Atomic Physics is Fantastic

Quentin Schibler
quentin.schibler@lkb.umpc.fr

Contents

Programming is like physics but you can cheat

- When in doubt, you can always ask god (the datasheet) for the answer.
- You can observe everything at once, no one will ever say to you $\sigma_p \sigma_x \geq \frac{\hbar}{2}$.
- You can even go back in time.

Observe your program

- `print()` is for printing, pdb is for debugging.
- `time()` is for **not** for timing, `perf_counter_ns()` obviously is
- Profilers are useful
- Always average over multiple measurements in a row, σ is usually huge

Computers like locality

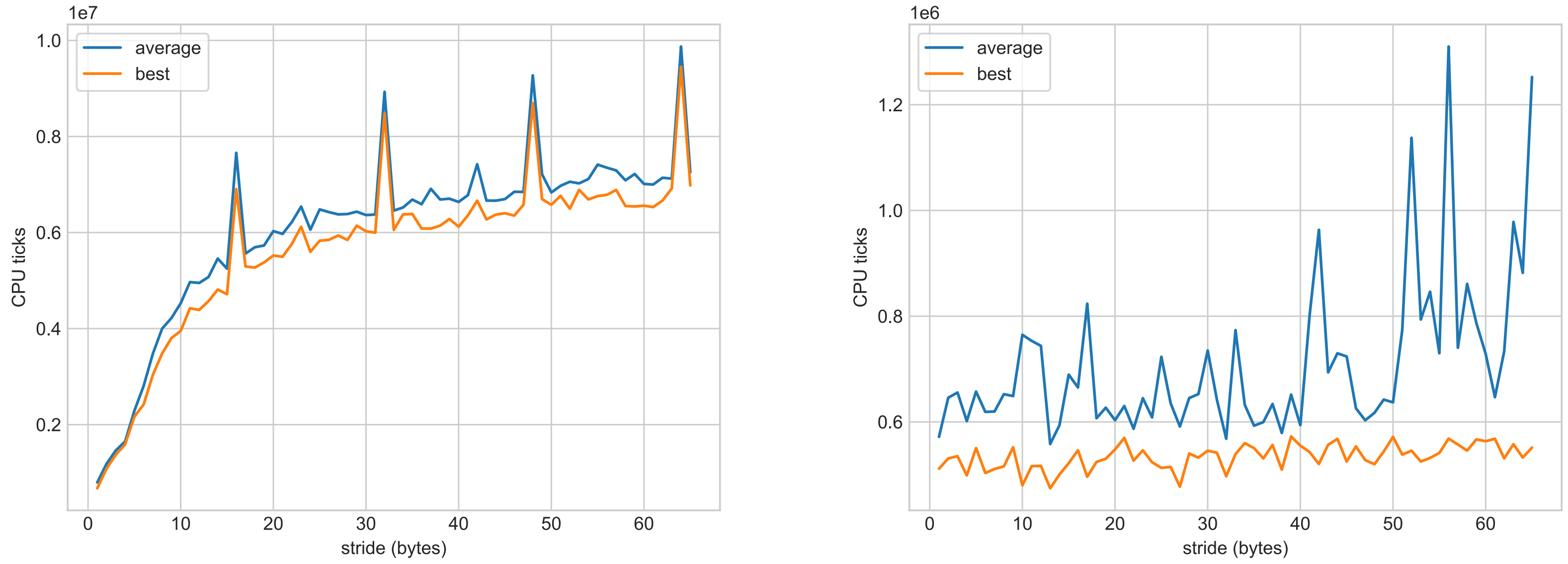

Figure 1: Performance of add-reducing an array for different strides, without and with copy

Programming like it’s 1999: Numpy

- Python is slow, sometimes of the order of 10^4 compared to regular C code.
- We have to use libraries like numpy to call into C code.
- You are on your own, you don’t have a compiler to optimise for you. Good luck captain.

