Large Data Analysis with Python

Francesc Alted

Freelance Developer and PyTables Creator

G-Node
November 24th, 2010. Munich, Germany

Where do I live?

Where do I live?

Some Words About PyTables

- Started as a solo project back in 2002. I had a necessity to deal with very large amounts of data and needed to scratch my itch.
- Focused on handling large series of tabular data:
    - Buffered I/O for maximum throughput.
    - Very fast selections through the use of Numexpr.
    - Column indexing for top-class performance queries.
- Incomes from selling PyTables Pro sponsors part of my invested time.

Some PyTables Users

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ACUSIM SOFTWARE

NOAA

GENERAL DYNAMICS
Strength On Your Side

UNIVERSITY OF Southampton
School of Engineering Sciences

MAX-PLANCK-GESELLSCHAFT

Germanischer Lloyd
OPERATING 24/7

SLAC

SarVision
applications in remote sensing

NASA
Jet Propulsion Laboratory
California Institute of Technology

cellzome

TeraView

Outline

Outline

Getting the Most Out of Computers: An Easy Goal?

- Computers nowadays are very powerful:
  - Extremely fast CPU's (multicores)
  - Large amounts of RAM
  - Huge disk capacities
- But they are facing a pervasive problem:
  An ever-increasing mismatch between CPU, memory and disk speeds (the so-called "**Starving CPU problem**")

This introduces tremendous difficulties in getting the most out of computers.

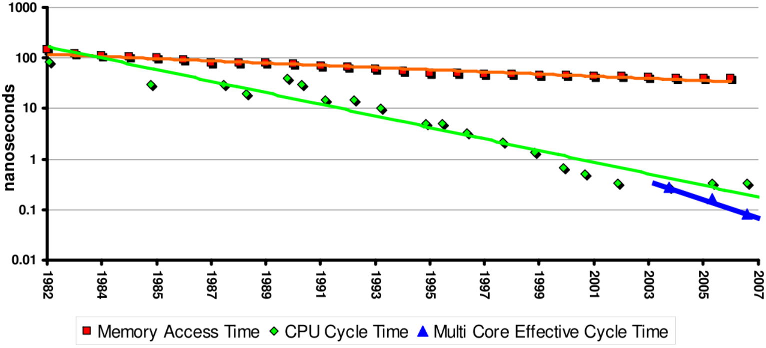
Once Upon A Time...

- In the 1970s and 1980s the memory subsystem was able to deliver all the data that processors required in time.
- In the good old days, the processor was the key bottleneck.
- But in the 1990s things started to change...

Once Upon A Time...

- In the 1970s and 1980s the memory subsystem was able to deliver all the data that processors required in time.
- In the good old days, the processor was the key bottleneck.
- But in the 1990s things started to change...

CPU vs Memory Cycle Trend



The CPU Starvation Problem

Known facts (in 2010):

- Memory latency is much higher (around 250x) than processors and it has been an essential bottleneck for the past twenty years.
- Memory throughput is improving at a better rate than memory latency, but it is also much slower than processors (about 25x).

The result is that CPUs in our current computers are suffering from a serious data starvation problem: *they could consume (much!) more data than the system can possibly deliver.*

Outline

1. The Starving CPU Problem
   - Getting the Most Out of Computers
   - Caches and Data Locality
   - Techniques For Fighting Data Starvation
2. High Performance Libraries
   - Why Should You Use Them?
   - In-Core High Performance Libraries
   - Out-of-Core High Performance Libraries

What Is the Industry Doing to Alleviate CPU Starvation?

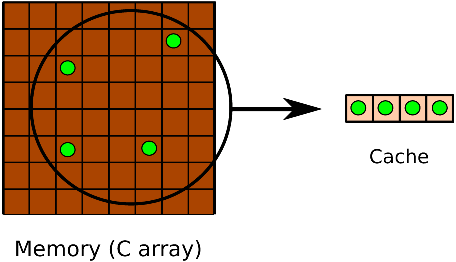
- They are improving memory throughput: cheap to implement (more data is transmitted on each clock cycle).
- They are adding big caches in the CPU dies.

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.

