NDPage: Efficient Address Translation for Near-Data Processing Architectures via Tailored Page Table

Qingcai Jiang*, Buxin Tu* and Hong An
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
Email: {jqc, tubuxin}@mail.ustc.edu.cn, han@ustc.edu.cn

***Abstract*—Near-Data Processing (NDP) has been a promising architectural paradigm to address the memory wall problem for data-intensive applications. Practical implementation of NDP architectures calls for system support for better programmability, where having virtual memory (VM) is critical. Modern computing systems incorporate a 4-level page table design to support address translation in VM. However, simply adopting an existing 4-level page table in NDP systems causes significant address translation overhead because (1) NDP applications generate a lot of address translations, and (2) the limited L1 cache in NDP systems cannot cover the accesses to page table entries (PTEs). We extensively analyze the 4-level page table design in the NDP scenario and observe that (1) the memory access to page table entries is highly irregular, thus cannot benefit from the L1 cache, and (2) the last two levels of page tables are nearly fully occupied.**

Based on our observations, we propose NDPage, an efficient page table design tailored for NDP systems. The key mechanisms of NDPage are (1) an L1 cache bypass mechanism for PTEs that not only accelerates the memory accesses of PTEs but also prevents the pollution of PTEs in the cache system, and (2) a flattened page table design that merges the last two levels of page tables, allowing the page table to enjoy the flexibility of a 4KB page while reducing the number of PTE accesses.

We evaluate NDPage using a variety of data-intensive workloads. Our evaluation shows that in a single-core NDP system, NDPage improves the end-to-end performance over the state-of-the-art address translation mechanism of 14.3%; in 4-core and 8-core NDP systems, NDPage enhances the performance of 9.8% and 30.5%, respectively.

***Index Terms*—Near-Data Processing, Virtual Memory, Address Translation**

I. Introduction

Data-intensive applications spend significant time and energy moving data between the CPU and memory, which causes the "memory wall" problem in modern computing systems [5]. Recent advances in 3D-stacked memory technologies, e.g., High Bandwidth Memory (HBM) [19], enable the practical implementation of **near-data processing (NDP)**, which is a promising solution to alleviate the memory wall problem. By placing computing units within the logic layer of 3D-stacked memory, NDP systems enhance performance and energy efficiency with low-latency and energy-efficient memory accesses [26].

Despite the significant performance and energy-efficiency potential of NDP, system challenges remain a major obstacle to the practical adoption of NDP architectures. Among these challenges, efficient support for **virtual memory** is often considered one of the most significant ones in NDP architectures [18]. Virtual memory allows the operating system to automatically map NDP's virtual memory addresses to the corresponding physical memory addresses. This flexible mapping provides several benefits for the NDP system, including programmer-transparent memory management and enhanced programmability.

Supporting virtual memory comes at the cost of **address translation**, which is a performance bottleneck for data-intensive applications in modern computing systems with a 4-level page table [22]. In this work, we find that the address translation overhead is even more profound in NDP systems if we simply apply the 4-level page table design in NDP systems due to two reasons: (1) The data-intensive NDP applications introduce large amounts of irregular memory accesses, which cause frequent translation lookaside buffer (TLB) misses, thus leading to many page table walks (PTWs). (2) The area and power constraints in NDP limit the cache size and levels in NDP systems, making it hard to effectively cache the recently used page table entries (PTEs).

Our goal in this work is to design an efficient address translation mechanism for NDP systems by adapting conventional page table design to the characteristics of NDP workloads. We aim to develop a practical technique that: (1) is highly efficient in both single-core and multi-core NDP systems, (2) remains transparent to NDP applications, and (3) requires only modest or no hardware modifications and ISA changes.

To this end, we present NDPage, a new, highly efficient, software-transparent address translation mechanism tailored for NDP systems. Our two **key observations** behind NDPage are: (1) The memory access pattern of PTEs, referred to as **metadata** throughout this paper, in NDP systems is highly irregular. This irregularity prevents metadata accesses from benefiting from the cache and significantly pollutes the cache, which negatively impacts the performance of normal data accesses. (2) The flexibility of the 4-level radix page table is unnecessary in NDP systems, as the last two levels of page tables are often fully occupied.

Based on our observations, we devise two **key mechanisms** in NDPage: (1) An L1 cache bypass mechanism for metadata that makes PTEs not cacheable in NDP's L1 cache system, accelerating the memory access of PTEs and preventing them from polluting normal data in the cache system. (2) A flattened L2/L1 page table design that merges the last two levels of the page table in NDP systems to reduce the sequential page table accesses during address translation.

*These authors contributed equally to this work.

