Finanziato dall'Unione europea
NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

Characterization of exoplanetary atmospheres with GUIBRUSH®

G. Guilluy, P. Giacobbe, A. S. Bonomo

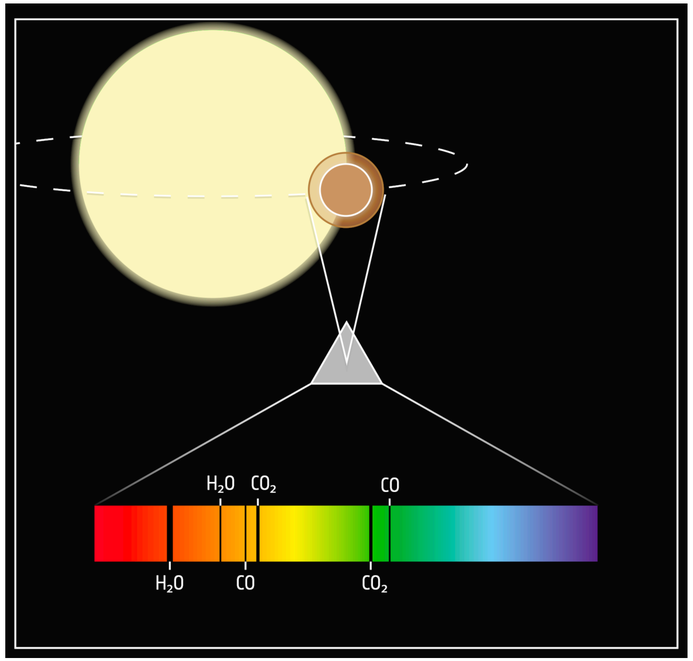
Monthly WP1-2 Meeting, March 20, 2024

Scientific Rationale

EXOPLANETARY ATMOSPHERES

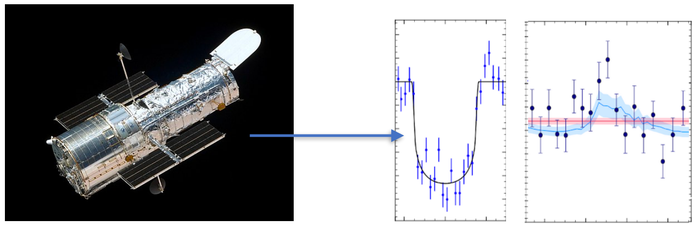
- Encoded within a planet spectrum there is information about the *formation and migration history*.

Transiting planets represent a gold booty to perform atmospheric studies.



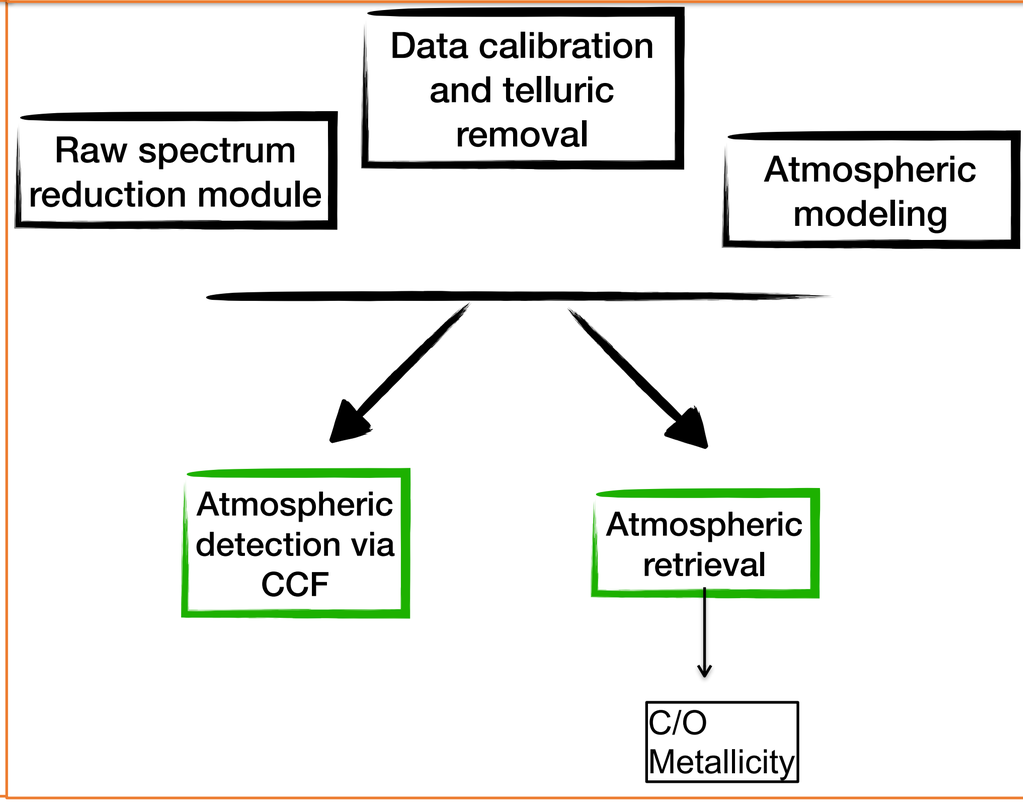
How do we probe exoplanetary atmospheres?

