Exploration of GPU-enabled lossless compressors

Stefan Rua, stefan.elias.rua@cern.ch

Contents

Introduction **1**

Algorithms **2**

Compressors **2**
- CPU . . . 2
- GPU . . . 2
  - BSC . . . 2
  - CULZSS . . . 3
  - nvCOMP . . . 3
  - DietGPU . . . 3
  - HuffmanCoding_MPI_CUDA . . . 3
  - cuda_bzip2 . . . 3
- NX . . . 3
  - libnxz . . . 3

Benchmarking **4**
- Stand-alone runs . . . 4
- Finer evaluation . . . 4
- Data . . . 5

Results **5**

Discussion **7**

Conclusion **7**

Introduction

The data produced by CMS's High Level Trigger (HLT) is compressed to speed up network transfers, and currently this is done using CPUs. However, as we have new computers with powerful GPUs, it would be nice to transfer some of the load onto the GPUs. This is the motivation for finding a good GPU-enabled compressor. GPU-enabled compressors aren't widely adopted yet, but in the last ten years some researchers and companies have showed interest in the topic.

Algorithms

Most full compressors use a combination of multiple more fundamental compression algorithms. These are the most common ones:

- **Burrows-Wheeler transform**[1,2]: also called block-sorting compression, sorts the data so that repeating patterns are near each other, making it easier to compress.
- **Huffman coding**[3,4]: gives characters prefix codes that vary in length based on their frequency, so that the most common ones take the least space.
- **The Lempel-Ziv family (LZ)**[5,6]: compression algorithms that replace patterns with references to previous occurrences.
- **Asymmetric numeral systems (ANS)**[7,8]: encodes the data into a natural number based on the frequency of different symbols.

Compressors

CPU

These are some of the most commonly used compressors:

- **zlib**[9]: a compression library that implements the DEFLATE algorithm, which is a combination of Huffman coding and LZ77, a Lempel-Ziv variant.
- **LZ4**[10]: an algorithm based on LZ77 that prioritizes speed.
- **XZ**[11] / **LZMA**[12]: the Lempel–Ziv–Markov chain algorithm, is an algorithm that prioritizes compression ratio.
- **Zstandard**[13]: a compressor that combines ANS and LZ77.

GPU

These are the GPU compressors I found on GitHub and papers during my exploration.

BSC

BSC[14] is a GPU accelerated block-sorting compressor. The default makefile doesn't compile the necessary files for CUDA support, so a few changes were needed. Multiple block sorting algorithms are available: the Burrows-Wheeler transform and sort transforms of order 3 to 8. The sort transform is a generalization of the Burrows-Wheeler transform that takes into account a different size context when

[1] https://en.wikipedia.org/wiki/Burrows-Wheeler_transform
[2] https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf
[3] https://en.wikipedia.org/wiki/Huffman_coding
[4] http://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf
[5] https://en.wikipedia.org/wiki/Lempel-Ziv-Welch
[6] https://courses.cs.duke.edu/spring03/cps296.5/papers/ziv_lempel_1978_variable-rate.pdf
[7] https://en.wikipedia.org/wiki/Asymmetric_numeral_systems
[8] https://arxiv.org/pdf/1311.2540.pdf
[9] https://en.wikipedia.org/wiki/Zlib
[10] https://en.wikipedia.org/wiki/LZ4_(compression_algorithm)
[11] https://en.wikipedia.org/wiki/XZ_Utils
[12] https://en.wikipedia.org/wiki/Lempel-Ziv-Markov_chain_algorithm
[13] https://en.wikipedia.org/wiki/Zstd
[14] https://github.com/IlyaGrebnov/libbsc

sorting. Only sort transforms of order 5 to 8 can use the GPU, of which 7 and 8 use it by default and 5 and 6 require and additional flag for it.

CULZSS

CULZSS[15,16] is a CUDA implementation of LZSS. CULZSS-bit[17], a more recent and reportedly better performing version exists, but no source code or binary is available for it. I emailed the author, but he couldn't find the code either. He also mentioned that the code for the original CULZSS on github might not work.

nvCOMP

nvCOMP[18] is a library for GPU compression by Nvidia, and it implements Deflate, LZ4, Cascaded, Snappy, BitComp, and ANS. I expect nvCOMP's compressors to perform well, as it is in Nvidia's interest to make their GPUs look good. Unfortunately nvCOMP was made proprietary in version 2.3.

DietGPU

DietGPU[19] is a work-in-progress ("very early alpha preview") compression library by Facebook which implements ANS. As its purpose they state speeding up network transfers in a data center environment, which matches our use case. DietGPU provides two codecs: a general byte-oriented entropy encoder and decoder, and another one specifically for floating point data. At the time of writing only a Python API for PyTorch tensors is ready, but the underlying encoders and decoders are implemented in C++ and CUDA, making it possible to utilize library for other data as well.

