Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет комп’ютерних інформаційних технологій

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни «Паралельні та розподілені комп’ютерні системи»

спеціальність: 123 - Комп’ютерна інженерія
галузь знань: 12 – Інформаційні технології
освітній рівень «Бакалавр»

2022

Піцун О.Й. Методичні вказівки до проведення лабораторних робіт з дисципліни «Паралельні та розподілені комп’ютерні системи» / О.Й. Піцун / Під ред. О.М. Березького, Тернопіль: ЗУНУ, 2022. 42 с.

Укладач: О.Й. Піцун, к.т.н., ст. викладач кафедри КІ ЗУНУ.

Відповідальний за випуск: Березький О.М., д.т.н., професор кафедри КІ ЗУНУ.

Рецензенти:

Трембач Р.Б. – к.т.н., доцент кафедри автоматизації технологічних процесів і виробництв Тернопільського національного технічного університету ім. І. Пулюя.

Івасьєв С.В. - к.т.н., доцент кафедри кібербезпеки ЗУНУ.

Методичні вказівки розглянуто та рекомендовано до друку на засіданні кафедри комп’ютерної інженерії протокол № 3 від 24.10.2022 р.

Методичні рекомендації затверджено на засіданні групи забезпечення спеціальності за напрямом підготовки "Комп’ютерна інженерія" протокол № 2 від 24.10.2022 р.

ЗМІСТ

Лабораторна робота №1 5

Лабораторна робота №2 12

Лабораторна робота №3 20

Лабораторна робота №4 29

Лабораторна робота №5 35

РЕКОМЕНДОВАНІ ДЖЕРЕЛА ІНФОРМАЦІЇ 40

ВСТУП

Програма та тематичний план дисципліни орієнтовані на отримання студентами навиків та знань щодо використання технологій паралельних та розподілених обчислень під час розробки програмного забезпечення.

Курс «Паралельні та розподілені комп’ютерні системи» призначений для розширення компетентностей випускників спеціальності 123 - Комп'ютерна інженерія в галузі прикладного застосування комп’ютерних систем штучного інтелекту в наукових дослідженнях та на виробництві. Введення курсу в навчальний план дозволяє надати студентам додаткові знання та практичні навички, які вони зможуть застосовувати як при подальшому навчанні, так і в майбутній професійній діяльності.

Дані методичні рекомендації призначені для студентів денної і заочної форм навчання освітнього рівня «Бакалавр».

Лабораторна робота №1

Тема: Організація багатозадачності за допомогою потоків на основі технології JAVA.

Мета: Навчитися розробляти програмне забезпечення з використанням багатопоточності на основі мови програмування JAVA.

Питання для обговорення:

1. Підходи для забезпечення багатопотоковості в Java
2. Інтерфейс Runnable
3. Клас Thread та його методи

Теоретичні відомості

Найбільш очевидна область застосування багатопоточності - це програмування інтерфейсів. Потоки незамінні тоді, коли необхідно, щоб графічний інтерфейс продовжував відгукуватися на дії користувача під час виконання певної обробки інформації. Наприклад, потік, який відповідає за інтерфейс, може чекати завершення іншого потоку, що завантажує файл з інтернету, і в цей час виводити деяку анімацію або оновлює прогрес-бар.

Процес - це сукупність коду і даних, які поділяють спільний віртуальний адресний простір. Найчастіше одна програма складається з одного процесу, але бувають і виключення (наприклад, браузер Chrome створює окремий процес для кожної вкладки, що дає йому деякі переваги, на кшталт незалежності вкладок один від одного). Для кожного процесу ОС створює так зване «віртуальний адресний простір», до якого процес має прямий доступ. Це простір належить процесу, містить тільки його дані і знаходиться в повному його розпорядженні. Операційна система ж відповідає за те, як віртуальний простір процесу проектується на фізичну пам'ять.

Потік - це одна одиниця виконання коду. Кожен потік послідовно виконує інструкції процесу, якому він належить, паралельно з іншими потоками цього процесу. На одне ядро процесора, в кожен момент часу, доводиться одна одиниця виконання. Тобто одноядерний процесор може обробляти команди тільки послідовно, по одній за раз. Однак запуск декількох паралельних потоків можливий і в системах з одноядерними процесорами. В цьому випадку система буде періодично перемикатися між потоками, по черзі даючи виконуватися то одному, то іншому потоку. Така схема називається *псевдо-паралелізмом*.

Графічно принцип псевдо-паралелізму наведено на рисунку 1.

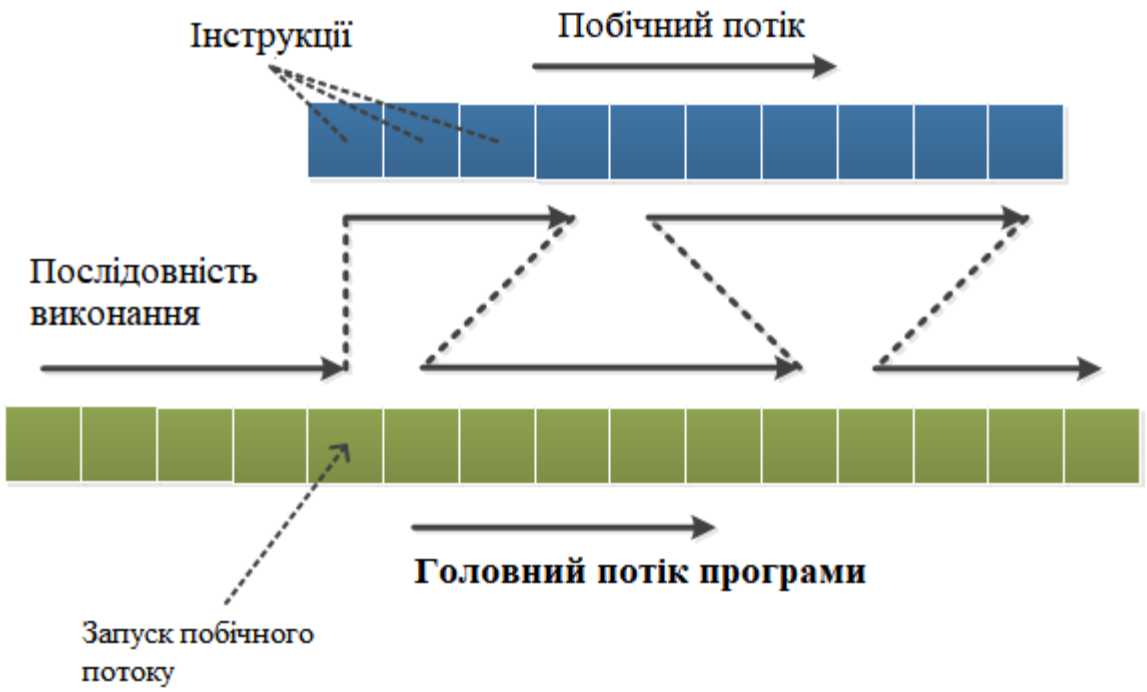

Рисунок 1.1 – Графічне представлення принципу псевдо-паралелізму

У Java потік представляється у вигляді об'єкта-нащадка класу ***Thread***. Цей клас інкапсулює стандартні механізми роботи з потоком. Створити новий потік можна такими способами:

1) З допомогою інтерфейсу ***Runnable***.

   Метод ***run()*** виконуватиметься у новому потоці.

   Оскільки інтерфейс Runnable функціональний, ми можемо використовувати лямбда-вирази, які з'явилися в Java 8. Виведемо ім'я поточного

потоку на консоль, і запустимо спочатку в головному потоці, а потім - в окремому.

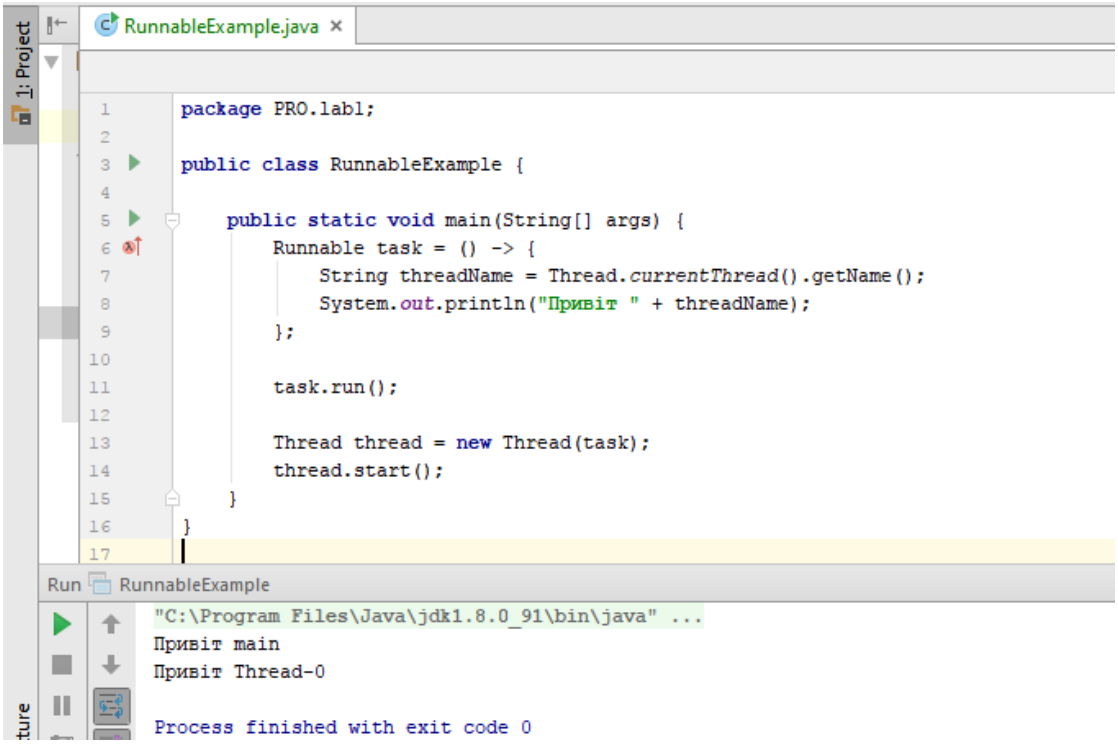

Рисунок 1.2 – Створення нового потоку засобами ***Runnable***

2) Створити нащадка класу **Thread** і перевизначити його метод ***run ()***:

