Toggle-Aware Compression for GPUs

Gennady Pekhimenko† Evgeny Bolotin⋆ Mike O'Connor⋆# Onur Mutlu† Todd C. Mowry† Stephen W. Keckler⋆#
†Carnegie Mellon University ⋆NVIDIA #University of Texas at Austin

Abstract—Memory bandwidth compression can be an effective way to achieve higher system performance and energy efficiency in modern data-intensive applications by exploiting redundancy in data. Prior works studied various 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). These works addressed two common shortcomings of compression: (i) compression/decompression overhead in terms of latency, energy, and area, and (ii) hardware complexity to support variable data size. In this paper, we make the new observation that there is another important problem related to data compression in the context of the communication energy efficiency: transferring compressed data leads to a substantial increase in the number of *bit toggles* (communication channel switchings from 0 to 1 or from 1 to 0). This, in turn, increases the dynamic energy consumed by on-chip and off-chip buses due to more frequent charging and discharging of the wires. Our results, for example, show that the bit toggle count increases by an average of 2.2× with some compression algorithms across 54 mobile GPU applications. We characterize and demonstrate this new problem across a wide variety of 221 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 six data compression algorithms we examine, while keeping most of their bandwidth reduction benefits.

1 Introduction and Background

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: improving 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/off-chip bandwidth consumption, while simultaneously avoiding negative effects on other key parameters, such as overall system cost, energy, and latency characteristics. One potential way of addressing multiple of the described constraints is to employ dedicated hardware-based *data compression* mechanisms (e.g., [27], [2], [7], [18], [4]) across various data links in the system. Compression exploits the high data redundancy observed in many modern applications [18], [20], [4], [26]. It can be used to improve both capacity (e.g., of caches, DRAM, non-volatile memories [27], [2], [7], [18], [4], [17], [22], [16], [26]) and bandwidth utilization (e.g., of on-chip and off-chip interconnects [8], [3], [24], [21], [17], [22], [26]). Several recent works focus on bandwidth compression to decrease memory traffic by transmitting data in a compressed form in both CPUs [17], [24], [3] and GPUs [21], [17], [26], which results in better system performance and energy consumption. Bandwidth compression proves to be particularly effective in GPUs because GPUs are often bottlenecked by memory bandwidth [15], [14], [13], [28], [26]. GPU applications also exhibit high degrees of data redundancy [21], [17], [26], leading to good compression ratios.

1.1 Why Data Compression Can Be Energy-inefficient

The benefits of data compression are well-studied. There are also two well-known overheads of data compression: (1) compression/decompression overhead [2], [18] in terms of latency, energy, and area, and (2) complexity/cost to support variable data sizes [12], [20], [17], [22]. Both problems have solutions: e.g., Base-Delta-Immediate compression [18] provides a low-latency, low-energy hardware-based compression algorithm, and Decoupled Compressed Cache [20] provides a mechanism to manage data recompaction and fragmentation in compressed caches.

In this paper, we make the new observation that there is yet another important problem with data compression that needs to be addressed in the context of communication channels: transferring data in compressed form (as opposed to in 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 leads to a higher switching activity [25], [5], [6] of wires, leading to higher dynamic energy consumed by on-chip or off-chip interconnects. We identify two reasons for the increase in bit toggle count: (i) higher per-bit entropy of compressed data (the same amount of information is now stored in fewer bits; hence, the "randomness" of a single bit grows), and (ii) variable-size nature of compressed data, which 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 tradeoff: energy savings due to reduced bandwidth requirements versus energy loss due to higher switching energy during compressed data transfers. This observation and the corresponding tradeoff are the key contributions of this work.

