QSLC: Quantization-Based, Low-Error Selective Approximation for GPUs

Sohan Lal, Jan Lucas, Ben Juurlink
Technische Universität Berlin, Germany

***Abstract*—GPUs use a large memory access granularity (MAG) that often results in a low effective compression ratio for memory compression techniques. The low effective compression ratio is caused by a significant fraction of compressed blocks that have a few bytes above a multiple of MAG. While MAG-aware selective approximation, based on a tree structure, has been used to increase the effective compression ratio and the performance gain, approximation results in a high error that is reduced by using complex optimizations. We propose a simple quantization-based approximation technique (QSLC) that can also selectively approximate a few bytes above MAG. While the quantization-based approximation technique has a similar performance to the state-of-the-art tree-based selective approximation, the average error for the quantization-based technique is 5× lower. We further trade-off the two techniques and show that the area and power overhead of the quantization-based technique is 12.1× and 7.6× lower than the state-of-the-art, respectively. Our sensitivity analysis to different block sizes further shows the opportunities and the significance of MAG-aware selective approximation.**

I. Introduction

Memory compression is being widely used to increase the memory bandwidth and/or effective capacity [1]–[3]. This is necessitated by the tremendous increase in the amount of data that need high-capacity storage and fast processing using accelerators. On the one hand, the large data require higher memory bandwidth and storage capacity, on the other hand, the large data has redundancy that can be exploited by memory compression techniques to achieve a higher compression ratio. Moreover, there are several emerging applications of machine learning in various domains such as computer vision, image processing that are tolerant to reduced precision.

While memory compression plays a significant role in meeting the demands of higher memory bandwidth and capacity, memory compression techniques often result in a low effective compression ratio due to the large memory access granularity (MAG) of GDDR/DDR memories employed in multi-/many-core systems such as GPUs [4]. The large memory access granularity helps to achieve peak memory bandwidth and reduce control overhead, however, it can cause a significant loss (on average about 20%) to the effective compression ratio [4]. The loss occurs because a compressed block can be of any size and not necessarily a multiple of a MAG, while a memory controller only fetches data in the multiple of a MAG. This means a memory controller needs to fetch a whole burst even when only a few bytes are actually required. As there is a significant fraction of blocks that have a few bytes above MAG, it causes a low effective compression ratio.

MAG-aware selective lossy compression (SLC) has been used to approximate a few bytes above MAG so that the compressed size is a multiple of a MAG, resulting in increased effective compression ratio and performance gain [4], [5]. SLC uses a tree-based approach, also known as tree-based selective lossy compression (TSLC), to select the symbols for approximation. While TSLC increases the effective compression ratio and the performance gain, TSLC may result in a higher error as it uses truncation for approximation. TSLC reduces the approximation error by using a value prediction and an optimization that reduces unneeded approximation. However, these optimizations increase the complexity of the design, while the occasional high error still remains an issue.

We propose a simple quantization-based selective approximation technique (QSLC) that can also selectively approximate a few bytes above MAG like TSLC, however, our method has a much lower approximation error as well as area and power overhead. While QSLC has a similar performance gain to TSLC, QSLC reduces the approximation error by 5×. The main reason for the higher error in TSLC is that it uses truncation for the approximation of selected symbols. Moreover, TSLC can also approximate high-order bits of a word. In QSLC, the approximation error is uniformly distributed across all symbols of a block as quantization is applied to low-order bits. QSLC provides a speedup of up to 17% normalized to state-of-the-art E^2MC [3] with a maximum error of 0.64% and an average error of only 0.2%. In summary, we make the following main contributions:

- We propose a simple quantization-based technique for MAG-aware selective approximation.
- We show that the quantization-based selective approximation has a similar performance to the state-of-the-art but drastically reduces the approximation error by 5×.
- We implement hardware and show that the area and the power cost of QSLC is 12.1× and 7.6× lower than the state-of-the-art, respectively.
- Our sensitivity analysis to different block sizes shows opportunities and the significance of MAG-aware selective approximation, widening its application.

