NOMAD

Non-blocking OS-Managed DRAM Cache

Authors

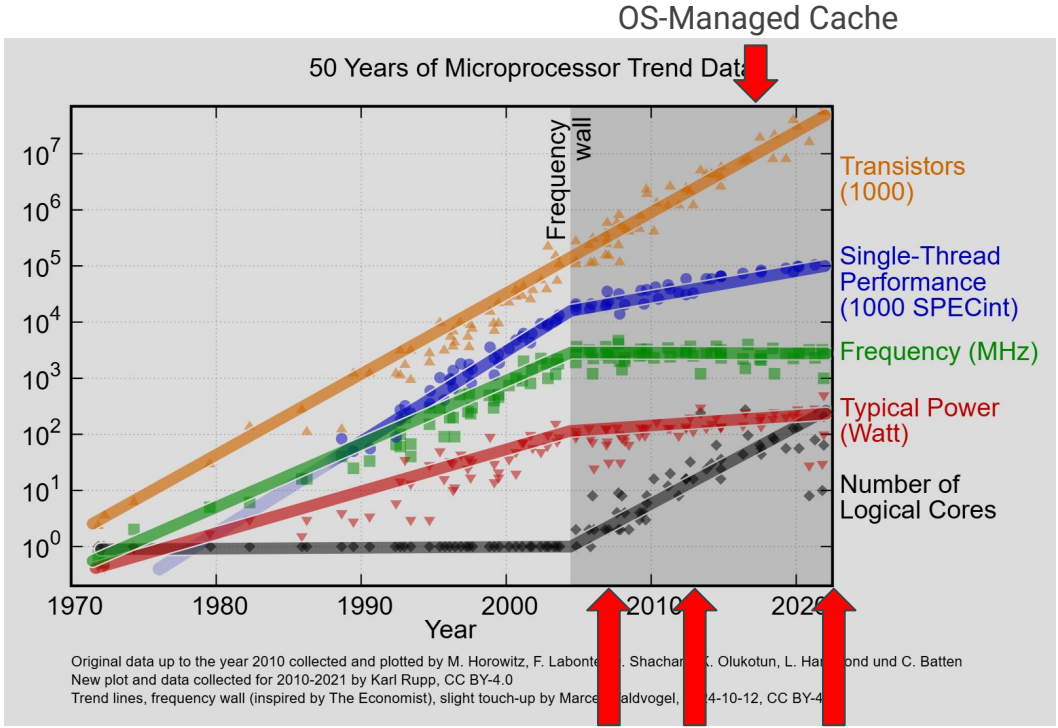
- Youngin Kim
  - Yonsei University PHD
  - System Architect at MangoBoost
- Hyeonjin Kim
  - Yonsei University PHD
  - Samsung Research
- William J. Song
  - Yonsei University Associate Professor

