Multi-Prediction Compression: An Efficient and Scalable Memory Compression Framework for GP-GPU

Hoyong Jin, Donghun Jeong, Taewon Park, Jong Hwan Ko, *Member, IEEE*, and Jungrae Kim

Abstract—Data-intensive applications and throughput-oriented processors demand more memory bandwidth. Memory compression can provide more data beyond physical limits, yet new data types and smaller block sizes are challenging. This paper presents a novel and lightweight memory compression framework, Multi-Prediction Compression (MPC), to increase the effective memory bandwidth. Based on multiple prediction models and data-driven algorithm tuning, MPC can provide 31.7% better compression than state-of-the-art (SOTA) algorithms for 32B blocks. Moreover, MPC is hardware-friendly and scalable to support a growing number of data patterns.

Index Terms—Graphics processors, memory hierarchy, data compaction and compression

1 Introduction

Throughput-oriented processors, such as GP-GPU can provide high computational throughput to data-intensive applications. While such processors rapidly increase their computational throughput, the ability to feed data grows less fast, leading to a symptom known as the *memory wall* [1].

Memory compression is promising for overcoming the wall by providing more bandwidth than the physical limits. An efficient compression algorithm, however, must address the following challenges. First, the compression must be lossless not to affect functionality. Second, decompression must be simple not to increase the read latency significantly. Finally, it must compress data at a block granularity to support random access. Finding such an algorithm has required a large amount of human labor, yet the output compression ratio is modest (e.g., 1.71 for 32B blocks). Moreover, the manual effort has to recur as digital transformation and AI bring new data patterns and types (e.g., int4, bfloat) to computing.

This paper proposes a novel compression framework, called *Multi-Prediction Compression (MPC)*, to improve the compression ratio, automate algorithm tuning based on data sets, and support an increasing number of data patterns. MPC is composed of multiple prediction models and a shared residue-encoder (Fig. 1). A prediction model predicts a value from another data within the block. If a prediction is accurate, the resulting residue between the original and predicted values is small and can be efficiently compressed by the encoder. A receiver can losslessly restore the original data with the model ID, the base value, and the compressed residues.

- *Hoyong Jin, Donghun Jeong, Jong Hwan Ko, and Jungrae Kim are with the Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, Korea. E-mail: {jin8495, dounghun22}@g.skku.edu, {jhko, dale40}@skku.edu.*
- *Taewon Park is with the Department of Semiconductor and Display Engineering, Sungkyunkwan University, Suwon 16419, Korea. E-mail: pdtowctor@g.skku.edu.*

Manuscript received 19 Apr. 2022; revised 15 May 2022; accepted 17 May 2022. Date of publication 24 May 2022; date of current version 14 July 2022.
This work was supported in part by the National Research Foundation of Korea (NRF) Grants funded by the Korea government (MSIT) under Grants 2020R1C1C1011419 and 2020M3H2A1076786 and in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korea government (MSIT) under Grant 2020-0-01821, and in part by the SungKyunKwan University and the BK21 FOUR (Graduate School Innovation) funded by the Ministry of Education (MOE, Korea) and National Research Foundation of Korea (NRF).
(Corresponding author: Jungrae Kim.)
Digital Object Identifier no. 10.1109/LCA.2022.3177419

While prediction accuracy is the key to improving compression ratio, a single model cannot provide high accuracy as data have diverse patterns. MPC utilizes multiple models to process different patterns differently. All models share the same structure, but each has customized parameters based on an offline analysis. The analysis automatically extracts parameters of a model from its target data set without human intervention. The models are lightweight such that MPC can instantiate multiple models in hardware at a meager cost (primarily metal wires). With parallel models, MPC can provide high accuracy to diverse patterns. Moreover, making the per-pattern prediction models inexpensive allows MPC supporting new and emerging patterns by adding prediction models without a significant cost.

