Machine Learning Driven Memory and Storage Systems

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
19 December 2023
IBM Research
The Problem

Computing is Bottlenecked by Data
Data is Key for AI, ML, Genomics, …

- Important workloads are all data intensive

- They require rapid and efficient processing of large amounts of data

- Data is increasing
  - We can generate more than we can process
  - We need to perform more sophisticated analyses on more data
Data is Key for Modern Workloads

- **In-memory Databases**
  [Mao+, EuroSys’12; Clapp+ (Intel), IISWC’15]

- **Graph/Tree Processing**
  [Xu+, IISWC’12; Umuroglu+, FPL’15]

- **In-Memory Data Analytics**
  [Clapp+ (Intel), IISWC’15; Awan+, BDCloud’15]

- **Datacenter Workloads**
  [Kanev+ (Google), ISCA’15]
Huge Demand for Performance & Efficiency

Exponential Growth of Neural Networks

Memory and compute requirements

- **2018**
  - BERT Base (110M)
  - BERT Large (340M)

- **2019**
  - GPT-2 (1.5B)
  - T-NLG (17B)
  - Megatron-LM (8B)

- **2020+**
  - GPT-3 (175B)
  - MT-NLG (530B)
  - MSFT-1T (1T)

1800x more compute
In just 2 years

Tomorrow, **multi-trillion** parameter models

Source: https://youtu.be/Bh13IdwcBOQ?t=283
Data Overwhelms Modern Machines

- In-memory Databases
  - In-memory Databases
    - [Clapp+ (Intel), IISWC’15; Awan+, BDCloud’15]

- Graph/Tree Processing
  - Datacenter Workloads
    - [Kanev+ (Google), ISCA’15]

Data → performance & energy bottleneck

In-Memory Data Analytics

Datacenter Workloads

SAFARI
Data is Key for Modern Workloads

**Chrome**
Google’s web browser

**TensorFlow Mobile**
Google’s machine learning framework

**VP9**
YouTube
- Video Playback
  - Google’s video codec
- Video Capture
  - Google’s video codec
Data Overwhelms Modern Machines

Chrome

TensorFlow Mobile

Data → performance & energy bottleneck

VP9

Video Playback

Google’s video codec

VP9

Video Capture

Google’s video codec
Data Movement Overwhelms Modern Machines


62.7% of the total system energy is spent on data movement

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand\textsuperscript{1}  Saugata Ghose\textsuperscript{1}  Youngsok Kim\textsuperscript{2}  Rachata Ausavarungnirun\textsuperscript{1}  Eric Shiu\textsuperscript{3}  Rahul Thakur\textsuperscript{3}  Aki Kuusela\textsuperscript{3}  Allan Knies\textsuperscript{3}  Parthasarathy Ranganathan\textsuperscript{3}  Daehyun Kim\textsuperscript{4,3}  Onur Mutlu\textsuperscript{5,1}
Data Movement Overwhelms Accelerators

- Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu,

"Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks"

Proceedings of the 30th International Conference on Parallel Architectures and Compilation Techniques (PACT), Virtual, September 2021.
[Slides (pptx) (pdf)]
[Talk Video (14 minutes)]

> 90% of the total system energy is spent on memory in large ML models
An Intelligent Architecture
Handles Data Well
How to Handle Data Well

- **Ensure data does not overwhelm the components**
  - via intelligent algorithms, architectures & system designs: algorithm-architecture-devices

- **Take advantage of vast amounts of data and metadata**
  - to improve architectural & system-level decisions

- **Understand and exploit properties of (different) data**
  - to improve algorithms & architectures in various metrics
Corollaries: Computing Systems Today …

- Are processor-centric vs. data-centric
- Make designer-dictated decisions vs. data-driven
- Make component-based myopic decisions vs. data-aware
Fundamentally Better Architectures

Data-centric

Data-driven

Data-aware
We Need to Revisit the Entire Stack

We can get there step by step
A Blueprint for Fundamentally Better Architectures

- Onur Mutlu,
  "Intelligent Architectures for Intelligent Computing Systems"
  Invited Paper in Proceedings of the Design, Automation, and Test in
  Europe Conference (DATE), Virtual, February 2021.
  [Slides (pptx) (pdf)]
  [IEDM Tutorial Slides (pptx) (pdf)]
  [Short DATE Talk Video (11 minutes)]
  [Longer IEDM Tutorial Video (1 hr 51 minutes)]

Intelligent Architectures for Intelligent Computing Systems

Onur Mutlu
ETH Zurich
omutlu@gmail.com
Data-Driven (Self-Optimizing) Architectures
System Architecture Design Today

- Human-driven
  - Humans design the policies (how to do things)

- Many (too) simple, short-sighted policies all over the system

- No automatic data-driven policy learning

- (Almost) no learning: cannot take lessons from past actions

**Can we design fundamentally intelligent architectures?**
An Intelligent Architecture

- Data-driven
  - Machine learns the “best” policies (how to do things)

- Sophisticated, workload-driven, changing, far-sighted policies

- Automatic data-driven policy learning

- All controllers are intelligent data-driven agents

We need to rethink design (of all controllers)
Self-Optimizing Memory Controllers


Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek¹,² Onur Mutlu² José F. Martínez¹ Rich Caruana¹

¹Cornell University, Ithaca, NY 14850 USA
²Microsoft Research, Redmond, WA 98052 USA
Self-Optimizing Memory Prefetchers


[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (1.5 minutes)]
[Pythia Source Code (Officially Artifact Evaluated with All Badges)]
[arXiv version]

Officially artifact evaluated as available, reusable and reproducible.

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera¹ Konstantinos Kanellopoulos¹ Anant V. Nori² Taha Shahroodi³,¹
Sreenivas Subramoney² Onur Mutlu¹

¹ETH Zürich ²Processor Architecture Research Labs, Intel Labs ³TU Delft

Learning-Based Off-Chip Load Predictors

- Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, and Onur Mutlu,

"Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction"

Proceedings of the 55th International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, October 2022.

[Slides (pptx) (pdf)]
[Longer Lecture Slides (pptx) (pdf)]
[Talk Video (12 minutes)]
[Lecture Video (25 minutes)]
[arXiv version]
[Source Code (Officially Artifact Evaluated with All Badges)]

Officially artifact evaluated as available, reusable and reproducible. Best paper award at MICRO 2022.

Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera¹ Konstantinos Kanellopoulos¹ Shankar Balachandran² David Novo³
Ataberk Olgun¹ Mohammad Sadrosadati¹ Onur Mutlu¹

¹ETH Zürich ²Intel Processor Architecture Research Lab ³LIRMM, Univ. Montpellier, CNRS

Self-Optimizing Hybrid SSD Controllers

Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gomez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu, "Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning"
[Slides (pptx) (pdf)]
[arXiv version]
[Sibyl Source Code]
[Talk Video (16 minutes)]

Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh¹  Rakesh Nadig¹  Jisung Park¹  Rahul Bera¹  Nastaran Hajinazar¹  
David Novo³  Juan Gómez-Luna¹  Sander Stuijk²  Henk Corporaal²  Onur Mutlu¹  
¹ETH Zürich  ²Eindhoven University of Technology  ³LIRMM, Univ. Montpellier, CNRS

Self Optimizing Memory Controllers
DRAM Controllers Difficult to Design

- Need to obey **DRAM timing constraints** for correctness
  - There are many (50+) timing constraints in DRAM
  - tWTR: Minimum number of cycles to wait before issuing a read command after a write command is issued
  - tRC: Minimum number of cycles between the issuing of two consecutive activate commands to the same bank
  - ...
- Need to keep track of many resources to prevent conflicts
  - Channels, banks, ranks, data bus, address bus, row buffers
- Need to handle **DRAM refresh**
- Need to manage power consumption
- Need to optimize performance & QoS (in the presence of constraints)
  - Reordering is not simple
  - Fairness and QoS needs complicates the scheduling problem
Many DRAM Timing Constraints

<table>
<thead>
<tr>
<th>Latency</th>
<th>Symbol</th>
<th>DRAM cycles</th>
<th>Latency</th>
<th>Symbol</th>
<th>DRAM cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precharge</td>
<td>( t_{RP} )</td>
<td>11</td>
<td>Activate to read/write</td>
<td>( t_{RCD} )</td>
<td>11</td>
</tr>
<tr>
<td>Read column address strobe</td>
<td>( CL )</td>
<td>11</td>
<td>Write column address strobe</td>
<td>( CWL )</td>
<td>8</td>
</tr>
<tr>
<td>Additive</td>
<td>( AL )</td>
<td>0</td>
<td>Activate to activate</td>
<td>( RC )</td>
<td>39</td>
</tr>
<tr>
<td>Activate to precharge</td>
<td>( t_{RAS} )</td>
<td>28</td>
<td>Read to precharge</td>
<td>( t_{RTP} )</td>
<td>6</td>
</tr>
<tr>
<td>Burst length</td>
<td>( t_{BL} )</td>
<td>4</td>
<td>Column address strobe to column address strobe</td>
<td>( CCD )</td>
<td>4</td>
</tr>
<tr>
<td>Activate to activate (different bank)</td>
<td>( t_{RRD} )</td>
<td>6</td>
<td>Four activate windows</td>
<td>( FAW )</td>
<td>24</td>
</tr>
<tr>
<td>Write to read</td>
<td>( t_{WTR} )</td>
<td>6</td>
<td>Write recovery</td>
<td>( WR )</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 4. DDR3 1600 DRAM timing specifications

More on DRAM Operation


![Diagram of DRAM access phases]

**Figure 5. Three Phases of DRAM Access**

<table>
<thead>
<tr>
<th>Phase</th>
<th>Commands</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACT → READ</td>
<td>tRCD</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td>ACT → WRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ACT → PRE</td>
<td>tRAS</td>
<td>37.5ns</td>
</tr>
<tr>
<td></td>
<td>READ → data</td>
<td>tCL</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td>WRITE → data</td>
<td>tCWL</td>
<td>11.25ns</td>
</tr>
<tr>
<td></td>
<td><em>data burst</em></td>
<td>tBL</td>
<td>7.5ns</td>
</tr>
<tr>
<td>3</td>
<td>PRE → ACT</td>
<td>tRP</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td>1 &amp; 3 ACT → ACT</td>
<td>tRCD (tRAS+tRP)</td>
<td>52.5ns</td>
</tr>
</tbody>
</table>

Table 2. Timing Constraints (DDR3-1066) [43]
Why So Many Timing Constraints? (I)

Figure 4. DRAM bank operation: Steps involved in serving a memory request [17] \((V_{PP} > V_{DD})\)

<table>
<thead>
<tr>
<th>Category</th>
<th>RowCmd↔RowCmd</th>
<th>RowCmd↔ColCmd</th>
<th>ColCmd↔ColCmd</th>
<th>ColCmd→DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name Commands</td>
<td>tRC</td>
<td>tRAS</td>
<td>tRP</td>
<td>tCCD</td>
</tr>
<tr>
<td>Scope</td>
<td>A→A</td>
<td>A→P</td>
<td>P→A</td>
<td>R(W)→R(W)</td>
</tr>
<tr>
<td>Value (ns)</td>
<td>~50</td>
<td>~35</td>
<td>13-15</td>
<td>5-7.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

A: ACTIVATE  P: PRECHARGE  R: READ  W: WRITE
* Goes into effect after the last write data, not from the WRITE command
† Not explicitly specified by the JEDEC DDR3 standard [18]. Defined as a function of other timing constraints.

Table 1. Summary of DDR3-SDRAM timing constraints (derived from Micron’s 2Gb DDR3-SDRAM datasheet [33])

Why So Many Timing Constraints? (II)

Figure 6. Charge Flow Between the Cell Capacitor ($C_C$), Bitline Parasitic Capacitor ($C_B$), and the Sense-Amplifier ($C_B \approx 3.5C_C$ [39])

DRAM Controller Design Is Becoming More Difficult

- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs
- Many timing constraints for various memory types
- Many goals at the same time: performance, fairness, QoS, energy efficiency, ...
Reality and Dream

- Reality: It is difficult to design a policy that maximizes performance, QoS, energy-efficiency, ...
  - Too many things to think about
  - Continuously changing workload and system behavior

- Dream: Wouldn’t it be nice if the DRAM controller automatically found a good scheduling policy on its own?
Memory Controller: Performance Function

Resolves memory contention by scheduling requests

How to schedule requests to maximize system performance?
Self-Optimizing DRAM Controllers

- **Problem:** DRAM controllers are difficult to design
  - It is difficult for human designers to design a policy that can adapt itself very well to different workloads and different system conditions.

- **Idea:** A memory controller that adapts its scheduling policy to workload behavior and system conditions using machine learning.

- **Observation:** Reinforcement learning maps nicely to memory control.