We evaluate NDPage with Victima, an extended version of the Sniper simulator [7], [8] used by prior works [21], [22], using 11 data-intensive applications from five diverse benchmark suites. Our evaluation yields 2 **key results** that demonstrate NDPage's effectiveness. **First**, in single-core environments, NDPage improves performance by 34.4% on average over the baseline system using a four-level radix-tree-based page table, yielding 14.3% performance improvements compared with the second-best address translation mechanism. **Second**, in multi-core environments, NDPage improves performance by 42.6% (40.7%) on average over the 4-core (8-core) baseline system using a four-level radix-tree-based page table, yielding 9.8% (30.5%) performance improvements compared with the second-best address translation mechanism.

This paper makes the following key contributions:

- We investigate the problems of applying the conventional 4-level page table design for address translation in Near-Data Processing architectures.
- We propose NDPage, an address translation mechanism specifically designed for NDP architectures, based on our two observations in address translation within NDP systems.
- We evaluate NDPage using a diverse set of data-intensive applications and demonstrate its effectiveness in both single-core and multi-core environments.

II. Background

A. Near-Data Processing Architecture

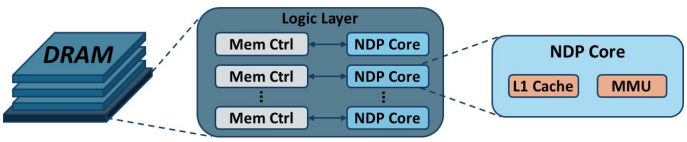

Fig. 1. Overview of NDP architecture.

Near-data processing is a promising technique to alleviate the memory wall in data-intensive applications [15], [26]. By placing the computing unit (referred to as an NDP core) near the main memory, NDP cores can access data with low latency and high throughput. Figure 1 shows a typical NDP architecture based on 3D-stacked memories like the Hybrid Memory Cube (HMC) [10] and HBM [19], [20]. In this architecture, NDP cores are placed in the logic layer of the 3D-stacked memory. An NDP core typically has a **shallow L1 cache** due to two reasons: (1) a strictly limited power and area budget in the logic layer, and (2) the memory access in NDP applications usually exhibits low data locality, which cannot benefit from deep cache hierarchies [6], [29]. Apart from that, to support virtual memory, an NDP core incorporates a memory management unit (MMU), which will be discussed in Section II-B.

B. Virtual Memory

Virtual memory is a fundamental concept in computer systems, designed to provide applications with the illusion of having unlimited memory [4], [12]. This abstraction is crucial for enabling several key functionalities in computing systems, such as process isolation, data sharing, and memory protection. To support these functionalities, **address translation** is essential for mapping virtual addresses to corresponding physical addresses. The page table stores the metadata for these mappings. A commonly used page table mechanism is the radix

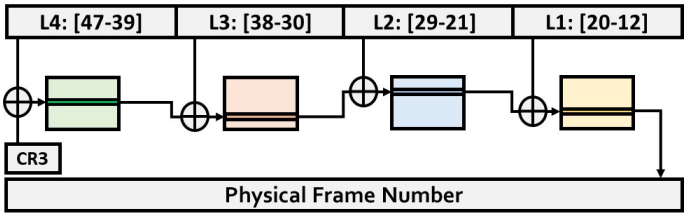

Fig. 2. Address translation in radix page table.

page table [13]. In x86-64 systems, when a virtual address needs to be mapped to a physical address, the operating system allocates memory for the page tables at each level only if they do not already exist. This involves allocating a 4KB page frame for each new table required. The flexibility provided by radix page tables results in significant space savings, which is one of their major advantages. Figure 2 depicts the address translation based on a 4-level radix page table in x86-64 systems. The OS divides the higher 36 bits of the virtual address into four parts and executes four sequential memory accesses to page tables from level 4 to level (PL4 - PL1) to obtain the physical frame number (PFN).

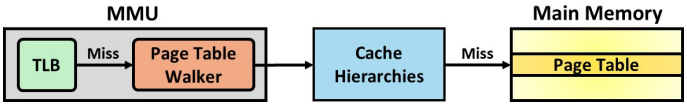

Fig. 3. Address translation workflow in modern processors.

To avoid frequent access to the main memory for the page tables, virtual memory systems usually adopt the MMU hardware to accelerate address translation. As shown in Figure 3, an MMU has two main components: (1) TLB, which caches recently used address translations. (2) PTW, which initiates a page table walk upon a TLB miss, searches for PTEs in the memory hierarchies including the cache, and finally obtains the corresponding physical address.

III. Motivation

Virtual memory provides an essential attraction for programmers to manage physical memory efficiently. In contrast to maintaining a separate physical address space [16], virtual memory allows NDP systems to be compatible with the mainstream programming models and benefit from existing advanced memory management mechanisms [14], [18].

