

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Лабораторна робота №2

з дисципліни: «Технології паралельного програмування в умовах великих даних»
з теми: «Паралельні обчислення в моделі з розподіленою пам'яттю»

Перевірив:
доцент
Жереб К.А.

Виконав:
студент групи ІТ-01мн
Корзун І.М.

Завдання

- Обрати задачу та реалізувати для неї послідовну, в одному процесі, реалізацію та паралельну версію з розподіленою пам'яттю.
- Забезпечити можливість змінювати кількість процесів, що використовуються для обчислень.
- Порівняти швидкодію послідовної та паралельної реалізації

Хід роботи

В якості задачі обрано пошук рядків у текстах. В умовах послідовного виконання час обчислення добутку прямо пропорційний довжині вхідного тексту і є несуттєвим, коли довжина мала. Хоч складність і лінійна, та зі збільшенням довжини даних час очікування стає неприпустимим.

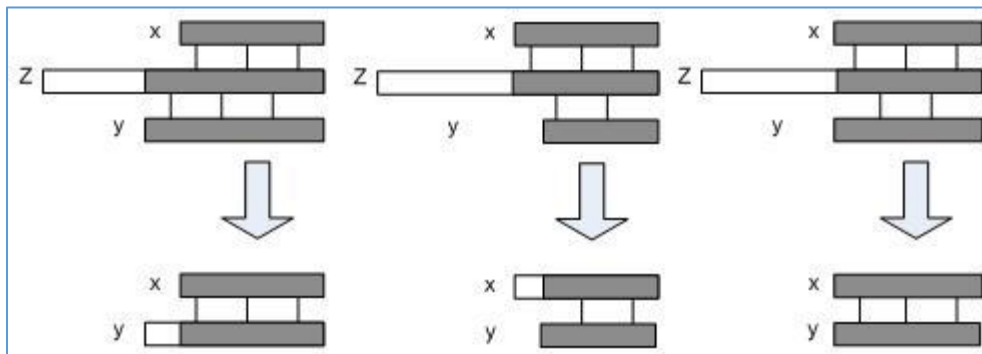


Рисунок 1. Пошук місця, де один або декілька текстових рядків входять у довший рядок або текст.

При виконанні поставленого завдання було вирішено наступні задачі:

- пошук, налаштування та вивчення засобів міжпроцесної взаємодії
 - встановлення Microsoft MPI SDK та бібліотеки mpi4py – для паралельної реалізації обраної задачі
 - встановлення бібліотеки mpi_master_slave для зручної організації міжпроцесної взаємодії при розподіленні задачі між процесами
- створено модель розподіленої системи для вирішенні даної задачі
- підготовлено два варіанти реалізації множення:
 - послідовне, в одному потоці (процесі)

- паралельне, у декількох потоках – за допомогою засобів MPI
- проаналізовано час пошуку рядків у тексті у запропонованій реалізації

Налаштування робочого середовища

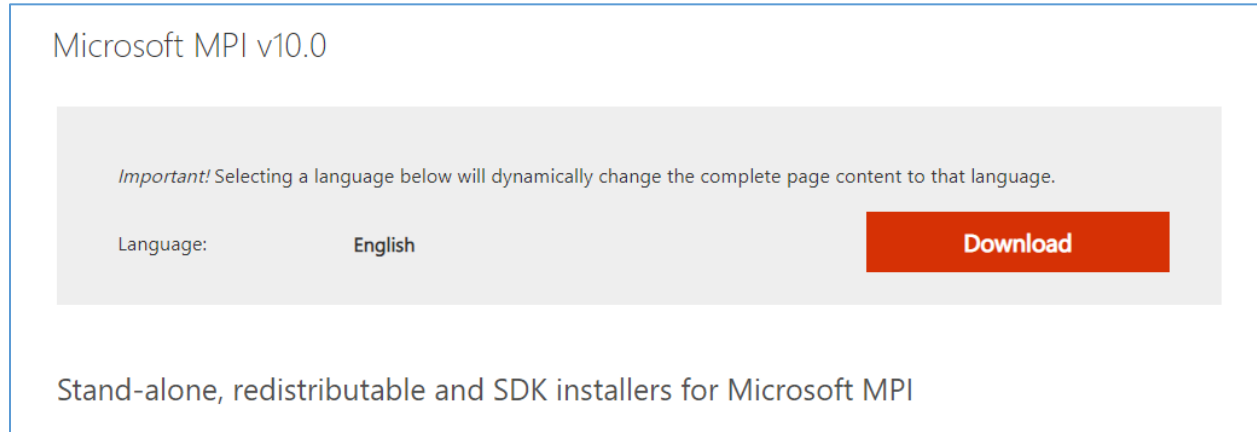


Рисунок 2. Завантаження інсталятора SDK з [офіційного сайту Microsoft](#) (для Windows 10)

