ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КЕРУЮЧИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Кафедра спеціалізованих комп’ютерних систем

КЛАСТЕРНІ ОБЧИСЛЕННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт
з дисципліни

«ОРГАНІЗАЦІЯ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ»

Харків – 2019

Методичні вказівки розглянуто і рекомендовано до друку

на засіданні кафедри спеціалізованих комп’ютерних систем 12 лютого 2018 р., протокол № 10.

Описано основи роботи з HPC системами, проектування і розробка програмного забезпечення з застосуванням стандарту OpenMP, створення скриптів запуску та використання їх для виконання програм на високопродуктивному обчислювальному кластері.

Призначено для студентів факультету ІКСТ зі спеціальності 123 «Комп’ютерна інженерія» першого рівня вищої освіти (бакалавр) усіх форм навчання. Можливе застосування методичних вказівок для інших дисциплін на інших рівнях вищої освіти та галузях (освітніх програмах).

Укладачі:

проф. С. В. Лістровий,
асист. Ю. В. Савін

Рецензент

проф. М. А. Мірошник

КЛАСТЕРНІ ОБЧИСЛЕННЯ

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни

«ОРГАНІЗАЦІЯ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ»

Відповідальний за випуск Савін Ю. В.

Редактор Решетилова В. В.

Підписано до друку 01.06.18 р.

Формат паперу 60х84 1/16. Папір писальний.

Умовн.-друк.арк. 2,5. Тираж 35. Замовлення №

Видавець та виготовлювач Український державний університет
залізничного транспорту,
61050, Харків-50, майдан Фейєрбаха, 7.
Свідоцтво суб’єкта видавничої справи ДК № 6100 від 21.03.2018 р.

ЗМІСТ

Вступ………………………………………………………… 4
Лабораторна робота 1. Кластер. Побудова і принципи роботи……………………………………………………….. 5
Лабораторна робота 2. Алгоритм Дейкстри. Знаходження найкоротшого шляху від однієї з вершин графа до решти з використанням стандарту OpenMP…………………………….. 22
Лабораторна робота 3. Алгоритм Форда. Знаходження найкоротшого шляху від однієї з вершин графа до решти з використанням стандарту OpenMP…………………………….. 37
Список літератури…………………………………………….. 54

ВСТУП

Методичні вказівки призначені для проведення лабораторних робіт, основною метою яких є оволодіння основними прийомами проектування паралельних програм для вирішення широкого кола обчислювальних завдань.

У наш час високопродуктивні обчислення набули поширення завдяки універсальності можливостей застосування, простоті програмування, скороченню часу отримання кінцевого результату, гнучкості методів, доступності засобів розробки.

Мета лабораторного практикуму – ознайомитися з основами проектування паралельних програм для вирішення складних обчислювальних завдань, їх реалізації та виконання на високопродуктивних кластерних системах.

Приклади, наведені в лабораторному практикумі, створено програмною мовою С++ з використанням стандарту паралельного напису програм OpenMP і вирішують задачі знаходження найкоротшого шляху в повнозв'язному неорієнтованому графі між заданою та усіма іншими вершинами графа та упорядковування великих масивів даних по зростанню.

ЛАБОРАТОРНА РОБОТА 1
Кластер. Побудова і принципи роботи

Мета: ознайомитися з улаштуванням кластера і набути навичок роботи з написання та компіляції програм, пакетних скриптів і постановки завдань в чергу менеджера ресурсів.

Теорія

У сучасному світі наука і промисловість для вирішення завдань, що виникають перед ними, вимагають все більших обсягів математичного моделювання. Обробка даних вже не може виконуватися в рамках одного сервера. Для вирішення великих завдань сервери об'єднують в єдину систему, яку називають **кластером**.

Кластер – сукупність серверів, з'єднаних між собою каналами зв'язку, що мають загальне управління і об'єднані для виконання єдиного завдання (рисунок 1.1).

