PISCES: Power-Aware Implementation of SLAM by Customizing Efficient Sparse Algebra

Bahar Asgari, Ramyad Hadidi, Nima Shoghi Ghaleshahi, Hyesoon Kim

Georgia Institute of Technology

Simultaneous Localization and Mapping (SLAM)

- A key real-time task in autonomous systems
- Categories of algorithms:
  - Direct methods
  - Indirect feature-based methods (e.g., EKF[1] and ORB[2]) ← Our focus
  - Semi-direct and semi-dense methods
- Categories of implementations:
  - Software
  - Hardware with focus on
    - Architecture
    - Microarchitecture ← Most related to our work

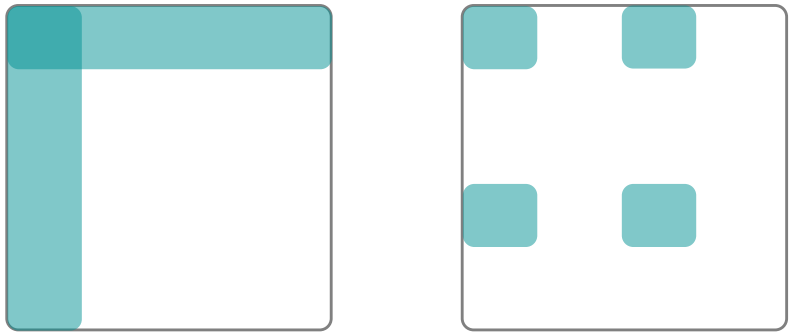
[1] EKF: extended Kalman filter
[2] ORB: oriented-fast and rotated-brief

Key Challenges and Our Observations

- Two aspects of SLAM create a performance bottleneck and increase power consumption:
  - The random accesses to the on-chip memory
  - The high data-reuse rate of compute-intensive matrix operations
- The building blocks of SLAM consist of ***sparse*** computations that
  - Cause the first aspect
  - Worsen the second one
- We take advantage of the sparseness to jointly improve
  - Latency
  - Power consumption

Sparse Matrix Algebra in EKF and ORB SLAM

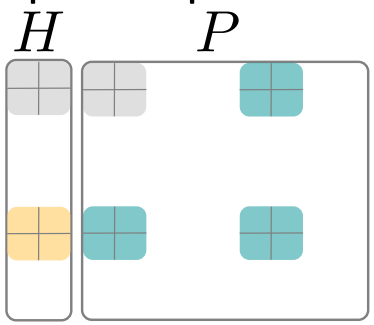
- The sparseness is ***structured*** and has common attributes:
  - A sequence of matrix operations on sparse operands
  - The sparse operand captures fixed-size, small, and dense blocks of data
  - The correlated dense blocks are scattered over *deterministic related locations*
- Example for EKF



PISCES – Key Insights

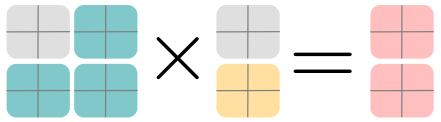
- To efficiently improve SLAM performance, we propose ***Pisces*** that
  - Aligns the dense blocks of sparse data that are processed together
  - Maps them to adjacent locations of the on-chip memory
  - Implements a chain of sparse operations as a sequence of ***dense*** operations
  - Reads a dense block once and applies all processes before writing it back
- Example for EKF

