Accelerating Page Migrations with Intel Data Streaming Accelerator

Jongho Baik[†], Jonghyeon Kim[†], Chang Hyun Park[‡], and Jeongseob Ahn[§]

† AJOU UNIVERSITY

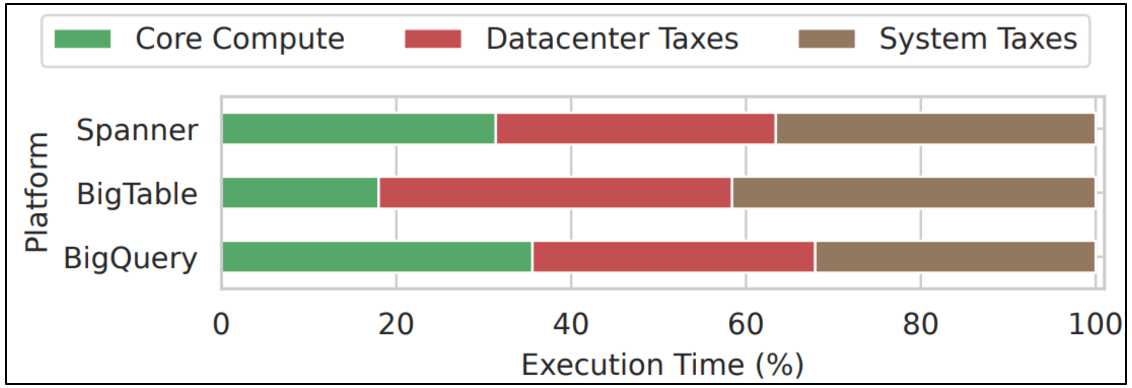
‡ UPPSALA UNIVERSITET

§ KOREA UNIVERSITY

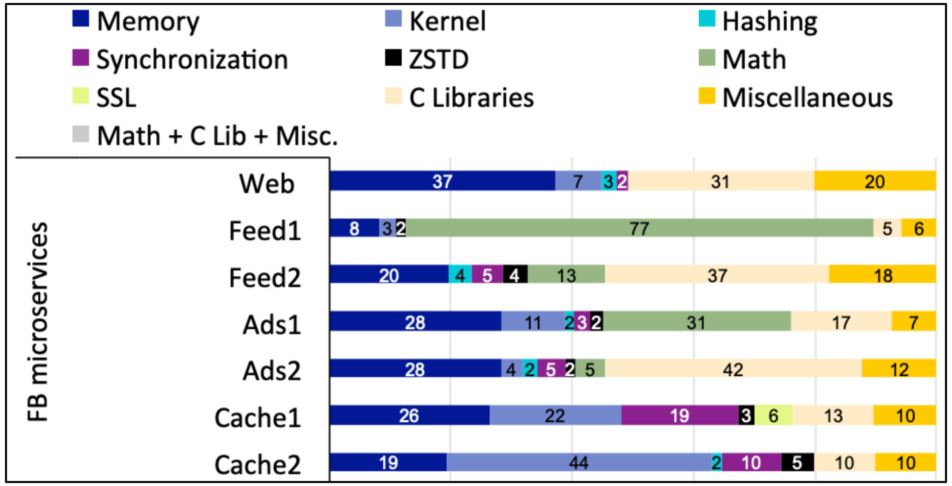
Appeared in IEEE Computer Architecture Letters, 24(1), 2025

Motivation : Traditional CPU-based Data Processing

- Data generation, transfer, and processing are escalating to unprecedented levels
- In the same way, Datacenter Taxes (e.g., memcpy, hashing) has grown fast
- NUMA System, Tiered Memory (PMEM, CXL) makes memory managing important
- **Google[ISCA '23] : up to 40% of CPU cycles are spent on datacenter taxes**
- **Facebook[ASPLOS '20] : up to 37% of datacenter taxes are spent on memory functions (e.g., `memcpy`, `malloc`, `memmove`).**



