Improving Python Script Execution Efficiency with PyPy

Jianlin Su

June 11, 2014

In the article "Sum of Primes below Two Million and Sum of the First Two Million Primes", we used Python to calculate the sum of primes below two million and the sum of the first two million primes. The execution times obtained in Python 3.3 were as follows:

> Sum of primes below two million:
> 142913828922
> time: 2.4048174478605646
>
> Sum of the first two million primes:
> 31381137530481
> time: 46.75734807838953

Consequently, I looked for ways to improve the execution efficiency of Python scripts. I felt that there was relatively little room for optimization in terms of algorithms, so I considered optimizations at the executor level. During my search, I accidentally came across a term—Psyco! This is an external module for Python that, once imported, can speed up the execution of `.py` scripts. There are articles online such as "Using Psyco to Make Python Run as Fast as C" and "Using Psyco to Speed Up Python Programs," which suggest that Psyco is indeed a viable option. I was eager to try it, but later learned that Psyco stopped development in 2012 and only supports up to Python 2.4. It has since been succeeded by PyPy. Therefore, I downloaded PyPy.

PyPy is no longer just an external module for Python, but an independent interpreter. This means we can execute `.py` scripts using the command `pypy prime.py`. It is currently based on Python 2.7.6 and has a Windows version. How is its efficiency? Take a look:

> Sum of primes below two million:
> 142913828922
> ('time:', 0.226615647130835)
>
> Sum of the first two million primes:
> 31381137530481
> ('time:', 9.033084048872064)

One was reduced from 2.4 seconds to 0.3 seconds, and the other from 47 seconds to 9 seconds—the speed increased several times over! Executing these two scripts with PyPy is indeed satisfying.

Does this mean all Python scripts should be switched to PyPy? Not necessarily. There are many cases where Python still runs faster. For example, consider the following script that calculates 10000! (calculation only, no output):

```
import time
start=time.clock()
s=1
for i in range(1,10000+1):
```

```
    s=s*i

end=time.clock()
print(end-start)
```

In Python 3.3, this takes only 0.06 seconds, while in PyPy it takes 0.3 seconds. If calculating 100000!, PyPy takes 39 seconds, while Python 3.3 takes only 7 seconds!

Regarding PyPy and Python, here is the best answer from Veedrac:

> As others have mentioned, PyPy has very weak C extension compatibility. It supports C extensions, but they run slower than in Python itself. Therefore, many modules themselves require the use of CPython.
>
> Data processing on CPython with NumPy is excellent, satisfying those who require both speed and the use of libraries like Pandas and SciPy for data analysis tasks.
>
> So, PyPy either does not support or has weak support for C extensions, or it slows down those data processing speeds. It simply cannot compete with CPython, which is both fast enough and easy to use.
>
> Second, Python 3 support is still in the experimental stage. Those using the latest features of Python are likely unwilling to give up those fresh and novel functions.
>
> Third, PyPy is not actually fast for "scripts," and most people using Python are writing scripts. These scripts are short programs. PyPy's greatest advantage is its Just-In-Time (JIT) compiler for long-running, simple numerical processing. To put it bluntly, PyPy's pre-compilation processing time is much longer than CPython's.
>
> Fourth, inertia. Moving to PyPy requires re-equipping machines. For many users or organizations, this is too much extra work.

However, for me, my current programming mainly involves scientific computing programs with a large number of repetitive calculations, where PyPy can indeed greatly improve efficiency! Therefore, I will likely try to use PyPy for execution as much as possible in the future.