# Starving CPUs
(and coping with that in PyTables)

Francesc Alted[1]

[1]Freelance developer and PyTables creator

Rijnhuizen, September 24th, Nieuwegein - The Netherlands

## Outline

## Outline

## First Commodity Processors
(Early and middle 1980s)

- Processors and memory evolved more or less in step.
- Memory clock access in early 1980s was at ~ 1MHz, the same speed than CPUs.

## Intel 8086, 80286 and i386
(Middle and late 1980's)

- Memory still pretty well matched CPU speed.
- The 16MHz i386 came out; memory still could keep up with it.

## Intel i486 and AMD Am486 (Early 1990s)

- Increases in memory speed started to stagnate, while CPU clock rates continued to skyrocket to 100 MHz and beyond.
- In a single-clock, a 100 MHz processor consumes a word from memory every 10 nsec. This rate is impossible to sustain even with *present-day* RAM.
- The first on-chip cache appeared (8 KB for i486 and 16 Kb for i486 DX).

## Intel Pentium and AMD K5/K6
(Middle and late 1990s)

- Processor speeds reached unparalleled extremes, before hitting the magic 1 GHz figure.
- A huge abyss opened between the processors and the memory subsystem: up to 50 wait states for each memory read or write.

## Intel Pentium 4 and AMD Athlon
(Early and middle 2000s)

- The strong competition between Intel and AMD continued to drive CPU clock cycles faster and faster (up to 0.25 ns, or 4 GHz).
- The increased impedance mismatch with memory speeds brought about the introduction of a second level cache.

## Intel Core2 and AMD Athlon X2
(Middle 2000s)

- The size of integrated caches is getting really huge (up to 12 MB).
- Chip makers realized that they can't keep raising the frequency forever $\rightarrow$ enter the multi-core age.
- Users start to scratch their heads, wondering how to take advantage of multi-core machines.

## Intel Core2 and AMD Athlon X2
(Middle 2000s)

- The size of integrated caches is getting really huge (up to 12 MB).
- Chip makers realized that they can't keep raising the frequency forever → enter the multi-core age.
- Users start to scratch their heads, wondering how to take advantage of multi-core machines.

## Intel Core i7 and AMD Phenom (Late 2000s)

- 4-core on-chip CPUs become the most common configuration.
- 3-levels of on-chip cache is the standard now.

## Where we are now
(2009)

- Memory latency is much slower (around 150x) than processors and has been an essential bottleneck for the past fifteen years.
- Memory throughput is improving at a better rate than memory latency, but it is also lagging behind processors (about 25x slower).
- In order to achieve better performance, CPU makers are implementing additional levels of caches, as well as increasing cache size.
- Recently, CPU speeds have stalled as well, limited now by power dissipation problems. So, in order to be able to offer more speed, CPU vendors are packaging several processors (cores) in the same die.

## Outline

## The CPU Starvation Problem

- Over the last 25 years CPUs have undergone an exponential improvement on their ability to perform massive numbers of calculations extremely quickly.
- However, the memory subsystem hasn't kept up with CPU evolution.
- The result is that CPUs in our current computers are suffering from a serious starvation data problem: *they could consume (much!) more data than the system can possibly deliver.*

## Can't Memory Latency Be Reduced to Keep Up with CPUs?

- To improve latency figures, we would need:
  - more wire layers
  - more complex ancillary logic
  - more frequency (and voltage):

    $$Energy = Capacity \; x \; Voltage^2 \; x \; Frequency$$
- That's too expensive for commodity SDRAM.

## What Is the Industry Doing to Alleviate CPU Starvation?

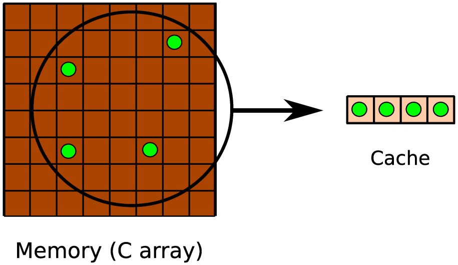
- They are improving memory throughput: cheap to implement (more data is transmitted on each clock cycle).
- They are adding big caches in the CPU dies.