Рисунок 1.1 — Обчислювальний кластер

Залежно від організації кластери поділяються на кілька видів:

• кластери високої доступності (відмовостійкі кластери);

• кластери розподілу навантаження (кластери з балансуванням мережевого навантаження);

• обчислювальні кластери (High Performance Computing Cluster, HPC). Китайський суперкомп'ютер Sunway TaihuLight (рисунок 1.2) очолив топ-500 найбільш продуктивних машин на планеті (листопад 2016 р.). Для створення Sunway TaihuLight було використано понад 10 млн процесорних ядер. Завдяки цьому суперкомп'ютер виконує 93 квадрильйони математичних операцій в секунду, т. ч. його продуктивність досягає 93 петафлопс. Така продуктивність затребувана для проектування автомобілів, кораблів розрахунку аеродинаміки літака, аналізу медичного препарату і т. п.

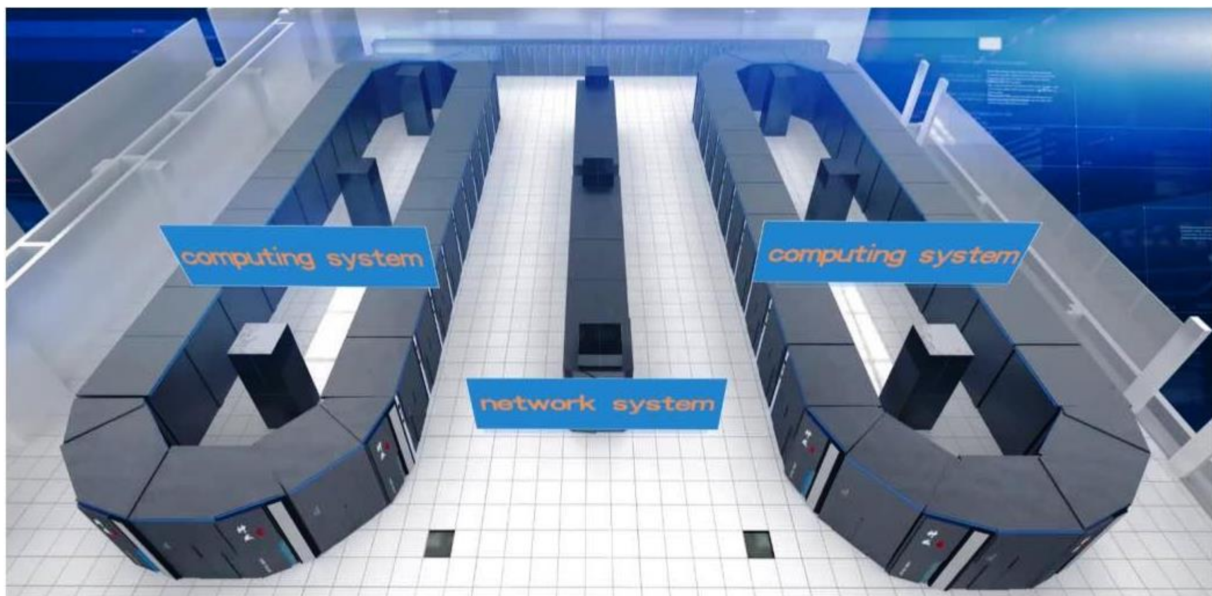

Рисунок 1.2 – Китайський суперкомп'ютер Sunway TaihuLight

В даному курсі розглядаються обчислювальні кластери. Основними вимогами до даного типу кластерів є:

• наявність високопродуктивних процесорів;

• високошвидкісні канали зв'язку (низька латентність).

При класичній побудові сервери всередині кластера об'єднуються між собою трьома мережами:

• обчислювальною;

• передачі даних;

• управління.

У багатьох сучасних рішеннях обчислювальна мережа і мережа передачі даних об'єднані. Обчислювальна мережа являє собою високопродуктивну систему, що має дуже велику пропускну спроможність і низьку затримку. Використовуються для високошвидкісного обміну виконуваних кодом програм, між обчислювальними вузлами, а також для моніторингу стану вузлів і роботи з чергами менеджером ресурсів. Як така мережа використовується InfiniBand. Параметри мережі наведені в таблиці 1.1.

