Toggle-Aware Bandwidth Compression for GPUs

Gennady Pekhimenko[†] Evgeny Bolotin[⋆] Mike O'Connor[⋆#]
Onur Mutlu[†] Todd C. Mowry[†] Stephen W. Keckler[⋆#]
[†]Carnegie Mellon University [⋆]NVIDIA [#]UT-Austin

Abstract—Hardware bandwidth compression can be an efficient way to achieve better system performance and energy efficiency for modern bandwidth intensive applications. Prior works studied the potential designs to exploit data redundancy through data compression to improve both the capacity (e.g., caches and main memory) and bandwidth utilization of the interconnects. These works addressed two common shortcomings of compression: (i) compression/decompression overhead in terms of latency, energy and area, and (ii) complexity to support variable size. In this work, we make a new observation that there is an additional important problem related to bandwidth compression that must be addressed in the context of the communication energy efficiency: a substantial growth in the number of *bit toggles* (communication channel switchings from 0 to 1 or from 1 to 0) when transferring compressed data: we show 2.2× average increase in bit toggles with some compression algorithms across 54 mobile applications). This change in bit toggles number leads to growth in dynamic energy consumed by on-chip/off-chip buses due to more frequent charging and discharging of the wires. We characterize the problem and propose two mechanisms to address this new challenge: Energy Control and Metadata Consolidation. We provide a detailed analysis and evaluation of a large spectrum of GPU applications that justify both the usefulness of data compression for bandwidth compression in many real applications, as well as the existence of the bit toggle problem for bandwidth compression. The proposed toggle-aware compression solutions can keep most of the bandwidth reduction benefits of bandwidth compression while leading to only minor increase in bit toggles, in contrast to the large increase when using baseline compression.

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* [7]), increasing memory latency (*memory latency wall* [27]), and scarce and expensive bandwidth resources (*bandwidth wall* [20]). 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 some of the described constraints is to employ dedicated hardware-based *data compression* mechanisms [28], [2], [8], [18], [4], specifically across various data links in the system. Compression exploits the high data redundancy observed in many modern applications [18], [21], [4]. It can be used to improve both the capacity (e.g., caches, DRAM, non-volatile memories [28], [2], [8], [18], [4], [17], [23]) and bandwidth utilization (e.g., on-chip and off-chip interconnects [9], [22], [17], [23]). Several recent works [22], [17], [23], [3], [25] focus on bandwidth compression approaches to decrease memory traffic by transmitting data in a compressed form for both CPUs [17], [25], [3] and GPUs [22], [16], which results in better system performance and energy consumption. Bandwidth compression proves to be particularly effective in GPUs because GPUs are often bottlenecked on memory bandwidth [15], [14], [29]. GPU applications also exhibit high degrees of data redundancy [22], [16], enabling good compression ratios to be exploited.

1.1 Why Data Compression Can Be Energy-inefficient

While the benefits of data compression are clear, there are also two commonly-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 [13], [21], [17], [23]. Both problems have solutions to make data compression practical: e.g., Base-Delta-Immediate compression [18] describes a very low-latency, low-energy hardware-based compression algorithm, and Decoupled Compressed Cache design [21] proposes an efficient way to manage data recompaction and fragmentation in compressed caches.

In this work, 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: a potential significant increase in the activity factor [26], [5], [6] or the number of charges/discharges on the physical wires (i.e., bit toggles), when transferring compressed data versus when transferring it in uncompressed form. A larger total number of *bit toggles* causes increased dynamic energy consumed by on-chip/off-chip channels due to more frequent switching on the channel wires. We identify two reasons for this increase in bit toggles: (i) higher per-bit entropy of the data after compression (the same amount of information is now stored in fewer bits, hence, the "randomness" of a single bit grows), and (ii) the variable-size nature of compression that can negatively affect the word/flit data alignment in the data structures. Thus, in contrast to the common wisdom that bandwidth compression saves energy (when effective), our key observation offers an interesting tradeoff: energy savings due to higher available bandwidth versus energy loss due to higher transfer energy for compressed data. This observation and the corresponding tradeoff are the key contributions of this work.