A fundamental aspect of virtual memory functionality is address translation, which facilitates the efficient mapping of virtual addresses to physical addresses. To better understand the performance overhead of address translation in NDP systems, we evaluate the address translation performance metrics and compare them to those of conventional CPU systems.

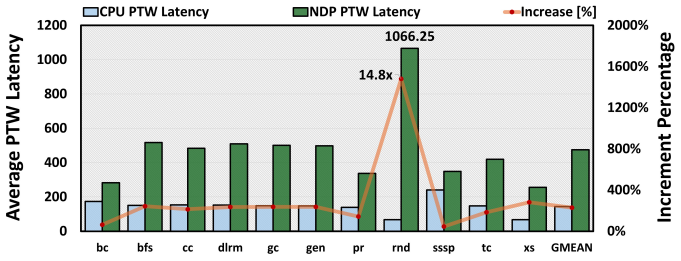

Fig. 4. Average PTW latency in 4-core systems and NDP's PTW latency increment compared with CPU.

Here we take a 4-core system for example*. Figure 4 demonstrates the average PTW latency in 4-core NDP and CPU systems. We observe that the average PTW latency in the NDP system is 474.56 cycles (up to 1066.25 cycles) across all the data-intensive applications, which is 229% higher than in the CPU system.

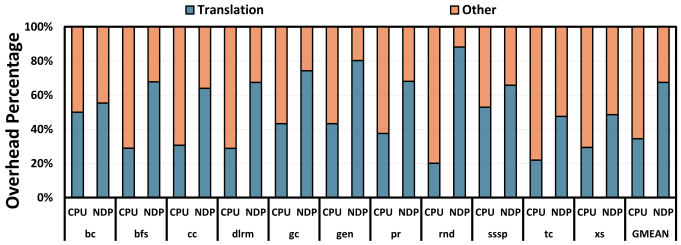

Fig. 5. Percentage of address translation overhead in 4-core systems. Blue bars denote the proportion of the address translation overhead and orange bars denote the execution time apart from address translation.

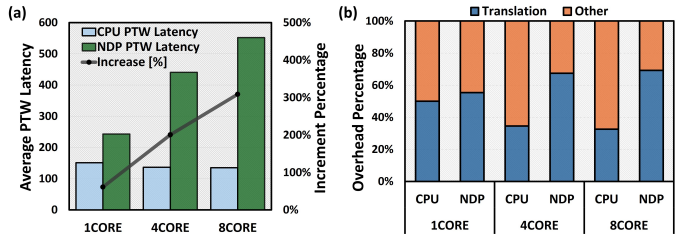

Fig. 6. As the number of cores scales, (a) shows the trend in average PTW latency, and (b) shows the trend in the average percentage of address translation overhead.

Figure 5 shows the percentage of total address translation overhead in 4-core NDP and CPU systems. We observe that the average address translation overhead in the NDP system occupies 67.1% of the total execution time, higher than the 34.51% observed in the CPU system. Furthermore, as the number of cores scales, the address translation overhead in NDP systems increases significantly. As Figure 6 (a) shows, from 1 core to 8 cores, the average PTW latency in the NDP system increases from 242.85 to 551.83 cycles. However, in the CPU system, the average PTW remains similar. Also, as Figure 6 (b) shows, from 1 core to 8 cores, the percentage of address translation overhead in the NDP system continues to increase, whereas in the CPU system, it also remains similar. We conclude that naively applying the address translation mechanism based on a 4-level radix page table on NDP systems creates huge overhead and nullifies the performance benefits of NDP architecture.

*Section VI shows the methodologies

IV. Key Observations

In this section, we study the root cause of the high address translation overhead in NDP systems. we make 2 key observations by analyzing the memory access patterns of the PTEs and the page table occupancy: (1) Although the memory access pattern of the normal data, i.e., the actual data that the program accesses, is irregular, the memory access pattern of **metadata**, i.e., the PTEs required to locate the **normal data**, is **even more irregular** , and (2) when adopting the 4-level radix page table mechanism in NDP systems, the L2 and L1 page tables are often **fully occupied**.

A. The Irregular PTE Accesses

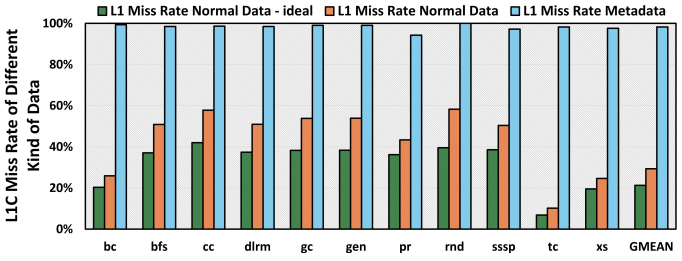

Fig. 7. Comparison of L1 data cache miss rates for normal data (ideal vs. actual) and metadata across different benchmarks in 4-core NDP systems.