## Why Is a Cache Useful?

- Caches are closer to the processor (normally in the same die), so both the latency and throughput are improved.
- However: the faster they run the smaller they must be.
- They are effective mainly in a couple of scenarios:
    - Time locality: when the dataset is reused.
    - Spatial locality: when the dataset is accessed sequentially.

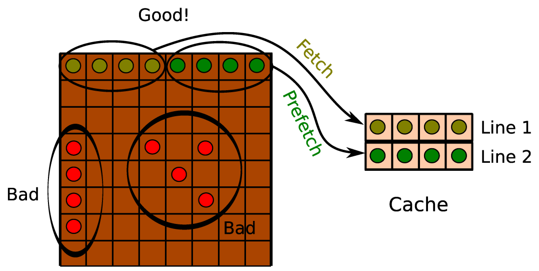
## Time Locality

Parts of the dataset are reused



## Spatial Locality

Dataset is accessed sequentially



Memory (C array)

## Outline

## Once Upon A Time...

- In the 1970s and 1980s many computer scientists had to learn assembly language in order to squeeze all the performance out of their processors.
- In the good old days, the processor was the key bottleneck.

## Nowadays...

- Every computer scientist must acquire a good knowledge of the hierarchical memory model (and its implications) if they want their applications to run at a decent speed (i.e. they do not want their CPUs to starve too much).
- Memory organization has become now the key factor for optimizing.

**The BIG difference is. . .**

. . . learning assembly language is relatively easy, but understanding how the hierarchical memory model works requires a considerable amount of experience (it's almost more an art than a science!)

## Nowadays...

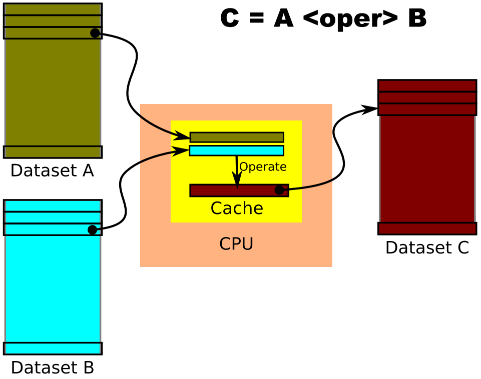
- Every computer scientist must acquire a good knowledge of the hierarchical memory model (and its implications) if they want their applications to run at a decent speed (i.e. they do not want their CPUs to starve too much).
- Memory organization has become now the key factor for optimizing.

**The BIG difference is. . .**

. . . learning assembly language is relatively easy, but understanding how the hierarchical memory model works requires a considerable amount of experience (it's almost more an art than a science!)

## The Blocking Technique I

When you have to access memory, get a contiguous block that fits in the CPU cache, operate upon it or reuse it as much as possible, then write the block back to memory:



## The Blocking Technique II

- This is not new at all: it has been in use for out-of-core computations since the dawn of computers.
- However, the meaning of *out-of-core* is changing, since the *core* does not refer to the main memory anymore: it now means something more like *out-of-cache*.
- Although this technique is easy to apply in some cases (e.g. element-wise array computations), it can be potentially difficult to *efficiently* implement in others.

**Good News!**

Fortunately, many useful algorithms using blocking have been developed by others that you can use ☺

## The Blocking Technique II

- This is not new at all: it has been in use for out-of-core computations since the dawn of computers.
- However, the meaning of *out-of-core* is changing, since the *core* does not refer to the main memory anymore: it now means something more like *out-of-cache*.
- Although this technique is easy to apply in some cases (e.g. element-wise array computations), it can be potentially difficult to *efficiently* implement in others.

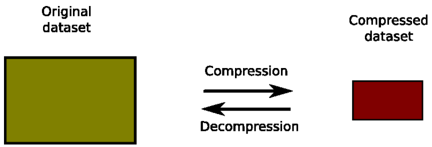
**Good News!**

Fortunately, many useful algorithms using blocking have been developed by others that you can use ☺

## Outline

## The Compression Process

- A compression algorithm looks in the dataset for redundancies and dedups them. The usual outcome is a smaller dataset:



## The Role Of Compression In Data Access

- Compression has *already* helped accelerate reading and writing large datasets from/to disks over the last 10 years.
- It generally takes less time to read/write a small (compressed) dataset than a larger (uncompressed) one, even taking into account the (de-)compression times.

