian Matrix

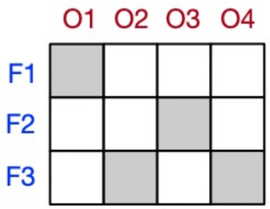
<feature point, observation> pairs have non-zero values



Data Reuse & Design Hierarchy

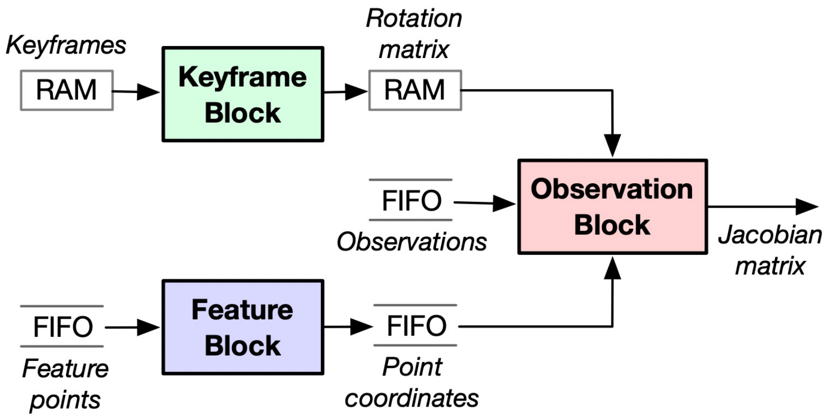


2 **K**eyframes
3 **F**eature Points (F1~F3)
4 **O**bservations (O1~O4)



Jacobian Matrix

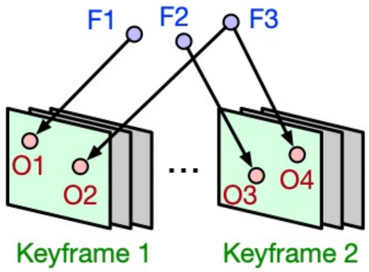
<feature point, observation> pairs have non-zero values



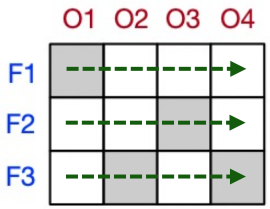
<u>Three-Level Block Designs</u>:

- Keyframe-level: Rotation matrix of keyframes
- Feature-level: 3D coordinates
- Observation-level: Jacobian matrix

Data Reuse & Design Hierarchy

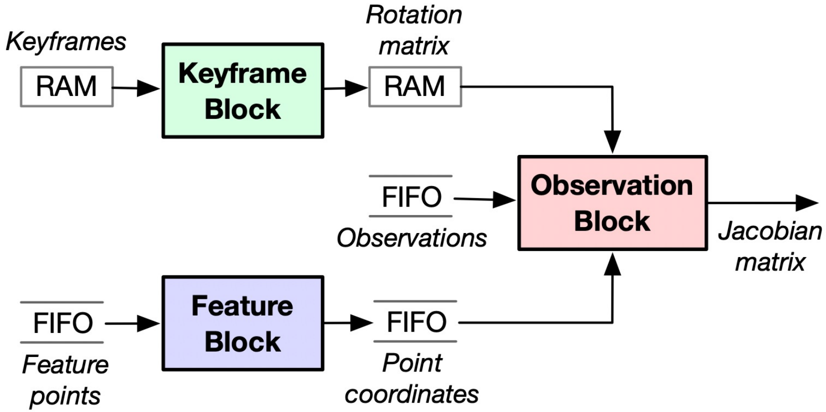


2 **K**eyframes
3 **F**eature Points (F1~F3)
4 **O**bservations (O1~O4)



Jacobian Matrix

<feature point, observation> pairs have non-zero values



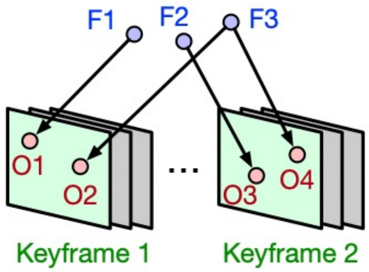
<u>Two-Level Data Reuses</u>:

- Feature-reuse: across associated observations
- Keyframe-reuse: over all obsn. within keyframe

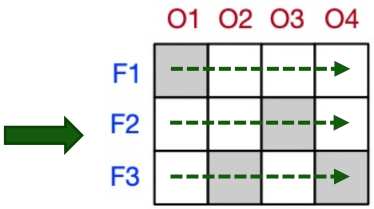
<u>Three-Level Block Designs</u>:

- Keyframe-level: Rotation matrix of keyframes
- Feature-level: 3D coordinates
- Observation-level: Jacobian matrix

Data Reuse & Design Hierarchy

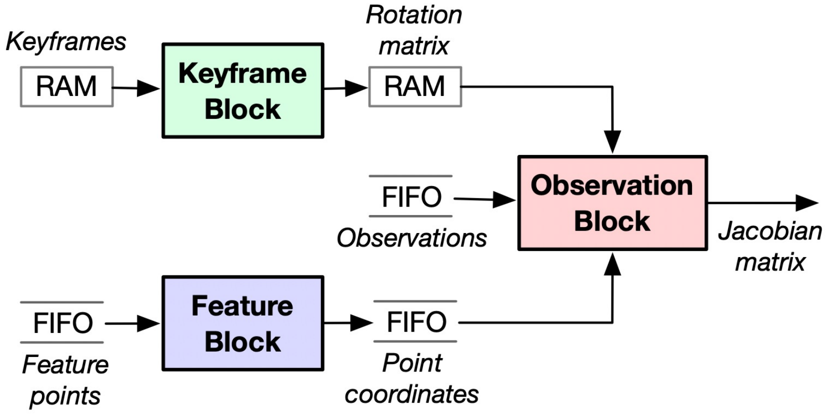


2 **K**eyframes
3 **F**eature Points (F1~F3)
4 **O**bservations (O1~O4)



Jacobian Matrix

<feature point, observation> pairs have non-zero values



Two-Level Data Reuses:

- Feature-reuse: across associated observations
  ➡ feature (row)-stationary
- Keyframe-reuse: over all obsn. within keyframe

Three-Level Block Designs:

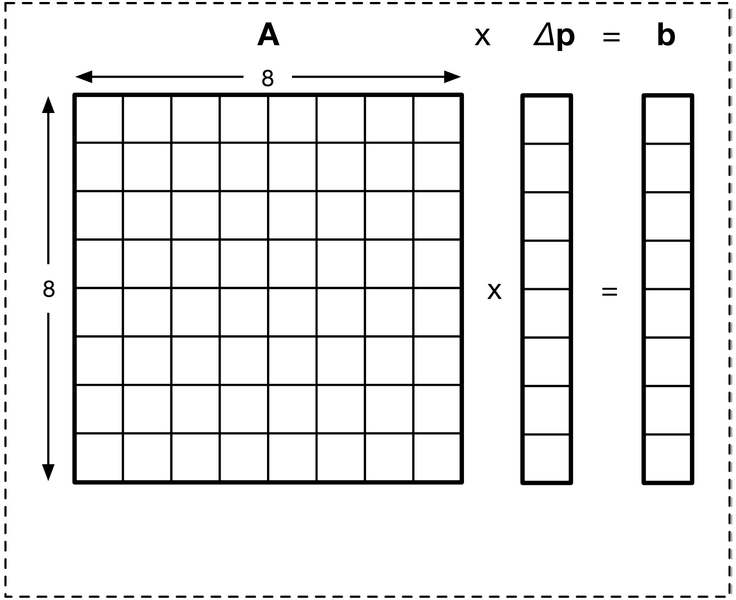
- Keyframe-level: Rotation matrix of keyframes
- Feature-level: 3D coordinates
- Observation-level: Jacobian matrix

Method 2

Symmetry & Sparsity

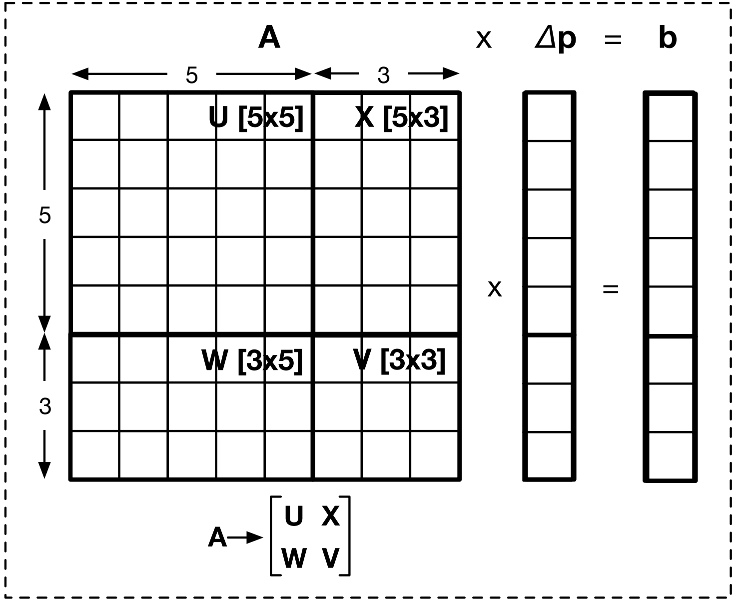
Diagonal Computation + Symmetry + Hardware Reuse

Shure Elimination:



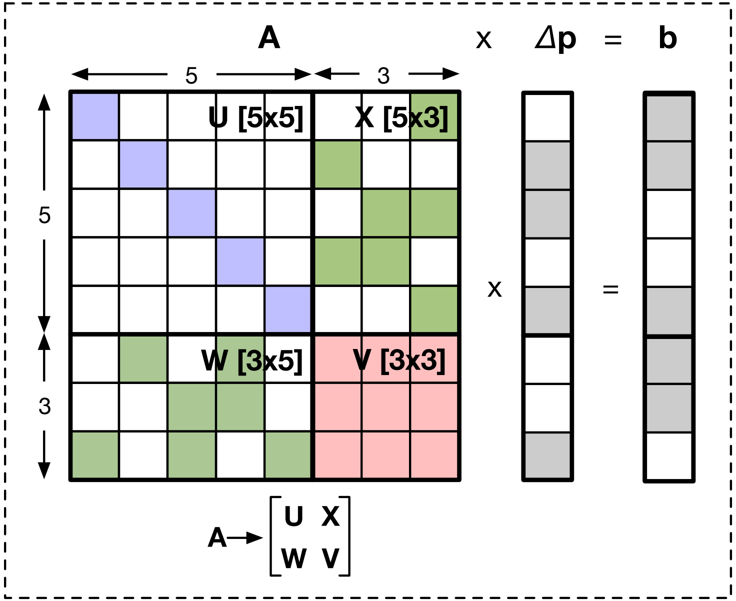
Diagonal Computation + Symmetry + Hardware Reuse

Shure Elimination:



Diagonal Computation + Symmetry + Hardware Reuse

Shure Elimination:



Make U as diagonal matrix:

