Buddy Compression: Enabling Larger Memory for Deep Learning and HPC Workloads on GPUs

Esha Choukse
Microsoft
esha.choukse@microsoft.com

Michael B. Sullivan
NVIDIA
misullivan@nvidia.com

Mike O'Connor
NVIDIA/University of Texas at Austin
moconnor@nvidia.com

Mattan Erez
University of Texas at Austin
mattan.erez@utexas.edu

Jeff Pool
NVIDIA
jpool@nvidia.com

David Nellans
NVIDIA
dnellans@nvidia.com

Stephen W. Keckler
NVIDIA
skeckler@nvidia.com

***Abstract*—GPUs accelerate high-throughput applications, which require orders-of-magnitude higher memory bandwidth than traditional CPU-only systems. However, the capacity of such high-bandwidth memory tends to be relatively small. Buddy Compression is an architecture that makes novel use of compression to utilize a larger buddy-memory from the host or disaggregated memory, effectively increasing the memory capacity of the GPU. Buddy Compression splits each compressed 128B memory-entry between the high-bandwidth GPU memory and a slower-but-larger buddy memory such that compressible memory-entries are accessed completely from GPU memory, while incompressible entries source some of their data from off-GPU memory. With Buddy Compression, compressibility changes never result in expensive page movement or re-allocation. Buddy Compression achieves on average 1.9× effective GPU memory expansion for representative HPC applications and 1.5× for deep learning training, performing within 2% of an unrealistic system with no memory limit. This makes Buddy Compression attractive for performance-conscious developers that require additional GPU memory capacity.**

I. Introduction

GPUs are widely used for high-memory-footprint applications, including those for High Performance Computing (HPC) and Deep Learning (DL). HPC applications like planet-scale simulations and the modeling of fluid and molecular dynamics have grown to require very large models [1], [2], [3], [4], [5]. DL networks are also growing such their model sizes are either too big to run on GPUs or large enough that the only a small batch size can fit on the GPU, possibly resulting in low utilization and accuracy issues [6], [7], [8], [9], [10], [11].

Despite increasing memory requirements, compute-class GPUs will likely continue to prioritize memory speed over capacity to keep their many parallel cores busy. High Bandwidth Memory (HBM) must fit in limited package space and chip periphery such that even the current highest-capacity GPUs have only 4 HBM stack sites, resulting in a maximum capacity of 32GB. Meanwhile, non-HBM graphics DDR memory cannot be driven at high speeds with more than 1 rank per channel, and the number of channels is already nearing the practical pin count limit. Thus accelerators like GPUs will always have a relatively limited maximum capacity as compared to CPUs or other network-attached devices.

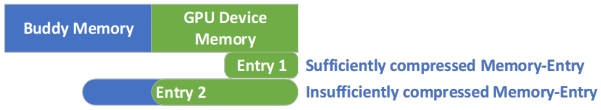


Fig. 1: If a memory-entry (128B) does not compress sufficiently, part of it is accessed from the buddy-memory.

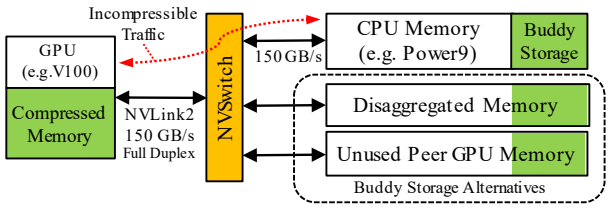


Fig. 2: A target system for Buddy Compression. Any larger NVLink-connected memory can be used as buddy storage. The overall organization of this system is similar to an NVIDIA DGX-2 node [26].

Currently, applications with large memory footprints must resort to unattractive options to compensate for limited GPU memory. They can scale out to many GPUs for capacity purposes alone (inefficiently utilizing resources) [12], [13], explicitly orchestrate data movement between the CPU and GPU to stay within device memory limitations (adding algorithmic complexity) [14], [7], [15], or rely on off-GPU memory accesses or Unified Memory [16] to automatically oversubscribe device memory (limiting performance) [17], [18]. This paper explores memory compression as a performant and general GPU memory-expansion alternative.

While main memory compression has been studied for CPUs [19], [20], [21], [22], [23], GPU architectures pose very different trade-offs. CPU compression techniques assume that compressed pages are of different sizes, and they re-allocate pages as compressibility changes [20], [21], [22], [23]. Such on-the-fly page re-allocations would be prohibitively expensive in GPUs due to their immense memory bandwidth [17]. Additionally, while domain-specific compression [24], [25] has been proposed for large-footprint GPU workloads, general-purpose compression for capacity remains unexplored.

Figure 1 shows that Buddy Compression divides compressed memory allocations between the GPU device memory

and a larger-but-slower *buddy-memory* connected with a high-bandwidth interconnect. If a cacheline-sized (128B) memory-entry is sufficiently compressed, it is sourced completely from device memory; if not, it is sourced from *both* device and buddy-memory. This design requires no re-allocations or page movement if the compressibility of the data changes over time. A high-bandwidth interconnect like NVLink [26], OpenCAPI [27], or CXL [28] enables this design, since it ensures low overhead accesses to the buddy-memory, so long as most of the data compresses to fit in GPU device memory. Figure 2 shows that any remote memory connected to the GPU with a high-bandwidth interconnect is suitable for use as a buddy-memory. This design maintains a good compression ratio and high performance while avoiding the complexity and performance concerns of using CPU memory compression approaches on GPUs. To summarize our contributions:

- We introduce the first design to use general-purpose compression to increase the memory capacity of GPUs. Buddy Compression is unique, since it does not require any additional data movement when the compressibility of the data changes.
- We provide an in-depth analysis of the memory values from GPU workloads with representative data and derive insights for effective GPU compression.
- Buddy Compression achieves on average 1.9× (HPC) or 1.5× (DL training) compression and performs within 2% of an unconstrained-memory-capacity GPU.
- Finally, we present a case study on DL training to understand the benefits and trade-offs of using Buddy Compression to expand the GPU memory capacity.

II. Overview and Background

A. Target Workloads and Related Work

HPC Workloads. Previous work in the HPC domain suggests that a larger GPU memory would benefit applications such as fluid dynamics and weather prediction [5], [29], [30], [31]. However, scientists are currently forced use either a shared virtual address space like Unified Memory [16], [32] or multiple GPUs [3], [4] to scale past the GPU capacity limit. Buddy Compression offers a general and performant alternative to these approaches. We use a subset of SpecACCEL OpenACC v1.2 [33] and CUDA versions of the DOE benchmarks HPGMG [34] and LULESH [35] as accessible and simulatable representatives for HPC applications. The subset is chosen based on our confidence in the representativeness of the data values used in the benchmarks. For each considered benchmark, we consulted a domain expert to affirm that the input data for the benchmark could be considered "reasonably representative" of real-world use.[1] We omit many other benchmarks that fail this test. SPEC benchmarks are also commonly used in prior compression studies [19], [21], [20], [36], [23].

DL Workloads. GPUs are currently the most popular choice for training deep neural networks. As networks grow deeper and wider, they require more data and inevitably hit the memory-capacity wall. We train a set of 5 convolutional neural networks to represent DL workloads: AlexNet [37], Inception v2 [38], SqueezeNetv1.1 [39], VGG16 [40], and ResNet50 [41], all running on the Caffe [42] framework with the ImageNet [43] dataset. Additionally we consider a long short-term memory network, BigLSTM [44], using the English language model.

Many domain-specific solutions have been proposed across the stack to address the DL memory capacity challenge, including DL-specific footprint reduction schemes [45], [46], [25] and asynchronous offloading of data during each training iteration [47], [14], [12], [48], [15], [49], [50]. Buddy Compression is an orthogonal and possibly-complementary approach to expand the effective GPU memory capacity. Our approach allows larger-footprint DL training runs with no algorithm-level changes. Buddy Compression can to elide all communication for compressible data, whereas DL offloading must transfer some data during every iteration. This reduction in bandwidth may improve performance, and it likely allows Buddy Compression to operate with significantly lower power overheads. In addition, Buddy Compression could potentially be used in tandem with these other approaches. For instance, Buddy Compression can be used in conjunction with vDNN [14] to allow a single layer to be larger than GPU memory while relying on vDNN to retire and prefetch other layers to and from host memory in the background.

Memory Compression. Memory compression has been used in various forms to expand the effective CPU memory capacity. Most modern operating systems compress the swap space to reduce paging to disk [51]. Numerous proposals have been made to accelerate CPU main memory compression in hardware [19], [20], [21], [22], [23]. However, explored in Section II-C, these approaches are inefficient and inapplicable for general-purpose GPU memory capacity compression due to architectural differences and frequent compressibility changes in GPU data. The graphics pipeline of most GPUs includes domain-specific lossy texture memory compression to reduce the footprint of graphics textures [24], [52]. To our knowledge, hardware compression is not currently used for general-purpose compute workloads on GPUs.

Buddy Compression Target System. Figure 2 shows a future GPU system that we envision for Buddy Compression. The system is composed of multiple GPU nodes connected with a high-bandwidth NVLink2 interconnect [53], [26] to a larger source of remote memory. In currently available systems, this remote memory could be the system memory of a Power9 CPU, an unused peer GPU memory, or an NVLink2-connected disaggregated memory appliance, a natural extension of the technology that is being explored for servers [31], [54], [49].

