More is Less: Improving the Energy Efficiency of Data Movement via Opportunistic Use of Sparse Codes

Yanwei Song and Engin Ipek
Department of Electrical and Computer Engineering
University of Rochester, Rochester, NY 14627 USA
{yanwei, ipek}@ece.rochester.edu

ABSTRACT

Data movement over long and highly capacitive interconnects is responsible for a large fraction of the energy consumed in nanometer ICs. DDRx, the most broadly adopted family of DRAM interfaces, contributes significantly to the overall system energy in a wide range of computer systems. To reduce the energy cost of data transfers, DDR4 adopts a pseudo open-drain IO circuit that consumes power only when transmitting or receiving a **0**, which makes the IO energy proportional to the number of **0**s transferred over the data bus. A data bus invert (DBI) coding technique is therefore supported by the DDR4 standard to encode each byte using a small number of **0**s. Although sparse coding techniques that are more advanced than DBI can reduce the IO power further, the relatively high bandwidth overhead of these codes has heretofore prevented their application to the DDRx bus.

This paper presents MiL (More is Less), a novel data communication framework built on top of DDR4, which exploits the data bus under-utilization caused by DRAM timing constraints to selectively apply sparse codes, thereby reducing the IO energy without compromising system performance. Evaluation results on a set of eleven parallel applications show that MiL can reduce the average IO interface energy by 49%, and the average DRAM system energy by 8% when added on top of a conventional DDR4 system, with less than 2% performance degradation on average.

Categories and Subject Descriptors

B.4.3 [**Hardware**]: Interconnections

Keywords

Energy-efficient Design, Memory Interfaces, Sparse Representation

MICRO-48, December 05-09, 2015, Waikiki, HI USA
ACM 978-1-4503-4034-2/15/12 ...$15.00.
DOI: http://dx.doi.org/10.1145/2830772.2830806

1. INTRODUCTION

Data movement is responsible for a substantial and growing fraction of the overall energy dissipation in nanometer ICs [1]. IO energy is a major component of the overall data movement energy. The off-chip interconnects typically span much longer routing distances than the on-chip interconnects, and are therefore more capacitive. Consequently, accessing the off-chip main memory can cost 250× more energy than accessing an on-chip cache [2]. Due to the increasing demand on off-chip memory bandwidth, IO energy is expected to become an even more critical problem in future systems. In Figure 1, a breakdown of the total DRAM power is depicted for different DRAM modules. Despite the incorporation of low-power signaling techniques into the DDRx standard, the IO interface still is responsible for 42% of the aggregate DRAM power in current generation DDR4 modules. Hence, architectural techniques that can increase the energy efficiency of data transfers over the DDRx IO interface can go a long way toward reducing DRAM energy.

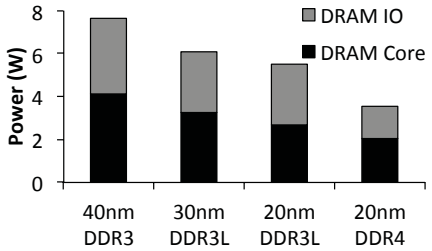

Figure 1: DRAM power breakdown [3].

This paper presents MiL (More is Less), a novel data communication architecture built on top of the latest generation of DDRx interfaces in commercial use (DDR4 and LPDDR3). MiL reduces the data movement energy by exploiting the asymmetry in the energy cost of transferring a **1** vs. a **0** over the DDRx IO interface. In V_{DDQ}-terminated IO interfaces such as DDR4, the energy cost of transferring a **0** over the interconnect is orders of magnitude higher than that of transferring a **1** [4], which makes the energy consumption proportional to the number of **0**s communicated through the interface. As a result, a sparse data representation that reduces the number of **0**s at the expense of a longer codeword can improve the energy efficiency. DDR4

supports one such sparse representation, the DBI code, at the expense of eight extra IO pins, increasing the data pin count from 64 to 72. It is possible to apply sparse codes that are more effective than DBI [5] to the IO power problem; however, accommodating the longer codewords that are typically encountered with such codes requires either a significant increase in the number of pins (a scarce resource), or in the DRAM burst length (which wastes off-chip bandwidth). Note that the latter approach would not only hurt the execution time, but would also lead to greater DRAM background energy. Figure 2 shows the effect of using the (8,17) 3-limited weight code (3-LWC) [6] instead of the DBI code on execution time, IO energy, and system energy for two representative benchmarks, CG and GUPS[1]. The 3-LWC code respectively reduces the IO energy by 1.7× and 3.1× on CG and GUPS; however, the execution times of the applications increase by 14% and 42% due to the increase in off-chip bandwidth demand, which results in marginal system energy savings.

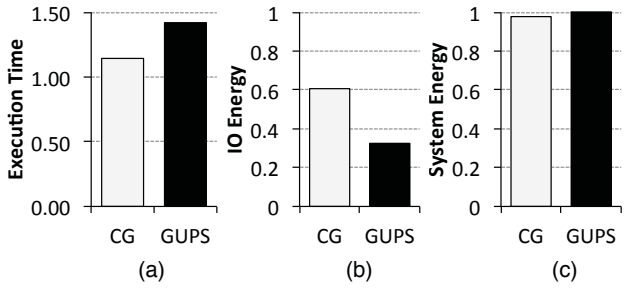

Figure 2: (a) Execution time, (b) IO energy, and (c) system energy, all normalized to the DBI baseline when using the (8,17) 3-LWC for CG and GUPS.

Key Idea. Due to DRAM timing constraints, the DDRx data bus is under-utilized in many situations, even during periods of high demand when multiple pending memory requests are queued at the memory controller. In such situations, the memory bus lies idle while waiting for the DRAM timing constraints to be resolved; as a result, the burst length for the request immediately preceding such an idle interval can be increased without hampering DRAM throughput. We propose to identify these periods of execution, and opportunistically use the (otherwise idle) data bus cycles to transfer an energy efficient, sparse coded version of the original data. Although in principle MiL can work with any sparse code, a simple yet effective code, MiLC (More is Less Code), is proposed here as one base coding scheme for use with MiL. One design configuration of MiL is evaluated over two types of energy constrained systems: DDR4 based microservers [7], and LPDDR3 based mobile systems [8].[2] The application of MiL to these two systems is evaluated on a set of eleven parallel applications. The results indicate that MiL reduces the interface energy by 49% and 46%, and overall DRAM system energy by 8% and 17% for DDR4 and LPDDR3 systems, respectively. These energy reductions are achieved at the expense of a 2% performance degradation in DDR4 based microservers, and a 4% performance degradation in LPDDR3 based mobile systems.

[1]The experimental setup is explained in Section 6.

[2]We choose LPDDR3 in our study of mobile memory interfaces due to a lack of detailed information on power and current draw for the next generation LPDDR4 interface.

2. BACKGROUND AND RELATED WORK

This paper aims at improving the energy efficiency of the main memory interface, which requires optimizing the high-bandwidth off-chip interconnect for low-energy data transfers. It is therefore important to understand the energy efficiency of data movement under different signaling techniques, and with different types of interconnect.

2.1 DDRx IO Interface

The DDRx standard defines different types of interfaces (*e.g.*, DDR4, LPDDR3) to address the performance and energy requirements of different market segments. To guarantee signal integrity at high clock speeds, on-die termination usually is adopted in systems with multiple ranks on a DRAM channel. All of the recent DDRx interfaces, including DDR2/3/4, and (optionally) LPDDR3/4, support on-die termination in different forms; however, on-die termination typically is unnecessary in mobile systems that employ a point-to-point configuration.

2.1.1 DDR4: A Terminated Interface

To reduce the IO energy, DDR4 adopts the pseudo open drain (POD) signaling scheme[3] [9] shown in Figure 3(a). The interesting property of the POD signaling scheme is that the transmission of only the **0**s costs energy due to the current flow from VDD to GND; the transmission of **1**s effectively is free. This asymmetry provides an opportunity to apply coding techniques that reduce the number of **0**s in the transferred data. DDR4 chips with x8 and x16 output widths hence support data bus inversion (DBI) coding [4], which reduces both the worst-case simultaneous switching output (SSO) noise and the IO interface energy. The DBI coding is applied at the byte granularity, and requires one extra pin to transmit a *DBI bit* for each group of eight data pins. When the number of **0**s in a given byte of data is greater than four, the DBI bit is set to **0**, and the ones' complement of the byte is transmitted (*i.e.*, the eight data bits are inverted). Otherwise, the original eight data bits are transferred with the DBI bit set to **1**. This guarantees the total number of **0**s in each byte of transferred data is less than five. A DDR4 chip with a x4 output width can reap only modest benefits from DBI coding; as a result, DBI coding is not supported by the x4 chips.

