

Blosc and Friends – PyGrunn

Valentin Haenel

Freelance Consultant and Software Developer

<http://haenel.co>

22.05.2015

Version: v0.2

<https://github.com/esc/blosc-and-friends>



This work is licensed under the *Creative Commons Attribution-ShareAlike 3.0 License*.

Outline

- 1 Blosc
- 2 Python-Blosc
- 3 Bloscpack
- 4 Bcolz

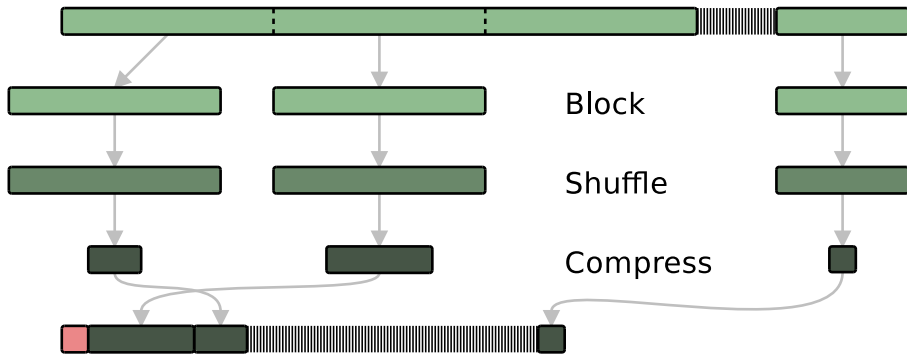
Disclaimer

- All examples with Py3
- No comparisons to other technologies

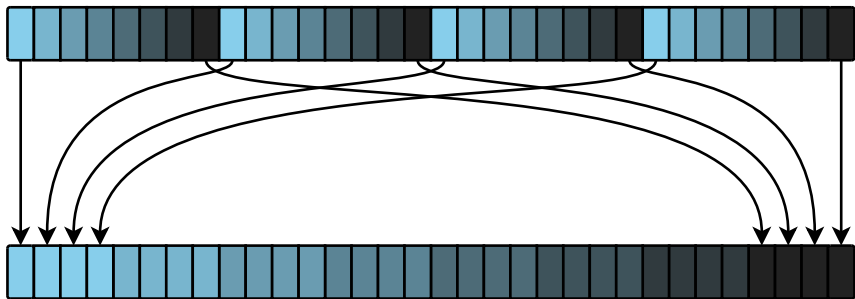
- Valentin Haenel
- Freelance developer and consultant
- Free-software enthusiast
- Check my homepage: <http://haenel.co>
- Follow me on Twitter: [esc_](#)

- **Blocking** by operating on data in blocks
- **Shuffling** by doing a byte shuffle
- **Fast** by multithreading across **blocks**
- **Metacodec** because it can drive **[lz4|snappy|zlib]**
- (But also comes with it's very own **blosclz**)

Schematic



Shuffle



- Reorder **bytes** within a **block** by significance
- Requires the **typesize**
- Can leverage **SSE2** or **AVX** if available

Blosc... Why?

- Accelerate computation by compression
- Keep data compressed in memory
- Mitigate the **starving CPU problem**
- Transmit data faster from memory → CPU

Outline

- 1 Blosc
- 2 Python-Blosc
- 3 Bloscpack
- 4 Bcolz

- **Python-Blosc** ← Python bindings
- Python C-API
- Blosc (and codecs) are vendored
- Can be dynamically linked

\$ pip install blosc

Code, Yo!

```
>>> import blosc
>>> b = b"b"*888
>>> c = blosc.compress(b, typesize=8)
>>> len(b)/len(c)
15.857142857142858
>>> c = blosc.compress(b, typesize=1, shuffle=False,
... clevel=9, cname='lz4')
>>> len(b)/len(c)
23.36842105263158
```

C'mmon man, your pulling my leg?!

```
>>> d = blosc.decompress(compressed)
>>> assert b == d
```

What Up?

- Long sequence of the same character
- Not hard to beat
- Feel free to try with some input of your own choosing
- Feel free to compare to `zlib`

- Doesn't yet support compressing via buffer-interface

Outline

- 1 Blosc
- 2 Python-Blosc
- 3 Bloscpack**
- 4 Bcolz

And now some «stuff» layered on top!