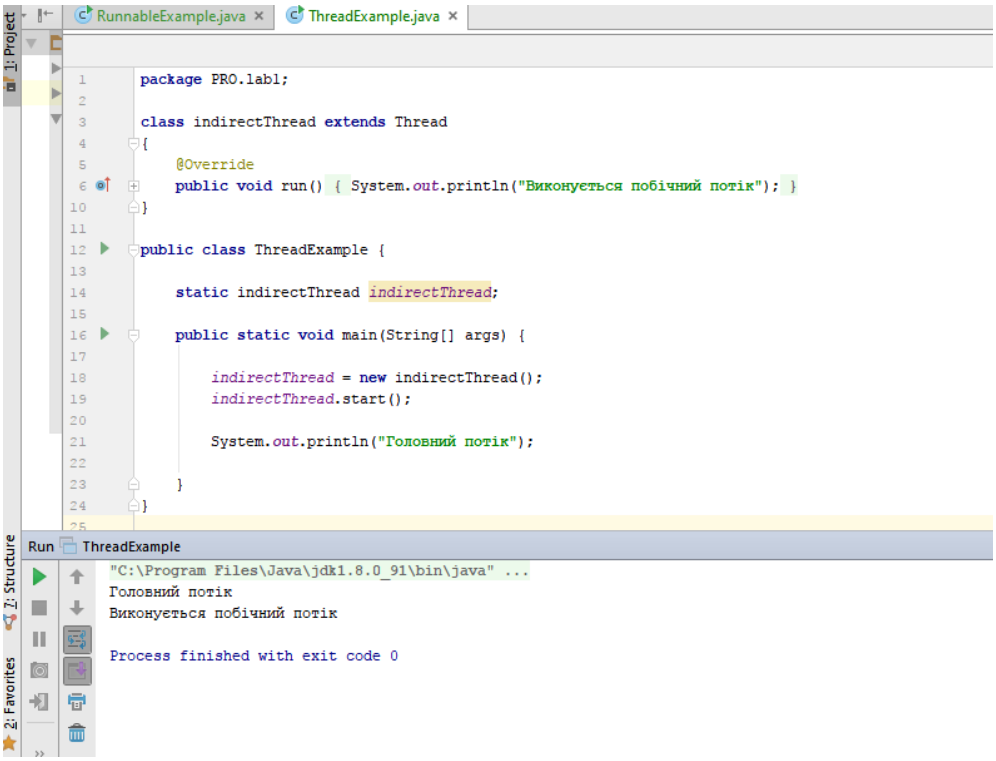

Рисунок 1.3 – Розпаралелення з допомогою класу **Thread**

Функції для роботи з потоками

Thread.stop() – завершення потоку. Потік може зупинитися або тоді, коли він закінчить виконання методу *run ()*, (main () - для головного потоку) або за сигналом з іншого потоку.

Thread.sleep ()– статичний метод класу **Thread**, який призупиняє виконання потоку, в якому він був викликаний. Під час виконання методу *sleep ()* система перестає виділяти потоку процесорний час, розподіляючи його між іншими потоками. Метод *sleep ()* може виконуватися або заданий кількість часу (мілісекунди або наносекунди) або до тих пір поки він не буде зупинений перериванням.

Thread.yield () – метод, що змушує процесор переключитися на обробку інших потоків системи. Метод може бути корисним, наприклад, коли потік очікує настання якої-небудь події і необхідно щоб перевірка його настання відбувалася якомога частіше. В цьому випадку можна помістити перевірку події і метод *Thread.yield ()* в цикл.

В Java передбачений механізм, що дозволяє одному потоку чекати завершення виконання іншого. Для цього використовується метод ***join ()***. Наприклад, щоб головний потік почекав завершення побічного потоку *myThready*, необхідно виконати інструкцію *myThready.join ()* в головному потоці. Як тільки потік *myThready* завершиться, метод ***join ()*** поверне управління, і головний потік зможе продовжити виконання.

boolean ***isAlive()*** - повертає true якщо *myThready ()* виконується і false якщо потік ще не був запущений або був завершений.

setName (String threadName) - задає ім'я потоку.

String ***getName ()*** - повертає ім'я потоку.

Лістинг коду програми обчислення завдання за допомогою одного та декількох потоків:

```
      public class Processor {

    public static final int STR_COUNT = 100;
    public static final int TASK_COUNT = 10000;

    public static void main(String[] args) {
        System.out.println("Старт");
        {
            BigTaskOneThread bt = new BigTaskOneThread();
            long d1 = System.currentTimeMillis();
            Long result = bt.startTask();
            long d2 = System.currentTimeMillis();
            System.out.println(result + ", Time 1:" + (d2 - d1));
        }/*
        {
            BigTaskManyThreads bt = new BigTaskManyThreads();
            long d1 = System.currentTimeMillis();
            Long result = bt.startTask();
            long d2 = System.currentTimeMillis();
            System.out.println(result + ", Time 2:" + (d2 - d1));
        }*/
        System.out.println("Фініш");

    }

    public Long process() {
        Long summa = 0L;

        SecureRandom random = new SecureRandom();
        for (int i = 0; i < Processor.TASK_COUNT; i++) {
            String g = new BigInteger(500, random).toString(32);
            for (char c : g.toCharArray()) {
                summa += c;
            }
        }
        return summa;
    }
}

class BigTaskOneThread {

    public Long startTask() {
        Long summa = 0L;
        for (int i = 0; i < Processor.STR_COUNT; i++) {
            Processor p = new Processor();
            summa += p.process();
        }
        return summa;
    }

}

class BigTaskManyThreads {

    public Long startTask() {
        int ap = Runtime.getRuntime().availableProcessors();
        ExecutorService es = Executors.newFixedThreadPool(ap);
```

```
        Long summa = 0L;
        try {
            List<MyCallable> threads = new ArrayList<MyCallable>();
            for (int i = 0; i < Processor.STR_COUNT; i++) {
                threads.add(new MyCallable());
            }
            List<Future<Long>> result = es.invokeAll(threads);

            for (Future<Long> f : result) {
                summa += f.get();
            }
            es.shutdown();
        } catch (InterruptedException | ExecutionException ex) {
            ex.printStackTrace(System.out);
        }
        return summa;
    }
}

class MyCallable implements Callable<Long> {

    @Override
    public Long call() throws Exception {
        Processor p = new Processor();
        return p.process();
    }
}
```

Синхронізація

У многопоточності можна визначати певний ресурс, що ексклюзивно надається одночасно тільки одному потоку. Такий ресурс називається «**монітором**». Це звичайний об'єкт, який потік повинен «захопити». Всі потоки, які хочуть отримати доступ до цього монітора (об'єкту) шикуються в чергу. Причому для цього не треба писати спеціальний код - досить просто спробувати «захопити» монітор. Слово **synchronized** говорить про те, що перш ніж потік зможе викликати певний метод у нашого об'єкта, він повинен «захопити» наш об'єкт і потім виконати потрібний метод.

Лістинг коду із синхронізованим методом `increaseCounter()`

```
public class CounterTester
{
    public static void main(String[] args) throws InterruptedException {
        Counter counter = new Counter();
        for(int i=0; i<200; i++) {
            CounterThread ct = new CounterThread(counter);
            ct.start();
        }
        Thread.sleep(1000);

        System.out.println("Counter:" + counter.getCounter());
```

```
    }
}

class Counter
{
    private long counter = 0L;

    public synchronized void increaseCounter() {
        counter++;
    }

    public long getCounter() {
        return counter;
    }
}

class CounterThread extends Thread
{
    private Counter counter;

    public CounterThread(Counter counter) {
        this.counter = counter;
    }

    @Override
    public void run() {
        for(int i=0; i<1000; i++) {
            counter.increaseCounter();
        }
    }
}
```

Завдання:

1. Освоїти теоретичний матеріал.
2. Запустити коди, наведені у методичці.
3. Провести порівняльний аналіз роботи програми з одним та багатьма потоками https://github.com/olehpitsun/testki/blob/master/src/PRO/lab1/Processor.java
4. Графічно відобразити результати роботи

Контрольні запитання

Які існують підходи до розпаралелення в JAVA ?

Наведіть основні методи класу Thread.

Основні підходи до застосування інтерфейсу Callable

Лабораторна робота №2

Тема: Розпаралелення секцій програми за допомогою технології OpenMP
Мета: Розпаралелення блоків і циклів послідовних програм засобами OpenMP, використовуючи Visual C++

Питання для обговорення:

1. Підходи для забезпечення багатопотоковості в C++
2. Директиви OpenMP
3. Розпаралелення циклів

Теоретичні відомості

Директива ***sections.*** *Я*к правило, OpenMP застосовується для розпаралелювання циклів. Однак, в OpenMP є також засоби для підтримки паралелізму на рівні функцій. Цей механізм називається секціями OpenMP (OpenMP sections). Для створення паралельного блоку секцій застосовується директива #pragma omp sections, або, аналогічно директиві #pragma omp parallel for, її скорочений варіант #pragma omp parallel sections . Секції в блоці створюються директивой #pragma omp section . Кожній секції ставиться у відповідність один потік, і всі секції виконуються паралельно.