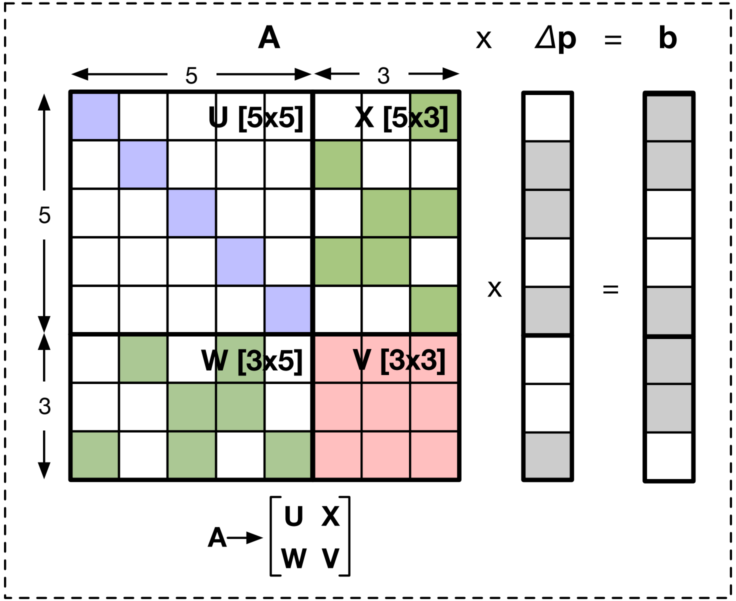
***O*(n³)->*O*(n)** computational complexity

X becomes the transpose of W:

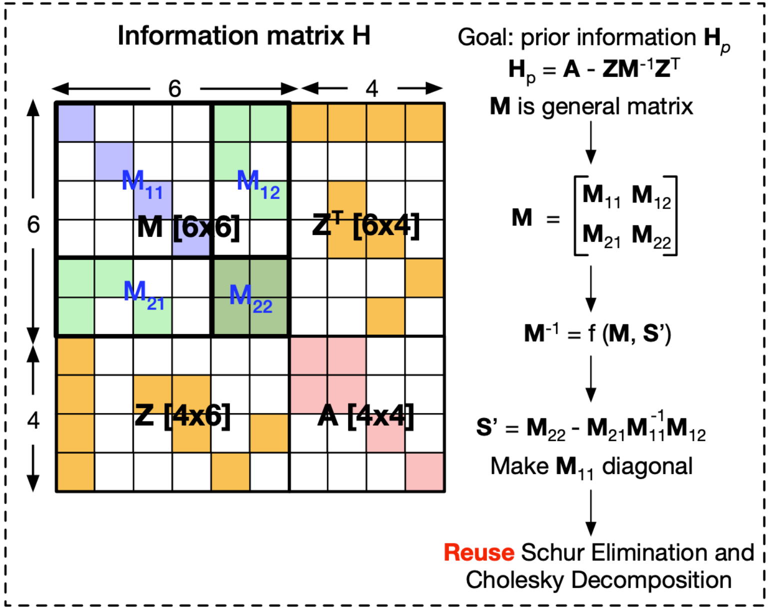
1.34x on-chip memory reduction

Diagonal Computation + Symmetry + Hardware Reuse

Shure Elimination:

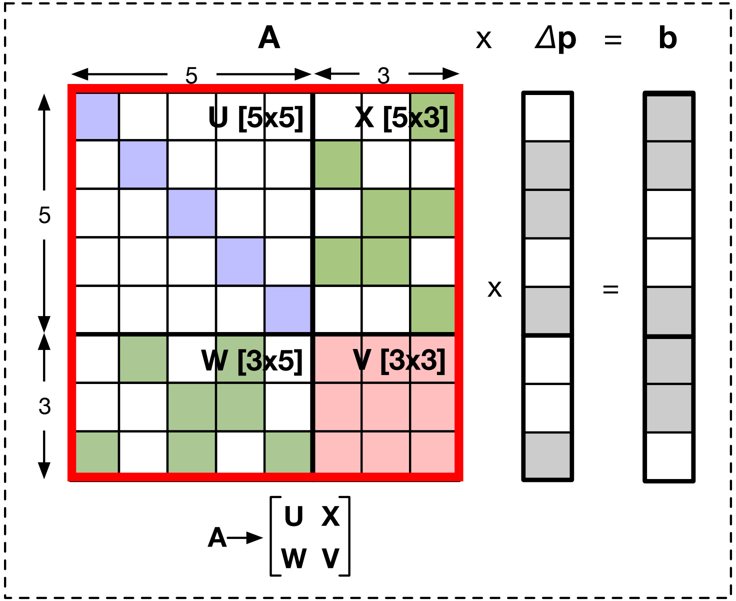


Marginalization:

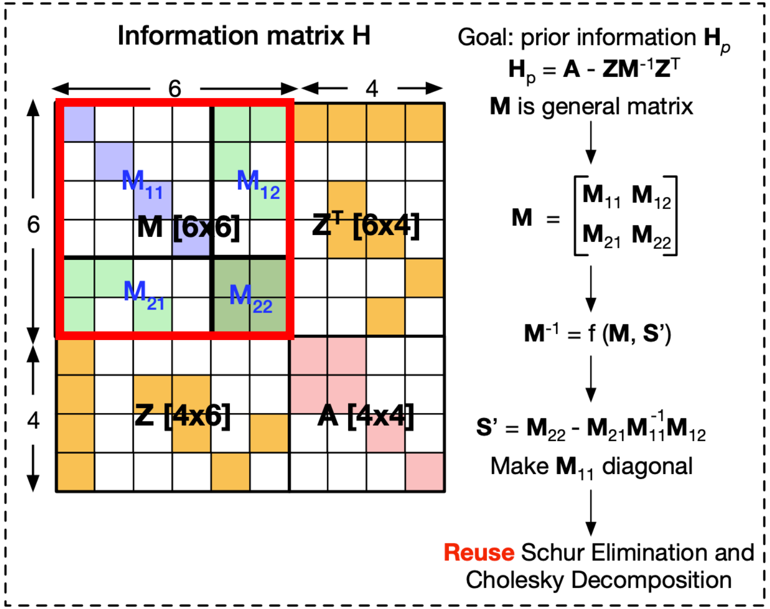


Diagonal Computation + Symmetry + Hardware Reuse

Shure Elimination:



Marginalization:



Diagonal Computation + Symmetry + Hardware Reuse

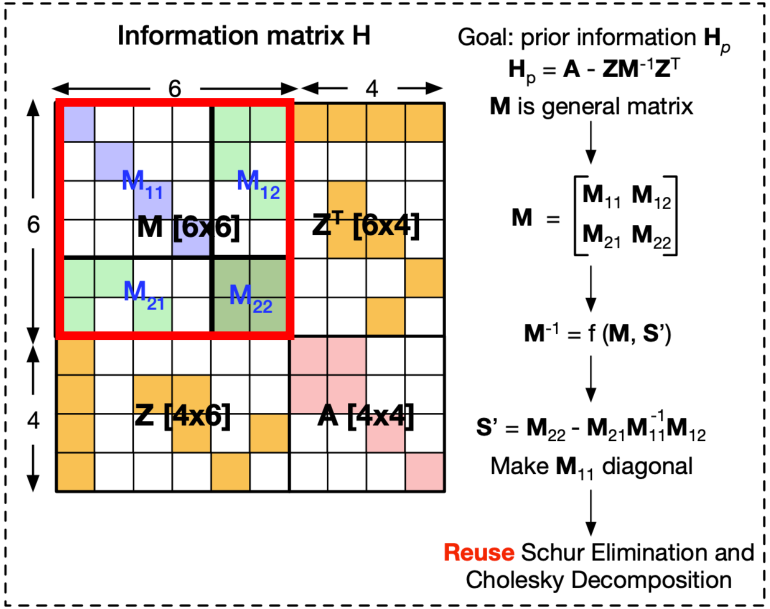
Make M as diagonal matrix:

$O(n^3)$->$O(n)$ computational complexity

Reuse Schur Elimination circuit in Marginalization:

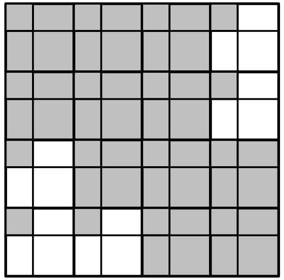
Reduce resource consumption without performance degradation

Marginalization:



Data Layout + Symmetry + Sparsity

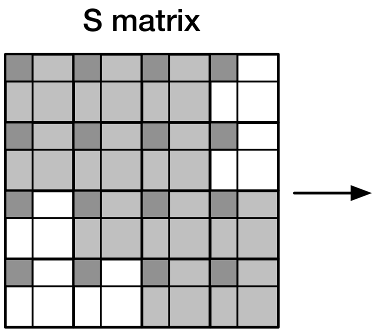
S matrix

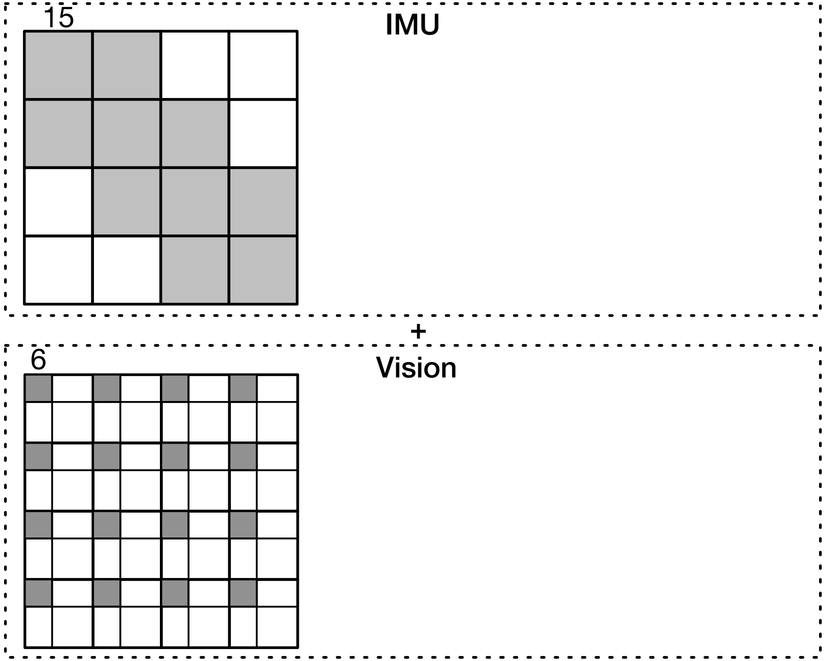


S matrix: store the parameters
for the system
(40%-80% of total storage)

720 kb

Data Layout + Symmetry + Sparsity

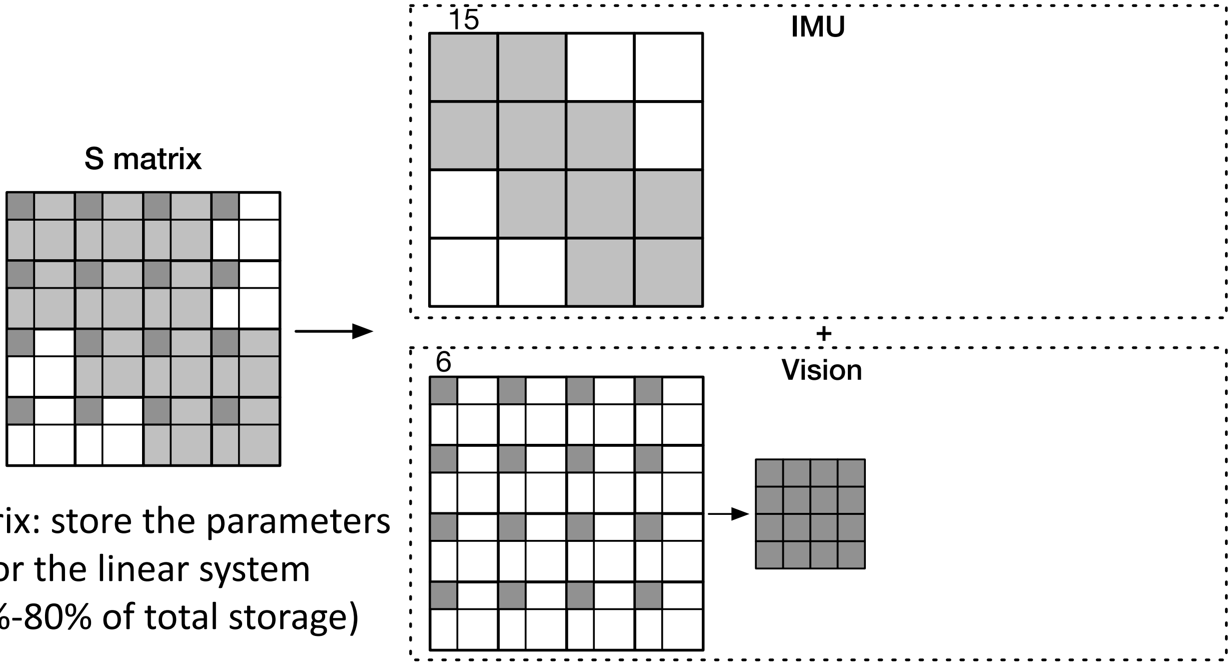




S matrix: store the parameters for the linear system (40%-80% of total storage)