To understand how applicable general-purpose data compression is for real applications, and also how severe the problem we identify is, we analyze a large (221 total) group of discrete and mobile graphics application traces from a major GPU vendor with six previously proposed compression algorithms. Our analysis shows that even though off-chip bandwidth compression is able to achieve a significant compression ratio (e.g., more than 47% average increase in effective bandwidth with C-Pack [8] for mobile GPU applications), it can also greatly increase the total number of bit toggles (e.g., 2.2× average increase with C-Pack across 54 mobile GPU applications). 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 Bandwidth Compression

In this work, we develop and analyze two mechanisms that make bandwidth compression more energy-efficient by limiting the overall growth in bit toggles in cases where it significantly increases after compression. We propose a new *Energy Control (EC)* mechanism that monitors the benefits and overheads of data compression, and decides whether it is better to send data in a compressed or uncompressed format, based achieved compression ratio and a relative change in bit toggles. The key insight behind EC is that the decision can be made locally (e.g., for every cache line) based on a model derived from the commonly used $Energy \times Delay$ and $Energy \times Delay^2$ metrics. In this model, $Energy$ is directly proportional to the number of bit toggles, and $Delay$ is inversely proportional to the compression

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

ratio. In addition to EC, we propose and develop a new *Metadata Consolidation* optimization for existing data compression algorithms. It reduces the negative effects of inserting the per-word metadata into the cache line data after compression with many existing compression algorithms [2], [8].

Our toggle-aware mechanisms are generic and applicable to different compression algorithms (e.g., Frequent Pattern Compression (FPC) [2] and Base-Delta-Immediate (BDI) [18] compression), to different communication channels (e.g., on-chip and off-chip busses), and potentially to different architectures (e.g., both GPUs and CPUs). In addition, we demonstrate that our proposed mechanisms are largely orthogonal to different encoding schemes also used to minimize the number of bit toggles (e.g., Data Bus Inversion [24]), and hence can be efficiently used together with them to enhance energy-efficient on-chip/off-chip bus designs.

2 Motivation and Analysis

In this work we examine six compression algorithms and assess their compression efficiency and applicability for bandwidth compression in GPU compute applications, taking into account bit toggles. The six algorithms are: (i) *FPC* (Frequent Pattern Compression) [2]; (ii) *BDI* (Base-Delta-Immediate Compression) [18]; (iii) *BDI+FPC* (combining FPC and BDI) [16]; (iv) *LZSS* (Lempel-Ziv compression) [30], [1]; (v) *Fibonacci* (graphics-specific compression algorithm) [19]; and (vi) *C-Pack* compression algorithm [8]. Moreover, to make our conclusions 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 reduction, 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 minimal flit size is 32 bytes and the packet size is 128 bytes).

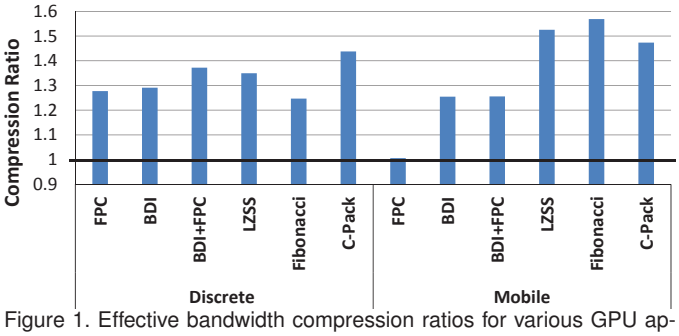

Figure 1. Effective bandwidth compression ratios for various GPU applications (higher bars are better).