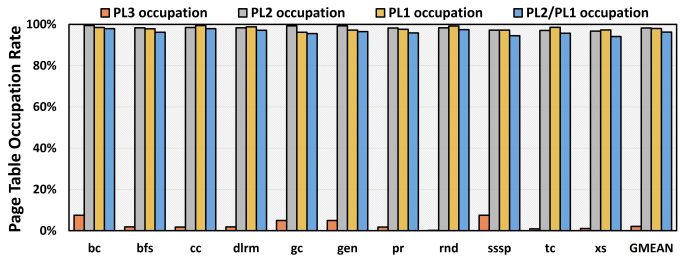

Fig. 8. The page table occupation rates at PL1, PL2, PL3, and combined PL2/PL1 levels for different benchmarks.

We conduct an analysis of PTE access by examining the cases of address translation on a 4-core NDP system. Firstly, we observe that the TLB miss rate reaches 91.27%, and 65.8% of the memory accesses are caused by PTEs. Thus, a significant amount of PTE accesses overwhelms the memory system, starting with the cache hierarchies. We evaluate the L1C miss rate of metadata in 4-core NDP systems. As Figure 7 shows, the blue bar demonstrates that the average PTE miss rate in the L1 cache reaches 98.28%. We conclude that irregular PTE access cannot effectively utilize the cache in NDP systems. What's worse is that excessive PTE misses will lead to an increase in normal data misses. As Figure 7 shows, comparing the green bars and orange bars, we observe that the L1C miss rate of

the normal data in the NDP 4-core system reaches 35.89%, which is 1.37x higher than the 26.16% in the ideal NDP 4-core system with no address translation. As a result, the main memory access caused by PTE in the NDP system increases by 200.4x compared to the CPU 4-core system. To conclude, our first key observation is that the memory access pattern of PTE metadata in an NDP system is **extremely irregular**, which results in the following: (1) the memory access of metadata cannot benefit from the cache, and (2) significantly pollutes the cache, negatively impacting the performance of accessing normal data.

B. Fully Occupied L2 and L1 Page Table

The key benefit of the four-level radix page table mechanism in the conventional system is its flexibility, which results in memory space savings, i.e., the next level of the page table is allocated **only when needed**. However, we conduct a study on the occupancy of four-level page tables in the NDP 4-core system.

As Figure 8 shows, the average occupancy rate of the L2 and L1 page tables reaches 98.24% and 97.97%, respectively, significantly higher than 0.43% and 3.12% of L4 and L3. In other words, the full PL2 and PL1 are **always needed** in NDP systems. The flexibility of the 4-level radix page table is overkill in the last 2 levels of fully occupied page tables, and the additional levels increase sequential accesses to the page tables. To conclude, our second key observation is that in NDP systems, **fully occupied** L2 and L1 page tables negate the necessity of maintaining the 4-level radix-tree structure between them.

These two key observations lead us to two ideas for designing efficient address translation mechanisms for NDP systems.

V. NDPage Design

In this section, we describe the mechanisms of NDPage, including an L1 cache-bypass mechanism for metadata and a novel page table architecture with the flattened L2/L1 page table.

A. L1 Cache-Bypass Mechanism for Metadata

NDPage implements a cache-bypass mechanism only for metadata, i.e., the PTEs, in NDP systems. This mechanism arises from the observation that highly irregular metadata cannot benefit from the cache in NDP systems, as analyzed in Section IV-A. Therefore, NDPage bypasses the PTEs from the NDP cache, directly accessing the main memory. The hardware implementation of the cache-bypass mechanism is based on well-established techniques [9], [25]. We reference these prior works to implement the technology in the hardware design of NDP systems. The key focus is on the careful recognition and selection of data to bypass. We modify the operating system to mark the regions of the PTEs, which is essentially a 4KB memory, i.e., 2^9 entries $\times$ 64 bits/entry, in the 4-level radix-tree page table design. Since 4KB is divisible by 64B, which is the conventional cache line size, making these regions 64B aligned ensures this mechanism does not affect normal data accesses within the cache. In the NDPage design, for each access to the metadata, the OS triggers a special load instruction to look up the PTEs, depending on the architecture. For instance, the x86 ISA provides the PFLD (pipelined floating-point load) [2] instruction to issue the memory request that bypasses the cache system. The cache bypassing violates the assumption of inclusion with inclusive multi-level cache hierarchies [24]. However, since there is only one level of cache in the NDP systems, the violation of the cache bypassing to the inclusive multi-level caches doesn't exist.

