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\(^2\))
  • Semi-direct and semi-dense methods

• Categories of implementations:
  • Software
  • Hardware with focus on
    • Architecture
    • Microarchitecture

Our focus

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 \textit{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

\[
H \times P \times H^T = B
\]

Sparse operation

\[
P_{4 \times 4} \times H^T_{4 \times 2} = A_{4 \times 2}
\]

Dense operation

\[
H^T_{2 \times 4} \times A_{4 \times 2} = B_{2 \times 2}
\]

Dense operation
PISCES – Microarchitecture

• Dense building blocks of Pisces and their latency:

<table>
<thead>
<tr>
<th>Dense matrix operation</th>
<th>$A_{2\times2}^T$</th>
<th>$A_{4\times4} \times B_{4\times2}$</th>
<th>$A_{2\times4} \times B_{4\times2}$</th>
<th>$A_{2\times2} \times B_{2\times4}$ or $A_{4\times2} \times B_{2\times2}$</th>
<th>$A_{2\times2} \pm B_{2\times2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency ($\mu s$)</td>
<td>0.06</td>
<td>0.66</td>
<td>0.33</td>
<td>0.58</td>
<td>0.09</td>
</tr>
</tbody>
</table>

• Pipelines and configurations for EKF and ORB:
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.
\(^3\) Liu et al, DAC, 2019.
Results – Resource Utilization and Power

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

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

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 $8 \times$ 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.