This paper is organized as follows. Section II further motivates the problem. Section III presents QSLC in detail. In Section IV, we explain our experimental setup. The results are presented in Section V. The related work is described in Section VI and finally, we conclude in Section VII.

978-3-9819263-5-4/DATE21/©2021 EDAA

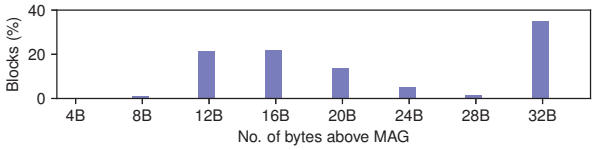


Fig. 1: Distribution of compressed blocks for *nn* benchmark.

TABLE I: Qualitative analysis of state-of-the-art MAG-aware approximation techniques.

Technique	Max Error	Average Error	Complexity
TSLC-SIMP	High	High	Simple
TSLC-PRED	High	Intermediate	Intermediate
TSLC-OPT	High	Low	Complex
QSLC	**Low**	**Low**	**Simple**

II. Motivation for Quantization-based Selective Approximation

We briefly talk about the distribution of compressed blocks and analyze the state-of-the-art MAG-aware selective approximation technique to show motivation for a quantization-based approximation as an alternative approach.

A. Analysis of Compressed Blocks at MAG

We use E^2MC [3] for compression and assume a MAG of 32 bytes and a block size of 128 bytes, which are typical values in current GPUs. Figure 1 shows the distribution of compressed blocks for the *nn* benchmark. The x-axis shows the number of bytes above a multiple of MAG and y-axis shows the percentage of compressed blocks with the corresponding number of bytes above MAG.

The figure shows that there are about 22% of blocks with a compressed size $\leqslant$ 12 bytes above a multiple of MAG and about 45% of blocks with a compressed size $\leqslant$ 16 bytes above a multiple of MAG. We see that blocks are not compressed to a multiple of MAG, but rather distributed across multiple sizes. This is also expected as the probability that a compressed block will be an exact multiple of a MAG is very low as a compressed block can be of any size. Ideally, for a high effective compression ratio, all blocks should be compressed to an exact multiple of MAG. However, we see there is a significant percentage of blocks that are not compressed to an exact multiple of MAG, but a few bytes above a multiple of MAG. As a memory controller cannot fetch the few bytes above MAG, it needs to fetch a whole 32-byte burst, causing a low effective compression ratio because more data than compressed size is fetched. Our analysis commensurates with Lal et al. [4] and we also propose to approximate the few bytes above MAG, however, by using an alternative approach that has a much lower error.

B. Qualitative Analysis of Tree-based SLC

Lal et al. [4] proposed a MAG-aware tree-based selective lossy compression (TSLC) technique to approximate the few bytes above MAG. TSLC has three variations aiming to reduce the approximation error. Table I shows the qualitative analysis of the three variations of TSLC. The first variation is called TSLC-SIMP, which uses a simple truncation (replaces selected symbols with zeros) to approximate the symbols. While it is relatively simple in complexity, it has a high maximum and high average error. The second variation is called TSLC-PRED, which uses a value prediction to reconstruct approximated symbols, aiming to reduce error. TSLC-PRED lowered the maximum and average error compared to TSLC-SIMP, but it increases the complexity. The third variation is called TSLC-OPT, which in addition to prediction, further optimizes TSLC-PRED by adding a few extra nodes. While TSLC-OPT has a low average error, it can still result in a high error in some benchmarks. Furthermore, it increases the complexity of the design. In contrast, our simple quantization-based approximation method (QSLC) has a low error and is simpler to implement.

III. Quantization-based Selective Approximation

We first provide an overview of the system and then present the details of quantization-based approximation technique.

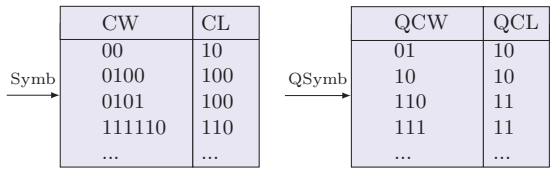
A. Overview

Quantization-based selective approximation aims to increase the effective memory bandwidth and not the capacity, similar to [3], [4]. The data is read/written from/to the DRAM in a compressed form, and compression and decompression take place in the memory controller, which means the compression is transparent to other units. Two bits are stored in a small metadata cache to read/write the required compressed bursts from/to the DRAM like [3], [4].