[3]GDDR5 also uses POD, while LPDDR4 uses a low voltage swing with VSSQ termination. However, the asymmetry in the energy when transferring different values (**0** or **1**) still exists in LPDDR4.

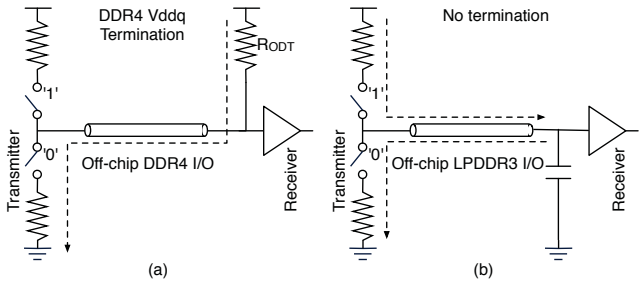

Figure 3: DDRx IO Interface: (a) a terminated interface (VDDQ termination) in DDR4, (b) an unterminated interface in LPDDR3.

2.1.2 LPDDR3: An Unterminated Interface

Figure 3(b) shows the unterminated IO interface supported by the LPDDR3 standard, which consumes energy by charging and discharging the load capacitance of the data bus [10]. Different from the DDR4 IO interface, the IO interface of LPDDR3 consumes energy on the **0**→**1** and **1**→**0** transitions. As as result, minimizing the number of zeroes in the transferred data is not necessarily a wise strategy for LPDDR3, and LPDDR3 does not support any native coding techniques. This problem, however, can be overcome relatively easily by relying on bus invert (BI) coding [11], an early predecessor of DBI. In bus invert coding, one extra BI pin is paired with each group of eight data pins. When the number of transitions is less than five, the original eight data bits are transferred with the BI bit set to **0**. When the number of transitions is greater than four, the data bits are inverted with the BI bit set to **1**. In addition, transition signaling (Section 4.5) can make the number of bit flips on the bus equal to the number of transmitted zeroes.

2.2 Low Energy Data Encoding Schemes

Energy efficient data encoding has been heavily studied in the literature. Bus-invert coding [11], the fundamental principle behind the data bus invert (DBI) coding [4] used in DDR4, was introduced in 1995. Transition signaling [12, 13] makes the number of bit flips on the bus (and therefore the bus energy) depend only on the hamming weight of the transmitted data, even if the bus is unterminated. The k-limited weight codes (k-LWC) [5] are a class of codes which constrain the maximum hamming weight of a given block of data to k. Stan et al. first proposed 3-LWC [6], a simple code which maps an 8-bit block of data to a 17-bit code, and later invented the perfect 3-LWC [14] as the dual of the Golay code, which maps 11 bits of data to a 23-bit sparse representation with at most three 1s. All of these coding schemes are applied by increasing the interconnect width, or transferring a given (long) codeword over multiple cycles.

Two survey papers [15, 16] give a comprehensive introduction to various encoding techniques for both on-chip and off-chip interconnects. Stan and Burleson [12] also present a general coding framework which takes into account factors such as space, time, and voltage amplitude. Various adaptive code-book based methods [17, 18, 19] exploit the temporal correlations among different cache blocks by caching the recently communicated data. Childers and Nakra [20] reorder the data in a burst to reduce the switching activity on the data bus.

Sparse data representation finds application not only in energy efficient data movement, but also in minimizing write energy and wear-out for emerging memory technologies. Flip-N-Write [21] applies bus invert coding to phase change memory (PCM). The recently published CAFO [22] is a form of two dimensional bus-invert coding, which alternates row and column bus inverting in an iterative fashion until no more Hamming weight reduction can be achieved. In contrast, the goal of the proposed MiL framework is to improve the energy efficiency of data movement over the DDRx data bus via the opportunistic use of sparse codes. As a coding scheme, CAFO is comparable to MiLC in the MiL framework. MiLC exploits the correlation among bytes that are spatially close to one another, and achieves a deterministic latency, which simplifies scheduling. CAFO exhibits a non-deterministic coding latency, which makes it much harder for the memory controller to schedule the commands. If the number of iterations for CAFO is fixed, the reduction in the number of zeroes is compromised. Moreover, the memory controller has to consider the worst-case coding latency at all times. In other words, CAFO suffers from limitations when used under the MiL framework. CAFO requires iterative search in two dimensions, which incurs higher design complexity and hardware overhead, and its encoding latency is higher than that of MiLC.

2.3 Circuit Level Low Power Signaling Techniques

Low-swing signaling can lower the interconnect energy by using a low output voltage swing at the expense of a higher latency. LPDDR4 has adopted this circuit optimization to improve the IO power [23]. Wide I/O [24] and High Bandwidth Memory (HBM) [25] are two memory standards with a wide interface operating at a low frequency to achieve higher power efficiency at the expense of extra pins and greater cost. Hybrid Memory Cube (HMC) [26, 27] changes the IO interface to SERDES to improve both the bandwidth and the energy efficiency when the bandwidth is highly utilized; however, when the bandwidth is underutilized, it costs more energy than the conventional parallel DDRx interface due to the constant static power consumption of SERDES. Li et al. [28] propose a photonic interconnect as a substitute for the conventional electrical memory interface to improve the signal integrity and power efficiency, which requires much more radical hardware changes as compared to MiL. MiL is not applicable to HMC or photonic interfaces.

3. DATA BUS UNDER-UTILIZATION AND THE POTENTIAL FOR SPARSE CODING

The DDRx data bus is often under-utilized due to

DRAM timing constraints, even when executing memory-intensive applications. The under-utilization creates an opportunity to apply sparse codes to data transfers without degrading system performance.

3.1 Data Bus Under-Utilization

Due to DRAM timing and resource constraints, the DDRx data bus utilization often falls well below 60% [29, 30, 31, 32, 33, 34, 35], even during periods in which many pending requests are queued at the memory controller. Over time, the DDRx family of standards has evolved toward a more heavily constrained interface to accommodate various circuit level constraints (*e.g.*, peak power, noise). The DDR4 interface, for instance, has introduced new timing constraints that are absent from the earlier generation DDR3 by dividing banks into multiple bank groups [36], and making the tRRD, tWTR, and tCCD parameters depend on whether consecutive commands are in the same bank group or in different groups.

Figure 4 shows the distribution of idle cycles between successive transactions on the data bus. The bus transactions occur back-to-back in only 13% of the cases. Notably, it is possible for the data bus to be idle due to two different reasons: 1) there may be no pending requests at the memory controller; and 2) timing constraints may prevent keeping the data bus utilized even when requests are pending. Figure 5 shows how frequently these two situations occur. The applications MG, FFT, SCALPARC, SWIM, OCEAN, CG, and GUPS are memory intensive, and result in pending requests at the memory controller a majority of the time. However, the data bus remains idle due to timing constraints in more than half of these cycles when the requests are pending.

MiL aims at increasing the energy efficiency of data movement by applying sparse codes during otherwise idle data bus cycles. Regrettably, not all idle cycles can be exploited without hurting system performance. Specifically, if two successive transactions on the data bus require a bus turnaround (and therefore are separated by idle bus cycles), applying sparse codes to the first transaction would delay the starting time of the second transaction. As such, the extra delay incurred by sparse codes cannot be hidden when the relevant timing constraints that cause the idle cycles are due to bus turnaround (*e.g.*, tWTR, tRTRS, and tOST). We define the *slack* between two successive data bus transactions as the number of cycles by which the end of the first transaction can be postponed (in order to use sparse codes) without delaying the beginning of the second transaction. Figure 6 shows the distribution of slack between successive data bus transactions. The results indicate that in many (but not all) cases, the bus turnaround does not limit the application of longer sparse codes. All in all, the wasted data bus bandwidth offers a good opportunity to improve the energy efficiency of data movement over the DDR4 interface.

