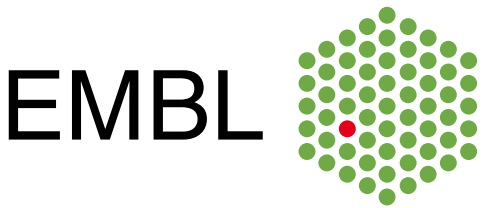
Jug: Executing Parallel Tasks in Python

Luis Pedro Coelho

EMBL

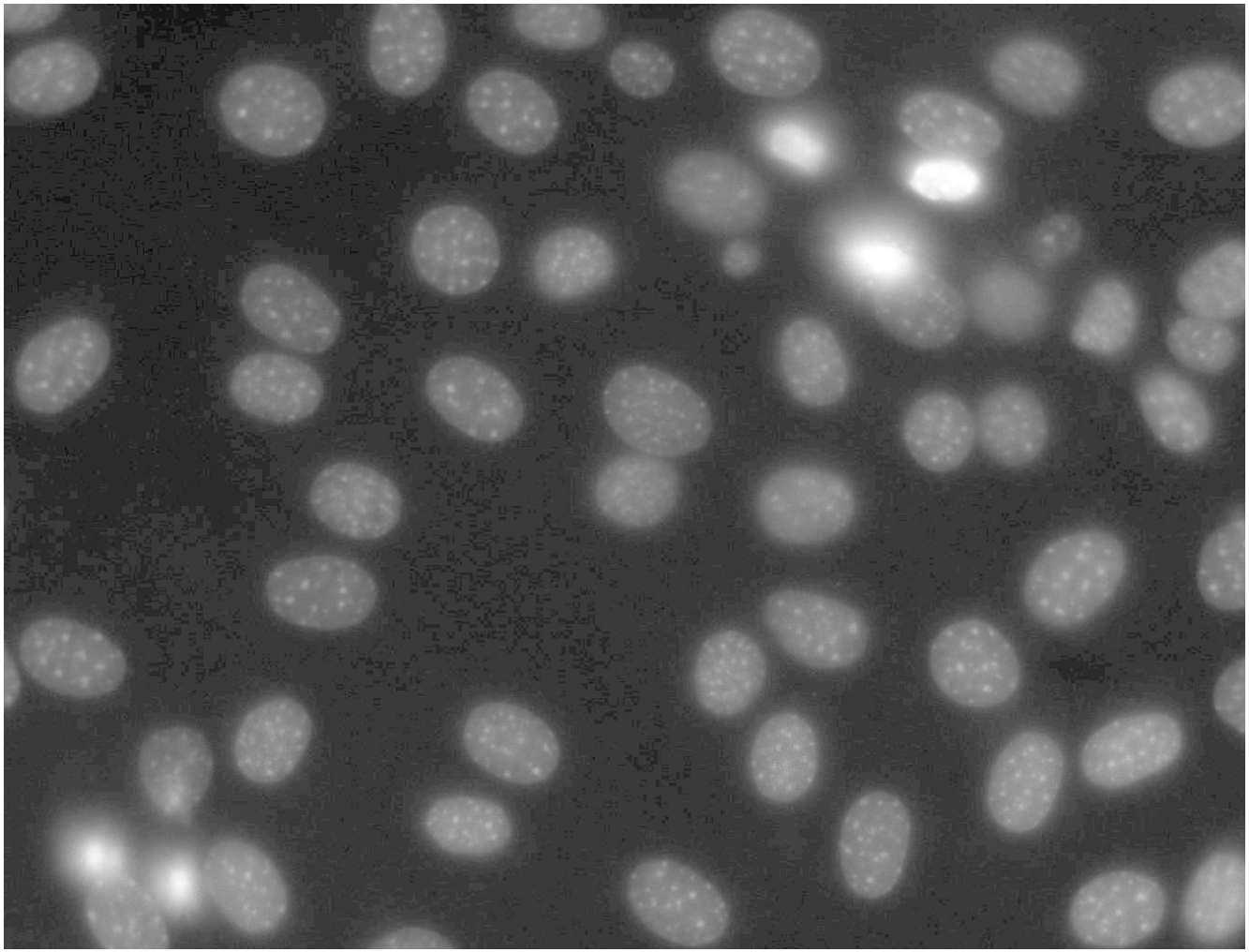
21 May 2013



Jug: Coarse Parallel Tasks in Python

- Parallel Python code
- Memoization

Example: Evaluating Segmentation Methods



Example: Evaluating Segmentation Methods

Problem Statement

1. You have images to segment
2. Many algorithms available
3. Which one is best?

Example: Evaluating Segmentation Methods

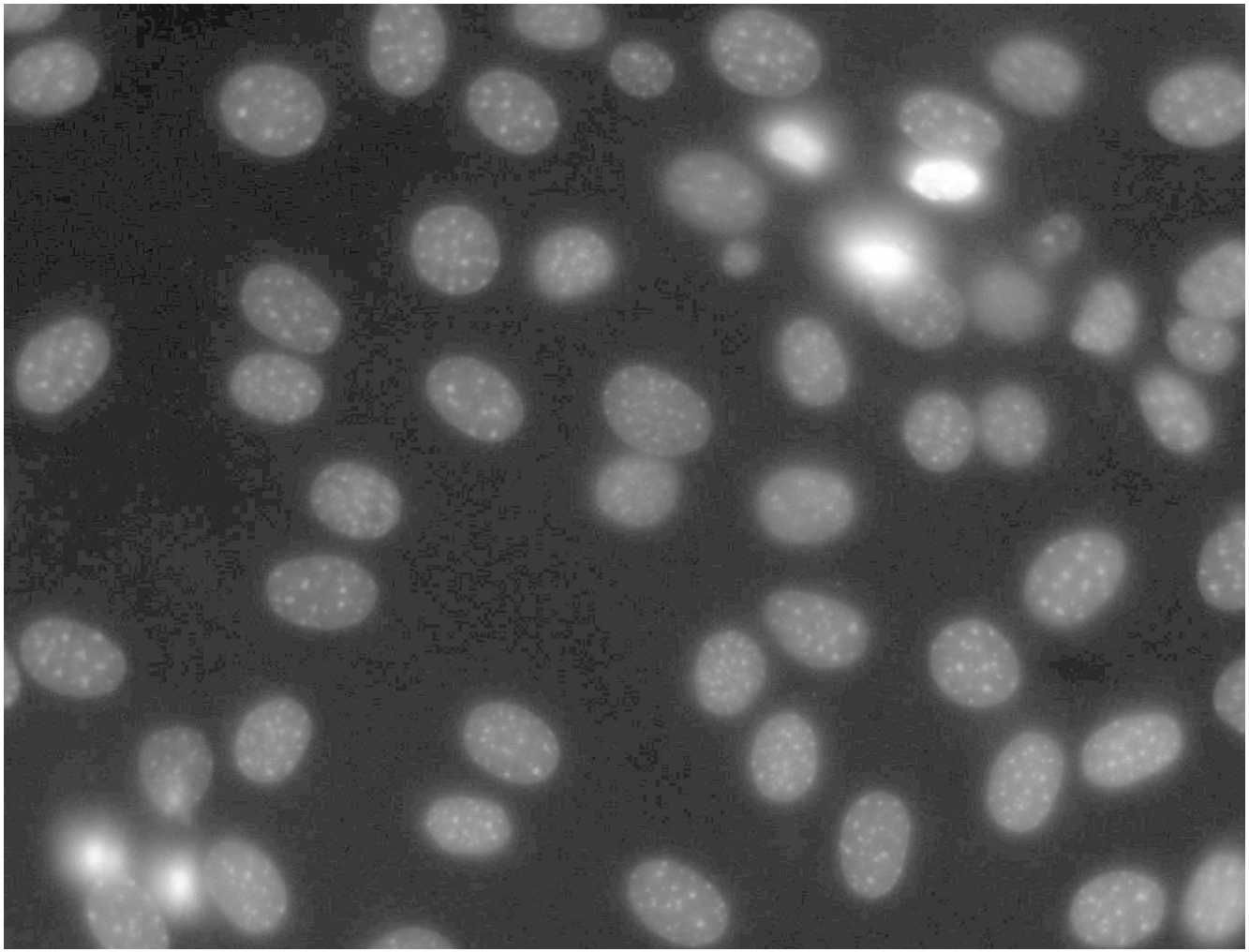
Problem Statement

