The Direct-to-Data (D2D) Cache: Navigating the Cache Hierarchy with a Single Lookup

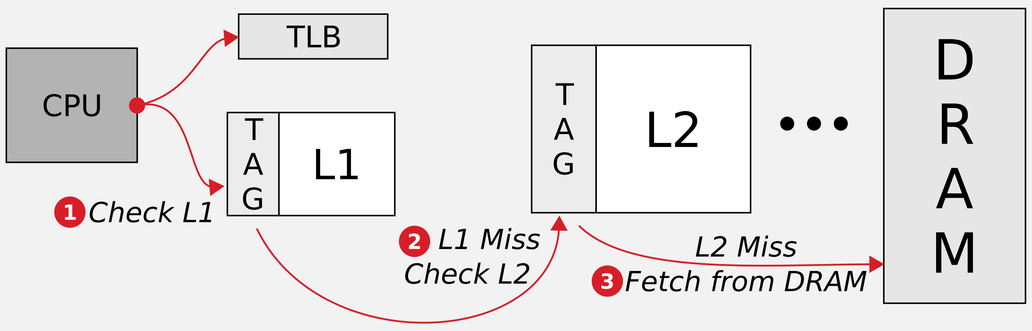
Andreas Sembrant, Erik Hagersten, David Black-Schaffer

UPMARC Uppsala Programming for Multicore Architectures Research Center

UPPSALA UNIVERSITET

1 Problem: Traditional Caches

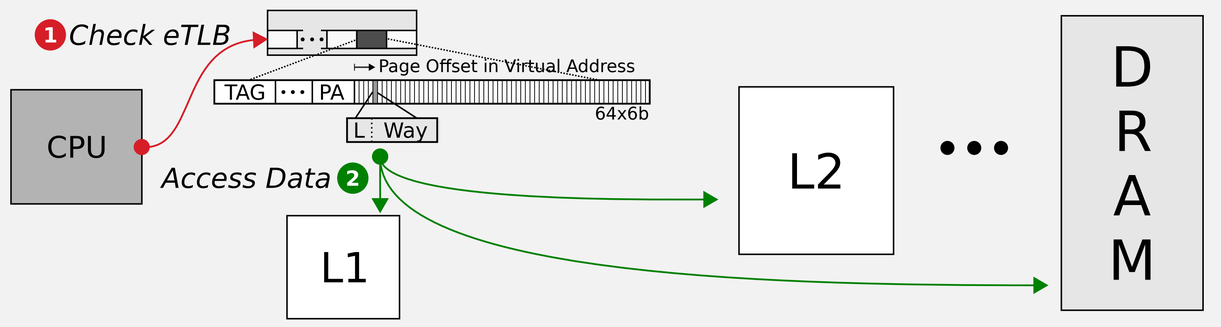
*Traditional cache hierarchies waste **time** and **energy** by sequentially searching for data.*



The cache hierarchy in modern processors consumes 12-45% of the core power. Typically, each cache level is unaware of what data the other levels contain. A request must therefore traverse the cache hierarchy, level-by-level, until the data is found. This wastes energy by probing levels that do not contain the data, and increases access latency for every level examined.

2 Solution: Go Directly to Data

D2D extends the TLB with per cache line level and way info.
→One lookup in the eTLB tells us where the data is located.

