Racetrack Queues for Extremely Low-Energy FIFOs

Donald Kline, Jr., *Member, IEEE*, Haifeng Xu, Rami Melhem, *Fellow, IEEE*, and Alex K. Jones, *Senior Member, IEEE*

***Abstract*— Networks-on-chip (NoCs) have become a leading energy consumer in modern multicore processors, with a considerable portion of this energy originating from the large number of virtual channel first-in–first-out (FIFO) buffers. Given this motivation, we propose control schemes that leverage the "shift-register" nature of spintronic domain-wall memory (DWM) to create extremely low-energy FIFO queues. In order to test these queues in the most relevant application context, replacing conventional memory buffers for NoCs, we perform design-space analysis over the different schemes in a network context and then analyze the best schemes with benchmark traffic. Our results indicate that the best shift-based buffer utilizes a dual-nanowire approach to ensure that reads and writes can be more effectively aligned with access ports for simultaneous access in the same cycle. Our approach provides a 2.93× speedup over a DWM buffer using a traditional FIFO memory control scheme with a 23.4% savings in energy. The resulting approach achieves a 39% reduction in energy-delay product compared to SRAM and a 24% reduction in energy-delay product versus spin-transfer torque magnetic memories.**

***Index Terms*— Computer networks-mesh networks, memory-buffer storage, memory-nonvolatile memory.**

I. Introduction

EMERGING memory technologies, such as spin-transfer torque magnetic memories (STT-MRAM), have been proposed for the replacement of conventional memory elements of the memory hierarchy for their nonvolatility, density, and static power advantages over conventional memories. Thus, as the number of cores scales, the local caches, and increased on-chip shared memory storage can consider STT-MRAM replacements to reduce static power and improve the power-density tradeoff from further scaling of the system cores. A limitation of this strategy is that unlike conventional SRAM, STT-MRAM is inherently asymmetric in its access: its relatively fast (SRAM speed) reads is offset by relatively slow, energy intensive (2–4 times worse) writes. Network-on-chip (NoC) first-in–first-out data buffers (FIFOs) perform most efficiently when read and write performance is symmetric, as packets are often read and written in the same cycle. Furthermore, the storage array for such buffers is often dominated by the size of the peripheral circuitry making the overall power benefit of STT-MRAM storage potentially limited by the small FIFO size.

Spintronic domain-wall "Racetrack" memory, a CMOS compatible memory [1] recently proposed and demonstrated by IBM [2], [3], provides a potential solution for the size issues apparent in STT-MRAM FIFOs. Domain-wall memory (DWM) is comprised of a ferromagnetic nanowire where the multiple bits of data are stored in the different domains along the nanowire. To read/write the data in the DWM, the appropriate bit must be shifted into alignment with an access point similar to the magnetic tunnel-junction (MTJ) of STT-MRAM. Additionally, DWM can replace the current-based write of STT-MRAM with a shift-based write [4] to improve the write time and energy. The nonuniform random access and shifting characteristics of DWM make the design of network buffers a unique challenge that is considerably different to memory buffers constructed from random-access storage arrays.

In this paper, we first focus on the use of DWM as a queue. We examine the multitude of different configurations and eliminate those that is not viable, as well as provide a framework for choosing the configurations that are given particular hardware limitations, such as shift speed, read speed, and power. Moreover, we provide proofs for various boundary conditions such as minimum and maximum access cycles based on the configuration parameters.

After narrowing the possible configurations through analysis, virtual queue traffic simulation, and trace-driven cycle-accurate NoC simulation, we provide the hardware implementations of leading queue configurations and test those with full-system simulation on synthetic and benchmark workloads.

In particular, in this paper, we make the following contributions.

1) We articulate the unique properties and mathematical limitations on DWM queue performance originating from DWM's unique behavior.
2) We create a "shift-register" style FIFO, including both the single- and dual-nanowire approaches, that leverages the properties of the DWM shifting operations for efficient NoC buffer implementation.
3) We conduct a detailed sensitivity study that considers different shift speeds, read speeds, read head distance, cycle times, and read head offsets with synthetic traffic.
4) We provide full-system evaluations of the highest performing DWM designs with benchmark traffic workloads in a mesh for a 64-core chip multiprocessor.

Our best performing DWM approach provides a 2.93× speedup over a DWM circular buffer (CB) implementation

Manuscript received September 21, 2017; revised January 31, 2018; accepted March 13, 2018. Date of publication April 5, 2018; date of current version July 24, 2018. This work was supported in part by the NSF Graduate Research Fellowship under Award 1747452, in part by SHREC Industry and Agency Members, and in part by the IUCRC Program of the National Science Foundation under Grant CNS-1738783. *(Corresponding author: Donald Kline.)*

The authors are with the University of Pittsburgh, Pittsburgh, PA 15213 USA (e-mail: dek61@pitt.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2819945

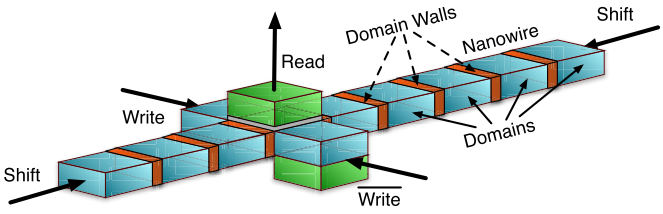


Fig. 1. DWM design.

with a 23.4% energy reduction. Compared to an SRAM FIFO, it provides a 43% energy reduction with 8% latency degradation. In our full-system performance experiments, this results in a 39% reduction in energy-delay product compared to SRAM and a 24% reduction in energy-delay product as compared to the leading STT-MRAM FIFO scheme.

II. Background and Related Work

DWM comprises an array of magnetic nanowires, where each nanowire consists of many magnetic domains separated by domain walls (DWs). Each domain has its own magnetization direction used in a similar manner to STT-MRAM. For a horizontally arranged planar strip (Fig. 1), several domains share one access point for read and write operations [5]. The DW motion is controlled by applying a short current pulse on the head or tail of the nanowire in order to align different domains with the access point. Since, the storage elements and access devices in a DWM do not have a one-to-one correspondence, a random access requires two steps to complete: step 1) shift the target bit and align it to an access transistor; and step 2) apply an appropriate voltage/current to read or write the target bit. Intrinsically, the read operation from step 2 is the same as STT-MRAM, however, the write can be a shift in the orthogonal dimension [4]. Thus, the tradeoff for DWM is reduced leakage power over STT-MRAM (fewer access transistors per bit) with increased dynamic power due to shifting domains.

Various forms of storage applications based on DWM have been demonstrated, such as array integration [1], lower level cache [6]–[8], content addressable memory (CAM) design and fabrication [9], [10], reconfigurable computing memory [11], [12], and a GPU register file [13]. These applications focus on using DWM in random access applications. In contrast, we explore the use of DWM in queue-oriented applications, which has considerably a different access behavior.

Due to the growing percentage of power consumption in many-core architectures contributed by the NoC, it has been a significant concern of many research groups to find ways to reduce power consumption and use high-density memories. A fixed-length shift register, realized by perpendicular magnetic anisotropy (PMA) technology, has been demonstrated [5], [14]. While it has been proposed to use DWM in FIFOs in an NoC [15], this proposal used the naive circular control scheme we will discuss, and did not optimize Racetrack control for FIFOs. We explore new control approaches that leverage the properties of queues rather than random-access structures.

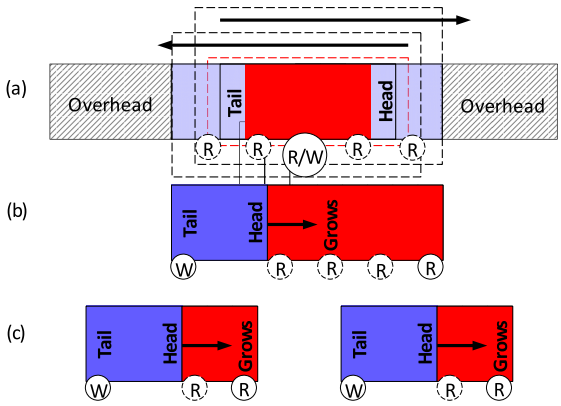


Fig. 2. FIFO queue structure with DWM. (a) Traditional CB. (b) Shift-register approach (LB). (c) Dual shift-register approach (Dual).

In addition to the improvements which focus on emerging memories, a substantial amount of research has been performed to reduce the number and optimize the usage of network buffers. For example, a network which actively adjusts the number of available buffers through flow control to save energy has been designed [16]. Furthermore, a completely bufferless high-performing NoC design has been realized [17] that performs well for light traffic, but has challenges maintaining high performance at medium and high network loads.

The leading scheme to reduce energy in NoC buffers while maintaining the original buffer capacity is to replace a large percentage of the FIFO's SRAM with STT-MRAM [18]. This approach writes into SRAM and then lazily migrates it to a reduced retention (i.e., a faster lower write effort) STT-MRAM [19], [20] when possible. An energy savings of 16% is demonstrated. In contrast, we demonstrate several DWM designs for NoC FIFOs that replace the entire SRAM buffer with spintronic memory with little (e.g., a single-flit) or no SRAM buffering required. We describe our DWM-based variable length queue designs in Section III.

III. Designing Queues With DWM

At first glance, DWM appears naturally suited to implement queue structures. In fact, DWM can naturally implement stacks and fixed-length queues. However, implementing efficient variable length FIFO queues, as required in network applications, is nontrivial. In this section, we describe our methodology for building variable length queues from DWM, and develop quantitative descriptions of these proposed queue approaches based on the physical parameters of the DWM.