Parallel models and automatic fine-tuning allow MPC high compression ratios despite the simplicity of the model. For 32B blocks, which is the access granularity of GP-GPU sector caches, MPC can achieve a compression ratio of 2.26 over integer and floating-point applications, which is 31.7% better than SOTA algorithms. GP-GPUs can gain 69.0% speed-up for I/O-bound applications in limited bandwidth situations, 14.5% faster than SOTA algorithms. MPC has similar hardware overheads in area and latency (3-cycle compression and 4-cycle decompression) to the SOTA algorithms.

In this paper, we make the following contributions:

- We propose a novel compression framework, called MPC, to improve the compression ratio for GP-GPU.
- We provide a prediction model and automatic parameter extraction algorithm to exploit value correlations better than manual efforts.
- We propose lightweight prediction models to support growing data patterns with parallel models at a low cost.

2 Background & Motivation

2.1 GP-GPU Memory Sub-System

GP-GPU memory sub-system is highly optimized to provide high data bandwidth to cores. NVIDIA GPUs utilize 128B cache lines to simultaneously provide 4B data to a group of 32 threads (i.e., a WARP). However, transferring an entire cache-line can result in wasted memory bandwidth with irregular memory accesses, and NVIDIA introduced sector cache since Maxwell [2]. NVIDIA sector cache splits a 128B cache-line into 4 32B sectors where each sector is the unit of memory access.

Although the reduced granularity can save memory bandwidth from overfetching, it brings challenges to memory compression. A larger data block is easier to compress because it has more redundancy (e.g., more repeated values) and can amortize encoding overheads (e.g., metadata).

2.2 Existing Memory Compression Algorithm

Academia and industry have studied memory compression to improve memory capacity and bandwidth. Most algorithms utilize frequent patterns to reduce size. Frequent Pattern Compression (FPC) [3] exploits patterns of zero value, narrow values, and repeated values. C-Pack [4] utilizes *Temporal locality*, where data is likely to reappear in the near future. Base-Delta-Immediate (BDI) [5] and Bit-Plane Compression (BPC) [6] exploit narrow values and *Low dynamic range*, where data within a block have similar values that can be compressed with delta-encoding. BDI uses a single value or zero-immediate basis, while BPC applies a delta-based transform to improve inherent compressibility. However, the delta-based algorithms do not fully utilize diverse value correlation by fixing the base positions and using its value directly as the base. As a result, their compression ratios are modest in GP-GPU (up to 1.71 for 32B blocks). Moreover, the above works are based on manual efforts to extract frequent patterns and are

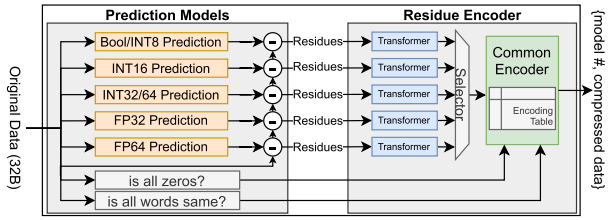

Fig. 1. An overview of the MPC compressor.

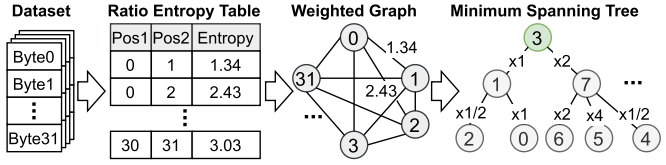

Fig. 3. The prediction parameter extraction flow (offline).

not scalable to new data patterns and types. SC2 [7] exploits repeated values with Huffman-coding and can expand cache capacity up to $4\times$. However, it requires rewriting the stored data on a significant change on frequent values.

HyComp [8] is similar to MPC in applying multiple algorithms and selecting the best output. However, it mechanically integrates independent algorithms and its hardware complexity grows as the number of algorithms increases. [9] proposed a prediction-based compression for float-point data streams. It utilizes a hashtable to predict a residue from history. Although it provides up to 1.66 compression ratio, it is not randomly accessible and requires significant resources in hardware.