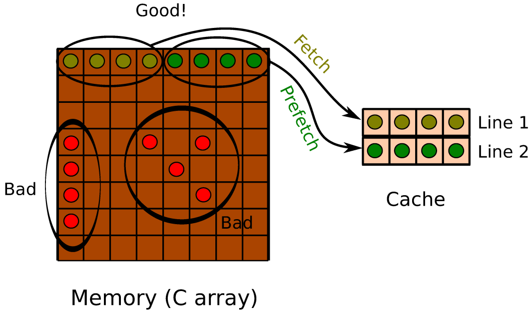
Time Locality

Parts of the dataset are reused



Spatial Locality

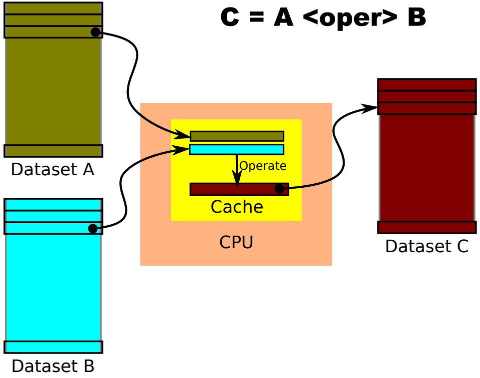
Dataset is accessed sequentially



Outline

The Blocking Technique

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:



Understand NumPy Memory Layout

Being "a" a squared array (4000x4000) of doubles, we have:

Summing up column-wise

```
a[:,1].sum()      # takes 9.3 ms
```

Summing up row-wise: more than 100x faster (!)

```
a[1,:].sum()      # takes 72 µs
```

Remember:

NumPy arrays are ordered row-wise (C convention)

Understand NumPy Memory Layout

Being “a” a squared array (4000x4000) of doubles, we have:

Summing up column-wise

```
a[:,1].sum()      # takes 9.3 ms
```

Summing up row-wise: more than 100x faster (!)

```
a[1,:].sum()      # takes 72 µs
```

Remember:

NumPy arrays are ordered row-wise (C convention)

Vectorize Your Code

Naive matrix-matrix multiplication: 1264 s (1000x1000 doubles)

```
def dot_naive(a,b):      #  1.5 MFlops
    c = np.zeros((nrows, ncols), dtype='f8')
    for row in xrange(nrows):
        for col in xrange(ncols):
            for i in xrange(nrows):
                c[row,col] += a[row,i] * b[i,col]
    return c
```

Vectorized matrix-matrix multiplication: 20 s (64x faster)

```
def dot(a,b):            #  100 MFlops
    c = np.empty((nrows, ncols), dtype='f8')
    for row in xrange(nrows):
        for col in xrange(ncols):
            c[row, col] = np.sum(a[row] * b[:,col])
    return c
```

The Consequences of the Starving CPU Problem

- The gap between CPU and memory speed is simply huge (and growing)
- Over time, an increasing number of applications will be affected by memory access

Fortunately, hardware manufacturers are creating novel solutions for fighting CPU starvation!

But vendors cannot solve the problem alone...

Computational scientists need another way to look at their computers:
Data arrangement, not code itself, is central to program design.

The Consequences of the Starving CPU Problem

- The gap between CPU and memory speed is simply huge (and growing)
- Over time, an increasing number of applications will be affected by memory access

Fortunately, hardware manufacturers are creating novel solutions for fighting CPU starvation!

But vendors cannot solve the problem alone...

Computational scientists need another way to look at their computers:
Data arrangement, not code itself, is central to program design.

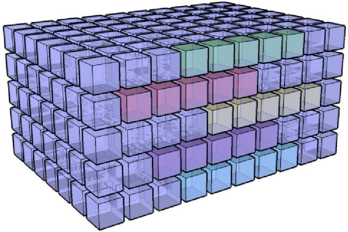
Outline

Why High Performance Libraries?

- High performance libraries are made by people that knows very well the different optimization techniques.
- You may be tempted to create original algorithms that can be faster than these, but in general, it is very difficult to beat them.
- In some cases, it may take some time to get used to them, but the effort pays off in the long run.

