Práctica #4

Concurrent Futures en Python

Rubén Escalante Chan (A01370880), Guillermo Pérez Trueba (A01377162)

10 de marzo, 2019.

Tabla de contenido

Este reporte fue elaborado para el curso de *Programación multinúcleo* del Tecnológico de Monterrey, Campus Estado de México.

1. Introducción

La práctica consta de estimar el logaritmos base *e* de 2 usando Python. Ésto se debe lograr tanto de manera secuencial como de manera paralela.

Para calcular un valor aproximado de loge(2) se utiliza la siguiente serie:

```
loge(2) ≈ Σ(1, n) = (−1)^(i+1)/i ≈ 1−1/2+1/3−1/4+1/5−1/6+⋯+(−1)^(n+1/)n ≈ 0.693147181
```

Hardware y software utilizado

Los programas se probaron en una computadora de escritorio con las siguientes características:

- Procesador Intel Core i7-6700 de 2.60GHz con cuatro núcleos y ocho *hyperthreads*.
- 16 GiB de memoria RAM.
- Sistema operativo Ubuntu 16.04, kernel de Linux 3.13.0-107 de 64 bits.
- Python 3.6.5.

2. Solución

La solución consiste en computar la sumatoria de -1) ** (i + 1 / i dónde *i* va desde 1 hasta *n*. El valor de *n* es 5_000_000.

loge2.py

```
#------------------------------------------------------------
# Práctica #4: Concurrent Futures en Python
# Fecha: 24-Feb-2019
# Autores:
#          A01370880 Rubén Escalante Chan
#          A01377162 Guillermo Pérez Trueba
#------------------------------------------------------------

from time import time
from concurrent.futures import ProcessPoolExecutor
from functools import reduce
from operator import add

N = 5_000_000
NUM_PROCS = 8
```

```
def measure_time(fun, *args):
  start = time()
  result = fun(*args)
  end = time()
  return (result, end - start)

def loge2_sequential(n):
  r = 0
  for i in range(1, n + 1):
    r += ((-1) ** (i + 1)) / i
  return r

def loge2_range(range_tuple):
  start, end = range_tuple
  r = 0
  for i in range(start, end + 1):
    r += ((-1) ** (i + 1)) / i
  return r

def make_ranges(total, chunks):
  assert total % chunks == 0, f'{total} is not exactly divisible by {chunks}.'
  size = total // chunks
  return [(i * size + 1, (i + 1) * size) for i in range(chunks)]

def loge2_parallel():
  with ProcessPoolExecutor() as pool:
    results = list(pool.map(loge2_range, make_ranges(N, NUM_PROCS)))
  return reduce(add, results)

def main():
  rs, ts = measure_time(loge2_sequential, N)
  print(f'T1={ts:.4f}, Result={rs}')

  rp, tp = measure_time(loge2_parallel)
  print(f'T{NUM_PROCS}={tp:.4f}, Result={rs}')

  print(f'S{NUM_PROCS}={ts/tp:.4f}')

main()
```

El programa produce esta salida:

```
T1=2.2647, Result=0.6931470805600676
T8=1.3367, Result=0.6931470805600217
S8=1.6942
```

3. Resultado

A continuación se muestran los tiempos de ejecución de varias corridas de los dos programas:

Tabla 1. Tiempos de ejecución del ordenamiento secuencial

# de corrida	Tiempo T_1 (segundos)
1	2.2262
2	2.2175
3	4.4203
4	4.3790
5	4.3783
Media aritmética	3.5242

Tabla 2. Tiempos de ejecución del ordenamiento paralelo

# de corrida	Tiempo T_8 (segundos)
1	T8=1.1130
2	T8=1.4578
3	T8=1.6161
4	T8=1.7201
5	T8=1.8648
Media aritmética	T8=1.5544

A partir de las medias aritméticas calculadas, el *speedup* obtenido en un CPU es:

$S_8 = T_1 / T_8 = 3.5242 / 1.5544 = 2.2789$

El *speedup* obtenido es bastante bueno, siendo mayor a 2. Se podría concluir que el uso de *concurrent.futures* en python nos ayudará a crear código en paralelo eficiente, sin embargo, esto no resultaría del todo correcto, ya que en esta misma práctica, y un $N = 200$ se obtubieron los siguientes resultados:

T_1=0.0001, Result=0.6906534304818243
T_8=0.0479, Result=0.6906534304818243
$S_8 = T_1 / T_8 = 0.0001 / 0.0479 = 0.0013$

Con estos resultados, se puede concluir que la librería *concurrent.futures* si nos ayuda a crear algoritmos paralelos eficientes siempre y cuando se tenga un N bastante grande.

4. Agradecimientos

Se agradece al profesor Ariel Ortiz por sus enseñanzas y ayuda para solucionar este problema.

5. Referencias

- Ariel Ortiz. (2019). Práctica #4: Concurrent Futures en Python Sitio web: http://34.212.143.74/apps/s201911/tc3039/practica_python_concurrent_futures/