Speed Comparison of C vs Python 2 vs PyPy2 vs Swift 4

Sieve of Eratosthenes

이재호

2018년 8월 17일

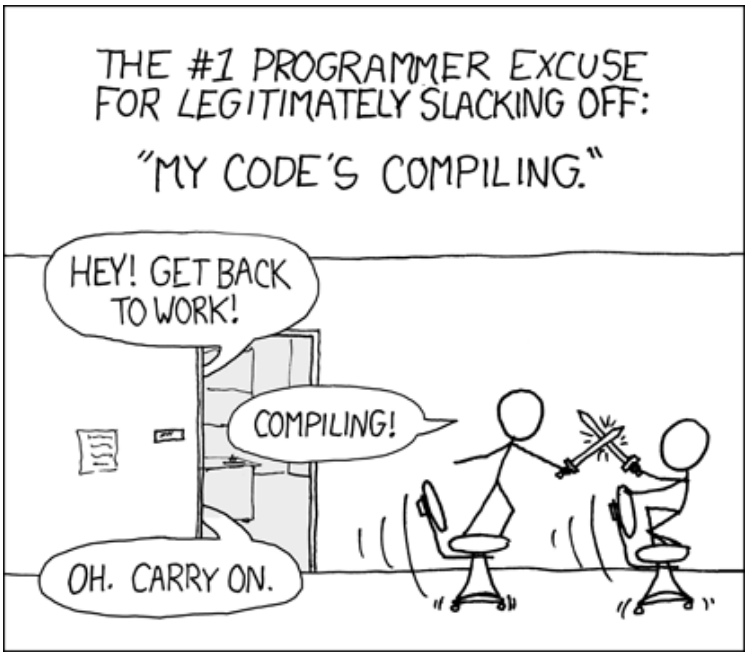

그림 1: Compiling (`https://xkcd.com/303/`)

Swift로 Eratosthenes의 체를 구현하였는데, Python(의 PyPy implementation)보다 실행 속도가 무려 10배 정도 느린 것을 확인하였다. 이에 따라 1. Swift가 굉장히 비효율적인 언어이거나, 2. 아직 Swift에 익숙하지 않아 최적화된 방식으로 작성하지 못한 것이라고 생각하였다.

한편 Swift로 Project Euler의 문제를 풀던 중 runtime error가 발생하는데도 불구하고, error message가 표시되지 않는 문제가 있었다. 이에 AppCode configuration을 확인해보니 release mode로 compile이 되고 있었고, 정상적으로 error message를 표시하려면 debug mode로 compile해야 했다. 그런데 위에서 이 설정이 문제가 되었던 것이, debug mode로 compile하면 debugging이 가능하도록 error message가 표시되지만 실행 속도는 현저히 느려지기 때문이다. 다시 debugging을 완료한 후 release mode로 compile을 하니 속도가 10배 정도 빨라졌다.

속도 문제는 해결하였지만, PyPy와 Swift의 속도가 엇비슷한 것을 보고 Swift의 속도가 어느 정도 빠른 것인지, 약간의 research를 해보았다. 검색을 해보니 Swift가 아주 빠른 것은 아니지만,

점점 최적화를 하여 최신 version인 Swift 4는 속도 문제가 상당히 개선되었다고 한다. 그래서 직접 C, PyPy, 그리고 Swift로 Eratosthenes의 체를 구현하여 실행 속도를 비교해보았다. C는 Apple LLVM 9.1.0로 compile했으며, Python은 Python 2.7.13의 PyPy v6.0.0을, Swift는 Swift 4.1.2를 사용하였다. Python의 경우 CPython 2.7.15에 대해서도 함께 시간을 측정하였다. 모두 100 000 000까지의 소수를 구하는 체를 구현하였고, 출력 시간으로 인한 병목 현상을 없애기 위해 출력은 주석 처리하였다. CPython을 제외하고는 100번씩 진행한 평균 시간을 측정하였다.