Фрагмент програми, що демонструє розпаралелювання на рівні функцій наведено нижче.

```
void quickSort ( int L, int R)
{
int i ;
if (R > L)
{
i = partition (L, R);
```

```
#pragma omp parallel sections
{
#pragma omp section
quickSort (L, i - 1);
#pragma omp section
quickSort ( i + 1 , R);
} /* #pragma omp parallel sections */
} /* if (R > l ) */
} /* void quickSort ( int l , int r ) */
```

Створення нового проекту у Visual Studio

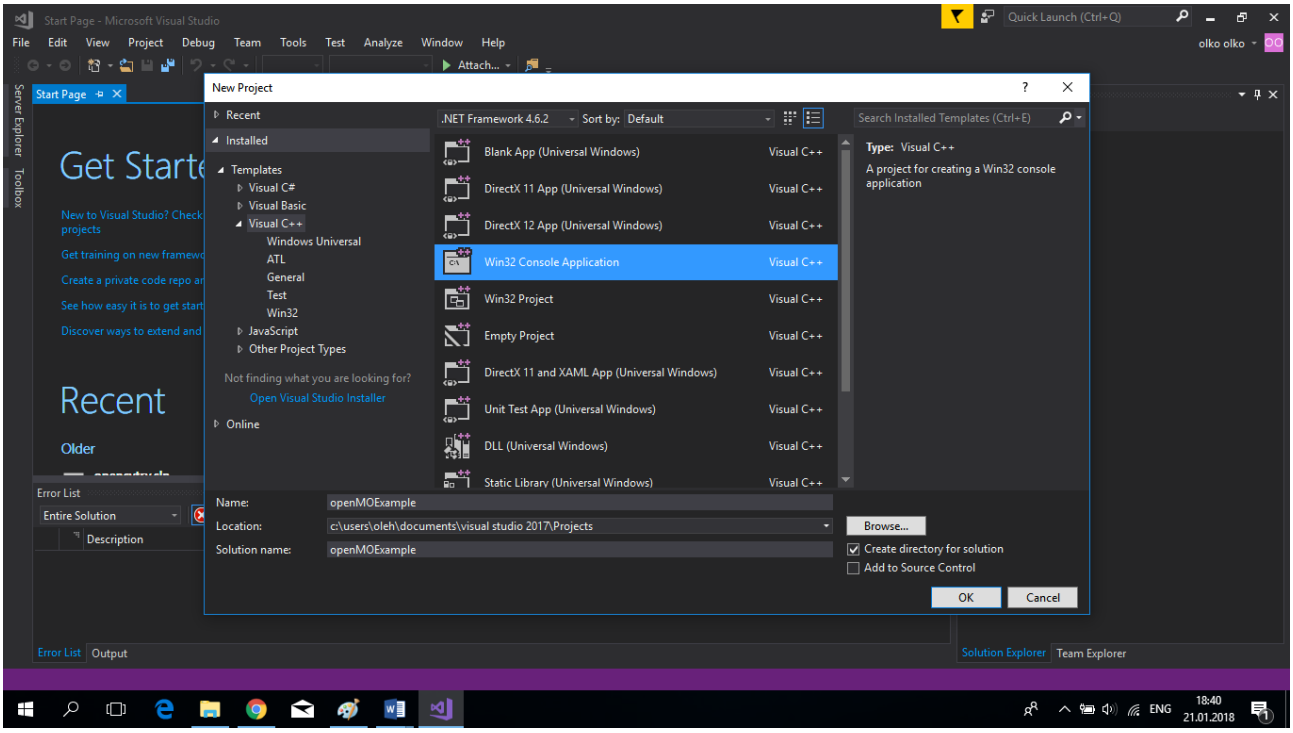

Рисунок 2.1 - Створення нового проекту у Visual Studio

Створення нового класу

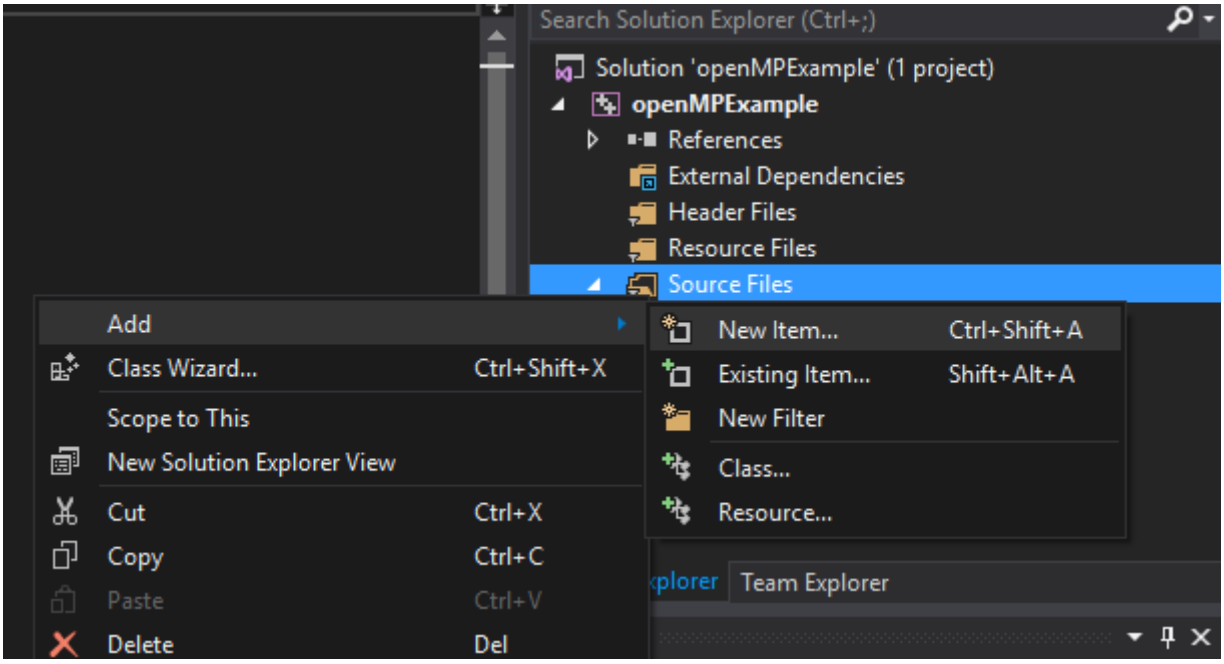

Рисунок 2.2 - Створення нового класу

Підключення бібліотеки OpenMP проекту

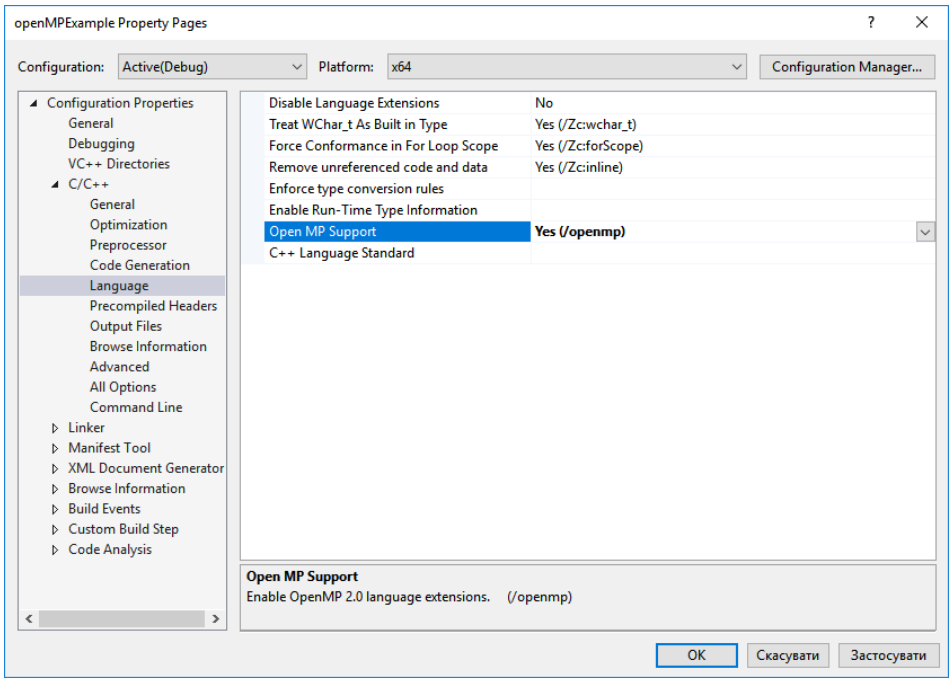

Рисунок 2.3 - Підключення бібліотеки OpenMP

Запуск програми з допомогою OpenMP

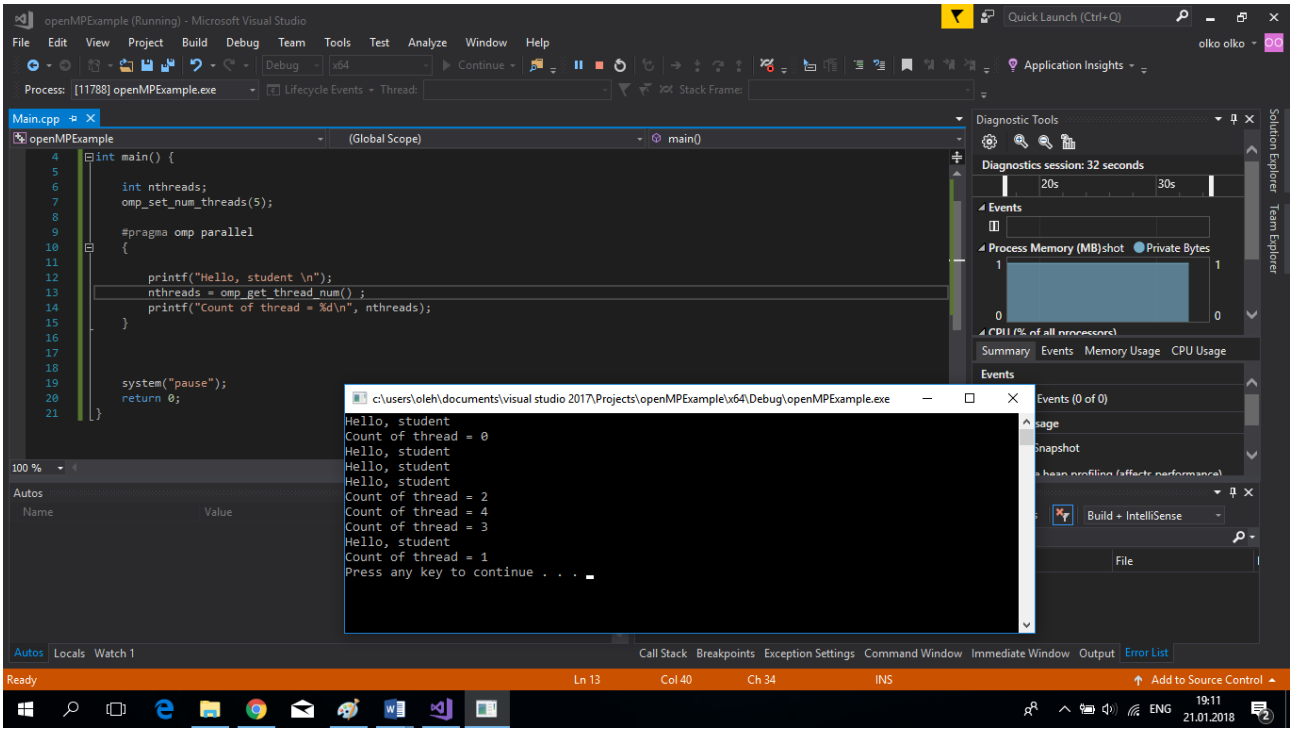

Рисунок 2.4 - Запуск програми з допомогою OpenMP

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Створити проект консольного застосування з назвою “OMPPi” і скопіювати в нього програму з додатку А. Дана програма призначена для знаходження числа Пі.
3. Відкомпілювати проект, виконати і знайти середній час виконання.
4. Розпаралелити послідовну програму за допомогою OpenMP: - Включити для даного проекту підтримку OpenMP: меню Project\OMPPi Properties, вкладка Configuration Properties\C/C++\Language, для властивості OpenMP support вказати значення Yes (/openmp). - У вікні файлу OMPPi.cpp підключити відповідну бібліотеку: #include "omp.h". - У текст програми вставити необхідні прагми OpenMP.