A. DWM Physical Design

Traditional FIFO architectures utilize head and tail counters to implement a CB, where the head and tail pointers (tracking the next write and read locations, respectively) can wrap around the array. While this configuration naturally lends itself to array-based memory technologies (e.g., SRAM), it does not naturally extend to DWM. DWM has a nonuniform access time due to the shifting required for alignment with an access port. In Fig. 2, we present three DWM-based queues where

the queue is implemented as a group of N simultaneously shifted Racetracks and N is the number of bits in a flit. Thus, each "domain" represents storage for a flit. Moving forward, we describe operations for a buffer assuming each operation will be completed simultaneously for all N Racetracks in parallel.

Fig. 2(a) shows the CB implementation using DWM. We start with a single read/write port in the center. An FIFO write shifts the Racetrack (if necessary) to align the tail domain with the access point (step 1) and then to write (step 2) using the orthogonal shift-write. For reading, the head domain is aligned with an access point and then read by applying a current. Immediately, several undesirable characteristics become apparent. First, to align the leftmost or rightmost domain with the access point requires a nanowire that is essentially twice as long as the useful storage in the device [regions indicated by "overhead" in Fig. 2(a)]. This makes the nanowire larger and requires more shifting. Also, it may be necessary to shift the full logical length of the Racetrack between subsequent writes. Furthermore, most FIFOs are assumed to be able to read and write simultaneously, which is not possible in most configurations.

To address these inefficiencies, we consider three approaches which can be applied independently or simultaneously, a linear buffer (LB) concept [Fig. 2(b)] that shifts data through the Racetrack such as a shift-chain, an increase in the number of access points, and the introduction of temporary SRAM storage to buffer reads or writes to move them off the critical path.

1) Linear Buffer: We propose a new hardware architecture, referred to as LB, to attempt to mitigate the inefficiencies with the CB approach. LB configurations write at one end of the queue, and then shift the data into the queue in analogous fashion to a shift register, keeping the data contiguous. LB saves space over CB because it does not require any additional overhead; it can read and write until it reaches maximum capacity without shifting valid data out of the Racetrack.

2) Increasing the Number of Access Points: It has been demonstrated that a multiple read port Racetrack does not detract significantly from the density achievable by the nanowire because of the small relative size of read ports [6], [21]. Thus, it is reasonable to introduce additional read access points to all physical schemes to increase performance at the cost of some additional static power. One possible configuration for each of CB and LB using additional read ports is displayed in Fig. 2 with dashed lines. Note that having a read port for every domain would be physically impractical, since it would result in the DWM losing its physical size advantage over STT-MRAM. In this configuration, back to back cycles with both read and write would incur delays if the cycle is sized to the read latency. This is further discussed and demonstrated mathematically in Section III-F.

3) Introduction of SRAM Storage: All physical DWM queues can be augmented with an additional SRAM (or STT-MRAM) buffer to attempt to improve their performance. However, while the buffer improves performance in most cases, it also comes with a significant negative tradeoff in power and the loss full nonvolatility of the buffer. This approach will be considered in greater detail in the full-system simulation in Section VIII.

While LB significantly reduces write delays compared to CB, consecutive reads in either scheme continue to introduce additional read latency even when including a single-flit SRAM head buffer and prioritizing reads due to the longer operation time compared to shifting/writing. In order to mitigate this concern, we propose using Dual LBs (Dual). Each of the nanowires is half the length, but also has half the read access points of the LB [Fig. 2(c)]. The two-Racetrack structure alternates consecutive reads (and consecutive writes) to each Racetrack, essentially creating the illusion of a dual port. The consecutive read delay can be eliminated with sufficient read heads because one Racetrack can shift to prepare for an access while the other is accessed.

B. Terminology and Assumptions

When discussing the physical topology of the DWM queue, it is helpful to define some terminology and parameters used in organizing the queue.

The read offset F is the space between the write head and the first read head. In order to simplify the control and design logic, the distance between all read heads is enforced to be uniform, and is denoted as by a read separation parameter N. In addition, the maximum gap G, is defined as the larger of either the read separation N or the read offset F, $G = \max(N, F)$. The length of the Racetrack denoted as L and is equivalent to the logical queue capacity. Finally, as demonstrated in several other macro-cell DWM systems [4], [6], we assume that each Racetrack only has one write port, in part to maintain an area advantage of DWM over SRAM and STT-MRAM.

To consider the performance of different designs, a second set of terms is required including the shift speed S, the reading time R, the writing speed W, and the chosen cycle time C of operation. Since, the write operation can be completed as a shift in an orthogonal dimension [4], from this point on we will assume that the write speed is equivalent to the shift speed. Currently, reading for DWM is the latency bottleneck for the technology [22], so in most of our control designs the C will be chosen to permit R along with the delay of the peripheral circuitry.

Finally, several other assumptions were made to keep the control logic manageable. First, we assume the data in the queue must remain in order. Second, the data must be maintained without gaps or empty domains between the valid data. For CB, this implies that if the head points to position H the data will be written in position $(H + 1) \bmod L$. For LB, this means that the data are always written into position 0, and only when the most recently written data are adjacent at position 1. For Dual, writes alternate between each half queue, but otherwise follow the same restrictions as LB. Also, we assume that reads and writes are permitted at any cycle where the queue is not empty or full, respectively.

C. Shifting Control and Policies

In traditional SRAM queues with uniform random access, read and write ports can be implemented independently.

However, queues composed of DWM may have to delay accesses because ports are busy due to misalignment with the requested data or storage location. As a result, the queues must also have the read-pending and write-pending signals, which is asserted on a read or write that cannot be completed because the port is busy. This signal forces the queue to focus on the pending access. For example, in the case of a pending write, the queue would shift to align with the write head (having its most recently written data in position 1) and wait until the write occurs before allowing another operation. In this case, the queue would only service reads if it did not interfere with being available to write. A similar case can be envisioned for a pending read.

For each configuration (LB, CB, and Dual), after performing a write, a read, or a no-op, it is possible that there may be enough time left in a cycle to also perform additional shifting to put the queue in better position to service the future accesses. The decision on what proactive shifting to complete is split into two main methods of control.

1) Stay-in-Place: With stay-in-place, the queue will remain in its current position, and not attempt to proactively shift to any position to anticipate a read or write. Instead, the stay-in-place scheme only shifts when either the write-pending or read-pending signal is asserted, or the queue is in a position where it can service a read or a write in the current cycle, and as part of that service it must shift. This strategy is common in many DWM designs for random access applications.

2) Shift-to-Home: Shift-to-home policies in general attempt to use spare cycles and shifting opportunities to align with a predetermined access point, known as its "home," whenever the queue does not have a read-pending or write-pending signal active. For a CB, there are two possible homes: the location at which the tail is aligned with the write head (shift-to-write), and the position where the head is aligned with the closest read head (shift-to-read). For the LB (and the underlying queues that form Dual), there are three possible homes: (shift-to-read-forward) aligning the head pointer with the closest read head to the right, (shift-to-read-back) aligning with the closest read head to its left (assuming sufficient space/padding to prevent data loss), or (shift-to-write) aligning the tail pointer to the write head.

D. Impacts of Shift Speed

This section establishes the correlation between the shifting speed of a Racetrack and its observed latency. Since the latency is data-dependent, we focus on the calculation of the maximum number of useful shifts in a cycle: the shift speed above which latency will not improve for any possible pattern of read and write accesses. For instances of CB, LB, and Dual controls where the cycle time is based on read latency, the following methods can be used to calculate this quantity. For stay-in-place or shift-to-write schemes, the maximum useful shifts-per-cycle can be calculated simply using a queue with one element in it, and adding the worst case number of shifts to align with the write head and write to the distance to shift from that location to the home location. For shift-to-read schemes, determining similar expressions becomes a maximization problem dependent on the number of elements in the queue and the location of the read heads.

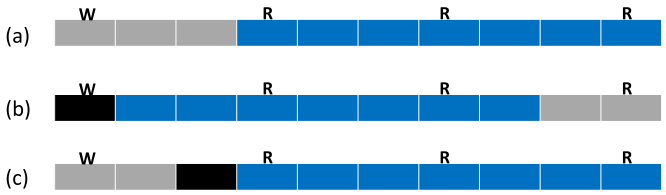


Fig. 3. LB shift-to-read-back maximum useful shifts/cycle writing example, 0 postread shifts, and 0 padding.

An example of the maximum useful shifts in a cycle for an LB with a shift-to-read-back is shown in Fig. 3 where blue indicates valid data and gray indicates unused space. The worst case start for an LB queue is that the tail is G positions away from being aligned with the write head, where $G = 2$ in Fig. 3(a). The reason for this worst case distance from aligning with the write head is that shift-to-read-back will always shift left to align with the read head if it is possible to do so while not blocking the write port. When a write arrives, the Racetrack now must shift G positions to align its tail with the write access point, shown in black in Fig. 3(b) and takes one additional shift to complete the write (as a part of the shift-based write). Following this, the queue attempts to realign with the closest access point. Since the Racetrack cannot shift left because there is no padding to the left of the tail, it shifts right two positions to align with the next read head requiring an additional G shifts. Thus, the total shifts for this operation is $2G + 1$ (or five, as shown in Fig. 3). The maximum useful shifts-per-cycle expressions can similarly be determined for other architectures and policies, and the results are reported in Table I.