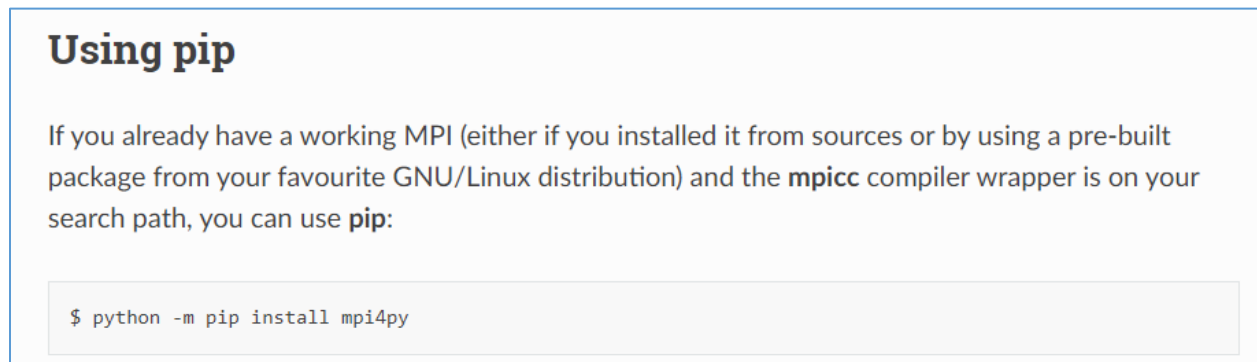


Рисунок 3. Встановлення бібліотеки mpi4py



Рисунок 4. . Встановлення бібліотеки mpi_master_slave

Лістинг 1. Реалізація програми засобами міжпроцесної взаємодії, яка здатна розподіляти роботу між іншими процесами

```
class MyApp(object):
    """
    This application has lots of work to do so it delegates tasks
    to slaves until everything is done
    """

    def __init__(self, slaves):
        # when creating the Master we tell it what slaves it can handle
        self.master = Master(slaves)
        # WorkQueue is a convenient class that run slaves on a tasks queue
        self.work_queue = WorkQueue(self.master)

    def terminate_slaves(self):
        """
        Call this to make all slaves exit their run loop
        """
        self.master.terminate_slaves()

    def run(self):
        """
        keep starting slaves as long as there is work to do
        """
        path = "text.txt"
        pattern = "love"
        results = []

        workers = self.master.num_slaves()
        text = ""
        with open(path, "r") as file:
            text = file.read()
        #
        # split text into equal parts
        #
        ratio = int(len(text) / workers)
        for i in range(workers):
            beg, end = max(0, ratio * i), min(len(text) - 1, ratio * (i + 1))
            text_part = text[beg:end]
            self.work_queue.add_work(data=(find_pattern_in_text, text_part, pattern))
        #
        # give more work to do to each idle slave (if any)
        #
        while not self.work_queue.done():
            self.work_queue.do_work()
            #
            # reclaim returned data from completed slaves
            #
            for work in self.work_queue.get_completed_work():
                done, report = work
                if done:
                    results.append(report)
        #
        # finally, show workers results
        #
        for result in results:
            print(
                f'Master: slave-{result[0]} found {result[1] if len(result[1]) else "nothing"}'
            )
```

Лістинг 2. Реалізація інтерфейсу залежних процесів, що шукатимуть рядки в даному тексті

```
class MySlave(Slave):
    """
    A slave process extends Slave class, overrides the 'do_work' method
    and calls 'Slave.run'. The Master will do the rest
    """

    def __init__(self):
        super(MySlave, self).__init__()
        self.id = MPI.COMM_WORLD.Get_rank()

    def do_work(self, data):
        find, text, pattern = data
        indices = find(text, pattern)

        return (True, (self.id, indices))
```

Лістинг 3. Реалізація вимірювання часу відпрацювання програми

```
import time

def timeit(proc):
    """Returns execution time of the nullary function in seconds"""

    beg = time.time()
    proc()
    end = time.time()
    return end - beg
```

Лістинг 4. Реалізація алгоритму пошуку рядків у тексті

```
def find_pattern_in_text(text: str, pattern: str, count: int = -1):
    """
    Finds 'count' of places where the pattern
    is located in the text
    """

    locations = list()

    idx = 0
    pad = 1

    while count == -1 or len(locations) < count:
        #
        # find another location in the text
        #
        i = __find_str(text[idx:], pattern)
        if i == -1:
            break

        #
        # note the location and the pattern with some neighbour characters around
        #

        beg = idx + i
        end = beg + len(pattern)
        locations.append((text[max(0, beg - pad) : min(len(text) - 1, end + pad)], beg))

        #

        idx = end

    return locations
```