3.2 Potential of Sparse Coding

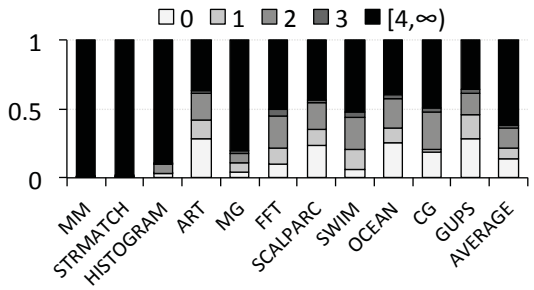

Figure 4: The distribution of idle cycles between successive transactions on the DDR4 data bus.

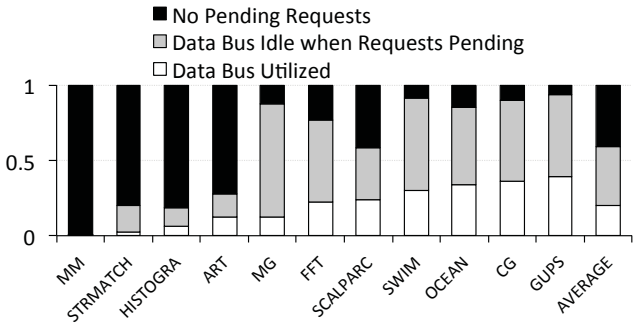

Figure 5: The distribution of time among cycles when there are no pending requests, cycles when the data bus is idle despite the presence of pending requests, and cycles when the data bus is utilized. Benchmarks are sorted based on the utilization from low to high.

Since the DDR4 interface has already adopted the DBI coding technique to improve the energy efficiency, it is useful to study how much potential there is to reduce the energy further by using more sophisticated coding techniques. To reduce the size of the design space, only static codes in which a code is mapped to a unique data pattern are studied. In order to make the comparison fair, all of the codes are configured to have the same overhead as DBI (*i.e.*, 8-bit data are paired with a single extra bit). Figure 7 shows the normalized number of zeroes achieved by different codes. Compared to DBI, these codes show significant potential to lower the IO energy; however, the codec logic for these codes are hard to implement algorithmically, while a look-up table based codec is impractical due to exorbitant capacity overheads. Therefore, MiL adopts simple algorithmic codes; the study of other coding schemes is left for future work.

4. MORE IS LESS: OPPORTUNISTIC DATA ENCODING

The proposed "More is Less" (MiL) framework is a general opportunistic data encoding interface for point-to-point communication, but this paper focuses exclusively on the DDRx interface. MiL exploits the idle cycles on the data bus due to the inherent timing con-

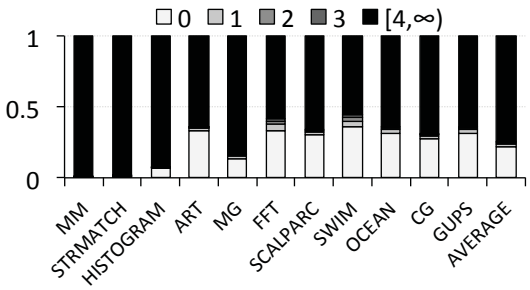

Figure 6: The distribution of slack between successive data bus transactions.

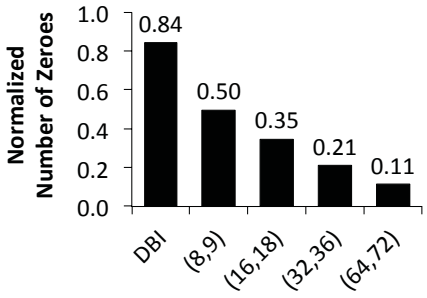

Figure 7: Relative reduction in the number of zeroes achieved by optimal static LWC codes. (8, 9) denotes an LWC which optimally encodes an 8-bit data pattern into a 9-bit code according to the frequency of different data patterns. The number of zeroes observed with different schemes are normalized to the number of zeroes observed on the original data.

straints of the DDRx interface to reduce the data movement energy.

4.1 Overview

Figure 8 illustrates an example DDRx system with and without the proposed MiL framework. MiL is engaged on column read or write commands. In a conventional DDRx system, timing constraints often result in idle cycles between two successive data bursts. In a MiL based DDRx system, the memory controller identifies an opportunity to exploit these idle cycles, and decides whether to apply the sparse coding (Section 4.2). If sufficiently many idle cycles are identified, a sparse encoded version of the original data is sent to save energy (Section 4.3). As a result, the data movement energy is reduced even though the bus utilization is increased (*i.e.*, more bits are sent over the bus with less overall energy). As shown in Figure 8, read *0* and read *1* are separated in time due to a timing constraint, which results in the idle cycles between data burst *0* and data burst *1* in a conventional DDRx system. In the MiL based DDRx system, the data movement energy saving is achieved by transferring data burst *0* in a sparse representation without hurting the performance. This approach is more cost-effective than adding data pins to the memory chip; moreover, unlike the case of DBI, x4 chips can benefit from MiL.

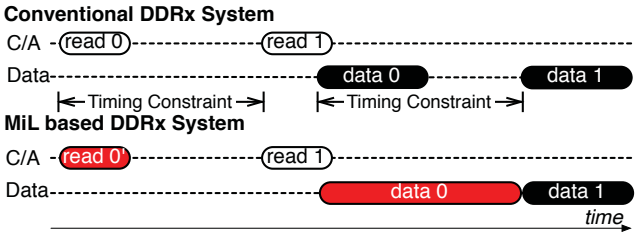

Figure 8: Illustrative example of the key idea. Note that MiL is applicable to both reads and writes.

The design of the MiL framework (Figure 9) is decoupled from that of the memory controller for the sake of simplicity. It consists of two major components: 1) the decision logic at the memory controller, which determines when the sparse coding should be applied; and 2) the codec logic, which implements specific coding algorithms on both the memory controller and the DRAM sides. Besides these two components, minor modifications are made to an existing memory controller.

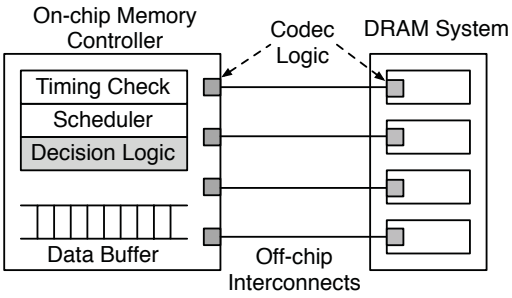

Figure 9: The design of a MiL-based memory system.

4.2 When to Code?

The decision logic is activated only when a ready read or write command is scheduled, and is used to decide which coding scheme to apply. Deciding when to apply a sparse representation affects both the data movement energy and the system performance. If the system performance degrades too much, the energy consumed by other components, such as the static energy due to DRAM background power, will be sufficiently large to nullify the energy savings on the IO interface. The decision to use sparse encoding is made based on an intuitive yet effective heuristic: 1) assume the data will be coded in an X-cycle sparse representation; 2) although incoming requests may create new scheduling opportunities that can fill otherwise idle bus cycles, use the current commands in the queue to infer the approximate behavior of the schedule in the next few cycles; 3) check whether there are any ready reads or writes that can issue in the next X cycles. If there is no such read or write, a wider sparse code is allowed, otherwise, a simpler code or the original data are transferred. This approach avoids significantly delaying pending requests, and prevents system performance degradation.

4.3 How to Code?

Based on the decision made by the decision logic, either the original data or a sparse coded version of that data is transmitted. We study two types of sparse codes: 1) the existing, low-overhead 3-limited-weight code (3-LWC) [6]; and 2) a newly proposed "More is Less Code" (MiLC). Other coding approaches can be adopted in the MiL framework so long as the codec latency is deterministic, which makes it possible for the memory controller to schedule all requests while delaying the DDRx timing constraints.

4.3.1 3-Limited-Weight Code

The limited-weight code (LWC) [5], which restricts the hamming weight of a codeword to be below a fixed number, is a sparse representation of the original data that can be used to greatly reduce the number of zeroes.[4] For example, the bus inverting code is an $\frac{n}{2}$-LWC (n denotes the number of bits in the original data), whereas a one-hot code is a 1-LWC. However, the algorithmic generation of an arbitrary LWC is hard. Stan et al. [6] propose a 3-LWC with a very low hardware overhead, which encodes 8-bit data to 17-bit codewords with a hamming weight of at most 3. We make a small improvement over the original coding algorithm (Section 5.2.2), and apply the modified coding scheme (3-LWC) to the data at an 8-bit granularity.