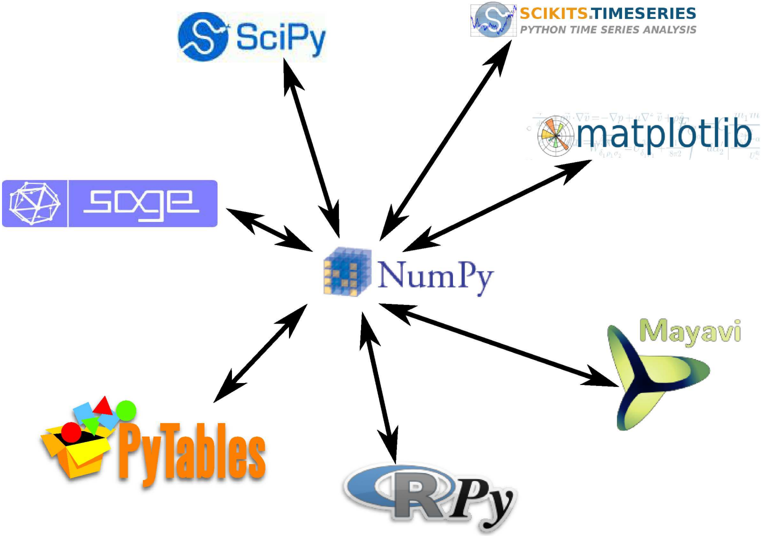
NumPy: A Powerful Data Container for Python

NumPy provides a very powerful, object oriented, multi-dimensional data container:



- `array[index]`: retrieves a portion of a data container
- `(array1**3 / array2) - sin(array3)`: evaluates potentially complex expressions
- `numpy.dot(array1, array2)`: access to optimized BLAS (*GEMM) functions

NumPy: The Cornerstone of Python Numerical Apps



Outline

Some In-Core High Performance Libraries

ATLAS/MKL (Intel's Math Kernel Library): Uses memory efficient algorithms as well as SIMD and multi-core algorithms $\rightarrow$ linear algebra operations.

VML (Intel's Vector Math Library): Uses SIMD and multi-core to compute basic math functions (sin, cos, exp, log...) in vectors.

Numexpr: Performs potentially complex operations with NumPy arrays without the overhead of temporaries. Can make use of multi-cores.

Blosc: A multi-threaded compressor that can transmit data from caches to memory, and back, at speeds that can be larger than a OS `memcpy()`.

ATLAS/Intel's MKL: Optimize Memory Access

Using integrated BLAS in NumPy: 5.6 s

```
numpy.dot(a,b)    # 350 MFlops
```

Using ATLAS: 0.19s (35x faster than integrated BLAS)

```
numpy.dot(a,b)    # 10 GFlops
```

Using Intel's MKL: 0.11 s (70% faster than ATLAS)

```
numpy.dot(a,b)    # 17 GFlops (2x12=24 GFlops peak)
```

Numexpr: Dealing with Complex Expressions

- Wears a specialized virtual machine for evaluating expressions.
- It accelerates computations by using blocking and by avoiding temporaries.
- Multi-threaded: can use several cores automatically.
- It has support for Intel's VML (Vector Math Library), so you can accelerate the evaluation of transcendental (sin, cos, atanh, sqrt. . .) functions too.

Numexpr Example

Be “a” and “b” are vectors with 1 million entries each:

Using plain NumPy

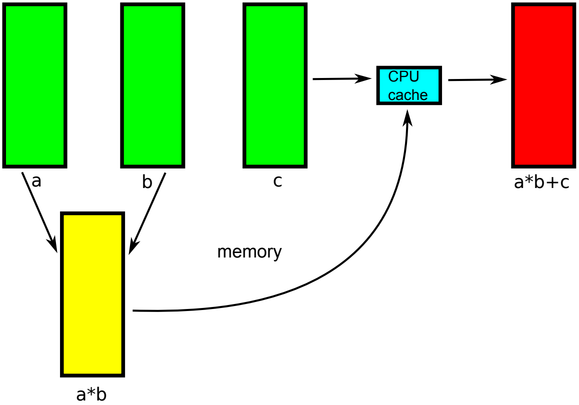
```
a**2 + b**2 + 2*a*b    # takes 33.3 ms
```

Using Numexpr: more than 4x faster!

```
numexpr.evaluate('a**2 + b**2 + 2*a*b')  # takes 8.0 ms
```

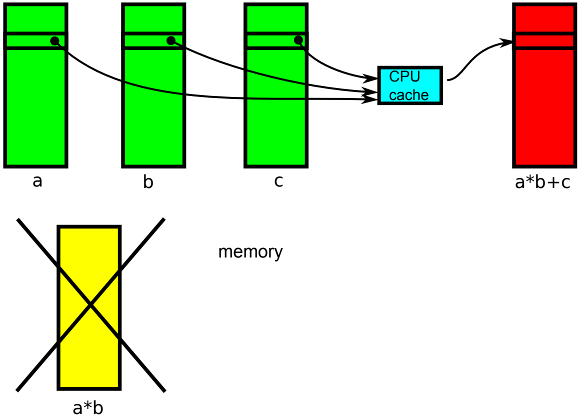
NumPy And Temporaries

Computing "a*b+c" with NumPy. Temporaries goes to memory.