B. Quantization-based SLC

Quantization-based selective approximation (QSLC) uses the same high-level approach as TSLC [4]. It also uses two compression modes: (1) lossless compression mode, and (2) lossy compression mode. The lossy compression mode is only applied when a compressed size has a few bytes above a multiple of MAG. Therefore, QSLC also requires a *bit budget*, *comp size*, *extra bits*, and a *threshold* to determine when to use approximation opportunistically like TSLC. The *bit budget* is a multiple of MAG i.e. 32, 64, 96, or 128 bytes. The number of bits above the *bit budget* gives us *extra bits*. We use "bytes above MAG" for simplicity of discussion, however, as a compressed block size is not always a multiple of a byte, the implementation uses size in bits. The *threshold* is the number of bits defined by a user that can be safely approximated and approximation is applied when the *extra bits* are within the *threshold*. A block is stored uncompressed when the *comp size* is more than its uncompressed size as well as when the *comp size* is less than 32 bytes as a memory controller cannot fetch less than 32 bytes. More details about the computation of the *bit budget*, *comp size*, *extra bits*, and *threshold* can be found in [4].

A key challenge of selective approximation is to find the number of symbols needed to be approximated to decrease the



(a) Normal Huffman table. (b) Quantized Huffman table.

Fig. 2: Quantization-based SLC.

compressed size to a multiple of MAG. While the state-of-the-art TSLC first determines the number of symbols and then only approximates these symbols, it not only requires additional hardware, but it also results in a higher error as it uses truncation for approximation. TSLC employs value prediction and further optimization that reduces unneeded approximation, but it increases its complexity. Since it is not trivial to find the symbols that contribute extra bytes, one simple way is to approximate the whole block. QSLC approximates the whole block using quantization. Quantization is applied to lower-order bits of all symbols of the block selected for approximation. As the quantization error is uniformly distributed across all symbols of a block, it results in substantially less error than TSLC. TSLC guarantees that approximated blocks are an exact multiple of MAG. While QSLC does not guarantee that the approximated block will be an exact multiple of MAG after the quantization, we can use TSLC as a reference to estimate the required approximation.

QSLC uses quantization to implement the selective lossy compression mode. We use the compressor and decompressor proposed by Lal et al. [3] as base designs and adapt them to implement QSLC. QSLC uses two Huffman tables for compression as shown in Figure 2. A Huffman table is a small lookup table that stores codewords (CWs) and their code lengths (CLs). The table is indexed by a symbol to get its CW and CL. The first table is a normal Huffman table. The second table is a quantized Huffman table that stores quantized codewords (QCWs) and their code lengths (QCLs). The second table is built in a similar way to the first table except that the symbols are quantized before generating the codewords. More details about the generation of Huffman codewords can be found in [3]. We quantize the least significant bits as they introduce the least error and vary the number of quantization bits (4, 8, 12, and 16). Our baseline E^2MC uses 16-bit symbols, thus, we only quantize the first symbol of a 32-bit word. The quantized Huffman table is used for the lossy compression mode while the normal Huffman table is used for the lossless compression mode. The quantized symbols are expected to have smaller codewords compared to no quantization as several unquantized symbols will be mapped to a single symbol with a higher probability. For example, the two symbols which only differ in a single least significant bit result in two separate symbols with their different codewords in a normal Huffman coding, however, when we quantize the least significant bit, it will result in a single quantized symbol with a probability equal to the sum of probabilities of two symbols. The quantized symbol with a higher probability will have a smaller codeword, leading to a smaller compressed size. Figure 2 shows that the quantized Huffman table, in general, has smaller codewords.