To understand (1) how applicable general-purpose data compression is for real GPU applications, and (2) how severe the problem we identify is, we analyze 221 discrete and mobile graphics application traces from a major GPU vendor using six compression algorithms. 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 [7] across 54 mobile GPU applications), it also greatly increases the bit toggle count (e.g., 2.2× average corresponding increase). This effect, in turn, can significantly increase the energy dissipated in the on-chip/off-chip interconnects, which constitute a significant portion of the memory subsystem energy.[1]

1.2 Our Approach: Toggle-Aware Compression

In this work, we develop two new techniques that make bandwidth compression more energy-efficient by limiting the overall increase in bit toggles due to compression. Our first technique, *Energy Control (EC)*, decides whether it is better to send data in compressed or uncompressed form, based on a model that takes into account the compression ratio and the increase in bit toggles. 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, and $Delay$ is inversely proportional to the compression ratio. Our second technique, *Metadata Consolidation (MC)*, reduces the negative effects of scattering of metadata in a compressed cache line, which happens with many existing compression algorithms [2], [7], by consolidating 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) [2] and Base-Delta-Immediate (BDI) compres-

[1]For example, up to 80% energy of the LLC caches is H-tree capacitance interconnects [6].

sion [18]), different communication channels (e.g., on-chip and off-chip buses), and, potentially, different architectures (e.g., GPUs, CPUs, and hardware accelerators). We demonstrate that our proposed mechanisms are also largely orthogonal to different data encoding schemes also used to minimize the bit toggle count (e.g., Data Bus Inversion [23]), and hence can be used together with them to enhance the energy efficiency of interconnects.

2 Motivation, Problem, 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) [2]; (ii) *BDI* (Base-Delta-Immediate Compression) [18]; (iii) *BDI+FPC* (combined FPC and BDI) [17]; (iv) *LZSS* (Lempel-Ziv compression) [29], [1]; (v) *Fibonacci* (a graphics-specific compression algorithm) [19]; and (vi) *C-Pack* [7]. To ensure our conclusions are more practically applicable, we analyze real GPU applications with actual data sets provided by a major GPU vendor.

Figure 1 shows the potential 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 handled by modern GPUs efficiently, and assume practical boundaries on bandwidth compression ratios (e.g., the highest possible compression ratio is 4.0, because the minimum flit size is 32 bytes and packet size is 128 bytes).

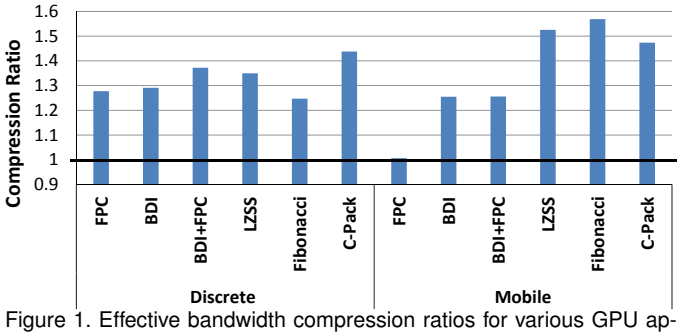

Figure 1. Effective bandwidth compression ratios for various GPU applications and compression algorithms (higher bars are better).

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). Especially interesting is that simple compression algorithms are very competitive with the more complex GPU-oriented *Fibonacci* algorithm and the software-based Lempel-Ziv algorithm [29]. Second, for the 54 mobile GPU applications (right side of Figure 1), bandwidth benefits are even more pronounced (reaching up to 57% on average with the Fibonacci algorithm). 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 both discrete and mobile GPU compute applications.

Unfortunately, the benefits of compression come with additional costs. Two overheads of compression are well-known: (i) additional data processing due to compression/decompression, and (ii) hardware changes to transfer variable-length cache lines. While these two problems are significant, multiple compression algorithms [2], [27], [18], [9] were proposed to minimize the overheads of data compression/decompression. Several designs [22], [21], [17], [26] were proposed to integrate bandwidth compression into existing memory hierarchies. In this work, we find 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 [25], [5], [6], 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 a given data. As a result, energy consumed for moving this data can change. Figure 2 shows the increase in bit toggle count for discrete and mobile GPU applications with the six compression algorithms over a baseline that does not employ compression. 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).