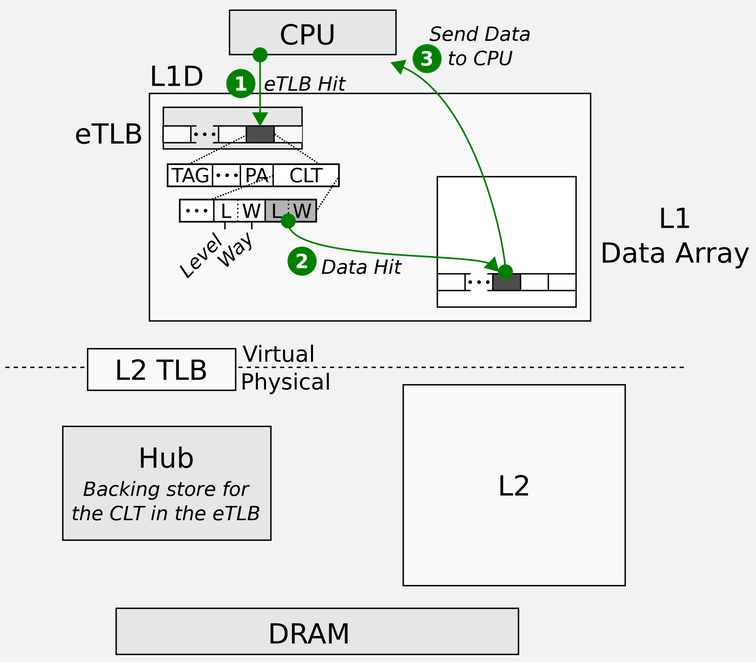


To address the inefficiency of having to traverse the whole hierarchy to find data, we propose the Direct-to-Data (D2D) cache design. This design uses a single lookup to an extended L1 TLB (the eTLB) to identify the cache level and way for the desired data. As a result we can eliminate the energy and latency overhead of traversing the cache hierarchy and the cost of tag comparisons. Because the D2D design removes the need for tag lookups in the L2 cache, it reduces L2 latency by 40% (4 cycles) compared to a standard 10-cycle phased L2 cache (4c tags + 6c data).

3 D2D: Accessing Data Directly

eTLB Hit + L1D Hit

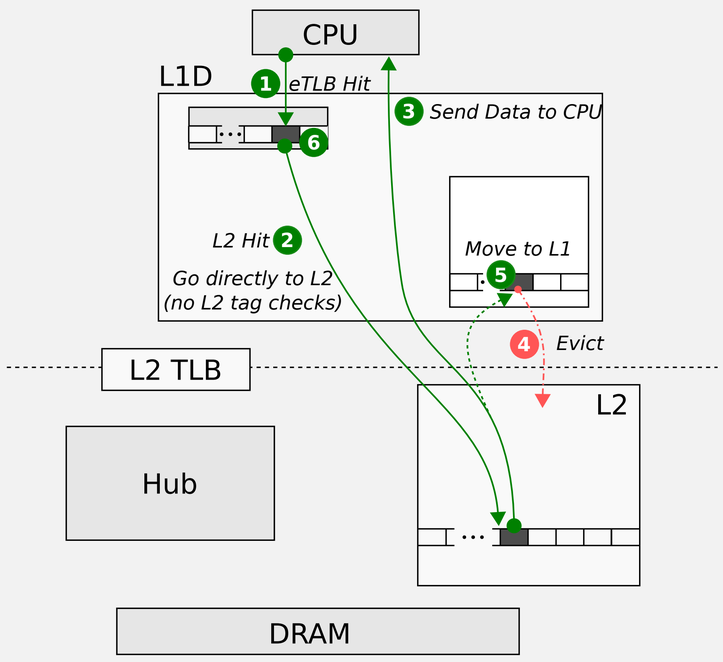
(94% of memory accesses)



The cache-level bits in the CLT entry indicate that the data is in the L1 cache. The L1 data-array is then accessed with the set index from the virtual address and the way index stored in the eTLB's cache-line location table (CLT) 2. The D2D cache accesses the correct L1 data array directly and sends the data to the CPU 3. Note that the set index from the virtual address is used to identify the correct cache line in the CLT, meaning that only 6 bits of the CLT (384 bits) needs to be read. Moreover, the physical address in the eTLB is not needed since the correct way is determined by the way information from the CLT (and not by comparing tags).

eTLB Hit + L2 Hit

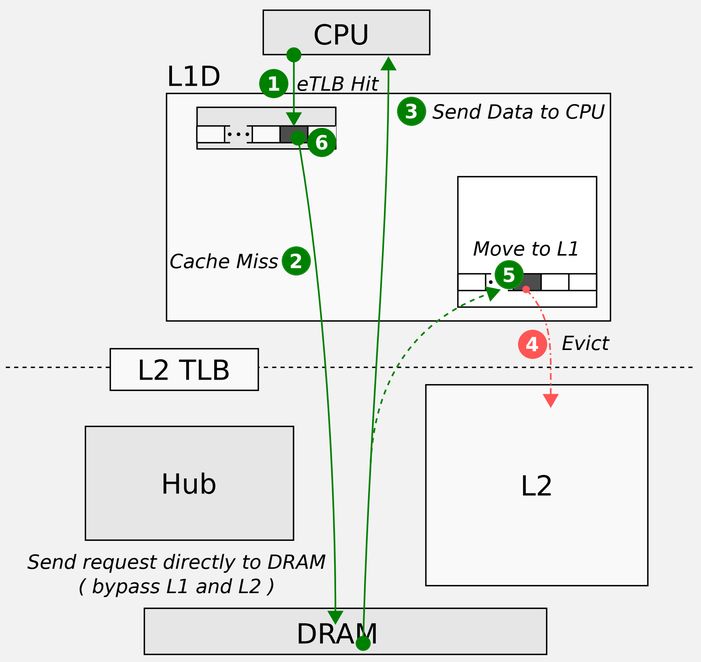
(3% of memory accesses)