- **Design:** Memory controller is a reinforcement learning agent
  - It dynamically and continuously learns and employs the best scheduling policy to maximize long-term performance.
Self-Optimizing DRAM Controllers

Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots (0 \leq \gamma < 1)$

Figure 2: (a) Intelligent agent based on reinforcement learning principles;
Self-Optimizing DRAM Controllers

- Dynamically adapt the memory scheduling policy via interaction with the system at runtime
  - Associate system states and actions (commands) with long term reward values: each action at a given state leads to a learned reward
  - Schedule command with highest estimated long-term reward value in each state
  - Continuously update reward values for \(<\text{state}, \text{action}>\) pairs based on feedback from system
Self-Optimizing DRAM Controllers


![Diagram of a high-level overview of an RL-based scheduler.](image)

**Figure 4:** High-level overview of an RL-based scheduler.
States, Actions, Rewards

❖ Reward function
  • +1 for scheduling Read and Write commands
  • 0 at all other times

Goal is to maximize long-term data bus utilization

❖ State attributes
  • Number of reads, writes, and load misses in transaction queue
  • Number of pending writes and ROB heads waiting for referenced row
  • Request’s relative ROB order

❖ Actions
  • Activate
  • Write
  • Read - load miss
  • Read - store miss
  • Precharge - pending
  • Precharge - preemptive
  • NOP
Performance Results

Large, robust performance improvements over many human-designed policies

Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers.

Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth.
Self Optimizing DRAM Controllers

+ **Continuous learning** in the presence of changing environment

+ **Reduced designer burden** in finding a good scheduling policy. Designer specifies:
  
  1) What system variables might be useful
  2) What target to optimize, but not how to optimize it

  -- How to specify different objectives? (e.g., fairness, QoS, ...)

  -- Hardware complexity?

  -- Design **mindset** and flow
Self-Optimizing Memory Controllers

Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana,
"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"

Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek¹,²  Onur Mutlu²  José F. Martínez¹  Rich Caruana¹

¹Cornell University, Ithaca, NY 14850 USA
² Microsoft Research, Redmond, WA 98052 USA
Pythia: Prefetching using Reinforcement Learning
Self-Optimizing Memory Prefetchers

Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu,
"Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning"
Proceedings of the 54th International Symposium on Microarchitecture (MICRO), Virtual, October 2021.

[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (1.5 minutes)]
[Pythia Source Code (Officially Artifact Evaluated with All Badges)]
[arXiv version]

Officially artifact evaluated as available, reusable and reproducible.

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera¹  Konstantinos Kanellopoulos¹  Anant V. Nori²  Taha Shahroodi³,¹
Sreenivas Subramoney²  Onur Mutlu¹

¹ETH Zürich  ²Processor Architecture Research Labs, Intel Labs  ³TU Delft

Pythia
A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori, Taha Shahroodi, Sreenivas Subramoney, Onur Mutlu

https://github.com/CMU-SAFARI/Pythia

Executive Summary

**Background**: Prefetchers predict addresses of future memory requests by associating memory access patterns with pieces of program and system information (called feature).

**Problem**: Three key shortcomings of prior prefetchers:
- Predict mainly using a single program feature
- Lack inherent system awareness (e.g., memory bandwidth usage)
- Lack in-silicon customizability

**Goal**: Design a prefetching framework that:
- Learns from multiple features and inherent system-level feedback
- Can be customized in silicon to use different features and/or prefetching objectives

**Contribution**: Pythia, which formulates prefetching as reinforcement learning problem
- Takes adaptive prefetch decisions using multiple features and system-level feedback
- Can be customized in silicon for target workloads via simple configuration registers
- Proposes a realistic and practical implementation of RL algorithm in hardware

**Key Results**:
- Evaluated using a wide range of workloads from SPEC CPU, PARSEC, Ligra, Cloudsuite
- Outperforms best prefetcher by 3.4%, 7.7% and 17% in 1/4/bw-constrained cores
- Customizing Pythia leads to up to 7.8% more performance over basic Pythia across Ligra workloads

**SAFARI**

[https://github.com/CMU-SAFARI/Pythia](https://github.com/CMU-SAFARI/Pythia)
Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

Pythia: Overview

Evaluation of Pythia and Key Results

Conclusion
Prefetching Basics

- 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 or system information.

  **Program Feature** → Access Pattern

- Example program features:
  - Program counter (PC)
  - Page number
  - Page offset
  - Cacheline delta
  - ...
  - Or a combination of these attributes.
Key Shortcomings in Prior Prefetchers

- We observe three key shortcomings that significantly limit performance benefits of prior prefetchers.

1. Predict mainly using a single program feature.

2. Lack inherent system awareness.

3. Lack in-silicon customizability.
(1) Single-Feature Prefetch Prediction

• Provides **good** performance **gains** mainly on workloads where the **feature-to-pattern correlation exists**

![Graph showing IPC improvement over baseline (%) for different benchmarks and SPP, Bingo, Pythia.]

Bingo [1] performs better

SPP [2] performs better

[1] Bakshalipour et al., HPCA’19

[2] Kim et al., MICRO’16
(1) Single-Feature Prefetch Prediction

- Provides **good** performance **gains** mainly on workloads where the **feature-to-pattern correlation** exists.

Relying on a **single feature** for prediction leaves significant performance improvement on table.

IPC improvement over baseline (%)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>482.sphinx3-417B</td>
<td>15.4%</td>
<td>3.5%</td>
<td>Bingo [1] performs better</td>
<td>SPP [2] performs better</td>
</tr>
<tr>
<td>PARSEC-Canneal</td>
<td>5.5%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARSEC-Facesim</td>
<td>4.6%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>459.GemsFDTD-765B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(2) Lack of Inherent System Awareness

- Little understanding of **undesirable effects** (e.g., memory bandwidth usage, cache pollution, ...)
  - Performance loss in **resource-constrained** configurations

**Bar Graph:**
- Fraction of LLC misses for different benchmarks:
  - SPP, Bingo, Pythia, Ligra-CC, PARSEC-Canneal
  - Coverage, Uncovered, Overpredicted

**Percentage Improvement:**
- IPC improvement over baseline (%)
  - SPP, Bingo, Pythia
  - Ligra-CC, PARSEC-Canneal

- **Similar coverage**
- **Lower overpredictions**
- **Yet, lower performance**
(2) Lack of Inherent System Awareness

- Little understanding of **undesirable effects** (e.g., memory bandwidth usage, cache pollution, ...)
- Performance loss in **resource-constrained** configurations

Prefetchers often **lose performance** due to lack of **inherent system awareness**
(3) Lack of In-silicon Customizability

• Feature **statically** selected at design time
  - **Rigid hardware** designed specifically to exploit that feature

• **No way to change** program feature and/or change prefetcher’s objective **in silicon**
  - **Cannot adapt** to a wide range of workload demands
Our Goal

A prefetching framework that can:

1. Learn to prefetch using multiple features and inherent system-level feedback information

2. Be easily customized in silicon to use different features and/or change prefetcher’s objectives
Our Proposal

Pythia

Formulates prefetching as a reinforcement learning problem
Talk Outline

- Key Shortcomings of Prior Prefetchers
- Formulating Prefetching as Reinforcement Learning
- Pythia: Overview
- Evaluation of Pythia and Key Results
- Conclusion
Basics of Reinforcement Learning (RL)

• Algorithmic approach to learn to take an action in a given situation to maximize a numerical reward

Agent

Environment

• Agent stores Q-values for every state-action pair
  - Expected return for taking an action in a state
  - Given a state, selects action that provides highest Q-value
Formulating Prefetching as RL

Agent

Environment

State ($S_t$)

Action ($A_t$)

Reward ($R_t + 1$)

Prefetcher

Processor & Memory Subsystem

Reward

Prefetch from address $A + o$ff set (O)

Features of memory request to address $A$ (e.g., PC)
What is State?

• **$k$-dimensional** vector of features
  \[ S \equiv \{\phi_S^1, \phi_S^2, \ldots, \phi_S^k\} \]

• Feature = control-flow + data-flow

• **Control-flow examples**
  - PC
  - Branch PC
  - Last-3 PCs, ...

• **Data-flow examples**
  - Cacheline address
  - Physical page number
  - Delta between two cacheline addresses
  - Last 4 deltas, ...
What is State?

Example of a state information

\[ S = \{ PC + \text{Delta}, \text{Sequence of last-4 deltas} \} \]

- **Feature-1** \( (\phi_1) \)
  - **PC** (Control-flow info.)
  - **Cacheline Delta** (Data-flow info.)

- **Feature-2** \( (\phi_2) \)
  - **Seq. of last-4 deltas** (Data-flow info.)
What is Action?

Given a demand access to address A the action is to **select prefetch offset “0”**

- **Action-space**: 127 actions in the range [-63, +63]
  - For a machine with 4KB page and 64B cacheline

- Upper and lower limits ensure prefetches do not cross physical page boundary

- A **zero offset** means no prefetch is generated

- We further **prune** action-space by design-space exploration
What is Reward?

• Defines the **objective** of Pythia

• Encapsulates two metrics:
  - **Prefetch usefulness** (e.g., accurate, late, out-of-page, ...)
  - **System-level feedback** (e.g., mem. b/w usage, cache pollution, energy, ...)

• We demonstrate Pythia with **memory bandwidth usage** as the system-level feedback in the paper
What is Reward?

- **Seven** distinct reward levels
  - **Accurate and timely** \((R_{AT})\)
  - **Accurate but late** \((R_{AL})\)
  - **Loss of coverage** \((R_{CL})\)
  - **Inaccurate**
    - With low memory b/w usage \((R_{IN-L})\)
    - With high memory b/w usage \((R_{IN-H})\)
  - **No-prefetch**
    - With low memory b/w usage \((R_{NP-L})\)
    - With high memory b/w usage \((R_{NP-H})\)

- Values are set at design time via **automatic design-space exploration**
  - Can be **customized** further in silicon for higher performance
Steering Pythia’s Objective via Reward Values

- Example reward configuration for
  - Generating **accurate prefetches**
  - Making **bandwidth-aware** prefetch decisions

1. **Highly prefers to generate accurate prefetches**
2. **Prefers not to prefetch if memory bandwidth usage is low**
3. **Strongly prefers not to prefetch if memory bandwidth usage is high**

\[
\begin{align*}
R_{IN-H} & = -14 \\
R_{IN-L} & = -8 \\
R_{NP-L} & = -4 \\
R_{NP-H} & = -2 \\
R_{AL} & = +12 \\
R_{AT} & = +20
\end{align*}
\]

AT = Accurate & timely; AL = Accurate & late; NP = No-prefetching; IN = Inaccurate;
H = High mem. b/w; L = Low mem. b/w
Steering Pythia’s Objective via Reward Values

• Customizing reward values to make Pythia **conservative** towards prefetching

AT = Accurate & timely; AL = Accurate & late; NP = No-prefetching; IN = Inaccurate; 
H = High mem. b/w; L = Low mem. b/w

1. **Highly prefers to generate accurate prefetches**

2. **Otherwise prefers not to prefetch**
Steering Pythia’s Objective via Reward Values

- Customizing reward values to make Pythia conservative towards prefetching

**Strict Pythia configuration**

- Accuracy and timeliness: **AT**
- Accuracy and late: **AL**
- No prefetching: **NP**
- Inaccurate: **IN**
- High memory bandwidth: **H**
- Low memory bandwidth: **L**

Highly prefers to generate accurate prefetches

Otherwise prefers not to prefetch

Server-class processors

Bandwidth-sensitive workloads
Pythia Overview

- **Q-Value Store**: Records Q-values for all state-action pairs
- **Evaluation Queue**: A FIFO queue of recently-taken actions

![Diagram](image_url)
Architecting QVStore

S = \{PC+Delta, \text{Sequence of last-4 deltas}\}
Architecting the QVStore

- Fast prefetch prediction
- Fast retrieval of Q-values from QVStore
- Efficient storage organization of Q-values in QVStore

$S = \{PC+\Delta, \text{Sequence of last-4 deltas}\}$
Organization of QVStore

- A **monolithic** two-dimensional table?
  - Indexed by state and action values
- State-space increases **exponentially** with #bits

\[ S = \{PC+\text{Delta}, \text{Sequence of last-4 deltas}\} \]

\[ 32b + 7b + 4\times7b = 67 \text{ bits} \]

- **SAFARI**
  - 2^{67} states
  - 127 actions
  - Design complexity
  - Access latency
Organization of QVStore

- We partition QVStore into \( k \) vaults \([k = \text{number of features in state}]\)
  - Each vault corresponds to one feature and stores the Q-values of feature-action pairs

To retrieve \( Q(S,A) \) for each action

- Query each vault in parallel with feature and action
- Retrieve feature-action Q-value from each vault
- Compute \( \text{MAX} \) of all feature-action Q-values

\( \text{MAX} \) ensures the \( Q(S,A) \) is driven by the constituent feature that has highest \( Q(\phi,A) \)
Organization of QVStore

• We further partition each vault into multiple **planes**
  - Each plane stores a **partial** Q-value of a feature-action pair

**To retrieve Q(ϕ,A) for each action**

• Query each plane in **parallel** with hashed feature and action
• Retrieve partial feature-action Q-value from each plane
• Compute **SUM** of all partial feature-action Q-values
Organization of QVStore

- We further partition each vault into multiple planes.
  - Each plane stores a partial Q-value of a feature-action pair.

1. **Enables sharing** of partial Q-values between similar feature values, shortens prefetcher training time.

2. **Reduces chances** of sharing partial Q-values across widely different feature values.