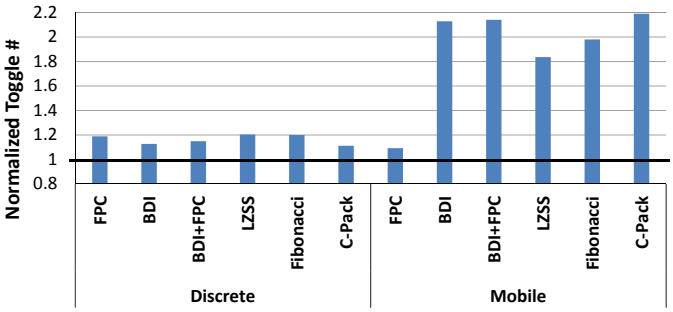

Figure 2. Bit toggle count increase due to compression.

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 toggle rates (31% on average across discrete applications) and is less amenable to compression. Second, the increase in bit toggle count is more dramatic for mobile applications (right half of Figure 2), exceeding 1.8× for all algorithms but one. The FPC algorithm is not as effective in compressing mobile application data in our pool, and hence does not greatly affect bit toggle count. In both 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 effectiveness of compression and the resultant 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. Clearly, there is a positive correlation between the compression ratio and the increase in bit toggle count. We observe similarly-shaped curves for other compression algorithms. This strongly suggests that successful compression can lead to increased dynamic energy dissipation by on-chip/off-chip communication channels due to increased toggle counts.

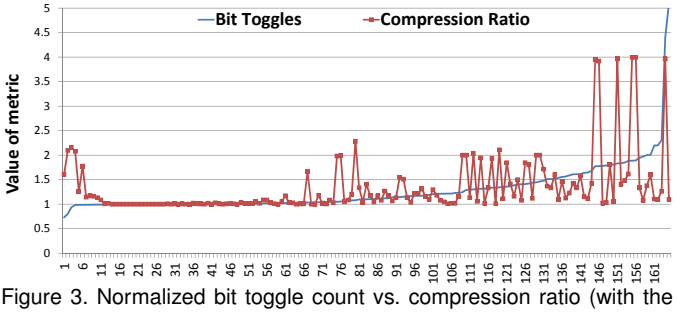

Figure 3. Normalized bit toggle count vs. compression ratio (with the FPC algorithm) for each of the discrete GPU applications.

To understand this phenomenon, 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, 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 low number of toggles (2 toggles per 8-byte flit). This low 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 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

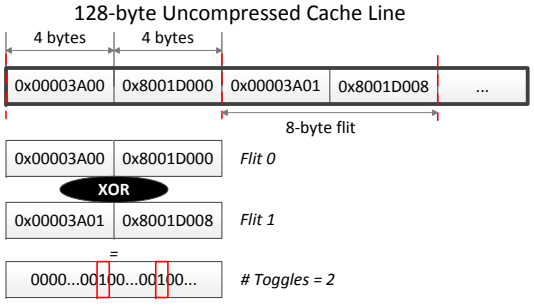

Figure 4. Bit toggles without compression.

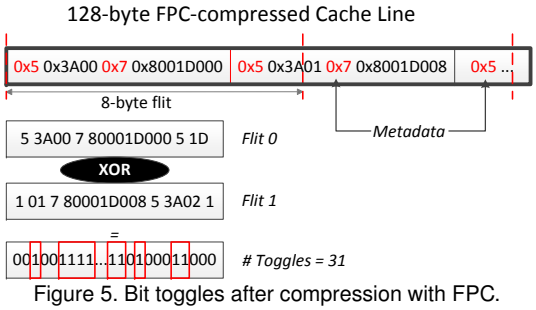

Figure 5. Bit toggles after compression with FPC.

entropy leads to higher "randomness" in data and, thus, a larger toggle count. Second, compression negatively affects the alignment of data both at the byte granularity (narrow values 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).

