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

Gerasimos Gerogiannis and Josep Torrellas

University of Illinois at Urbana-Champaign

gg24@illinois.edu
Reinforcement Learning for Microarchitecture

- Reinforcement Learning (RL) gaining traction in microarchitecture

- Subclass of ML for action selection problems
- Agent, environment, action, feedback (reward)
- Goal: maximize accumulated reward in the long-term

- Prefetching, memory controllers, cache coherence for heterogeneous accelerators …
Reinforcement Learning is attractive for microarchitecture:

- Can learn, adapt and generalize online at runtime
- No need for offline data collection
- No need for offline static system model

Although effective current RL-based microarchitecture agents:

- Are associated with high complexity introduced by decomposing the environment in a complex set of states
- Are not reusable across different use-cases
Contributions

- We introduce a necessary property of microarchitectural problems that enables the usage of the lightweight Multi-Armed Bandit RL algorithms.

- We propose a hardware agent called **Micro-Armed Bandit (Bandit)** that is based on the Multi-Armed Bandit algorithms and is:
  - Lightweight (only 100B!)
  - Reuseable across different microarchitecture problems

- We evaluate Bandit for 2 different problems:
  - Data prefetching
  - Instruction fetch thread selection in Simultaneous Multithreaded processors
RL problem formulations

decreasing complexity

MDP-RL

Contextual Bandits

Multi-Armed Bandits

- Multi-Armed Bandits (MAB):
  - Single state
  - Only need to track action-values for a single state
Property: Temporal Homogeneity in the Action Space

If same action is repeatedly optimal for enough time
- We do not need multiple states
- Multi-Armed Bandits (MAB) is good enough
Multi-Armed Bandits for microarchitecture are good when:

1. action space is temporally homogeneous and
2. different actions are optimal across different benchmarks or benchmark phases

- Microarchitecture problems with temporal homogeneity considered in this work:
  - data prefetching
  - SMT instruction fetch thread selection
Temporal Homogeneity in Prefetching

- Pythia [Bera-MICRO’21]: state of the art MDP-RL based prefetcher
- Uses 16 different offsets and 4 different degrees (64 total actions)

- Top-2 actions in each application account for 75% of the action selections

Prefetching has high temporal homogeneity
SMT instruction fetch thread selection

- SMT processors: which thread to fetch instructions from?
  - Gate thread if it uses too many resources [Choi-ISCA’06]
  - Give priority to certain threads over others [Tullsen-ISCA’96]
Adapting SMT instruction fetch

Examples of fetch priority-gating policies

<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>

- Different fetch priority-gating policy combinations work best for different benchmarks – static oracle adaptation can provide up to 30% performance improvement.

The SMT instruction fetch shows adaptation opportunities that can be exploited with Multi-Armed Bandits at runtime.
Multi-Armed Bandit algorithms

- **arm**: action of the Multi-Armed Bandit agent
- **bandit step**: time duration for which agent is idle waiting to observe the reward from its previous action selection
- **$r_{step}$**: reward sample received at the end of bandit step

**Initial Round Robin Phase**

- $arm = a_1$
- $r_1 = r_{step}$
- $n_1 = 1$

- $arm = a_2$
- $r_2 = r_{step}$
- $n_2 = 1$

- $arm = a_3$
- $r_3 = r_{step}$
- $n_3 = 1$

**Main Phase**

- $arm = nextArm$
- $r_{arm} = updRew$
- $n_{arm} = updSels$

$n$: number of arm selections, $r$: average reward of arm
Key Functions in Multi-Armed Bandit (MAB) algorithms

- **nextArm**: selects the next arm by tackling the exploration-exploitation tradeoff

- **updSels**: updates the number of arm selections

- **updRew**: updates the arm reward after the bandit step is over
Three MAB algorithms for microarchitecture

- **ε-Greedy:**
  - *nextArm:* random, exploration rate does not decrease with time
  - *updSels:* selections are simply incremented
  - *updRew:* reward sample is added to running average

- **Upper Confidence Bound (UCB):**
  - *nextArm:* (1) exploration accounts for past reward and (2) exploration rate decreases with time
  - Actions that have resulted in very poor performance (e.g. significant IPC drops) have smaller exploration probability than near-optimal actions

- **Discounted Upper Confidence Bound (DUCB):**
  - *updSels:* selections are discounted (gradually forgotten)
  - Allows for adapting to dynamic program phases

- More details and microarchitecture-inspired algorithmic modifications in the paper
The Micro-Armed Bandit Microarchitecture

- Hardware agent
- Consists of 2 tables:
  - nTable: contains # times an arm has been selected
  - rTable: contains average reward for the arm
  - As many entries as arms (storage: 100B)
- Selects the prefetching and SMT fetching action
- Reward: IPC during a bandit step
Methodology

- Simulation with ChampSim for prefetching and gem5 for SMT
- We use the DUCB algorithm as it shows the best performance
- Traces from SPEC06, SPEC17, Ligra, PARSEC and Cloudsuite for prefetching (1B instructions)
- Simpoints from SPEC17 for SMT (150M instructions)
- Skylake-like simulated processor parameters
Evaluation Highlights

- Arm selection with Bandit has very similar geomean performance with a static oracle (99.1% for prefetching and 98.6% for SMT) that selects the best arm.

