ndzip-gpu

Efficient Lossless Compression of Scientific Floating-Point Data on GPUs

Fabian Knorr, Peter Thoman and Thomas Fahringer

University of Innsbruck, Austria
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!
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 → 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**.
**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.

**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

**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.

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

\[\times\]

\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

\[\Rightarrow\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

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

Chunks are mapped onto warps in a **round-robin** fashion.

Zero-chunks have a **trivial encoding**.

Divergence between chunks allows **tighter scheduling**.
Output positions are dependent on length of the previous blocks – use a scratch buffer:

1. **Compression**
   - Scratch Buffer
     - Chunk 0
     - Chunk 1
     - Chunk 2
   - worst-case compressed chunk size

2. **Compaction**
   - Compressed Stream
     - Offsets
     - Chunk 0
     - Chunk 1
     - Chunk 2
Reference System

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

Peak transfer rates in the system:

<table>
<thead>
<tr>
<th>Component</th>
<th>Type</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU memory</td>
<td>NVIDIA HBM2</td>
<td>900 GB/s</td>
</tr>
<tr>
<td>Host memory</td>
<td>8-chan DDR4-2666</td>
<td>170 GB/s</td>
</tr>
<tr>
<td>GPU → CPU</td>
<td>3× NVLink</td>
<td>150 GB/s</td>
</tr>
<tr>
<td>GPU/CPU → NIC</td>
<td>2× PCIe 4.0 ×8</td>
<td>32 GB/s</td>
</tr>
<tr>
<td>Network</td>
<td>2× Infiniband EDR</td>
<td>25 GB/s</td>
</tr>
</tbody>
</table>

For maximum inter-node throughput, software I/O should be able to saturate the slowest link (25 GB/s).
## Test Data from various scientific domains [1]:

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

<table>
<thead>
<tr>
<th>dataset</th>
<th>single</th>
<th>double</th>
<th>extent</th>
</tr>
</thead>
<tbody>
<tr>
<td>asteroid</td>
<td>✓</td>
<td></td>
<td>500 × 500 × 500</td>
</tr>
<tr>
<td>astro_mhd</td>
<td>✓</td>
<td></td>
<td>128 × 512 × 1024</td>
</tr>
<tr>
<td>astro_mhd</td>
<td>✓</td>
<td></td>
<td>130 × 514 × 1026</td>
</tr>
<tr>
<td>astro_pt</td>
<td>✓ ✓</td>
<td></td>
<td>512 × 256 × 640</td>
</tr>
<tr>
<td>flow</td>
<td>✓</td>
<td></td>
<td>16 × 7,680 × 1,0240</td>
</tr>
<tr>
<td>hurricane</td>
<td>✓</td>
<td></td>
<td>100 × 500 × 500</td>
</tr>
<tr>
<td>magrecon</td>
<td>✓</td>
<td></td>
<td>512 × 512 × 512</td>
</tr>
<tr>
<td>miranda</td>
<td>✓</td>
<td></td>
<td>1,024 × 1,024 × 1,024</td>
</tr>
<tr>
<td>redsea</td>
<td>✓ ✓</td>
<td></td>
<td>50 × 500 × 500</td>
</tr>
<tr>
<td>sma_disk</td>
<td>✓</td>
<td></td>
<td>301 × 369 × 369</td>
</tr>
<tr>
<td>turbulence</td>
<td>✓</td>
<td></td>
<td>256 × 256 × 256</td>
</tr>
<tr>
<td>wave</td>
<td>✓ ✓</td>
<td></td>
<td>512 × 512 × 512</td>
</tr>
</tbody>
</table>
Compression Performance on NVIDIA V100

Throughput [GB/s] vs. Compression ratio

**single-precision**
- ndzip-gpu: 0.565, 135 GB/s
- MPC [6]:
- GFC [4] (partial):
- nvCOMP [3] LZ4:
- nvCOMP Cascaded:
- cudppCompress [5]:

**double-precision**
- ndzip-gpu: 0.500, 216 GB/s
- MPC [6]:
- GFC [4] (partial):
- nvCOMP [3] LZ4:
- nvCOMP Cascaded:
- cudppCompress [5]:

ndzip-gpu: Efficient Lossless Compression of Scientific Floating-Point Data on GPUs
Decompression Performance on NVIDIA V100

**Throughput [GB/s]**

**Compression ratio**

<table>
<thead>
<tr>
<th>Method</th>
<th>Ratio</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndzip-gpu</td>
<td>0.565</td>
<td>196 GB/s</td>
</tr>
<tr>
<td>double-precision</td>
<td>0.500</td>
<td>235 GB/s</td>
</tr>
</tbody>
</table>

**Graphs**

- **single-precision**
- **double-precision**

**Legend**
- ndzip-gpu
- MPC
- GFC
- nvCOMP LZ4
- nvCOMP Cascaded
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-aligned data reduction
4. A separate **compaction kernel** avoids synchronization on output positions
Now for your questions, please!

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

\[ \textbf{ndzip-gpu} \text{ is available at } \text{https://github.com/fknorr/ndzip}. \]

\[ \textbf{Celerity} \text{ is available at } \text{https://celerity.github.io}. \]
F. Knorr, P. Thoman, and T. Fahringer.  
**Datasets for Benchmarking Floating-Point Compressors.**  

F. Knorr, P. Thoman, and T. Fahringer.  
**ndzip: A high-throughput parallel lossless compressor for scientific data.**  

NVIDIA.  
**NVCOMP – High Speed Data Compression Using NVIDIA GPUs.**  

M. A. O’Neil and M. Burtscher.  
**Floating-point data compression at 75 Gb/s on a GPU.**  
Parallel lossless data compression on the GPU.  

A. Yang, H. Mukka, F. Hesaaraki, and M. Burtscher.  
MPC: a massively parallel compression algorithm for scientific data.  