B. Flattened L2/L1 Page Table

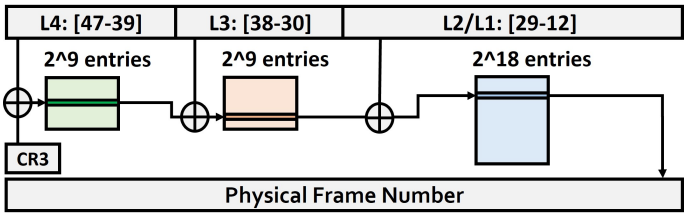

Fig. 9. PTE access flow with new page table design in NDPage.

The conventional radix page table is organized as a 4-level structure for high flexibility. However, as shown in Section IV-B, the 4-level structure is excessive for data-intensive NDP applications since the last two levels of the page table are nearly fully occupied. Based on this observation, we propose a flattened L2/L1 page table. Flattening uses the radix nature of the page table to naturally merge levels into single, larger levels, resulting in a shallower tree. We merge the last two levels of the page table to reduce the number of sequential memory accesses for each PTW from 4 to 3. Figure 9 illustrates the address translation in NDPage with the flattened L2/L1 page table. By combining the L2 and L1 levels, a single 2 MB flattened node replaces each L2 node and its 512 L1 child nodes, encompassing all $2^9 \times 2^9 = 262,144$ entries. Although this increases the size of individual page table entries, the overall impact is minimal due to the small fraction of the page table relative to the actual data size. To support flattened page tables, minor modifications are needed in the system architecture. Specifically, control registers and page table entries require a single bit to indicate flattened nodes. The hardware page walker uses these bits to determine the appropriate index bits from the virtual address for each level. To implement the flattened page tables, we modify the encoding of the virtual address bits for indexing. Each flattened L2/L1 table structure comprises $2^9 \times 2^9 = 262,144$ entries, fitting into a single 2 MB page. Therefore, as shown in Figure 9, 18 bits are required for indexing each combined L2/L1 table.

C. Page Walk Caches in NDPage

The Page Walk Cache (PWC) [3] is a small dedicated cache in modern processors that accelerates address translation by storing recently accessed Page Table Entries (PTEs). In a modern processor, each level of the page table has its own PWC. We also implement the PWC in the NDPage design to further optimize PTE access. We observe that the hit rates of the

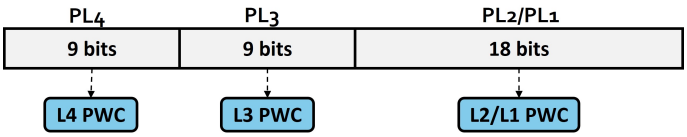

Fig. 10. PWC design in NDPage

PWCs in L4 and L3 are nearly 100% and 98.6%, respectively, while the hit rates in L2 and L1 are relatively low, averaging 15.4% in our evaluation. This result is consistent with previous works [3], [21], [22], which also report very high PWC hit rates in L4 and L3 and low hit rates in L2 and L1. We conclude that NDPage flattens the last two levels of the page table while retaining the advantages of PWCs in L3 and L4. By restricting the high PWC miss rate to a flattened PL2/PL1, instead of two separate PL2 and PL1 levels, NDPage significantly reduces memory accesses.

D. Address Translation Workflow with NDPage

Figure 11 demonstrates the address translation and corresponding data access flow in an NDP system with NDPage. The core initiates a virtual-to-physical address translation via the MMU, which first incurs a TLB lookup. When a TLB miss occurs ❶, the MMU triggers a Page Table Walk (PTW) to look up the page tables in the main memory. Unlike normal data accesses, the metadata accesses caused by the PTW bypass the first-level cache and begin directly at the main memory ❷. The PTW requires three sequential page table accesses: first to the PWCs, and if there is a miss, then to the main memory, progressing from the L4 page table to L3, and finally to the flattened L2/L1 page table ❸. In the end, the page table walker fetches the physical address ❹. Once the address translation is complete, the NDP core initiates the normal data access using the physical address at the L1 cache ❺, and if the L1 cache misses, it proceeds to access the main memory ❻.

VI. Evaluation Methodology

We implement NDPage, including the L1 cache-bypass mechanism for metadata, the flattened L2/L1 page table, and the corresponding PWC design using Victima, the simulator used by a prior related work [21], [22]. Table I summarizes the system configuration in CPU and NDP systems.

Workloads. Table II shows all benchmarks we use to evaluate NDPage. We choose 11 data-intensive applications compatible with NDP architecture which are also used in previous works about address translation [1], [17], [30], [33]. We simulate the execution of 500M instructions per core.