First, for the 167 discrete GPU applications, all algorithms provide substantial reduction in bandwidth consumption (25%–44% on average for different compression algorithms). Especially interesting is that simple compression algorithms show very competitive results with the more complex GPU-oriented *Fibonacci* compression algorithm and the software-based Lempel-Ziv algorithm [30]. Second, for the 54 mobile GPU applications, the benefits of compression 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, compression benefits come with additional costs. Two aspects of the overheads of compression are well-known: (i) additional data processing due to compression/decompression, and (ii) some hardware changes to transfer variable-length cache lines. While these two problems are significant, multiple compression algorithms [2], [28], [18], [10] were proposed to minimize the overheads of data compression/decompression. Several designs [23], [22], [16] were proposed to integrate bandwidth compression into existing memory hierarchies. In this work, we make an additional important observation: there is yet another 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 [26], [5], [6] to the number of bit toggles on the communication channel due to charging and discharging the channel wire capacitance. Data compression may increase or decrease the overall bit toggling on the communication channel for a given datum, and as a result the overall energy that is consumed for moving/storing this datum can also change. To understand whether there is any significant change in the bit toggling, we measure the total number of bit toggles with and without compression. Figure 2 shows the results of this experiment for discrete and mobile GPU applications for the six compression algorithms, normalized to the number observed in the uncompressed baseline. The total number of bit toggles already includes the effect of compression due to the decrease in the total number of bits sent when data is compressed.

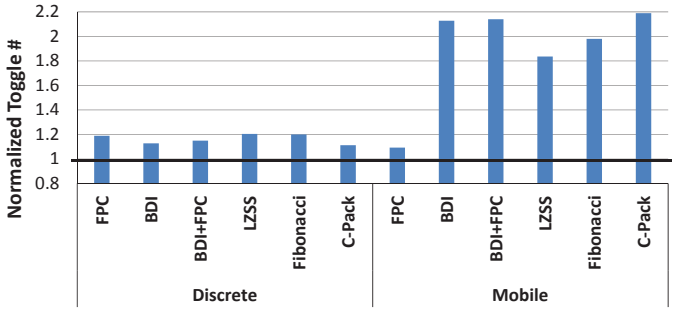

Figure 2. Toggle number increase for different applications.

We make two observations from this figure. First, we observe a steady increase in the total number of toggles across all compression algorithms. The average growth is lower for discrete applications, mostly because many of the applications include floating-point data that are less amenable to compression and have already high toggle rates (31% on average across discrete applications), but it is still quite significant on average, within 12%–20%, depending on the algorithm. Second, for mobile applications (right half of Figure 2), the growth is more dramatic, exceeding 1.8× for all but one algorithm. The FPC algorithm is not effective in compressing mobile applications from our pool, hence does not greatly affect bit toggles. In both cases, this increase in number of bit toggles can potentially lead to significant growth in the dynamic energy consumed by the communication channels.

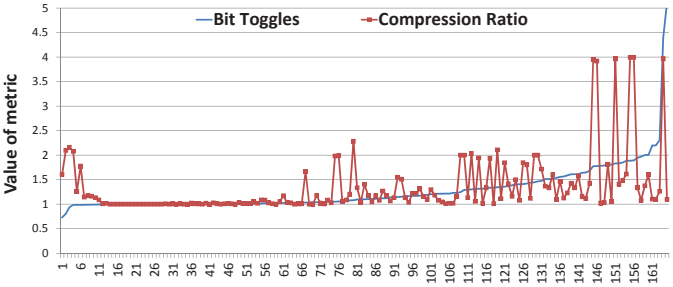

Figure 3. Normalized number of bit toggles vs. compression ratio (FPC compression algorithm) for each of the discrete GPU applications.

In addition, we study the possible correlation between the effectiveness of a compression algorithm and the resultant growth in the number of bit toggles. Figure 3 summarizes the results on a per application basis for the FPC compression algorithm, for all discrete applications. We observe similar behavior for other compression algorithms. For each workload, we plot both its compression ratio and the normalized number of toggles after compression. Figure 3 shows a correlation between the increase in the number of toggles and

the compression ratio. We observe that the number of toggles usually correlates with compressibility. This observation strongly suggests that successful compression can potentially lead to an increase in the number of toggles, which in turn would lead to an increase in the dynamic energy dissipated by on-chip/off-chip communication channels.

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.

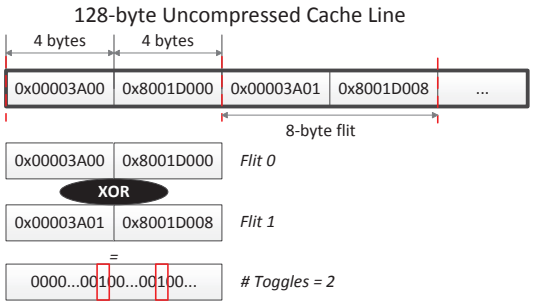

Figure 4. Number of bit toggles without compression.

