2018 51st Annual IEEE/ACM International Symposium on Microarchitecture

CABLE: a CAche-Based Link Encoder for Bandwidth-starved Manycores

Tri M. Nguyen
Department of Electrical Engineering
Princeton University
Princeton, USA
trin@princeton.edu

Adi Fuchs
Department of Electrical Engineering
Princeton University
Princeton, USA
adif@princeton.edu

David Wentzlaff
Department of Electrical Engineering
Princeton University
Princeton, USA
wentzlaf@princeton.edu

***Abstract*—Off-chip bandwidth is a scarce resource in modern processors, and it is expected to become even more limited on a per-core basis as we move into the era of high-throughput and massively-parallel computation. One promising approach to overcome limited bandwidth is off-chip link compression. Unfortunately, previously proposed latency-driven compression schemes are not a good fit for latency-tolerant manycore systems, and they often do not have the dictionary capacity to accommodate more than a few concurrent threads. In this work, we present CABLE, a novel CAche-Based Link Encoder that enables point-to-point link compression between coherent caches, re-purposing the data already stored in the caches as a massive and scalable dictionary for data compression. We show the broad applicability of CABLE by applying it to two critical off-chip links: (1) the memory link interface to off-chip memory, and (2) the cache-coherent link between processors in a multi-chip system. We have implemented CABLE's search pipeline hardware in Verilog using the OpenPiton framework to show its feasibility. Evaluating with SPEC2006, we find that CABLE increases effective off-chip bandwidth by 7.2× and system throughput by 3.78× on average, 83% and 258% better than CPACK, respectively.**

***Index Terms*—cache memory, data compression, parallel processing**

I. Introduction

As modern throughput-oriented processors integrate hundreds if not thousands of threads within a single chip [1]–[7] or a multi-socket system [8]–[10], off-chip bandwidth has become the dominant limitation to achieve higher performance. In single-socket systems, this bottleneck is at the chip-to-memory (DRAM) interface where thousands of threads compete for a minuscule share of memory bandwidth. In multi-chip systems, limited chip-to-chip cache coherence bandwidth (e.g., QPI [11], Infinity Fabric [12]) also limits the amount of sharing and inter-chip coherence requests.

Limited off-chip bandwidth is a major challenge because I/O technology has not been improving as fast as cores can be added to a single chip. Adding I/O bandwidth comes at either high monetary cost (e.g., HBM [13] needing an interposer layer), or significant energy consumption, as high-speed serial links (SERDES) require complex clock recovery schemes, equalization, and 10GHz+ PLLs to operate. If left unsolved, the bandwidth-wall [14]–[16] will effectively halt further progress for high throughput computing systems.

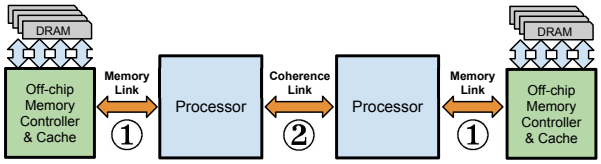


Fig. 1. Two bandwidth-starved off-chip links are shown: ① memory link between the processor and memory, and ② coherent link between processors.

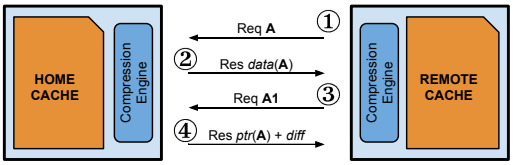


Fig. 2. Shared cache lines are used as dictionaries. ① request and ② response for line `A` occur uncompressed, but ③ a request for A1, a similar data, ④ can be compressed as a pointer to `A` plus a `diff`.

One promising approach is the compression of off-chip links to increase the effective bandwidth. Link compression has been well-studied in the literature [17]–[22], ranging from simple zero-encoders [23, 24] to sophisticated LZ77/gzip engines [22, 25]. Unfortunately, there are two primary shortcomings of prior work when viewed in the context of manycore processors. First, prior work tends to optimize for single-threaded performance, favoring simple non-dictionary algorithms to minimize compression latency. They were not designed for throughput-oriented manycore processors and GPUs which are explicitly designed to tolerate high memory latency. Second, in shared systems, dictionary pollution increases as the number of concurrent threads grows, nullifying the benefits of having a dictionary in the first place.

To address the above issues, this paper proposes CABLE, a novel point-to-point CAche-Based Link Encoding framework that exploits shared data already stored in caches as a massive and scalable dictionary. Shown in Fig. 1, CABLE compression can be applied between any two coherent caches, such as ① at the memory link between the LLC and off-chip L4, or ② at the coherence links between LLCs in a multi-chip system.

CABLE's key mechanism is depicted in Fig. 2. In this example, line `A` and `A1` are similar in content but are otherwise

978-1-5386-6240-3/18/$31.00 ©2018 IEEE
DOI 10.1109/MICRO.2018.00033

located in unassociated memory addresses. Since `A` already exists in the *remote cache* (e.g., the on-chip LLC) from a prior request, instead of sending the raw data of `A1`, the *home cache* (e.g., the off-chip L4 or an LLC in another chip) can send a much shorter `DIFF` and a pointer to `A`, from which the remote cache can then reconstruct the content of `A1`.

To our knowledge, CABLE is the first work to use the shared cache data itself (in-place) as a dictionary for data compression. By having a much larger dictionary to find similarity across, the compression ratio of off-chip links can be significantly increased. Evaluating with SPEC2006, we demonstrate that CABLE is well-suited for emerging manycore architectures, increasing effective off-chip bandwidth by 7.2× and system throughput by 3.78× on average, 83% and 258% better than CPACK [26], respectively. Having a dictionary that scales with the amount of on-chip cache, CABLE is also well-suited for shared systems: its compression performance is further enhanced by 60% in multiprogram workloads, whereas `gzip` suffers by 15% due to dictionary pollution. Lastly, while energy efficiency is not the main focus of this work, we find that CABLE can reduce memory subsystem energy by 15% on average while incurring less than 5% single-threaded performance loss.

By having a more complex compression scheme, with CABLE, we consciously trade compression latencies for off-chip bandwidth. The key argument is that since modern manycores have already sacrificed single-threaded performance to have better throughput and energy efficiency, it makes sense to spend more time to find more commonality at the compression step to increase the effective bandwidth. By saving more off-chip bandwidth, more threads can be consolidated onto a single processor thereby increasing computational throughput. This paper makes the following contributions:

- We design an architectural framework to exploit data commonality in caches as a massive and scalable compression dictionary.
- We provide a methodology of how to exploit cache-specific data patterns and identify data similarity by use of data *signatures*.
- We show the broad applicability of CABLE by applying CABLE at the memory link and coherence link–two critical off-chip links.
- We implement CABLE's search pipeline in Verilog using the OpenPiton framework to show its implementation feasibility.
- We evaluate CABLE across SPEC2006 and demonstrate that CABLE is 91% better at compression than a scalable state-of-the-art solution (CPACK), consequently resulting in a 378% throughput increase on average.

II. CABLE

A. Motivation & Challenges

We started this project with the observation that there exists significant data similarity across cache lines. Prior work in cache compression and cache deduplication [27]–[32] has shown that there is significant data redundancy in the last-level

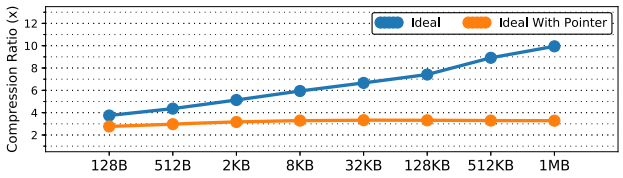


Fig. 3. Compression ratios of two ideal dictionary algorithms against increasing dictionary sizes: one with and one without pointer overhead.

cache. Building upon this intuition, we wanted to see how high compression ratio scales with increasing dictionary (cache) size. As plotted in Fig. 3 [1], and ignoring pointer overhead for a moment, `Ideal` encouragingly shows that the compression ratio increases with dictionary size.

Challenge: Pointer overhead. However, pointer overhead[2] is a critical issue. Typically, data is compressed by replacing each matching 32-bit word with a pointer. Because pointers become larger as dictionary increases in size, when accounting for these pointers, `Ideal With Pointer` in Fig. 3 shows no improvement in compression ratio. This result mirrors that of prior work where the optimal dictionary size was found to be around a modest 128 bytes [33].

CABLE tackles the pointer overhead challenge with two key mechanisms: (1) amortizing a single pointer over multiple data words by pointing to 64-byte cache lines instead of 32-bit word unlike prior work, and (2) reduction of pointer size by more than 50% using a novel Way Map Table.

Challenge: Finding similarity. Another challenge is identifying similarity in a huge dictionary–a brute-force approach is simply infeasible. Instead, like software compression [22, 34], we build a search index in hardware using signatures (§III-A) and hash tables (§III-B). Unlike general purpose compression though, we tailor the signature mechanism to exploit the sparsity in data cache lines.

Challenge: Synchronization. Lastly, synchronizing dictionaries across caches can be hard. In this work, instead of restricting cache eviction policy or adding complexity to existing cache coherence protocols, we assume that the home cache (e.g., off-chip L4) is inclusive of the remote cache (e.g., on-chip LLC), which aids in identifying which line is present in both caches. Additionally, we do not use cache lines in exclusive and modified states as dictionary reference data because modified lines can be changed silently leading to incorrect decompression.

B. High-level Design

Note that CABLE is a compression *framework* and not a compression algorithm. Its main function is to find similar cache lines; the actual compression operation is delegated to existing compression algorithms such as CPACK [26], LBE [29], or

[1]Using CPACK (modified with configurable dictionary size) minus symbol overheads. Profiled with non-trivial benchmarks in SPEC2006.