- For prefetching:
  - Outperforms Bingo[Bakhshalipour-HPCA’19] and MLOP[Shakerinava-DPC3] by 2.6% and 2.3% and matches the performance of Pythia[Bera-MICRO’21] and IPCP[Pakalapati-ISCA’20]

- For SMT:
  - Outperforms ICount[Tullsen-ISCA’96] by 7% and Hill Climbing[Choi-ISCA’06] by 2.2%

- Less than 0.003% area and power overhead on top of a conventional multi-core (equipped with stride/stream/NL prefetchers)
We propose a hardware agent based on Multi-Armed Bandits that is (1) lightweight and (2) reusable and evaluate it for:

- Data prefetching
- SMT instruction fetch

**Learning 1:** Very simple ML algorithms can be beneficial for microarchitecture, reducing area cost and implementation complexity

**Learning 2:** Overhead can be further reduced if we design ML agents that are reusable across different use-cases
Micro-Armed Bandit:
Lightweight & Reusable Reinforcement Learning for Microarchitecture Decision-Making

Gerasimos Gerogiannis and Josep Torrellas
University of Illinois at Urbana-Champaign
gg24@illinois.edu
The Micro-Armed Bandit

- Hardware agent that implements MAB algorithms
- Consists of 2 tables and a simple arithmetic unit → 100B storage overhead
- Used to select the degree and type of a set of lightweight L2 prefetchers (NL, Stream, Stride)
- Used to select the instruction fetch priority-gating policy of the SMT front-end
- The IPC during a bandit step is used as the reward
- 4 basic functionalities
- Most of the functionalities are implemented in the background resulting in very low latency (50 cycles)
Multi-Armed Bandit algorithms

- Single state
- Different actions available
- Goal is to find the best action while minimizing time spent in suboptimal actions (exploration vs exploitation)
- arm: action available to the MAB agent
- bandit step: duration for which agent is idle waiting to observe the reward from its previous action selection
- $r_i$: average reward previous selections of arm $i$ have yielded
- $n_i$: total selections of arm $i$ in the past
- $n_{total}$: total selections of all arms in the past
- $r_{step}$: reward sample received at the end of bandit step
- Different MAB algorithms differ on the nextArm, updSels and updRew functions

**Algorithm 1** General template for MAB algorithms

1: Inputs: $M$ arms
2: Variables: $r_i$: average reward of arm $i$,
   $n_i$: number of times that arm $i$ has been selected

<table>
<thead>
<tr>
<th>Initial Round Robin Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{total} \leftarrow 0$</td>
</tr>
<tr>
<td>for $t = 1$ to $M$ do</td>
</tr>
<tr>
<td>arm $\leftarrow t$</td>
</tr>
<tr>
<td>$n_{arm} \leftarrow 1$</td>
</tr>
<tr>
<td>$n_{total} \leftarrow n_{total} + 1$</td>
</tr>
<tr>
<td>// receive reward at the end of the bandit step</td>
</tr>
<tr>
<td>$r_{arm} \leftarrow r_{step}$</td>
</tr>
<tr>
<td>end for</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Main Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>for $t = M + 1$ to $\infty$ do</td>
</tr>
<tr>
<td>arm $\leftarrow$ nextArm()</td>
</tr>
<tr>
<td>updSels(arm)</td>
</tr>
<tr>
<td>// receive reward at the end of the bandit step</td>
</tr>
<tr>
<td>$r_{arm} \leftarrow$ updRew($r_{step}$)</td>
</tr>
<tr>
<td>end for</td>
</tr>
</tbody>
</table>
Methodology

- Arms used in prefetching:

<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>On</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>Stride Degree</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</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>8</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

- Arms used in SMT:

<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></td>
<td></td>
<td>IQ</td>
</tr>
<tr>
<td>IC_0000</td>
<td>ICount</td>
<td>no</td>
</tr>
<tr>
<td>BrC_1000</td>
<td>Branch Count</td>
<td>yes</td>
</tr>
<tr>
<td>IC_1110</td>
<td>ICount</td>
<td>yes</td>
</tr>
<tr>
<td>IC_1111</td>
<td>ICount</td>
<td>yes</td>
</tr>
<tr>
<td>LSQC_1111</td>
<td>LSQ Count</td>
<td>yes</td>
</tr>
<tr>
<td>RR_1111</td>
<td>Round Robin</td>
<td>yes</td>
</tr>
</tbody>
</table>
Opportunities for adapting the fetch PG policy

- The best policy heavily depends on the application mix
$\epsilon$-Greedy

\begin{align*}
\text{nextArm} & \quad \text{arm} \leftarrow \begin{cases} 
\arg\max\{r_i\} & \text{with prob. } 1 - \epsilon \\
\text{random arm} & \text{with prob. } \epsilon 
\end{cases} \\
\text{updSels} & \quad n_{\text{arm}} \leftarrow n_{\text{arm}} + 1 \\
& \quad n_{\text{total}} \leftarrow n_{\text{total}} + 1 \\
\text{updRew} & \quad r_{\text{arm}} \leftarrow \frac{(r_{\text{arm}} \cdot (n_{\text{arm}} - 1) + r_{\text{step}})}{n_{\text{arm}}} 
\end{align*}

- Random non-decaying exploration
- Selections incremented
- Step reward added to the running average
Upper Confidence Bound (UCB)

Upper Confidence Bound

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

\[ \text{updSels} \]
\[ n_{arm} \leftarrow n_{arm} + 1 \]
\[ n_{total} \leftarrow n_{total} + 1 \]

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

- Very bad actions and nearly-optimal actions have different exploration chances
Discounted Upper Confidence Bound (DUCB)

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

\[
\text{updSels:} \quad n_i \leftarrow \gamma \times n_i, \forall i \in [1, M] \\
\text{ } \quad \text{ } n_{arm} \leftarrow n_{arm} + 1 \\
\text{ } \quad \text{ } n_{total} \leftarrow \gamma \times n_{total} + 1
\]

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

- Can adapt to dynamic workload phases
- More in the paper: microarchitecture-inspired modifications