ndzip-gpu

Efficient Lossless Compression of Scientific Floating-Point Data on GPUs

Fabian Knorr, Peter Thoman and Thomas Fahringer

University of Innsbruck, Austria

SC21 St. Louis, MO | science & beyond.

Context

Celerity

High-level C++ for Accelerator Clusters

Runtime system for **GPU clusters**

- Based on SYCL
- Purely declarative data flow
- Well-suited for multidimensional algorithms on dense arrays

Current **development goals:**

- Transfer latency optimization
- Fast automatic checkpointing

. . . all without user intervention!

Speeding up Data Transfers via Compression

1

Scientific applications primarily work on **floating-point data**

⇓

Compressor must be **effective on floats**

2

For Celerity, operation must be **transparent** to the user

⇓

Compression must be **lossless**

3

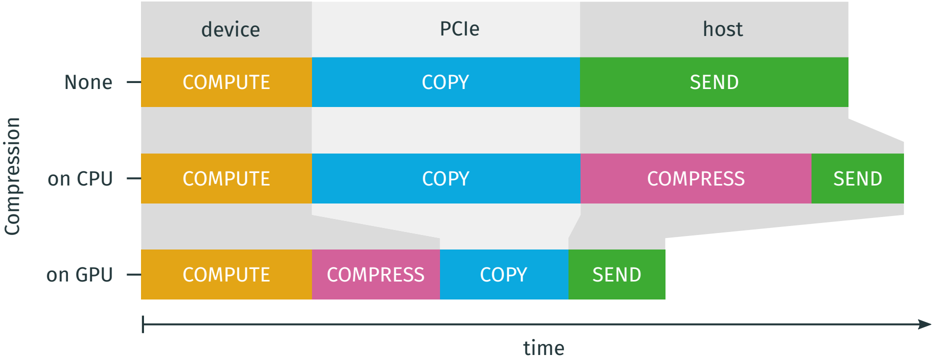
Primary goal is to **maximize throughput**, not minimize storage

⇓

Compressor must be **fast** enough to saturate the link

Benefits of Compression on the GPU

Compressing data directly on the GPU will accelerate PCIe transfers and save CPU time.



Depending on the hardware, compressing on the device allows direct GPU $\rightarrow_{\text{PCIe}}$ NIC copy of compressed data without going through system memory.

Data Compression Challenges on GPU

Mutable Encoder / Decoder State

In general-purpose compression,

- compressor updates its probability model with each symbol
- decompressor reconstructs the model in the same way

Variable Length Encoding

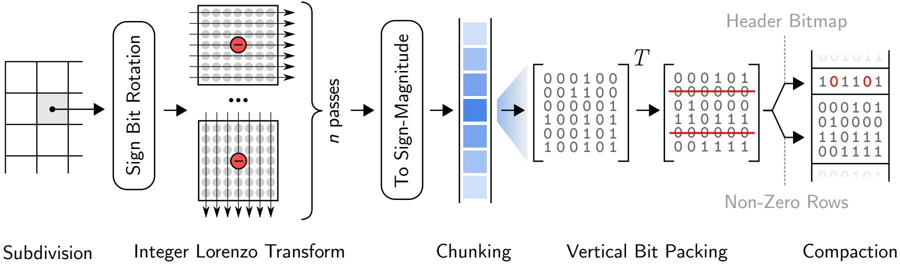
In lossless compression,

- output stream positions are not known ahead of time
- common encoders output symbols with arbitrary bit-length

For high-throughput GPU compression, we explore

1. **local decorrelation** schemes with minimal state
2. an encoder that only requires **coarse-granular addressing**.