Youngin Kim

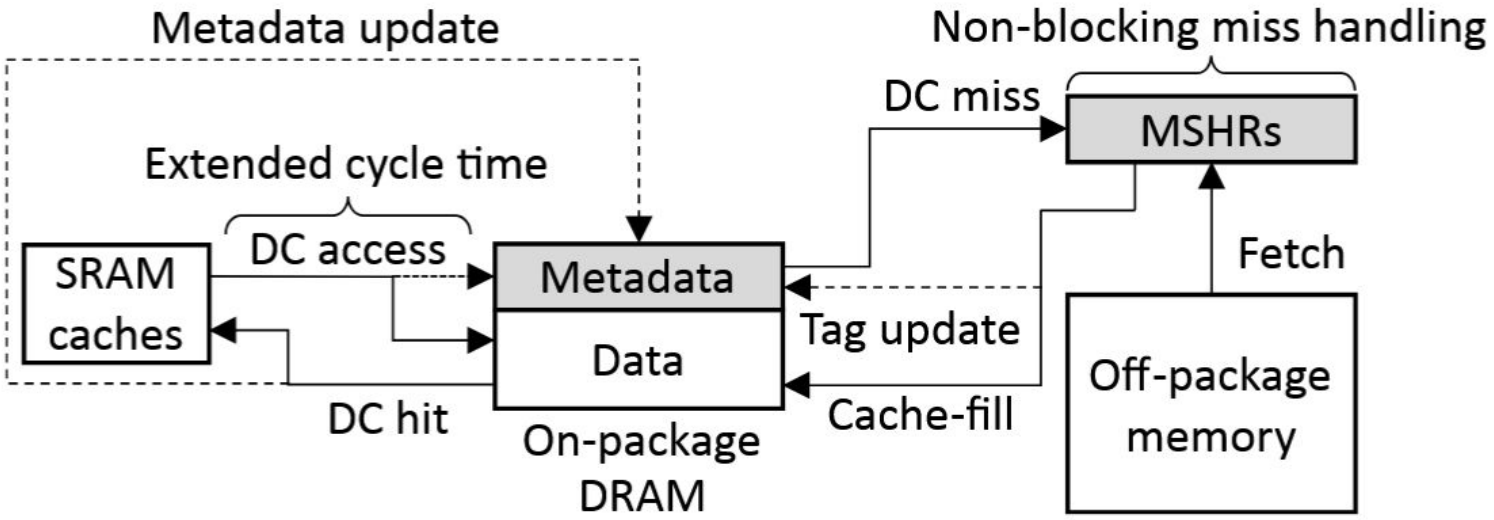
Hyeonjin Kim

William J. Song

Historical Context

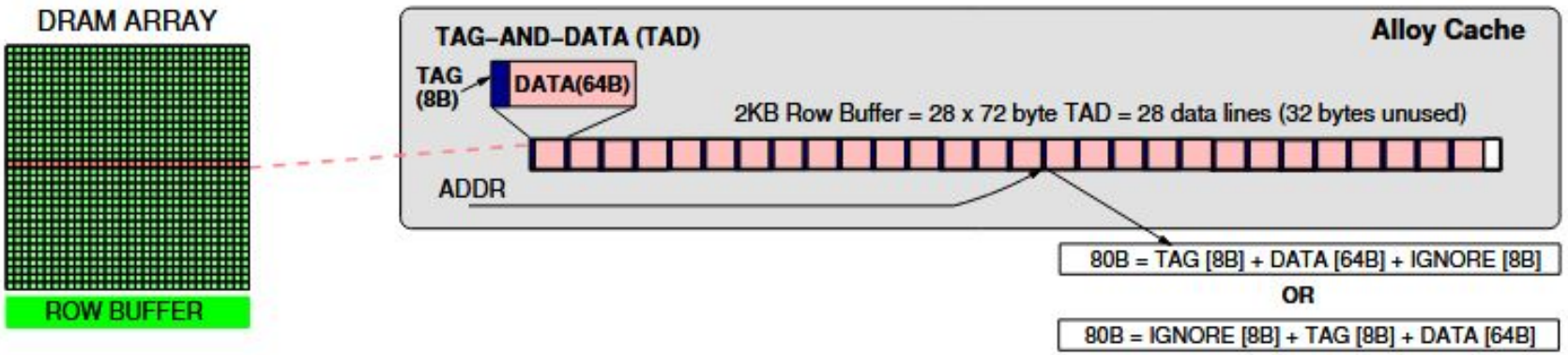


Hardware Managed DRAM Cache



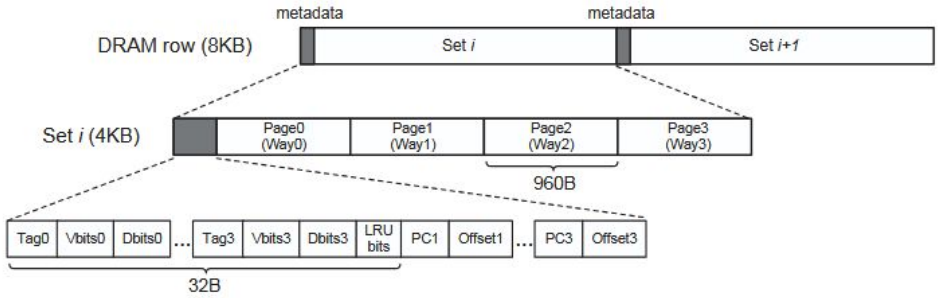
- Non-blocking + Good Parallelism
- Metadata Tag Store + Bandwidth

Hardware Managed - Alloy Cache (2012)



- Stores tag and data in same DRAM row
- Saves tag lookup bandwidth

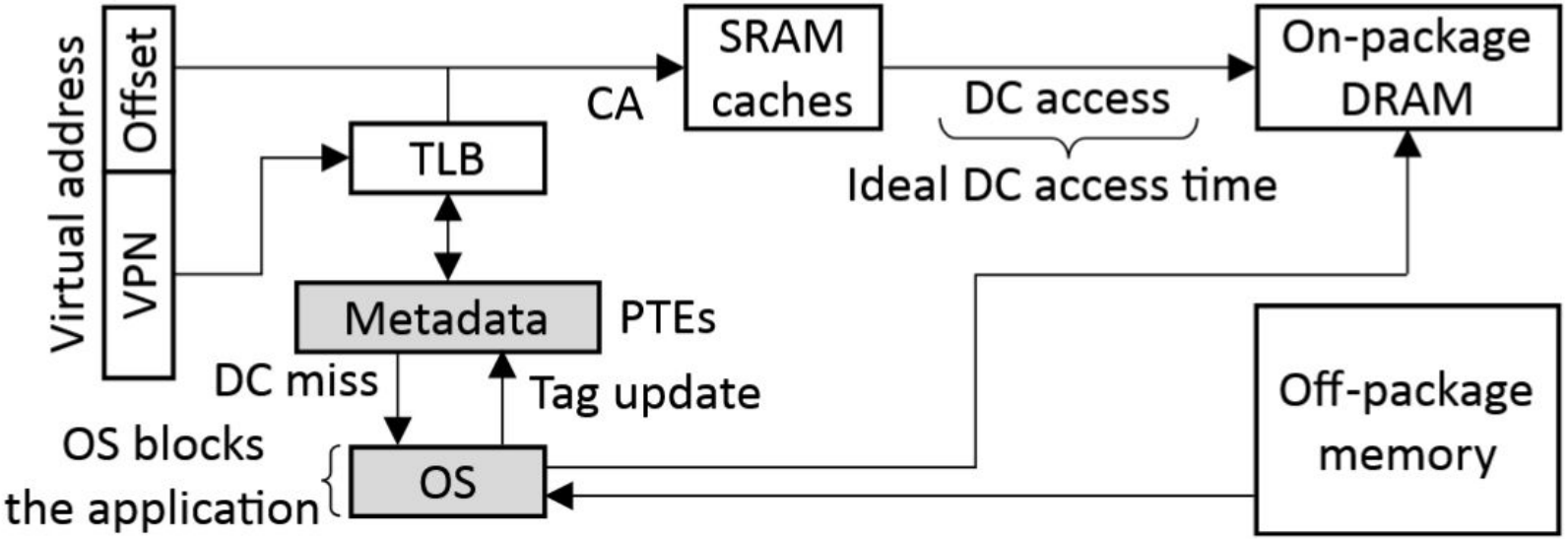
Hardware Managed - Unison Cache (2014)