**Crazy question:**

Given the gap between processors and memory speed, could compression accelerate the transfer from memory to the processor, also?

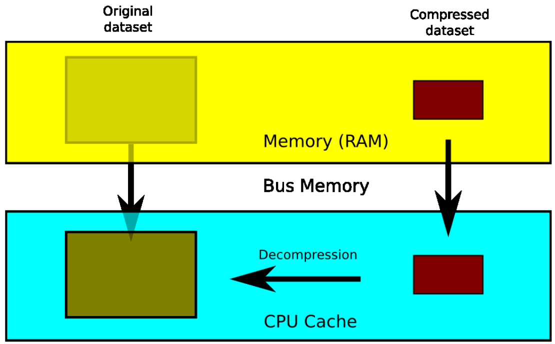
## The Role Of Compression In Data Access

- Compression has *already* helped accelerate reading and writing large datasets from/to disks over the last 10 years.
- It generally takes less time to read/write a small (compressed) dataset than a larger (uncompressed) one, even taking into account the (de-)compression times.

**Crazy question:**

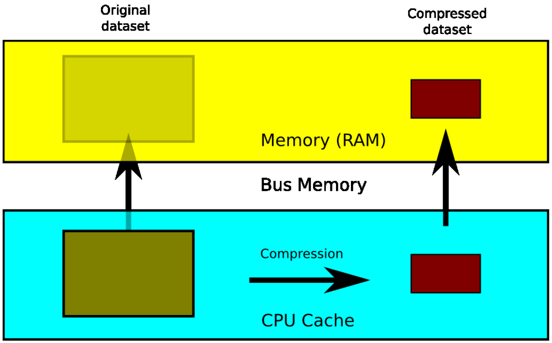
Given the gap between processors and memory speed, could compression accelerate the transfer from memory to the processor, also?

## Reading Compressed Datasets



Transmission + decompression processes faster than direct transfer?

## Writing Compressed Datasets



Compression + transmission processes faster than direct transfer?

## The Challenge: Faster Memory I/O By Using Compression?

**What we need:**

Extremely fast compressors/decompressors.

**What we should renounce:**

High compression ratios.

## The Challenge: Faster Memory I/O By Using Compression?

**What we need:**

Extremely fast compressors/decompressors.

**What we should renounce:**

High compression ratios.

## Applications for Fast In-Memory Compression

- We could store more data in a given amount of RAM.
- When a large quantity of data needs to be accessed sequentially, access time could be reduced (if compression is fast enough).

**Not good for random access. . .**

To get a single word, you would need to uncompress an entire compressed block.

**But. . .**

Many algorithms out there have already been blocked: it should be easy to implement compression for them.

## Applications for Fast In-Memory Compression

- We could store more data in a given amount of RAM.
- When a large quantity of data needs to be accessed sequentially, access time could be reduced (if compression is fast enough).

**Not good for random access. . .**

To get a single word, you would need to uncompress an entire compressed block.

**But. . .**

Many algorithms out there have already been blocked: it should be easy to implement compression for them.

## Applications for Fast In-Memory Compression

- We could store more data in a given amount of RAM.
- When a large quantity of data needs to be accessed sequentially, access time could be reduced (if compression is fast enough).

**Not good for random access. . .**

To get a single word, you would need to uncompress an entire compressed block.

**But. . .**

Many algorithms out there have already been blocked: it should be easy to implement compression for them.

## The Current State Of Compressors

- Generally speaking, current compressors do not yet achieve speeds that would allow programs to handle compressed datasets faster than uncompressed data.
- As CPUs have become faster, the trend has been to shoot for high compression ratios, and not so much to reach faster speeds.
- There are some notable exceptions like LZO, LZF and FastLZ, which are very fast compressor/decompressors, but they're still not fast enough to hit our goal.

We need something better!

## The Current State Of Compressors

- Generally speaking, current compressors do not yet achieve speeds that would allow programs to handle compressed datasets faster than uncompressed data.
- As CPUs have become faster, the trend has been to shoot for high compression ratios, and not so much to reach faster speeds.
- There are some notable exceptions like LZO, LZF and FastLZ, which are very fast compressor/decompressors, but they're still not fast enough to hit our goal.

We need something better!