[2]Not to be confused with pointers in programming, in data compression, a pointer refers to the reference data. It can be either a dictionary table index or a buffer offset. In CABLE a pointer points to a cache block.

TABLE I
GLOSSARY OF CABLE-SPECIFIC TERMS

Home Cache	Larger cache that services and compresses memory requests
Remote Cache	Smaller cache that receives and decompresses data
References	Shared lines that exist in both home and remote caches
DIFF	A compressed representation using cache lines as references
Signature	Shortened (32b), unique representation of a cache line
HomeLID	Index + way of a cache line stored in the home cache
RemoteLID	Index + way of a cache line stored in the remote cache
Hash table	A table mapping *hash*(signature) → HomeLID
Way-map table	A tag-like table used to translate HomeLID → RemoteLID

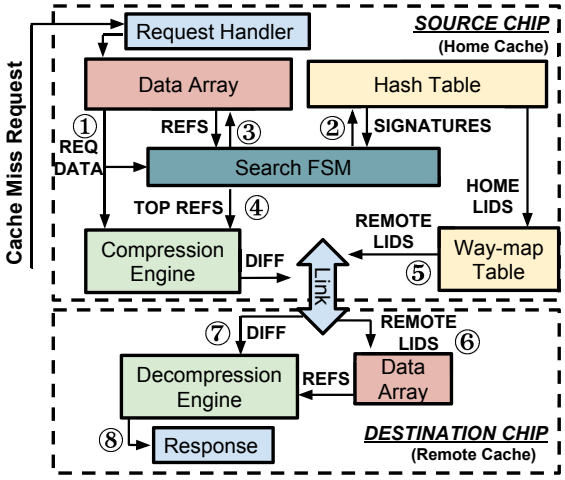


Fig. 4. Datapath of CABLE showing main components for a home cache communicating with a remote cache.

LZ77/gzip. The framework is composed of four distinct tasks (see Table I for definitions): (i) searching for similarities using signatures, (ii) compressing using references as seeds, (iii) transmitting the compressed *DIFF* and reference pointers, and (iv) synchronizing valid signatures in the hash tables between the caches.

Fig. 4 gives a walk-through of the compression sequence from the home cache (the larger cache) to the remote cache (the smaller cache) in response to a cache request. ① First, given a cache line to search for, CABLE analyzes and extracts *signatures* from it. ② CABLE uses these signatures to find similar cache lines (termed *references*) that share the same signatures, which is done by indexing into the hash table to obtain the cache index+way (*HomeLIDs*) of these cache lines. ③ CABLE then uses the HomeLIDs to read the reference candidates from the data array, ranks them by data similarity, and ④ selects the best three to produce the *DIFF*. Concurrently, to minimize pointer overhead, CABLE reduces cache tags into *RemoteLIDs* using the way-map table (WMT), and ⑤ sends these with the *DIFF* over the link. ⑥ To decompress the *DIFF*, the remote cache uses the *RemoteLIDs* to read the reference lines, ⑦ decompresses the *DIFF*, and ⑧ reconstructs the original data before installing it in the cache.

Structurally, there are two new data structures: the hash table and the WMT. Given a source signature, the hash table can be used to locate other cache lines with similar signatures, and the WMT is used to compress cache line tags into RemoteLIDs. Note that these structures are not content addressable memories (CAMs), but rather standard SRAMs, and that the WMT only exists at the home caches.

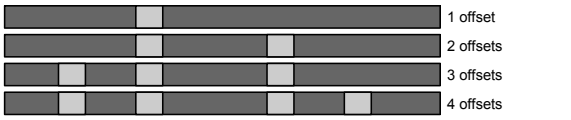


Fig. 5. Example default signature offsets. For searching, all potential signatures are extracted and checked.

Fig. 6. To find representative 32-bit signatures, CABLE skips *trivial* values.

C. Assumptions & Limitations

To simplify the design, we make several assumptions based on the characteristics of today's modern architectures. First, we assume an *inclusive* cache hierarchy. Practically, for the memory-link compression use case, this means the off-chip L4 [3] contains the on-chip LLC entirely. For coherence-link compression, such as between chip modules in an MCM package [10], it implies that the remote cache (L3) is inclusive of the local cache (L2). Assuming inclusivity simplifies data synchronization, though we also discuss *non-inclusive* extensions in §IV-C. Note that CABLE does not (and should not) use dirty lines as references.

We assume that the off-chip links are *point-to-point ordered* to simplify correctness; further discussions of race conditions and out-of-order links are in §IV-A. In general, a *non-silent eviction* cache coherence design is assumed to reduce synchronization complexity, but certain memory layouts allow *silent evictions* (§IV-B). Also, CABLE by itself does not change the underlying coherence protocol–it only modifies and reduces the data payload transmitted. As such, CABLE is decoupled from replacement policies because it tracks cache line evictions precisely, although one requirement is that the remote cache (smaller cache) needs to include the replacement way info in the cache request, as done in some modern architectures like the UltraSPARC T1/T2 [36, 37].

III. ARCHITECTURE

In this section, we discuss how the home cache, given a request, uses the hash table to search for commonalities in the local cache and uses the WMT to reduce reference pointer size. First, we describe the process of generating signatures and how signatures can be used to find and rank shared data.

A. Signature Extraction

Signatures are succinct and unique representations of a cache line. However, the requirements for a good signature

[3]Such as the IBM POWER8/9 Centaur [9] and the Intel Haswell eDRAM cache [35].

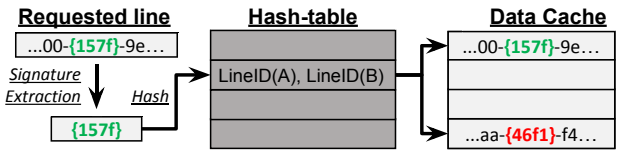


Fig. 7. An example of searching the hash table. The hash entry contains LineIDs to two lines in the cache. The first line has the correct signature and is a perfect match. The second line is a hash collision and shares no similarity.

representation are often conflicting: not only does it need to be short to minimize storage and unique to find similarities with high accuracy, but also flexible enough to ignore slight edits, shifts, and permutations.

In CABLE, we explore the possibility of sampling 32-bit words as signatures. We choose one or more offsets in the cache line as shown in Fig. 5, and compute the hash of those 32-bit words for use as the signatures. We found empirically that a well-selected 32-bit signature can represent the content of a full cache line well. In most benchmarks, while zeroes are abundant, non-zero words are distinct, and the sequence of these words tend to stay the same. For example, copies of an object or objects of in an array typically have the same data layout with minimal modifications but might be byte-shifted.

To avoid hashing zeroes as signatures, we move the hash offset forward when the data word is *trivial*, defined as having 24 bits or more of leading zeroes or ones. Fig. 6 shows examples of how signature offsets are moved forward when the current offset points to a trivial data. Unlike general-purpose compression like gzip [22], CABLE shifts offsets by four bytes instead of one based on the observation that shifting by a byte does not improve compression performance significantly, presumably as data objects in many programming languages such as C++ are aligned to 32-bit or 64-bit boundaries.

B. Hash table

The hash table is a standard key-value data structure that maps *signatures* to *LineID*. We use it to find *reference candidates* given one or more signatures of the requested data.

When building the search index, two signatures per cache line are extracted and inserted into the hash table. When searching, all non-*trivial* signatures generated from the requested data are used to search for references. Signatures are invalidated when caches desynchronize, such as when a line is evicted. CABLE computes the signatures for these lines and removes the LineID of the invalidated line from the corresponding hash entry if found. Each entry stores two LineIDs by default.

The hash table is inherently inexact and simply gives possible matches. Hash collision–when different signatures hash into the same entry–is one such source of inaccuracy. Fig. 7 demonstrates an example where the hash table puts two dissimilar cache lines in the same entry, resulting in a false-positive result. Keeping hash collision low is one reason only two signatures are inserted during synchronization.

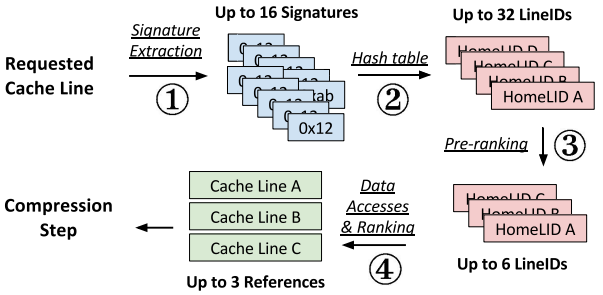


Fig. 8. Data flow of the search step, where signatures are extracted and searched in the hash table for up to six references, then ranked to get the best three.

TABLE II
ENERGY COMPARISON BETWEEN MEMORY OPERATIONS FOR 64 BYTES.

Operation	Energy	Scale
CPACK Compression [26]	50 pJ	1×
Cache access (1MB slice) [38]	100 pJ	2×
Off-chip IO link [39]	15 nJ	300×
DRAM access [40]	50.6 nJ	1000×

C. Search

Fig. 8 summarizes CABLE's searching process. Assuming 64-byte cache lines and 32-bit signatures, CABLE extracts up to 16 signatures (often much less due to zeroes, and potentially non-unique signatures) from the requested cache line ①. Then using these signatures to index the hash table, it gets up to 32 LineIDs (assuming bucket size of two) ②.

Of these LineIDs, CABLE *pre-ranks* and selects the top six most duplicated LineIDs ③. When references are very similar to the requested data, different signatures often map to the same LineIDs, resulting in duplicated entries in the outputs of the hash table. These entries are prioritized as they are more likely to contain more similarities than the others.

