Leveraging Access Port Positions to Accelerate Page Table Walk in DWM-based Main Memory

Hoda Aghaei Khouzani, Pouya Fotouhi, Chengmo Yang, and Guang R. Gao
Department of Electrical and Computer Engineering
University of Delaware
Newark, DE, USA
Email: {hoda, pfotouhi, chengmo, ggao}@udel.edu

***Abstract*—Domain Wall Memory (DWM) with ultra-high density and comparable read/write latency to DRAM is an attractive replacement for CMOS-based devices. Unlike DRAM, DWM has non-uniform data access latency that is proportional to the number of shift operations. While previous works have demonstrated the feasibility of using DWM as main memory and have proposed different ways to alleviate the impact of shift operations, none of them have addressed the performance-critical metadata accesses, in particular page table accesses. To bridge this gap, this paper aims at accelerating page table walk in DWM-based main memory from two innovative aspects. First of all, we propose a new page table layout and leverage the positions of access ports in DWM to differentiate the state of page table entries. In addition, we propose a technique to pre-align the access ports to the positions to be accessed in the near future, thus hiding shift latency to the maximum extent. Since both address translation and context switching are affected by page table access latency, the proposed technique can effectively improve system performance and user experience.**

I. Introduction

The demand for larger memory capacity and higher memory performance keeps increasing as modern applications are more data-centric rather than computation-centric. On the other hand DRAM, the current CMOS-based main memory technology, fails to scale below 20nm due to the constraints of leakage current and capacitor placement [1]. These facts lead the researchers to investigate new non-volatile memory (NVM) technologies as alternatives to DRAM.

One promising NVM technology is *Domain Wall Memory (DWM)*, also known as *Racetrack memory*. DWM uses ferromagnetic nanowires on a silicon substrate and offers ultra-high storage density [2]. Its read and write latency and write endurance are comparable to DRAM [3], which makes it attractive to build DWM-based main memory for next generation computer systems. The main challenge to employ DWM is due to its sequential access structure. Specifically, to read/write data from/to DWM, it is necessary to shift the track containing the data to align it with an access port. As a result, both the data access latency and energy of DWM are proportional to the number of shift operations. While previous research has proposed different ways to alleviate the impact of shift operations on system metrics [4], [5], [6], [7], to the best of our knowledge, no work has addressed the performance-critical page table accesses in DWM-based main memory.

Previously when the application's data set was small, accesses to the page table were limited since more than 99% of address translations are handled by the *translation lookaside buffer* (TLB) [8]. Unfortunately, the trend towards data-intensive applications leads to a larger footprint in memory and the need of more page table entries, thus engendering more TLB misses. Meanwhile, the TLB miss penalty also increases as multiple memory accesses are required to traverse the multi-level page table. For example, in a 64-bit architecture such as x86_64 and modern ARM, at least four memory accesses are needed to handle a TLB miss. The latency of virtual-to-physical address translation can reach to 50% of total execution time [9]. The page table is on the critical path of data accesses. So its access latency is crucial to both the overall memory performance and context switch latency, since upon a context switch the entire data TLB has to be flushed back to the page table. This effect becomes more significant as users expect to run more and more applications on their devices simultaneously.

The goal of this paper is to reduce the latency of page table accesses in a DWM-based main memory. The observation is that not all the fields in a page table entry (PTE) are required per each access. Instead, based on the current state of a PTE, one can determine the upcoming access as well as the exact fields to be accessed, and pre-shift the tracks to align those bits with the access port. This approach accelerates page table accesses from two aspects. First, the alignment shifts are eliminated from the critical path. Second, without reading the PTE, its state is known *a priori* since the position of the access port can be used to interpret the state of PTE before reading the data. This paper also proposes a new approach to place page table pages (called *PTpages*) in DWM-based main memory. To minimize the potential conflicts between the PTEs that share the same track, only PTEs with non-overlapping access time are placed on the same track. Trace driven experiments confirm that the proposed techniques effectively reduce both the latency and the enegy for accessing the page table.

The rest of the paper is organized as follows: Section II provides a brief background on DWM, TLB, and page table. Section III introduces the proposed page table organization in DWM main memory as well as the pre-align and page replacement algorithms. Section IV presents results of our experimental studies, while Section V concludes the paper.