2.3 Value Correlation

Memory compression reduces data size by exploiting redundancy. Among redundancies, prediction models in MPC primarily target *Value Correlation*, where a value is highly related to values at other positions and can be predicted from them. To quantify value correlation, we define a metric called *Ratio Entropy (RE)*. For two positions within a block, i and j, we define RE as the information entropy of quantized ratios of their values (i.e., $RE(i,j) = InfoEntropy\,(\lfloor log2(Data[i]/Data[j])\rfloor)$) for a given data set. A low RE indicates that the two positions have values highly correlated by a fixed ratio.

Fig. 2 presents REs of the first 8 bytes out of 32B blocks in our data sets, classified by data types. If two positions are highly correlated (i.e., low RE), the pair is marked in a bright color. If two positions have unrelated values, it is marked in dark. 32b integer (INT32) blocks have high correlations among the same positions across words. Byte 3 and 7 have high correlation as they are the most significant bytes (our data sets are in the little-endian format), and a small difference across words does not change MSB. Similarly, (byte 2, byte 6), (byte 1, byte 5), and (byte 0, byte 4) show reduced but good REs. Moreover, byte 2 and byte 3 are correlated because a small value has the same sign information in those bytes.

In 32b floating point (FP32) blocks, bytes at the same positions still show correlation at reduced levels. It is because FP32 changes its mantissa significantly and its exponent modestly with a small difference. 64b floating point (FP64) show similar correlations among sign and exponent fields across words (not shown in the figure). Unlike others, FP64 blocks show relatively high correlations among byte 0, 1, 2, and 3. FP64 is often overprovisioned to

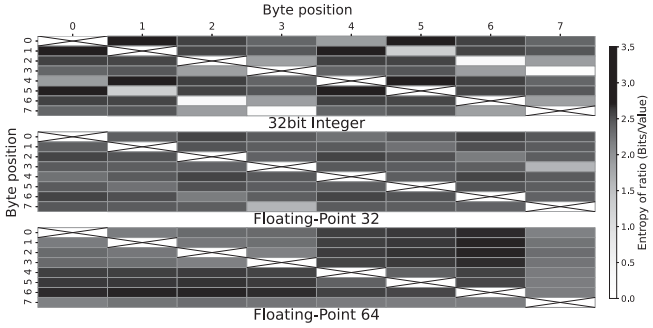

Fig. 2. A table of ratio entropies measured in our data set.

hold FP32 data, in which case the lower mantissa bytes have the same value of 0.

Different data types show different correlations. Some of the above correlations are apparent such that prior work hand-picked them in developing compression algorithms, while others are blurry such that a naïve algorithm may miss them. MPC provides an automated approach to extract correlations and fine-tune its predictions.

3 Multi-Prediction Compression

MPC is a compression framework that can efficiently compress diverse data patterns in GP-GPU even with small (i.e., 32B) blocks. MPC is composed of a set of prediction models and a common residue-encoder. A model predicts a value from another value within the same block with an optional shift operation. The predicted value is used to generate a residue, which is the difference between the actual and the predicted values. If the prediction is accurate, the common encoder can efficiently compress the resulting residues block with small and similar values. MPC applies multiple models in parallel and selects the best prediction whose residue can be compressed the most. The encoder selects the best residue and uses a shared encoding engine to generate compressed data.

3.1 Prediction Models

Prediction models must have high accuracy to generate small residues, while their complexity must be low to reduce latency increases and hardware overheads. To meet these conflicting requirements, prediction models in MPC are simple yet optimized. Moreover, the models are auto-generated from data sets instead of human efforts.

A MPC prediction model predicts a value from another value at a different position (i.e., base position) and a ratio factor, using $Pred_i = Data[BasePosition_i] \times RatioFactor_i$. The ratio factors are constrained to power-of-2 s. This simple structure eliminates hardware overheads in implementing the models. Once an offline analysis extracts the parameters (base position and ratio factor), compressor/decompressor hardware can implement the model using metal wires only for the fixed selection and power-of-2 multiplication.

