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

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN i-acoma group

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 for Microarchitecture

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

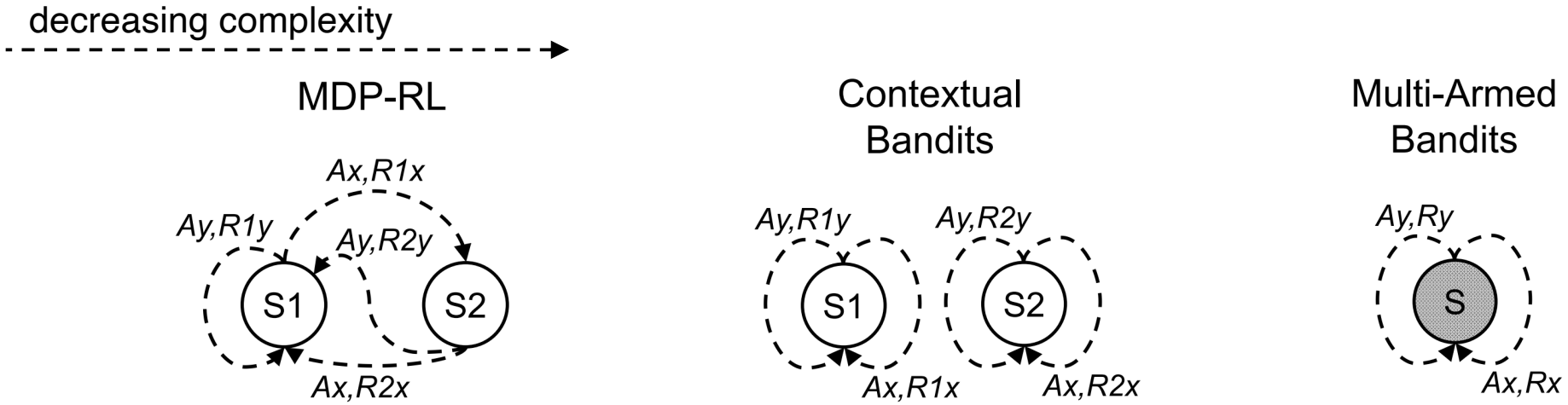
Contributions

- 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

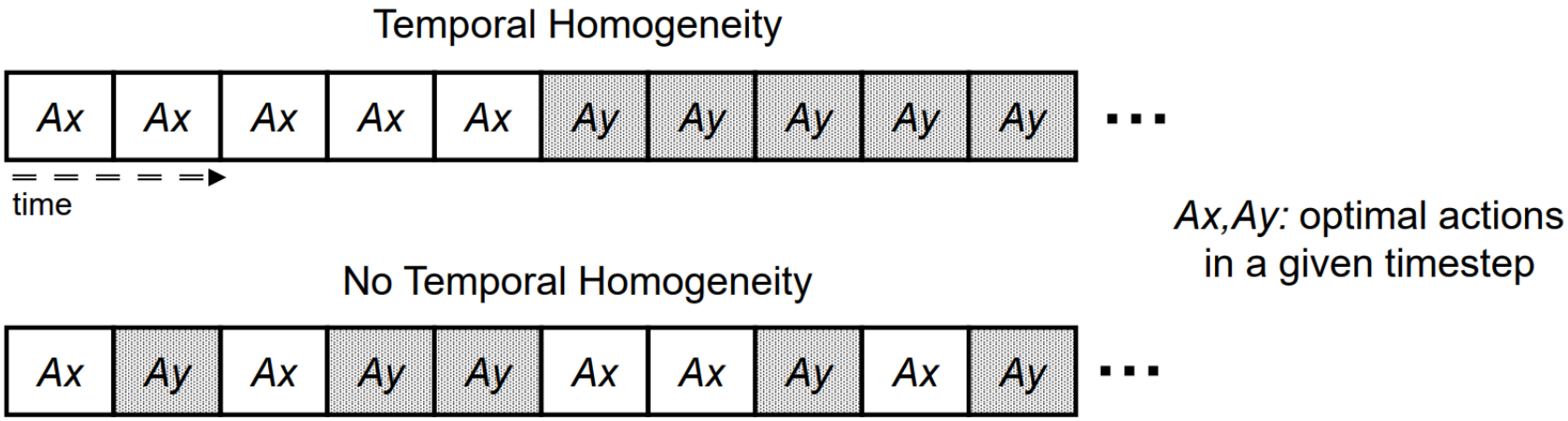
- We evaluate Bandit for 2 different problems:
  - ✓ Data prefetching
  - ✓ Instruction fetch thread selection in Simultaneous Multithreaded processors

