OSDI '25

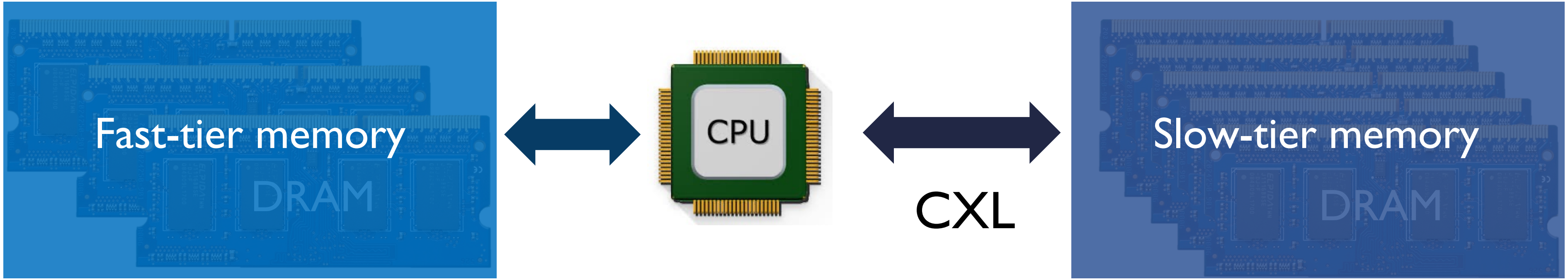
Tiered Memory Management Beyond Hotness

Jinshu Liu, Hamid Hadian, Hanchen Xu, Huaicheng Li

VIRGINIA TECH

Tired Memory Architecture is a New Norm

Growing demand from memory-intensive applications



Limited size

Low memory access latency (~100ns)

Memory expansion

High memory access latency (200~300ns)

Decades of Memory Tiering Research

I. First-touch allocation

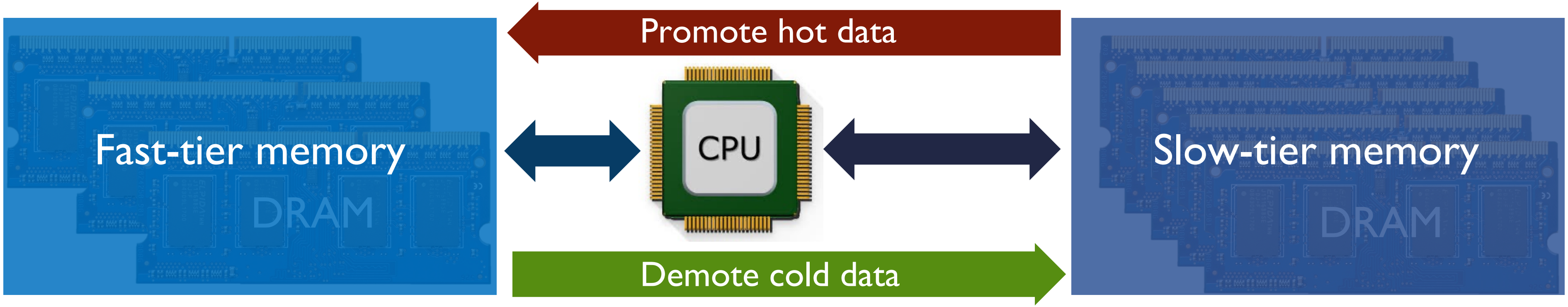
Maximize the usage of fast-tier memory

II. Hotness based data placement

A simple assumption to generalize the affect of memory access

III. Page migration

Migrate hot pages to fast-tier memory



Common Assumptions of State-of-the-Art

Hotness = Performance

High frequency memory access on the slow-tier

?

High performance slowdown

Is hot data always performance-critical?

HeMem [SOSP'21], TPP [ASPLOS'23], MEMTIS [SOSP'23], Nomad [OSDI'24], Memstrata [OSDI'24] Colloid [SOSP'24], NeoMem [Micro'24], Chrono [EuroSys'25], M5 [ASPLOS'25], …

Hotness = Performance ?

High frequency memory access on the slow-tier

High performance slowdown

I. Page migration

Hot but non-performance-critical pages can be promoted with no gains

II. First-touch allocation

Overlook the varying performance contribution from various objects

Our Goal

Three key research questions:

1. Why cannot hotness represent performance?
2. Which metrics should be used to guide tiering?
3. How to apply the new metrics on memory allocation and migration?