5. Відкомпілювати проект, виконати і знайти середній час виконання.
6. Створити проект консольного застосування з назвою “OMPSections” і

скопіювати в нього програму з додатку Б. Дана програма здійснює дві секції обчислень над масивами.

7. Виконати кроки 3-5 для проекту “OMPSections”. При розпаралеленні даної програми необхідно, щоб випадкова ініціалізація масивів відбувалася паралельно, обидві секції обчислень (Section 1, Section 2) теж виконувалися одночасно. При оцінці часу виконання послідовної і розпаралеленої версій програми доцільно відключати вивід наступних повідомлень:

```
printf("Thread %d: C[%d]= %f\n", tid, i, C[i]);
printf("Thread %d: D[%d]= %f\n", tid, i, D[i]);
```

8. Результати виконання лабораторної роботи оформити у вигляді звіту.

Зміст звіту

1. Короткі теоретичні відомості по лабораторній роботі.
2. Лістинги розпаралелених програм OMPPi та OMPSections.
3. Копії екрану з результатами виконання послідовних і паралельних версій програм OMPPi та OMPSections.
4. Діаграми або таблиці з порівнянням швидкодії послідовних і паралельних версій програм OMPPi та OMPSections.
5. Висновки.

Контрольні питання

1. Якими директивами виконується розпаралелювання циклів в OpenMP?
2. Які змінні в паралельних блоках є спільними, а які — приватними?
3. Яким чином діє параметр private?
4. Яким чином діє параметр firstprivate?
5. Яким чином діє параметр lastprivate?
6. Яким чином діє параметр reduction?

7. Які оператори можуть застосовуватись з параметром reduction?

8. Яким чином виконується розпаралелювання коду на рівні функцій?

ВАРІАНТИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

№ варіанту	OMPPi
1.	num_steps = 1000000000 кількість потоків = 2
2.	num_steps = 500000000 кількість потоків задається динамічно
3.	num_steps = 1000000000 кількість потоків = 4
4.	num_steps = 100000000
5.	num_steps = 1000000000 кількість потоків задається динамічно
6.	num_steps = 500000000 кількість потоків = 2
7.	num_steps = 1000000000 кількість потоків задається динамічно
8.	num_steps = 1000000000 кількість потоків = 2
9.	num_steps = 500000000 кількість потоків задається динамічно
10.	num_steps = 1000000000 кількість потоків = 4
11	num_steps = 100000000 кількість потоків задається динамічно
12.	num_steps = 1000000000 кількість потоків = 8
13.	num_steps = 500000000 кількість потоків задається динамічно
14.	num_steps = 1000000000 кількість потоків задається динамічно
15	num_steps = 100000000 кількість потоків задається динамічно
16.	num_steps = 1000000000 кількість потоків = 2
17.	num_steps = 500000000 кількість потоків задається динамічно
18.	num_steps = 1000000000
19	num_steps = 100000000 кількість потоків задається динамічно
20.	num_steps = 1000000000 кількість потоків задається динамічно

Додаток А
Програма для розрахунку числа Пі

```
// OMPPi.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include <stdio.h>
#include <time.h>
#include "conio.h"
int main()
{
long long num_steps = 1000000000;
double step;
clock_t start, stop;
double x, pi, sum=0.0;
int i;
step = 1./(double)num_steps;
start = clock();
for (i=0; i<num_steps; i++)
{
x = (i + .5)*step;
sum = sum + 4.0/(1.+ x*x);
}
pi = sum*step;
stop = clock();
printf("The value of PI is %15.12lf\n", pi);
printf("The time to calculate PI was %lf seconds\n", ((double)(stop - start)/1000.0));
printf("Press any key...");
_getch();
return 0;
}
```

Додаток Б
Програма з обчисленнями над масивом

```
// OMPSections.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include "time.h"
#include "conio.h"
#include "math.h"
int main ()
{
const int N = 30000;
int i, nthreads, tid;
float A[N], B[N], C[N], D[N];
clock_t start, stop;
srand(time(0));
// Random initializations for arrays
```

```
for (i=0; i<N; i++)
{
A[i] = (float) rand() / RAND_MAX;
B[i] = (float) rand() / RAND_MAX;
C[i] = (float) rand() / RAND_MAX;
}
start = clock();
tid = omp_get_thread_num();
if (tid == 0)
{
nthreads = omp_get_num_threads();
printf("Number of threads = %d\n", nthreads);
}
printf("Thread %d starting...\n", tid);
// Section 1
printf("Thread %d doing section 1\n", tid);
for (i=0; i<N; i++)
{
C[i] = cos(A[i]) + sin(B[i]) - sqrt(fabs(A[i] + B[i]));
printf("Thread %d: C[%d]= %f\n", tid, i, C[i]);
}
// Section 2
printf("Thread %d doing section 2\n", tid);
for (i=0; i<N; i++)
{
D[i] = sqrt(fabs(A[i] * B[i])) - sin(A[i]*A[i]) / cos (B[i]);
printf("Thread %d: D[%d]= %f\n", tid, i, D[i]);
}
stop = clock();
printf("Thread %d done.\n", tid);
printf("The computing time is %.2lf ms\n\n", ((double)(stop - start)));
printf("Press any key...");
_getch();
}
```

Лабораторна робота №3

Тема: Організації розподіленого зберігання і обробки наборів великих даних з використанням технології Apache Hadoop.

Мета: Навчитися працювати з технологіями для забезпечення розподіленої обробки даних.

Питання для обговорення:

1. Основні складові Apache Hadoop
2. Команди Linux для роботи з користувачами
3. Технологія MapReduce

Теоретичні відомості

Apache Hadoop — вільна програмна платформа і каркас для організації розподіленого зберігання і обробки наборів великих даних з використанням моделі програмування MapReduce, при якій завдання ділиться на багато дрібніших відособлених фрагментів, кожен з яких може бути запущений на окремому вузлі кластера, що складається з серійних комп'ютерів. Всі модулі в Hadoop спроектовані з врахуванням припущення, що злам апаратного забезпечення трапляється часто і це повинно автоматично враховуватись фреймворком.

Ядро системи Apache Hadoop складається з розподіленої файлової системи Hadoop Distributed Filesystem (HDFS), та системи обчислень на основі моделі програмування MapReduce. Hadoop розділяє файли на великі блоки і розподіляє їх між вузлами кластера. Тоді він передає запакований код на вузли для паралельної обробки даних. Цей підхід користується локальністю даних, коли вузли маніпулюють лише даними до яких мають доступ. Це дозволяє обробляти набір даних швидше і ефективніше ніж в традиційнішій суперкомп'ютерній

архітектурі яка покладається на паралельну файлову систему в якій обчислення та дані для них передаються через високошвидкісну мережу.

Основний фреймворк Apache Hadoop складається з наступних модулів:

Hadoop Common — містить бібліотеки та утиліти потрібні іншим модулям Hadoop;

Hadoop Distributed File System (HDFS) — розподілена файлова система, яка зберігає дані на звичайних машинах, надаючи дуже високу загальну пропускну здатність на кластері загалом;

Hadoop YARN — платформа що відповідає за керування обчислювальними ресурсами в кластерах і їх використання для користувацьких завдань.

Hadoop MapReduce — реалізація моделі програмування MapReduce для обробки великих об'ємів даних.

Технологію MapReduce придумали в Google для системи пошуку в Інтернет. Мета - зберігати і обробляти великі обсяги даних на звичайних комп'ютерах, об'єднаних мережею

Особливості MapReduce:

Автоматизація розпаралелювання

• Розподілена файлова система для зберігання даних на внутрішніх дисках серверів кластера

• Орієнтація на потокову обробку великих обсягів даних

• Захист від збоїв обладнання

• Автоматичний перезапуск завдань на інших вузлах

• Обробка даних там, де вони зберігаються

• Переміщення обчислень до даних

• Один алгоритм обробки даних - MapReduce

Архітектура Hadoop v2 наведено на рисунку 3.1.