- Query each plane in parallel with hashed feature and action.
- Compute SUM of all partial feature-action Q-values.
More in the Paper

• **Pipelined search** operation for QVStore

• Reward assignment and **QVStore update**

• **Automatic design-space exploration**
  - Feature types
  - Actions
  - Reward and Hyperparameter values
More in the Paper

• Pipelined search operation for QVStore

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera\textsuperscript{1} \quad Konstantinos Kanellopoulos\textsuperscript{1} \quad Anant V. Nori\textsuperscript{2} \quad Taha Shahroodi\textsuperscript{3,1}
Sreenivas Subramoney\textsuperscript{2} \quad Onur Mutlu\textsuperscript{1}
\textsuperscript{1}ETH Zürich \quad \textsuperscript{2}Processor Architecture Research Labs, Intel Labs \quad \textsuperscript{3}TU Delft

- Reward and Hyperparameter values

Simulation Methodology

• **Champsim** [3] trace-driven simulator

• **150** single-core memory-intensive workload traces
  - SPEC CPU2006 and CPU2017
  - PARSEC 2.1
  - Ligra
  - Cloudsuite

• Homogeneous and heterogeneous multi-core mixes

• **Five** state-of-the-art prefetchers
  - SPP [Kim+, MICRO’16]
  - Bingo [Bakhshalipour+, HPCA’19]
  - MLOP [Shakerinava+, 3rd Prefetching Championship, 2019]
  - SPP+DSPatch [Bera+, MICRO’19]
  - SPP+PPF [Bhatia+, ISCA’20]

Basic Pythia Configuration

• Derived from **automatic design-space exploration**

**State:** 2 features
- PC+Delta
- Sequence of last-4 deltas

**Actions:** 16 prefetch offsets
- Ranging between -6 to +32. Including 0.

**Rewards:**
- $R_{AT} = +20$; $R_{AL} = +12$; $R_{NP-H} = -2$; $R_{NP-L} = -4$
- $R_{IN-H} = -14$; $R_{IN-L} = -8$; $R_{CL} = -12$

SAFARI
### Table 3: List of program control-flow and data-flow components used to derive the list of features for exploration

<table>
<thead>
<tr>
<th>Control-flow Component</th>
<th>Data-flow Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) PC of load request</td>
<td>(1) Load cacheline address</td>
</tr>
<tr>
<td>(2) PC-path (XOR-ed last-3 PCs)</td>
<td>(2) Page number</td>
</tr>
<tr>
<td>(3) PC XOR-ed branch-PC</td>
<td>(3) Page offset</td>
</tr>
<tr>
<td>(4) None</td>
<td>(4) Load address delta</td>
</tr>
<tr>
<td></td>
<td>(5) Sequence of last-4 offsets</td>
</tr>
<tr>
<td></td>
<td>(6) Sequence of last-4 deltas</td>
</tr>
<tr>
<td></td>
<td>(7) Offset XOR-ed with delta</td>
</tr>
<tr>
<td></td>
<td>(8) None</td>
</tr>
</tbody>
</table>
## Basic Pythia Configuration

Table 2: Basic Pythia configuration derived from our automated design-space exploration

<table>
<thead>
<tr>
<th>Features</th>
<th>PC+Delta, Sequence of last-4 deltas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefetch Action List</td>
<td>{-6,-3,-1,0,1,3,4,5,10,11,12,16,22,23,30,32}</td>
</tr>
<tr>
<td>Reward Level Values</td>
<td>( R_{AT}=20 ), ( R_{AL}=12 ), ( R_{CL}=-12 ), ( R_{IN}^{H}=-14 ), ( R_{IN}^{L}=-8 ), ( R_{NP}^{H}=-2 ), ( R_{NP}^{L}=-4 )</td>
</tr>
<tr>
<td>Hyperparameters</td>
<td>( \alpha = 0.0065 ), ( \gamma = 0.556 ), ( \epsilon = 0.002 )</td>
</tr>
</tbody>
</table>
Performance with Varying Core Count

Geomean speedup over no prefetching vs. Number of cores

- Bingo
- MLOP
- SPP
- Pythia

3.4% speedup at 2 cores for Bingo
7.7% speedup at 12 cores for Pythia
1. Pythia consistently provides the highest performance in **all core configurations**

2. Pythia’s gain **increases with core count**
Performance with Varying DRAM Bandwidth

Geomean speedup over no prefetching

DRAM MTPS (in log scale)

- ~Intel Xeon 6258R
- ~AMD EPYC Rome 7702P
- ~AMD Threadripper 3990x

Baseline

Bingo
MLOP
SPP

Pythia

3%
17%
3%
Pythia outperforms prior best prefetchers for a wide range of DRAM bandwidth configurations
Performance Improvement via Customization

• Reward value customization

• **Strict Pythia configuration**
  - **Increase** the rewards for **no prefetching**
  - **Decrease** the rewards for **inaccurate prefetching**

  ![Diagram showing reward values for different states](image)

• **Strict Pythia** is **more conservative** in generating prefetch requests than the basic Pythia

• Evaluate on all **Ligra graph processing workloads**
Performance Improvement via Customization

IPC normalized to no prefetching

- Basic Pythia
- Strict Pythia

3.1%  2.8%  3.4%  7.8%  5.2%  2%

PageRank  PageRankDelta  CC  BFS  BC  GEOMEAN
Performance Improvement via Customization

Pythia can extract even higher performance via customization **without changing hardware**
Pythia’s Overhead

• **25.5 KB** of total metadata storage **per core**
  - Only simple tables

• We also model functionally-accurate Pythia with full complexity in **Chisel** [4] HDL

- **1.03% area overhead**
- **0.4% power overhead**
- Satisfies prediction latency of a desktop-class 4-core Skylake processor (Xeon D2132IT, 60W)

More in the Paper

• Performance comparison with unseen traces
  - Pythia provides equally high performance benefits

• Comparison against multi-level prefetchers
  - Pythia outperforms prior best multi-level prefetchers

• Understanding Pythia’s learning with a case study
  - We reason towards the correctness of Pythia’s decision

• Performance sensitivity towards different features and hyperparameter values

• Detailed single-core and four-core performance
Performance on Previously-Unseen Workloads

- Evaluated with 500 traces from value prediction championship
  - No prefetcher has been trained on these traces

Pythia outperforms MLOP and Bingo by
  - 8.3% and 3.5% in single-core
  - 9.7% and 5.4% in four-core
More in the Paper

• Performance comparison with unseen traces
  - Pythia provides equally high performance benefits

• Comparison against multi-level prefetchers
  - Pythia outperforms prior best multi-level prefetchers

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera¹  Konstantinos Kanellopoulos¹  Anant V. Nori²  Taha Shahroodi³,¹
Sreenivas Subramoney²  Onur Mutlu¹

¹ETH Zürich  ²Processor Architecture Research Labs, Intel Labs  ³TU Delft

• Performance sensitivity towards different features and hyperparameter values


• Detailed single-core and four-core performance
Pythia is Open Source

https://github.com/CMU-SAFARI/Pythia

- MICRO’21 artifact evaluated
- Champsim source code + Chisel modeling code
- All traces used for evaluation
Talk Outline

- Key Shortcomings of Prior Prefetchers
- Formulating Prefetching as Reinforcement Learning
- Pythia: Overview
- Evaluation of Pythia and Key Results
- Conclusion
Executive Summary

- **Background**: Prefetchers predict addresses of future memory requests by associating memory access patterns with pieces of program and system information (called feature).

- **Problem**: Three key shortcomings of prior prefetchers:
  - Predict mainly using a single program feature
  - Lack inherent system awareness (e.g., memory bandwidth usage)
  - Lack in-silicon customizability

- **Goal**: Design a prefetching framework that:
  - Learns from multiple features and inherent system-level feedback
  - Can be customized in silicon to use different features and/or prefetching objectives

- **Contribution**: Pythia, which formulates prefetching as reinforcement learning problem
  - Takes adaptive prefetch decisions using multiple features and system-level feedback
  - Can be customized in silicon for target workloads via simple configuration registers
  - Proposes a realistic and practical implementation of RL algorithm in hardware

- **Key Results**:
  - Evaluated using a wide range of workloads from SPEC CPU, PARSEC, Ligra, Cloudsuite
  - Outperforms best prefetcher by 3.4%, 7.7% and 17% in 1/4/bw-constrained cores
  - Customizing Pythia leads to up to 7.8% more performance over basic Pythia across Ligra workloads

SAFARI

https://github.com/CMU-SAFARI/Pythia
Pythia
A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori, Taha Shahroodi, Sreenivas Subramoney, Onur Mutlu

https://github.com/CMU-SAFARI/Pythia

Pythia Discussion

• FAQs
  - Why RL?
  - What about large page?
  - What’s the prefetch degree?
  - Can customization happen during workload execution?
  - Can runtime mixing create problem?

• Simulation and Methodology
  - Basic Pythia configuration
  - System parameters
  - Configuration of prefetchers
  - Evaluated workloads
  - Feature selection

• Detailed Design
  - Reward structure
  - Design overview
  - QVStore Organization

• More Results
  - Comparison against other adaptive prefetchers
  - Comparison against Context prefetcher
  - Feature combination sensitivity
  - Hyperparameter sensitivity
  - Comparison with multi-level prefetchers
  - Performance in unseen workloads
  - Single-core s-curve
  - Four-core s-curve
  - Detailed performance analysis
  - Benefit of bandwidth awareness
  - Case study
  - Customizing rewards
  - Customizing features
Self-Optimizing Memory Prefetchers

Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu,
"Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning"
Proceedings of the 54th International Symposium on Microarchitecture (MICRO), Virtual, October 2021.
[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (1.5 minutes)]
[Pythia Source Code (Officially Artifact Evaluated with All Badges)]
[arXiv version]
Officially artifact evaluated as available, reusable and reproducible.

Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning
Rahul Bera¹ Konstantinos Kanellopoulos¹ Anant V. Nori² Taha Shahroodi³,¹
Sreenivas Subramoney² Onur Mutlu¹
¹ETH Zürich ²Processor Architecture Research Labs, Intel Labs ³TU Delft

Hermes: Perceptron-Based Off-Chip Load Prediction
Learning-Based Off-Chip Load Predictors

- Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, and Onur Mutlu,

"Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction"

Proceedings of the 55th International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, October 2022.

[Slides (pptx) (pdf)]
[Longer Lecture Slides (pptx) (pdf)]
[Talk Video (12 minutes)]
[Lecture Video (25 minutes)]
[arXiv version]
[Source Code (Officially Artifact Evaluated with All Badges)]

Officially artifact evaluated as available, reusable and reproducible. Best paper award at MICRO 2022.

Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera\textsuperscript{1} Konstantinos Kanellopoulos\textsuperscript{1} Shankar Balachandran\textsuperscript{2} David Novo\textsuperscript{3}
Ataberk Olgun\textsuperscript{1} Mohammad Sadrosadati\textsuperscript{1} Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich \textsuperscript{2}Intel Processor Architecture Research Lab \textsuperscript{3}LIRMM, Univ. Montpellier, CNRS

\url{https://arxiv.org/pdf/2209.00188.pdf}
Hermes Talk Video

Hermes Overview

1. **Predict**
   - Core
   - L1-D
   - L2
   - POPET

2. **Issue a Hermes request**
   - LLC
   - MC

3. **Wait**
   - Off-Chip Memory
   - Main Memory

**Perceptron-based off-chip load predictor**

**Latency tolerance limit of ROB**

**Processor is stalled**

**Baseline**

**Hermes**

**Saved stall cycles**

**Main Memory**

Computer Architecture - Lecture 18: Cutting-Edge Research in Computer Architecture (Fall 2022)

https://www.youtube.com/watch?v=PWWBtrL60dQ&t=3609s
Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, Onur Mutlu

https://github.com/CMU-SAFARI/Hermes

The Key Problem

Long-latency **off-chip** load requests

Often **stall** processor by **blocking instruction retirement** from Reorder Buffer (ROB)

Limit performance
Traditional Solutions

1. Employ sophisticated *prefetchers*

2. Increase size of on-chip caches
Key Observation 1

Many loads still go off-chip

50% successfully prefetched

50% still go off-chip even with a state-of-the-art prefetcher

70% of the off-chip loads block the ROB

# off-chip loads without any prefetcher
Key Observation 2

On-chip cache access latency significantly contributes to off-chip load latency.

L1 | L2 | LLC | Main Memory

40% of the stalls can be eliminated by removing on-chip cache access latency from critical path.
Caches are Getting Bigger and Slower...