	AC	FC	UC
No SRAM tag overhead	✓	✗	✓
Low hit latency	✓	✗	✓
High hit rate	✗	✓	✓
High effective capacity	✗	✓	✓
Scalability	✓	✗	✓

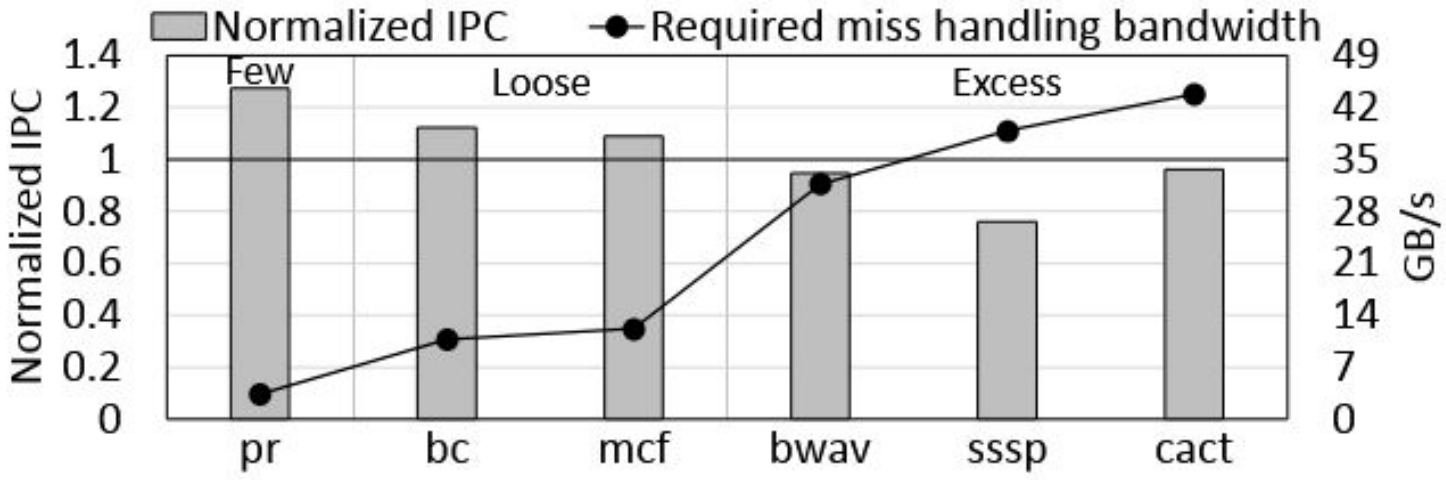
- Based on Alloy Cache and Footprint Cache
- Organizes metadata at start
- Use footprint predictor to learn which block is likely to be accessed

Software Managed DRAM Cache



- No metadata traffic + fast lookup
- On DRAM cache miss, OS updates PTE and stalls

OS to HW Managed Comparison



- OS-managed DRAM relative to HW-based
- OS better for low RMHB, worse for high RMHB

Core Problem

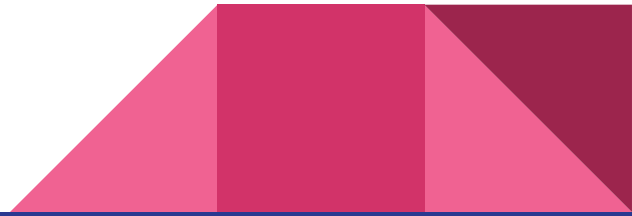
Hardware-managed DRAM caches: non-blocking but suffers metadata bandwidth overhead