RL problem formulations



- 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

Property: Temporal Homogeneity in the Action Space

Multi-Armed Bandits for microarchitecture are good when:

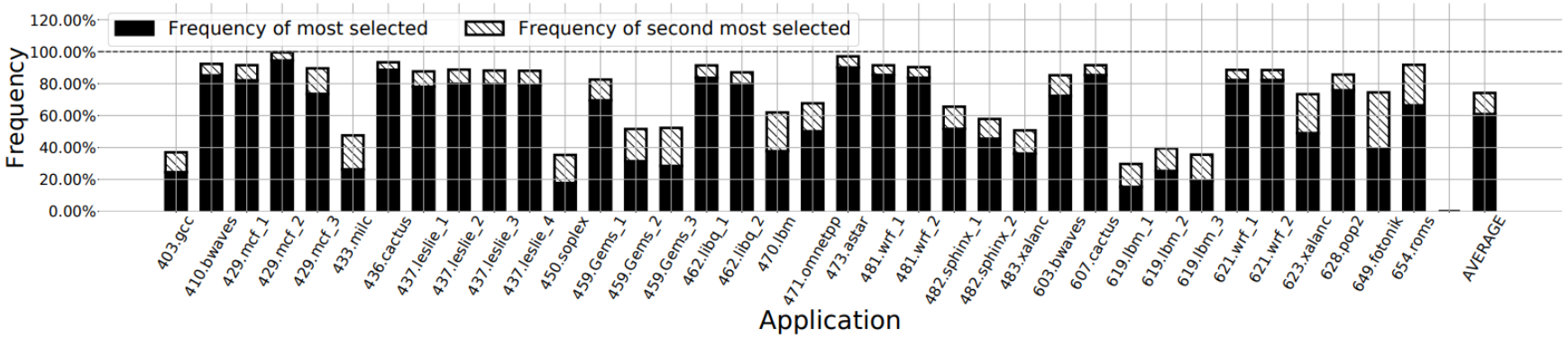
(1) action space is temporally homogeneous and

(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

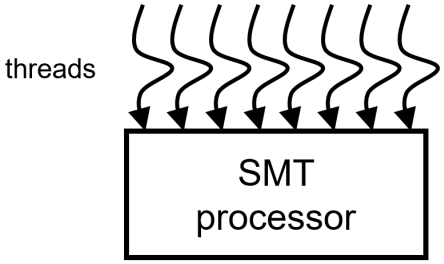
- Pythia [Bera-MICRO'21]: state of the art MDP-RL based prefetcher
- Uses 16 different offsets and 4 different degrees (64 total actions)