![Graph showing L2 Size (KB) vs. Year]

- Skylake (2015)
- Sunny Cove (2019)
- Willow Cove (2020)
- Golden Cove P-core (2021)
- Raptor Lake P-core (2022)

![Graph showing L2 Latency (processor cycles) vs. Year]

- Skylake (2015)
- Sunny Cove (2019)
- Willow Cove (2020)
- Golden Cove P-core (2021)
- Raptor Lake P-core (2022)
Our Goal

Improve processor performance by removing on-chip cache access latency from the critical path of off-chip loads.
HERMES

Predicts which load requests are likely to go off-chip

Starts fetching data directly from main memory while concurrently accessing the cache hierarchy
Key Contribution

Hermes employs the first perceptron-based off-chip load predictor

That predicts which loads are likely to go off-chip

By learning from multiple program context information
Hermes Overview

Core

L1-D

L2

LLC

MC

Off-Chip
Main Memory

Baseline

Latency tolerance limit of ROB

Processor is stalled

L1  L2  LLC

Main Memory
Hermes Overview

1. Predict
   - Core ➔ POPET

2. Issue a Hermes request
   - L1-D ➔ L2 ➔ LLC

3. Wait
   - MC ➔ Off-Chip Main Memory

4. Train
   - POPET ➔ Main Memory

Perceptron-based off-chip load predictor

Latency tolerance limit of ROB

Saved stall cycles

Processor is stalled

Baseline
- L1 ➔ L2 ➔ LLC ➔ Main Memory

Hermes
- L1 ➔ L2 ➔ LLC ➔ Main Memory

SAFARI
Designing the Off-Chip Load Predictor

**History-based prediction**

- HMP [Yoaz+, ISCA'99] for the L1-D cache
- Using branch-predictor-like hybrid predictor: Global, Gshare, and GSkew

### POPET provides
- both **higher accuracy** and **higher performance**
  - than predictors inspired from these previous works

- Metadata size increases with cache hierarchy size
- May need to track **all** cache operations
  - Gets complex depending on the cache hierarchy configuration (e.g., inclusivity, bypassing,...)

**Learning from program behavior**

- Correlate different program features with off-chip loads
- Low storage overhead
- Low design complexity
**POPET**: Perceptron-Based Off-Chip Predictor

- **Multi-feature** hashed perceptron model [1]
  - Each feature has its own *weight table*
  - Stores correlation between feature value and off-chip prediction

---

Predicting using POPET

- Uses simple **table lookups, addition, and comparison**

Extract features from the load request:

- Feature_1: \(0x7ffe0+12\)
- Feature_2
- ... 
- Feature_N

<table>
<thead>
<tr>
<th>Feature</th>
<th>Weight Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature_1</td>
<td>Weight Table_1</td>
</tr>
<tr>
<td>Feature_2</td>
<td>Weight Table_2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Feature_N</td>
<td>Weight Table_N</td>
</tr>
</tbody>
</table>

**Stage 1:**

**Stage 2:**

**Stage 3:**

\( \geq \tau_{act} \geq \tau_{act} \)

Predict that the load would go off-chip.

Core L1 - D L2 LLC Off-Chip Main Memory

**SAFARI**
Training POPET

- Uses simple **increment or decrement** of feature weights

<table>
<thead>
<tr>
<th>Feature</th>
<th>Weight Table</th>
<th>Index</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature_1</td>
<td>Table_1</td>
<td>42</td>
<td>-4</td>
</tr>
<tr>
<td>Feature_2</td>
<td>Table_2</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>Feature_N</td>
<td>Table_N</td>
<td>-5</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Activation Sum**

$$\sum \text{weights} \geq \tau_{act}$$

**Predict to go off-chip**

$$3 \geq -2$$

**Shouldn’t be activated**

$$\text{Cumulative weight} < \tau_{act}$$
### Features Used in Hermes

Table 1: The initial set of program features used for automated feature selection. \( \oplus \) represents a bitwise XOR operation.

<table>
<thead>
<tr>
<th>Features without control-flow information</th>
<th>Features with control-flow information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Load virtual address</td>
<td>8. Load PC</td>
</tr>
<tr>
<td>2. Virtual page number</td>
<td>9. PC ( \oplus ) load virtual address</td>
</tr>
<tr>
<td>3. Cacheline offset in page</td>
<td>10. PC ( \oplus ) virtual page number</td>
</tr>
<tr>
<td>4. First access</td>
<td>11. PC ( \oplus ) cacheline offset</td>
</tr>
<tr>
<td>5. Cacheline offset + first access</td>
<td>12. PC + first access</td>
</tr>
<tr>
<td>6. Byte offset in cacheline</td>
<td>13. PC ( \oplus ) byte offset</td>
</tr>
<tr>
<td>7. Word offset in cacheline</td>
<td>14. PC ( \oplus ) word offset</td>
</tr>
<tr>
<td></td>
<td>15. Last-4 load PCs</td>
</tr>
<tr>
<td></td>
<td>16. Last-4 PCs</td>
</tr>
</tbody>
</table>

Table 2: POPET configuration parameters

<table>
<thead>
<tr>
<th>Selected features</th>
</tr>
</thead>
<tbody>
<tr>
<td>• PC ( \oplus ) cacheline offset</td>
</tr>
<tr>
<td>• PC ( \oplus ) byte offset</td>
</tr>
<tr>
<td>• PC + first access</td>
</tr>
<tr>
<td>• Cacheline offset + first access</td>
</tr>
<tr>
<td>• Last-4 load PCs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Threshold values</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_{act} = -18, \ T_N = -35, \ T_P = 40 )</td>
</tr>
</tbody>
</table>
Evaluation
Simulation Methodology

- **ChampSim** trace driven simulator
- **110 single-core** memory-intensive traces
  - SPEC CPU 2006 and 2017
  - PARSEC 2.1
  - Ligra
  - Real-world applications
- **220 eight-core** memory-intensive trace mixes

### LLC Prefetchers
- Pythia [Bera+, MICRO’21]
- Bingo [Bakshalipour+, HPCA’19]
- MLOP [Shakerinava+, 3rd Prefetching Championship’19]
- SPP + Perceptron filter [Bhatia+, ISCA’20]
- SMS [Somogyi+, ISCA’06]

### Off-Chip Predictors
- **History-based**: HMP [Yoaz+, ISCA’99]
- **Tracking-based**: Address Tag-Tracking based Predictor (TTP)
- **Ideal Off-chip Predictor**
Latency Configuration

- **Cache round-trip latency**
  - L1-D: 5 cycles
  - L2: 15 cycles
  - LLC: 55 cycles

- **Hermes request issue latency**
  (incurred after address translation)

  Depends on
  - Interconnect between POPET and MC

<table>
<thead>
<tr>
<th>Latency (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off-Chip Main Memory</td>
</tr>
<tr>
<td>Core</td>
</tr>
<tr>
<td>L1-D</td>
</tr>
<tr>
<td>Issue Hermes request</td>
</tr>
<tr>
<td>LLC</td>
</tr>
<tr>
<td>Wait</td>
</tr>
<tr>
<td>POPET</td>
</tr>
<tr>
<td>Interconnect</td>
</tr>
<tr>
<td>MC</td>
</tr>
<tr>
<td>Main Memory</td>
</tr>
</tbody>
</table>
Hermes alone provides nearly **90%** performance benefit of **Ideal Hermes** that has an ideal off-chip load predictor with only 2/3rd storage overhead.
For **every 1% performance** benefit, increase in main memory requests

<table>
<thead>
<tr>
<th></th>
<th>Pythia</th>
<th>2%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermes on top of Pythia</td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td>Hermes alone</td>
<td>0.5%</td>
<td></td>
</tr>
</tbody>
</table>

Hermes is more **bandwidth-efficient** than even an efficient prefetcher like Pythia.
Performance with Varying Memory Bandwidth

In bandwidth-constrained configurations, Hermes alone outperforms Pythia. Hermes+Pythia outperforms Pythia across all bandwidth configurations.
Hermes consistently improves performance on top of a wide range of baseline prefetchers.
Hermes can provide **even higher performance** benefit in **future processors** with bigger and slower on-chip caches.
Effect of ROB Size

![Graph showing the effect of ROB size on performance. The x-axis represents different ROB sizes (256, 512, 768, 1024), and the y-axis represents the geometric mean speedup over the No-prefetching system. The bars are color-coded to represent Hermes, Pythia, and Pythia+Hermes. There is a 6.7% increase in performance at the 256 ROB size compared to the No-prefetching system, and a 5.3% increase at the 1024 ROB size.](image-url)
Effect of LLC Size

The chart illustrates the geomean speedup over the No-prefetching system for different LLC sizes per core. The labels are:

- Hermes
- Pythia
- Hermes + Pythia

The speedup values are indicated for LLC sizes of 3MB, 6MB, 12MB, and 24MB.

- For an LLC size of 3MB, the speedup is 1.3%.
- For an LLC size of 6MB, the speedup is 2.5%.
- For an LLC size of 12MB, the speedup is 1.3%.
- For an LLC size of 24MB, the speedup is not explicitly marked but appears to be close to the 1.3% value.

The chart highlights the performance improvements with increasing LLC sizes.
Accuracy and Coverage with Different Prefetchers

POPET’s accuracy and coverage increases significantly in absence of a data prefetcher.
Overhead of Hermes

- 4 KB storage overhead
- 1.5% power overhead*

*On top of an Intel Alder Lake-like performance-core configuration

More in the Paper

• Performance sensitivity to:
  - Cache hierarchy access latency
  - Hermes request issue latency
  - Activation threshold
  - ROB size (in extended version on arXiv)
  - LLC size (in extended version on arXiv)

• Accuracy, coverage, and performance analysis against HMP and TTP

• Understanding usefulness of each program feature

• Effect on stall cycle reduction

• Performance analysis on an eight-core system
More in the Paper

Performance sensitivity to:
- Cache hierarchy access latency
- Hermes request issue latency
- Activation threshold
- ROB size (in extended version at arXiv)
- LLC size (in extended version at arXiv)

Accuracy, coverage, and performance analysis against HMP and TTP

Understanding usefulness of each program feature

Effect on stall cycle reduction

Performance analysis in eight-core system

Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera\textsuperscript{1} Konstantinos Kanellopoulos\textsuperscript{1} Shankar Balachandran\textsuperscript{2} David Novo\textsuperscript{3}
Ataberk Olgun\textsuperscript{1} Mohammad Sadrosadati\textsuperscript{1} Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich \textsuperscript{2}Intel Processor Architecture Research Lab \textsuperscript{3}LIRMM, Univ. Montpellier, CNRS

Long-latency load requests continue to limit the performance of modern high-performance processors. To increase the latency tolerance of a processor, architects have primarily relied on two key techniques: sophisticated data prefetchers and large on-chip caches. In this work, we show that: (1) even a sophisticated state-of-the-art prefetcher can only predict half of the off-chip load requests on average across a wide range of workloads, and (2) due to the increasing size and complexity of on-chip caches, a large fraction of the latency of an off-chip load request is spent accessing the on-chip cache hierarchy to solely determine that it needs to go off-chip.

The goal of this work is to accelerate off-chip load requests by removing the on-chip cache access latency from their critical path. To this end, we propose a new technique called Hermes, whose key idea is to: (1) accurately predict which load requests off-chip main memory (i.e., an off-chip load) often stalls the processor core by blocking the instruction retirement from the reorder buffer (ROB), thus limiting the core’s performance [88, 91, 92]. To increase the latency tolerance of a core, computer architects primarily rely on two key techniques. First, they employ increasingly sophisticated hardware prefetchers that can learn complex memory address patterns and fetch data required by future load requests before the core demands them [28, 32, 33, 35, 75]. Second, they significantly scale up the size of the on-chip cache hierarchy with each new generation of processors [10, 11, 16].

Key problem. Despite recent advances in processor core design, we observe two key trends in new processor designs that leave a significant opportunity for performance improvement on the table. First, even a sophisticated state-of-the-art

To Summarize...
Hermes enables **off-chip load prediction**, a different form of speculation than **load address prediction** employed by prefetchers.

Off-chip load prediction can be applied **by itself** or **combined with load address prediction** to provide performance improvement.
Summary

Hermes employs the first perceptron-based off-chip load predictor

- **High accuracy** (77%)
- **High coverage** (74%)
- **Low storage overhead** (4KB/core)
- **High performance improvement** over best prior baseline (5.4%)
- **High performance per bandwidth**
Hermes is Open Sourced

All workload traces

13 prefetchers
- Stride [Fu+, MICRO’92]
- Streamer [Chen and Baer, IEEE TC’95]
- SMS [Somogyi+, ISCA’06]
- AMPM [Ishii+, ICS’09]
- Sandbox [Pugsley+, HPCA’14]
- BOP [Michaud, HPCA’16]
- SPP [Kim+, MICRO’16]
- Bingo [Bakshalipour+, HPCA’19]
- SPP+PPF [Bhatia+, ISCA’19]
- DSPatch [Bera+, MICRO’19]
- MLOP [Shakerinava+, DPC-3’19]
- IPCP [Pakalapati+, ISCA’20]
- Pythia [Bera+, MICRO’21]

9 off-chip predictors

Predictor type | Description
--- | ---
Base | Always NO
Basic | Simple confidence counter-based threshold
Random | Random Hit-miss predictor with a given positive probability
HMP-Local | Hit-miss predictor [Yoaz+, ISCA’99] with local prediction
HMP-GShare | Hit-miss predictor with GShare prediction
HMP-GSkew | Hit-miss predictor with GSkew prediction
HMP-Ensemble | Hit-miss predictor with all three types combined
TTP | Tag-tracking based predictor
Perc | Perceptron-based OCP used in this paper

https://github.com/CMU-SAFARI/Hermes
Easy To Define Your Own Off-Chip Predictor

• Just extend the **OffchipPredBase** class

```cpp
class OffchipPredBase
{
public:
    uint32_t cpu;
    string type;
    uint64_t seed;
    uint8_t dram_bw; // current DRAM bandwidth bucket

    OffchipPredBase(uint32_t _cpu, string _type, uint64_t _seed) : cpu(_cpu), type(_type), seed(_seed)
    {
        srand(seed);
        dram_bw = 0;
    }

    ~OffchipPredBase() {}

    void update_dram_bw(uint8_t _dram_bw) { dram_bw = _dram_bw; }

    virtual void print_config();
    virtual void dump_stats();
    virtual void reset_stats();
    virtual void train(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry);
    virtual bool predict(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry);
};

#endif /* OFFCHIP_PRED_BASE_H */
```
Easy To Define Your Own Off-Chip Predictor

• Define your own `train()` and `predict()` functions

```cpp
19  void OffchipPredBase::train(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
20  {
21    // nothing to train
22  }
23
24  bool OffchipPredBase::predict(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
25  {
26    // predict randomly
27    // return (rand() % 2) ? true : false;
28    return false;
29  }
```

• Get statistics like **accuracy** (stat name `precision`) and **coverage** (stat name `recall`) out of the box

```
Core_0_offchip_pred_true_pos 2358716
Core_0_offchip_pred_false_pos 276883
Core_0_offchip_pred_false_neg 132145
Core_0_offchip_pred_precision 89.49
Core_0_offchip_pred_recall 94.69
```
Off-Chip Prediction Can Further Enable...