Overview

Memory allocation/migration for tiered memory beyond hotness

AOL: Amortized Offcore Latency

- Key factor for accurate performance prediction
- Quantifies the actual impact of memory accesses on performance slowdown

SOAR: Static Object Allocation based on Ranking

- Near-optimal object placement after profiling based on AOL

ALTO: AOL-based Layered Tiering Orchestration

- Regulate page migration for hot but less performance-critical pages

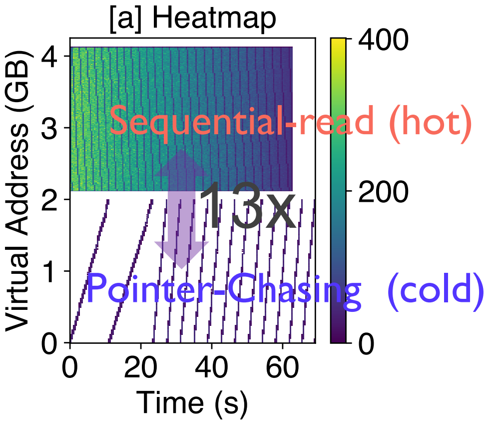
Outline

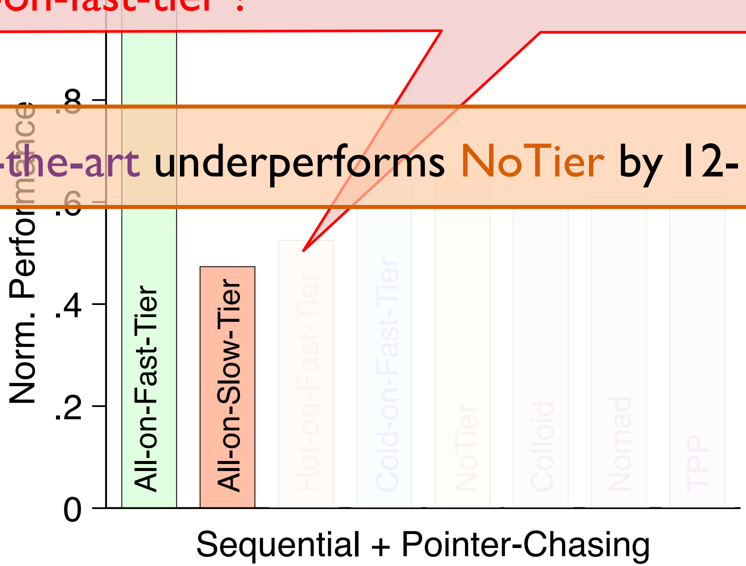
Hotness != Performance

One thread pointer-chasing + one thread sequential read

Hot-on-fast-tier gives worse (34%) performance than Cold-on-fast-tier !

State-of-the-art underperforms NoTier by 12-14%





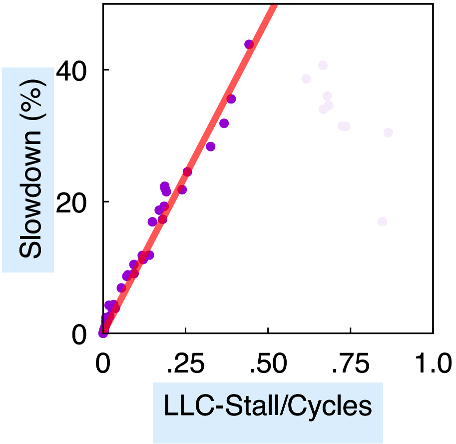
Norm. Performance = $\text{Time}_X / \text{Time}_{\text{All-On-Fast-Tier}}$

Hotness (or access frequency) itself is not enough for guiding memory tiering

Tiered Memory Performance Modeling

I. LLC-stalls for performance prediction

Stalled cycles caused by LLC misses



LLC-stalls/Cycles is measured under all-on-fast-tier

Slowdown = $\text{Time}_{\text{All-on-Slow-Tier}}$ / $\text{Time}_{\text{All-on-Fast-Tier}}$