The ndzip floating-Point Compressor

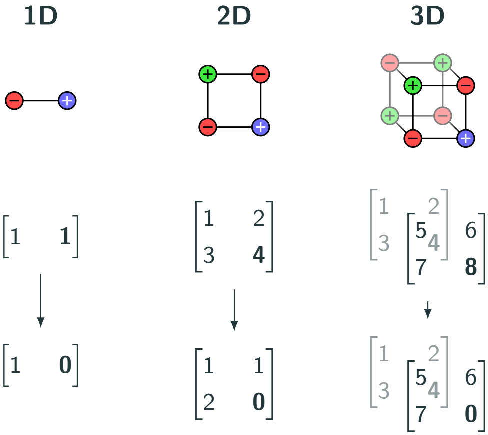


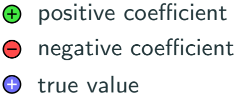
ndzip[2]: Lossless block compressor for dense multi-dimensional floating-point data

- Model: Data is smooth locally in multiple dimensions
- Impressive single-core performance (2.2–3.0 GB/s on AMD Ryzen 9 3900X)
- So far CPU only, but designed for highly-parallel implementations

Decorrelation: The Integer Lorenzo Transform

Compute residuals by replacing each data point with the difference to its predecessor.
In the multi-dimensional case, repeat along each axis. If data is smooth, **residuals are small**.



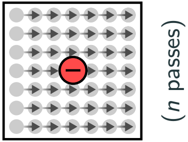


Since this transform is not reversible in floating-point arithmetic, it is **approximated in the integer domain**.

Integer Lorenzo Transform on GPU

Forward Transform

Each pass is fully parallel—assign threads to data points freely.

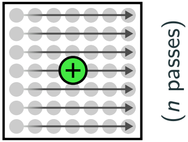


difference

Inverse Transform

The inverse pass corresponds to a prefix sum per lane.

- 1D case: Use a parallel scan
- 2D/3D case: Sum up sequentially, parallelize over lanes



prefix sum

Memory Access Patterns are Performance Critical

- Keep block in fast GPU shared memory between passes
- Be careful about memory layout to avoid bank conflicts

Residual Encoding: Vertical Bit Packing

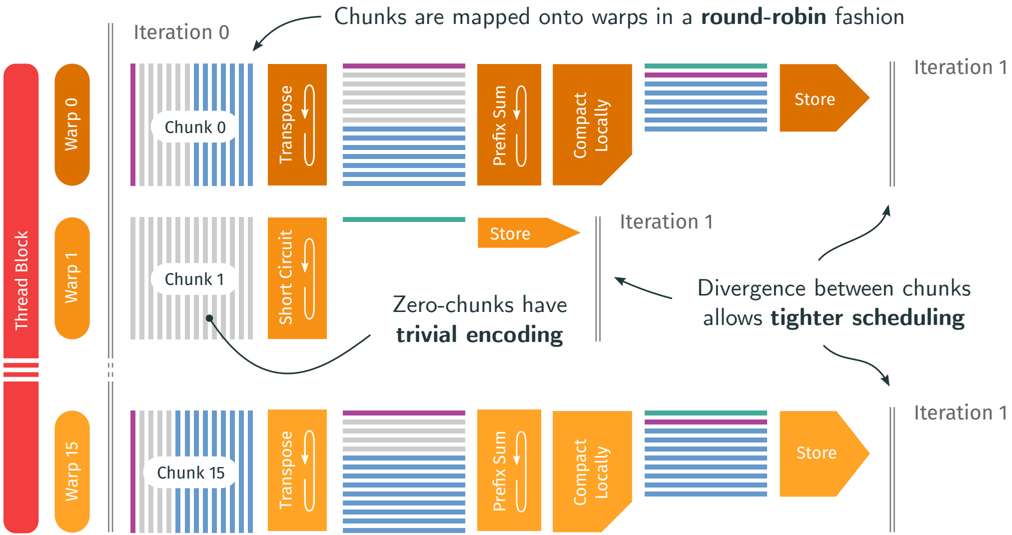
In sign-magnitude representation, small integer residuals have many leading-zero bits.

$$
\begin{matrix} \text{Word 0} \\ \text{Word 1} \\ \text{Word 2} \\ \text{Word 3} \\ \text{Word 4} \\ \text{Word 5} \\ \text{Word 6} \\ \text{Word 7} \end{matrix}
\begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}^T
=
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}
\rightleftharpoons
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}
\begin{matrix} \text{Head} \\ \text{Bit 0} \\ \text{Bit 4} \\ \text{Bit 5} \\ \text{Bit 6} \\ \text{Bit 7} \end{matrix}
$$

Hardware-friendly **Vertical Bit Packing** encoder:

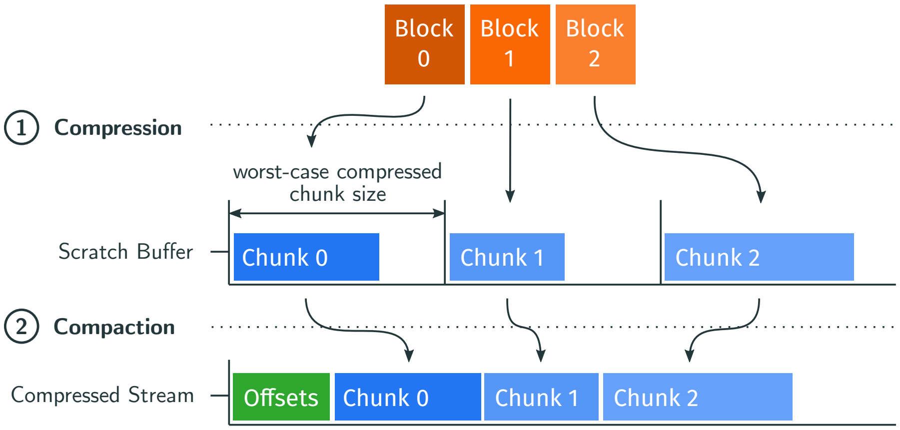
1. Group into chunks of 32 32-bit (64 64-bit) integers
2. Transpose the chunk to obtain one 32-bit (64-bit) word for each bit position
3. Strip zero words, communicate which positions were eliminated through a header bitmap

Vertical Bit Packing on GPUs



Fully Parallel Variable-Length Compression

Output positions are dependent on length of the previous blocks – use a scratch buffer:



Reference System

980 IBM POWER9 ACC922 nodes à 4× NVIDIA V100

Marconi-100 (Italy, TOP500 #14 in 2021-06)

Peak transfer rates in the system:

GPU memory	NVIDIA HBM2	900 GB/s
Host memory	8-chan DDR4-2666	170 GB/s
GPU → CPU	3× NVLink	150 GB/s
GPU/CPU → NIC	2× PCIe 4.0 x8	32 GB/s
Network	2× Infiniband EDR	25 GB/s

For maximum inter-node throughput, software I/O should be able to saturate the slowest link (25 GB/s).

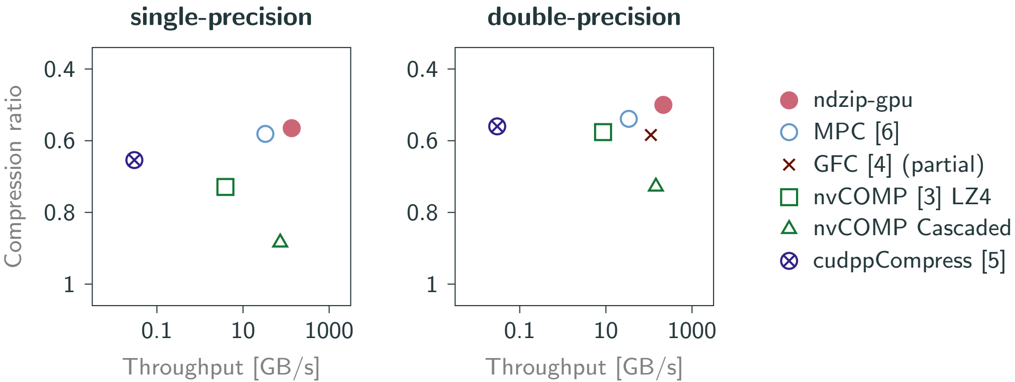
Test Data

Test Data from various scientific domains [1]:

dataset	single	double	extent
msg_sppm	✓	✓	34,874,483
msg_sweep3d	✓	✓	15,716,403
snd_thunder	✓		7,898,672
ts_gas	✓		4,208,261
ts_wesad	✓		4,588,553
hdr_night	✓		8,192 × 16,384
hdr_palermo	✓		10,268 × 20,536
hubble	✓		6,036 × 6,014
rsim	✓	✓	2,048 × 11,509
spitzer_fls_irac	✓		6,456 × 6,389
spitzer_fls_vla	✓		8,192 × 8,192
spitzer_frontier	✓		3,874 × 2,694