3 Toggle-Aware Bandwidth Compression

3.1 Energy vs. Performance Tradeoff

Data compression can reduce energy consumption and improve performance by reducing communication 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 tradeoff, we examine commonly-used metrics like $Energy \times Delay$ and $Energy \times Delay^2$ [10]. We estimate these metrics with a simple model, which aids us in making compression-related performance/energy tradeoffs.[2] 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). Based on the observations above, we have developed two techniques to enable *toggle-aware compression* to reduce the negative effects of increased bit toggle count.

3.2 Energy Control (EC)

We propose a generic *Energy Control* (EC) mechanism that can be applied along with any current (or future) compression algorithm.[3] 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), and (iii) compression ratio (CR) to decide whether to transmit the data compressed or uncompressed. We can calculate the toggle count very energy efficiently (by expending 4pJ per 128-byte cache line with 32-byte flits, based on initial results from our Verilog implementation). Using this approach, it is possible to achieve a desirable tradeoff 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 $Energy \times Delay^2$). The decision function can also use as inputs other metrics, including application bandwidth requirements, power limitations, available voltage/frequency scaling options, and other system power management opportunities. We leave detailed explorations to future work.

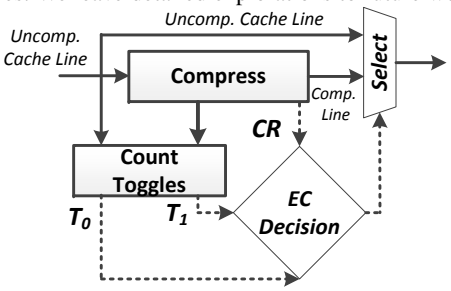

Figure 6. Energy Control decision mechanism.

3.3 Metadata Consolidation

Traditional energy-oblivious compression algorithms are not optimized to minimize the bit toggle count. However, it is possible to extend existing 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. Recall that one major reason for increased bit toggle count was the misalignment of compressed data with flit width due to alignment issues caused by compression-related metadata (Section 2). 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 toggle count increase can be avoided.

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

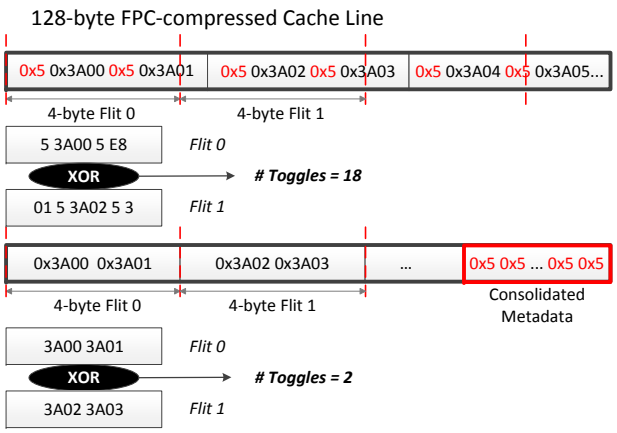

Figure 7. Bit toggle count w/o and with Metadata Consolidation.

4 Evaluation

4.1 Methodology

We analyze 221 memory traces from discrete (167) and mobile (54) GPU applications. We quantify *bit toggle count*, which reflects energy consumption, and *compression ratio*, which serves as a proxy for bandwidth consumption (and also a proxy for performance, for bandwidth-limited applications). Different data encoding techniques (e.g., DBI [23] or DESC [6]) can be applied to decrease the baseline bit toggle count of any data transfer (uncompressed or compressed). We find that the benefits of these data encoding techniques are largely

[2] We are currently working on verifying our estimations by implementing our techniques in the open-source simulator GPGPU-Sim [11].

