Pracownia Technik Obliczeniowych

Wydajność i współbieżność

Paweł Daniluk

Wydział Fizyki

Wiosna 2016

Pomiar wydajności programu

Mityczna zasada 20/80

Z reguły czas działania programu zależy od wydajności niewielkiego fragmentu. 20% kodu zajmuje 80% czasu wykonania.

Profilowanie (ang. *profiling*)

Pomiar czasu działania programu w rozbiciu na poszczególne funkcje (metody) lub linie.

Profilowanie w Pythonie...

... jest bardzo łatwe.

Moduł cProfile

```
python -m cProfile [-o output_file] [-s sort_order] myscript.py
```

Można również uruchamiać profilowanie określonych fragmentów z poziomu programu.

Przykład

profile_test.py

```
def slow(N=1000000):
    total = 0
    for i in range(N):
        total += i
    return total

def pythonic(N=1000000):
    total = sum(range(N))
    return total

if __name__ == '__main__':
    slow()
    pythonic()
```

Przykład

```
$ python -m cProfile ./profile_test.py
         7 function calls in 0.075 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.075    0.075 profile_test.py:3(<module>)
        1    0.035    0.035    0.053    0.053 profile_test.py:3(slow)
        1    0.006    0.006    0.022    0.022 profile_test.py:9(pythonic)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler
        2    0.027    0.013    0.027    0.013 {range}
        1    0.008    0.008    0.008    0.008 {sum}
```

Wizualizacja

Jest wiele narzędzi do wizualizowania wyniku profilowania. Bardzo przyjemny jest `snakeviz`.

Użycie

```
$ python -m cProfile -o prof_result ./profile_test.py
$ snakeviz prof_result
```

Współbieżność

Komputery mają więcej niż jeden procesor. Czy można to łatwo wykorzystać?

Rodzaje współbieżności

- podział danych
- podział pracy

Techniki

- Symmetric MultiProcessing (SMP) – równoprawne procesory wewnątrz jednej maszyny
- przekazywanie komunikatów – procesy mogą być uruchomione gdziekolwiek

OpenMP, pymp

Technika pozwalająca na łatwe zrównoleglenie wykonywania pętli. Nie wymaga znaczących zmian w programie.

```
ex_array = np.zeros((100,), dtype='uint8')
for index in range(0, 100):
    ex_array[index] = 1
    print('Yay! {} done!'.format(index))
```

```
ex_array = pymp.shared.array((100,), dtype='uint8')
with pymp.Parallel(4) as p:
    for index in p.range(0, 100):
        ex_array[index] = 1
        # The parallel print function takes care of asynchronous
        p.print('Yay! {} done!'.format(index))
```

pymp

Identyfikacja wątku

```
with pymp.Parallel(4) as p:
    p.print(p.num_threads, p.thread_num)
```

pymp

Komunikacja pomiędzy wątkami

Komunikacja pomiędzy wątkami odbywa się poprzez zmienne dzielone. Należy samodzielnie zapewnić ochronę dostępu do nich.

```
ex_array = pymp.shared.array((1,), dtype='uint8')
with pymp.Parallel(4) as p:
    for index in p.range(0, 100):
        with p.lock:
            ex_array[0] += 1
```

pymp

Zagnieżdżone pętle

```
with pymp.Parallel(2) as p1:
    with pymp.Parallel(2) as p2:
        p.print(p1.thread_num, p2.thread_num)
```

Równoległe wykonywanie różnych kodów

```
with pymp.Parallel(4) as p:
    for sec_idx in p.xrange(4):
        if sec_idx == 0:
            p.print('Section 0')
        elif sec_idx == 1:
            p.print('Section 1')
        ...
```

Zadanie 1

Zbadaj wydajność programu liczącego kolejne liczby Fibonacciego.

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def fib_seq(n):
    seq = [ ]
    if n > 0:
        seq.extend(fib_seq(n-1))
    seq.append(fib(n))
    return seq

fib_seq(20)
```

Zadanie 2

Jak zmienia się wydajność programu z zadania 1 po dodaniu do funkcji `fib` dekoratora `memoize`.

```
class memoize:
    def __init__(self, function):
        self.function = function
        self.memoized = {}

    def __call__(self, *args):
        try:
            return self.memoized[args]
        except KeyError:
            self.memoized[args] = self.function(*args)
            return self.memoized[args]
```

Zadanie 3

Obliczyć metodą niewydajną (z zadania 1) 100 liczb Fibonacciego na zadanej liczbie procesorów. Zmierzyć czas wykonania programu w zależności od liczby procesorów.