Evaluated Address translation Mechanisms. We evaluate six different mechanisms in NDP systems: (1) Radix: Conventional x86-64 system with a four-level radix-based page table. (2) Elastic Cuckoo Hash Table (ECH) [33]: the state-of-the-art hash-based page table. ECH increases the parallelism of PTE access and reduces PTW latency. (3) Huge Page [11]: the operating system allocates memory in 2MB chunks. (4) Ideal: every address translation request hits the L1 TLB, and the access latency to the L1 TLB is zero. This scenario denotes the performance limit of address translation mechanisms. (5) NDPage: this work.

TABLE I
Simulation Configuration

system	CPU	NDP
Core	1/4/8 x86-64 2.6GHz core(s)	
Cache	L1I/D: 32KB, 8-way, 4-cycle latency	
	L2: 512KB, 16-way, 16-cycle latency L3: 2MB/core, 16-way, 35-cycle latency	No L2 No L3
MMU	L1 ITLB: 128-entry, 4-way, 1-cycle latency L1 DTLB: 64-entry, 4-way, 1-cycle latency	
	L2 TLB: 1536-entry, 12-cycle latency	
Interconnect	Mesh, 4 cycle hop latency, 512-bit link width	
Memory	DDR4-2400, 16GB	HBM2 [20], 16GB

TABLE II
Evaluated Workloads

Suite	Workload	Dataset size
GraphBIG [27]	Betweenness Centrality (BC), Breadth-first search (BFS), Connected components (CC), Coloring (GC), PageRank (PR), Triangle counting (TC), Shortest-path (SP)	8 GB
XSBench [35]	Particle Simulation (XS)	9 GB
GUPS [32]	Random-access (RND)	10 GB
DLRM [28]	Sparse-length sum (DLRM)	10 GB
GenomicsBench [34]	k-mer counting (GEN)	33 GB

VII. Evaluation Results

A. Performance in the Single-Core System

Figure 12 shows the speedup provided by the evaluated address translation mechanisms compared to the Radix baseline in single-core NDP systems. We make two key observations based on the results: (1) On average, NDPage outperforms the second-best address translation mechanism (ECH) by 14.3% and Radix by 34.4%, demonstrating NDPage's capability to mitigate the address translation overhead in the NDP architecture. (2) Although Huge Page also requires only three-level page tables, reducing one sequential page table access compared with Radix, NDPage outperforms Huge Page by 24.4%, highlighting the performance advantage of NDPage's Metadata Bypass mechanism.

B. Performance in the Multi-Core System

Figure 13 and Figure 14 show the execution speedup provided by the evaluated address translation mechanisms in 4-core and 8-core NDP systems, respectively. We make several key observations based on the results: (1) on average, NDPage outperforms the second-best address translation mechanism (ECH) by 9.8% and 30.5% in 4-core and 8-core NDP systems, respectively, demonstrating NDPage's significant performance benefits and scalability with NDP multi-core architectures. (2) In the 8-core system, Huge Page achieves only 90.1% of

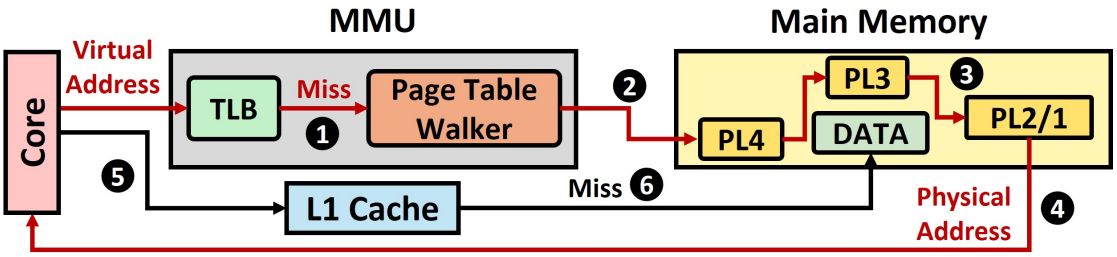

Fig. 11. Address translation and data access flow in an NDP system with NDPage. Red lines indicate the metadata access flow and black lines indicate the normal data access flow.

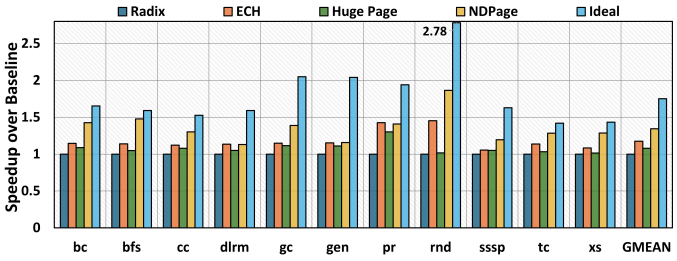

Fig. 12. Speedup provided by ECH, Huge Page, NDPage, and Ideal over Radix in single-core execution.