[1]Through personal communications, we received a positive affirmation for the data of LULESH and HPGMG from N. Sakharnykh (NVIDIA) on 30-May-2018 and for some SpecACCEL benchmarks from M. Colgrove (PGI/NVIDIA) on 29-May-2018.

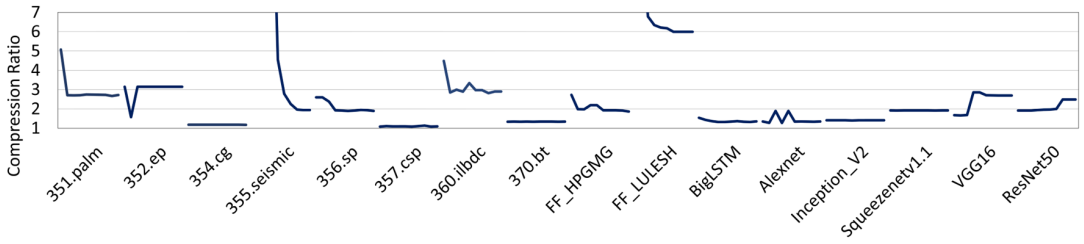


Fig. 3: The average compression ratio of the allocated memory for the complete run of each benchmark. Ten equally distributed memory-snapshots are taken during the entire run of each benchmark, and the compression ratio calculated.

B. Relevant Modern GPU Technologies

High-Bandwidth Interconnects. High-bandwidth interconnects are a key enabler for Buddy Compression, and can be applied on any GPU with a high-bandwidth interconnect to host memory. In recent years, high-bandwidth interconnects like NVLink [53], OpenCAPI [27], and NVLink2 [26] have been used to alleviate the communication bottleneck in multi-GPU systems. NVLink2 provides 25GBps of full-duplex unidirectional bandwidth per *channel*.[2] Modern compute-class V100 GPUs support six NVLink2 channels per GPU, offering a bidirectional bandwidth of up to 150GBps (full-duplex), much higher than a 16GBps $\times$16 PCIe3.0 or 32GBps $\times$16 PCIe4.0 connection. The NVIDIA DGX-2 [26] workstation has sixteen V100 GPUs connected through an NVLink2 switch that support six NVLink2 channels, allowing high-speed remote access to system memory [55]. Similar solutions are available for AMD and ARM GPUs using Heterogeneous System Architecture (HSA) and CCIX [57], [58], and more recently for Intel GPUs using Compute eXpress Link (CXL) [28].

Unified Memory (UM). These high-bandwidth interconnects also enable a shared virtual address space between the host processor and multiple accelerators. NVIDIA's Unified Memory (UM), introduced in CUDA 8 for Pascal-class GPUs [16], is a prominent example. Non-local UM requests either remotely access data through the GPU interconnect or result in transparent data migration with the placement of any piece of data being determined by a variety of heuristics [16], [32]. UM supports memory oversubscription, allowing UM-managed regions that are larger than the GPU device memory to be accessed without the programmer explicitly managing data movement. However, this capability is not widely used for high-performance applications, since large hot working sets experience frequent page faults and thrashing with Unified Memory, causing significant slowdowns [32], [59], [60].

C. Compressibility of GPU Workloads

To estimate the possible gains from compression, we first determine how compressible the high-footprint GPU workloads are by collecting memory dumps of the workloads running on a Tesla P100 GPU. We intercept each GPU *malloc* and *free* API call (including variants for pinned and UM-managed memory) to dynamically track the allocated regions in device memory. We divide the entire runtime of the workload into 10 regions, and collect a memory dump of the allocated device memory at the kernel boundary closest to each region.

Figure 3 shows the compression ratio of each benchmark using Bit-Plane Compression (BPC) [61] over its entire run. These compression ratios are optimistic, since they assume eight available compressed memory-entry sizes (0B, 8B, 16B, 32B, 64B, 80B, 96B, and 128B) and assume no page-packing overheads. Each memory-entry is individually compressed and allowed to occupy any of these sizes. On average, the geometric mean of compression ratios is 2.51$\times$ for the HPC benchmarks and 1.85$\times$ for DL training. This average compressibility is higher than prior reports for CPU workloads [23], which we attribute to the higher percentage of homogeneous data allocations (with a single uniform datatype). Prior work has established that BPC works well for homogeneous data, and such homogeneity is prevalent in GPU workloads [61].

Compressibility Changes. Changes in compressibility are more frequent for GPU benchmarks than they are in previously-studied CPU workloads [21], [23], [20]. As an example, 355.seismic begins with many zero values but slowly reaches an asymptote at a 2$\times$ compression ratio over its execution. Although the overall compression ratio of the DL workloads stays roughly constant, compressibility changes frequently for individual memory entries. This variability is because DL frameworks perform their own asynchronous GPU memory allocation with software-managed memory pools and may reuse the same memory location for a variety of purposes over program execution [42].

Comparison to CPU Trends. If data compressibility decreases, prior CPU memory approaches may be forced to allocate more space for the same memory-entry, causing a *memory-entry overflow* and suffering additional data movement [23]. We observe that memory-entry overflows would happen more often and be more painful in GPUs than CPUs. While CPU workloads like SPECcpu2006, Pagerank, and Graph500 experience an overflow rate of 0.4% per instruction on average [23], the overflow rate in GPUs is 8% per warp-instruction on average across our workloads. This difference stems from a higher write throughput in GPU memories, as well as differing workload characteristics (e.g. DL training). Furthermore, previous work has shown that GPU page allocation incurs a hefty GPU-driver latency and is a sequential bottleneck in an otherwise highly-parallel system [17]. Since all previous work in CPU memory compression relies on page re-allocation and data movement, it is not a good fit for GPUs. Buddy Compression addresses

[2]We use the term channel to denote a single NVLink2 $\times$8 connection. Other papers call this NVLink2 connection a brick [55] or link-slot [56].

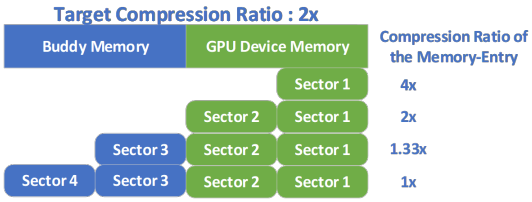


Fig. 4: A 128B memory-entry compresses to occupy 1–4 32B sectors. If an entry does not compress to the target compression ratio, left-over sectors are accessed from buddy-memory.

this challenge by avoiding any page-level movement due to memory-entry overflows.

III. Buddy Compression

Buddy Compression allows a programmer or DL framework to annotate memory allocations to use less device memory than the allocation size. For instance, if a user has 24GB of data and a GPU with only 12GB of memory capacity, the data can be allocated with a target of 2× compression so that only half of the full data size is allocated on the GPU device memory. We use fine-grained compression to opportunistically fit data into this reduced device-resident allocation. If a memory-entry does not compress sufficiently, an NVLink2-connected larger-but-slower *buddy-memory* is used as overflow storage. Data from compressible memory-entries are sourced completely from GPU device memory, while incompressible memory entries are sourced from both device and buddy-memory.

Figure 4 shows Buddy Compression striping the data using 32B sectors which matches the access granularity of GPU memory on GDDR5, GDDR5X, GDDR6, and HBM2-based GPUs. For example, if an allocation targets a compression ratio of 2×, the first two sectors per 128B memory-entry are mapped to device memory, and the last two are mapped to buddy-memory. Therefore, if a memory-entry is compressed by 2× or more, it fits completely in device memory. Otherwise, the latter 64B of the entry are stored in a pre-allocated buddy-memory location.

Compression Algorithms. A hardware memory compression algorithm should be fast and require little energy, yet result in high compression rates. Figure 5a compares several low-cost compression algorithms [62], [63], [65], [61], [64]. Bit-Plane Compression (BPC) [61] is the most attractive for Buddy Compression as it achieves robust compression ratios across both HPC and DL workloads. As Buddy Compression is algorithm-agnostic, it will work with any low-cost hardware compression algorithm.

Compression Granularity and Sizes. Most CPU main memory compression strategies operates at the cache-block granularity to avoid read-modify-write (RMW) overheads. Buddy Compression shares this design decision and uses a 128B compression granularity to match the GPU cache block size [66]. Figure 5b shows a histogram of compressed memory-entry sizes across our workloads. Buddy Compression supports the most frequently occurring sizes of 0B, 32B, 64B, and 128B. We also support 96B compression as it aligns nicely with the 32B access granularity of GPU memory.

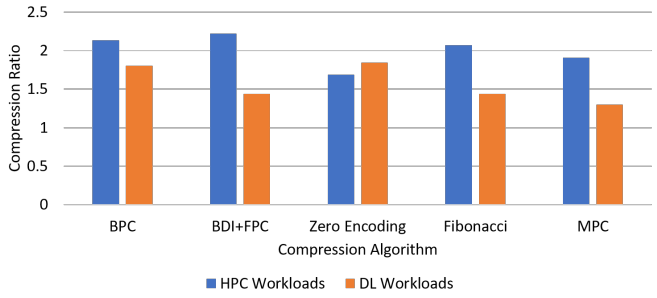
Buddy-Memory Carve-Out Region. At boot time, the driver carves out a contiguous chunk of pinned buddy-memory for each GPU. These regions are never directly accessed by the host CPU, eliminating any coherence issues, and making the address translation for buddy-memory simple and fast. The buddy-memory size corresponds to the maximum target compression ratio for the GPU. As an example, if the maximum target compression ratio is 4×, then the carve-out region should be 3× as large as GPU device memory, to allow each memory-entry to have 3 sectors in buddy-memory (in the worst case) and only 1 on the GPU.