HuffmanCoding_MPI_CUDA

HuffmanCoding_MPI_CUDA[20] performs Huffman compression with the option of using CUDA, MPI, CUDA and MPI, or no parallelization at all. I used it on the CUDA setting.

cuda_bzip2

cuda_bzip2[21] is a modified version of bzip2 with CUDA support. bzip2 is a compressor based on the Burrows-Wheeler transform.

NX

libnxz

IBM's POWER9 processor has a dedicated hardware accelerator for compression called NX. The library that provides an interface for compressing on this accelerator is called libnxz[22], and is a drop-in

[15] https://github.com/adnanozsoy/CUDA_Compression
[16] https://web.cs.hacettepe.edu.tr/~aozsoy/papers/2011-ppac.pdf
[17] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7079027
[18] https://developer.nvidia.com/nvcomp
[19] https://github.com/facebookresearch/dietgpu
[20] https://github.com/smadhiv/HuffmanCoding_MPI_CUDA
[21] https://github.com/aditya12agd5/cuda_bzip2
[22] https://github.com/libnxz/power-gzip https://dl.acm.org/doi/pdf/10.1109/ISCA45697.2020.00012

replacement for zlib.

Benchmarking

Stand-alone runs

As a first step I ran the stand-alone programs for the compressors, measuring the time taken by them using the `time` command. This gave me a rough idea of their performance, based on which I selected the ones to examine in more detail. libnxz and DietGPU are excluded from these because I found out about them later, and they seemed promising enough right away. The only CPU compressors included in this part are XZ, Zstandard, and LZ4 because they give a good enough idea of the compression ratios and throughputs achievable with CPUs.

Finer evaluation

lzbench[23] is an in-memory benchmarking tool for compressors. Excluding the time taken to read and write files to and from the disk gives a more accurate idea of the compressor's throughput. All compressors are also compiled with the same options when possible, reducing variation from compiler optimizations and such. I added DietGPU, libnxz, and the most promising compressor form the stand-alone tests, BSC, to lzbench. DietGPU and libnxz were compiled using their own makefiles and thus different compiler options. The throughputs shown are the highest ones from 5 runs. For the GPU compressors, copying from RAM to GPU memory and back is included.

Adding the more recent proprietary version of nvCOMP to lzbench was taking more time than what I was willing to spend on it, so it is excluded from the tests. The LZ4 implementation from nvCOMP's older open-source version is included since it was already available in lzbench.

The lzbench benchmarks were run on two machines, one that resembles the actual HLT ones and another with a POWER9 CPU for testing libnxz.

HLT-like

- AMD EPYC 75F3
  - 32 cores
  - max. 4 GHz
  - 256 MB L3 cache
- Nvidia Tesla T4
  - 2560 CUDA cores
  - 16 GB GDDR6
  - 8.1 TFLOPS

POWER9

- 8335-GTH / IBM Power System AC922
- IBM POWER9
  - 32 cores
  - max. 4 GHz
  - 320 MB L3 cache
- 4 x Nvidia Tesla V100
  - 5120 CUDA cores
  - 32 GB HBM2

[23] https://github.com/inikep/lzbench

– 15.7 TFLOPS

The libnxz benchmark are only in the POWER9 results, as that is the chip with the NX accelerator.

Data

The data that was compressed in these benchmarks consists of raw event files as they come from the HLT. All lzbench benchmarks were run on 100 proton-proton collision files and 100 heavy ion collision files. The stand-alone tests were run only on the proton-proton files.

100 PP events

- HadronsTaus stream from 2022
- pileup $\approx$ 50
- 100 files
- 170 MB
- 1.4 MB to 2.1 MB each

100 HI events

- from 2018
- 100 files
- 131 MB
- 644 KB to 5.5 MB each

Results

The labels in the plots are the command or library names, some of which differ from the more verbose names used earlier. See the table below for clarification.

Abbreviation	Compressor	Device
bsc	BSC	GPU
culzss	CULZSS	GPU
culzss_hcmc	CULZSS + HuffmanCoding_MPI_CUDA	GPU
dietgpu	DietGPU	GPU
hcmc	HuffmanCoding_MPI_CUDA	GPU
libnxz	libnxz	IBM NX
lz4	LZ4	CPU
lzma	LZMA	CPU
nvcomp_lz4	nvCOMP's LZ4	GPU
xz	XZ	CPU
zlib	zlib	CPU
zstd	Zstandard	CPU