Matrix multiplication

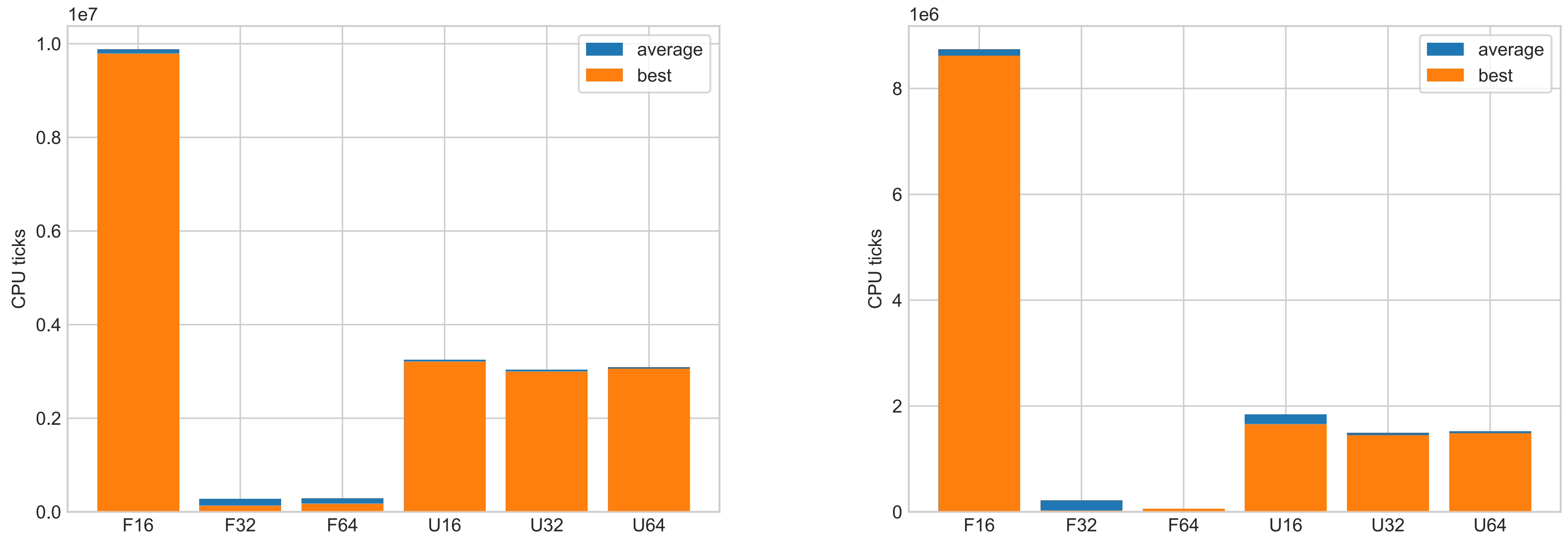

Figure 2: Performance of matrix multiply using numpy blas (left) and mkl (right)