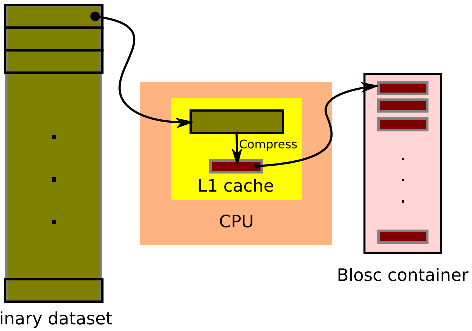
## Outline

## Blosc: A BLOcking, Shuffling & Compression Library

- Blosc is a new, lossless compressor for binary data. It's optimized for speed, not for high compression ratios.
- It is based on the FastLZ compressor, but with some additional tweakings:
  - It works by splitting the input dataset into blocks that fit well into the level 1 cache of modern processors.
  - It can shuffle bytes very efficiently for improved compression ratios (using the data type size metainformation).
  - Makes use of SSE2 vector instructions (if available).
- Free software (MIT license).

## Blocking: Divide And Conquer

Blosc achieves very high speeds by making use of the well-known blocking technique:



## Pros And Cons Of Blocking

**Very fast**

Compresses/decompresses at L1 cache speeds.

**Lesser compression ratio**

The block is the maximum extent in which redundant data can be identified and de-dup'd.

## Pros And Cons Of Blocking

**Very fast**

Compresses/decompresses at L1 cache speeds.

**Lesser compression ratio**

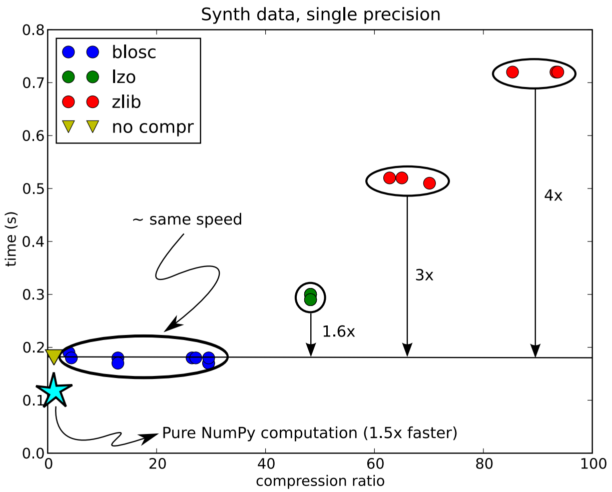
The block is the maximum extent in which redundant data can be identified and de-dup'd.

## Outline

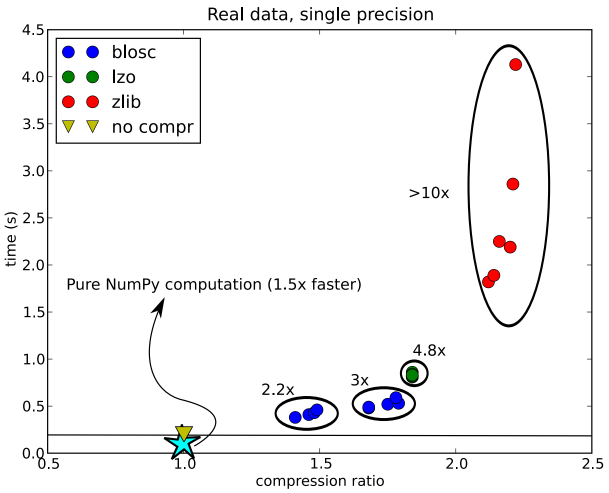
## Blosc and In-Memory Datasets

- Several benchmarks have been conducted in order to analyze how Blosc performs in comparison with other compressors when data is *in-memory*.
- The benchmarks consist in reading a couple of datasets from OS filesystem cache, operating upon them and writing the result to the filesystem cache again.
- Datasets analyzed are synthetic (low entropy, so highly compressibles) and real-life (medium/high entropy, difficult to compress well), in both single and double-precision versions.
- Synthetic datasets do represent important corner use cases: sparse matrices, regular grids. . .