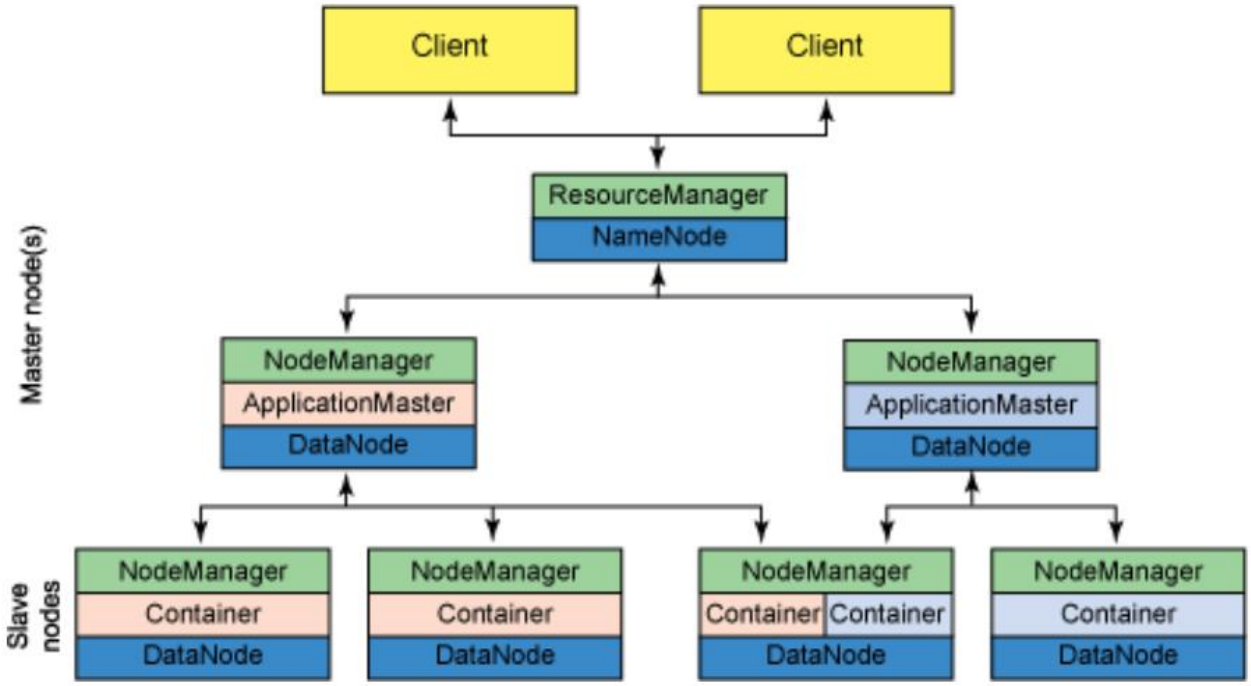

Рисунок 3.1 - Архітектура Hadoop v2

Режими роботи Hadoop

• ***Локальний***

- Один процес на одному комп'ютері
- Mapper і Reducer у вигляді потоків

• Псевдо-розподілений

- Всі процеси Hadoop працюють на одному комп'ютері
- Працюють демони Hadoop (NameNode, DataNode, ResourceManager, NodeManager і ін.)
- Mapper і Reducer працюють в окремих процесах

• Розподілений

- Процеси Hadoop працюють на декількох комп'ютерах
- Працюють демони Hadoop (NameNode, DataNode, ResourceManager, NodeManager і ін.)
- Mapper і Reducer працюють в окремих процесах на різних комп'ютерах

Налаштування Hadoop на Ubuntu

1. **Налаштування Java**

```
sudo apt-get update
sudo apt-get install default-jre
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get install oracle-java8-installer
```

2. **Створення нового користувача**

```
adduser hadoop
passwd Hadoop
```

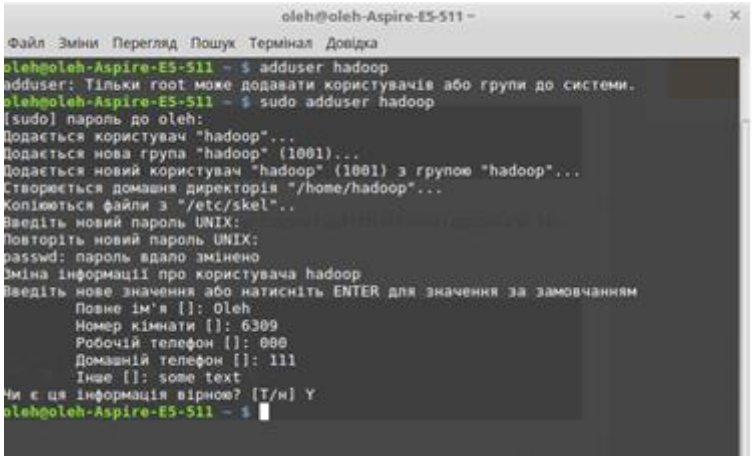

Рисунок 3.2 – Створення нового користувача

3. **Налаштування ssh – доступу**

```
su - hadoop
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
```

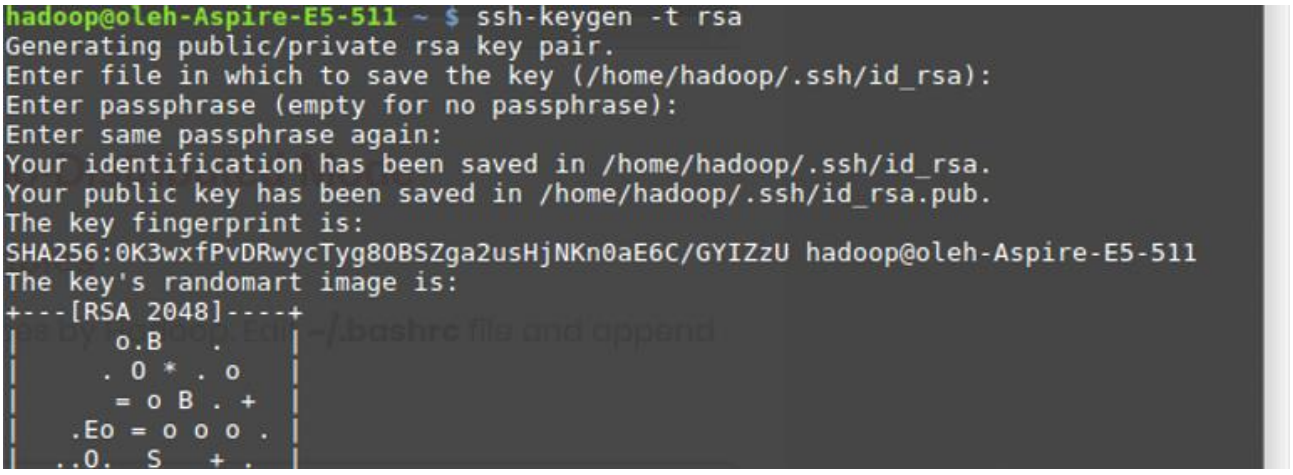

Рисунок 3.3 – Генерування ключа

4. **Завантаження Hadoop**

```
cd ~
 wget      http://www-eu.apache.org/dist/hadoop/common/hadoop-
2.6.5/hadoop-2.9.0.tar.gz
 tar xzf hadoop-2.9.0.tar.gz
 mv hadoop-2.9.0 hadoop
```

5. **Налаштування змінних середовища Hadoop**

Відредагуйте ~/.bashrc файл. Додайте в кінець файлу

```
export HADOOP_HOME=/home/hadoop/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

6. **source ~/.bashrc**
7. **Відредагуйте файл** **$HADOOP_HOME/etc/hadoop/hadoop-env.sh** та введіть власну версію Java
   export JAVA_HOME=/opt/jdk1.8.0_131/
8. **Відредагуйте** *cd $HADOOP_HOME/etc/hadoop*

Edit core-site.xml

```
    <configuration>

<property>
  <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
</property>
</configuration>
```

Edit hdfs-site.xml

```
<configuration>
<property>
 <name>dfs.replication</name>
 <value>1</value>
</property>

<property>
  <name>dfs.name.dir</name>
<value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
</property>

<property>
  <name>dfs.data.dir</name>
<value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
</property>
```

```
</configuration>
```

Edit mapred-site.xml

```
<configuration>
 <property>
  <name>mapreduce.framework.name</name>
   <value>yarn</value>
 </property>
</configuration>
```

Edit yarn-site.xml

