ECS Semester Project

Python code for numerically finding solutions for surface seismic waves

Chirayu Gupta

IISER Pune

December 20, 2021

Objective of the project

- Translating a legacy Fortran code(called srgram) into python and introduce any possible changes in the algorithm to improve readability and performance.
- Parallelize earthsr(a code I worked on during the summer).

Srgramf

Srgramf is a program for computing seismic surface wave Green functions and synthetic seismogram in 1-D Earth models.

Earthsr

Earthsr is a program to compute Rayleigh and Love wave dispersion curves in a one-d medium for sources and receivers at specified depths. Relation to srgram?

Work done on Earthsr

The translation work of earthsr was finished during the summer. However certain updates were made to the code to improve performance and readability of the code.

- The whole code was parallelized thus reducing the runtime of the code by a factor of n.
- The procedural like code was broken up into various modules including wrapper classes, helper classes and provided extra provisions in the modification of parameters
- Other minor changes were made as suggested by Sir Arjun Datta.

Brief overview of srgramf

- Surface wave greens function program to be used with a moment tensor as specified for normal modes.
- Works in the frequency domain
- 6 Moment tensor specified in input file or strike/dip/rake to recover source mechanism.
- Algorithm uses dispersion values from earthsr along with momentum tensor and others
- Algorithm works in accordance to equations form mendiguren's paper(1977)
- This algorithm also agrees with Aki and Richards derivation (1981).

Work done on srgram

- Interpreted and translated the 1000+ lines of code.
- Changes in way input worked. Original code manually wrote a parser for input file to read input parameters(approx. 300 lines of code). Now uses **.ini** files
- Added functionality of getting solutions in the time domain by using DFT(through numpy), revised values of constants of nature(from libraries like numpy, astropy), converted all arrays to numpy arrays/ numpy matrices(Use C/C++ backend)
- No more common blocks! Use global variables/ objects instead
- OOP $>$ Procedural Programming.

Example of a minute change

```python
class mkhomogsrparams:
    def __init__(self,angle) -> None:
        self.angle = angle
        self.cos = math.cos(angle)
        self.sin = math.sin(angle)
        self.cos2 = math.cos(angle*2)
        self.sin2 = math.sin(angle*2)
        self.c2p1 = 0.5*(1+self.cos2)
        self.c2m1 = 0.5*(1 - self.cos2)
        self.f1 = f1(angle)
    def f1(angle):
        #Some operation here
        pass

mkhomogsr(angle) #Approch 1( Original code)
mkhomogsr(mkhomogsrparams(angle)) #Approch 2(requires OOP)

def mkhomogsr(angle):

    anglesin=math.sin(angle) #Approch 3: Increases number of varibles and decreases readability, modularity and increases repetitiveness especially if multiple angles and more complex operations involved

    for i in range(..,..):
#         involves use of sin/cos/f1 in multiple places
        math.sin(angle) #Approch 1:Calculates sin multiple times
        angle.sin #Uses precalculated values of sin(no change in modularity/ readability of code
```

Impact of the minute change

- Output of a profiler on the code.
- Numbers show number of calls to the functions and the time taken
- Time spend on sin functions will be reduced if already calculated values are used since number of calls will reduce.
- Will scale if used along with parallelizing

```
      7    0.000    0.000    0.000    0.000 {built-in method builtins.vars}
    288    0.000    0.000    0.000    0.000 {built-in method from_bytes}
     48    0.000    0.000    0.000    0.000 {built-in method fromkeys}
    149    0.002    0.000    0.002    0.000 {built-in method io.open_code}
      4    0.009    0.002    0.009    0.002 {built-in method io.open}
      2    0.000    0.000    0.000    0.000 {built-in method maketrans}
    117    0.013    0.000    0.013    0.000 {built-in method marshal.loads}
 540841    0.061    0.000    0.061    0.000 {built-in method math.atan}
   2843    0.001    0.000    0.001    0.000 {built-in method math.copysign}
1818534    0.153    0.000    0.153    0.000 {built-in method math.cos}
2916183    0.221    0.000    0.221    0.000 {built-in method math.exp}
      2    0.000    0.000    0.000    0.000 {built-in method math.log}
1818534    0.151    0.000    0.151    0.000 {built-in method math.sin}
5825531    0.391    0.000    0.391    0.000 {built-in method math.sqrt}
    124    0.000    0.000    0.000    0.000 {built-in method numpy.array}
      1    0.000    0.000    0.000    0.000 {built-in method numpy.asarray}
      1    0.000    0.000    0.000    0.000 {built-in method numpy.core._mul
```

Future Plans/Possible improvements[1]

Current code is slow !

Python is not a compiled language and doesnt support multithreading(GIL=1).

1. Link earthsr and srgram into one module.
2. Dockerise the code so that the problem the original code had with dependencies being obselute can be resolved. Of course this can also be done by using virtual environments.
3. Parallelize srgram
4. Performance can be further improved by making the code run on GPU. This can provide a tremendous improvement in performance since a GPU can run upto millions of threads at once.
5. Further improve parallelization of earthsr(Currently parts of code that generate a race condition on parallelizing are not parallized. The use of faster compilers such as pypy3 can also be considered.

[1] Points 4 and 5 can reduce the readability of the code