Source: Profiling Hyperscale Big Data Processing, Gonzalez et al., ISCA '23

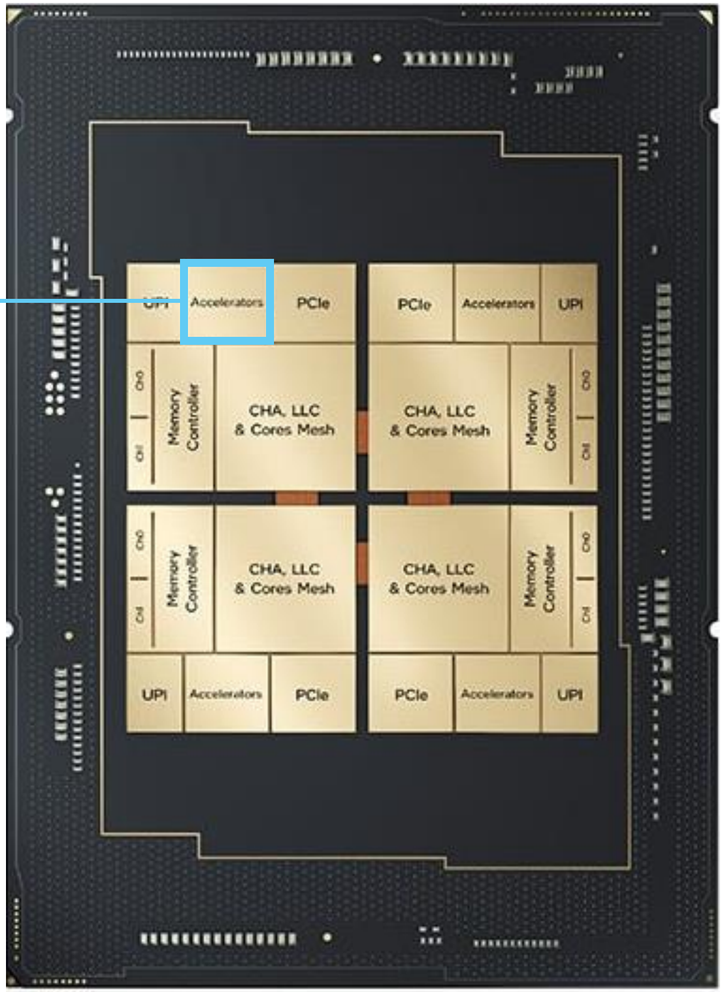


Source: Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale, Sriraman et al., ASPLOS '20

Intel® Data Streaming Accelerator

- On-chip accelerator since 4th Gen Xeon® processor
  - Data move, fill, compare, and more
  - Offloads data copy and data transformation operations
  - Freeing up CPU resources (Increasing compute capacity)
  - Accelerate data movement throughput

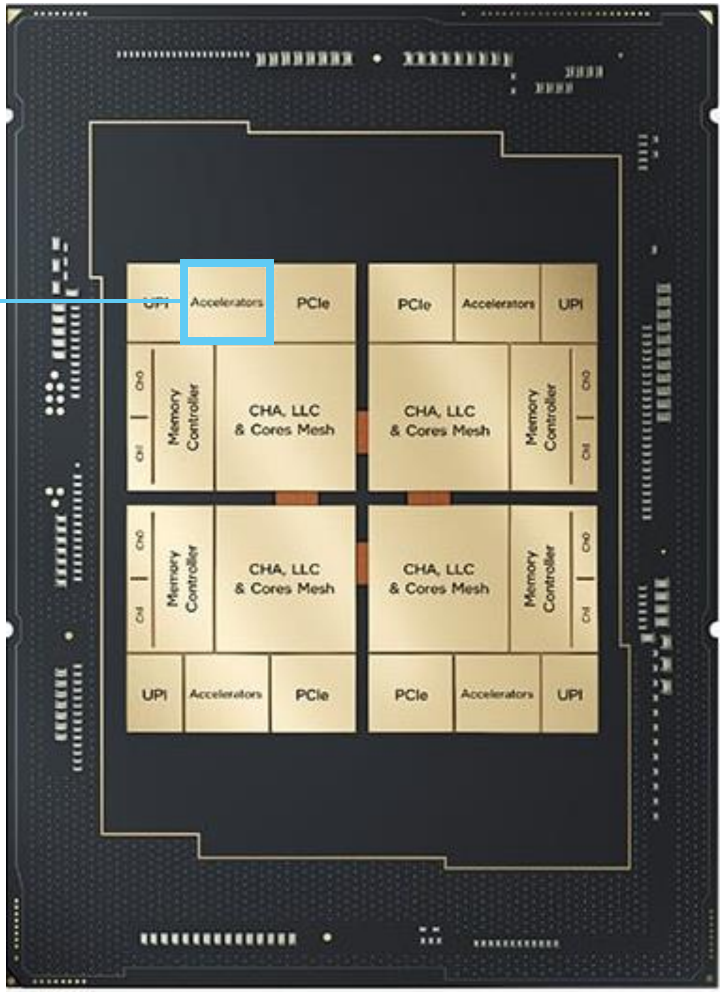
On-chip accelerators
- Intel DSA



Intel® Data Streaming Accelerator

- On-chip accelerator since 4th Gen Xeon® processor
  - Data move, fill, compare, and more
  - Offloads data copy and data transformation operations
  - Freeing up CPU resources (Increasing compute capacity)
  - Accelerate data movement throughput

