CACHES HAVE PERFORMANCE CLIFFS

libquantum

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 MB LLC</td>
<td>0%</td>
</tr>
<tr>
<td>31 MB LLC</td>
<td>100%</td>
</tr>
<tr>
<td>28 MB LLC</td>
<td>100%</td>
</tr>
<tr>
<td>25 MB LLC</td>
<td>100%</td>
</tr>
<tr>
<td>20 MB LLC</td>
<td>100%</td>
</tr>
<tr>
<td>16 MB LLC</td>
<td>100%</td>
</tr>
<tr>
<td>10 MB LLC</td>
<td>100%</td>
</tr>
<tr>
<td>8 MB LLC</td>
<td>100%</td>
</tr>
<tr>
<td>4 MB LLC</td>
<td>100%</td>
</tr>
<tr>
<td>2 MB LLC</td>
<td>100%</td>
</tr>
</tbody>
</table>

Cache size vs. Miss rate graph showing a sharp decline in miss rate for cache sizes above 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?

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Miss Rate</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 MB</td>
<td>0%</td>
<td>libquantum #1</td>
</tr>
<tr>
<td>16 MB</td>
<td>100%</td>
<td>libquantum #2</td>
</tr>
<tr>
<td>32 MB</td>
<td>0%</td>
<td>libquantum #1</td>
</tr>
</tbody>
</table>
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 $\Rightarrow$ 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

![Graph showing cache size vs. misses (MPKI)]
TALUS EXAMPLE

![Chart showing cache size vs. misses (MPKI) with different cache sizes and target at 6 MPKI @ 4MB]
TALUS EXAMPLE

![Graph showing cache size vs. misses (MPKI) for Original and Talus with 2MB and 5MB cache sizes. The graph includes a target of 6 MPKI at 4MB.]
(HYPOTHETICAL) BASELINE CACHE AT 2 MB

Accesses (APKI): 24

Misses (MPKI): 12

Sets x Ways

Graph: Misses (MPKI) vs. Cache size (MB)
(HYPOTHETICAL) BASELINE CACHE AT 5 MB

Accesses (APKI): 24

Misses (MPKI): 3
(HYPOTHETICAL) BASELINE CACHE AT 5 MB

Accesses (APKI)

24

8

16

Misses (MPKI)

3

1

2

5/3 MB

10/3 MB
**TALUS AT 4 MB**

Combine hypothetical baseline 2 MB & 5 MB

<table>
<thead>
<tr>
<th></th>
<th>2/3 MB</th>
<th>4/3 MB</th>
</tr>
</thead>
</table>

**Graph:**
- 5/3 MB
- 10/3 MB
TALUS AT 4 MB

Spread accesses *disproportionally* across partitions to match baselines

<table>
<thead>
<tr>
<th>Accesses (APKI)</th>
<th>Misses (MPKI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
</tr>
</tbody>
</table>

Diagram showing 2/3 MB and 10/3 MB partitions with arrows indicating spread across partitions.
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.

$\alpha$: $s_1$ lines

$\beta$: $s - s_1$ lines

Total misses

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
SHADOW PARTITIONING INTERPOLATES MPKI OF THE ORIGINAL MISS CURVE
TALUS GUARANTEES CONVEXITY

Just interpolate the convex hull of the original miss curve!
THERE’S MATH!

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:
- Y-axis: Misses (MPKI)
- X-axis: Cache size (MB)
- Two curves:
  - Original
  - Not bypassed (sampled)
BYPASSING PRODUCE 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
EFFICIENT TALUS IMPLEMENTATION

Hardware:
- Partitioned Cache
- Miss curve monitors

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

Talus additions
EFFICIENT TALUS IMPLEMENTATION

Partitioned Cache

Hardware

Miss curve monitors

Software

Pre-processing

Convex hulls

Partitioning Algorithm

Desired allocations

Post-processing

Shadow partition sizes & sampling rate
EFFICIENT TALUS IMPLEMENTATION

Hardware

Partitioned Cache

Miss curve monitors

Partitioning Algorithm

Pre-processing

Convex hulls

Desired allocations

Post-processing

Software

β

α

Shadow partition sizes & sampling rate
EFFICIENT TALUS IMPLEMENTATION
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 (MB)
  - MPKI decreases with increasing LLC size.

- **perlbench**
  - MPKI vs. LLC Size (MB)
  - MPKI decreases with increasing LLC size.

- **IBM**
  - MPKI vs. LLC Size (MB)
  - MPKI decreases with increasing LLC size.

- **mcf**
  - MPKI vs. LLC Size (MB)
  - MPKI decreases with increasing LLC size.
EVALUATION: SINGLE-THREADED

Talus is convex in practice!
EVALUATION: SINGLE-THREADED

PDP performs similarly but is not always convex

- xalancbmk
- perlbench
- PDP
- LRU
- Talus +V/LRU
RRIP policies avoid most cliffs, but their performance depends on access pattern.
**EVALUATION: SINGLE-THREADED**

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

Weighted Speedup

Workload Mix

LRU

0
20
40
60
80
100

0.0
1.0
1.1
1.2
1.3
1.4

48
Talus is convex \(\Rightarrow\) naïve hill climbing yields large performance gains.
Hill climbing alone does not improve performance much.
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!
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
THANK YOU!

- 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*