CABLE then ranks the reference candidates by data similarity to get the best three (or less) references. Ranking mechanism is as followed. First, a 16-bit coverage bit vector (CBV) is computed for each candidate, indicating exact 32-bit word matches when compared to the requested cache line. Then, CABLE greedily compares the CBVs to select up to 3 lines that maximize coverage of the requested data. For instance, suppose we have CBVs 1100 and 0110, the combined CBV is 1110 with a coverage of 3. If a third CBV is 0011, CABLE drops 0110 and selects 1100 and 0011 for a combined CBV of 1111 and coverage of 4.

As a baseline parameter, six cache lines are accessed from the data array for the final ranking step. We choose this number partly because of empirical results (Fig. 22), and because is inexpensive to access the data array directly without tag checks. Latency-wise, SRAMs usually take one cycle to access, while eDRAMs take between 1ns [41] and 3ns [42]. Energy-wise, it is also advantageous to be generous with searching as link energy is typically hundreds of times more energy consuming (shown in Table II).

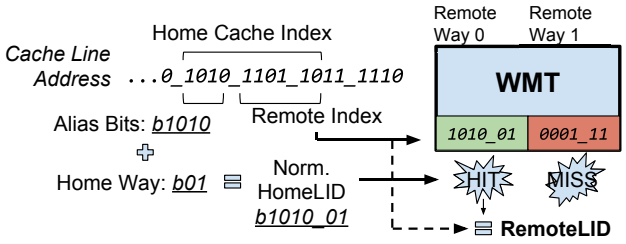


Fig. 9. An example access where a hit in the WMT indicates that the cache line exists in way 0 of the remote cache.

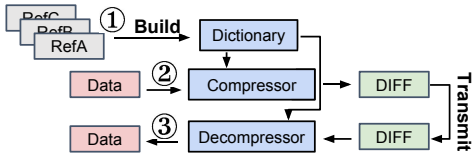


Fig. 10. Compression and decompression steps. ① CABLE builds a temporary dictionary using references; ② This dictionary is used to make the DIFF. ③ The same dictionary is also used to reconstruct data from the DIFF at the receiver.

D. Way-Map Table

Cache tags could be used as reference pointers, but we can do better. HomeLIDs and RemoteLIDs (index+way of a line locally and remotely) are a means to reduce transmission overhead: at 17 bits, LineIDs reduce pointer overhead by 57.5% versus 40-bit tags.

The WMT is a data structure in the home cache that tracks the cache ways of the *remote* caches. The structure mirrors the layout of the remote cache such that a tag hit in the WMT indicates the index and way of the remote cache. The entries themselves, however, are HomeLIDs which uniquely map to cache lines in the home cache (and not the remote cache). Furthermore, to reduce the size of the WMT, the HomeLIDs are *normalized* (alias+way, where "alias" is the home cache index minus remote index bits, instead of index+way).

Fig. 9 shows how the WMT reduces a cache tag to a RemoteLID. First CABLE computes the normalized HomeLID by appending the alias bits and the home way. Using the remote cache indexing bits to access the WMT, CABLE checks for a HomeLID match. If not found, the line is not guaranteed to exist in the remote cache. If found, the position of the WMT entry indicates the remote way, which can be used to form the RemoteLID.

E. Compression & Transmission

Having found the top three references in the ranking step, the compression engine builds a temporary dictionary using the references to compress the requested data. Fig. 10 shows the data path of the compression step. Concurrently with the described search pipeline, we also compress the requested data without any seed. If the resulting compression ratio is above a certain threshold (ie., 16×) or better than the *DIFF*, we send the compressed data without searching for references.

Payload overheads are minimal: a 1-bit flag is needed to denote whether the data is compressed or uncompressed, 2 bits to specify the number of references, which are followed by the RemoteLIDs and the variable-length DIFF. The DIFF length is not needed because the decompressed data length is fixed. In the evaluation, a 16-bit bus is assumed which limits max compression to 32×, but if a smaller width link is used (e.g., PCIe x1/x2), there is potential to save even more bandwidth as CABLE could save sub-16b portions of data across the link.

Once data is received across the link, the receiving cache uses the provided LineIDs to fetch the references to seed a temporary dictionary and decompress the DIFF (Fig. 10).

F. Synchronization

CABLE needs to update the hash tables and the WMTs for two events: sending/receiving cache lines, and cache invalidations. CABLE compresses both clean and dirty requests, but only cache lines sent in the "shared" state are incorporated into the hash table of the home cache for future compressions. Correspondingly, the same cache line is inserted into the remote cache's hash table for future write-back compressions. Simultaneously, by checking the provided way-replacement info and the WMT, the home cache can determine whether this request displaces some other cache line in the remote cache. If so, CABLE reads this cache line, extracts its signatures, and invalidates these entries from the hash table.

Hash table and WMT invalidations also happen for coherency events that invalidate shared cache lines from the remote cache, such as a snoop invalidation, a cache eviction at the home cache, or an upgrade request (from shared to dirty). It is worth reiterating that since CABLE tracks synchronization precisely with the WMT, its operations are decoupled from cache replacement policy.

G. Write-back Compression

Write-back compression from the remote to home cache follows a similar sequence with some minor differences. Unlike the home cache, the remote cache does not have a WMT; it simply sends its own LineIDs. At the home cache, this LineID is split into remote index and way bits, which are then used to retrieve the HomeLIDs stored in the home cache's WMT. Second, the remote cache updates its hash table only when receiving data from the home cache, and not for write-backs.

IV. Discussion

A. Race Conditions

An important corner case that CABLE must solve is when the home cache selects a reference, and concurrently it is being evicted from the remote cache–CABLE cannot decompress a response that points to missing (evicted) references.

A simple solution is to implement a small *eviction buffer* at the remote cache to temporarily hold a copy of unacknowledged evictions, which will work even with an out-of-order link transport such as Intel's QPI. For instance, each eviction is assigned a sequence number, `EvictSeq`, and a copy is inserted in the *eviction buffer*. This tag is embedded in the next, unrelated memory request, and the home cache acknowledges by embedding the last seen `EvictSeq` in the response to indicate which entries are safe to be removed.

B. Silent Evictions

Baseline CABLE assumes notifying (non-silent) evictions, but is not a strict requirement. In certain memory configurations where no two home caches track the same remote cache set, silent eviction can be used. Here are two common cases:

1-1 mapping occurs when a single home cache exclusively backs a remote cache, like between the LLC and a single DRAM channel seen at ① in Fig. 1. In architectures like the SPARC T1/T2 [36, 37], eviction messages are implicitly embedded in the cache requests as the requests contain the way-replacement info (where the requested data would be placed in). Embedding replacement-way info allows both caches to track evicted data without an explicit eviction notice.

Linear address interleaving of the remote cache across multiple home caches occurs when address interleaving is linear and the number of home caches is a power of two. For instance, when a LLC is linearly interleaved across four DRAM channels, each cache set is exclusively backed by one DRAM channel. As such, like 1-1 mapping, the replacement-way info can be used to implement implicit eviction without incurring race conditions among the DRAM caches.

C. Non-inclusive cache extension

While CABLE assumes inclusive caches, it can be adapted to work with non-inclusive cache hierarchies, like the Intel Haswell-EP [43] multi-chip NUMA architecture. Here, there is a single home cache (called Home Agent) for each memory address, but remote caches (called Caching Agents) can keep private copies using MESIF states. The home cache is not inclusive of remote caches.

The non-inclusiveness is fundamentally not a problem since home caches keep track of all copies in directories for cache coherence purposes even if they don't have the actual data. CABLE can opportunistically use the sharing between home and remote caches to compress traffic between LLCs or between LLC-L4.

The issue with non-inclusiveness is with write-back compression. In inclusive caches, a remote node implicitly knows that its cache lines are guaranteed to exist at the home node. This assumption is no longer true for non-inclusive caches. Solutions include disabling write-back compression, or compressing write-backs with a non-dictionary algorithm.

D. Latency & Area Overhead

While CABLE is primarily a throughput technique where memory latency is not the most important parameter, we characterize it as we think it is useful to know. To further validate the feasibility of our design, we implement the search pipeline in OpenPiton [44], an open-source manycore framework.

Search Latency: The search stage includes hashing signatures, accessing the hash table, reading the data cache, building coverage vectors, and ranking coverage. Except for the final ranking step, the processing of each signature is independent and highly parallelizable. Nevertheless, throughput is limited by the number of read ports of the hash table SRAM. Assuming the hash table is 2-way banked, only two signatures can be checked concurrently. At four cycles for data array reads[4] and one cycle for other steps, the search latency for one signature is eight cycles. With 16 signatures and throughput of two signatures per cycle, the total search latency is 16 cycles. Note that this is the *worst-case* latency; when the requested data is highly redundant or filled with zeroes, there would be fewer signatures that need to be checked, reducing the total search latency to as little as eight cycles. As latencies are implementation dependent, we conservatively modeled CABLE with the worst-case latency in the results.

Search Implementation: Compared with gzip/LZ77 which can be highly complex to implement in hardware [45], CABLE is relatively simple. We built the entire search pipeline for the L2 cache of OpenPiton [44], an open-source manycore framework. To compute the signatures from 32-bit words, we implemented H_3 [46, 47], a simple yet high performance hash function. In OpenPiton, the data array access latency is one cycle per read, and even without aggressive pipelining, we were able to achieve the nominal operating frequency of OpenPiton using the IBM 32nm SOI standard cell library.

