The Extra Project Assignment - Reproducibility of Published Results
Vinícius Silva Pacheco

To get the five extra points, the student must:

1. Choose a paper presented in a recent compiler related conference, such as CGO, PLDI or CC.

2. Download the material that the authors have made publicly available for the paper. If there is no such material, but the student still wants to reproduce the paper's results, he/she can write to the authors of the paper, asking for their implementation.

3. Run the implementation, trying to reproduce at least one of the experiments in the paper. Even if it is not possible to reproduce the experiments, the student can still write a report about his/her tries, to claim the extra points.

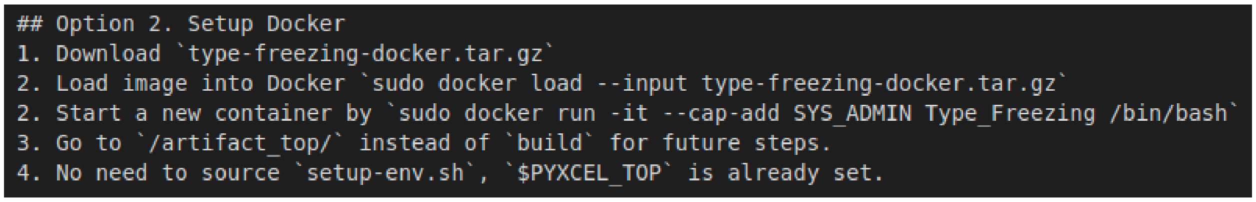
4. Write a short report about the entire procedure. This report must contain the URL of the material that has been downloaded, plus a brief description of the student's experience with it. If the student has not been able to reproduce those experiments, than the report must explain the reasons for this failure.

I chose the following paper:

Lin Cheng, Berkin Ilbeyi, Carl Friedrich Bolz-Tereick, and Christopher Batten. 2020. Type freezing: exploiting attribute type monomorphism in tracing JIT compilers. In Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization (CGO 2020). Association for Computing Machinery, New York, NY, USA, 16–29. DOI:https://doi.org/10.1145/3368826.3377907

The authors made the material publicly available, and an appendix in the paper describes how to use them to reproduce the experiments. To get the material, you can use the link https://doi.org/10.5281/zenodo.3542289. It contains three files, a README explaining in more details how to get the results, a docker image containing an environment with most of the prerequisites already installed, and a source file to build the project in any other machine.

I chose to use the docker image, in this case the installation is straightforward. From the README:

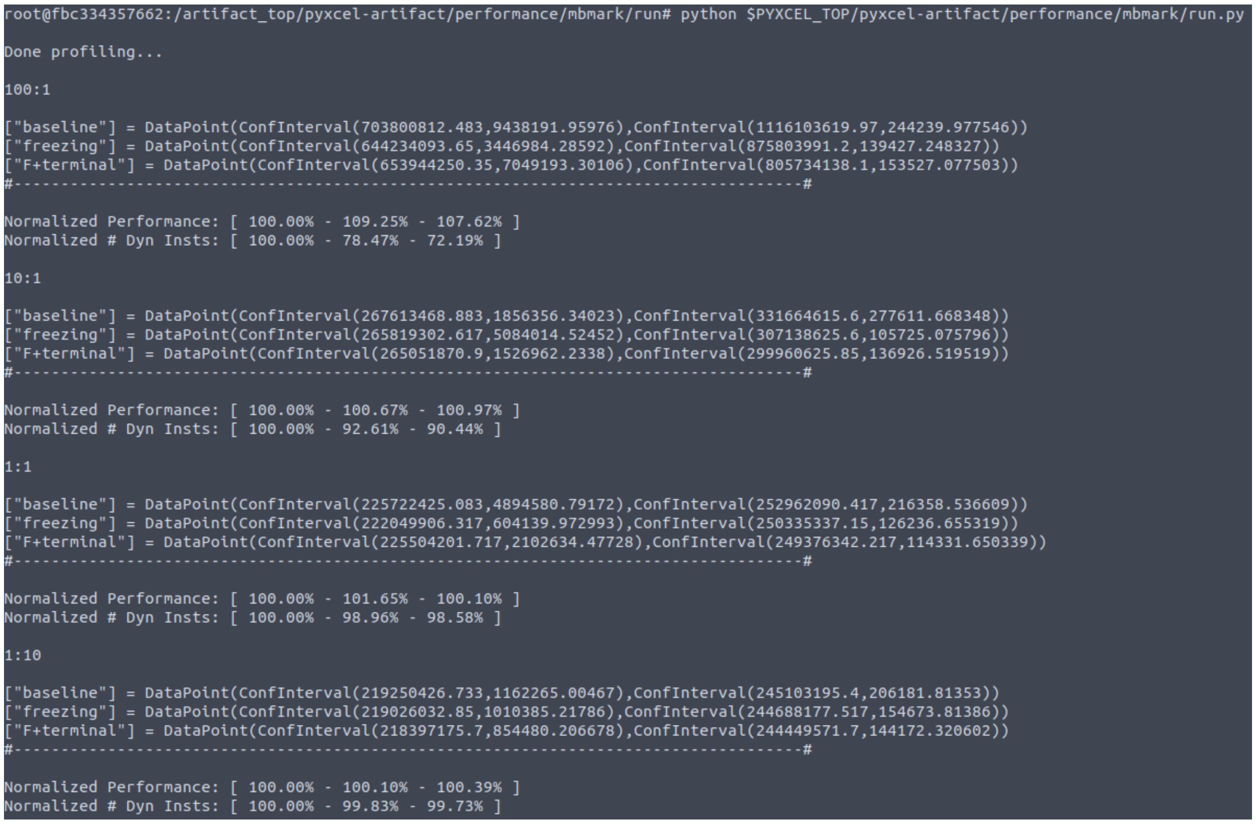


```
## Option 2. Setup Docker
1. Download `type-freezing-docker.tar.gz`
2. Load image into Docker `sudo docker load --input type-freezing-docker.tar.gz`
2. Start a new container by `sudo docker run -it --cap-add SYS_ADMIN Type_Freezing /bin/bash`
3. Go to `/artifact_top/` instead of `build` for future steps.
4. No need to source `setup-env.sh`, `$PYXCEL_TOP` is already set.
```