Space-borne Low-Resolution Spectroscopy



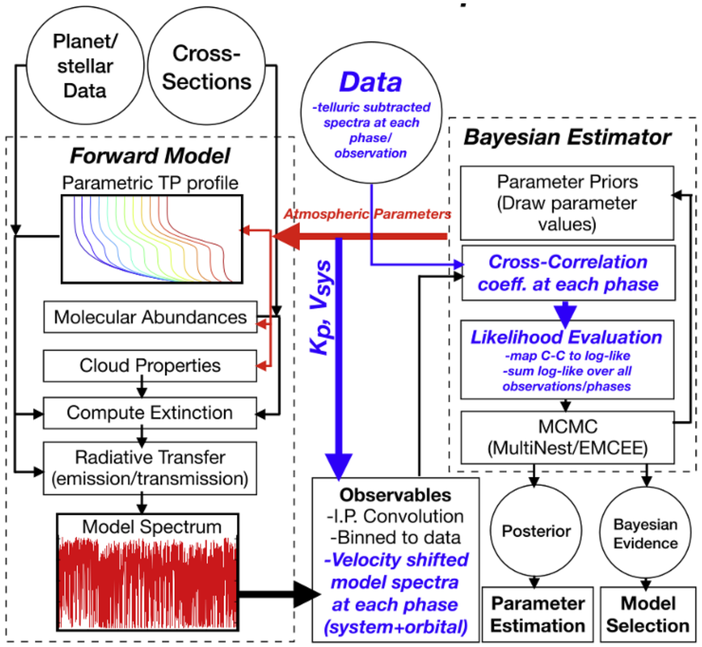
Ground-based High-Resolution Spectroscopy (R>20 000)

