Dust emission ... in parallel!

Edgar Ortiz
edgar.ortiz@ua.cl

Abstract

Following on the previously "Dust emission" work, in the present work, I implement this code to fit dust emission models for a set of 21 galaxies using the multiprocessing.Pool tool of python to do the computations in parallel.

Introduction

Modern CPUs contain different cores. When it comes to computations and taking advantage of these multicore architectures, we can, if possible, perform the calculations in parallel. Doing independent computations in different cores is what is called parallelism. Python offers a high-level parallelism library called multiprocessing. During this work, I'll be using the tool Pool from this library to perform parallelism. I'll be using this technique to fit dust emission models to a set of 21 galaxies.

Data

The data for the emission models that I'll be using comes from Drain & Li (2007) as was the case for the previous work. Additionally, I have a table with 21 galaxies that have observed fluxes in [mJy] in 7 different filters, the errors and the distances in [Mpc] as well.

Science questions, code & results

My first task is to compute a grid of models in parallel by sampling the values of the different parameters that create a model. To compute the grid of models I implement the following lines of code:

```
with Pool() as pool:
    models = pool.starmap(Model, params)
    lum_densities = [model.L_density() for model in models]
```

Here Model corresponds to the class Model defined in the previous work and params is a list of tuples, where each tuple contains the sampled values that initialize a model. The method **starmap** maps each of these tuples to Model to instantiate the class, meaning we will have as many instances as the number of tuples we have. Pool() is in charge of creating the processes that will run in parallel, in this case since there is no argument, it will create as many as logical cores there are. After the grid of models has been computed in parallel, I implement the method L_density which corresponds to the monochromatic luminosity of a model in a band (see code).

Finally, I have to fit the models in parallel, to do so, I defined a class that contains the method to fit the models and in a similar fashion I ran it with the following code:

```
with Pool() as pool:
    fits = pool.starmap(Fit, params)
    chi2s = [fit.chi2_2() for fit in fits]
```

Here the constructor of Fit accepts four parameters: the measured flux, the corresponding error to that flux, its distance and an n-dimensional array with the fluxes coming from the different computed models. The distance parameter is used to convert the flux of the models to the units of [mJy].

Given to circumstances of slow computations in my computer, I had to create a mock data set for the models' fluxes to test my code, which was done with the function **mock_ms().** The chi2_2 method returns the chi-squared for a given galaxy. Since in my case I'm dealing with 21, I obtain 21 chi-squared values, but each chi-squared is an n-dimensional array, where n corresponds to the number of models that were fitted to the galaxy. After that, I use the NumPy min method to obtain the smallest chi-squared per galaxy. And that concludes the work I was able to do for this graded practical.