TALUS: A SIMPLE WAY TO REMOVE PERFORMANCE CLIFFS IN CACHES
CACHES HAVE PERFORMANCE CLIFFS

- 32 MB array
- 4 MB LLC
- 100% miss

- 16 MB LLC
- 100% miss

- 31 MB LLC
- 100% miss

- 32 MB LLC
- 0% miss

Diagram showing miss rate against cache size with a peak at 32 MB.
CLIFFS ARE A PROBLEM

Cliffs are wasteful
Cliffs cause annoying performance bugs
Cliffs complicate cache partitioning
- NP-hard problem
PRIOR WORK: HIGH-PERFORMANCE REPLACEMENT VS. CACHE PARTITIONING

Individual apps: High-performance replacement
- E.g., RRIP [ISCA’10]

Shared caches: Cache partitioning
- E.g., UCP [MICRO’06]

For shared caches, partitioning > replacement
- Both in performance & flexibility

Partitioning is hard to use with high-performance replacement!

Can partitioning help individual apps?

32 MB
libquantum #1
0% miss

Miss rate

LRU
SRRIP
DRRIP

Cache size
IN THIS TALK WE WILL...

Give a simple technique to eliminate cliffs (Talus)
  - Talus partitions within a single access stream

Prove it works under simple assumptions
  - Agnostic to app or replacement policy

No cliffs → Simpler cache partitioning

Talus combines the benefits of high-performance replacement and partitioning
ROAD MAP

Talus example

Theory

Implementation

Evaluation
Talus uses miss curves

Cliffs occur under a variety of access pattern and replacement policies.

Talus works on miss curves only;
Talus is agnostic to app and replacement policy.
TALUS EXAMPLE
TALUS EXAMPLE
TALUS EXAMPLE

![Graph showing cache misses vs. cache size with a target of 6 MPKI at 4MB]
TALUS EXAMPLE

![Graph showing cache size vs. misses (MPKI) for different cache sizes: 2MB and 5MB. The graph compares Original and Talus performance, with Talus showing better performance.]
(HYPOTHETICAL) BASELINE CACHE AT 2 MB

Accesses (APKI): 24

Misses (MPKI): 12
(HYPOTHETICAL) BASELINE CACHE AT 2 MB

Accesses (APKI): 24
Misses (MPKI): 12

2/3 MB

4/3 MB
(HYPOTHETICAL) BASELINE CACHE AT 5 MB

Accesses (APKI) 24

Misses (MPKI) 3

Graph showing cache size (MB) vs. misses (MPKI) for original and Talus.
(HYPOTHETICAL) BASELINE CACHE AT 5 MB

Accesses (APKI)

- 24
- 16
- 8

Misses (MPKI)

- 3
- 2
- 1

5/3 MB

10/3 MB

Graph showing Misses (MPKI) versus Cache size (MB)
TALUS AT 4 MB

Combine hypothetical baseline 2 MB & 5 MB
TALUS AT 4 MB

Spread accesses disproportionately across partitions to match baselines

Accesses (APKI) 24

Misses (MPKI) ??

2/3 MB

10/3 MB
EXAMPLE SUMMARY

Talus avoids cliffs by combining efficient cache sizes of baseline

Does not know or care about app or replacement details

- Just needs miss curve!

Nothing special about set partitioning; Talus works on other partitioning techniques

But how to choose partition configuration?
ROAD MAP

Talus example

Theory
- Proof sketch
- Talus vs prior policies

Implementation

Evaluation
GOAL: **CONVEXITY AVOIDS CLIFFS**

Convex miss curves do not have cliffs
Talus divides the cache (of size $s$) into shadow partitions, invisible to software.

Talus ensures convexity under simple assumptions.
ASSUMPTIONS

Miss curves are stable (eg, across tens of milliseconds)

Cache size is the dominant factor in miss rate (ie, not associativity)

Pseudo-random sampling of an access stream yields a statistically self-similar stream

These assumptions are implicit in prior work (see paper)
SAMPLING SCALES THE MISS CURVE

APKI

2/3 MB

MPKI

5/3 MB

8

24

16

4

12

8

1

2/3 MB

Misses (MPKI)

Cache size (MB)

Original
Sample rate $\rho = 1/3$
SHADOW PARTITIONING INTERPOLATES MPKI OF THE ORIGINAL MISS CURVE
TALUS GUARANTEES CONVEXITY

Just interpolate the convex hull of the original miss curve!
Miss curve scaling:

\[ m'(s') = \rho \cdot m\left(\frac{s'}{\rho}\right) \]

Shadow partitioned miss rate:

\[ m_{\text{shadow}}(s) = \rho \cdot m\left(\frac{s_1}{\rho}\right) + (1 - \rho) \cdot m\left(\frac{s - s_1}{1 - \rho}\right) \]

How to interpolate between \( \alpha \) and \( \beta \):

\[ \rho = \frac{\beta - s}{\beta - \alpha}, \quad s_1 = \rho \alpha \]
ROAD MAP

Motivation

Talus example

Theory
  • Proof sketch
  • Talus vs prior policies

Implementation

Evaluation
PRIOR TECHNIQUE: BYPASSING

Bypassing is a common replacement technique to avoid thrashing
- E.g., BIP [ISCA’07] bypasses 31/32 accesses