OS-managed DRAM caches: avoids metadata overhead but blocks DRAM cache miss

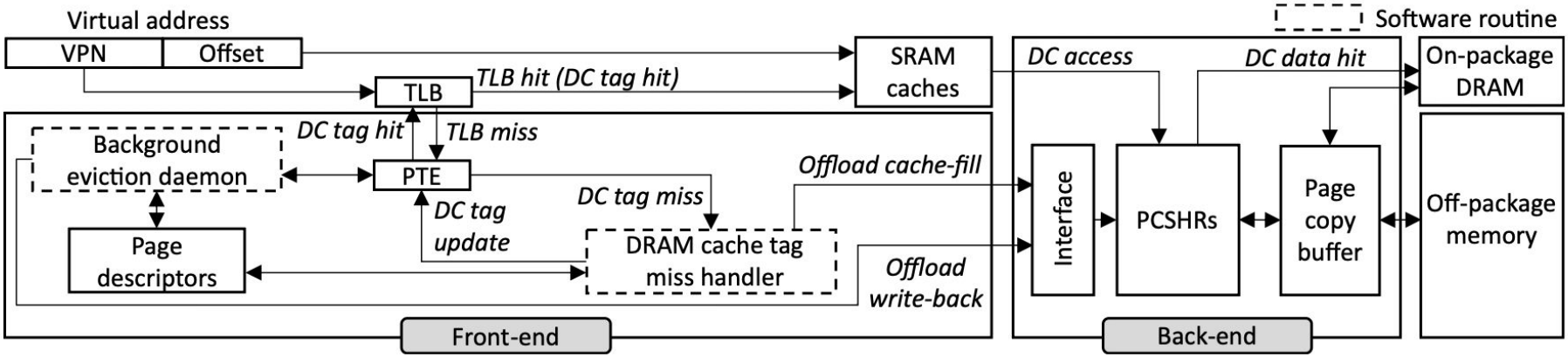
NOMAD Design

Overall Structure

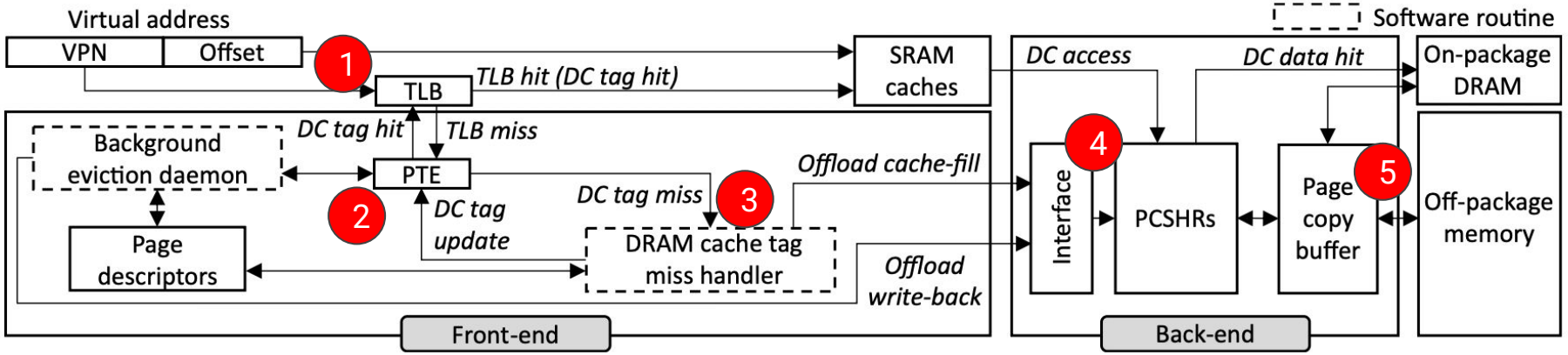
- Decoupled tag and data management
- Front-end OS routines
  - Virtual address mapping (page frames)
  - Issues commands to back-end
  - Allows thread to continue execution
- Back-end hardware
  - Data management (cache fill, writebacks…)
  - Non-blocking cache



NOMAD Overall Structure



Miss in TLB and DRAM Cache



1. Miss in TLB - start page table walk
2. Miss in page table, issue miss to handler
3. Allocate new cache frame, issue cache-fill & restore execution
4. Allocate page copy status holding register (PCSHR)
5. Update PCSHR state and service requests