Sparse operation



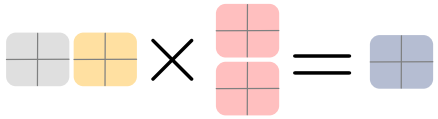
$H \times P \times H^T = B$

Dense operation



$P_{4\times4} \times H^T{}_{4\times2} = A_{4\times2}$

Dense operation



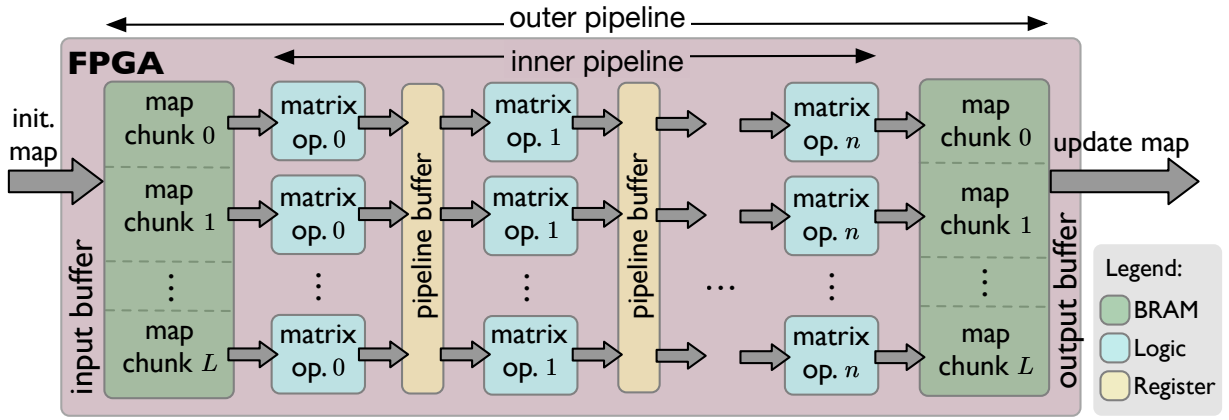
$H^T{}_{2\times4} \times A_{4\times2} = B_{2\times2}$

PISCES – Microarchitecture

- Dense building blocks of Pisces and their latency:

Dense matrix operation	$A^T_{2\times2}$	$A_{4\times4}\times B_{4\times2}$	$A_{2\times4}\times B_{4\times2}$	$A_{2\times2}\times B_{2\times4}$ or $A_{4\times2}\times B_{2\times2}$	$A_{2\times2} \pm B_{2\times2}$
Latency (μs)	0.06	0.66	0.33	0.58	0.09

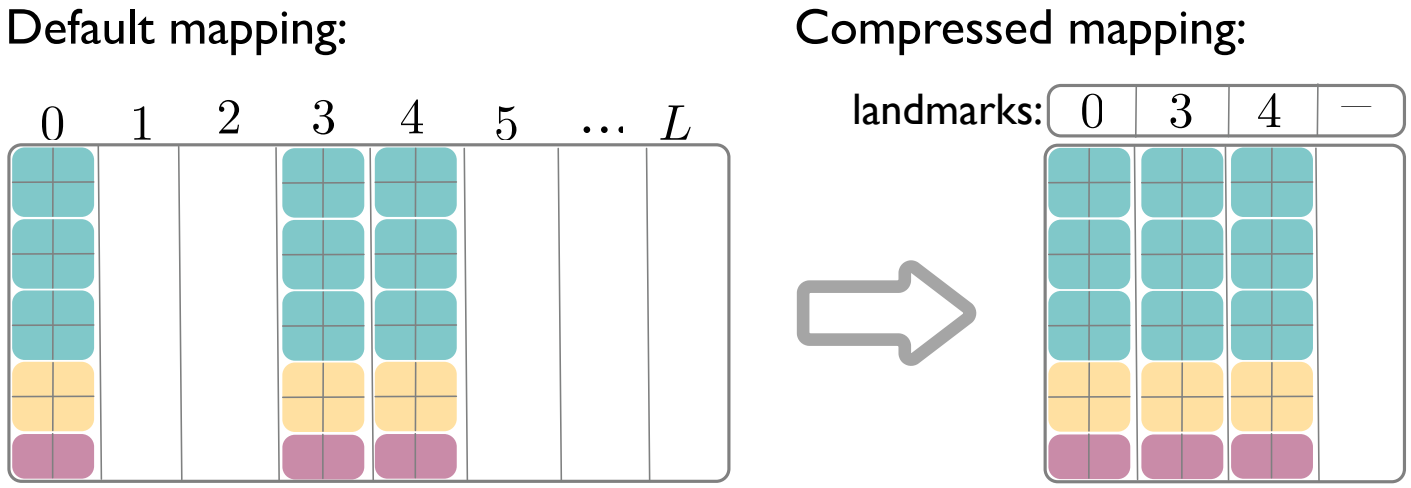
- Pipelines and configurations for EKF and ORB:



	op. 0	op. 1	op. 2	op. 3	op. 4	op. 5
EKF update ($n = 5$)	AB^T	$AB + C$	AB^{-1}	AB	AB^T	$AB - C$
ORB bundle adj. ($n = 3$)	AB^T	$AB^T + C$	AB^{-1}	$C - AB^T$	N/A	N/A