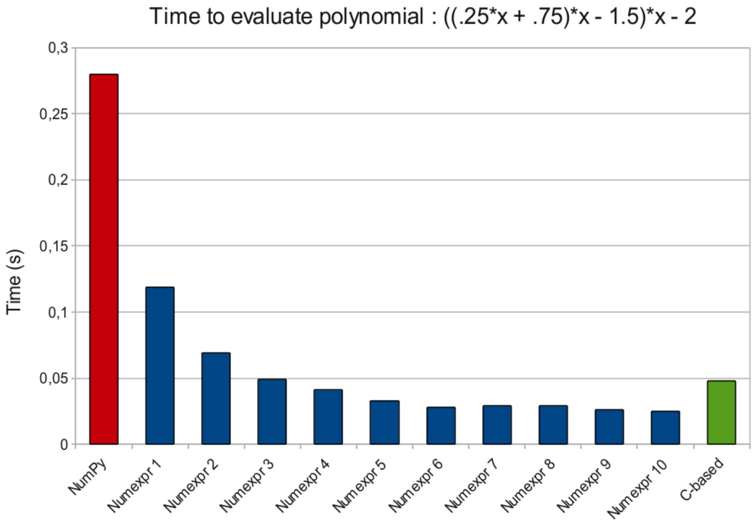


Numexpr Avoids (Big) Temporaries

Computing "a*b+c" with Numexpr. Temporaries in memory are avoided.



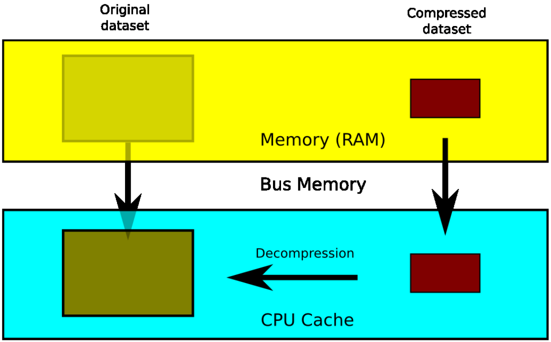
Numexpr Performance (Using Multiple Threads)



Blosc: A Blocked, Shuffling and Loss-Less Compression Library

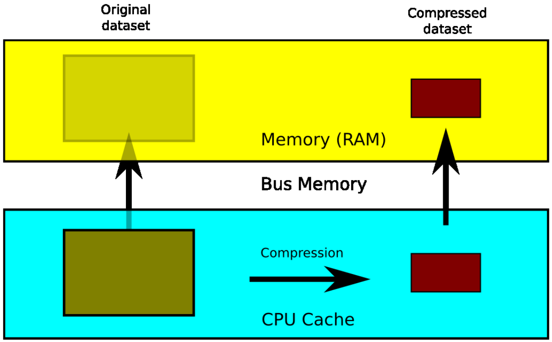
- Blosc (http://blosc.pytables.org/) is a new, loss-less compressor for binary data. It's optimized for speed, not for high compression ratios.
- It is based on the FastLZ compressor, but with some additional tweaking:
  - It works by splitting the input dataset into blocks that fit well into the level 1 cache of modern processors.
  - Makes use of SSE2 vector instructions (if available).
  - Multi-threaded (via pthreads).
- Has a Python wrapper (http://github.com/FrancescAlted/python-blosc)
- Free software (MIT license).