Таблиця 1.1

Покоління:	SDR	DDR	QDR	FDR-10	FDR	EDR	HDR	NDR
Ефективна пропускна спроможність, Гбіт/с, на 1x шину	2	4	8	10	14	25	50	
Ефективні швидкості для 4x і 12x шин, Гбіт/с	8 24	6 48	32 96	41.25 123.75	54.54 163.64	100 300	200 600	
Кодування (біт)	8/10	8/10	8/10	64/66	64/66	64/66		
Типові затримки, мкс	5	2.5	1.3	0.7	0.7	0.5		
Рік появи	2001	2005	2007		2011	2014	2017	після 2020

Мережі передачі даних будуються на основі швидкісних систем, представлених мережами Infiniband, 10G Ethernet. Використовується для швидкісного обміну між обчислювальними вузлами і сховищем даних.

Мережі управління будуються на основі стандартних 1G Ethernet або Fast Ethernet. Використовуються для віддаленого управління сервером доступу і обчислювальними вузлами. В сучасних серверах підключається до IPMI інтерфейсу. На рисунку 1.3 подано структурну схему кластера.

Як видно на рисунку, користувачі кластера мають доступ до сервера доступу (ведучого вузла) через мережу Інтернет. На даній схемі зображено два сервери доступу. До них підключено загальне файлове сховище. На серверах встановлено менеджер розподілених ресурсів. Він дозволяє працювати з кластером в пакетному режимі.

