pyDAMPF: a Python package for modeling mechanical properties of hygroscopic materials under interaction with a nanoprobe

Willy Menacho N.[‡], Gonzalo Marcelo Ramírez- Ávila[‡], Horacio V. Guzman[‡§*]

Abstract—pyDAMPF is a tool oriented to the Atomic Force Microscopy (AFM) community, which allows the simulation of the physical properties of materials under variable relative humidity (RH). In particular, pyDAMPF is mainly focused on the mechanical properties of polymeric hygroscopic nanofibers that plays an essential role in designing of tissue scaffolds for implants and filtering devices. Those mechanical properties have been mostly studied from a very coarse perspective reaching a micrometer scale. However, at the nanoscale, the mechanical response of polymeric fibers becomes cumbersome due to both experimental and theoretical challenges. For example, the response of polymeric fibers to RH demands advanced models that consider sub-nanometric changes in the local structure of each single polymer chain. While from an experimental viewpoint, choosing the optimal cantilevers to scan the fibers under variable RH is not trivial.

In this article, we show how to use pyDAMPF to choose one optimal nanoprobe for planed experiments with an hygroscopic polymer. Along these lines, We show how to evaluate common and non-trivial operational parameters from an AFM cantilever from several manufacturers. Our results show in a stepwise approach the most relevant parameters to compare the cantilevers based on a non-invasive criterion of measurements. The computing engine is written in Fortran in order to reuse physics code and wrapped to Python to interoperate with high-level packages. We also introduce an in-house and transparent for the user multi-thread approach of the pyDAMPF code, which compares for various computing architectures (PC, Google Colab and an HPC facility) very favorable in comparison to former AFM simulators.

Index Terms—Materials science, Nanomechanical properties, AFM, f2py, multi-threading CPUs, numerical simulations, polymers

Introduction and Motivation

This article provides an overview of pyDAMPF, which is a BSD licensed, Python and Fortran simulation tool that enables AFM users to simulate the interaction between a probe (cantilever) and materials at the nanoscale under diverse environments. The code is packaged in a bundle and hosted on GitHub at https://github.com/govarguz/pyDAMPF.

Despite the recent open-source availability of dynamic AFM simulation packages [**?**], [**?**], a broad usage for the assessment and planning of experiments has yet to come. One of the problems is that it is often hard to simulate several operational parameters at once. For example, most scientists evaluate different AFM cantilevers before starting new experiments. A typical evaluation criterion is the maximum exerted force that prevents invasivity of the nanoprobe into the sample. The variety of AFM cantilevers depends on the geometrical and material characteristics used for its fabrication. Moreover, manufacturers' nanofabrication techniques may change from time to time, according to the necessities of the experiments, like sharper tips and/or higher oscillation frequencies. From a simulation perspective, evaluating observables for reaching optimal results on upcoming experiments is nowadays possible for *tens* or *hundreds* of cantilevers. On top of other operational parameters in the case of dynamic AFM like the oscillation amplitude A_0, set-point A_{sp}, among other materials expected properties that may feed simulations and create simulations batches of easily *thousands* of cases. Given this context, we focus this article on choosing a cantilever out of an initial pyDAMPF database of 30. In fact, many of them are similar in terms of spring constant k_c, cantilever volume V_c and also Tip's radius R_T. Then we focus on seven archetypical and distinct cases/cantilevers to understand the characteristics of each of the parameters specified in the manufacturers' datasheets, by evaluating the maximum (peak) forces.

We present four scenarios comparing a total of seven cantilevers and the same sample, where we use as a test-case Poly-Vinyl Acetate (PVA) fiber. The first scenario (Figure 1(a)) illustrates the difference between air and a moist environment. On the second one, a cantilever, only a very soft and stiff cantilever spring constants are compared (see Figure 1(b)).At the same time, the different volumes along the 30 cantilevers is depicted in Figure 1(a). A final and mostly very common comparison is scenario 4, by comparing one of the most sensitive parameters to the force of the tip's radii (see Figure 1(d)).