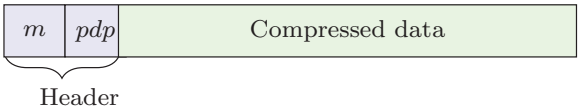


Fig. 3: Simplified structure of a compressed block for QSLC.

TABLE II: Comparison of the area and power overhead.

	Compressor			Decompresor		
Technique	Freq (GHz)	Area (mm^2)	Power (mW)	Freq (GHz)	Area (mm^2)	Power (mW)
TSLC-OPT	1.43	0.00830	1.620	0.80	0.00030	0.210
QSLC-4b	1.43	0.15000	4.200	0.80	0.05200	1.230
QSLC-8b	1.43	0.09000	2.800	0.80	0.02600	0.910
QSLC-12b	1.43	0.00350	1.150	0.80	0.00370	0.600
QSLC-16b	1.43	0.00070	0.240	0.80	0.00001	0.001

The base Huffman decompressor is also modified to implement QSLC. The main modification is the addition of a second decompressor lookup table (De-LUT) to store the quantized symbols along with unquantized symbols and some arbitration logic to select the appropriate De-LUT.

C. Simplified Structure of a Compressed Block

Figure 3 shows that QSLC greatly simplifies the structure of a compressed block. The header consists of 1-bit (*m*) to indicate the compression mode that could be either lossless or lossy and 3 parallel decoding pointers (*pdp*) to reduce the decompression latency [3], [4]. The size of a pointer is N bits, where 2^N is the block size in bytes. QSLC removed 10 bits from the header which are required for TSLC implementation. TSLC uses 6 bits to store the index of the first approximated symbol and 4 bits to store the number of approximated symbols. These fields are not required for QSLC as it does not need to select the symbols for approximation.

D. Quantitative Comparison of Hardware Overhead

We implemented TSLC and QSLC in RTL and then synthesized the designs using Synopsis design compiler version K-2015.06-SP4 targeting 32 nm technology node to compare the overhead of the proposed QSLC and the state-of-the-art TSLC. Table II shows the frequency at which we can run these designs along with the hardware overhead of extending E^2MC with TSLC and QSLC with a different number of quantization bits. The estimated area and power of QSLC-4b and QSLC-8b is higher than the TSLC as QSLC requires additional SRAM tables to store the quantized codewords and symbols to implement the lossy compression mode. The overhead decreases for higher quantization as the number of quantized codewords and symbols decreases. We recommend using QSLC-12b or QSLC-16b as they have a similar performance to TSLC (see Section V). The table shows that the area and power overhead of QSCL-16b are 12.1× and 7.6× lower than the area and power overhead of TSLC, respectively.

TABLE III: Simulator configuration.

#SMs	16	L1 $ size/SM	16 KB
SM freq (MHz)	822	L2 $ size	768 KB
Max #Threads/SM	1536	#Registers/SM	32 K
Max CTA size	512	Shared memory/SM	48 KB
Memory type	GDDR5	# Memory controllers	6
Memory clock	1002 MHz	Memory bandwidth	192.4 GB/s
Bus width	32-bit	Burst length	8
E^2MC compressor	46 cycles	E^2MC decompressor	20 cycles
TSLC compressor	60 cycles	TSLC decompressor	20 cycles
QSLC compressor	60 cycles	QSLC decompressor	20 cycles

TABLE IV: Benchmarks used for experimental evaluation.

Name	Short Description	Input	Error Metric
JM	Intersection of tri. [10]	400 K tri. pairs	Miss rate
BS	Options pricing [8]	4 M options	MRE
DCT	Discrete trans. [8]	1024×1024 img.	Image diff.
FWT	Fast walsh trans. [8]	8 M elements	NRMSE
TP	Matrix transpose [8]	1024×1024	NRMSE
BP	Perceptron train. [9]	64 K elements	MRE
NN	Nearest neigh. [9]	20 M records	MRE
SRAD1	Anisotropic diff. [9]	1024×1024 img.	Image diff.
SRAD2	Anisotropic diff. [9]	1024×1024 img.	Image diff.