II. Preliminaries

A. Domain Wall Memory

Domain Wall Memory (DWM), also known as Racetrack memory, is a non-volatile memory [2] made of hundreds of millions of ferromagnetic nanowires on a silicon substrate. Each nanowire can store many bits in the form of magnetized regions or domains which are separated from each other by ultra-narrow domain walls. As shown in Fig. 1, to read/write data from/to

978-3-9815370-8-6/17/$31.00

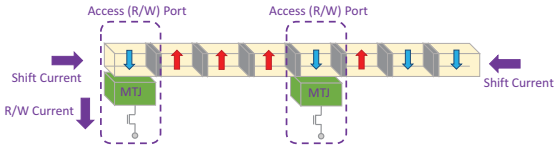

Fig. 1. A DWM cell structure

a track, strongly magnetized ferromagnetic layers called access (R/W) ports are placed at certain fixed positions on the track and are separated by insulator layers. Such structure forms a magnetic-tunnel-junction (MTJ), through which the stored information can be read/written [10]. Since multiple bits share one access port, electronic current pulses can be applied at either end of the track to move the magnetic domains forward or backward to align a certain bit with an appropriate access port. Such access mechanism implies that the access latency and energy of DWM are non-uniform and proportional to the number of shifts. While some work proposes to use spin orbit torque methods [11] instead of current-induced domain wall motions to reduce shift latency, architectural approaches are still needed to mitigate the adverse impact of shift operations.

Since the width of the access transistor is larger than the width of the nanowire, there will be unused space in the layout of one DWM cell. To improve area utilization and increase data storage density, multiple DWM cells are typically overlapped with each other [12] to form a Macro Unit (MU), which is the basic building block of a DWM array. The configuration of MU, denoted as $\text{MU}_{ND\_NP\_NRT}$, is defined by three parameters: ND is the number of domains on each track, NP is the number of access ports in an MU, and NRT is the number of tracks in an MU. In [4], many different MU configurations are studied experimentally, and the results show that $\text{MU}_{64\_32\_4}$ is an efficient configuration that balances area, performance, and energy. The studies in the rest of this paper are based on this configuration.

B. Previous Utilizations of DWM

Previous research has tried to use DWM to implement various components on the memory hierarchy, including register file, cache, and main memory. In addition, researchers have also proposed to use DWM to implement the adder in computational unit [13] or the buffer in a NoC [14]. While these two works exploit the shift property of DWM as a positive factor to develop their implementation, most of the other previous works try to tackle the adverse impact of shifts on access latency.

In [7], DWM is used as GPGPU's register files. The work alleviates the impact of shifts on the execution latency through a register remapping and wrap scheduling algorithm. The logical-to-physical register mapping is rearranged to reduce shift distances. In [5] and [15], DWM is used as a cache. The former adds a few low-cost read-only ports to each track to reduce the performance-critical read latency, while the latter leverages compression techniques to reduce the number of bits to be traversed. The scratchpad memory (SMP) in [16] combines DWM cells with STT-RAM cells to optimize both area and performance. Data that tend to cause more shifts in DWM are stored in STT-RAM. It further proposes an algorithm to schedule the instructions that access the same data in DWM adjacently. In [4], DWM is used as main memory, and data access latency is reduced through transposing the rows and columns in the memory sub-arrays. Instead of placing all the 8 bits of a byte on the same racetrack, it distributes the bits on 8 racetracks at aligned positions, allowing all the bits of a byte to be read or written in parallel.

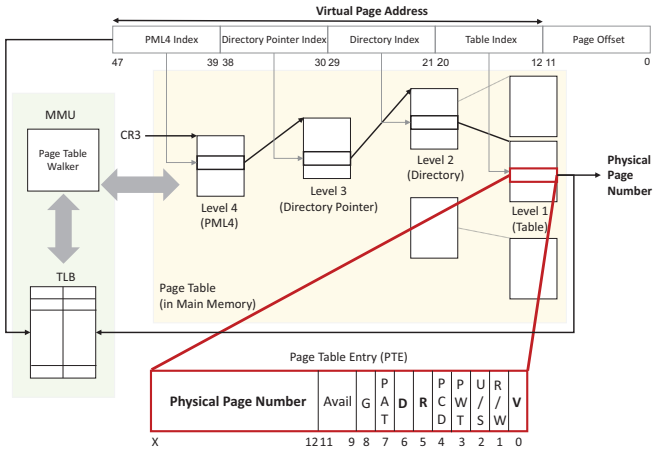

Fig. 2. Page table walk in x86_64 architecture

All the aforementioned works treat both data and metadata equally, without considering the distinct characteristics and behavior of metadata accesses. In contrast, this paper focuses on accelerating page table accesses in DWM. It leverages the different positions of access ports to distinguish between the various states of PTEs, and pre-aligns the access ports to accelerate the performance-critical address translation and context switch.