```
<configuration>
 <property>
  <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
 </property>
</configuration>
```

Запустіть команду **hdfs namenode -format**

Запустіть Hadoop **cd $HADOOP_HOME/sbin/**

Запустіть start-dfs.sh скріпт: **start-dfs.sh**

Запустіть start-yarn.sh скріпт: **start-yarn.sh**

Перейдіть в браузер: **localhost:50070**

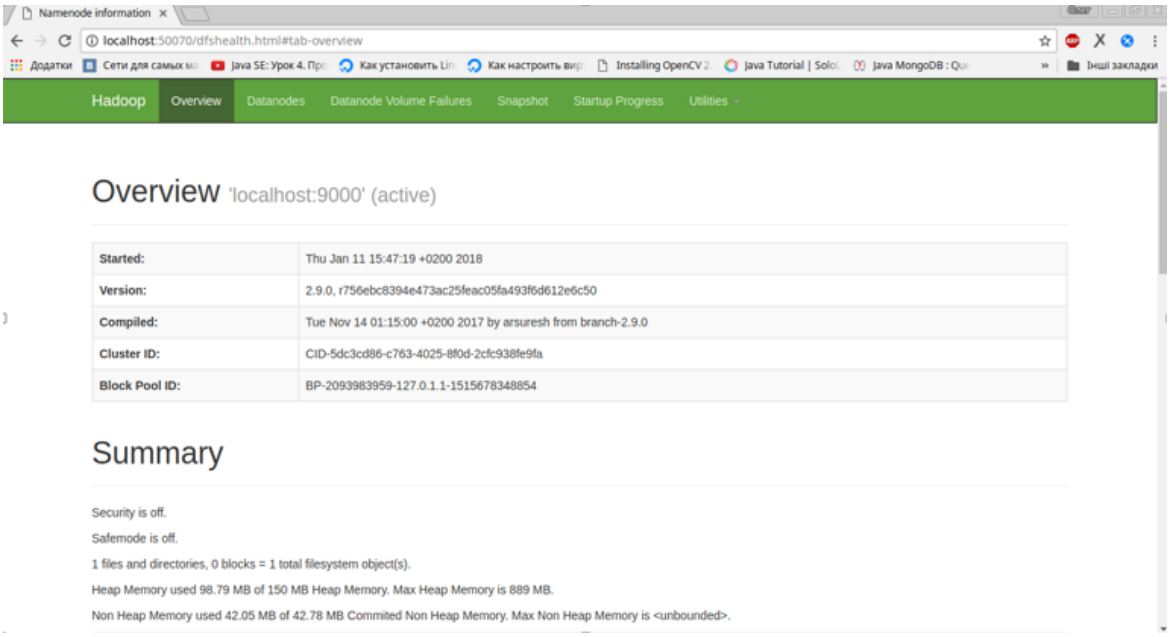

Рисунок 3.4 – Головна сторінка

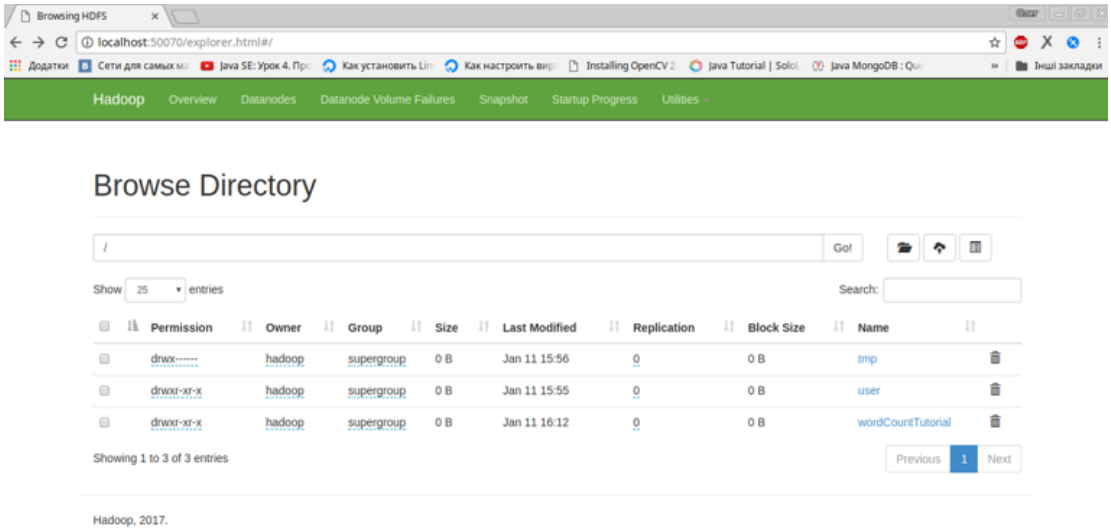

Рисунок 3.5 – Перегляд директорій

Перегляд інформації про запущені ноди:

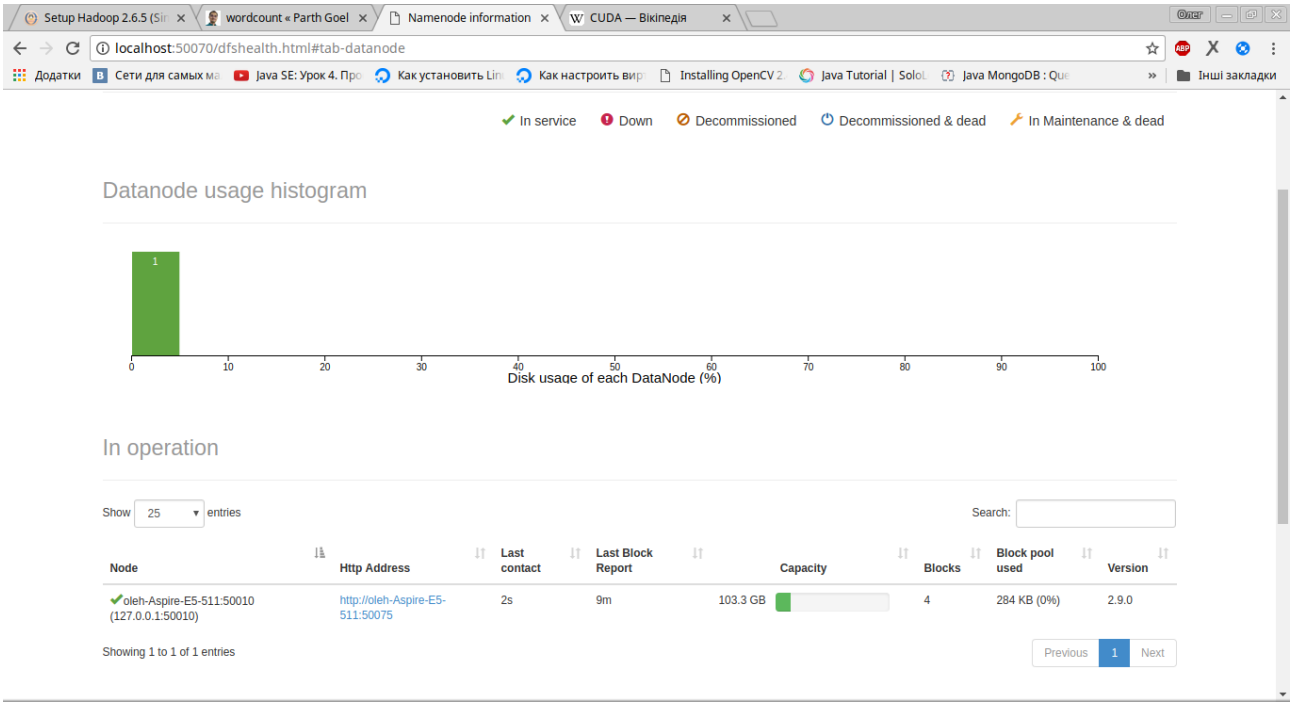

Рисунок 3.6 – Відображення datanode

Результат роботи програми підрахунку слів в тексті в середовищі Eclipse. Вхідний текст знаходиться у файлі «input.txt». Результатом роботи програми є вихідний файл із списком слів та їх кількістю в тексті.

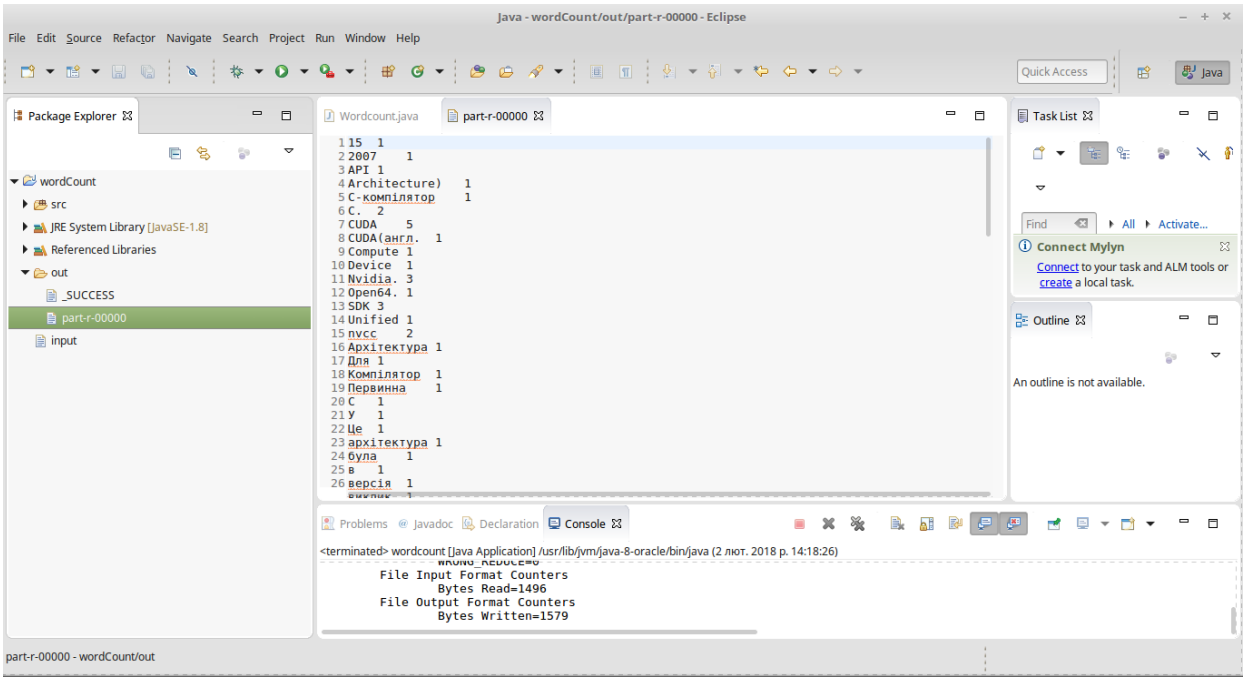

Рисунок 3.7 – Результат роботи програми

Завдання

1. Налаштувати hadoop в ОС сімейства Linux
2. Відобразити працездатність hadoop, навівши скрін-шоти веб інтерфейсу
3. Запустити програму підрахунку кількості слів
4. Навести результати роботи

Додаток А

Код програми підрахунку унікальних слів в тексті

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Wordcount {

 public static class TokenizerMapper
 extends Mapper<Object, Text, Text, IntWritable>{
```

```
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
        }
        }
        }

        public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,Context context) throws IOException,
InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
        sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
        }
        }

        public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
       }
```

Лабораторна робота №4

Тема: CUDA: Інсталяція та основи технології.

Мета: Одержати практичні навички інсталяції та використання технології CUDA.

Теоретичні відомості

Складність обчислювальних завдань вимагає різкого збільшення ресурсів і швидкодії комп’ютерів. Найбільш перспективним напрямком збільшення швидкості розв’язку прикладних завдань є широке впровадження ідей паралелізму в роботу обчислювальних систем. Сьогодні спроектовано і випробувано сотні різних комп'ютерів, що використовують у своїй архітектурі той чи інший вид паралельної обробки даних. Основна складність при проектуванні паралельних програм – забезпечення правильної послідовності взаємодій між різними обчислювальними процесами, а також координація ресурсів, що розділяються між ними.

CUDA розшифровується як Compute Unified Device Architecture і є розширенням мови програмування C, створеним Nvidia. Використання CUDA дозволяє програмісту скористатися потужністю паралельних обчислень відеокарти Nvidia для виконання обчислень загального призначення.

При використанні цієї технології слід знати такі основні поняття:

- **пристрій (device)** – сама відеокарта, графічний процесор (GPU) – виконує команди центрального процесора;
- **хост (host)** – центральний процесор (CPU) – запускає різні завдання на пристрої, виділяє пам'ять тощо;
- **ядро (kernel)** – функція (завдання), що буде виконуватися на GPU.

Запуск та виконання програми на графічному процесорі відбувається у декілька кроків:

1. Хост виділяє необхідну кількість пам’яті на пристрої.

2. Хост копіює дані із своєї пам’яті в пам’ять пристрою.
3. Хост запускає ядро на пристрої.
4. Пристрій виконує це ядро.
5. Хост копіює результати із пам’яті пристрою в свою пам’ять.

