Compress Me, Stupid!

Valentin Haenel

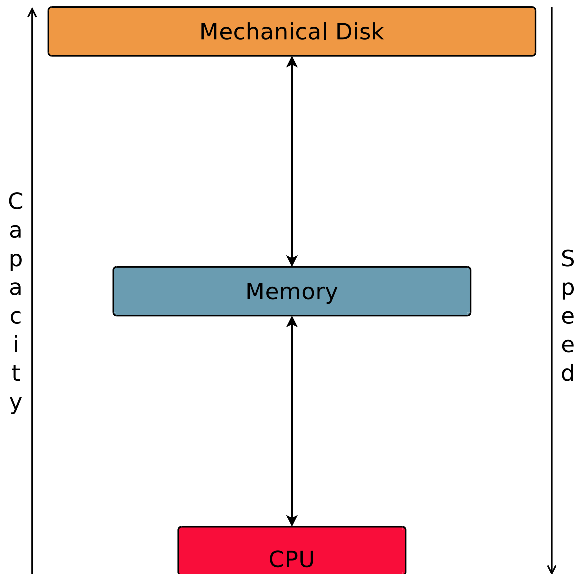
Freelance Consultant and Software Developer
@esc___

23 July 2014 - EuroPython Berlin (EP14)

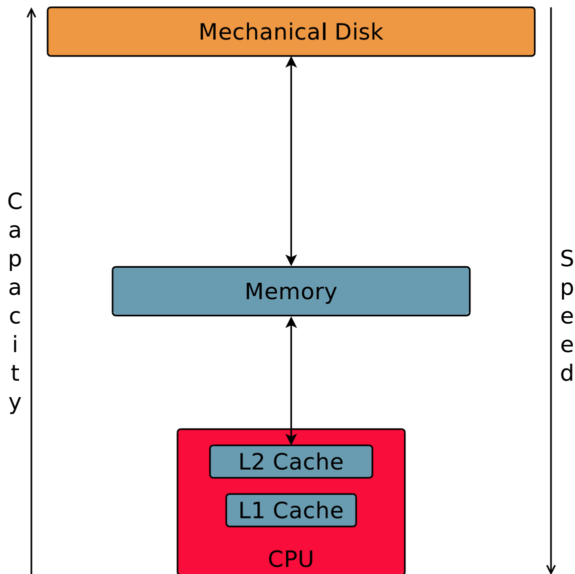
Version: 2014-EuroPython https://github.com/esc/compress-me-stupid
This work is licensed under the *Creative Commons Attribution-ShareAlike 3.0 License*.

A Historical Perspective

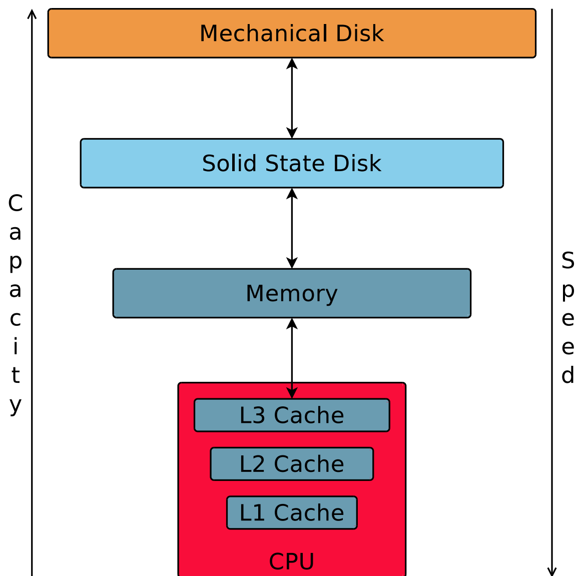
The Memory Hierarchy – Up to end of 80's



The Memory Hierarchy – 90's and 2000's



The Memory Hierarchy – 2010's



Starving CPUs

The Status of CPU Starvation in 2014:

- Memory latency is much slower (between 100x and 500x) than processors.
- Memory bandwidth is improving at a better rate than memory latency, but it is also slower than processors (between 30x and 100x).
- Net effect: CPUs are often waiting for data

It's the memory, Stupid

Problem: *It's the memory, Stupid!* [1]