C. Page Table and Translation Lookaside Buffer (TLB)

Page table, which records the virtual-to-physical address mapping, is on the critical path of memory accesses. Due to the considerably large size of page table, modern systems usually adopt a hierarchical organization, which makes the translation go through a series of memory accesses called *page table walk*. As a concrete example, Fig. 2 shows the page table organization in x86_64, which has four levels of indexing and hence requires the accesses to four different PTpages to perform one address translation.

The *page table walk* procedure is performed by the memory management unit (MMU). It returns a page table entry (PTE) that contains the physical page number (PPN) as well as 12 control bits, as shown in Fig. 2. The size of the PPN is determined by the size of the main memory. For a 4GB main memory which has 2^{20} physical pages, the PPN field is 20 bits. Among the control bits, the most frequently accessed ones are the *Validity (V) bit*, which indicates whether the page locates in the memory or not, the *Referenced (R) bit*, which indicates whether the page is recently accessed or not, and the *Dirty (D) bit*, which indicates whether a writeback to the secondary storage is necessary upon replacing the page.

To hide the latency of *page table walk* to the maximum extent, many systems use instruction and data *translation lookaside buffers* (TLBs) to store recently accessed PTEs. In

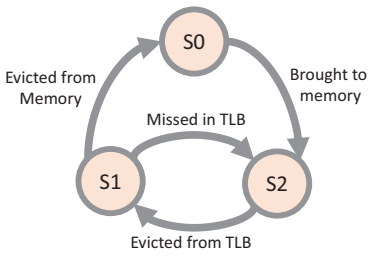

Fig. 3. PTE states and inter-state transitions

TABLE I. FIELDS ACCESSED IN EACH PTE ACCESS

Transition	Actions	Accessed Field in PTE
$S0 \rightarrow S2$	Write	The entire entry
$S2 \rightarrow S1$	Update	Referenced (R) bit and/or Dirty (D) bit
$S1 \rightarrow S2$	Read	The entire entry
$S1 \rightarrow S0$	Update	Validity (V) bit

the presence of TLB, the page table is accessed only upon TLB misses. Unfortunately, with the increasing application footprint and application count, TLB misses are becoming more common, which again makes page table access latency critical.

III. PROPOSED SCHEME

A. Technical Motivation

In a DWM-based main memory, the default page table access policy incurs a large number of shift operations. Specifically, to access a PTE, first it is necessary to shift the track to align the access port with the head of the PTE, then the track needs to be shifted bit by bit to read/write the entire PTE. This uniform and inefficient access policy is due to the fact that prior to the access request, the MMU is not aware of the exact PTE state and hence cannot determine whether all the bits in the PTE have to be accessed or not.

However, our observation is that not all PTE fields need to be accessed for each PTE access. In a system equipped with TLB, a PTE has three different states that are shown in Fig. 3 together with the inter-state transitions. At the beginning, the page is not in main memory and the PTE state is $S0$. Upon a page fault, OS brings the page into memory and the PTE is cached in TLB. This state is represented as $S2$. When there is no more space in TLB, one PTE in it, typically the least recently used one, is evicted and the PTE state becomes $S1$. In that state, the PTE can either transit to state $S2$ when it is brought back to TLB, or transit to state $S0$, if the page is evicted from memory due to the lack of space. The fields in PTE which are necessary to access during each state transition are summarized in Table I. As can be seen, not all the state transitions need to access all the fields in the PTE. In particular, $S2 \rightarrow S1$ and $S1 \rightarrow S0$ transitions only need to update two bits and one bit, respectively.