The cache-level bits in the CLT entry indicate that the data is in the L2 cache. The L2 data-array is physically indexed and accessed by combining: 1) parts of least significant bits of the virtual address, 2) the parts of the physical address bits from the eTLB (4 bits for 1MB 16way L2), and 3) the way index bits stored in the eTLB's cache-line location table (CLT) 2. The D2D cache accesses the L2 data array and sends the data to the CPU 3. The data is then installed into the L1 cache by first evicting a cache line from the L1 cache 4, and then migrating the data from the L2 cache into the L1 cache and updating the Hub pointer (HP) to point to the cache line's page in the Hub 5. The CLT is updated to point to the new location in the L1 cache 6.

eTLB Hit + Cache Miss

(2% of memory accesses)



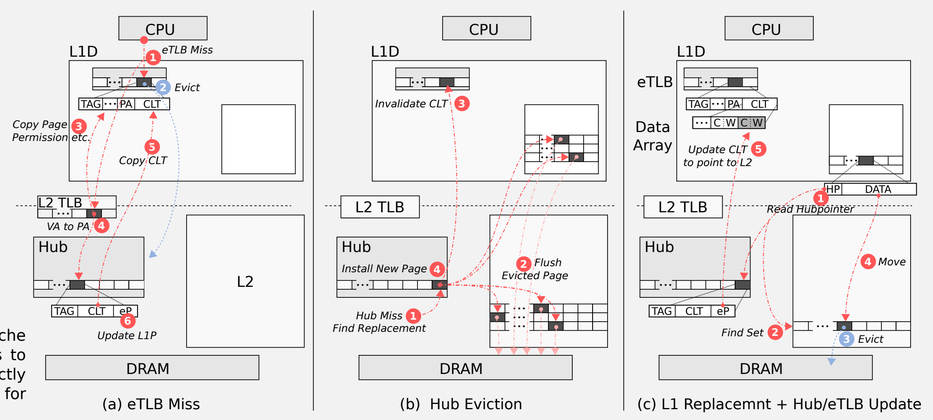
The cache-level bits indicate that the data is not in any of the caches. D2D then reads the physical address from the eTLB, and sends a memory request directly to the DRAM controller 2. The data is fetch from memory, and D2D sends the data to the CPU 3. The data is then installed into the L1 cache by first evicting a victim from the L1 cache 4, and then moving the data into the L1 cache and updating the Hub pointer (HP) to point to the cache line's page in the Hub 5. The CLT is updated to point to the new location in the L1 cache 6.

4 D2D: Extra Cache Management

A. eTLB Miss
→Fetch CLT from Hub
B. Hub Eviction
→Flush cache lines (no tags)
C. Cache Line Replacement
→Move data and update eTLB

Off Critical Path

These events are off the critical path. That is, they are done during cache misses (B, C) or TLB misses (A). Regular accesses, when the CPU wants to fetch data from L1, L2, or DRAM are fast since the data is accessesed directly using the eTLB. D2D therefore trades some extra cache management for faster and more energy efficient cache accesses.



5 Summary

1. **Skip levels in the cache hierarchy**
by determining the correct level from the TLB
2. **Eliminate extra data-array reads**
by determining the correct way from the TLB
3. **Eliminate the tag-arrays**
by avoiding tag comparisons

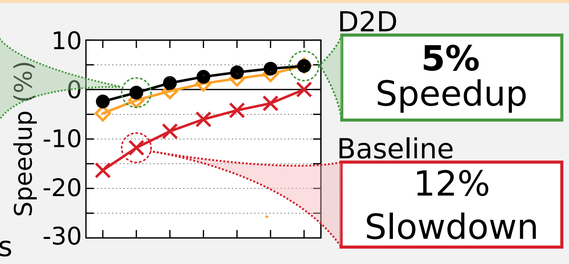
40% Lower L2 Latency
21% Lower Cache Energy

7 Use Case 1: L1 Cache Size

Trade off D2D's lower L2 latency for energy by using a smaller L1 cache size (without hurting performance).

