

<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper published in . This paper has been peer-reviewed but does not include the final publisher proof-corrections or journal pagination.

Citation for the original published paper (version of record):

Baik, J., Kim, J., Park, C H., Ahn, J. (2025)  
Accelerating Page Migrations in Operating Systems with Intel DSA  
*IEEE Computer Architecture Letters*, 24(1): 37-40  
<https://doi.org/10.1109/lca.2025.3530093>

Access to the published version may require subscription.

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-547285>

# Accelerating Page Migrations in Operating Systems with Intel DSA

Jongho Baik<sup>†</sup>, Jonghyeon Kim<sup>†</sup>, Chang Hyun Park<sup>‡</sup>, and Jeongseob Ahn<sup>§</sup>  
<sup>†</sup>Ajou University <sup>‡</sup>Uppsala University <sup>§</sup>Korea University

**Abstract**—Modern server-class CPUs are introducing special-purpose accelerators on the same chip to improve performance and efficiency for data-intensive applications. This paper presents a case for accelerating data migrations in operating systems with the Data Streaming Accelerator (DSA), a new feature by Intel. To the best of our knowledge, this is the first study that exploits a hardware-assisted data migration scheme in the operating system. We identify which Linux kernel components can benefit from the hardware acceleration, particularly focusing on the kernel subsystems that rely on the `migrate_pages()` kernel function. As the hardware accelerator is not suitable for transferring a small amount of data due to the HW setup overhead, this preliminary study concentrates on the design and implementation of accelerating `migrate_pages()` with DSA. We prototype a DSA-enabled Linux kernel and evaluate its effectiveness through two benchmarks demonstrating real-world page compaction (`kcompactd`) and promotion (`kdamond`) scenarios. In both cases, our prototype demonstrates improved throughput in page migration, benefiting both the kernel subsystem and applications.

**Index Terms**—Data migration, Hardware accelerator, Linux

## I. INTRODUCTION

In today’s data-centric computing, the amount of data generation, transfer, and processing is escalating to unprecedented levels. Traditionally, data movement and processing relies on the CPU, which loads data from the source memory location, into its registers and stores the data from its registers to the destination memory. This is increasingly becoming a bottleneck, impeding system performance and efficiency. To address this issue, modern Intel CPUs incorporate a specialized hardware technique, called Data Streaming Accelerator (DSA), accelerating data movement tasks and freeing CPU resources [3], [9]. It markedly enhances overall system throughput and efficiency. Despite its potential performance benefits, the DSA has not been evaluated in accelerating data movement tasks of operating systems (OSes), such as memory compaction and task (process or virtual machine) migrations.

In this paper, we introduce a new hardware-assisted memory migration method to accelerate data movement tasks that

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) (RS-2024-00460762), funded by the Ministry of Science and ICT (MSIT) and the Swedish Research Council (2023-03272).

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

This is the author version. The final version is available at <https://doi.org/10.1109/LCA.2025.3530093>

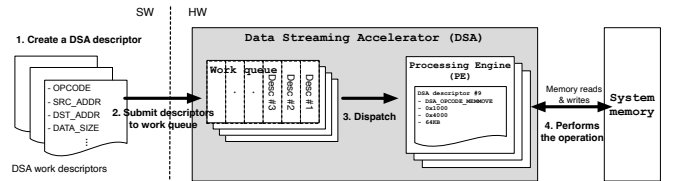


Fig. 1. SW and HW organization of Intel DSA

are currently performed using the CPU. We identify that `migrate_pages()` is the key function in the Linux kernel to benefit substantially from DSA because it is widely used in various kernel subsystems such as memory reclaiming and NUMA balancing [2], [7], [8]. When utilizing DSA for page migration tasks, a technical challenge is to amortize the hardware (DSA) setup overhead which can limit its effectiveness in data movement scenarios, particularly when migrating a small amount of data. Our approach dynamically determines the offloading decision based on the volume of pages, thus optimizing the use of DSA for page migrations.

We have implemented our prototype on Linux kernel version 6.8 and conducted evaluations using two benchmarks. We first select `kcompactd`, which rearranges physical pages to mitigate the memory fragmentation issue, leading to larger contiguous free memory pages [1]. With DSA, the migration throughput of `kcompactd` is increased by 1.2×. Second, we evaluate DSA-assisted page promotions in `kdamond` of DAMON for two applications running in a tiered memory environment. Our revised migration function processes a batch of pages that is more efficient for DSA and outperforms the CPU-based migration. The execution time of `XSbench` is improved by 16%. In future work, we will explore the other potential places for applying DSA and the design of operating systems for such hardware-assisted memory management.