While the size of the carve-out is considerable, most systems have a significantly larger host system memory than the aggregated GPU memory capacity. For example, DGX-2 nodes have an aggregated 512GB GPU versus 1.5TB CPU memory (3×) [26], Sierra nodes have 64GB GPU versus 256GB CPU memory (4×) [67], Summit nodes have 96GB GPU versus 512GB CPU memory (5.3×) [68], and ACBI nodes have 64GB GPU versus 384GB CPU memory (6×) [69]. Prior work [70] reports that host memory remains underutilized in such systems. Disaggregated memory pools [54], [31], [49] can be sized to an even larger capacity.

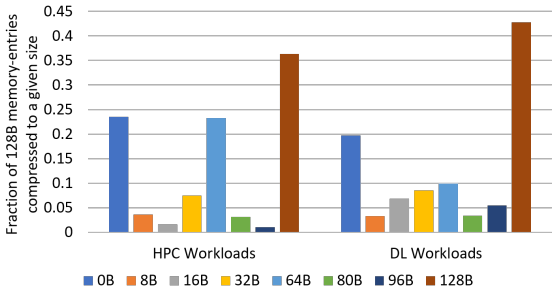
Address Translation. Accessing compressed data requires additional address translation and metadata which informs (i) the target compression ratio, (ii) whether a particular memory-entry was compressed to the target ratio, and (iii) the address in buddy-memory to be accessed for memory-entries that did not compress to the target ratio. A global base physical address for the buddy-memory carve-out region is stored in a Global Buddy Base-address Register (GBBR). The page-table and TLBs are augmented to store the information about whether the page is compressed or not (1 bit), the target compression ratio (3 bits), and the offset of the buddy-page from the global base address (16/20 bits for 2MB/64KB GPU pages, a 4× maximum target compression ratio, and a 32GB maximum GPU capacity). GPUs differ from CPUs in that they use a larger page table entry (PTE) which already contains potentially-unused metadata bits. Open documentation [71] shows NVIDIA Pascal GPUs to have 64b PTEs with 27 bits that are unused or dedicated to graphics-specific attributes and texture compression. These 27 bits can be repurposed for general-purpose workloads to store the 16–20 bits of Buddy Compression address translation metadata.

Buddy Compression works for GPU-pinned (cudaMalloc'ed) memory. CPU-shared (UM [16] or ATS [72], [73]) memory support requires additional address translation mechanisms and system-level simulation and is left for future work. Buddy Compression stores address translation metadata in the GPU page tables, and non-GPU peers do not have access to this translation metadata so they cannot make requests to Buddy-Compressed GPU memory. Other aspects of shared memory support that are ripe for for future work, such as compressibility-aware migration heuristics and the ability to dynamically change compression targets during migration.

Memory-Entry Size Metadata. The compressed size of each 128B memory-entry uses 4 bits of metadata. We store this metadata in a dedicated driver-allocated region of device

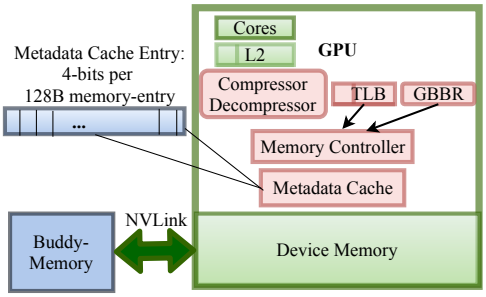


(a) Compression Algorithms

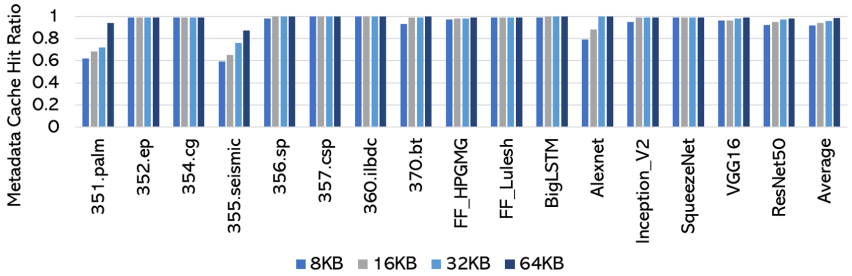


(b) Compressed Sizes (BPC, 128B Granularity)

Fig. 5: The average compression ratio of different algorithms, and a histogram of compressed sizes across all workloads (BPC, 128B compression). Algorithms: Bit-Plane Compression (BPC) [61], Base-Delta-Immediate and Frequent Pattern Compression (BDI+FPC) [62], [63], Zero Value Compression [15], Fibonacci Compression [64], and Massively Parallel Compression (MPC) [65].



(a) Architectural overview



(b) Metadata cache hit rates with different sizes of total metadata cache

Fig. 6: Compression metadata handling architecture; GBBR is Global Buddy Base-address Register.

memory, amounting to 0.4% storage overhead. While our chosen compression sizes strictly require only 5 states (3 bits) per compressed cache line, we reserve 4 bits of metadata to align to a power-of-two and to provision reserved states for other uses or future compression changes. The metadata storage overheads of Buddy Compression are comparable to or less than those of CPU compression schemes [23], [21], [19], [20], [22]. Figure 6a shows a high-level view of the metadata setup and translation. The simple GBBR-offset based addressing makes the overall translation mechanism straightforward to implement.

A cache is used to avoid metadata traffic for workloads with good locality. Figure 6b shows the metadata cache hit ratios as a function of the metadata cache size. Most applications have high hit ratios. We use a 4-way 64KB metadata cache that is split into 8 slices, 1 per memory controller. Each 128B metadata cache entry is split into four 32B sectors, thereby causing a prefetch of metadata corresponding to 63 neighboring memory-entries on every metadata sector miss. The metadata is interleaved across the HBM2 channels using the same hashing mechanism as regular physical-address interleaving.

A. Benefits of the Buddy Compression Design

No Page-Faulting Expense. The immense parallelism of a GPU increases its overall throughput. However, driver-based page-fault handling is remote and non-distributed, making GPU page-faults expensive [17]. The compressibility of data in memory can decrease, requiring new page allocations for prior CPU-based memory compression schemes. The page fault overhead in GPUs renders these prior CPU-based compression schemes untenable. Buddy Compression is unique in that the compressibility of each memory-entry affects only its own allocation, and compressibility changes never result in page movement or re-allocation. If the compressed size of a memory-entry exceeds the device allocation, its upper sectors spill over to the buddy-memory; if the compressed size then decreases, stale-but-never-accessed values are left in buddy-memory until the allocation is freed.

Low Translation Overhead. Memory bandwidth is an occasional bottleneck for GPUs. Accordingly, there has been fruitful research on bandwidth compression of GPU main memory [61], [74]. Buddy Compression uses compression to amplify both the bandwidth and capacity of GPU memory. However, as discussed earlier, compression-for-capacity requires additional metadata accesses for translation into the compressed address space, emphasizing the importance of reducing the metadata size and keeping translation simple. Buddy Compression requires only 0.4% metadata, and because the buddy-memory carve-out region is physically contiguous, addressing into it is offset-based and straightforward.

B. Reducing Buddy Compression Overheads

With the design of Buddy Compression, the major overhead comes from accessing the slower buddy-memory in cases of unexpectedly low compression.

Profiling Target Compression Ratios. Choosing the right target compression ratio is important, since aggressive compression ratios will lead to more memory-entries exceeding

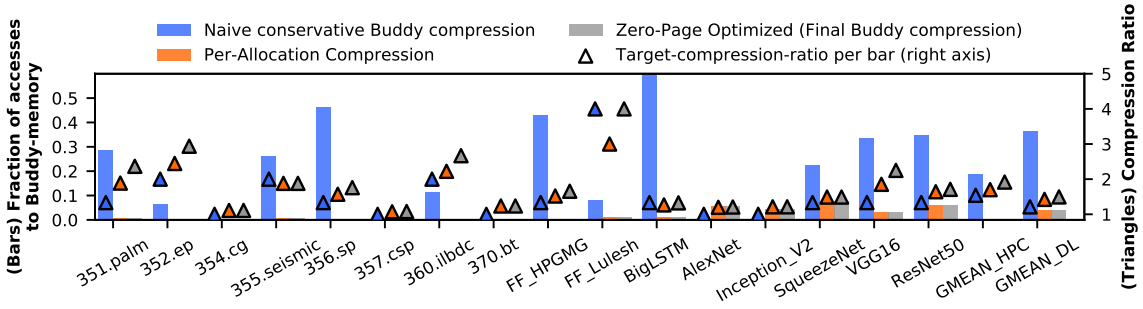


Fig. 7: Sensitivity of the compression ratio and buddy-memory accesses to design optimizations. Optimizations are applied successively, such that the zero-page optimized design also contains per-allocation compression.

the allocated device memory and requiring buddy-memory accesses. To choose the target compression ratio, we use a simple profiling pass on a representative dataset. For HPC workloads, the profiling pass is run using a smaller dataset, like the *train* dataset for SpecACCEL. For DL workloads, a brief profiling pass is run with a smaller batch size, which can be embedded in training platforms like PyTorch or TensorFlow.