MPC utilizes a novel algorithm to extract the parameters automatically from a data set (Fig. 3). For a data set, it first calculates REs for every position pair within a block. A byte position in 32B block has 31 REs with the remaining positions. A low RE indicates that the two positions can be a good base for each other. However, constructing a model using the position pairs with the lowest REs can lead to an irreversible model with a circular dependency. MPC utilizes the Minimum Spanning Tree (MST) search algorithm to avoid circular dependency and minimize the number of root bases. The REs can turn into a weighted graph, where a node represents a position, and an undirected edge represents the RE between the two nodes. A tree in the graph can represent a base relation in MPC with a parent as the base of the child. Furthermore, an MST is a block-level base relation whose sum of REs is the minimum and has no circular dependency.

To support different data types, MPC classifies blocks in the data set into five groups based on their primary data type: Boolean or INT8, INT16, INT32 or INT64, FP32, and FP64. Then MPC generates a prediction model for each group. Once parameter extraction is complete, a compressor/decompressor implements the parameters using wires. In addition to the five auto-generated prediction models, we

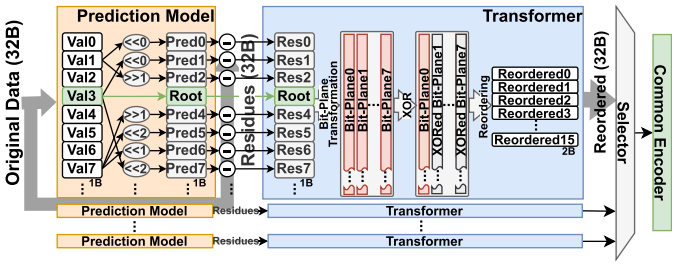

Fig. 4. Hardware implementation of the prediction model and the transformer. The prediction model is based on Fig. 3 (only the first 8 bytes shown in the figure)

TABLE 1
Encoding Table Used by the Common Encoder

Pattern	Length	Code
0	4-bit	{4'b0011}
0 s (run length 2∼15)	7-bit	{3'b010, RunLength[3:0]}
Single 1	7-bit	{3'b011, OnePosition[3:0]}
Consecutive two 1 s	8-bit	{4'b0000, StartingOnePosition[3:0]}
Front half 0 s	12-bit	{4'b0001, NoneZeros[7:0]}
Back half 0 s	12-bit	{4'b0010, NoneZeros[7:0]}
Default	17-bit	{1'b1, Default[15:0]}

add two custom models for two of the most frequent patterns: all-zero block and all-words-same. Note that data type information is unavailable during compression, and MPC applies multiple models in parallel and selects the best output (Fig. 1). The selected model number is stored along with the compressed data.

3.2 Residue Encoder

The shared encoder received residues and generates compressed data (Fig. 4). Instead of compressing individual residues, it encodes an entire residue block for better outputs. It first transforms a residue block into a compression-friendly format. A selector chooses a transformed residue that can be compressed the most. Then the shared engine does the actual compression based on an encoding table.

The first stage transforms a block of small values into frequent patterns and long sequences of 0 s. It first transposes 32 8-bit residues into 8 32-bit bit-planes, similar to [6]. Residue bits at the same position build a 32-bit bit-plane. Then it XORs each bit-plane with its upper bit-plane to increase the number of 0 s. Then based on an offline analysis on conditional probability across bit positions, it places the XORed bits in an order that bits that are likely to be 0 s at the same time are placed next to each other to generate consecutive 0 s. Different data types have different probabilities, so that each prediction model uses its own transformer. However, transformers are inexpensive in hardware as transposing and reordering require no logic gate.