On-chip accelerators
- Intel DSA



No attempt has been made to utilize DSA for operating systems

Where to assist OS as an `memcpy` accelerator?

- In the Linux kernel, `memcpy()` operates based on per page (e.g., 4 KB)
- To maximize the efficiency with DSA, we need to find tasks that moves large amounts of data in the kernel

Where to assist OS as an `memcpy` accelerator?

- In the Linux kernel, `memcpy()` operates based on per page (e.g., 4 KB)

- To maximize the efficiency with DSA, we need to find tasks that moves large amounts of data in the kernel

`migrate_pages()!`

Where to assist OS as an `memcpy` accelerator?

- `migrate_pages()` in the Linux kernel :

Module	Usage
Memory reclaiming (e.g., kswapd)	Move pages to different memory locations, facilitating the freeing of contiguous blocks of memory and improving memory availability
kcompactd	Relocate pages, compacting memory by consolidating free space into contiguous blocks, which is beneficial for large allocation requests and performance
NUMA balancing	Move pages to memory nodes closer to the CPU that accesses them most frequently, impr-oving memory access latency and overall performance
Memory hotplug	During memory offlining (removing memory), migrate_pages is used to move pages out of the memory regions that are being removed
DAMON	Promote hot pages classified by DAMON from the lower-tier to the upper-tier memory and demote cold pages from the upper-tier to the lower-tier memory

Where to assist OS as an `memcpy` accelerator?

- `migrate_pages()` in the Linux kernel :