720 kb

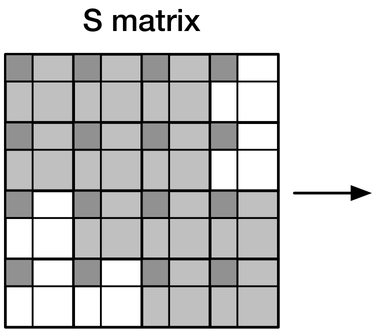
Data Layout + Symmetry + Sparsity

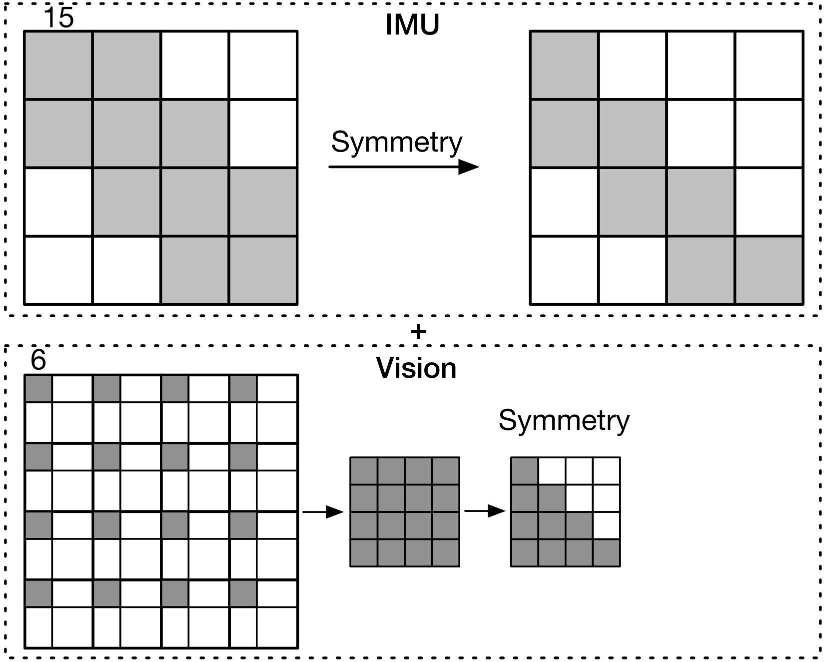


S matrix: store the parameters for the linear system (40%-80% of total storage)

720 kb

Data Layout + Symmetry + Sparsity

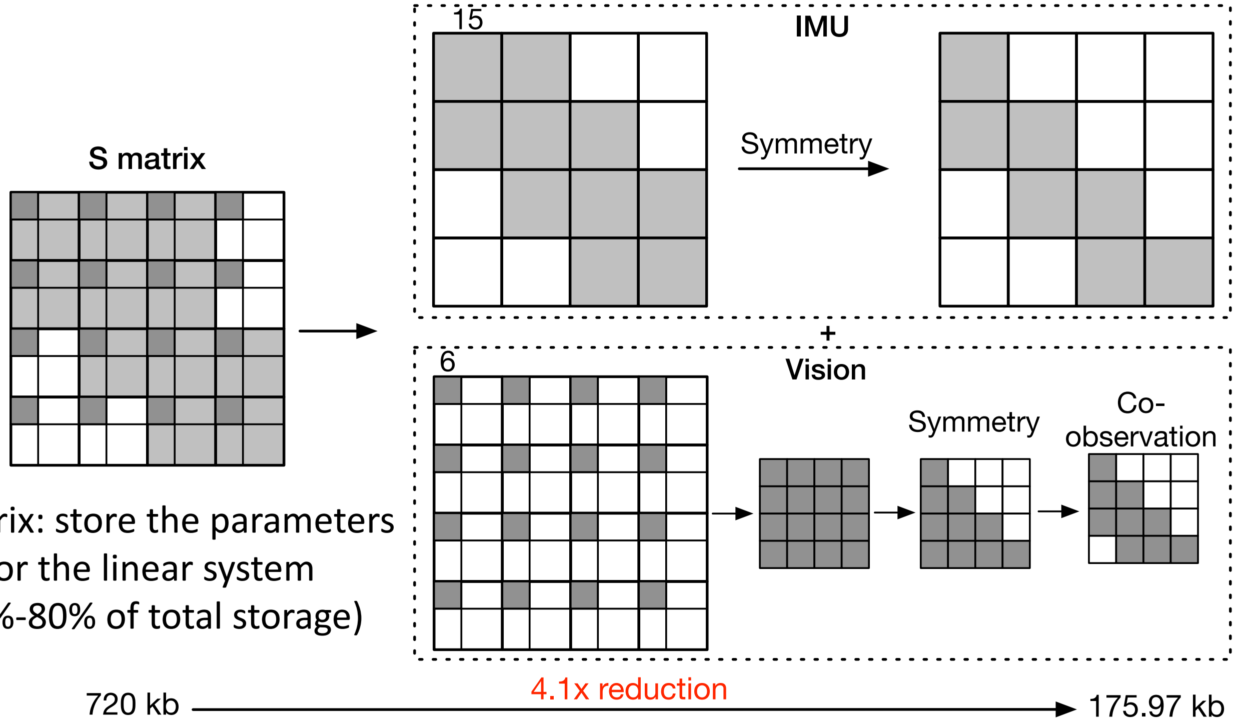




S matrix: store the parameters for the linear system (40%-80% of total storage)

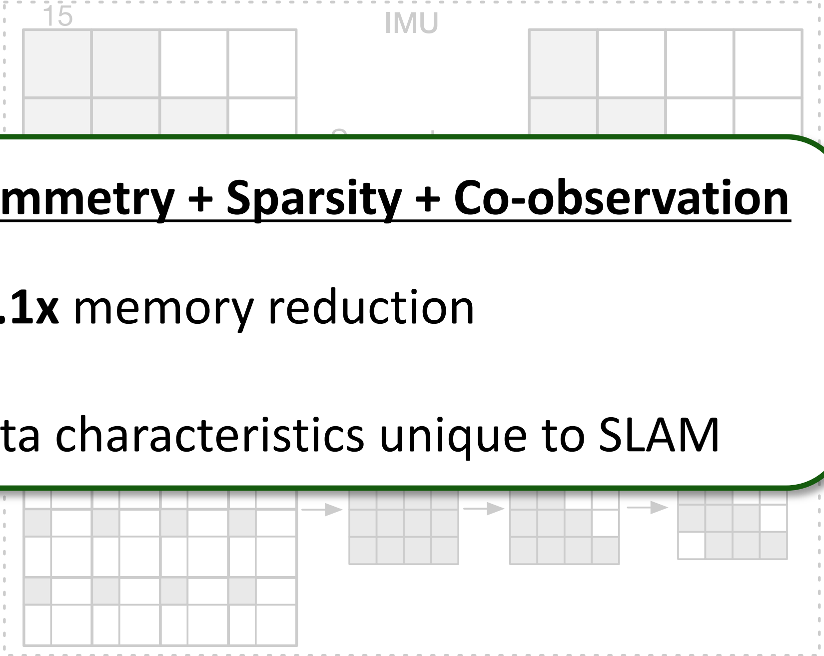
720 kb

Data Layout + Symmetry + Sparsity



S matrix: store the parameters for the linear system (40%-80% of total storage)

Data Layout + Symmetry + Sparsity



Data Layout + Symmetry + Sparsity + Co-observation

4.1x memory reduction

Exploiting data characteristics unique to SLAM

S matrix: store the parameters for the linear system (40%-80% of total storage)

720 kb → 175.97 kb

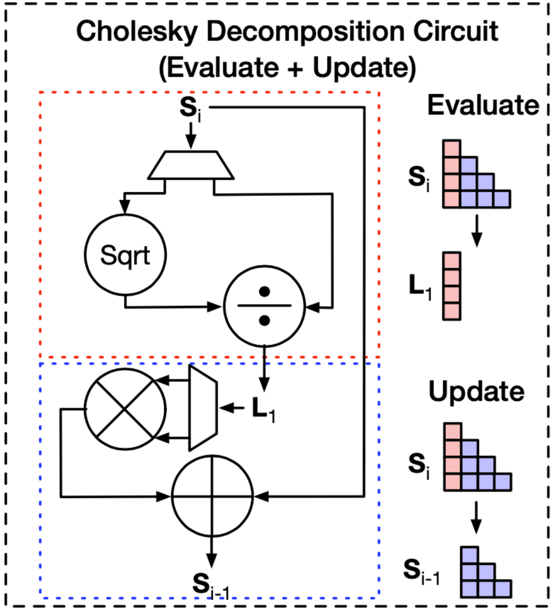
4.1x reduction

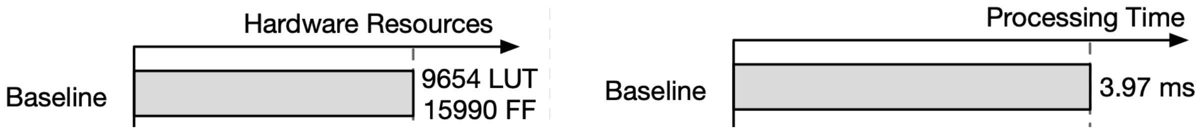
Method 3

Time-Multiplex & Pipeline

Time-Multiplexed + Pipeline Processing

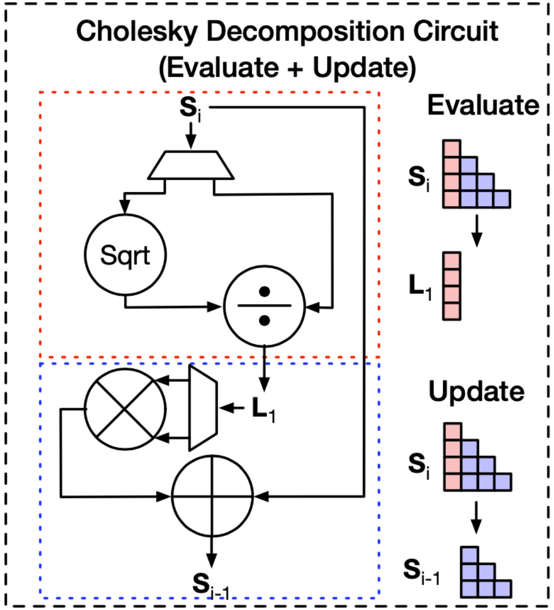
Cholesky decomposition: $\boldsymbol{S} = \boldsymbol{L}\boldsymbol{L}^T$ ($\boldsymbol{S}$: symmetric matrix; $\boldsymbol{L}$: lower triangular matrix)