The quantitative analysis for these four scenarios is presented and also the advantages of computing several simulation cases at once with our in-house development. Such a comparison is performed under the most common computers used in science, namely, personal computers (PC), cloud (Colab) and supercomputing (small Xeon based cluster). We reach a Speed-up of 20 over the former implementation [**?**].

Another novelty of pyDAMPF is the detailed [**?**] calculation of the environmental-related parameters, like the quality factor Q.

‡ *Instituto de Investigaciones Físicas. Carrera de Física, Universidad Mayor de San Andrés. Campus Universitario Cota Cota. La Paz, Bolivia*
§ *Department of Theoretical Physics, Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia.*
∗ *Corresponding author: horacio.guzman@ijs.si*

Here, we summarize the main features of pyDAMPF are:

- Highly efficient structure in terms of time-to-result, at least one order of magnitude faster than existing approaches.
- Easy to use for scientists without a computing background, in particular in the use of multi-threads.
- It supports the addition of further AFM cantilevers and parameters into the code database.
- Allows an interactive analysis, including a graphical and table-based comparison of results through Jupyter Notebooks.

The results presented in this article are available as Google Colaboratory notebook, which facilitates to explore pyDAMPF and these examples.

Methods

Processing inputs

pyDAMPF counts with an initial database of 30 cantilevers, which can be extended at any time by accessing to the file *cantilevers_data.txt* then, the program inputs_processor.py reads the cantilever database and asks for further physical and operational variables, required to start the simulations. This will generate *tempall.txt*, which contains all cases *e.g.* 30 to be simulated with pyDAMPF

```
def inputs_processor(variables,data):
    a,b = np.shape(data)
    final = gran_permutador( variables, data)
    f_name = ' tempall.txt'
    np.savetxt(f_name,final)
    directory = os.getcwd()
    shutil.copy(directory+'/tempall.txt',directory+'
              /EXECUTE_pyDAMPF/')
    shutil.copy(directory+'/tempall.txt',directory+'
              /EXECUTE_pyDAMPF/pyDAMPF_BASE/nrun/')
```

The variables inside the argument of the function *inputs_processor* are interactively requested from a shell command line. Then the file *tempall.txt* is generated and copied to the folders that will contain the simulations.

Execute pyDAMPF

For execution on a single or multi-thread way, we require first to wrap our numeric core from Fortran to Python by using f2py [**?**]. Namely, the file *pyDAMPF.f90* within the folder *EXECUTE_pyDAMPF*.

Compilation with f2py: This step is only required once and depends on the computer architecture the code for this reads:

```
f2py -c --fcompiler=gnu95 pyDAMPF.f90 -m mypyDAMPF
```

This command-line generates mypyDAMPF.so, which will be automatically located in the simulation folders.

Once we have obtained the numerical code as Python modules, we need to choose the execution mode, which can be serial or parallel. Whereby parallel refers to multi-threading capabilities only within this first version of the code.

Serial method: This method is completely transparent to the user and will execute all the simulation cases found in the file *tempall.txt* by running the script *inputs_processor.py*. Our in-house development creates an individual folder for each simulation case, which can be executed in one thread.

```
def serial_method(tcases, factor, tempall):
    lst = gen_limites(tcases, factor)
    change_dir()
    for i in range(1,factor+1):
        direc = os.getcwd()
        direc2 = direc+'/pyDAMPF_BASE/'
        direc3 = direc+'/SERIALBASIC_0/'+str(i)+'/'
        shutil.copytree ( direc2,direc3)
    os.chdir ( direc+'/SERIALBASIC_0/1/nrun/')
    exec(open('generate_cases.py').read())
```

As arguments, the serial method requires the total number of simulation cases obtained from *tempall.txt*. In contrast, the *factor* parameter has, in this case,a default value of 1.