Matrix multiplication

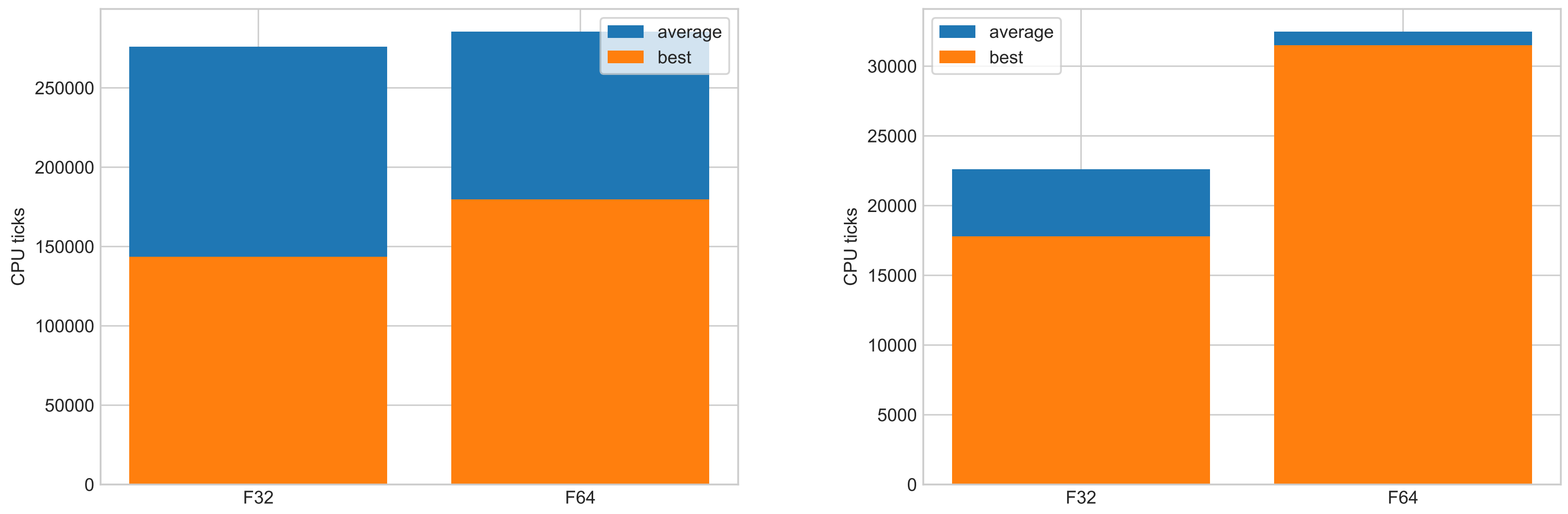

Figure 3: Performance of matrix multiply for floats using numpy blas (left) and mkl (right)

