A Case for Toggle-Aware Compression for GPU Systems

Gennady Pekhimenko[†], Evgeny Bolotin[⋆], Nandita Vijaykumar[†],
Onur Mutlu[†], Todd C. Mowry[†], Stephen W. Keckler[⋆#]

[†]Carnegie Mellon University [⋆]NVIDIA [#]University of Texas at Austin

ABSTRACT

Data compression can be an effective method to achieve higher system performance and energy efficiency in modern data-intensive applications by exploiting redundancy and data similarity. Prior works have studied a variety of data compression techniques to improve both capacity (e.g., of caches and main memory) and bandwidth utilization (e.g., of the on-chip and off-chip interconnects). In this paper, we make a new observation about the energy-efficiency of communication when compression is applied. While compression reduces the amount of transferred data, it leads to a substantial increase in the number of bit toggles *(i.e., communication channel switchings from 0 to 1 or from 1 to 0). The increased toggle count increases the dynamic energy consumed by on-chip and off-chip buses due to more frequent charging and discharging of the wires. Our results show that the total bit toggle count can increase from 20% to 2.2× when compression is applied for some compression algorithms, averaged across different application suites. We characterize and demonstrate this new problem across 242 GPU applications and six different compression algorithms. To mitigate the problem, we propose two new* toggle-aware compression *techniques:* Energy Control *and* Metadata Consolidation. *These techniques greatly reduce the bit toggle count impact of the data compression algorithms we examine, while keeping most of their bandwidth reduction benefits.*

1. Introduction

Modern data-intensive computing forces system designers to deliver good system performance under multiple constraints: shrinking power and energy envelopes (*power wall*), increasing memory latency (*memory latency wall*), and scarce and expensive bandwidth resources (*bandwidth wall*). While many different techniques have been proposed to address these issues, these techniques often offer a trade-off that improves one constraint at the cost of another. Ideally, system architects would like to improve one or more of these system parameters, e.g., on-chip and off-chip[1] bandwidth consumption, while simultaneously avoiding negative effects on other key parameters, such as overall system cost, energy, and latency characteristics. One potential way to address multiple constraints is to employ dedicated hardware-based *data compression* mechanisms (e.g., [71, 4, 14, 52, 6]) across different data links in the system. Compression exploits the high data redundancy observed in many modern applications [52, 57, 6, 69] and can be used to improve both capacity (e.g., of caches, DRAM, non-volatile memories [71, 4, 14, 52, 6, 51, 60, 50, 69, 74]) and bandwidth utilization (e.g., of on-chip and off-chip interconnects [15, 5, 64, 58, 51, 60, 69]). Several recent works focus on bandwidth compression to decrease memory traffic by transmitting data in a compressed form in both CPUs [51, 64, 5] and GPUs [58, 51, 69], which results in better system performance and energy consumption. Bandwidth compression proves to be particularly effective in GPUs because they are often bottlenecked by memory bandwidth [47, 32, 31, 72, 69]. GPU applications also exhibit high degrees of data redundancy [58, 51, 69], leading to good compression ratios.

While data compression can dramatically reduce the number of bit symbols that must be transmitted across a link, compression also carries two well-known overheads: (1) latency, energy, and area overhead of the compression/decompression hardware [4, 52]; and (2) complexity and cost to support variable data sizes [22, 57, 51, 60]. Prior work has addressed solutions to both of these problems. For example, Base-Delta-Immediate compression [52] provides a low-latency, low-energy hardware-based compression algorithm. Decoupled and Skewed Compressed Caches [57, 56] provide mechanisms to efficiently manage data recompaction and fragmentation in compressed caches.

1.1. Compression & Communication Energy

In this paper, we make a new observation that yet another important problem with data compression must be addressed to implement energy-efficient communication: transferring data in compressed form (as opposed to uncompressed form) leads to a significant increase in the number of *bit toggles*, i.e., the number of wires that switch from 0 to 1 or 1 to 0. An increase in bit toggle count causes higher switching activity [65, 9, 10] for wires, causing higher dynamic energy to be consumed by on-chip or off-chip interconnects. The bit toggle count of compressed data transfer increases for two reasons. First, the compressed data has a higher per-bit entropy because the same amount of information is now stored in fewer bits (the "randomness" of a single bit grows). Second, the variable-size nature of compressed data can negatively affect the word/flit data alignment in hardware. Thus, in contrast to the common wisdom that bandwidth compression saves energy (when it is effective), our key observation reveals a new trade-off: energy savings obtained by reducing bandwidth versus energy loss due to higher switching energy during compressed data transfers. This observation and the corresponding trade-off are the key contributions of this work.

To understand (1) how applicable general-purpose data compression is for real GPU applications, and (2) the severity of the problem, we use six compression algorithms [4, 52, 14, 51, 76, 53] to analyze 221 discrete and mobile graphics appli-

[1]Communication channel between the last-level cache and main memory.

978-1-4673-9211-2/16/$31.00

cation traces from a major GPU vendor and 21 open-source, general-purpose GPU applications. Our analysis shows that although off-chip bandwidth compression achieves a significant compression ratio (e.g., more than 47% average effective bandwidth increase with C-Pack [14] across mobile GPU applications), it also greatly increases the bit toggle count (e.g., a corresponding 2.2× average increase). This effect can significantly increase the energy dissipated in the on-chip/off-chip interconnects, which constitute a significant portion of the memory subsystem energy.

1.2. Toggle-Aware Compression

In this work, we develop two new techniques that make bandwidth compression for on-chip/off-chip buses more energy-efficient by limiting the overall increase in compression-related bit toggles. *Energy Control (EC)* decides whether to send data in compressed or uncompressed form, based on a model that accounts for the compression ratio, the increase in bit toggles, and current bandwidth utilization. The key insight is that this decision can be made in a fine-grained manner (e.g., for every cache line), using a simple model to approximate the commonly-used $Energy \times Delay$ and $Energy \times Delay^2$ metrics. In this model, $Energy$ is directly proportional to the bit toggle count; $Delay$ is inversely proportional to the compression ratio and directly proportional to the bandwidth utilization. Our second technique, *Metadata Consolidation (MC)*, reduces the negative effects of scattering the metadata across a compressed cache line, which happens with many existing compression algorithms [4, 14]. Instead, MC consolidates compression-related metadata in a contiguous fashion.

Our toggle-aware compression mechanisms are generic and applicable to different compression algorithms (e.g., Frequent Pattern Compression (FPC) [4] and Base-Delta-Immediate (BDI) compression [52]), different communication channels (on-chip and off-chip buses), and different architectures (e.g., GPUs, CPUs, and hardware accelerators). We demonstrate that our proposed mechanisms are mostly orthogonal to different data encoding schemes also used to minimize the bit toggle count (e.g., Data Bus Inversion [63]), and hence can be used together with them to enhance the energy efficiency of interconnects.

Our extensive evaluation shows that our proposed mechanisms can significantly reduce the negative effect of the increase in bit toggles (e.g., our mechanisms almost completely eliminate the 2.2× increase in bit toggle count in one workload), while preserving most of the benefits of data compression when it is useful (such that the reduction in performance benefits from compression is usually within only 1%). This efficient trade-off leads to significant reductions in (i) DRAM energy (of up to 28.1%, and 8.3% on average), and (ii) total system energy (of up to 8.9%, and 2.1% on average). Moreover, our mechanisms can greatly reduce the energy cost to support data compression over the on-chip interconnect. For example, our toggle-aware compression mechanisms can reduce the original 2.1× average increase in on-chip interconnect energy consumption with C-Pack compression algorithm to a much more acceptable 1.1× increase.

2. Background

Data compression is a powerful mechanism that exploits the existing redundancy in the applications' data to relax capacity and bandwidth requirements for many modern systems. Hardware-based data compression was explored in the context of on-chip caches [71, 4, 14, 52, 57, 6] and main memory [2, 64, 18, 51, 60], but mostly for CPU-oriented applications. Several prior works [64, 51, 58, 60, 69] examined the specifics of memory bandwidth compression, where it is critical to decide where and when to perform compression and decompression.

While these works evaluated the energy/power benefits of bandwidth compression, the overhead of compression was limited to the examined overheads of 1) the compression/decompression logic and 2) the newly-proposed mechanisms/designs. To our knowledge, this is the first work that examines the energy implications of compression on the data transferred over the on-chip/off-chip buses. Depending on the type of the communication channel, the transferred data bits have different effect on the energy spent on communication. We provide a brief background on this effect for three major communication channel types.

On-chip Interconnect. For the full-swing on-chip interconnects, one of the dominant factors that defines the energy cost of a single data transfer (commonly called a flit) is the activity factor—the number of *bit toggles* on the wires (communication channel switchings from 0 to 1 or from 1 to 0). The bit toggle count for a particular flit depends on both the flit's data and the data that was previously sent over the same wires. Several prior works [63, 10, 73, 66, 9] examined more energy-efficient data communication in the context of on-chip interconnects [10], reducing the number of bit toggles. The key difference between our work and these prior works is that we aim to address the effect of increase (sometimes a dramatic increase, see Section 3) in bit toggle count *specifically due to data compression*. Our proposed mechanisms (described in Section 4) are mostly orthogonal to these prior mechanisms and can be used together with them to achieve even larger energy savings in data transfers.