Solution: *Compress me, Stupid!*

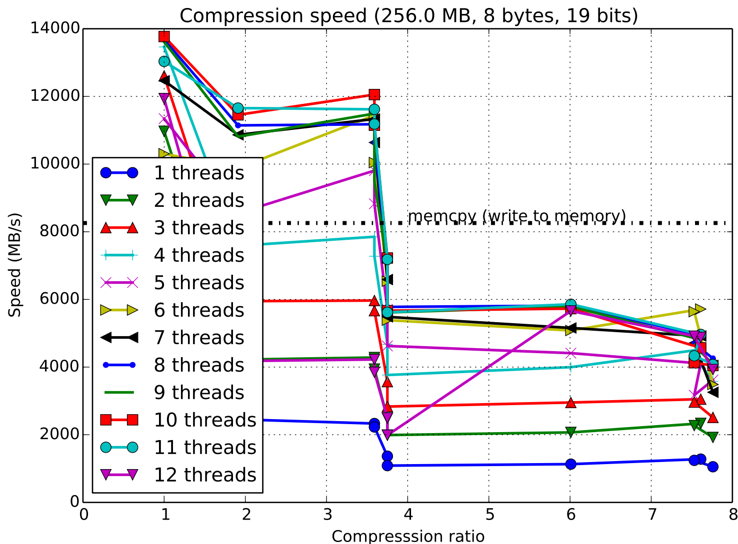
[1] R. Sites. It's the memory, stupid! MicroprocessorReport, 10(10),1996

Blosc

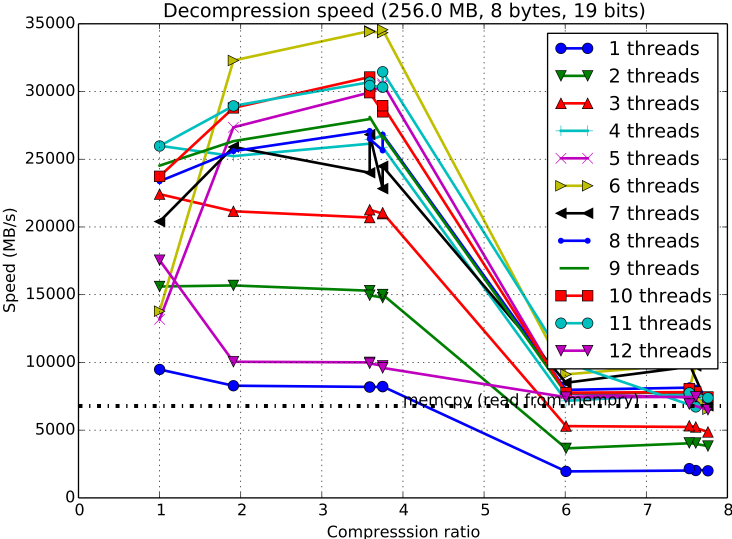
Blosc

- Designed for: in-memory compression
- Addresses: the starving CPU Problem
- (In fact, it also works well in general purpose scenarios)
- Written in: C

Faster-than-memcpy



Faster-than-memcpy

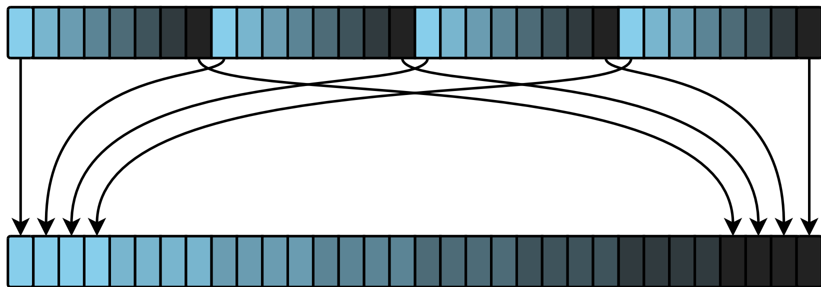


Blosc is a Metacodec

- Blosc does not actually compress anything
  - *Cutting* data into blocks
  - Application of filters
  - Management of threads
- Can use ‘real’ codecs under the hood.
- Filters and codecs are applied to each block (blocking)
- Thread-level parallelism on blocks

Shuffle Filter

- Reorganization of bytes within a block
- Reorder by byte significance



Shuffle Filter Example – Setup

Imagine we have the following array as `uint64` (8 byte, unsigned integer):

```
[0, 1, 2, 3]
```

Reinterpret this as `uint8`:

```
[0, 0, 0, 0, 0, 0, 0, 0,
 1, 0, 0, 0, 0, 0, 0, 0,
 2, 0, 0, 0, 0, 0, 0, 0,
 3, 0, 0, 0, 0, 0, 0, 0]
```