E. Impact of Read to Shift Time Ratio and Postread Shifts

When a Racetrack can perform x number of shifts in addition to performing a read within a cycle, then we say that the queue has x postread shifts. One solution to reduce the latency for any Racetrack queue is to increase postread shifts in a cycle; however, at a certain point, there will be a cycle time $C_{\max}$ (defined as the maximum useful cycle time) which will have no latency degradation for any combination of inputs. Equivalently, this means that $C_{\max}$ is the minimum cycle time at which the queue is guaranteed to be able to both read and write every cycle. In this section, we calculate the maximum useful cycle times for each combination of shift policy and Racetrack hardware.

There are three primary queue positions which contribute to the calculation of the maximum useful cycle time. The first originates from the time it takes for LB or Dual with two elements at the far end of the queue to both read and write in that cycle (represented by α and α_h, respectively, in Table II). One example of this situation is shown in Fig. 4(a), which contributes the time for $L - 2$ shifts ($L - 3$ actual shifts and one shift delay to write) and one read delay R. For $L = 10$, this results in $\alpha = 8S + R$. The shift-to-write and shift-to-read-back schemes do not include this term, because they are

TABLE I

MAXIMUM USEFUL SHIFTS-PER-CYCLE. ASSUMES QUEUE CANNOT SHIFT AND READ IN THE SAME CYCLE, NO PADDING FOR LINEAR/DUAL QUEUES, AND THAT WRITES CAN BE COMPLETED AS SHIFTS [4]

Circular	**Stay-in-Place**	**Shift-to-Write**	**Shift-to-Read**	
Shifts to Align with Write	L-1	L-1	L-1*	
Home Shifts	0	1	G+1*	
Total (including writes)	L	L+1	L+G+1*	
Linear	**Stay-in-Place**	**Shift-to-Write**	**Shift-to-Read-Forward**	**Shift-to-Read-Back**
Shifts to Align with Write	L-2	G	L-2*	G
Home Shifts	0	1	G*	G
Total (including writes)	L-1	G+2	L-1+G*	2G+1
Dual	**Stay-in-Place**	**Shift-to-Write**	**Shift-to-Read-Forward**	**Shift-to-Read-Back**
Shifts to Align with Write	$\frac{L}{2}-2$	G	$\frac{L}{2}-2^*$	G
Home Shifts	0	1	G*	G
Total (including writes)	$\frac{L}{2}-1$	G+2	$\frac{L}{2}-1+G^*$	2G+1

* Indicates worst case but not the general case. Certain configurations of read heads may result in this number being reduced, but the order (e.g., linear with $L+G$) remains the same.

TABLE II

MAXIMUM USEFUL CYCLE TIMES (NOT INCLUDING PERIPHERAL CIRCUITRY DELAY)

Circular	**Stay-in-Place**	**Write**	**Read**	
	$max(L*S+R,\gamma)$			
Linear	**Stay-in-Place**	**Write**	**Read Fwd**	**Read Back**
	$max(\alpha,\beta,\gamma)$	$max(\beta,\gamma)$	$max(\alpha,\beta,\gamma)$	$max(\beta,\gamma)$
Dual	**Stay-in-Place**	**Write**	**Read Fwd**	**Read Back**
	$max(\alpha_h,\beta,\gamma)$	$max(\beta,\gamma)$	$max(\alpha_h,\beta,\gamma)$	$max(\beta,\gamma)$
$\alpha:(L-2)*S+R$		$\beta:(1+G)*S+R$		
$\alpha_h:(\frac{L}{2}-2)*S+R$		$\gamma:(\frac{3*G}{2}-(G-1)\%2)*S+R$		

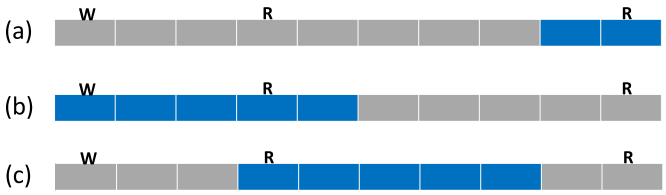


Fig. 4. Maximum useful cycle examples for LB.

guaranteed to have a maximum difference between the tail data and the write head of at most G. CB has a slightly a different term, $LS+R$, because it is dependent on the situation where the write head must shift the logical length of the Racetrack and conduct a read and a write sequentially within a cycle.

The second contributing factor occurs when the queue must shift the maximum gap, G, in order to read and write in a cycle (represented by β in Table II). An example of this is demonstrated in Fig. 4(b) where $G=5$ requiring G shifts, one shift delay to write, and one read delay R, totaling $\beta=6S+R$. The final scenario occurs when the data are located in the middle of the gap and cannot be read on the way to write (represented as γ in Table II). This scenario is demonstrated for LB in Fig. 4(c). In these examples where $G=5$, the queue shifts right two locations, reads, shifts left four locations and writes, contributing $\gamma=7S+R$.

The home policy dictates which of these conditions contribute to the maximum cycle time. It turns out that for LB, shift-to-read-back and shift-to-write are only dependent on conditions two and three (β, γ), while the others also depend on α. The maximum useful cycle time for a control scheme is the maximum of the conditions which apply to that scheme. Therefore, in the example in Fig. 4, LB stay and LB shift-to-read-forward would have a max cycle time of $8S+R$, while LB shift-to-write and shift-to-read-back would have a max useful cycle time of $7S+R$. The maximum cycle time follows this logic for the other schemes as well, and the summary of the times can be seen in Table II.

F. Impacts of Sizing Cycle to Read Latency

Since the read latency is the limiting performance characteristic (highest latency operation) in DWM, as mentioned previously, it is natural to choose the cycle time as close to the read time as possible. However, if there is not enough time to both read and shift in a cycle, then there is no possible configuration of writing speeds, shifting speeds, and distance between read heads for a single Racetrack, which can always service both a read and a write request in a cycle. This limitation arises from two origins: only having one write head per Racetrack, and not being able to shift the Racetrack in the same cycle as it is read. This can be easily proved by contradiction by considering a situation where the Racetrack begins with at least one element and then must service both a read and a write in two consecutive cycles. In the best case, the Racetrack begins with the first value aligned with the read head, and successfully reads and writes in the first cycle. Because there is not enough time to read and shift, the written data element remains under the write head at the start of the next cycle. In the next cycle, when a read and write request is made, because there is not enough time to read and then shift in the cycle, as the FIFO started unaligned with the next write position, the system does not have time to both be read and shift to satisfy the new write.

IV. DWM STATE-MACHINE DESIGN

The algorithms presented in Section III are not designed for optimized hardware implementation. Therefore, in this section, we present a hardware optimized methods to implement control focusing on the specific example of two shifts-per-cycle, a read offset of zero, a read separation of one, and no postread

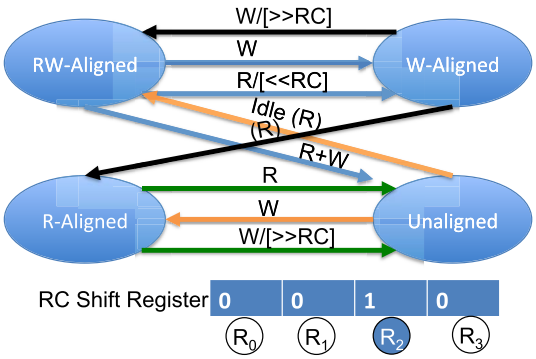


Fig. 5. LB FSM. R = read, (R) = read align RT, W = write, R + W = read and write, and Idle = neither read nor write. [$\ll RC$] and [$\gg RC$] also shift the read port left and right, respectively, depicted in the *RC* shift register where the highlighted ("1") port is currently selected.

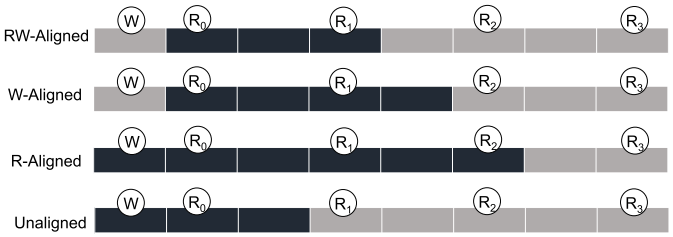


Fig. 6. Example queue conditions corresponding to LB FSM controller. Black regions are valid flits.

shifts in a cycle. All other parameter combinations can be similarly expanded into the hardware control implementations.

The CB scheme uses traditional head/tail pointers to determine shift locations, the LB scheme requires a finite state machine (FSM) for control, as shown in Fig. 5. There are four states possible for the buffer (see Fig. 6): RW-Aligned, where the queue head is aligned with a read access point and the tail pointer is aligned with the write access point, W-Aligned where the tail pointer is aligned with the write access port but the head is not aligned with a read access point, R-Aligned where the queue head is aligned with a read access point but the tail pointer is not, and Unaligned where neither the head nor tail is aligned with an access point. This FSM can easily be expanded for a larger gap (more domains) between read access points through additional "unaligned" states.