DRAM bus. In the case of DRAM (e.g., GDDR5 [26]), the energy attributed to the actual data transfer is usually less than the background and access energy, but still significant (16% on average based on our estimation with the Micron power calculator [44]). The second major distinction between on-chip and off-chip buses, is the definition of bit toggles. In case of DRAM, bit toggles are defined as the number of zero bits. Reducing the number of signal lines driving a low voltage level (zero bit) results in reduced power dissipation in the termination resistors and output drivers [26]. To reduce the number of zero bits, techniques like DBI (data-bus-inversion) are usually used. For example, DBI is part of the standard for GDDR5 [26] and DDR4 [28]. As we will show later in

Section 3, these techniques are *not effective enough* to handle the significant increase in bit toggles due to data compression.

PCIe and SATA. For SATA and PCIe, data is transmitted in a serial fashion at much higher frequencies than typical parallel bus interfaces. Under these conditions, bit toggles impose different design considerations and implications. Data is transmitted across these buses without an accompanying clock signal which means that the transmitted bits need to be synchronized with a clock signal by the receiver. This *clock recovery* requires *frequent* bit toggles to prevent information loss. In addition, it is desirable that the *running disparity*—which is the difference in the number of one and zero bits transmitted—be minimized. This condition is referred to as the *DC balance* and prevents distortion in the signal. Data is typically scrambled using encodings like the 8b/10b encoding [70] to balance the number of ones and zeros while ensuring frequent transitions. These encodings have high overhead in terms of the amount of additional data transmitted but they obscure any difference in bit transitions with compressed or uncompressed data. As a result, we do not expect further compression or toggle-rate reduction techniques to apply well to interfaces like SATA and PCIe.

Summary. With on-chip interconnect, *any bit toggle* increases the energy expended during data transfer. In the case of DRAM, energy spent during data transfers increases with an increase in *zero* bits. Data compression exacerbates the energy expenditure in both these channels. For PCIe and SATA, data is scrambled before transmission and this obscures any impact of data compression and, hence, our proposed mechanisms are not applicable to these channels.

3. Motivation and Analysis

In this work, we examine the use of six compression algorithms for bandwidth compression in GPU applications, taking into account bit toggles: (i) *FPC* (Frequent Pattern Compression) [4]; (ii) *BDI* (Base-Delta-Immediate Compression) [52]; (iii) *BDI+FPC* (combined FPC and BDI) [51]; (iv) *LZSS* (Lempel-Ziv compression) [76, 2]; (v) *Fibonacci* (a graphics-specific compression algorithm) [53]; and (vi) *C-Pack* [14]. All of these compression algorithms explore different forms of redundancy in memory data. For example, FPC and C-Pack algorithms look for different static patterns in data (e.g., high order bits are zeros or the word consists of repeated bytes). At the same time, C-Pack allows partial matching with some locally defined dictionary entries, which usually gives it better coverage than FPC. In contrast, the BDI algorithm is based on the observation that the whole cache line of data can be commonly represented as one or two bases and respective deltas from these bases. This allows compression of some cache lines much more efficiently than FPC and even C-Pack, but potentially leads to lower coverage. For completeness of our analysis of compression algorithms, we also examine the well-known software-based mechanism, called LZSS, and the recently proposed graphics-oriented Fibonacci algorithm.

To ensure our conclusions are practically applicable, we analyze both the real GPU applications (both *discrete* and *mobile* ones) with actual data sets provided by a major GPU vendor and *open-source* GPU computing applications [48, 13, 24, 11]. The primary difference is that discrete applications have more single and double precision floating point data, mobile applications have more integers, and open-source applications are in-between. Figure 1 shows the effect of these six compression algorithms in terms of effective bandwidth increase, averaged across all applications. These results exclude simple data patterns (e.g., zero cache lines) that are already efficiently handled by modern GPUs, and assume practical boundaries on bandwidth compression ratios (e.g., for on-chip interconnect, the highest possible compression ratio is 4.0, because the minimum flit size is 32 bytes while the uncompressed packet size is 128 bytes).

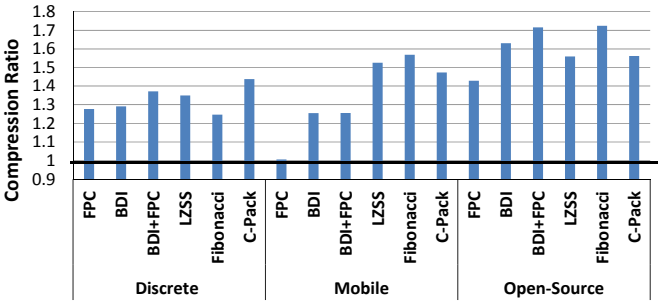

Figure 1: Effective bandwidth compression ratios for various GPU applications and compression algorithms (higher bars are better).

First, for the 167 discrete GPU applications (left side of Figure 1), all algorithms provide substantial increase in available bandwidth (25%–44% on average for different compression algorithms). It is especially interesting that simple compression algorithms like BDI are very competitive with the more complex GPU-oriented *Fibonacci* algorithm and the software-based Lempel-Ziv algorithm [76]. Second, for the 54 mobile GPU applications (middle part of Figure 1), bandwidth benefits are even more pronounced (reaching up to 57% on average with the Fibonacci algorithm). Third, for the 21 open-source GPU computing applications, the bandwidth benefits are the highest (as high as 72% on average with the Fibonacci and BDI+FPC algorithms). Overall, we conclude that existing compression algorithms (including simple, general-purpose ones) can be effective in providing high on-chip/off-chip bandwidth compression for GPU applications. Unfortunately, the benefits of compression come with additional costs. Two overheads of compression are well-known: (i) additional processing due to compression/decompression, and (ii) hardware changes due to supporting and transfering variable-length cache lines. While these two problems are significant, multiple compression algorithms [4, 71, 52, 17] have been proposed to minimize these two overheads of data compression/decompression. Several designs [60, 58, 51, 69] integrate bandwidth compression into existing memory hierarchies. In this work, we identify a new challenge with data compression that needs

to be addressed: the increase in the total number of bit toggles as a result of compression.

On-chip data communication energy is directly proportional to the number of bit toggles on the communication channel [65, 9, 10], due to the charging and discharging of the channel wire capacitance with each toggle. Data compression may increase or decrease the bit toggle count on the communication channel for any given data. As a result, energy consumed for moving this data can change. Figure 2 shows the increase in bit toggle count for all GPU applications in our workload pool with the six compression algorithms over a baseline that employs zero cache line compression (as this is already efficiently done in modern GPUs). The total number of bit toggles is computed such that it already includes the positive effects of compression (i.e., the decrease in the total number of bits sent due to compression).

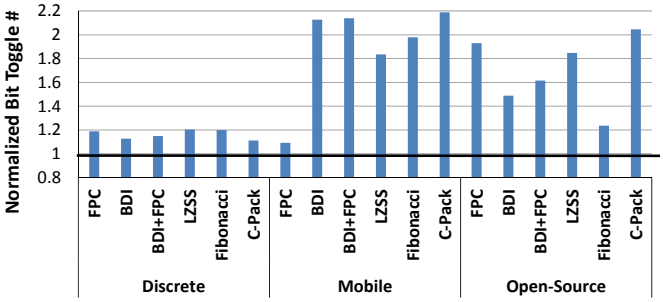

Figure 2: Bit toggle count increase due to compression.

We make two observations. First, all compression algorithms consistently increase the bit toggle count. The effect is significant yet smaller (12%–20% increase) in discrete applications, mostly because they include floating-point data, which already has high toggle rates (31% on average across discrete applications) and is less amenable to compression. This increase in bit toggle count happens even though we transfer less data due to compression. If this effect were due only to the higher density of information per bit, we would expect an increase in bit toggle *rate* (the relative fraction of bit toggles per data transfer), but not in bit toggle *count* (the total *number* of bit toggles).

Second, the increase in bit toggle count is more dramatic for mobile and open-sourced applications (the rightmost two-thirds of Figure 2), exceeding 2× in four cases.[2] For all types of applications, the increase in bit toggle count can lead to significant increase in the dynamic energy consumption of the communication channels.

We study the relationship between the achieved compression ratio and the resulting increase in bit toggle count. Figure 3 shows the compression ratio and the normalized bit toggle count of each discrete GPU application after compression with the FPC algorithm.[3] Clearly, there is a positive correlation between the compression ratio and the increase in bit toggle count, although it is not a simple direct correlation—higher compression ratio does *not* necessarily mean higher increase in bit toggle count. To make things worse, the behavior might change within an application due to phase and data patterns changes.

