CompressPoints: An Evaluation Methodology for Compressed Memory Systems

Esha Choukse*, Mattan Erez, and Alaa Alameldeen

Abstract—Current memory technology has hit a wall trying to scale to meet the increasing demands of modern client and datacenter systems. Data compression is a promising solution to this problem. Several compressed memory systems have been proposed in the past years [1], [2], [3], [4]. Unfortunately, a reasonable methodology to evaluate these systems is missing. In this paper, we identify the challenges for evaluating main memory compression. We propose an effective methodology to evaluate a compressed memory system by proposing mechanisms to: (i) incorporate correct virtual address translation, (ii) choose a region in the application that is representative of the compression ratio, in addition to regular metrics like IPC and cache hit rates, and (iii) choose a representative region for multi-core workloads, bringing down the correlation error from 12.8 to 3.8 percent.

Index Terms—Compression, memory, DRAM, evaluation, methodology, representative regions, compressed memory, translation, multi-core, workloads

1 INTRODUCTION

Research in computer systems often involves slow, cycle-based simulations, since prototyping is a time and resource-intensive process. In order to draw conclusions from these simulations, they need to reasonably represent real systems. However, we found that current research practices are not suitable for simulating a compressed memory system. For instance, for most simulation strategies, the focus is on the control flow, and seldom on the data. A compressed memory system’s performance, however, highly depends on the data it is operating on.

Fig. 1 shows the schematic of a typical compressed main memory system. The compressibility of the data depends on its redundancy. A higher redundancy implies a higher compression ratio for the data and results in lower memory storage in a compressed memory system. In general, the data used by an application is initially highly compressible if it is zero-initialized, and as the run proceeds, the compressibility tends to monotonically decrease. Therefore, the compression ratio, as well as the memory usage of an application, do not tend to display phase behavior. This is unlike IPC and other performance-related metrics. Hence, to get a good idea of the dependency of an application on the memory capacity, it is important to run a benchmark to completion. This requires runs on a real system, since simulations are too slow to run the complete benchmark. This is the first kind of evaluation.

Compressed main memory adds overhead to a system, due to the additional address translation, compression and decompression latencies, and compressed data movement. Due to writebacks from the LLC to memory, the data can change over time, leading to changes in the compressibility and memory required to store it. This causes compressed data movement in the form of page overflows and underflows in the compressed address space. The amount of compressed data movement in the system depends on the compression algorithm and other design decisions. The evaluation of these performance overheads requires a cycle-level simulation—which needs to be done on a small, representative region of the benchmark. This is the second kind of evaluation. Combining the results from these two kinds of evaluation requires the representative region to have a high correlation with the metrics of the complete run. It is also important to evaluate a compressed memory system with multi-core workloads that are a mix of compressible and incompressible applications to analyse their interplay.

We address these challenges and present mechanisms to make the evaluation of a compressed memory system reasonably representative. First, we present CompressPoints, an extension of the SimPoints [5] methodology to choose a representative region of a benchmark in a compressed memory system. We then present a mechanism to avoid being misled due to inaccurate virtual address translation by using the actual translation mechanism of the Linux OS. This is imperative for accurately calculating the compressibility of an application’s memory at any given time. Finally, we provide a methodology to choose representative regions for multi-core workloads to maximize the correlation with real hardware runs.

2 COMRESSPOINb

Hardware designers heavily rely on detailed cycle-based simulators to evaluate novel features and changes to existing microprocessors. However, these cycle-level simulators tend to be very slow, taking weeks to simulate complete runs of applications. The most popular solution to this challenge is to simulate only a set of small representative regions of the benchmark. The most widely-adopted mechanism for choosing representative regions is SimPoints [5].

2.1 SimPoints and Data-Representativeness

SimPoints relies on the assumption that most of the important metrics of an application are a function of the code executed, and therefore uses basic block vectors (BBVs) to characterize different regions of execution. The complete execution of an application is divided into intervals based on the number of retired instructions. The code-based of the application is divided into basic blocks and the BBVs track the number of times each basic block is executed in each interval. SimPoints uses a random projection to decrease the dimension of the BBVs from thousands to 15, before applying the well-known clustering algorithm K-Means. Note that there is no concept of elapsed time in the BBVs. The whole mechanism is hence oblivious to the simulated hardware, which makes it easy to use.

