Reducing Data Transfer Energy by Exploiting Similarity within a Data Transaction

Donghyuk Lee*† Mike O'Connor*†‡ Niladrish Chatterjee†

†*NVIDIA* ‡*University of Texas at Austin*

***Abstract*—Modern highly parallel GPU systems require high-bandwidth DRAM I/O interfaces that can consume a significant amount of energy. This energy increases in proportion to the number of `1` values in the data transactions due to the asymmetric energy consumption of Pseudo Open Drain (POD) I/O interface in contemporary Graphics DDR SDRAMs.**

In this work, we describe a technique to save energy by reducing the energy-expensive `1` values in the DRAM interface. We observe that multiple data elements within a single cache line/sector are often similar to one another. We exploit this characteristic to encode each transfer to the DRAM such that there is one reference copy of the data, with remaining similar data items being encoded predominantly as `0` values. Our proposed low energy data transfer mechanism, *Base + XOR Transfer*, encodes the data-similar portion by performing *XOR* operations between data elements within a single DRAM transaction. We address two challenges that influence the efficiency of our mechanism, *i)* the frequent appearance of zero data elements in transactions, and *ii)* the diversity in the underlying size of data types within a transaction. We describe two techniques, *Zero Data Remapping* and *Universal Base + XOR Transfer*, to efficiently address these issues. Our proposed encoding scheme requires *no* additional metadata or changes to existing DRAM devices.

We evaluate our mechanism on a modern high performance GPU system with a variety of graphics and compute workloads. We show that our mechanism reduces energy-expensive `1` values by 35.3 % with minimal overheads, and combining our mechanism with Dynamic Bus Inversion (DBI) reduces `1` values by 48.2 % on average. These `1` value reductions lead to 5.8 % and 7.1 % DRAM energy savings, respectively.

I. Introduction

Increasing DRAM bandwidth has been a key challenge to enable higher throughput in GPU systems. To meet this high bandwidth demand, Graphics DDR (GDDR) DRAMs use a very high speed I/O interface. The most recent generation of Graphics DDR DRAM, GDDR5X, offers a road-map up to 14 Gbps per pin bandwidth [1, 2], which is 1.75× the bandwidth of the previous GDDR5 generation. This trend towards increasing bandwidth, however, has not been matched by reductions of a similar magnitude in the energy required for each access. Figure 1 shows that while bandwidth has increased 2× from GDDR5 to the latest GDDR5X, the energy per access has only been reduced 19 %. As a result, total power at peak bandwidth has increased by 63 %. This growing DRAM power demand is an increasingly important issue facing these high-bandwidth systems.

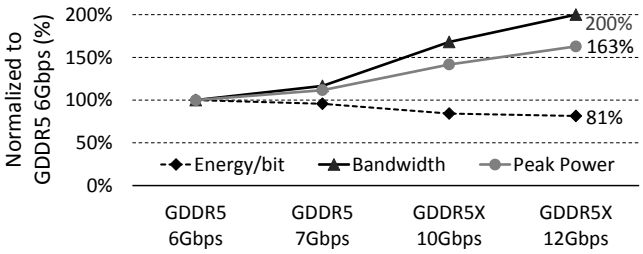


Figure 1: Hypothetical GPU memory system trend

A large portion of DRAM energy is dissipated in the I/O channel (Section V-A and [3, 4]). To enable reliable high-speed operation, contemporary DRAM adopts termination-based I/O schemes. For example, GDDR5/GDDR5X uses a *Pseudo Open Drain (POD)* I/O interface that introduces a termination resistor between the I/O channel and the V_{DD} voltage source. This termination resistor creates a static current flow when driving the channel to $0\,V$, which dissipates significant fraction of the I/O energy. Due to the asymmetric termination scheme, transferring one value (*e.g.,* a `1`) consumes more energy than transferring the other value (*e.g.,* a `0`).[1] GDDR5/GDDR5X leverages this asymmetric characteristic by adopting a data encoding technique, Dynamic Bus Inversion (DBI) [5], which conditionally transfers the inverted form of data if it is expected to reduce data transfer energy.

Our goal in this work is to further reduce the energy consumption on terminated DRAM I/O interfaces by reducing the number of energy-expensive `1` values. We leverage the facts that *i)* multiple data elements are simultaneously transmitted in a single transaction together over a DRAM channel and *ii)* there often exists a large amount of similarity among these elements. Similar to observations leading to some compression schemes (*e.g.,* [6, 7]), this data similarity among adjacent data elements allows the same information to be encoded into a more energy efficient form, such as one with reduced switching activity or fewer `1` values. SILENT [8] leverages this data similarity among adjacent data words to reduce switching activity for better energy efficiency in a *serial* interface within an SoC. The key idea is transferring the bulk of the data words as the bitwise difference from their adjacent words. To take advantage of the intra-transaction similarity among the data elements,

* Both authors contributed equally to the paper.

[1] In this work, we use the common convention that a `1` value is transferred as $0\,V$ and a `0` value is transferred as V_{DD}. Therefore, transferring a `1` is more energy expensive than a `0`.

we build upon the basic approach described in SILENT to reducing the energy consumption on the *parallel* I/O interface on conventional DRAM channels.

Like SILENT, our base data encoding mechanism, *Base + XOR Transfer*, encodes the similar data elements by performing a simple XOR operation of values within a transaction. An unmodified data element is called the *base element* and the remaining elements are encoded as *XORed* values with the base or the adjacent element. We observe two issues that adversely impact the effectiveness of this base version of our mechanism, and we provide ways to optimize the encoding to address these issues. First, there are often a large number of zero data elements (*e.g.,* a 4-byte zero data element `00000000h`) interspersed among the data that can introduce *more* `1` values in the encoded result. This potentially increases the overall data transfer energy. To avoid this, we special-case zero data elements separately by *remapping* encoded zero values to a *low-weight constant* that includes only one `1` bit (minimizing the additional increase in `1` values). We also remap the value that would be encoded to this low-weight constant to the encoded value that a zero data element would have had. Swapping these encoded symbols requires us to add no additional metadata. This *Zero Data Remapping* is described in Section IV-A.

Second, the efficiency of our mechanism highly depends on properly determining the size of the base element (*base size*) for the XOR operation. Since there are many data structures that have different sizes, the size of repeated similar data in a transaction could vary. If the base size for XOR operations does not match to the size of the repeated similar data, our mechanism either fails to realize the maximum potential benefit or, alternatively, can *increase* the number of `1` values. A naive mechanism that exhaustively performs XOR operations with all possible base sizes might find the proper base size that maximizes the benefit. This approach, however, induces significant overheads in energy consumption, latency, and logic complexity. Furthermore, metadata to communicate the selected granularity would be required. Instead, we propose an *Universal Base* approach that efficiently extracts the similar data portion without *a priori* knowledge of the underlying data structure size. We explain the details of this mechanism in Section IV-C.

We evaluate our mechanism on a high performance GPU system executing a variety of graphics and compute workloads. Our evaluation shows that our optimized Base + XOR Transfer encoding significantly reduces `1` values on average by 35.3 % with a corresponding 5.8 % memory system energy reduction, compared to the conventional data transfer scheme. We achieve this at low cost, without any metadata, and without any noticeable performance degradation. We also consider the existing `1` value reduction techniques like Dynamic Bus Inversion (DBI), and we show that our techniques can reduce energy more than DBI by itself. However, since DBI does provide guarantees on the maximum number of `1` values and can be used to limit power-supply noise, our scheme may not replace DBI. Applying Base + XOR Transfer encoding in addition to DBI results in a cumulative benefit, providing a 48.2 % reduction in `1` values and a corresponding 7.1 % memory system energy reduction.

Our contributions of this work follow.

- We propose a new low power data transfer mechanism, Base + XOR Transfer, which reduces terminated POD I/O interface energy. Since the major operation is simple *XOR* and our mechanism does *not* require any *metadata*, systems can easily integrate our mechanism using existing DRAMs and at low cost.
- We only leverage the intra-transaction similarity (within a sector/a cacheline), the encoded data can be stored at any data repository (*e.g.,* cache or DRAM cells).
- We evaluate our mechanism on data from a high performance GPU system and show that Base + XOR Transfer can significantly reduce `1` values (by 35.3 %). Coupled with DBI, `1` values can be reduced by 48.2 %. DRAM energy is reduced 5.8 % and 7.1 %, respectively.

