Qiang Liu¹,*, Zishen Wan²,*, Bo Yu³,*, Weizhuang Liu¹, Shaoshan Liu³, Arijit Raychowdhury²

* Equally Contributed Authors

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

• **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.

Email: zishenwan@gatech.edu
Homepage: https://zishenwan.github.io
Motivation: Autonomous Systems

Drones

Self-Driving Cars

Robots

[Source: V. Sze]
Motivation: Autonomous Systems

Drones  Self-Driving Cars  Robots

Search & Rescue  Package Delivery  Surveillance

[Source: V. Sze]
How Does Autonomous System Work?

- Perception
- Localization
- Motion Planning
- Control

(Source: V. Sze)
How Does Autonomous System Work?

Localization is computationally intensive
Challenges

Large Factor Graph:

4000+ factors

[Source: V. Sze]

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

Large Factor Graph:

4000+ factors

Real-Time Requirement:

Execution time

CPU  GPU  FPGA

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

**Large Factor Graph:**
- IMU_1
- IMU_2
- \( x_1 \)
- \( x_2 \)
- \( x_3 \)
- \( x_4 \)
- \( x_5 \)

4000+ factors

**Real-Time Requirement:**
- Execution time vs. Resources:
  - CPU
  - GPU
  - FPGA

**Low Power Budget:**
- Big battery
- CPU/GPU: 10-100W

[Source: V. Sze]
Challenges

Large Factor Graph:

- 4000+ factors

Real-Time Requirement:

- CPU
- GPU
- FPGA

Execution time

Low Power Budget:

- Big battery
- CPU/GPU: 10-100W

Dynamic Changing Environments:

- Sparse
- Dense

[Source: A. Lele]
Energy-Efficient Localization and Mapping

- 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

Estimated States

Simultaneous Localization and Mapping (SLAM)

Localization (6 DoF poses)

Mapping (3D coordinates)

IMU (Inertial Measurement Unit)

[Source: V. Sze]
Localization and Mapping Using SLAM

SLAM is computationally intensive:

**ORB-SLAM**
- FrontEnd: ORB (54%)
- BackEnd: SLAM (46%)

**LK-SLAM**
- FrontEnd: LK (22%)
- BackEnd: SLAM (78%)
How Does SLAM Work?

Camera

Feature Tracks

IMU (Inertial Measurement Unit)

Estimated States
How Does SLAM Work?

- Camera
- IMU (Inertial Measurement Unit)
- Feature Tracks
- Estimated States

ORB-SLAM
  FrontEnd: ORB
  BackEnd: SLAM
  ORB 54%
  SLAM 46%

LK-SLAM
  FrontEnd: LK
  BackEnd: SLAM
  LK 22%
  SLAM 78%
How Does SLAM Work?

Maximum a Posteriori (MAP) Estimation

Nonlinear least squares (NLS) optimization problem:

\[
\min_p \left\{ \sum_{i=1}^{N} \left( \| r_{ip} - H_{ip} p \|^2 + \| o_i - \mathcal{P}(p) \|^2 \| \right) \right\}
\]

State:
6 DoF poses (location)
3D coordinates (mapping)
How Does SLAM Work?

Camera

Feature Tracks

Marginalization

Maximum a Posteriori (MAP) Estimation

Estimated States

Nonlinear least squares (NLS) optimization problem:

\[
\min_{\mathbf{p}} \left\{ \sum_{i=1}^{N} \left( \| \mathbf{r}_p - \mathbf{H}_p \mathbf{p} \|^2 + \| \mathbf{o}_i - \mathbf{P}_i(p) \|^2 \right) \right\}
\]

State:
6 DoF poses (location)
3D coordinates (mapping)
How Does SLAM Work?

Nonlinear least squares (NLS) optimization problem:
\[
\min_p \left\{ \sum_{i=1}^{N} \| r_p - H_p p \|^2 + \| o_i - \mathcal{P}_i(p) \|^2 \right\}
\]

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

Camera
IMU (Inertial Measurement Unit)

Feature Tracks
Estimated States
Marginalization
Maximum a Posteriori (MAP) Estimation

ORB-SLAM
FrontEnd: ORB
BackEnd: SLAM
ORB 54% SLAM 46%

LK-SLAM
FrontEnd: LK
BackEnd: SLAM
LK 22% SLAM 78%

Jacobian Matrix
Schur Elimination
Cholesky Decomposition

State:
- 6 DoF poses (location)
- 3D coordinates (mapping)
How Does SLAM Work?