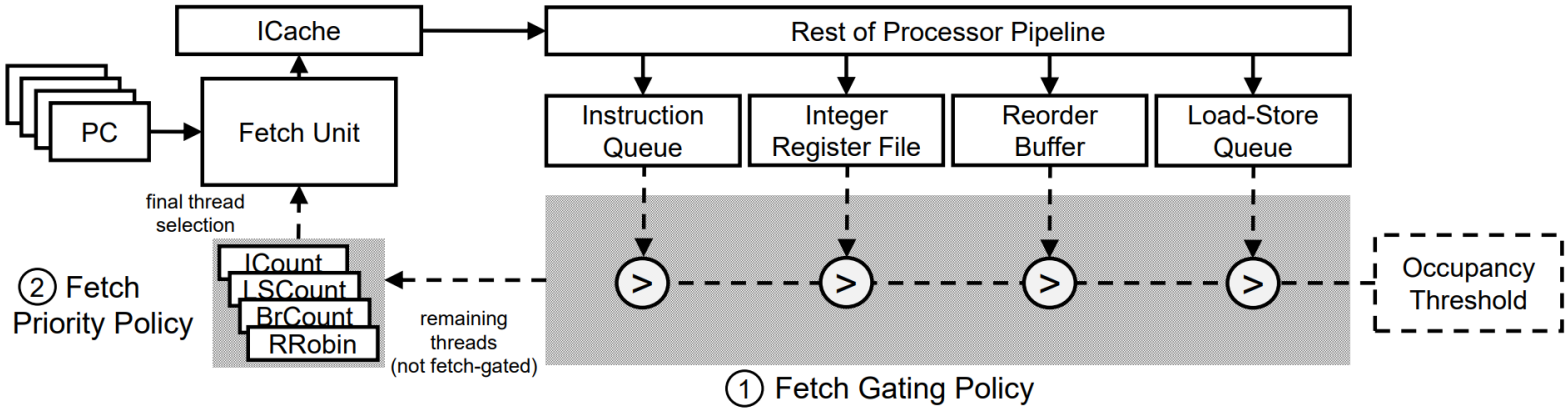
- Top-2 actions in each application account for 75% of the action selections

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

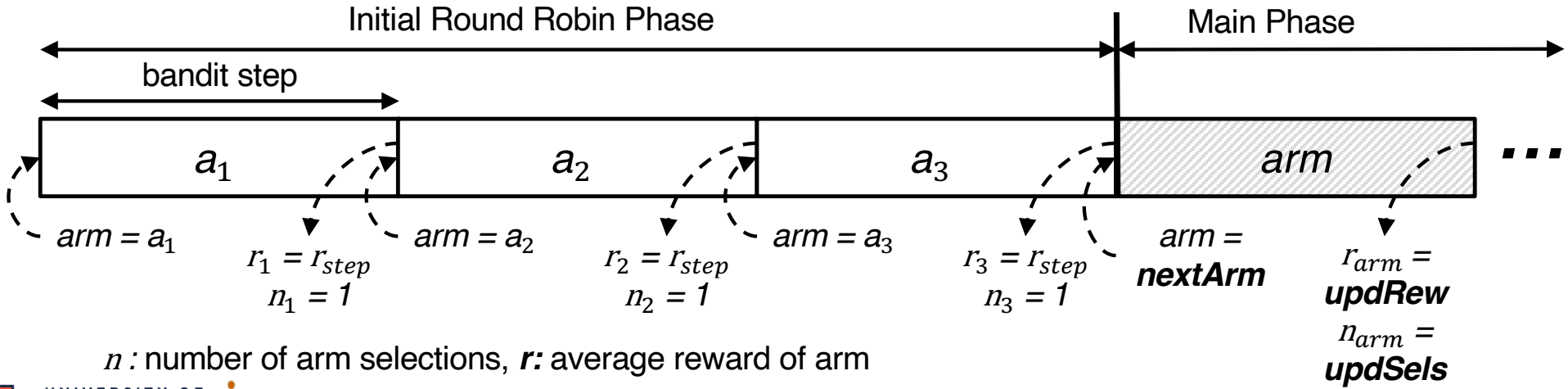
Policy Mnemonic	Fetch Priority	Fetch-gate if occupancy of any of these structures exceeds threshold: IQ	LSQ	ROB	IRF
IC_0000	ICount	no	no	no	no
BrC_1000	Branch Count	yes	no	no	no
IC_1110	ICount	yes	yes	yes	no
IC_1111	ICount	yes	yes	yes	yes
LSQC_1111	LSQ Count	yes	yes	yes	yes
RR_1111	Round Robin	yes	yes	yes	yes