The only prerequisite not installed in the image was *perf,* as it depends on the kernel version of the host machine, but installing it is pretty easy.

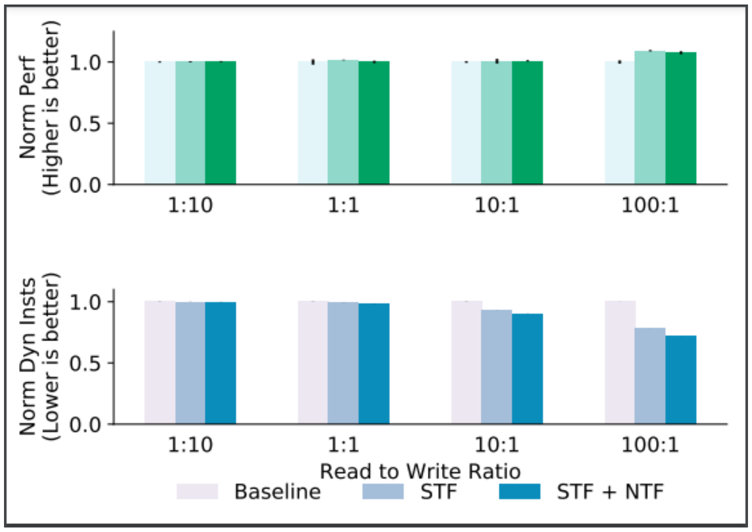
- Reproducing the results of Figure 10 (Section 4.3)

We do it by running the proposed technique on microbenchmarks:



```
root@fbc334357662:/artifact_top/pyxcel-artifact/performance/mbmark/run# python $PYXCEL_TOP/pyxcel-artifact/performance/mbmark/run.py

Done profiling...

100:1

["baseline"] = DataPoint(ConfInterval(703800812.483,9438191.95976),ConfInterval(1116103619.97,244239.977546))
["freezing"] = DataPoint(ConfInterval(644234093.65,3446984.28592),ConfInterval(875803991.2,139427.248327))
["F+terminal"] = DataPoint(ConfInterval(653944250.35,7049193.30106),ConfInterval(805734138.1,153527.077503))
#-------------------------------------------------------------------------------------#

Normalized Performance: [ 100.00% - 109.25% - 107.62% ]
Normalized # Dyn Insts: [ 100.00% - 78.47% - 72.19% ]

10:1

["baseline"] = DataPoint(ConfInterval(267613468.883,1856356.34023),ConfInterval(331664615.6,277611.668348))
["freezing"] = DataPoint(ConfInterval(265819302.617,5084014.52452),ConfInterval(307138625.6,105725.075796))
["F+terminal"] = DataPoint(ConfInterval(265051870.9,1526962.2338),ConfInterval(299960625.85,136926.519519))
#-------------------------------------------------------------------------------------#

Normalized Performance: [ 100.00% - 100.67% - 100.97% ]
Normalized # Dyn Insts: [ 100.00% - 92.61% - 90.44% ]

1:1

["baseline"] = DataPoint(ConfInterval(225722425.083,4894580.79172),ConfInterval(252962090.417,216358.536609))
["freezing"] = DataPoint(ConfInterval(222049906.317,604139.972993),ConfInterval(250335337.15,126236.655319))
["F+terminal"] = DataPoint(ConfInterval(225504201.717,2102634.47728),ConfInterval(249376342.217,114331.650339))
#-------------------------------------------------------------------------------------#

Normalized Performance: [ 100.00% - 101.65% - 100.10% ]
Normalized # Dyn Insts: [ 100.00% - 98.96% - 98.58% ]

1:10

["baseline"] = DataPoint(ConfInterval(219250426.733,1162265.00467),ConfInterval(245103195.4,206181.81353))
["freezing"] = DataPoint(ConfInterval(219026032.85,1010385.21786),ConfInterval(244688177.517,154673.81386))
["F+terminal"] = DataPoint(ConfInterval(218397175.7,854480.206678),ConfInterval(244449571.7,144172.320602))
#-------------------------------------------------------------------------------------#

Normalized Performance: [ 100.00% - 100.10% - 100.39% ]
Normalized # Dyn Insts: [ 100.00% - 99.83% - 99.73% ]
```

