TLC: A Tag-Less Cache for Reducing Dynamic First Level Cache Energy

Andreas Sembrant, Erik Hagersten, David Black-Schaffer
Uppsala University, Sweden
Motivation: Standard VIPT L1

Problems:
- What do we do?
  - Read the TLB, 8 tags and data from 8 cache lines
- What do we need?
  - Data from one cache line

Can we do better?
Tag-based → Tag-less Cache

CPU

Virtual Address

TLB

Physical Address

L1

Read 8 tags and 8 lines

Use 1 line
Tag-based $\rightarrow$ Tag-less Cache

CPU

Virtual Address

eTLB

L1

Read 8 tags and 8 lines

Use 1 line

hit/miss

valid

3 way index bits

VTAG | PA ...

Cache-line Location Table

Page offset in Virtual Address
Tag-based $\rightarrow$ Tag-less Cache

CPU

Virtual Address

eTLB

No Tags

L1

Data

Use 1 line

hit/miss

valid

3 way index bits

VTAG  PA ...

Cache-line Location Table

Page offset in Virtual Address

Read 8 lines

Use 1 line

PA ...
Tag-based → Tag-less Cache

- **CPU**
- **Virtual Address**
  - **Page offset**
  - **Way**
- **Data**
- **eTLB**
- **hit/miss**
- **Valid**
  - **3 way index bits**
  - **1 110**
- **VTAG PA ...**
- **Cache-line Location Table**
- **Page offset in Virtual Address**
- **Use 1 line**
- **Read 1 line**

**Key Points**
- **Virtual Address**
- **Page offset**
- **Way**
- **Valid**
- **3 way index bits**
- **Cache-line Location Table**
- **Page offset in Virtual Address**
- **Use 1 line**
- **Read 1 line**
Tag-based → Tag-less Cache

- CPU
- Virtual Address
- page offset
- way
- valid
- 3 way index bits
- VTAG
- PA ...
- Cache-line Location Table
- Use 1 line

Better SRAM Shape (faster, less energy)

Page offset in Virtual Address

Use 1 line

Better SRAM Shape (faster, less energy)
Tag-less Cache

- eTLB
- Virtual Address
- Page offset
- Valid
- VTAG
- PA
- Cache-line Location Table
- Page offset in Virtual Address
- 3 way index bits
- Only 1 line
- Better SRAM Shape
- Filter out misses
- No tags
- Use 1 line
- Hit/miss
- 1 110
Sanity Check (Bitcount)

L1: 32kB, 8-way  TLB: 64 entry, 8-way

- **Storage / Area:** +1%
  - + Cache line location data in TLB
  - No cache tags

- **Read hit energy:** −64%
  - Don’t need the physical address from the TLB on hits
  - No cache tags
  - Only read the correct cache line

Slightly more storage (+1%), but many fewer bit reads (-64%)
Using an eTLB, we can:

1. Eliminate extra data-array reads
   - by determining the correct way from the TLB

2. Eliminate the tag-array
   - by avoiding tag comparisons

3. Filter out cache misses
   - by checking in the eTLB

4. Faster data-array SRAM shape
   - by only reading one way
But …

- We need new ways to handle:
  1. Cache-line replacement
  2. eTLB replacement
1. Cache-line Replacement

To evict this line:
1. Invalidate eTLB entry
2. Move it to L2

No tags → No way to find the right eTLB entry
1. **Cache-line Replacement**

- **eTLB** as an extension to the L1 cache
  - **VTAG** and **PA** fields
  - Page offset to Cache-line Location Table

---

**To evict this line:**
1. Invalidate eTLB entry
2. Move it to L2

---

**Move data to L2 using the PA in the eTLB.**

---

**Extend each cache line with an eTLB Backpointer to the page it belongs to**

---

1. **Backpointer much smaller than a tag (6 vs. ~28bits)**
2. **Only used during replacement (infrequent)**
But ...

- To make this work we need to handle:
  1. Cache-line replacement
  2. eTLB replacement

**Problem:** The only way to locate a cache line is via the eTLB.
- We can not keep data in the cache without an eTLB entry.

**Solution:** Need to flush the evicted eTLB entry’s cache lines on eTLB replacement.
2. eTLB Replacement

To evict this page:
1. Flush page’s cache lines
2. Invalidate eTLB entry

Flush data to L2 using the PA in the eTLB.

Seems expensive?
This is rare!

1. We do this rarely (SPEC2006):
   - L1 miss ratio ~3% (use backpointers)
   - eTLB miss ratio ~0.3% (need to flush lines from a page)

2. Off critical path (read hit)
   - Optimize for the common case (hit) 97%
     - Accesses minimum data on hits
     - Fast on hits
     - Low energy on hits
Improving TLC

1. eTLB Replacement Policy
   - Minimize forced cache line evictions

2. uPages (Sparse Data)
   - Minimize area when increasing eTLB entries

3. uPage Banking
   - Faster eTLB $\rightarrow$ Data-array communication
1. eTLB Replacement Policy

**Problem:** We can not keep data in the cache without an eTLB entry.

Evict the eTLB entry with least data in the cache.
2. uPages (Sparse Data)

Some applications with sparse access patterns need more eTLB entries.