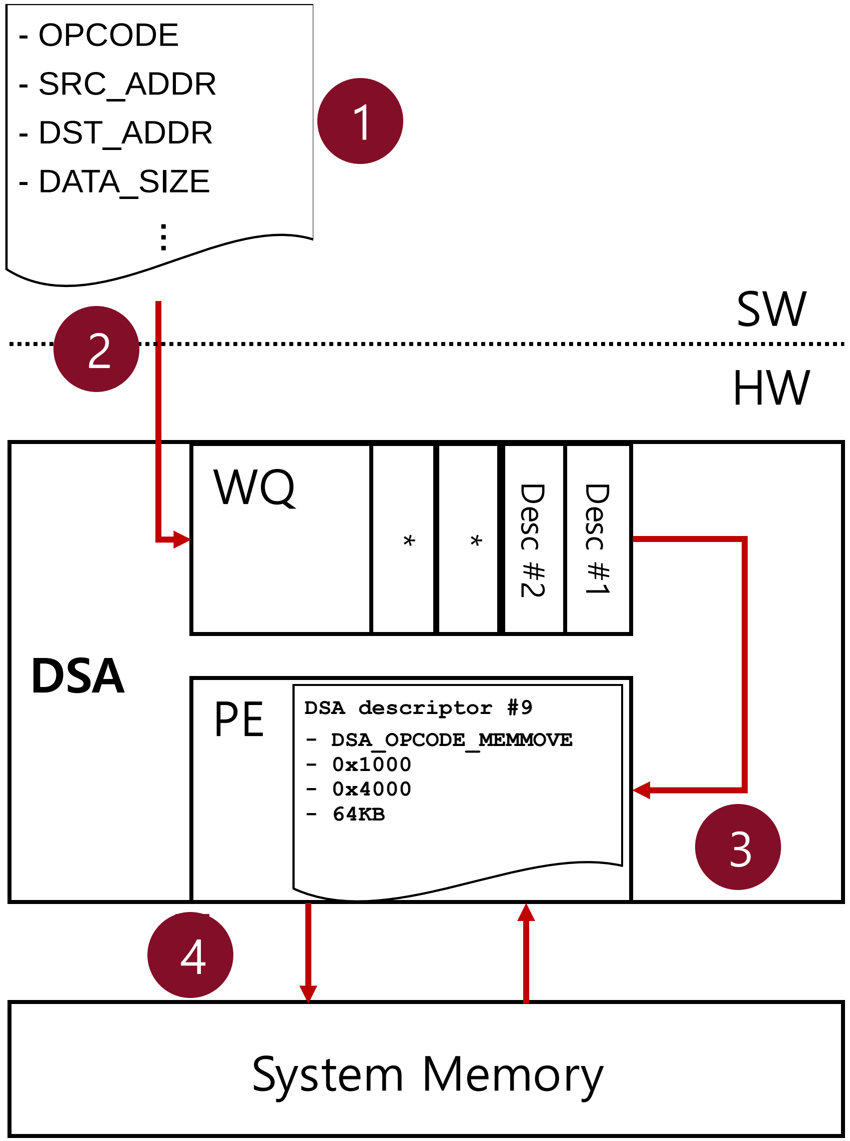
Such tasks in OS are critical

- **Directly impact system performance and resource management**
- **Especially in HPC and large-scale data centers**

Offloading Costs to DSA

- Offloading memory operations to DSA :

1. Create DSA descriptors that specifies the tasks
2. Submit the descriptors to DSA workqueue
3. A processing engine (PE) of DSA dispatches a descriptor from its workqueue
4. Perform the operation
5. Wait for completion



Offloading Costs to DSA

- To create a descriptor in OS:

  1. Create 2 scatter-gather(`SG`) tables
     - Tracking `src` and `dst` pages, respectively
     - Pages may not be contiguous

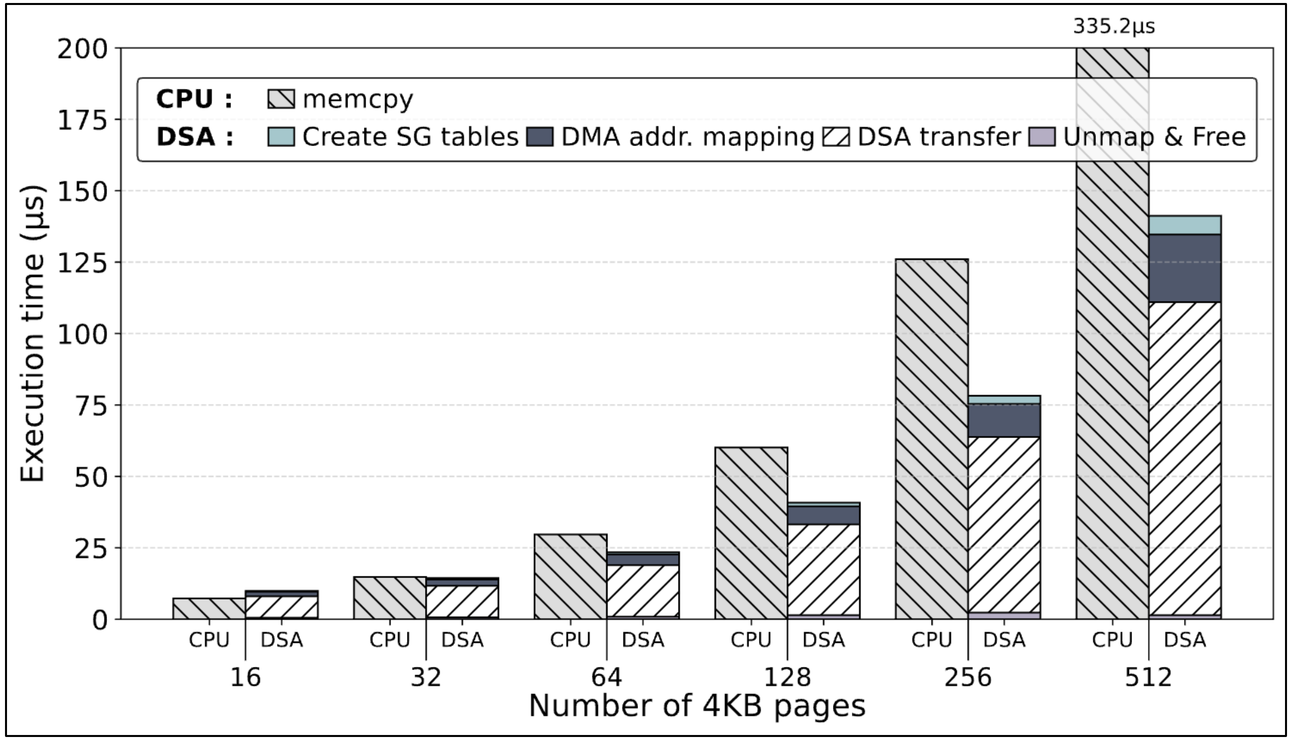
  2. Map the `SG` tables to get DMA-capable addresses(IOVA)

