Energy-Efficient Data Compression for GPU Memory Systems

Gennady Pekhimenko[†] Evgeny Bolotin[⋆] Mike O'Connor[⋆#]

Onur Mutlu[†] Todd C. Mowry[†] Stephen W. Keckler[⋆#]

[†]Carnegie Mellon University [⋆]NVIDIA Research [#]UT-Austin

1. Introduction and Background

Modern data-intensive computing forces system designers to deliver good performance under several major constraints: limited system power/energy budget (*power wall* [6]), high memory latency (*memory wall* [22]), and on-chip/off-chip memory bandwidth (*bandwidth wall* [15]). Multiple different techniques were proposed to address these issues, but, unfortunately, these techniques usually offer a trade-off: improving one constraint at the cost of another one.

Ideally, system designers would like to improve one or several of the system parameters, e.g., on-chip/off-chip bandwidth consumption, with minimal (if any) negative impact on other key parameters. One potential way of achieving this is hardware-based *data compression* mechanisms [23, 1, 8, 14, 4], and more specifically, *bandwidth or link compression*. Data compression exploits high data redundancy observed in many modern applications [1, 14, 16, 4], and can be used to improve both the capacity (e.g., caches, DRAMs, non-volative memories [23, 1, 8, 14, 4, 13, 18] and bandwidth utilization of the interconnects (e.g., on-chip and off-chip buses [17, 13, 18]). Several recent works [17, 13, 18, 3, 21] apply data compression to decrease memory traffic by sending/receiving data in a compressed form for both CPUs [13, 21, 3], and GPUs [17, 12] that result in better system performance and/or energy consumption. Bandwidth compression is especially effective for GPU applications [17, 12] where the limited main memory bandwidth usually becomes the major bottleneck to achieve high performance [10], and there is also significant redundancy in transferred data [17, 12].

2. Data Compression Can Be Energy Inefficient

While the above benefits of data compression are clear, there are also two common shortcomings of data compression that prior work have looked at: (i) compression/decompression overhead [1, 14] (in terms of latency, energy and area) and (ii) complexity/cost to support variable size [9, 16, 13, 18]. Both problems have reasonable solutions to make data compression practical (e.g., Base-Delta-Immediate compression [14] demonstrates very low-latency low-energy hardware-based compression algorithm, and Decoupled Compressed Cache design [16] proposes an efficient way to deal with data recompaction and fragmentation in compressed caches).

In this work, we make a *new observation* that there is yet another important problem with data compression that needs to be addressed in the context of communication channels – *increase in the number of bit-toggles (bit-flips)* when transferring compressed data. This increase in the bit-toggles (i.e., switching a bit from 0 to 1 or from 1 to 0) or in the activity factor [5] then leads to increase in dynamic energy consumed by on-chip/off-chip buses (due to more frequent charging and discharging the channel wire capacitance). Hence data compression offers a tradeoff – more available bandwidth vs. potentially higher dynamic energy of the data transfers that has to be properly analyzed.

The reason for this increase in bit-toggles is two-fold: (i) higher per-bit entropy of the data after compression (the same amount of information is now stored in less bits, hence, the "randomness" of a single bit increases), (ii) variable-size nature of compression that negatively affects data that was originally more efficiently word-/flit-alligned.

In order to understand (i) how applicable is general-purpose data compression for real applications (i.e., outside of previously analyzed SPEC [19] and small GP-GPU applications [7, 11]), and how severe is the observed problem – the increased number of bit-toggles, we analyze a large (231 total) group of application traces from major GPU vendor (both general-purpose and mobile applicatons) with *six* previously proposed compression algorithms. Our analysis shows that even though data compression offers a significant increase in compression ratio (e.g., more than 47% average increase in effective bandwidth with C-Pack compression algorithm [8] for mobile applications), it can also significantly increase the total number of toggles (e.g., more than 2.2X average increase with C-Pack for mobile applications). This, in turn, can significantly increase the energy of on-chip/off-chip interconnects which constitute a significant portion of the memory subsystem energy.[1]

3. Our Approach: Key Idea

In this work, we aim to build a new set of mechanisms to make bandwidth compression more energy-efficient by reducing the number of toggles that significantly increases due to compression. First, 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 compressed or uncompressed form (based on both compression ratio and relative change in bit-toggle rate). The key insight behind EC is that the decision can be made locally (e.g., for every cache line) based on the cost-benefit model derived from the commonly used $Energy * Delay$ and $Energy * Delay^2$ metrics. In this model, $Energy$ is directly proportional to the bit-toggle number, and $Delay$ is inversely proportional to compression ratio. Second, we define a new *Local (Flit)* and *Global (Packet)* reordering mechanisms that can be used in existing on-chip interconnects to further reduce the number of bit-toggles. Our proposed solutions are especially effective when the number of flits is small (e.g., due to data compression). Third, we propose a new *Metadata Consolidation* optimization for existing data compression algorithms to reduce the negative effects of inserting per word metadata into the cache line data after compression.