The area and power consumption of GTX 580 is 540 mm^2 and 240 W, respectively. The area and power cost of TSLC is about 0.0015% and 0.0008% of GTX580. In contrast, the area and power cost of QSLC-16b is about 0.0001% and 0.0001% of GTX580. TSLC adds 5.6% of the area of E^2MC, while QSLC-12b and QSLC-16b only add 4.7% and 0.5% of the area of E^2MC, respectively.

IV. Experimental Setup

A. Simulator

We use gpgpu-sim v3.2.1 [6] and modify it to integrate E^2MC, TSLC, and QSLC techniques. We configure gpgpu-sim to simulate a GPU similar to NVIDIA's GTX580. Table III shows the simulator configuration along with (de)compression latencies. More details of the simulator can be found in [6].

We use E^2MC with 16-bit symbols, 4 PDWs, and 20 million sampling instructions as a baseline lossless compression [3]. We synthesize RTL designs of E^2MC, TSLC, and QSLC to find the number of cycles required for compression and decompression. While E^2MC takes 46 cycles to compress and 20 cycles to decompress a block, TSLC and QSLC require 60 cycles for compression and 20 cycles for decompression for a memory clock of 1002 MHz. We modify GPUSimPow [7] with power models to estimate energy consumption.

B. Benchmarks

We use the same benchmarks as well as the same metrics (speedup and application-specific error) as used by Lal et al. [4], which facilitates a direct comparison. Table IV shows the benchmarks along with the error metrics. These are memory-bound and amenable to approximation benchmarks from CUDA SDK [8], Rodinia [9], and AxBench [10].

Like Lal et al. [4] and similar to others [11]–[13], QSLC requires programmer annotations to find the loads that can be safely approximated and a value of lossy threshold that meets the quality requirements. We adopt Lal et al. [4] coarse-grain model of extended cudaMalloc() to specify if a memory region is safe to approximate or not and to set a lossy threshold.

V. Results

We present results for four variations of QSLC for four different quantization levels (4, 8, 12, and 16 bits). We provide speedup and application-specific error for individual benchmarks and the geometric mean of the speedup of all benchmarks. We conduct experiments using a lossy threshold of 16 bytes and 32 bytes MAG size.

A. Speedup

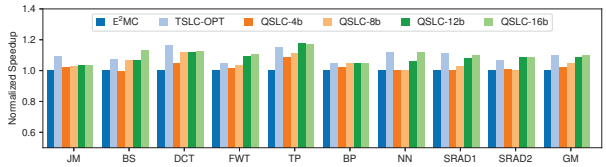
Figure 4 shows the speedup and error for TSLC-OPT and QSLC using 4, 8, 12, and 16-bit quantization. The speedup is normalized to E^2MC [3]. The geometric mean of the speedup is 2%, 4.8%, 8.5%, and 10.1% for 4, 8, 12, and 16-bit quantization, respectively. Both the speedup and error increase with the increase in the number of quantization bits which is expected because more blocks get compressed to a multiple of MAG when the number of quantized bits is increased. The results are interesting as it is possible to achieve up to 17% speedup and an average speedup of more than 10% with only 0.64% maximum loss in accuracy.

The average speedup of QSLC with 12-bit quantization (8.5%) and 16-bit quantization (10.1%) is close to the average speedup of TSLC-OPT (9.5%). As discussed before, QSLC does not guarantee that a compressed block will be a multiple of MAG after approximation, however, by comparing the speedup of TSLC-OPT and QSLC, we can infer that 12-bit and 16-bit quantizations are roughly approximating the same number of blocks to a multiple of MAG as TSLC-OPT.

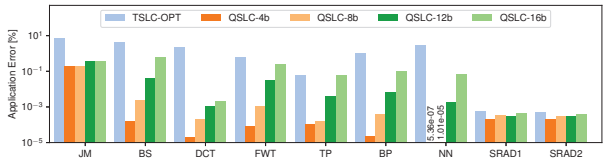
While QSLC provides a similar speedup to TSLC-OPT, the approximation error of QSLC is drastically lower as shown in Figure 4. TSLC-OPT first uses a value similarity-based prediction to reconstruct the approximated symbols with the aim to reduce approximation error and then further adds extra nodes in the tree to reduce unneeded approximation. While these optimizations significantly reduce the approximation error, they also increase the complexity of the TSLC-OPT. Moreover, even after the complex optimizations, TSLC-OPT can result in a higher error. For example, *JM* has a 7.3% error, while *BS* has a 4.4% error.