Nonlinear least squares (NLS) optimization problem:

\[
\min_{p} \left\{ \sum_{i=1}^{N} ||r_{ip} - H_{ip}p||^2 + ||o_{i} - \mathcal{P}_{i}(p)||^2 \right\}
\]

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

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

Diagram showing the flow of data and processes:
- Front End
  - Camera
  - IMU
- Input Buffer
  - DDR
  - Old $H_p, r_p$
  - IMU Jacobian and Residual Update
  - DTD Evaluate
  - Schur Elimination
  - RAM
  - Hessian Matrix Calculation
  - Prior Information Accumulation
  - RAM
- Output Buffer
  - Substitution and Solve $\delta p$
  - Cholesky Decomposition
  - RAM
  - Marginazation
Hardware Architecture – Perception

Sensor Input:
Camera + IMU, process in host
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 Keyframes
3 Feature Points (F1~F3)
4 Observations (O1~O4)
Data Reuse & Design Hierarchy

2 Keyframes
3 Feature Points (F1~F3)
4 Observations (O1~O4)

<feature point, observation> pairs have non-zero values
Data Reuse & Design Hierarchy

2 Keyframes
3 Feature Points (F1~F3)
4 Observations (O1~O4)

Jacobian Matrix

<feature point, observation> pairs have non-zero values
Data Reuse & Design Hierarchy

Three-Level Block Designs:
- Keyframe-level: Rotation matrix of keyframes
- Feature-level: 3D coordinates
- Observation-level: Jacobian matrix
Data Reuse & Design Hierarchy

Two-Level Data Reuses:
- Feature-reuse: across associated observations
- 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
Data Reuse & Design Hierarchy

**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:

\[ \mathbf{A} \mathbf{x} + \Delta \mathbf{p} = \mathbf{b} \]
Diagonal Computation + Symmetry + Hardware Reuse

Shure Elimination:

\[
\begin{bmatrix}
U & X \\
W & V
\end{bmatrix}
\begin{bmatrix}
x \\
\Delta p
\end{bmatrix}
= b
\]
Diagonal Computation + Symmetry + Hardware Reuse

Shure Elimination:

\[
\begin{align*}
\mathbf{A} & \rightarrow \begin{bmatrix} \mathbf{U} & \mathbf{X} \\ \mathbf{W} & \mathbf{V} \end{bmatrix} \\
\begin{bmatrix} 5 & \Delta p \\ 3 & = \\
\end{bmatrix} \\
\end{align*}
\]

Make \( \mathbf{U} \) as diagonal matrix:
\( O(n^3) \rightarrow O(n) \) computational complexity

\( \mathbf{X} \) becomes the transpose of \( \mathbf{W} \):
1.34x on-chip memory reduction
Diagonal Computation + Symmetry + Hardware Reuse

Shure Elimination:

\[
\begin{bmatrix}
U & X & W & V
\end{bmatrix}
\]

\[
A \times \Delta p = b
\]

Marginalization:

Goal: prior information \( H_p \)

\[
H_p = A - ZM^{T}Z^{T}
\]

\( M \) is general matrix

\[
M = \begin{bmatrix}
M_{11} & M_{12} \\
M_{21} & M_{22}
\end{bmatrix}
\]

\( M^{-1} = f(M, S') \)

\( S' = M_{22} - M_{21}M_{11}^{-1}M_{12} \)

Make \( M_{11} \) diagonal

Reuse Schur Elimination and Cholesky Decomposition
Diagonal Computation + Symmetry + Hardware Reuse

Shure Elimination:

\[ A \times \Delta p = b \]

Marginalization:

Goal: prior information \( H_p \)
\[ H_p = A - ZM^T Z' \]
M is general matrix
\[ M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \]
\[ M^{-1} = f(M, S') \]
\[ S' = M_{22} - M_{21} M_{11}^{-1} M_{12} \]
Make \( M_{11} \) diagonal

Reuse Schur Elimination and Cholesky Decomposition
Diagonal Computation + Symmetry + Hardware Reuse

**Make M as diagonal matrix:**

$O(n^3) \rightarrow O(n)$ computational complexity

**Reuse Schur Elimination circuit in Marginalization:**

Reduce resource consumption without performance degradation
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)

S matrix: 720 kb

S matrix:

15

IMU

Vision

δ
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)

