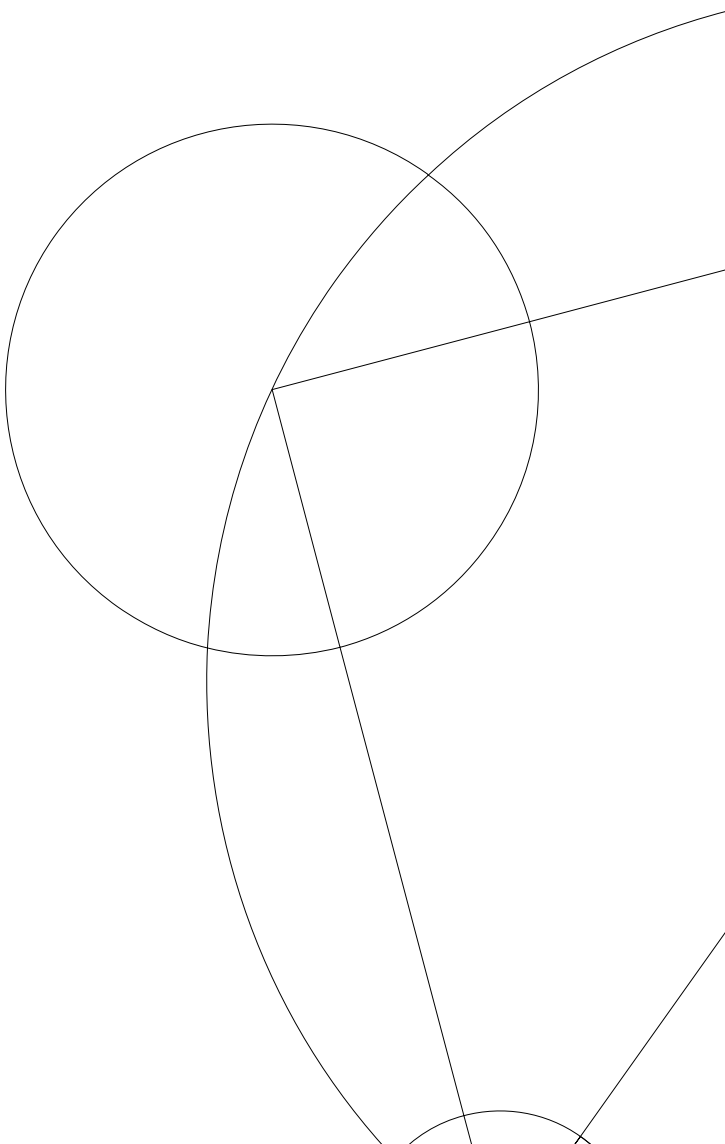
UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

Check-pointing Long-running Applications in Python

Bachelor project

Jacob Olesen (slc458)
Leeann Quynh Do (hlp848)



Kenneth Skovhede

Handed in: March 4, 2020

1 Abstract

2 Forewords

Contents

3 Introduction

WELCOME TO A BACHELOR PROJECT WITH JACOB AND LEEANN!!

TODO: INSERT ADDITIONAL TEXT HERE AS INTRODUCTION

3.1 Motivation

The motivation behind this project is primarily to aid researchers and scientists in their studies, as most would currently need to write their own checkpointing function in a program. This is not easy, and is not meant to be part of their work, so a dynamic `python` library which can checkpoint every python application is highly desirable. This is primarily relevant in terms of research and work which uses simulations such as molecular dynamics or climate, which can run for weeks, months or even years. Having an easy, dynamic, and efficient way to checkpoint these programs would make their lives much easier.

4 Theory

4.1 Serialization and deserialization

Serialization of data is the process of translating a state of an object or a data structure into a format, e.g. a byte stream, that can be stored on a file or transferred, and can later again be reconstructed, even on another machine. This opposite operation of translating a byte stream into an object or a data structure, is called deserialization [1].

4.2 Python

Interpreted vs compiled
no program counter.