Compression Latency: Compression and decompression are done in two steps: (1) build the dictionary from references, and (2) compress or decompress the *DIFF*. Assuming a compression rate of 8B/cycle [26] and 64B cache lines, both steps take 8 cycles each for a total of 16 cycles. Adding the latency of the search step, the end-to-end latency is 48 cycles and is summarized in Table IV.

Energy Efficiency: As data transfers across the off-chip links often consume hundreds of times more energy than on-chip operations (Table II), energy reduction from saving link energy outweighs the compression and cache access energy consumption.

The search step reads up to six reference candidates, plus up to three data array accesses at the receiving cache for a total of nine cache reads, or 0.9nJ. The compression and decompression steps process up to 3 references each to build the temporary dictionary, and during search up to six references are processed to construct the coverage vector for ranking. Scaling known numbers [26] to *32nm*, a conservative estimation of each compression operation is 0.7nJ. In total, the worst-case energy consumption is 1.6nJ per request, a little over $\frac{1}{10}$ of an off-chip link transfer (15nJ).

Area Overhead: In addition to the compressor engines which can be estimated as 0.02mm^2 in *32nm* [26], CABLE has two additional SRAM structures: the hash table and the way-map table. They impose between 1% and 3% of SRAM overhead. When implemented and synthesized with OpenPiton in *32nm*, the CABLE search pipeline adds 1.48% of logic overhead to an L2 slice. Interestingly, the pre-ranking engine was the most complex piece of logic in this implementation as it needs to accept and sort four LineIDs per cycle. All overheads are

[4] A data array access without tag check typically takes around 1ns [41, 42] for eDRAMs.

TABLE III
CABLE AREA OVERHEADS.

	Off-chip		Multi-chip
	Buffer	On-chip Cache	Last-level caches
Hash table	1.76%	3.32%	2.50%
Way-map table	0.4%	-	1.74%
RemoteLID width	17b	18b	17b

Search Logic Overheads

	CABLE (area)	Per-L2(%)	Per-Tile(%)
Combinational	3377	0.71%	0.28%
Buffers	1247	0.26%	0.10%
Noncombinational	2407	0.51%	0.20%
Total	7031	1.48%	0.58%

reported as a percentage of the data cache size assuming 128KB L2, summarized in Table III.

For off-chip link compression (assuming 8-way 8MB LLC, 8-way 16MB DRAM buffer), each WMT entry is 4 bits (1 alias + 3 associativity bits) hence the storage overhead is 0.4% at the home cache (remember that only home caches have WMTs). Similarly, for multi-chip use case, WMT overhead is 0.58%, but since each processor has three WMTs, the total is 1.74%.

For coherence compression among multiple processors, we elected to have one WMT per link-pair for small configurations. For large systems, WMT information can be pooled into a single, competitively shared super-WMT/hash-table managed like a cache to decrease storage overheads and improve scalability.

Hash table sizing is independent of cache sizes and scales gracefully with more or fewer entries: scaling downward, a table with half as many entries can retain signatures of the most recent half, while scaling upward alleviates hash conflicts. We define a hash table as "full-sized" if it has as many entries as there are cache lines in the home cache. Generally, each full-sized hash table is 3.5% the size of the data cache (16MB cache, 18-bit HomeLIDs). With 128-byte lines (as common in some architectures) this overhead drops to 1.6%.

V. USE CASES FOR CABLE

Fig. 1 shows two bandwidth-deprived off-chip links that CABLE can alleviate. In this section, we discuss how CABLE can be used in these systems.

A. Off-chip memory link

To alleviate the bandwidth wall, modern architectures have been implemented with memory buffers just before the DRAM chips. Examples include the IBM POWER8/9 [9] which incorporates a 16MB off-chip eDRAM buffer at each DRAM channel, and the Intel Skylake which can be configured with a 128MB eDRAM memory-side cache [35]. Furthermore, interposer-based 3D-stacked memories such as HBM/HMC have allowed SRAM caches in the DRAM base logic layer [48]. For these systems, CABLE can be applied to compress request and write-back traffic between the CPU and the off-chip L4 caches. Data sent from L4 to LLC is compressed on both hit and misses. For misses, first the L4 fetches data from main memory, then compression continues as if it was a hit. For POWER8/9, the benefits include increased bandwidth and throughput. For Skylake, CABLE obviates the need of an on-package high-speed link and reduces cost by moving the buffer off-package. For both designs, CABLE can reduce link frequency while maintaining the same effective bandwidth for energy savings.

B. Multi-chip cache coherent link

Cache coherent multi-chip chip-multiprocessor (CMP) designs are prevalent, especially in enterprise settings where major manufacturers all ship multi-socket systems at the high end of their product range. GPU-based architectures are also going coherent across nodes, such as NVIDIA's cache coherent NVLINK [49] and NVIDIA's MCM [10] where multiple coherent chips are placed on an interposer layer. An advantage of these systems over the single-chip variant is the ability to run applications with large memory footprint by interleaving the memory space across chips and memory channels. The energy cost and communication latency between chips limit the performance and scalability, however.

We propose to compress the coherence-link by applying CABLE between each chip-to-chip point-to-point (PTP, eg., QPI [11], NVLINK [49]) link. We assume a NUMA system where memory pages are interleaved across nodes, and that a node can cache data copies homed in other chips and that caching is inclusive (like seen with Intel Haswell-EP [50] and NVIDIA MCM [10]). Like memory-link compression, CABLE compresses data both on data request and write-back between remote and home nodes. In a four-chip system, for instance, the system is fully-connected where each chip has three PTP links directly connecting it to the other three chips for a total of six PTP links and CABLE pipelines.

VI. RESULTS

A. Methodology

We modified PriME [51] simulator and SPEC2006 benchmarks to evaluate CABLE. SimPoint [52] traces of 130M and 1B instructions are used, both of which are available publicly online [53, 54]. The 130M instruction traces (10 phases, 100M warm-up, 30M profiled for a total of 1300M instructions) are used for most single-threaded studies because they are both faster and more representative. Multi-program studies instead use 1B traces, where each program is run for at least 250M instructions but is kept running until all have finished 250M instructions to sustain loads on other programs. This methodology matches that of prior work [55, 56]. To further increase the fidelity of the study, we remove phases that consist mostly of loading and storing zeroes as they can artificially boost the overall compression ratio [5]

To model system performance, we configured the core frequency and memory subsystem as detailed in Table IV. Latency studies use numbers in Table IV, which for CABLE

[5]Our findings hold whether zero-dominant phases are included or not. These phases are identified with the criteria that CPACK achieves 10×+ off-chip link compression with either 64KB LLC or 128KB LLC.

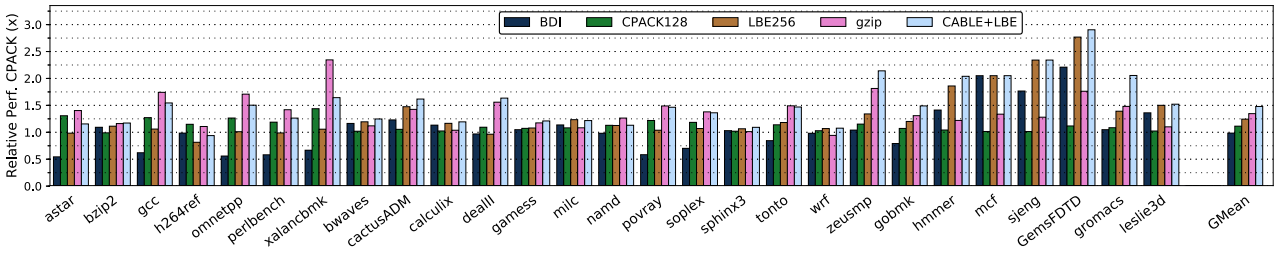


Fig. 11. Off-chip link compression (normalized to CPACK)

TABLE IV
DEFAULT SYSTEM CONFIGURATION

Core	2.0GHz, in-order x86 1 CPI non-memory instructions
L1	32KB per-core, private, single-cycle 4-way set associative
L2	128KB per-core, private 8-way set associative, 4-cycle
LLC	1MB per-core 8-way set associative, 30-cycle shared inclusive (within chip)
Off-chip Link	16-bit wide @ 9.6GHz (19.2GB/s) Single-threaded studies: single-channel Throughput studies: quad-channel 20ns setup latency
DRAM Buffer	4MB per-core 16-way set associative, 30-cycle
NUMA	4 nodes, round-robin page allocation
DRAM Link	64-bit wide @ 1.6GHz (12.8GB/s)
DRAM	4 MCs per chip/buffer FCFS controller, closed-page DDR3 1600MHz, 9-9-9 sub-timings
Compression Latencies (comp/decomp)	CPACK: 8/8 cycles gzip (LZSS): 64/32 cycles CABLE: 32/16 cycles

TABLE V
ENERGY SIMULATION PARAMETERS

	Static	Dynamic
L1	7.0mW	61.0pJ
L2	20.0mW	32.0pJ
LLC	169.7mW	92.1pJ
DRAM Buffer	22.0mW	149.4pJ
CABLE+LBE Comp		1000pJ
CABLE+LBE Decomp		200pJ

is always its worst-case search latency at 48 cycles total per transfer. Off-chip link parameters are modeled after Intel QPI and AMD HyperTransport. For single-threaded studies, each thread has a 1MB share of the LLC. Energy calculations though still use the 8MB cache model. For the throughput studies, to account for statistical multiplexing of bandwidth that a purely static bandwidth partitioning model does not capture, we split the threads into groups of eight and allow them to share bandwidth competitively within a group. The evaluated memory system is quad-channel (76.8GB/s total). Cache contention due to additional cache accesses to find similarity is also simulated for 128-bank eDRAM chips [42] at one chip per channel. L4 is four times as large as the LLC for off-chip compression studies. Last, compression ratios are

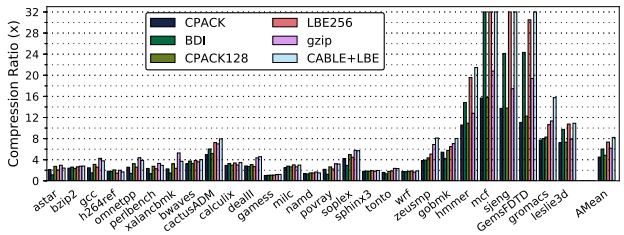


Fig. 12. Off-chip link compression (raw compression ratios)

represented as $uncompressed\_size \div compressed\_size$.