Лістинг 5. Код основної програми

```
from mpi4py import MPI
from app import MyApp, MySlave

def main():
    #
    # gets the process Id
    #
    rank = MPI.COMM_WORLD.Get_rank()

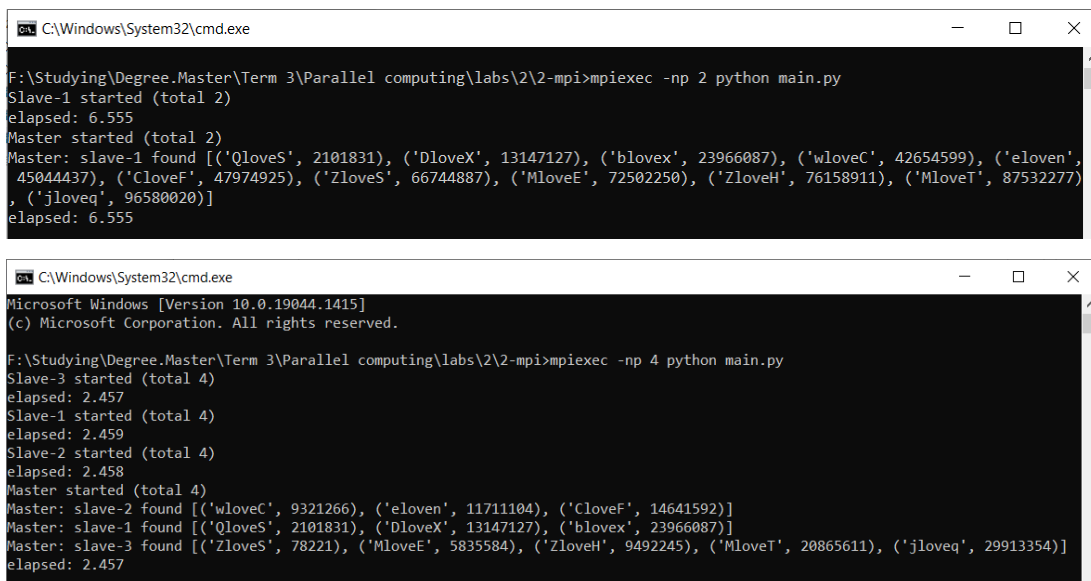
    #
    # gets the total count of managed processes
    #
    size = MPI.COMM_WORLD.Get_size()

    #
    # assigns a role to the process based on its Id
    #
    role = "Master" if rank == 0 else f"Slave-{rank}"
    print(f"{role} started (total {size})")

    #
    # 0. Master routine
    #
    if rank == 0:
        app = MyApp(slaves=range(1, size))
        app.run()
        app.terminate_slaves()

    #
    # 1-. Slaves routine
    #
    else:
        MySlave().run()

if __name__ == "__main__":
    print("elapsed: %.3f" % timeit(main))
```



```
C:\Windows\System32\cmd.exe
F:\Studying\Degree.Master\Term 3\Parallel computing\labs\2\2-mpi>mpiexec -np 2 python main.py
Slave-1 started (total 2)
elapsed: 6.555
Master started (total 2)
Master: slave-1 found [('QloveS', 2101831), ('DloveX', 13147127), ('blovex', 23966087), ('wloveC', 42654599), ('eloven', 45044437), ('CloveF', 47974925), ('ZloveS', 66744887), ('MloveE', 72502250), ('ZloveH', 76158911), ('MloveT', 87532277), ('jloveq', 96580020)]
elapsed: 6.555

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.1415]
(c) Microsoft Corporation. All rights reserved.
F:\Studying\Degree.Master\Term 3\Parallel computing\labs\2\2-mpi>mpiexec -np 4 python main.py
Slave-3 started (total 4)
elapsed: 2.457
Slave-1 started (total 4)
elapsed: 2.459
Slave-2 started (total 4)
elapsed: 2.458
Master started (total 4)
Master: slave-2 found [('wloveC', 9321266), ('eloven', 11711104), ('CloveF', 14641592)]
Master: slave-1 found [('QloveS', 2101831), ('DloveX', 13147127), ('blovex', 23966087)]
Master: slave-3 found [('ZloveS', 78221), ('MloveE', 5835584), ('ZloveH', 9492245), ('MloveT', 20865611), ('jloveq', 29913354)]
elapsed: 2.457
```

Рисунок 5. Результати порівняння послідовної та паралельної реалізації задачі з розподіленою пам'яттю.

Висновок

У результаті виконання лабораторної роботи реалізовано рішення для виконання пошуку рядків у тексті (обрана задача) засобами міжпроцесної взаємодії. Зі створеною програмою проведено аналіз тривалості виконання. Час роботи послідовної версії (з одним підрядним процесом) достатній для одноразового застосування на малих даних – при залученні паралельної версії в даних умовах програма деградує через необхідність синхронізації потоків. Але, коли розміри тексту достатньо великі (100 мільйонів символів), паралельна версія показує помітно кращі результати і їй слід надавати перевагу при вирішенні даного типу задач.