There are four possible permutations of operations for each state: buffer read, write, both read and write, or idle. In the RW-Aligned state all requests can be handled directly. On a read, the head is read and the state moves to W-Aligned. On a write, the FSM writes and shifts right the Racetrack and the state also moves to W-Aligned. For both a read and write, the FSM simultaneously writes and reads and moves to the unaligned state. In unaligned, if idle (or an R occurs) the Racetrack buffer shifts right and moves to the RW-Aligned state in the next cycle (read still pending). If a write (or read and write) occurs, the queue shifts right and writes in a cycle moving to R-Aligned (read still pending). The other states proceed in a similar way with certain options able to be serviced directly, and those in parenthesis requiring multiple cycles to complete. We track the current active read head using

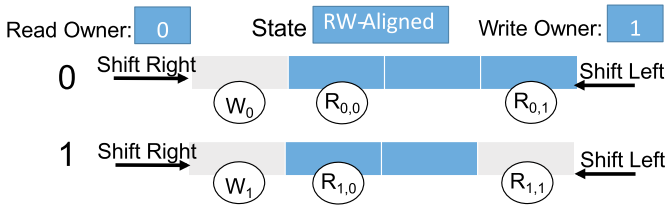


Fig. 7. Dual Racetrack design. The two Racetracks together comprise one NoC buffer.

a simple shift register of the same length as the number of flits that shifts in the same direction as data of the Racetrack queue. This simple shift register could also be implemented using a DWM, although during our analysis of the peripheral circuitry we assume it is implemented in traditional CMOS. These operations (noted by $\ll$ and $\gg$) as well as other state transitions not explicitly described are enumerated in Fig. 5.

Dual and LB have very similar FSM structure. For Dual, we include two additional control bits, "Read Owner" and "Write Owner" to manage which Racetrack is accessed in each cycle. For each access, the owner bit is flipped. In the example from Fig. 7, the queue holds five flits, it is in the RW-Aligned state, the next read comes from queue "1" and the next write goes to queue "0," both of which may proceed in parallel. The read head is indexed by the "Read Owner" and a single bit *RC* (shown in Fig. 7 as the R indices). Overall Dual has a similar control overhead to LB.

V. Racetrack Overheads

The overhead of the CB scheme includes read and write pointers [$lg(N)$ bits each], a stored current offset value for Racetrack [$lg(N)$ bits], comparator, increment and decrement circuitry, $N-1$ extra domains to prevent loss of data when shifting to the ends of the Racetrack [see Fig. 2(a)], and the stored locations of the read and read/write access points. Note, it does not require the one-hot head and tail pointer storage because it does not use these bits to energize a word-line as in traditional array-based storage. If the CB scheme is augmented with an SRAM buffer, the additional overhead includes a one-flit buffer plus an additional one valid bit per Racetrack.

For LB, only N domains are required as compared to the $2N-1$ domains per Racetrack for CB. The overhead for this scheme includes two bits of storage to represent four states, and the same number of bits as the number of read heads (1 hot, in a shift register) for the currently used read head plus the same overhead for an additional SRAM buffer as CB. Similarly, in the Dual scheme, N domains are required per buffer ($(N/2)$ for each Racetrack). The overhead for the Dual scheme includes two bits per Racetrack to represent the four states, one bit per buffer to represent the valid read head and two additional bits to indicate which Racetrack is controlling the buffer reads and writes (i.e., one bit to each represent the read and write owners of the buffer, respectively, in Fig. 7). The Dual scheme does add additional peripheral circuitry to write to and shift two half-length Racetracks, which are accounted for the energy calculations presented.

VI. Experimental Methodology

In order to analyze the performance of DWM queues, we began by evaluating the queues independently based on different read and write traffic patterns. We simulated each queue (e.g., CB, LB, and Dual) for different parameters with different synthetically generated read and write traffic patterns. For uniform random traffic, at each time step the chance of reading was K, and the chance of writing was also K, as long as the queue was not empty/full, respectively. Access latency is determined by the average delay from when the access request arrives and when the access is completed.

Based on the best performing parameters in the queue simulator, we further tested DWM buffers by implementing CB, LB, and Dual schemes to serve as the virtual channel buffers in an NoC using the cycle-accurate HORNET multicore simulator [27] to compute both average flit latency and energy consumption. In addition, peripheral circuitry power calculations for SRAM, STT-MRAM, and different Racetrack FIFO schemes were analyzed using data from [4], [23] and a modified version of NVSim [24]. Sniper [28] was used to generate workload traces of the Princeton Application Repository for Shared-Memory Computers (PARSEC) benchmark suite [29] for the modified HORNET simulator. To estimate the full-system performance impact of the NoC, the HORNET generated latencies were then used in a second full-system simulation in Sniper to determine the performance impact via IPC.

The tests were performed on a simulated 64-core network using one-queue-per-flow o1-turn routing. In addition, each ingress port connecting a core to its neighbor has eight virtual channels, each of which can store eight flits. CB, LB, and Dual were tested both with and without a single-flit SRAM storage to buffer the queue's head flit. The full-system simulated out-of-order cores with multiple instructions issue per-cycle with private first level and a distributed shared last level (L2) cache. When running the simulations, we assumed that there was no performance difference between SRAM and STT-MRAM as an optimistic implementation of the leading STT-MRAM NoC buffer proposal [18]. Also, to compare the implementation of the DWM technology with a more energy efficient, and for STT-MRAM area-equivalent counter parts, both SRAM and STT-MRAM were also tested with half (4) the number of queues per channel (referred to as SRAMHalf and STTHalf).

To determine the control circuitry overhead from implementing LB and Dual, we created an implementation of the FSM in 45-nm standard cell ASIC hardware and compared it with the traditional one-hot read and write points for traditional FIFOs. The RT control had a nominal overhead increase in terms of area and energy (i.e., < 0.5 μW) compared to the traditional FIFO. The full energy parameters for the virtual queues of eight flits and 128 bits per flit for Racetrack, SRAM, and STT-MRAM are reported in Table III. The increase in static power for LB and Dual, respectively, over CB results from small increases in control circuitry and peripheral circuitry. Adding the single-flit SRAM storage to the Racetrack schemes increases both the static and dynamic powers for reads and writes. The Racetrack LB approach (7.9 μW) requires about half the static power of an STT-MRAM array (14.9 μW) and one-third of an SRAM array (22.6 μW). Dual still provides 36% and 57% static power reduction over STT-MRAM and SRAM, respectively. The area overhead for the shift-to-read variation of the control schemes examined in Section VIII is shown in Table IV. The state-machine methods of control (LB, Dual) are more efficient than the adaptation of the head and tail pointer method (CB) by nearly a factor of four and two, respectively.

We assume an NoC clock speed of 1 GHz which allows SRAM, STT-MRAM, and Racetrack reads in a single cycle based on latency data from NVSIM and the literature [22]. STT-MRAM writes are also optimistically assumed to be a single cycle [18] similar to SRAM, while Racetrack shifts and writes take half a cycle allowing a Racetrack to conservatively write and shift, or shift twice in a cycle [22]. The detailed architecture parameters are shown in Table V.

TABLE III

Buffer Power for Different Technologies Using Synthesized Control Logic, Data From [4], [23] and Calculated Using a Modified Version of NVSim [24]

	Static Power (μW)			Dynamic Energy (pJ/bit)		
	mem. array & control			read	write	shift
SRAM	22.60			0.27	0.12	—
STT	14.91			0.10	0.24	—
	CB	LB	Dual			
RT	7.76	7.91	9.61	0.10	0.062	0.062
RT+SRAM	10.23	10.39	11.40	0.37[2]	0.36[1]	0.062

[1]This is a special case for Dual where each Racetrack has two read heads specified by that valid read head bit, otherwise each Racetrack would require an *RC* Shift-Register circuit as in the LB control.

[2]Empty buffer accesses incur SRAM-only cost.

TABLE IV

Area for Different Racetrack Shift-to-Read Schemes Using Synopsys Design Compiler [25] and a 45-nm process design kit [26]

Area (um^2)	CB	LB	DUAL
	915.135	250.136	493.704

TABLE V

Architecture Parameters

CPU		Cache	
64 out-of-order cores 4 issue width, 4GHz clk		Private L1 32K Inst, 32K Data Dist. Shared L2 4M (256K/tile)	
NoC			
1GHz clk 8 VCs	1-cycle read 3-cycle pipe	1-cycle write 8 flits/queue	0.5-cycle RT write 0.5-cycle RT shift

VII. Design-Space Analysis

In Section III, we evaluated mathematically the limits for the useful range of parameters in DWM queue designs. In this section, we evaluate the practical sets of parameters from the design space using experiments with common synthetic workloads for NoCs defined by the relationships between the source and destination addresses of the traffic [30]. Results for the pattern "shuffle" are shown, as the bit-complement and transpose workloads had the same trends. The resulting

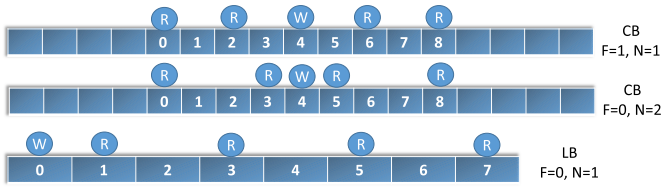


Fig. 8. Physical layout of the different configurations used in the design-space analysis of the different shift control schemes.

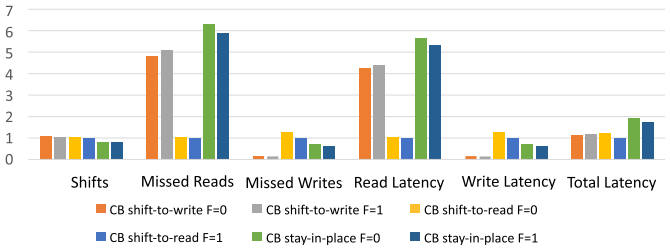


Fig. 9. Normalized results to shift-to-read F = 1 for the CB using four shifts-per-cycle, four read heads, and nine elements per queue for different shift policies with under evenly distributed random traffic at 10% intensity. Distance between write and first read port (F) is considered at 0 and 1.

configurations are then evaluated using cycle-level analysis with application workloads in Section VIII.

