Update on Blosc2

New Features And What We Are Working On

Francesc Alted / @FrancescAlted
The Blosc Development Team / @Blosc2
CEO [iron] ironArray / @ironArray

LEAPS Innov WP7 (data reduction and compression) meeting
March 29th 2023
What is Blosc2?

- Next generation of Blosc(1), a high performance compressor.
- Blosc2 adds 63-bit containers that expand over the existing 31-bit containers (chunks) in Blosc1.
- Metalayers for adding info for apps and users.
The Blosc Development Team

Aleix Alcacer
Oscar Guiñón
Marta Iborra
Alberto Sabater
Nathan Moinvaziri
Francesc Alted (BDFL)
Agenda

- The second partition in Blosc2/HDF5: what’s it providing?
- New Blosc2 NDim and NDArray objects
- Bytedelta: enhancing your compression toolset
- Ongoing Work
The second partition in Blosc2/HDF5

What is it providing?
Chunks and Blocks: allow better granularity during compression/decompression

Modern processors work best when workloads fit in internal caches

Chunks can be set to fit in L3 cache, and blocks to fit in L1/L2

Better performance!
Second partition at work

- Using data from ERA5 datasets
- Inkernel queries => full scan on disk

- Much higher speed than default ZLIB + shuffle in HDF5
- 2x performance than plain HDF5 with no compression
- Almost reaching perf of efficient, pure in-memory libraries like pandas
Bypassing the HDF5 pipeline

- HDF5 pipeline implementation is powerful but slow
- PyTables has support for bypassing it via the H5Dwrite_chunk / H5Dread_chunk
- Unleash the full I/O parallel in Blosc2
Bypassing the HDF5 pipeline: Writing

Blosc2 optimized -> bypass the HDF5 pipeline. Almost 2x faster!
Bypassing the HDF5 pipeline: Reading

Blosc2 optimized -> bypass the HDF5 pipeline. Almost 2x faster!
NDim And NDArray

Blosc2 Goes Multidimensional
C-Blosc2 NDim: Multidimensions for C

- Each NDim array is split in chunks
- Each chunk is split in blocks
- All the partitions are multidimensional!
- Metalayer representing both multidimensionality and **data types** (new!)

import blosc2

a = blosc2.full((4, 4), fill_value=9)
a.resize((5, 7))
a[3:5, 2:7] = 8
print(a[:])

Features:

- Create arrays in memory or on disk
- Flexible resize (including shrinking)
- Support for all NumPy data types
- Efficient conversion from/to NumPy
- Mimic NumPy API
- Version 2.1 out; meant for production

Output:

```
[[9 9 9 9 0 0 0]
 [9 9 9 9 0 0 0]
 [9 9 9 9 0 0 0]
 [9 9 8 8 8 8 8]
 [0 0 8 8 8 8 8]]
```

Blosc2 NDim read/write performance

4-d array:
- shape: (50, 100, 300, 250)
- chunk shape: (10, 25, 50, 50)
- block shape: (3, 5, 10, 20)
- data type: float64

- Doing a complete read is generally faster
- Writing is more expensive because of the overhead of double partitioning

https://www.blosc.org/posts/blosc2-ndim-intro/
Leveraging the second partition in Blosc2 NDim

Much more selective and faster queries!
Blosc2 NDim partial read performance

Faster slicing due to higher data selectivity in double partitioning
Bytedelta
A new filter for Blosc2
Bytedelta: How it works

Blosc2 Filter Pipeline

shuffle ->

bytedelta ->

Based on initial work by Aras Pranckevičius

https://www.blosc.org/posts/bytedelta-enhance-compression-toolset/
Bytedelta: How does it perform?

Tested on 5 different [ERA5 datasets](https://docs-assets.developers.google.com/cloud Platform/blobs/documents/aerospace/erosa المجيبات online help center) (atmospheric reanalysis of the global climate): wind, snow, flux, precip and pressure

- Some show some complex structure (wind)
- Others are simpler (snow)
Bytedelta: How does it perform?

Bytedelta compress better than shuffle or bitshuffle on average
Median for bytedelta (best): 6.36 x
Median for bitshuffle (second best): 5.66 x
Bytedelta: How does it perform?

- Compression ratio versus dataset
- Points for all the general codecs in Blosc2 (BLOSCLZ, LZ4, LZ4HC, ZLIB, ZSTD), and for different filters
Bytedelta: How does it perform?

Compression ratio for the datasets with more complexity (entropy)

For pressure, bytedelta achieves up to 37% better results than second best (!)
Bytedelta works best in combination with ZSTD codec (and high clevels of ZLIB)
Bytedelta shows excellent compression speed when using ZSTD with clevel 1
Bytedelta + ZSTD level 1: a good default
(specially for complex datasets)
Ongoing Work
Fine Tuning Performance with BTune

https://btune.blosc.org

- BTune can fine tune the different parameters of the underlying Blosc2 storage to perform as best as possible.
- Can be trained to find the best codec & filter with deep learning.
- Looking for beta testers!
Support for High Throughput JPEG 2000 (HTJ2K)

- Experimenting with OpenHTJ2K, an open source HTJ2K implementation by Osamu Watanabe.
- We already have a working implementation, but:
  - It cannot leverage Blosc2 multithreading
  - Big library: cannot be included straight in C-Blosc2

Proposal:
Use a smarter plugin system that can load plugins dynamically in runtime. Already working on that, but need more resources.
Conclusion
Blosc2 is making rapid progress

The Blosc2 development team has recently:

• Implemented **native support for Blosc2 in HDF5**, bypassing the HDF5 pipeline
• **New Ndim and NDArray** objects for easy handling of multidimensional data containers
• **New bytedelta** filter
• **BTUNE**, a tool for automatically select best Blosc2 parameters, is **in beta**
• Prospective work done for High Throughput JPEG 2000

Blosc2: a highly efficient and flexible tool for **compressing your data, your way**
Proposals

1. Use Blosc2 in combination with HDF5 direct chunking mechanism for efficient compression and parallel I/O.

2. Help in determining optimal compression pipelines by adapting to user data and using machine learning techniques.

3. Support for High Throughput JPEG 2000
Thanks to donors & contracts!

Without them, we could not have possibly put Blosc2 into production status: Blosc2 2.0.0 came out in June 2021; now at 2.8.0.
Enjoy data!

https://blosc.org/