Shuffle Filter Example – Application

What the shuffle filter does is:

```
[0, 1, 2, 3, 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, 0, 0, 0]
```

Which, reinterpreted as `uint64` is:

```
[50462976,          0,          0,          0]
```

Shuffle Filter Benefits

- Works well for multibyte data with small differences
  - e.g. Timeseries
- Exploit similarity between elements
- Lump together bytes that are alike
- Create longer streams of similar bytes
- Better for compression
- Shuffle filter implemented using SSE2 instructions

Shuffle Fail

It does not work well on all datasets, observe:

```
[18446744073709551615, 0, 0, 0]
```

Or, as `uint8`:

```
[255, 255, 255, 255, 255, 255, 255, 255,
 0,   0,   0,   0,   0,   0,   0,   0,
 0,   0,   0,   0,   0,   0,   0,   0,
 0,   0,   0,   0,   0,   0,   0,   0]
```

Shuffle Fail in Action

When shuffled yields:

```
[1095216660735, 1095216660735,
 1095216660735, 1095216660735]
```

Or, as `uint8`:

```
[255,   0,   0,   0, 255,   0,   0,   0,
 255,   0,   0,   0, 255,   0,   0,   0,
 255,   0,   0,   0, 255,   0,   0,   0,
 255,   0,   0,   0, 255,   0,   0,   0]
```

OK, so what else is *under the hood*?

- By default it uses **Blosclz** – derived from **Fastlz**
- Alternative codecs
  - **LZ4 / LZ4HC**
  - **Snappy**
  - **Zlib**

Support for other codecs (LZO, LZF, QuickLZ, LZMA) possible, but needs to be implemented.

Blosc + X

So... using Blosc + X can yield **higher compression ratios** using the shuffle filter and **faster compression/decompression** time using multithreading.

That's pretty neat!

Python-Blosc

Python API

- It's a codec
  - Naturally we have a `compress`/`decompress` pair
- Can operate on byte strings or pointers (encoded as integers)
  - `compress` vs. `compress_ptr`
- Tutorials
  - `http://python-blosc.blosc.org/tutorial.html`
- API documentation
  - `http://python-blosc.blosc.org/`
- Implemented as a C-extension using the Python-C-API

Example – Setup

```
>>> import numpy as np
>>> import blosc
>>> import zlib

>>> bytes_array = np.linspace(0, 100, 1e7).tostring()
>>> len(bytes_array)
80000000
```

Example – Compress

```
>>> %timeit zpacked = zlib.compress(bytes_array, 9)
1 loops, best of 3: 14.7 s per loop

>>> %timeit bzpacked = blosc.compress(bytes_array,
...                                   typesize=8,
...                                   cname='zlib',
...                                   clevel=9)
1 loops, best of 3: 317 ms per loop
```

Example – Ratio

```
>>> zpacked = zlib.compress(bytes_array, 9)
>>> len(zpacked)
52945925

>>> bzpacked = blosc.compress(bytes_array,
...                           typesize=8,
...                           cname='zlib',
...                           clevel=9)
>>> len(bpacked)
1011304

>>> len(bytes_array) / len(zpacked)
1.5109755849954458
>>> len(bytes_array) / len(bzpacked)
79.10578817052044
>>> len(zpacked) / len(bzpacked)
52.35411409427828
```

Example – Decompress

```
>>> %timeit zupacked = zlib.decompress(zpacked)
1 loops, best of 3: 388 ms per loop

>>> %timeit bupacked = blosc.decompress(bzpacked)
10 loops, best of 3: 76.2 ms per loop
```

Example – Demystified

- Blosc works really well for the `linspace` dataset
- Shuffle filter and multithreading bring benefits

Example – Speed Demystified

- Use a single thread and deactivate the shuffle filter

```
>>> blosc.set_nthreads(1)
>>> %timeit bzpacked = blosc.compress(bytes_array,
...                                   typesize=8,
...                                   cname='zlib',
...                                   clevel=9,
...                                   shuffle=False)
1 loops, best of 3: 12.9 s per loop
```

Example – Ratio Demystified

```
>>> bzpacked = blosc.compress(bytes_array,
...                           typesize=8,
...                           cname='zlib',
...                           clevel=9,
...                           shuffle=False)
>>> len(zpacked) / len(bzpacked)
0.9996947439311876
```