4.1x reduction

720 kb ──> 175.97 kb
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 → 4.1x reduction → 175.97 kb
Method 3

Time-Multiplex & Pipeline
Time-Multiplexed + Pipeline Processing

Cholesky decomposition: \( S = LL^T \) (\( S \): symmetric matrix; \( L \): lower triangular matrix)
Cholesky decomposition: $S = LL^T$ ($S$: symmetric matrix; $L$: lower triangular matrix)
Time-Multiplexed + Pipeline Processing

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

**Time-Multiplexed + Pipeline:**

- **Hardware resources reduction**: 3.3x
- **Processing time reduction**: 5.75x
Method 4

Runtime Reconfiguration & Clock Gating
Runtime Reconfiguration + Clock Gating

Feature Points
- Levenberg-Marquardt (LM) Algorithm
- Marginalization Calculation
- 6 DoF poses + 3D coordinates

Marginalization Accelerator
- State Vector (Localization + Mapping)

Sensors
- NLS Solver Accelerator

Software Processing
Hardware Operation

Time

KITTI Dataset
Power: 5.47W
Baseline
EuRoC Dataset
Power: 5.48W
Baseline
Runtime Reconfiguration + Clock Gating

Feature Points
Levenberg-Marquardt (LM) Algorithm
Marginalization Calculation
6 DoF poses + 3D coordinates
State Vector (Localization + Mapping)

# Feature points ↓ → Accuracy ↓ → NEED # iterations

Software Processing
Levenberg-Marquardt (LM) Algorithm
Marginalization Calculation
State Vector (Localization + Mapping)

Hardware Operation
NLS Solver Accelerator
Marginalization Accelerator

Feature Points
Levenberg-Marquardt (LM) Algorithm
Marginalization Calculation
State Vector (Localization + Mapping)

Power
Baseline
KITTI Dataset
5.47W
EuRoC Dataset
1.47x
Baseline
5.48W

Power
Baseline
5.47W
EuRoC Dataset
1.47x
Baseline
5.48W
Runtime Reconfiguration + Clock Gating

Feature Points

Levenberg-Marquardt (LM) Algorithm

Marginalization Calculation

6 DoF poses + 3D coordinates

State Vector (Localization + Mapping)

Table:

<table>
<thead>
<tr>
<th>Feature Points</th>
<th># Iterations in NLS</th>
<th># Schur blocks</th>
<th># Update blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-200</td>
<td>6</td>
<td>47</td>
<td>97</td>
</tr>
<tr>
<td>200-250</td>
<td>5</td>
<td>42</td>
<td>63</td>
</tr>
<tr>
<td>250-300</td>
<td>4</td>
<td>35</td>
<td>42</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Lookup Table (0.3 kb)

Automated Self-Update with New Environments

Asynchronous

Feature Points

Software Processing

Hardware Operation

Power

Baseline

KITTI Dataset 5.47W

EuRoC Dataset 5.48W

5.47W

5.48W
Runtime Reconfiguration + Clock Gating

Software Processing
- Feature Points
- Levenberg-Marquardt (LM) Algorithm
- Marginalization Calculation
- 6 DoF poses + 3D coordinates

Hardware Operation
- Sensors: Runtime Reconfig. + Clock Gating (RR + CG)
- NLS Solver Accelerator
- Marginalization Accelerator
- State Vector (Localization + Mapping)

Asynchronous
- Feature Points
- Lookup Table (0.3 kb)
- Automated Self-Update with New Environments

KitTI Dataset
- Baseline 5.47W
- RR+CG 3.73W
- 1.47x

EuRoC Dataset
- Baseline 5.48W
- RR+CG 3.45W
- 1.59x
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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation Frequency</strong></td>
<td>143 MHz</td>
</tr>
<tr>
<td><strong>LUT</strong></td>
<td>144108 (65.92%)</td>
</tr>
<tr>
<td><strong>Flip-Flop</strong></td>
<td>172935 (39.56%)</td>
</tr>
<tr>
<td><strong>BRAM</strong></td>
<td>268 (49.17%)</td>
</tr>
<tr>
<td><strong>DSP</strong></td>
<td>869 (96.56%)</td>
</tr>
</tbody>
</table>

FPGA Zynq-7000 SoC ZC706 with XC7Z045 FFG900-2
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*

![Processing time vs Energy Graph](image)