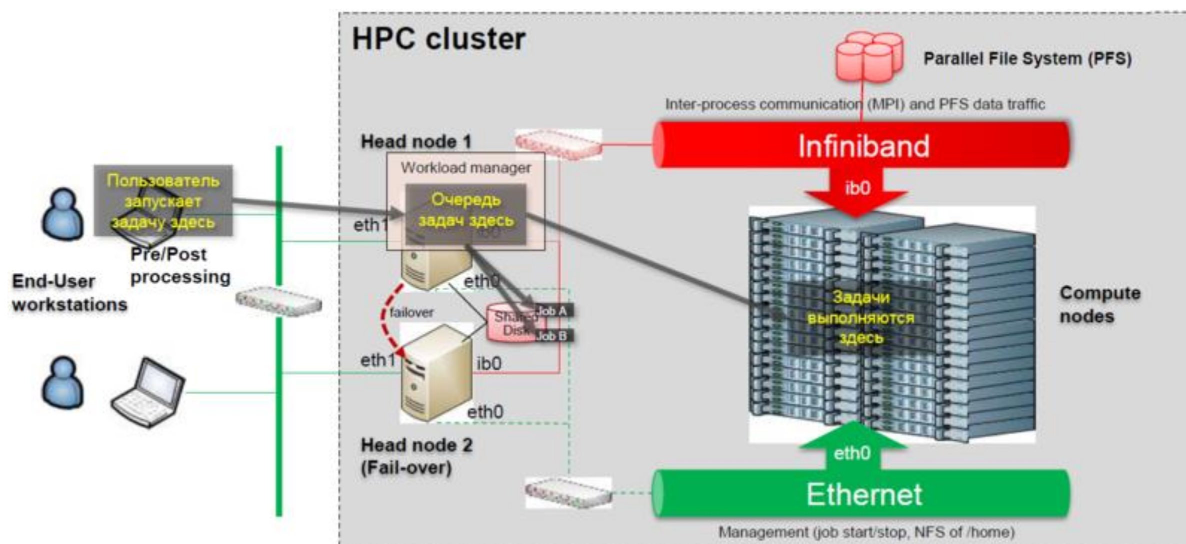

Рисунок 1.3 – Структурна схема кластера

Найпоширенішим вільним менеджером ресурсів вважається **TORQUE** (Terascale Open-source Resource and QUEue Manager). У файловому сховищі містяться дані користувачів, системні дані, менеджер ресурсів, черги завдань і планувальник. Сервери доступу не беруть участі в обчисленнях. Вони виконують завдання, пов'язані з функціонуванням і життєзабезпеченням кластера. Далі йдуть обчислювальні вузли. На кожному з них запущена система моніторингу стану вузла, роботи з обчислювальними ресурсами і чергами. На вузлах виконуються тільки обчислення. Як правило, вони не мають свого дискового простору. Останнім елементом кластера є файлове сховище з розподіленою файловою системою, яке використовується для зберігання коду, виконуваного вузлами в даний момент, тимчасових файлів програми, файлу підкачування. Всі файлові сховища побудовані з використанням RAID 5, 6, 10, 50 рівнів.

Кластер в своєму складі повинен мати рідинну або повітряну систему охолодження, використовувати системи безперебійної подачі живлення і клімат-контролю.

На рисунку 1.4 подано найпотужніші суперкомп'ютери в світі за останні 25 років, які очолюють рейтинг TOP500 і перші 10 місць рейтингу (листопад 2016 р.).

Їхні характеристики подано в таблиці 1.2.

No.1 in Jun 2016 Sunway TaihuLight: National Supercomputing Center in Wuxi

No.1 in Jun 2016 Tianhe-2 (MilkyWay-2): National University of Defense Technology

No.1 from Jun 2013 until Nov 2015 Titan: Oak Ridge National Laboratory

No.1 in Nov 2012 Sequoia: Lawrence Livermore National Laboratory

Рисунок 1.4, аркуш 1

No.1 in Jun 2012 K Computer: RIKEN Advanced Institute for Computational Science

No.1 from Jun 2011 until Nov 2011 Tianhe-1A: National Supercomputing Center in Tianjin

No.1 in Nov 2010 Jaguar: Oak ridge National Laboratory

No.1 from Nov 2009 until Jun 2010 Roadrunner: Los Alamos National Laboratory

Рисунок 1.4, аркуш 2

No.1 from Jun 2008 until Jun 2009
BlueGene/L: Lawrence Livermore National Laboratory

No.1 from Nov 2004 until Nov 2007
The Earth Simulator: Earth Simulator Center

No.1 from Jun 2002 until Jun 2004
ASCI White: Lawrence Livermore National Laboratory

No.1 from Nov 2000 until Nov 2001
ASCI Red: Sandia National Laboratory

Рисунок 1.4, аркуш 3

No.1 from Jun 1997 until Jun 2000 _ CP-PACS: University of Tsukuba

No.1 in Nov 1996 _Hitachi SR2201: University of Tokyo

No.1 in Jun 1996 _Intel XP/S 140 Paragon: Sandia National Labs

No.1 in Jun 1994 _Numerical Wind Tunnel: National Aerospace Laboratory of Japan

Рисунок 1.4, аркуш 4

No.1 in Nov 1993 CM-5: Los Alamos National Lab

Рисунок 1.4, аркуш 5

Таблиця 1.2

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890

Продовження таблиці 1.2

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
5	DOE/SC/LBNL/NERSC United States	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	622,336	14,014.7	27,880.7	3,939
6	Joint Center for Advanced High Performance Computing Japan	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path Fujitsu	556,104	13,554.6	24,913.5	2,719
7	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
8	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	206,720	9,779.0	15,988.0	1,312
9	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
10	DOE/NNSA/LANL/SNL United States	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	301,056	8,100.9	11,078.9	4,233

Маємо зауважити, що наявність в країні потужного HPC кластера свідчить про її високий науково-технологічний рівень розвитку і є предметом національної гордості.