4.3.2 More is Less Code

MiLC, a very efficient coding scheme with a low bandwidth overhead, is proposed in this section. MiLC exploits the spatial correlations in the data, and manages to reduce the number of zeroes in a data block better than the recent work on CAFO [22]. The coding scheme is intuitive and exhibits a fixed latency, whereas CAFO requires a variable number of iterations to reduce the number of zeroes. Figure 10 shows the basic idea behind MiLC. The original data at the 64-bit granularity are laid out as an 8×8 square, and the final code is 80 bits (*i.e.*, the square plus the last two columns). Each row of eight bits has four candidate codes as shown at the bottom: 1) inverted and XORed with the previous row, 2) XORed with the previous row, 3) inverted, and 4) original. Two extra mode bits denote the chosen combination. The mode value is selected to achieve the fewest number of zeroes, after which the data are encoded. The decoding step is even simpler: MiLC first decides whether the inversion is needed, and then decides on the application of the XOR based on the selected mode. The gray-colored bit in the first row is used to perform the bus inverting over the other seven xor mode bits in the same column to further reduce the number of **0**s.

MiL applies MiLC as the base coding scheme when the 3-LWC code cannot be applied to achieve the best energy savings.

[4]Strictly speaking, minimizing the number of zeroes requires inverting all of the bits after LWC is applied, because by default LWC minimizes the number of ones as opposed to zeroes.

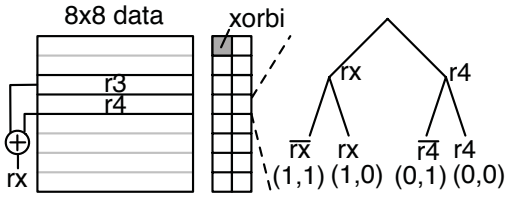

Figure 10: MiLC coding schemes.

4.4 Modifications to the Memory Controller

Minor modifications to an existing memory controller are needed to support different data burst lengths. Fortunately, two of the required features (dynamic burst length, and codec latency) can be readily supported by DDR4 [37].

Dynamically changing the burst length is already supported by the DDR4 interface. Specifically, an extra address pin called "burst chop" specifies a burst length of eight (no burst chop) or four (burst chop), on the fly. Supporting a greater variety of burst lengths requires more pins to differentiate among the alternatives. In this paper, we try to avoid the addition of any pins. Based on a performance sensitivity study to the burst length and the available coding schemes, the new burst lengths are respectively chosen to be 10 and 16 to support MiLC and 3-LWC. This means that MiLC is used when the decision logic decides not to apply the more expensive 3-LWC coding. Note that this is only one design point; the burst length can even be made application-specific with a few candidate coding schemes, which is left for future work.

The DDR4 interface also supports programmable preambles for column read and write commands. This feature can be used when the coding scheme requires a variable number of cycles to finish. MiLC and 3-LWC both exhibit low codec latencies close to that of DBI [5], so no extra preamble is needed at runtime; more sophisticated coding schemes can reuse the preamble to guarantee the right timing for the data burst. Notably, the one or two cycle latency added to the memory request has little effect on the performance of the system.

4.5 Transition Signaling Logic for LPDDR3

In the unterminated LPDDR3 interface, the data movement energy is consumed by the bit flips on the wires. It is possible to apply coding schemes such as bus inversion (BI) directly to the interface; alternatively, transition signaling [12] can convert the energy problem on the unterminated interface to the same logical problem encountered on the terminated interface, in which the optimization only needs to consider the current data. Either the level or the transition of voltage can be used to represent logic **1** and **0**. *Level signaling* represents information by high vs. low voltage, respectively, whereas *transition signaling* represents logic **1** by a voltage level transition on a wire, and logic **0** by the absence of a level transition. Therefore, an energy-efficient coding

[5]One more DRAM cycle is added to the fixed tCL based on the systhesis results.

scheme with transition signaling needs to minimize only the number of **1**s in each data bock to reduce the data movement energy. This makes the MiL framework readily applicable to LPDDR3.

4.6 Optimizations

Occasionally, the sparse data representation generated by MiLC exhibits fewer zeroes than the sparse data representation that results from applying LWC to the same original data. One optimization that the memory controller can apply to writes is to encode the data with both coding schemes ahead of time, and pick the better option (*i.e.*, the code that results in fewer zeroes, with no extra latency imposed on the following column command). This write optimization is implemented in MiL; however, the optimization is not applicable to reads, since the memory controller cannot inspect the data at the moment the read command is scheduled.

More changes can be made to the memory controller to improve the results. This paper takes the strategy of decoupling functional components completely in the memory controller for low complexity, thereby requiring no modifications to the command scheduler of the memory controller. A coding-aware memory controller can combine the decision logic with the scheduler, which should result in better coding decisions based on greater knowledge of scheduling decisions. For instance, the memory writes, which are not on the critical path, can be coded by finding the right available slots on the data bus. In the current version of MiL, we forego such optimizations for simplicity: only a simple write drain mode [38] is implemented to mitigate the effect of the tWTR bus turnaround constraint.

5. IMPLEMENTATION

The implementation of MiL is straightforward and carries low overhead. This section discusses each component (the decision logic at the memory controller, and the codec logic on both sides of the DRAM interface) of the MiL framework.

5.1 Decision Logic

A high-level explanation of the decision logic is given in Section 4.2, and the implementation turns out to be straightforward. In MiL, a group of comparators is used to check whether a column command (read or write) will become ready within the next X cycles, where X is the burst length (in cycles) for 3-LWC. When there is more than one column command, the decision logic picks MiLC rather than 3-LWC to avoid delaying other column commands.

Figure 11 illustrates the concept. All of the timing constraints for the column commands are guaranteed with saturating down counters that decrement every cycle. A zero in a counter indicates that the corresponding timing constraint is met in the current cycle. Similarly, the timing constraint is met within X cycles when the corresponding counter has a value less than or equal to X. By ANDing the ready signals of all of the timing constraints for a column command, as shown by the

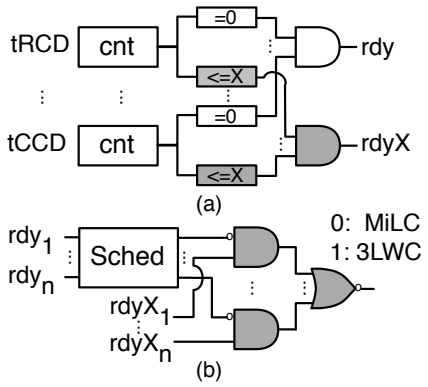

Figure 11: Illustrative example of the implementation of the decision logic: (a) the rdyX generation logic, and (b) the selection logic. Note that only the gray portions are newly added to a conventional memory controller.

"rdyX" signal in Figure 11(a), the readiness within X cycles is known. If the number of ready column commands within X cycles is greater than one, MiLC is picked (*i.e.*, decision logic outputs a **0**). This is shown in Figure 11(b); the output of the scheduler is one-hot encoded to indicate which command is scheduled.

5.2 Codec Logic

The codec logic handles the data encoding and decoding for different coding schemes.

5.2.1 Data Layout

In a DDR4 memory system, a block of data comes from a rank of either eight (x8) or four chips (x16). In order to support critical word first, the data layout is commonly organized as shown in Figure 12(a) [39]. When the data block is transferred on the interface, each chip ships the data in the fashion shown in Figure 12(b), but with the encoded format. For the LPDDR3-based point-to-point system, since all of the data are supplied by a single chip, it is easy to apply a data layout similar to what is shown here. Therefore, without loss of generality, the DDR4 data layout is assumed in the following sections.

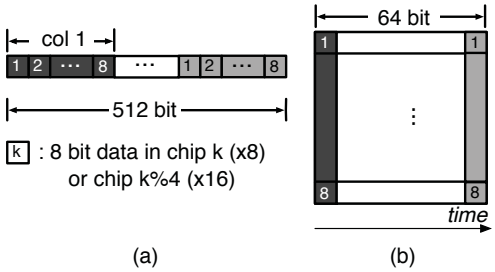

Figure 12: (a) The 512-bit data block layout for DRAM chips, and (b) data block padding for 3-LWC.