- **eTLB**
- **Cache-line Location Table**
- **Micro pages (< 4kB)**

Sparse data is limited to the number of eTLB entries (\(< \# \text{L1 cache lines}\))

- + More eTLB entries
  - Larger area
  - Low CLT utilization

- + More eTLB entries
  - More tags
  + Smaller CLT area
  + Higher utilization

In paper: Large Pages
3. uPage Banking

Use uPages to improve the eTLB → Data-array communication

Virtual Address

Page offset

H-tree

eTLB

Bank on page offset (uPage number bits)

Faster + less energy
eTLB → Data-array communication
Improving TLC

1. eTLB Replacement Policy
   - Minimize forced cache line evictions

2. uPages (Sparse Data)
   - Minimize area when increasing eTLB entries

3. uPage Banking
   - Faster eTLB $\rightarrow$ Data-array communication
Compatibility

- Drop in replacement to a standard VIPT L1 cache
  - No software modification
  - No processor core modification

- Need to handle:
  1. Coherency
  2. Synonyms
1. Coherency

- L1
- L2
- eTLB
- BLT
- Reverse Address Translation
- Virtually Indexed
- Mutation Inclusive
- Filter request on BLT misses
- ~20% of eTLB size (only tag + bp)

Coherency Request (Physical Address)
Compatibility

- Drop in replacement to a standard VIPT L1 cache
  - No software modification
  - No processor core modification

- Need to handle:
  1. Coherency
  2. Synonyms
2. Synonyms

No synonyms
To reduce number of pointers

The data does not move
Only the location info in the eTLB

Want to install new page
1. Detect and remove synonym
2. Install new page

L1

eTLB

BLT

Virtual

Physical

L2 TLB

L2
Compatibility

- Drop in replacement to a standard VIPT L1 cache
  - No software modification
  - No processor core modification

- Need to handle:
  1. Coherency (BLT to handle physical $\rightarrow$ virtual translation)
  2. Synonyms (No synonyms to reduce number of pointers)
Sanity Check 2 (Bitcount)

**L1:** 32kB, 8-way  
**TLB:** 64 entry, 8-way

### Storage / Area:
- + Cache line location data in TLB
  - No cache tags
- + eTLB backpointers
- + BLT
- + (64 → 256 entry 1kB uPage) eTLB

| +1% | +1% | +0.6% | +2.6% | +8.4% | +11% |

- Trade-off between area and optimizing for sparse data

### Read hit energy: −64%  
- Don’t need the physical address from the TLB on hits
- No change (Backpointer + BLT is off read hit path)
- Only read the correct cache line

More area (2.6–11%), but same energy reduction (-64%)
Access Time and Read Energy

- CACTI 6.5 cache model

- Here: 3 configurations
  - Baseline (Standard VIPT cache)
    - 64 entry TLB
  - Basic TLC
    - 64 entry eTLB
  - Optimized TLC
    - Smart data-aware replacement, μPages, μPage-banking
    - 512 entry uPage eTLB

- 32kB L1 cache

More configurations in the paper
Results: Read Hit

Access Time (ns)

Baseline (VIP) Basic TLC Optimized TLC

Dynamic read energy (pJ)

Baseline (VIP) Basic TLC Optimized TLC

Slower due to eTLB

Less energy since no tags and only reading the correct way.

uPage-bank
H-Tree
Data

Tags
eTLB

39%
142%
Results: Read Hit

Access Time (ns)

Dynamic read energy (pJ)

Faster due to uPage-banking

But significant H-tree energy

uPage-bank H-Tree

Data

Tags eTLB

65% of the energy
Results: Read Hit

Access Time (ns)

- Baseline (VIPT)
- Basic TLC
- Optimized TLC

Dynamic read energy (pJ)

- Baseline (VIPT)
- Basic TLC
- Optimized TLC

uPage-bank
H-Tree

Comparable to VIPT

Much lower energy

Data

Tags
eTLB

+14%
Runtime Performance and Energy

- Gem5 simulator
- SPEC2006
- Simulated SPEC’s 307 longest phases
- Sorted the simulation points (phases) in ascending order (e.g. performance or energy)
Relative Performance (%)

TLC Slower

Basic TLC

Simulation Points in gcc

Optimized TLC uPages / repl. / etc.

Why slower?
- evict lines due to eTLB replacement

TLC Faster

Why faster?
- proactive line eviction with pages
- Higher TLB granularity

Sorted Simulation Points
H-tree high: optimized for latency.

Reduce total L1D dynamic energy by 78% without hurting performance.
Related Work

- **Way-prediction**
  - We eliminate tags
  - We always know the correct way from the eTLB
  - More in paper

- **Coarse-grained cache line tracking**
  - Region-tracker (Zebchuk) and Sector-cache (Seznec)
  - Focus on tag-area, coherency and L2/L3
  - We focus on L1 caches and dynamic energy
More in paper

- **eTLB page prediction**
  - Reduce eTLB energy even more

- **uPage optimizations**
  - Macro-page preloading to reduce TLB miss ratio

- **Super pages**
  - Split L2 TLB pages on load

- **Synonym optimizations**
  - Reduce cost of updating eTLB backpointers

- **More configurations**
  - Simulation results for different CPUs (ROB etc.)
  - More cache sizes (16kB, 32kB and 64kB)
  - VIPT with phased lookups
Summary

Reduce total L1D dynamic energy by 78% without hurting performance
Questions?