- 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



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

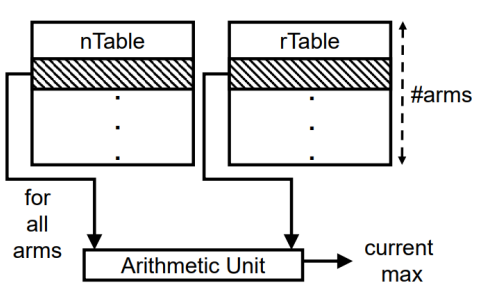
- Discounted Upper Confidence Bound (DUCB):
  - ***updSels:*** selections are discounted (gradually forgotten)

  Allows for adapting to dynamic program phases

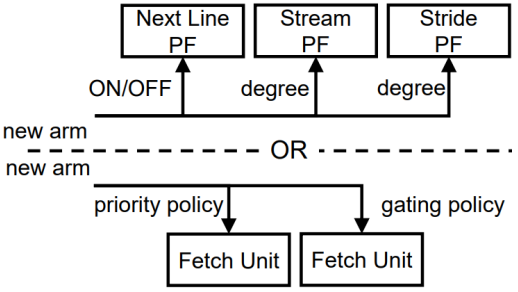
- More details and microarchitecture-inspired algorithmic modifications in the paper

The Micro-Armed Bandit Microarchitecture

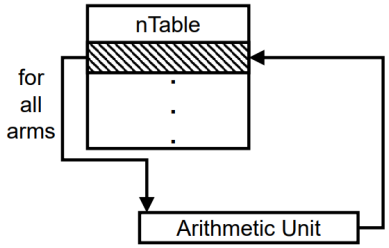
- Hardware agent
- Consists of 2 tables:
  - nTable: contains # times an arm has been selected
  - rTable: contains average reward for the arm
  - As many entries as arms (storage: 100B)
- Selects the prefetching and SMT fetching action
- Reward: IPC during a bandit step



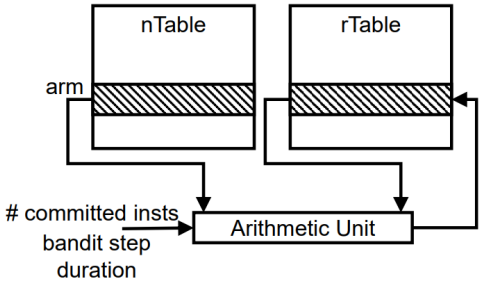
(a) Next Arm Selection



(b) Communicate Arm Selection



(c) Update Selections



(d) Update Reward

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)

Summary and Learnings

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

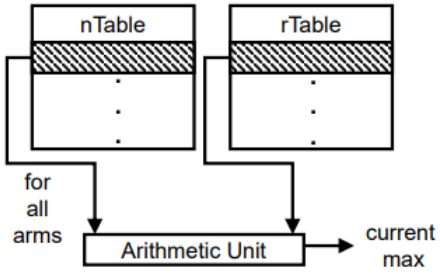
University of Illinois at Urbana-Champaign

gg24@illinois.edu

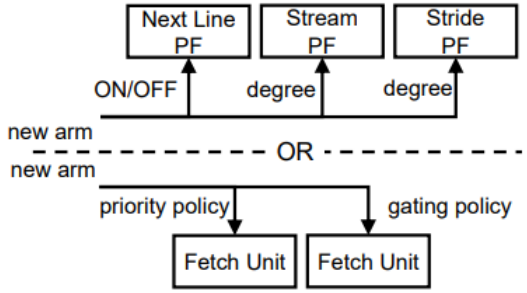
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN i-acoma group