After transformation, most data blocks, regardless of their data type, have long sequences of 0 s from the per-data-type prediction and transformation. This shared property enables efficient sharing of the encoder, which occupies the largest chip area in MPC, and allows MPC scaling out to diverse patterns. Among transformed residues, MPC selects one with the longest zero-run as an input to the encoder. The encoder receives a transformed bitstream as a sequence of 16-bit symbols. It encodes each 16-bit symbol based on the encoding table in Table 1. There are seven patterns, including one for the unmatched pattern (default). As inputs are zero-dominant, zero symbols can be encoded via run-length encoding (the 2nd pattern in the table) to achieve multi-symbol savings per encoding. Single zero symbol is also frequent to assign a shorter encoding (4 bits). The remaining 4 are used to cover the most frequent patterns with some 1 s within a symbol.

4 Evaluation

We evaluate benefits and costs of MPC by measuring its compression ratio, execution time, chip area, and latency. We compare MPC against state-of-the-art memory compression algorithms (BPC, FPC, SC2, BDI, and C-Pack), showing that MPC can significantly improve compression ratio and execution time with a similar silicon footprint and latency.

Because MPC is data-dependent, we use a large number of applications to improve confidence in results (Table 2). We use a cycle-level simulator [17], GPGPU-Sim 4.0, to capture memory traffic between LLC and DRAM in the NVIDIA Titan-V configuration. We ran 59 applications from 7 benchmark suites to collect 36.23 GB of 32B blocks. For MPC training, we sorted the applications based on the information entropy of the memory traffic (excluding all-zero blocks and all-words-same blocks). The first 3 out of every 4 applications in descending order are used to extract model parameters. In those applications, 80% of randomly-selected data blocks are used to extract model parameters, and the remaining 20% are used as the seen test set. The fourth applications are used as the unseen test set. We measure compression ratios only on the seen and unseen test sets and do not include blocks used in the parameter extraction.

Fig. 5 presents compression ratios of BPC, FPC, SC2, BDI, C-Pack, and MPC. BPC shows the lowsest compression ratio (0.79). While it shows the best results for 128B blocks, 32B blocks have only eight 4B symbols, leading to fewer deltas and less redundancy to exploit for BPC. FPC, which uses zero value, narrow value, and repeated value patterns, shows a low overall compression ratio (1.50). SC2 achieves a moderate compression ratio (1.56), utilizing statistics of data blocks. BDI exploits the low dynamic range and can compress more efficiently (compression ratio = 1.67). Overall, C-Pack performs the best among the prior work by exploiting spatial locality (compression ratio = 1.71). MPC achieves superior compression ratios of 2.31, 2.06, and 2.26 for the seen test set, the unseen test set, and overall, respectively. The overall compression ratio is 31.72% better than the best of the prior work (C-Pack). In 40 out of 59 workloads, MPC is the best-performing algorithm. In 5 out of the remaining 19, C-Pack shows the best compression by exploiting the spatial locality, which is not used in the MPC. Note that MPC provides significantly better compression ratios for both integers (2.87) and floating points (2.09).

To estimate performance benefits, we apply MPC to compress link traffic on a packet-based memory interface (e.g., HMC, NVLink, CXL) with reduced bandwidth, in a similar way to [6]. One of the key challenges in future GPUs is decreasing memory bandwidth per computation throughput. NVIDIA GPUs increased their TOP/s 317× during 2012-2020 [18], yet their memory bandwidth modestly increased 8× in the same period. To reflect the decreasing bandwidth per TOP/s trend, we reduce the DRAM bandwidth of the baseline GPGPU-Sim configuration (NVIDIA Titan-V) to 1/8× while keeping the SM count the same. The ratio

TABLE 2
GPGPU-Sim Configuration (Based on NVIDIA Titan-V) and the Evaluated Benchmarks (Abbreviations in Brackets)

Core	1200 MHz, 80 Shader cores, SIMD width=128 (32 x 4), 2048 Threads (64 warps / SM), 96 KB Shared Memory
L1 Cache	32B Line Size (4 Sectors of 128B), 32 KB L1 D-cache, 128 KB L1 I-cache
L2 Cache	1200 MHz, 4.5 MB (96 KB x 48 sub partitions)
Memory	850 MHz 3 stacks of HBM2 (637.5 GB/s total bandwidth)
Memory link	3 full-duplex links, 26.6 GB/s undirectional bandwidth per link, 79.7 GB/ bidirectional bandwidth in total
Benchmark Suites	ISPASS 2009 (I) [10], Lonestar 2.0 (L) [11], Parboil (PA) [12], Polybench (PO) [13], Pytorch (PY) [14], Rodinia 3.1 (R) [15], Shoc (S) [16]

