Звіт до лабораторної роботи

з дисципліни "Хмарні обчислення"

на тему:

"Факторизація чисел алгоритмом Ро-Поларда з використанням технологій PARCS"

Виконав студент групи ТТП-41

Грабар Олександр Олександрович

**Постановка задачі**

Реалізувати виконання алгоритму факторизації чисел алгоритмом Ро-Поларда з використання системи PARCS для мови програмування Python. Програма має приймати число і повертати добуток двох чисел з найбільшим простим множником. Проаналізувати прискорення, забезпечене використанням worker-ів та швидкість виконання програми в залежності від кількості.

**Алгоритм:**

Нехай {\displaystyle N} складене ціле додатне число, яке потрібно розкласти на множники. Алгоритм виглядає таким чином:

1. Вибираємо число та будуємо послідовність{ }, n = 0, 1, 2, …, визначаючи кожне наступне як
2. Одночасно на кожному і-ому кроці обчислюємо d = GCD (N, ) для будь-яких, i, j таких, що j<i, наприклад i=2j
3. Якщо виявили, що d>1, то обчислення закінчується, і знайдене на попередньому кроці число d є дільником N. Якщо N/d не є простим числом, то процедуру пошуку дільників можна продовжити, узявши як N число N` = N/d

Функція F(x) має бути не надто складною для обчислення, але в той же час не має бути лінійним многочленом, а також не повинна породжувати взаємно однозначне відображення.

Зазвичай за F(x) беруть функцію F(x) = (mod N) або F(x) = (mod N). Однак не слід застосовувати

**Реалізація та код**

Нехай у нас є n ≥ 1 воркерів. Розбиваємо вхідний масив на n частин, які порівну розподіляємо між воркерами.

Тепер, реалізуємо обробку даних(факторизація чисел методом Ро-Поларда) для кожного воркера після того, як він приймає масив даних:

def mymap(N, step):

def gcd(x, y):

while (y):

x, y = y, x % y

return x

x = randint(1, N - 2)

y = 1

i = 0

stage = 2

while gcd(N, abs(x - y)) == 1:

if i == stage:

y = x

stage \*= 2

x = (x \* x + step) % N

i += 1

return gcd(N, abs(x - y))

І додаємо метод, який збирає результат роботи даних в один масив, повертає його на вихід та записує в output.txt

def myreduce(mapped):

output = []

for x in mapped:

if x.value != 1:

output.append(x.value)

return output

**Отже, маємо такий вихідний код:**

from Pyro4 import expose

from random import randint

import time

class Solver:

def \_\_init\_\_(self, workers=None, input\_file\_name=None, output\_file\_name=None):

self.input\_file\_name = input\_file\_name

self.output\_file\_name = output\_file\_name

self.workers = workers

self.N = 1

def solve(self):

self.N = self.read\_input()

N = self.N

# map

mapped = []

steps = [1, 3, 5, 7]

start\_time = time.time()

for i in range(0, len(steps)):

worker\_index = i % len(self.workers)

mapped.append(self.workers[worker\_index].mymap(N, steps[i]))

reduced = self.myreduce(mapped)

finish\_time = time.time()

# output

self.write\_output(reduced)

@staticmethod

@expose

def mymap(N, step):

def gcd(x, y):

while (y):

x, y = y, x % y

return x

x = randint(1, N - 2)

y = 1

i = 0

stage = 2

while gcd(N, abs(x - y)) == 1:

if i == stage:

y = x

stage \*= 2

x = (x \* x + step) % N

i += 1

return gcd(N, abs(x - y))

@staticmethod

@expose

def myreduce(mapped):

output = []

for x in mapped:

if x.value != 1:

output.append(x.value)

return output

def read\_input(self):

f = open(self.input\_file\_name, 'r')

line = f.readline()

f.close()

return int(line)

def write\_output(self, output):

f = open(self.output\_file\_name, 'w')

if len(output) == 0:

f.write("Could not find the divider")

else:

f.write(str(self.N) + " = " + str(output[0]) + " \* " + str(self.N / output[0]))

f.write('\n')

f.close()

**Результати спостереження:**

1) Залежність часу виконання від найбільшого простого множника числа.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10000000019 | 2550183799 | 193707721 |
| 1 worker | 0:0:10 | 0:0:6 | 0:0:1 |
| 3 workers | 0:0:6 | 0:0:2 | 0:0:1 |
| 7 workers | 0:0:5 | 0:0:2 | 0:0:1 |

**Висновок:**

При виконанні лабораторної роботи було досліджено систему PARCS. Було зрівняно час виконання програми, в залежності від кількості воркерів та величини числа. З'ясовано, що зі збільшенням найбільшого простого множника, час виконання зростає.

Збільшення кількості воркерів зменшує час виконання програми, але тут немає шаблонної залежності, оскільки на це впливає як і алгоритм, так і вхідні дані та їх величина. В моєму випадку, три та сім воркерів відпрацювали майже за однаковий час. Проте, якщо порівнювати з одним воркером, то різниця може бути більш суттєва.