Reading Compressed Datasets



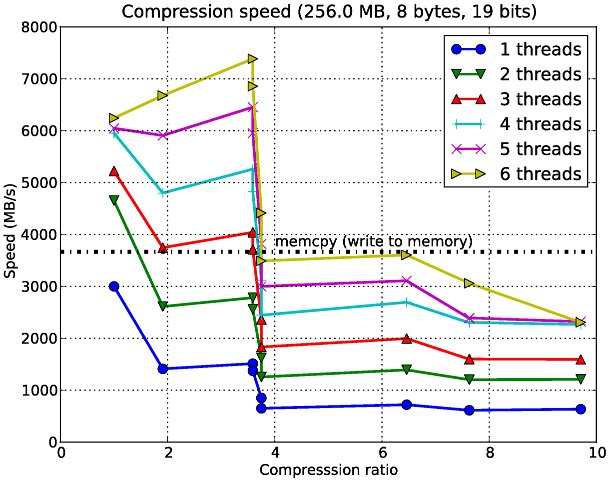
Transmission + decompression processes faster than direct transfer?

Writing Compressed Datasets

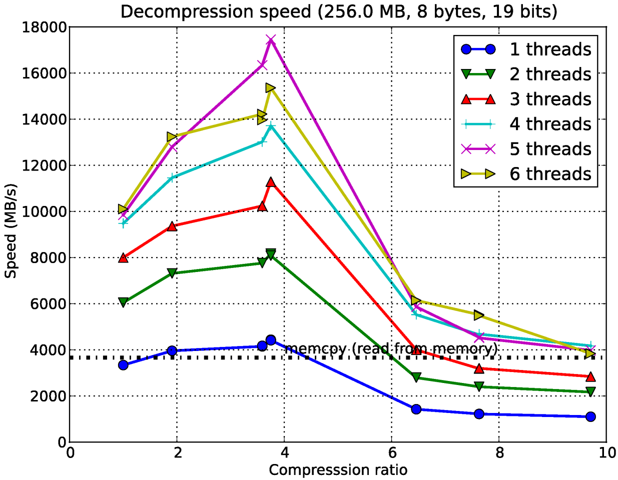


Compression + transmission processes faster than direct transfer?

Blosc: Beyond `memcpy()` Performance (I)



Blosc: Beyond `memcpy()` Performance (II)

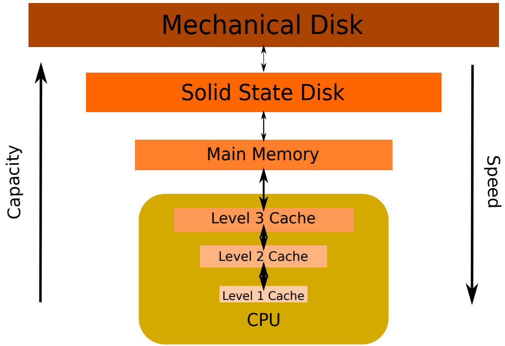


Outline

When Do You Need Out-Of-Core?

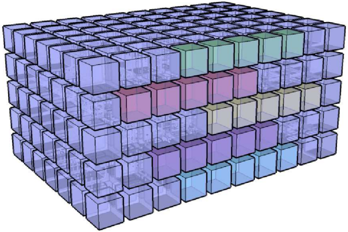
- Situations where your datasets do not fit in memory are increasingly common:
  - Datasets are continuously growing (e.g. better and more comprehensive sensors)
  - Finer precision in results normally requires larger storage size
- Persistence is needed

Disk VS Memory



- Disk access is more complicated than memory access
- OOC libraries should provide an easier interface

Easing Disk Access Using the NumPy OO Paradigm



- `array[index]`
- `(array1**3 / array2) - sin(array3)`
- `numpy.dot(array1, array2)`

Many existing OOC libraries are already mimicking **parts** of this abstraction.

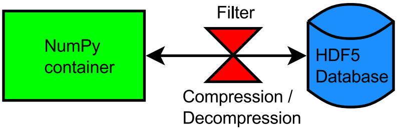
Some OOC Libraries Mimicking NumPy Model

Interfaces to binary formats (HDF5, NetCDF4):

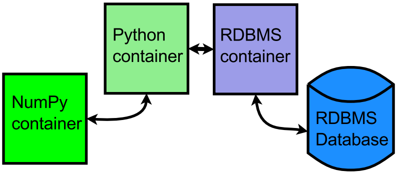
- Interfaces to HDF5:
  - h5py
  - PyTables
- Interfaces to NetCDF4:
  - netcdf4-python
  - Scientific.IO.NetCDF

Using NumPy As Default Container for OOC

All the previous libraries are using NumPy as default container (and they can also use compression filters for improved I/O).



Interfaces for RDBMS in Python lacks support for direct NumPy containers (very inefficient!).