We draw two major conclusions from this study. First, it strongly suggests that successful compression may lead to higher dynamic energy dissipation by on-chip/off-chip communication channels due to increased bit toggle count. Second, these results show that any efficient solution for this problem should probably be dynamic in its nature to adopt for data pattern changes during application execution.

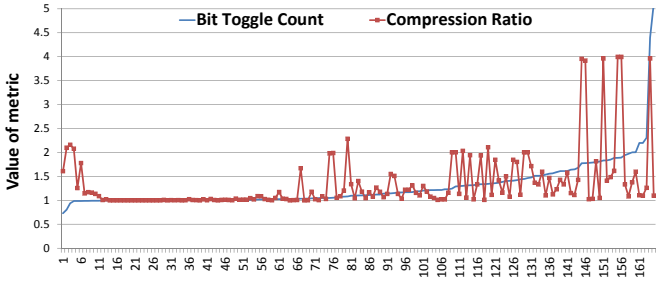

Figure 3: Normalized bit toggle count vs. compression ratio (with the FPC algorithm) for each of the 167 discrete GPU applications.

To understand the phenomenon of bit toggle count increase, we examined several example cache lines where bit toggle count increases significantly after compression. Figures 4 and 5 show one of these cache lines with and without compression (FPC), assuming 8-byte flits.

Without compression, the example cache line in Figure 4, which consists of 8-byte data elements (4-byte indices and 4-byte pointers), has a very small number of bit toggles (2 toggles per 8-byte flit). This small number of bit toggles is due to the favorable alignment of the uncompressed data with the boundaries of flits (i.e., transfer granularity in the on-chip interconnect). With compression, the bit toggle count of the same cache line increases significantly, as shown in Figure 5 (e.g., 31 toggles for a pair of 8-byte flits in this example). This increase is due to two major reasons. First, because compression removes zero bits from narrow values, the resulting higher per-bit entropy leads to higher "randomness" in data and, thus, a larger bit toggle count. Second, compression negatively affects the alignment of data both at the byte granularity (narrow values are replaced with shorter 2-byte versions) and bit granularity (due to the 3-bit metadata storage; e.g., 0x5 is the encoding metadata used to indicate narrow values for the FPC algorithm).

4. Toggle-Aware Compression

We introduce the key ideas of our mechanisms to combat bit toggle count increases due to compression. To this end, we first examine the energy-performance trade-off introduced due to larger bit toggle counts caused by compression.

4.1. Energy vs. Performance Trade-off

Data compression can significantly reduce energy consumption and improve performance by reducing communication

[2]The FPC algorithm is not as effective in compressing mobile application data in our pool, and hence does not greatly affect the bit toggle count.
[3]We observe similarly-shaped curves for other compression algorithms.

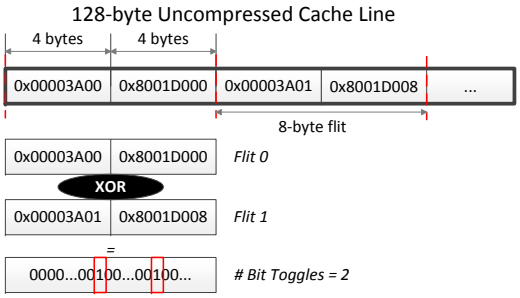

Figure 4: Bit toggles without compression.

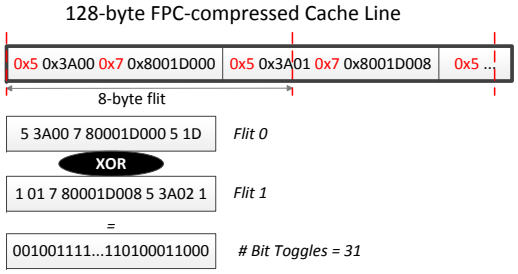

Figure 5: Bit toggles after compression.

bandwidth demands. At the same time, data compression can potentially lead to significantly higher energy consumption due to increased bit toggle count. To properly evaluate this trade-off, we examine commonly-used metrics like $Energy \times Delay$ and $Energy \times Delay^2$ [19]. We estimate these metrics with a simple model, which can facilitate compression-related performance/energy trade-offs. We define the $Energy$ of a single data transfer to be proportional to the bit toggle count associated with it. Similarly, $Delay$ is defined to be inversely proportional to performance, which we assume is proportional to bandwidth reduction (i.e., compression ratio) and bandwidth utilization. The intuition behind this heuristic is that compression ratio affects how much additional bandwidth we can get, while bandwidth utilization shows how useful this additional bandwidth could be in improving performance. Based on the observations above, we develop two techniques to enable *toggle-aware compression* to reduce the negative effects of increased bit toggle count.

4.2. Energy Control (EC)

We propose a generic *Energy Control* (EC) mechanism that can be applied along with any current (or future) compression algorithm.[4] It aims to achieve high compression ratio while minimizing the bit toggle count. As shown in Figure 6, the Energy Control mechanism uses a generic decision function that considers (i) the bit toggle count for transmitting the original data (T_0), (ii) the bit toggle count for transmitting the data in compressed form (T_1), (iii) compression ratio (CR), (iv) current bandwidth utilization (BU), and possibly other metrics of interest that can be gathered and analyzed dynamically to decide whether to transmit the data compressed or uncompressed. Using this approach, it is possible to achieve a desirable trade-off between overall bandwidth reduction and increase/decrease in communication energy. The decision function that compares the compression ratio (A) and toggle ratio (B) can be linear ($A \times B > 1$, based on $Energy \times Delay$) or quadratic ($A \times B^2 > 1$, based on $Energy \times Delay^2$).[5] Specifically, when the bandwidth utilization (BU) is very high (e.g., $BU > 50\%$), we incorporate it into our decision function by multiplying the compression ratio with $\frac{1}{1-BU}$, thereby allocating more weight to the compression ratio. Since the data patterns during application execution could change drastically, we expect our mechanism to be applied dynamically (either per cache line or a per region of execution) rather than statically (for the whole application execution).

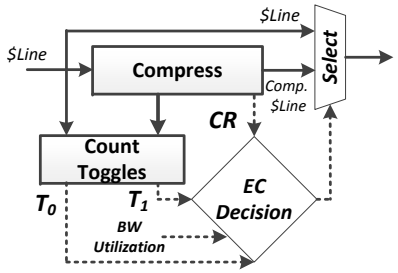

Figure 6: Energy Control decision mechanism.

4.3. Metadata Consolidation

Traditional energy-oblivious compression algorithms are not optimized to minimize the bit toggle count. Most of these algorithms [14, 4, 53] have metadata that is shared together with each piece of compressed data to efficiently track the redundancy in data, e.g., several bits per word to represent the current pattern used for encoding. These metadata bits can significantly increase the bit toggle count as they disturb the potentially good alignment between different words within a cache line (Section 3). It is possible to enhance these compression algorithms (e.g., FPC and C-Pack) such that the increase in bit toggle count would be less after compression is applied. Metadata Consolidation (MC) is a new technique that aims to achieve this. The key idea of MC is to consolidate compression-related metadata into a *single contiguous metadata block* instead of storing (or, scattering) such metadata in a fine-grained fashion, e.g., on a per-word basis. We can locate this single metadata block either before or after the actual compressed data (this can increase decompression latency since the decompressor needs to know the metadata). The major benefit of MC is that it eliminates misalignment at the bit granularity. In cases where a cache line has a majority of similar patterns, a significant portion of the bit toggle count increase can thus be avoided.

Figure 7 shows an example cache line compressed with the FPC algorithm, with and without MC. We assume 4-byte

[4] In this work, without loss of generality, we assume that only memory bandwidth is compressed, while on-chip caches and main memory still store data in uncompressed form.

[5] We empirically find the specific coefficient determining the relative weights of $Energy$ and $Delay$ in this equation.

flits. Without MC, the bit toggle count between the first two flits is 18 (due to per-word metadata insertion). With MC, the corresponding bit toggle count is only 2, showing the effectiveness of MC in reducing bit toggles.

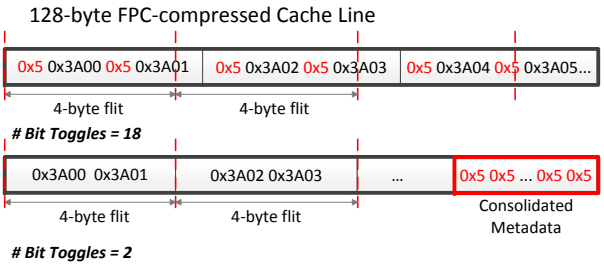

Figure 7: Bit toggle count w/o and with Metadata Consolidation.