Understanding Compressibility Patterns. The granularity at which the programmer annotates memory is also important—the best annotation granularity depends on the spatial compressibility patterns. Naive Buddy Compression considers a single, conservative target compression ratio for the whole-program. Figure 7 shows that this granularity is too coarse. The naive mechanism achieves an overall compression ratio of just 1.57× for HPC workloads and 1.18× for DL workloads, requiring 8% accesses over the interconnect to the buddy-memory for HPC, and 32% for DL.

To investigate the appropriate Buddy Compression annotation granularity, Figure 8 shows a spatial plot (in the virtual address space) of each workload's compressibility. Each sub-plot is a spatial heat map that shows the compressibility of the memory allocated by a benchmark. A colder color (blue), signifies high compressibility and a hotter color (red) denotes low compressibility. The plot is structured with each row having 64 128B memory-entries. Figure 8 shows that the spatial locality of compressibility varies significantly across benchmarks. While most HPC benchmarks have large homogeneous regions of similar compressibility, the distribution is more random in DL workloads. FF_HPGMG shows specific patterns of compressibility that can be correlated to its use of arrays of heterogeneous structs for data structures. Although the DL workloads do not show the level of homogeneity that can be seen in HPC workloads, the graph still contains some mostly-red or mostly-blue regions. Based on the insights from these plots, we propose further optimizations to the design of Buddy Compression below.

Per-Allocation Compression Targets. Figure 8 shows that most benchmarks have large homogeneous regions of similar compressibility. We find that most of these region boundaries overlap with GPU malloc() boundaries. A special allocation API for compressed regions allows us to capture this behavior and eliminate the futile effort of compressing the red regions.

During profiling, we periodically take snapshots of memory, to track the compression ratios per allocation. At the end of profiling, we decide target compression ratios per allocation using heuristics to trade-off the compression ratio with the buddy-memory accesses. The compression ratio is chosen conservatively to aggressively reduce buddy-memory accesses. For example, 355.seismic from Figure 3 exhibits high average compressibility, but our per-allocation heuristics target most allocations to 2× compression to compensate for the lesser end-of-application compressibility.

We evaluate Buddy Compression using a static target compression ratio per allocation. A dynamic target compression ratio would require reallocating and moving around memory, making the compression management more complicated and less performant. DL frameworks reuse memory allocations for different purposes, and the compressibility of a single allocation can change over time. To investigate the effect of changing DL memory allocations despite using a single target compression ratio, Figure 9 shows the buddy-memory access rate of ResNet50 and SqueezeNet over a training interval. While both programs exhibit frequent compressibility changes per memory-entry, the buddy-memory access rates do not fluctuate. Even though the individual memory-entries frequently change in compressibility, the changes are relatively unbiased and the net effect is negligible.

The Buddy Threshold Meta-Heuristic. Most benchmarks have allocations that are highly homogeneous in their compressibility, making the per-allocation target ratio decision straightforward. However, for benchmarks like AlexNet and ResNet50, memory allocations have mixed compressibility, meaning we must trade-off between the compression ratio and the frequency of buddy-memory accesses. To guide our per-allocation heuristics, we define a Buddy Threshold meta-heuristic that limits the expected fraction of buddy-memory accesses. A higher Buddy Threshold achieves a higher compression ratio at the cost of more buddy-memory accesses and lower performance.

Figure 10 shows a sensitivity sweep of the Buddy Threshold (10% to 40%), alongside the best achievable compression ratio assuming no Buddy Threshold constraints. Buddy-memory accesses remain infrequent for HPC benchmarks regardless of the Buddy Threshold selection, due to the largely-homogeneous allocations in these benchmarks. DL workloads suffer from more frequent buddy-memory accesses and are sensitive to the Buddy Threshold. With the exception of FF_HPGMG, we are able to achieve near-optimal compression with a 40%

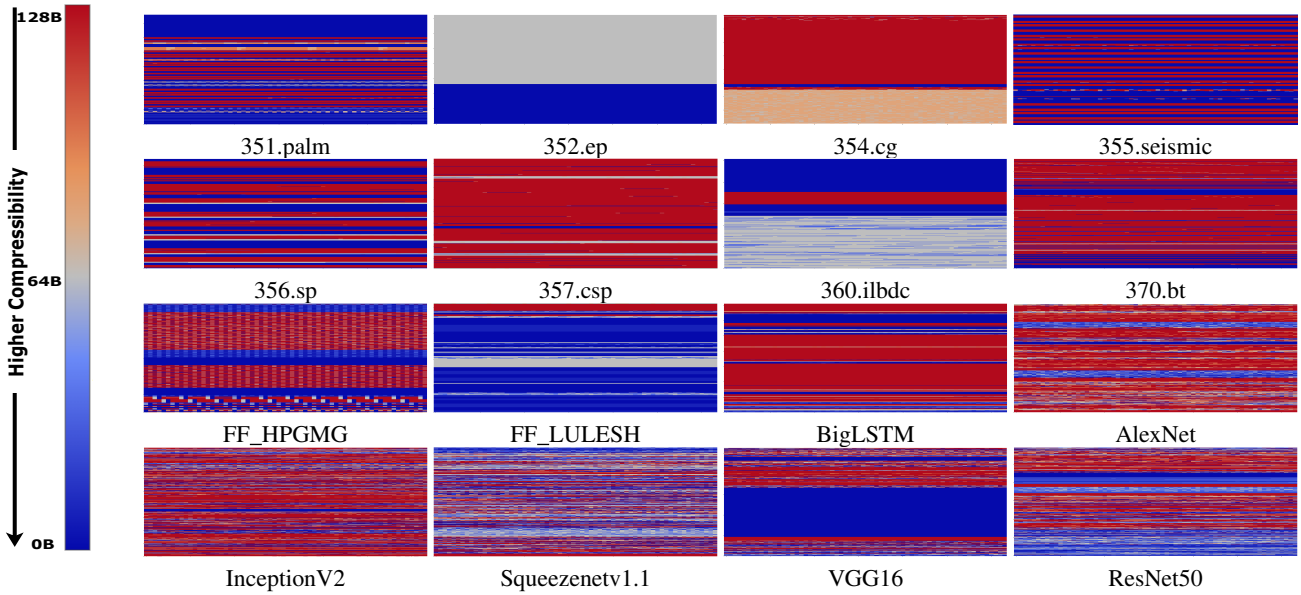


Fig. 8: Spatial compressibility patterns. Each heatmap shows the compressibility of all allocated GPU memory. Each row represents 64 128B memory-entries, ordered by their virtual addresses.

Buddy Threshold, as can be seen in comparison with the red marker. As discussed earlier, FF_HPGMG has a peculiar striped compressibility pattern resulting from its use of arrays of structs. To capture the maximum compression, FF_HPGMG requires more than 80% Buddy Threshold. Overall, since a 30% Buddy Threshold achieves a good balance between the compression and buddy-memory accesses, we choose this parameter for our Buddy Compression evaluation.

Special Case For Mostly-Zero Allocations. The spatial plots show memory regions that remain mostly-zero across the entire benchmark. To capture the capacity-expanding opportunity of such allocations, we add an aggressive target compression ratio of 16× where we keep only 8B out of each 128B in device memory. The only change is an additional encoding for page size in the TLB.

This optimization increases the compression ratio for benchmarks such as 352.ep and VGG16 with mostly-zero regions. This optimization does not have much impact on the buddy-memory accesses rate, since such highly-compressible data almost always fits in device memory. Figure 7 shows the effect of this optimization. For HPC benchmarks, the compression ratio goes up from 1.7× to 1.9×, while for DL, from 1.4× to 1.5×. To enable this optimization, the profiler marks regions that are mostly zero and remain so for the entire benchmark run. Because of the limited size of the buddy-memory carve-out region, however, the profiler is constrained to keep the overall compression ratio less than 4×.

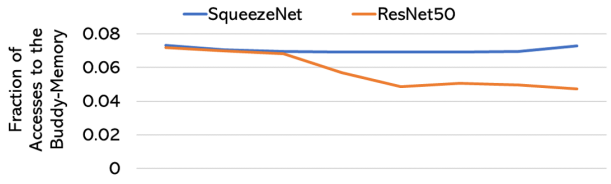


Fig. 9: The fraction of buddy-memory accesses over a complete DL training interval does not vary significantly.

C. Evaluated Design

For our evaluation, Buddy Compression uses a 30% Buddy Threshold for its profiling heuristics, a 4KB metadata cache per memory controller, and a buddy-memory carve-out that is 3× the size the GPU device memory (supporting a 4× maximum expansion of GPU memory, or more with mostly-zero allocations). We profile the application with a smaller dataset, and the profiler reports target compression ratios that are used by the DL framework or HPC user to annotate GPU malloc API calls. Figure 7 shows the compression ratio and buddy-memory accesses for the final design. We achieve 1.9× memory compression for HPC and 1.5× compression for DL workloads. The average proportion of buddy-memory accesses are 0.08% for HPC data traces and 4% for DL workloads.