Parallel method: The parallel method uses more than one computational thread. It is similar to the serial method; however, this method distributes the total load along the available threads and executes in a parallel-fashion. This method comprises two parts: first, a function that takes care of the bookkeeping of cases and folders:

```
def Parallel_method(tcases, factor, tempall):
    lst = gen_limites(tcases, factor)
    change_dir()
    for i in range(1,factor+1):
        lim_inferior=lst[i-1][0]
        lim_superior=lst[i-1][1]
        direc =os.getcwd()
        direc2 =direc+'/pyDAMPF_BASE/'
        direc3 =direc+'/SERIALBASIC_0/'+str(i)+'/'
        shutil.copytree ( direc2,direc3)
        factorantiguo = ' factor=1'
        factornuevo='factor='+str(factor)
        rangoantiguo = '( 0,paraleliz)'
        rangonuevo='('+str(lim_inferior)+','
                +str(lim_superior)+')'
        os.chdir(direc+'/PARALLELBASIC_0/'+str(i))
        pyname =' nrun/generate_cases.py'
        newpath=direc+'/PARALLELBASIC_0/'+str(i)+'/'
                +pyname
        reemplazo(newpath,factorantiguo,factornuevo)
        reemplazo(newpath,rangoantiguo,rangonuevo)
        os.chdir(direc)
```

This part generates serial-like folders for each thread's number of cases to be executed.

The second part of the parallel method will execute pyDAMPF, which contains at the same time two scripts. One for executing pyDAMPF in a common *UNIX* based desktop or laptop. While the second is a python script that generated *SLURM* code to launch jobs in HPC facilities.

- Execution with *SLURM*
  It runs pyDAMPF in different threads under the *SLURM* queuing system.

```
def cluster(factor):
    for i in range(1,factor+1):
        with open('jobpyDAMPF'+str(i)+'.x','w')
                as ssf :
            ssf.write('#/bin/bashl|n ')
            ssf.write('#SBATCH--time=23:00:00
\n')
            ssf.write('#SBATCH--constraint=
epyc3\n')
            ssf.write('#\n')
            ssf.write('\n')
            ssf.write('ml Anaconda3/2019.10\n')
            ssf.write('\n')
            ssf.write('ml foss/2018a\n')
            ssf.write('\n')
            ssf.write('cd/home/$<USER>/pyDAMPF/
EXECUTE_pyDAMPF/PARALLELBASIC_0/'+str(i)+'/nrun
\n')
```

```
            ssf.write('\n')
            ssf.write('echo$pwd\n')
            ssf.write('\n')
            ssf.write('python3 generate_cases.py
\n')
            ssf.close();
        os.system(sbatch jobpyDAMPF)'+str(i)+'
.x;')
        os.system(rm jobpyDAMPF)'+str(i)+'.x;')
```

The above script generates *SLURM* jobs for a chosen set of threads; after launched, those jobs files are erased in order to improve bookkeeping

- Parallel execution with *UNIX* based Laptops or Desktops
  Usually, microscopes (AFM) computers have no SLURM pre-installed; for such a configuration, we run the following script:

```
def compute(factor):
    direc = os.getcwd()
    for i in range(1,factor+1):
        os.chdir(direc+'/PARALLELBASIC_0/'+
                 str(i)+'/nrun')
        os.system('python3 generate_cases.py
                  &')
        os.chdir(direc)
```

This function allows the proper execution of the parallel case without a queuing system and where a slight delay might appear from thread to thread execution.

Analysis

Graphically:

- With static graphics, as shown in Figures 2, 3, 4 and 5.

```
python3 Graphical_analysis.py
```

- With interactive graphics, as shown in 6.

```
pip install plotly

jupyter notebook Graphical_analysis.ipynb
```

Quantitatively:

- With static data table

```
python3 Quantitative_analysis.py
```

- With interactive tables
  Quantitative_analysis.ipynb uses a minimalistic dashboard application for tabular data visualization tabloo with easy installation.:

```
pip install tabloo

jupyter notebook Quantitative_analysis.ipynb
```

Results and discussions

In Figure 1, we show four scenarios to be tackled in this test-case for pyDAMPF. As described in the introduction, the first scenario (Figure 1(a)), compares between air and moist environment, the second tackles soft and stiff cantilevers(see Figure 1(b)), next is Figure Figure 1(c) with the cantilever volume comparison and the force the tip's radio (see Figure 1(d)). Further details of the cantilevers depicted here are included in Table 1.