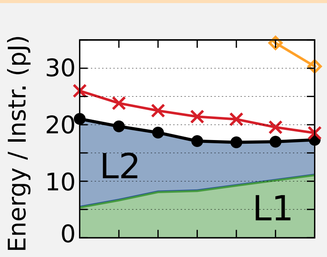
D2D
No perf. loss with smaller L1

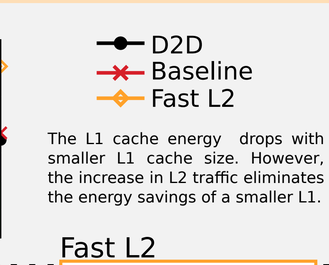
D2D's lower L2 latency results in a 5-14% speedup for L2 sensitive applications compared to a traditional cache hierarchy.



D2D
5% Speedup

Baseline
12% Slowdown



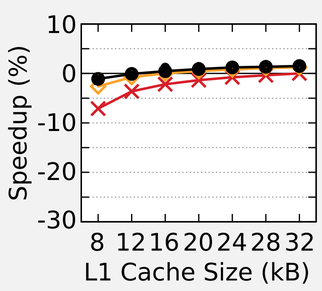


The L1 cache energy drops with smaller L1 cache size. However, the increase in L2 traffic eliminates the energy savings of a smaller L1.

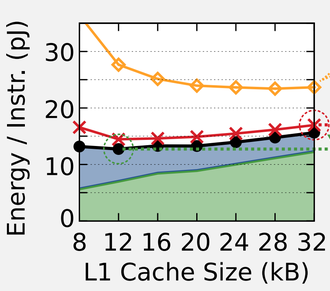
L2 Sensitive Programs

Overall
(average across all of SPEC, ...)

Many applications have low L2 utilization. We therefore see lower performance improvement when looking at the average across all applications.



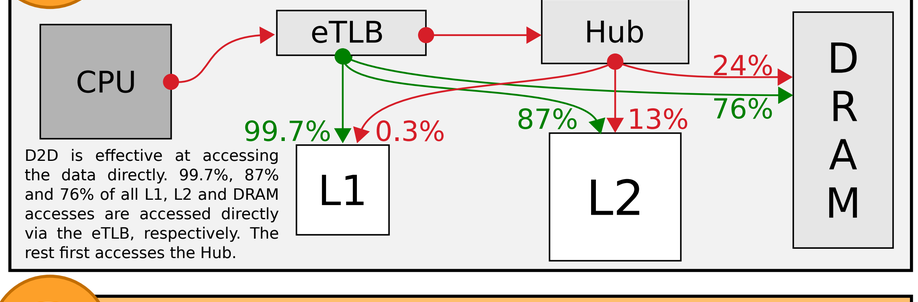
L1 cache energy drops with smaller cache sizes, resulting in a total cache energy reduction.



Fast L2
+Low Latency
-High Energy

D2D
21% Lower Energy

6 eTLB Effectiveness

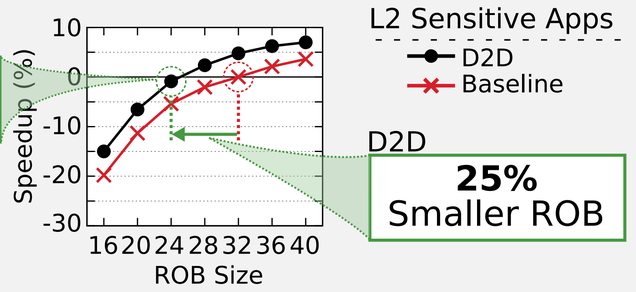


D2D is effective at accessing the data directly. 99.7%, 87% and 76% of all L1, L2 and DRAM accesses are accessed directly via the eTLB, respectively. The rest first accesses the Hub.

8 Use Case 2: ROB Size

D2D
No perf. loss with smaller ROB

D2D's lower L2 latency enable us to use a smaller reorder buffer without hurting performance, since the ROB does not need to hide as much cache latency.



D2D
25% Smaller ROB

1. TLC: A Tag-Less Cache for Reducing Dynamic First Level Cache Energy, In International Symposium on Microarchitecture (MICRO), December 2013
2. The Direct-to-Data (D2D) Cache: Navigating the Cache Hierarchy with a Single Lookup, In International Symposium on Computer Architecture (ISCA), June 2014