**Prioritizing** loads that are likely go off-chip in cache queues and **on-chip network routing**

**Better instruction scheduling** of data-dependent instructions

**Other ideas to improve** performance and **fairness** in multi-core system design...
Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, Onur Mutlu

https://github.com/CMU-SAFARI/Hermes

Hermes Discussion

• FAQs
  - What are the selected set of program features?
  - Can you provide some intuition on why these features work?
  - What happens in case of a misprediction?
  - What’s the performance headroom for off-chip prediction?
  - Do you see a variance of different features in final prediction accuracy?

• Simulation Methodology
  - System parameters
  - Evaluated workloads

• More Results
  - Percentage of off-chip requests
  - Reduction in stall cycles by reducing the critical path
  - Fraction of off-chip load requests
  - Accuracy and coverage of POPET
  - Effect of different features
  - Are all features required?
  - 1C performance
  - 1C performance line graph
  - 1C performance against prior predictors
  - Effect on stall cycles
  - 8C performance
  - Sensitivity:
    - Hermes request issue latency
    - Cache hierarchy access latency
    - Activation threshold
    - ROB size
    - LLC size
  - Power overhead
  - Accuracy without prefetcher
  - Main memory request overhead with different prefetchers
Hermes Paper [MICRO 2022]

- Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, and Onur Mutlu,

"Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction"

Proceedings of the 55th International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, October 2022.

[Slides (pptx) (pdf)]
[Longer Lecture Slides (pptx) (pdf)]
[Talk Video (12 minutes)]
[Lecture Video (25 minutes)]
[arXiv version]

[Source Code (Officially Artifact Evaluated with All Badges)]

Officially artifact evaluated as available, reusable and reproducible.

Best paper award at MICRO 2022.

Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera\textsuperscript{1} Konstantinos Kanellopoulos\textsuperscript{1} Shankar Balachandran\textsuperscript{2} David Novo\textsuperscript{3} Ataberk Olgun\textsuperscript{1} Mohammad Sadrosadati\textsuperscript{1} Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich  \textsuperscript{2}Intel Processor Architecture Research Lab  \textsuperscript{3}LIRMM, Univ. Montpellier, CNRS

Sibyl: Reinforcement Learning based Data Placement in Hybrid SSDs
Self-Optimizing Hybrid SSD Controllers

Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gomez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu,
"Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning"
[Slides (pptx) (pdf)]
[arXiv version]
[Sibyl Source Code]
[Talk Video (16 minutes)]

Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh¹ Rakesh Nadig¹ Jisung Park¹ Rahul Bera¹ Nastaran Hajinazar¹
David Novo³ Juan Gómez-Luna¹ Sander Stuijk² Henk Corporaal² Onur Mutlu¹
¹ETH Zürich ²Eindhoven University of Technology ³LIRMM, Univ. Montpellier, CNRS

Sibyl
Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gómez Luna, Sander Stuijk, Henk Corporaal, Onur Mutlu
Executive Summary

• **Background**: A hybrid storage system (HSS) uses multiple different storage devices to provide high and scalable storage capacity at high performance

• **Problem**: Two key shortcomings of prior data placement policies:
  - Lack of adaptivity to:
    • Workload changes
    • Changes in device types and configurations
  - Lack of extensibility to more devices

• **Goal**: Design a data placement technique that provides:
  - *Adaptivity*, by continuously learning and adapting to the application and underlying device characteristics
  - *Easy extensibility* to incorporate a wide range of hybrid storage configurations

• **Contribution**: Sibyl, the first reinforcement learning-based data placement technique in hybrid storage systems that:
  - Provides adaptivity to changing workload demands and underlying device characteristics
  - Can easily extend to any number of storage devices
  - Provides ease of design and implementation that requires only a small computation overhead

• **Key Results**: Evaluate on real systems using a wide range of workloads
  - Sibyl improves performance by 21.6% compared to the best previous data placement technique in dual-HSS configuration
  - In a tri-HSS configuration, Sibyl outperforms the state-of-the-art-policy policy by 48.2%
  - Sibyl achieves 80% of the performance of an oracle policy with storage overhead of only 124.4 KiB

[https://github.com/CMU-SAFARI/Sibyl](https://github.com/CMU-SAFARI/Sibyl)
Talk Outline

Key Shortcomings of Prior Data Placement Techniques

- Formulating Data Placement as Reinforcement Learning
- Sibyl: Overview
- Evaluation of Sibyl and Key Results
- Conclusion
Hybrid Storage System Basics

Address Space (Application/File System View)

Logical Pages

Read

Write

Storage Management Layer

Read

Write

Promotion

Eviction

Fast Device

Slow Device

Hybrid Storage System
Performance of a hybrid storage system **highly depends** on the ability of the storage management layer.
Key Shortcomings in Prior Techniques

We observe **two key shortcomings** that significantly limit the performance benefits of prior techniques.

1. Lack of **adaptivity to**:
   a) Workload changes
   b) Changes in device types and configuration

2. Lack of **extensibility** to more devices
Lack of Adaptivity (1/2)

Workload Changes

Prior data placement techniques consider only a few workload characteristics that are statically tuned.
Lack of Adaptivity (2/2)

Changes in Device Types and Configurations

Do not consider **underlying storage device characteristics** (e.g., changes in the level asymmetry in read/write latencies, garbage collection)
**Lack of Extensibility (1/2)**

**Rigid techniques** that require significant effort to accommodate more than two devices

*Change in storage configuration*

*Dual-HSS*
Lack of Extensibility (2/2)

**Rigid techniques** that require significant effort to accommodate more than two devices

- Change in storage configuration
- Design a new policy

**Tri-HSS**
Our Goal

A **data-placement mechanism** that can provide:

1. **Adaptivity**, by continuously learning and adapting to the application and underlying device characteristics

2. **Easy extensibility** to incorporate a wide range of hybrid storage configurations
Our Proposal

Sibyl

Formulates data placement in hybrid storage systems as a reinforcement learning problem

Sibyl is an oracle that makes accurate prophecies
https://en.wikipedia.org/wiki/Sibyl
Talk Outline

- Key Shortcomings of Prior Data Placement Techniques
- Formulating Data Placement as Reinforcement Learning
- Sibyl: Overview
- Evaluation of Sibyl and Key Results
- Conclusion
Agent learns to take an **action** in a given **state** to maximize a numerical **reward**
Formulating Data Placement as RL

Agent

Environment

State ($S_t$)

Reward ($R_{t+1}$)

Action ($A_t$)

Hybrid Storage System

Sibyl

Features of the current request and system

Request latency (of last served request)

Select storage device to place the current page

Features of the current request and system

Request latency (of last served request)
What is State?

• **Limited number of state features:**
  - Reduce the implementation overhead
  - RL agent is more sensitive to reward

• **6-dimensional** vector of state features
  \[ O_t = (\text{size}_t, \text{type}_t, \text{intr}_t, \text{cnt}_t, \text{cap}_t, \text{curr}_t) \]

• We **quantize the state representation** into bins to reduce storage overhead
## Selected State Attributes

### Table 1: State features used by Sibyl

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th># of bins</th>
<th>Encoding (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$size_t$</td>
<td>Size of the requested page (in pages)</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>$type_t$</td>
<td>Type of the current request (read/write)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$intr_t$</td>
<td>Access interval of the requested page</td>
<td>64</td>
<td>8</td>
</tr>
<tr>
<td>$cnt_t$</td>
<td>Access count of the requested page</td>
<td>64</td>
<td>8</td>
</tr>
<tr>
<td>$cap_t$</td>
<td>Remaining capacity in the fast storage device</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>$curr_t$</td>
<td>Current placement of the requested page (fast/slow)</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
What is Reward?

• Defines the **objective** of Sibyl

• We formulate the reward as a function of the request latency

• Encapsulates three key aspects:
  - **Internal state of the device** (e.g., read/write latencies, the latency of garbage collection, queuing delays, ...)
  - **Throughput**
  - **Evictions**

• More details in the paper
Reward Function

**Reward.** After every data placement decision at time-step \( t \), Sibyl gets a reward from the environment at time-step \( t + 1 \) that acts as a feedback to Sibyl’s previous action. To achieve Sibyl’s performance goal, we craft the reward function \( R \) as follows:

\[
R = \begin{cases} 
\frac{1}{L_t} & \text{if no eviction of a page from the fast storage to the slow storage} \\
\max(0, \frac{1}{L_t} - R_p) & \text{in case of eviction}
\end{cases} \quad (1)
\]

where \( L_t \) and \( R_p \) represent the last served request latency and eviction penalty, respectively. If the fast storage is running out of free space, there might be evictions in the background from the fast storage to the slow storage. Therefore, we add an eviction penalty \( (R_p) \) to guide Sibyl to place only performance-critical pages in the fast storage. We empirically select \( R_p \) to be equal to \( 0.001 \times L_e \) \( (L_e \) is the time spent in evicting pages from the fast storage to the slow storage), which prevents the agent from aggressively placing all requests into the fast storage device.

---

\( ^4 \)In HSS, a time-step is defined as a new storage request.
What is Action?

• At every new page request, the action is to **select a storage device**

• Action can be **easily extended** to any number of storage devices

• Sibyl learns to **proactively evict or promote** a page
Sibyl Execution

RL Training Thread

State, Reward, and Action Information

Periodic Policy Weight Update

Storage Request (from OS)

RL Decision Thread

Data Placement Decision

Asynchronous Execution
Sibyl Design: Overview

- **Inference Network**
- **Experience Buffer (in host DRAM)**
- **State**
- **Observation Vector**
- **HSS**
- **RL Decision Thread**
- **RL Training Thread**

**Periodic Policy Weight Update**

**Training Network** → **Training Dataset** → **Batch**

**Observation Vector** → **State** → **Inference Network** → **Max** → **Action** → **Reward** → **Collect Experiences** → **HSS** → **Collect Experiences**

**Storage Request (from OS)**
RL Decision Thread

Inference Network

Max

HSS

Collect Experiences

Observation Vector

State

Storage Request (from OS)

Periodic Policy Weight Update

Training Network

Training Dataset

Batch

Experience Buffer (in host DRAM)

Reward

Collect Experiences

SAFARI
RL Decision Thread

Observation Vector

State

Storage Request (from OS)

Inference Network

Max

Sibyl Policy

Action

HSS

Reward

Collect Experiences

Experience Buffer (in host DRAM)

Batch

Training Dataset

Periodic Policy Weight Update

Training Network

RL Training Thread

RL Decision Thread
RL Decision Thread

1. Training Network
2. Inference Network
3. RL Training Thread
4. Periodic Policy Weight Update
5. Training Dataset
6. Batch
7. RL Decision Thread
8. Sibyl Policy
9. Experience Buffer (in host DRAM)
10. Observation Vector
11. State
12. Inference Network
13. Max
14. Action
15. HSS
16. Reward
17. Collect Experiences
18. Storage Request (from OS)
RL Decision Thread

Periodic Policy Weight Update

Training Network

Inference Network

Max

Observation Vector

Storage Request (from OS)

State

Reward

Collect Experiences

HSS

Experience Buffer (in host DRAM)

Action

Sibyl Policy

Training Dataset

Batch

RL Training Thread

State

Reward

Collect Experiences

HSS

Experience Buffer (in host DRAM)

Action

Sibyl Policy

Training Dataset

Batch

RL Training Thread

Periodic Policy Weight Update

Training Network

Inference Network

Max

Observation Vector

Storage Request (from OS)

State

Reward

Collect Experiences

HSS

Experience Buffer (in host DRAM)

Action

Sibyl Policy

Training Dataset

Batch

RL Training Thread

Periodic Policy Weight Update

Training Network

Inference Network

Max

Observation Vector

Storage Request (from OS)

State

Reward

Collect Experiences

HSS

Experience Buffer (in host DRAM)

Action

Sibyl Policy

Training Dataset

Batch

RL Training Thread

Periodic Policy Weight Update

Training Network

Inference Network

Max

Observation Vector

Storage Request (from OS)

State

Reward

Collect Experiences

HSS

Experience Buffer (in host DRAM)

Action

Sibyl Policy

Training Dataset

Batch

RL Training Thread

Periodic Policy Weight Update

Training Network

Inference Network

Max

Observation Vector

Storage Request (from OS)