56 workloads

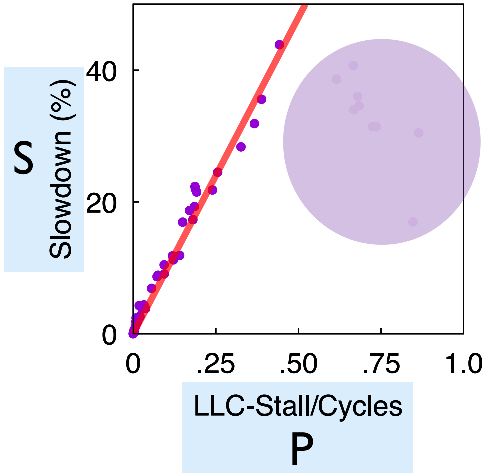
The slowdown for some workloads can be modeled by LLC-stalls

Tiered Memory Performance Modeling

I. LLC-stalls for performance prediction

II. Memory-Level-Parallelism (MLP)

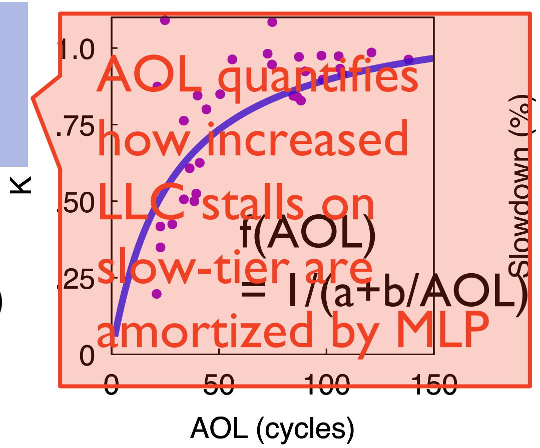
MLP can mask the latency penalty



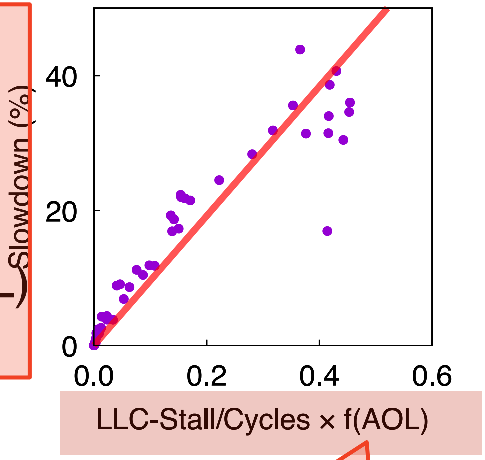
$$\mathrm{AOL} = \frac{\text{Latency}}{\text{MLP}}$$

K = S / P
= f(MLP-metric)

= f(AOL)



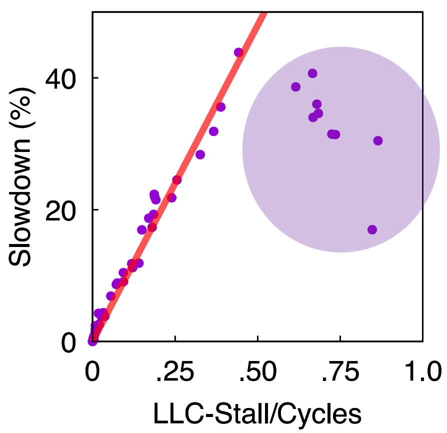
AOL quantifies how increased LLC stalls on slow-tier are amortized by MLP



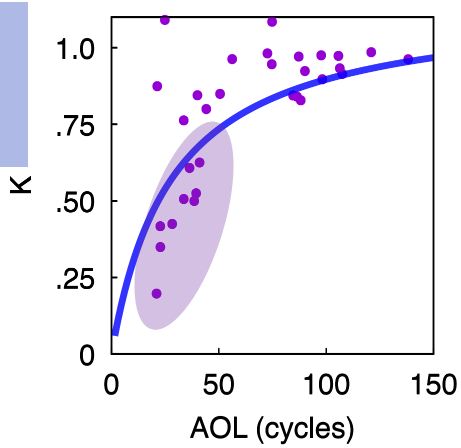
AOL can be used to predict tiered memory performance with high accuracy

Low AOL -> Performance slowdown is amortized by high MLP

High AOL -> Minimal MLP impact



$$\text{AOL} = \frac{\text{Latency}}{\text{MLP}}$$



Outline

Overview

AOL: Amortized Offcore Latency

SOAR: Static Object Allocation based on Ranking

ALTO: AOL-based Layered Tiering Orchestration

Evaluation

Memory Allocation for Tiering

First-touch allocation

- The allocation order is based on the timing requested by runtime
- No awareness of the various performance slowdown from different objects