II. Background

A. Pseudo Open Drain (POD) I/O Interface

In the GDDR DRAM interface, data is represented as two different voltage levels (*e.g.*, $0\,V$ or V_{DD}). Figure 2 shows the organization of a DRAM interface, driving data from the left output driver to the right input buffer, which are connected over a wire that has a parasitic capacitance (C_C). To improve channel integrity operating at high frequency, DRAM integrates termination resistors on its channel. A termination scheme, *Pseudo Open Drain* (POD), puts a termination resistor (R_T) between the wire and V_{DD}. Based on our analysis (more details in Section V-A) and other prior work [3, 4], the energy consumption on a POD I/O interface is significant.

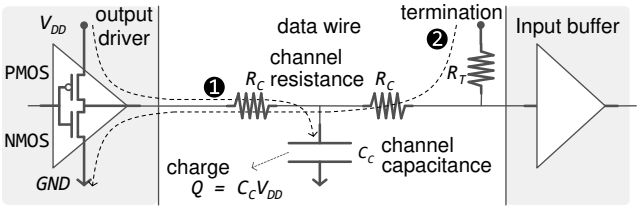


Figure 2: Energy consumption for moving data over wire

There are two major sources of energy consumption. The first one is the parasitic capacitance (C_C). To transfer a `0` value (represented as a V_{DD}) over a wire that was initially $0\,V$, the PMOS transistor in the driver is turned on, charging C_C by driving charge through ❶. Then, to transfer a `1` value (represented as a $0\,V$), the NMOS transistor is turned on, discharging the charge in C_C. From these two steps of charging and discharging the parasitic capacitance of the wire for transferring a sequence of data, $1 \rightarrow 0 \rightarrow 1$, the

amount of charge Q flows from the V_{DD} power source to the ground, consuming energy.[2]

The second source of energy consumption is the termination resistor (R_T). Transferring a `1` value over the wire forms a current path from V_{DD} to ground over R_T (as shown in current path ❷). This energy consumption depends on how often bits with a `1` value are transferred. As we discuss more detail in Section V-A, transferring a `1` value in GDDR5X (10 Gbps per-pin data rate) dissipates 13.5 mA additional current during the data period (100 ps). Due to this additional current flow through R_T, transferring a `1` value consumes 37 % more energy than transferring a `0` value.

B. Dynamic Bus Inversion (DBI)

To reduce the power consumption on the POD I/O interface, GDDR5/5X supports Dynamic Bus Inversion (DBI) [5], which conditionally inverts the data to be transferred so as to minimize the number of `1` values. When transferring an n-bit element that has more than $n/2$ `1` bits, inverting the data and transferring this inverted value ensures that no more then $n/2$ bits in the encoded result are `1`. The polarity information, indicating whether the current element is inverted or not, is transferred as an additional bit of metadata over an additional wire. Therefore, an n-bit data element can be transferred as an $(n+1)$-bit encoded result in which no more than $n/2$ bits are a `1`. The element size, n, is a major design parameter for DBI. Smaller element sizes enable simpler implementations (an n-bit pop-count is effectively required to determine whether inversion is necessary) at the expense of a larger metadata overhead. GDDR5/GDDR5X applies DBI over each 8 bits of data on the I/O interface, thus requiring one additional wire for every 8 data signals.

III. Reducing Data Transfer Energy by Exploiting Data Similarity

Our goal in this work is to improve the energy efficiency in the DRAM interface by reducing the number of `1` values that must be sent across the interface. We observe that often a transaction sent across the DRAM interfaces consists of several similar data elements. By exploiting this intra-transaction data similarity, we propose an encoding mechanism that can reduce the number of `1` values. We describe some challenges that limit the effectiveness of this approach, and propose optimizations to address these issues.

A. Data Similarity within a Transaction

Within a GPU, a typical single DRAM transaction is a 32-byte cache sector. This transaction will be sent across a 32-bit interface to a GDDR5 DRAM over eight beats. Typically, this 32-byte transaction will consist of several adjacent data elements of a given size. For instance, it might consist of four 64-bit double-precision floating-point values, eight 32-bit integers, or sixteen 16-bit floating-point values. The SIMD nature of GPU execution units favors a programming style that uses *structures of arrays* (SoA), rather than arrays of structures commonly found in scalar CPU code [9]. As a result, data fetched is very commonly from an array of fixed-sized elements of a given data type.

Many prior works [6, 7, 10, 11] have observed significant similarities in adjacent data elements stored in on-chip caches and off-chip memories. Due to the similarity among data elements within a transaction (*intra-transaction similarity*), identical or highly similar data could be transferred multiple times, repetitively sending energy-expensive `1` values, and increasing overall energy consumption in the I/O interface. In Figure 3, we plot two transactions of 32-byte data over the DRAM I/O interface in a GPU system. Examining the eight beats of `transaction 0`, the top 16-bit chunks of all elements are `0x390c`. Repetitively transferring the top 16-bit chunks of each element leads to the six `1` bits being sent seven additional times. A simple encoding scheme that prevents this redundant transfer of `1` bits is desired.

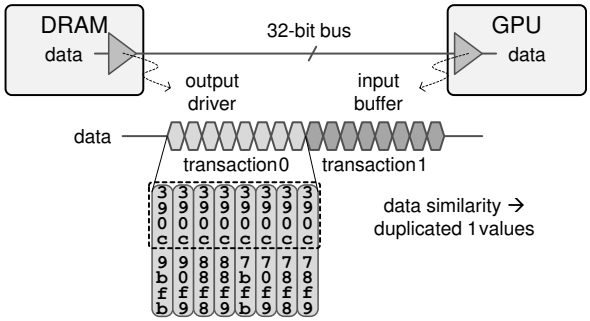


Figure 3: Data transfer between DRAM and GPU: Data similarity leads to duplicated data

B. Base + XOR Transfer

We exploit this data similarity within DRAM transactions to reduce the number of `1` values in each DRAM transfer. We leverage an earlier approach, SILENT [8], that encodes data to reduce switching activity in a serial I/O interface, and we modify it for a conventional DRAM channel. The key idea is to simply transfer the bitwise difference (`XOR`) from each adjacent data element rather than the element itself. If the data is highly similar, this leads to fewer `1` values. Figure 4 shows the basic idea behind our mechanism (illustrated on a 16-byte transaction for simplicity). The transaction is divided into four 4-byte (32-bit) elements, labeled `element 0` to `element 3` from the left to the right. The key operation follows.

- The left most 4-byte element (`element 0`) is transferred with no change. We call this the *base element*. We call the size of the base element the *base size*.

[2] The parasitic resistance also consumes a small amount of energy that is dissipated as thermal energy, increasing the temperature of the wire.

- The data in the second element (`element 1`) is transferred as a bitwise difference (XOR) with respect to `element 0`. As a result, bits in `element 1`, which have the same values as the corresponding bits in `element 0`, are set to `0` value that is less energy expensive. The other elements (`element 2` and `element 3`) are also encoded as a difference from their left adjacent elements. We call these three elements *XORed elements*.
- The original values of the XORed elements can be decoded by simply performing the same XOR operations in the reverse sequence with the XORed elements and their left adjacent elements.

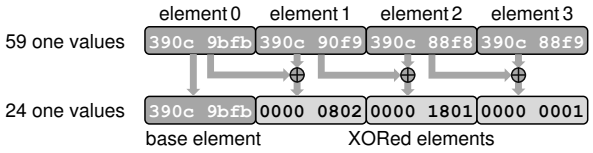


Figure 4: Base + XOR Transfer

Due to the data similarity within the transaction, performing XOR operations between adjacent elements in a transaction tends to introduce many `0` values (which consume less energy than `1` values) in the XORed elements. As a result, our mechanism reduces the number of `1` values from 59 to 24 in this example. Note that we divide a transaction into multiple elements by using a fixed base size and perform XOR operations with these base-sized elements. We call this basic mechanism when using an *N*-byte base: *N-byte Base + XOR Transfer*.