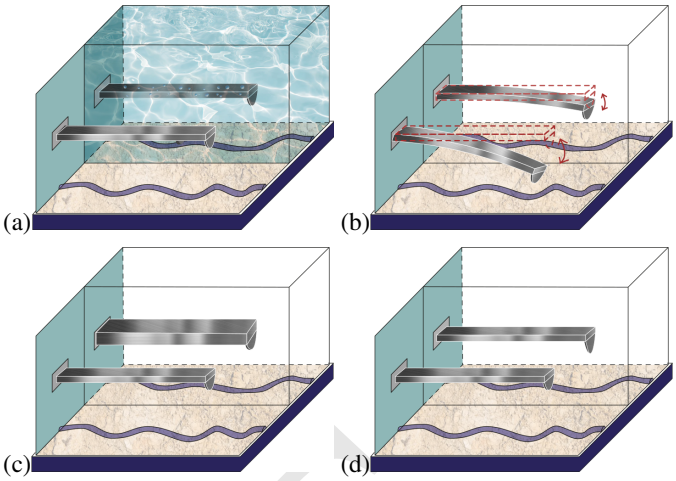

Fig. 1: *Schematic of the tip-sample interface comparing: (a) air at a given Relative Humidity with air; (b) a hard (stiff) cantilever with a soft cantilever; (c) a cantilever with a high volume compared with a cantilever with a small volume; (d) a cantilever with a wide tip with a cantilever with a sharp tip.*

The AFM is widely used for mechanical properties mapping of matter [**?**]. Hence, the first comparison of the four scenarios points out to the force response versus time according to a Hertzian interaction [**?**]. In Figure 2a, we see the humid air (RH = 60.1%) changes the measurement conditions by almost 10%. Using a stiffer cantilever ($k_c = 2.7[N/m]$) will also increase the force by almost 50% from the softer one ($k_c = 0.8[N/m]$), see Figure 2b. Interestingly, the cantilever's volume, a smaller cantilever, results in the highest force by almost doubling the force by almost five folds of the smallest volume (Figure 2c). Finally, the Tip radius difference between 8 and 20 nm will impact the force in roughly 40 pN.

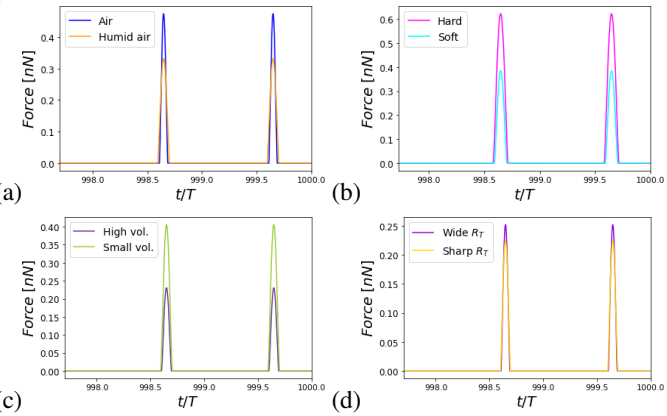

Fig. 2: *Time-varying force for PVA at RH = 60.1% for different cantilevers. The simulations show elastic (Hertz) responses. For each curve, the maximum force value is the peak force. Two complete oscillations are shown corresponding to the conditions in: (a) Fig. 1a, (b) Fig. 1b, (c) Fig. 1c, and (d) Fig. 1d. The simulations were performed for $A_{sp}/A_0 = 0.8$.*

Now, if we consider literature values for different RH [**?**], [**?**], we can evaluate the Peak or Maximum Forces. This force in all cases depicted in Figure 3 shows a monotonically increasing behaviour with the higher Young modulus. Remarkably, the force

varies in a range of 25% from dried PVA to one at RH = 60.1% (see Figure 3a).