[3] In this work, we assume that only memory bandwidth is compressed, while on-chip caches still store data in uncompressed form.

orthogonal to whether or not data compression is used. We use DBI [23] as part of our baseline, for transferring both compressed and uncompressed data.

4.2 Effect of Energy Control

Figure 8 and Figure 9) show, respectively, the normalized bit toggle count and compression ratio with and without *Energy Control* (using various compression algorithms). Results are normalized to a system that does not employ data compression and averaged across all discrete and mobile workloads. The number on top of bars indicates the reduction EC (right bar for each compression algorithm) provides over the system that employs compression (left bar). The EC decision function based on the $Energy \times Delay^2$ metric (Section 3.2).

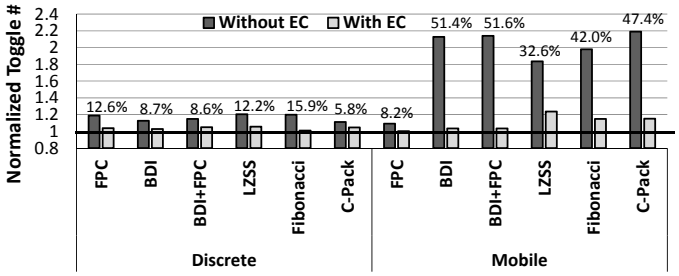

Figure 8. Effect of Energy Control on bit toggle count.

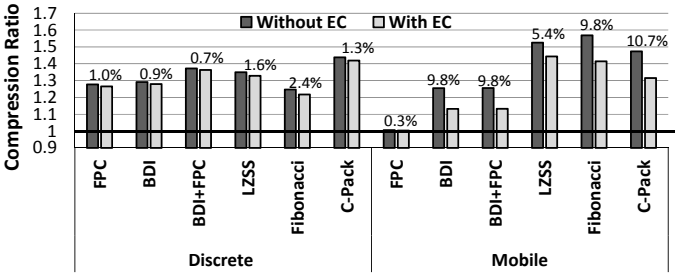

Figure 9. Effect of Energy Control on compression ratio.

EC significantly reduces the bit toggle count increase due to compression in both discrete and mobile GPU applications (Figure 8). For discrete workloads, EC's toggle count reduction varies from 5.8% to 15.9% on average, for different compression algorithms. EC almost completely eliminates the toggle count increase due to compression with the Fibonacci algorithm. For mobile workloads, which suffer much more from the bit toggle count increase, EC is even more effective: it reduces toggle count by as much as 51% on average for the BDI+FPC compression algorithm, which corresponds to a 32× reduction in *extra* bit toggles due to compression.

EC's toggle count benefits come with only a modest reduction in compression ratio[4] (Figure 9). In discrete workloads, EC reduces the compression ratio almost negligibly, e.g., by 0.7% for the BDI+FPC algorithm. In mobile workloads, EC's compression ratio reduction is more noticeable, e.g., 9.8% for BDI+FPC. However, EC's trade-off of compression ratio with bit toggle count is still very attractive since the 2.2× growth in bit toggle count is reduced to less than 10% for BDI+FPC. We conclude that EC offers a new and effective way to control the energy efficiency of data compression by applying it judiciously: it enables data compression when it provides a compression ratio at only a small increase in bit toggle count.

4.3 Effect of Metadata Consolidation

Metadata Consolidation (MC) is able to reduce bit-level misalignment for several compression algorithms (e.g., FPC and C-Pack). The additional toggle reduction (on top of EC) is 3.2%/2.9% for FPC/C-Pack compression algorithms correspondingly. We also observe that even though MC can hide some negative effects of bit-level misalignment after compression, it is not effective in cases where data compression compresses data values within the cache line 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 minimize the misalignment effects, this would go against the primary goal of compression, i.e., to minimize the data size after compression. We leave the investigation of this potential tradeoff to future work.

