SwiftRL:
Towards Efficient Reinforcement Learning on Real Processing-In-Memory Systems

Kailash Gogineni, Sai Santosh D, Juan Gómez Luna, Karthik Gogineni, Peng Wei, Tian Lan, Mohammad Sadrosadati, Onur Mutlu, Guru Venkataramani
Outline

- Reinforcement learning algorithms
- Processing-in-memory (PIM)
- PIM implementation of RL algorithms
- Evaluation
- Conclusion
Reinforcement Learning (RL) Algorithms

- Offline RL is vital in healthcare, finance, and robotics, ensuring safe optimization where real-time learning is risky.
- Processing large datasets requires significant computational power and memory.
- Representative RL algorithms:
  - Q-learning
    - Applications: AlphaGo, Robotics
  - SARSA
    - Applications: Robotics, Adaptive Control Systems

Source: [https://adabhishek dabas.medium.com/rl-world-3fc4dc38a73d](https://adabhishek dabas.medium.com/rl-world-3fc4dc38a73d)
Reinforcement Learning Algorithms: Roofline Analysis

Roofline plot for RL workloads:
- Constrained by DRAM bandwidth due to repeated memory accesses during the training phase.

Observation: All workloads fall in the memory-bound area of the Roofline.

The Roofline model derives the performance characteristics of CPU versions in RL workloads, where “Q” refers to Q-learner, “S” signifies the SARSA learner, and “1M” and “20M” indicate the data size in millions of transitions.
Outline

- Reinforcement learning algorithms
- Processing-in-memory (PIM)
- PIM implementation of RL algorithms
- Evaluation
- Conclusion
Processing-in-Memory (PIM)

- PIM advocates for memory-centric computing, positioning processing elements near or inside memory arrays.

- Real-world PIM architectures are becoming a reality
  - UPMEM PIM, Samsung HBM-PIM, Samsung AxDIMM, SK Hynix AiM, Alibaba HB-PNM

- These PIM systems have some common characteristics:
  1. There is a host processor (CPU or GPU) with access to (1) standard main memory, and (2) PIM-enabled memory
  2. PIM-enabled memory contains multiple PIM processing elements (PEs) with high bandwidth and low latency memory access
  3. PIM PEs run only at a few hundred MHz and have a small number of registers and small (or no) cache/scratchpad
  4. PIM PEs may need to communicate via the host processor
In our work, we use the UPMEM PIM architecture

- General-purpose processing cores called DRAM Processing Units (DPUs)
  - Up to 24 PIM threads, called tasklets
  - 32-bit integer arithmetic, but multiplication/division are emulated, as well as floating-point operations
  - 8-bit integer multiplication is natively supported
- 64-MB DRAM bank (MRAM), 64-KB scratchpad (WRAM)
The execution phase comprises four main steps:

1. Loading the input dataset chunks into individual DRAM banks of PIM-enabled memory,
2. Executing the RL workload (kernel) on PIM cores in parallel operating on different chunks of data,
3. Retrieving partial results from DRAM banks to the host CPU, and
4. Aggregating partial results on the host processor.
Tabular Q-learning Implementation

Tabular Q-learning is a widely-used model-free and off-policy RL workload that learns through a trial-and-error approach.

Our PIM implementation divides the training dataset \((X)\) equally among PIM cores.

- **Q-Learning Updates in PIM Cores**: PIM cores perform **Q-learning updates** concurrently, each core handling its assigned data chunk.

- **Result Aggregation**: Combine partial results from PIM cores to form the final Q-table on the host processor.
Multi-agent training is increasingly applied across diverse fields.

Developed a variant of Q-learning optimized for hardware adaptability.

Multi-agent Q-learning PIM implementation:

1. Data Loading and Training: Agent-specific datasets loaded into PIM cores for concurrent training of independent learners.

2. Independent Operation: Each agent pinned to a core, trained iteratively, with final Q-tables retrieved directly.
SARSA Implementation

Algorithm 2 SARSA (State-Action-Reward-State-Action) Algorithm

1: Inputs:
2: Experience data collected offline.
3: Initialize a Q-table with arbitrary/zero values.
4: Hyper-parameters: Learning Rate ($\alpha$), Discount Factor ($\gamma$), Number of episodes.
5: for iteration from 1 to Number of episodes do
6:   Batched updates:
7:   for each experience in selected batch do
8:     Q-learning Update:
9:       Update Q-values for the current state-action:
10:      $Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma Q(s', a') - Q(s, a))$
11:   end for
12: end for
13: Output:
14: Final Q-table with the learned Q-values.