State

Reward

Collect Experiences

HSS

Experience Buffer (in host DRAM)

Action

Sibyl Policy

Training Dataset

Batch

RL Training Thread

Periodic Policy Weight Update

Training Network

Inference Network

Max

Observation Vector

Storage Request (from OS)

State

Reward

Collect Experiences

HSS

Experience Buffer (in host DRAM)

Action

Sibyl Policy

Training Dataset

Batch

RL Training Thread
RL Decision Thread

- **Training Network**
  - Periodic Policy Weight Update
- **Training Dataset**
- **Batch**
- **Experience Buffer (in host DRAM)**
  - **HSS**
  - **Collect Experiences**

- **Observation Vector**
  - **State**
  - Storage Request (from OS)
  - **Inference Network**
    - Max
  - **Sibyl Policy**
  - **Action**
  - **Reward**

- **RL Decision Thread**
RL Training Thread

Training Network

Periodic Policy Weight Update

Training Dataset

Batch

Experience Buffer (in host DRAM)

RL Training Thread

RL Decision Thread

State

Observation Vector

State

Inference Network

Max

Sibyl Policy

Action

HSS

Reward

Collect Experiences

Storage Request (from OS)
Periodic Weight Transfer

Periodic Policy Weight Update

Observation Vector → Inference Network

State → Max

Action → HSS

Reward → Collect Experiences

Experience Buffer (in host DRAM)

Batch

Training Dataset

Training Network

Periodic Policy Weight Update

SAFARI
Training and Inference Networks

- Training and inference networks **allow parallel execution**

- Observation vector as the input

- Produces probability distribution of Q-values
**Algorithm 1** Sibyl’s reinforcement learning-based data placement algorithm

1: **Initialize**: the experience buffer $EB$ to capacity $e_{EB}$
2: **Initialize**: the training network with random weights $\theta$
3: **Initialize**: the inference network with random weights $\hat{\theta}$
4: **Initialize**: the observation vector $O_t = O(s_t)$ with storage request $s_t = \{req_t\}$, and host and storage features
5: **for all** storage requests **do**
6: \[ \text{if } \text{rand() } < \epsilon \text{ then} \quad \triangleright \text{ with probability } \epsilon, \text{ perform exploration} \]
7: \[ \text{random action } a_t \]
8: \[ \text{else} \quad \triangleright \text{ with probability } 1-\epsilon, \text{ perform exploitation} \]
9: \[ a_t = \operatorname{argmax}_a Q_t(a) \quad \triangleright \text{ select action with the highest } Q_t \text{ value from inference network} \]
10: \[ \text{execute } a_t \quad \triangleright \text{ place the requested page to fast or slow storage} \]
11: \[ \text{if no eviction then} \]
12: \[ r_t \leftarrow \frac{1}{L_t} \quad \triangleright \text{ reward, given no eviction of a page from fast to slow storage} \]
13: \[ \text{else} \]
14: \[ r_t \leftarrow \max(0, \frac{1}{L_t} - R_P) \quad \triangleright \text{ reward with an eviction penalty in case of an eviction} \]
15: \[ \text{store experience } (O_t, a_t, r_t, O(t+1)) \text{ in } EB \]
16: \[ \text{if (num requests in } EB = e_{EB}) \text{ then} \quad \triangleright \text{ train training network when } EB \text{ is full} \]
17: \[ \text{sample random batches of experiences from } EB, \text{ which are in format } (O_j, a_j, r_j, O(j+1)) \quad \triangleright \text{ where } O_j \text{ represents an observation at a time instant } j \text{ from } EB \]
18: \[ \text{Perform stochastic gradient descent} \quad \triangleright \text{ update the training network weights} \]
19: \[ \hat{\theta} \leftarrow \theta \quad \triangleright \text{ copy the training network weights to the inference network} \]
Hyperparameter Tuning

Table 2: Hyper-parameters considered for tuning

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Design Space</th>
<th>Chosen Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discount factor ($\gamma$)</td>
<td>0-1</td>
<td>0.9</td>
</tr>
<tr>
<td>Learning rate ($\alpha$)</td>
<td>$1e^{-5} - 1e^{0}$</td>
<td>$1e^{-4}$</td>
</tr>
<tr>
<td>Exploration rate ($\epsilon$)</td>
<td>0-1</td>
<td>0.001</td>
</tr>
<tr>
<td>Batch size</td>
<td>64-256</td>
<td>128</td>
</tr>
<tr>
<td>Experience buffer size ($e_{EB}$)</td>
<td>10-10000</td>
<td>1000</td>
</tr>
</tbody>
</table>

Figure 14: Sensitivity of Sibyl throughput to: (a) the discount factor ($\gamma$), (b) the learning rate ($\alpha$), (c) the exploration rate ($\epsilon$), averaged across 14 workloads (normalized to Fast-Only)
Evaluation Methodology (1/3)

• **Real system** with various HSS configurations
  - Dual-hybrid and tri-hybrid systems
Evaluation Methodology (2/3)

Cost-Oriented HSS Configuration

- High-end SSD
- Low-end HDD

Performance-Oriented HSS Configuration

- High-end SSD
- Middle-end SSD
Evaluation Methodology (3/3)

• **18 different workloads** from:
  - MSR Cambridge and Filebench Suites

• **Four** state-of-the-art data placement baselines:
  - CDE [Matsui+, Proc. IEEE’17]
  - HPS [Meswani+, HPCA’15]
  - Archivist [Ren+, ICCD’19]
  - RNN-HSS [Doudali+, HPDC’19]
Performance Analysis

Cost-Oriented HSS Configuration

![Graph showing normalized average request latency for different configurations and workloads. The x-axis represents various workloads including `hm_1`, `mds_0`, `prn_1`, `proj_0`, `proj_2`, `proj_3`, `prxy_0`, `prxy_1`, `rsrc_0`, `srcl_0`, `stg_1`, `usr_0`, `wdev_2`, `web_1`, and `AVG`. The y-axis represents the normalized average request latency ranging from 0 to 200. Different configurations are color-coded: Slow-Only, CDE, HPS, Archivist, RNN-HSS, Sibyl, and Oracle.](image-url)
Sibyl consistently outperforms all the baselines for all the workloads.
Performance Analysis

Performance-Oriented HSS Configuration

- Slow-Only
- CDE
- HPS
- Archivist
- RNN-HSS
- Sibyl
- Oracle

Normalized Average Request Latency

- hm_1
- mds_0
- prn_1
- proj_0
- proj_2
- proj_3
- prxy_0
- prxy_1
- rsrch_0
- src1_0
- stg_1
- usr_0
- wdev_2
- web_1
- AVG
Performance Analysis

Performance-Oriented HSS Configuration

Sibyl provides 21.6% performance improvement by dynamically adapting its data placement policy.
Performance Analysis

Performance-Oriented HSS Configuration

Normalized Average Request Latency

1. Slow-Only
2. CDE
3. HPS
4. Archivist
5. RNN-HSS
6. Sibyl
7. Oracle

High-end SSD
Mid-end SSD
Sibyl achieves **80% of the performance of an oracle policy** that has complete knowledge of future access patterns.
Performance on Tri-HSS

Extending Sibyl for **more devices**:

1. **Add a new action**
2. **Add the remaining capacity** of the new device as a state feature

![Graph showing performance on Tri-HSS with different types of storage devices](image)

**Legend:**
- **Heuristic\text{Tri-hybrid}**
- **Sibyl\text{Tri-hybrid}**

**Axis:**
- X-axis: Dataset names (`hm_1`, `mds_0`, `prn_1`, `proj_0`, `proj_2`, `proj_3`, `prxy_0`, `prxy_1`, `rsrc_0`, `src1_0`, `stg_1`, `usr_0`, `wdev_2`, `web_1`, `avg`)
- Y-axis: Normalized Average Request Latency
Performance on Tri-HSS

Extending Sibyl for more devices:

1. Add a new action
2. Add the remaining capacity of the new device as a state feature
Performance on Tri-HSS

Extending Sibyl for more devices:
1. Add a new action
2. Add the remaining capacity of the new device as a state feature

Sibyl **outperforms** the state-of-the-art data placement policy by **48.2%** in a real tri-hybrid system.

Sibyl reduces the system architect's burden by providing **ease of extensibility**.
Sibyl’s Overhead

- **124.4 KiB** of total storage cost
  - Experience buffer, inference and training network
- **40-bit** metadata overhead per page for state features
- Inference latency of **~10ns**
- Training latency of **~2us**

- Small area overhead
- Small inference overhead
- Satisfies prediction latency
More in the Paper (1/3)

• Throughput (IOPS) evaluation
  - Sibyl provides high IOPS compared to baseline policies because it indirectly captures throughput (size/latency)

• Evaluation on unseen workloads
  - Sibyl can effectively adapt its policy to highly dynamic workloads

• Evaluation on mixed workloads
  - Sibyl provides equally-high performance benefits as in single workloads
More in the Paper (2/3)

• Evaluation using **different features**
  - Sibyl *autonomously decides* which features are important to maximize the performance

• Evaluation with **different hyperparameter values**

• Sensitivity to **fast storage capacity**
  - Sibyl *provides scalability* by dynamically adapting its policy to available storage size

• **Explainability analysis** of Sibyl's decision making
  - Explain Sibyl’s actions for different workload characteristics and device configurations
Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh\textsuperscript{1}, Rakesh Nadig\textsuperscript{1}, Jisung Park\textsuperscript{1}, Rahul Bera\textsuperscript{1}, Nastaran Hajinazar\textsuperscript{1}
David Novo\textsuperscript{3}, Juan Gómez-Luna\textsuperscript{1}, Sander Stuijk\textsuperscript{2}, Henk Corporaal\textsuperscript{2}, Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich \quad \textsuperscript{2}Eindhoven University of Technology \quad \textsuperscript{3}LIRMM, Univ. Montpellier, CNRS


https://github.com/CMU-SAFARI/Sibyl
**Talk Outline**

- Key Shortcomings of Prior Data Placement Techniques
- Formulating Data Placement as Reinforcement Learning
- Sibyl: Overview
- Evaluation of Sibyl and Key Results
- Conclusion
Conclusion

• We introduced Sibyl, the first reinforcement learning-based data placement technique in hybrid storage systems that provides
  - Adaptivity
  - Easily extensibility
  - Ease of design and implementation

• We evaluated Sibyl on real systems using many different workloads
  - Sibyl improves performance by 21.6% compared to the best prior data placement policy in a dual-HSS configuration
  - In a tri-HSS configuration, Sibyl outperforms the state-of-the-art data placement policy by 48.2%
  - Sibyl achieves 80% of the performance of an oracle policy with a storage overhead of only 124.4 KiB

SAFARI https://github.com/CMU-SAFARI/Sibyl
Sibyl
Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gómez Luna, Sander Stuijk, Henk Corporaal, Onur Mutlu
Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh1, Rakesh Nadig1, Jisung Park1, Rahul Bera1, Nastaran Hajinazar1, David Novo3, Juan Gómez-Luna1, Sander Stuijk2, Henk Corporaal2, Onur Mutlu1

1ETH Zürich 2Eindhoven University of Technology 3LIRMM, Univ. Montpellier, CNRS

SSD Course (Spring 2023)

- **Spring 2023 Edition:**

- **Fall 2022 Edition:**
  - https://safari.ethz.ch/projects_and_seminars/fall2022/doku.php?id=modern_ssds

- **Youtube Livestream (Spring 2023):**
  - https://www.youtube.com/watch?v=4VTwOMmsnJY&list=PL5Q2soXY2Zi_8qOM5Icpp8hB2SHtm4z57&pp=iAQB

- **Youtube Livestream (Fall 2022):**
  - https://www.youtube.com/watch?v=hqLrd-Uj0aU&list=PL5Q2soXY2Zi9BJhenUq4JI5bwhAMpAp13&pp=iAQB

- **Project course**
  - Taken by Bachelor’s/Master’s students
  - SSD Basics and Advanced Topics
  - Hands-on research exploration
  - Many research readings

https://www.youtube.com/onurmutlulectures
Comp Arch (Fall 2021)

- **Fall 2021 Edition:**
  - [https://safari.ethz.ch/architecture/fall2021/doku.php?id=schedule](https://safari.ethz.ch/architecture/fall2021/doku.php?id=schedule)

- **Fall 2020 Edition:**

- **Youtube Livestream (2021):**
  - [https://www.youtube.com/watch?v=4yfkM_5EFgo&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF](https://www.youtube.com/watch?v=4yfkM_5EFgo&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF)

- **Youtube Livestream (2020):**
  - [https://www.youtube.com/watch?v=c3mPdZA-Fmc&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN](https://www.youtube.com/watch?v=c3mPdZA-Fmc&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN)

- **Master’s level course:**
  - Taken by Bachelor’s/Masters/PhD students
  - Cutting-edge research topics + fundamentals in Computer Architecture
  - 5 Simulator-based Lab Assignments
  - Potential research exploration
  - Many research readings

https://www.youtube.com/onurmutlulectures
Machine Learning Driven Memory and Storage Systems

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
19 December 2023
IBM Research
PYTHIA BACKUP
Reward Assignment to EQ Entry

- **Every** action gets inserted into EQ
- Reward is assigned to each EQ entry *before or during* the eviction

- **During EQ insertion**: for actions
  - Not to prefetch
  - Out-of-page prefetch
Reward Assignment to EQ Entry

- **Every** action gets inserted into EQ
- Reward is assigned to each EQ entry **before or during** the eviction

- **During EQ insertion**: for actions
  - Not to prefetch
  - Out-of-page prefetch

- **During EQ residency**:  
  - In case address of a demand matches with address in EQ (*signifies accurate prefetch*)
Reward Assignment to EQ Entry

- **Every** action gets inserted into EQ
- Reward is assigned to each EQ entry **before or during** the eviction

- **During EQ insertion:** for actions
  - Not to prefetch
  - Out-of-page prefetch

- **During EQ residency:**
  - In case address of a demand matches with address in EQ
  (signifies accurate prefetch)

- **During EQ eviction:**
  - In case no reward is assigned till eviction
  (signifies inaccurate prefetch)
Performance S-curve: Single-core

Speedup over no prefetching

Workload number

SPP  Bingo  MLOP  Pythia

623.xalancbmk_s-592B  603.bwaves_s-2931B  462.libquantum

streamcluster  fluidanimate-9500M

BFSCC-22B  pagerank-51B  429.mcf
Performance S-curve: Four-core

Speedup over no prefetching

Workload number

- SPP
- Bingo
- MLOP
- Pythia

- 429.mcf-184B
- Bingo
- 462.libquantum-1343B
- Pythia
- raytrace-23.75B
- Mix-240
- Mix-59
- pagerank

SAFARI
FAQs
Pythia Discussion

• FAQs
  - Why RL?
  - What about large page?
  - What’s the prefetch degree?
  - Can customization happen during workload execution?
  - Can runtime mixing create problem?