## II. BACKGROUND AND MOTIVATION

### A. Related Work

There are some prior studies on accelerating data migrations in the operating systems. With the introduction of heterogeneous memory and tiered memory systems, accelerating the `migrate_pages()` function is crucial for enhancing the efficiency and performance of the Linux kernel’s memory management system. Yan et al. proposed a parallelization technique to achieve speedup for transferring pages between different types of memory [8]. HeMem utilized the DMA engine to offload the data migration, freeing the CPU resource [7]. Meanwhile, vector instructions such as AVX in x86 have been

widely used in implementing memory copy operations because they can benefit from the parallel processing of multiple data in a single instruction. However, the Linux kernel does not leverage such optimizations due to the additional overhead of saving and restoring the AVX registers.

### B. Intel’s Data Streaming Accelerator

Recently, Intel has unveiled a specialized hardware unit called Data Streaming Accelerator (DSA) for accelerating data movement operations and freeing up CPU cycles. Fig. 1 presents the overview of how DSA works. The DSA hardware unit includes work queues (WQs) that hold DSA descriptors, each of which specifies an operation type (e.g., *copy*, *fill*, *compare*, and *checksum*), source address, destination address, and the size of the data transfer. Intel DSA exposes portals, which are MMIO registers, for the programmers to submit the DSA descriptors. In DSA, the processing engines (PEs) are responsible for executing the operation by fetching a descriptor from the work queue. The engine is designed to read the source data from memory, perform the specified operation, and write the resulting data back to the destination memory [3].

### C. Data Migrations in Linux

Since the introduction of the Data Streaming Accelerator (DSA), researchers have applied this technology in various applications [5]. However, to the best of our knowledge, no attempt has been made to utilize DSA for operating systems (OSes). Since the memory management unit in modern OSes is a page size (e.g., 4KB in the x86 architecture), this preliminary study focuses on accelerating page migrations. TABLE I presents where the page migration function (e.g., `migrate_pages()` in `mm/migrate.c`) is used in the Linux kernel. Data movement tasks in modern OSes, such as memory reclamation, compaction, and migration, are critical as they directly impact system performance and resource management, especially in high-performance computing and large-scale data centers. Thus, accelerating `migrate_pages()` with DSA can enhance the performance of kernel subsystems, leading to more efficient memory management and improved overall system performance. Meanwhile, migrating pages requires TLB shutdown operations. To reduce the migration overhead, Linux v6.3 added features to migrate pages in a batch through `migrate_pages_batch()` which is also utilized in DAMON with a list of pages (folios) [4].

## III. ACCELERATING DATA MIGRATION WITH DSA

In this section, we introduce our design for accelerating data migration, particularly focusing on `migrate_pages()` in the Linux kernel through the use of Intel DSA. We characterize the performance advantages and offloading cost of the DSA feature. Based on the cost model, we design our offloading strategy which opportunistically utilizes the hardware acceleration whenever we predict the memory movement task to benefit from the accelerator.

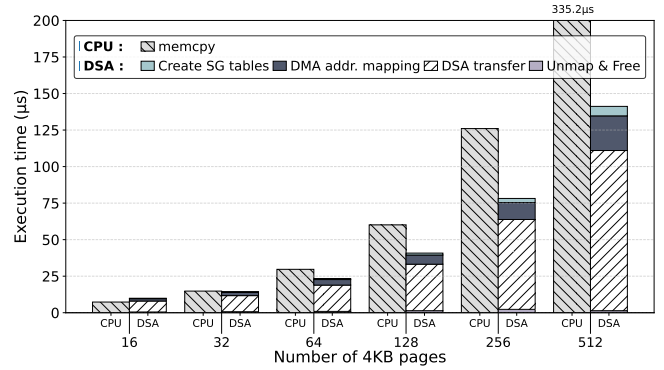


Fig. 2. Performance breakdown of copying pages: CPU vs. DSA

### A. Design Considerations

When looking under the hood of `migrate_pages()`, we observe that the `memcpy()` function is the core primitive for copying data and this is widely used in various kernel components such as device drivers and filesystems. For instance, when transferring data from the kernel to the user in a `read()` system call, the designated size of data in the system call is transferred through `memcpy()`. It is also common in device drivers to copy data from the device-specific memory to the system memory via `memcpy()`. However, instead of directly applying DSA acceleration to `memcpy()`, our first-order consideration is to study and amortize the HW offloading cost. For this reason, this study focuses on `migrate_pages()` as a case study.