A. Control Schemes

In this section, we evaluate the effectiveness of the shifting policy, specifically stay-in-place, shift-to-read, and shift-to-write, and their buffer architecture design specific variants such as shift-to-read-forward and shift-to-read-back (see Section III-C) on overall performance of the CB and LB schemes.

1) Circular Buffer: To begin the analysis of the CB, we first compare $F = 0$, $N = 2$ (read ports adjacent to the central write port and a spacing of two between read heads) and $F = 1$, $N = 1$ (read heads with an offset of one from the central write port and a spacing of one between read heads). The physical layout of each of these cases are shown in Fig. 8; as shown, each configuration has four read heads.

Fig. 9 demonstrates normalized read and write latencies, as well as total shifts for CB with four shifts-per-cycle and length $L = 9$ under low (10%) traffic. The traffic intensity is the probability a write and/or read request, each calculated independently, are issued in each cycle. Shifts represent the total shifts the Racetrack made during the simulation, Missed Reads are the total number of times the Racetrack received a read request and could not service it in the same cycle, and Missed Writes represent the same for writes. The Read Latency is the average amount of cycles a data element has to wait before it can be read, and Write Latency is the same for writes. Since every element is both written and read in a simulation, the Total Latency is defined as the sum of the read and write latency.

In all cases, the number of shifts in the $F = 1$, $N = 1$ scheme is less than the $F = 0$, and $N = 2$ counterpart. In this comparison, the total latency advantage for the $F = 1$, $N = 1$ is 18% and 9% for stay-in-place and shift-to-read, respectively, with a nominal 2% increase for shift-to-write compared to $F = 0$, $N = 2$. When different schemes are compared for the same F and N choices, we can see that, as expected, shift-to-read has substantially lower (over 4×) read latency, while shift-to-write shows similar improvements for write latency. However, as the 10% traffic intensity is far from saturation, even significantly higher missed reads and read latency does not dramatically increase overall latency because those missed reads are quickly serviced, often in the next cycle. For higher traffic intensity, these read delays can dramatically increase latency.

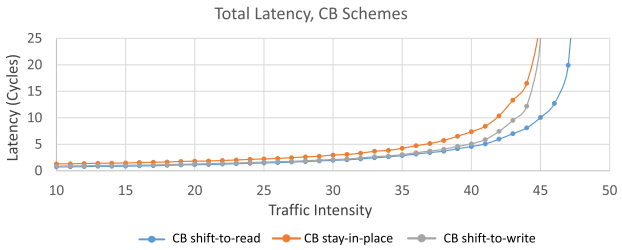


Fig. 10. Latency results as traffic increases beyond 10% (Fig. 9) for CB at four shifts-per-cycle, four read heads, and nine elements per queue.

Thus, in Fig. 10, we evaluate the impact of different traffic loads on queue latency for the different shifting policies in CB. As in Fig. 9, the traffic percentage is the probability of reading and/or writing in each cycle and the read and write probabilities are equivalent to mimic the behavior that all data written to the queue are eventually read. Each shifting scheme is evaluated with its best performing read/write head layout ($F = 0$, $N = 2$ for shift-to-write, and $F = 1$, $N = 1$ otherwise). As can be observed in the figure, CB with the shift-to-read control scheme has a later onset of saturation, as well as a consistently reduced total latency before saturation. For example, at the 10% Traffic point (shown in detail in Fig. 9), the best CB stay-in-place scheme has a 76% higher latency than the best CB shift-to-read scheme. While the stay-in-place control scheme shifts 21% less and has a simpler control design, the 76% latency degradation until the onset of saturation makes it less attractive compared to the shift-to-read scheme. The best shift-to-read control scheme also demonstrates better presaturation latency than the shift-to-write scheme. At the same 10% point, the shift-to-write scheme has a 15% higher latency, as well as 8% more shifts. Given this analysis, we conclude that the best performing control scheme for CB is shift-to-read with $F = 1$ and $N = 1$. From this point forward, when we refer to CB, we are referring to this configuration of CB.

2) Linear Buffer: To evaluate the shifting performance for the LB, the best read head placement was more obvious, $F = 0$, $N = 1$, for a length $L = 8$ queue with four read heads. Under these conditions, $F = 0$ provides the ability to write one data element and read it in the following cycle, while $F \geq 1$ prevents reading the cycle after writing. In this configuration, we show in Fig. 11 when LB is observed for low (10%) synthetic traffic (e.g., the presaturation region), the results follow a very similar trend to those for the CB.

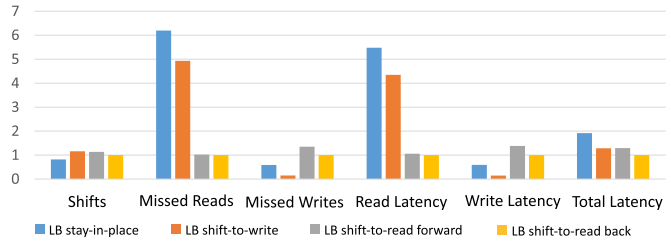


Fig. 11. Synthetic traffic results for LB using two shifts-per-cycle, four read heads, and eight elements per queue for different shift policies under evenly distributed random traffic at 10% intensity.

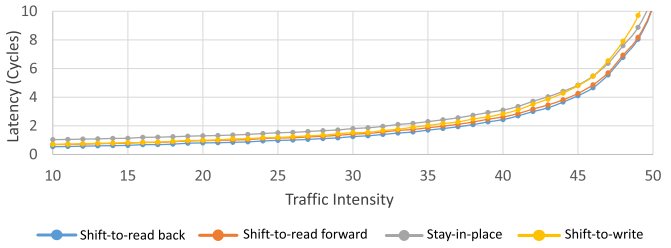


Fig. 12. Latency results as traffic increases for different LB schemes at two shifts-per-cycle, four read heads, and eight elements per queue.

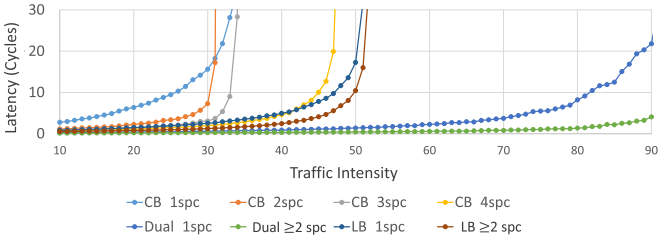


Fig. 13. Latency as traffic increases with varying shifts-per-cycle for CB ($L = 9$) and LB ($L = 8$), each with four read heads per queue.

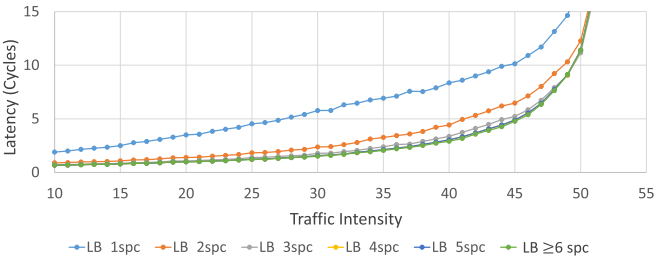


Fig. 14. Latency as traffic increases for LB for various shifts-per-cycle, with two read heads and eight elements per queue.

LB with the shift-to-read-back control scheme consistently has lower access latency than all the other schemes, shift-to-read-forward, shift-to-write, and stay-in-place for low traffic. Shift-to-read-back also have the fewest shifts except stay-in-place. While shift-to-write tends to favor writes and the two shift-to-read tends to favor reads, in a queue it is expected that each queue element will be written and then read, so we focus on overall latency.

As demonstrated in Fig. 12, all of the saturation regions are quite similar, but shift-to-read-back saturates the latest, thus giving a slight edge in the postsaturation region. Therefore with both CB and LB, we conclude that aligning with the read head (read-back for LB) is the most efficient design choice. Because of these factors, from this point forward, LB will refer to LB with the shift-to-read-back control scheme. Moreover, an analysis of the Dual scheme mirrored the trends of the LB, thus, we also selected and will refer to Dual as using a shift-to-read-back control scheme.

B. Shift Speed

In this section, we examine the effect of altering the number of shifts-per-cycle on the latency of LB, CB, and Dual queues. For CB queues, increasing the shifts-per-cycle can have tremendous impacts in the presaturation region. As can be seen in Fig. 13, as the number of shifts-per-cycle (shown as shifts-per-cycle Fig. 13) increases, the presaturation region significantly increases. Furthermore, the latency in the presaturation region (in terms of cycles per access) improves as well. For example, the latency is reduced by 5× at 10% traffic when increasing from one to two shifts-per-cycle. For CB, Table I predicts a maximum useful shifts-per-cycle of at least $L - 1$ (eight shifts-per-cycle in this $L = 9$ configuration), so it is reasonable that all transitions from one to four shown in Fig. 13 improve read and write latency.

LB queues also exhibit significant improvement in the presaturation region, with a 48% improvement in total latency at 10% traffic. Unlike CB, LB's improvement in latency plateaus after two shifts-per-cycle. While it was mathematically predicted that the maximum performance would be found at three shifts-per-cycle, the traffic pattern necessary to take advantage of that capability is rare.