IV. Performance Evaluation

Having demonstrated that Buddy Compression provides good compression with infrequent buddy-memory accesses (Figure 7, Zero-Page Optimized design), we next discuss the performance benefits of Buddy Compression from GPU memory capacity expansion. We show that Buddy Compression approaches the performance of an unrealistic unconstrained-capacity GPU, far outperforming UM-based oversubscription. We then present a case-study of DL training to estimate the end-to-end performance benefits from an increased GPU memory capacity.

A. Evaluation Methodology

Workloads. As previously described in Section II-A, we evaluate Buddy Compression's effectiveness on a set of 10 HPC and 6 DL network training workloads. We collect a representative trace from each benchmark running its reference dataset, following the compression-aware subsetting

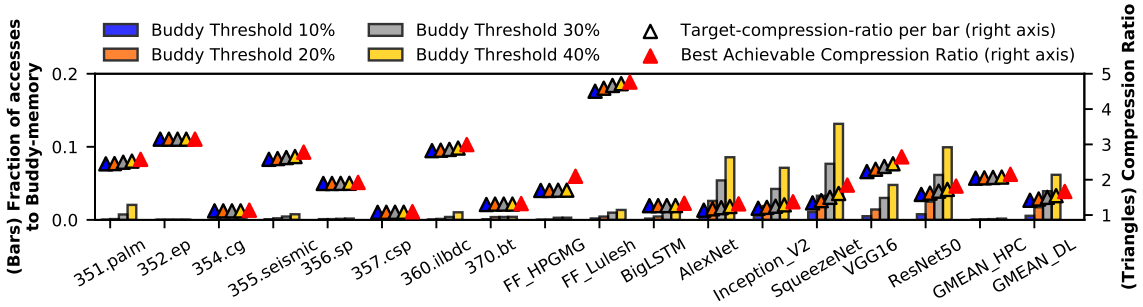


Fig. 10: Sensitivity of the compression ratio and buddy-memory accesses to the Buddy Threshold parameter.

methodology of prior work [36]. Each trace contains 1–9 billion warp instructions and corresponds to the dominant kernel of each benchmark at an execution point that exhibits the average compression ratio for that entire benchmark execution. The trace for each DL workload spans one full training iteration.

Simulation Infrastructure. We use a dependency-driven GPU performance simulator, similar to the one used by Arunkumar et al. and others [75], [76], [77]. Tab. I shows our simulator configurations which are based on public information about NVIDIA's P100 Pascal GPU [78] and the interconnect characteristics of recent Volta GPUs [79]. Non-public microarchitectural details are configured using microbenchmark results from prior work [66], [56]. Each SM is modeled as an in-order processor with greedy-then-oldest warp scheduling. We model a multi-level cache hierarchy with private L1 caches and a shared sectored L2 cache with 128B lines and 32B sectors. Caches are banked to provide the necessary parallelism to saturate DRAM bandwidth. We model software-based cache coherence in the private caches, similar to state-of-the-art GPUs. Device memory consists of 32 HBM2 channels split across 8 memory controllers (MCs), and each GPU is connected to buddy-memory with 6 NVLink2 channels (unless otherwise noted).

We conservatively model decompression latency as 11 DRAM cycles, as discussed in prior work [61]. The metadata cache is 4KB and 4-way set associative per MC. To determine its performance impact, we also evaluate bandwidth-only compression between the L2 cache and device memory. Such compression does not increase the effective memory capacity, but it can alter the DRAM bandwidth and latency, affecting performance. Our simulator also models the latency overheads of accessing the buddy memory; this latency is set to be twice that of a local GPU HBM2 access, based on prior characterization work [56], [80].

Figure 11 (left) shows agreement (correlation coefficient 0.989) between the simulated and actual cycles spent on a V100 GPU across a wide variety of benchmarks.[3] Corresponding numbers from GPGPUSim (correlation coefficient 0.948), a widely-used academic simulator, are also shown. Our motivation in using a proprietary simulator comes from the two orders-of-magnitude speed benefit shown in Figure 11 (right), which allows us to simulate larger and more realistic workloads.

[3]To compare to GPGPUSim results, we show simulator correlation results with a slightly different configuration than is used for evaluation. The P100-based configuration used in the paper also correlates well with silicon.

Tab. I: Performance simulation parameters.

Core	1.3 GHz; 2 greedy-then-oldest warp schedulers per SM Max 64 32-thread warps per SM
Caches	24KB private L1/texture cache per SM, 128B lines 64KB dedicated scratchpad per SM, 4MB shared L2, 32 slices, 128B lines, 16 ways
Off-Chip	8 MCs, each with 4 875MHz HBM2 channels (900 GBps) 6 NVLink2 channels (150 GBps full-duplex*)
Buddy	64KB metadata cache (4KB slices), 128B lines, 4 ways Compression/Decompression latency = +11 cycles

* Interconnect bandwidth is swept in later parts of the evaluation.

B. Performance Relative to an Unconstrained-Capacity GPU

We compare the performance of Buddy Compression to an unrealistic GPU with no memory limits. This unconstrained-capacity GPU is idealized and it either represents a system with a higher memory density than is currently possible, or a system with multiple ranks per channel (disregarding any impact on signal integrity and memory interface speed). Such a large memory is not generally feasible in today's GPUs.

Apart from increasing the effective memory capacity, Buddy Compression has various performance side effects. GPU DRAM bandwidth compression, alone, can help performance by amplifying the memory bandwidth for programs with good DRAM locality, or it can hinder performance through (de-)compression latency and overfetch for fine-grained random memory accesses. Buddy compression potentially leads to further performance penalties due to the relatively low bandwidth and long latency of buddy-memory accesses, extra DRAM traffic and latency due to metadata accesses, and interconnect contention.

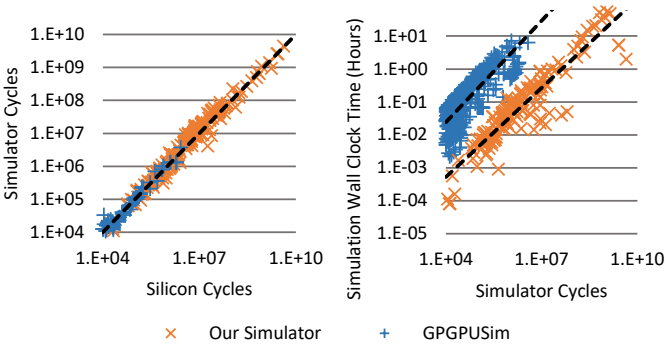


Fig. 11: Our simulator correlates with a V100 GPU (left, with slope=1 line drawn). It is two orders of magnitude faster than GPGPUSim [81], enabling the simulation of longer programs (right, linear regression lines drawn).

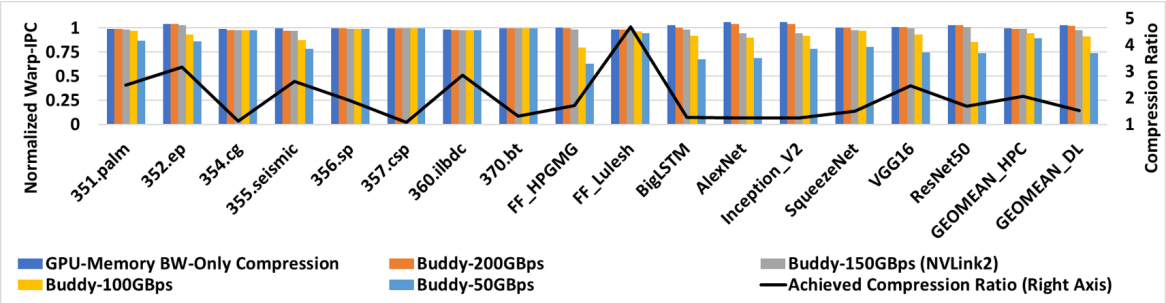


Fig. 12: The performance effects of compression, not accounting for capacity benefits. Interconnect bandwidths are swept and all results are normalized to an uncompressed GPU with a 150GBps interconnect and no memory limits.

To separate these effects, Figure 12 evaluates the performance of Buddy Compression alongside bandwidth-only compression, sweeping the buddy-memory interconnect bandwidth from 50 to 200GBps full-duplex, where 150GBps represents current NVLink2 speeds. We analyze the various performance contributors below. The application-specific performance benefits of a larger memory capacity are not present in these experiments; we consider this effect for DL training later in Section IV-D.

Bandwidth-Only Compression. While bandwidth-only compression does not increase the effective memory capacity, it does achieve an overall speedup of 5.5%. Most of this speedup comes from the DL training workloads, which are memory intensive with regular memory accesses. On the other hand, the HPC applications 354.cg and 360.ilbdc experience slight slowdowns with bandwidth compression because of their random and irregular access patterns. Most memory requests in these programs require only one sector, but bandwidth compression fetches the entire compressed cache-line, squandering bandwidth. FF_Lulesh experiences a slowdown despite having a regular memory access pattern due largely to the compression and decompression latency.

Buddy Compression Performance. Buddy Compression offers 1.5–1.9× higher effective memory capacity while possibly introducing additional overheads in the form of metadata cache misses and buddy-memory accesses. Figure 12 shows that while an interconnect bandwidth of 200GBps still achieves a 2% average speedup using Buddy Compression, all lower interconnect bandwidths experience some slowdown relative to the unconstrained-capacity GPU.

Metadata Accesses. Figure 6b shows that the benchmarks 351.palm and 355.seismic experience slowdowns due to a higher metadata cache miss rate. Since the other benchmarks have high metadata cache hit rates, metadata accesses do not have a discernible impact on their performance.