SOAR: Static Object Allocation based on Ranking

- Assign predicted performance to each object per time period
- Use AOL to determine when the MLP-impact is effective and the degree of its impact
- Adjust slowdown attribution for objects based on AOL (MLP effect)

Offline Analysis Workflow

Object tracking
Object allocation/deallocation → **Object flow**

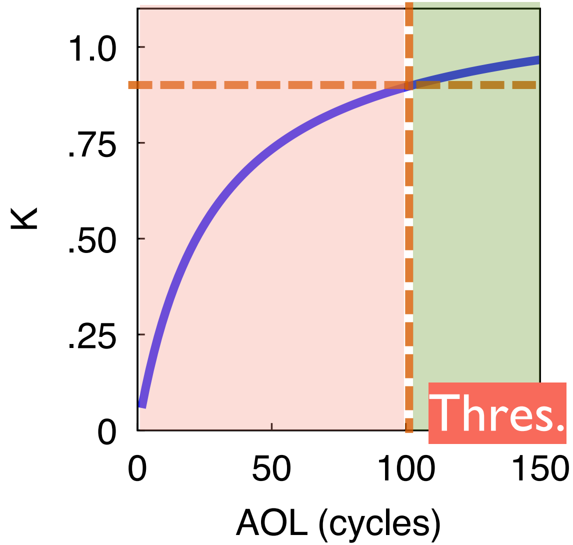
Memory access tracking
PEBS sampling → **Memory access flow**
[time, address]

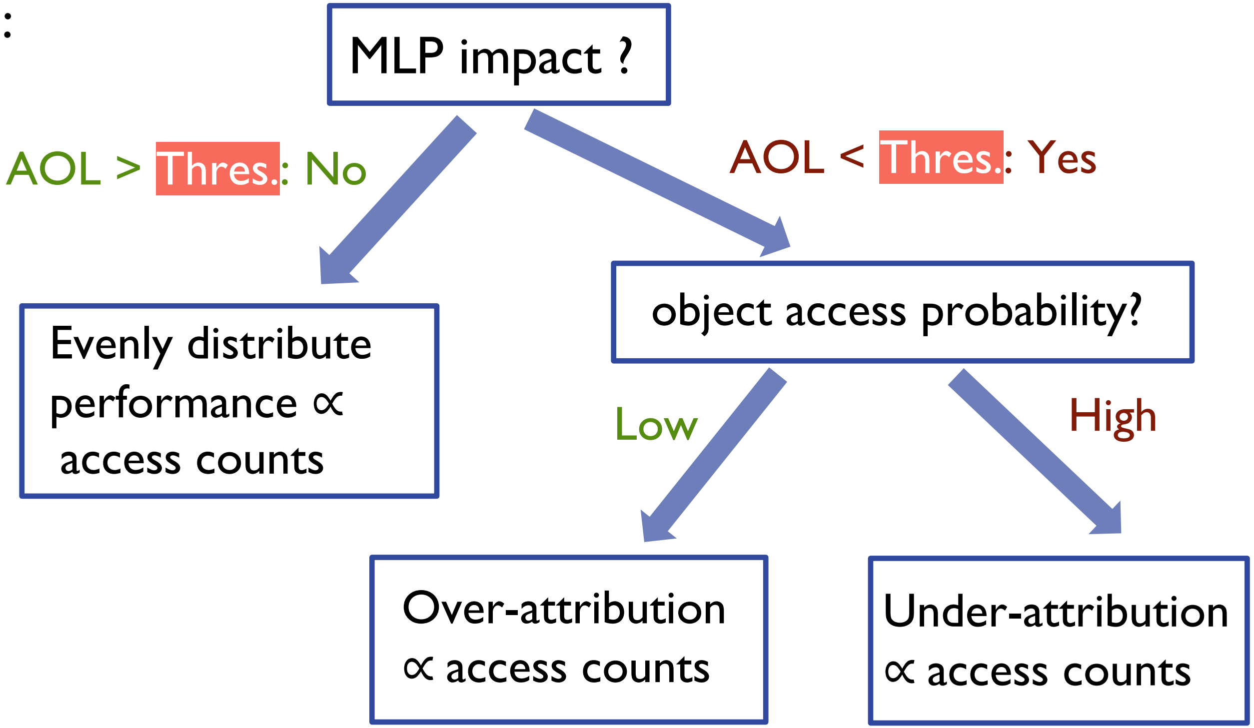
Performance monitoring
LLC-stalls and *AOL* → **Predicted performance flow**
[time, performance, *AOL*]

For each time period, how to attribute the performance to each object with the predicted performance, access counts and AOL?