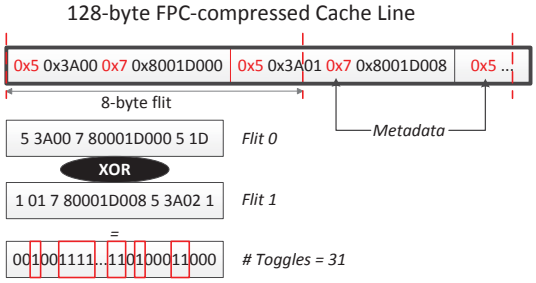

Figure 5. Number of bit toggles after compression with FPC.

Without compression, an example cache line that 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 favorable alignment of the original data with width of a flit (i.e., transfer granularity in the on-chip interconnect). In contrast, Figure 5 shows the FPC algorithm where the toggle count increases significantly (e.g., 31 toggles for a pair of 8-byte flits in this example). This happens due to two major reasons. First, because compression removes zero bits for narrow index values, the higher per-bit entropy leads to higher "randomness" of the data and thus a larger number of toggles. Second, the originally good alignment of the data is negatively affected both at the byte granularity (narrow values replaced with shorter 2-byte versions) and bit granularity (due to 3-bit metadata storage, e.g., $101 = 0x5$ metadata is used to indicated narrow values for the FPC compression algorithm).

3 Toggle-aware Energy Efficient Compression

3.1 Energy vs. Performance Tradeoff

Data compression can reduce energy consumption and improve performance by reducing communication bandwidth demands and alleviating potential bandwidth bottlenecks. At the same time, data compression can potentially lead to a significantly higher energy consumption due to an increased number of bit toggles. To properly evaluate this tradeoff, metrics like $Energy \times Delay$ and $Energy \times Delay^2$ are commonly used [11]. We estimate these metrics to perform compression related performance/energy tradeoffs using a simple model.[2] We define the $Energy$ of a single data transfer to be proportional to the number of toggles associated with its transfer. Similarly, the $Delay$ is defined to be inversely proportional to performance, which we assume is proportional to the bandwidth reduction (or compression ratio). Based on the observations above, we have developed two techniques to enable toggle-aware bandwidth compression to reduce the negative effects of increased bit toggles.

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 while minimizing the overall toggle energy. As shown in Figure 6, the Energy Control mechanism uses a generic decision function that considers (i) the number of bit toggles for transmitting the original datum (T_0), (ii) the number of bit toggles for transmitting it in compressed form (T_1), and (iii) compression ratio (CR) to decide whether to transmit the data in a compressed or uncompressed form. We can calculate the toggle count very energy efficiently (4pJ per 128-byte cache line and 32-byte flits based on our initial results in Verilog). 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 $Energy \times Delay$ heuristic), or quadratic ($A \times B^2 > 1$, based on $Energy \times Delay^2$). The decision function can also use any other sensible decision metric, assuming application bandwidth requirements, power limitations, and available voltage/frequency scaling options, and other system power management opportunities.

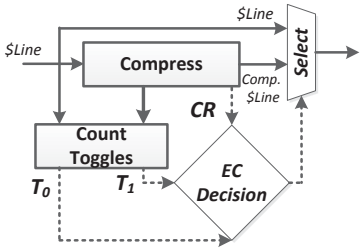

Figure 6. Energy Control decision mechanism.

3.3 Metadata Consolidation

Thus far, we only considered traditional energy-oblivious compression algorithms that are not optimized to minimize the number of toggles. However, it is possible to extend existing algorithms (e.g. FPC and C-Pack) such that the increase in bit toggles would be smaller. We propose Metadata Consolidation (MC) where the metadata that is usually stored at a per word-granularity, is consolidated into a single contiguous metadata block. We can locate this block either before or after the actual data. The tradeoff is between the toggle count and the increase in decompression latency since the decompression needs to know the metadata. The major benefit of such an approach is that there is no misalignment at the bit granularity after this optimization is applied. In cases where a cache line has a majority of similar patterns, a significant portion of the toggle overhead can be avoided.