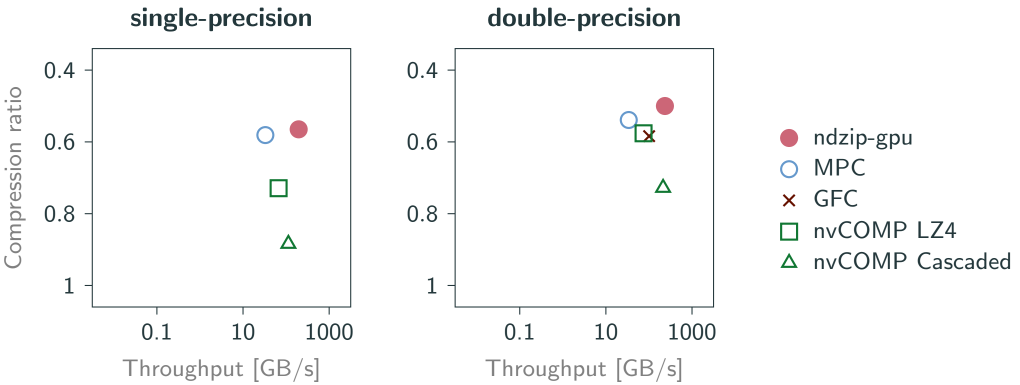
dataset	single	double	extent
asteroid	✓		500 × 500 × 500
astro_mhd	✓		128 × 512 × 1024
astro_mhd		✓	130 × 514 × 1026
astro_pt	✓	✓	512 × 256 × 640
flow		✓	16 × 7,680 × 1,0240
hurricane	✓		100 × 500 × 500
magrecon	✓		512 × 512 × 512
miranda	✓		1,024 × 1,024 × 1,024
redsea	✓	✓	50 × 500 × 500
sma_disk	✓		301 × 369 × 369
turbulence	✓		256 × 256 × 256
wave	✓	✓	512 × 512 × 512

Compression Performance on NVIDIA V100



	Ratio	Throughput
ndzip-gpu compress single-precision	**0.565**	**135 GB/s**
double-precision	**0.500**	**216 GB/s**

Decompression Performance on NVIDIA V100



	Ratio	Throughput
ndzip-gpu decompress single-precision	**0.565**	**196 GB/s**
double-precision	**0.500**	**235 GB/s**

Conclusion

On the reference hardware, **ndzip-gpu** outperforms state-of-the art GPU compressors on floating-point data both in throughput and compression ratio achieved.

Key takeaways

1. **Local decorrelation** allows efficient subdivision of the input space
2. In-place **Integer Lorenzo Transform** makes residual computation parallel
3. **Vertical Bit Packing** provides fast, word-alinged data reduction
4. A separate **compaction kernel** avoids synchronization on output positions

Thank You!

Now for your questions, please!

Livestream (without captions): view video . . . or feel free to contact me any time at
fabian@dps.uibk.ac.at.

ndzip-gpu is available at https://github.com/fknorr/ndzip.

Celerity is available at https://celerity.github.io.

F. Knorr, P. Thoman, and T. Fahringer.
Datasets for Benchmarking Floating-Point Compressors.
arXiv e-prints, page arXiv:2011.02849, Nov. 2020.

F. Knorr, P. Thoman, and T. Fahringer.
ndzip: A high-throughput parallel lossless compressor for scientific data.
In *2021 Data Compression Conference*. IEEE, 2021.

NVIDIA.
NVCOMP – High Speed Data Compression Using NVIDIA GPUs.
`https://developer.nvidia.com/nvcomp`.

M. A. O'Neil and M. Burtscher.
Floating-point data compression at 75 Gb/s on a GPU.
In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, pages 1–7, 2011.

References ii

R. A. Patel, Y. Zhang, J. Mak, A. Davidson, and J. D. Owens.
Parallel lossless data compression on the GPU.
IEEE, 2012.

A. Yang, H. Mukka, F. Hesaaraki, and M. Burtscher.
MPC: a massively parallel compression algorithm for scientific data.
In *2015 IEEE International Conference on Cluster Computing*, pages 381–389. IEEE, 2015.