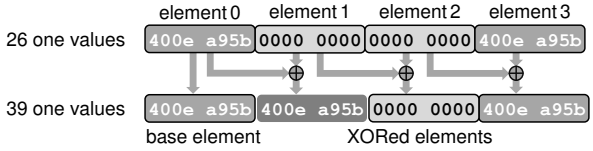
IV. Efficiently Extracting Data Similarity

The intra-transaction data similarities our scheme relies on vary based on the application and the data set. Using data collected from a large number of applications, we found two key challenges that can negatively impact the efficacy of our mechanism. First, there are many zero valued data elements within transactions, often interspersed among other data. This prevalence of zero data elements has been observed in many prior works [11, 12, 13]. These zero valued data elements are often poorly handled in our baseline scheme. In Section IV-A, we describe why the problem occurs and we detail our approach to better handle zero data elements.

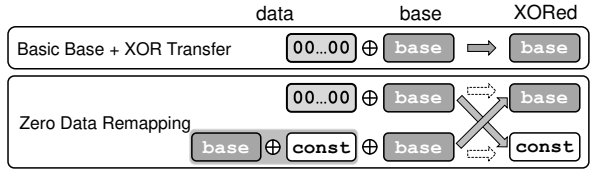
Second, since the elements within a transaction can be of different sizes, the data similarity can manifest at different granularities. Therefore, determining the most appropriate base size is critical for the XOR encoding. In Sections IV-B and , we discuss the impact of the base size selection for the XOR encoding, and introduce our low-overhead approach that naturally exploits data similarity over the range of base sizes.

A. Zero Data Remapping (ZDR)

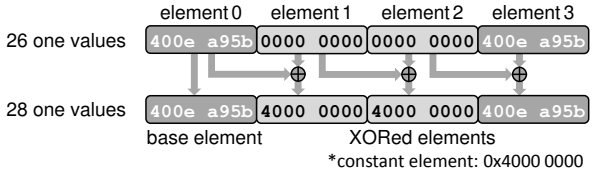
We observe many zero data elements in data traces of applications. Figure 5a illustrates a case in which applying our mechanism makes things worse on a transaction that includes zero data elements. Rather than keeping a zero data element, another copy of a non-zero value to the left is created during the encoding process (see the XORed result of `element 1` in Figure 5a). This increases the total number of `1` bits we must transmit, diminishing the benefits from the encoding scheme.



(a) Zero data interferes with extracting data similarity



(b) Key Idea: Swapping the encoding results of two special cases



(c) The effect of Zero Data Remapping

Figure 5: Base + XOR Transfer with Zero Data Remapping

To avoid the interference from zero valued data elements, we propose a refined encoding that *remaps* the encoded result of a zero data element to a *low-weight constant*. In this mechanism, we must ensure that other data is *not* encoded to this low-weight constant. Figure 5b shows the basic operations. During encoding, the input data element is checked if it is equal to the base element XORed with the low-weight constant. In this case, the XOR-encoded result would have ended up being the low-weight constant. Instead, we make the encoded result equal to the base value. By swapping the encoded results of these two inputs (zero data element and the base element $\oplus$ the constant), we make the common case of zero data elements cheap, while making an uncommon case more expensive. We call this *Zero Data Remapping*.

Any constant value can potentially be used for this mechanism. With the goal of reducing `1` values in the XORed element as much as possible, there are two major requirements for the constant. First, it should be a *low-weight* value with a small number of `1` bits. This is necessary to make the incremental cost of replacing zero data elements have low overhead. Second, data elements that are equal to the base element $\oplus$ the constant should be rare. A constant like `00000000h`, for instance, would preserve `0` values but would eliminate any benefit in our scheme for repeated non-

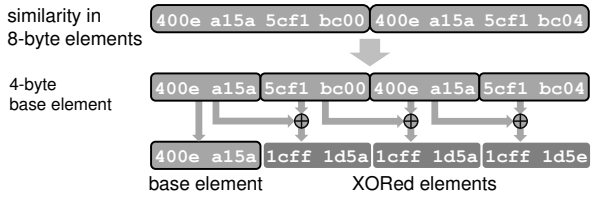
zero data elements (another common case). Similarly, small constants like `00000001h` or `00000004h` would make data elements with small power-of-two offsets susceptible to sub-optimal encodings. We find that a constant like `40000000h` works well.

As shown in Figure 5c, XOR encoding with Zero Data Remapping only introduces two '1' bits in the results, increasing `1` values from 26 to 28. Considering that original XOR encoding increases `1` values from 26 to 39 in Figure 5a, Zero Data Remapping successfully mitigates the interference from the zero valued data element.

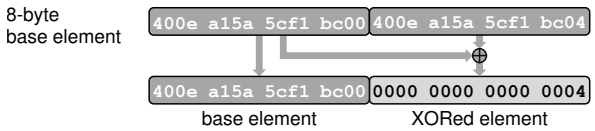
B. Proper Base Size Selection

Matching the base size of the encoding scheme to the underlying size of the data elements in a transaction can have a significant effect on the effectiveness of the encoding scheme. If a base size is not matched to the underlying data size, two different effects can occur, which reduce the effectiveness of the encoding.

In particular, if we choose a *smaller* base size than the underlying data elements, our XOR encoding does not find the data similarity, possibly consuming more energy. We show this effect in Figure 6a. In this example, two 8-byte similar elements form a 16-byte transaction that are XOR-encoded with a 4-byte base size. As we see, our mechanism could not extract the data similarity, failing to introduce any `0` values in the XORed elements. Selecting a proper base size that is aligned to the underlying data elements (8 bytes in the example in Figure 6b) successfully introduces the mostly `0` values in the XORed elements.



(a) Smaller Base (4 bytes) than Similar Data Chunks (8 bytes)



(b) The Same Size Base as Similar Data Chunks (8 bytes)

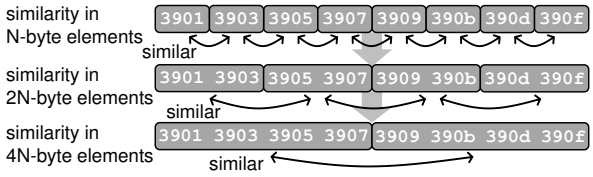
Figure 6: Base + XOR Transfer with different base size

If we choose a *larger* base size than the underlying data elements, our mechanism misses opportunities to introduce more `0` values. Figure 4 shows `Transaction 0` with repeated data patterns every 4 bytes. When using an 8-byte base size, we can still find significant similarity between the two 8-byte regions, but we lose the opportunity to exploit the similarity between the upper and lower 4 bytes of the 8-byte base. This loss in effectiveness is smaller than the consequences of choosing a base size that is too small, but it is a significant effect.

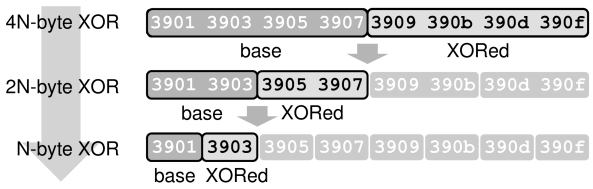
There are several possible mechanisms that could identify the best base size. First, the most intuitive solution is to encode the data with multiple base sizes and select the results with the fewest `1` bits. A second potential solution is periodically profiling a per-page preferred base size and using the profiled information to encode transactions in the page. A third solution is selecting the proper base size for the compute units that use a predefined data structure, for example, floating-point units. Unfortunately, all these potential base selection mechanisms require non-trivial overhead (*e.g.,* similarity check units and storage for base size information) or are applicable only to the interconnect that always transfers predefined data structures.

C. Universal Base + XOR Transfer

Instead of explicitly figuring out the proper base size, we introduce a simple mechanism that is broadly applicable to any base size, thereby not requiring any *a priori* information of the base size. Our mechanism exploits the fact that *in a transaction, if every N-byte element is similar to each other, then 2N-byte elements are also similar to each other.* As Figure 7a shows, all 2-byte elements in a transaction are similar to each other (in the first row). If two 2-byte elements form one 4-byte element (shown in the second row), all 4-byte elements are also similar to each other. We leverage this insight to build our mechanism.