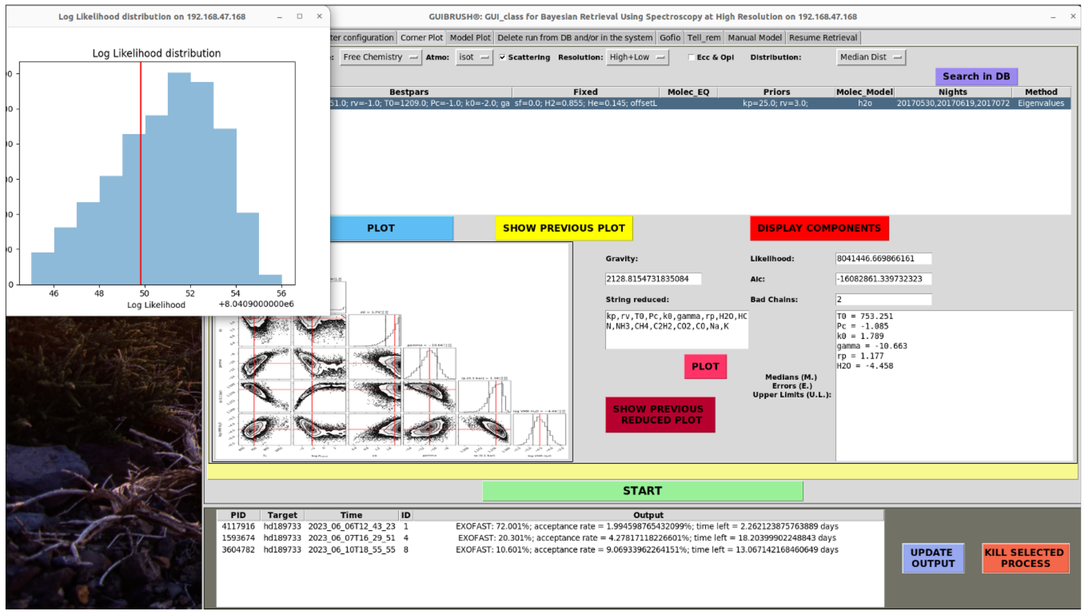
GUIBRUSH®:
Is a user friendly workspace to study and characterize exoplanetary atmospheres



Technical Objectives, Methodologies and Solutions

-**The Code: GUIBRUSH®** is coded in Python > 3.8 and makes use of the Bayesian differential evolution Markov chain Monte Carlo (DE-MCMC) technique to explore the parameter space and derive the posterior distributions of the free parameters such as the abundances of the probed chemical species.

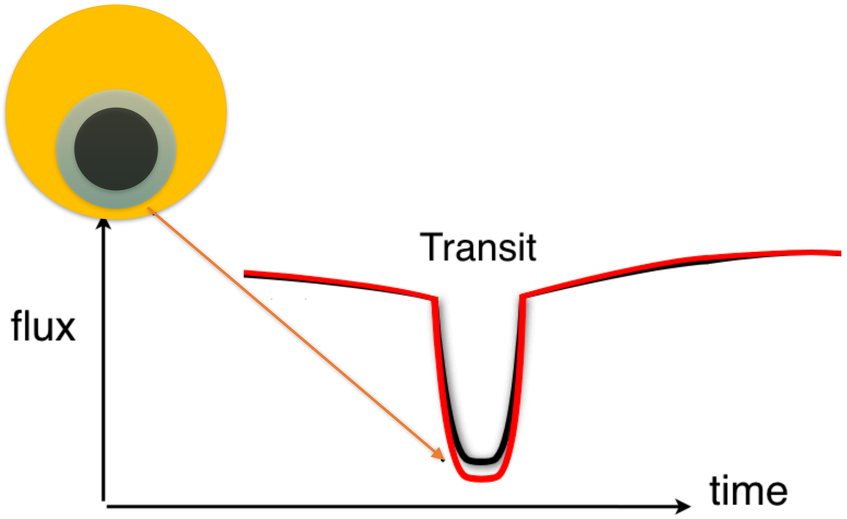




-GUIBRUSH® is currently run on the HPE Proliant DL560 Gen10 Server at INAF-Osservatorio Astrofisico di Torino, which was purchased with the PRIN-INAF 2019 project "HOT-ATMOS" (PI: A. S. Bonomo) and currently has 2 processors, 48 2.3Ghz cores and 256GB R AM; we will expand it to 6 processors, 144 cores, 750GB RAM in the next months thanks to an INAF minigrant (PI: P. Giacobbe).

Technical Objectives, Methodologies and Solutions

-The main bottleneck is due to the slowness of the radiative transfer code, which takes ~10 s to produce a single atmospheric model at each step of each DE-MCMC chain to be compared with the observed spectrum and thus compute the likelihood function. The currently employed radiative transfer code is the publicly available, open-source tool **petitRADTRANS** (Mollière 2019).



$$\text{Transit depth} = \frac{\Delta F_\lambda}{F_\lambda} = \frac{F_\lambda^{\text{out}} - F_\lambda^{\text{in}}}{F_\lambda^{\text{out}}}$$

RADIATIVE TRANSFER:

$$I_\lambda(\tau) = I_0 e^{-\tau_\lambda}$$

$$\text{Transit depth} = \frac{1}{R_*^2}\left[R_{\text{top}}^2 - 2\int_0^{R_{\text{top}}} e^{-\tau_\lambda} b\,\mathrm{d}b\right]$$

-The **goal** would be to reduce the computation time of a single atmospheric model by at least a factor of 10, that is about or less than 1 s.