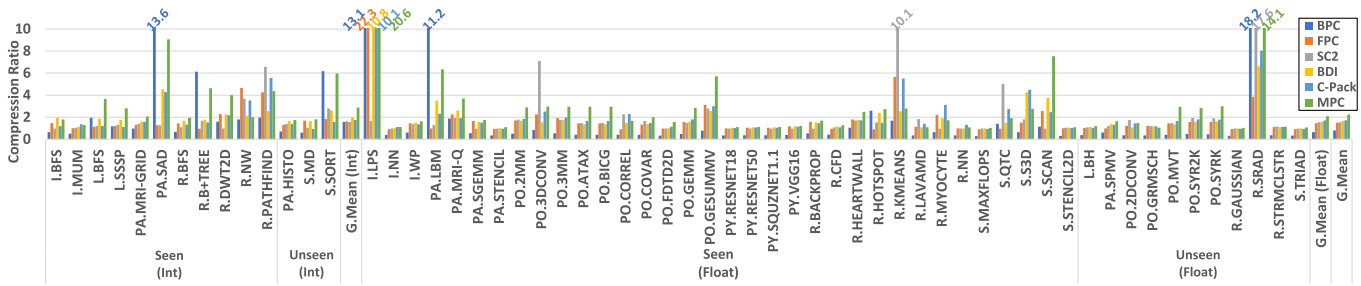

Fig. 5. A comparison of compression ratios (32B blocks).

roughly matches to NVIDIA Hopper architecture, which has $3.4\times$ DRAM bandwidth and $16\times$ FP16 TOP/s than NVIDIA Volta architecture [19], [20]. A processor compresses 32B blocks and packs the compressed data into packets for write operations. The packets can have up to 4 KB payloads with 8B header overhead. Each compressed data has a 9b tag to match out-of-order responses to their requests. DRAM decompresses the received packets and stores the original data. A similar flow applies to reads. Note that MPC is a general compression framework and is applicable to other usages (e.g., cache compression, compressed memory storage) or different block sizes in GP-GPUs.

Fig. 6 presents the execution time measurements with MPC and the best of the prior work, C-Pack. The execution times are normalized to ones without a compression. We classify applications whose execution time degrades by $> 25\%$ in the limited bandwidth as I/O-bound and the others as compute-bound. C-Pack provides 2.2 compression ratio and 54.5% speed-up than no-compression for I/O-bound applications. However, it slows down compute-bound applications by 0.9%, despite its 1.9:1 compression. These applications are less sensitive to enhanced bandwidth from compression and more sensitive to the increased read latency from decompression (9 cycles with C-pack). MPC achieves 69.0% and 4.1% speed-ups for I/O- and compute-bound applications, respectively (34.5% overall). For I/O-bound applications, the better compression ratio (2.8 with MPC versus 2.2 with C-Pack) makes MPC outperform C-Pack. For compute-bound applications, the smaller read latency increase (4-cycle decompression) and higher compression ratio (2.7 with MPC versus 1.9 with C-Pack) minimizes the negative impact of compression.

To evaluate hardware overheads, we synthesized our design of MPC on UMC 28 nm. The overall area for compressor and decompressor are 19,000 μm^2 and 34,000 μm^2, which corresponds to 38 K and 68 K NAND2 gates, respectively. MPC are comparable to C-Pack in area, which requires 40 K gates for its compressor [4]. MPC consumes 13.5 mW and 33.6 mW for compression and decompression, respectively. The combined 47.1 mW takes only 0.019% in 250 W TDP of NVIDIA Titan-V, and can be easily offset by the significant savings in execution time.