Front-end OS Routines - Page Descriptors

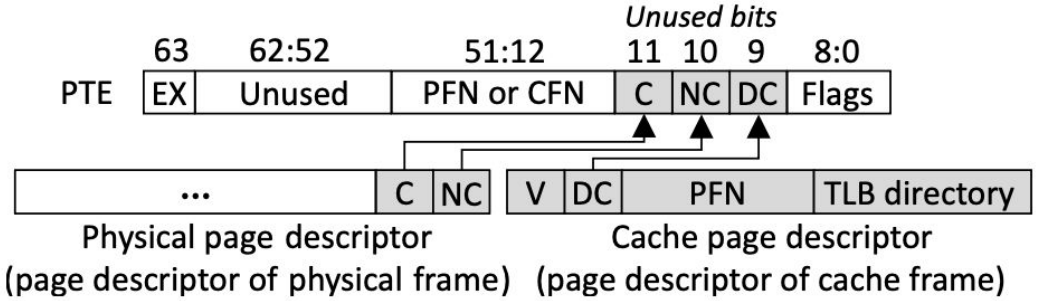
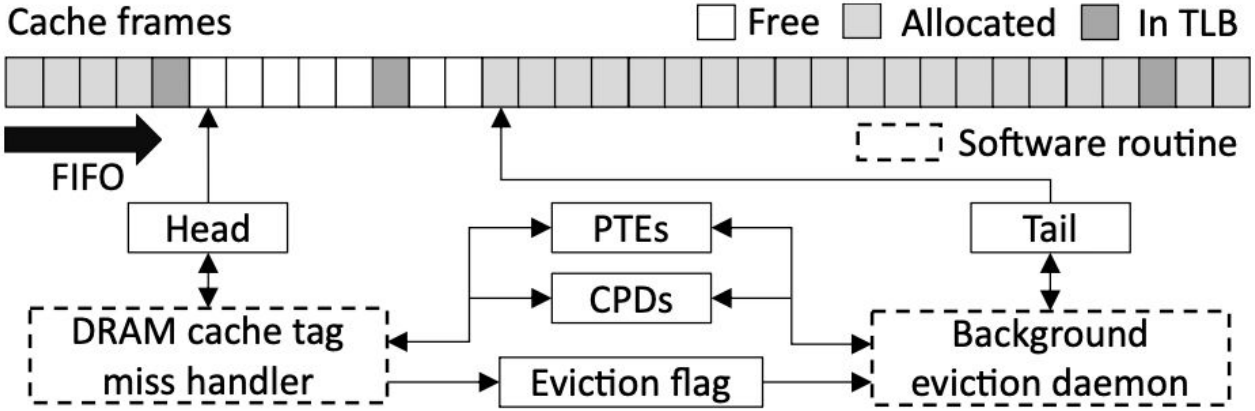


Fig. 4: Page descriptor and PTE extension [1] in NOMAD.

- PPD - cacheable and non-cacheable
- CPD - valid, dirty
  - TLB - avoid TLB shootdown
  - PFN - reverse mapping when deallocated

Stores metadata as page table entry in OS

Miss Handling



- Utilizes FIFO replacement policy
- Miss handler allocates to the head
- Background eviction proactively from tail

OS keeps track of free frames using FIFO
Fully associative lookup

Miss Handling

Algorithm 1. DRAM Cache Tag Miss Handling Routine

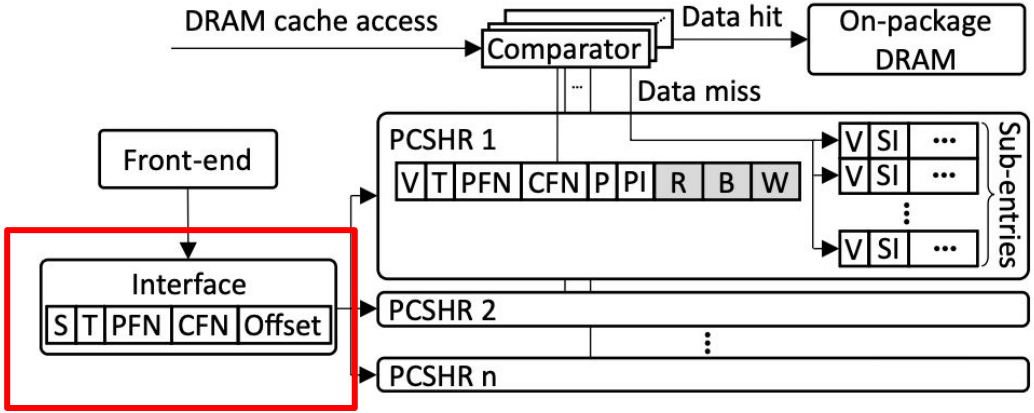
```
Input: pte (page table entry), va (virtual address)
 1: mutex_lock(&cache_frame_management_mutex)
    /* Find free cache frame */
 2: cpd = cache_page_descriptors[head]
 3: while unlikely cpd.valid do
 4:    cpd = cache_page_descriptors[++head]
 5: end while
    /* Offload data management task to back-end */
 6: send_cachefill(head, pte.page_frame_num, offset(va))
    /* Tag management */
 7: cpd.valid = true
 8: cpd.pfn = pte.page_frame_num
 9: pte.cached = true
10: pte.page_frame_num = head
    /* Set eviction flag */
11: num_free_cache_frames--
12: if num_free_cache_frame < eviction_threshold then
13:    eviction_flag = true
14: end if
15: mutex_unlock(&cache_frame_management_mutex)
```

Algorithm 2. Background Eviction Routine

```
 1: mutex_lock(&cache_frame_management_mutex)
    /* Reset flag */
 2: eviction_flag = false
    /* Flush cache for data consistency */
 3: flush_cache_range(tail,n) /* (start,victims) */
    /* Evict pages */
 4: for i = 0 to n − 1 do
 5:    cpd = cache_page_descriptors[tail]
       /* Skip this page if it is in any of TLB. */
 6:    if unlikely cpd.tlb_directory != 0 then
 7:       tail++; continue
 8:    end if
       /* Perform writeback for dirty victims */
 9:    if cpd.dirty-in-cache then
10:       send_writeback(tail, cpd.pfn)
11:    end if
       /* Recover PTEs */
12:    rmap = get_reverse_mapping(cpd.pfn)
13:    for all pte in rmap do
14:       pte.page_frame_num = cpd.pfn
15:    end for
16:    cpd.valid = false
17:    num_free_cache_frames++
18: end for
19: mutex_unlock(&cache_frame_management_mutex)
```