1. You have images to segment
2. Many algorithms available
3. Which one is best?

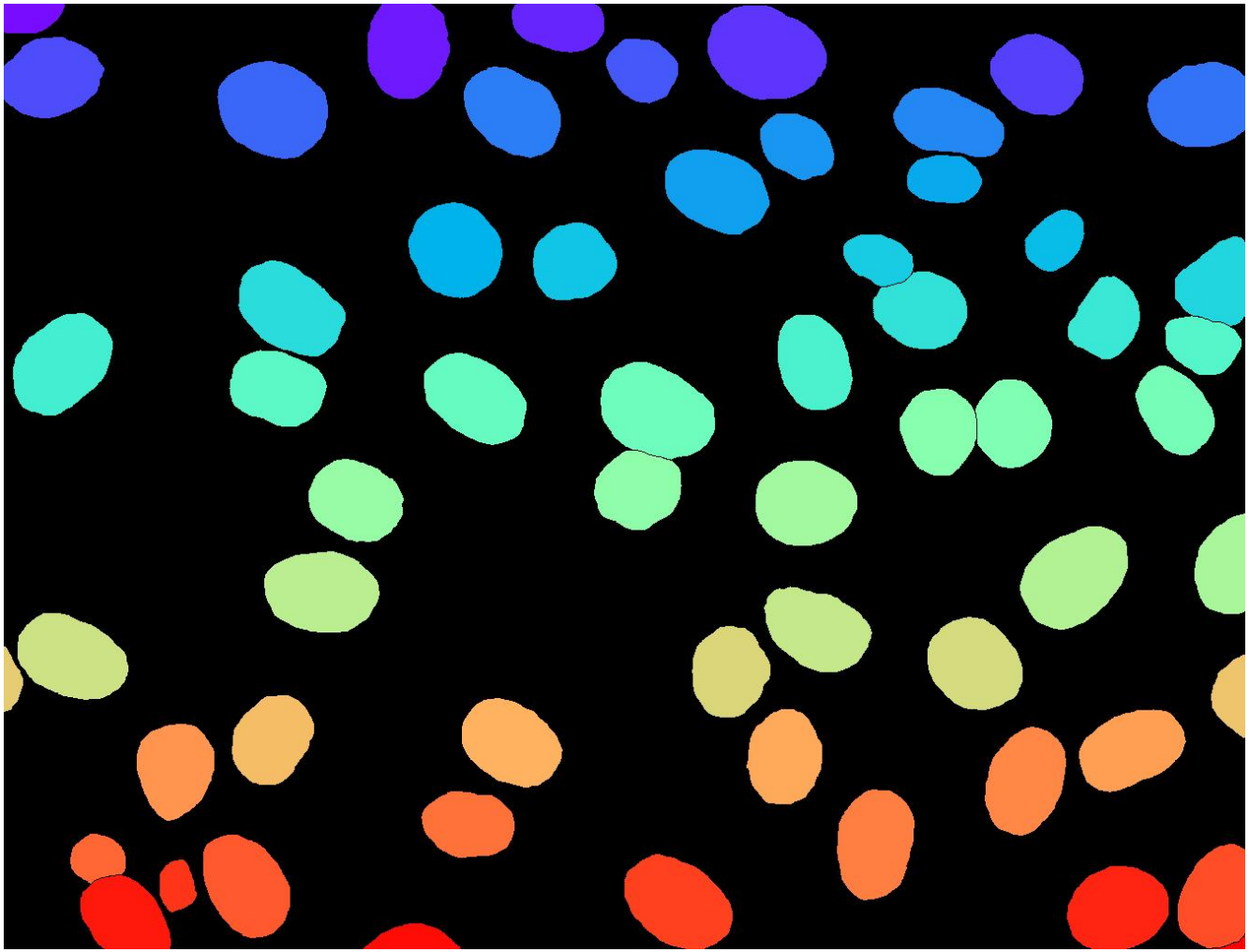
Solution

1. Manually segment a few images (reference)
2. Run algorithms on these images
3. Compare with reference

Reference Segmentations



Reference Segmentations



Segmentation Live Demo

If your software is really that good, you don't fear a live demo!

Methods

```
import mahotas as mh
def method1(image, sigma):
    image = mh.imread(image)[:,:,0]
    image  = mh.gaussian_filter(image, sigma)
    binimage = (image > image.mean())
    labeled, _ = mh.label(binimage)
    return labeled
```

mahotas is my computer vision/image processing package.

Segmentation Methods In Demo

Methods Under Study

1. Threshold with Otsu
2. Threshold with mean

Segmentation Methods In Demo

Methods Under Study

1. Threshold with Otsu
2. Threshold with mean

This is a Demo!

- Neither of these methods is very good!
- They are easy to explain & demo
- Read our paper for what methods actually work.
  (or just come talk to me).