5 Conclusion

We present a novel data-driven compression framework called MPC. MPC provides significantly better compression with parallel prediction models and automatic parameter tuning. Despite the improvement, it is hardware-friendly to have small hardware overheads and scalable to support a growing number of data patterns. MPC can provide more data bandwidth and reduce execution time in GP-GPU against diversifying data patterns.

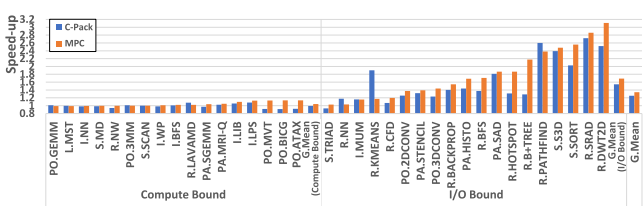

Fig. 6. Execution times of C-Pack and MPC in limited bandwidth situation (1/8×). The execution times are normalized to ones without a compression.

References

[1] M. Thuresson, L. Spracklen, and P. Stenstrom, "Memory-link compression schemes: A value locality perspective," *IEEE Trans. Comput.*, vol. 57, no. 7, pp. 916–927, Jul. 2008.

[2] M. Khairy *et al.*, "Exploring modern GPU memory system design challenges through accurate modeling," 2018, *arXiv:1810.07269*.

[3] A. Alameldeen and D. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," University of Wisconsin-Madison, Madison, Wisconsin, Tech. Rep. 1500, 2004.

[4] X. Chen, L. Yang, R. P. Dick, L. Shang, and H. Lekatsas, "C-pack: A high-performance microprocessor cache compression algorithm," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 8, pp. 1196–1208, Aug. 2009.

[5] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. 21st Int. Conf. Parallel Architect. Compilation Techn.*, 2012, pp. 377–388.

[6] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-plane compression: Transforming data for better compression in many-core architectures," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Architect.*, 2016, pp. 329–340.

[7] A. Arelakis and P. Stenstrom, "SC 2: A statistical compression cache scheme," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Architect.*, 2014, pp. 145–156.

[8] A. Arelakis, F. Dahlgren, and P. Stenstrom, "HyComp: A hybrid cache compression method for selection of data-type-specific compression methods," in *Proc. 48th IEEE/ACM Int. Symp. Microarchitecture*, 2015, pp. 38–49.

[9] P. Ratanaworabhan, J. Ke, and M. Burtscher, "Fast lossless compression of scientific floating-point data," in *Proc. Data Compression Conf.*, 2006, pp. 133–142.

[10] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2009, pp. 163–174.

[11] M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on GPUs," in *Proc. IEEE Int. Symp. Workload Characterization*, 2012, pp. 141–151.

[12] J. Stratton *et al.*, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center Reliable High-Perf. Comput.*, vol. 127, p. 29, 2012.

[13] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to GPU codes," in *Proc. Innov. Parallel Comput.*, 2012, pp. 1–10.

[14] J. Lew *et al.*, "Analyzing machine learning workloads using a detailed GPU simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2019, pp. 151–152.

[15] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization*, 2009, pp. 44–54.

[16] A. Danalis *et al.*, "The scalable heterogeneous computing (SHOC) benchmark suite," in *Proc. 3rd Workshop Gen.-Purpose Comput. Graph. Process. Units*, 2010, pp. 63–74.

[17] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated GPU modeling," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Architect.*, 2020, pp. 473–486.

[18] W. J. Dally, S. W. Keckler, and D. B. Kirk, "Evolution of the graphics processing unit (GPU)," *IEEE Micro*, vol. 41, no. 6, pp. 42–51, Nov./Dec. 2021.

[19] "Nvidia tesla V100 GPU architecture, the world's most advanced data center GPU," NVIDIA Corp., Santa Clara, CA, USA, Tech. Rep. WP-08608-001_v1.1, 2017.

[20] M. Andersch *et al.*, Nvidia hopper architecture in-depth, Mar. 2022. [Online]. Available: https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/