Figure 4b shows that, in contrast to TSLC-OPT, QSLC has a much lower error. The error ranges from a minimum of 10^{-7}% for *nn* with 4-bit quantization to the maximum of 0.64% for *BS* with 16-bit quantization. The main reason for the high error in TSLC-OPT is that it approximates symbols using truncation which ensures that a compressed block is a multiple of MAG, however, it can result in a higher error. In addition, the consecutive symbols are truncated which can also lead to increased error. In QSLC, the approximation error is uniformly distributed as quantization is applied to lower-order bits of all symbols of a block. As a result, the maximum error is limited to 0.64% and the average error is only 0.2%.

In addition to the application-specific error, we also calculated the mean relative error for an individual benchmark



(a) Speedup



(b) Error

Fig. 4: Speedup and error of TSLC-OPT and QSLC.

which enables averaging the error across all benchmarks. While the geometric mean of the mean relative error for all benchmarks is 0.99% for TSLC-OPT (the best error after all optimizations), the geometric mean of the mean relative error for QSLC is 0.0006%, 0.004%, 0.04%, and 0.2% for 4, 8, 12, and 16-bit quantization, respectively. QSLC with 16-bit quantization provides 5× lower error than TSLC-OPT.

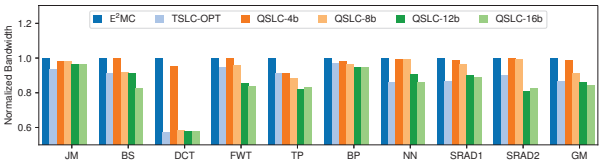
The main reason for differences in the error between TSLC-OPT and QSLC is that both techniques approximate blocks differently. While TSLC-OPT introduces a high error due to truncation which is significantly reduced after optimizations, QSLC results in a low error as only the least significant bits are quantized and the error is uniformly distributed.

B. Bandwidth and Energy Reduction

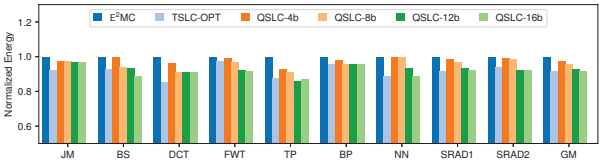
Figure 5 shows the bandwidth, energy, and energy-delay-product (EDP) reductions for QSLC normalized to E^{2}MC. The savings in bandwidth, energy, and EDP are due to the reductions in 32-byte burst transactions. The geometric mean of the reduction in memory bandwidth is 2%, 8.6%, 14.2%, and 15.8% for 4, 8, 12, and 16-bit quantization, respectively. Figure 5b and Figure 5c show the reduction in energy and EDP for QSLC. The geometric mean of the energy and EDP reductions is 2.0%, 6.0% for 4-bit, 4.0%, 10.0% for 8-bit, 7.2%, 16% for 12-bit, and 8.4%, 18.2% for 16-bit quantizations, respectively. The geometric mean of the bandwidth, energy, and EDP reductions is 13.3%, 8.3%, and 17.3% for TSLC-OPT, respectively. QSLC with 16-bit quantization has slightly higher savings in bandwidth, energy, and EDP compared to TSLC-OPT. These results also commensurate with the speedup results presented in Section V-A.

