Micro-Armed Bandit
Lightweight & Reusable Reinforcement Learning for Microarchitecture Decision-Making

Gerasimos Gerogiannis, Josep Torellas

MICRO 2023

presented by Talu Güloglu

Seminar in Computer Architecture
16.05.2024
Executive Summary

**Background:** Online Reinforcement Learning has been effectively integrated into various diverse challenges in microarchitecture.

**Problem:** Two key shortcomings of prior Utilizations:
- **High complexity and storage overhead:**
  - Bookkeeping of states and action values
- RL agents are tailored for specific applications and lack reusability

**Goal:** Design an RL agent with the following characteristics:
- **Lightweight,** which emphasizes efficient microarchitectural design and minimal resource consumption
- **Reusable,** effortlessly applicable to multiple microarchitectural problem domains

**Contribution:** Micro-Armed Bandit, a lightweight and, across different microarchitectural problem domains, reusable RL agent, that:
- Provides lightweight design due to only considering a small fraction of the action space in a given time window (temporal homogeneity)
- Provides ease of design and implementation easily applicable to different problem areas

**Evaluation:** Compared to sota baselines in 1) Data Prefetching and 2) Instruction Fetching in simultaneous multithreaded processors
- MAB outperforms non-RL prefetchers by 2.3-2.6% and attains similar performance to RL prefetcher Pythia
- Imposes a significantly lower storage requirement of only 100 bytes on the system
- MAB outperformed Choi policy in SMT by 2.2%
Talk Outline

Key Shortcomings of current approaches

- Formulating DP and IF as Reinforcement Learning
- Micro-Armed Bandit: Overview
- Evaluation and Key Results
- Conclusion
- Critical Analysis and Discussion
Data prefetching

• Predicts addresses of long-latency memory requests and fetches data before the program demands it

• Associates access patterns from past memory requests with program context information

• Example program features
  - Program counter (PC)
  - Page number
  - Page offset
  - ...

• Usual shortcomings: predict mainly using a single program feature and lack inherent system awareness
Pythia: Customizable Hardware Prefetching Framework using Online Reinforcement Learning

Formulates prefetching as Reinforcement Learning Problem

Pythia has negligible power overhead but requires about 26KB of metadata storage per core

Prefetch degree is decided by a simple degree selector separate from the RL agent
Formulating Prefetching as RL

Agent Environment

State ($S_t$)

Action ($A_t$)

Reward ($R_{t+1}$)

Prefetcher Processor & Memory Subsystem

Reward

Prefetch from address $A+\text{offset (O)}$
Instruction fetching in simultaneous multithreaded processors

• Fetching instructions from **multiple threads** concurrently
• Fetch-priority policies focus on various metrics such as:
  - Branch Count
  - Instruction Count
  - #entries in Load-Store Queue
  - ...
• Choi and Yeung introduced an **adaptive** mechanism for Fetch Priority and Gating of threads
Instruction fetching in simultaneous multithreaded processors

- Fetching instructions from **multiple threads** concurrently
- Fetch-priority policies focus on various metrics such as:
  - Branch Count
  - Instruction Count
  - #entries in Load-Store Queue
  - …

Choi and Yeung introduced an adaptive mechanism for Fetch Priority and Gating of threads.

**Low adaptability**

Only Instruction Count is used for priority policy

Fetch gating policy is fixed
Markov Decision Process Reinforcement Learning:
- Multiple states
- Actions lead to non-deterministic transitions between different states
- Actions (depending on state) lead to a different reward

High complexity from bookkeeping action values and transition probabilities for each state (e.g. Q-Learning)
**RL problem formulations**

**Removes** state transition probabilities

Transitions can still occur but are **not influenced** by agent's actions

**Randomly transition** between states according to other effects
Collapses to a **single state**
Reward only depends on the taken agent action
Each agent choice is made **independently** from context and external factors
RL problem formulations

More Lightweight

MDP-RL

Contextual Bandits

Multi-Armed Bandits

(a)

(b)

(c)

More expressive
Is Multi-Armed Bandit good enough?
Temporal homogeneity in the action space

Consider a scenario where the **same actions** is **repeatedly optimal** for a large enough period inside an RL episode.

- Action is **temporally optimal** regardless of the environment.

**Temporal homogeneity of the action space**

- High temporal homogeneity allows us to **reduce the environment states** to a single state since a few actions suffice to model the action space **in a given time window**.