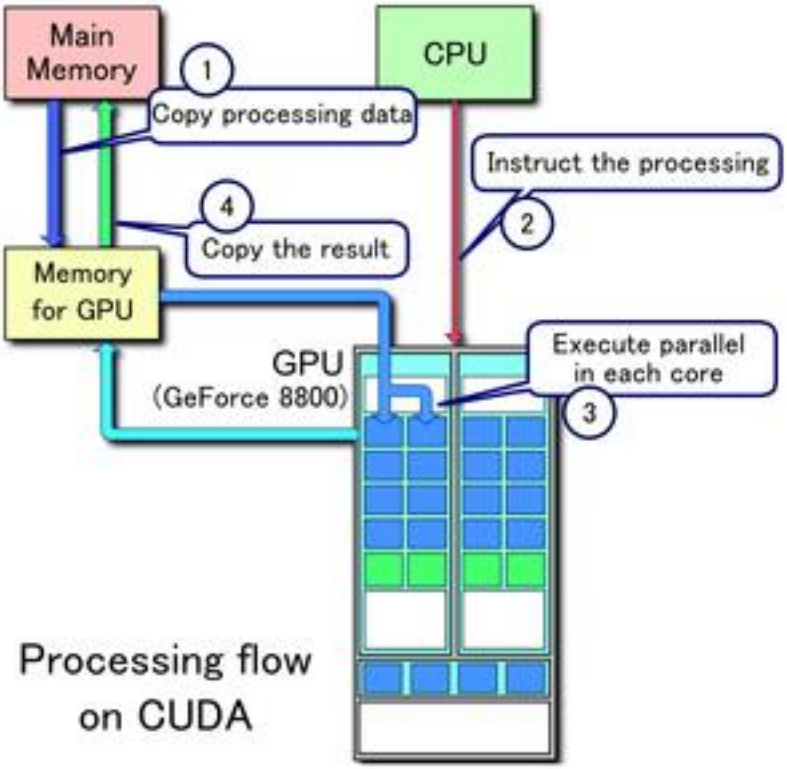

Рисунок 4.1 - Етапи запуску та виконання програми

Хост взаємодіє з графічним процесором через CUDA Runtime API, CUDA Driver API та CUDA Libraries. Runtime та Driver API відрізняються між собою рівнем абстракції. Тобто, перший варіант є більш високого рівня у плані програмування, більш абстрактний, а другий – навпаки більш низького (на рівні драйвера). Загалом, Runtime API є абстрактною обгорткою над Driver API. Під час програмування, ви можете використовувати будь-який варіант.

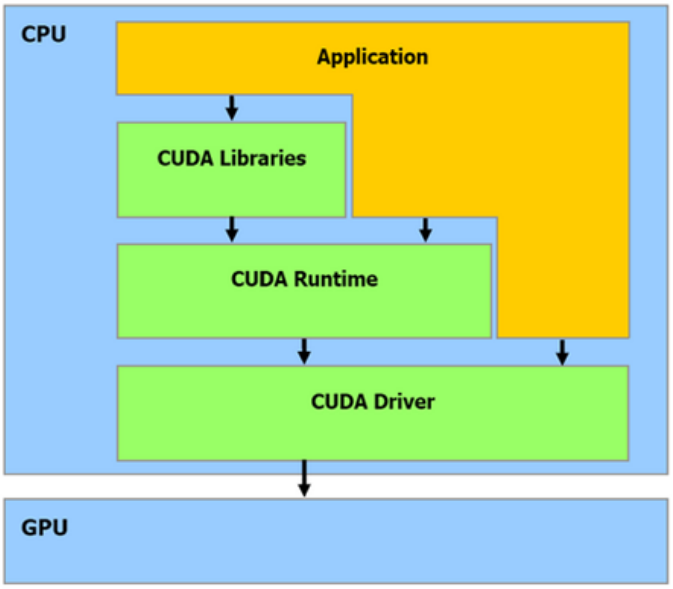

Рисунок 4.2 - Взаємодія CPU та GPU

Такі процесори, як Intel Core 2 Duo та AMD Opteron, добре справляються з одним або двома завданнями одночасно, і виконують ці завдання дуже швидко. З іншого боку, графічні карти добре справляються з великою кількістю завдань одночасно, і виконують ці завдання відносно швидко, бо архітектура GPU побудована трохи інакше, ніж CPU. Оскільки графічні процесори спочатку використовувалися тільки для графічних розрахунків, що припускають незалежну паралельну обробку даних, то GPU призначений саме для паралельних обчислень. Він спроектований так, щоб виконувати велику кількість потоків (елементарних паралельних процесів). GPU містить дуже багато простих арифметико-логічних пристроїв (АЛП), які об'єднані в декілька груп і мають спільну пам'ять (рис. 3). Це допомагає підвищити продуктивність в обчислювальних завданнях, але трохи ускладнює програмування. ***Для досягнення найкращого прискорення необхідно продумувати стратегії доступу до пам'яті й враховувати апаратні особливості.***

GPU орієнтований на виконання програм з великим об'ємом даних та розрахунків і являє собою масив потокових процесорів (Streaming Processor Array), що складається з кластерів текстурних процесорів (Texture Processor Clusters, TPC) (рис. 4). TPC складається з набору мультипроцесорів (SM – Streaming Multi-processor), кожен з яких містить кілька потокових процесорів (SP

– Streaming Processors) або ядер (*у сучасних графічних процесорах к-сть ядер перевищує 1024*). Набір ядер кожного мультипроцесора працює за принципом SIMD (проте, з деякою відмінністю) – реалізація, що дозволяє групі процесорів, що працюють паралельно, працювати з різними даними, але при цьому всі вони в будь-який момент часу повинні виконувати однакову команду. Якщо по-простому, то декілька потоків виконують одне і те ж саме завдання.

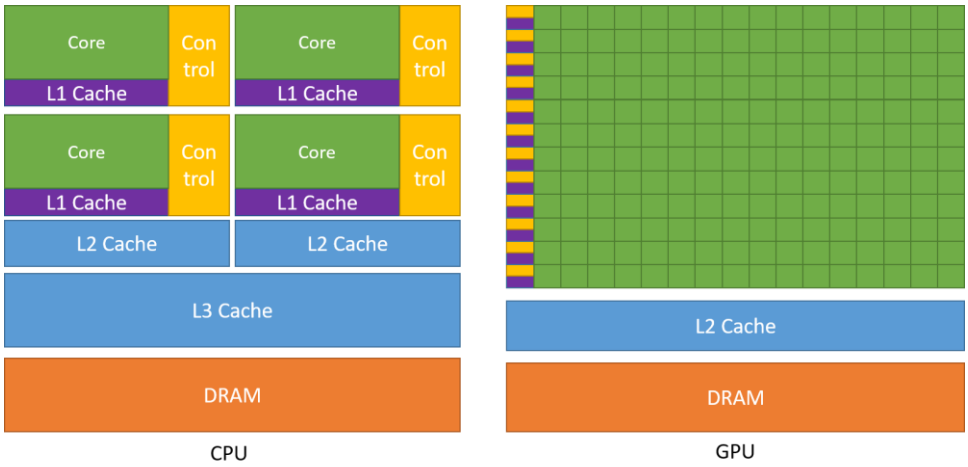

Рисунок 4.3 - Відмінності архітектури CPU від GPU

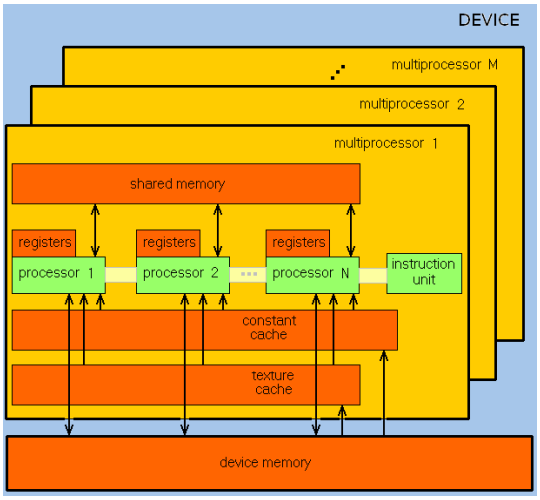

Рисунок 4.4 - Мультипроцесори GPU

CUDA використовує велику кількість окремих потоків для розрахунків. Всі вони групуються в ієрархію – grid / block / thread. Верхній рівень – grid – відповідає ядру й об'єднує всі потоки, що виконують дане ядро. Grid – одновимірний або

двомірний масив блоків (block). Кожен блок (block) являє собою 1 / 2 / 3 -мірний масив потоків (threads). При цьому кожен блок являє собою повністю незалежний набір скоординованих між собою потоків. Потоки з різних блоків не можуть між собою взаємодіяти.

Є ще таке поняття, як warp – група з 32 потоків. Так ось, тільки потоки в межах однієї групи (warp) можуть фізично виконуватися одночасно. Потоки різних варпів можуть знаходитися на різних стадіях виконання програми. Такий метод обробки даних позначається терміном SIMT (Single Instruction – Multiple Theads). Управління роботою варпів виконується на апаратному рівні (рис. 5).

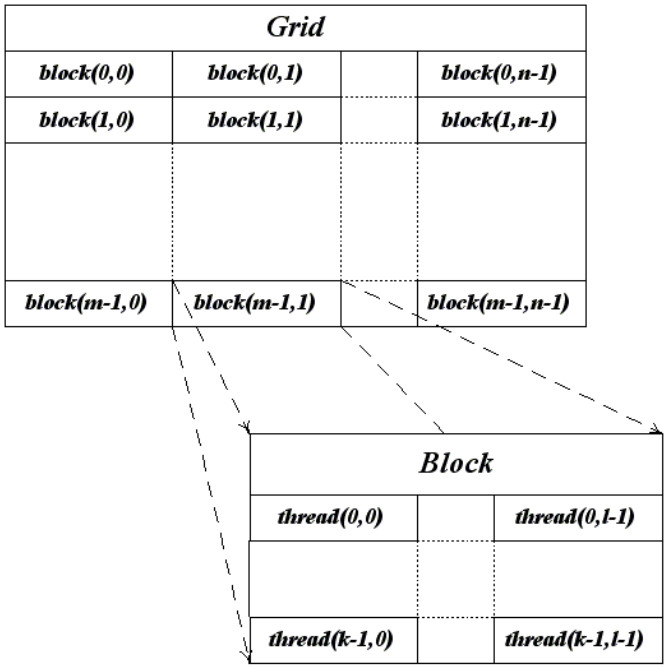

Рисунок 4.5 - Ієрархія потоків.