For memory-link compression, CABLE is configured with one "half-sized", 2-deep hash table at the buffer and one "full-sized", 2-deep hash table on-chip. For coherence-link compression, the hash tables are "quarter-sized", and the WMTs are "full-sized". Unless otherwise stated, data access count was set at six for the study in §VI-B, and sixteen in other studies. This parameter is studied in §VI-E.

The power model includes both static and dynamic power of the caches, eDRAM buffer, and DRAM, based on CACTI 5.3 [38] at *32nm*. DRAM access energy consumption is based on the Micron DDR3 power calculator [40], assuming two ranks, nine chips per rank, and 100% utilization. I/O link energy is estimated to be 25nJ per 64-byte, based on prior work that estimates it to be 50% of DRAM access energy [57] and at 30nJ [39]. Table II & Table V summarizes the power model.

We evaluated CABLE against three classes of algorithms: **non-dictionary** (CPACK [26], BDI [55]), **small dictionary** (CPACK128 and LBE256 [29]), and **big dictionary** (`gzip` [22]). To represent compression algorithms with small dictionary [20, 21, 33, 58], we modify CPACK and LBE to have 128-byte and 256-byte dictionaries respectively, with FIFO replacement policy. Larger sizes were not chosen because larger dictionaries can lead to *worse* performance [33]. `gzip` was evaluated with a 32KB dictionary (max configurable size). Power and timing parameters are in Table V & IV. Power and latency of `gzip` are estimations of IBM's ASIC LZ77 [25].

B. Single-program Compression

Off-chip Memory Link Compression: Fig. 11 & 12 show the relative and raw compression ratios of different compres-

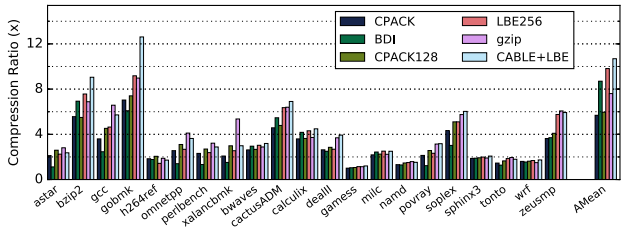


Fig. 13. Compression ratio for a 4-chip CMP. Only non-trivial workloads are shown, but averages are across all.

sion schemes when applied to the off-chip memory link [6]. Compared with `gzip`, CABLE fares favorably. Despite being simpler, CABLE only loses to `gzip` in a few benchmarks, while achieving higher compression in other workloads like *dealII*, *tonto*, *zeusmp*, and *gobmk*. Compared with LBE256, CPACK128, and other simpler schemes, CABLE consistently provides 25% to 100% more bandwidth. Overall, as shown in Fig. 12, CABLE achieves a $8.2\times$ compression ratio on average, increasing the effective bandwidth capacity by $7.2\times$. This result is 82% better than CPACK which only has a $4.5\times$ compression ratio. Relative to a system which already has CPACK applied, shown in Fig. 11, CABLE provides 46.9% better compressions.

In Fig. 12, note that some benchmarks (e.g., *mcf*) are easier to compress than others because their off-chip traffic is often dominated by zeroes or repeated values; we group these benchmarks to the right. For these workloads, both CABLE and other schemes do very well, consistently reaching $16\times$ and higher compression ratios, consistent with the results of prior work in link and memory compression [56, 58, 59]. As a side node, as mentioned in §III-E, CABLE does not send RemoteLIDs when the data can be compressed well without a dictionary.

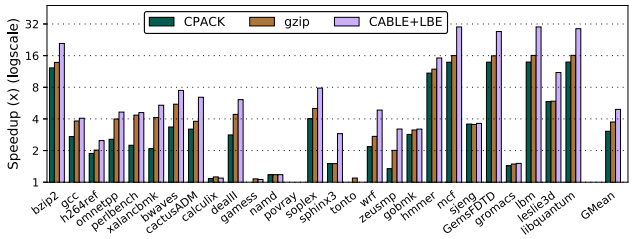
Multi-chip Coherence Link Compression: Fig. 13 compares the compression performance of link compression when applied to the coherent links between processors in a four-chip CMP system. As mentioned in §V-B, SPEC2006 single-threaded benchmarks are used to gauge the performance of a system with memory load balancing by interleaving pages across nodes. We observe similar trends in the memory-link compression results in the previous section, although compression ratios are slightly lower due to more dirty line transfers which are harder to compress. On average, CABLE+LBE achieves an average $10.6\times$ compression ratio, 86.4% better than CPACK.

Throughput Improvements: By saving more off-chip bandwidth, CABLE significantly increases throughput despite having higher compression latency. Fig. 14a shows that a 2048-thread system gets a throughput increase of 378% on average, and up to almost 3000% for certain workloads[7]. Memory-intensive workloads like *mcf* and *lbm* benefit the most while compute-intensive workloads like *povray* and *gobmk* generally do not benefit despite achieving high compression ratios. Fig. 14b shows the average throughput speedups at other thread counts. At 256 threads, bandwidth over-subscription is not significant and thus CABLE is only marginally better than CPACK and `gzip`. It is at a high thread count that the system can really benefit from CABLE.

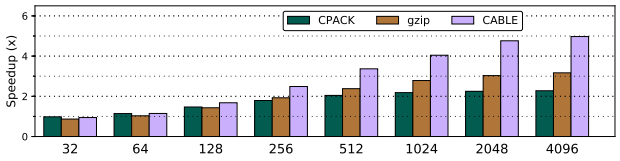
Such high thread count systems are rapidly emerging. Examples include the Intel Knight Landing with 288 threads and 72 cores [5], the Sunway TaihuLight processor with 256 cores per processor [6] which powers one of the fastest computers in the world [60], and the Epiphany-V 64-bit RISC processor with 1024 cores [7]. Besides, many GPUs have hundreds to thousands of threads in a single chip [61], and they are moving toward coherent off-chip links [62].

[6]Max effective compression ratio is to $32\times$ because the physical link width is 16-bit (or $1/32$ of a 64-byte payload).

[7]These are multi-threaded simulations with replicated workloads, but we configured compression to not be done across programs. For multiprogram studies with sharing effects, please see §VI-C



(a) At 2048 threads (log_2 scale)



(b) At other thread counts

Fig. 14. Throughput speedups with link compression.

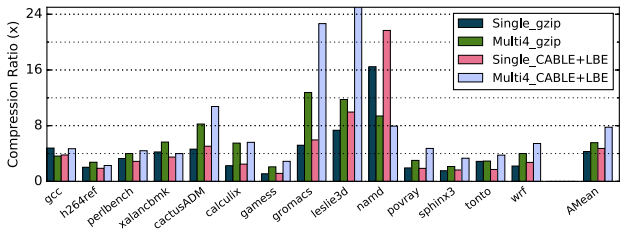


Fig. 15. Compression ratios when programs are run alone (`Single`), and replicated 4 times (`Multi4`).

C. Multiprogram Compression

We conducted two studies with multiprogram simulations: (1) cooperative workloads that, because of similar data structures, should result in *better* compression, and (2) unrelated, destructive workloads that can pollute the dictionary and result in *worse* compression. We examine the off-chip memory link use case. We only include `gzip` and CABLE in these studies as they have the most to gain or lose from compressing similar or dissimilar data streams. We also remove zero-dominant workloads from these results.

Cooperative Multiprogram: To simulate a cooperative multiprogram environment, we run four copies of the same program

TABLE VI
MULTIPROGRAM MIX; RANDOMLY CHOSEN.

MIX0	h264ref, soplex, hmmer, bzip2	**MIX1**	gcc, gobmk, gcc, soplex
MIX2	bzip2, lbm, gobmk, perlbench	**MIX3**	gcc, bzip2, tonto, cactusADM
MIX4	perlbench, wrf, gobmk, gcc	**MIX5**	omnetpp, bzip2, bzip2, gobmk
MIX6	gcc, tonto, gamess, cactusADM	**MIX7**	gcc, wrf, gcc, bzip2

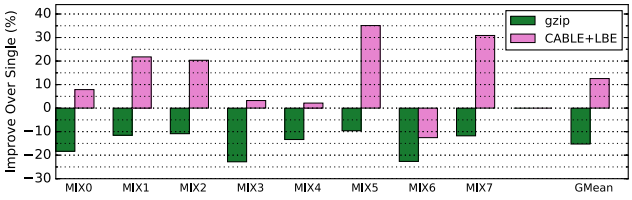


Fig. 16. Compression improvements over single-threaded compression when running program mixes in Table VI.

concurrently, same input, *SPECrate* style. These simulations approximate throughput-oriented workloads where multiple copies of the same program are run with slightly different inputs, or same inputs but with different configurations much like the architectural simulations of this paper.