C. QSLC Sensitivity to Block Size

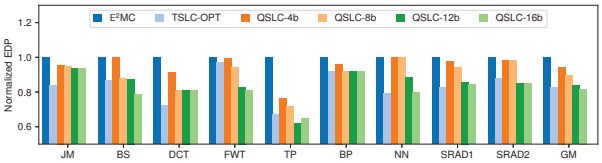
Lal et al. [4] showed the significance of MAG-aware approximation to different MAG sizes. We extend this study and show that MAG-aware approximation is also needed across different block sizes, thus, widening its application. To conduct sensitivity analysis to different block sizes, we performed experiments for the baseline E^{2}MC and QSLC



(a) Bandwidth reduction



(b) Energy reduction

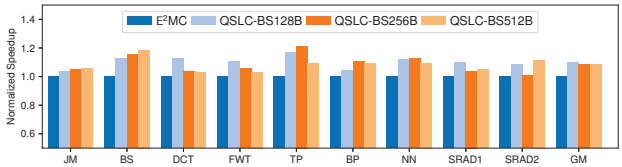


(c) EDP reduction

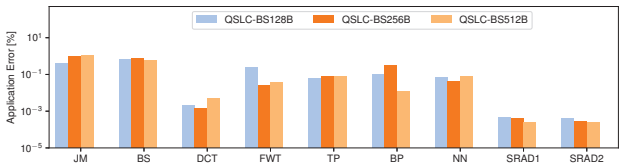
Fig. 5: Bandwidth, energy, and EDP reduction for QSLC.

with block sizes of 128, 256, 512 bytes using a MAG of 32 bytes. The geometric mean of the raw compression ratio for E^{2}MC is 1.54, 1.57, 1.58 and the effective compression ratio is 1.31, 1.45, 1.47 for the block size of 128, 256, and 512 bytes, respectively. We notice that for a larger block size the difference between the raw and effective compression ratio shrinks, leading to a relatively higher compression ratio. The reason for the smaller difference with a larger block size is two-fold, first, there is a smaller number of blocks that need to be compressed, and second, a larger block has a higher probability to be a multiple of MAG. For example, assume there are two compressed blocks with a size of 36 bytes each. This will result in four MAG fetches for a block size of 128 bytes, while only three for a block size of 256 bytes, assuming the same compressibility for the two consecutive smaller blocks and one larger block.

Our compression technique operates at the granularity of L2 cache line size as that is the size of memory requests processed by the memory controller. Thus, for conducting the block size sensitivity analysis, we increase the L2 cache line size and reduce the number of sets to keep the L2 cache size the same for different block sizes. The compression and decompression latency is also adjusted accordingly to the block size. Figure 6 shows the change in speedup and error for different block sizes for QSLC using 32-byte MAG. The average speedup slightly decreases due to the reduced gap between the effective and raw compression ratio, however, the decrease in speedup is rather small. Moreover, for some benchmarks, the speedup is even higher compared to smaller block sizes. This is likely caused by a change in the cache configuration. For example,



(a) Speedup



(b) Error

Fig. 6: Speedup and error for different block sizes for QSLC using 32-byte MAG.

performance for the block size of 256 and 512 bytes even without compression, is on an average 15% and 70% lower compared to the block size of 128 bytes. We also conducted experiments with 4× and 8× L2 cache size, however, the trend remains the same as it is not completely possible to isolate the effects of change in cache configuration. In general, a bigger block can reduce the difference between the raw and effective compression ratio, however, as long as a large MAG exists we can use QSLC to gain performance. Figure 6b shows that, in general, the error for a larger block is slightly lower which is kind of expected due to less approximation.

VI. Related Work

Approximate computing usage for GPUs has increased as GPUs have become popular accelerators [12], [14]–[17]. Approximate computing has been used at software, hardware, and hybrid levels. For example, Samadi et al. [14], [15] used static compilers to automatically generate different versions of GPU kernels. A run-time system selects the appropriate kernel based on application requirements. NPU [13] uses program transformation to select and train a neural network to mimic a region of imperative code. A compiler replaces the original code with an invocation of a low-power accelerator called a neural processing unit (NPU). Arnau et al. [17] proposed a hardware-based memoization technique to remove redundant fragment computations. Sathish et al. [18] used lossy compression for GPU. They always truncate the least significant bits specified by using a modified *cudaMemCpy()* API. Warp Approximation [12] dynamically detects intra-warp value similarity using hardware and executes only a single representative thread and uses its value to approximate the values of other threads in the same warp. Sartori and Kumar [16] used approximate computing to eliminate the control and memory divergence on GPUs. The most closely related work is the MAG-aware selective lossy compression (TSLC) by Lal et al. [4]. While both TSLC and QSLC selectively approximate a few bytes above MAG to increase the performance and energy efficiency, QSLC drastically reduces the approximation error.