- Proactive eviction
- Skip entries in TLB to avoid TLB shootdown
- Reverse map PFN to recover page table entry (PTE)

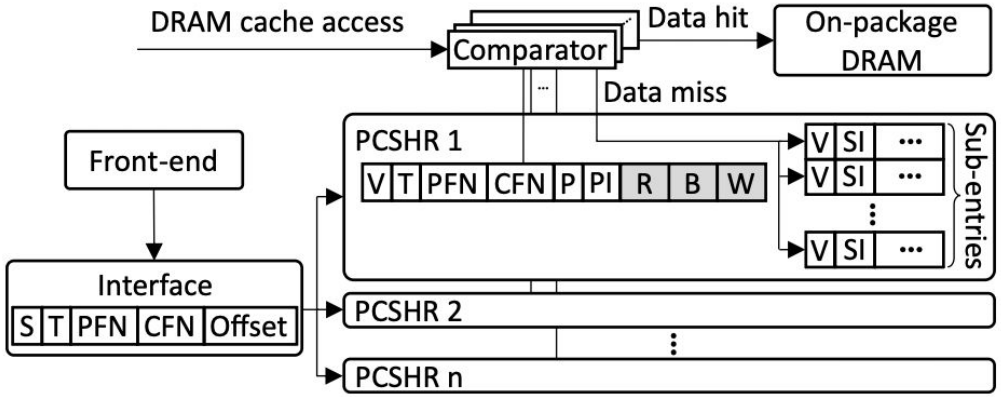
Back-end Hardware - Interface



- Accessible by OS
- State - busy when working or no PCSHRs
- If idle, allocates a PCSHR - holds status of page copy

Hardware receives requests from OS and processes them concurrently

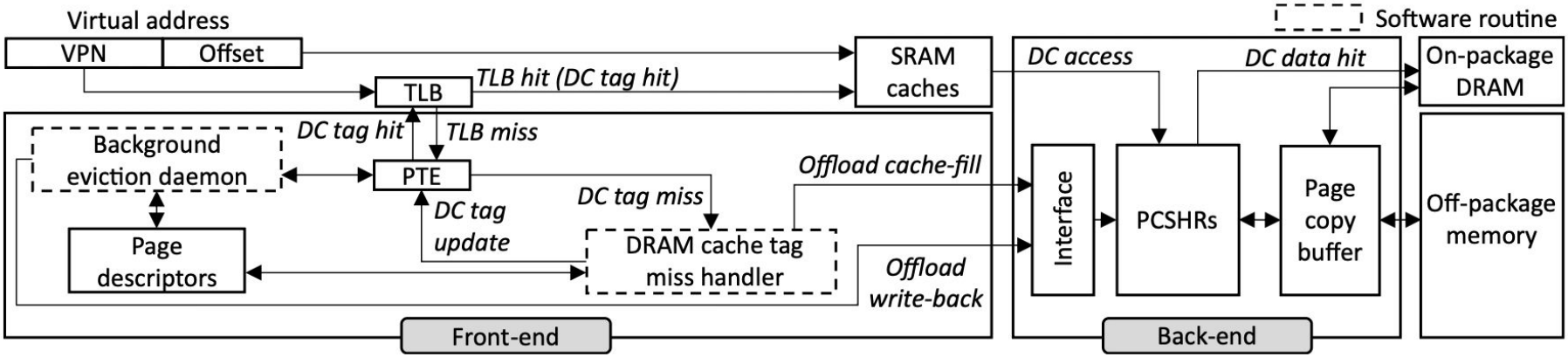
Back-end Hardware - Handling DRAM Access



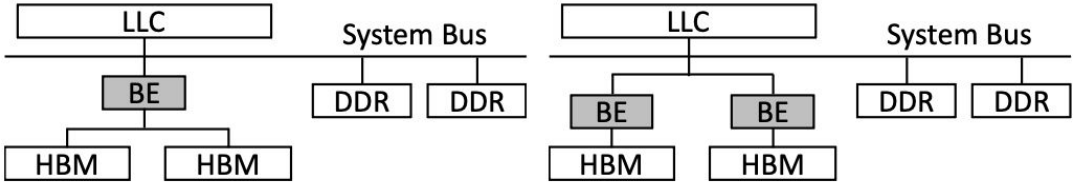
- PCSHR holds outstanding cache-fills at page granularity
- Matched tags implies miss
  - Schedules a request in sub-entry
- Can use metadata to “hit” in page copy buffer

Each PCSHR fills a page, which can have multiple requests at the block-level

NOMAD Overall Structure



Other Design Details



(a) Centralized back-end (b) Distributed back-ends

- Distributed back-end for each HBM
  - Won't become imbalanced because of FIFO allocation policy
- Supports caching of shared pages
- Supports superpage allocation to extend TLB reach
  - Pages much larger than 4KB, MB or GB