In order to observe the changes with LB in greater detail, we also examined the change in latency with LB queues with $L = 8$, $N = 3$, and $F = 2$, shown in Fig. 14. Table I predicts the maximum useful shifts-per-cycle for this configuration is seven; however, the state transition where having an extra shift from six to seven shifts-per-cycle is similarly rare. Thus, six and seven shifts-per-cycle provide nearly identical results for this configuration of LB. While at the maximum shifts-per-cycle the latency reaches its minimum, all increases after four shifts-per-cycle are nominal (i.e., less than 0.1%). In our experimentation, similar results were found for different read head configurations leading to the conclusion that even though there is a mathematical limit of maximum shifts-per-cycle, LB configurations effectively reaches its practical minimum latency at $G + 1$ shifts-per-cycle, recalling that G is the maximum read head separation $[max(N, F)]$.

Dual queues, such as LB queues for four read heads, have their latency improvement plateau at two shifts-per-cycle, as can be seen in Fig. 13. Even with one shift-per-cycle, it takes high traffic (i.e., a read and/or write more than 80% of the cycles) for Dual to enter saturation. At 10% traffic, two shifts-per-cycle has a 52% latency improvement over one shift-per-cycle, and the percentage latency improvement provided by two shifts steadily increases with the traffic.

When comparing CB, LB, and Dual queues in Fig. 13, we observe that CB only outperforms LB with one shift-per-cycle in the presaturation region once CB has four or greater

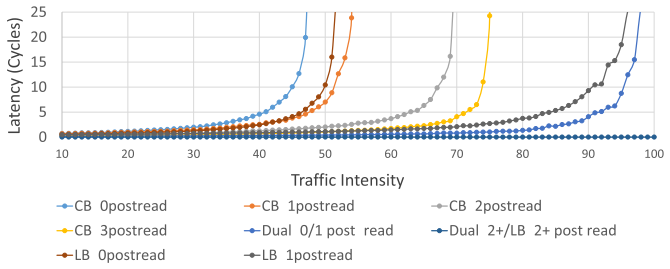


Fig. 15. Latency results in cycles as traffic increases with different postread shift amounts, with four shifts-per-cycle, four read heads, and eight elements per queue.

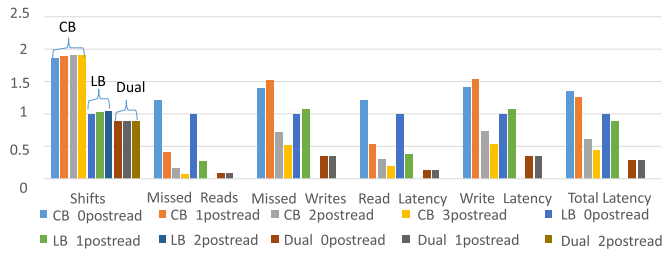


Fig. 16. Normalized synthetic traffic results with low traffic, four shifts-per-cycle, four read heads, and eight elements per queue for different amounts of postread shifts in a cycle.

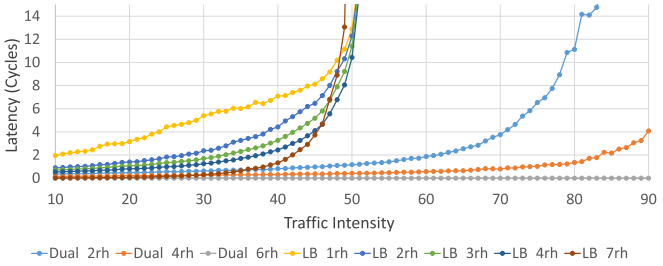


Fig. 17. Latency results as traffic increases over different read heads, with two shifts-per-cycle and eight elements per queue.

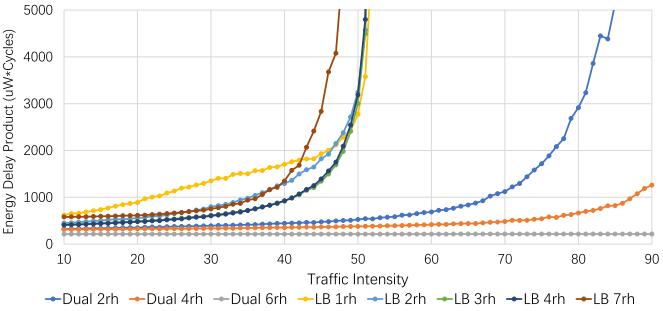


Fig. 18. Energy-delay product as traffic increases over different read heads, with two shifts-per-cycle and eight elements per queue.

shifts-per-cycle. However, even in this case, LB with one shift-per-cycle has a later onset of saturation. While LB queues consistently begin the onset of saturation around the 50% traffic mark (since they cannot read and write in consecutive cycles), Dual queues are able to remain in the presaturation region until around 80% traffic. This difference primarily arises from the parallel nature of Dual, where one queue can prepare for an operation while the other is being accessed.

C. Postread Shifts

Another consideration of cycle latency is the combination of a read access with shifting within a single cycle. Thus, we examine the effect of sizing the cycle to allow postread shifts on the access latency (in terms of cycles) of the LB, CB, and Dual queues. Fig. 15 demonstrates the effect of adding postread shifts (labeled as #postread in the figure) for a case with $N = 1$, $F = 0$, four shifts-per-cycle, and $L = 8$. Table II predicts the minimum cycle time that can give performance without delay is given by $(1 + G) \times S + R$ resulting in $2S + R$ or two postread shifts-per-cycle for LB and Dual in this configuration. With $N = 1$, increasing the cycle time from R to $R+S$ has a significant positive impact on LB performance as well as significantly delaying the onset of saturation from 50% to about 95% traffic, which is comparable to the onset of saturation for Dual with zero postread shifts. We quantified the latency improvement in Fig. 16 for 10% traffic.

An interesting result is that CB, which typically falls far short of LB, with one postread shift actually outperforms LB with 0 postread shifts. This lends credence to the importance of including shifting and reading in the same cycle. However, CB, even with three postread shifts in a cycle still is outperformed by Dual and LB with one postread shift-per-cycle. Of course, increasing the cycle time results in an improvement in queue performance from a queue state behavior perspective, but it comes at the obvious tradeoff of fewer cycles per unit time, which may make the original cycle time more efficient for Dual, despite the queue delays it incurs. However, if a particular queue realization changes the relationship of read access latency to shift latency, this provides interesting insights on how to size a cycle.

D. Read Heads

The number of read access points also provides an interesting tradeoff between static power, area, and queue latency. Thus, we performed a design-space exploration on the number of read heads in the queue, and examine the effect it has on total latency. Fig. 17 demonstrates the effect of varying the number of read access points on latency for LB and Dual schemes. CB schemes were also considered, but they again performed significantly worse than LB in the presaturation region, and consistently had an earlier onset of saturation and as a result are not displayed in detail in this section. For LB queues, as observed in this section with increasing shift speeds, all queues begin to saturate at or before 50% traffic, even with a read head in seven out of eight positions. While the presaturation latency steadily decreases as the number of read heads increases for LB, the 50% traffic limit is a fundamental limitation of not being able to both read and write in consecutive cycles.

As the number of read heads grows to reduce the latency, the energy, in large part from static energy, of the buffer increases. The resulting energy-delay product of these same configurations, shown in Fig. 18, shows that among the LB

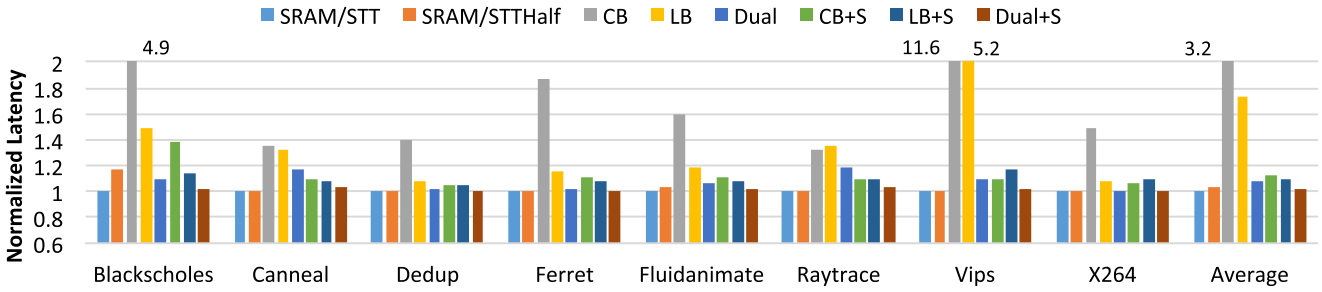


Fig. 19. Flit latency of Racetrack buffer schemes normalized to SRAM.

schemes that three or four read heads are nearly equivalent and have the minimum energy-delay product. Thus, LB with four read heads provides the best choice to minimize latency and energy-delay product. Note, adding seven read heads dramatically degrades energy-delay product over four read heads as the energy increase does not compensate for the minor latency improvement.

As previously mentioned, because Dual does not have the limitation of not being able to both read and write in consecutive cycles, it can break the 50% traffic saturation barrier with only two read access points (one per DWM). The energy-delay product also reflects the latency advantage of Dual over the LB. From Dual, the most significant finding is that Dual with six read heads (three per DWM) results in no additional latency and can guarantee a read + write every cycle. However, operating in this configuration would eliminate the size advantage over STT-MRAM due to the port every cell, and thus will not be seriously considered. Therefore, for practical purposes in our hardware implementation, Dual, CB, and LB with four read heads is used, since they have lowest total latency while still maintaining a significant area and power advantage over STT-MRAM. In Section VIII, we evaluate these configurations for application workloads in the context of a full-system simulation using Dual, CB, and LB DWM queues for NoC buffers.