(a) **Key Observation:** Similarity in 2N-byte element → Similarity in 4N-byte element



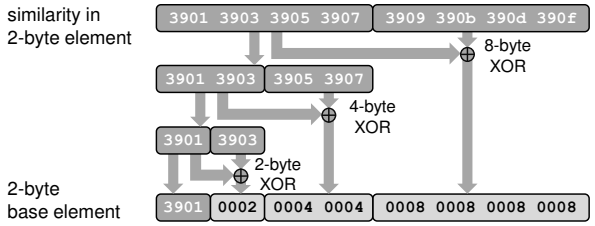
(b) **Key Idea:** XOR Encoding for the base of previous stage

Figure 7: Universal Base + XOR Transfer

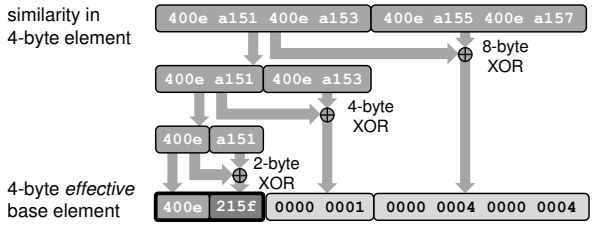
Our encoding mechanism consists of multiple steps that can operate in parallel. Each step takes an N-byte data and performs the XOR encoding by using half of the data as the base (*i.e.*, a $N/2$ base size), generating a $N/2$-byte base element and a $N/2$-byte XORed element. If there exists a repeated pattern in two $N/2$-byte elements of the input data, the $N/2$-byte XORed element turns out to have mostly `0` values. The same process is applied to the $N/2$-byte base element of the previous step. Our mechanism applies this XOR encoding for each base size down to the smallest base size output (*e.g.,* 2 bytes).

Figure 7b shows how our mechanism works. In the first step, the 16-byte transaction is divided into two 8-byte elements. Using these two elements, our mechanism performs the XOR encoding, which will generate 8 bytes of the final encoded result. In the next step (the middle row in Figure 7b), the base structure from the previous step (the left 8-byte data) is divided into two 4-byte elements. Using these two elements, our mechanism performs the XOR encoding, producing 4 bytes of the final XOR-encoded result. Our mechanism continues, performing the same operations, until generating the left most 2-byte data turns to be base element.

In Figure 8, we demonstrate the encoding process for two example transactions. Applying our mechanism to a transaction that consists of eight similar 2-byte elements (Figure 8a) successfully generates a 2-byte base element and 14 bytes of mostly `0` XOR-encoded elements. Applying our mechanism to a transaction that consists of four similar 4-byte elements (Figure 8b) leads to one 4-byte data result (`0x400e 215f`) and a 12 bytes of mostly `0` XOR-encoded elements, which is similar to the result that we would achieve by performing the 4-byte XOR encoding with the knowledge of the most suitable base size. Therefore, we call this 4-byte data the *effective base element*. Thus, since our mechanism automatically generates a reasonable encoding for any effective base element, we call this mechanism *Universal Base + XOR Transfer*.



(a) 2-byte similar element case



(b) 4-byte similar element case

Figure 8: Case study: Universal Base + XOR Transfer

Using Base + XOR Transfer coupled with the two optimizations, we have developed a simple encoding scheme that can significantly reduce the number of `1` values in the resulting data. This scheme requires no *a priori* knowledge of the data and requires no metadata to be communicated with the encoded data.

V. System Configuration and Implementation

Our mechanism is generally applicable to any compute system that transfers data over a POD I/O interface. We evaluate the benefits of our mechanisms on a highly parallel GPU system that integrates a large number of compute units.

A. Evaluated System Configuration

We model an NVIDIA Titan X graphics card based on NVIDIA's Pascal architecture [14]. This GPU system integrates many channels of GDDR5X which utilize a POD I/O interface. The detailed system configuration is provided in Table I.

Component	Parameters
Compute Units	56 stream multi processors
Last-Level Cache	4 MB in total
Memory System	384 bit total bus, 12 GB GDDR5X 480 GBps total channel bandwidth 4 32-byte sectors per cacheline
GDDR5X	Data Rate (per pin): 10 Gbps Power Supply: V_{DD}/V_{DDQ} = 1.35 V Output Driver: R_{PullUp}/R_{PullDn} = 60/40 Ω Termination: R_T = 60 Ω

Table I: Configuration of evaluated GPU system

In this system, transferring a `1` value drives the corresponding wire to $0\,V$, which creates a static current path over the termination resistor and the pull down transistor. As shown in Table I, since GDDR5X uses a 60 Ω termination resistor (R_T) and a pull-down transistor that has 40 Ω of turn-on resistance, the static current for a `1` value is about 13.5 mA ($I = V/R = 1.35V/100\Omega$). Due to this additional current for a `1` value, there is a significant energy imbalance between transferring `1` values and `0` values. Considering that a GPU system uses very wide bus in DRAM channel, the additional energy for `1` values is significant (*e.g.,* 432 mA peak current for 32-bit bus in a GDDR5X chip or 5.2 A peak current for 384-bit bus in the GPU memory system).

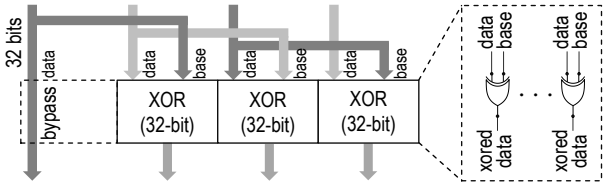
We estimate the overall memory system energy consumption considering this data dependent energy consumption. For the overall memory system energy estimation, we use modified versions of the Micron DRAM Power Calculator [15] and Rambus DRAM Power Calculator [16]. We report the energy impact of our mechanisms in Section VI-F.

B. Implementation Costs

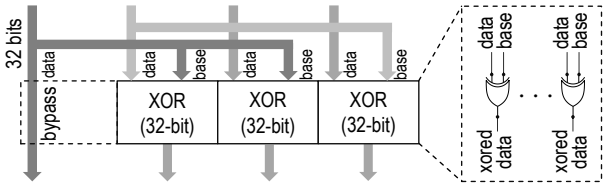
We designed detailed gate-level implementations of our proposals and analyzed the additional area, latency, and energy consumption of the encode and decode logic.

N-byte Base + XOR Transfer. Figure 9a shows an implementation of the 4-byte Base + XOR Transfer *encoding* scheme that is applied to 16-byte transactions. As shown in the right of the figure, the *only* required change is adding 96 XOR gates (one XOR gate per wire to generate the XORed elements). Two inputs of the XOR gates are connected to two wires, one from the original element and the other from the left 4-byte element. The *decoding* process is simply performing the same XOR operation between adjacent 4-byte elements. The only difference in the decoding scheme

from the encoding scheme is that an XORed element (*e.g.,* the third element) should be decoded with the decoded values of its base element (*e.g.,* the second element). As a result, a decoding takes longer time than a encoding. Alternatively, it is also possible to use a fixed base (*e.g.,* the first element) for all XOR operations. While this fixed base scheme is simpler and has lower latency than our proposal, we observe that using the adjacent element as a base element shows better energy reduction, due to higher similarity in adjacent elements. *N*-byte Base + XOR Transfer can be easily implemented similarly to the 4-byte Base + XOR Transfer example.



(a) 4-byte Base + XOR Transfer (16-byte)



(b) Universal Base + XOR Transfer (16-byte)

Figure 9: Implementation details of Base + XOR Transfer

Universal Base + XOR Transfer. While we explained Universal Base + XOR Transfer as multiple encoding steps in Section IV-B, those steps operate in parallel. Figure 9b shows an implementation of Universal Base + XOR Transfer that consists of two stages and is applied to 16-byte transactions. As the figure shows, the only implementation difference from N-byte Base + XOR Transfer is organizing the base elements in an asymmetric way. Due to the fact that the the left-most values are used as the base element for several of these operations, elements that are closer to the left end fan out to more XOR operations, causing some additional area due to additional routing. This implementation can be easily extended to the cases of a different transaction size (*e.g.,* 32 bytes) and a different number of stages (*e.g.,* 3 stages).