PISCES – Design Optimization

- An agent observes new landmarks and stops observing some of old ones
- We need to dynamically replace old landmarks with new ones
- Pisces exploits a simple compression scheme:



Evaluation Methodology

- Tools and system setup:
  - PYNQ-z1 including Zynq XCZ020 FPGA
  - Xilinx Vivado HLS to generate Pisces
  - AXI stream interface
  - 100MHz clock frequency
- Baselines:
  - One-dimensional systolic array (1DSA)[1]
  - Faddeev systolic array (FSA)[2]
  - eSLAM[3]
  - Raspberry Pi 4 including Cortex-A72 ARM processor
- Benchmarks, algorithms, and datasets:
  - For EKF: synthetic emulated environment
  - For ORB: EuRoC dataset

[1] Tertei et al, Elsevier computers & electrical Engineering, 2016.
[2] Souza et al, Springer Journal of Signal Processing Systems, 2018.
[3] Liu et al, DAC, 2019.

Results – Resource Utilization and Power

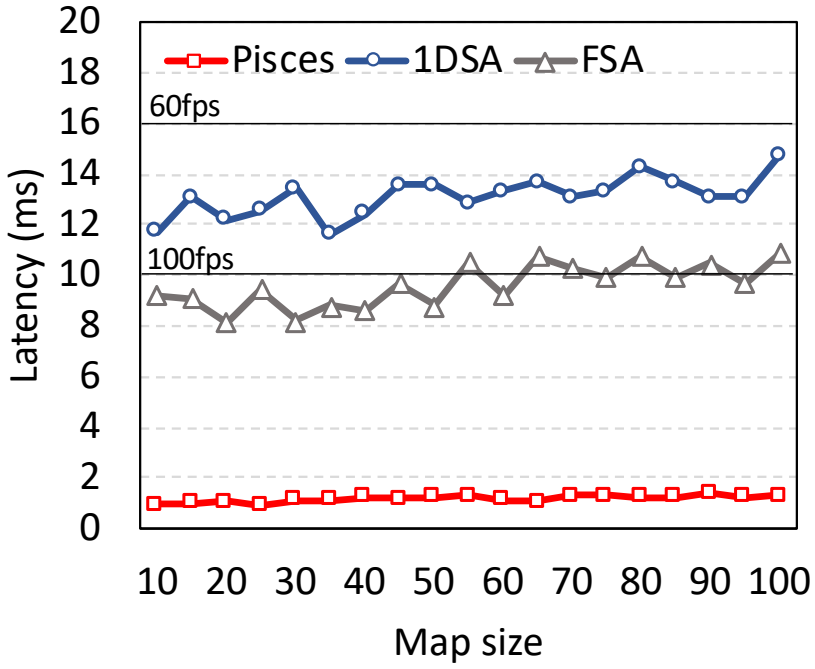
- Pisces trades BRAM for FF and LUT to more efficiently consume the power budget:
  - 3.3× and 2.5× less power consumption compared to 1DSA and FSA.

