Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №8

Python.

Виконав:

студент групи ІП-32

Ковальчук О. М.

Перевірив:

Корочкін О. В.

Київ - 2015

Лабораторна робота №8. Python

Мета роботи: вивчення засобів мови Python для роботи з потоками

Мова програмування: Python3

Завдання: Розробити програму, що містить паралельні потоки, кожен з яких реалізує функції F1, F2, F3 з лабораторної роботи №1. Вимоги до потоків такі ж, як в лабораторній роботі № 2.

Функції:
F1: C = A - B * (MA * MD)
F2: o = Min(MK * MM)
F3: T = (MS * MZ) * (W + X)

Лістинг програми

src/parallelprog/data.py

```
"""
Parallel programming
Lab 8

Functions:
F1: C := A - B * (MA * MD)
F2: o := Min(MK * MM)
F3: T := (MS * MZ) * (W + X)

@since 2015-12-09
@author Olexandr Kovalchuk
IP-32
"""
from operator import add, mul
from functools import reduce


def make_vector(sz, filler=1):
    return [filler for _ in range(sz)]


def make_sq_matrix(sz, filler=1):
    return make_matrix(sz, sz, filler)


def make_matrix(r, c, filler=1):
    return [[filler for _ in range(c)] for _ in range(r)]


def transpose(m):
    return list(zip(*m))


def vm_mult(v, m):
    return [reduce(add, map(mul, v, c)) for c in transpose(m)]


def mv_mult(m, v):
    return [reduce(add, map(mul, r, v), 0) for r in m]


def mm_mult(ma, mb):
    return [
        [sum(ea*eb for ea, eb in zip(a, b)) for b in transpose(mb)]
        for a in ma
    ]
```

```
47
48
49  def v_add(a, b):
50      return [ea + eb for ea, eb in zip(a, b)]
51
52
53  def v_subs(a, b):
54      return [ea - eb for ea, eb in zip(a, b)]
55
56
57  def func1(a, b, ma, md):
58      return v_subs(a, vm_mult(b, mm_mult(ma, md)))
59
60
61  def func2(mk, mm):
62      return min(map(min, mm_mult(mk, mm)))
63
64
65  def func3(ms, mz, w, x):
66      return mv_mult(mm_mult(ms, mz), v_add(w, x))
```

src/parallelprog/lab08.py

```
1   """
2   Parallel programming
3   Lab 8
4
5   Functions:
6   F1: C := A - B * (MA * MD)
7   F2: o := Min(MK * MM)
8   F3: T := (MS * MZ) * (W + X)
9
10  @since 2015-12-09
11  @author Olexandr Kovalchuk
12  IP-32
13  """
14  from multiprocessing import Process
15  import logging
16  import os
17
18  from data import func1, func2, func3, make_sq_matrix, make_vector
19
20
21  def verbose(f):
22      def _wrapper(*args, **kwargs):
23          det = kwargs.get('det', args[0] if args else 0)
24          pid = os.getpid()
25          logging.info('Task %s in process %s started', det, pid)
26          result = f(*args, **kwargs)
27          logging.info('Task %s in process %s finished', det, pid)
28          return result
29
30      return _wrapper
31
32
33  @verbose
34  def task(det, size=4):
35      assert 1 <= det <= 3
36      assert size > 0
37
38      if det == 1:
39          ma = make_sq_matrix(size)
40          md = make_sq_matrix(size)
41          a = make_vector(size)
42          b = make_vector(size)
43          result = func1(a, b, ma, md)
44      elif det == 2:
45          mk = make_sq_matrix(size)
```

```
            mm = make_sq_matrix(size)
            result = func2(mk, mm)
        elif det == 3:
            w = make_vector(size)
            x = make_vector(size)
            ms = make_sq_matrix(size)
            mz = make_sq_matrix(size)
            result = func3(ms, mz, w, x)
        if (size < 8):
            print('task %s: %s' % (det, result))


def runapp(sz):
    assert sz > 0
    ps = [Process(target=task, args=(d, sz)) for d in range(1, 4)]
    for p in ps:
        p.start()
    for p in ps:
        p.join()


if __name__ == '__main__':
    logging.getLogger().setLevel(logging.NOTSET)
    sz = 1000
    runapp(sz)
```