---

(a) Fully Temporal Homogeneous Action Space

(b) Temporal Heterogeneous Action Space
Temporal Homogeneity in Prefetching

- Identifying **temporal homogeneity** of prefetching action space is done by monitoring the most common actions from Pythia (64 actions; 16 different offsets and 4 different degrees)
Temporal Homogeneity in Prefetching

- Identifying **temporal homogeneity** of prefetching action space is done by monitoring the most common actions from Pythia (64 actions: 16 different offsets and 4 different degrees).

Most selected action accounts for **60%**
- Second most selected action for **15%**
- **Sufficient Temporal homogeneity allows us to use MAB as coordinator** of various conventional lightweight prefetchers.
Extending the Choi algorithm

- Add variations to the Fetch priority and the Fetch-gate policies

Table 1: Examples of different fetch PG policies.

<table>
<thead>
<tr>
<th>Policy Mnemonic</th>
<th>Fetch Priority</th>
<th>Fetch-gate if occupancy of any of these structures exceeds threshold:</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC_0000</td>
<td>ICount</td>
<td>IQ  LSQ  ROB  IRF</td>
</tr>
<tr>
<td>BrC_1000</td>
<td>Branch Count</td>
<td>yes  no  no  no</td>
</tr>
<tr>
<td>IC_1110</td>
<td>ICount</td>
<td>yes  yes yes no</td>
</tr>
<tr>
<td>IC_1111</td>
<td>ICount</td>
<td>yes  yes yes yes</td>
</tr>
<tr>
<td>LSQC_1111</td>
<td>LSQ Count</td>
<td>yes  yes yes yes</td>
</tr>
<tr>
<td>RR_1111</td>
<td>Round Robin</td>
<td>yes  yes yes yes</td>
</tr>
</tbody>
</table>
Extending the Choi algorithm

- Add variations to the Fetch priority and the Fetch-gate policies

Different priority gate policies work best for different workloads
Talk Outline

- Key Shortcomings of current RL techniques
- Formulating DP and IF as Reinforcement Learning
- Micro-Armed Bandit: Overview
- Evaluation and Key Results
- Conclusion
- Critical Analysis and Discussion
Multi-armed Bandit algorithm

Requires: m arms, two tables \( r \) and \( n \)
- table \( n \): Number of times arm \( i \) has been selected
- table \( r \): Average reward of arm \( i \)

1. Initial round-robin phase
   All arms are consecutively tried once to fill \( n \) and \( r \)

2. Main Loop
   a) Choose the next arm
   b) Update the Selection
   c) Update the reward of the selected arm
Three MAB algorithms

**$\epsilon$-Greedy**

- **nextArm**
  \[ \text{arm} \leftarrow \begin{cases} \arg \max \{ r_i \} & \text{with prob. } 1 - \epsilon \\ \text{random arm} & \text{with prob. } \epsilon \end{cases} \]

- **updSels**
  \[ \begin{align*} n_{\text{arm}} & \leftarrow n_{\text{arm}} + 1 \\ n_{\text{total}} & \leftarrow n_{\text{total}} + 1 \end{align*} \]

- **updRew**
  \[ r_{\text{arm}} \leftarrow \frac{(r_{\text{arm}} \times (n_{\text{arm}} - 1) + r_{\text{step}})}{n_{\text{arm}}} \]

**Upper Confidence Bound**

- **nextArm**
  \[ \text{arm} \leftarrow \arg \max \{ r_i + c \sqrt{\frac{\ln(n_{\text{total}})}{n_i}} \} \]

- **updSels**
  \[ \begin{align*} n_{\text{arm}} & \leftarrow n_{\text{arm}} + 1 \\ n_{\text{total}} & \leftarrow n_{\text{total}} + 1 \end{align*} \]

- **updRew**
  \[ r_{\text{arm}} \leftarrow \frac{(r_{\text{arm}} \times (n_{\text{arm}} - 1) + r_{\text{step}})}{n_{\text{arm}}} \]

**Discounted Upper Confidence Bound**

- **nextArm**
  \[ \text{arm} \leftarrow \arg \max \{ r_i + c \sqrt{\frac{\ln(n_{\text{total}})}{n_i}} \} \]

- **updSels**
  \[ \begin{align*} n_i & \leftarrow \gamma \times n_i, \forall i \in \{1, M\} \\ n_{\text{arm}} & \leftarrow n_{\text{arm}} + 1 \\ n_{\text{total}} & \leftarrow \gamma \times n_{\text{total}} + 1 \end{align*} \]

- **updRew**
  \[ r_{\text{arm}} \leftarrow \frac{(r_{\text{arm}} \times (n_{\text{arm}} - 1) + r_{\text{step}})}{n_{\text{arm}}} \]
Microarchitecture and Functionality