Based on the analysis above, the shift operations performed to access the page table in DWM can be classified into three categories. The *necessary shifts* are required to read/write the PTE fields listed in Table I. The *alignment shifts* are performed to align the access port with the head of the PTE. Finally, the

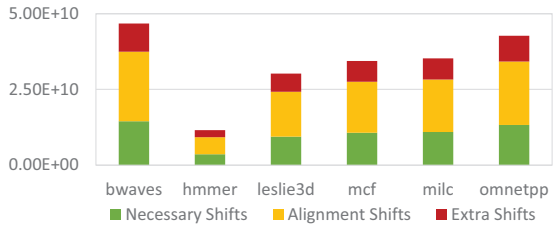

Fig. 4. Classification of shift operations to access page table in the baseline DWM-based main memory

extra shifts are performed on the fields that are unnecessary to access, i.e., the fields that are not listed in Table I. To quantitatively illustrate the potential to reduce the shifts, a preliminary study is conducted to classify the shifts performed under the default access policy. Traces of six SPEC 2006 benchmarks are extracted using Pin [17], with a pair of 4-way associative 128-entry instruction and data TLBs modeled in Pin during trace extraction. The different types of shift operations in the baseline organization are shown in Fig. 4. As can be seen, only 31% of shifts are the *necessary shifts*, indicating the existence of significant potential for reduction. The *alignment shifts* account for 50% of all the shifts, thus dominating PTE access latency. Performing these shifts ahead of the access will eliminate its adverse impact on latency. Finally, the remaining 19% of shifts are *extra shifts*. If the MMU can predict the state of each PTE *a priori*, these shifts can be eliminated as well.

B. PTE State-aware Access Pre-alignment

The proposed technique leverages access port positions for two purposes: differentiating PTE states and reducing the shifts during access. Precisely, since the PTE states are known *a priori* and the type of the upcoming access is predictable based on the current PTE state, it is possible to pre-shift the track so as to align the required fields with the access ports. The proposed technique therefore adopts the following pre-align policy:

- State $S2$. As shown in Fig. 3 and Table I, the upcoming access to the PTE is a TLB replacement, which causes a transition to state $S1$ and only R and D bits need to be accessed. These two bits are placed adjacently in the x86 PTE organization. Therefore, the R bit will be aligned with the access port, which precludes the need for any *alignment shift* or *extra shift* upon TLB replacement.
- State $S1$. The upcoming access could be either a transition to state $S2$ or a transition to state $S0$, as shown in Table I. However, in either case, the V bit which locates at the head of the PTE needs to be accessed. Therefore, by aligning the V bit with the access port, no *alignment shift* or *extra shift* is needed in either case.
- State $S0$. The upcoming access is a transition to state $S2$ and the entire PTE entry needs to be written. However, since state $S1$ already aligns the V bit with the access port, a different position is needed for this state. To minimize shifts, the bit adjacent to the V bit is aligned with the access port, and one *alignment shift* is performed upon a page fault.

Benefits of the proposed pre-align policy are illustrated in Table II, which compares the default and the proposed access

TABLE II. THE NUMBER OF DIFFERENT SHIFTS IN EACH TRANSITION W/ AND W/O PROPOSED SCHEME

Transition	Default			Proposed				Savings	
	necessary	extra	align	necessary	extra	align	pre-align	latency saving	energy saving
$S0 \rightarrow S2$	MaxShift	MaxShift	2×MaxShift	MaxShift	0	1	MaxShift-Pos(R)	3×MaxShift-1	2×MaxShift-Pos(R)-1
$S2 \rightarrow S1$	1	MaxShift-1	MaxShift	1	0	0	Pos(R)-Pos(V)	2×MaxShift-1	2×MaxShift+Pos(V)-Pos(R)-1
$S1 \rightarrow S2$	MaxShift	0	MaxShift	MaxShift	0	0	MaxShift-Pos(R)	MaxShift	Pos(R)
$S1 \rightarrow S0$	0	MaxShift	MaxShift	0	0	0	1	2×MaxShift	2×MaxShift-1

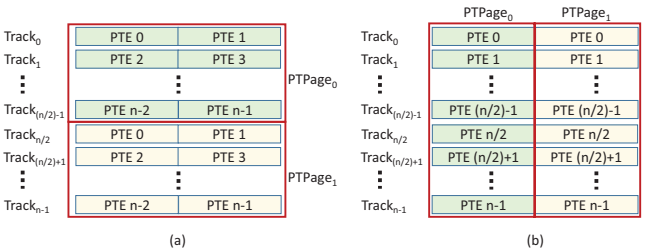

Fig. 5. PTpage placement: (a) contiguous placement of PTEs in one PTpage, (b) stride-k placement, k is number of PTEs on one track.