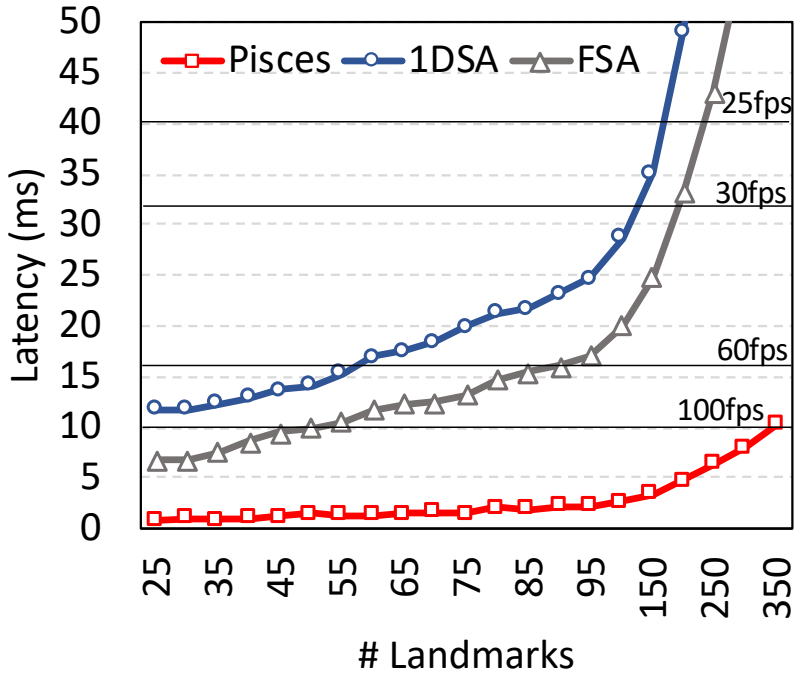
	EKF			ORB		
	1DSA	FSA	Pisces EKF	eSLAM	Pisces ORB	Available
BRAM(Kb)	756	297	252	78	180	2520
LUT	7824	3073	14472	56954	11898	53200
FF	4223	5176	16686	67809	12178	106400
DSP	32	2	75	111	114	220
Power(W)	1.302	0.986	0.384	1.936	0.292	N/A

1DSA: Tertei et al, Elsevier computers & electrical Engineering, 2016.
FSA: Souza et al, Springer Journal of Signal Processing Systems, 2018.
eSLAM: Liu et al, DAC, 2019.

Results – Latency vs. Hardware Accelerators

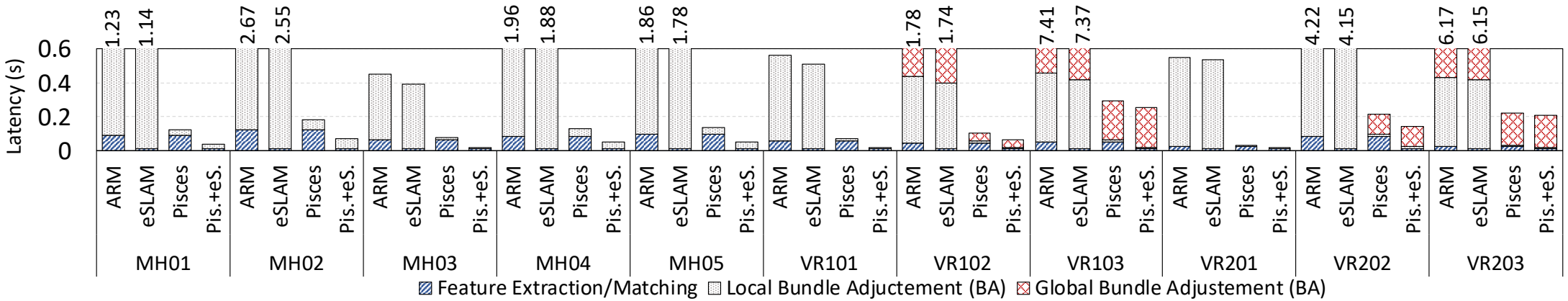
- Power consumption is not the only concern about the prior hardware accelerators
- Pisces executes EKF 11× and 7.4× faster than 1DSA and FSA





Results – Latency vs. Hardware Accelerators

- For ORB:
  - eSALM extracts/matches the features 8× as fast as ARM implementation
  - Pisces significantly reduces the latency of local and global bundle adjustment
  - to meet real-time constraints, two approaches can be combined



Conclusions

- The navigation of autonomous systems relies on SLAM.
- With advancement in sensor technologies, SLAM performance is becoming a bottleneck for a faster and more accurate navigation.
- With a limited power budget in autonomous systems, performing the compute-intensive SLAM in real time is the key challenge.
- We proposed Pisces, a new approach to accelerate SLAM.
- To improve power consumption and latency, Pisces
  - Transforms the sparse matrix operations into a chain of fixed-size dense matrix operations.
  - Reduces the accesses to on-chip memory by enabling the data exchange between the functions.