Two Tables (n table, r table)

One Arithmetic unit for nextarm, updateSels, updRew

Control logic triggers:
- selection of the next arm
- communicates selected arm
- computes rewards
Prefetching Use Case

• MAB as coordinator for **NL** prefetcher, **Stream** prefetcher and **PC-based** stride prefetcher

• Actions:
  - On/Off for NL prefetcher
  - Degree of Stream prefetcher
  - Degree of Stride prefetcher

• Reward: IPC

<table>
<thead>
<tr>
<th>Arm id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL On/Off</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>Stride Degree</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>Streamer Deg.</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>
SMT Instruction Fetch Use Case

- Control Fetch **Priority** and **Gating policy** of an SMT processor

- **Actions:**
  - Fetch Gating policy (which structures do we monitor?)
  - Fetch Priority policy (how do we choose from the remaining threads)

- **Reward:** IPC

- Additionally **Hill-climbing** for the best occupancy threshold to trigger Fetch Gating

<table>
<thead>
<tr>
<th>Policy Mnemonic</th>
<th>Fetch Priority</th>
<th>IQ</th>
<th>LSQ</th>
<th>ROB</th>
<th>IRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC_0000</td>
<td>ICount</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>BrC_1000</td>
<td>Branch Count</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>IC_1110</td>
<td>ICount</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>IC_1111</td>
<td>ICount</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>LSQC_1111</td>
<td>LSQ Count</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>RR_1111</td>
<td>Round Robin</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Talk Outline

- Key Shortcomings of current RL techniques
- Formulating DP and IF as Reinforcement Learning
- Micro-Armed Bandit: Overview
- Evaluation and Key Results
- Conclusion
- Critical Analysis and Discussion
Simulation Methodology

• **Champsim** trace-driven simulator for data prefetching

• Single-core and 4-core memory-intensive workload traces
  - SPEC CPU2006 and CPU2017
  - PARSEC 2.1
  - Ligra
  - Cloudsuite

• Five state-of-the-art prefetchers (IP-Stride, Pythia, MLOP, Bingo, IPCP)

• **Gen5v20** [3] execution-driven simulator for SMT
  - Simpoints from 22 SPEC17 apps captured using reference input set
  - 226 2-threaded combinations created for simulation
  - Simulation run until each thread completes 150M instructions

• Evaluate against ICount and Choi policy
Performance Analysis for Prefetching Single-core

Bandit outperforms Stride by 9%, Bingo by 2.6%, MLOP by 2.3% and Pythia by 0.2%
Performance Analysis for Prefetching Multi-Core

Bandit outperforms Stride by 6% Bingo by 4%, MLOP by 2.4%

Bandit performs worse than Pythia by 0.2%

Reward is more noisy!
Performance on SMT Thread Fetching

Bandit reduces the fraction of stalled cycles
Eliminates conservative fetch gating

Bandit outperforms Choi policy by 2.2%, and sole ICount by 7%
Talk Outline

- Key Shortcomings of current RL techniques
- Formulating DP and IF as Reinforcement Learning
- Micro-Armed Bandit: Overview
- Evaluation and Key Results
- Conclusion
- Critical Analysis and Discussion
Conclusion

• Introduce Micro-Armed Bandit, an RL agent based on Multi-Armed Bandit algorithms that is lightweight and reusable

• Contributions:
  - Concept of temporal homogeneity in the action space
  - Adaption of Multi-Armed Bandits as an efficient and reusable learning mechanism for such problems
  - RL agent that implements the Discounted Upper confidence bound in hardware
  - Simulation-based evaluation for data prefetching and SMT instruction fetching
  - Customizing Fetch priority and Fetch gating policy

• Evaluations:
  - MAB outperforms non-RL prefetchers by 2.3-2.6% and attains similar performance to RL prefetcher Pythia
  - Imposes a significantly lower storage requirement of only 100 bytes on the system
  - MAB outperformed Choi policy in SMT by 2.2%
Strengths

• Addresses **real world challenges** of reinforcement learning in computer architecture (high complexity and storage overhead, limited reusability)

• Is **easily integrable** with different computer architectural decision-making problems

• Evaluated multiple **different configurations, problem domains**, and various traces

• **Exploration analysis** of different MAB algorithms essential for understanding decision-making of different algorithms
Weaknesses

• **Temporal Homogenity:**
  a) Temporal homogeneity might be worse for more complex workloads

b) The motivational analysis for temporal homogeneity in data prefetching is done for Pythia! (Pythia might make **suboptimal** decisions; temporal homogeneity might not be a characteristic of the problem domain but the “best” solution Pythia came up with)