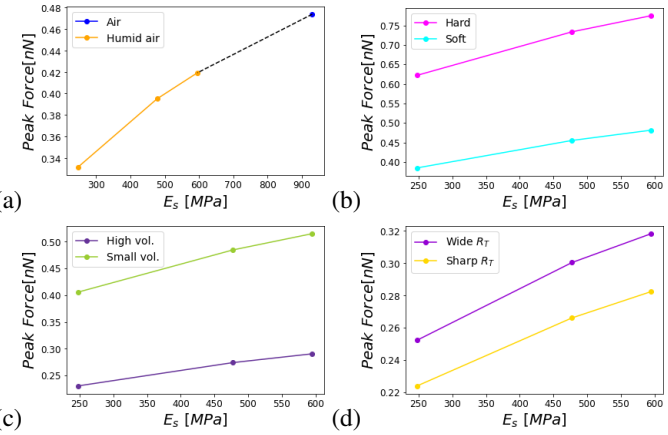

Fig. 3: *Peak force reached for a PVA sample subjected to different relative humidities 0.0%, 29.5%, 39.9% and 60.1% corresponding to the conditions in (a) Fig. 1a, (b) Fig. 1b, (c) Fig. 1c, and (d) Fig. 1d. The simulations were performed for $A_{sp}/A_0 = 0.8$.*

In order to properly describe operational parameters in dynamic AFM we analyze the peak force dependence with the set-point amplitude A_{sp}. In Figure 4, we have the comparison of peak forces for the different cantilevers as a function of A_{sp}. The sensitivity of the peak force varies more with the type of cantilevers with contrary k_c and V_c. Nonetheless, the peak force dependence given by the Hertzian mechanics has a dependence with the square root of the tip radius, and for those Radii on Table 1 are not influencing much the force. However, they could strongly influence resolution [**?**].

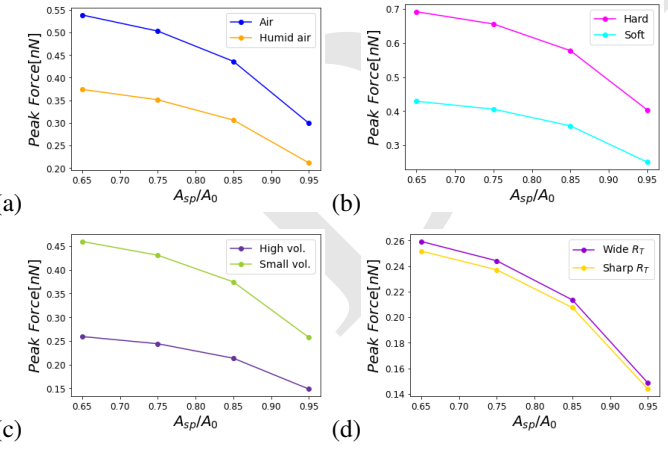

Fig. 4: *Dependence of the maximum force on the set-point amplitude corresponding to the conditions in: (a) Fig. 1a, (b) Fig. 1b, (c) Fig. 1c, and (d) Fig. 1d.*

Figure 5 shows the dependence of the peak force as a function of k_c, V_c, and R_T, respectively, for all the cantilevers listed in Table 1; constituting a graphical summary of the seven analyzed cantilevers for completeness of the analysis.

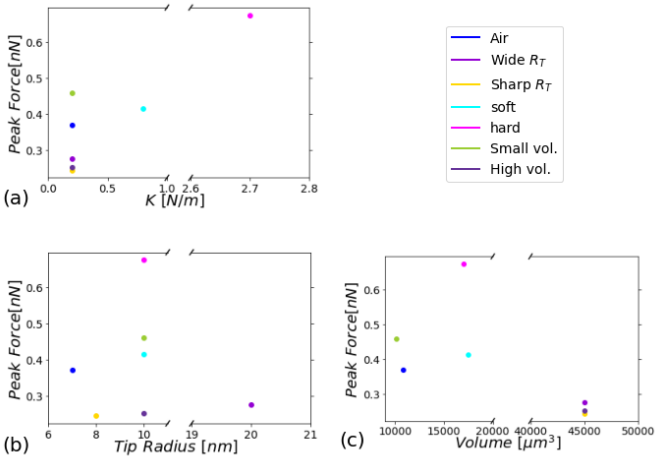

Fig. 5: *Dependence of the maximum force with the most important characteristics of each cantilever, filtering the cantilevers used for the scenarios , the figure shows maximum force dependent on the: (a) force constant k, (b) cantilever tip radius, and (c) cantilever volume, respectively. The simulations were performed for $A_{sp}/A_0 = 0.8$.*