5. EC Architecture

In this work, we assume a system where global on-chip network and main memory communication channels are augmented with compressor and decompressor units as shown in Figure 8 and Figure 9. While it is possible to store data in the compressed form as well (e.g., to improve the capacity of on-chip caches [71, 4, 52, 14, 57, 6]), the corresponding changes come with potentially significant hardware complexity that we would like to avoid in our design. We first attempt to compress the data traffic coming in and out of the channel with one (or a few) compression algorithms. The results of the compression, both the compressed cache line size and data, are then forwarded to the Energy Control (EC) logic that is described in detail in Section 4. EC decides whether it is beneficial to send data in the compressed or uncompressed form, after which the data is transferred over the communication channel. It is then decompressed if needed at the other end, and the data flow proceeds normally. In the case of main memory bus compression (Figure 9), additional EC and compressor/decompressor logic can be implemented in an existing base-layer die, assuming 3D-stacked memory organization [29, 25, 35, 43], or in an additional layer between DRAM and the main memory bus. Alternatively, the data can be stored in the compressed form but without any capacity benefits [58, 60].

5.1. Bit Toggle Count Computation for On-Chip Interconnect

As described in Section 4, our proposed mechanism, EC, aims to decrease the negative effect of data compression on bit toggle count while preserving most of the compression benefits. GPU on-chip communication is performed by exchanging packets at a cache line size granularity. However, the physical width of an on-chip interconnect channel is usually several times smaller than the size of a cache line (e.g., a 32-byte wide channel for a 128-byte cache line). As a result, the communication packet is divided into multiple *flits* that are stored at the transmission queue buffer before being transmitted over the communication channel in a sequential manner. Our approach adds a simple bit toggle count computation logic

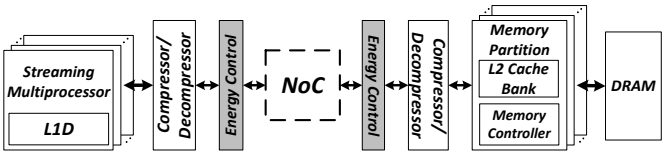

Figure 8: System overview with on-chip interconnect bandwdith compression and EC.

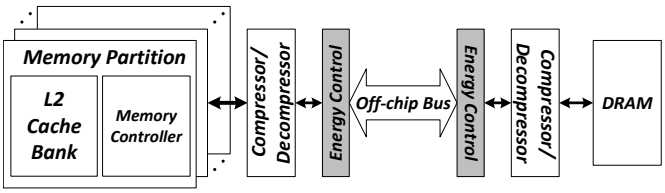

Figure 9: System overview with off-chip bus bandwidth compression and EC.

that computes the bit toggle count across flits awaiting transmission. This logic consists of a flit-wide array of XORs and a tree-adder to compute the *Hamming distance*, the number of bits that are different, between two flits. We perform this computation for both compressed and uncompressed data, and the results are then fed to the EC decision function (as described in Figure 6). This computation can be done sequentially while reusing the transition queue buffers to store intermediate compressed or uncompressed flits, or in parallel with the addition of some dedicated flit buffers (to reduce the latency overhead). In this work, we assume the second approach.

5.2. Bit Toggle Count Computation for DRAM

For modern DRAMs [26, 28], the bit toggle definition is different from the definition we use for on-chip interconnects. As we described in Section 2, in the context of the main memory bus, what matters is the number of zero bits per data transfer. This defines how we compute the bit toggle count for DRAM transfers by simply counting the zero bits—which is known as the *Hamming weight* or the *population count* of the inverted value. The difference in the definition of the bit toggle count on the DRAM bus does not require previously-transmitted data to be known to determine the bit toggle count of the current transmission. Hence, no additional buffering is required to perform this computation.

5.3. EC and Data Bus Inversion

Modern communication channels use different techniques to minimize (or maximize) the bit toggle count to reduce energy consumption or/and preserve signal integrity. We now briefly summarize two major techniques used in existing on-chip/off-chip interconnects: *Data Bus Inversion* and *Data Scrambling*, and their effect on our proposed EC mechanism.

5.3.1. Data Bus Inversion. Data Bus Inversion is an encoding technique proposed to reduce the power consumption in data channels. Two commonly used DBI algorithms include *Bus invert coding* [63] and *Limited-weight coding* [61, 62]. *Bus invert coding* places an upper-bound on the number of bit flips

while transmitting data along a channel. Consider a set of N bitlines transmitting data in parallel. If the Hamming distance between the previous and current data values being transmitted exceeds *N/2*, the data is transmitted in the inverted form. This limits the number of bit flips to *N/2*. To preserve correctness, an additional bitline carries the inverted status of each data tranmission. By reducing the number of bit flips, *Bus invert coding* reduces the switching power associated with charging and discharging of bitlines.

Limited weight coding is a DBI technique that helps reduce power when one of the two different bus states is more dissipative than the other. The algorithm observes only the *current* state of data. It decides to invert or leave the data inverted to minimize either the number of *zeros* or *ones* being transmitted.

Implementing *Bus invert coding* requires much the same circuitry as bit toggle count in our EC mechanism. Hardware logic is required to compute the XOR between the previous current transmitted data at a fixed granularity. The Hamming distance is then computed by summing the number of 1's using a simple adder. Similar logic is required to compute the bit toggle count for compressed versus uncompressed data in the Energy Control mechanism. We expect that EC and DBI can efficiently coexist. After compression is applied, we can first apply DBI (to minimize the bit toggles), and after that we can apply the EC mechanism to evaluate the tradeoff between the compression ratio and bit toggle count.

5.3.2. Data Scrambling. To minimize signal distortion, some modern DRAM designs [30, 46] use a *data scrambling* technique that aims to minimize the running data disparity, i.e., the difference between the number of 0s and 1s, in the transmitted data. One way to "randomize" the bits is by XORing them with pseudo-random values generated at boot time [46]. While techniques like data scrambling can potentially decrease signal distortion, they also increase the dynamic energy of DRAM data transfers. This approach therefor contradicts what several designs aim to achieve by using DBI for GDDR5 [26] and DDR4 [28], since data scrambling causes the bits to become much more random.

Using pseudo-random data scrambling techniques can also reduce certain pathological data patterns [46], where signal integrity requires much lower operational frequency. However, such patterns can usually be handled well with data compression algorithms that can provide the appropriate data transformation to avoid repetitive failures at a certain frequency. For these reasons, we assume a GDDR5 memory system without scrambling.

5.4. Complexity Estimation

Bit toggle count computation is the main hardware addition introduced by the EC mechanism. We modeled and synthesized the bit toggle count computation block in Verilog. Our results show that the required logic is energy-efficient way: it consumes 4pJ per 128-byte cache line with 32-byte flits for 65nm process. This is significantly lower than the corresponding energy for compression and decompression [60].

6. Methodology

We analyze two distinct groups of applications. First, we evaluate a group of 221 applications from a major GPU vendor in the form of memory traces with real application data. This group consists of two subgroups: *discrete* applications (e.g., HPC, physics, and general-purpose applications) and *mobile* applications. As there is no existing simulator that can run these traces for cycle-accurate simulation, we use them to demonstrate (i) the benefits of compression on a large pool of existing applications operating on real data, and (ii) the existence of the bit toggle count increase problem. Second, we use 21 *open-source* GPU compute applications derived from CUDA SDK [48] (*BFS, CONS, JPEG, LPS, MUM, RAY, SLA, TRA*), Rodinia [13] (*hs, nw*), Mars [24] (*KM, MM, PVC, PVR, SS*), and Lonestar [11] (*bfs, bh, mst, sp, sssp*) workload suites.

We evaluate the performance of our proposed mechanisms with the second group of applications using the GPGPU-Sim 3.2.2 [7] cycle-accurate simulator. Table 1 provides all the details of the simulated system. We use GPUWattch [38] for energy analysis, with proper modifications to take into account bit toggle counts. We run all applications to completion or 1 billion instructions (whichever comes first). Our evaluation in Section 7 demonstrates detailed results for applications that exhibit at least 10% bandwidth compressibility.

System Overview	15 SMs, 32 threads/warp, 6 memory channels
Shader Core Config	1.4GHz, GTO scheduler [55], 2 schedulers/SM
Resources / SM	48 warps/SM, 32K registers, 32KB Shared Mem.
L1 Cache	16KB, 4-way associative, LRU
L2 Cache	768KB, 16-way associative, LRU
Interconnect	1 crossbar/direction (15 SMs, 6 MCs), 1.4GHz
Memory Model	177.4GB/s BW, 6 GDDR5 Memory Controllers, FR-FCFS scheduling, 16 banks/MC
GDDR5 Timing [26]	$t_{CL} = 12, t_{RP} = 12, t_{RC} = 40, t_{RAS} = 28, t_{RCD} = 12, t_{RRD} = 6, t_{CLDR} = 5, t_{WR} = 12$

Table 1: Major Parameters of the Simulated System.