Fig. 15 shows that CABLE benefits more from cooperative multiprogram than `gzip` does because CABLE has a much bigger data window to look for similarity. The plot also shows that compression can get worse, with the most obvious example being *namd* where both CABLE and `gzip` lose. With *gcc*, `gzip` also loses performance but not CABLE. One explanation is that even with identical programs running at the same rate, threads can desynchronize and execute dissimilar program phases.

Destructive Multiprogram: To measure the effect of dictionary pollution, we run mixes of programs as in Table VI. For each mix, we measure the compression ratios of each program separately, then normalize them to the single-threaded results presented in §VI-B. The trend in Fig. 16 shows that dictionary pollution is a severe problem with `gzip`. Having a fixed-size 32KB dictionary, `gzip` can perform as much as 25% worse under contention from multiple data streams. These results imply that `gzip` and other dictionary algorithms are not scalable, and will under-perform in manycore systems.

Meanwhile, CABLE and its data structures (hash table and WMT) are more scalable with increasing cache sizes and thread counts. CABLE not only maintains similar compression ratios to single-threaded compressions (*MIX3*, *MIX4*), but can improve up to 35% (*MIX5*).

D. Latency and Energy

Compression Latency: While single-threaded performance is not the focus of CABLE, it is still useful to understand the effect of increasing the compression latency. Fig. 17 shows the performance overheads of different schemes due to the compression latencies (as listed in Table IV). The overhead is proportional to the compression and decompression latency: at 48 cycles for CABLE, it is 5% on average and 10% at most. Multi-chip coherence compression (not shown) has similar degradation curves.

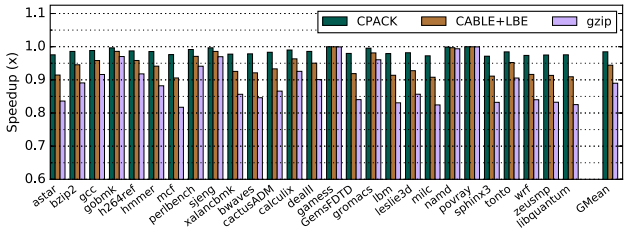


Fig. 17. Single-threaded performance degradation from link compression.

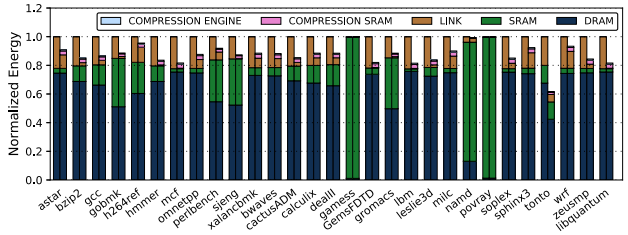


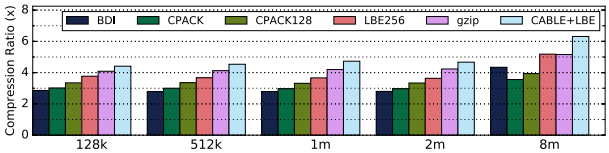
Fig. 18. Normalized memory subsystem energy breakdowns. The left bars indicate the uncompressed baseline, and the right bars CABLE+LBE.

To mitigate this deficiency, we tried a simple on/off compression control scheme where, when sampled with a 1ms period, compression is turned off when effective bandwidth usage is below 80% and turned on when it is over 90%. With this scheme, the single-threaded performance degradation of CABLE is effectively nullified, while only decreasing throughput by an average of 2.3% (not shown).

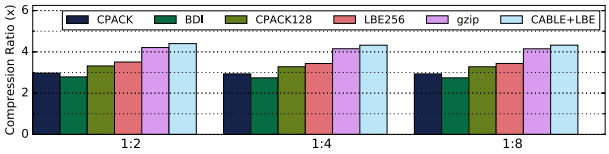
Energy Comparison: Data coding techniques–such as the 128b/130b coding [63] often used in high-speed serial links (SERDES) like PCI-Express–scramble transmitted data to guarantee edge density, error correction, and clock recovery. When data is scrambled, the energy used for data transmission is correlated less with the data patterns and more with the number of transactions. Fig. 18 compares the normalized energy costs (left bar) of off-chip links and other components to CABLE (right bar). The `SRAM` label indicates both static (leakage) and dynamic components, though the static energy of shared resources like the LLC is not included. CABLE compression energy is further broken down into the engine portion (`COMPRESSION ENGINE`) and the eDRAM search (`COMPRESSION SRAM`).

Overall, CABLE saves link energy significantly. Except for compute-intensive benchmarks, link energy accounts for roughly 20% of memory subsystem energy. Compression energies are small compared to the link energy it saves, leading to 16% overall energy savings. This result is promising for future energy-efficient computing systems where link energy is expected to grow to 13% of total system energy [57].

Bit Toggle Reduction: If the memory link does not scramble data, bit toggling rate can be a useful metric as high bit toggles result in higher energy consumption and lower reliability [64]. We find that when applied to 16-bit links, CABLE reduces bit toggles by 30.2% on average, 16.9% less than CPACK (not shown).



(a) Across LLC sizes



(b) Across LLC to L4 ratios. The LLC is fixed at 1MB.

Fig. 19. Memory link compression across cache sizes.

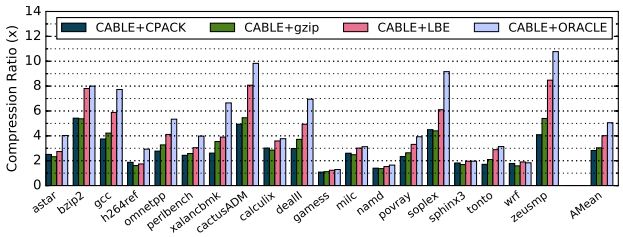


Fig. 20. Compression performance of CABLE using different engines.

E. Sensitivity Studies

These studies exclude zero-dominant benchmarks.

Cache Sizes: Fig. 19a varies the LLC allocation for each program thread from 128KB to 8MB while keeping a 1:2 ratio between LLC and L4. We see that compression ratios are mostly static across cache sizes. For `gzip` and CABLE, compression ratios get slightly better with increasing cache sizes because there are less hard to compress data patterns as a result of fewer spills and fills to memory.

L4 Ratio: Changing LLC-to-L4 ratio from 1:2 to 1:8 (Fig. 19b) while keeping the LLC size constant, we observed that while the compression ratios of individual workloads fluctuate due to the working set size effect, the averages vary within 1% across configurations. This result is expected because the amount of shared data accessible to CABLE depends on the smaller of the two caches (i.e., the LLC), which is constant in this study.

NUMA Count: For coherence link compression, varying NUMA node count between 2 to 8 processors (not shown), we also found compression ratios largely unaffected.

Other Compression Engines: Although we found that CABLE has the best performance with LBE, we show in Fig. 20 CABLE's performance when paired with CPACK128, `gzip`, and ORACLE. Generally, LBE is better than `gzip`, which is better than CPACK128. Inspecting more closely, we make the following insights. First, pointer overhead matters. One significant difference between LBE and CPACK is that LBE can copy large aligned data blocks with lower overheads. Second, there is room to improve, as demonstrated with CABLE+ORACLE. CABLE+ORACLE has the same reference cache lines as the other schemes but can compress any data patterns such as byte shifts and unaligned duplicates, resulting in significantly higher compression ratios.

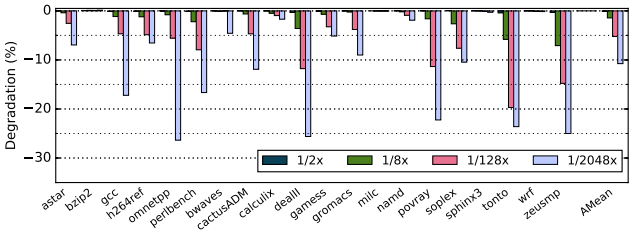


Fig. 21. Compression degradation from decreasing the hash table's size, compared to a 2× hash table.

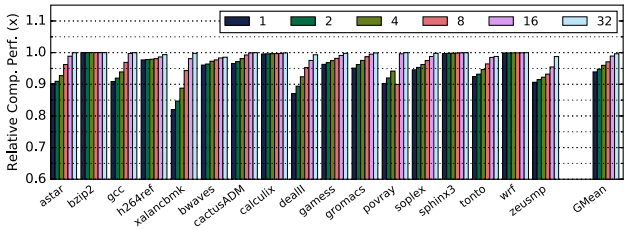


Fig. 22. Compression with other data access counts, relative to 64 accesses.

Hash table size: Fig. 21 varies the hash table size between 1/2048× to 2×, where 1× (not shown) denotes a "full-sized" table. We use the 2× table as the baseline. We see graceful degradation of performance, even at the extreme end ($1/2048\times$). For most benchmarks, the 1/8× table size presents an advantageous trade-off between area and performance, with less than 7% performance loss in the worst case.

Data access count: In the search stage (§III-C), CABLE selects the top six references after the pre-ranking step to access the data array with. Fig. 22 varies this threshold and plots the compression ratio relative to 64 accesses. We find lower access counts quite resilient, with the one-access case within 80%, at worst, of the 64-access case. The main reason fewer accesses can still have good performance is that when the references are near-duplicates of the requested line, the pre-ranking step is effective at filtering out hash-collided LineIDs.