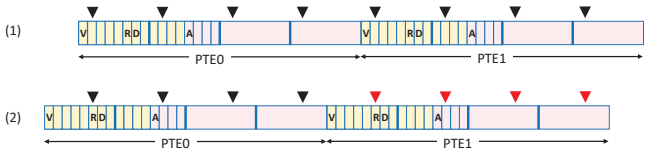

Fig. 6. Conflict between PTEs on the same track: (1) both PTEs are invalid (state $S0$) and access port is aligned with the bit adjacent to the V bit; (2) PTE0 is cached in TLB (state $S2$) and access port is aligned with the R bit of PTE0. Yet PTE1 is still in state $S0$.

policies regarding the numbers of the three types of shifts defined before as well as *pre-alignment shifts*. "MaxShift" equals the distance between two adjacent access ports on the same racetrack. Given the MU configuration characterized in Section II-A, this value equals $ND \times NRT/NP$. "Pos(X)" refers to the position of X bit on the track modulo MaxShift. As can be seen, by performing pre-alignment shifts, the proposed technique completely eliminates *extra shifts* and removes *alignment shifts* from PTE access latency. Savings in latency and energy of each transition are reported in Table II as well. They are computed using the following equations:

$$\Delta_{latency} = \Delta_{necessary} + \Delta_{extra} + \Delta_{alignment} \quad (1)$$

$$\Delta_{energy} = \Delta_{latency} - \#_{pre-align} \quad (2)$$

Table II confirms that the proposed technique effectively accelerates address translation by reducing TLB miss penalty, represented as the latency and energy savings in the $S0 \rightarrow S2$ and $S1 \rightarrow S2$ transitions. The proposed technique also accelerates context switches, which flush the PTEs from TLB and hence induce $S2 \rightarrow S1$ transitions to the PTE states.

It is worth mentioning that the pre-alignment technique proposed for PTEs can be applied to not only the PTEs in the last-level page table pages (called *PTpages*), but also entries in the higher levels of PTpages. Although these entries do not have R or D bit in their control sections, they are also cached in TLB and hence have the three states of $S0$, $S1$, and $S2$ as well. These states can be determined by aligning either the V bit or its adjacent bits with the corresponding access port.

C. PTE and PTpage placement

When the size of a PTE is smaller than the size of a racetrack, more than one PTE can share one track to improve storage density. Intuitively, PTEs in one PTpage can be stored continuously along the tracks, as shown in Fig. 5(a). Unfortunately, this placement degrades the performance of the proposed pre-alignment technique. Specifically, neighboring PTEs in a PTpage display high spacial locality and they are usually accessed sequentially. However, since the distance between access ports on the same track is fixed, if the PTEs sharing the same track are not in the same state, their pre-alignment requirements conflict and only the need of one PTE can be satisfied. To illustrate this issue, Fig. 6 shows an example of PTE0 and PTE1. At the beginning, both of them are invalid (state $S0$) and the bits adjacent to their V bits are aligned with the access ports, as shown in Fig. 6(1). Upon an access to PTE0, the corresponding page is brought to the memory, and PTE0 is cached in TLB and its state becomes $S2$. According to the pre-alignment policy, the track will be shifted to align the R bit of PTE0 with the access port, as shown in Fig. 6(2). However, this conflicts with the need of PTE1 which is still in state $S0$. Giving the privilege to one PTE will lead to incorrect interpretation of the states of the other PTE, incurring extra shifts and alignment shifts during following accesses.

To tackle this issue, this paper proposes a stride-k placement policy for PTEs in one PTpage, as shown in Fig. 5(b). In this organization, a track capable of storing k PTEs is shared among k PTpages. PTEs in one PTpage do not have any conflict, but PTEs in different PTpages may conflict. Since those PTEs are less likely to be accessed sequentially, this organization effectively reduces the shifts induced by PTE conflicts. To further reduce conflicts between PTpages, one can exploit the co-running constraints in the system. Precisely, context switches divide the CPU time between different processes, and in each period only one process is actively executed in a core. Accordingly, placing PTEs of different processes on the same track will help reduce the conflicts bewteen neighboring PTEs.

Under the proposed stride-k PTE placement policy, the PTpage replacement algorithm can be either stride-based or global. Taking Fig. 5(b) as an example, assume the green space and the yellow space respectively belong to two different processes A and B. When process A requests a new PTpage and there is no free page in the green area, the stride-based replacement algorithm, denoted as *stride-based clock*, is constrained to select a green page to evict. Same constraint applied to the process B as well. This policy ensures low runtime correlation among neighboring pages, with the drawback of possibly inducing more page faults. The alternative is to adopt a global replacement algorithm [18], called *global clock*, which searches both the green and the yellow space for a least recently accessed page to hold an incoming PTpage of either process.