Bandwidth Sensitivity. Most HPC benchmarks have rare buddy-memory accesses (Figure 7), leading to negligible performance losses with a high-bandwidth interconnect. When the interconnect bandwidth is reduced, however, even 1% accesses buddy-memory accesses can cause considerable slowdown for bandwidth-sensitive applications like 352.ep and 355.seismic. Because FF_HPGMG spends significant time performing synchronous CPU-GPU memory copies, the link bandwidth dramatically affects its performance for reasons other than Buddy Compression. All results are normalized to a baseline system with a 150 GBps interconnect.

DL training uses a relatively higher rate of buddy-memory accesses, as can be seen in Figure 7. These buddy-memory accesses are caused by a lack of compression locality in the workloads. For example, AlexNet accesses buddy-memory at 5.4% of locations, leading to a 6.5% slowdown relative to the unconstrained-capacity GPU when combined with a 150GBps full-duplex interconnect. Performance degenerates with lower interconnect bandwidths, with the 50GBps full-duplex connection seeing a 35% slowdown.

These results show that recently-developed high-speed GPU interconnects are an enabling technology for Buddy Compression. The slowest link we evaluate (50 GBps full-duplex) is still faster than the most recent PCIe generation (×16 PCIe4.0, providing 32GBps full-duplex bandwidth). Yet it suffers from more than 20% average slowdown relative to the unconstrained-capacity GPU. However, high-bandwidth interconnects such as NVLink2 (150GBps full-duplex) allow Buddy Compression to come within 1% of the performance of the unconstrained-capacity GPU for HPC benchmarks, and within 2.2% for DL training.

Insignificant Factors. Due to the low buddy-memory access frequency (Fig. 10) and varying program memory intensities, the Buddy-150GBps interconnect bandwidth utilization is only 5.6% on average across all benchmarks (average 4% for HPC and 9.8% for DL). While we simulate interference between buddy-memory and regular interconnect traffic, interconnect contention does not seem to be a major factor in program performance. We simulate CPU-GPU traffic in our single-GPU workloads, while the high-speed interconnect is potentially also used for GPU-GPU communication. We expect that multi-GPU programs with GPU-GPU communication may see some slowdown due to interconnect contention, but will only affect program portions with overlapped communication and computation. At worst the performance loss should be proportional to the bandwidth utilization of the buddy-memory traffic. We also simulate the additional latency to buddy-memory, which ultimately has little effect on performance. This result is apparent from the 200GBps interconnect results, which offer a net speedup despite the increased buddy-memory access latency.

Energy. Apart from its system-level performance effects (with a proportional effect on leakage energy), Buddy Com-

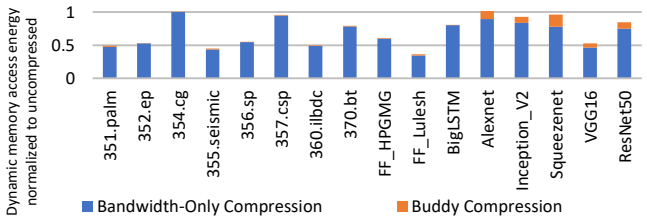


Fig. 13: Dynamic memory access energy benefits of compression.

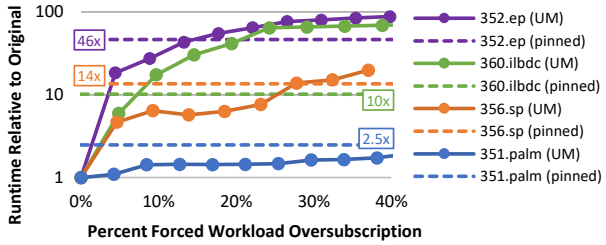


Fig. 14: Measured overheads of using UM oversubscription on a Power9 CPU connected via 3 NVLink2 channels (75 GBps full-duplex) to an NVIDIA V100 GPU. Dotted lines show the performance with all allocations pinned in host memory.

pression decreases the energy required to access DRAM. Figure 13 estimates the dynamic access energy of bandwidth-only compression and Buddy Compression relative to the uncompressed GPU with no memory limit. These results model the energy to read a memory snapshot of each program, assuming average row-buffer locality and the 64KB metadata cache hit rates from Figure 6b. We assume that HBM2 consumes 909 pJ per row activation, 1.51 pJ/bit on column access, and 1.97 pJ/bit on data sensing and I/O [82], while a high-bandwidth interconnect consumes roughly 11 pJ/bit in total [83]. We conservatively assume a 100% toggle rate for compressed data to account for increased entropy, and double the interconnect energy to represent ingress and egress from a switch. We also account for the worst-case 32B NVLink2 control header per buddy-memory access packet [84], and model remote buddy-memory and metadata accesses as HBM2 transactions. Despite these generally-conservative assumptions, Buddy Compression uses at worst roughly as much memory energy as the uncompressed baseline, and generally saves some energy through less HBM2 data movement.

C. Comparison with Unified Memory on NVIDIA GPUs

Faithfully comparing Buddy Compression and Unified Memory in simulation is not feasible due to the complex host-driver interactions and page migration policies within UM. Instead we choose to understand UM performance in oversubscribed scenarios by running SpecACCEL on real hardware. Figure 14 shows the measured performance of four SpecACCEL applications[4] with varying levels of oversubscription to illustrate the limitations of UM. The programs are run on an IBM Power9 system, connected to a Tesla V100 GPU via NVLink2 (interconnect topology with 3 channels, 75 GBps full-duplex CPU-GPU bandwidth). We compile the SpecACCEL applications with the *managed* PGI compiler flag, and force varying levels of oversubscription through an interposer that hogs GPU memory at application startup. We also run the applications using a compiler flag to pin all allocations in host memory, showing the slowdown in dotted lines.

Our results indicate that slowdown varies widely from negligible (354.cg and 370.bt, not shown in Figure 14 for clarity) to 87× slowdown (352.ep). Thrashing hampers UM performance for the most severely-slowed applications relative to running in pinned host memory, perhaps because UM is primarily intended for ease of programming and has not yet been tuned for high-performance memory oversubscription. Prior work [32], [17] supports our observation that UM oversubscription slowdowns can be excessive without more extensive hardware support. Figure 12 shows that Buddy Compression suffers from at most 1.67× slowdown for these programs when oversubscribing by >50%, even with a conservative 50 GBps NVLink speed. This result indicates that Buddy Compression is a better approach for managing high-performance memory oversubscription than software-based UM.

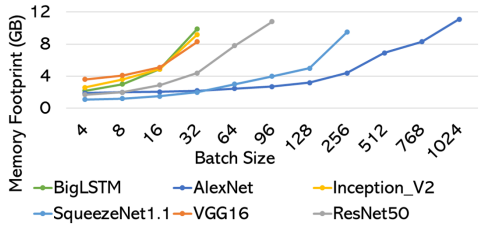
[4]We omit 354.cg and 370.bt for visibility because they show negligible slowdowns, even when pinned in system memory. We omit 357.csp due to failed compilation on the Power9 system, and 355.seismic because it hangs when run with the memory-hogging interposer.

D. Case Study: DL Training with Increased Memory Capacity

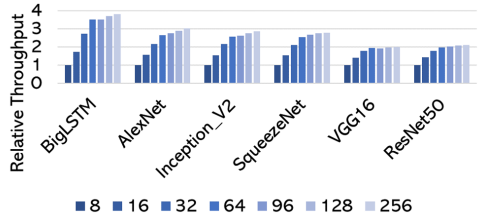
Comparing the performance of Buddy Compression to an uncompressed baseline with no memory limits (as in Figure 12) ignores the benefits of memory capacity amplification. For HPC benchmarks, more capacity allows a larger problem to be solved. Such benefits are important yet difficult to quantify. Accordingly, we instead study DL training workloads to quantify the performance benefits from compression. Buddy Compression increases the maximum supported mini-batch size, training large networks with fewer GPUs and higher efficiency per GPU.

Memory Footprints of DL Workloads. The memory footprint of a network during training depends on the mini-batch size. Larger mini-batch sizes place a larger part of the dataset in device memory, along with more intermediate data (activations and gradients). Figure 15a shows memory footprint measurements for each DL training workload as the mini-batch size is increased. The mini-batch sizes are increased up to the maximum that can fit on a Titan V GPU (12GB device memory). Initially there is not much difference as the batch size is doubled. Eventually, the memory footprint grows almost linearly with increasing mini-batch size. This transition point depends on the size of the network parameters, which do not vary with mini-batch size. For example in AlexNet, the network parameters consume a large portion of the overall memory due to the three large fully-connected layers and relatively few (five) convolutional layers. This attribute leads to a later transition point for AlexNet at a batch-size of 96; all other networks transition to an increasing memory footprint at a batch size of 32 or below.

Performance Impact of Larger Mini-Batches. A larger batch size is often beneficial as it allows more work to be performed per iteration, better utilizing resources [44], [6]. Figure 15b projects the speedup for each network with increasing mini-batch sizes. The graph is generated using a detailed analytical model very similar to [85], [86], to project performance with batch sizes that do not fit on current GPUs. Increasing the

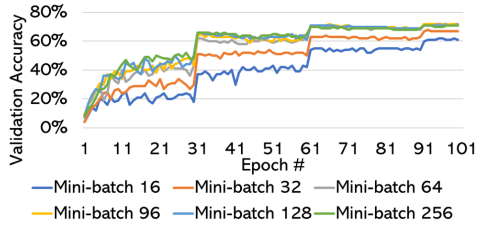


(a) Memory footprint as a function of batch size (PyTorch, Titan Xp).