Another way to summarize the results in AFM simulations if to show the Force vs. Distance curves (see Fig. 7), which in these case show exactly how for example a stiffer cantilever may penetrate more into the sample by simple checking the distance cantilever e reaches. On the other hand, it also jumps into the eyes that a cantilever with small volume f has less damping from the environment and thus it also indents more than the ones with higher volume. Although these type of plots are the easiest to make they carry lots of experimental information. In addition, pyDAMPF can plot such 3D figures interactively that enables a detailed comparison of those curves.

As we aim a massive use of pyDAMPF, we also perform the corresponding benchmarks on four different computing platforms, where two of them resembles the standard PC or Laptop found at the labs, and the other two aim to cloud and HPC facilities, respectively (see Table 2 for details).

Figure 7a shows the average run time for the serial and parallel implementation. Despite a slightly higher performance for the case of the HPC cluster nodes, a high-end computer (PC 2) may also reach similar values, which is our current goal. Another striking aspect observed by looking at the speed-up, is the maximum and minimum run times, which notoriously show the on-demand character of cloud services. As their maxima and minima show the highest variations.

To calculate the speed up we use the following equation:

$$S = \frac{t_{total}}{t_{thread}} \tag{1}$$

Where S is the speed up , t_{Thread} is the execution time of a computational thread, and t_{Total} is the sum of times, shown in the table 3. For our calculations we used the highest, the average and the lowest execution time per thread.

Limitations

The main limitation of dynamic AFM simulators based in continuum modeling is that some times a molecular behaviour is overlooked. Such a limitation comes from the multiple time and length scales behind the physics of complex systems, as it is the

Tabla 1: *Data used for Figs. 2, 3 and 4 with an $A_0 = 10[nm]$. Observe that the quality factor and Young's modulus have three different values respectively for RH1 = 29.5%, RH2 = 39.9% y RH3 = 60.1%. ** The values presented for Quality Factor (Q) were calculated at Google Colaboratory notebook Q_calculation,using the method proposed by [?], [?]*

Scenario	Model	R[nm]	Q^{**}	E_s[MPa]	k[N/m]	f[kHz]	Volume [μm^3]	Cantilever
Air	PPP-CONTSCR	7	28.65	930	0.2	25	10800	a
Humid air	PPP-CONTSCR	7	28.74 - 28.77 - 28.83	595.6 - 477.8 - 247.8	0.2	25	10800	a
Hard	XCS11-B	10	167.73 - 167.89 - 168.28	595.6 - 477.8 - 247.8	2.7	80	17010	e
Soft	HQ:CSC37-A	10	95.15 - 95.24 - 94.45	595.6 - 477.8 - 247.8	0.8	40	17500	d
High vol.	PPP-XY-CONTR	10	48.37 - 48.41 - 48.51	595.6 - 477.8 - 247.8	0.2	13	45000	g
Small vol.	SD-PXL-CONT-SC	10	23.61 - 23.64 - 23.68	595.6 - 477.8 - 247.8	0.2	8	10125	f
Wide R_T	SD-CONT-SiN*	20	48.37 - 48.41 - 48.51	595.6 - 477.8 - 247.8	0.2	13	45000	b
Sharp R_T	CONTR	8	48.37 - 48.41 - 48.51	595.6 - 477.8 - 247.8	0.2	13	45000	c

Tabla 2: *Computers used to run pyDAMPF and Former work [?], * the free version of Colab provides this capability, there are two paid versions which provide much greater capacity, these versions known as Colab Pro and Colab Pro+ are only available in some countries.*

Computer	CPU	RAM	pyDAMPF	Forme Work
PC 1	Core i5-5200U @ 2.2 GHz	8 GB	✓	✓
PC 2	Ryzen 7 4800H @ 2.9 GHz	16GB	✓	✓
Colab	Intel(R) Xeon(R) CPU @ 2.2GHz	6.4GB*	✓	-
Cluster	Intel(R) Xeon(R) E5-2620 v4 @ 2.1GHz	32 GB	✓	-