1 C Implentation–Base Case

C는 비교적 기계어에 근접한 low-level 언어임으로 실행 속도가 가장 빠를 것으로 기대할 수 있다. 표 1에 C언어로 100 000 000까지의 소수를 Eratosthenes의 체를 사용해 구하는데 걸리는 시간이 나와있다. 최적화를 하지 않은 `-O0` option flag와 `-O1`, `-O2`, `-O3`, 그리고 `-Os` 최적화 option flag를 적용한 경우에 대해서 측정하였다. 최적화를 하지 않았을 경우에는 평균 2초의 수행시간이 걸렸으며, 최적화를 하였을 경우에는 1.5초 정도의 시간이 걸렸다.

표 1: Average execution time of sieve of Eratosthenes of 100 runs. Written in C, using `-O0`, `-O1`, `-O2`, `-O3`, and `-Os` GCC optimization option flag.

Option	Average Time [s]
-O0	1.919 700
-O1	1.578 928
-O2	1.531 234
-O3	1.543 294
-Os	1.543 973

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void sieve_eratosthenes(unsigned int n)
{
        int *sieve = calloc(n, sizeof(unsigned int));
        sieve[0] = 1;
        for (int p = 2; p <= n; ++p) {
                if (sieve[p - 1] == 0) {
                        /* printf("%d\n", p); */
                        for (int i = p * p; i <= n; i += p)
                                sieve[i - 1] = 1;
                }
        }
}

```

```c
int main(void)
{
        unsigned int n = 100000000;
        double average = 0.0;
        for (int i = 0; i < 100; i++) {
                clock_t start = clock();
                sieve_eratosthenes(n);
                clock_t end = clock();
                average += (double) (end - start) / CLOCKS_PER_SEC;
        }
        printf("%f\n", average / 100.);
        return 0;
}
```

2 Python Implentation

Python의 경우, CPython보다 PyPy 구현이 훨씬 빠를 것으로 예상된다. PyPy로는 평균적으로 2.200 786초가 걸렸고, 기본적인 CPython으로는 무려 22.031 530초가 걸렸다. CPython은 code를 수정하여 1회만 측정하였다. PyPy의 경우 최적화하지 않은 C 언어보다 15% 가량, 최적화한 C 언어에 비해서는 40% 정도 느린 결과이다. CPython은 수행하는데 PyPy보다 10배 정도 오랜 시간이 걸렸다. 표 2에 결과가 정리되어 있다.

표 2: Average execution time of sieve of Eratosthenes of 100 runs. Written in Python, using PyPy and CPython implementation.

Version	Average Time [s]
CPython	22.031 530
PyPy	2.200 786

```python
import time

def sieve_eratosthenes(n):
    sieve = [1] * n
    sieve[0] = 0
    for p in xrange(2, n + 1):
        if sieve[p - 1]:
            # print p
            for i in xrange(p ** 2, n + 1, p):
                sieve[i - 1] = 0


```

```
n = 100000000
average = 0
for _ in xrange(100):
    start = time.time()
    sieve_eratosthenes(n)
    end = time.time()
    average += end - start
print average / 100.
```

3 Swift Implementation

Swift로는 1.975 862초가 걸렸다. 최적화하지 않은 C 언어와 비슷한 수준의 수행 시간이 걸렸으며, 최적화한 C 언어에 비해서도 30% 정도밖에 느리지 않다.

```
import QuartzCore

func sieveEratosthenes(_ n: Int) {
    var sieve = Array(repeating: 1, count: n)
    sieve[0] = 0
    for p in 2...n {
        if sieve[p - 1] == 1 {
            // print(p)
            for i in stride(from: p * p, through: n, by: p) {
                sieve[i - 1] = 0
            }
        }
    }
}

let n = 100000000
var average = 0.0
for _ in 1...100 {
    let start = CACurrentMediaTime()
    sieveEratosthenes(n)
    let end = CACurrentMediaTime()
    average += end - start
}
print(average / 100.0)
```