Our proposed mechanisms are applicable to different compression algorithms (e.g., FPC [2] and BDI [14] compression), to different communication channels (e.g., on-chip and off-chip buses), and potentially to different architectures (e.g., both GPUs and CPUs). Some of our proposed techniques (Local and Global reordering) can be efficiently applied to data transfers even without compression. We also demonstrate that our mechanisms are largely orthogonal to different encoding schemes (e.g., Data Bus Invertion (DBI) [20]) also used to minimize the number of bit-toggles, and hence can be efficiently used together to obtain the benefits of both types of techniques.

4. Novelty and Contributions

In summary, this work makes following contributions:

- We make the new observation that hardware-based data compression applied to on-chip/off-chip buses poses a new challenge for system designers – a significant increase in the number of bit-toggles after compression. Without proper care, this increase can lead to significant energy

[1] For example, up to 80% energy of the LLC caches is H-tree capacitance interconnects [5].

overheads when transferring compressed data that was not accounted for in prior works.

- We propose a set of new mechanisms to address this new challenge: Energy Control, Local/Global Reordering, and Metadata Consolidation.

- We provide a detailed analysis and evaluation of a large spectrum of GP-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. Our proposed solutions can deliver most of benefits of bandwidth compression with only minor increase in energy consumption (in contrast to 2.2X increase in the energy consumption with the baseline compressed design).

References

[1] A. R. Alameldeen and D. A. Wood. Adaptive Cache Compression for High-Performance Processors. In *ISCA-31*, 2004.

[2] A. R. Alameldeen and D. A. Wood. Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches. *Tech. Rep.*, 2004.

[3] A. R. Alameldeen and D. A. Wood. Interactions between compression and prefetching in chip multiprocessors. *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 0:228–239, 2007.

[4] A. Arelakis and P. Stenstrom. Sc2: A statistical compression cache scheme. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 145–156, Piscataway, NJ, USA, 2014. IEEE Press.

[5] M. N. Bojnordi and E. Ipek. DESC: Energy-efficient Data Exchange Using Synchronized Counters. In *MICRO*, 2013.

[6] P. Bose. Power Wall. In *Encyclopedia of Parallel Computing*. 2011.

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

[8] X. Chen, L. Yang, R. Dick, L. Shang, and H. Lekatsas. C-pack: A high-performance microprocessor cache compression algorithm. In *VLSI Systems, IEEE Transactions on*, volume 18, pages 1196 –1208, Aug. 2010.

[9] E. G. Hallnor and S. K. Reinhardt. A Unified Compressed Memory Hierarchy. In *HPCA-11*, 2005.

[10] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt. Improving gpu performance via large warps and two-level warp scheduling. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 308–317, New York, NY, USA, 2011. ACM.

[11] NVIDIA. CUDA C/C++ SDK Code Samples, 2011.

[12] 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.

[13] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. Linearly Compressed Pages: A Low Complexity, Low Latency Main Memory Compression Framework. In *MICRO-46*, 2013.

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

[15] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: Challenges in and avenues for cmp scaling. *SIGARCH Comput. Archit. News*, 37(3):371–382, June 2009.

[16] S. Sardashti and D. A. Wood. Decoupled Compressed Cache: Exploiting Spatial Locality for Energy-optimized Compressed Caching. In *MICRO-46*, 2013.

[17] V. Sathish, M. J. Schulte, and N. S. Kim. Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads. In *PACT*, 2012.

[18] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis. MemZip: Exploring Unconventional Benefits from Memory Compression. In *HPCA-20*, 2014.

[19] SPEC CPU2006 Benchmarks. http://www.spec.org/.

[20] M. Stan and W. Burleson. Bus-invert coding for low-power I/O. In *IEEE Transactions on VLSI Systems*, volume 3, pages 49–58, March 1995.

[21] M. Thuresson, L. Spracklen, and P. Stenstrom. Memory-Link Compression Schemes: A Value Locality Perspective. *IEEE Trans. Comput.*, 57(7), July 2008.

[22] W. A. Wulf and S. A. McKee. Hitting the Memory Wall: Implications of the Obvious. *SIGARCH Comput. Archit. News*, 1995.

[23] J. Yang, Y. Zhang, and R. Gupta. Frequent Value Compression in Data Caches. In *MICRO-33*, 2000.