### B. Offloading Cost

To exploit the performance benefits of the hardware accelerator, we need to understand the costs associated with offloading. We conducted a performance characterization study for copying pages using CPU and DSA, respectively. Fig. 2 shows the time for copying 4KB pages. In the CPU configuration, we measured the time spent copying pages through `copy_highpage()` in `/include/linux/highmem.h`, which consists of map, copy, and unmap operations. We observe that DSA is preferred when the number of pages exceeds 32. In other words, if the amount of data is not sufficiently large, it is more efficient to use the conventional CPU copying method due to the offloading cost, which may negatively impact the overall throughput of page migrations.

The offloading procedure includes the following steps: (1) we need to create two scatter-gather (SG) tables to keep track of the source and destination pages, respectively as those pages may not be contiguously allocated in the physical memory. Each entry in the SG tables uses the *struct page* pointer types to refer to actual physical pages in the system. (2) We need to configure the DMA engine to map the source and destination pages specified in the scatter lists using the `dma_map_sgtable()` function, producing DMA-capable addresses. Since DSA works on an IO virtual address, a mapping needs to be made before a DSA transfer and unmapped after the completion of the transfer. (3) This enables us to create DSA descriptors through DMA-mapped addresses, and then submit the descriptors sequentially. Once

TABLE I  
migrate\_pages() USED IN THE LINUX KERNEL

Module	Location	Usage
Memory reclaiming (e.g., kswapd)	mm/vmscan.c	Move pages to different memory locations, facilitating the freeing of contiguous blocks of memory and improving memory availability
kcompactd	mm/compaction.c	Relocate pages, compacting memory by consolidating free space into contiguous blocks, which is beneficial for large allocation requests and performance
NUMA balancing	mm/mempolicy.c	Move pages to memory nodes closer to the CPU that accesses them most frequently, improving memory access latency and overall performance
Memory hotplug	mm/memory_hotplug.c	During memory offlining (removing memory), migrate_pages is used to move pages out of the memory regions that are being removed
DAMON [4]	mm/damon/paddr.c	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

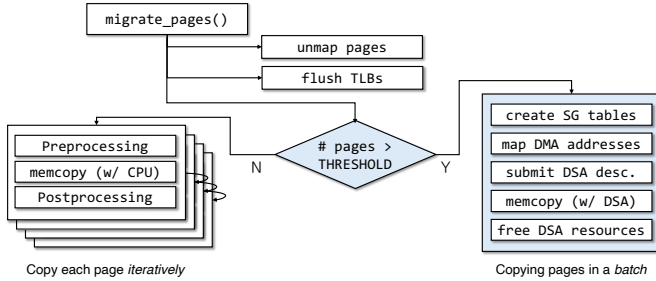


Fig. 3. Workflow of migrate\_pages() with DSA. The blue-shaded region is added in this study.

the copy operation is done, (4) we unmap the DMA-mappings and clean up the SG tables. Such offloading costs are non-negligible when the amount of data to be moved small. Therefore, since the number of pages to be migrated through the migrate\_pages() function depends on the use cases of how many pages are transferred, replacing this function entirely with DSA does not necessarily enhance the performance of page migrations.

### C. Accelerated migrate\_pages() with DSA

To effectively utilize the hardware acceleration, it is important to make an offloading decision based on the amount of data movement. Fig. 3 shows our revised migrate\_pages() function designed to exploit the performance advantages of DSA. We make an additional path (right-hand side of the figure) for migrating pages with DSA and the migration path with CPU remains as a fallback mechanism that is used when the number of pages is less than the threshold (e.g., 32 in our implementation). The CPU path migrates each page iteratively, while we design the DSA path to perform the migrations in a batch to minimize the offloading overhead explained in the earlier section. We build the scatter-gather tables for all the pages to be migrated and then generate DMA-capable addresses at once.