However, we note that the BBVs fail to capture the characteristics of the data being used. In the evaluation of compression, the redundancy/compressibility of the data is a very important metric that directly impacts the resources available in the system. In our evaluation, we add 4 metrics of compression to the projected space, increasing the dimension of the input to the K-means to 19. The 4 metrics added are: Compression ratio, number of page overflows and number of page underflows and total memory usage per interval. These augmented vectors are then used for clustering the execution intervals of the application, leading to better representative regions. We call these CompressPoints. Fig. 2 shows the compression ratio of GemsFDTD and astar over time, and the representative regions selected by SimPoints and CompressPoints. Figs. 2 and 3 exhibit the higher correlation of compression ratio with CompressPoints.

The first three of the new metrics we use are dependent on the system design. While for a design space search, the metrics from all the candidate designs should be included separately, we found that the CompressPoints found based on one compression algorithm represent other algorithms within an error margin of 4 percent.
2.2 Importance of Truly Representative Regions for Compressed Memory Evaluation

For some research purposes, it can be acceptable for an evaluation to not use regions that accurately represent a benchmark. However, for a complete evaluation of a compressed memory system, the results of two kinds of experiments must be combined: (i) A complete run of the application with different constraints on available memory, in order to quantify the benefits of memory savings on the performance of the application. This experiment requires a complete run, since the compressibility of the data in an application is generally a slowly and monotonically changing value, rather than changing in phases like IPC and cache hit rates. (ii) A cycle-based simulation of a region of the application to evaluate the performance impact of the compressed system on the processor pipeline. In order to be able to combine the two results, it is important that the regions used for simulation accurately represent the complete application.

Most systems researchers today use a single SimPoint of 1 Billion instructions. A single SimPoint, however, is not very representative of the complete benchmark, hence, we use 10 SimPoints, instead of 1, for better correlation. The length of the SimPoint is important too. For instance, we found that increasing the number of SimPoints with a 1 Billion instruction interval does not lead to better correlation. This is because the phases in the application are shorter than this size. We found 200M-instruction SimPoints to be closely correlated to real behavior in terms of IPC (1.6 percent average error as compared to 13 percent from 1B-instruction SimPoints).

3 Multi-Core Workloads

Multi-core performance evaluation often requires SimPoints for multi-programmed workloads. Several computer architecture papers use the SimPoints captured for single core benchmarks and reuse those to represent workloads. However, this does not take into account the interplay between the complete benchmarks running together. Instead, each sampled region is used as a distinct benchmark, and does not represent the full original application.

In order to replicate the notion of full benchmarks running together, we concatenate the basic-block vectors from the individual applications, interval-by-interval, to get workload-BBVs, and run KMeans on these new BBVs. SimPoints collected this way exhibit much better representativeness of the benchmarks under contention. For correlation study of the multi-core workloads, we reran the individual applications till the longest application finishes, so that all benchmarks were always under contention.

For identifying the CompressPoints of the multi-core workloads, we append their concatenated BBVs with weighted average of compression metrics of the individual benchmarks, weighted by the memory usage of the benchmark during the interval.

4 Methodology

**Exact Address Translation.** One of the common memory-saving techniques used in operating systems is to map all the zero virtual pages to the same physical frame and mark it for copy-on-write. Since zero pages are highly compressible, for accurate estimation...
of a benchmark’s compressibility, it is imperative to have correct and unique physical addresses. If uniqueness is not maintained, it can have a two-fold impact on the results. First, if the count of zero pages is overestimated, the compression ratios can be misleadingly high. Second, a compressed memory system is generally optimized for zero-page access by having the required information encoded in the metadata. Using virtual addresses multiplies the space occupied by zero pages, leading to cache misses and main memory accesses. These main memory accesses to zero pages convert into inflated bandwidth savings with compression.

To show the effect of inaccurate virtual to physical address translation, let us consider the benchmark GemsFDTD of the SPEC2006 suite. Previous work [2] reports the compression ratio during the 1-billion instruction SimPoint of GemsFDTD to be \(\frac{C_2}{C_24} = \frac{200}{2}\). However, we observe that the compression ratio of the physical memory used during the SimPoint is only \(\frac{13}{2}\), despite the virtual address space seemingly being \(\sim 200 \times\) compressible.

To solve this problem, we use Linux’s proc files. We use the pagemap to get the physical addresses and the kpageflags file to check whether a page is resident in main memory. Furthermore, we make sure that every physical address is used uniquely while taking the memory snapshot for compression ratio calculations. A recently published paper from Sardashti and Wood [6] uses a similar approach for translation of addresses. CompressPoints can also be used in combination with their described tool to get better representative regions.