Taxonomy of numpy operations

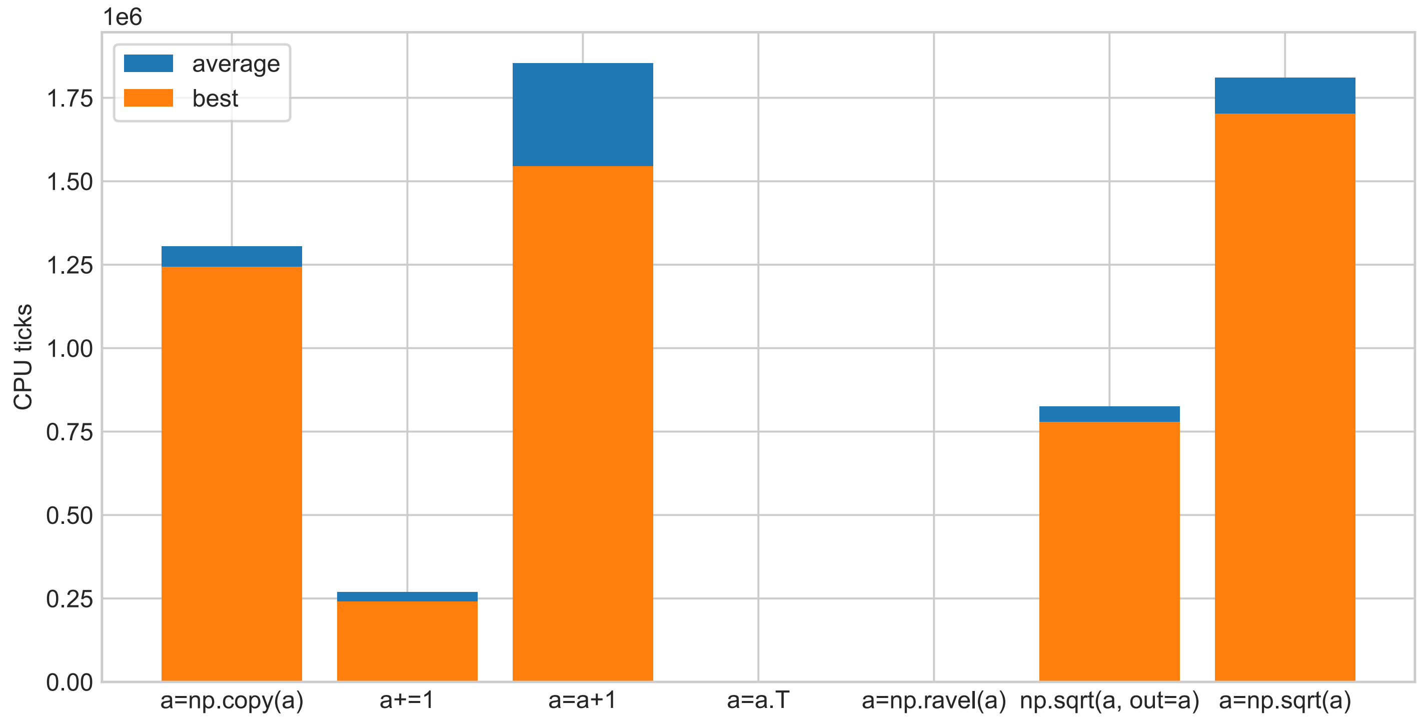

Figure 4: Performance of various numpy operations

Faster is better: Norm 2

- We'll play the let's make it faster game
- The problem is to compute the sum of the norm 2 of a bunch of complex number
- $s = -\sum_{i,j} \sqrt{R(i,j)^2 + I(i,j)^2}$

- Trivial implementation: `np.sum(np.sqrt(A**2 + B**2))`
- What can we improve based on what we just learned ?
- Wait until the end before you say numba Tanguy

Getting rid of the copies :

```
A**=2
B**=2
A+=B
np.sqrt(A, out=A)
np.sum(A)
```

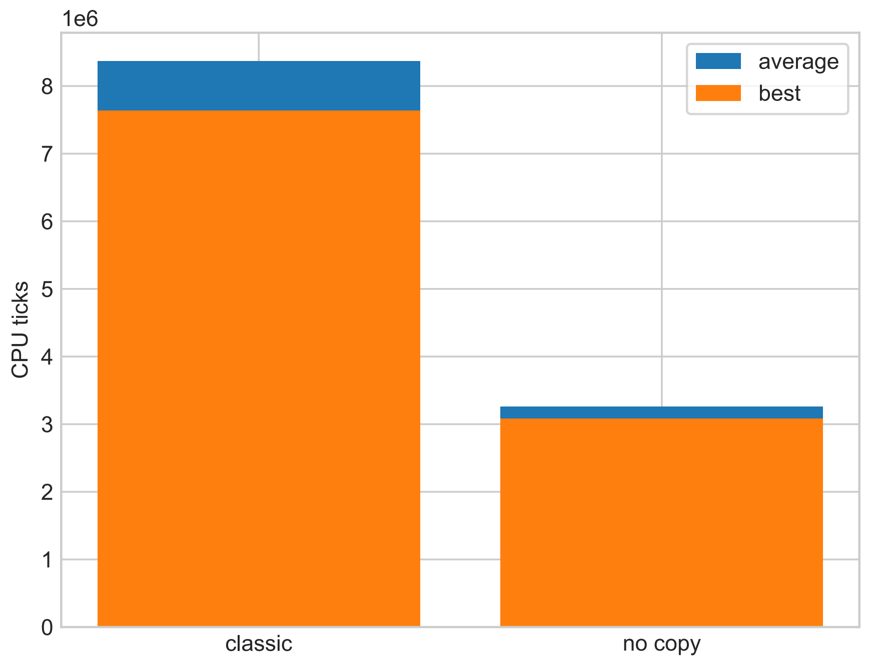

Figure 5: Performance of different norm 2 implementations

Faster sum:

```
A**=2
B**=2
A+=B
np.sqrt(A, out=A)
v = np.ones(A.shape[0])
v @ A @ v.T
```