Zero Data Remapping. The key operation of this mechanism is swapping the encoded results of the two special inputs, *i)* zero data element and *ii)* the base element $\oplus$ the light-weight constant. Figure 10 shows a possible Verilog pseudocode of our mechanism applied to a 32-bit element. Our mechanism first determines if the input is a zero data element by simply performing `OR` operations with all bits in the input. If all bits are zero, the output is the constant. We, then, determine if the input is the same as the result of the base $\oplus$ the constant. To this end, we leverage the fact that an XOR operation of two the same values results in a `0` value. Therefore, if the input is the same as the result of the base $\oplus$ the constant, simply performing XOR operations with the two data leads to all zero values, which can be detected by a simple tree of `OR` operations. If so, the output is the base element. If the input is not the two special values, the output is the result of the base $\oplus$ the input.

```
in[31:0], base[31:0], const[31:0] = 40000000h

if in[31:0] == 0 then
   out[31:0] ⇐ const[31:0]
else if in[31:0] ⊕ (base[31:0] ⊕ const[31:0]) == 0 then
   out[31:0] ⇐ base[31:0]
else
   out[31:0] ⇐ in[31:0] ⊕ base[31:0]
end if
```

Figure 10: Zero Data Remapping

System Organization. The encoding/decoding logic is contained entirely within the memory controller on the GPU and our schemes do not require any modifications to the DRAM devices. The data is encoded by the memory controller before being written to DRAM, stored in encoded form in the DRAM, and is decoded after it has been brought into the memory controller on a read.

Table II summarizes the area, latency, and energy consumption of our mechanisms by using TSMC 16 nm FinFET process parameters from [17, 18]. We estimate the worst case of the energy consumption and encoding/decoding latency by considering all the possible inputs. The area estimation includes both additional logic and wires. We observe that the area overhead is very small (*e.g.,* 2,232 μm^2 for both encoding and decoding logic in the most sophisticated mechanism). For the evaluated GPU system, with twelve 32-bit DRAM channels, the total additional chip area would be 0.027 mm^2 (less than 0.01 % additional die area). The estimated latency varies from 24 ps to 360 ps based on the complexity of the mechanisms. However, the most advanced mechanism (237 ps of the decoding scheme in Universal Base + XOR Transfer with Zero Data Remapping) is still short enough to be completed the operation within one clock cycle of DRAM (*e.g.,* 400 ps clock period in 10 Gbps. GDDR5X provides data at quadruple the clock frequency). Our encoding/decoding mechanisms consume very small additional energy (up to 201 fJ) compared to the overall energy consumption in the memory system, *e.g.,* the additional 1.82 pJ for transferring a `1` value.

VI. Evaluation

We evaluate the energy impact of our proposed mechanisms for running both compute and graphics workloads on a modern GPU system. To this end, we use a GPU simulator that models the streaming multiprocessors, cache hierarchy, and DRAM memory system. The simulator consists of a functional model that includes both graphics and compute

Mechanism (encode/decode)	Area (μm^2)	Energy (fJ/32B)	Latency (ps)	Config.
2-byte XOR	214/214	43/43	24/360	
4-byte XOR	289/289	73/73	24/168	
8-byte XOR	341/341	97/97	24/72	
Universal XOR	355/355	98/98	24/72	3 stage
ZDR	761/761	103/103	165/165	4B base
4-byte XOR + ZDR	1050/1050	176/176	189/333	
Universal XOR + ZDR	1116/1116	201/201	189/237	3 stage

Table II: Area, energy, and latency overhead for implementing Base + XOR Transfer to 32-byte transactions

pipelines, and the cycle-accurate timing model. The configuration of the evaluated system is summarized in Table I. We extract the data value of each transaction that is transferred to or from the DRAM at the memory controller for 187 compute and graphics workloads.

For compute workloads, we use 106 CUDA applications from Rodinia [19], Lonestar [20] benchmark suites, and Exascale workloads [21, 22, 23, 24, 25, 26] that include CoMD, HPGMG, lulesh, MCB, MiniAMR, and Nekbone. For graphics workloads, we use 81 applications that include a suite of popular DirectX games, 3D graphic rendering benchmarks, and workstation graphics applications.

A. N-Byte Base + XOR Transfer

We evaluate how many 1 values are reduced by using Base + XOR Transfer with Zero Data Remapping for three different base sizes (2, 4, and 8 bytes). Figure 11 plots the number of 1 values for 187 applications, normalized to the baseline without any mechanism. Applications that show their normalized 1 values are lower than the baseline (100 %) get benefits from our mechanism. We categorize applications into three groups based on the most beneficial base size.

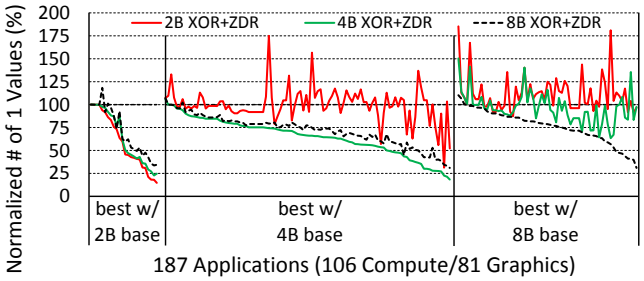


Figure 11: 2-/4-/8-byte Base + XOR Transfer

We draw three observations. First, each application shows a different 1 value reduction, due to its own characteristics of data similarity. Some applications show the largest 1 value reduction when applying a 2-byte base (the left group), while others show the best benefit when applying a 4-byte base (the middle group). Second, most of applications show 1 value reduction when using a proper base size, since the normalized 1 values of the best beneficial base case are mostly below the 100 %. Third, in most cases, applying either a 4-byte base or a 8-byte base provides 1 value reduction, however, applying a 2-byte base increases the number of 1 values in many applications. As we explained in Section IV-B, this is because *i)* using larger base size than the similar data structure size leads to losing the opportunities for better 1 value reduction, but still provides some 1 value reduction, and *ii)* using smaller base size than the underlying data element size leads to generating more 1 values. On average, 2-/4-/8-byte Base + XOR Transfer reduces 1 values by 6.5 %/29.7 %/29.6 %, respectively.

B. Universal Base + XOR Transfer

We evaluate the 1 value reduction with Universal Base + XOR Transfer that generates an *effective base element* and XORed elements without any *a priori* knowledge of the repeated pattern size. Figure 12 plots the normalized 1 values of Universal Base + XOR Transfer (red line) and the best 1 value reduction (black line) among the fixed three base size cases, shown in Figure 11. Since the red line closely tracks the black line, Universal Base + XOR Transfer successfully achieves the best of three base cases, in general.

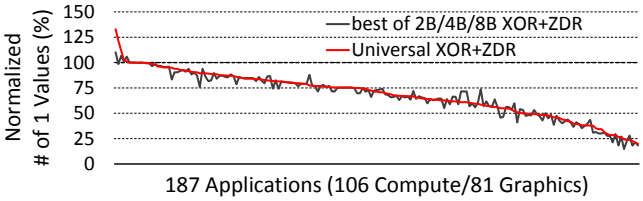


Figure 12: Universal Base + XOR Transfer

In some applications, the 1 value reduction in Universal Base + XOR Transfer is smaller than that of a fixed base case. This is because *N*-byte Base + XOR Transfer extracts the similar data structure between adjacent elements, while Universal Base + XOR Transfer performs XOR operations with elements that are longer distances apart. We observe that two adjacent elements typically show more similarities than two non-adjacent elements which enables better 1 value reduction in Base + XOR Transfer with a proper data structure. We also observe that Universal Base + XOR Transfer provides better 1 value reductions in some applications than the best of *N*-byte Base + XOR Transfer. These applications usually operate on different data structures with different sized elements. By generating the effective base for any repeated data patterns, universal base mechanism is able to provide better 1 value reduction than a fixed base mechanism. On average, Universal Base + XOR Transfer reduces 1 values by 35.3 %, a much higher reduction than that of the fixed base mechanisms (*e.g.,* 29.7 % with 4-byte Base + XOR Transfer).

In Figure 13, we categorize applications based on the amount of 1 value reduction they experience with the four mechanisms. Each bar in the figure represents the portion of applications which normalized 1 values are within a 20 % range. For example, the left most green bar in each mechanism shows the portion of applications that had the

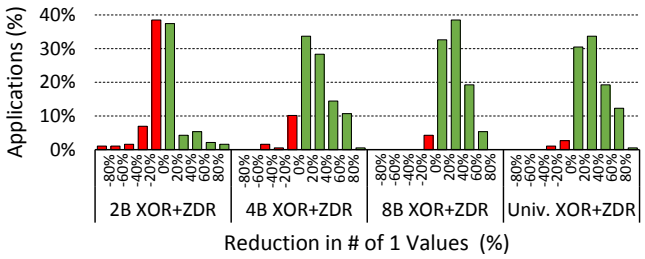


Figure 13: Application distribution of 1 value reduction

normalized 1 values in the range of 100 % to 80 % compared to the baseline.