4.3 Existing solutions

`python` already have some existing solutions to serializing and deserializing objects or data structures. These are the `python` modules `pickle` and `dill`.

4.3.1 `pickle`

The `pickle` module is a module in `python` for serialization and deserialization. The module uses the terms `pickling` and `unpickling` .
The format that the `pickle` module translate data to, is specific to `python`. This means that programs written in anything else than `python`, are not able to reconstruct pickled `python` objects [2].

Pickle is written in pure `python`, but has an optimized version which is written in pure C called `cPickle`, and is up to 1000 times faster. `Pickle` attempts to import and use `cPickle`, but falls back to the pure `python` version in case of errors. `cPickle` does not support quite as many features and types as `Pickle`.
https://docs.python.org/3/library/pickle.htmlperformance
https://docs.python.org/2/library/pickle.html
https://docs.python.org/release/3.1.5/whatsnew/3.0.html

4.3.2 `dill`

`dill` is an extension to the `pickle`-module. In addition to pickling and unpickling `python` objects, `dill` also allows the user to save the state of the interpreter

session, so that if the program were to continue on another machine, it is possible to continue from the saved state of the 'original' interpreter.
`dill` is very flexible, as it allows arbitrary user defined functions and classes to be serialized. The user thus has to be aware if the data they are trying to serialize is trustworthy, as `dill` does not protect against maliciously constructed data.

Some types that `dill` does not support are `frame`, `generator` and `traceback` [3].

5 Implementation

In order to have a proper, iteratively structured program for use and benchmarking throughout this project, we were advised to use a heat equation simulation program written specifically for benchmarking [4]. This is an ideal program for this project, as it is structured the same way as the simulation programs which we aim to be able to checkpoint.

5.1 `dill`

Our initial implementation efforts were made in collaboration with our initial research in order to gain a proper understanding of exactly what was happening under the hood of the program, and, more specifically, how `dill` works. `dill` is initially able to make a perfect checkpoint at an arbitrary point in the program, but due to `python` having no program counter, the program is unable to continue from a specific instruction. Thus, we had to rewrite the program to be able to properly continue from a checkpoint by calling a function. What this meant in practice was that the `main()` function was rewritten from

```
def main():
    H = bench.args.size[0]
    W = bench.args.size[0]
    I = 50000

    grid = bench.load_data()
    if grid is None:
            grid = init_grid(H, W, dtype=bench.dtype)
    ...
```

to

```
def main():
    global H, W, I, grid

    if grid is None:
            grid = init_grid(H, W, dtype=bench.dtype)
    ...
```

This was done to allow resuming the program from a checkpoint by simply calling `main()`. Prior to this change, calling `main` after de-serializing a state would simply re-initialize the variables used, which in practice means that the program starts over.

This allowed us to define the first limitation of `dill`.

fase0: vi fik udleveret et godt iterativt program af vores vejleder, ligesom de fleste scientific programmer.
for at få det til at virke blev vi nødt til at gøre variable globale. altså initialisere dem uden for de funktioner som bliver kaldt, således at de ikke bliver overskredet når de bliver kaldt igen.

6 Testing

7 Discussion

7.1 Program counter

How did it affect us, that there was no program counter?

7.2 Security

Most efficient serialization and deserialization modules in `python` are vulnerable to arbitrary code execution as they store the data as bytes and load them without any 'validation'. This can, of course, be a major point of concern in certain environments and use cases. However, we do not consider this to be a cause of concern in our project, as the purpose of this program is for users to only serialize and deserialize their own programs in case of errors during runtime, which should never lead to any dangerous code getting executed.

8 Conclusion

9 Appendix

References

[1] March 4, 2020. URL: https://en.wikipedia.org/wiki/Serialization.

[2] March 4, 2020. URL: https://docs.python.org/3/library/pickle.html.

[3] March 4, 2020. URL: https://pypi.org/project/dill/.

[4] February 7, 2020. URL: https://benchpress.readthedocs.io/autodoc_benchmarks/heat_equation.html.