# MBZip: A Case for Compressing Multiple Data Blocks

Raghavendra K, Biswabandan Panda, Madhu Mutyam, Department of CSE, IIT Madras

## 1 Introduction and Motivation

Compression techniques at the LLC and DRAM play an important role in improving system performance by increasing their effective capacities. Existing compression techniques, such as B$\triangle$I [1] and LCP [2], compress a single cache block/DRAM column independently. These techniques are oblivious to the data patterns spread across multiple blocks. Applications exhibit data locality that spread across multiple consecutive data blocks. We observe that there is significant opportunity available for compressing multiple consecutive data blocks both at the LLC and at the DRAM. On an average, across 22 SPEC 2000/2006 applications, around 25% of the cache blocks, when grouped together in groups of 2 to 8 blocks, can be compressed into a single cache block. In DRAM, more than 30% of the columns residing in a single page can be grouped together in groups of 2 to 6 and compressed.

We propose a mechanism that can compress multiple cache blocks into one single block (*zipped* block) at the LLC. We also propose a tag structure to index into these cache blocks. This indexing does not incur any redirectional delay. To complement this mechanism, we compress multiple columns into one zipped column at the DRAM and communicate the same to the LLC.

## 2 Our Approach

In B$\triangle$I, a single block is compressed and stored along with the associated base and the encoding bits representing the data pattern exploited. In MBZip, multiple consecutive blocks that share common data pattern are compressed together into a single zipped block using B$\triangle$I, and thus need only one set of encoding bits and the common base, in total.

*At cache (MBZip-C):* A zipped cache contains 3 types of blocks: uncompressible (64 bytes), compressible (single block $<$ 64 bytes) and zipped (8 to 64 bytes). Similar to B$\triangle$I, in MBZip we double the number of tags per set. To handle these different types of blocks, half of the tags retain the generic index function, whereas the remaining half employ zipped index function (the index bits are shifted by the maximum number of blocks a zipped block is allowed to hold). Note, a zipped tag can also point to an uncompressed/compressed block.

*At DRAM (MBZip-M):* With MBZip, we intend to zip utmost 6 data blocks into one DRAM column. The first block in a zipped column is always the block that would have been present in a generic DRAM. In DRAM, a block of data is either in uncompressed or zipped format. If the block can be zipped together with it's consecutive blocks, it will share the column space with those other blocks, else it is stored alone. For each column, we store 8 bits of metadata information (3 encoding & 5 valid bits) in a reserved DRAM space. We also use a metadata cache to store metadata of frequently used DRAM pages. Using MBZip-M, we can service multiple block requests with a single read, and hence improve performance. Note, in MBZip-M the same block of data might be present in 6 different columns. This replication of data does not change the generic DRAM address mapping policy apart from reserving space for metadata.

*MBZip-CM:* A page brought into DRAM is stored in the zipped format. On a request, either an uncompressed or a zipped block is transferred to the cache along with the corresponding metadata bits. Depending on whether the block is uncompressed or zipped, the indexing function is chosen (generic or zipped). If an uncompressed block from DRAM is compressible, it is compressed & stored in the cache. Depending on whether the dirty data written back from higher levels of cache is zippeable or not, it is either re-zipped or stored as a compressed/uncompressed block. When a zipped block containing dirty data (might contain other clean blocks) is evicted from the cache, the entire block is written to the write buffer. This dirty zipped block is written back to the DRAM, and it's valid bits & those of the previous 5 columns are updated accordingly. In effect, multiple blocks of data compressed into one single block is fetched from the DRAM, handled in the cache, written back to the write buffer and then to the write queue and from thereon again to the DRAM.

## 3 Results

We evaluate the effectiveness of MBZip-C and MBZip-CM, in terms of harmonic speedup (HS). We compare our techniques with a system that uses no compression. We use 70 4-core and 25 8-core workloads, which contains a mix of SPEC CPU 2000/2006 benchmarks. On an average (geomean), for 4-core workloads, MBZip-C and MBZip-CM provide 15.4% and 21.9% improvement in HS. For 8-core workloads, MBZip-C and MBZip-CM provide HSs of 10.2% and 17.1%, respectively. In contrast, BDI provides an improvement of 11.4% and 7.3% for 4- and 8-core.

**References**

[1] Pekhimenko et al., "Base-delta-immediate compression:practical data compression for on-chip caches", in PACT 2012.

[2] Pekhimenko et al., "Linearly compressed pages: a low-complexity, low-latency main memory compression framework", in MICRO 2013