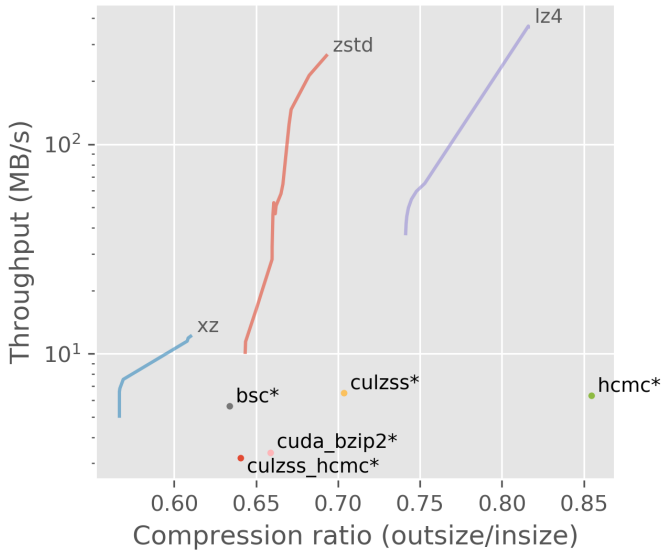

Figure 1: Stand-alone tests. GPU compressors are marked with ∗.

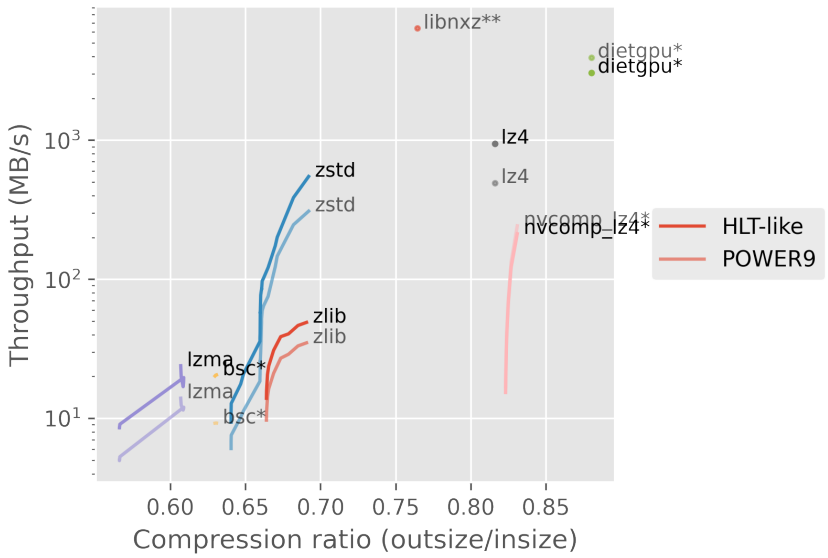

Figure 2: In-memory benchmark, proton-proton events. GPU compressors are marked with ∗, NX with ∗∗.

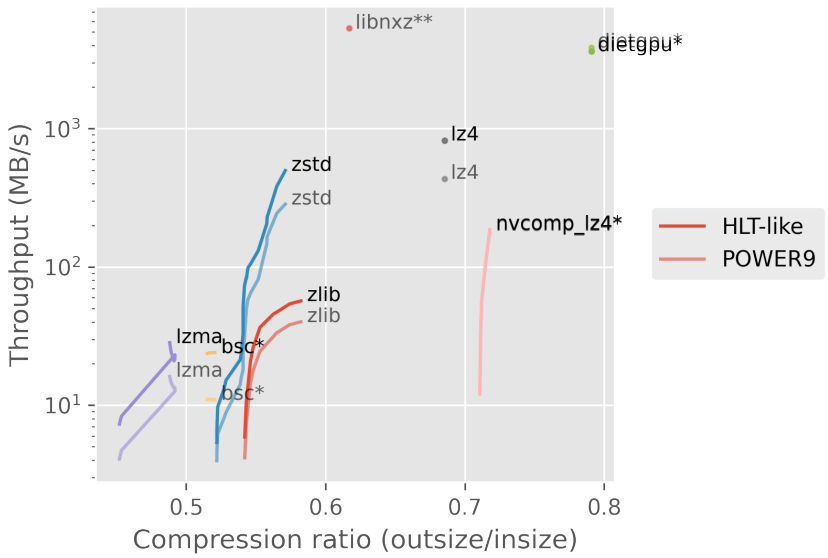

Figure 3: In-memory benchmark, heavy ion events. GPU compressors are marked with ∗, NX with ∗∗.

Discussion

The best candidate from the GPU compressors included in the stand-alone tests seems to be BSC, which on its lowest setting performs similarly to zstd on its highest setting.

Out of the results produced by lzbench, libnxz stands out with its combination of extreme throughput and reasonable compression ratio. BSC is outperformed by XZ and LZMA in both ratio and throughput, and DietGPU is very fast but suffers from relatively poor compression.

It would be nice to use libnxz, but POWER9 machines are prohibitively expensive and as stated in the introduction, the whole point is to make use of the GPUs that we already have. DietGPU would make use of them, but even ignoring the suboptimal compression ratio, throughput would most likely be worse than for Zstandard which can be run on a massive number of threads at once for separate event files.

Conclusion

Using the CPU compressor Zstandard seems to be the reasonable choice with our current hardware and the state of GPU compressors. An eye should be kept on DietGPU, as it is still under rapid development.