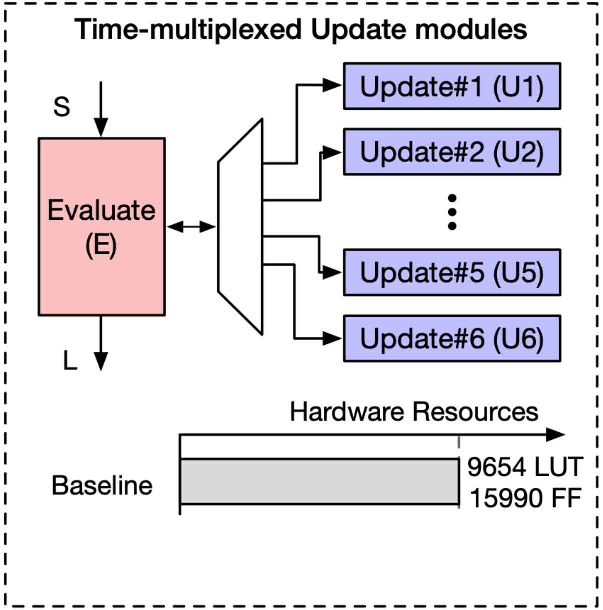


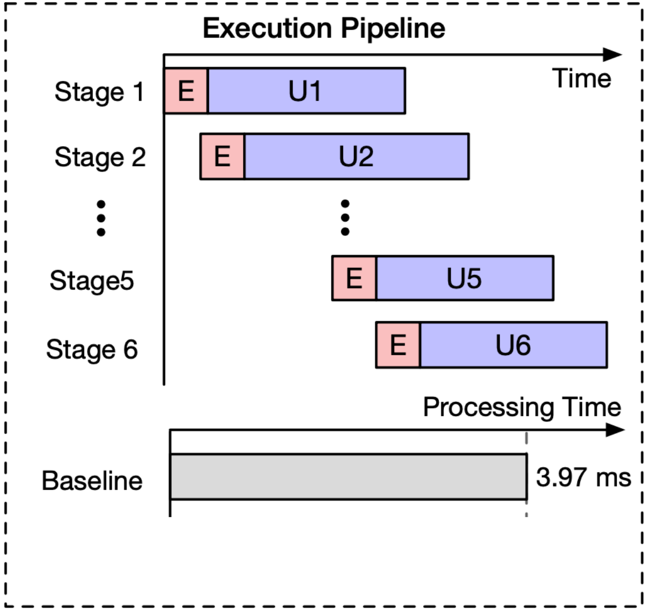


Time-Multiplexed + Pipeline Processing

Cholesky decomposition: $\boldsymbol{S} = \boldsymbol{L}\boldsymbol{L}^T$ (***S***: symmetric matrix; ***L***: lower triangular matrix)

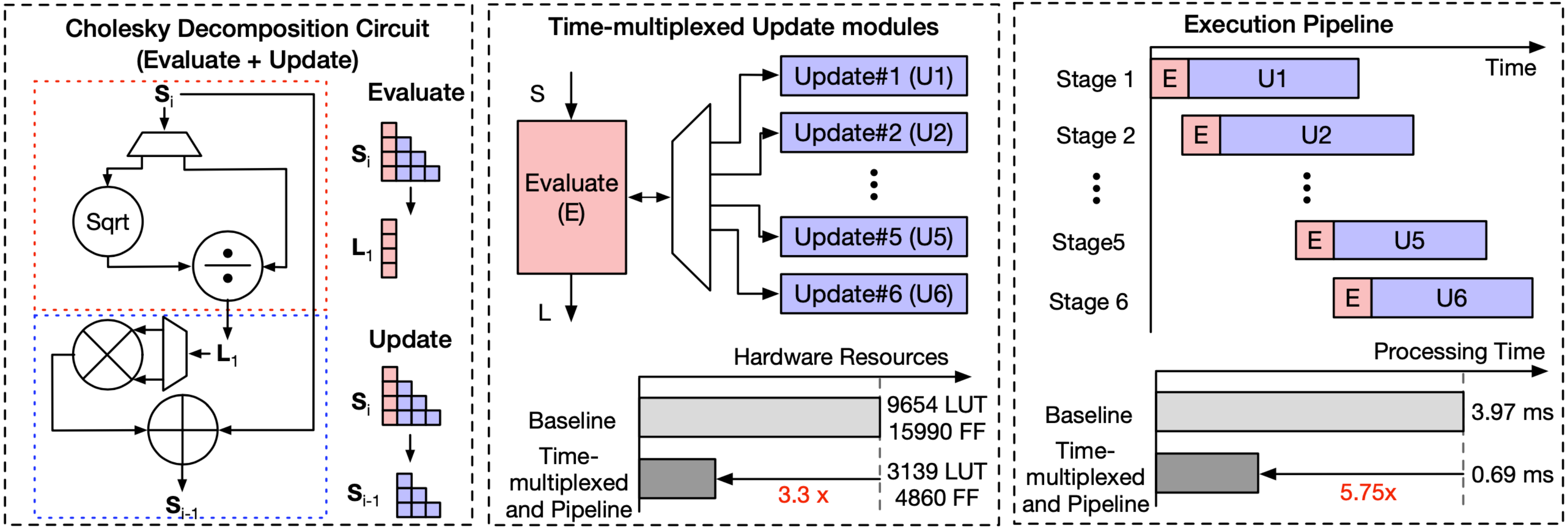






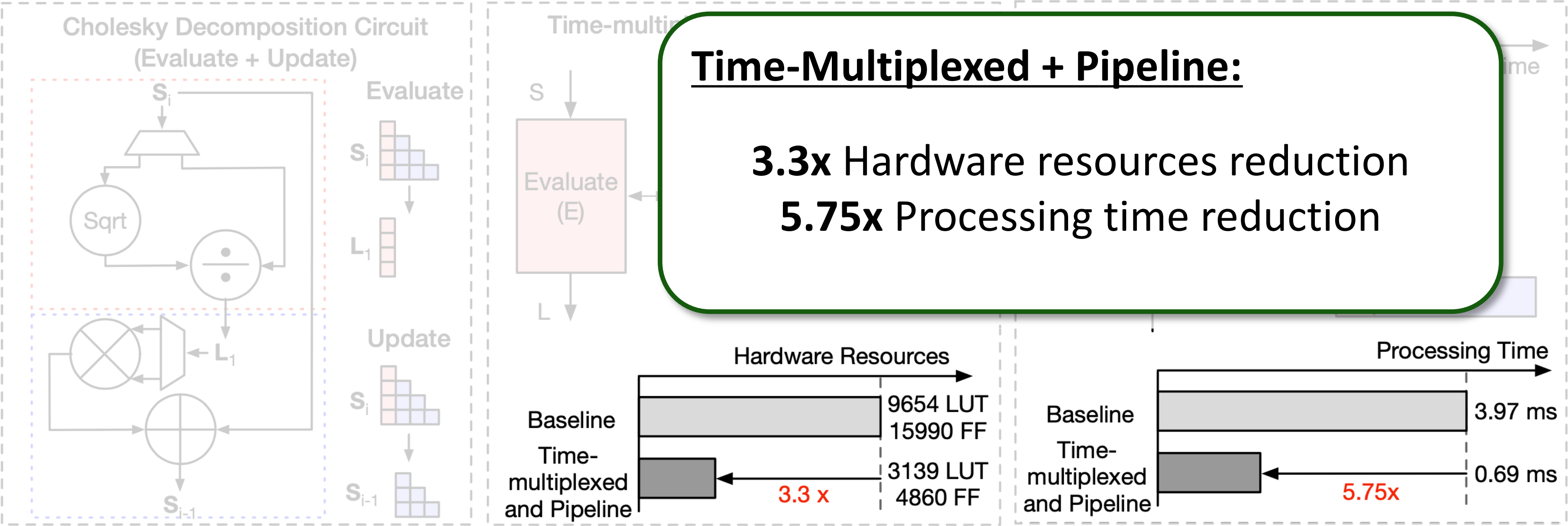
Time-Multiplexed + Pipeline Processing

Cholesky decomposition: $\boldsymbol{S} = \boldsymbol{L}\boldsymbol{L}^T$ ($\boldsymbol{S}$: symmetric matrix; $\boldsymbol{L}$: lower triangular matrix)



Time-Multiplexed + Pipeline Processing

Cholesky decomposition: $\boldsymbol{S} = \boldsymbol{LL}^T$ ($\boldsymbol{S}$: symmetric matrix; $\boldsymbol{L}$: lower triangular matrix)



Time-Multiplexed + Pipeline:

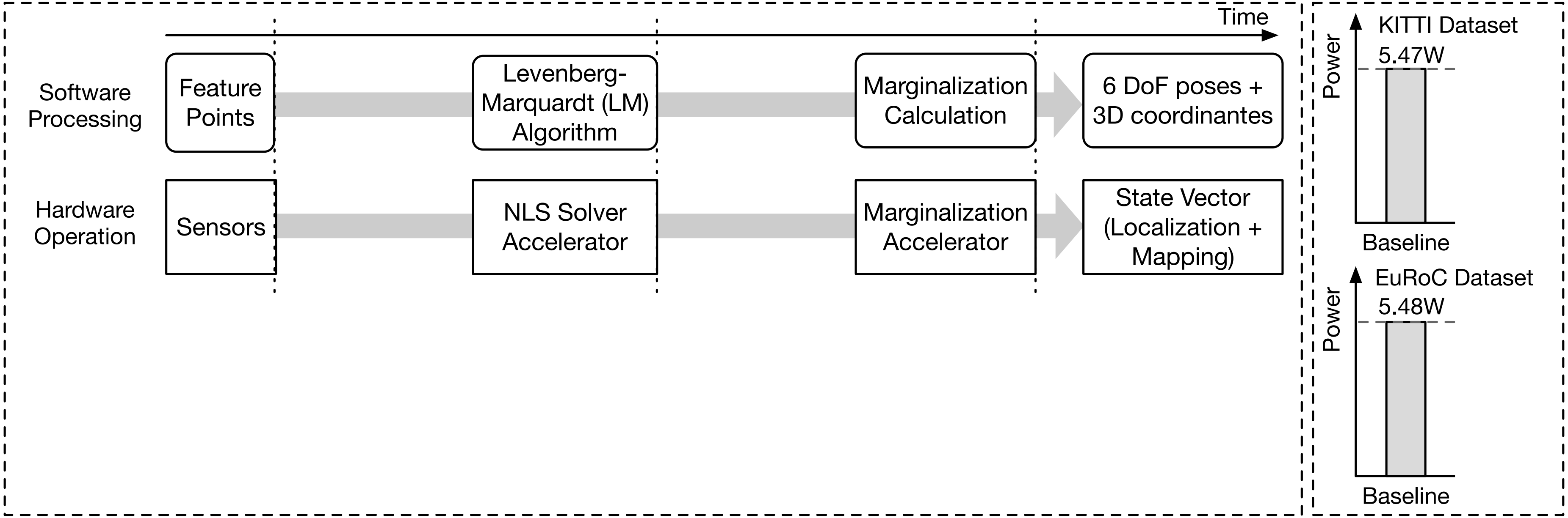
3.3x Hardware resources reduction

5.75x Processing time reduction

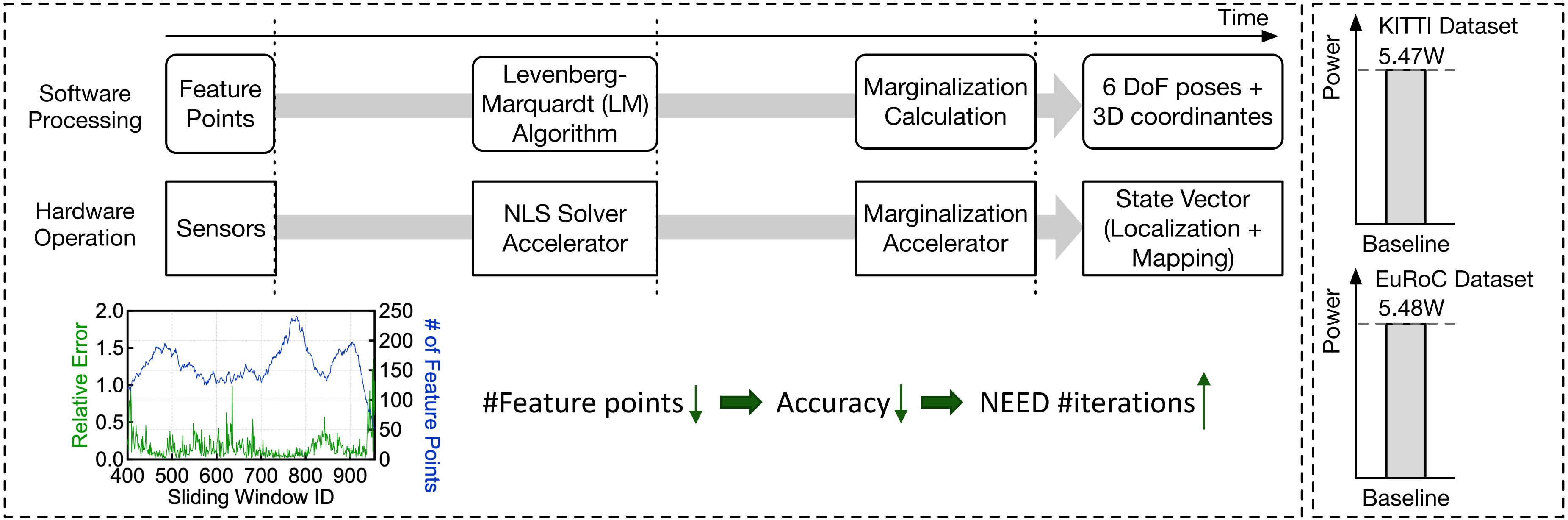
Method 4

Runtime Reconfiguration & Clock Gating

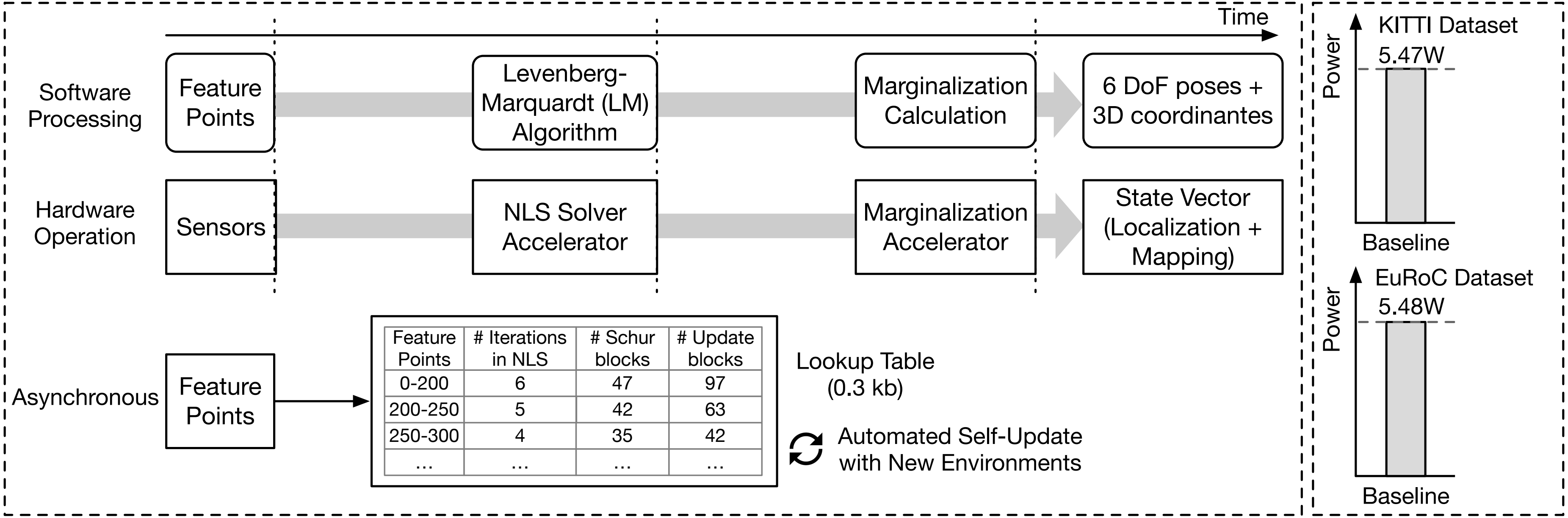
Runtime Reconfiguration + Clock Gating



Runtime Reconfiguration + Clock Gating

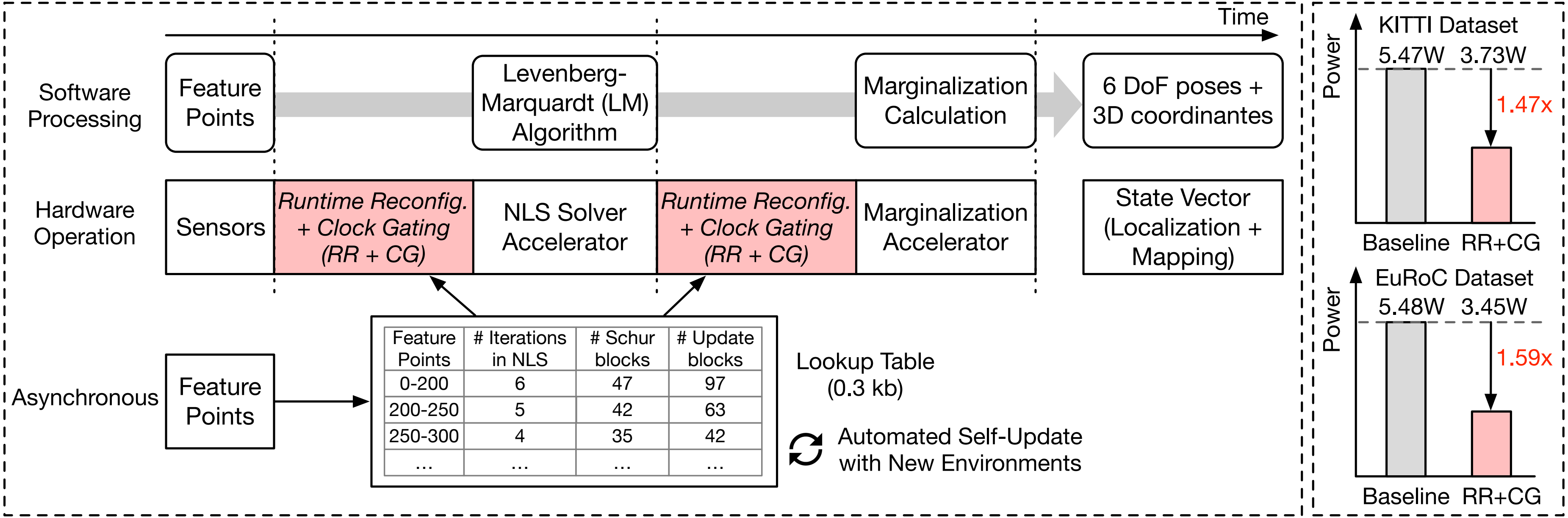


Runtime Reconfiguration + Clock Gating



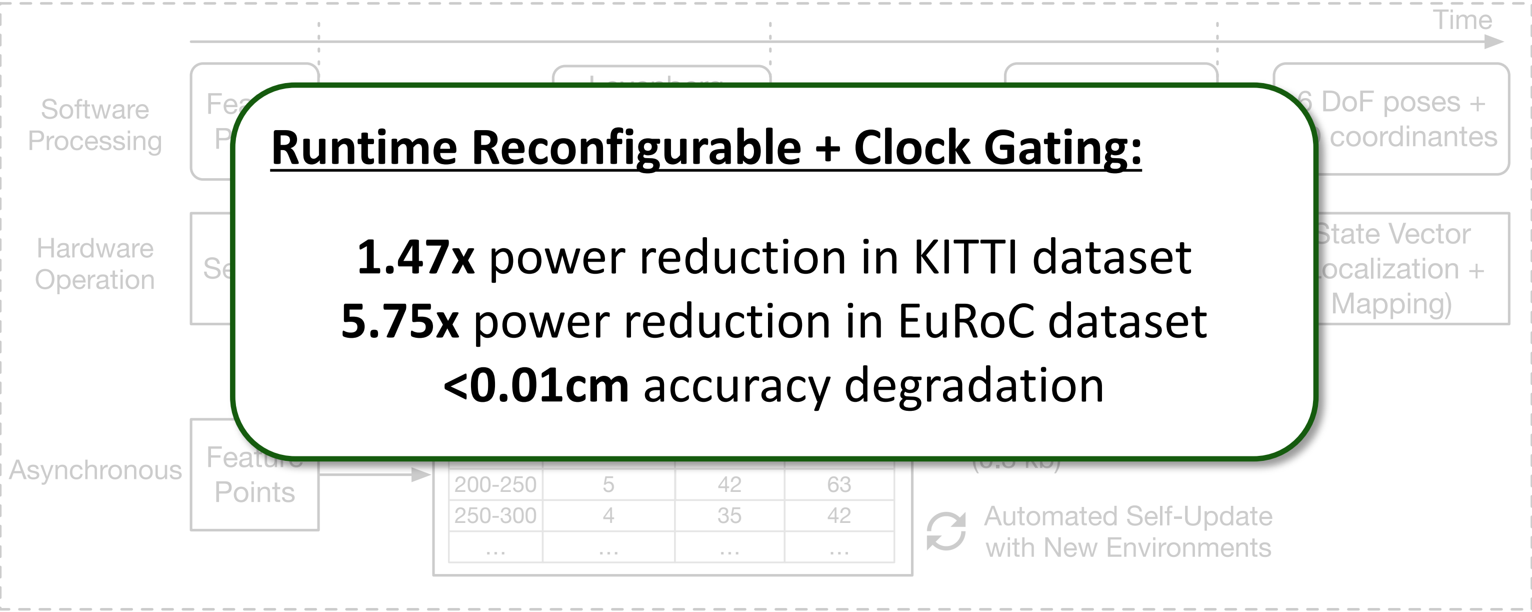
Feature Points	# Iterations in NLS	# Schur blocks	# Update blocks
0-200	6	47	97
200-250	5	42	63
250-300	4	35	42
...	...	...	...

Runtime Reconfiguration + Clock Gating



Feature Points	# Iterations in NLS	# Schur blocks	# Update blocks
0-200	6	47	97
200-250	5	42	63
250-300	4	35	42
...	...	...	...