Основною операційною системою, використовуваною на сучасних кластерах, є різні різновиди Лінукс в режимі командного рядка (**bash**). Для роботи в даному середовищі необхідно знати основний набір команд, перерахуємо їх:

• **ls** або **ll** (**ls -l**) - показати вміст каталогу. У першому випадку всі неприховані файли і папки виводяться в рядок, у другому кожен файл виводиться з повним переліком атрибутів по одному файлу в рядку;

• **cd** <ім'я каталогу> або **cd** .. - перехід в (вихід з) каталог (а);

• **touch** <ім'я файлу> - створення нового пустого файлу з заданим ім'ям;

• **mcedit** <ім'я файлу> - редагування файлу з заданим ім'ям;

• **mc** - виклик файлового менеджера (аналог програми FAR);

• **less** <ім'я файлу> - перегляд вмісту заданого файлу з можливістю скролінгу;

• **exit** – закінчення сеансу роботи.

Для компіляції вихідного тексту програм використовуються вільні компілятори, що входять до складу ОС: **gcc** і **g++**. Якщо файл створено мовою **C** і має розширення **.c**, то він компілюється командою **gcc**. Якщо файл створено мовою **C++** і має розширення **.cpp**, то він компілюється командою **g++**. Формати команд багато в чому збігаються. Наведемо приклад команди:

gcc (**g++**) <**ім'я файлу з вихідним текстом програми з розширенням .c** (**.cpp**)> - <**ключі компілятора**> **-o** <**ім'я виконуваного файлу з розширенням .bin**>

Щоб користувач міг взаємодіяти з кластером, необхідно знати команди роботи з чергами менеджера ресурсів. Наведемо основні з них:

• **pbsnodes** або **qnodes** – команда виводить інформацію про склад і стан всіх вузлів кластера;

• **qstat** – команда виводить інформацію про стан черг на кластері;

• **qsub** <ім'я файлу скрипта з розширенням **.pbs**> – команда постановки завдання в чергу на кластер;

• **qdel** <номер завдання в черзі> – команда зняття завдання із зазначеним номером з черги на кластері.

Для постановки завдання в чергу використовується скрипт. Розглянемо формат і вміст необхідного скрипта. Приклад скрипта наведено нижче:

#! / Bin / sh – обов'язковий рядок для bash скриптів

#PBS -d / home / student1 / <прізвище студента> / lab1 / – шлях до виконуваної програми. У цей каталог зберігаються файли результату і помилок.

#PBS -l walltime = 01: 00: 00 – максимальний час виконання програми. Після досягнення даного часу завдання зупиняється, результати зберігаються.

#PBS -l nodes = 1: ppn = 3 – необхідна кількість вузлів, необхідна кількість ядер.

#PBS -N <ім'я файлу> – ім'я з яким зберігаються файли результату і помилок.

echo "File containing nodes:" – файл містить інформацію про запущену задачу.

echo $ PBS_NODEFILE

echo "Nodes for computing:" – ім'я вузла, на якому виконується обчислення.

cat $ PBS_NODEFILE

echo "Start date:` date` " – час старту програми.

./<імя файлу> .bin – ім'я виконуваного файлу.

echo "End date:` date` " – час завершення програми.

Хід виконання лабораторної роботи

1 Вивчити теоретичний розділ лабораторної роботи.

2 Підключитися до кластеру. Для цього запустити програму PuTTy.exe.