SOAR: How to Rank Objects?

For each time period t_i, t_{i+1} :





SOAR ranks the objects based on attributed scores. It allocates top ranking objects on the the fast-tier, the rest will be on the slow-tier.

Outline

Page Migration for Tiering

Memory allocation: initial data placement

SOAR: Offline profiling for deciding data allocation between tiers

Page migration: online approach during runtime

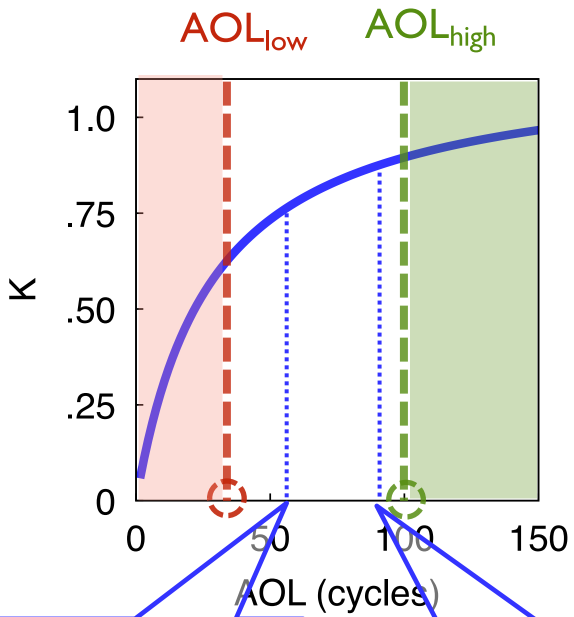
Promote hot but non-performance-critical pages results in minimal or negative performance gain

Low AOL → less performance slowdown caused by high MLP

Use *AOL* to determine when there is unnecessary page migration

ALTO: AOL-based Layered Tiering Orchestration

Reduce unnecessary page migration



e.g., AOL = 60 cycles
25% page promotion

e.g., AOL = 90 cycles
75% page promotion

For each time period t_i, t_{i+1} :

> AOL_{high}	Enable page promotion
[AOL_{low}, AOL_{high}]	Partial page promotion *Scale <- f(AOL)*
< AOL_{low}	Disable page promotion

Outline

Evaluation

Workloads:

- CPU SPEC
- Graph (GAPBS)
- Redis
- GPT-2

Hardware:

- SKX (*slow-tier: coreless-NUMA, 96GB*), SPR (*slow-tier: CXL-DRAM, 128GB*)

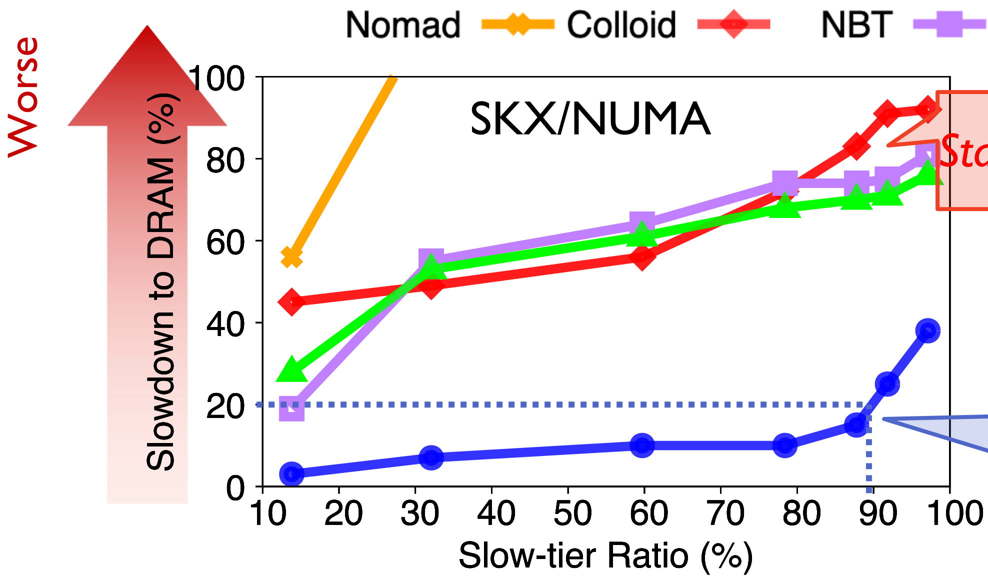
Comparison with:

- NUMA Balancing Tiering (NBT)
- TPP [*ASPLOS'23*]
- Nomad [*OSDI'24*]
- Colloid [*SOSP'24*]

Evaluation

How does SOAR perform under different fast/slow tier ratio?