TABLE III. BENCHMARKS IN EACH GROUP

Group Name	benchmarks
Group1	gemsFDTD, gobmk, gromacs, h264ref
Group2	astar, bwaves, bzip, cactus
Group3	calculix, deal, gamess, gcc
Group4	hmmer, zeusmp, leslie3d, libquantum
Group5	mcf, milc, namd, omnetpp
Group6	perlbench, sjeng, soplex, tonto

TABLE IV. IMPLEMENTED SCHEMES

Name	Pre-aligned	Placement	PTpage Allocation
Baseline	No	Fig. 5(a)	Global Clock
Cont-Plc	Yes	Fig. 5(a)	Global Clock
Strd-2-Plc	Yes	Fig. 5(b)	Global Clock
Strd-2-Plc+StrdClk	Yes	Fig. 5(b)	Stride-Based Clock

Finally, in the proposed DWM-based main memory, there is a clean cut between the storage space of data and metadata. In other words, data pages and PTpages are not allowed to share the same set of tracks. We enforce this constraint because of the different access characteristics of data and metadata: for data accesses, all the bits are read/written sequentially, *au contraire* to PTEs whose states are predictable and the exact bits to access can be pre-aligned with the access port using the proposed policies. Accordingly, part of the DWM-based main memory is dedicated to storing PTpages, and the size of the reserved space will be dynamically adjusted based on the numbers of data page faults and metadata page faults.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

We have developed a memory management system to simulate the proposed DWM-based main memory and page table organization. The memory size is 4GiB, while the upper limit of page table size within the memory is set to 2MiB. The DWM organization is $MU_{64\_32\_4}$, reported in [4] as the most efficient one that balances area, performance, and energy.

The evaluation set is composed of 24 SPEC 2006 benchmarks, which are relatively memory-intensive and engender non-trivial amount of TLB misses. The page table access trace of each benchmark is collected with Pin [17], including the accessing address, the operation type (eviction, read, or flush), and the corresponding instruction count which is used to simulate execution time. A pair of 4-way associative 128-entry instruction and data TLBs are modeled in Pin when extracting these traces. Every four benchmarks are grouped together to simulate a multi-process environment. A round robin policy is used to simulate OS context switches at the frequency of once per million cycles. Table III lists benchmarks in each group.

B. DWM Shift Reductions

Four different schemes have been implemented, with their specifications summarized in Table IV. *Baseline* stores PTEs of a PTpage continuously as shown in Fig. 5(a), does not perform pre-alignment, and uses global clock algorithm as the page allocation policy. *Cont-Plc* applies the proposed pre-alignment to baseline. Both *Strd-2-Plc* and *Strd-2-Plc+StrdClk* use the proposed pre-alignment and stride-k PTpage placement in shown Fig. 5(b). The former uses the global clock page replacement algorithm, while the latter uses stride-based clock described in Section III-C.

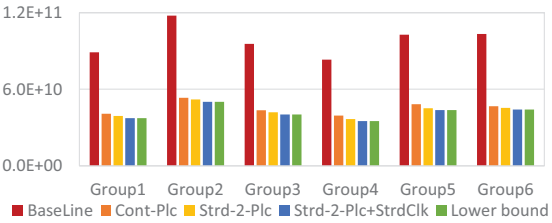

Fig. 7. Number of shifts that affect page table access latency

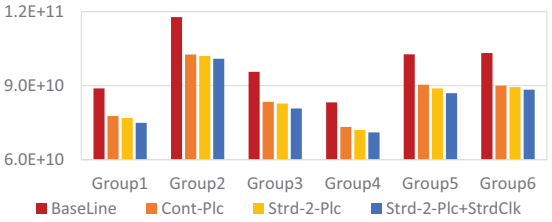

Fig. 8. Number of shifts that affect page table access energy

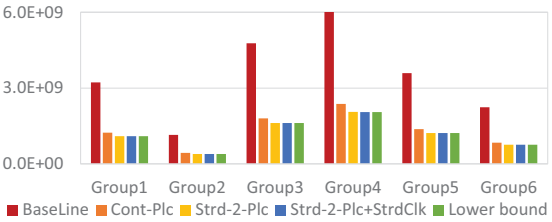

Fig. 9. Number of shifts that affect context switch latency and energy

1) Page Table Walk: Fig. 7 shows the total number of shift operations that affect the latency of accessing the page table. The four schemes are presented together with the *lower bound*, which only includes necessary shifts and fully eliminates alignment shifts, extra shifts, as well as inter-PTE conflicts. By just adopting the proposed pre-alignment policy, *Cont-Plc* is able to reduce the shifts in baseline by 54% on average. *Strd-2-Plc* achieves 1% more reduction on top of *Cont-Plc*, while *Strd-2-Plc+StrdClk* achieves 4% more reduction than *Strd-2-Plc*. These results show that the proposed stride-k organization achieves better performance when applied together with stride-based page replacement. The overall reduction in shift operations in 69% (achieved by *Strd-2-Plc+StrdClk*), whose difference to the theoretical lower bound is negligible.