[4]Compression ratio reduces because EC decides to transfer some compressible lines in the uncompressed form, as it does not optimize *solely* for compression ratio.

5 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 221 workloads show these techniques are effective: 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 acknowledge the support of Intel Science and Technology Center for Cloud Computing; NSF grants 1212962, 1409723, and 1423172; and the US Department of Energy.

References

[1] B. Abali et al. Memory Expansion Technology (MXT): Software Support and Performance. *IBM J.R.D.*, 2001.
[2] A. R. Alameldeen and D. A. Wood. Adaptive Cache Compression for High-Performance Processors. In *ISCA*, 2004.
[3] A. R. Alameldeen and D. A. Wood. Interactions Between Compression and Prefetching in Chip Multiprocessors. In *HPCA*, 2007.
[4] A. Arelakis and P. Stenstrom. SC2: A Statistical Compression Cache Scheme. In *ISCA*, 2014.
[5] B. M. Beckmann and D. A. Wood. TLC: Transmission line caches. In *MICRO*, 2003.
[6] M. N. Bojnordi and E. Ipek. DESC: Energy-efficient Data Exchange Using Synchronized Counters. In *MICRO*, 2013.
[7] X. Chen et al. C-pack: A high-performance microprocessor cache compression algorithm. *IEEE Transactions on VLSI Systems*, 18(8):1196 –1208, Aug. 2010.
[8] R. Das et al. Performance and power optimization through data compression in Network-on-Chip architectures. In *HPCA*, 2008.
[9] J. Dusser et al. Zero-content Augmented Caches. In *ICS*, 2009.
[10] R. Gonzalez and M. Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, Sep 1996.
[11] GPGPU-Sim v3.2.1. GPGPU-Sim Manual.
[12] E. G. Hallnor and S. K. Reinhardt. A Unified Compressed Memory Hierarchy. In *HPCA*, 2005.
[13] A. Jog et al. Owl: cooperative thread array aware scheduling techniques for improving gpgpu performance. In *ASPLOS*, 2013.
[14] A. Jog et al. Orchestrated Scheduling and Prefetching for GPGPUs. In *ISCA*, 2013.
[15] V. Narasiman et al. Improving GPU Performance via Large Warps and Two-level Warp Scheduling. In *MICRO-44*, 2011.
[16] G. Pekhimenko et al. Exploiting Compressed Block Size as an Indicator of Future Reuse. In *HPCA*, 2015.
[17] G. Pekhimenko et al. Linearly Compressed Pages: A Low Complexity, Low Latency Main Memory Compression Framework. In *MICRO*, 2013.
[18] G. Pekhimenko et al. Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches. In *PACT*, 2012.
[19] J. Pool et al. Lossless Compression of Variable-precision Floating-point Buffers on GPUs. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2012.
[20] S. Sardashti and D. A. Wood. Decoupled Compressed Cache: Exploiting Spatial Locality for Energy-optimized Compressed Caching. In *MICRO*, 2013.
[21] V. Sathish et al. Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads. In *PACT*, 2012.
[22] A. Shafiee et al. MemZip: Exploring Unconventional Benefits from Memory Compression. In *HPCA*, 2014.
[23] M. Stan and W. Burleson. Bus-invert Coding for Low-power I/O. *IEEE Transactions on VLSI Systems*, 3(1):49–58, March 1995.
[24] M. Thuresson et al. Memory-Link Compression Schemes: A Value Locality Perspective. *IEEE Trans. Comput.*, 57(7), July 2008.
[25] A. Udipi et al. Non-uniform power access in large caches with low-swing wires. In *HiPC*, 2009.
[26] N. Vijaykumar et al. A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps. In *ISCA*, 2015.
[27] J. Yang et al. Frequent Value Compression in Data Caches. In *MICRO*, 2000.
[28] G. L. Yuan et al. Complexity effective memory access scheduling for many-core accelerator architectures. In *MICRO*, 2009.
[29] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 1977.