(b) Projected speedup as a function of mini-batch size.

(c) Projected speedup using larger Buddy Compression batch sizes.



(d) Accuracy across mini-batch sizes (ResNet50, CIFAR100).

Fig. 15: Increasing DL training mini-batch size.

mini-batch size leads to higher relative training speed until the point of full GPU utilization, after which the effect plateaus.

Buddy Compression allows a larger mini-batch to fit in GPU memory. Figure 15c shows the relative speedup projected by our analytical model for this larger mini-batch size on a Titan V GPU (12GB device memory). The average speedup is 14%, while BigLSTM and VGG16 achieve higher speedups of 28% and 30%, respectively. The higher speedup in these workloads follows from Figures 15a and 15b. Without compression, both networks are unable to fit the mini-batch size of 64 that is needed for good resource utilization. The average speedup of 14% from larger mini-batch sizes is much higher than the 2.2% performance used to enable Buddy Compression (Figure 12). Thus Buddy Compression can significantly improve performance for capacity-constrained GPUs by enabling better GPU utilization through larger mini-batch sizes.

Better Convergence with Larger Mini-Batches. Apart from improving computational throughput with better resource utilization, the mini-batch size can also affect DL training accuracy. To investigate, we trained ResNet50 on the CIFAR100 [87] dataset for 100 epochs on a Titan Xp GPU with different mini-batch sizes. Figure 15d shows the validation accuracy results for these runs. Small mini-batches of 16 and 32 do not reach the maximum accuracy, despite using individually-tuned hyperparameters. Additionally, the mini-batch size of 64 trains to the maximum accuracy but converges more slowly than larger mini-batches. With batch normalization, the jitter in accuracy is also higher with small mini-batch sizes. While we observe good validation accuracy up to a batch size of 256, which is consistent with previous results [6], some prior work reports that increasing the mini-batch beyond a certain size can be detrimental to the network's generalization. However, other work indicates that tuning loss functions and hyperparameters can enable successful training with large mini-batches [88], [89].

Huge DL Networks. Recent object detection networks like MegDet [6] and natural language processing networks like GPT-2 [11] and BERT [8] exceed GPU memory capacities with even 2–4 input samples per GPU. Such capacity limitations are hurdles for developers, since batch normalization requires a batch size above 32 to be effective [90]. As a result, developers resort to horizontal scaling by spreading a mini-batch across many GPUs. For example, MegDet [6], [91] performs batch normalization across 128 GPUs, resulting in high communication overheads. This work also presents results showing that larger mini-batches lead to higher accuracy and are faster to train. Using horizontal scaling alone to support larger batches is not sustainable due to the inter-GPU communication bottleneck. While our simulation infrastructure is unable to support such huge DL training networks, Buddy Compression enables modest vertical scaling, which could be combined with horizontal scaling for more sustainable solutions.

V. Conclusions

This work describes Buddy Compression, the first general-purpose mechanism to increase user-visible memory capacity on GPUs. Buddy Compression is enabled by modern high-bandwidth interconnects that allow a remote memory pool to be used as overflow storage for incompressible memory entries. Buddy Compression achieves 1.5–1.9× GPU capacity expansion across a wide range of HPC and DL workloads with only a 1–2% performance penalty relative to an unconstrained-capacity GPU, due to its unique design where compressibility changes require no additional data movement. This combination of high performance and compression ratios makes Buddy Compression an attractive and performant alternative to existing technologies like Unified Memory oversubscription.

VI. Acknowledgments

This research was funded in part by the U.S. Government under the DoE PathForward program. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. We thank Nikolay Sakharnykh and Mat Colgrove for helping us to curate a benchmark suite.

REFERENCES

[1] L. Gu, J. Siegel, and X. Li, "Using GPUs to Compute Large Out-of-Card FFTs," in *Proceedings of the International Conference on Supercomputing (ICS)*, 2011.

[2] F. Song, S. Tomov, and J. Dongarra, "Enabling and Scaling Matrix Computations on Heterogeneous Multi-core and Multi-GPU Systems," in *Proceedings of the International Conference on Supercomputing (ICS)*, 2012.

[3] O. Fuhrer, T. Chadha, T. Hoefler, G. Kwasniewski, X. Lapillonne,D. Leutwyler, D. Luthi, C. Osuna, C. Schar, T. C. Schulthesset, "Near-Global Climate Simulation at 1 KM Resolution: Establishing a Performance Baseline on 4888 GPUs with COSMO 5.0," in *Geoscientific Model Development*, 2018.

[4] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, Prabhat, and M. Houston, "Exascale Deep Learning for Climate Analytics," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2018.

[5] V. Allombert, D. Michea, F. Dupros, C. Bellier, B. Bourgine, H. Aochi, and S. Jubertie, "An Out-of-Core GPU Approach for Accelerating Geostatistical Interpolation," *Procedia Computer Science*, 2014. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01133110

[6] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun, "Megdet: A Large Mini-Batch Object Detector," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[7] X. Chen, D. Z. Chen, and X. S. Hu, "moDNN: Memory optimal DNN training on GPUs," in *Proceedings of the Conference on Design, Automation, and Test in Europe (DATE)*, Mar. 2018.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

[10] R. Prenger, R. Valle, and B. Catanzaro, "WaveGlow: A Flow-based Generative Network for Speech Synthesis," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Better Language Models and Their Implications," 2019. [Online]. Available: https://openai.com/blog/better-language-models/

[12] M. Wang, C.-C. Huang, and J. Li, "Supporting Very Large Models using Automatic Dataflow Graph Partitioning," *arXiv preprint 1807.08887*, 2018.

[13] T. Akiba, T. Kerola, Y. Niitani, T. Ogawa, S. Sano, and S. Suzuki, "PFDet: 2nd Place Solution to Open Images Challenge 2018 Object Detection Track," *arXiv preprint arXiv:1809.00778*, 2018.

[14] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vDNN: Virtualized Deep Neural Networks for Scalable, Memory-efficient Neural Network Design," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016.

[15] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[16] M. Harris, "Unified memory for CUDA beginners," NVIDIA Blog, 2016. [Online]. Available: https://devblogs.nvidia.com/unified-memory-cuda-beginners/

[17] T. Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler, "Towards High Performance Paged Memory for GPUs," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[18] A. A. Awan, C.-H. Chu, H. Subramoni, X. Lu, and D. K. Panda, "Can Unified-Memory Support on Pascal and Volta GPUs Enable Out-of-Core DNN Training?" in *Proceedings of the International Supercomputing Conference (ISC)*, 2018.

[19] R. B. Tremaine, P. A. Franaszek, J. T. Robinson, C. O. Schulz, T. B. Smith, M. E. Wazlowski, and P. M. Bland, "IBM Memory Expansion Technology (MXT)," in *IBM Journal of Research and Development*, vol. 45, no. 2, 2001.

[20] M. Ekman and P. Stenstrom, "A Robust Main-Memory Compression Scheme," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2005.

[21] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. Gibbons, M. Kozuch, and T. Mowry, "Linearly Compressed Pages: A Low-Complexity, Low-Latency Main Memory Compression Framework," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2013.

[22] J. Zhao, S. Li, J. Chang, J. L. Byrne, L. L. Ramirez, K. Lim, Y. Xie, and P. Faraboschi, "Buri: Scaling Big-Memory Computing with Hardware-Based Memory Expansion," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 12, no. 3, 2015.

[23] E. Choukse, M. Erez, and A. R. Alameldeen, "Compresso: Pragmatic Main Memory Compression," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2018.

[24] J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson, "Adaptive Scalable Texture Compression," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2012.

[25] A. Jain, A. Phanishayee, J. Mars, L. Tang, and G. Pekhimenko, "Gist: Efficient Data Encoding for Deep Neural Network Training," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, Jun. 2018, pp. 776–789.

[26] NVIDIA, "NVIDIA DGX-2: The World's Most Powerful AI System for the Most Complex AI Challenges." [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-2/

[27] OpenCAPI Consortium, "OpenCAPI 3.0 Data Link Specification." [Online]. Available: https://opencapi.org/wp-content/uploads/2016/09/OC-DL-Specification.10.14.16.pdf

[28] U. Pirzada, "Intel Hints Towards An X^e 'Coherent Multi-GPU' Future With CXL Interconnect." [Online]. Available: https://wccftech.com/intel-xe-coherent-multi-gpu-cxl/

[29] P. Messmer, "Large Scale Visualization on GPU-Accelerated Supercomputers," in *Proceedings of the Workshop on Ultrascale Visualization (UltraVis)*, 2015. [Online]. Available: http://vis.cs.ucdavis.edu/Ultravis15/slides/UltraVis_Messmer.pdf

[30] ——, "In Situ Visualization with Accelerators," NVIDIA Developer Blog, 2014. [Online]. Available: https://devblogs.nvidia.com/interactive-supercomputing-in-situ-visualization-tesla-gpus/

[31] P. Markthub, M. E. Belviranli, S. Lee, J. S. Vetter, and S. Matsuoka, "DRAGON: Breaking GPU Memory Capacity Limits with Direct NVM Access," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2018.