Example: bc-urand

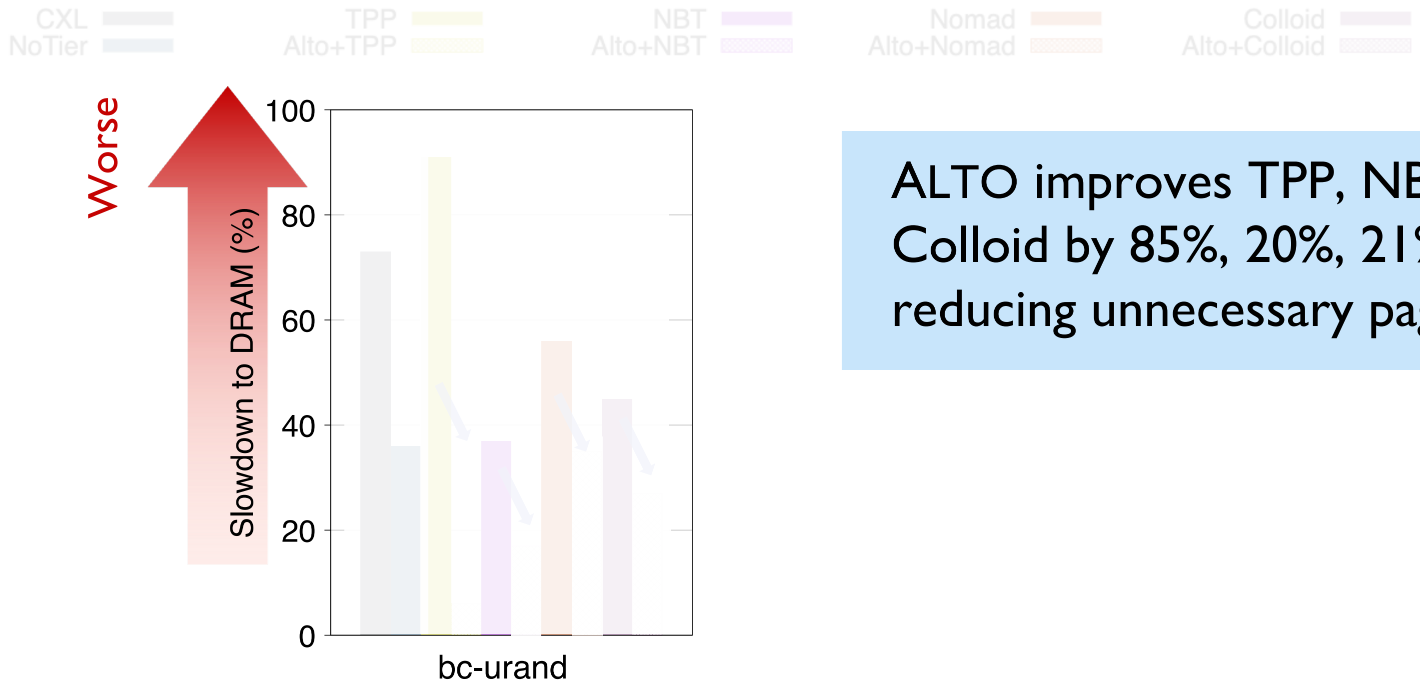


State-of-the-art is similar to or worse than NoTier

SOAR maintains <20% slowdown with up to 90% slow-tier memory

More in the paper: SPR/CXL

How does ALTO perform?



ALTO improves TPP, NBT, Nomad, and Colloid by 85%, 20%, 21%, and 18%, by reducing unnecessary page promotions

More in the paper: other workloads under different setups

Summary

Hotness != Performance

AOL for quantifying MLP impact on slowdown

SOAR: Profile-guided static allocation policy

Identify performance-critical objects

ALTO: Page migration regulation policy

Reduce unnecessary page promotions

Paper

Code

https://github.com/MoatLab/SoarAlto

Thank you! Questions?