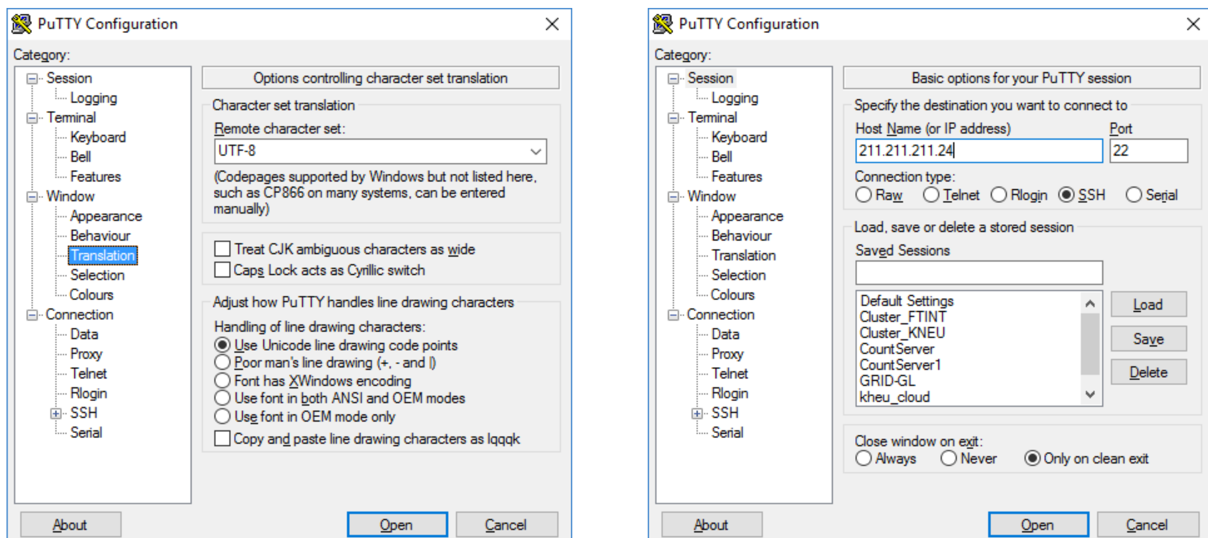

Рисунок 1.5 – Інтерфейс програми PuTTy.exe.

Перевірити налаштування мови кодування (рисунок 1.5). Повинна бути встановлена в **UTF-8**. Для підключення до кластера використовуйте IP-адресу 211.211.211.24.

У вікні введіть логін і пароль:

login: studentX

password: studentX

де X – номер курсу. Пароль при введенні на екрані не відображається.

3 Після входу на кластер створіть каталог зі своїм прізвищем:

mkdir <прізвище студента>

4 Перейдіть в нього: **cd** <прізвище студента>

5 Створіть всередині каталогу ще один з ім'ям **lab1**.

6 Перейдіть в нього.

7 Виконайте команду **pbsnodes (qnodes)** для перевірки працездатності кластера.

8 Створіть файл з розширенням **.cpp**.

9 Наберіть в ньому текст програми-прикладу наведеної нижче:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <cstdlib>
#include <omp.h>
using namespace std;

int main(int argc, char**argv)
{
	if (argc != 3)
		cout << "!!!ERROR!!! Example of command: ./sortBetcer.bin
<dimension of array> <number of processor cores> !!!ERROR!!!" <<
endl, exit(1);
	cout <<
"**********************************************************
***************************************************" << endl;
	const int N = atoi(*++argv); // Enter the size array
	const int threads = atoi(*++argv); // Enter the number of processors
	cout << "\t\t\t\tDimension of array = " << N << "\tNumber of
processor cores = " << threads << endl;
	int i, t, p = 1, q, r, d, tmp;
	vector<int> massiv(N);
	t = log(N) / log(2);
```

```
	for (i = 0; i < N; i++)
		massiv[i] = abs(100000 - rand() % 100000);
	/* cout << "\t\t\tmassiv:";
	for (i = 0; i < N; i++)
	cout << " " <<  massiv[i];
	cout << endl; */
	p <<= --t;
	double t1 = omp_get_wtime();
	while (p > 0)
	{
		r = 0;
		d = p;
		q = 1, q <<= t;
		while (q >= p)
		{
#pragma omp parallel for private (i, tmp) shared (massiv, p, r, d)
num_threads(threads)
			for (i = 0; i < (N - d); i++)
			{
				if ((i & p) == r)
					if (massiv[i] > massiv[i + d])
					{
						tmp = massiv[i + d];
						massiv[i + d] = massiv[i];
						massiv[i] = tmp;
					}
			}
			d = q - p;
			q >>= 1;
			r = p;
		}
		p >>= 1;
	}
	double t2 = omp_get_wtime();
	/* cout << "\t\t   sort massiv:";
	for (i = 0; i < N; i++)
	cout << " " <<  massiv[i];
	cout << endl;*/
```