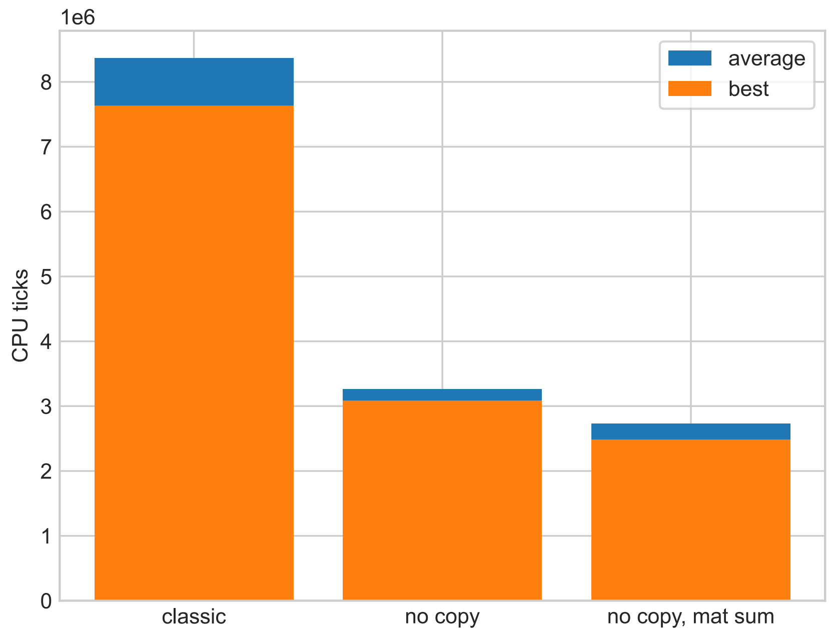

Figure 6: Performance of different norm 2 implementations

Numba loop:

```
sum = 0
for j in range(n):
  for i in range(n):
    a = A[i,j]**2
    b = B[i,j]**2
    sum += (a+b)**0.5
```