E. Results of Design-Space Analysis

As discussed in Sections VII-A and VII-D, the shift-to-read and shift-to-read-back schemes consistently outperform the other shifting policies, and therefore will be the schemes used during the benchmark simulations in Section VIII. While having a read head every domain of the queues would eliminate the advantage over STT-MRAM, previous work suggests it would be feasible to place read heads every other element of the queue [6]. Doing so results in 24% and 56% improvement for LB and Dual, respectively, at 10% traffic over having a read head every third element. Because the standard FIFO queue size in an NoC is typically eight, we will use four read heads for CB, LB, and Dual during the benchmark simulations.

In Section VII-B, the latency of the queue saturates at two shifts-per-cycle for four read heads with a queue size of eight; therefore, it makes sense to choose a shifting speed that allows two shifts-per-cycle in Section VIII. Finally, in Section VII-C, while adding postread shifts provided significant latency improvements, Dual already has very low latency with four read heads and two shifts-per-cycle, and therefore increasing the cycle time would most likely result in an overall decrease of throughput. We simulate benchmark traffic with zero postread shifts in Section VIII.

VIII. Results and Discussion

To evaluate the impact of the proposed Racetrack buffer schemes, we compared the CB, LB, and Dual schemes with the parameters resulting from the design-space analysis discussed in Section VII with an SRAM and STT-MRAM baseline for flit latency, overall system performance (IPC) and energy-delay product based on the architectural parameters from Section VI using the Hornet and Sniper simulators. SRAM represents SRAM FIFO queues of equivalent number and capacity as the DWM queues, and SRAMHalf represents SRAM FIFO queues with half the number of queues as the DWM queues. STT and STTHalf are similarly defined for STT-MRAM FIFOs.

Fig. 19 summarizes the average flit latency of the Racetrack buffer schemes normalized to all SRAM[1] for the PARSEC benchmarks. As expected, the translation of a head/tail pointer FIFO concept (CB) performed poorly, resulting in a more than 3× increase in latency over SRAM. In contrast, LB was much more competitive than CB, but still increased latency by 73%. By adding a single-flit SRAM storage element for the head flit to the Racetrack, the CB (CB + S) and LB (LB + S) latency overhead can be reduced to 13% and 10%, respectively. The SRAM storage allows the Racetrack FIFO to support a limited number of concurrent reads and writes reducing the performance overhead of the Racetrack-only approach.

The Dual scheme *without* adding SRAM actually outperforms the CB + S and LB + S schemes but still requires an 8% latency overhead compared to the all SRAM FIFO. Dual, by alternating between two Racetracks, allows most cases of successive reads, writes, and concurrent reads and writes to be handled without introducing stalls. By adding similar SRAM storage to Dual (Dual + S), the latency drops to within 2% of the all SRAM case, which is equivalent to the overhead of reducing the number of SRAM VCs in half.

The impact of these latencies on full-system performance is shown in Fig. 20 as IPC normalized to all SRAM buffers. CB resulted in a dramatic 22% IPC degradation, while LB also

[1]We assume the most optimistic case for the STT-MRAM buffer scheme has identical performance as SRAM.

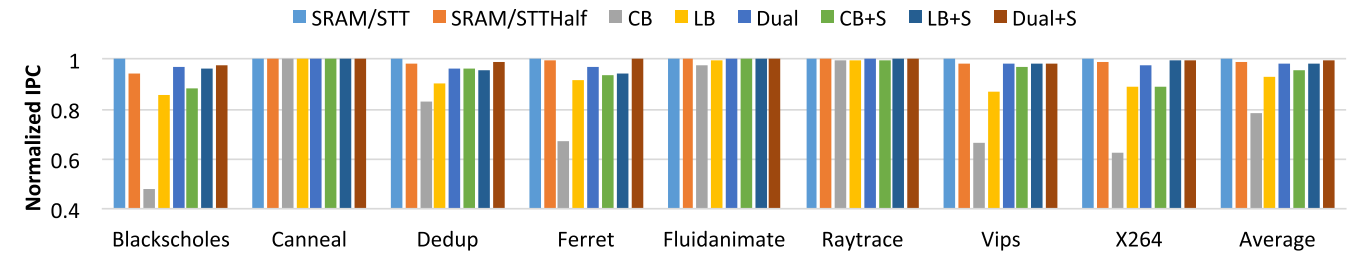


Fig. 20. Full-system performance (IPC) for different Racetrack buffer schemes normalized to SRAM.

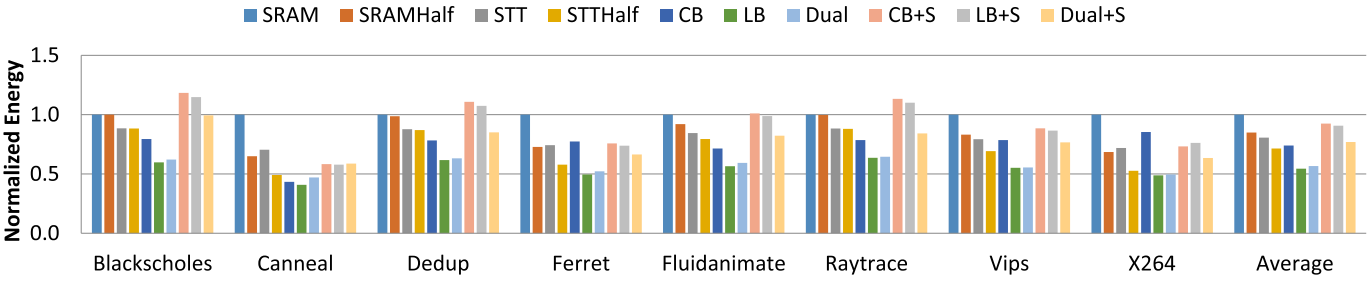


Fig. 21. NoC buffer energy results normalized to SRAM.

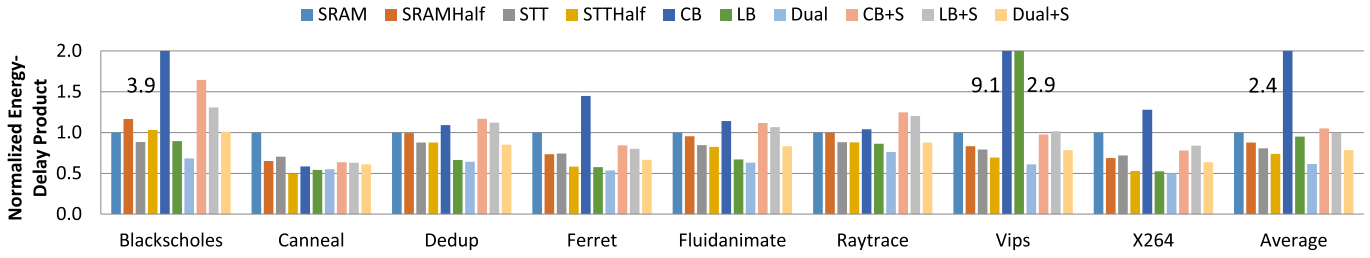


Fig. 22. NoC buffer energy-delay product normalized to SRAM.

had a significant 7.2% IPC reduction compared to SRAM. Adding the SRAM head flit storage, the LB + S and Dual scheme were nearly indistinguishable, requiring a nominal 1.7% overhead over SRAM and were within 0.5% of the SRAMHalf scheme. Dual + S was nearly indistinguishable from SRAM.

The energy consumption of the buffers computed from the information in Table III and the access behavior from each benchmark run, is shown in Fig. 21 and normalized to SRAM. In particular, this energy computation utilizes all the shifts, reads, and writes by the virtual queues over the length of the benchmark for each workload. As expected, SRAMHalf, STT, and STTHalf progressively reduce energy. Of the Racetrack schemes CB does considerably worse than the other Racetrack schemes despite its reduced static power, while LB and Dual perform the best. CB + S, LB + S, and Dual + S consume much higher energy due to the added static power from the SRAM, but also because most flits are both written to and read from the Racetrack and SRAM buffer.

The energy-delay product, reported in Fig. 22, shows that CB is a poor choice, causing a more than 2.4× increase over SRAM in energy-delay product. LB only provides a 5% reduction over SRAM, and performs worse than all of the STT-based schemes. However, Dual has the best result with a more than 35% savings over SRAM and more than 20% over STT-MRAM. When adding an additional SRAM buffer, CB + S and LB + S are both within 5% of the original SRAM. The Dual + S is better than STT-MRAM implementation by only 3%. The added energy degradation of LB + S and Dual + S make them less valuable than their non-SRAM buffer counterparts for the energy/performance tradeoff.

IX. Conclusion

DWM queues provide new control solutions without an obvious analog from constructing queues using traditional array-based memories. While many different control schemes are possible, it appears that in all cases for DWM, spending idle cycles to align the leading data with the read access point (the shift-to-read control strategy) outperforms aligning to the write access point or staying in place (not proactively shifting). Also, while for LB the maximum useful shifts in a cycle provides a clear performance ceiling, when the cycle is sized to the read access latency, one more than the size of the maximum gap between read heads $G + 1$ provides a practical ceiling on how many shifts-per-cycle are beneficial. Furthermore, this read latency-based cycle time limitation results in the saturation region occurring at 50% reads/writes per cycle, regardless of how many read access points can be included. Increasing the cycle time to allow shifts before or after a read in a cycle removes this 50% limit for LB, and would allow the queue to have consistent performance in a system at the cost of reduced operating frequency. Dual, which does not have the

limitation of being unable to perform consecutive read + write operations, always has a significantly later onset of saturation and likely can result in a faster operation frequency than LB.