- **Bloscpack**
- Command-line interface
- File-format
- **Numpy** array serialization
- Pure-python based on Python-Blosc

More Code, Yo!

```
>>> import bloscpack, numpy
>>> b = numpy.arange(5e6)
>>> c = bloscpack.pack_ndarray_str(b)
>>> len(b)/len(c)
15.359992135684026

>>> c = bloscpack.pack_ndarray_str(b,
... blosc_args=bloscpack.BloscArgs(clevel=9, cname='lz4'))
>>> len(b)/len(c)
17.693979092794304
```

And Back Again

```
>>> d = bloscpack.unpack_ndarray_str(c)
>>> assert (b == d).all()
```

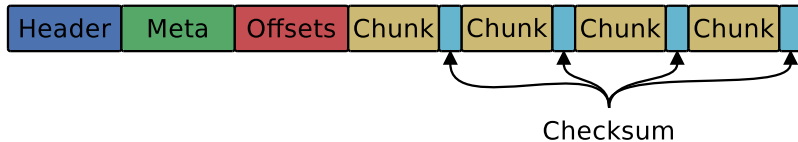
Command-line Usability

```
>>> np.save('b.npy', b)

$ ./blpk compress b.npy
$ ./blpk compress -l 9 -c lz4 b.npy b.npy.lv9lz4.blp
$ ls -lh b.*
-rw----- 1 esc esc 39M May 13 21:50 b.npy
-rw----- 1 esc esc 319K May 13 21:52 b.npy.blp
-rw----- 1 esc esc 277K May 17 20:39 b.npy.lv9lz4.blp
```

Format

- A **simple** file format
- Optional
 - **Checksum**
 - **Metadata**
 - **Offsets**



Numpy Support

- Python-Blosc can also pack an array:

```
>>> b = numpy.arange(5e6)
>>> %timeit c = blosc.pack_array(b)
10 loops, best of 3: 40.2 ms per loop

>>> %timeit c = bloscpack.pack_ndarray_str(b)
100 loops, best of 3: 9.81 ms per loop
```

- Internal copy vs. pointer operation
- Numpy metadata is stored in metadata section

```
>>> from io import BytesIO
>>> %timeit bio=BytesIO(); np.save(bio, b)
...
```

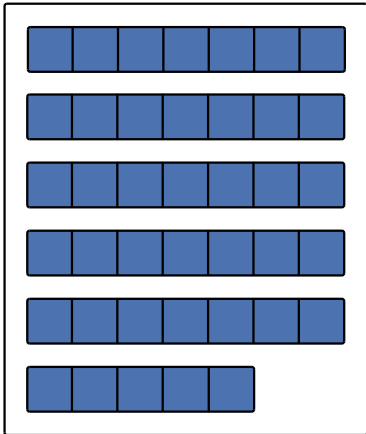
- `pack_ndarray_str` \leftarrow `str`, really?!
- Py3 support not merged :(

Outline

- 1 Blosc
- 2 Python-Blosc
- 3 Bloscpack
- 4 Bcolz

- Chunked container
- In-memory and out-of-core
- `carray` and `ctable`
- Good for **medium data**
- Support basic analytics
- Uses Blosc and Bloscpack under the hood
- Python/Cython

Layout of the carray



- Discontiguous
- Blosc-compressed
- Homogeneously typed
- **Chunks**

Creating a carray

```
>>> b = bcolz.arange(5e6)
>>> b
carray((5000000,), float64)
  nbytes: 38.15 MB; cbytes: 1.45 MB; ratio: 26.33
  cparams := cparams(clevel=5, shuffle=True, cname='blosclz')
[ 0.00000000e+00  1.00000000e+00  2.00000000e+00 ...,  4.99999700e+06
 4.99999800e+06  4.99999900e+06]
```

The chunks

```
>>> b.chunks[0]
chunk(float64)  nbytes: 524288; cbytes: 18698; ratio: 28.04
'[ 0.00000000e+00  1.00000000e+00  2.00000000e+00 ...,  6.55330000e+04
 6.55340000e+04  6.55350000e+04]'
>>> b.chunks[0].to_numpy_array() # currently available as pull-request only
array([ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00, ...,
        6.55330000e+04,  6.55340000e+04,  6.55350000e+04])
>>> type b.chunks
list
```