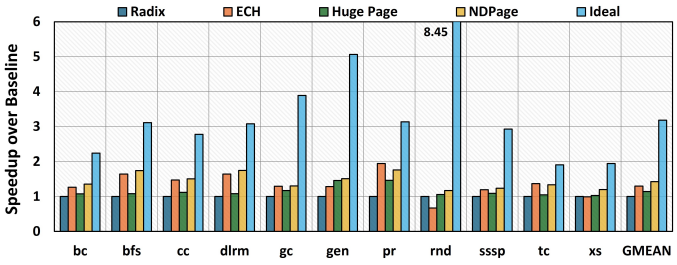

Fig. 13. Speedup provided by ECH, Huge Page, NDPage, and Ideal over Radix in 4-core execution.

the performance of the Radix baseline. We conclude that as the workload scale and the number of NDP cores increase, Huge Page brings increased page fault latency, bloat memory footprint, and rapid consumption of available physical memory contiguity [23]. As a result, Huge Page leads to performance degradation. However, based on our second key observation, NDPage reduces sequential page table accesses while maintaining the flexibility of a 4KB page, performing well (56.2% speedup compared to Huge page on average) in the 8-core NDP system.

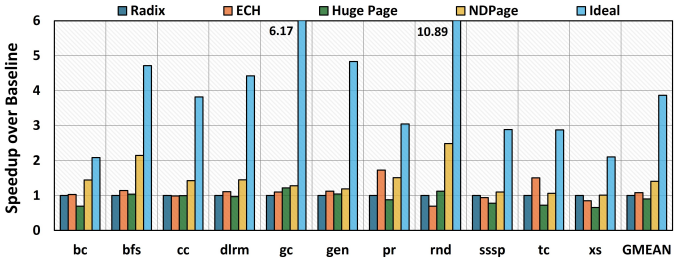

Fig. 14. Speedup provided by ECH, Huge Page, NDPage, and Ideal over Radix in 8-core execution.

VIII. Related Work

To our knowledge, NDPage is the first work to provide a highly efficient, programmer-transparent virtual-to-physical translation mechanism for NDP systems, achieved through a novel page table design with modest hardware modifications. Several prior works have focused on reducing address translation overhead in NDP systems. vPIM [14] optimizes page table accesses using a *network-contention-aware hash* page table and allocates some NDP cores for *pre-translation*. However, the hash-based page table limits certain virtual memory functionalities, such as page data sharing, and requires programmers to modify their applications to conform to the vPIM framework. DIPTA [31] improves page table access by strategically placing page tables closer to the data. Nevertheless, DIPTA restricts page mapping associativity, potentially leading to significant performance degradation in applications that frequently encounter page conflicts. In contrast, NDPage provides a programmer-transparent solution that imposes no restrictions on page mapping.

IX. Conclusion

NDP architecture makes efficient use of high memory bandwidth and is compatible with data-intensive workloads. However, data-intensive workloads experience irregular memory accesses and long-latency page table walks, leading to excessive overheads in address translation using the conventional table mechanism in NDP systems. We propose NDPage, an address translation mechanism that (1) bypasses the L1 cache in NDP systems when accessing PTEs, and (2) flattens the fully occupied L2 and L1 page tables. Our evaluation demonstrates that NDPage provides substantial performance improvements in NDP systems, offering an efficient address translation mechanism.

Acknowledgment

This work is partly supported by Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDB0500102). The computing resources are financially supported by Laoshan Laboratory (LSKJ202300305).

REFERENCES

[1] S. Ainsworth and T. M. Jones. Compendia: reducing virtual-memory costs via selective densification. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on Memory Management*, pages 52–65, 2021.

[2] M. Atkins. Performance and the i860 microprocessor. *IEEE Micro*, 11(5):24–27, 1991.

[3] T. W. Barr, A. L. Cox, and S. Rixner. Translation caching: skip, don't walk (the page table). *ACM SIGARCH Computer Architecture News*, 38(3):48–59, 2010.

[4] A. Bhattacharjee and D. Lustig. *Architectural and operating system support for virtual memory*. Morgan & Claypool Publishers, 2017.

[5] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, et al. Google workloads for consumer devices: Mitigating data movement bottlenecks. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 316–331, 2018.

[6] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, R. Ausavarungnirun, K. Hsieh, N. Hajinazar, K. T. Malladi, H. Zheng, et al. Conda: Efficient cache coherence support for near-data accelerators. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 629–642, 2019.

[7] T. E. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[8] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout. An evaluation of high-level mechanistic core models. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(3):1–25, 2014.

