# Linearly Compressed Pages: A Main Memory Compression Framework with Low Complexity and Low Latency

Gennady Pekhimenko, Todd C. Mowry, and Onur Mutlu
gpekhime@cs.cmu.edu, tcm@cs.cmu.edu, onur@cmu.edu

**Categories and Subject Descriptors:** B.3.2 [**Design Styles**]: Primary Memory, Cache Memories
**Keywords:** Main memory compression, cache compression

## 1. Introduction and Motivation

Main memory is a critical resource in modern systems. Its capacity must be sufficiently provisioned to prevent the target application's working set from overflowing into the backing store (e.g., hard disk, flash storage) that is slower by orders of magnitude compared to DRAM. Adding more DRAM to meet the increasing main memory demand is ideally not desirable, because this will significantly increase both the system's cost and the power consumption.

One potential way of addressing DRAM capacity problem is by using data compression. Data compression was previously used to increase the effective cache size [2, 3, 5, 6, 7], and DRAM capacity [1, 4], and to decrease bandwidth consumption [6]. One of the primary concerns with data compression is that *decompression* lies on the critical path of accessing any compressed data. To counter this problem, prior work [2, 6, 7] has proposed specialized compression algorithms that exploit some regular patterns present in in-memory data, and has shown that such specialized algorithms have high compression ratios while incurring relatively lower decompression latencies (1-5 cycles).

One of the primary reasons, why applying these works directly to main memory is difficult, is a virtual to physical page mapping. When a virtual page is mapped to a compressed physical page, the virtual page offset of a cache line can be different from the corresponding physical page offset since each physical cache line is expected to be smaller than the virtual cache line. In fact, where a compressed cache line resides in a main memory page is dependent on the sizes of the compressed cache lines that come before it in the page. As a result, accessing a cache line within a compressed page in main memory requires an additional layer of indirection to compute the location of the cache line in main memory. This indirection not only increases complexity and cost, but also can increase the latency of main memory access, and thereby degrade system performance.

**Our goal** in this work is to build a main memory compression framework that neither incurs the latency penalty nor requires power-inefficient hardware. To this end, we propose a new approach to compress pages, which we call *Linearly Compressed Pages* (LCP). The key idea of LCP is that if all the cache lines within a page are compressed to the *same size*, then the location of a cache line within a compressed page is simply the product of the index of the cache line and the compressed cache line size. Based on this idea, each compressed page has a target compressed cache line size. Cache lines that cannot be compressed to this target size are called *exceptions*. All exceptions, along with the metadata required to locate them, are stored separately on the same compressed page. We adapt two previously proposed compression algorithms (Frequent Pattern Compression [2] and Base-Delta-Immediate Compression [6]) to fit the requirements of LCP, and show that the resulting designs can significantly improve effective main memory capacity on a wide variety of workloads.


*PACT'12,* September 19–23, 2012, Minneapolis, Minnesota, USA.
ACM 978-1-4503-1182-3/12/09.

## 2. Main Memory Compression Background

In summary, prior works on hardware-based main memory compression mitigate the performance degradation due to the main memory address computation problem by either adding large hardware structures (e.g., 32MB L2 cache) that consume significant area and power [1] or by using techniques that require energy-inefficient hardware and lead to wasted energy [4]. In contrast, in this work, we propose a new main memory compression framework that simplifies the nature of the main memory address computation, thereby significantly reducing the associated latency (compared to prior approaches) without requiring energy-inefficient hardware structures.

## 3. Our Approach: Key Idea

The key idea of LCP is to *use a fixed size for compressed cache lines within a page* (which eliminates complex and long-latency main memory address calculation problem that arises due to variable-size cache lines), and still enable a page to be compressed even if not all cache lines within the page can be compressed to that fixed size (which enables high compression ratios). Since all the cache lines within a page are compressed to the same size, then the location of a compressed cache line within the page is simply the product of the index of the cache line within the page and the size of the compressed cache line – essentially a linear scaling using the index of the cache line.

It is clear, that some cache lines within a page may not be compressible to a specific fixed size (using some simple compression mechanism as BDI). Also, a cache line which is originally compressed to the target size may later become uncompressible due to a write. LCP stores such uncompressible cache lines of a page separately from the compressed cache lines (but still within the page), along with the metadata required to locate them.

There are two major benefits of having such a compressed page organization. First, it is possible to avoid the accesses to the special pages with metadata (as in MXT), and, hence, potentially avoid additional DRAM row misses. Second, when the data and the metadata are both located on the same page (e.g., 4kB in size), it is then possible to pipeline the requests to both of the them, since they are located on the same active DRAM row. Both these benefits decrease the negative effect of a higher access latency.

We also propose three simple optimizations to our LCP framework: (1) metadata cache, (2) memory bandwidth reduction, and (3) exploiting zero pages and zero cache lines.

## References

[1] B. Abali and et al. Memory expansion technology (MXT): software support and performance. *IBM J. Res. Dev.*, 45:287–301, 2001.
[2] A. R. Alameldeen and D. A. Wood. Adaptive cache compression for high-performance processors. In *ISCA-31*, 2004.
[3] J. Dusser, T. Piquet, and A. Seznec. Zero-content augmented caches. In *ICS*, 2009.
[4] M. Ekman and P. Stenstrom. A robust main-memory compression scheme. In *ISCA-32*, 2005.
[5] E. G. Hallnor and S. K. Reinhardt. A unified compressed memory hierarchy. In *HPCA-11*, 2005.
[6] G. Pekhimenko and et al. Base-delta-immediate compression: A practical data compression mechanism for on-chip caches. In *PACT*, 2012.
[7] J. Yang, Y. Zhang, and R. Gupta. Frequent value compression in data caches. In *MICRO-33*, 2000.