  3. Create DSA descriptors through DMA-mapped addresses

- After the operation is done:

  4. Unmap the DMA-mappings and Clean up the `SG` tables

Offloading Costs to DSA

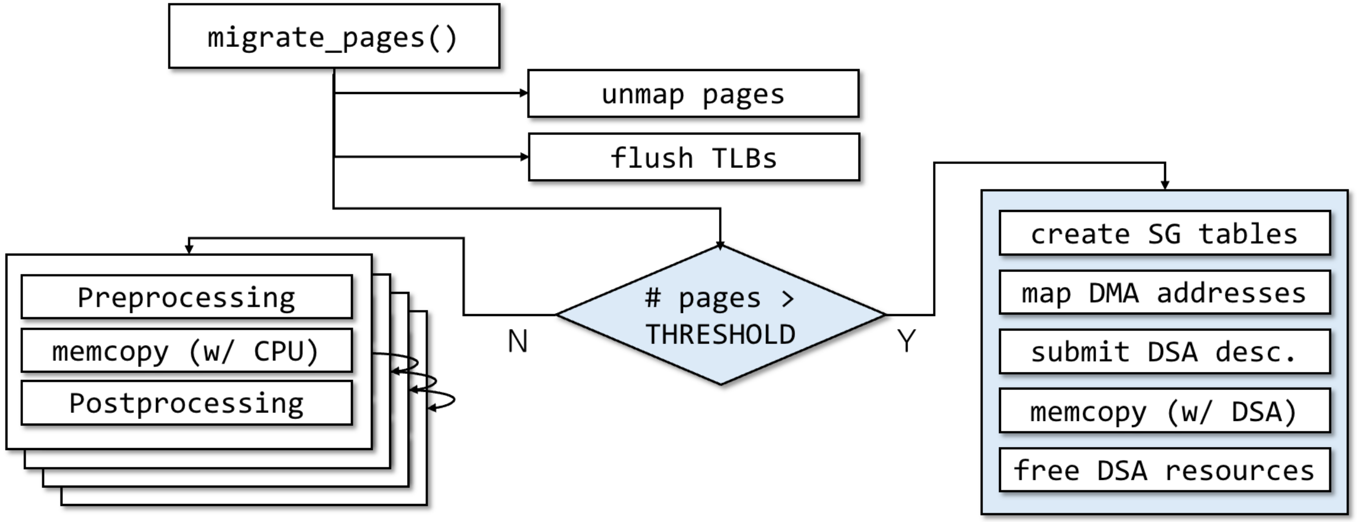


<Performance breakdown of copying pages: CPU vs DSA>

Due to the offloading costs, DSA is preferred when the number of pages exceeds 32

Accelerated `migrate_pages()` with DSA

- Designed to exploit the performance advantages of DSA
  - THRESHOLD : 32
  - # pages > THRESHOLD
    - Migrating pages with DSA