Variation

```
>>> b = bcolz.arange(5e6, cparams=bcolz.cparams(clevel=9, cname='lz4'))

>>> b
carray((5000000,), float64)
  nbytes: 38.15 MB; cbytes: 849.54 KB; ratio: 45.98
  cparams := cparams(clevel=9, shuffle=True, cname='lz4')
[ 0.00000000e+00  1.00000000e+00  2.00000000e+00 ...,  4.99999700e+06
 4.99999800e+06  4.99999900e+06]
```

On-Disk

```
>>> b = bcolz.arange(5e6, rootdir='bexample', chunklen=1000**2)
```

```
>>> ls -lh bexample/*
```

```
-rw----- 1 esc esc    3 May 17 22:19 bexample/__attrs__
```

```
bexample/data:
```

```
total 992K
```

```
-rw----- 1 esc esc 168K May 17 22:19 __0.blp
```

```
-rw----- 1 esc esc 138K May 17 22:19 __1.blp
```

```
-rw----- 1 esc esc 190K May 20 19:29 __2.blp
```

```
-rw----- 1 esc esc 195K May 20 19:29 __3.blp
```

```
-rw----- 1 esc esc 286K May 20 19:29 __4.blp
```

```
bexample/meta:
```

```
total 16K
```

```
-rw----- 1 esc esc   60 May 17 22:19 sizes
```

```
-rw----- 1 esc esc  122 May 17 22:19 storage
```

- *.blp ← Bloscpack files

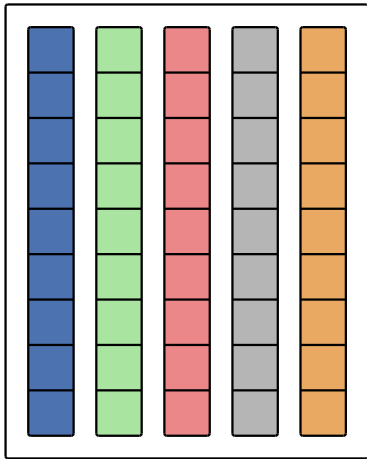
Compute: Witness the power of numexpr

- Uses numexpr under the hood for computation
- ... because numexpr can do **blocking** on **chunks**

```
>>> a = bcolz.arange(2e8, rootdir='a')
>>> b = bcolz.arange(2e8, rootdir='b')
>>> c = bcolz.eval('a**2 + 2 * b', rootdir='c')
>>> c
carray((200000000,), float64)
  nbytes: 1.49 GB; cbytes: 386.22 MB; ratio: 3.95
  cparams := cparams(clevel=5, shuffle=True, cname='blosclz')
  rootdir := 'c'
  mode    := 'a'
[ 0.00000000e+00  3.00000000e+00  8.00000000e+00 ...,  3.99999992e+16
 3.99999996e+16  4.00000000e+16]
```

- Both a and b are about 1.5GB uncompressed

Layout of the ctable



- Columnar
- Heterogeneously typed
- **Columns**
- Every column \rightarrow carray

Philosophy of Bcolz

- Keep it simple
 - Expose a Cython interface to `carray`
 - Layer everything else on-top
-
- **bquery** \leftarrow out of core groupby
 - **dask** \leftarrow out-of-core abstraction
 - **odo** \leftarrow convert from one format to another

Whats coming (maybe)

- Networked storage, e.g. S3
- Bcolz based server / database
- Better Cython interface
- True n-dimensionality (really?)

Summary (I)

- Everyone always asks me about the relationship of our toolstack

Summary (II)

- **Blosc**
 - The codec
- **Python-Blosc**
 - The Python bindings
- **Bloscpack**
 - Serialization format
 - Command-line interface
 - Fast Numpy serializer
- **Bcolz**
 - Compressed container(s)
 - In-memory and out-of-core
 - Analytics engine

Summary (III)

- **Bcolz** uses a vendored, older version of Bloscpack
- **Bcolz** interfaces with Blosc using it's own Cython bindings (not Python-Blosc)

Famous last words

- Links:
 - <http://blosc.org>
 - <https://github.com/blosc>
 - <https://github.com/esc/blosc-and-friends>
- Open source tools used to make this presentation:
 - Wiki2beamer
 - L^AT_EXbeamer
 - Dia
 - Pygments
 - Minted
 - Solarized theme for pygments