NOMAD Results

Experiment Details

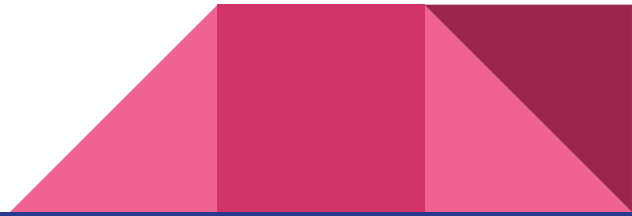
- modeled NOMAD on the gem5 simulator
- 1 GB shared DRAM cache
- HBM2 On Package DRAM
- DDR4 Off Package Memory
- Multi-level TLBs

TABLE I: Workload Characteristics

Class	Abbr.	Workloads	RMHB (GB/s)	LLC MPMS	Memory footprint (GB)
Excess	cact	cactusADM [3]	43.8	486.6	11.9
	sssp	sssp [3]	38.8	511.1	2.3
	bwav	bwaves [16]	31.7	588.1	4.5
Tight	les	leslie3d [16]	26.5	532.8	7.5
	libq	libquantum [16]	25.1	210.6	4.0
	gems	gemsFDTD [16]	24.8	269.2	6.3
	bfs	bfs [3]	23.1	298.5	2.4
Loose	cc	cc [3]	13.5	183.1	2.3
	lbm	lbm [16]	12.4	270.5	3.2
	mcf	mcf [16]	12.2	472.0	2.8
	bc	bc [3]	10.8	533.7	1.3
Few	ast	astar [16]	6.9	72.1	1.0
	pr	pr [3]	3.4	691.9	4.8
	sop	soplex [16]	1.7	310.2	1.2
	tc	tc [3]	1.66	226.3	2.3

Compared Memory Schemes

- Baseline
  - Only off-package memory
- TiD (tags in DRAM)
  - Unison Cache
  - HW-based,
  - 1KB cache line,
  - four-way set-associative
- TDC (tagless Dram Cache)
  - OS-managed DC design,
  - blocking miss handling
- Ideal
  - Ideal OS-managed DC,
  - no latencies for tag miss handling, page copy



Evaluation

- NOMAD improved IPC by 16.7% over TDC, 25.5% over TiD
- With RMHB level in Few class, TDC performance similar to NOMAD

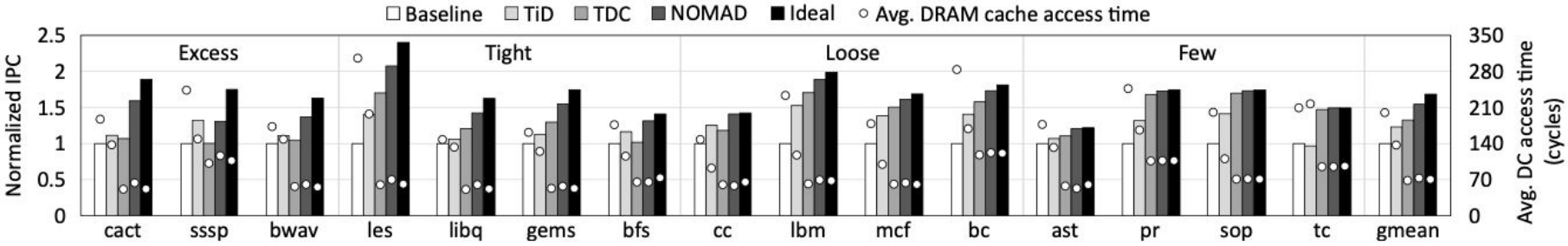


Fig. 9: IPC normalized to the baseline and average DRAM cache access time in CPU cycles.

Evaluation

- TiD has high bandwidth usage which lead to longer DRAM cache access time

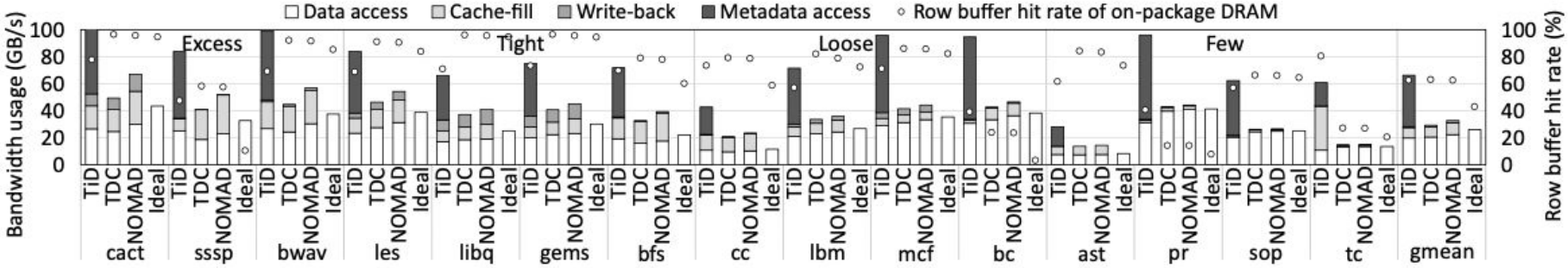


Fig. 10: Breakdown of on-package DRAM bandwidth usage and row buffer hit rates of the on-package DRAM.