Runtime Reconfiguration + Clock Gating

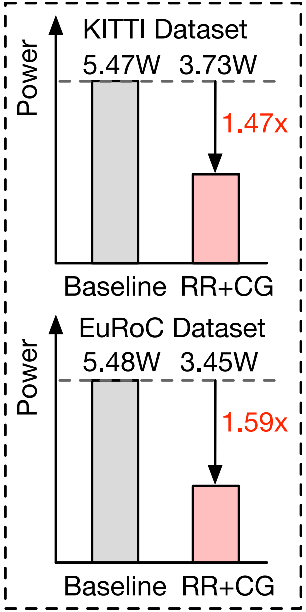


Runtime Reconfigurable + Clock Gating:

1.47x power reduction in KITTI dataset

5.75x power reduction in EuRoC dataset

<0.01cm accuracy degradation



Outline

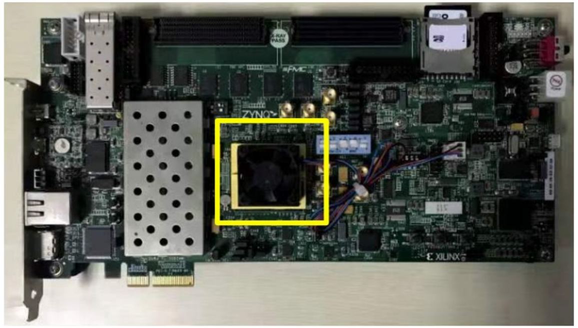
- SLAM: Simultaneously Localization & Mapping
- Hardware Architecture
- Main Contributions
- Evaluations and Comparisons
- Summary

Evaluation - Dataset

- EuRoC Dataset (for drone)
  - A very challenging, and widely used UAV dataset
  - 11 sequences with three categories: easy, medium & difficult
  - This work: Machine Hall sequences

- KITTI Dataset (for self-driving car)
  - A widely used autonomous driving vision benchmark
  - Task of interest: stereo, optical flow, visual odometry, 3D object detection and 3D tracking
  - This work: odometry (grayscale sequence)

Evaluation – FPGA Platform

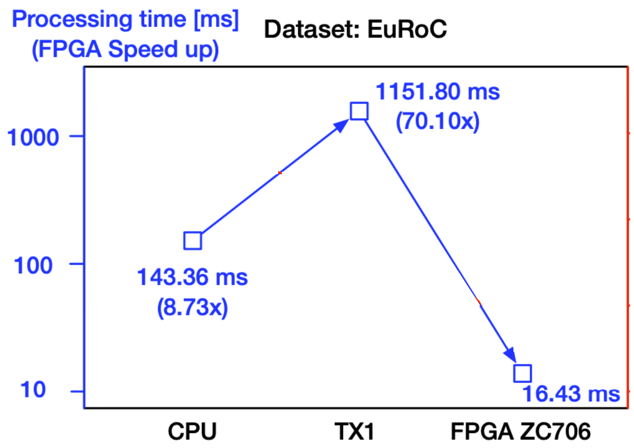


FPGA Zynq-7000 SoC ZC706
with XC7Z045 FFG900-2

Operation Frequency	143 MHz
LUT	144108 (65.92%)
Flip-Flop	172935 (39.56%)
BRAM	268 (49.17%)
DSP	869 (96.56%)

Evaluation

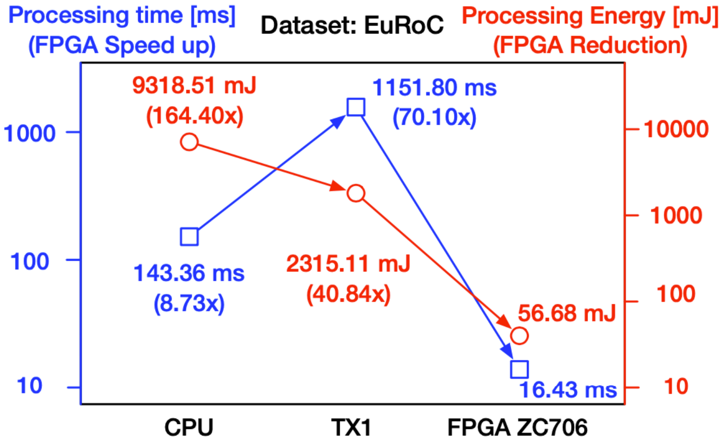
- Processing Latency and Energy of FPGA, CPU, and GPU



- FPGA: Xilinx Zynq-7000 SoC ZC706 @ 143 MHz
- CPU: Intel Comet Lake processor, 12 cores @ 2.9 GHz
- TX1: quad-core Arm Cortex-A57 processor @ 1.9 GHz

Evaluation

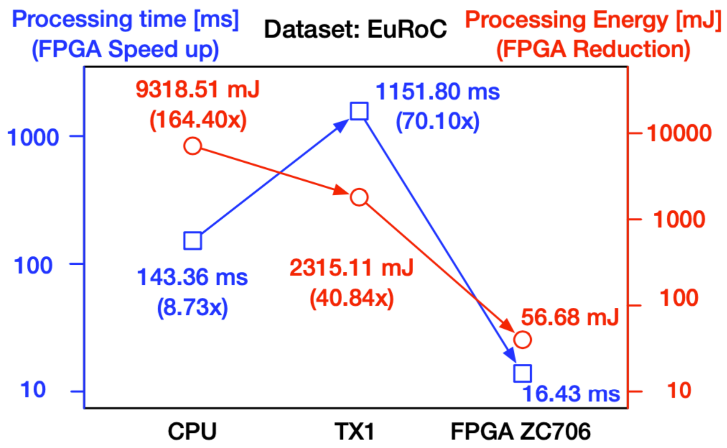
- Processing Latency and Energy of FPGA, CPU, and GPU

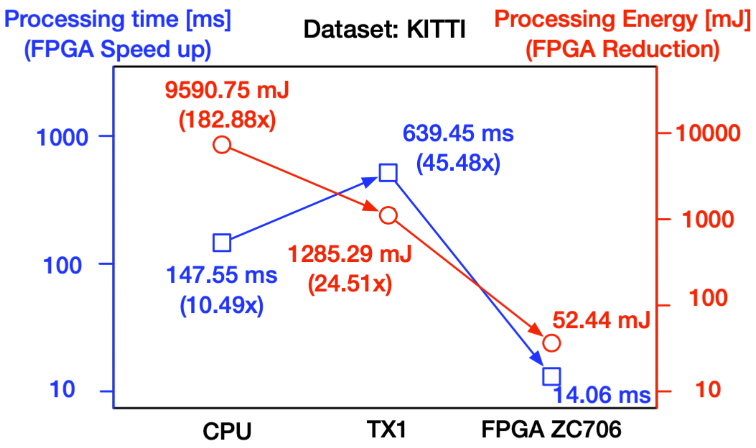


- FPGA: Xilinx Zynq-7000 SoC ZC706 @ 143 MHz
- CPU: Intel Comet Lake processor, 12 cores @ 2.9 GHz
- TX1: quad-core Arm Cortex-A57 processor @ 1.9 GHz

Evaluation

- Processing Latency and Energy of FPGA, CPU, and GPU

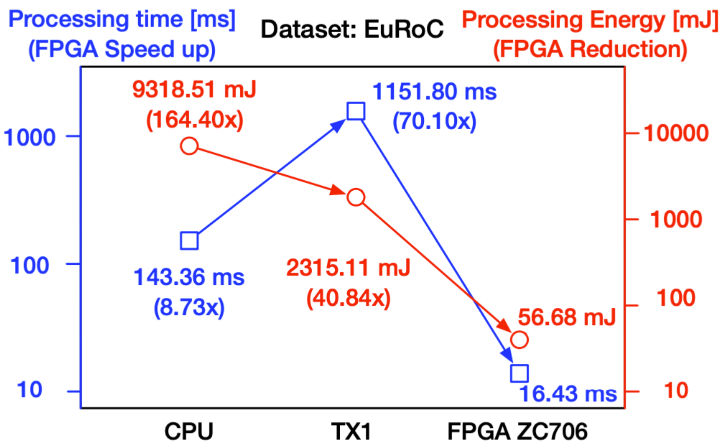




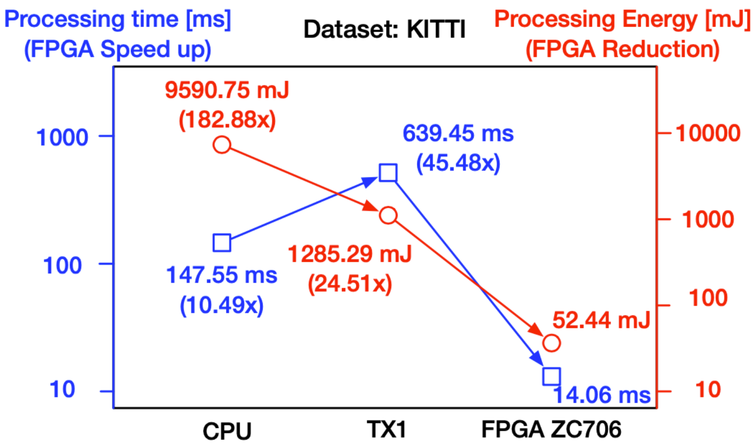
- FPGA: Xilinx Zynq-7000 SoC ZC706 @ 143 MHz
- CPU: Intel Comet Lake processor, 12 cores @ 2.9 GHz
- TX1: quad-core Arm Cortex-A57 processor @ 1.9 GHz

Evaluation

- Processing Latency and Energy of FPGA, CPU, and GPU



EuRoC Dataset (For drone)	FPGA Speedup		FPGA Energy Reduction	
	Over CPU	Over TX1	Over CPU	Over TX1
FPGA ZC706	8.73x	70.10x	164.40x	40.84x
Kintex-7 Series (XC7K160tfbg484)	7.01x	56.30x	180.73x	44.90x
Virtix-7 Series (XC7VX690tffg1761)	10.75x	86.34x	172.05x	42.75x



KITTI Dataset (For car)	FPGA Speedup		FPGA Energy Reduction	
	Over CPU	Over TX1	Over CPU	Over TX1
FPGA ZC706	10.49x	45.48x	182.88x	24.51x
Kintex-7 Series (XC7K160tfbg484)	8.27x	35.82x	196.09x	26.28x
Virtix-7 Series (XC7VX690tffg1761)	12.71x	55.08x	188.60x	25.28x

Evaluation

- Comparison with Related Work

	This work	ISSCC'19 CNN-SLAM [1]	JSSC'19 Navion [2]	TC'20 pi-BA [3]	RSS'17 VIO on Chip [4]	HPCA'21 Eudoxus [5]
Platform	FPGA	ASIC	ASIC	FPGA	FPGA	FPGA
Technology	28 nm	28 nm	65 nm	28nm	28nm	16nm
Design	digital	digital	digital	digital	digital	digital
Type	SLAM	SLAM	SLAM	SLAM	SLAM	SLAM
Algorithm	Levenberg-Marquardt (optimization-based)	Levenberg-Marquardt (optimization-based)	Gaussian-Newton (optimization-based)	Levenberg-Marquardt (optimization-based)	Gaussian-Newton (optimization-based)	Kalman Filter (Filter-based)
DoF	6-DoF	6-DoF	6-DoF	6-DoF	6-DoF	6-DoF
Voltage	1 V	0.63-0.9V	1.2V	1 V	1 V	0.85 V
Power	3.45W	243.6mW @ 0.9V 61.75mW @ 0.63V	24mW	5.50W	1.46 W	8.96W
Frequency	143 MHz	240 MHz	62.5/83.3 MHz	143 MHz	100 MHz	180 MHz
Throughput	55.8 GOPS	879.6 GOPS @ 0.9V 329.8 GOPS @ 0.63V	10.5-59.1 GOPS	N/A	4.4-24.6 GOPS	N/A
Latency	16.43 ms	N/A	30.8 ms	110 ms	200 ms	44.6 ms
Energy per Frame	56.6 mJ	N/A	739.2 uJ	605 mJ	292 mJ	399.6 mJ
Dynamic Optimiza-tion	Yes	N/A	N/A	No	No	No

Outline

- SLAM: Simultaneously Localization & Mapping
- Hardware Architecture
- Main Contributions
- Evaluations and Comparisons
- Summary

Summary

- **Energy-efficient** and **runtime-reconfigurable** FPGA accelerator for robotic localization and mapping.