[32] N. Sakharnykh, "Beyond GPU Memory Limits with Unified Memory on Pascal," NVIDIA Developer Blog, 2016. [Online]. Available: https://devblogs.nvidia.com/beyond-gpu-memory-limits-unified-memory-pascal/

[33] G. Juckeland, W. C. Brantley, S. Chandrasekaran, B. M. Chapman, S. Che, M. E. Colgrove, H. Feng, A. Grund, R. Henschel, W. mei W. Hwu, H. Li, M. S. Müller, W. E. Nagel, M. Perminov, P. Shelepugin, K. Skadron, J. A. Stratton, A. Titov, K. Wang, G. M. van Waveren, B. Whitney, S. Wienke, R. Xu, and K. Kumaran, "SpecACCEL: A Standard Application Suite for Measuring Hardware Accelerator Performance," 2014.

[34] Lawrence Berkeley National Laboratory (LBL), "HPGMG: High-Performance Geometric Multigrid," 2017.

[35] Lawrence Livermore National Laboratory (LLNL), "Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH)," 2015.

[36] E. Choukse, M. Erez, and A. R. Alameldeen, "CompressPoints: An Evaluation Methodology for Compressed Memory Systems," *IEEE Computer Architecture Letters*, vol. 17, 2018.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, 2012.

[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[39] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5 MB Model Size," *arXiv preprint arXiv:1602.07360*, 2016.

[40] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[42] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proceedings of the International Conference on Multimedia*, 2014.

[43] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[44] R. Józefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the Limits of Language Modeling," *arXiv preprint arXiv:1602.02410*, 2016.

[45] L. Liu, L. L. Deng, X. Hu, M. Zhu, G. Li, Y. Ding, and Y. Xie, "Dynamic Sparse Graph for Efficient Deep Learning," *arXiv preprint arXiv:1810.00859*, 2018.

[46] C. D. Sa, M. Leszczynski, J. Zhang, A. Marzoev, C. R. Aberger, K. Olukotun, and C. Ré, "High-Accuracy Low-Precision Training," *arXiv preprint arXiv:1803.03383*, 2018.

[47] A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves, "Memory-Efficient Backpropagation Through Time," in *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, 2016.

[48] Y. Ito, R. Matsumiya, and T. Endo, "ooc_cuDNN: Accommodating Convolutional Neural Networks Over GPU Memory Capacity," in *Proceedings of the International Conference on Big Data (Big Data)*, 2017.

[49] Y. Kwon and M. Rhu, "Beyond the Memory Wall: A Case for Memory-Centric HPC System for Deep Learning," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2018.

[50] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, and T. Kraska, "Superneurons: Dynamic GPU Memory Management for Training Deep Neural Networks," in *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2018.

[51] "zram: Compressed RAM Based Block Devices," 2012. [Online]. Available: https://www.kernel.org/doc/Documentation/blockdev/zram.txt

[52] NVIDIA, "NVIDIA Turing GPU Architecture." [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[53] T. Trader, "TSUBAME 3.0 Points to Future HPE Pascal-NVLink-OPA Server," HPC Wire, 2017. [Online]. Available: https://www.hpcwire.com/2017/02/17/tsubame3-0-points-future-hpe-pascal-nvlink-opa-server/

[54] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated Memory for Expansion and Sharing in Blade Servers," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2009.

[55] A. B. Caldeira, "IBM Power System AC922 Introduction and Technical Overview," IBM International Technical Support Organization, Tech. Rep. REDP-5472-00, 2018. [Online]. Available: https://www.redbooks.ibm.com/redpapers/pdfs/redp5472.pdf

[56] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. R. Tallent, and K. J. Barker, "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 31, no. 1, 2019.

[57] The HSA Foundation, "Heterogenous System Architecture (HSA)." [Online]. Available: https://www.hsafoundation.com

[58] Advanced Micro Devices, "AMD Kaveri: Support for Heterogenous System Architecture (HSA)." [Online]. Available: http://www.amd.com/en-us/products/processors/desktop/a-series-apu

[59] N. Sakharnykh, "Everything you Need to Know About Unified Memory," 2018, GPU Technology Conference (GTC). [Online]. Available: http://on-demand.gputechconf.com/gtc/2018/presentation/s8430-everything-you-need-to-know-about-unified-memory.pdf

[60] ——, "Unified Memory On Pascal And Volta," 2017, GPU Technology Conference (GTC). [Online]. Available: http://on-demand.gputechconf.com/gtc/2017/presentation/s7285-nikolay-sakharnykh-unified-memory-on-pascal-and-volta.pdf

[61] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-Plane Compression: Transforming Data for Better Compression in Many-Core Architectures," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016.

[62] G. Pekhimenko, V. Seshadri, O. Mutlu, P. Gibbons, M. Kozuch, and T. Mowry, "Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[63] A. R. Alameldeen and D. A. Wood, "Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches," Computer Sciences Department, University of Wisconsin-Madison, Tech. Rep. 1500, 2004.

[64] J. Pool, A. Lastra, and M. Singh, "Lossless Compression of Variable-Precision Floating-Point Buffers on GPUs," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2012.

[65] A. Yang, H. Mukka, F. Hesaaraki, and M. Burtscher, "MPC: A Massively Parallel Compression Algorithm for Scientific Data," in *Proceedings of International Conference on Cluster Computing (CLUSTER)*, 2015.

[66] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking," *arXiv preprint arXiv:1804.06826*, 2018.

[67] Lawrence Livermore National Laboratory (LLNL), "Sierra," 2019. [Online]. Available: https://hpc.llnl.gov/hardware/platforms/sierra

[68] F. Foertter, "Preparing GPU-Accelerated Applications for the Summit Supercomputer," GPU Technology Conference (GTC), 2017. [Online]. Available: http://on-demand.gputechconf.com/gtc/2017/presentation/s7642-fernanda-foertter-preparing-gpu-accelerated-app.pdf

[69] National Institute of Advanced Industrial Science and Technology (AIST), "About ACBI: Computing Resources," 2019. [Online]. Available: https://abci.ai/en/about_abci/computing_resource.html

[70] G. Panwar, D. Zhang, Y. Pang, M. Dahshan, N. DeBardeleben, B. Ravindran, and X. Jian, "Quantifying Memory Underutilization in HPC Systems and Using It to Improve Performance via Architecture Support," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2019.

[71] NVIDIA, "Pascal MMU Format Changes," Open GPU Documents, 2016. [Online]. Available: https://nvidia.github.io/open-gpu-doc/pascal/gp100-mmu-format.pdf

[72] N. Sakharnykh, "Memory Management on Modern GPU Architectures," 2019, GPU Technology Conference (GTC). [Online]. Available: https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9727-memory-management-on-modern-gpu-architectures.pdf

[73] NVIDIA, "CUDA 9.2 Update," OLCF User Group Call, 2018. [Online]. Available: https://www.olcf.ornl.gov/wp-content/uploads/2018/03/olcf_cuda9_2_update.pdf

[74] G. Pekhimenko, E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry, and S. W. Keckler, "A Case for Toggle-Aware Compression for GPU Systems," *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[75] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

[76] U. Milic, O. Villa, E. Bolotin, A. Arunkumar, E. Ebrahimi, A. Jaleel, A. Ramirez, and D. Nellans, "Beyond the Socket: NUMA-Aware GPUs," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2017.

[77] V. Young, A. Jaleel, E. Bolotin, E. Ebrahimi, D. Nellans, and O. Villa, "Combining HW/SW Mechanisms to Improve NUMA Performance of Multi-GPU Systems," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2018.

[78] NVIDIA, "NVIDIA Pascal GPU Architecture." [Online]. Available: https://www.nvidia.com/object/pascal-architecture-whitepaper.html

[79] ——, "NVIDIA Volta GPU Architecture." [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[80] E. Eshelman. (2017) Comparing NVLink vs PCI-E with NVIDIA Tesla P100 GPUs on OpenPOWER Servers. [Online]. Available: https://www.microway.com/hpc-tech-tips/comparing-nvlink-vs-pci-e-nvidia-tesla-p100-gpus-openpower-servers/

[81] A. Bakhoda, G. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.

[82] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2017.

[83] N. Beck, S. White, M. Paraschou, and S. Naffziger, "Zeppelin: An SoC for Multichip Architectures," in *Proceedings of the International Solid State Circtuits Conference (ISSCC)*, 2018.

[84] D. Foley and J. Danskin, "Ultra-Performance Pascal GPU and NVLink Interconnect," *IEEE MICRO*, vol. 37, no. 2, 2017.

[85] H. Qi, E. R. Sparks, and A. S. Talwalkar, "Paleo: A Performance Model for Deep Neural Networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
[86] S. Lym, D. Lee, M. O'Connor, N. Chatterjee, and M. Erez, "DeLTA: GPU Performance Model for Deep Learning Applications with In-Depth Memory System Traffic Analysis," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.
[87] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-100 Dataset (Canadian Institute for Advanced Research)," 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html
[88] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," *arXiv preprint arXiv:1706.02677*, 2017.
[89] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the Effects of Data Parallelism on Neural Network Training," *arXiv preprint arXiv:1811.03600*, 2018.
[90] Y. Wu and K. He, "Group Normalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.
[91] T.-Y. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.