5.2.2 3-LWC

The new 3-LWC coding algorithm, which converts 8-bit data to a 17-bit codeword, is applied to each small square (8 bits) of every column, as shown in Figure 12(b). The total number of bits for transmitting 512 original data bits with 3-LWC is 1088 bits; these bits are transferred by reusing some of the existing DBI pins and increasing the burst length to 16 (8 cycles). Figure 13 illustrates the algorithm to generate the code word: 1) a group of 8 data bits is split into two halves, and one hot encoding is applied to generate the 15-bit left and right intermediate forms; and 2) left and right data are ORed to generate the final code, and the mode value is set based on the code, left, and right conditions as shown in Table 1.

The code generation algorithm is from prior work [6], and we further reduce the number of zeroes by reassigning the mode values.[6] Therefore, the maximum number of zeroes within the codeword, including the code and mode bits, is 3. The decoder simply implements the inverse operation based on Table 1, which requires a low hardware overhead.

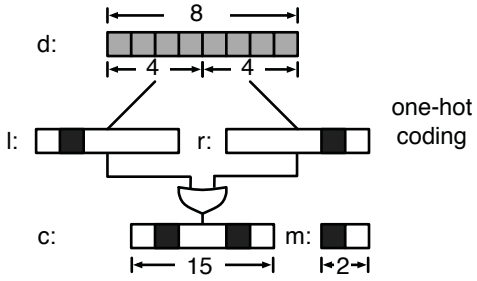

Figure 13: Illustrative example of a 3-LWC encoder: d, l, r, c, m respectively denote data, left, right, code, and mode bits.

Mode	Code	Left	Right
00	all 0s	all 0s	all 0s
01	single 1	single 1	single 1
00	single 1	single 1	all 0s
10	single 1	all 0s	single 1
10	two 1s	greater	smaller
00	two 1s	smaller	greater

Table 1: 3-LWC Mode Generation Table.

5.2.3 MiLC

Encoder. MiLC takes each 64-bit row in Figure 12(b), and converts it into an 8×8 square (Figure 10). Each eight-bit row is encoded in parallel by the row encoder (Figure 14). MiLC picks the code for each eight-bit row with the minimum number of zeroes among the four candidates. Notably, a simplified implementation is used for the first row in Figure 10, in which the xorbi bit is used for bus inverting over the other seven XOR mode bits in the same column.

[6]The insight is that different types of codes can reuse the same mode value, and they can still be distinguished by the code.

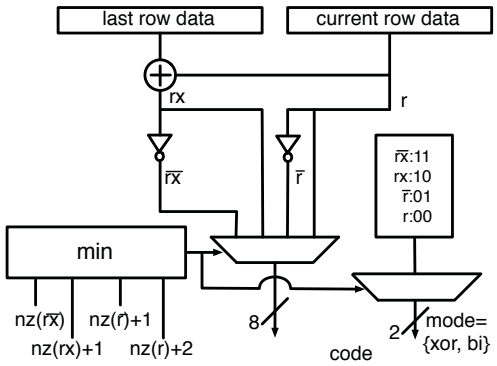

Figure 14: Illustrative example of the MiLC row encoder. The nz function counts the number of 0s in the input 8-bit data, and the additional constant corresponds to the number of 0s in the mode value.

Decoder. The MiLC decoder operates in two steps: 1) perform bit inversion over the 8×8 region and the xor column based on the bus invert bits, including the xorbi bit, in parallel, and 2) XOR the result with the previous row if the XOR bit is set.

5.3 Transition Signaling for LPDDR3

For LPDDR3, MiL uses transition signaling to make the energy of data movement a function of the hamming weight of the original data: transferring a **1** requires the inversion of the voltage level on the wire, and transferring a **0** retains the previous voltage. This is realized by XORing the input bit with the previous value on the wire at the encoder side. The code is then recovered by XORing the current value on the wire with the previous one. Figure 15 shows the implementation of the encoder (left) and the decoder (right). In the figure, C_n represents the actual signal on the interface.

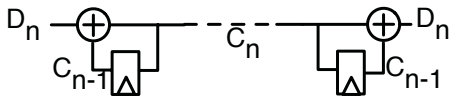

Figure 15: Transition signaling.

6. EXPERIMENTAL SETUP

Assessing the performance, energy, and area of MiL requires both circuit and architecture level design and evaluation. The MiL framework is designed and verified in Verilog RTL with Cadence NCSim [44], and synthesized using the Synopsys Design compiler [45] at 45nm using the FreePDK standard cell library [46]. The numbers are then scaled to a 22nm DRAM process using the methodology [47] [7] and FO4 factors reported in prior work [48, 49]. The efficacy of MiL depends on both the contribution of the memory interface to system power

[7]The peripheral circuitry within the memory array is modeled with ITRS LSTP devices [47].

	Snapdragon-like Mobile System [40, 8]	Niagara-Like Microserver System [41]
Core	8 out-of-order cores 1.6GHz single thread in each core, issue width: 3	8 in-order cores 3.2GHz 4 threads in each core, fetch/issue width: 4/2
IL1 cache (per core)	32KB, direct mapped, 64B line, hit/miss delay 1/1	
DL1 cache (per core)	32KB, 4-way, 64B line, WB, hit/miss delay 2/2	
Coherence	MESI	
L2 cache	2MB, 8-way, 8 banks, 64B line, hit/miss delay 8/4	4MB, 8-way, 8 banks, 64B line, hit/miss delay 16/4
Stream Prefetcher	nstreams/distance/degree: 64/8/1	nstreams/distance/degree: 64/32/4
Memory Controller	scheduling:FR-FCFS [42], address mapping: page-interleaving read queue:64 entries, write queue:64 entries, write drain high/low watermark: 60/50	
Memory	LPDDR3-1600 chips [43, 40] channels/ranks/banks=2/2/8, pageSize:4KB	DDR4-3200 chips [37] channels/rank/banks=2/2/8, pageSize:8KB
DRAM System	CL/WL/CCD_S/CCD_L/RC/RTP/RP/RCD/RAS/WR/RTRS/WTR_S/WTR_L/RRD_S/RRD_L/FAW/REFI/RFC	
DDR4-3200	20/16/4/8/72/12/20/20/52/4/2/4/12/9/11/48/12480/416	
LPDDR3-1600	12/6/4/4/51/6/16/15/34/6/1/6/6/8/8/40/3120/104	

Table 2: Simulated system parameters.

and the bandwidth utilization of the applications. MiL is less effective if the memory interface power is low or if the bandwidth utilization is too high.[8]

6.1 Architecture

We model two systems: a DDR4 based server system, and an LPDDR3 based mobile system (system parameters are shown in Table 2). Simulations are performed on a heavily-modified version of the SESC simulator [50].A stream prefetcher [51] is simulated with the best performing configuration. To mitigate the adverse effect of tWTR on performance, write drain mode [38] is implemented, and the best performing memory controller configuration is picked for the baseline. DRAM is modeled with a detailed, cycle-accurate DDRx timing model that includes the new timing constraints introduced in DDR4, such as the bank group dependent tRRD, tWTR, and tCCD. We use McPAT 1.0 [52] and DDR4/LPDDR3 power calculators [53, 10] to estimate the energy. Table 2 lists all of the modeled DDRx timing parameters. The data bus is modeled with the dynamic burst length feature (10 or 16) when MiL is applied, and with a fixed burst length (8) for the baseline with DBI.

6.2 Applications

In order to assess the impact of MiL on applications with different characteristics, we evaluate 11 applications with different bandwidth utilization and memory intensity. These applications and the corresponding input data sets are shown in Table 3.

7. EVALUATION

The MiL framework is evaluated herein.

7.1 MiL Hardware Overhead

As discussed in Section 5, both the decision logic and the transition signaling circuits exhibit low design complexity. We focus on the synthesis results for the two coding schemes, MiLC and 3-LWC. Table 4 shows the area, power, and latency of the codec logic for a 22nm DRAM process[9]. The power and area costs are negligible; however, the extra latency (up to 0.35ns) is a significant fraction of the DDR4 clock period (0.63ns), and is taken into account by increasing tCL by one clock cycle. MiL also eliminates the need for extra DBI pins by exploiting the idle cycles, which results in a lower pin cost.

[8]This would be the case for streaming memory accesses; however, even memory intensive benchmarks (e.g., GUPS and CG) exhibit abundant idle cycles to be exploited.