Evaluated Metrics. We present Instructions per Cycle (*IPC*) as the primary performance metric. We also use *average bandwidth utilization* defined as the fraction of total DRAM cycles that the DRAM data bus is busy, and *compression ratio*, defined as the effective bandwidth increase. For both on-chip interconnect and DRAM we assume the highest possible compression ratio is 4.0. For on-chip interconnect, this is because we assume a flit size of 32 bytes for a 128-byte packet. For DRAM, existing GPUs (e.g., GeForce FX series) are known to support 4:1 data compression [1].[6]

[6]For DRAM, there are multiple ways of achieving the desired flexibility in data transfers: (i) increasing the size of a cache line (from 128 bytes to 256 bytes), (ii) using sub-ranking as was proposed for DDR3 in the MemZip design [60], (iii) transferring multiple compressed cache lines instead of one uncompressed line as in the LCP design [51], and (iv) any combination of the first three approaches.

7. Evaluation

We present our results for the two communication channels described earlier: (i) off-chip DRAM bus and (ii) on-chip interconnect. We exclude LZSS compression algorithm from our detailed evaluation since its hardware implementation is not practical due to compression/decompression latencies [2] of hundreds of cycles.

7.1. DRAM Bus Results

7.1.1. Effect on Bit Toggle Count and Compression Ratio. We analyze the effectiveness of the proposed EC optimization by examining how it affects both the number of bit toggles (Figure 10) and the compression ratio (Figure 11) on the DRAM bus for five compression algorithms. In both figures, results are averaged across all applications within the corresponding application subgroup and normalized to the baseline design with no compression. Unless specified otherwise, we use the EC mechanism with the decision function based on the $Energy \times Delay^2$ metric using our model from Section 4.2. We make two observations from these figures.

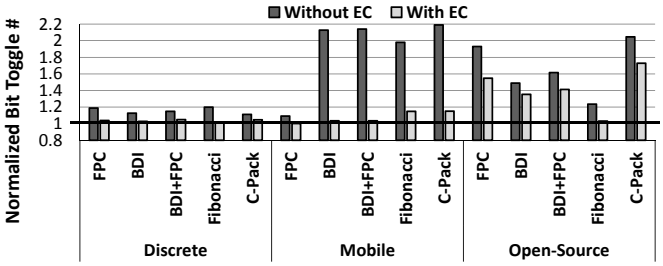

Figure 10: Effect of Energy Control on the bit toggle count on the DRAM bus.

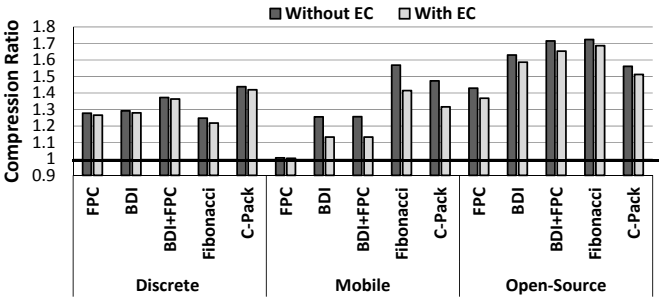

Figure 11: Effect of Energy Control on compression ratio (i.e., effective bandwidth increase) on the DRAM bus.

First, we observe that EC effectively reduces the bit toggle count overhead for both discrete and mobile GPU applications (Figure 10). For discrete GPU applications, the bit toggle count reduction varies from 6% to 16% on average, and the bit toggle count increase due to compression is almost completely eliminated in the case of the Fibonacci compression algorithm. For mobile GPU applications, the bit toggle count reduction is as high as 51% on average for the BDI+FPC compression algorithm (i.e., more than 32× reduction in *extra* bit toggles), with only a modest reduction[7] in compression ratio.

Second, the reduction in compression ratio with EC is usually small. For example, in discrete GPU applications, this reduction for the BDI+FPC algorithm is only 0.7% on average (Figure 11). For mobile and open-source GPU applications, the reduction in compression ratio is more noticeable (e.g., 9.8% on average for Fibonacci with mobile applications), which is still a very attractive trade-off since the very large (e.g., 2.2×) growth in bit toggle count is greatly reduced or practically eliminated in many cases. We conclude that EC offers an effective way to control the energy efficiency of data compression for the DRAM bus by applying it only when it provides a high compression ratio along with a small increase in bit toggle count.

While the average numbers presented express the general effect of the EC mechanism on both the bit toggle count and compression ratio, it is also important to see how the results vary for individual applications. To perform this deeper analysis, we pick one compression algorithm (*C-Pack*), and a single subgroup of applications (*Open-Source* from different application suites: *CUDA*, *lonestar*, *Mars*, and *rodinia*), and show the effect of compression with and without EC on the toggle count (Figure 12) and compression ratio (Figure 13). We also study two versions of the EC mechanism: (i) *EC1* which uses the $Energy \times Delay$ metric and (ii) *EC2* which uses the $Energy \times Delay^2$ metric. We make three major observations from these figures.

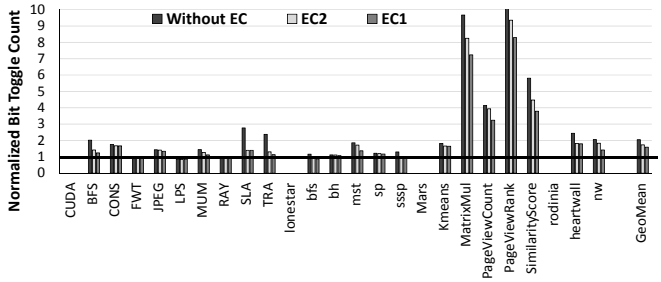

Figure 12: Effect of Energy Control with C-Pack compression algorithm on bit toggle count on the DRAM bus.

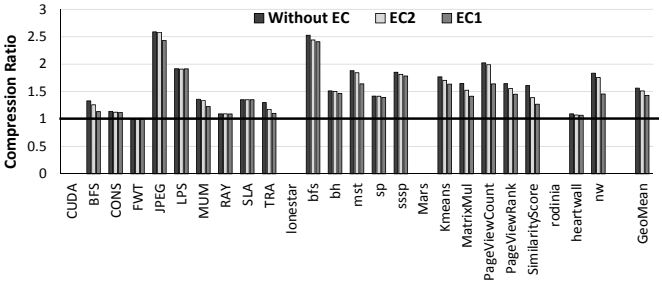

Figure 13: Effect of Energy Control on compression ratio on the DRAM bus.

[7] Compression ratio reduces because EC decides to transfer some compressible lines in the uncompressed form.

First, both the increase in bit toggle count and compression ratio vary significantly for different applications. For example, *bfs* from the Lonestar application suite has a very high compression ratio of more than 2.5×, but its increase in bit toggle count is relatively small (only 17% for baseline C-Pack compression without EC mechanism). In contrast, *PageViewRank* application from the Mars application suite has more than 10× increase in bit toggle count with a 1.6× compression ratio. This is because different data is affected differently from data compression. There can be cases where the overall bit toggle count is slightly lower than in the uncompressed baseline even without our EC mechanism (e.g., *LPS*).

Second, for most of the applications in our workload pool, the proposed mechanisms (EC1 and EC2) can significantly reduce the bit toggle count while retaining most of the benefits of compression. For example, for *heartwall* our EC2 mechanism reduces the bit toggle count from 2.5× to 1.8× by sacrificing only 8% of the compression ratio (from 1.83× to 1.75×). This could significantly reduce the energy overhead of the C-Pack algorithm while preserving most of its bandwidth (and thus, likely performance) benefits.

Third, as expected, EC1 is more aggressive in disabling compression, because it weighs bit toggles and compression ratio equally in the trade-off, while in the EC2 mechanism, compression ratio has higher weight (squared in the formula) than bit toggle count. Hence, for many of our applications (e.g., *bfs*, *mst*, *Kmeans*, *nw*, etc.) we see a gradual reduction in bit toggle count, with a corresponding small reduction in compression ratio, when moving from baseline to EC1 and then EC2. This means that depending on the application characteristics, we have multiple options with varying aggressiveness to trade off between bit toggle count with compression ratio. As we will show in the next section, we can achieve these trade-offs with minimal effect on performance.

7.1.2. Effect on Performance. While previous results show that EC1 and EC2 mechanisms are very effective in trading off bit toggle count with compression ratio, it is still important to understand how much this trade-off "costs" in actual performance. This is especially important for the DRAM bus, which is a major bottleneck in many GPU applications' performance, and hence even a minor degradation in compression ratio can potentially lead to a noticeable degradation in performance and overall energy consumption. Figure 14 shows the effect of C-Pack compression algorithm of both EC1 and EC2 mechanisms on performance in comparison to a mechanism without energy control (results are normalized to the performance of the uncompressed baseline). We make two observations.