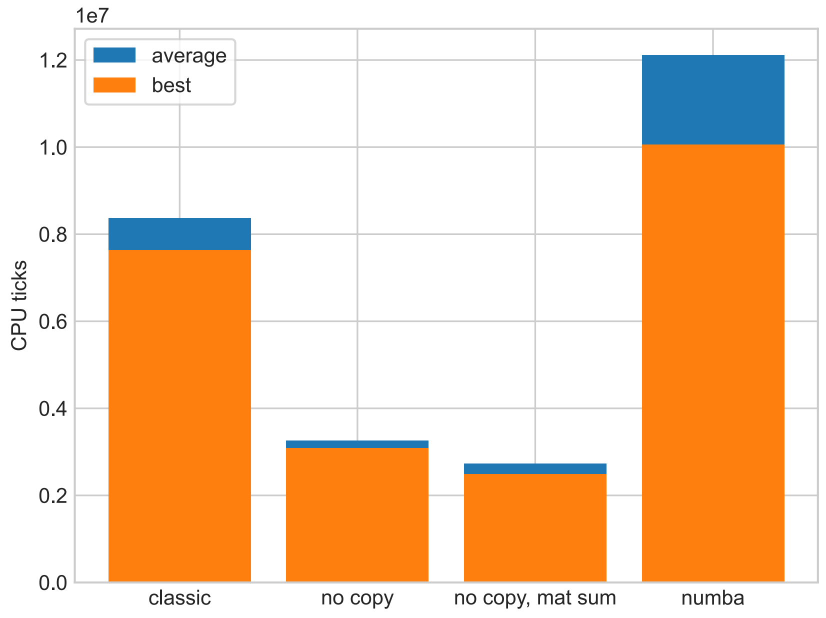

Figure 7: Performance of different norm 2 implementations

Putting everything together

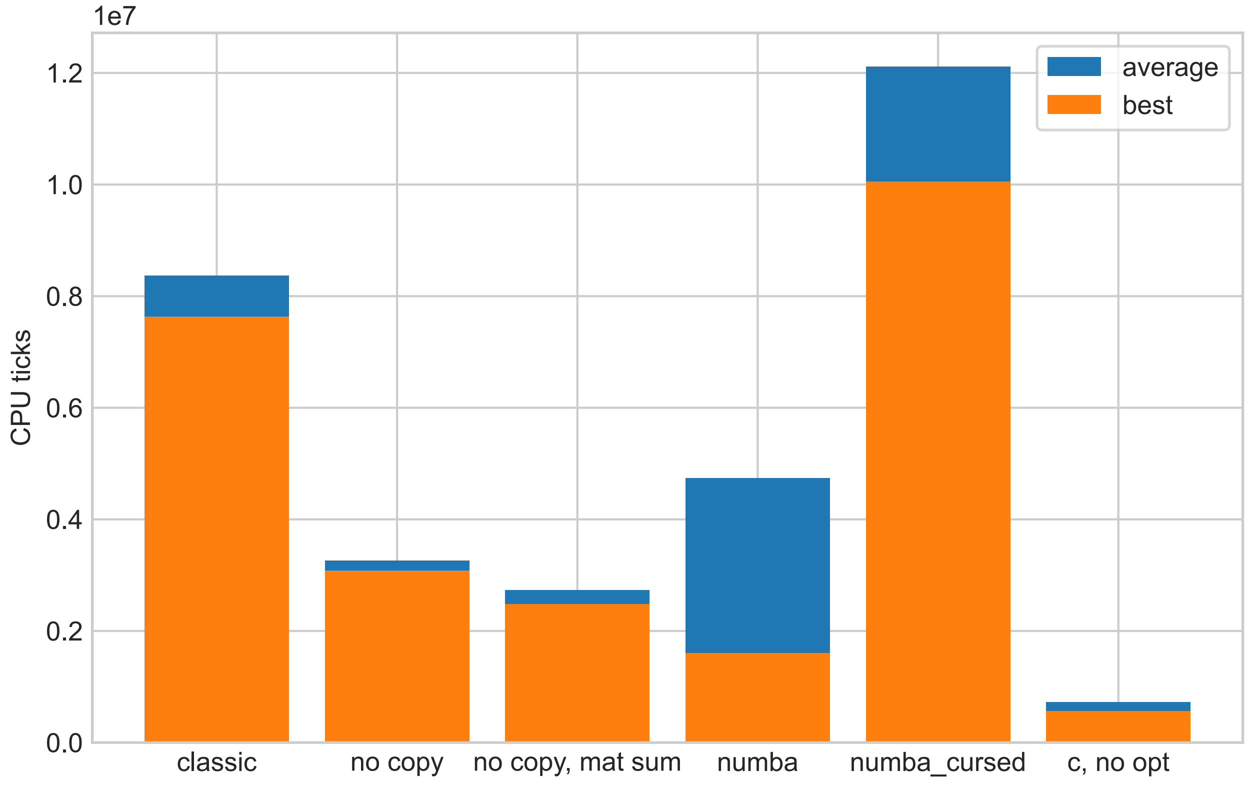

Figure 8: Performance of different norm 2 implementations

Random useful thought

Taxonomy of optimisations methods

- Check what your objective function looks like
- If you have the freedom to change it, try to make it convex
- Compute the gradient if you can, it often contains part of the function.
- CG is better when you have a cheap cost function, BFGS when you don't

Assert, assert, assert.

- Basically what the title says, use assertions everywhere
- `assert idx < arr.size`
- `assert arr.size < 1e4 "the algorithm scale in n², find a better one"`
- `assert arr.dtype == np.float64 "numpy use BLAS which only supports float64 for fast matrix multiply"`

Thanks