VII. Conclusions

Memory compression helps to reduce the memory bandwidth and storage requirement demands of ever-increasing large data sets, however, memory compression techniques lose a significant compression ratio (average 20%) due to the large memory access granularity (MAG) of multi-/many-core systems. The loss in the compression ratio is caused by a significant fraction of compressed blocks that have a few bytes above a multiple of MAG. While MAG-aware selective approximation has been used to increase the effective compression ratio and the performance gain, it results in a high error which is reduced by using complex optimizations. We propose a simple quantization-based approximation technique (QSLC) that can also selectively approximate a few bytes above MAG. While the quantization-based approximation technique has a similar performance to the state-of-the-art tree-based selective approximation, the average error for the quantization-based technique is 5× lower. We implement hardware and show that the area and power overhead of the quantization-based technique is 12.1× and 7.6× lower than the state-of-the-art, respectively. Our sensitivity analysis to different block sizes shows that bigger block sizes can reduce the difference between raw and effective compression ratio, however, as long as a large MAG exists we can use a selective approximation to gain significant performance.

References

[1] Chen *et al.*, "C-Pack: A High-Performance Microprocessor Cache Compression Algorithm," *IEEE Transactions on VLSI Systems*, 2010.
[2] Kim *et al.*, "Bit-plane Compression: Transforming Data for Better Compression in Many-core Architectures," in *Proc. 43rd ISCA*, 2016.
[3] Lal *et al.*, "E^{2}MC: Entropy Encoding Based Memory Compression for GPUs," in *Proc. 31st IPDPS*, 2017.
[4] Lal *et al.*, "SLC: Memory Access Granularity Aware Selective Lossy Compression for GPUs," in *Proc. DATE*, 2019.
[5] Lal *et al.*, "A Case for Memory Access Granularity Aware Selective Lossy Compression for GPUs," in *ACM SRC, MICRO*, 2018.
[6] Bakhoda *et al.*, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *Proc. ISPASS*, 2009.
[7] Lucas *et al.*, "How a Single Chip Causes Massive Power Bills - GPUSimPow: A GPGPU Power Simulator," in *Proc. ISPASS*, 2013.
[8] NVIDIA, "CUDA: Compute Unified Device Architecture," 2007, http://developer.nvidia.com/object/gpucomputing.html.
[9] Che *et al.*, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proc. IISWC*, 2009.
[10] Yazdanbakhsh *et al.*, "AxBench: A Multiplatform Benchmark Suite for Approximate Computing," *Proc. DATE*, 2017.
[11] Esmaeilzadeh *et al.*, "Architecture Support for Disciplined Approximate Programming," in *Proc. 17th ASPLOS*, 2012.
[12] Wong *et al.*, "Approximating Warps with Intra-warp Operand Value Similarity," in *Proc. HPCA*, 2016.
[13] Esmaeilzadeh *et al.*, "Neural Acceleration for General-Purpose Approximate Programs," in *Proc. 45th MICRO*, 2012.
[14] Samadi *et al.*, "SAGE: Self-tuning Approximation for Graphics Engines," in *Proc. 46th MICRO*, 2013.
[15] Samadi *et al.*, "Paraprox: Pattern-based Approximation for Data Parallel Applications," in *Proc. 19th ASPLOS*, 2014.
[16] Sartori *et al.*, "Branch and Data Herding: Reducing Control and Memory Divergence for Error-tolerant GPU Applications," in *Proc. PACT*, 2012.
[17] Arnau *et al.*, "Eliminating Redundant Fragment Shader Executions on a Mobile GPU via Hardware Memoization," in *Proc. 41st ISCA*, 2014.
[18] Sathish *et al.*, "Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads," in *Proc. PACT*, 2012.