Figure 7 shows one example cache line compressed with FPC algorithm with and without the MC optimization. In this example we assume 4-byte flits. Without the MC, the bit toggle count between first two flits is 18 (due to per-word metadata insertion), while with MC the corresponding bit toggle count is only 2. This example shows the potential of MC in decreasing the bit toggle count.

4 Evaluation

4.1 Methodology

We analyze 221 memory traces from compute (167) and graphics (54) application traces. We collect the information about the bit toggle count that reflects energy consumption and compression ratio that serves as a proxy for bandwidth consumption. Different encoding techniques (e.g., DBI [24] or DESC [6]) can be applied to decrease the baseline bit toggle count of the data transfers. In our experiments, we found that the benefits of these techniques are largely orthogonal

[2]We are currently working on verifying our estimations by implementing our techniques in the open-sourced GPGPU simulator called GPGPU-Sim [12].

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

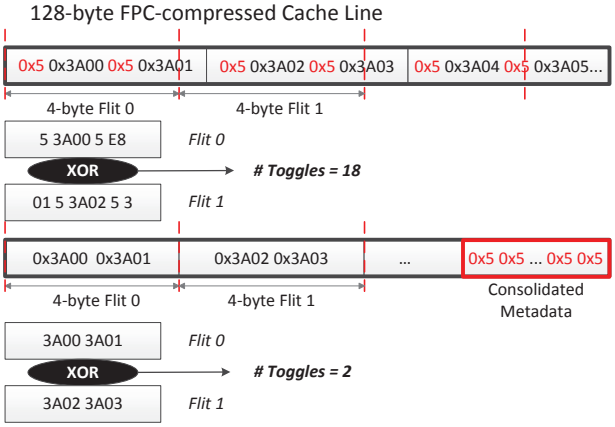

Figure 7. Number of bit toggles after compression with FPC and Metadata Consolidation.

to whether or not data compression is applied. In our evaluation we use DBI as a part of the baseline for transferring both compressed and uncompressed data.

4.2 Effect of EC on Toggles and Compression Ratio

We analyze the effectiveness of the proposed EC optimization by examining how it affects both the number of toggles (Figure 8) and the compression ratio (Figure 9). In both figures, results are normalized to the baseline design with no compression (the numbers on top of the bars are the relative change between these bars). We also use an EC mechanism with the decision function based on the $Energy \times Delay^2$ metric using our simple model from Section 3.2.

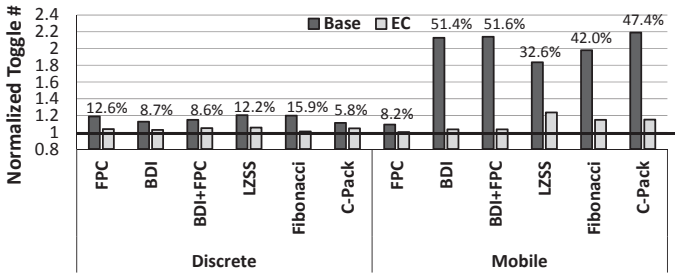

Figure 8. Effect of Energy Control on the number of toggles.

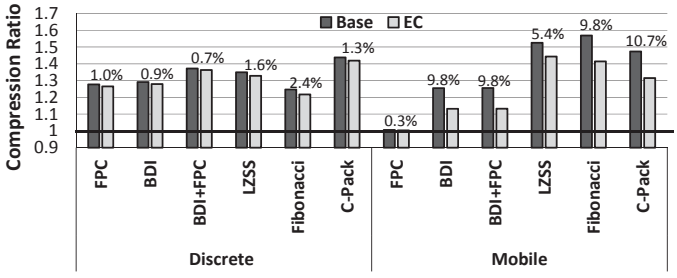

Figure 9. Effective bandwidth increase for different applications.