Link width: Generally, link compression is most effective with small-width interconnects. As shown in Fig. 23, at larger link widths, effective bandwidth degrades because more bits are wasted on left-over and small payloads. That said, there are many examples of narrow interconnects: Intel QPI is 20-bit, AMD Zen Infinity InterSocket is 16-bit, and a PCI-E ×8 lane is 8-bit wide. A physical link can be bifurcated further into smaller logical channels, like with HBM where the 1024-bit bus is divided into eight 128-bit channels. Alternatively, the transport protocol can pack multiple transactions in one transfer by adding a 6-bit value specifying the length in bytes of each compressed data, as shown with `64-bit Packed`.

VII. Related Work

Besides the dictionary approach used in CABLE and `gzip`, non-dictionary compression is often used because it is fast. These algorithms are evaluated for cache compression [30, 56,

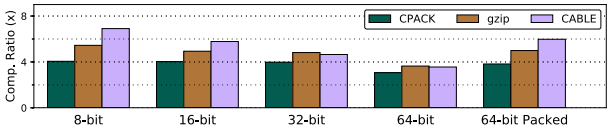


Fig. 23. Compression at other link widths (all workloads).

65]–[71], main memory [59, 72]–[75], link [17, 18, 20], and other components such as NOC and register [18, 19, 55, 76, 77]. In this work, we evaluated the performance of three representative algorithms: BDI [55], CPACK [26], and LBE [29].

In contrast, dictionary-based compression often performs better due to significant redundancies between cache lines, a phenomenon observed in cache compression studies [27, 29, 31, 32]. It has been applied to main memory [32, 45, 78], link compression [21, 58], and cache [27, 29]–[32, 79]. Inter-block similarity encoding has also been explored to save DRAM interface energy [80]. *gzip/LZ77*, the gold standard for software compression algorithms, is evaluated, and CABLE is shown to have comparable performance. This is encouraging because gzip is well-studied in domains including HPC which has been shown to have $1.8\times$ to $160\times$ compression ratios [81]. We also evaluated LZMA [34] which can be configured with up to 4GB of dictionary storage but we found its performance to be subpar due to inefficient output flushing.

Last-level cache (LLC) compression is promising direction to increase the effective cache capacity and decrease miss rate to memory [27, 29]–[32, 56, 65]–[71, 79]. Compared to *link* compression, *cache* compression is harder to design because of issues such as cache fragmentation. As a result, compression ratios in cache compression are typically much lower, often capped at $2\times$ or $4\times$ due to limited tags. Otherwise, cache and link compression are orthogonal and can work in tandem to maximize off-chip bandwidth.

Data deduplication has been studied in contexts such as storage [82], replication [83], and network systems [84]. More closely related, data deduplication has been applied to the last-level cache [28], which like CABLE also utilizes hashing to detect duplicated lines. The main differences are (a) CABLE is designed to compress memory links, not memory caches, and (b) CABLE detects similarities between lines at the sub-block granularity using 32-bit signatures, whereas the mentioned scheme can only deduplicate exact cache lines. Doppelgänger Cache [85] exploits data approximation to deduplicate similar cache lines. Given a cache line containing a sequence of values, it uses a few hash functions to get value ranges and averages to determine proximity with other cache lines. It also depends on programmers to annotate memory regions to be approximated. Nevertheless, approximation can be employed to increase the efficacy of link compression.

A recent retrospective review of compression methodology shows that simulating a single memory trace often does not capture the varying compression behavior of the whole workload [86]. We agree with this assertion, and in this work we simulate a set of publicly available SPECK2k6 SimPoint traces [53, 54] to avoid reporting unrepresentative results.

In the context of cloud computing, CABLE provides a greater degree of bandwidth control beyond cache [87] and simple bandwidth throttling [88] that both cloud providers and tenants alike can exploit to optimize costs and profits [87, 89, 90].

VIII. Conclusion

Evaluating with the SPEC2006 benchmark suite, we find CABLE markedly improves compression ratios over prior work in link compression: it increases effective memory bandwidth by $7.2\times$ and throughput by $3.78\times$ on average for a 2048-threaded system. We also demonstrated CABLE's resiliency to dictionary pollution, and framework flexibility across two different off-chip links. With off-chip bandwidth becoming scarce, cache-based link compression presents an excellent solution to extract the last bits of bandwidth.

Acknowledgments

This material is based on research sponsored by the NSF under Grants No. CNS-1823222 and CCF-1438980, AFOSR under Grant No. FA9550-14-1-0148, Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement No. FA8650-18-2-7846 and FA8650-18-2-7852. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA), the NSF, AFOSR, DARPA, or the U.S. Government.

References

[1] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *Solid-State Circuits, IEEE Journal of*, vol. 43, 2008.

[2] J. S. Kim, M. B. Taylor, J. Miller, and D. Wentzlaff, "Energy characterization of a tiled architecture processor with on-chip networks," in *Proceedings of the 2003 international symposium on Low power electronics and design*. ACM, 2003.

[3] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the Tile Processor," *IEEE Micro*, vol. 27, Sep. 2007.

[4] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C. C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "Tile64 - processor: A 64-core soc with mesh interconnect," in *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, Feb 2008.

[5] A. Sodani, "Knights landing (knl): 2nd generation intel® xeon phi processor," in *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE, 2015.

[6] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, W. Zhao, X. Yin, C. Hou, C. Zhang, W. Ge, J. Zhang, Y. Wang, C. Zhou, and G. Yang, "The sunway taihulight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11432-016-5588-7

[7] A. Olofsson, "Epiphany-v: A 1024 processor 64-bit risc system-on-chip," *arXiv preprint arXiv:1610.01832*, 2016.

[8] J. Feehrer, S. Jairath, P. Loewenstein, R. Sivaramakrishnan, D. Smentek, S. Turullols, and A. Vahidsafa, "The oracle sparc t5 16-core processor scales to eight sockets," *IEEE Micro*, 2013.

[9] W. J. Starke, J. Stuecheli, D. Daly, J. Dodson, F. Auernhammer, P. Sagmeister, G. Guthrie, C. Marino, M. Siegel, and B. Blaner, "The cache and memory subsystems of the ibm power8 processor," *IBM Journal of Research and Development*, vol. 59, 2015.

[10] 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 44th Annual International Symposium on Computer Architecture*. ACM, 2017.

[11] D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek, "Intel® quickpath interconnect architectural features supporting scalable system architectures," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*. IEEE, 2010.

[12] K. Lepak, G. Talbot, S. White, N. Beck, and S. Naffziger, "The next generation amd enterprise server product architecture," *IEEE Hot Chips*, vol. 29, 2017.

[13] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin *et al.*, "25.2 a 1.2 v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014.

[14] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: challenges in and avenues for cmp scaling," in *Proceedings of the International Symposium on Computer Architecture*, 2009.

[15] A. Kagi, J. Goodman, and D. Burger, "Memory bandwidth limitations of future microprocessors," in *Computer Architecture, 1996 23rd Annual International Symposium on*, May 1996.

[16] J. Huh, D. Burger, and S. W. Keckler, "Exploring the design space of future cmps," in *Parallel Architectures and Compilation Techniques, 2001. Proceedings. 2001 International Conference on*. IEEE, 2001.

[17] A. R. Alameldeen and D. A. Wood, "Interactions between compression and prefetching in chip multiprocessors," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, Feb 2007.

[18] V. Sathish, M. J. Schulte, and N. S. Kim, "Lossless and lossy memory I/O link compression for improving performance of GPGPU workloads," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 2012.

[19] J. Zhan, M. Poremba, Y. Xu, and Y. Xie, "Noδ: Leveraging delta compression for end-to-end memory access in noc based multicores," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 2014.

[20] C. Alverti, G. Goumas, K. Nikas, A. Arelakis, N. Koziris, and P. Stenström, "Memory link compression to speedup scientific workloads," in *8th Workshop on Programmability Issues for Heterogeneous Multicores*, 2015.

[21] B. S. An, M. Lee, K. H. Yum, and E. J. Kim, "Efficient data packet compression for cache coherent multiprocessor systems," in *Data Compression Conference (DCC), 2012*. IEEE, 2012.

[22] L. P. Deutsch, "GZIP file format specification version 4.3," 1996.

[23] L. Villa, M. Zhang, and K. Asanović, "Dynamic zero compression for cache energy reduction," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*. ACM, 2000.

[24] J. Dusser, T. Piquet, and A. Seznec, "Zero-content augmented caches," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009.

[25] 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)," *IBM Journal of Research and Development*, vol. 45, 2001.

[26] X. Chen, L. Yang, R. P. Dick, L. Shang, and H. Lekatsas, "C-pack: A high-performance microprocessor cache compression algorithm," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, 2010.

[27] A. Arelakis and P. Stenstrom, "SC2: A statistical compression cache scheme," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*. IEEE, 2014.

[28] Y. Tian, S. M. Khan, D. A. Jiménez, and G. H. Loh, "Last-level cache deduplication," in *Proceedings of the 28th ACM international conference on Supercomputing*. ACM, 2014.

[29] T. M. Nguyen and D. Wentzlaff, "MORC: A manycore-oriented compressed cache," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2830772.2830828

[30] A. Arelakis, F. Dahlgren, and P. Stenstrom, "Hycomp: a hybrid cache compression method for selection of data-type-specific compression methods," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015.

[31] B. Panda and A. Seznec, "Dictionary sharing: An efficient cache compression scheme for compressed caches," in *49th Annual IEEE/ACM International Symposium on Microarchitecture, 2016*, 2016.

[32] R. Kanakagiri, B. Panda, and M. Mutyam, "Mbzip: Multiblock data compression," *ACM Trans. Archit. Code Optim.*, vol. 14, Dec. 2017. [Online]. Available: http://doi.acm.org/10.1145/3151033

[33] A. Arelakis and P. Stenstrom, "A case for a value-aware cache," *Computer Architecture Letters*, vol. 13, 2014.