First, our proposed mechanisms (EC1 and EC2) usually have minimal negative impact on application performance. The baseline C-Pack algorithm (*Without EC*) provides 11.5% average performance improvement, while the least aggressive EC2 mechanism reduces the performance benefit by only 0.7%, and the EC1 mechanism - by only 2.0%. This is significantly

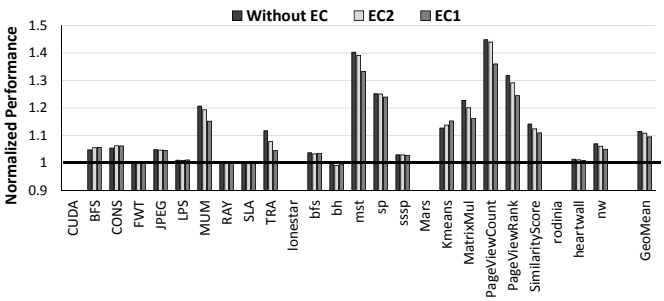

Figure 14: Effect of Energy Control on performance with the C-Pack compression algorithm.

smaller than the corresponding loss in compression ratio (shown in Figure 13). The primary reason is a successful trade-off between compression ratio, bit toggle count and performance. Both EC mechanisms consider current DRAM bandwidth utilization, and only trade off compression when it is unlikely to hurt performance.

Second, while there are applications (e.g., *MatrixMul*) where we could lose up to 6% performance using the most aggressive mechanism (EC1). Such a loss is likely justified because we also reduce the bit toggle count from almost 10× to about 7×. It is hard to avoid any degradation in performance for such applications since they are severely bandwidth-limited, and any loss in compression ratio is reflected conspicuously in performance. If such performance degradation is unacceptable, then a less aggressive version of the EC mechanism, EC2, can be used. Overall, we conclude that our proposed mechanisms EC1 and EC2 are both very effective in preserving most of the performance benefit of data compression while significantly reducing it negative effect of bit toggle count (and hence the energy overhead of date compression).

7.1.3. Effect on DRAM and System Energy. Figure 15 shows the effect of the C-Pack compression algorithm on the DRAM energy consumption with and without energy control (normalized to the energy consumption of the uncompressed baseline). These results include the overhead of the compression/decompression hardware [14] and our mechanism (Section 5.4). We make two observations. First, as expected, many applications' energy consumption significantly reduces with EC (e.g., *SLA*, *TRA*, *heartwall*, *nw*). For example, for *TRA*, the 28.1% reduction in the DRAM energy (8.9% reduction in total system energy) is the direct effect of the significant reduction in bit toggle count (from 2.4× to 1.1× as shown in Figure 12). Overall, DRAM energy is reduced by 8.3% for both EC1 and EC2. As DRAM energy constitutes on average 28.8% out of total system energy (ranging from 7.9% to 58.3%), and the decrease in performance is less than 1%, this leads to a total system energy reduction of 2.1% on average across all applications using our EC1/EC2 mechanisms.

Second, many applications that have significant growth in their bit toggle count due to compression (e.g., *MatrixMul* and *PageViewRank*) are also very sensitive to the available

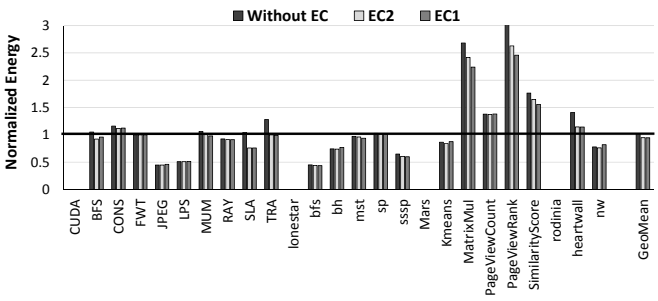

Figure 15: Effect of Energy Control on the DRAM energy the with C-Pack compression algorithm.

DRAM bandwidth. Therefore, to provide energy savings for these applications, it is very important to dynamically monitor their current bandwidth utilization. We observe that without the integration of the *current bandwidth utilization* metric into our mechanisms (described in Section 4.2), even a minor reduction in compression ratio for these applications could lead to a severe degradation in performance, and system energy.

We conclude that our proposed mechanisms can efficiently trade off compression ratio and bit toggle count to improve both the DRAM and overall system energy.

7.2. On-Chip Interconnect Results

7.2.1. Effect on Bit Toggle Count and Compression Ratio. Similar to the off-chip bus, we evaluate the effect of five compression algorithms on bit toggle count (Figure 16) and compression ratio (Figure **??**) for the on-chip interconnect using GPGPU-sim and open-source applications as described in Section 6. We make three major observations from these figures.

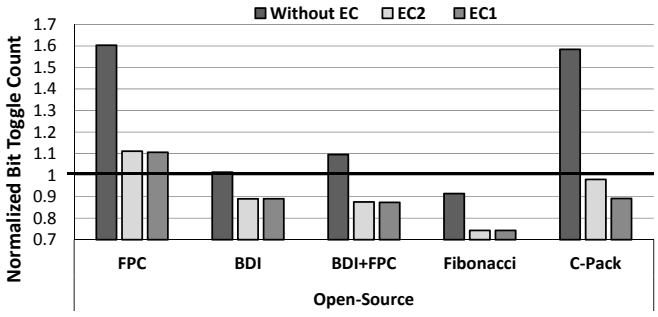

Figure 16: Effect of Energy Control on bit toggle count on the on-chip interconnect.

First, the most noticeable difference when compared with the DRAM bus is that the increase in bit toggle count is not as significant for all compression algorithms. Bit toggle count still increases for all but one algorithm (*Fibonacci*), but we observe steep increases in bit toggle count (e.g., around 60%) only for the FPC and C-Pack algorithms. The reason for this behavior is twofold. First, the on-chip data and its access pattern are different from those of the off-chip data for some applications, and hence these data sets have different

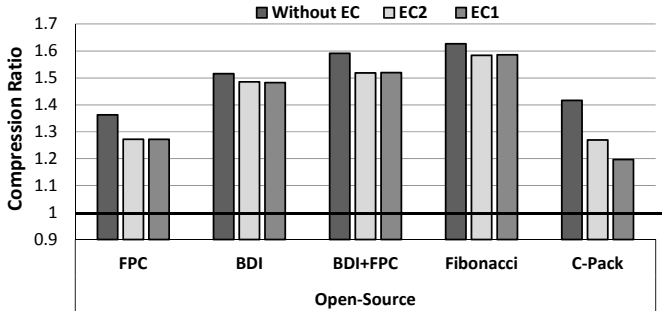

Figure 17: Effect of Energy Control on compression ratio on the on-chip interconnect.

characteristics. Second, the definition of *bit toggles* is different for these two channels (as discussed in Section 2).

Second, despite the variation in how different compression algorithms affect the bit toggle count, both of our proposed mechanisms are effective in reducing the bit toggle count (e.g., from 1.6× to 0.9× with C-Pack). Moreover, both mechanisms, EC1 and EC2, preserve most of the compression ratio achieved by the C-Pack algorithm. Therefore, we conclude that our proposed mechanisms are effective in reducing bit toggles for both on-chip interconnect and off-chip buses.

Third, in contrast to our evaluation of the DRAM bus, our results with the interconnect show that for all but one algorithm (C-Pack), both EC1 and EC2 are almost equally effective in reducing the bit toggle count while preserving the compression ratio. This means that in the on-chip interconnect, there is no need to use more aggressive decision functions to trade off bit toggles with compression ratio, because the EC2 mechanism—the less aggressive of the two—already provides most of the benefit.

Finally, while the overall achieved compression ratio is slightly lower than in DRAM, we still observe impressive compression ratios in on-chip interconnect, reaching up to 1.6× on average across all open-source applications. While DRAM bandwidth traditionally is a primary performance bottleneck for many applications, on-chip interconnect is usually designed such that its bandwidth will not be the primary performance limiter. Therefore, the achieved compression ratio in the on-chip interconnect is expected to translate directly into overall area and silicon cost reduction, assuming fewer ports, wires and switches are required to provide the same effective bandwidth. Alternatively, the compression ratio can translate into lower power and energy by using lower clock frequency due to lower bandwidth demands the from on-chip interconnect.

7.2.2. Effect on Performance and Interconnect Energy. While it is clear that both EC1 and EC2 are effective in reducing the bit toggle count, it is important to understand how they affect performance and interconnect energy in our simulated system. Figure 18 shows the effect of both techniques on performance (normalized to the performance of the uncompressed baseline). The key takeaway from this figure is that for all compression algorithms, both EC1 and EC2 are

within less than 1% of the performance of the designs without the energy control mechanisms. There are two reasons for this. First, both EC1 and EC2 are effective in deciding when compression is useful to improve performance and when it is not. Second, the on-chip interconnect is less of a bottleneck in our example configuration than the off-chip DRAM bus. Hence, disabling compression, in some cases, has smaller impact on overall performance.