Evaluation

- TDC stalled more for workloads with higher RMHB
- NOMAD has slightly higher tag manage latency from its front end routines

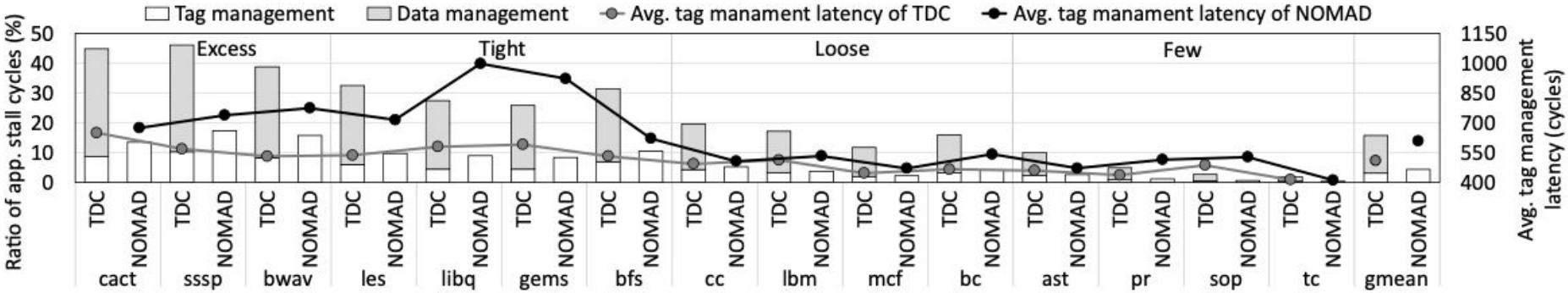


Fig. 11: Breakdown of application stall cycle ratios and the average tag management latency of OS-managed schemes.

Sensitivity to Number of PCSHRs

- PCSHR could bottleneck miss handling bandwidth
- 8 PCSHRs offer near-max performance
- IPC independent of CPU core counts as off package DRAM bandwidth bottleneck performance

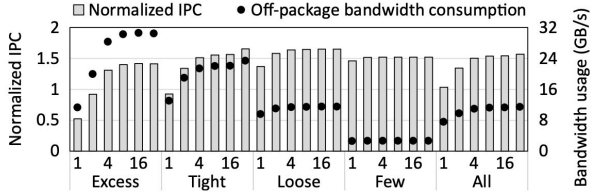


Fig. 12: Per-class average IPC relative to the baseline and the average off-package memory bandwidth consumption of NOMAD with respect to the number of PCSHRs.

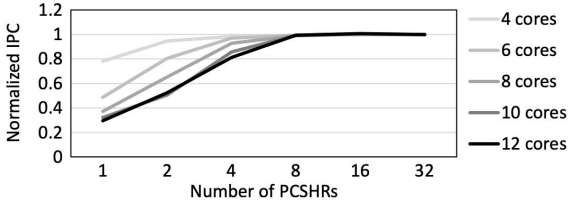


Fig. 13: Average IPC of Excess-class benchmarks with different number of PCSHRs for increasing CPU core count.

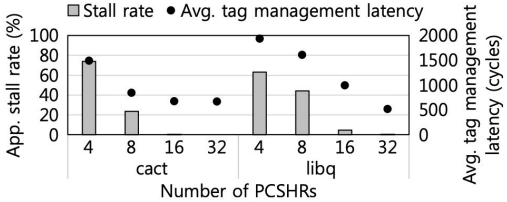


Fig. 14: Application stall rates and the average tag management latency of cact (highest RMHB) and libq (bursty RMHB) with respect to the number of PCSHRs.

Area Optimized Design

- Area Overhead is mainly from page copy buffers (4KB × PCSHR count)
- Increasing PCSHR count help burst-RMHB workloads
- Number of page copy buffer is independent

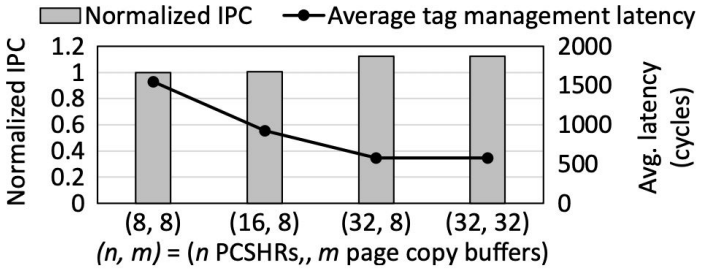


Fig. 15: Normalized IPC and the tag management latency of burst-RMHB workloads (i.e., libq, gems) with different configurations of (n PCSHRs, m page copy buffers).

Key Takeaways

- Decouple Tag Management and Data Management
- Tag Management in Front End using TLB and PTE
- Data Management in Backend
- Track Misses with PCSHRs
- Copy Pages in subblocks

Limitations

- Front end OS tag management work introduce more latencies

- Backend Hardware costs from 4KB Page Copy Buffer Area