The Micro-Armed Bandit

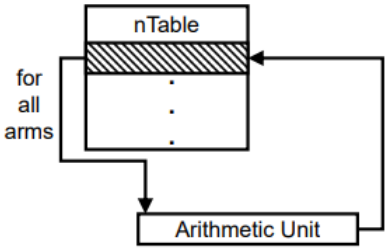
- Hardware agent that implements MAB algorithms
- Consists of 2 tables and a simple arithmetic unit → 100B storage overhead
- Used to select the degree and type of a set of lightweight L2 prefetchers (NL, Stream, Stride)
- Used to select the instruction fetch priority-gating policy of the SMT front-end
- 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)



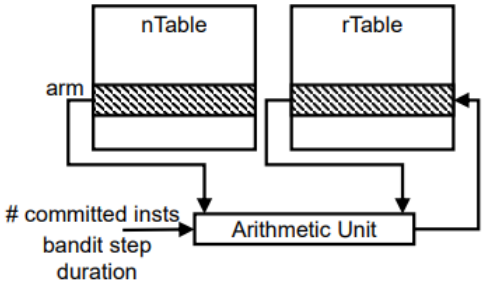
(a) Next Arm Selection



(b) Communicate Arm Selection



(c) Update Selections



(d) Update Reward

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

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

Initial Round Robin Phase

$n_{total} \leftarrow 0$
for $t = 1$ to M **do**
    $arm \leftarrow t$
    $n_{arm} \leftarrow 1$
    $n_{total} \leftarrow n_{total} + 1$
    // receive reward at the end of the bandit step
    $r_{arm} \leftarrow r_{step}$
end for

Main Loop

for $t = M + 1$ to ∞ **do**
    $arm \leftarrow$ ***nextArm()***
    updSels*(arm)*
    // receive reward at the end of the bandit step
    $r_{arm} \leftarrow$ ***updRew****(r_{step})*
end for

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN i-acoma group

Methodology

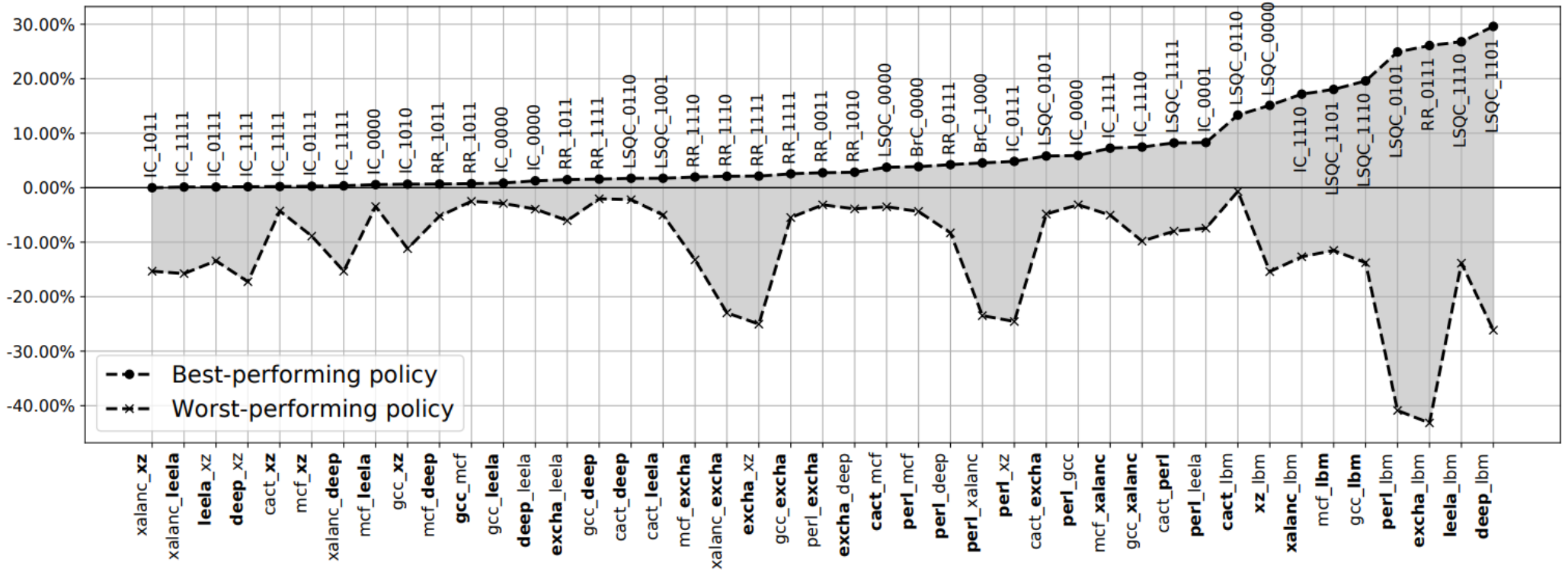
- Arms used in prefetching:

Arm id	0	1	2	3	4	5	6	7	8	9	10
NL On/Off	Off	Off	On	Off	Off	Off	Off	Off	On	Off	Off
Stride Degree	0	0	0	0	2	4	0	8	0	0	15
Streamer Deg.	4	0	0	2	2	4	6	6	8	15	15

- Arms used in SMT:

Policy Mnemonic	Fetch Priority	Fetch-gate if occupancy of any of these structures exceeds threshold: IQ	LSQ	ROB	IRF
IC_0000	ICount	no	no	no	no
BrC_1000	Branch Count	yes	no	no	no
IC_1110	ICount	yes	yes	yes	no
IC_1111	ICount	yes	yes	yes	yes
LSQC_1111	LSQ Count	yes	yes	yes	yes
RR_1111	Round Robin	yes	yes	yes	yes

Opportunities for adapting the fetch PG policy



- The best policy heavily depends on the application mix

ε-Greedy

ϵ-Greedy

nextArm

$$arm \leftarrow \begin{cases} \arg\max\{r_i\} & with\ prob.\ 1-\epsilon \\ random\ arm & with\ prob.\ \epsilon \end{cases}$$

updSels

$$n_{arm} \leftarrow n_{arm} + 1$$
$$n_{total} \leftarrow n_{total} + 1$$

updRew

$$r_{arm} \leftarrow \frac{(r_{arm} * (n_{arm} - 1) + r_{step})}{n_{arm}}$$

Random non-decaying exploration

Selections incremented

Step reward added to the running average

Upper Confidence Bound (UCB)

Upper Confidence Bound

nextArm

$$arm \leftarrow \arg\max\{r_i + c\sqrt{\frac{\ln(n_{total})}{n_i}}\}$$

updSels

$$n_{arm} \leftarrow n_{arm} + 1$$
$$n_{total} \leftarrow n_{total} + 1$$

updRew

$$r_{arm} \leftarrow \frac{(r_{arm} * (n_{arm} - 1) + r_{step})}{n_{arm}}$$

Selection and past reward aware decaying exploration

Selections incremented

Step reward added to the running average

- Very bad actions and nearly-optimal actions have different exploration chances

Discounted Upper Confidence Bound (DUCB)

Discounted Upper Confidence Bound

nextArm

$$arm \leftarrow \arg\max\{r_i + c\sqrt{\frac{\ln(n_{total})}{n_i}}\}$$

updSels

$$n_i \leftarrow \gamma \times n_i, \forall i \in [1, M]$$
$$n_{arm} \leftarrow n_{arm} + 1$$
$$n_{total} \leftarrow \gamma \times n_{total} + 1$$

updRew

$$r_{arm} \leftarrow \frac{(r_{arm} * (n_{arm} - 1) + r_{step})}{n_{arm}}$$

Selection and past reward aware decaying exploration

Selections discounted and incremented

Step reward added to the running average

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