[34] I. Pavlov, "LZMA SDK," *www.7-zip.org/sdk.html*, 2007.

[35] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar *et al.*, "Haswell: The fourth-generation intel core processor," *IEEE Micro*, vol. 34, 2014.

[36] M. Shah, J. Barreh, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Saha, D. Sheahan, L. Spracklen, and A. Wynn, "Ultrasparc t2: A highly-treaded, power-efficient, sparc soc," in *Solid-State Circuits Conference, 2007. ASSCC '07. IEEE Asian*, Nov 2007.

[37] I. Parulkar, A. Wood, J. C. Hoe, B. Falsafi, S. V. Adve, J. Torrellas, and S. Mitra, "Opensparc: An open platform for hardware reliability experimentation," in *Fourth Workshop on Silicon Errors in Logic-System Effects (SELSE)*. Citeseer, 2008.

[38] S. Thoziyoor, N. Muralimanohar, and N. P. Jouppi, "CACTI 5.0," *HP Laboratories, Technical Report*, 2007.

[39] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "Gpus and the future of parallel computing," *IEEE Micro*, vol. 31, 2011.

[40] Micron Technology, "DDR3 System-Power Calculator," *www.micron.com/support/power-calc*.

[41] G. Fredeman, D. W. Plass, A. Mathews, J. Viraraghavan, K. Reyer, T. J. Knips, T. Miller, E. L. Gerhard, D. Kannambadi, C. Paone *et al.*, "A 14 nm 1.1 mb embedded dram macro with 1 ns access," *IEEE Journal of Solid-State Circuits*, vol. 51, 2016.

[42] F. Hamzaoglu, U. Arslan, N. Bisnik, S. Ghosh, M. B. Lal, N. Lindert, M. Meterelliyoz, R. B. Osborne, J. Park, S. Tomishima *et al.*, "13.1 a 1gb 2ghz embedded dram in 22nm tri-gate cmos technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014.

[43] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton, "Haswell: The fourth-generation intel core processor," *IEEE Micro*, vol. 34, 2014.

[44] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang *et al.*, "OpenPiton: An open source manycore research framework," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2. ACM, 2016.

[45] B. Abali, H. Franke, D. E. Poff, R. A. Saccone, C. O. Schulz, L. M. Herger, and T. B. Smith, "Memory expansion technology (mxt): Software support and performance," *IBM Journal of Research and Development*, vol. 45, March 2001.

[46] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of computer and system sciences*, vol. 18, 1979.

[47] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Transactions on Computers*, vol. 46, Dec 1997.

[48] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, and F. Franchetti, "A 3d-stacked logic-in-memory accelerator for application-specific data intensive computing," in *3D Systems Integration Conference (3DIC), 2013 IEEE International*. IEEE, 2013.

[49] L. Durant, O. Giroux, and M. Harris. (2017) Inside volta: The worlds most advanced data center gpu. [Online]. Available: https://devblogs.nvidia.com/parallelforall/inside-volta/

[50] D. Molka, D. Hackenberg, R. Schöne, and W. E. Nagel, "Cache coherence protocol and memory performance of the intel haswell-ep architecture," in *Parallel Processing (ICPP), 2015 44th International Conference on*. IEEE, 2015.

[51] Y. Fu and D. Wentzlaff, "PriME: A parallel and distributed simulator for thousand-core chips," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 2014.

[52] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using simpoint for accurate and efficient simulation," in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '03. New York, NY, USA: ACM, 2003. [Online]. Available: http://doi.acm.org/10.1145/781027.781076

[53] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulations," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2011.

[54] H. Patil and T. E. Carlson, "Pinballs: Portable and Shareable User-level Checkpoints for Reproducible Analysis and Simulation," in *Proceedings of the Workshop on Reproducible Research Methodologies (REPRODUCE)*, 2014.

[55] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: practical data compression for on-chip caches," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 2012.

[56] S. Sardashti and D. A. Wood, "Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013.

[57] A. Gara, "Energy efficiency challenges for exascale computing," in *ACM/IEEE Conference on Supercomputing: Workshop on Power Efficiency and the Path to Exascale Computing*, 2008.

[58] M. Thuresson, L. Spracklen, and P. Stenstrom, "Memory-link compression schemes: A value locality perspective," *Computers, IEEE Transactions on*, vol. 57, 2008.

[59] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly compressed pages: a low-complexity, low-latency main memory compression framework," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013.

[60] J. Dongarra, M. Meuer, H. Simon, and E. Strohmaier, "Top500 supercomputer ranking," 2017.

[61] M. Harris. (2016) Inside pascal: Nvidia's newest computing platform. [Online]. Available: https://devblogs.nvidia.com/parallelforall/inside-pascal/

[62] M. Harris, "How nvlink will enable faster, easier multi-gpu computing," https://devblogs.nvidia.com/parallelforall/how-nvlink-will-enable-faster-easier-multi-gpu-computing/, 2014.

[63] R. Walker and R. Dugan, "64b/66b low-overhead coding proposal for serial links," *IEEE*, vol. 802, 2000.

[64] G. Pekhimenko, E. Bolotin, M. O'Connor, O. Mutlu, T. C. Mowry, and S. Keckler, "Toggle-aware bandwidth compression for gpus."

[65] A. R. Alameldeen and D. Wood, "Adaptive cache compression for high-performance processors," in *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*. IEEE, 2004.

[66] S. Kim, J. Lee, J. Kim, and S. Hong, "Residue cache: a low-energy low-area L2 cache architecture via compression and partial hits," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011.

[67] S. Sardashti, A. Seznec, and D. Wood, "Skewed compressed caches," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*. IEEE, 2014.

[68] G. Pekhimenko, T. Huberty, R. Cai, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Exploiting compressed block size as an indicator of future reuse," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015.

[69] S. Sardashti, A. Seznec, and D. A. Wood, "Yet another compressed cache: A low-cost yet effective compressed cache," *ACM Trans. Archit. Code Optim.*, vol. 13, Sep. 2016. [Online]. Available: http://doi.acm.org/10.1145/2976740

[70] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep*, vol. 1500, 2004.

[71] V. Young, P. J. Nair, and M. K. Qureshi, "Dice: Compressing dram caches for bandwidth and capacity," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080243

[72] M. Ekman and P. Stenstrom, "A robust main-memory compression scheme," in *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2. IEEE Computer Society, 2005.

[73] E. G. Hallnor and S. K. Reinhardt, "A unified compressed memory hierarchy," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*. IEEE, 2005.

[74] P. M. Palangappa and K. Mohanram, "Compex++: Compression-expansion coding for energy, latency, and lifetime improvements in mlc/tlc nvms," *ACM Trans. Archit. Code Optim.*, vol. 14, Apr. 2017. [Online]. Available: http://doi.acm.org/10.1145/3050440

[75] A. Arunkumar, S. Y. Lee, V. Soundararajan, and C. J. Wu, "Latte-cc: Latency tolerance aware adaptive cache compression management for energy efficient gpus," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018.

[76] N. Vijaykumar, G. Pekhimenko, A. Jog, A. Bhowmick, R. Ausavarungnirun, C. Das, M. Kandemir, T. C. Mowry, and O. Mutlu, "A case for core-assisted bottleneck acceleration in gpus: enabling flexible data compression with assist warps," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, 2015.

[77] S. Lee, K. Kim, G. Koo, H. Jeon, W. W. Ro, and M. Annavaram, "Warped-compression: enabling power efficient gpus through register compression," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, 2015.

[78] M. Kjelso, M. Gooch, and S. Jones, "Design and performance of a main memory hardware data compressor," in *EUROMICRO 96. Beyond 2000: Hardware and Software Design Strategies., Proceedings of the 22nd EUROMICRO Conference*. IEEE, 1996.

[79] Y. Zhang, J. Yang, and R. Gupta, "Frequent value locality and value-centric data cache design," in *ACM SIGOPS Operating Systems Review*, vol. 34, no. 5. ACM, 2000.

[80] H. Seol, W. Shin, J. Jang, J. Choi, J. Suh, and L. S. Kim, "Energy efficient data encoding in dram channels exploiting data value similarity," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016.

[81] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016.

[82] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system." in *Fast*, vol. 8, 2008.

[83] N. Jain, M. Dahlin, and R. Tewari, "Taper: Tiered approach for eliminating redundancy in replica synchronization," in *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies-Volume 4*. USENIX Association, 2005.

[84] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5. ACM, 2001.

[85] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelgänger: a cache for approximate computing," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015.

[86] S. Sardashti and D. A. Wood, "Could compression be of general use? evaluating memory compression across domains," *ACM Trans. Archit. Code Optim.*, vol. 14, Dec. 2017. [Online]. Available: http://doi.acm.org/10.1145/3138805

[87] Y. Zhou, H. Hoffmann, and D. Wentzlaff, "CASH: Supporting IaaS customers with a sub-core configurable architecture," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016.

[88] Y. Zhou and D. Wentzlaff, "MITTS: Memory inter-arrival time traffic shaping," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016. [Online]. Available: https://doi.org/10.1109/ISCA.2016.53

[89] Y. Zhou and D. Wentzlaff, "The sharing architecture: Sub-core configurability for IaaS clouds," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2541940.2541950

[90] M. Shahrad, C. Klein, L. Zheng, M. Chiang, E. Elmroth, and D. Wentzlaff, "Incentivizing self-capping to increase cloud utilization," in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3127479.3128611