• Simulation and Methodology
  - Basic Pythia configuration
  - System parameters
  - Configuration of prefetchers
  - Evaluated workloads
  - Feature selection

• Detailed Design
  - Reward structure
  - Design overview
  - QVStore Organization

• More Results
  - Comparison against other adaptive prefetchers
  - Comparison against Context prefetcher
  - Feature combination sensitivity
  - Hyperparameter sensitivity
  - Comparison with multi-level prefetchers
  - Performance in unseen workloads
  - Single-core s-curve
  - Four-core s-curve
  - Detailed performance analysis
  - Benefit of bandwidth awareness
  - Case study
  - Customizing rewards
  - Customizing features
Why RL? Why Not Supervised Learning?

• Determining the **benefits of prefetching** (i.e., whether a decision was good for performance or not) is **not easy**
  - Depends on a complex set of metrics
    • Coverage, accuracy, timeliness
    • Effects on system: b/w usage, pollution, cross-application interference, ...
  - **Dynamically-changing environmental conditions** change the benefit
  - **Delayed feedback due to long latency** (might not receive feedback at all for inaccurate prefetches!)

• Differs from classification tasks (e.g., branch prediction)
  - Performance strongly correlates mainly to accuracy
  - Does not depend on environment
  - Bounded feedback delay
What About Large Pages?

• Pythia’s framework can be easily extended to incorporate additional prefetch actions (i.e., possible prefetch offsets for the page size)

• To decrease the storage overhead
  - Prune action space via automatic design-space exploration
  - Hash action values to retrieve Q-values
What is the Prefetch Degree? Is It Managed by the RL Agent?

• Pythia employs a simple degree selector, separate from the RL agent
  - If the agent has selected the same prefetch action (O) multiple times in a row, Pythia increases the degree (A+2O, A+3O, ...)
  - At most degree 4

• Future works on managing degree by the RL agent
Can the Customization Be Done While the Workload is Running?

• Certainly.

• Pythia, being an **online learning** technique, will autonomously adapt (and optimize) its policy to use the new program features or the modified reward values.
Can Runtime Workload Mix Create an Issue?

- We implement the bandwidth usage feedback using a counter in the memory controller. Thus Pythia already has a **global view of the memory bandwidth usage** that incorporates all workloads running on a multi-core system.

- We evaluate a diverse set (300 of each category) of four-core, eight-core, twelve-core random workload mixes.

- Based on our evaluation, we observe that **Pythia dynamically adapts** itself to varying workload demands.
How does Pythia Compare Against Other Adaptive Prefetching Solutions?

• We compare Pythia against IBM POWER7\textsuperscript{[5]} prefector
  - Adaptively selects prefetcher degree/configuration by monitoring program IPC

![Geomean speedup over baseline](Image)

(a) single-core

![Geomean speedup over baseline](Image)

(b) four-core

\[5\] Jimenez et al., TOPC’14
How Does Pythia Compare Against the Context Prefetcher?

- Pythia widely differs from the Context Prefetcher (CP)\textsuperscript{[6]} in all three aspects: state, action, and reward. The key differences are:
  - CP \textbf{does not consider system-level feedback}
  - CP models the agent as a contextual bandit which \textbf{takes myopic prefetch decisions} as compared to Pythia
  - CP \textbf{requires compiler support} to extract software-level features

\begin{figure}
\centering
\includegraphics[width=\textwidth]{comparison.png}
\caption{Comparison of Pythia and CP-HW on various benchmarks.}
\end{figure}

Pythia outperforms CP-HW by \textbf{5.3\% in single-core} and \textbf{7.6\% in four-core system}.

\textsuperscript{[6]} Leeor et al., ISCA’15
How Pythia’s Performance Changes With Various State Definitions You Have Swept?

• In total we evaluate state defined as any-one, any-two, and any-three combinations of 32 features

Performance gain ranges from 20.7% to 22.4%

Coverage ranges from 66.2% to 71.5%

Overprediction ranges from 26.7% to 32.2%
Is Pythia Sensitive to Hyperparameters?

• Not setting hyperparameters can significantly impact the overall performance improvement

Changing $\varepsilon$ from 0.002 to 1.0 drops perf. by 16%

Changing $\alpha$ from 0.0065 to 1.0 drops perf. by 5.4%
How Does Pythia Compare Against Commercial Multi-level Prefetchers?

Pythia outperforms IPCP [7] by **14.2%** on average in 150-MTPS

[6] Prakashapati et al., ISCA’20
Does Pythia Perform Equally Well for Unseen Workloads?

• Evaluated with 500 traces from value prediction championship
  - No prefetcher has been trained on these traces

Pythia outperforms MLOP and Bingo by
8.3% and 3.5% in single-core

And 9.7% and 5.4% in four-core
# Basic Pythia Configuration

Table 2: Basic Pythia configuration derived from our automated design-space exploration

<table>
<thead>
<tr>
<th>Features</th>
<th>PC+Delta, Sequence of last-4 deltas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefetch Action List</td>
<td>{-6,-3,-1,0,1,3,4,5,10,11,12,16,22,23,30,32}</td>
</tr>
<tr>
<td>Reward Level Values</td>
<td>$\mathcal{R}<em>{AT}=20$, $\mathcal{R}</em>{AL}=12$, $\mathcal{R}<em>{CL}=-12$, $\mathcal{R}</em>{IN}^{H}=-14$, $\mathcal{R}<em>{IN}^{L}=-8$, $\mathcal{R}</em>{NP}^{H}=-2$, $\mathcal{R}_{NP}^{L}=-4$</td>
</tr>
<tr>
<td>Hyperparameters</td>
<td>$\alpha = 0.0065$, $\gamma = 0.556$, $\varepsilon = 0.002$</td>
</tr>
</tbody>
</table>
# System Parameters

## Table 5: Simulated system parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core</strong></td>
<td>1-12 cores, 4-wide OoO, 256-entry ROB, 72/56-entry LQ/SQ</td>
</tr>
<tr>
<td><strong>Branch Pred.</strong></td>
<td>Perceptron-based [69], 20-cycle misprediction penalty</td>
</tr>
<tr>
<td><strong>L1/L2 Caches</strong></td>
<td>Private, 32KB/256KB, 64B line, 8 way, LRU, 16/32 MSHRs,  4-cycle/14-cycle round-trip latency</td>
</tr>
<tr>
<td><strong>LLC</strong></td>
<td>2MB/core, 64B line, 16 way, SHiP [133], 64 MSHRs per LLC Bank, 34-cycle round-trip latency</td>
</tr>
<tr>
<td><strong>Main Memory</strong></td>
<td>1C: Single channel, 1 rank/channel; 4C: Dual channel, 2 ranks/channel; 8C: Quad channel, 2 ranks/channel; 8 banks/rank, 2400 MTPS, 64b data-bus/channel, 2KB row buffer/bank, tRCD=15ns, tRP=15ns, tCAS=12.5ns</td>
</tr>
</tbody>
</table>
## Configuration of Prefetchers

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPP [78]</td>
<td>256-entry ST, 512-entry 4-way PT, 8-entry GHR</td>
<td>6.2 KB</td>
</tr>
<tr>
<td>Bingo [27]</td>
<td>2KB region, 64/128/4K-entry FT/AT/PHT</td>
<td>46 KB</td>
</tr>
<tr>
<td>MLOP [111]</td>
<td>128-entry AMT, 500-update, 16-degree</td>
<td>8    KB</td>
</tr>
<tr>
<td>DSPatch [30]</td>
<td>Same configuration as in [30]</td>
<td>3.6 KB</td>
</tr>
<tr>
<td>PPF [32]</td>
<td>Same configuration as in [32]</td>
<td>39.3 KB</td>
</tr>
<tr>
<td>Pythia</td>
<td>2 features, 2 vaults, 3 planes, 16 actions</td>
<td>25.5 KB</td>
</tr>
</tbody>
</table>
## Evaluated Workloads

### Table 6: Workloads used for evaluation

<table>
<thead>
<tr>
<th>Suite</th>
<th># Workloads</th>
<th># Traces</th>
<th>Example Workloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEC06</td>
<td>16</td>
<td>28</td>
<td>gcc, mcf, cactusADM, lbm, ...</td>
</tr>
<tr>
<td>SPEC17</td>
<td>12</td>
<td>18</td>
<td>gcc, mcf, pop2, fotonik3d, ...</td>
</tr>
<tr>
<td>PARSEC</td>
<td>5</td>
<td>11</td>
<td>canneal, facesim, raytrace, ...</td>
</tr>
<tr>
<td>Ligra</td>
<td>13</td>
<td>40</td>
<td>BFS, PageRank, Bellman-ford, ...</td>
</tr>
<tr>
<td>Cloudsuite</td>
<td>4</td>
<td>53</td>
<td>cassandra, cloud9, nutch, ...</td>
</tr>
</tbody>
</table>
## Table 3: List of program control-flow and data-flow components used to derive the list of features for exploration

<table>
<thead>
<tr>
<th>Control-flow Component</th>
<th>Data-flow Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) PC of load request</td>
<td>(1) Load cacheline address</td>
</tr>
<tr>
<td>(2) PC-path (XOR-ed last-3 PCs)</td>
<td>(2) Page number</td>
</tr>
<tr>
<td>(3) PC XOR-ed branch-PC</td>
<td>(3) Page offset</td>
</tr>
<tr>
<td>(4) None</td>
<td>(4) Load address delta</td>
</tr>
<tr>
<td></td>
<td>(5) Sequence of last-4 offsets</td>
</tr>
<tr>
<td></td>
<td>(6) Sequence of last-4 deltas</td>
</tr>
<tr>
<td></td>
<td>(7) Offset XOR-ed with delta</td>
</tr>
<tr>
<td></td>
<td>(8) None</td>
</tr>
</tbody>
</table>
MORE RESULTS
Performance S-curve: Single-core

The graph illustrates the speedup over no prefetching for various workloads on a single-core processor. The x-axis represents the workload number, while the y-axis shows the speedup. Different lines represent different workloads:

- Orange line: SPP
- Blue line: Bingo
- Green line: MLOP
- Black line: Pythia

Workloads include:
- 623.xalancbmk_s-592B
- 603.bwaves_s-2931B
- 462.libquantum
- streamcluster
- fluidanimate-9500M
- pagerank-51B
- BFSCC-22B
- 429.mcf

The graph shows a trend of increasing speedup with workload number, indicating improved performance as the workload is executed.
Performance S-curve: Four-core

Speedup over no prefetching

Workload number

- SPP
- Bingo
- MLOP
- Pythia

- 429.mcf-184B
- 462.libquantum-1343B
- 437.leslie3d-271B
- Mix-59
- Mix-240
- Mix-201
- raytrace-23.75B
- pagerank
Single-core Coverage & Overprediction

SPEC06
SPEC17
PARSEC
Ligra
Cloudsuite
AVG

Fraction of LLC misses

Covered
Uncovered
Overpredicted
Detailed Performance

![Bar chart showing geomean speedup over baseline for different benchmarks and configurations.](chart.png)

(a) SPEC06, SPEC17, PARSEC, Ligra, Cloudsuite, GEOMEAN

(b) St, St+S, St+S+B, St+S+B+D, St+S+B+D+M, Pythia

SAFARI
Benefit of Bandwidth Awareness

Performance normalized to basic Pythia

Memory BW-oblivious Pythia

DRAM MTPS (in log scale)
Figure 13: Q-value curves of PC+Delta feature values (a) 0x436a81+0 and (b) 0x4377c5+0 in 459.GemsFDTD-1320B.
Customizing Rewards

Figure 14: Performance and main memory bandwidth usage of prefetchers in Ligra-CC.