## Performing A Certain Computation (Synth Data)



## Performing A Certain Computation (Real Data)



## Outline

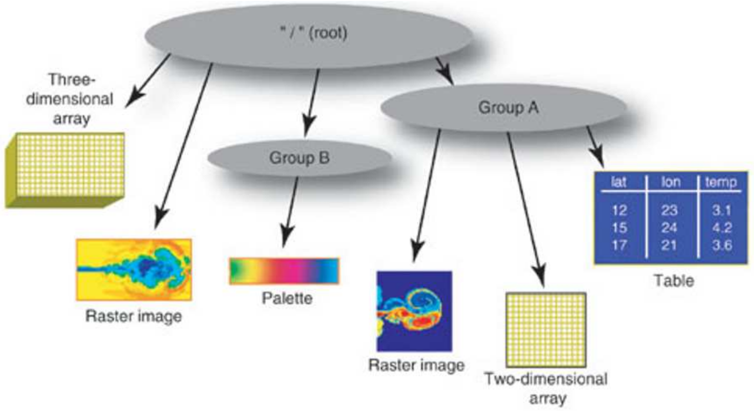
## Motivation

- Many applications need to save and read very large amounts of data. Coping with this is a real challenge!
- Most computers today can deal with such large datasets. However, we should ask for an interface that should be usable by *human beings*.
- Requirements:
  - **Interactivity**: data analysis is an iterative process.
  - Need to re-read many times the data: **efficiency**.
  - Easy **categorization** of data.
  - Ability to **keep data for long time**.

## What Does PyTables Offer?

| | |
|---|---|
| Interactivity | You can take immediate action based on previous feedback. |
| Efficiency | Improves your productivity (very important for interactive work). |
| Hierarchical structure | It allows you to categorize your data into smaller, related chunks. |
| Backward/Forward compatibility | Based on HDF5, a general purpose framework with a great commitment with backward/forward compatibility. |

## Example Of Hierachical Structure



| lat | lon | temp |
|---|---|---|
| 12 | 23 | 3.1 |
| 15 | 24 | 4.2 |
| 17 | 21 | 3.6 |

## PyTables Highlights (I)

- High level of flexibility for structuring your data:
  - Datatypes: scalars (numerical & strings), records, enumerated, time...
  - Tables support multidimensional cells
  - Tables support nested records
  - Mutidimensional arrays
  - Variable length arrays

## PyTables Highlights (II)

- Transparent data compression support (Zlib, LZO, Bzip2...).
- Support of full 64-bit addressing in files, even on 32-bit platforms.
- Can handle generic HDF5 files (most of them).
- Aware of little/big endian issues (data is portable).

## Easy Of Use

### Natural naming

```
# access to file:/group1/table
table = file.root.group1.table
```

### Support for generalized and fancy indexing

```
array[idx, start:stop, :, start:stop:step]     # hyperslicing
array[1, [1,5,10], ..., -1]       # sparse reads (since 2.2)
```

### Support for iterators

```
# get the values in col1 that satisfy a certain condition
[r['col1'] for r in table.where((1.3 < col3) & (col2 <= 2.))]
```

## How PyTables Fights CPU Starvation?

Basically, by applying blocking techniques and by leveraging high performance packages like:

| | |
|---|---|
| HDF5 | A library & format thought out for managing very large datasets in an efficient way. |
| NumPy | A Python package for handling large homogeneous and heterogeneous datasets. |
| Numexpr | Increase the performance of NumPy in complex operations by applying blocking. |
| Blosc | A high-performance compressor meant for binary data (available in the short future). |

## PyTables Pro

It is an enhanced version of PyTables. It sports:

**Column indexing** Queries in tables having up to 1 billion rows can be typically done in less than 1 second.

**Customizable index quality** The indexes can be created with an optimization level (specified as a number ranging from 0 to 9).

**Improved cache system** for both metadata and regular data. Allows for maximum speed during intensive node browsing and data queries.

## Outline

## Some PyTables Use Cases

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Thomas Jefferson National Accelerator Facility
Jefferson Lab

CALTECH

university of groningen

UNIVERSITY OF Southampton

MAX-PLANCK-GESELLSCHAFT

OBJEXX ENGINEERING

FOM Institute for Plasma Physics Rijnhuizen

NOAA

## Risk Analysis In Dam Safety

- We need a database capable of dealing with several GB of information and PyTables is working great for us.
- Best points:
  - Easy integration of PyTables with our existing code.
  - Speed.
  - Seamlessly support for compressed data.
  - Great user support.