UPDATES from the last SPOKE 3 technical meeting

M7

--Decision on the radiative transfer code to use:

-We generated a transmission spectrum for an atmosphere composed of H2O, H2 and He with both PYRATBAY and PetitRADTRANS (the latter was currently used in GUIBRUSH). The investigated atoms and molecule have a uniform distribution at different pressures (atmospheric layers) with fixed volume mixing ratio as reported in Figure1. We simulated 100 atmospheric layers with a pressure between 10^{-6} — 10^{+2} bar

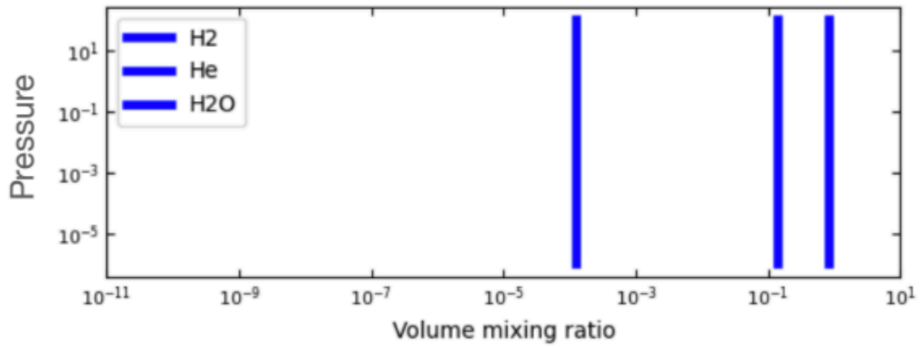


Figure 1, VMR of the investigated molecule/Atoms as a function of the atmospheric pressure

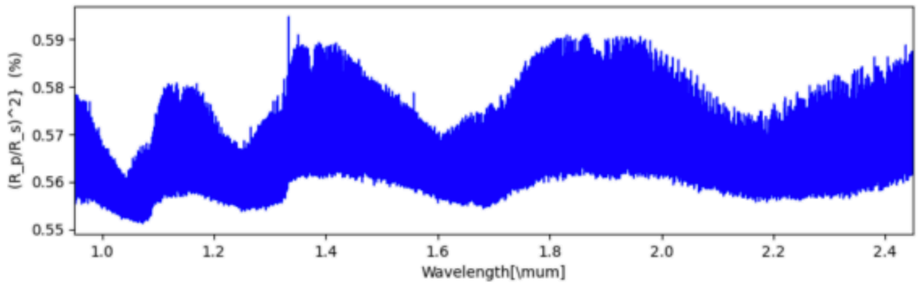


Figure 2, Transmission spectrum as a function of wavelength

-We computed the opacity for H2O from the HITEMP linelist at a resolution of R=250,000. We used a wavelength range [0.95-2.45] μm, which is the same range of the GIANO-B spectrograph at the Telescopio Nazionale Galileo (La Palma island).

FINAL RESULT=> PetitRADTRANS takes: 2.630 s, while PYRATBAY takes: 2.405 s.

-We thus decided to use the RADIATIVE TRANSFER CODE implemented in PYRATBAY, as it is a bit faster and the developer of the code is a close collaborator.