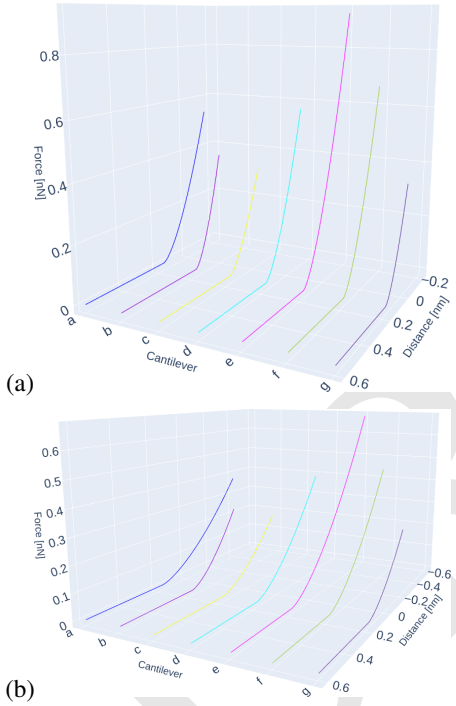

Fig. 6: *Three-dimensional plots of the various cantilevers provided by the manufacturer and those in the pyDAMPF database that establish a given maximum force at a given distance between the tip and the sample for a PVA polymer subjected to : (a) RH= 0% with E = 930 [MPa]. and (b) RH = 60.1% with E = 248.8 [MPa].*

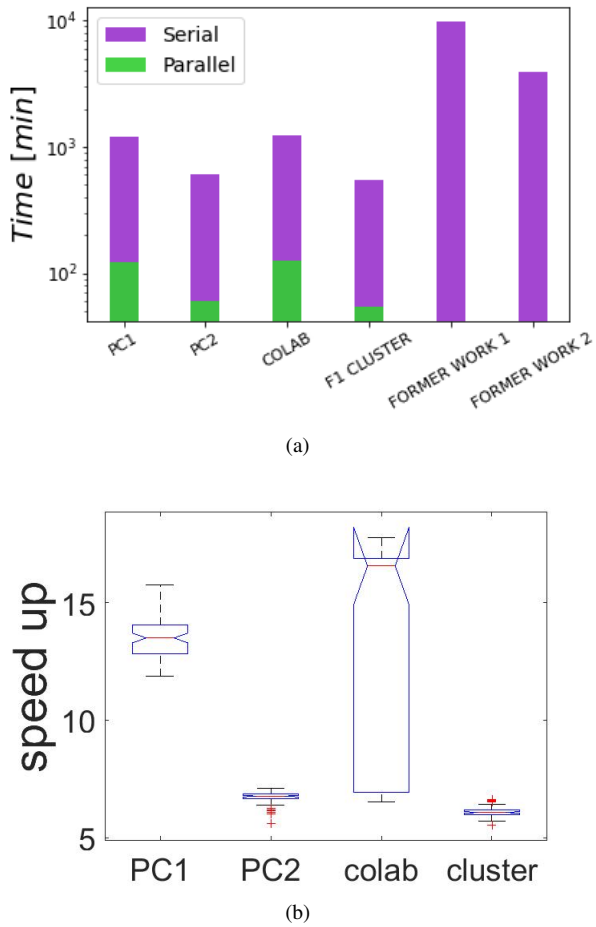

Fig. 7: *Comparison of execution times: (a) Comparison of times taken by both the parallel method and the serial method. (b) Speed up parallel method.*

case of polymers and biopolymers. In this regard, several efforts on the multiscale modeling of materials have been proposed, joining mainly efforts to stretch the multiscale gap [?]. We also plan to do so, within a current project, for modeling the polymeric fibers as molecular chains and providing "feedback" between models from a top-down strategy. Code-wise, the implementation will be also gradually improved. Nonetheless, to maintain scientific code is a challenging tasks. In particular without the support for our students once they finish their thesis. In this respect, we will seek for software funding and more community contributions.