Copy each page *iteratively*

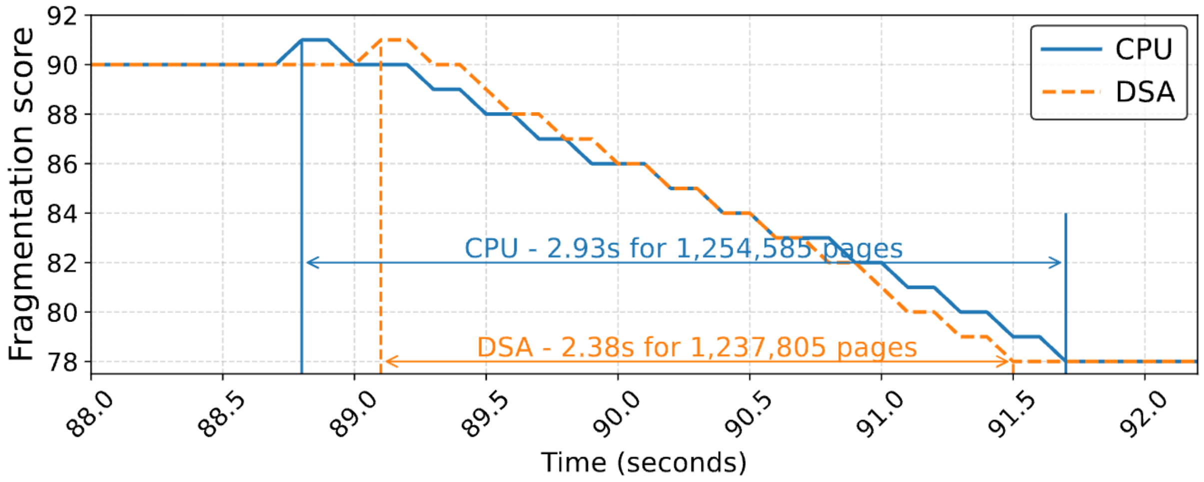
Copying pages in a *batch*

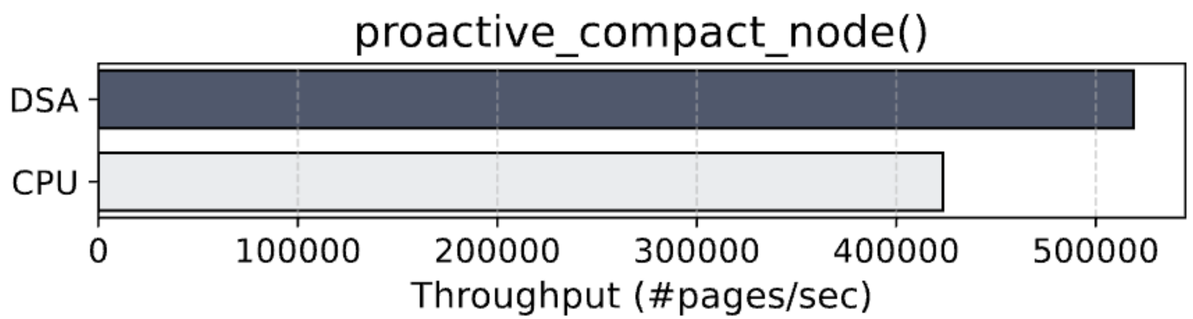
Evaluation

- System
  - Intel Xeon Gold 6430 CPU @ 2.10GHz x 2 sockets
    - Each socket has 1 DSA device
  - 128GB DDR5 DRAM per socket
  - Linux kernel 6.8 with Ubuntu 22.04
- Benchmark
  - Memory Compaction: `kcompactd`
  - Memory Promotion: `DAMON`
    - XSBench
    - GAP Benchmark(SSSP)

Evaluation: Memory Compaction (kcompactd)

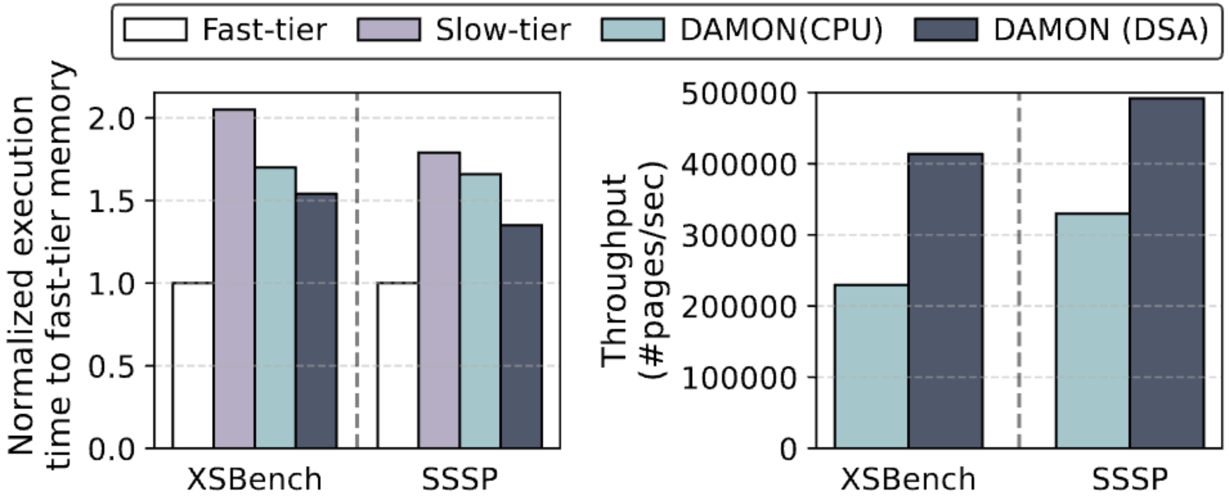
- Proactive Compaction
  - Start when the node's fragmentation score exceeds the high threshold(90)
  - For defragmentation and Reduce higher-order memory allocation latencies
- DSA shows an improved throughput of 1.2x