UPDATES from the last SPOKE 3 technical meeting

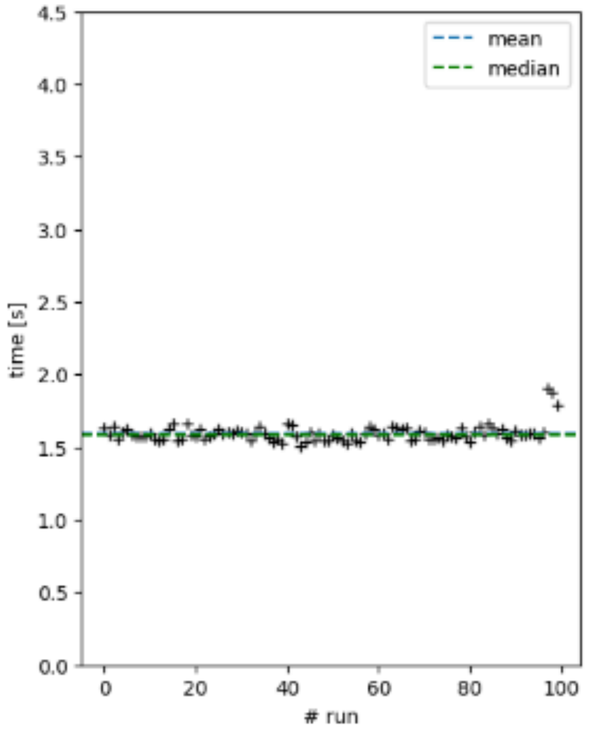
--Porting on GPU:

-- We are currently working with PyOpenCL and with a AMD Radeon Pro 5500M Compute Engine GPU.

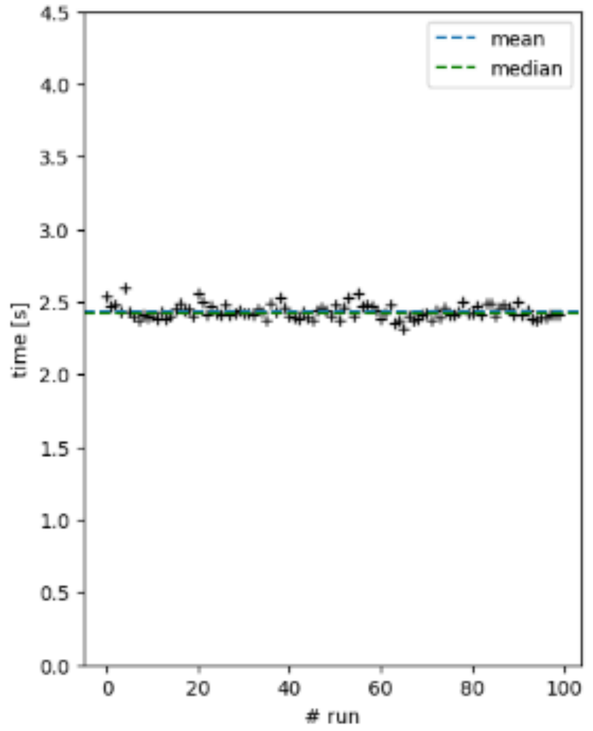
-- We tested the new code to generate a model with only water for the GIANO-B range, we recorded a slight increase in speed compared to the original code.

However, the code is still much slower than desired

Code in PyOpenCL on GPU



Original Code



UPDATES from the last SPOKE 3 technical meeting

--Porting on GPU:

Benchmarking

-Copying on the GPU: 6.9906711157836914e-05 s

```
    #print(padded_flatten)

start_copy_GPU = time.time()


data_buffer = cl.Buffer(ctx, mf.READ_ONLY | mf.USE_HOST_PTR, hostbuf=np.double(data_flatten))
intervals_buffer = cl.Buffer(ctx, mf.READ_ONLY | mf.USE_HOST_PTR, hostbuf=np.double(padded_arrays))
#ideep_buffer = cl.Buffer(ctx, mf.READ_WRITE | mf.USE_HOST_PTR, hostbuf=ideep)
#tau_buffer = cl.Buffer(ctx, mf.READ_WRITE | mf.USE_HOST_PTR, hostbuf=np.double(od_depth_flatten))  # Buffer for tau values
ideep_buffer = cl.Buffer(ctx, mf.WRITE_ONLY, ideep.nbytes)
tau_buffer = cl.Buffer(ctx, mf.WRITE_ONLY, od_depth_flatten.nbytes)  # Buffer for tau values

   #print(len(od.depth[r]),nwave)


end_copy_GPU = time.time()
```