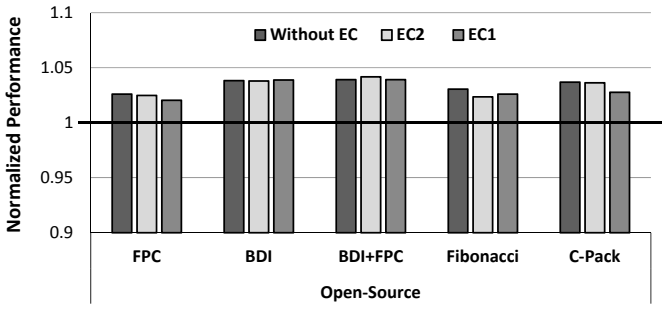

Figure 18: Effect of Energy Control on performance when compression is applied to the on-chip interconnect.

Figure 19 shows the effect of data compression and bit toggling on the energy consumed by the on-chip interconnect. Results are normalized to the energy of the uncompressed interconnect. As expected, compression algorithms that have a higher bit toggle count have much higher energy cost to support data compression, because bit toggle count is the dominant part of the on-chip interconnect energy consumption. From this figure, we observe that our proposed mechanisms, EC1 and EC2, are both effective in reducing the energy overhead of compression. The most notable reduction is for the *C-Pack* algorithm, where we reduce the overhead from 2.1× to only 1.1×.

We conclude that our mechanisms are effective in reducing the energy overhead due to higher bit toggle count caused by compression, while preserving most of the bandwidth and performance benefits achieved via compression.

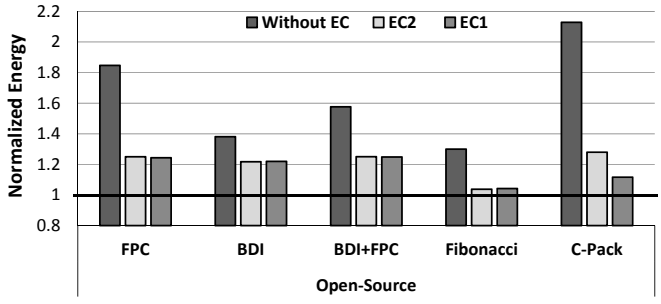

Figure 19: Effect of Energy Control on on-chip interconnect energy.

7.3. Effect of Metadata Consolidation

Metadata Consolidation (MC) is able to reduce the bit-level misalignment for several compression algorithms. We currently implemented MC for FPC and C-Pack compression algorithms. We observe additional toggle reduction on the *DRAM bus* from applying MC (over EC2) of 3.2% and 2.9% for FPC and C-Pack respectively across the discrete and mobile GPU applications. Even though MC can mitigate some negative effects of bit-level misalignment after compression, it is not effective in cases where data values within the cache line are compressed to different sizes. These variable sizes frequently lead to misalignment at the byte granularity. While it is possible to insert some amount of padding into the compressed line to reduce the misalignment, this would counteract the primary goal of compression to minimize data size.

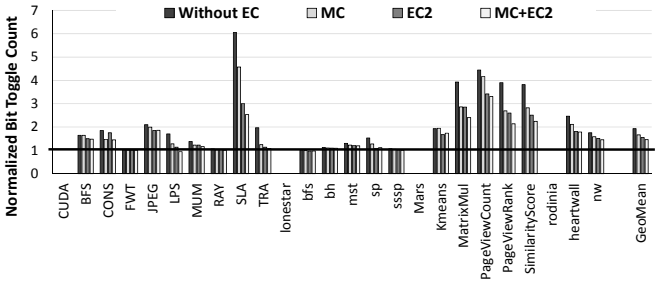

Figure 20: Effect of Metadata Consolidation on DRAM bus bit toggle count with the FPC compression algorithm.

We also conducted an experiment with open-source applications where we compare the impact of MC and EC separately, as well as together, for the FPC compression algorithm (Figure 20). We observe similar results with the C-Pack compression algorithm. We make two observations from Figure 20. First, when EC is not employed, MC substantially reduces the bit toggle count, from 1.93× to 1.66× on average. Hence, if the hardware changes due to EC are undesirable, MC can be used to avoid some of the increase in the bit toggle count. Second, when energy control is employed (*EC2* and *MC+EC2*), the additional reduction in bit toggle count is relatively small. This means that the EC2 mechanism provides most of the benefits that MC can provide.

We conclude that Metadata Consolidation is effective in reducing the bit toggle count when energy control is not used. It does not require significant hardware changes other than the minor modifications in the compression algorithm itself. At the same time, in the presence of Energy Control, the additional effect of MC in bit toggle count reduction is small.

8. Related Work

To our knowledge, this is the first work that (i) identifies increased bit toggle count in communication channels as a major drawback in enabling efficient data compression in modern systems, (ii) evaluates the impact and causes for this inefficiency in modern GPU architectures for different channels across multiple compression algorithms, and (iii) proposes and extensively evaluates different mechanisms to mitigate this effect to improve overall energy efficiency. We first discuss prior works that (i) propose more energy efficient designs for DRAM and interconnects, (ii) mechanisms for energy efficient data communication in on-chip/off-chip buses and other communication channels. We then discuss prior

works that aim to address different challenges in efficiently applying data compression.

Low Power DRAM and Interconnects. A wide range of previous works propose mechanisms and architectures to enable more energy-efficient operation of DRAM. Examples of these proposals include activating fewer bitlines [67], using shorter bitlines [37], adapting latency to common-case operation conditions [36] and workload characteristics [23], more intelligent refresh policies [40, 42, 49, 3, 34, 54, 68, 33], dynamic voltage and frequency scaling [16], energy-efficient data movement mechanisms [59, 12] and better management of data placement [75, 39, 41]. For interconnects, Balasubramonian et al. [8], Mishra et al. [45], and Grot et al. [21, 20] propose heterogeneous interconnects comprising wires or network designs with different latency, bandwidth, and power characteristics for better performance and energy efficiency. Previous works also propose different schemes to enable and exploit *low-swing* interconnects [73, 66, 9] where reduced voltage swings during signalling enables better energy efficiency. These works do not consider energy efficiency in the context of data compression and are usually data-oblivious, hence the proposed solutions cannot alleviate the negative impact of increased bit toggle count with data compression.

Energy-Efficient Encoding Schemes. *Data Bus Inversion (DBI)* is an encoding technique proposed to enable energy efficient data communication. Widely used DBI algorithms include *bus invert coding* [63] and *limited-weight coding* [61, 62] which selectively invert all the bits within a fixed granularity to either reduce the number of bit flips along the communication channel or reduce the frequency of either 0's or 1's when transmitting data. Recently, *DESC* [10] was proposed in the context of on-chip interconnects to reduce power consumption by representing information by the delay between two consecutive pulses on a set of wires, thereby reducing the bit toggle count. Jacobvitz et al. [27] applied *coset coding* to reduce the number of bit flips while writing to memory by mapping each dataword into a larger space of potential encodings. These encoding techniques do not tackle the excessive bit toggle count generated by data compression and are largely orthogonal to our mechanisms for toggle-aware data compression. Note that our mechanisms for the DRAM bus assume DBI for the baseline, and hence our improvements are on top of a baseline that employs DBI.

Efficient Data Compression. Several prior works [64, 5, 58, 51, 60, 2, 50] study main memory and cache compression with several different compression algorithms [4, 52, 14, 57, 6]. These works exploit the capacity and bandwidth benefits of data compression to enable higher performance and energy efficiency. They primarily tackle improving compression ratios, reducing the performance/energy overheads of processing data for compression/decompression, or propose more efficient architectural designs to integrate data compression. These works address different challenges in data compression and are orthogonal to our proposed toggle-aware compression mechanisms. To our knowledge, this is the first work to study the energy implications of bit toggle count increase when transferring compressed data over different on-chip/off-chip channels.

9. Conclusion

We observe that data compression, while very effective in improving bandwidth efficiency in GPUs, can greatly increase the bit toggle count in the on-chip/off-chip interconnect. Based on this new observation, we develop two new *toggle-aware compression* techniques to reduce bit toggle count while preserving most of the bandwidth reduction benefits of compression. Our evaluations across six compression algorithms and 242 workloads show that these techniques are effective as they greatly reduce the bit toggle count while retaining most of the bandwidth reduction advantages of compression. We conclude that toggle-awareness is an important consideration in data compression mechanisms for modern GPUs (and likely CPUs as well), and encourage future work to develop new solutions for it.