In our implementation, we reserve a DSA engine for the operating system to exclusively use one work queue and one PE unit during boot time. Once a DSA descriptor is submitted to the queue, the corresponding PE unit fetches an item and starts the data movement without the CPU’s involvement. To keep the implementation simple, our migrate\_page operations were run synchronously, where the kernel thread

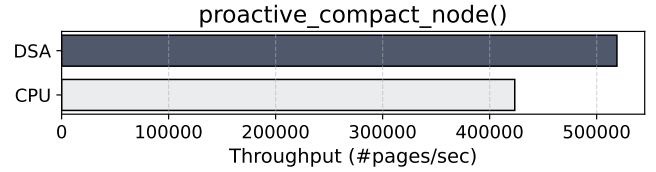


Fig. 4. Throughput of kcompactd: CPU vs. DSA

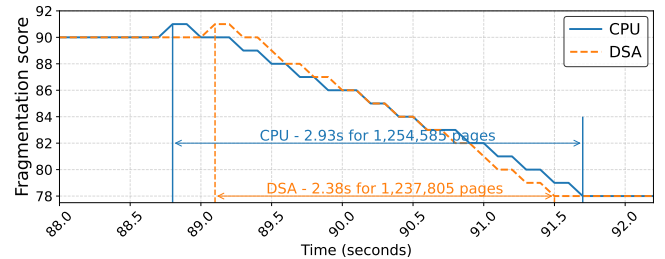


Fig. 5. Changes in the fragmentation score for a compaction period

waited for the completion of the transfer through busy waiting. We will investigate an asynchronous approach switching to other kernel tasks in future work.

## IV. PRELIMINARY RESULTS

### A. Experimental Environment

Our evaluation platform has two Intel Xeon Gold 6430, each of which has one DSA engine. Each CPU socket includes four DDR5 32GB DIMMs, constructing 128GB per socket. The Linux kernel version 6.8 and the Ubuntu distribution 22.04 are used as our baseline and our proposed scheme is implemented on top of the kernel. We configure our DSA engine through the IDXG driver (/drivers/dma/idxd) in the kernel. Our revised kernel can be found at <https://github.com/Sys-KU/DSA-Linux>.

### B. Performance Evaluation

**Benchmark results:** To evaluate the effectiveness of accelerating page movement with DSA, we select two scenarios: memory compaction and memory promotion. First, we measure the page compaction throughput of kcompactd. We design a microbenchmark that allocates 120GB memory with 2MB unit through mmap and then sequentially free the even number page of these 2MB blocks. Once we release about 40GB of memory, we observe that the fragmentation score exceeds the high threshold, which in turn

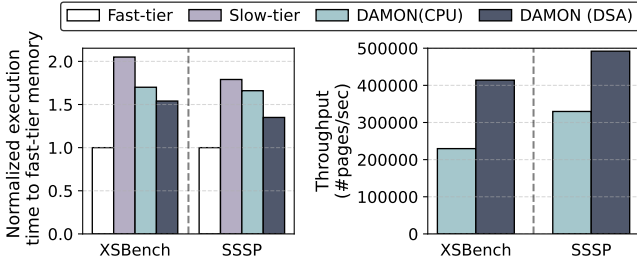


Fig. 6. Performance benefit of DSA in DAMON-based memory tiering [4]

triggers the proactive compaction [1]. We measure the time spent in `proactive_compact_node()`, which invokes `migrate_pages()`. Fig. 4 presents how many pages are processed per second. Compared to the CPU version, DSA shows an improved throughput of  $1.2\times$ , resulting in less time spent in the kernel. Fig. 5 shows the changes in the fragmentation score during a compaction period. We observe that DSA can quickly lower the score back to 78 compared to CPU. DSA completes it in 2.38 seconds while the CPU takes 2.93 seconds. In terms of throughput, DSA processes 520,086 pages per second ( $1,254,585 / 2.38$ ) while CPU completes 428,186 pages per second.

Second, we evaluate page promotions of HMSDK which is used in tiered memory environments [4]. Since the Linux kernel version 6.11, HMSDK has been merged into DAMON of the mainline kernel, so we backport it to our kernel 6.8. As DAMON manages memory in regions that consist of multiple 4KB pages, and are typically larger than 2MB [6], using DSA is beneficial to accelerate the page promotions. To mimic a tiered memory environment, we configure the remote NUMA memory as a slow-tier memory by lowering the uncore frequency of the remote NUMA node to lower the memory bandwidth and increase the access latency. Fig. 6 (left) shows the execution time of XSBench and SSSP workloads when using CPU and DSA versions of DAMON, respectively. Each bar is normalized to the ideal case, where all the allocations are in `fast-tier` memory. The lower-bound performance is modeled where all allocations are made to the `slow-tier` memory. While both CPU and DSA versions with DAMON promotion (`kdamond` improve performance compared to the `slow-tier` case), the DSA version exhibits further improved performance because the throughput of page promotion is increased, as shown in Fig. 6 (right). For SSSP, it improves the execution time by 31% compared to the CPU version.