Tiered Memory Management Beyond Hotness

Jinshu Liu Hamid Hadian Hanchen Xu Huaicheng Li

Virginia Tech

Abstract

Tiered memory systems often rely on access frequency ("hotness") to guide data placement. However, hot data is not always performance-critical, limiting the effectiveness of hotness-based policies. We introduce amortized offcore latency (AOL), a novel metric that precisely captures the true performance impact of memory accesses by accounting for memory access latency and memory-level parallelism (MLP). Leveraging AOL, we present two powerful tiering mechanisms: SOAR, a profile-guided allocation policy that places objects based on their performance contribution, and ALTO, a lightweight page migration regulation policy to eliminate unnecessary migrations. SOAR and ALTO outperform four state-of-the-art tiering designs across a diverse set of workloads by up to 12.4×, while underperforming in a few cases by no more than 3%.

1 Introduction

Driven by the growing demands of memory-intensive workloads, such as graph processing and machine learning, tiered memory architectures that integrate a **fast-tier** (*e.g.*, DRAM) and **slow-tier** (*e.g.*, CXL memory) are becoming standard in cloud datacenters [1–5]. While this approach improves memory capacity scaling, it also introduces significant performance challenges. Effective data tiering is critical to mitigating the 2–3× performance disparity between tiers [6–12].

Existing tiering designs are grounded in the assumption that frequently-accessed ("hot") data is more performance-critical than cold data and should reside in the fast-tier. Thus, tiered memory management primarily focuses on hotness tracking, memory allocation, and migration policies to detect, allocate, and relocate hot data across tiers efficiently [4, 13–29].

We argue that hot data is not always performance-critical and can reside in the slow-tier without degrading performance (§2.1). In modern out-of-order CPU designs, latency mitigation techniques, such as memory-level parallelism (**MLP**), obscure the true cost of memory accesses [13, 30–32]. Not all memory accesses contribute equally to performance (vary by 4×, §3); overlapping requests (high MLP) often mask slow-tier latency penalties, leading to less pronounced slowdowns.

Although MLP is a well-established concept within the architecture community [30–32], its implications for tiered memory management have been largely overlooked. Prior classification efforts across objects, pages, and data structures [13, 19, 33, 34] often implicitly reflect the effects of MLP through coarse heuristics or indirect indicators of memory access costs. However, they do not *explicitly* model or quantify MLP impact. What remains missing is a principled, accurate, and MLP-aware performance metric that enables more effective, performance-driven tiering policies across online and offline scenarios, and generalizes to diverse workloads.

Existing tiering systems also suffer from heavyweight and imprecise hotness sampling and page migration mechanisms. Two key limitations are prevalent [1, 4, 16, 17, 19, 21–24, 35]: **(a)** *Suboptimal data placement.* Existing coarse-grained allocation policies prioritize fast-tier placement for newly allocated data, but under fast-tier pressure, performance-critical data is often displaced to the slow-tier, necessitating costly migrations later to correct the placement errors; **(b)** *Excessive migration overhead.* Existing systems often employ aggressive migration policies, incurring substantial overhead by frequently relocating non-critical pages. This overhead can erode or negate the performance benefits of tiering (§2.1).

We propose **Amortized Offcore Latency** (**AOL**), a novel performance metric that accurately quantifies the performance impact of memory accesses by integrating memory latency and MLP. While latency measures the impact of individual memory requests, it does not capture the latency-masking effects of MLP. By considering both factors, AOL, expressed as "`Latency/MLP`" combined with CPU stalls, offers a more precise representation of the true performance contribution of memory accesses (validated across 56 workloads, §3).

We leverage AOL to redesign memory allocation and migration policies, introducing two novel tiering mechanisms: a static memory allocation policy, **SOAR**, and a dynamic page migration regulation policy, **ALTO**. SOAR employs AOL-based profiling to rank objects by assessing their accumulative contributions to application performance. High-ranking objects are placed in the fast-tier, achieving near-optimal placement while eliminating runtime migration overhead. ALTO adaptively regulates page migrations based on AOL, ensuring that only performance-critical pages are promoted, regardless of their hotness. ALTO seamlessly integrates with four representative tiering systems with minimal code changes, including TPP [4], Nomad [22], Linux NUMA Balancing Tiering (NBT) [36–38], and Colloid [23].

We evaluate SOAR and ALTO across a range of realistic