Summary

- **Energy-efficient** and **runtime-reconfigurable** FPGA accelerator for robotic localization and mapping.
- Leverage data sparsity, locality, and parallelism inherent in localization.
  - **4.1x** memory reduction with symmetry and sparsity
  - **5.7x** compute time reduction with time-multiplexed and pipeline processing
  - **5.8x** power reduction with runtime reconfiguration and clock gating

Summary

- **Energy-efficient** and **runtime-reconfigurable** FPGA accelerator for robotic localization and mapping.
- Leverage data sparsity, locality, and parallelism inherent in localization.
  - **4.1x** memory reduction with symmetry and sparsity
  - **5.7x** compute time reduction with time-multiplexed and pipeline processing
  - **5.8x** power reduction with runtime reconfiguration and clock gating
- Our design is **2 orders of magnitude** more energy efficient than CPU and GPU.

Reference

An Energy-Efficient and Runtime-Reconfigurable FPGA-Based Accelerator for Robotic Localization Systems

Qiang Liu*[1], Zishen Wan*[2], Bo Yu*[3], Weizhuang Liu[1], Shaoshan Liu[3], Arijit Raychowdhury[2]

[1]Tianjin University, China, [2]Georgia Institute of Technology, USA, [3]PerceptIn, USA

*Equally-Credited Authors (ECAs)

A robot usually localizes itself in an environment by estimating the collection of its position and rotation states, while constructing a map of unknown surroundings, giving rise to the notion of Simultaneous Localization and Mapping (SLAM). SLAM is a fundamental kernel in autonomous machines at all computing scales, from drones, AR, VR to self-driving cars. Principled mathematical solutions for SLAM involve filtering-based or non-linear optimization-based (Fig. 1a), where the latter recently shows higher robustness but with intensive computation. Prior ASICs [1,2] and FPGAs [3,4,5] have accelerated SLAM on hardware, but they usually target one specific design. In this work, we present a runtime-reconfigurable FPGA accelerator for robotic localization tasks. We exploit SLAM-specific data locality, sparsity, reuse, and parallelism, and achieve >5x performance improvement over the state-of-the-art. Especially, our design is reconfigurable at runtime according to the environment and platform to save power while sustaining accuracy and performance.

Fig.1b shows the SLAM system compute latency characterization on software. SLAM consists of a vision frontend to extract features and an optimization backend to estimate the pose. We find that the localization computation accounts for 46% and 78% latency on two commonly-used SLAM systems, indicating a lucrative acceleration target. The localization is usually formulated as a constrained non-linear optimization problem, often through bundle adjustment, which minimizes the pose projection errors from 2D features to 3D points in the map. The optimization problem is solved using the Levenberg-Marquardt (LM) method, consisting of 1) a nonlinear least squares (NLS) solver that solves maximum a posteriori estimation, and 2) marginalization that generates the prior of NLS solver. We will accelerate both phases through software-hardware co-design by leveraging SLAM-specific data patterns and inherent parallelism.

The proposed robotic localization design accelerates both NLS solver and marginalization algorithm (Fig. 2a). The NLS solver circuit first calculates Jacobians, following by Schur elimination and Cholesky decomposition. Marginalization is performed after NLS solver. Fig. 2b shows the circuit for visual Jacobian. We divide the computation into three levels: keyframe, feature, and observation. The keyframe-level solves each keyframe's rotation matrix. The feature-level uses pixel coordinates and inverse depth to obtain feature spatial coordinates. The observation-level is divided into two phases. The first phase uses coordinates from feature-level and the second phase uses rotation matrix from keyframe-level to calculate final Jacobian and residual. This three-level computation enables two unique SLAM data reuse. First, each keyframe's rotation matrix is reused over all observations within the keyframe. Second, each feature's coordinate is reused across its associated observations. Since the number of features is 10x more than keyframes, we prioritize feature reuse over keyframe reuse, thus calculating Jacobian matrix in feature (row)-stationary dataflow. Fig. 2c shows the circuit for IMU Jacobian, which consists of two pipeline stages. The first stage contains three parallel blocks for Jacobian matrix calculation, and the second stage calculates the residual and stores Jacobian and residual. Zero and identities of IMU Jacobian matrix will not be stored, which can reduce memory by 72%.

SLAM requires us to solve the linear system $\mathbf{A}\Delta\mathbf{p}=\mathbf{b}$. We use Schur elimination to simplify the equation, where the visual Jacobian matrix is divided into four blocks (Fig. 3a). Blocks **U**, **W**, and **X** only relate to visual observations, and **V** relates to IMU and prior information. Thus, when calculating Schur complement matrix $\mathbf{V} - \mathbf{W}\mathbf{U}^{-1}\mathbf{X}$, it can be considered that we first calculate the visual part and then add IMU and prior information to it. Two optimization schemes are proposed in the Schur elimination block. First, we make **U** as a diagonal matrix to reduce the computational complexity of $\mathbf{U}^{-1}$ from $O(n^3)$ to $O(n)$. Second, when **U** is a diagonal matrix, **X** becomes the transpose of **W**, reducing the on-chip memory storage requirement by 1.34x. After Schur elimination, Cholesky decomposition decomposes the symmetric matrix **S** into a lower triangular matrix **L** such that $\mathbf{L}\mathbf{L}^T=\mathbf{S}$. Fig. 3b illustrates the circuit for Cholesky Decomposition, where the hardware iteratively generates the *i*-th column of matrix **L** (Evaluate) and updates **S** for calculating (*i-1*)-th column of **L** (Update). We find that at *i*-th iteration, the number of operations of Evaluate and Update are i and $i(i-1)/2$, respectively. Thus, we propose to pipeline Evaluate and Update, where multiple Update units are time-multiplexed with the Evaluate unit. With pipelining and time-multiplexing, the latency is reduced by 5.75x with 3.3x less resources consumption.

Marginalization uses NLS solver outputs and performs $\mathbf{A} - \mathbf{Z}\mathbf{M}^{-1}\mathbf{Z}^T$ to generate the priors for the next window computation (Fig. 4a). The difficulty lies in $\mathbf{M}^{-1}$ computation. We propose to divide **M** into four blocks and make $\mathbf{M}_{11}$ as a diagonal matrix. In this way, the Schur elimination and Cholesky decomposition circuits for NLS solver can be reused in marginalization, greatly reducing resource consumption without performance degradation. During marginalization, **S** matrix that stores the parameters for linear system, contributes 60% of total memory (Fig. 3a). We notice **S** is a symmetric matrix, so the memory can be reduced by half. To further reduce the storage, we leverage the unique SLAM data structured sparsity. Since **S** is obtained by integrating camera and IMU, we propose to store their contributions separately. IMU's observation only relates to adjacent keyframes, so the non-zero elements are in diagonal and sub-diagonal blocks. The non-zero elements of camera contributions only exist in the 6x6 sub-block of each state, donating 6 DoF. The camera storage is further reduced by limiting the number of keyframes that capture the feature (co-observations). The storage is reduced by 4.1x in this process.

The design is dynamically optimized at runtime to adapt to different surroundings and save power while maintaining accuracy (Fig. 4b). When entering new environments with various feature points, the number of NLS iterations is dynamically adjusted to meet target accuracy based on the offline constructed lookup table. Along with NLS solver iterations, the number of Schur elimination modules and update modules during Cholesky decomposition will be dynamically reconfigured for less resource consumption. Since the lookup table is updated asynchronously, this runtime reconfiguration has minimal overhead. Instead of reconfiguring bitstream to FPGA, we applied clock gating for dynamically adjusted modules, enabling 1.59x power reduction with only 0.15% overhead. This runtime optimization has little impact on accuracy with <0.01cm degradation and sometimes even improves the accuracy due to its stochastic nature.

The proposed hardware is implemented on Xilinx ZC706 FPGA, with a fixed operational frequency at 143 MHz (Fig. 5a). We evaluate the design with two datasets: EuRoC for drones and KITTI Odometry for cars (Fig. 5b). Compared with CPU operating at 2.9 GHz, our FPGA design achieves 164x (183x) speedup and 8.73x (10.49x) energy reduction on EuRoC (KITTI). Compared with TX1 operating at 1.9 GHz, our FPGA design achieves 70x (45x) speedup and 41x (25x) energy reduction on EuRoC (KITTI). To validate the generalization of our design, we evaluate two additional Xilinx FPGAs: Kintex-7 and Virtix-7 series. Evaluated on EuRoC, our design achieves 7x and 11x speed up as well as 56x and 86x energy reduction over CPU on two boards. The significant efficiency gains are consistently found on KITTI dataset. Fig.6 demonstrates that our design achieves >5x better performance against recent prior SLAM accelerators.

Acknowledgements:

This work was supported in part by National Natural Science Foundation of China under Grant U21B2031, and C-BRIC, one of six centers in JUMP, a SRC program sponsored by DARPA.

References:

[1] Z. Li *et al.*, "An 879GOPS 243mw 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration," ISSCC, Feb. 2019.

[2] A. Suleiman *et al.*, "Navion: A 2-mw Fully Integrated Real-Time Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones," JSSC, Apr. 2019.

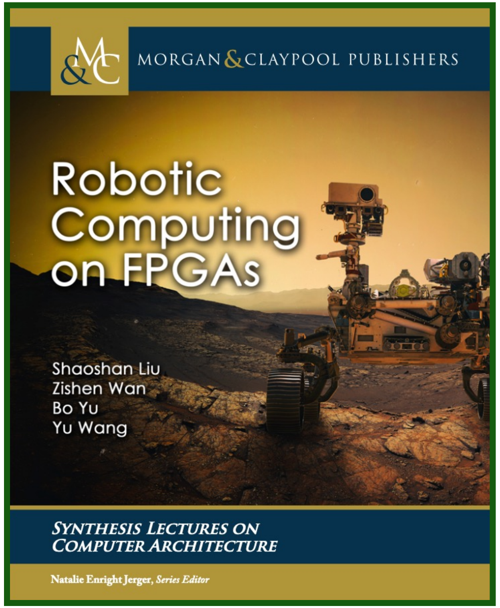
[3] Q. Liu *et al.*, "π-BA: Bundle Adjustment Hardware Accelerator Based on Distribution of 3D-Point Observations," TC, July. 2020.

[4] Z. Zhang *et al.*, "Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach," RSS, July. 2017.

[5] Y. Gan *et al.*, "Eudoxus: Characterizing and Accelerating Localization in Autonomous Machines," HPCA, Mar. 2021.