Benchmarks	Suite	Input
GUPS	HPCC [54]	2^{25} table, 1048576 updates
CG	NAS OpenMP [55]	Class A
MG	NAS OpenMP	Class A
SCALPARC	NuMineBench [56]	F26-A32-D125K.tab,
HISTOGRAM	Phoenix [57]	small
Matrix Mult	Phoenix	3000 x 3000 matrix
String Match	Phoenix	50MB file
ART-OMP	SPEC OpenMP [58]	MinneSpec-Large
SWIM-OMP	SPEC OpenMP	MinneSpec-Large
FFT	SPLASH-2 [59]	2^{20} complex data points
OCEAN	SPLASH-2	514×514 ocean

Table 3: Applications and data sets.

	Area (um2)	Power (mW)	Latency (ns)
MiLC Enc	1429	3.32	0.35
MiLC Dec	188	0.16	0.39
3-LWC Enc	173	0.44	0.10
3-LWC Dec	81	0.70	0.12

Table 4: Area, power and latency for the MiL codec.

7.2 Performance

The performance results for the DDR4 and LPDDR3 systems are presented in this section. CAFO [22] is a recently published coding scheme that aims at improving the write endurance of non-volatile memories by reducing the number of bit flips. It can be adapted to the MiL framework to reduce the number of zeroes on DDRx bus transfers. CAFO [22] is applied under the MiL framework on 8×8 square data to achieve the same bandwidth overhead as MiLC. Since CAFO benefits from iterative search, the encoding logic requires a clock to synchronize each iteration. We assume one DRAM cycle per iteration for CAFO in this paper. For example, CAFO2 denotes two iterations, including one row and one column search, such that the encoding

[9]Note that MiLC is at the 64-bit granularity, whereas 3-LWC is at the 8-bit granularity.

latency is deterministic (*i.e.*, 2 DRAM cycles) and is added to the tCL. Therefore, CAFO is evaluated with different numbers of iterations and compared to MiLC. It is observed that CAFO with four iterations achieves a comparable number of *0*s as the original CAFO without an extra latency penalty; consequently, two CAFO variations (with two and four iterations) are shown in the following results. In the Figures, the MiLC-only scheme always encodes the data with MiLC.

Figure 16 shows the execution times normalized to the DBI baseline for CAFO2, CAFO4, MiLC-only, and MiL on both DDR4 and LPDDR3 memory systems. All of the results are sorted by memory bus utilization from low to high. The results exhibit a clear trend: the more data-intensive the application is, the more performance degradation the sparse coding incurs. In Figure 16(a), the performance degradation of the highly memory-intensive benchmarks SWIM, OCEAN and CG, are within 4% of a conventional DDR4 system. On average, MiL respectively outperforms CAFO2, CAFO4, and MiLC-only by 2%, 3%, and 1%. CG and GUPS are sensitive to the additional tCL cycles, so MiL and MiLC-only outperform CAFO. Note that MiL is 9% faster than MiLC-only in GUPS, and even outperforms the baseline in STRMATCH. The MiL framework increases the latency of memory requests by applying either MiLC or 3-LWC. This increased latency can result in a higher command queue occupancy, which allows the scheduling policy to pick among more candidate commands. The performance results achieved on the LPDDR3 system are shown in Figure 16(b). The single-threaded core of the evaluated mobile system is more sensitive to the additional memory traffic than the DDR4-based server system. With the look-ahead decision logic, the performance degradation is constrained to less than 4%. Due to the out-of-order execution capability of the processor core, the performance degradation incurred by CAFO2 and CAFO4 on GUPS is reduced to around 10%. HISTOGRAM exhibits trends similar to GUPS.

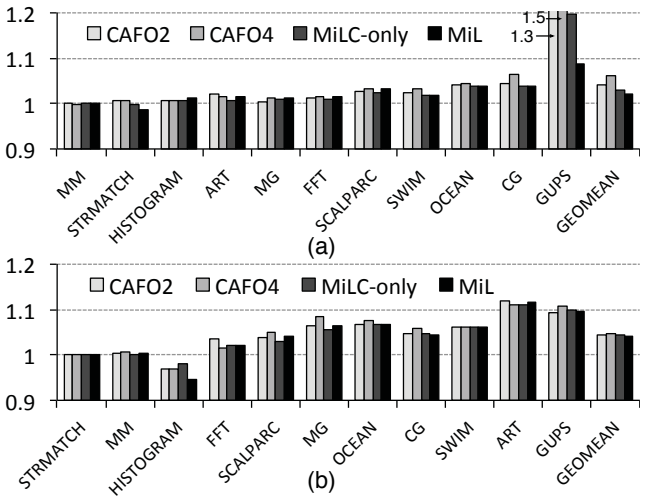

Figure 16: Execution Time normalized to the DBI baseline: CAFO2, CAFO4, MiLC-only, and opportunistic MiL (a) with DDR4, and (b) with LPDDR3.

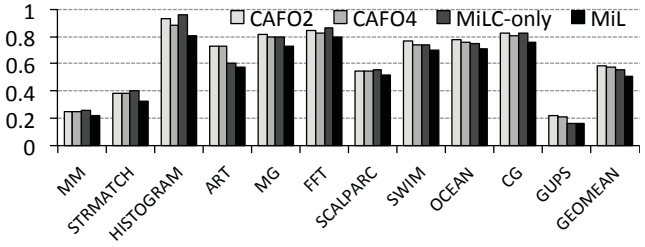

Figure 17: Number of zeroes normalized to the DDR4 DBI baseline.

7.3 Energy Impact on DDR4

Figure 17 illustrates the number of zeroes transferred under four different schemes: CAFO2, CAFO4, MiLC-only, and MiL normalized to the DDR4 DBI code. As compared with DBI, all of these schemes significantly reduce the number of 0s on MM, STRMATCH, and GUPS. On average, MiL respectively outperforms DBI, CAFO2, CAFO4, and MiLC-only by 49%, 12%, 11%, and 9%.

A DRAM energy breakdown is shown in Figure 18(a). The DDR4 background energy is a big contributor to the overall DDR4 energy due to the lack of a fast power down mode, which offsets the IO energy savings. MiL reduces the DDR4 system energy by 8% on average.

The average system energy savings shown in Figure 19(a) are 2.2%, 1.6%, 3.1%, and 3.7% for CAFO2, CAFO4, MiLC-only, and MiL on a server system. The system energy savings are determined by the percentage of memory system energy, data bus bandwidth, and the reduction in the number of *0*s. This explains why MM and STRMATCH (memory non-intensive applications where the memory system energy is not significant) exhibit only small energy savings, although the reduction in the number of *0*s is significant. Memory intensive benchmarks, such as GUPS and SCALPARC, show greater savings due to the relatively high reduction in the number of *0*s. Better sparse coding schemes, and tighter integration of the scheduler and the decision logic, may further improve the energy efficiency. In addition, the new power modes proposed by Malladi *et al.* [60] can reduce background power, and help increase the percentage of system energy savings that MiL can provide.

7.4 Energy Impact on LPDDR3

When applied to LPDDR3, MiL outperforms DBI, CAFO2, CAFO4, and MiLC-only by 46%, 13%, 10%, and 9% in the number of transitions, respectively. This result is similar to the savings achieved on DDR4, because the effect of the coding scheme depends on the characteristics of the application data. Since LPDDR3 is optimized aggressively to reduce the background power consumption, the IO energy is a major contributor to the LPDDR3 DRAM system energy. The average en-

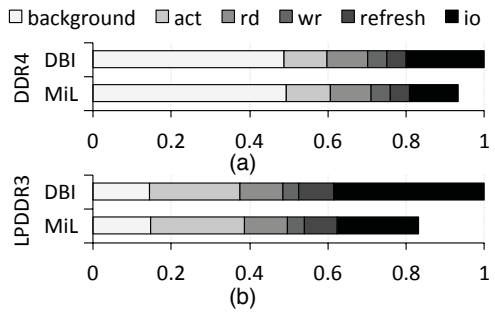

Figure 18: Energy breakdown of DBI vs MIL in (a) DDR4, and (b) LPDDR3.