– *Armando Serrano*

Institute of Water Engineering and Environment

## Finite Element Solver For Micromagnetic Problems

- We do not know at coding time what type of data the user may use and how often they decide to save the data, so flexibility is crucial. PyTables provides just that flexibility.
- Best points:
  - We get very significant space savings when saving field data.
  - Inbuilt compression reduces memory consumption significantly without slowing the process down.
  - Allows to quickly write code that saves complicated and hard-to-predict data structures.

– *Hans Fangohr*

School of Engineering Sciences

UNIVERSITY OF Southampton

## Atmospheric sciences

- PyTables is used in the assimilation of radar data into storm-resolving models to produce 3D analysis of severe storms for research and potentially forecasts.
- Best points:
  - Fast data searches.
  - Allow us doing things like only assimilating certain variables at certain levels of the atmosphere, or changing what variables are assimilated and when.
  - I liked the API.

– *Louis J. Wicker*

NOAA National Severe Storms Laboratory in Norman Oklahoma

## Neuroscience And Behavioural Science

- I save raw neuroelectrical signals and filtered versions of them in a HDF5 file; I use tables to register *events* (i.e. where in the data there are stimuli, which can be tens of thousands).
- Best points:
  - Ability to work with large numeric arrays (8 hours of recording 32 channels at 14000 Hz).
  - The entire database fits in a single file: easy to manage.
  - Flexible queries.

– *Gabriel J.L. Beckers*

Ornithology Institute, Department of Behavioural Neurobiology

MAX-PLANCK-GESELLSCHAFT

## Operations Research

- I store results of numerical experiments. I also use it to store the state spaces of large markov chains.
- Best points:
    - Format is free.
    - Supports metadata for annotation purposes.
    - One single file is enough for carrying all the data.

*– Nicky Van Foreest*

University of Groningen

## Multi-Camera Tracking Of Flying Flies

- We save our data as PyTables files as a matter of course. This is typically 3 GB/day/experiment.
- Best points:
  - Its integration with NumPy and its fast searching set it apart from other possible solutions.
  - The ease and speed to analyze data interactively far surpasses other systems I've worked with.
  - Extremely responsive development team in responding to bug reports and feature requests.

– *Andrew D. Straw*

California Institute of Technology

CALTECH

## Nuclear Fuel Rod FEM Modeling

- PyTables is used to create the file containing the model geometry/numbering specifications. Our Python post-processor uses PyTables to extract the table data for visualization.
- Best points:
  - Easy to use.
  - Best HDF5 interface for our needs.
  - Simple to interface with Fortran legacy applications via HDF5 files.

*– Stuart Mentzer*

Objexx Engineering, Inc.

## Particle Physics

- I store the relevant information from these events in a PyTables file where I can easily make additional cuts or make plots with Matplotlib.
- Best points:
  - PyTables, NumPy, SciPy, and Matplotlib are together my replacement for ROOT.
  - Possibility to use variable-length arrays (as well as fixed-length ones).
  - I have far more confidence in PyTables than I ever did in ROOT's TTree libraries.

*– DEMOLISHOR!*

Thomas Jefferson National Accelerator Facility

## Plasma Physics

- I'm writing a database implementation where PyTables is used in concert with Ice to provide an easy-to-use interface to large amounts of data.
- Best points:
  - Open Source.
  - Provides the necessary performance.
  - Support for continuous growing datasets.

– *Han Genuit*

FOM-Institute for Plasma Physics Rijnhuizen

FOM Institute for Plasma Physics Rijnhuizen

## Final Words

- Newer processors will surely improve uncompressed data processing, but the memory access bottleneck issue will prevent users from seeing much improvement in performance.
- Extremely fast compressors will be able to effectively accelerate in-memory computations, allowing a much more effective use of the CPU and memory resources.
- PyTables uses a series of techniques, including blocking and compression, so as to alleviate CPU starvation. This allows to store and process extremely large datasets more effectively.

## More Info

Ulrich Drepper
What Every Programmer Should Know About Memory
RedHat Inc.,2007

PyTables: Hierarchical Datasets in Python
http://www.pytables.org

Blosc: A blocking, shuffling and lossless compression library
http://blosc.pytables.org

## Thank You!

Questions?