### EuRoC Dataset (For drone)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>FPGA Speedup</th>
<th>FPGA Energy Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Over CPU</td>
<td>Over TX1</td>
</tr>
<tr>
<td>FPGA ZC706</td>
<td>8.73x</td>
<td>164.40x</td>
</tr>
<tr>
<td>Kintex-7 Series (XC7K160tfg484)</td>
<td>7.01x</td>
<td>180.73x</td>
</tr>
<tr>
<td>Virtix-7 Series (XC7VX690tfg1761)</td>
<td>10.75x</td>
<td>172.05x</td>
</tr>
</tbody>
</table>

### KITTI Dataset (For car)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>FPGA Speedup</th>
<th>FPGA Energy Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Over CPU</td>
<td>Over TX1</td>
</tr>
<tr>
<td>FPGA ZC706</td>
<td>10.49x</td>
<td>182.88x</td>
</tr>
<tr>
<td>Kintex-7 Series (XC7K160tfg484)</td>
<td>8.27x</td>
<td>196.09x</td>
</tr>
<tr>
<td>Virtix-7 Series (XC7VX690tfg1761)</td>
<td>12.71x</td>
<td>188.60x</td>
</tr>
</tbody>
</table>
## Evaluation

### Comparison with Related Work

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Platform</strong></td>
<td>FPGA</td>
<td>ASIC</td>
<td>ASIC</td>
<td>FPGA</td>
<td>FPGA</td>
<td>FPGA</td>
</tr>
<tr>
<td><strong>Technology</strong></td>
<td>28 nm</td>
<td>28 nm</td>
<td>65 nm</td>
<td>28 nm</td>
<td>28 nm</td>
<td>16 nm</td>
</tr>
<tr>
<td><strong>Design</strong></td>
<td>digital</td>
<td>digital</td>
<td>digital</td>
<td>digital</td>
<td>digital</td>
<td>digital</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>SLAM</td>
<td>SLAM</td>
<td>SLAM</td>
<td>SLAM</td>
<td>SLAM</td>
<td>SLAM</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>Levenberg-Marquardt (optimization-based)</td>
<td>Levenberg-Marquardt (optimization-based)</td>
<td>Gaussian-Newton (optimization-based)</td>
<td>Levenberg-Marquardt (optimization-based)</td>
<td>Gaussian-Newton (optimization-based)</td>
<td>Kalman Filter (Filter-based)</td>
</tr>
<tr>
<td><strong>DoF</strong></td>
<td>6-DoF</td>
<td>6-DoF</td>
<td>6-DoF</td>
<td>6-DoF</td>
<td>6-DoF</td>
<td>6-DoF</td>
</tr>
<tr>
<td><strong>Voltage</strong></td>
<td>1 V</td>
<td>0.63-0.9V</td>
<td>1.2V</td>
<td>1 V</td>
<td>1 V</td>
<td>0.85 V</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>3.45W</td>
<td>243.6mW @ 0.9V</td>
<td>61.75mW @ 0.63V</td>
<td>24mW</td>
<td>5.50W</td>
<td>1.46 W</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>143 MHz</td>
<td>240 MHz</td>
<td>62.5/83.3 MHz</td>
<td>143 MHz</td>
<td>100 MHz</td>
<td>180 MHz</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>55.8 GOPS</td>
<td>879.6 GOPS @ 0.9V</td>
<td>329.8 GOPS @ 0.63V</td>
<td>10.5-59.1 GOPS</td>
<td>N/A</td>
<td>4.4-24.6 GOPS</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>16.43 ms</td>
<td>N/A</td>
<td>30.8 ms</td>
<td>110 ms</td>
<td>200 ms</td>
<td>44.6 ms</td>
</tr>
<tr>
<td><strong>Energy per Frame</strong></td>
<td>56.6 mJ</td>
<td>N/A</td>
<td>739.2 µJ</td>
<td>605 mJ</td>
<td>292 mJ</td>
<td>399.6 mJ</td>
</tr>
<tr>
<td><strong>Dynamic Optimiza-</strong></td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

[CICC]
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

[Wan, CICC 2022]
Reference

[Wan, Synthesis Lectures on Comp Arch 2021] [Wan, CICC 2022] [Wan, Circuits and Systems Magazine 2021]
Thank You

gracias
merci
dank
Danke
谢谢
多谢
谢谢
多谢
Thank you

Obrigado
Mesi
Dank
Tak
Vielen Dank
Many thanks
Muchas gracias

ARIGATO
Grazie
Dzieki
Gracias
多谢
多谢
多谢
多谢

CICC