# A Case for Memory Access Granularity Aware Selective Lossy Compression for GPUs

Sohan Lal
Technische Universität Berlin, Germany
Email: sohan.lal@tu-berlin.de

Ben Juurlink
Technische Universität Berlin, Germany
Email: b.juurlink@tu-berlin.de

## I. Motivation and Background

Memory compression has been demonstrated as a promising alternative to increase memory bandwidth [1], [2], [3], [4], however, memory compression techniques often exhibit a low effective compression ratio. The main reason for the low effective compression ratio is the large memory access granularity (MAG) exhibited by GPUs due to wide memory interface width and large burst length. For example, MAG of GDDR5/5X/6 is 32B resulting from 32-bit interface width and burst length of 8. MAG refers to the amount of data read from or written to a memory by a single read or write command. MAG reduces the compression ratio as data can only be fetched in a multiple of MAG but a compressed block is often not a multiple of a MAG. For example, for a compressed size of 36B, we always fetch 64B. Thus, a compression ratio that seems close to 4× (3.6, assuming a typical block size of 128B used in contemporary GPUs) is actually only 2×. This leads to a significant difference between the raw and the effective compression ratio actually gained by a system. The raw compression ratio is calculated without considering the MAG by simply dividing the uncompressed size by compressed size and the effective compression ratio is calculated by scaling up the compressed size to the nearest multiple of a MAG.

Figure 1 shows the raw and effective compression ratios of Base Delta-Immediate (BDI) [5], Frequent-Pattern Compression (FPC) [6], Cache Packer (C-PACK) [1], and Entropy Encoding Based Memory Compression ($E^2$MC) [7] techniques for benchmarks from CUDA SDK and Rodinia. We see that the geometric mean (GM) of the effective compression ratio of BDI, FPC, C-PACK and $E^2$MC is 22%, 19%, 18% and 23% less than the raw compression ratio, respectively. The low effective compression ratio reduces performance benefits, otherwise possible from a higher raw compression ratio. Interestingly, our study of the distribution of compressed blocks shows a significant percentage of blocks are compressed to a size that is only a few bytes above MAG. However, as there is no way to fetch a few bytes, a whole burst is fetched from the memory, causing low effective compression ratio.

Figure 1 quantitatively shows that four state-of-the-art memory compression techniques suffer due to MAG. There are three other techniques: $SC^2$ [8], HyComp [9] and BPC [3] that can also be applied for memory compression. $SC^2$ is

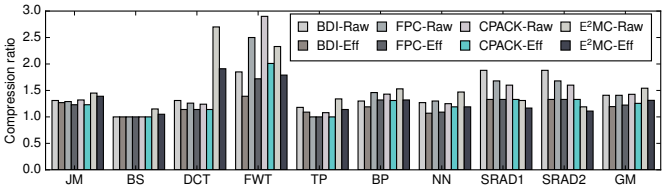


Fig. 1: Raw and effective compression ratio of BDI, FPC, C-PACK and $E^2$MC using block size of 128B and 32B MAG.

a statistical cache compression technique and is similar to $E^2$MC because both are based on Huffman encoding. The former is proposed for CPUs, while the later is proposed for GPUs. Therefore, $SC^2$ will suffer due to MAG. HyComp is a hybrid compression method which improves the compression ratio by selecting a suitable compression method based on the specific data-type. HyComp will also suffer from MAG as two (BDI and $SC^2$) out of the four compression methods that HyComp selectively uses are already shown to suffer. The third method called FP-H divides a floating-point number into three fields: Exponent, Mantissa-High, and Mantissa-Low and then employs $SC^2$ to compress each of these fields in the isolation, that means FP-H will also suffer from MAG. BPC stands for bit plane compression that uses transformation to increase the compressibility and then uses either run-length or frequent pattern encoding to compress the transformed data. While transformation increases compressibility, BPC will still suffer from MAG as both the run length and frequent pattern encodings exploit patterns similar to FPC and C-PACK which are already shown to suffer in Figure 1. Therefore, several memory compression techniques suffer from MAG.

## II. MAG Aware Selective Approximation

With the goal to increase the effective compression ratio, we propose a novel Selective Lossy Compression (SLC) for GPUs which is aware of MAG. The key idea of SLC is that when a lossless compression yields a compressed size with a few extra bytes, we use lossy compression to approximate these extra bytes such that the compressed size is a multiple of MAG. This way, we selectively introduce a small approximation error, however, we significantly increase the compression ratio. While lossless memory compression is desired, lossless compression alone is not sufficient to meet the bandwidth requirements and there are several GPU applications that are

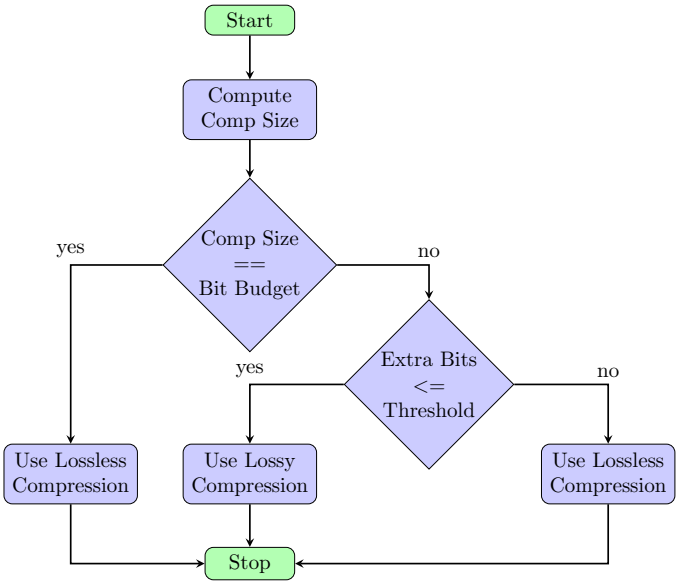


Fig. 2: Overview of selective lossy compression.