Figure 15: Performance of the basic and strict Pythia configurations on the Ligra workload suite.
Customizing Features

Figure 16: Performance of the basic and feature-optimized Pythia on the SPEC CPU2006 suite.
Hermes Discussion

• FAQs
  - What are the selected set of program features?
  - Can you provide some intuition on why these features work?
  - What happens in case of a misprediction?
  - What’s the performance headroom for off-chip prediction?
  - Do you see a variance of different features in final prediction accuracy?

• More Results
  - Percentage of off-chip requests
  - Reduction in stall cycles by reducing the critical path
  - Fraction of off-chip load requests
  - Accuracy and coverage of POPET
  - Effect of different features
  - Are all features required?
  - 1C performance
  - 1C performance line graph
  - 1C performance against prior predictors
  - Effect on stall cycles
  - 8C performance
  - Sensitivity:
    - Hermes request issue latency
    - Cache hierarchy access latency
    - Activation threshold
    - ROB size
    - LLC size
  - Power overhead
  - Accuracy without prefetcher
  - Main memory request overhead with different prefetchers

• Simulation Methodology
  - System parameters
  - Evaluated workloads
## Initial Set of Program Features

<table>
<thead>
<tr>
<th>Features without control-flow information</th>
<th>Features with control-flow information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Load virtual address</td>
<td>8. Load PC</td>
</tr>
<tr>
<td>2. Virtual page number</td>
<td>9. PC ⊕ load virtual address</td>
</tr>
<tr>
<td>3. Cacheline offset in page</td>
<td>10. PC ⊕ virtual page number</td>
</tr>
<tr>
<td>4. First access</td>
<td>11. PC ⊕ cacheline offset</td>
</tr>
<tr>
<td>5. Cacheline offset + first access</td>
<td>12. PC + first access</td>
</tr>
<tr>
<td>6. Byte offset in cacheline</td>
<td>13. PC ⊕ byte offset</td>
</tr>
<tr>
<td>7. Word offset in cacheline</td>
<td>14. PC ⊕ word offset</td>
</tr>
<tr>
<td></td>
<td>15. Last-4 load PCs</td>
</tr>
<tr>
<td></td>
<td>16. Last-4 PCs</td>
</tr>
</tbody>
</table>
Selected Set of Program Features

**Five features**

- PC ⊕ cacheline offset
- PC ⊕ byte offset
- PC + first access
- Cacheline offset + first access
- Last-4 load PCs

A **binary hint** that represents whether or not a cacheblock has been recently touched.
When A Feature Works/Does Not Work?

Trace: 462.libquantum-1343B

PC: 0x401442

With a simple stride prefetcher
• Cacheline offset + first access

Without prefetcher
• PC + first access
• Cacheline offset + first access
What Happens in case of a Misprediction?

• Two cases of mispredictions:

• Predicted on-chip but actually goes off-chip
  - Loss of performance improvement opportunity

No need for misprediction detection and recovery

• Predicted off-chip but actually is on-chip
  - Memory controller forwards the data to LLC if and only if a load to the same address have already missed LLC and arrived at the memory controller

No need for misprediction detection and recovery
Performance Headroom of Off-Chip Prediction

(a) Geometric speedup over the No-Prefetching system

- Ideal Hermes
- Pythia (baseline)
- Pythia + Ideal Hermes

(b) Geometric speedup over the No-Prefetching system

- Prefetcher-only
- Prefetcher + Ideal Hermes

SPEC06: (a) Pythia + Ideal Hermes: 1.29
SPEC17: (a) Pythia + Ideal Hermes: 1.29
PARSEC: (a) Pythia + Ideal Hermes: 1.29
Ligra: (a) Pythia + Ideal Hermes: 1.24
CVP: (a) Pythia + Ideal Hermes: 1.29
GEOMEAN: (a) Pythia + Ideal Hermes: 1.20

Pythia: (b) Prefetcher-only: 1.20, Prefetcher + Ideal Hermes: 1.29
Bingo: (b) Prefetcher-only: 1.19, Prefetcher + Ideal Hermes: 1.29
SPP: (b) Prefetcher-only: 1.14, Prefetcher + Ideal Hermes: 1.23
MLOP: (b) Prefetcher-only: 1.13, Prefetcher + Ideal Hermes: 1.24
SMS: (b) Prefetcher-only: 1.06, Prefetcher + Ideal Hermes: 1.19

SAFARI
## System Parameters

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core</strong></td>
<td>1 and 8 cores, 6-wide fetch/execute/commit, 512-entry ROB, 128/72-entry LQ/SQ, Perceptron branch predictor [61] with 17-cycle misprediction penalty</td>
</tr>
<tr>
<td><strong>L1/L2 Caches</strong></td>
<td>Private, 48KB/1.25MB, 64B line, 12/20-way, 16/48 MSHRs, LRU, 5/15-cycle round-trip latency [25]</td>
</tr>
<tr>
<td><strong>LLC</strong></td>
<td>3MB/core, 64B line, 12 way, 64 MSHRs/slice, SHiP [122], 55-cycle round-trip latency [24, 25], Pythia prefetcher [32]</td>
</tr>
<tr>
<td><strong>Main Memory</strong></td>
<td>1C: 1 channel, 1 rank per channel; 8C: 4 channels, 2 ranks per channel; 8 banks per rank, DDR4-3200 MTPS, 64b data-bus per channel, 2KB row buffer per bank, tRCD=12.5ns, tRP=12.5ns, tCAS=12.5ns</td>
</tr>
<tr>
<td><strong>Hermes</strong></td>
<td>Hermes-O/P: 6/18-cycle Hermes request issue latency</td>
</tr>
</tbody>
</table>
## Evaluated Workloads

Table 5: Workloads used for evaluation

<table>
<thead>
<tr>
<th>Suite</th>
<th>#Workloads</th>
<th>#Traces</th>
<th>Example Workloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEC06</td>
<td>14</td>
<td>22</td>
<td>gcc, mcf, cactusADM, lbm, ...</td>
</tr>
<tr>
<td>SPEC17</td>
<td>11</td>
<td>23</td>
<td>gcc, mcf, pop2, fotonik3d, ...</td>
</tr>
<tr>
<td>PARSEC</td>
<td>4</td>
<td>12</td>
<td>canneal, facesim, raytrace, ...</td>
</tr>
<tr>
<td>Ligra</td>
<td>11</td>
<td>20</td>
<td>BFS, PageRank, Radii, ...</td>
</tr>
<tr>
<td>CVP</td>
<td>33</td>
<td>33</td>
<td>integer, floating-point, server, ...</td>
</tr>
</tbody>
</table>
Observation: **Not All Off-Chip Loads are Prefetched**

Nearly **50%** of the loads are still not prefetched.
Observation: **Not All Off-Chip Loads are Prefetched**

70% of these off-chip loads blocks ROB
Observation: With Large Cache Comes Longer Latency

- On-chip cache access latency significantly contributes to the latency of an off-chip load.

![Graph showing the impact of on-chip cache hierarchy access latency on stall cycles due to off-chip load blocking instruction retirement from ROB. The graph compares SPEC06, SPEC17, PARSEC, Ligra, CVP, and AVG.]
Observation: With Large Cache Comes Longer Latency

• On-chip cache access latency significantly contributes to the latency of an off-chip load

40% of stall cycles caused by an off-chip load can be eliminated by removing on-chip cache access latency from its critical path.
What Fraction of Load Requests Goes Off-Chip?

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Off-chip rate</th>
<th>LLC MPKI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEC06</td>
<td>6%</td>
<td></td>
</tr>
<tr>
<td>SPEC17</td>
<td>4%</td>
<td></td>
</tr>
<tr>
<td>PARSEC</td>
<td>4%</td>
<td></td>
</tr>
<tr>
<td>Ligra</td>
<td>2%</td>
<td></td>
</tr>
<tr>
<td>CVP</td>
<td>10%</td>
<td>8%</td>
</tr>
<tr>
<td>AVG</td>
<td>5%</td>
<td></td>
</tr>
</tbody>
</table>

SAFARI
Off-Chip Prediction Quality: Defining Metrics

Accuracy

Coverage

Predicted off-chip

Actual off-chip

Predicted and actual off-chip
Off-Chip Prediction Quality: Analysis

Accuracy

Coverage

SPEC06  SPEC17  PARSEC  Ligra  CVP  AVG

HMP  TTP  POPET

SPEC06  SPEC17  PARSEC  Ligra  CVP  AVG

HMP  TTP  POPET

SAFARI
POPET provides off-chip predictions with high-accuracy and high-coverage.
Effect of Different Features

Combination of features provides both higher accuracy and higher coverage than any individual feature.
Are All Features Required? (1)

No single feature individually provides highest prediction accuracy across all workloads.
Are All Features Required? (2)

No single feature individually provides highest prediction coverage across all workloads.
Single-Core Performance

Hermes in combination with Pythia outperforms Pythia alone in every workload category
Single-Core Performance Line Graph

![Graph showing single-core performance with various workloads and benchmarks.](image)
Single-Core Performance Against Prior Predictors

POPET provides higher performance benefit than prior predictors.

Hermes with POPET achieves nearly 90% performance improvement of the Ideal Hermes.
Effect on Stall Cycles

Hermes reduces off-chip load induced stall cycles on average by 16.2% (up-to 51.8%).
Eight-Core Performance

Hermes in combination with Pythia outperforms Pythia alone by 5.1% on average.
Effect of Hermes Request Issue Latency

Hermes in combination with Pythia outperforms Pythia alone even with a **24**-cycle Hermes request issue latency.
Effect of Cache Hierarchy Access Latency

Hermes can provide **even higher performance** benefit in future processors with bigger and slower on-chip caches.

On-chip cache hierarchy access latency (in processor cycles)
Effect of Activation Threshold

With increase in activation threshold

1. Accuracy increases
2. Coverage decreases
Power Overhead

![Bar chart showing runtime dynamic power overhead normalized to the No-prefetching system across different benchmarks and configurations.](Image)
Effect of ROB Size

The diagram shows the geometric mean speedup over the No-prefetching system for different ROB sizes (256, 512, 768, 1024) for three different models: Hermes, Pythia, and Pythia+Hermes. The speedup percentages are 6.7% for ROB size 256, 5.3% for ROB size 1024, and the comparison is highlighted in green.
Effect of LLC Size

![Graph showing the effect of LLC size on performance]

- Hermes
- Pythia
- Hermes + Pythia

Geometric speedup over the No-prefetching system

- LLC size per core (in MB):
  - 3 MB
  - 6 MB
  - 12 MB
  - 24 MB

- Speedup:
  - 2.5%
  - 1.3%
POPET’s **accuracy and coverage increases significantly** in absence of a data prefetcher.
Increase in Main Memory Requests

- **Pythia**: 5.9% increase for Prefetcher, 7.6% for Prefetcher+Hermes
- **Bingo**: 5.8% increase for Prefetcher, 8.6% for Prefetcher+Hermes
- **SPP**: 5.8% increase for Prefetcher, 8.6% for Prefetcher+Hermes
- **MLOP**: 8.6% increase for Prefetcher, 15.6% for Prefetcher+Hermes
- **SMS**: 15.6% increase for Prefetcher+Hermes

*SAFARI*
SIBYL BACKUP
Performance on Unseen Workloads

H&M (H&L) HSS configuration, Sibyl outperforms RNN-HSS and Archivist by 46.1% (54.6%) and 8.5% (44.1%), respectively.
Baseline policies **are ineffective** for many workloads even when compared to Slow-Only.
Performance on Mixed Workloads

- Slow-Only
- CDE
- HPS
- Archivist
- RNN-HSS
- Sibyl\textsubscript{Def}
- Sibyl\textsubscript{Opt}
- Oracle

Performance-Oriented

Cost-Oriented

Normalized Average Request Latency

SAFARI
Performance on Mixed Workloads

Sibyl\textsubscript{Def} outperforms baseline data placement techniques by up to 27.9%
Performance on Mixed Workloads

Sibyl\textsubscript{Def} outperforms baseline data placement techniques by up to 27.9%

Sibyl\textsubscript{Opt} provides 7.2% higher average performance than Sibyl\textsubscript{Def}
Sibyl autonomously decides which features are important to maximize the performance of the running workload
Sensitivity to Fast Storage Capacity

(a) H&M

(b) H&L

Available capacity in fast storage

Normalized Average Request Latency
Explainability Analysis

![Bar chart showing preference for fast storage across different categories like \textit{hm}, \textit{mds}, \textit{prn}, \textit{proj}, etc., with two categories labeled \textit{H&M} and \textit{H&L}.

The chart displays the preference for fast storage for each category, with categories such as \textit{hm}, \textit{mds}, \textit{prn}, \textit{proj}, \textit{proj} (0 and 2), \textit{prxy}, \textit{rsrch}, \textit{src}, \textit{usr}, \textit{wdev}, and \textit{web}. The bars for \textit{H&M} and \textit{H&L} are color-coded, with \textit{H&M} represented by light blue and \textit{H&L} by dark blue. The y-axis represents the preference for fast storage, ranging from 0.00 to 1.00.
Training and Inference Network

• Training and inference networks **allow parallel execution**

• Observation vector as the input

• Produces probability distribution of Q-values

Observation vector
<\(size_t, type_t, intr_t, cnt_t, capt, curr\)>