**Metric-Collection Methodology.** The compression metrics are collected over a complete run of each application. We use pintool [7] to periodically pause the application. We then use the Linux proc files (pagemap, kpageflags) to get the resident pages of the application. For taking snapshots of the memory being used by the benchmark, we use these physical addresses with the Linux *Disk Dump (dd)* tool. The compression ratio, along with the other proposed metrics is then calculated. To track the page overflows and underflows, we maintain a dictionary of the current compression ratio per page.

Not all the pages of a benchmark are touched or written to during an interval. Hence, processing all the physical frames for every interval can significantly slow down data collection for large-footprint benchmarks. We use the `clear_refs` Linux feature to track the active pages, of each interval. Aggregate metrics are computed for all pages though, including the inactive ones.

**Correlation Methodology.** In order to correlate the behavior of the representative region with the complete application, we run each application to completion, locked to a single core of an Intel Xeon E5 v3 CPU. The hardware counters of that core are read and stored every 250 ms. We program the performance monitor to read the counters for instructions retired, IPC, and L1/L2/L3 hit rates. We then compare the counter readings from just the SimPoints/CompressPoints to the readings averaged over the complete run of the benchmark.

### 5 Evaluation

Fig. 4 shows the relative error percentage of the representative compression ratio (using BPC Compression [8]) for SimPoints and CompressPoints. The error is much higher when using SimPoints for
Another important observation is that the error in compression ratio does not always decrease when we use 10 SimPoints instead of 1. For instance, in the gcc benchmark, the error goes up from 3 to 12.3 percent when the number of SimPoints is increased. There is a similar trend in bzip2, cactusADM, and sjeng. If SimPoints were representative of the compression ratio, this divergence would not be common. Overall, using 10 CompressPoints, instead of 10 SimPoints, brings down the average error in representative compression ratio from 5.8 to 0.6 percent, while bringing down the maximum error from 35 to 3 percent.

It is important that including these compression metrics does not decrease the representativeness of traditional metrics, like IPC and cache hit rates. We therefore present the correlation studies for IPC and L1 hit rate in Figs. 5 and 6. Although we see a spike in the error when only 1 CompressPoint is used, using 10 CompressPoints performs well. In fact, the average IPC error actually decreases as compared to SimPoints, for mcf, milc, tonto, and xalancbmk. The inclusion of main memory usage in the CompressPoint vectors brings this benefit for these memory-intensive benchmarks. We see similar results for them even when only memory usage is added to the feature vectors. Overall, using 10 CompressPoints, instead of 10 SimPoints, maintains the average error in IPC correlation at 0.2 percent, while bringing down the maximum error from 13.2 to 9.29 percent.

Fig. 6 shows the error percentage of the representative L1 cache hit rate for SimPoints and CompressPoints. The results are mixed, as we see slight increase in errors for calculix and gcc, and decreases in errors for hmmer, h264ref, and tonto. Overall, the average error decreases from 16.4 to 10.6 percent.

Fig. 7 shows the correlation of IPC when using the CompressPoints collected per benchmark, as compared to the workload CompressPoints that are calculated as described in Section 3. We construct 8 mixes of benchmarks, based on their memory intensity and compression ratio, as shown in Table 1. The IPC for the workload is calculated as the geometric mean of the IPC of the individual benchmarks. Our methodology of finding the SimPoints and CompressPoints using the workload as a whole shows much higher correlation than the per-benchmark SimPoints, across all mixes. Workload SimPoints achieve an average error of 2.8 percent as compared to the 8 percent error from per-benchmark SimPoints. Note that this methodology is applicable to any multi-core system setup and not just for a compressed memory evaluation. Workload CompressPoints exhibit an average IPC error of just 2.5 percent.

6 CONCLUSION
In this paper, we strive to bring forth good practices for a reasonable evaluation of compressed memory systems. We show that using these methodologies results in higher correlation in compression ratio of a compressed memory system, in addition to the regular metrics of IPC and cache hit rate. The benefits of the methodology to get the exact address translation from an operating system are not limited to a compressed memory system, and should be used for any DRAM studies. Similarly, the multi-programmed-workload SimPoints methodology is also applicable to any multi-core evaluation.

REFERENCES

TABLE 1
Workloads for Multi-Core Evaluation

| Mix1   | mcf, GemsFDTD, libquantum, soplex |
| Mix2   | milc, asar, gomembench, tonto     |
| Mix3   | zeusmp, lbm, leslie3d, hmmer       |
| Mix4   | sjeng, omnetpp, gcc, namd         |
| Mix5   | xalancbmk, cactusADM, calculix, sphinx3 |
| Mix6   | perlbench, bzip2, gromacs, gobmk  |
| Mix7   | bwbases, povray, h264ref, lbm     |
| Mix8   | mcf, bwbases, milc, perlbench     |