We make two observations. First, using a larger base size for fixed Base + XOR Transfer results in fewer applications that see an *increase* in 1 values. For example, 4-byte Base + XOR Transfer has fewer applications that see increased 1 values than 2-byte Base + XOR Transfer. If only the simplest scheme can be adopted, an 8-byte base will result in the fewest applications with an increase in 1 values. Second, Universal Base + XOR Transfer enables *i)* the fewest applications that see an undesirable increase in 1 values and *ii)* the largest 1 value reduction on average.

C. Zero Data Remapping

Without Zero Data Remapping (ZDR), our Base + XOR Transfer scheme and previous work like SILENT [8] can increase the number of 1 values in applications that have a large number of zero valued elements. We found that in more than 20 % of the evaluated applications, greater than 30 % of transactions contain *mixed data*, *i.e.,* zero and non-zero elements interspersed together. ZDR is particularly important for these transactions. Figure 14 plots the relationship between the ratio of the mixed data transactions and the number of 1 values with/without ZDR. We observe that without ZDR, applications having more mixed data show less benefit in general from our scheme and even experience undesirable increases in 1 values on average. For example, applications in which more than 70 % of transactions are mixed data increase 1 values by 24 % on average without ZDR. As explained in Section IV-A, ZDR helps to avoid situations in which already-efficient zero data elements are encoded into values containing a large number of 1 values.

Overall, we found that without ZDR, a small number of applications experience a dramatic *increase* in the number of 1 values, with the worst case application suffering a 100% increase. Applying ZDR, this worst case application only sees an 8.4% increase in 1 values. The benefits of ZDR primarily apply to the subset of applications that have a large amount of mixed data which do poorly with the baseline scheme. Overall, applying ZDR reduces the number of applications that experience an increase in 1 values by 33%, and reduces the number of additional 1 values by 53.8%. This illustrates that ZDR is crucial for ensuring that the negative side effect of XORing the base with zero elements does not erode the benefits of Base + XOR Transfer.

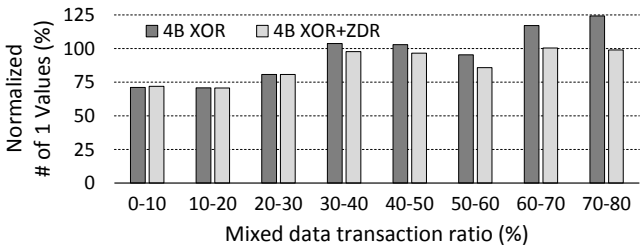


Figure 14: The impact of Zero Data Remapping on applications that have a different partial zero element ratio

D. Base + XOR Transfer vs. Previous Work

In Figure 15, we compare our mechanisms to two previous energy reduction mechanisms, Data Bus Inversion (DBI) and a cache-based 1 value reduction mechanism, Bitwise Difference Encoding (BD-Encoding) [4]. BD-Encoding holds 64 recently transferred 8-byte words in a temporal data repository and transfers data as a bitwise difference from the most similar one in the cache. BD-Encoding requires 8 bits of metadata to transfer 8 bytes of data, which includes the index of the most similar entry in the cache. We study multiple DBI implementations, dividing a 32-bit channel into multiple groups of different granularities (*e.g.,* four 1-byte groups) and applying DBI per group. Since each group with DBI requires one bit of polarity information, dividing a channel into more smaller groups can reduce 1 values more, but with higher overheads for additional polarity metadata. We study two different organizations of our mechanism, *i)* Universal Base + XOR Transfer with ZDR and *ii)* Universal Base + XOR Transfer with ZDR followed by N-byte DBI.

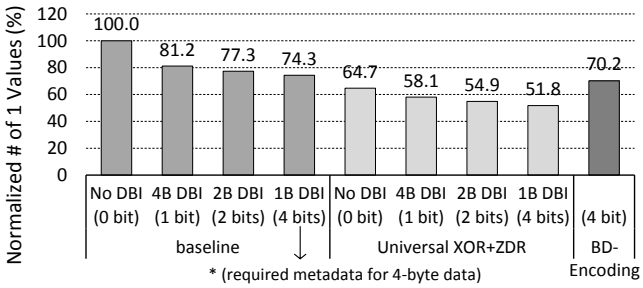


Figure 15: Base + XOR Transfer vs. Previous Works

We first compare our mechanism and DBI. Applying DBI to smaller size groups reduces more energy-expensive 1 values. For example, 4-byte DBI that requires 1 bit metadata per 32-bit bus reduces 1 values by 18.8 %, while 1-byte DBI that requires 4 bits of metadata per 32-bit bus reduces 1 values by 25.7 %. Our mechanism outperforms these DBI approaches *without any metadata or DRAM-side circuitry*, reducing 1 values by 35.3 %.

Second, we combine our mechanism and DBI by simply performing both encoding operations. We first apply Universal Base + XOR Transfer, then, perform DBI on the resulting encoded value. This hybrid mechanism enables much more substantial 1 value reductions. For example, Universal Base + XOR Transfer with 1-byte DBI (requiring

4 bits of metadata per 32-bit bus) achieves 48.2 % 1 value reduction compared to the baseline, and 22.5 % more 1 value reduction compared the 1-byte DBI alone. Combining these two mechanisms leverages the complementary nature of their 1 value reduction strategies. DBI reduces the number of one values within a single data element (*e.g.,* limiting the number of 1 values on a byte to no more than 4). Our mechanism, on the other hand, leverages the data similarity between elements in a transaction to reduce the number of 1 values. Furthermore, combining these approaches retains the important feature of DBI that guarantees no more than half of the data bits are ever simultaneously an 1 value.

Third, we compare our mechanism and BD-Encoding that requires 4 bits of metadata per 32-bit bus. Universal Base + XOR Transfer shows similar 1 value reduction to BD-Encoding (35.3 % vs. 29.8 %). Our mechanism achieves this without the metadata and repository for previous transactions required for BD-Encoding. Therefore, our mechanism provides similar energy reduction of DRAM channel at much lower cost compared to BD-Encoding. Moreover, considering schemes with the same metadata overhead (Universal Base + XOR Transfer with 1-byte DBI vs. BD-Encoding), our mechanism reduces 18.4 % more 1 values.

We observe two major reasons why our mechanism shows better 1 value reduction. First, BD-Encoding considers that two data elements are similar when the bitwise difference is lower than a predetermined threshold (*e.g.,* less than 12-bit bitwise differences in two data elements), which makes the mechanism very sensitive to the threshold. For example, if there exists 0x00000ffe in the cache repository, BD-Encoding determines that all 0x00000000 elements are considered to be similar to 0x00000ffe (since the bitwise difference is 11 bits), transferring bitwise difference of these two elements, 0x0000ffe. Second, BD-Encoding introduces metadata that also introduces additional signals that often can contain 1 values.

E. Channel Switching Activity Reduction

While our mechanism reduces energy-expensive 1 values, it also reduces the switching activities (*toggles*) on the I/O interface. Our mechanism significantly increases the probability that any given data bit to be transferred is a 0 value. This also has the effect of increasing the probability two successive data values to be transferred on the bus are identical. Figure 16 summarizes the toggle reduction impacts from different mechanisms. In general, DBI increases the number of toggles despite the 1 value reductions because additional toggles are introduced in the added metadata signals.[3] Universal Base + XOR Transfer provides 23.0 % toggle reduction by itself and 21.0 % applied together with the 1-byte DBI that exists in GDDR5/GDDR5X. Therefore, our mechanism also reduces data transfer energy by reducing channel switching activities as well as reducing energy-expensive 1 values.