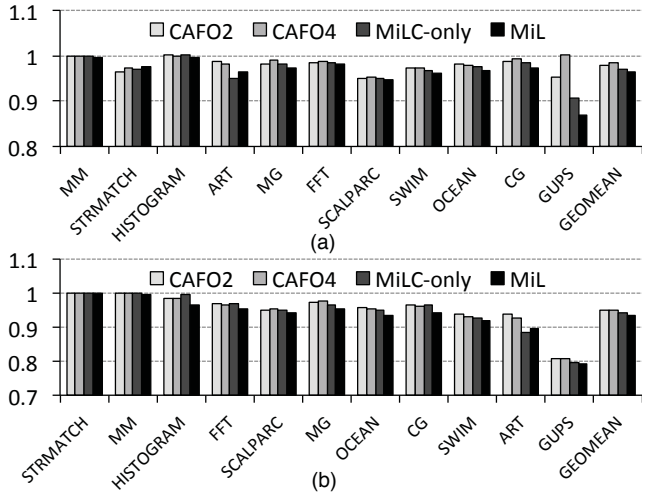

Figure 19: (a) DDR4 and (b) LPDDR3 MiL-based system energy normalized to the DBI baseline.

ergy reduction observed on the LPDDR3 system is 17%, as shown in Figure 18(b). Due to the energy efficiency of the mobile cores, the system energy is not sensitive to the small performance degradation. Applications that heavily utilize the data bus (*e.g.*, SWIM, ART, GUPS) expend a greater fraction of the memory system energy at the IO interface, which results in greater system energy savings with MiL. Figure 19(b) shows that MiL achieves 7% average system-wide energy savings, which is greater than the savings achieved by CAFO2 (5%), CAFO4 (5%) and MiLC-only (6%).

7.5 Analysis and Discussion

Sensitivity studies were performed on both DDR4 and LPDDR3 systems. Only the DDR4 results are shown here for brevity; the LPDDR3 based system exhibits similar characteristics.

7.5.1 *Fixed Burst Length: Coding With A Single Scheme*

Recall that the distribution of the idle cycles between successive data bursts was analyzed in Section 3.1. A naïve approach to exploit the idle cycles is to perform a specific sparse coding at all times to improve the energy efficiency. Intuitively, the longer the sparse code is, the less energy is consumed by the memory interface. However, the longer burst length required by the sparse code can delay ready column commands and degrade the performance, which in turn can increase the background energy consumption of the system. In other words, a careful balance must be struck between the DRAM interface energy and the background energy.

Figure 20 shows the execution time results based on different fixed burst lengths. SWIM, OCEAN, CG, and GUPS are highly data-intensive benchmarks, which suffer from the increased burst length (BL). STRMATCH (String Match) actually runs slightly faster when the burst length is 14. The extra latency due to the contention on the data bus allows more requests to be queued up, which provides more ready commands that the memory scheduler can inspect when deriving the command schedule. On average, the execution times are increased by 3%, 6%, 6.5%, and 9.3% with BL10, BL12, BL14, and BL16, respectively. Therefore, always coding with a long burst length is not attractive. However, there are indeed opportunities to exploit long periods of idle cycles as shown in Figure 4. As a result, a hybrid scheme (with MiLC as the base scheme and 3-LWC as the opportunistic scheme) is a natural choice[10] to increase the energy savings without hurting performance.

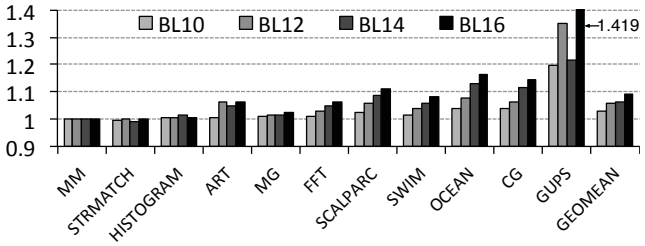

Figure 20: Sensitivity of execution time to burst length. The results are normalized to burst length=8.

7.5.2 *Look-ahead Distance*

In MiL, the decision logic (Section 4.2) checks whether there are any column commands that will become ready within the next X cycles, which is referred to as the "look-ahead distance" hereforth. Since MiL uses the 3-LWC as the optional coding scheme (which occupies the data bus for $2 \times 4 = 8$ cycles during each transaction), it is natural to set $X = 8$. This guarantees that no ready column command has to be postponed at the moment the current command is scheduled with 3-LWC.

Figure 21 illustrates how the system performs as the look-ahead distance is varied. The geometric mean of the execution times are all within 4% of one another for $X \geqslant 6$. Note that at $X = 14$ (rather than $X = 8$), the system performs the best, with a 2% performance degradation; this is because the simple decision logic cannot perfectly predict the future commands in the

[10]We eliminate the need for the DBI pin and reuse the burst chop pin, as discussed in Section 4.4.

next eight cycles at the time a command is scheduled. A more sophisticated decision logic is possible, and may achieve better results as discussed in Section 4.6.

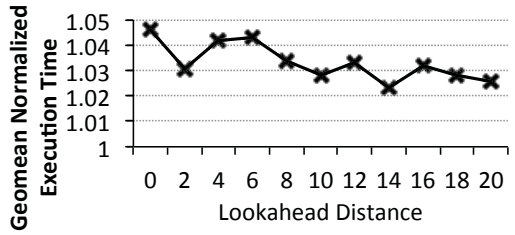

Figure 21: Impact of the look-ahead distance "X" on execution time.

7.5.3 MiLC vs 3-LWC: How Often are They Used?

We study how often MiLC and 3LWC are at runtime; the results are shown in Figure 22. The results indicate that the opportunity to apply long sparse codes decreases when the data bus utilization increases. The data-intensive benchmarks cannot fully benefit from the 3-LWC, which indicates that an intermediate sparse code with code length in between that of MILC and 3-LWC may improve the energy efficiency of data movement.

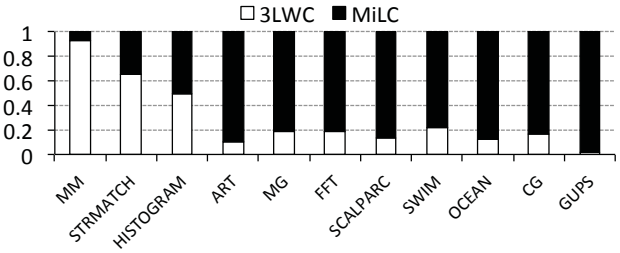

Figure 22: The fraction of time that MiLC and 3-LWC coding are chosen at runtime.

8. CONCLUSION

This paper presents MiL, a novel data communication framework built on top of the DDRx interface, which exploits the data bus under-utilization due to DRAM timing constraints. The experimental results show a 2× energy reduction in the DDR4 IO interface, and an 8% DDR4 system energy reduction with a less than 2% performance degradation. We conclude that MiL opens up a new dimension for energy optimization over the DDRx IO interface.

9. ACKNOWLEDGEMENTS

The authors would like to thank anonymous reviewers for useful feedback. This work was supported in part by NSF grant CCF-1217418.

10. REFERENCES

[1] B. Dally, "Power, programmability, and granularity: The challenges of exascale computing," in *Test Conference (ITC), 2011 IEEE International*, 2011.

[2] S. Keckler, W. Dally, B. Khailany, M. Garland, and D. Glasco, "Gpus and the future of parallel computing," *Micro, IEEE*, vol. 31, pp. 7–17, Sept 2011.

[3] "Samsung DDR4 Brochure." `http://www.samsung.com/global/business/semiconductor/file/media/DDR4_Brochure-0.pdf`.

[4] J. Feng, B. Dhavale, J. Chandrasekhar, Y. Tretiakov, and D. Oh, "System level signal and power integrity analysis for 3200mbps ddr4 interface," in *Electronic Components and Technology Conference (ECTC), 2013 IEEE 63rd*, pp. 1081–1086, IEEE, 2013.

[5] M. R. Stan and W. P. Burleson, "Limited-weight codes for low-power i/o," in *International Workshop on low power design*, 1994.

[6] M. R. Stan and W. P. Burleson, "Coding a terminated bus for low power," in *VLSI, 1995. Proceedings., Fifth Great Lakes Symposium on*, pp. 70–73, IEEE, 1995.

[7] "Flexible, Low Power Microservers for Lightweight Scale-Out Workloads." `http://www.intel.com/newsroom/kits/atom/c2000/pdfs/Intel_Microserver_Whitepaper.pdf`.