**Base page vs. large page:** To further understand the performance benefit of DSA, we decompose the migration time into the setup cost and actual data transfer. Fig. 7a compares the performance of migrating 2MB data with the base pages and a large page when using DSA. Although the changes in copying data are negligible, the setup cost for DSA is significantly improved when using large pages. With the base page configuration, we prepare the scatter-gather table for 512 pages. On the other hand, in the 2MB page option, a single large page entry is required in the scatter-gather table. The amount of SG table entries also reduces the time spent in DMA address mappings and unmappings.

**Contention in DSA:** We evaluate the latency changes when

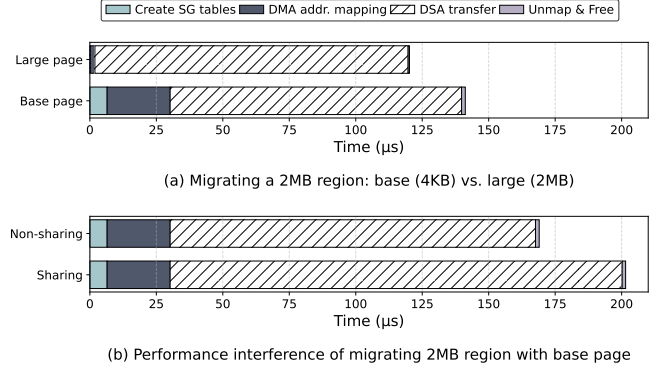


Fig. 7. Time decomposition for migrating a 2MB region

a user application thread and a kernel thread utilize DSA at the same time. We measure the latency of the kernel thread that migrates a 2MB region from one NUMA node to the other NUMA node with the 64KB chunk granularity while configuring the user thread to repeatedly migrate one 64KB chunk at a time. Fig. 7b presents two cases. The `non-sharing` bar presents the case where the kernel and user threads use two different DSA PEs (Processing Engines). Compared to base page (Fig. 7a), the migration time is increased by 25%. Note that the DSA setup cost is not changed while the transfer time increases. When sharing a PE between the user and kernel thread, the `sharing` bar presents a further increased latency of 24% over the `non-sharing` case.

## V. CONCLUSIONS AND FUTURE WORK

This study presented a HW-accelerated page migration scheme for the Linux kernel. We found that entirely replacing the migrate function with DSA was not desirable due to the HW setup overhead. To take advantage of the performance benefits of DSA, we revised the migration function to make the offloading decision based on the amount of data movement. Our evaluation demonstrated the performance benefits in two representative scenarios: `kcompactd` and `kdamond`.

In future work, we will investigate the other places copying data. For instance, DSA can accelerate `filemap_read()`, reading data from the page cache to the user-level buffer since the function is designed to copy a group of pages in a batch. Also, when dealing with large network packets, the network stack uses `skb_frag_foreach_page()` to iterate over the pages associated with a socket buffer. These components can benefit from DSA.

## REFERENCES

- [1] Proactive compaction for the kernel, 2020. <https://lwn.net/Articles/817905/>.
- [2] NUMA balancing: optimize memory placement for memory tiering system, 2021. <https://lwn.net/Articles/849095/>.
- [3] Intel. Intel® Data Streaming Accelerator Architecture Specification, 2022.
- [4] Honggyu Kim. DAMON based 2-tier memory management for CXL memory, 2024. <https://lwn.net/Articles/958323/>.
- [5] Reese Kuper, Ipoom Jeong, Yifan Yuan, Ren Wang, Narayan Ranganathan, Nikhil Rao, Jiayu Hu, Sanjay Kumar, Philip Lantz, and Nam Sung Kim. A quantitative analysis and guidelines of data streaming accelerator in modern intel xeon scalable processors. In *ASPLOS*, 2024.
- [6] SeongJae Park, Madhuparna Bhowmik, and Alexandru Uta. Daos: Data access-aware operating system. In *HPDC*, 2022.

- [7] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. Hemem: Scalable tiered memory management for big data applications and real nvm. In *SOSP*, 2021.
- [8] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. Nimble page management for tiered memory systems. In *ASPLOS*, 2019.
- [9] Yifan Yuan, Ren Wang, Narayan Ranganathan, Nikhil Rao, Sanjay Kumar, Philip Lantz, Vivekanathan Sanjeevan, Jorge Cabrera, Atul Kwatra, Rajesh Sankaran, Ipoom Jeong, and Nam Sung Kim. Intel accelerators ecosystem: An soc-oriented perspective : Industry product. In *ISCA*, 2024.