inherently resilient to error and small approximation will not degrade their output quality to an unacceptable level [10]. We choose E$^2$MC [7] as the baseline lossless compression as it provides the highest compression ratio and performance gain and adapt it to implement the lossy compression mode.

Figure 2 shows an overview of the SLC technique. Basically, SLC is a budget-based compression technique which allows selection between different compression modes depending upon *bit budget*, *extra bits*, and a *threshold*. SLC selectively uses a combination of lossless and lossy compression modes. When lossless compression yields a compressed size which is a few bytes (*extra bits*) above a multiple of MAG and *extra bits* within the *threshold*, lossy compression mode is used to approximate these *extra bits* such that the compressed size is a multiple of MAG. Thus, SLC retains the quality of a lossless compression when the compressed size is a multiple of MAG or the *extra bits* are above the *threshold*, otherwise, lossy compression is used to achieve the desired compression.

SLC requires compressed block size (*comp size*) to choose a compression mode. The *bit budget* is a multiple of MAG i.e. 32B, 64B, 96B, or 128B. When the *comp size* of a block is more than its uncompressed size, the block is always stored uncompressed and the *bit budget* is 128B. Since it is not possible to fetch less than 32B from memory, we also use lossless compression when the *comp size* is less than 32B and here, the *bit budget* is also 32B. The *extra bits* are the number of bits above the *bit budget* and the *threshold* is the number of bits defined by a user that can be safely approximated.

Once we know the *comp size*, we check if it is equal to *bit budget*. We use lossless compression when the *comp size* is equal to *bit budget*. When the *comp size* is not equal to *bit budget*, we use lossy compression if the *extra bits* $\leqslant$ *threshold* and lossless compression if the *extra bits* $>$ *threshold*. A block may be stored uncompressed when the *extra bits* $>$ *threshold*. In nutshell, based on a *bit budget*, *extra bits*, and a *threshold*, SLC selects an appropriate compression mode.

We know how many bytes (*extra bits*) are above a MAG, but the problem is that these *extra bits* are codewords and not symbols. The challenge is to find the number of symbols that need to be approximated to decrease the compressed size by *extra bits* to make it a multiple of MAG. Since it is not trivial to find the symbols that contribute extra bytes, one simple way is to approximate the whole block using quantization when lossy compression mode is selected. However, in this way, we approximate a whole block. The other method is to first determine the number of symbols needed to be approximated to decrease the compressed size and then only approximate these symbols. This requires extra logic to select the symbols, however, it only approximates a few symbols.

## III. Results and Contributions

We extend gpgpu-sim for experiments. MAG aware selective lossy compression with 16-bit quantization can achieve a speedup of up to 35% and 17% normalized to E$^2$MC for 64B and 32B MAG, respectively, with an accuracy loss of $< 1.0\%$. In summary, we make the following main contributions:

- We quantitatively show that low effective compression ratio due to MAG exists in four state-of-the-art techniques and qualitatively in three more.
- This is the first study that highlights the importance of MAG aware compression by quantitatively studying the distribution of compressed blocks above MAG.
- We show a significant performance gain with a minimal loss in accuracy.

## References

[1] X. Chen, L. Yang, R. Dick, L. Shang, and H. Lekatsas, "C-Pack: A High-Performance Microprocessor Cache Compression Algorithm," *IEEE Transactions on VLSI Systems*, 2010.

[2] V. Sathish, M. J. Schulte, and N. S. Kim, "Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads," in *Proc. 21st PACT*, 2012.

[3] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-plane Compression: Transforming Data for Better Compression in Many-core Architectures," in *Proc. 43rd ISCA*, 2016.

[4] N. Vijaykumar, G. Pekhimenko, A. Jog, A. Bhowmick, R. Ausavarungnirun, C. Das, M. Kandemir, T. Mowry, and O. Mutlu, "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps," in *Proc. 42nd ISCA*, 2015.

[5] 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 *Proc. 21st PACT*, 2012.

[6] A. Alameldeen and D. Wood, "Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches," in *Technical Report, University of Wisconsin-Madison*, 2004.

[7] S. Lal, J. Lucas, and B. Juurlink, "E$^2$MC: Entropy Encoding Based Memory Compression for GPUs," in *Proc. 31st IPDPS*, 2017.

[8] A. Arelakis and P. Stenstrom, "SC$^2$: A Statistical Compression Cache Scheme," in *Proc. 41st ISCA*, 2014.

[9] A. Arelakis, F. Dahlgren, and P. Stenstrom, "HyComp: A Hybrid Cache Compression Method for Selection of Data-type-specific Compression Methods," in *Proc. 48th MICRO*, 2015.

[10] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural Acceleration for General-Purpose Approximate Programs," in *Proc. 45th MICRO*, 2012.