Одна важлива річ, про яку слід пам’ятати, полягає в тому, що всю вашу програму не потрібно писати на CUDA. Якщо ви пишете велику програму з інтерфейсом користувача та багатьма іншими функціями, то більша частина вашого коду буде написана мовою C++ або іншою мовою, яку ви виберете. Тоді, коли потрібно щось надзвичайно інтенсивне з точки зору обчислень, ваша програма може просто викликати функцію ядра CUDA, яку ви написали.

Отже, основна ідея полягає в тому, що CUDA слід використовувати лише для найбільш інтенсивних обчислювальних частин вашої програми.

Найважливішим моментом є те, що CUDA добре підходить лише для обчислень, які можуть бути розбиті та виконані тисячами потоків паралельно.

Хід роботи.

Інсталяція CUDA на Windows ПК із графічним процесором Nvidia

1) Перевірити чи графічний процесор, що використовується, підтримує технологію CUDA. Зробити це можна на сайті https://developer.nvidia.com/cuda-gpus.
2) Перейти на сайт https://developer.nvidia.com/cuda-downloads та завантажити інсталяційний файл відповідно до операційної системи.
3) Запустити інсталятор та дотримуватися усіх кроків (оновити драйвери графічного процесора, якщо потрібно).
4) Після успішної інсталяції виконати в терміналі команду nvcc -V. На екрані повинна відобразитися інформація про версію CUDA.

Інсталяція CUDA у віртуальному середовищі Google Colab

1) Відкрити файл https://colab.research.google.com/drive/1uLpXx-qlgUN-AntDZtMI7yKfF8pVOoxa?usp=sharing та виконати інструкції.

Структура звіту лабораторної роботи.

- Титульна сторінка.
- Тема та мета роботи.
- Копія екрану з результатом виконання команди nvcc -V.
- Висновки.

Контрольні запитання

1. Що таке CUDA?
2. Основні системні вимоги для інсталяції CUDA?
3. Переваги GPU над CPU?

Лабораторна робота №5

Тема: Розробка клієнт-серверних асинхронних веб – додатків.

Мета: Навчитися використовувати засоби php vue.js, mysql, php для розробки асинхронних веб – додатків.

Теоретичні відомості

AJAX є одним із підходів до розробки коробки користувацьких веб-інтерфейсів, який пропонує «фоновий» обмін даними між браузерами та серверними ресурсами. Асинхронний спосіб оновлення даних веб-сторінки дозволяє не завантажувати їх зміст повністю. Завдяки цьому веб-ресурс працює більше плавно і швидко. Також це може дозволити економічний трафік, що особливо актуально в умовах, якщо немає безлімітного інтернету. Але коли на сайті використовується технологія AJAX, користувачі помічають приватну змінену сторінку. Тому розробникам слід подумати про впровадження індикації цих змін. Користувач повинен розуміти, що зараз відбувається фоновий обмін даними із сервером. Ще одна важлива особливість полягає в тому, що не всі браузери працюють з Аяксом. Його не підтримують, наприклад, старі та текстові версії браузерів. Що ще іноді Javascript блокує користувацькими налаштуваннями. Тому розробники часто рекомендують шукати для технологій більше гнучкі альтернативи та інші методи відображення даних на веб-ресурсах.

Основні переваги та недоліки AJAX

AJAX не завжди рекомендує використовувати розробників. У окремих варіантах технології приносять свої плюси, але іноді можна погано познайомитись на роботі веб-ресурсу**.**

До основного перевагу підходу належать:

зменшення трафіку - за допомогою приватного оновленого ресурсу не потрібно перезавантажувати сторінку;

зниження завантажень на сервері - технологія може робити менше записів до бази даних;

швидка робота сторінки веб-сайту - відгук про дії користувачів, що динамічно;

поліпшення функціональності веб-ресурсу.

Недоліки AJAX

Але при цьому, застосування AJAX може бути загрожує суттєвими недоліками. Основні мінуси роботи веб-сайту з використанням технологій Аякс буде:

- відсутність гарантійної безпеки - весь програмний код скриптів, доступний для перегляду в браузері і цим може скористатися зловмисниками;
- історія оцінки сторінки не працює, і це погано для ведення статистики та аналізу поведінки оцінки;
- погіршена індексація - сторінки, на яких використовується AJAX, можуть бути гіршими сприйматися пошуковими роботами. Іноді бувають випадки, коли вони разом виглядають з індексу.

Завдання до виконання лабораторної роботи:

1) Налаштувати веб сервер apache2 на ОС сімейста лінукс або на ОС сімейства windows з використаннях спеціалізовапних програмних пакетів (Денвер, XAMP тощо).
2) Використовуючи даний ресурс http://ki.tneu.edu.ua/?c=showArticle&f=show&p=362 реалізувати сайт для відправки повідомлень з допомогою технології AJAX на основі технологій vue.js, php, mysql.
3) Оформити звіт.

Контрольні запитання

Поняття асинхронності.

Асинхронні запити до веб – сервера

Засоби мови програмування JavaScript для реалізації асинхронності

Директиви бібліотеки vue.js

Додаток А

Приклад коду для забезпечення асинхронної передаці даних на сервер.

```
<script>

    var app = new Vue({
        el: '#vueapp',
        data: {
            name: '',
            email: '',
            country: '',
            city: '',
            job: '',
            contacts: []
        },
        mounted: function () {
            console.log('Hello from Vue!')
            //this.getContacts()
        },

        methods: {
            getContacts: function(){
```

```
    axios.get('api/contacts.php')
        .then(function (response) {
            //console.log(response.data);
            app.contacts = response.data;

        })
        .catch(function (error) {
            console.log(error);
        });
},
createContact: function(){
    console.log("Create contact!")

    let formData = new FormData();
    console.log("name:", this.name)
    formData.append('name', this.name)
    formData.append('email', this.email)
    formData.append('city', this.city)
    formData.append('country', this.country)
    formData.append('job', this.job)

    var contact = {};
    formData.forEach(function(value, key){
        contact[key] = value;
    });

    axios({
        method: 'post',
        url: 'api/contacts.php',
        data: formData,
```

```
                config: { headers: {'Content-Type': 'multipart/form-data' }}
            })
                .then(function (response) {
                    //handle success
                    console.log(response)
                    app.contacts.push(contact)
                    //app.resetForm();
                })
                .catch(function (response) {
                    //handle error
                    console.log(response)
                });
        },
        resetForm: function(){
            this.name = '';
            this.email = '';
            this.country = '';
            this.city = '';
            this.job = '';
        }
    }
  })
</script>
```

РЕКОМЕНДОВАНІ ДЖЕРЕЛА ІНФОРМАЦІЇ

1. Аксак Н.Г. Паралельні та розподілені обчислення: підручник / Н.Г. Аксак, О.Г. Руденко, А.М. Гуржій. - Х.: Компанія СМІТ, 2009. - 480с.
2. Вільямс Ентоні Паралельне програмування на C++ в дії. практика розробки багатопотокових програм/Пер. з англ. ДМК Прес – 2016 – 672 с.
3. Лафорі Роберт Структури даних та алгоритми в Java. – 2016 – 720 с.
4. Daniel Kusswurm Modern Parallel Programming with C++ and Assembly Language. 1st Ed. – Apress – pp. 633 – 2022
5. Баранов М. А. Паралельна версія алгоритму кластеризації – 2014
6. Гергель В.П. Теорія та практика паралельних обчислень/Гергель В.П. - М.: ІНТУІР.РУ Інтернет-Університет Інформаційних технологій, 2007.
7. Х'юз К., Х'юз Т. Паралельне та розподілене програмування з використанням C++: Пер. з англ. - М: Видавничий дім «Вільямс», 2011. – 672с.
8. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services 1st Edition. O'Reilly Media – p 166 – 2018
9. Domingos P. The Master Algorithm - Penguin., 2017. - 877 с.
10. Cheng John Professional CUDA C Programming / John Cheng, Max Grossman, Ty McKercher/John Wiley & Sons, Inc., Indianapolis, Indiana - 2014.
11. Eijkhout V. Introduction to High Performance Scientific Computing / Victor Eijkhout // Paperback – December 28, 2015
12. R. Trobec . Introduction to Parallel Computing: From Algorithms to Programming on State-of-the-Art Platforms - Springer (October 4, 2018) - 270 pages.
13. Згуровський М.З., Петренко А.І. Е-наука на шляху до семантичного Грід. Частина 1: Об’єднання Web- і Грід- технологій // Системні дослідження та інформаційні технології. - Київ, №1, 2010. - с.26-38.
14. Згуровський М.З., Петренко А.І. Е-наука на шляху до семантичного Грід. Частина 2: Семантичний Web- і семантичний Грід // Системні дослідження та інформаційні технології. - Київ, №2, 2010. - с. 7-25.
15. Java – Multithreading. Електронний ресурс. Режим доступу: https://www.tutorialspoint.com/java/java_multithreading.htm.19.05.2022
16. Laravel AJAX Tutorial Example. Електронний ресурс. Режим доступу: https://appdividend.com/2018/02/07/laravel-ajax-tutorial-example/.19.05.2022

О.Й. Піцун

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт

з дисципліни «Паралельні та розподілені комп’ютерні системи»

спеціальність: 123 - Комп’ютерна інженерія

галузь знань: 12 – Інформаційні технології

освітній рівень «Бакалавр»

Підписано до друку 24.10.2022 р.
Формат 60х84/16. Папір офсетний.
Друк на дублікаторі.
Умов.- друк арк. 1.4 Обл.-вид. арк 1.5.
Тираж 25 прим.

Віддруковано ФОП Шпак В.Д.
Свідоцтво про державну реєстрацію
серія В02 №924434 від 11.12.2006 р.
Свідоцтво платника податку: Е № 897220