-Time for programming: 0.005542576313018799 s

```
program2.parallelize_tau_gloria_for(queue, (nwave,), None,
                                    tau_buffer,
                                    ideep_buffer,
                                    intervals_buffer,
                                    np.double(od.maxdepth),
                                    data_buffer,
                                    np.int32(max_length),
                                    np.int32(nwave),
                                    np.int32(rbottom-rtop),
                                    np.int32(rtop))
```

-Copying from the GPU: 0.13318092346191407 s

```
cl.enqueue_copy(queue, ideep, ideep_buffer).wait()
cl.enqueue_copy(queue, od.depth[rtop:rbottom+1], tau_buffer).wait()
end_copy_from_GPU = time.time()
```

UPDATES from the last SPOKE 3 technical meeting

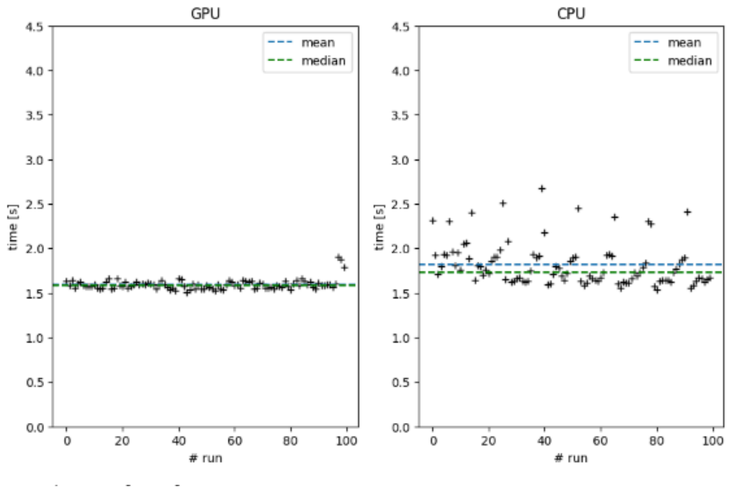
--Porting on GPU:

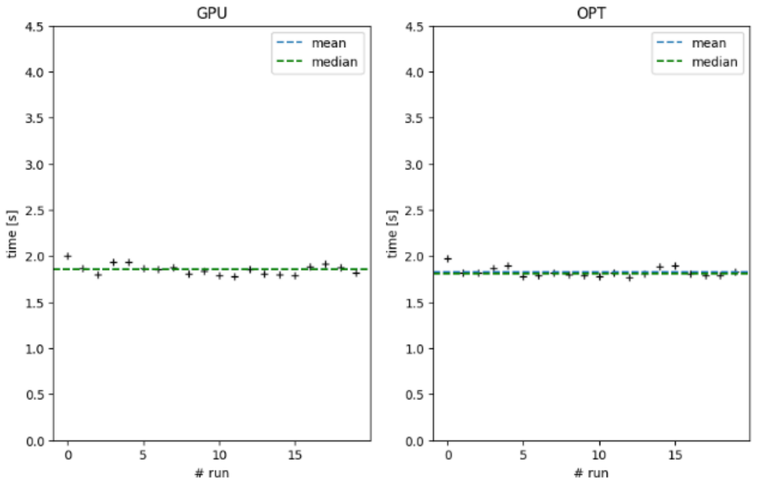
Possible problems:

-the program is not optimized for GPU computation.

-It is as if we are currently running a single thread of #wavelengths elements (236582).

-We also attempted to create multiple threads working simultaneously: specifically, we used a local size of 58, so we got 4079.0 workgroups. However, we did not observe a decrease in the execution time.





#program2.parallelize_tau_gloria_for(queue, (nwave,), None,

program2.parallelize_tau_gloria_for(queue, (nwave,), (58,),