EC can effectively reduce the overhead in the number of toggles for both discrete and mobile GPU applications (Figure 8). For discrete GPUs, the toggle reduction varies from 6% to 16% on average, and is able to eliminate the toggle growth due to compression almost completely in the case of the Fibonacci compression algorithm. For mobile GPUs, the reduction is as high as 51% on average for the BDI+FPC compression algorithm (more than 32× reduction in *extra* bit toggles), with only a modest reduction[4] in compression ratio. For example, in discrete GPUs, the reduction in compression ratio for the BDI+FPC algorithm is only 0.7% on average (Figure 9). For mobile GPUs, the reduction in compression ratio is more noticeable (e.g., 9.8% on average for Fibonacci), but this is still a very attractive tradeoff since the 2.2× growth in the number of toggles is practically eliminated. We conclude that EC offers a new and effective way to control the energy efficiency of data compression by applying it only when it provides a high compression ratio with only a small increase in the number of toggles.

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

4.3 Effect of Metadata Consolidation

Metadata Consolidation (MC) is able to reduce the bit-level misalignment for several compression algorithms (e.g., FPC and C-Pack). The additional toggle reduction (on top of EC) was 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 the 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 to minimize the data size after compression. We leave the investigation of this potential tradeoff to future work.

5 Conclusion

In this paper, we observe that data compression, while very effective in improving bandwidth efficiency in GPUs, can greatly increase the number of bit toggles in the on-chip/off-chip interconnect. Based on this new observation, we develop two techniques to reduce the number of toggles while preserving most of the bandwidth reduction benefits of compression. The proposed toggle-aware bandwidth compression solutions are able to retain most of the bandwidth reduction advantages, while leading to a controlled increase in energy consumption. We conclude that toggle-awareness is an important consideration in bandwidth compression mechanisms for modern GPUs.

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-31*, 2004.
[3] A. R. Alameldeen and D. A. Wood. Interactions Between Compression and Prefetching in Chip Multiprocessors. In *HPCA'07*, 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] P. Bose. Power Wall. In *Encyclopedia of Parallel Computing*. 2011.
[8] X. Chen et al. C-pack: A high-performance microprocessor cache compression algorithm. *IEEE Transactions on VLSI Systems*, 18(8):1196 –1208, Aug. 2010.
[9] R. Das et al. Performance and power optimization through data compression in Network-on-Chip architectures. In *HPCA*, 2008.
[10] J. Dusser et al. Zero-content Augmented Caches. In *ICS*, 2009.
[11] R. Gonzalez and M. Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, Sep 1996.
[12] GPGPU-Sim v3.2.1. GPGPU-Sim Manual.
[13] E. G. Hallnor and S. K. Reinhardt. A Unified Compressed Memory Hierarchy. In *HPCA-11*, 2005.
[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 and et al. Linearly Compressed Pages: A Main Memory Compression Framework with Low Complexity and Low Latency. In *SAFARI Technical Report No. 2012-002*, 2012.
[17] G. Pekhimenko et al. Linearly Compressed Pages: A Low Complexity, Low Latency Main Memory Compression Framework. In *MICRO-46*, 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*, I3D '12, 2012.
[20] B. M. Rogers et al. Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling. In *ISCA*, 2009.
[21] S. Sardashti and D. A. Wood. Decoupled Compressed Cache: Exploiting Spatial Locality for Energy-optimized Compressed Caching. In *MICRO-46*, 2013.
[22] V. Sathish et al. Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads. In *PACT*, 2012.
[23] A. Shafiee et al. MemZip: Exploring Unconventional Benefits from Memory Compression. In *HPCA-20*, 2014.
[24] M. Stan and W. Burleson. Bus-invert Coding for Low-power I/O. *IEEE Transactions on VLSI Systems*, 3(1):49–58, March 1995.
[25] M. Thuresson et al. Memory-Link Compression Schemes: A Value Locality Perspective. *IEEE Trans. Comput.*, 57(7), July 2008.
[26] A. Udipi et al. Non-uniform power access in large caches with low-swing wires. In *HiPC*, 2009.
[27] W. A. Wulf and S. A. McKee. Hitting the Memory Wall: Implications of the Obvious. *SIGARCH Comput. Archit. News*, 1995.
[28] J. Yang et al. Frequent Value Compression in Data Caches. In *MICRO-33*, 2000.
[29] G. L. Yuan et al. Complexity effective memory access scheduling for many-core accelerator architectures. In *MICRO*, 2009.
[30] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 1977.