The number of shifts that affect the energy consumed in page table accesses are shown in Fig. 8. Compared to latency reduction, the amount of energy reduction is less because pre-alignment does not completely eliminate *alignment shifts*, but instead performs them prior to the access. As shown in Equation (2), pre-alignment shifts contribute to energy but not latency. Overall, the average energy reduction achieved by *Cont-Plc*, *Strd-2-Plc*, and *Strd-2-Plc+StrdClk* are 12%, 13%, and 15%, respectively.

2) Context Switch: The shift operations performed to flush the TLB during context switches are presented in Fig. 9. These shifts constrain both the latency and energy consumption of context switches. Same as before, pre-alignment has the most significant impact, shown in *Cont-Plc* which already reduces baseline context switch overhead by 66% on average. *Strd-2-*

TABLE V. DRAM AND DWM PARAMETERS

Device	Parameters
DRAM	4 GiB, 16 banks, 32 nm Access Latency: 37.85 ns Energy to read/write a word: 1.81 nJ Total standby leakage power: 4.57 W
DWM	MU configuration: $MU_{64\_32\_4}$ Latency: Read 0.46 ns, Write 5.18 nJ, Shift 0.5 ns Energy: Read 0.037 nJ, Write 0.46 nJ, Shift 0.31 nJ Total standby Leakage Power: 163 mW

TABLE VI. DRAM-BASED VS. DWM-BASED PAGE TABLES

		Traditional DRAM-based	Proposed DWM-based
Page Table Walk Latency (ns)	Ave.	146.18	23.31
	Max.	227.10	136.32
Page Table Walk Energy (nJ)	Ave.	674.36	10.61
	Max.	1048.7	40.70
Context Switch Latency (μs)	Ave.	10.51	1.58
	Max.	32.09	4.82
Context Switch Energy (μJ)	Ave.	48.47	0.64
	Max.	148.03	1.84

Plc achieves 5% more reduction on top of *Cont-Plc*, indicating that the stride-k placement is effective in preserving PTEs of other processes when one process is running. The difference between the global clock and stride-based clock is negligible since both of them are very close to the lower bound.

C. Comparison with DRAM-based Page table

The proposed DWM-based page table, specifically the *Strd-2-Plc+StrdClk* scheme, is compared against the traditional DRAM-based implementation. Table V shows the latency and energy parameters of DRAM and DWM. The former is retrieved with CACTI [19], while the latter is reported in [12].

Table VI compares the traditional DRAM-based and the proposed DWM-based page tables in terms of the latency and energy of page table walk and context switch. For each metric, the average and maximum values across all groups are computed and reported. It can be observed that the proposed DWM-based page table is much faster than the traditional DRAM-based one. Using it, the average and maximum latencies of page table walk are respectively reduced by 84% and 40%, while the both the average and maximum latencies of context switch are reduced by 85%. The DWM-based page table is also more energy efficient. The average and maximum amount of DWM energy consumed in page table walk are respectively reduced by 98% and 96% compared to DRAM, while both the average and maximum amount of DWM energy consumed in context switch are reduced by 98%. Such significant reduction is mainly due to the impact of standby power. While the dynamic energy of two devices is at about the same level, the leakage power of DWM, as reported in Table V, is orders of magnitude lower than that of DRAM.

V. CONCLUSIONS

This paper has presented a scheme to accelerate address translations in main memory implemented with Domain Wall Memory (DWM), which is a promising memory technology with non-uniform access latency proportional to the number of shift operations performed during the access. To reduce the latency of page table walk, the proposed scheme leverages the positions of access ports in DWM to distinguish between states of a page table entry (PTE), and pre-aligns the access port with the exact bit(s) to be accessed next. A stride-k PTE placement strategy and a stride-based page replacement algorithm are also proposed, aiming at reducing the conflicts between PTEs that share the same track. Experiments on SPEC 2006 benchmarks show that the proposed DWM-based page table is able to reduce the number of shifts performed during page table walk by 69% and speed up page table walk by 84% compared to a DRAM-based page able. It also reduces the shifts in context switch by 71% and speed up context switch by 84% compared to DRAM. These results confirm the advantage of DWM-based main memory over DRAM and the ability of the proposed scheme in accelerating metadata accesses in DWM.