[9] C.-H. Chi and H. Dietz. Improving cache performance by selective cache bypass. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume 1: Architecture Track*, volume 1, pages 277–278. IEEE Computer Society, 1989.

[10] H. M. C. Consortium. Hmc specification 2.0, 2014.

[11] J. Corbet. Transparent Huge Pages in 2.6.38. https://lwn.net/inproceedingss/423584/, 2011.

[12] P. J. Denning. Virtual memory. *ACM Computing Surveys (CSUR)*, 2(3):153–189, 1970.

[13] K. El-Ayat and R. Agarwal. The intel 80386 architecture and implementation. *IEEE Micro*, 5(06):4–22, 1985.

[14] A. Fatima, S. Liu, K. Seemakhupt, R. Ausavarungnirun, and S. Khan. vpim: Efficient virtual address translation for scalable processing-in-memory architectures. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[15] C. Giannoula, N. Vijaykumar, N. Papadopoulou, V. Karakostas, I. Fernandez, J. Gómez-Luna, L. Orosa, N. Koziris, G. Goumas, and O. Mutlu. Syncron: Efficient synchronization support for near-data-processing architectures. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 263–276. IEEE, 2021.

[16] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu. Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture. *arXiv preprint arXiv:2105.03814*, 2021.

[17] S. Gupta, A. Bhattacharyya, Y. Oh, A. Bhattacharjee, B. Falsafi, and M. Payer. Rebooting virtual memory with midgard. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 512–525. IEEE, 2021.

[18] B. Hyun, T. Kim, D. Lee, and M. Rhu. Pathfinding future pim architectures by demystifying a commercial pim technology. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 263–279. IEEE, 2024.

[19] JEDEC. High Bandwidth Memory (HBM) DRAM. https://www.jedec.org/standards-documents/docs/jesd235a, 2021.

[20] JEDEC Solid State Technology Assn. JESD23-5D: High Bandwidth Memory (HBM) DRAM Standard, March 2021.

[21] K. Kanellopoulos, R. Bera, K. Stojiljkovic, F. N. Bostanci, C. Firtina, R. Ausavarungnirun, R. Kumar, N. Hajinazar, M. Sadrosadati, N. Vijaykumar, et al. Utopia: Fast and efficient address translation via hybrid restrictive & flexible virtual-to-physical address mappings. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1196–1212, 2023.

[22] K. Kanellopoulos, H. C. Nam, N. Bostanci, R. Bera, M. Sadrosadati, R. Kumar, D. B. Bartolini, and O. Mutlu. Victima: Drastically increasing address translation reach by leveraging underutilized cache resources. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1178–1195, 2023.

[23] Y. Kwon, H. Yu, S. Peter, C. J. Rossbach, and E. Witchel. Coordinated and efficient huge page management with ingens. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 705–721, 2016.

[24] L. Li, D. Tong, Z. Xie, J. Lu, and X. Cheng. Optimal bypass monitor for high performance last-level caches. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 315–324, 2012.

[25] S. Mittal. A survey of cache bypassing techniques. *Journal of Low Power Electronics and Applications*, 6(2):5, 2016.

[26] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun. A modern primer on processing in memory. In *Emerging computing: from devices to systems: looking beyond Moore and Von Neumann*, pages 171–243. Springer, 2022.

[27] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin. Graphbig: understanding graph computing in the context of industrial solutions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2015.

[28] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[29] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu. Damov: A new methodology and benchmark suite for evaluating data movement bottlenecks. *IEEE Access*, 9:134457–134502, 2021.

[30] C. H. Park, I. Vougioukas, A. Sandberg, and D. Black-Schaffer. Every walk's a hit: making page walks single-access cache hits. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 128–141, 2022.

[31] J. Picorel, D. Jevdjic, and B. Falsafi. Near-memory address translation. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 303–317. Ieee, 2017.

[32] S. J. Plimpton, R. Brightwell, C. Vaughan, K. Underwood, and M. Davis. A simple synchronous distributed-memory algorithm for the hpcc randomaccess benchmark. In *2006 IEEE International Conference on Cluster Computing*, pages 1–7. IEEE, 2006.

[33] D. Skarlatos, A. Kokolis, T. Xu, and J. Torrellas. Elastic cuckoo page tables: Rethinking virtual memory translation for parallelism. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1093–1108, 2020.

[34] A. Subramaniyan, Y. Gu, T. Dunn, S. Paul, M. Vasimuddin, S. Misra, D. Blaauw, S. Narayanasamy, and R. Das. Genomicsbench: A benchmark suite for genomics. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 1–12. IEEE, 2021.

[35] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz. Xsbench-the development and verification of a performance abstraction for monte carlo reactor analysis. *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)*, 2014.