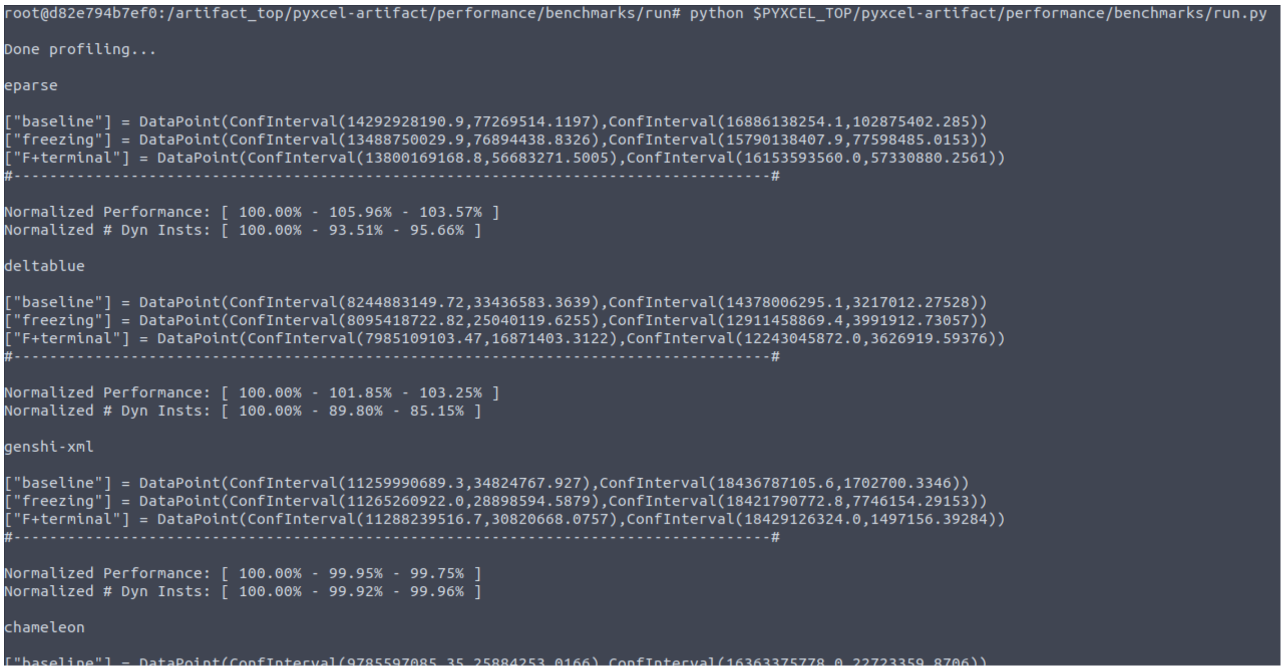
We then can generate a graph, similar to the one in the paper:



As we can see, the results are indeed the same as the authors'.