[8] "Qualcomm Snapdragon 808." `http://www.qualcomm.com/products/snapdragon/processors/808`.

[9] K. Sohn, T. Na, I. Song, Y. Shim, W. Bae, S. Kang, D. Lee, H. Jung, S. Hyun, H. Jeoung, K.-W. Lee, J.-S. Park, J. Lee, B. Lee, I. Jun, J. Park, J. Park, H. Choi, S. Kim, H. Chung, Y. Choi, D.-H. Jung, B. Kim, J.-H. Choi, S.-J. Jang, C.-W. Kim, J.-B. Lee, and J. S. Choi, "A 1.2 v 30 nm 3.2 gb/s/pin 4 gb ddr4 sdram with dual-error detection and pvt-tolerant data-fetch scheme," *Solid-State Circuits, IEEE Journal of*, vol. 48, pp. 168–177, Jan 2013.

[10] "MICRON LPDDR3 Power Calculator." `http://www.micron.com/`.

[11] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *TVLSI*, vol. 3, no. 1, pp. 49–58, 1995.

[12] M. Stan and W. Burleson, "Low-power encodings for global communication in cmos vlsi," *TVLSI*, vol. 5, no. 4, pp. 444–455, 1997.

[13] M. Anders, N. Rai, R. K. Krishnamurthy, and S. Borkar, "A transition-encoded dynamic bus technique for high-performance interconnects," *JSSC*, vol. 38, no. 5, pp. 709–714, 2003.

[14] M. R. Stan and Y. Zhang, "Perfect 3-limited-weight code for low power i/o," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, pp. 79–89, Springer, 2004.

[15] S. Pasricha and N. Dutt, *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann Publishers Inc., 2008.

[16] W.-C. Cheng and M. Pedram, "Memory bus encoding for low power: a tutorial," in *Quality Electronic Design, International Symposium on*, pp. 199–199, IEEE Computer Society, 2001.

[17] S. Komatsu, M. Ikeda, and K. Asada, "Low power chip interface based on bus data encoding with adaptive code-book method," in *GLSVLSI*, 1999.

[18] J. Yang, R. Gupta, and C. Zhang, "Frequent value encoding for low power data buses," *TODAES*, vol. 9, no. 3, pp. 354–384, 2004.

[19] D. Suresh, B. Agrawal, J. Yang, and W. Najjar, "Tunable and energy efficient bus encoding techniques," *TC*, vol. 58, no. 8, pp. 1049–1062, 2009.

[20] B. R. Childers and T. Nakra, "Reordering memory bus transactions for reduced power consumption," in *LCTES*, 2000.

[21] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *International Symposium on Microarchitecture*, (New York, NY), Dec 2009.

[22] R. Maddah, S. M. Seyedzadeh, and R. Melhem, "Cafo: Cost aware flip optimization for asymmetric memories," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 320–330, IEEE, 2015.

[23] D. Skinner, "Lpddr4 moves mobile," in *In JEDEC Mobile Forum Conference*, 2013.

[24] J.-S. Kim, C. S. Oh, H. Lee, D. Lee, H. R. Hwang, S. Hwang, B. Na, J. Moon, J.-G. Kim, H. Park, J.-W. Ryu, K. Park, S. K. Kang, S.-Y. Kim, H. Kim, J.-M. Bang, H. Cho, M. Jang, C. Han, J.-B. Lee, J. S. Choi, and Y.-H. Jun, "A 1.2 v 12.8 gb/s 2 gb mobile wide-i/o dram with 4x128 i/os using tsv based stacking," *Solid-State Circuits, IEEE Journal of*, vol. 47, pp. 107–116, Jan 2012.

[25] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong, "25.2 a 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 432–433, Feb 2014.

[26] J. Jeddeloh and B. Keeth, "Hybrid Memory Cube new DRAM architecture increases density and performance," in *VLSI Technology (VLSIT), 2012 Symposium on*.

[27] T. Pawlowski, "Hybrid Memory Cube (HMC)." HotChips, 2011.

[28] Z. Li, R. Zhou, and T. Li, "Exploring high-performance and energy proportional interface for phase change memory systems," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 210–221, IEEE, 2013.

[29] S. Saini, J. Chang, and H. Jin, "Performance evaluation of the intel sandy bridge based nasa pleiades using scientific and engineering applications," in *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, pp. 25–51, Springer, 2014.

[30] E. Ipek, O. Mutlu, J. Martinez, and R. Caruana, "Self-optimizing memory controllers : A reinforcement learning approach," in *International Symposium on Computer Architecture*, (Beijing, China), Jun 2008.

[31] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile dram," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, (Washington, DC, USA), pp. 37–48, IEEE Computer Society, 2012.

[32] B. Jacob, "The memory system: you can't avoid it, you can't ignore it, you can't fake it," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–77, 2009.

[33] E. Ipek, *Reconfigurable and Self-optimizing Multicore Architectures*. PhD thesis, Cornell University, 2008.

[34] E. Saule, K. Kaya, and Ü. V. Çatalyürek, "Performance evaluation of sparse matrix multiplication kernels on intel xeon phi," in *Parallel Processing and Applied Mathematics*, pp. 559–570, Springer, 2014.

[35] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM international conference on Autonomic computing*, pp. 31–40, ACM, 2011.

[36] "DDR4 bank groups in embedded applications." `http://www.synopsys.com/Company/Publications/DWTB/Pages/dwtb-ddr4-bank-groups-2013Q2.aspx`.

[37] Micron, *4Gb DDR4 SDRAM Data Sheet: EDY4016A*.

[38] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. H. Pugsley, A. N. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "Usimm: the utah simulated memory module a simulation infrastructure for the jwac memory scheduling championship," 2012.

[39] B. L. Jacob, S. W. Ng, D. T. Wang, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008.

[40] "Qualcomm Snapdragon 805." `http://www.anandtech.com/show/8035/qualcomm-snapdragon-805-performance-preview`.

[41] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded sparc processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, 2005.

[42] S. Rixner *et al.*, "Memory access scheduling," in *ISCA*, 2000.

[43] Micron, *16Gb 216-Ball, Dual-Channel Mobile LPDDR3 SDRAM Data Sheet: EDFA164A*.

[44] "Cadence NCSim." `http://www.cadence.com/products/fv/enterprise_simulator/pages/default.aspx`.

[45] "Synopsys Design Compiler." `http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/default.aspx`.

[46] "Free PDK 45nm open-access based PDK for the 45nm technology node." `http://www.eda.ncsu.edu/wiki/FreePDK`.

[47] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. Brockman, and N. Jouppi, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pp. 51–62, June 2008.

[48] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ISCA*, 2011.

[49] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "Fabscalar: composing synthesizable rtl designs of arbitrary cores within a canonical superscalar template," in *Proceeding of the 38th annual international symposium on Computer architecture*, ISCA '11, (New York, NY, USA), pp. 11–22, ACM, 2011.

[50] J. Renau *et al.*, "SESC simulator," Jan. 2005. `http://sesc.sourceforge.net`.

[51] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pp. 63–74, IEEE, 2007.

[52] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *ISCA*, 2009.

[53] "MICRON DDR4 Power Calculator." `http://www.micron.com/~/media/documents/products/power-calculator/ddr4_power_calc.xlsm`.

[54] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, "Introduction to the hpc challenge benchmark suite," *Lawrence Berkeley National Laboratory*, 2005.

[55] D. H. Bailey *et al.*, "NAS parallel benchmarks," tech. rep., NASA Ames Research Center, March 1994. Tech. Rep. RNR-94-007.

[56] J. Pisharath, Y. Liu, W. Liao, A. Choudhary, G. Memik, and J. Parhi, "NU-MineBench 2.0," tech. rep., Northwestern University, August 2005. Tech. Rep. CUCIS-2005-08-01.

[57] R. M. Yoo, A. Romano, and C. Kozyrakis, "Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system," in *IISWC*, 2009.

[58] L. Dagum and R. Menon, "OpenMP: An industry-standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, pp. 46–55, Jan.–Mar. 1998.

[59] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *ISCA-22*, 1995.

[60] K. Malladi, I. Shaeffer, L. Gopalakrishnan, D. Lo, B. Lee, and M. Horowitz, "Rethinking dram power modes for energy proportionality," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pp. 131–142, Dec 2012.