PyTables: Retrieving a Portion of a Dataset

`array[index]`, where index can be one of the following:

- scalar: array[1]
- slice: array[3:1000, ..., :10]
- list (or array) of indices (fancy indexing): array[[3,10,30,1000]]
- array of booleans: array[array2 $>$ 0]

All these selection modes are supported by PyTables.

PyTables: Operating With Disk-Based Arrays

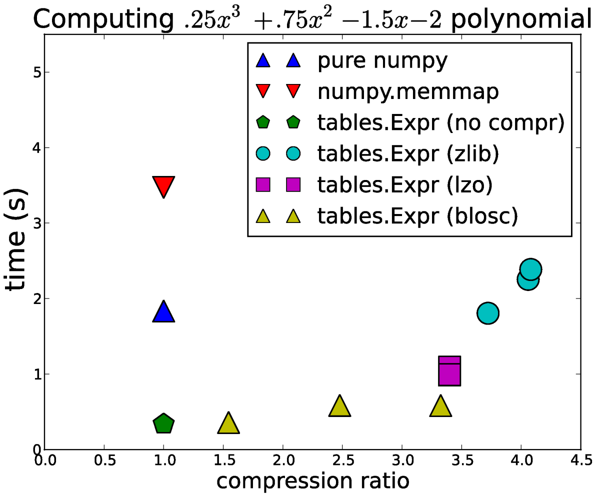
- `tables.Expr` is an optimized evaluator for expressions of disk-based arrays.
- It is a combination of the Numexpr advanced computing capabilities with the high I/O performance of PyTables.
- Similarly to Numexpr, disk-temporaries are avoided, and multi-threaded operation is preserved.

tables.Expr in Action

Evaluating `.25*x**3 + .75*x**2 - 1.5*x - 2`

```
import tables as tb
f = tb.openFile(h5fname, "a")
x = f.root.x     # get the x input
r = f.createCArray(f.root, "r", atom=x.atom, shape=x.shape)
ex = tb.Expr('.25*x**3 + .75*x**2 - 1.5*x - 2')
ex.setOutput(r)     # output will got to the CArray on disk
ex.eval()     # evaluate!
f.close()
```

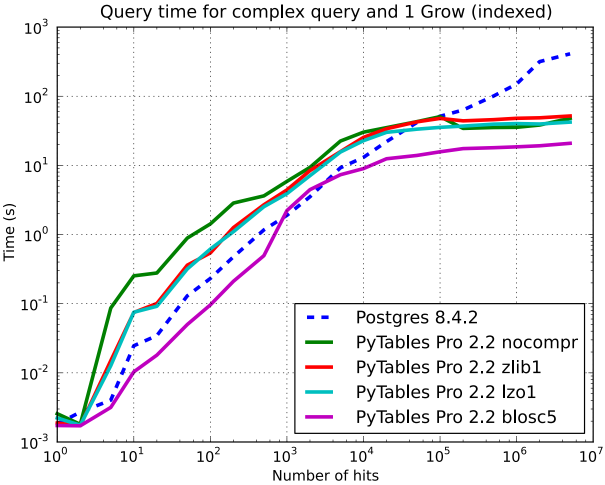
tables.Expr Performance (In-Core Operation)



Other Features of PyTables

- Allows organizing datasets on a **hierarchical structure**
- Each dataset or group can be complemented with user **metadata**
- Powerful query engine allowing **ultra-fast queries** (based on Numexpr and OPSI)
- Advanced compression capabilities (Blosc)

PyTables Pro Query Performance



Summary

- These days, you should understand that there is a CPU starvation problem if you want to get decent performance.
- Make sure that you use NumPy as the basic building block for your computations.
- Leverage existing memory-efficient libraries for performing your computations optimally.

More Info

- Francesc Alted
  Why Modern CPUs Are Starving and What Can Be Done about It
  Computing in Science and Engineering, IEEE, March 2010
  http://www.pytables.org/docs/CISE-March2010.pdf

- NumPy crew
  NumPy manual
  http://docs.scipy.org/doc/numpy

- PyTables site
  http://www.pytables.org

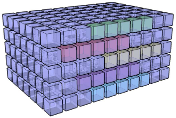
Questions?

Contact:

faltet@pytables.org

Acknowledgments

Thanks to Stéfan van der Walt for giving permission to use his cool multidimensional container picture:



This was made using a Ruby plugin for Google SketchUp.