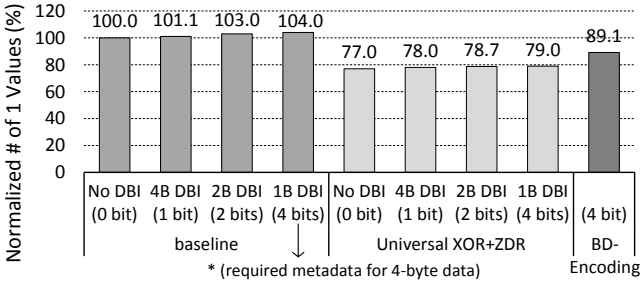


Figure 16: I/O switching activity reduction

F. DRAM Energy Reduction

We evaluate the energy consumption of a GDDR5X-based memory system. We model the detailed energy consumption including background energy (*e.g.,* leakage, clock, DLL, etc.), the row activation energy, and the on-chip/off-chip data transfer energy. In this evaluation, we assume that the DRAM bandwidth utilization is 70 % on average. Figure 17 shows the energy reduction using our mechanism, DBI, and BD-Encoding. Compared to the baseline, Universal Base + XOR Transfer achieves 5.8 % energy reduction with no metadata. Combining our mechanism and 1-byte DBI achieves a 7.1 % energy reduction, which is much bigger than the energy reduction with 1-byte DBI alone (2.7 %) or BD-Encoding (4.2 %). These energy savings are delivered from both reducing the number of 1 values and reducing the I/O switching activity (toggles).

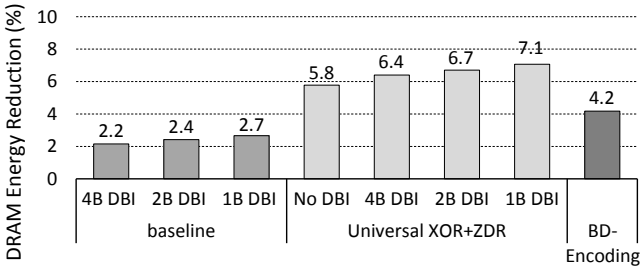


Figure 17: Energy reduction in memory system

G. CPU Workload Evaluation

Base + XOR Transfer can be applied without any modification in CPUs. We evaluate the impact of our mechanisms on a CPU system that has a single core, 4-Mbyte last-level cache, and DDR4-based memory system. As shown in Figure 18, our mechanism reduces the number of 1 values in 68 % of the applications and by 12 % on average across 28 applications in SPEC CPU2006. However, these reductions are much smaller than those for GPU applications. This is mainly because GPU applications have higher intra-transaction (or intra-cacheline) data similarity for our scheme to exploit, as shown in Section III-A. Furthermore,

[3] There are two DBI mechanisms, *i)* one of which reduces the number of 1 values (DBI-DC) and *ii)* the other of which reduces the number of toggles (DBI-AC). Since GDDR5/GDDR5X uses DBI-DC, we evaluate the impact of our mechanism on the POD I/O interface with *DBI-DC*.

GPU applications often use much more memory bandwidth and therefore dissipate higher DRAM power compared to CPUs, providing a stronger motivation for our proposals.

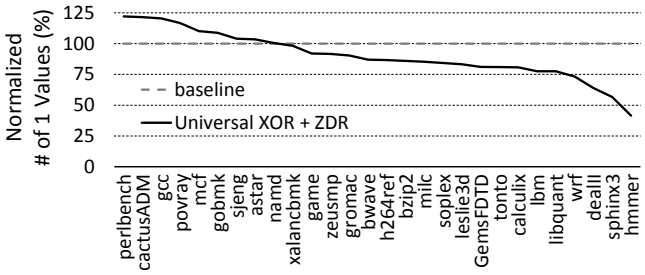


Figure 18: Base + XOR Transfer with CPU workloads

VII. Related Work

We reduce the DRAM energy consumption by leveraging the data similarity in a transaction. In this section, we describe the prior work related to our proposal.

Data Encoding that Leverages Bitwise Differences in Adjacent Data. SILENT [8] transfers the bitwise difference between adjacent words on a serial interface. By filtering out similar data, SILENT reduces switching activity and `1` values. However, as shown in Sections IV and VI-A, without a priori knowledge of the repeated data pattern size and a way to handle `0` data words, it is difficult to achieve the best energy efficiency. Our work highlights the importance of these considerations and proposes appropriate low cost solutions, improving upon SILENT.

BD-Encoding [4] reduces the energy consumption on the DRAM interface by transferring the bitwise differences from the most similar data that was recently transferred and stored in a cache. This work is similar to our work in terms of extracting similar data to transfer the same information with fewer number of `1`s. However, central to the BD-Encoding mechanism is a caching scheme which leads to high implementation overhead and complexity because it requires additional repositories in both the DRAM and in the memory controller to cache previously transmitted data words, comparison logic between requested data and all the cached data, and cache management logic. Our mechanism achieves the same goal with significantly lower implementation cost since it does not require any metadata. Of particular importance is the fact that, unlike BD-encoding, our mechanism needs no changes in the DRAM. We also compare our mechanism to BD-Encoding quantitatively in Section VI-D and show that our mechanism outperforms BD-Encoding when executing many compute and graphics workloads.

Data Encoding Techniques Leveraging Data Equality. Several previously proposed techniques [27, 28, 29, 30, 31, 32, 33, 34] store specific data in a cache repository and transfer an energy-efficient encoded form of the data when a transaction is matched to the cached data (*e.g.,* sending the index information of matching data in the cache). Our mechanism is different from this approach in several aspects. First, our mechanism leverages *data similarity*, while these approaches seek to exploit *data equality*. Therefore, even when there are small differences between subsequent data words, our mechanism can exploit the common portion of the data, reducing I/O energy. Second, we leverage the similarity within a transaction (*e.g.,* a sector). Therefore, our mechanism does not require any temporal repository, metadata, and comparison logic between the current transaction and previously transferred data. Therefore, our mechanism has much lower implementation overhead. Furthermore, since our encoding granularity is the same as data access granularity (*e.g.,* a sector or a cacheline), memories can store the encoded form of the data. Therefore, our mechanism does not require any changes in DRAM. Without any change on DRAM, our mechanism also can be synergistically combined with DBI that already exists in modern Graphics DDR DRAMs.

Reducing Overhead for Transferring Metadata. MiL [3] proposes a low cost implementation of Limited-Weight coding [35] in DRAM channel. By exploiting the underutilized DRAM channel bandwidth for transferring metadata, MiL mitigates the metadata overhead. Our mechanism is orthogonal to MiL and could be combined together for better energy efficiency in DRAM channel.

Cache Compression. Many compression mechanisms [6, 7, 10, 11, 12, 13, 36, 37, 38, 39, 40] enable *i)* storing more data in on-chip cache or memory (capacity benefit) and *ii)* transferring more data through interconnects (bandwidth benefit). While those cache compression mechanisms exploit the data similarity in a cacheline, which we also leverage in our mechanism, they differ in two ways. First, our mechanism focuses on reducing `1` values for reducing data transfer energy, while the compression mechanisms aim to enable capacity and bandwidth benefits. Thus, the achieved benefits are different. Prior work observed [41] that applying compression mechanisms does not provide energy efficiency benefits in general. Second, our mechanism can simply be applied for all data transactions, while compression mechanisms first need to determine if data structures (*e.g.,* cacheline) can be compressed, and then compress only data structures that are eligible for their mechanisms. Thus, our mechanism is much simpler and can be implemented with much lower overhead.

VIII. Conclusion

The energy consumption of high speed DRAM interfaces is a significant issue as future GPU systems continue to increase bandwidth. In this work, we focus on reducing the data transfer energy in the terminated, POD-based high speed DRAM I/O interfaces that consume more energy to transfer `1` values than `0` values. Our proposed encoding scheme exploits the intrinsic data similarity commonly found in GPU DRAM transactions, and, with our *Zero Data Remapping* and *Universal Base* techniques, is able

to significantly reduce the number of energy-expensive 1 bits transferred between the DRAM and the GPU for a broad range of GPU compute and graphics applications. Our proposed mechanism requires no metadata and very little area, energy, or delay. As a result, our approach can be easily used in future systems with existing DRAM devices. In the system we evaluated, our scheme applied in conjunction with the existing DBI scheme of GDDR5X, saves an additional 4.4 % of the memory system energy with less than 0.01 % area overhead for the encode/decode logic on the GPU.

While not the primary objective of our scheme, we also show that it is effective at reducing the toggle rate. Thus, our low-overhead approach may prove useful to save energy for a range of other non-terminated interconnects in which power is dominated by the dynamic capacitive switching effects. The current High-Bandwidth Memory (HBM) interface and large on-chip interconnect buses are potential applications. Adapting this technique to target toggle reduction and applying it to these other domains remains promising future work.