So, What about other Codecs? – Compress

- Zlib implements a comparatively slow algorithm (DEFLATE), let's try LZ4

```
>>> %timeit bzpacked = blosc.compress(bytes_array,
...                                   typesize=8,
...                                   cname='zlib',
...                                   clevel=9)
1 loops, best of 3: 329 ms per loop

>>> %timeit blpacked = blosc.compress(bytes_array,
...                                   typesize=8,
...                                   cname='lz4',
...                                   clevel=9)
10 loops, best of 3: 20.9 ms per loop
```

So, What about other Codecs? – Ratio

- Although this speed increase comes at the cost of compression ratio

```
>>> bzpacked = blosc.compress(bytes_array,
...                           typesize=8,
...                           cname='zlib',
...                           clevel=9)
>>> blpacked = blosc.compress(bytes_array,
...                           typesize=8,
...                           cname='lz4',
...                           clevel=9)
>>> len(bzpacked) / len(blpacked)
0.172963927766
```

So, What about other Codecs? – Decompress

```
>>> %timeit bzupacked = blosc.decompress(bzpacked)
10 loops, best of 3: 74.3 ms per loop

>>> %timeit blupacked = blosc.decompress(blpacked)
10 loops, best of 3: 25.3 ms per loop
```

C-extension Notes

- Uses `_PyBytesResize` to resize a string after compressing into it
- Release the GIL before compression and decompression.

Installation and Compilation

Installation via Package – PyPi/pip

Using `pip` (inside a virtualenv):

```
$ pip install blosc
```

Provided you have a `C++` (not just `C`) compiler..

Installation via Package – binstar/conda

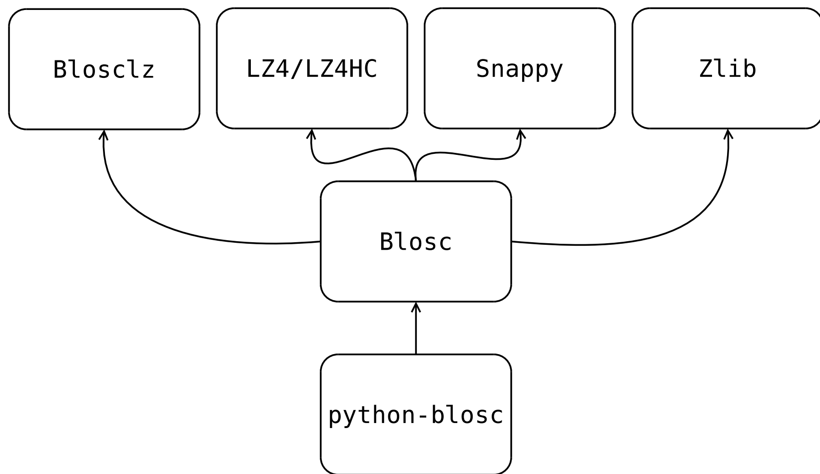
Using `conda`:

```
$ conda install -c https://conda.binstar.org/esc python-blo
```

Experimental, Numpy 1.8 / Python 2.7 only..

Compilation / Packaging

Blosc is a metacodec and as such has various dependencies



Compilation / Packaging – Flexibility is Everything

- Blosc uses CMake and ships with all codec sources
  - Try to link against existing codec library
  - If not found, use shipped sources
- Python-Blosc comes with Blosc sources
  - Compile everything into Python module
  - Or link against Blosc library
- Should be beneficial for packagers

Outro

Other Projects that use Blosc

- **PyTables** HDF Library
- **Bloscpack** Simple file-format and Python implementation
- **bcolz** In-memory and out-of-core compressed array-like structure

The Future

- What might be coming...
  - More codecs
  - Alternative filters
  - Auto-tune at runtime
  - Multi-shuffle
  - A Go implementation
- How can I help?
  - Run the benchmarks on your hardware, report the results
  - `http://blosc.org/synthetic-benchmarks.html`
  - Incorporate Blosc into your application

Advertisment

- EuroPython
  - Francecs Alted - Out of Core Columnar Datasets - Friday 11:00 C01
- PyData Berlin
  - Francecs Alted - Data Oriented Programming - Saturday 13:30 B05
  - Valentin Haenel - Fast Serialization of Numpy Arrays with Bloscpack - Sunday 11:00 am B05

Getting In Touch

- Main website: `http://blosc.org`
- Github organization: `http://github.com/Blosc`
- python-bloc: `http://github.com/Blosc/python-blosc`
- Google group: `https://groups.google.com/forum/#!forum/blosc`
- This talk: `https://github.com/esc/compress-me-stupid`