We compute optimal bypassing rate from miss curve

Bypassing handles some kinds of cliffs, but not all

⇒ Talus outperforms bypassing on some access patterns
BYPASSING PRODUCES COMPETING EFFECTS
BYPASSING PRODUCES COMPETING EFFECTS

- Bypassing reduces misses for sampled accesses

![Graph showing the effect of bypassing on cache misses vs. cache size. The graph compares the original curve with the curve for not bypassed (sampled) accesses.]
BYPASSING PRODUCES COMPETING EFFECTS

- Bypassing reduces misses for sampled accesses
- ...But adds misses for bypassed accesses

See paper for details!
TALUS VS BYPASSING

Talus reduces miss rate

Talus is convex
- i.e., avoids cliffs!
ROAD MAP

Talus example

Theory

Implementation

Evaluation
CONVENTIONAL PARTITIONED CACHE

Partitioning Algorithm

Eg, UCP [MICRO’06]
EFFICIENT TALUS IMPLEMENTATION

Hardware
- Miss curve monitors
- Partitioned Cache
- Partition

Software
- Pre-processing
- Partitioning Algorithm
- Post-processing

Talus additions
EFFICIENT TALUS IMPLEMENTATION

Hardware

Miss curve monitors

Partitioned Cache

Partition

Software

Miss curves

Pre-processing

Convex hulls

Partitioning Algorithm

Desired allocations

Post-processing

Shadow partition sizes & sampling rate
EFFICIENT TALUS IMPLEMENTATION

- **Hardware**
  - Miss curve monitors
  - Partitioned Cache
  - Partition
    - $\alpha$
    - $\beta$

- **Software**
  - Pre-processing
  - Convex hulls
  - Partitioning Algorithm
  - Desired allocations
  - Post-processing
  - Shadow partition sizes & sampling rate

- Miss curves
EFFICIENT TALUS IMPLEMENTATION

Address 0x074705

H3 hash

Limit Reg

Limit Reg

Partitioned Cache
TALUS IMPOSES LOW OVERHEADS

Computing convex hulls is cheap: $O(N)$

Computing shadow partition sizes is cheap: $O(1)$

*Talus reduces software overheads by making simple algorithms perform well*

Shadow partitioning is cheap: similar monitors to prior work (see paper), 1 bit per tag, 8 bits per partition, simple hash function

*Talus improves cache performance and adds <1% state*
EVALUATION CLAIMS

We compare Talus to high-performance replacement policies and partitioning schemes.

Talus is convex in practice.

Single-program: Talus gets similar performance to prior replacement policies.

Multi-program: Talus greatly simplifies cache partitioning and slightly outperforms prior, complex partitioning algorithms.

Talus combines the benefits of high-performance replacement and partitioning.
METHODOLOGY

Evaluate 1- and 8-core system similar to Silvermont on zsim
- See paper for details

Individual SPEC CPU2006 benchmarks + random mixes

Talk only shows Talus on LRU with Vantage partitioning (Talus +V/LRU)
EVALUATION: SINGLE-THREADED

- **xalancbmk**
  - MPKI vs. LLC Size

- **IBM**
  - MPKI vs. LLC Size (MB)

- **perlbench**
  - MPKI vs. LLC Size (MB)

- **mcf**
  - MPKI vs. LLC Size (MB)
EVALUATION: SINGLE-THREADED

Talus is convex in practice!
EVALUATION: SINGLE-THREADED

PDP performs similarly but is not always convex.
RRIP policies avoid most cliffs, but their performance depends on access pattern.
RRIP policies avoid most cliffs, but their performance depends on access pattern.
GMEAN IPC IMPROVEMENT VS LRU

Talus on LRU gets similar speedups to prior policies.
MULTI-PROGRAMMED PERFORMANCE

Talus is convex $\rightarrow$ naïve hill climbing yields large performance gains.
Hill climbing alone does not improve performance much.
MULTI-PROGRAMMED PERFORMANCE

Lookahead is close to Talus, but more expensive
Partitioning techniques outperform high-performance policies on shared caches.
TALUS SIMPLIFIES PARTITIONING ALGORITHMS AND REDUCES OVERHEADS

Efficient alternatives to Lookahead add significant complexity!
Talus with fair (equal-sized) partitions decreases execution time without degrading fairness.

See paper for other apps & schemes!
MORE CONTENT IN PAPER!

Detailed proofs

Prove optimal replacement is convex

Evaluation:
- Talus works on way partitioning
- Talus works with SRRIP
- More benchmarks
- Talus works with pre-fetching and multi-threading
Talus avoids cliffs and ensures convexity
- Proven under simple assumptions
- Verified by experiment

Analysis of shadow partitioning shows advantages vs bypassing

Talus improves performance and simplifies cache partitioning

Talus combines the benefits of high-performance replacement and partitioning