REFERENCES

[1] Micron, "Tn-ed-02: Gddr5x: The next-generation graphics dram introduction," https://www.micron.com/~/media/documents/products/technical-note/dram/tned02_gddr5x.pdf, 2005.

[2] M. Brox, M. Balakrishnan, M. Broschwitz, C. Chetreanu, S. Dietrich, F. Funfrock, M. A. Gonzalez, T. Hein, E. Huber, D. Lauber, M. Ivanov, M. Kuzmenka, C. Mohr, F. E. Munoz, J. O. Garrido, S. Padaraju, S. Piatkowski, J. Pottgiesser, P. Pfefferl, M. Plan, J. Polney, S. Rau, M. Richter, R. Schneider, R. O. Seitter, W. Spirkl, M. Walter, J. Weller, and F. Vitale, "23.1 An 8Gb 12Gb/s/pin GDDR5X DRAM for cost-effective high-performance applications," in *ISSCC*, 2017.

[3] Y. Song and E. Ipek, "More is Less: Improving the Energy Efficiency of Data Movement via Opportunistic Use of Sparse Codes," in *MICRO*, 2015.

[4] H. Seol, W. Shin, J. Jang, J. Choi, J. Suh, and L. S. Kim, "Energy Efficient Data Encoding in DRAM Channels Exploiting Data Value Similarity," in *ISCA*, 2016.

[5] M. R. Stan and W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," in *VLSI*, 1995.

[6] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-Delta-Immediate Compression: A Practical Data Compression Mechanism for On-Chip Caches," in *PACT*, 2012.

[7] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly Compressed Pages: A Low-complexity, Low-latency Main Memory Compression Framework," in *MICRO*, 2013.

[8] K. Lee, S.-J. Lee, and H.-J. Yoo, "SILENT: serialized low energy transmission coding for on-chip interconnection networks," in *ICCAD*, 2004.

[9] M. Harris, "Optimizing CUDA," http://gpgpu.org/static/sc2007/SC07_CUDA_5_Optimization_Harris.pdf, 2007.

[10] Y. Zhang, J. Yang, and R. Gupta, "Frequent Value Locality and Value-Centric Data Cache Design," in *ASPLOS*, 2000.

[11] J. Yang, Y. Zhang, and R. Gupta, "Frequent Value Compression in Data Caches," in *MICRO*, 2000.

[12] S. Balakrishnan and G. S. Sohi, "Exploiting Value Locality in Physical Register Files," in *MICRO*, 2003.

[13] M. Ekman and P. Stenström, "A Robust Main-Memory Compression Scheme," in *ISCA*, 2005.

[14] NVIDIA, "NVIDIA Titan X," https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal, 2016.

[15] Micron, "Mobile DRAM Power-Saving Features and Power Calculations," http://www.micron.com/~/media/documents/products/technical-note/dram/tn4612.pdf, 2005.

[16] Rambus, "DRAM Power Model," http://www.rambus.com/energy, 2010.

[17] S. Y. Wu, C. Y. Lin, M. C. Chiang, J. J. Liaw, J. Y. Cheng, S. H. Yang, M. Liang, T. Miyashita, C. H. Tsai, B. C. Hsu, H. Y. Chen, T. Yamamoto, S. Y. Chang, V. S. Chang, C. H. Chang, J. H. Chen, H. F. Chen, K. C. Ting, Y. K. Wu, K. H. Pan, R. F. Tsui, C. H. Yao, P. R. Chang, H. M. Lien, T. L. Lee, H. M. Lee, W. Chang, T. Chang, R. Chen, M. Yeh, C. C. Chen, Y. H. Chiu, Y. H. Chen, H. C. Huang, Y. C. Lu, C. W. Chang, M. H. Tsai, C. C. Liu, K. S. Chen, C. C. Kuo, H. T. Lin, S. M. Jang, and Y. Ku, "A 16nm FinFET CMOS technology for mobile SoC and computing applications," in *IEDM*, 2013.

[18] S. Y. Wu, C. Y. Lin, M. C. Chiang, J. J. Liaw, J. Y. Cheng, S. H. Yang, S. Z. Chang, M. Liang, T. Miyashita, C. H. Tsai, C. H. Chang, V. S. Chang, Y. K. Wu, J. H. Chen, H. F. Chen, S. Y. Chang, K. H. Pan, R. F. Tsui, C. H. Yao, K. C. Ting, T. Yamamoto, H. T. Huang, T. L. Lee, C. H. Lee, W. Chang, H. M. Lee, C. C. Chen, T. Chang, R. Chen, Y. H. Chiu, M. H. Tsai, S. M. Jang, K. S. Chen, and Y. Ku, "An enhanced 16nm CMOS technology featuring 2nd generation FinFET transistors and advanced Cu/low-k interconnect for low power and high performance applications," in *IEDM*, 2014.

[19] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *IISWC*, 2009.

[20] M. Burtscher and K. Pingali, "An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-body Algorithm," in *GPU Computing Gems Emerald Edition, Morgan Kaufmann*, 2011.

[21] S. Layton, N. Sakharnykh, and K. Clark, "GPU Implementation of HPGMG-FV," in *HPGMG BoF, Supercomputing*, 2015.

[22] M. F. Adams, J. Brown, J. Shalf, B. V. Straalen, E. Strohmaier, and S. Williams, "HPGMG 1.0: A Benchmark for Ranking High Performance Computing Systems," in *Techreport LBNL-6630E, Lawrence Berkley National Laboratory*, 2014.

[23] J. Mohd-Yusof and N. Sakharnykh, "Optimizing CoMD: A Molecular Dynamics Proxy Application Study," in *GTC*, 2014.

[24] "Coral benchmarks," https://asc.llnl.gov/CORAL-benchmarks/.

[25] "Mantevo benchmark suite," https://mantevo.org/packages.

[26] O. Villa, D. R. Johnson, M. O'Connor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, P. Wang, P. Micikevicius, A. Scudiero, S. W. Keckler, and W. J. Dally, "Scaling the Power Wall: A Path to Exascale," in *SC*, 2014.

[27] K. Basu, A. Choudhary, J. Pisharath, and M. Kandemir, "Power protocol: reducing power dissipation on off-chip data buses," in *MICRO*, 2002.

[28] J. Yang, R. Gupta, and C. Zhang, "Frequent Value Encoding for Low Power Data Buses," in *ACM Trans. Des. Autom. Electron. Syst.*, 2004.

[29] B. Bishop and A. Bahuman, "A low-energy adaptive bus coding scheme," in *IWV*, 2001.

[30] J. Yang and R. Gupta, "FV encoding for low-power data I/O," in *LPE*, 2001.

[31] V. Wen, M. Whitney, Y. Patel, and J. D. Kubiatowicz, "Exploiting prediction to reduce power on buses," in *HPCA*, 2004.

[32] C. Liu, A. Sivasubramaniam, and M. Kandemir, "Optimizing bus energy consumption of on-chip multiprocessors using frequent values," in *EMPDP*, 2004.

[33] D. C. Suresh, B. Agrawal, W. Najjar, and J. Yang, "VALVE: variable length value encoder for off-chip data buses," in *ICCD*, 2005.

[34] D. C. Suresh, B. Agrawal, J. Yang, and W. A. Najjar, "Tunable and Energy Efficient Bus Encoding Techniques," in *TC*, 2009.

[35] M. R. Stan and W. P. Burleson, "Limited-weight codes for low-power I/O," in *International Workshop on Low Power Design*, 1994.

[36] E. G. Hallnor and S. K. Reinhardt, "A Unified Compressed Memory Hierarchy," in *HPCA*, 2005.

[37] J. Dusser, T. Piquet, and A. Seznec, "Zero-Content Augmented Caches," in *ICS*, 2009.

[38] A. R. Alameldeen and D. A. Wood, "Adaptive Cache Compression for High-Performance Processors," in *ISCA*, 2004.

[39] B. Abali, H. Franke, D. E. Poff, R. A. Saccone, C. O. Schulz, L. M. Herger, and T. B. Smith, "Memory Expansion Technology (MXT): Software support and performance," in *IBM Journal of Research and Development*, 2001.

[40] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis, "MemZip: Exploring Unconventional Benefits from Memory Compression," in *HPCA*, 2014.

[41] G. Pekhimenko, E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry, and S. W. Keckler, "A case for toggle-aware compression for GPU systems," in *HPCA*, 2016.