Acknowledgments

We thank the reviewers for their valuable suggestions. We thank Mike O'Connor, Wishwesh Gandhi, Jeff Pool, Jeff Boltz, and Lacky Shah from NVIDIA and Suvinay Subramanian from MIT for their helpful comments during the early steps of this project. We thank the SAFARI group members for the feedback and stimulating research environment they provide. Gennady Pekhimenko is supported by a NVIDIA Graduate Fellowship. This research was supported by NSF grants 1212962, 1320531, 1409723, 1423172, and in part by the United States Department of Energy. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

[1] "NVIDIA GeForce GTX 980 Review," http://www.anandtech.com/show/8526/nvidia-geforce-gtx-980-review/3.
[2] B. Abali *et al.*, "Memory Expansion Technology (MXT): Software Support and Performance," *IBM J.R.D.*, 2001.
[3] J.-H. Ahn *et al.*, "Adaptive self refresh scheme for battery operated high-density mobile DRAM applications," in *ASSCC*, 2006.
[4] A. R. Alameldeen and D. A. Wood, "Adaptive Cache Compression for High-Performance Processors," in *ISCA*, 2004.
[5] A. R. Alameldeen and D. A. Wood, "Interactions Between Compression and Prefetching in Chip Multiprocessors," in *HPCA*, 2007.
[6] A. Arelakis and P. Stenstrom, "SC2: A Statistical Compression Cache Scheme," in *ISCA*, 2014.
[7] A. Bakhoda *et al.*, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *ISPASS*, 2009.
[8] R. Balasubramonian *et al.*, "Microarchitectural Wire Management for Performance and Power in Partitioned Architectures," in *HPCA*, 2005.
[9] B. M. Beckmann and D. A. Wood, "TLC: Transmission line caches," in *MICRO*, 2003.
[10] M. N. Bojnordi and E. Ipek, "DESC: Energy-efficient Data Exchange Using Synchronized Counters," in *MICRO*, 2013.
[11] M. Burtscher *et al.*, "A quantitative study of irregular programs on GPUs," in *IISWC*, 2012.

[12] K. K. Chang *et al.*, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.
[13] S. Che *et al.*, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *IISWC*, 2009.
[14] X. Chen *et al.*, "C-pack: A high-performance microprocessor cache compression algorithm," *TVLSI*, 2010.
[15] R. Das *et al.*, "Performance and power optimization through data compression in Network-on-Chip architectures," in *HPCA*, 2008.
[16] H. David *et al.*, "Memory power management via dynamic voltage/frequency scaling," in *ICAC*, 2011.
[17] J. Dusser *et al.*, "Zero-content Augmented Caches," in *ICS*, 2009.
[18] M. Ekman and P. Stenstrom, "A Robust Main-Memory Compression Scheme," in *ISCA*, 2005.
[19] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *JSCC*, 1996.
[20] B. Grot *et al.*, "Express Cube Topologies for on-Chip Interconnects," in *HPCA*, 2009.
[21] B. Grot *et al.*, "Kilo-NOC: A Heterogeneous Network-on-chip Architecture for Scalability and Service Guarantees," in *ISCA*, 2011.
[22] E. G. Hallnor and S. K. Reinhardt, "A Unified Compressed Memory Hierarchy," in *HPCA*, 2005.
[23] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[24] B. He *et al.*, "Mars: A MapReduce Framework on Graphics Processors," in *PACT*, 2008.
[25] Hybrid Memory Cube Consortium, *HMC Specification 1.1*, Feb. 2014.
[26] Hynix. Hynix GDDR5 SGRAM Part H5GQ1H24AFR Revision 1.0. [Online]. Available: {http://www.hynix.com/datasheet/pdf/graphics/H5GQ1H24AFR(Rev1.0).pdf}
[27] A. Jacobvitz *et al.*, "Coset coding to extend the lifetime of memory," in *HPCA*, 2013.
[28] JEDEC, "DDR4 SDRAM Standard," 2012.
[29] JEDEC, *JESD235 High Bandwidth Memory (HBM) DRAM*, Oct. 2013.
[30] JEDEC, "Standard No. 79-3F. DDR3 SDRAM Specification, July 2012." July 2009.
[31] A. Jog *et al.*, "OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance," in *ASPLOS*, 2013.
[32] A. Jog *et al.*, "Orchestrated Scheduling and Prefetching for GPGPUs," in *ISCA*, 2013.
[33] S. KHan *et al.*, "The efficacy of error mitigation techniques for DRAM retention failures: a comparative experimental study," in *SIGMETRICS*, 2014.
[34] J. Kim and M. C. Papaefthymiou, "Dynamic memory design for low data-retention power," in *PATMOS*, 2000.
[35] D. Lee *et al.*, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," in *TACO*, 2016.
[36] D. Lee *et al.*, "Adaptive-latency DRAM: Optimizing DRAM timing for the common-case," in *HPCA*, 2015.
[37] D. Lee *et al.*, "Tiered-latency DRAM: A low latency and low cost DRAM architecture," in *HPCA*, 2013.
[38] J. Leng *et al.*, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *ISCA*, 2013.
[39] C.-H. Lin *et al.*, "PPT: Joint Performance/Power/Thermal Management of DRAM Memory for Multi-core Systems," in *ISLPED*, 2009.
[40] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[41] S. Liu *et al.*, "Hardware/software techniques for DRAM thermal management," in *HPCA*, 2011.
[42] S. Liu *et al.*, "Flikker: saving DRAM refresh-power through critical data partitioning," in *ASPLOS*, 2011.
[43] G. H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in *ISCA*, 2008.
[44] Micron, "DDR3 SDRAM System-Power Calculator," 2010.
[45] A. K. Mishra *et al.*, "A heterogeneous multiple network-on-chip design: an application-aware approach," in *DAC*, 2013.
[46] P. Mosalikanti *et al.*, "High performance DDR architecture in Intel Core processors using 32nm CMOS high-K metal-gate process," in *VLSI-DAT*, 2011.
[47] V. Narasiman *et al.*, "Improving GPU Performance via Large Warps and Two-level Warp Scheduling," in *MICRO*, 2011.
[48] NVIDIA, "CUDA C/C++ SDK Code Samples," 2011.
[49] T. Ohsawa *et al.*, "Optimizing the DRAM refresh count for merged DRAM/logic LSIs," in *ISLPED*, 1998.
[50] G. Pekhimenko *et al.*, "Exploiting Compressed Block Size as an Indicator of Future Reuse," in *HPCA*, 2015.
[51] G. Pekhimenko *et al.*, "Linearly Compressed Pages: A Low Complexity, Low Latency Main Memory Compression Framework," in *MICRO*, 2013.
[52] G. Pekhimenko *et al.*, "Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches," in *PACT*, 2012.
[53] J. Pool *et al.*, "Lossless Compression of Variable-precision Floating-point Buffers on GPUs," in *Interactive 3D Graphics and Games*, 2012.
[54] M. K. Qureshi *et al.*, "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems," in *DSN*, 2015.
[55] T. G. Rogers *et al.*, "Cache-Conscious Wavefront Scheduling," in *MICRO*, 2012.
[56] S. Sardashti *et al.*, "Skewed Compressed Caches," in *MICRO*, 2014.
[57] S. Sardashti and D. A. Wood, "Decoupled Compressed Cache: Exploiting Spatial Locality for Energy-optimized Compressed Caching," in *MICRO*, 2013.
[58] V. Sathish *et al.*, "Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads," in *PACT*, 2012.
[59] V. Seshadri *et al.*, "RowClone: Fast and Energy-efficient in-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[60] A. Shafiee *et al.*, "MemZip: Exploring Unconventional Benefits from Memory Compression," in *HPCA*, 2014.
[61] M. R. Stan and W. P. Burleson, "Limited-weight codes for low-power I/O," in *International Workshop on Low Power Design*, 1994.
[62] M. R. Stan and W. P. Burleson, "Coding a terminated bus for low power," in *Proceedings of Fifth Great Lakes Symposium on VLSI*, 1995.
[63] M. Stan and W. Burleson, "Bus-invert Coding for Low-power I/O," *IEEE Transactions on VLSI Systems*, vol. 3, no. 1, pp. 49–58, March 1995.
[64] M. Thuresson *et al.*, "Memory-Link Compression Schemes: A Value Locality Perspective," in *TOC*, 2008.
[65] A. Udipi *et al.*, "Non-uniform power access in large caches with low-swing wires," in *HiPC*, 2009.
[66] A. N. Udipi *et al.*, "Non-uniform power access in large caches with low-swing wires," in *HiPC*, 2009.
[67] A. N. Udipi *et al.*, "Rethinking DRAM design and organization for energy-constrained multi-cores," in *ISCA*, 2010.
[68] R. Venkatesan *et al.*, "Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM," in *HPCA*, 2006.
[69] N. Vijaykumar *et al.*, "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps," in *ISCA*, 2015.
[70] A. X. Widmer and P. A. Franaszek, "A DC-balanced, partitioned-block, 8B/10B transmission code," *IBM Journal of Research and Development*, 1983.
[71] J. Yang *et al.*, "Frequent Value Compression in Data Caches," in *MICRO*, 2000.
[72] G. L. Yuan *et al.*, "Complexity effective memory access scheduling for many-core accelerator architectures," in *MICRO*, 2009.
[73] H. Zhang and J. Rabaey, "Low-swing interconnect interface circuits," in *ISPLED*, 1998.
[74] J. Zhao *et al.*, "Buri: Scaling Big-memory Computing with Hardware-based Memory Expansion," *TACO*, 2015.
[75] Q. Zhu *et al.*, "Thermal management of high power memory module for server platforms," in *ITHERM*, 2008.
[76] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE TOIT*, 1977.