Writing a jugfile

jugfile.py

```
from jug import TaskGenerator

@TaskGenerator
def method1(image, sigma):
    ...
```

Your code can be in multiple files

segmentation.py

```
import mahotas as mh

def method1(image, sigma):
    ...
```

jugfile.py

```
from jug import TaskGenerator
from segmentation import method1

method1 = TaskGenerator(method1)
```

Comparing Automated & Reference Segmentation

```
from glob import glob
inputs = glob('images/*.jpg')
results = []
for im in inputs:
    m1 = method1(im, 2)
    m2 = method2(im, 4)
    ref = im.replace('images','references') \
            .replace('jpg','png')
    v1 = compare(m1, ref)
    v2 = compare(m2, ref)
    results.append( (v1,v2) )
```

The above code looks like pure Python!

Demo Again

Also, ask questions...
By the way, if you're following at home, (i.e., downloaded the slides); you can see the code on github.

Task hashing saves structure of computation

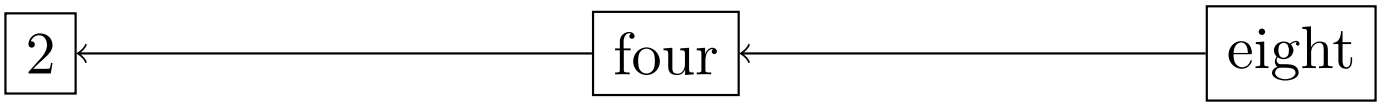
Let's look under the hood

```
@TaskGenerator
def double(x):
    return x*2

four = double(2)
eight = double(four)
```

converts to

```
def double(x):
    return x*2

four = Task(double, 2)
eight = Task(double, four)
```

Computation Structure



Compute Hash

```
def hash-of(task):
    return crypto-hash( {
                task.function,
                task.args,
                task.kwargs })
```

- If task.args are other tasks, recurse!
- That's pseudo-code
- Real-life code slightly more complex

The task hash encodes whole computation path

```
@TaskGenerator
def double(x):
    return x*2

four = double(2)
eight = double(four)
```

- four encodes double(2)
- eight encodes double(double(2))

Running a Task

```
def maybe-run-task(task, backend):
    h = task.hash()
    if backend.can_load(h):
        # Nothing to do
        return
```

Running a Task

```
def maybe-run-task(task, backend):
    h = task.hash()
    if backend.can_load(h):
        # Nothing to do
        return

    f = task.function
    args = []
    for a in task.args:
        if is-immediate-value(a):
            args.append(a)
        else:
            args.append(backend.load(a.hash()))
    # Same thing for kwargs
    return f(*args)
```

- Again, this is pseudo-code

Two Backends Are Available

Filesystem

- Default backend
- Carefully designed to work on NFS
- Anything pickle()able can be used as Task output/input.
- Numpy arrays are special-cased (for speed and disk-space savings).

Redis (NoSQL Database)

- Redis is a file-backed store
- Ideal for many small “files”
- All workers talk to same database

Jug Processes are Separate Processes!

- No GIL (Global Interpreter Lock) issues
- Can run on separate machines
- Do not need to start at the same time

Invalidate All Downstream Results

Case Study

1. You fix a bug in method1.
2. Now, you need to recompute all method1 calls.
3. Also, print_results

Jug Enhances Reproducibility

Typical Dark Side of Computational Analysis

- "What was the parameter that generated this result? I think it was ½, right? Had to be."
- "Deleted the intermediate results, reran; now everything is different."
- "We cannot reproduce the table in our own paper."

Advantages of Jug

- With jug, changing parameters will trigger recomputation of all downstream results.
- jug invalidate handles all dependencies
- Unlike make, you can use any Python function

How Much is Jug Used?

- It started as stereotypical scratch an itch software:
  I wrote it because I needed it
- Not very widely used at the moment
- Slowly picking up (by now 4.5 years old)
- 43,000 PyPI downloads
  Was at 13,000 than a year ago

Summary

Jug is Good For

- Coarse tasks (at least 1 second, ideally a few more)
- Data that fits on one disk
- Fan-out/Reduce/Fan-out modes
- Batch systems with shared network filesystems

Jug is Not Appropriate For

- Parallelization at micro level
- Data that does not fit in one disk

Finding Out More About Jug...

- http://metarabbit.wordpress.com
  My blog, latest posts are about jug
- http://github.com/luispedro/jug
  the code
- http://jug.rtfd.org
  read the fine documentation
- http://groups.google.com/group/jug-users
  google mailing list
- http://luispedro.org/software/jug
- luis@luispedro.org