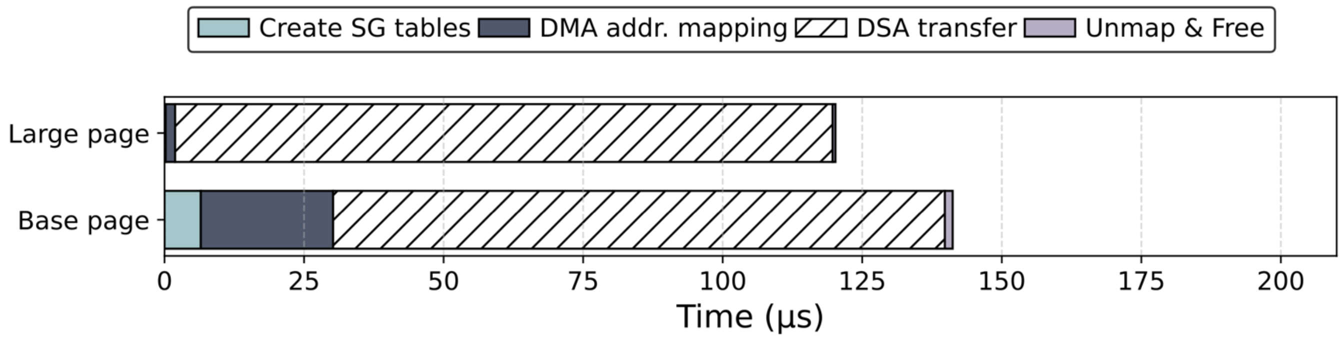
Evaluation: Memory Promotion (DAMON)

- HMSDK merged into DAMON for memory tiering (e.g., CXL Memory)
  - Emulation: lowering the uncore frequency of the remote NUMA node
- Memory Promotion : Promoting data from slow-tier to fast-tier memory
  - For SSSP, it improves the execution time by 31%

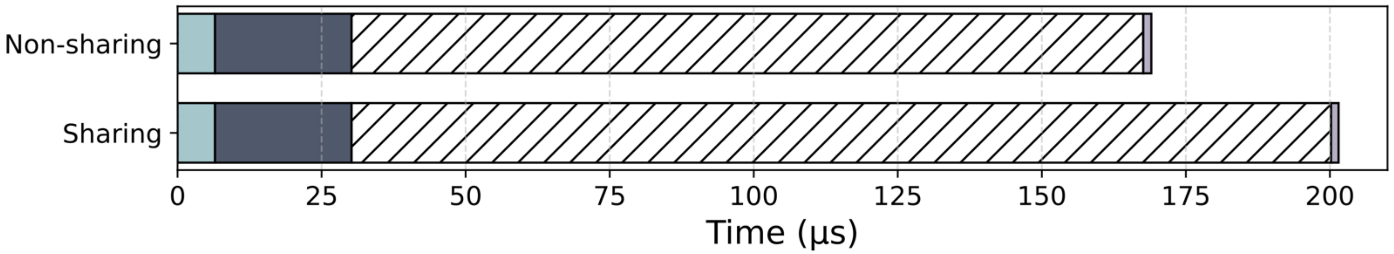


Evaluation: Page size and Performance Interference

- Base (4KB) page vs. large (2MB) page)



- Contention in DSA
  - Non-sharing: user and kernel threads use two different PEs, respectively
  - Sharing: a single PE is shared by the user and kernel threads



Summary & Future work

- Traditional CPU-based memory operations are not negligible in system performance and efficiency
- Accelerating `migrate_pages()` with DSA can lead to more efficient memory management and improve overall system performance
- We plan to explore the other kernel components that can benefit from DSA
    - Storage and network related memory copy operations
- Also, we need to search for the right abstraction for such accelerators
    - Coordination for both applications and kernel

Thank You!