[Wan, CICC 2022]

Reference



[Wan, Synthesis Lectures on Comp Arch 2021]

1 IEEE CICC 2022

An Energy-Efficient and Runtime-Reconfigurable FPGA-Based Accelerator for Robotic Localization Systems

Qiang Liu*[1], Zishen Wan*[2], Bo Yu*[3], Weizhuang Liu[1], Shaoshan Liu[3], Arijit Raychowdhury[2]

[1]Tianjin University, China, [2]Georgia Institute of Technology, USA, [3]PerceptIn, USA

*Equally-Credited Authors (ECAs)

A robot usually localizes itself in an environment by estimating the collection of its position and rotation states, while constructing a map of unknown surroundings, giving rise to the notion of Simultaneous Localization and Mapping (SLAM). SLAM is a fundamental kernel in autonomous machines at all computing scales, from drones, AR, VR to self-driving cars. Principled mathematical solutions for SLAM involve filtering-based or non-linear optimization-based (Fig. 1a), where the latter recently shows higher robustness but with intensive computation. Prior ASICs [1,2] and FPGAs [3,4,5] have accelerated SLAM on hardware, but they usually target one specific design. In this work, we present a runtime-reconfigurable FPGA accelerator for robotic localization tasks. We exploit SLAM-specific data locality, sparsity, reuse, and parallelism, and achieve >5x performance improvement over the state-of-the-art. Especially, our design is reconfigurable at runtime according to the environment and platform to save power while sustaining accuracy and performance.

Fig.1b shows the SLAM system compute latency characterization on software. SLAM consists of a vision frontend to extract features and an optimization backend to estimate the pose. We find that the localization computation accounts for 46% and 78% latency on two commonly-used SLAM systems, indicating a lucrative acceleration target. The localization is usually formulated as a constrained non-linear optimization problem, often through bundle adjustment, which minimizes the pose projection errors from 2D features to 3D points in the map. The optimization problem is solved using the Levenberg-Marquardt (LM) method, consisting of 1) a nonlinear least squares (NLS) solver that solves maximum a posteriori estimation, and 2) marginalization that generates the prior of NLS solver. We will accelerate both phases through software-hardware co-design by leveraging SLAM-specific data patterns and inherent parallelism.

The proposed robotic localization design accelerates both NLS solver and marginalization algorithm (Fig. 2a). The NLS solver circuit first calculates Jacobians, following by Schur elimination and Cholesky decomposition. Marginalization is performed after NLS solver. Fig. 2b shows the circuit for visual Jacobian. We divide the computation into three levels: keyframe, feature, and observation. The keyframe-level solves each keyframe's rotation matrix. The feature-level uses pixel coordinates and inverse depth to obtain feature spatial coordinates. The observation-level is divided into two phases. The first phase uses coordinates from feature-level and the second phase uses rotation matrix from keyframe-level to calculate final Jacobian and residual. This three-level computation enables two unique SLAM data reuse. First, each keyframe's rotation matrix is reused over all observations within the keyframe. Second, each feature's coordinate is reused across its associated observations. Since the number of features is 10x more than keyframes, we prioritize feature reuse over keyframe reuse, thus calculating Jacobian matrix in feature (row)-stationary dataflow. Fig. 2c shows the circuit for IMU Jacobian, which consists of two pipeline stages. The first stage contains three parallel blocks for Jacobian matrix calculation, and the second stage calculates the residual and stores Jacobian and residual. Zero and identities of IMU Jacobian matrix will not be stored, which can reduce memory by 72%.

SLAM requires us to solve the linear system $\mathbf{A}\Delta\mathbf{p}=\mathbf{b}$. We use Schur elimination to simplify the equation, where the visual Jacobian matrix is divided into four blocks (Fig. 3a). Blocks **U**, **W**, and **X** only relate to visual observations, and **V** relates to IMU and prior information. Thus, when calculating Schur complement matrix $\mathbf{V} - \mathbf{W}\mathbf{U}^{-1}\mathbf{X}$, it can be considered that we first calculate the visual part and then add IMU and prior information to it. Two optimization schemes are proposed in the Schur elimination block. First, we make **U** as a diagonal matrix to reduce the computational complexity of $\mathbf{U}^{-1}$ from $O(n^3)$ to $O(n)$. Second, when **U** is a diagonal matrix, **X** becomes the transpose of **W**, reducing the on-chip memory storage requirement by 1.34x. After Schur elimination, Cholesky decomposition decomposes the symmetric matrix **S** into a lower triangular matrix **L** such that $\mathbf{L}\mathbf{L}^T=\mathbf{S}$. Fig. 3b illustrates the circuit for Cholesky Decomposition, where the hardware iteratively generates the *i*-th column of matrix **L** (Evaluate) and updates **S** for calculating (*i-1*)-th column of **L** (Update). We find that at *i*-th iteration, the number of operations of Evaluate and Update are *i* and *i*(*i-1*)/2, respectively. Thus, we propose to pipeline Evaluate and Update, where multiple Update units are time-multiplexed with the Evaluate unit. With pipelining and time-multiplexing, the latency is reduced by 5.75x with 3.3x less resources consumption.

Marginalization uses NLS solver outputs and performs $\mathbf{A} - \mathbf{Z}\mathbf{M}^{-1}\mathbf{Z}^T$ to generate the priors for the next window computation (Fig. 4a). The difficulty lies in $\mathbf{M}^{-1}$ computation. We propose to divide **M** into four blocks and make $\mathbf{M}_{11}$ as a diagonal matrix. In this way, the Schur elimination and Cholesky decomposition circuits for NLS solver can be reused in marginalization, greatly reducing resource consumption without performance degradation. During marginalization, **S** matrix that stores the parameters for linear system, contributes 60% of total memory (Fig. 3a). We notice **S** is a symmetric matrix, so the memory can be reduced by half. To further reduce the storage, we leverage the unique SLAM data structured sparsity. Since **S** is obtained by integrating camera and IMU, we propose to store their contributions separately. IMU's observation only relates to adjacent keyframes, so the non-zero elements are in diagonal and sub-diagonal blocks. The non-zero elements of camera contributions only exist in the 6x6 sub-block of each state, donating 6 DoF. The camera storage is further reduced by limiting the number of keyframes that capture the feature (co-observations). The storage is reduced by 4.1x in this process.

The design is dynamically optimized at runtime to adapt to different surroundings and save power while maintaining accuracy (Fig. 4b). When entering new environments with various feature points, the number of NLS iterations is dynamically adjusted to meet target accuracy based on the offline constructed lookup table. Along with NLS solver iterations, the number of Schur elimination modules and update modules during Cholesky decomposition will be dynamically reconfigured for less resource consumption. Since the lookup table is updated asynchronously, this runtime reconfiguration has minimal overhead. Instead of reconfiguring bitstream to FPGA, we applied clock gating for dynamically adjusted modules, enabling 1.59x power reduction with only 0.15% overhead. This runtime optimization has little impact on accuracy with <0.01cm degradation and sometimes even improves the accuracy due to its stochastic nature.

The proposed hardware is implemented on Xilinx ZC706 FPGA, with a fixed operational frequency at 143 MHz (Fig. 5a). We evaluate the design with two datasets: EuRoC for drones and KITTI Odometry for cars (Fig. 5b). Compared with CPU operating at 2.9 GHz, our FPGA design achieves 164x (183x) speedup and 8.73x (10.49x) energy reduction on EuRoC (KITTI). Compared with TX1 operating at 1.9 GHz, our FPGA design achieves 70x (45x) speedup and 41x (25x) energy reduction on EuRoC (KITTI). To validate the generalization of our design, we evaluate two additional Xilinx FPGAs: Kintex-7 and Virtix-7 series. Evaluated on EuRoC, our design achieves 7x and 11x speed up as well as 56x and 86x energy reduction over CPU on two boards. The significant efficiency gains are consistently found on KITTI dataset. Fig.6 demonstrates that our design achieves >5x better performance against recent prior SLAM accelerators.

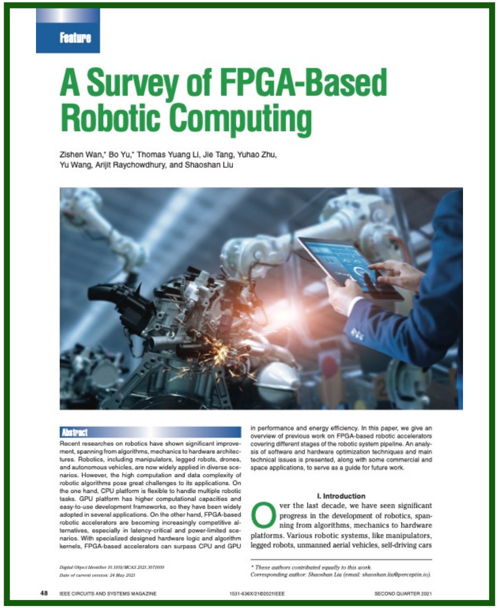
Acknowledgements:

This work was supported in part by National Natural Science Foundation of China under Grant U21B2031, and C-BRIC, one of six centers in JUMP, a SRC program sponsored by DARPA.

References:

[1] Z. Li *et al.*, "An 879GOPS 243mw 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration," ISSCC, Feb. 2019.
[2] A. Suleiman *et al.*, "Navion: A 2-mw Fully Integrated Real-Time Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones," JSSC, Apr. 2019.
[3] Q. Liu *et al.*, "π-BA: Bundle Adjustment Hardware Accelerator Based on Distribution of 3D-Point Observations," TC, July. 2020.
[4] Z. Zhang *et al.*, "Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach," RSS, July. 2017.
[5] Y. Gan *et al.*, "Eudoxus: Characterizing and Accelerating Localization in Autonomous Machines," HPCA, Mar. 2021.

[Wan, CICC 2022]



Feature

A Survey of FPGA-Based Robotic Computing

Zishen Wan,* Bo Yu,* Thomas Yuang Li, Jie Tang, Yuhao Zhu, Yu Wang, Arijit Raychowdhury, and Shaoshan Liu

Abstract

Recent researches on robotics have shown significant improvement, spanning from algorithms, mechanics to hardware architectures. Robotics, including manipulators, legged robots, drones, and autonomous vehicles, are now widely applied in diverse scenarios. However, the high computation and data complexity of robotic algorithms pose great challenges to its applications. On the one hand, CPU platform is flexible to handle multiple robotic tasks. GPU platform has higher computational capacities and easy-to-use development frameworks, so they have been widely adopted in several applications. On the other hand, FPGA-based robotic accelerators are becoming increasingly competitive alternatives, especially in latency-critical and power-limited scenarios. With specialized designed hardware logic and algorithm kernels, FPGA-based accelerators can surpass CPU and GPU in performance and energy efficiency. In this paper, we give an overview of previous work on FPGA-based robotic accelerators covering different stages of the robotic system pipeline. An analysis of software and hardware optimization techniques and main technical issues is presented, along with some commercial and space applications, to serve as a guide for future work.

I. Introduction

Over the last decade, we have seen significant progress in the development of robotics, spanning from algorithms, mechanics to hardware platforms. Various robotic systems, like manipulators, legged robots, unmanned aerial vehicles, self-driving cars

Digital Object Identifier 10.1109/MCAS.2021.3071609
Date of current version: 24 May 2021

** These authors contributed equally to this work.*
Corresponding author: Shaoshan Liu (email: shaoshan.liu@perceptin.io).

48 IEEE CIRCUITS AND SYSTEMS MAGAZINE 1531-636X/21©2021IEEE SECOND QUARTER 2021

[Wan, Circuits and Systems Magazine 2021]


THANK YOU
OBRIGADO
gracias
どうも
hvala
DANK U
takk
MERSI
merci
obrigado
danke schön
köszi
سپاس
PALDIES
muchas gracias
ありがとう
TEŞEKKÜR EDERIM
MOLTE GRAZIE
GO RAIBH MAITH AGAT
謝謝
danke
ARIGATO
grazas
GRAZZI
THANKS
qujan
TAK
благодаря
asante
TAK
どうも
muchas gracias
vielen dank
PALDIES
DANK U
danke
OBRIGADO
mesi
감사합니다
köszi
grazie
TACK
DZIEKI
Gràcies
спасибо
MULTUMESC
多謝
شكراً
TEŞEKKÜR EDERIM
NA GODE
muchas gracias
obrigado