In the context of using DWM to build NoC queues, while the inherent composition of Racetrack memory can result in a significant energy reduction from traditional SRAM, a direct replacement of Racetrack memory with well-established SRAM FIFO techniques (CB) results in significantly reduced performance (over 300% increase in message latency without an SRAM buffer, and 13% on average with an additional SRAM buffer). A conventional FIFO implementation performs poorly whereas the Dual provided a 56% improvement over SRAM FIFOs with a nominal performance disadvantage. Due to the reduced number of read–write heads in DWM as compared to SRAM or STT-MRAM, 2X the number of DWM queues can occupy the same space as X STT-MRAM queues. When comparing the Dual results to the configurations with half the number of queues for STT-MRAM or SRAM, Dual improves the energy-delay product by 17% and 30%, respectively.

Of the three enhancements to the DWM design discussed in Section III-A, each of which can be applied orthogonally, we note that logic design optimization provides the most significant improvements in both the benchmark and the sensitivity study. For example, while CB saturates at $<30\%$, LB and Dual delay saturation to $>50\%$ and $>90\%$ load, respectively, while also providing latency improvements in the presaturation region (Fig. 13). In comparison, adding additional read heads and/or an SRAM buffer can help to improve the presaturation latency at the cost of increased energy, and for the latter case, loss of nonvolatility, they do not provide nearly the same benefit as physical design improvements such as Dual compared to LB. Thus, potential future work might include prediction, possibly collaboratively designed with the scheduler, to preshift for read or write alignment to reduce the average access latency as a method to reduce adding additional read heads or SRAM buffers to achieve latency goals.

REFERENCES

[1] A. Annunziata *et al.*, "Racetrack memory cell array with integrated magnetic tunnel junction readout," in *IEDM Tech. Dig.*, 2011, pp. 3–24.

[2] S. S. P. Parkin, M. Hayashi, and L. Thomas, "Magnetic domain-wall racetrack memory," *Science*, vol. 320, no. 5873, pp. 190–194, 2008.

[3] S. Parkin, "Racetrack Memory: A storage class memory based on current controlled magnetic domain wall motion," in *Proc. DRC*, 2009, pp. 3–6.

[4] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "DWM-TAPESTRI—An energy efficient all-spin cache using domain wall shift based writes," in *Proc. DATE*, 2013, pp. 1825–1830.

[5] Y. Zhang, W. S. Zhao, D. Ravelosona, J.-O. Klein, J. V. Kim, and C. Chappert, "Perpendicular-magnetic-anisotropy CoFeB racetrack memory," *J. Appl. Phys.*, vol. 111, no. 9, p. 093925, 2012.

[6] 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. ISLPED*, 2012, pp. 185–190.

[7] Z. Sun, W. Wu, and H. H. Li, "Cross-layer racetrack memory design for ultra high density and low power consumption," in *Proc. ACM 50th Annu. Design Autom. Conf.*, 2013, pp. 1–6.

[8] H. Xu, Y. Li, R. Melhem, and A. K. Jones, "Multilane racetrack caches: Improving efficiency through compression and independent shifting," in *Proc. ASPDAC*, 2015, pp. 417–422.

[9] Y. Zhang, W. Zhao, J.-O. Klein, D. Ravelsona, and C. Chappert, "Ultra-high density content addressable memory based on current induced domain wall motion in magnetic track," *IEEE Trans. Magn.*, vol. 48, no. 11, pp. 3219–3222, Nov. 2012.

[10] R. Nebashi *et al.*, "A content addressable memory using magnetic domain wall motion cells," in *Proc. VLSIC*, Jun. 2011, pp. 300–301.

[11] W. Zhao, D. Ravelosona, J. Klein, and C. Chappert, "Domain wall shift register-based reconfigurable logic," *IEEE Trans. Magn.*, vol. 47, no. 10, pp. 2966–2969, Oct. 2011.

[12] W. Zhao, N. Ben Romdhane, Y. Zhang, J.-O. Klein, and D. Ravelosona, "Racetrack memory based reconfigurable computing," in *Proc. FTFC*, 2013, pp. 1–4.

[13] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. H. Li, "Exploration of GPGPU register file architecture using domain-wall-shift-write based racetrack memory," in *Proc. DAC*, 2014, pp. 1–6.

[14] L. Thomas *et al.*, "Racetrack memory: A high-performance, low-cost, non-volatile memory based on magnetic domain walls," in *IEDM Tech. Dig.*, Dec. 2011, pp. 24.2.1–24.2.4.

[15] R. Venkatesan, V. K. Chippa, C. Augustine, K. Roy, and A. Raghunathan, "Energy efficient many-core processor for recognition and mining using spin-based memory," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jun. 2011, pp. 122–128.

[16] F. Jafari, Z. Lu, A. Jantsch, and M. H. Yaghmaee, "Buffer optimization in network-on-chip through flow regulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 12, pp. 1973–1986, Dec. 2010.

[17] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proc. ISCA*, New York, NY, USA, 2009, pp. 196–207.

[18] H. Jang, B. S. An, N. Kulkarni, K. H. Yum, and E. J. Kim, "A hybrid buffer design with STT-MRAM for on-chip interconnects," in *Proc. NoCS*, 2012, pp. 193–200.

[19] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proc. HPCA*, pp. 50–61, Feb. 2011.

[20] Z. Sun *et al.*, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *Proc. ACM MICRO*, 2011, pp. 329–338.

[21] Z. Sun, X. Bi, A. K. Jones, and H. Li, "Design exploration of racetrack lower-level caches," in *Proc. ACM ISLPED*, 2014, pp. 263–266.

[22] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan, "STAG: Spintronic-Tape Architecture for GPGPU cache hierarchies," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 253–264.

[23] W. S. Zhao *et al.*, "Magnetic domain-wall racetrack memory for high density and fast data storage," in *Proc. IEEE ICSICT*, Oct./Nov. 2012, pp. 1–4.

[24] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[25] *Synopsys Design Compiler*, Synopsys Inc., Mountain View, CA, USA, 2000.

[26] J. E. Stine *et al.*, "FreePDK: An open-source variation-aware design kit," in *Proc. MSE*, 2007, pp. 173–174.

[27] M. Lis *et al.*, "Scalable, accurate multicore simulation in the 1000-core era," in *Proc. ISPASS*, Apr. 2011, pp. 175–185.

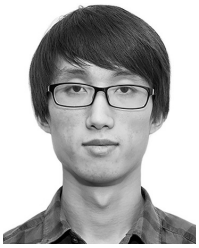
[28] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. SC*, Nov. 2011, pp. 1–12.

[29] C. Bienia, S. Kumar, and K. Li, "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *Proc. IEEE Int. Symp. Workload Characterization*, Sep. 2008, pp. 47–56.

[30] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. Amsterdam, The Netherlands: Elsevier, 2004.

Donald Kline, Jr., (S'13–M'15) received the B.Sc. degree in computer engineering from the University of Pittsburgh, Pittsburgh, PA, USA, in 2015, where he is currently working toward the Ph.D. degree in electrical and computer engineering under the guidance of Dr. A. Jones.

Mr. Donald is a John A. Jurenko Graduate Fellow of the University of Pittsburgh and an NSF Graduate Research Fellow. His current research interests include computer architecture, reliability, emerging memories, compilers, and machine learning.



Haifeng Xu received the B.S. degree in electronic and information engineering from Zhejiang University, Hangzhou, China, in 2009 and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Pittsburgh, Pittsburgh, PA, USA, in 2012 and 2015, respectively.

His current research interests include wireless sensor network and distributed computing with commodity computers, high-performance computer architecture, and emerging memory technologies.

Rami Melhem (S'82–M'83–SM'97–F'00) received the B.E. degree in electrical engineering from Cairo University, Giza, Egypt, in 1976 and the M.A. degree in mathematics and the M.S. and Ph.D. degrees in computer science from the University of Pittsburgh, Pittsburgh, PA, USA, in 1981 and 1983, respectively.

In 1986, he was with the faculty of the University of Pittsburgh. He was an Assistant Professor at Purdue University, West Lafayette, IN, USA, where he is currently a Professor at the Computer Science Department, which he chaired from 2000 to 2009. His current research interests include power management, parallel computer architectures, real-time and fault-tolerant systems, optical networks, and high performance.

Dr. Melhem is a member of the Association for Computing Machinery (ACM). He served and is serving on program committees of numerous conferences and workshops and on the editorial boards of the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE COMPUTER ARCHITECTURE LETTERS, the *Journal of Parallel and Distributed Computing*, and the *Journal of Sustainable Computing, Informatics and Systems*.

Alex K. Jones (S'02–M'03–SM'08) received the B.S. degree in physics from the College of William and Mary, Williamsburg, VA, USA, in 1998 and the M.S. and Ph.D. degrees in electrical and computer engineering from Northwestern University, Evanston, IL, USA, in 2000 and 2002, respectively.

He is currently an MCSI Sustainability Faculty Fellow, a Professor of Electrical and Computer Engineering and Computer Science (by courtesy), and the Director of Computer Engineering at the University of Pittsburgh, Pittsburgh, PA, USA. His research is funded by the U.S. National Science Foundation, DARPA, CCC, the Mascaro Center for Sustainable Innovation, the NSF SHREC Center, and industry. His current research interests include compilation techniques for configurable systems and architectures, behavioral and low-power synthesis, parallel architectures and networks, radio frequency identification and sensor networks, sustainable computing, and reliable computing and memories. He has authored over 150 publications in these areas.

Dr. Jones is a Walter P. Murphy Fellow of Northwestern University and a Senior Member of the Association for Computing Machinery (ACM).