- SARSA, an on-policy reinforcement learning algorithm, directly updates Q-values using the next action chosen according to the policy and its associated Q-value.
## Tabular Q-learning & SARSA Variants

<table>
<thead>
<tr>
<th>Q-learner</th>
<th>SARSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>FP32</td>
</tr>
<tr>
<td>INT32</td>
<td>INT32</td>
</tr>
<tr>
<td>STR</td>
<td>STR</td>
</tr>
</tbody>
</table>

### Q-learner Variants
- Q-learner-SEQ-FP32, Q-learner-SEQ-INT32
- Q-learner-STR-FP32, Q-learner-STR-INT32
- Q-learner-RAN-FP32, Q-learner-RAN-INT32

### SARSA Variants
- SARSA-SEQ-FP32, SARSA-SEQ-INT32
- SARSA-STR-FP32, SARSA-STR-INT32
- SARSA-RAN-FP32, SARSA-RAN-INT32
Outline

- Reinforcement learning algorithms
- Processing-in-memory (PIM)
- PIM implementation of RL algorithms
- Evaluation
- Conclusion
Evaluation Methodology

• Algorithms & Datasets:
  – Tabular Q-learning, SARSA, Multi-Agent Q-learners (Independent)
  – Frozen Lake & Taxi from Open AI GYM – 1 million & 5 million experiences

• Evaluated systems:
  – UPMEM PIM system with 2,524 PIM cores @ 425 MHz and 158 GB of DRAM
  – Intel Xeon Silver 4110 CPU
  – NVIDIA RTX 3090 – Ampere Architecture

• Evaluated Performance of RL Algorithm on PIM using:
  – Performance scaling across PIM cores
  – Comparison with FP32 & fixed-point representation + scaling optimization
  – Comparison to CPU & GPU
  – More details in the paper
Evaluation: Strong Scaling-FL

- Tabular Q-learning & SARSA: Strong scaling - 125 to 2,000 PIM cores
- Datasets: Frozen lake

The PIM kernel time decreases by 2x as we linearly increase the number of PIM cores.
Evaluation: Strong Scaling-FL (FP32 & INT32)

- Tabular Q-learning & SARSA: Strong scaling - 125 to 2,000 PIM cores
- Datasets: Frozen lake

With 125 PIM cores, fixed-point (INT32) representation accelerates the PIM kernel time by about 11× over FP32
Evaluation: Strong Scaling-Taxi

• Tabular Q-learning & SARSA: Strong scaling - 125 to 2,000 PIM cores
• Datasets: Taxi

Similar PIM kernel time trends seen in taxi environment.
Evaluation: Strong Scaling-Taxi (Commn.)

• Tabular Q-learning & SARSA: Strong scaling - 125 to 2,000 PIM cores
• Datasets: Taxi

In taxi task, Q-learner-STR-INT32 with 2000 PIM cores peaks at 21.19% for inter-PIM core synchronization.

Taxi environment requires 47× more PIM core data exchange (Q-values) than frozen lake.
Evaluation: Comparing PIM to CPU (SEQ & STR)

Key Takeaway 1. Our INT32 PIM implementation with sequential and stride-based sampling techniques exhibits slower execution times than CPU due to enhanced cache locality and lower CPU-cache latencies.
Evaluation: Comparing PIM to CPU (RAN)

Key Takeaway 2.
In both taxi and FL environments, our fixed-point PIM implementation with random sampling demonstrates superior performance.
**Evaluation: Comparing PIM to GPU (SEQ)**

**Q-learner-SEQ-INT32-FL** achieves a 4.84× speedup over the GPU version due to INT32 instructions.
Evaluation: Multi-agent Q-learning (I)

• UPMEM architecture accelerates training of multiple independent Q-learners with 10,000 transitions (frozen lake) each.

• Utilizes fixed-point representation and scaling optimization.

<table>
<thead>
<tr>
<th># Agents</th>
<th># PIM Cores</th>
<th>Speedup (times)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1000</td>
<td>11.23 ×</td>
</tr>
<tr>
<td>2000</td>
<td>2000</td>
<td>21.92 ×</td>
</tr>
</tbody>
</table>
Evaluation: Multi-agent Q-learning (II)

Key Takeaway 3. Memory-intensive RL algorithms with minimal inter-PIM core communication, such as multi-agent Q-learning, are best suited for UPMEM PIM architecture.
Conclusion

• **Adapted and implemented** RL algorithms on a PIM architecture for exploring memory-centric systems in RL training

• Explored **optimization strategies** for enhancing RL workload performance across
  • various data types,
  • sampling strategies (SEQ, RAN, STR)

• Compared **PIM-based** Q-learning & SARSA on UPMEM PIM (2000 cores) to **CPU & GPU**

• Achieved near-linear **scaling of 15x** in performance with a **16x increase in PIM cores** (125 to 2000)
Thank You!

Q & A

Contact: Kailash Gogineni (kailashg26@gwu.edu)