REFERENCES

[1] ITRS, "International Technology Roadmap for Semiconductors," Emerging Research Devices (ERD). [Online]. Available: http://www.itrs2.net/2013-itrs.html

[2] S. S. Parkin, M. Hayashi, and L. Thomas, "Magnetic Domain-Wall Racetrack Memory," *Science*, vol. 320, pp. 109–194, Apr. 2008.

[3] M. H. Kryder and C. S. Kim, "After Hard DrivesWhat Comes Next?" *IEEE Trans. on Magnetics*, vol. 45, pp. 3406–3413, Oct. 2009.

[4] Q. Hu, G. Sun, J. Shu, and C. Zhang, "Exploring Main Memory Design Based on Racetrack Memory Technology," in *ACM Great Lakes Symposium on VLSI (GLVLSI)*, 2016, pp. 397–402.

[5] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "TapeCache: A High Density, Energy Efficient Cache Based on Domain Wall Memory," in *Proc. Intl. Symp. Low Power Electron. & Design (ISLPED)*, 2012, pp. 185–190.

[6] S. Motaman, A. Iyengar, and S. Ghosh, "Synergistic Circuit and System Design for Energy-efficient and Robust Domain Wall Caches," in *Proc. Intl. Symp. Low Power Electron. & Design (ISLPED)*, 2014, pp. 195–200.

[7] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. Li, "Exploration of GPGPU Register File Architecture Using Domain-wall-shift-write based Racetrack Memory," in *Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[8] D. A. Patterson and J. L. Hennessy, *Computer Organization And Design. Hardware/Software Interface*. Morgan Kaufmann Publishers, 2009.

[9] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Unsal, "Redundant Memory Mappings for Fast Access to Large Memories," in *Intl. Symp. Comput. Archit. (ISCA)*, 2015, pp. 66–78.

[10] S. Motaman, A. Iyengar, and S. Ghosh, "Domain Wall Memory- Layout, Circuit and Synergistic Systems," *IEEE Trans. on Nanotechnology*, vol. 14, Mar. 2015.

[11] Y. Zhang, W. Zhao, J.-O. Klein, C. Chappert, and D. Ravelosona, "Peristaltic perpendicular-magnetic-anisotropy racetrack memory based on chiral domain wall motions," *J. of Physics D: Applied Physics*, vol. 48, Feb. 2015.

[12] Z. Sun, W. Wu, and H. Li, "Cross-Layer Racetrack Memory Design for Ultra High Density and Low Power Consumption," in *Design Autom. Conf. (DAC)*, 2013, pp. 1–6.

[13] H.-P. Trinh, W. Zhao, J.-O. Klein, Y. Zhang, D. Ravelsona, and C. Chappert, "Magnetic Adder Based on Racetrack Memory," *IEEE Trans. on Circuits and Syst. (TCS)*, vol. 60, Jun. 2013.

[14] D. Kline, H. Xu, R. Melhem, and A. K. Jones, "Domain-Wall Memory Buffer for Low-Energy NoCs," in *Design Autom. Conf. (DAC)*, 2015.

[15] H. Xu, Y. Li, R. Melhem, and A. K. Jones, "Multilane Racetrack Caches: Improving Efficiency through Compression and Independent Shifting," in *Asia & South Pacific Design Autom. Conf. (ASP-DAC)*, 2015, pp. 417–422.

[16] S. Gu, E. H.-M. Sha, Q. Zhuge, Y. Chen, and J. Hu, "Area and Performance Co-optimization for Domain Wall Memory in Application-specific Embedded Systems," in *Design Autom. Conf. (DAC)*, 2015.

[17] "Programmable Instrumentation - A Dynamic Binary Instrumentation Tool," 2012. [Online]. Available: http://software.intel.com/en-us/articles/pintool

[18] A. Tanenbaum, *Modern Operating Systems (Second Edition)*. Prentice Hall, 2001.

[19] S. J. E. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE J. Solid-State Circuits*, vol. 31, pp. 677–688, 1996.