- Reproducing the results of Figure 11 (Section 5.1)

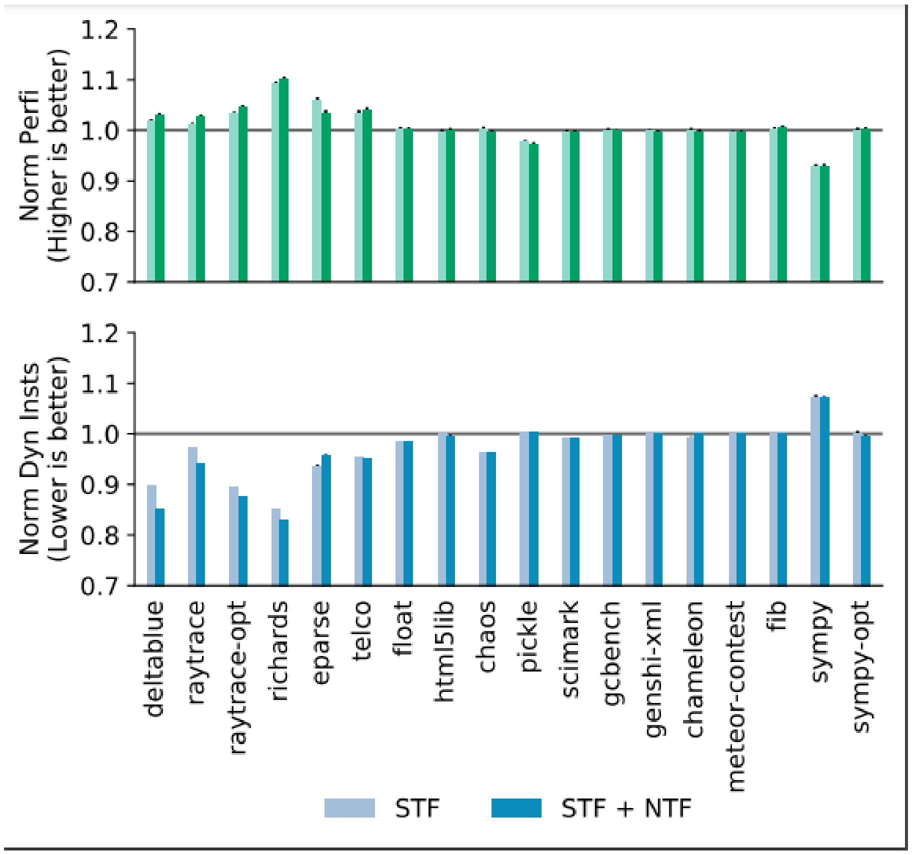
We do it by running the proposed technique on the PyPy benchmarks:



```
root@d82e794b7ef0:/artifact_top/pyxcel-artifact/performance/benchmarks/run# python $PYXCEL_TOP/pyxcel-artifact/performance/benchmarks/run.py

Done profiling...

eparse

["baseline"] = DataPoint(ConfInterval(14292928190.9,77269514.1197),ConfInterval(16886138254.1,102875402.285))
["freezing"] = DataPoint(ConfInterval(13488750029.9,76894438.8326),ConfInterval(15790138407.9,77598485.0153))
["F+terminal"] = DataPoint(ConfInterval(13800169168.8,56683271.5005),ConfInterval(16153593560.0,57330880.2561))
#------------------------------------------------------------------------------------#

Normalized Performance: [ 100.00% - 105.96% - 103.57% ]
Normalized # Dyn Insts: [ 100.00% - 93.51% - 95.66% ]

deltablue

["baseline"] = DataPoint(ConfInterval(8244883149.72,33436583.3639),ConfInterval(14378006295.1,3217012.27528))
["freezing"] = DataPoint(ConfInterval(8095418722.82,25040119.6255),ConfInterval(12911458869.4,3991912.73057))
["F+terminal"] = DataPoint(ConfInterval(7985109103.47,16871403.3122),ConfInterval(12243045872.0,3626919.59376))
#------------------------------------------------------------------------------------#

Normalized Performance: [ 100.00% - 101.85% - 103.25% ]
Normalized # Dyn Insts: [ 100.00% - 89.80% - 85.15% ]

genshi-xml

["baseline"] = DataPoint(ConfInterval(11259990689.3,34824767.927),ConfInterval(18436787105.6,1702700.3346))
["freezing"] = DataPoint(ConfInterval(11265260922.0,28898594.5879),ConfInterval(18421790772.8,7746154.29153))
["F+terminal"] = DataPoint(ConfInterval(11288239516.7,30820668.0757),ConfInterval(18429126324.0,1497156.39284))
#------------------------------------------------------------------------------------#

Normalized Performance: [ 100.00% - 99.95% - 99.75% ]
Normalized # Dyn Insts: [ 100.00% - 99.92% - 99.96% ]

chameleon

["baseline"] = DataPoint(ConfInterval(9785597085.35,25884253.0166),ConfInterval(16363375778.0,22723359.8706))
```

The above image shows some of the output.

And we can also generate the graph:



Once again, the results convey the same information as the ones in the paper.

Overall I had a nice experience working with the artifacts of the paper, the scripts were easy to use and the guidelines in the README file really helped. I was able to successfully reproduce the experiments.