Solutions for the exercises for Parallel Cython

Computing the Mandelbrot set

Exercise 1

The parallel loop is:

```
for pix in prange(npixels, nogil=True, schedule="dynamic"):
```

at *cython-mandel/mandel.pyx*.

Exercise 2

- Which is the speed-up of the latter with respect to the default run?

It depends on your machine, but pretty linear speed-ups are expected.

- How many threads do you think is the default for OpenMP?

The default is to use all the cores in your machine.

Exercise 3

- Do you find that it scales smoothly?

In general, yes.

Computing a polynomial

Exercise 4

- Does it scale the same way than the mandel program?

Not quite. It scales much more poorly. See *Mandel-Poly.ods* or *Mandel-Poly.pdf* for details.

- Why do you think this is so?

The answer for this is in the forthcoming *Starving CPUs* lecture.

Static vs dynamic scheduling

Exercise 5

5. In sources above, the OpenMP scheduling setting for threads has been set to its optimal value. Play with 'static' and 'dynamic' values for it.

    - Which one is better for each case?

The answer of the question is: it depends. Dynamic scheduling is better for the Mandel package, while a static scheduling is optimal for the Poly one. See *Mandel-Poly.ods* or *Mandel-Poly.pdf* for details.

    - Could you explain why?

First, we need a succinct explanation about the difference between dynamic and static schedules. A 'static' schedule means that, if your loop has to iterate, say, 100 times, and you use 3 threads, then 33 times will go to thread 1, 33 to thread 2 and 34 to thread 3. Now, a 'dynamic' division means that there is a

master thread sending work to slave threads, and does so in a first-finished, first-served basis.

The reason why the dynamic scheduling is best for the Mandel case is that the computational cost of the Mandelbrot set strongly depends on the region of that set. For example, regions near 'borders' of the Mandelbrot set are much more costly to compute that regions that are far from them. Hence, a dynamic schedule leads to a more even distribution of the work among threads.

On its hand, the cost for computing points for the Poly case is the same along all the range of points where the polynomial needs to be computed. A static division then leads to an optimal partition of the work among threads, and besides, it does not have the overhead of the dynamic scheduling approach (that is, send a task, wait for finish, send another one...). This is why 'static' wins here.