• Choi algorithms exhibit **nonadaptive features** (e.g. fetch gating threshold)

• Hyperparameter **tuning** is done **for chosen workloads**
Discussion (1/3)

Machine Learning in Computer Architecture

![Bar chart showing the paper count per year from Before 2004 to 2016-2019. The paper count increases significantly from 2012-2015 to 2016-2019.](chart_image)
Discussion (1/3)

Machine Learning in Computer Architecture

1. Complexity and Interpretability
2. Performance and Efficiency
3. Data Dependence and Generalization
4. Robustness and Reliability
Discussion (2/3)

• What other problem domains in Microarchitecture could be modeled using RL (specifically MAB) (that might exhibit temporal homogeneity)

1. Caches and Prefetching
2. Schedulers and Control
3. Branch Prediction
What **Security Vulnerability** could ML-based approaches lead to in Microarchitecture?

- **Denial of Service attacks**
  - Malicious workloads in parallel to real workloads to learn different access patterns
  - Significantly slowing down performance

- **Solutions:**
  - Improve training stability by limiting the changes to the policy at each training epoch
  - May not work for workloads with large variation
BACKUP SLIDES
Extending the Choi algorithm

- Revisit Choi algorithm shortcomings (**Fixed priority policy** and **Fetch gating policy**)
- Assume a 2-threaded scenario where we limit the occupancy of all three structures to 50%
  One thread wants to use more entries from the ROB and needs fewer entries from the IQ, while the other thread has asymmetric behavior
  Normally, both threads are fetch-gated
  ➔ Exclude monitoring of ROB or the IQ (or both) from the fetch gating policy to attain higher performance
Performance on Tune set

- Tune hyperparameters (bandit step duration, Hill climbing hyperparameters in SMT)
- Single: Stop exploring after initial round-robin phase and keeps the best-performing arm
- Periodic: Alternates between periodic phases of round robin exploration and the best arm

<table>
<thead>
<tr>
<th></th>
<th>Pythia</th>
<th>Single</th>
<th>Periodic</th>
<th>$\epsilon$-Greedy</th>
<th>UCB</th>
<th>DUCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>88.7</td>
<td>72.8</td>
<td>80.3</td>
<td>89.8</td>
<td>88.6</td>
<td>95.0</td>
</tr>
<tr>
<td>max</td>
<td>102.5</td>
<td>100.0</td>
<td>99.8</td>
<td>99.9</td>
<td>100.0</td>
<td>101.6</td>
</tr>
<tr>
<td>gmean</td>
<td>98.4</td>
<td>96.5</td>
<td>94.1</td>
<td>97.3</td>
<td>98.8</td>
<td>99.1</td>
</tr>
</tbody>
</table>

Data prefetching

<table>
<thead>
<tr>
<th></th>
<th>Choi</th>
<th>Single</th>
<th>Periodic</th>
<th>$\epsilon$-Greedy</th>
<th>UCB</th>
<th>DUCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>77.2</td>
<td>77.8</td>
<td>88.4</td>
<td>92.0</td>
<td>90.9</td>
<td>92.2</td>
</tr>
<tr>
<td>max</td>
<td>101.0</td>
<td>101.1</td>
<td>100.4</td>
<td>100.5</td>
<td>101.1</td>
<td>101.4</td>
</tr>
<tr>
<td>gmean</td>
<td>94.5</td>
<td>96.8</td>
<td>97.2</td>
<td>97.8</td>
<td>98.4</td>
<td>98.6</td>
</tr>
</tbody>
</table>

SMT (percentage of IPC relative best static arm)
Exploration for different algorithms

- **cactus (Prefetch)**
  - (a-0) IPC: 1.24
  - (a-1) IPC: 0.90
  - (a-2) IPC: 1.24
  - (a-3) IPC: 1.23

- **mcf (Prefetch)**
  - (b-0) IPC: 0.20
  - (b-1) IPC: 0.20
  - (b-2) IPC: 0.20
  - (b-3) IPC: 0.21

- **gcc-ibm (SMT)**
  - (c-0) IPC: 0.90
  - (c-1) IPC: 0.70
  - (c-2) IPC: 0.89
  - (c-3) IPC: 0.89

- **cactus-ibm (SMT)**
  - (d-0) IPC: 0.76
  - (d-1) IPC: 0.76
  - (d-2) IPC: 0.77
  - (d-3) IPC: 0.77