Tabla 3: *Execution times per computational thread, for each computer. Note that each Thread consist of 9 simulation cases, with a sum time showing the total of 90 cases for evaluating 3 different Young moduli and 30 cantilevers at the same time.*

Time of execution for thread [min]				
Thread	PC 1	PC 2	Colab	Cluster
1	123.81	60.19	151.10	55.19
2	116.12	61.23	151.94	54.96
3	123.73	60.09	148.94	52.87
4	120.41	60.26	62.01	54.24
5	122.27	60.35	61.39	55.48
6	119.93	59.97	59.91	52.96
7	122.70	60.68	149.89	54.72
8	124.19	60.55	166.58	54.54
9	121.34	60.99	136.69	54.36
10	123.36	60.57	152.88	57.75
Average	121.78	60.49	124.13	54.71
Sum	1217.85	604.89	1241.33	547.07

Future work

There are several improvements that are planned for pyDAMPF.

- We plan to include a link to molecular dynamics simulations of polymer chains in a multiscale like approach.
- We plan to use experimental values with less uncertainty to boost semi-empirical models based on pyDAMPF.
- The code is still not very clean and some internal cleanup is necessary. This is especially true for the Python backend which may require a refactoring.
- Some AI optimization was also envisioned, particularly for optimizing criteria and comparing operational parameters.

Conclusions

In summary, pyDAMPF is a highly efficient and adaptable simulation tool aimed at analyzing, planning and interpreting dynamic AFM experiments.

It is important to keep in mind that pyDAMPF uses cantilever manufacturers information to analyze, evaluate and choose a certain nanoprobe that fulfills experimental criteria. If this will not be the case, it will advise the experimentalists on what to expect from their measurements and the response a material may have. We currently support multi-thread execution using in-house development. However, in our outlook, we plan to extend the code to GPU by using transpiling tools, like compyle [**?**], as the availability of GPUs also increases in standard workstations. In addition, we have shown how to reuse a widely tested Fortran code [**?**] and wrap it as a python module to profit from pythonic libraries and interactivity via Jupyter notebooks. Implementing new interaction forces for the simulator is straightforward. However, this code includes the state-of-the-art contact, viscous, van der Waals, capillarity and electrostatic forces used for physics at the interfaces. Moreover, we plan to implement soon semi-empirical analysis and multiscale modeling with molecular dynamics simulations.

Acknowledgments

H.V.G thanks the financial support by the Slovenian Research Agency (Funding No. P1-0055). We gratefully acknowledge the fruitful discussions with Tomas Corrales.

REFERENCES

[Ble90] Guy E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, November 1990.

[CGK11] Bryan Catanzaro, Michael Garland, and Kurt Keutzer. Copperhead: compiling an embedded data parallel language. *ACM SIGPLAN Notices*, 46(8):47–56, February 2011. URL: https://doi.org/10.1145/2038037.1941562, doi:10.1145/2038037.1941562.

[Ram16] Prabhu Ramachandran. PySPH: a reproducible and high-performance framework for smoothed particle hydrodynamics. In Sebastian Benthall and Scott Rostrup, editors, *Proceedings of the 15th Python in Science Conference*, pages 127 – 135, 2016.

[RP+19] Prabhu Ramachandran, , Kunal Puri, Aditya Bhosale, Dinesh Adepu, Abhinav Muta, Pawan Negi, Rahul Govind, Suraj Sanka, Pankaj Pandey, Chandrashekhar Kaushik, Anshuman Kumar, Ananyo Sen, Rohan Kaushik, Mrinalgouda Patil, Deep Tavker, Dileep Menon, Vikas Kurapati, Amal S Sebastian, Arkopal Dutt, and Arpit Agarwal. PySPH: a Python-based framework for smoothed particle hydrodynamics. *arXiv preprint arXiv:1909.04504*, 2019. URL: https://arxiv.org/abs/1909.04504.

[Sch15] Daniel V. Schroeder. Interactive molecular dynamics. *American Journal of Physics*, 83(3):210–218, February 2015. Publisher: American Association of Physics Teachers. doi:10.1119/1.4901185.