```
	cout << "Runtime = " << t2 - t1 << "sec." << endl;
	return 0;
}
```

10 Відкомпілюйте програму-приклад. При компіляції використовуйте ключ **-fopenmp**.

11 Створіть файл скрипта з розширенням **.pbs** для запуску програми-прикладу на кластері.

12 Запустіть завдання в чергу на кластер: **qsub** <ім'я файлу скрепта> .pbs.

13 Перевірте стан черги командою: **qstat**.

14 Перевірте, чи створюються файли звіту і помилок.

15 Розберіть текст програми-прикладу. З’ясуйте призначення усіх змінних, функцій і директив.

16 За результатами роботи програми-прикладу, заповніть таблицю 1.3 показниками часу роботи при різних параметрах кількості ядер і довжини масиву.

Таблиця 1.3

Кіль. ядер, шт.	Довжина масиву, шт.			
	1000	100000	1000000	10000000
1				
2				
3				
6				

Контрольні питання

1 Що називається кластером?
2 Види кластерів.
3 Вимоги до обчислювальних кластерів.
4 Організація мережі в обчислювальному кластері.
5 Опис структурної схеми кластера.
6 Основні команди Лінукс для роботи з кластером.
7 Основні команди для роботи з менеджером ресурсів.
8 Як скомпілювати програми на кластері?
9 Що таке TORQUE?
10 Призначення скриптів.
11 Де зберігається результат роботи виконаного завдання?

12 Що є результатом роботи виконаного завдання?

13 Як перевірити склад кластера і докладний опис його стану?

14 Що робить програма-приклад?

Індивідуальні завдання

Створіть скрипт для запуску програми example.bin з такими вимогами:

варіант 1:

- шлях до виконуваної програми – назва групи;
- максимальний час виконання програми – 5 хвилин;
- необхідна кількість вузлів - 1, необхідна кількість ядер – 4;
- ім'я, під яким зберігаються файли результату і помилок – назва групи;

варіант 2:

- шлях до виконуваної програми – код групи;
- максимальний час виконання програми – 3 хвилини;
- необхідна кількість вузлів – 2, необхідна кількість ядер – 3;
- ім'я, під яким зберігаються файли результату і помилок – код групи;

варіант 3:

- шлях до виконуваної програми – назва кафедри;
- максимальний час виконання програми – 10 хвилин;
- необхідна кількість вузлів – 3, необхідна кількість ядер – 2;
- ім'я, під яким зберігаються файли результату і помилок - назва кафедри;

варіант 4:

- шлях до виконуваної програми – код кафедри;
- максимальний час виконання програми – 3 хвилини;
- необхідна кількість вузлів – 4, необхідна кількість ядер – 1;
- ім'я, під яким зберігаються файли результату і помилок – код кафедри;

варіант 5:

- шлях до виконуваної програми – назва групи;

- максимальний час виконання програми – 5 хвилин;
- необхідна кількість вузлів – 5, необхідна кількість ядер – 6;
- ім'я, під яким зберігаються файли результату і помилок – код кафедри;

варіант 6:
- шлях до виконуваної програми – код групи;
- максимальний час виконання програми – 3 хвилини;
- необхідна кількість вузлів – 6, необхідна кількість ядер – 5;
- ім'я, під яким зберігаються файли результату і помилок – назва групи;

варіант 7:
- шлях до виконуваної програми – назва групи;
- максимальний час виконання програми – 5 хвилин 10 секунд;
- необхідна кількість вузлів – 2, необхідна кількість ядер – 4;
- ім'я, під яким зберігаються файли результату і помилок – назва факультету;

варіант 8:
- шлях до виконуваної програми - код факультету;
- максимальний час виконання програми – 3 хвилини 40 секунд;
- необхідна кількість вузлів - 2, необхідна кількість ядер – 5;
- ім'я, під яким зберігаються файли результату і помилок – код групи;

варіант 9:
- шлях до виконуваної програми – назва кафедри;
- максимальний час виконання програми – 7 хвилин;
- необхідна кількість вузлів - 3, необхідна кількість ядер – 2;
- ім'я, під яким зберігаються файли результату і помилок – назва факультету;

варіант 10:
- шлях до виконуваної програми – код факультету;

- максимальний час виконання програми – 7 хвилини 45 секунд;
- необхідна кількість вузлів – 4, необхідна кількість ядер – 1;
- ім'я, під яким зберігаються файли результату і помилок – код кафедри.

Примітка – назва групи, кафедри, факультету має бути в скороченій формі, наприклад, кафедра спеціалізованих комп'ютерних систем — СКС.

ЛАБОРАТОРНА РОБОТА 2
Алгоритм Дейкстри. Знаходження найкоротшого шляху від однієї з вершин графа до решти з використанням стандарту OpenMP

Мета: ознайомитися з алгоритмом Дейкстри; створити програму, що реалізує програму знаходження найкоротшого шляху від однієї з вершин графа до решти, використовуючи стандарт OpenMP.

Теорія

Заданий простий неорієнтовний зважений граф ***G(V, E)*** (рисунок 2.1), що складається з ***V*** вершин і ***E*** дуг. Кожна дуга ***(u, v)*** має невід'ємні ваги ***w(u, v), w(u, v) ≥ 0*** (таблиця 2.1). Потрібно знайти найкоротші шляхи з деякою стартовою вершиною ***s*** до решти вершин графа ***G(V, E)***. Графічним еквівалентом простору можливих станів системи є стягнуте дерево ***D(i)*** всіх шляхів графа ***G(V, E),*** яке можна будувати від довільної вершини ***i***. (μ^r_{sj}) в графі ***D(s = i)***, від вершини ***s*** до вершини ***j*** рангу ***r***, локальних екстремумів рангу ***r***, а довжину найкоротшого шляху до вершини ***j*** на множині $\min_r d(\mu^r_{sj})^*$ локальних екстремумів на горизонтальній лінійці графа ***D (s = i)***. Найкоротший шлях $d(\mu^r_{sj})^{**}$ у графі ***D(s = i)*** від вершини ***s*** до деякої довільної вершини ***j*** назвемо глобальним екстремумом.

Тоді задача визначення найкоротших шляхів еквівалентна визначенням глобального екстремуму в просторі можливих станів системи, що задається графом ***D (s = i)***. З вершини ***s*** формуємо безлічі шляхів наступного рангу до вершин ***j*** і визначаємо в кожній безлічі локальні. З усіх локальних екстремумів на ярусі вибираємо найменший до вершини ***j*** і вершину ***j*** (назвемо його глобальним на ярусі) виключаємо з подальшого аналізу. На основі глобального екстремуму поточного рангу формуємо безлічі шляхів наступного рангу і в них знову виділяємо локальні екстремуми, підтягуємо найкоротший шлях з попереднього ярусу, а потім серед локальних екстремумів виділяємо знову глобальний екстремум на ярусі до вершини ***j*** і вершину j виключаємо з подальшого аналізу. Перевіряємо ***r = n - 1***. Якщо немає, то переходимо до виконання попередньої дії, інакше алгоритм закінчує роботу (таблиця 2.2).

Вхідний граф ***G(V, E)*** зручно подавати в пам'яті обчислювальної системи у вигляді двовимірного масиву **arcs**, індекси якого є номери вершин вихідного графа, і в які було здійснено перехід. Значення в полі матриці являють собою вагові коефіцієнти дуг, що з'єднують вершини і відповідних їх положенню в графі ***G(V, E)***. Для подання графа мовою ***C++*** використовується масив, кожен елемент якого є даними типу ***int***. Змінна **startver** позначає стартову вершину, від якої шукається відстань до всіх інших вершин ***v є V***. Масив **minvertices** використовується для зберігання поточні найкоротшої відстані від стартової вершини **startver** до решти ***v є V*** вершин. Кожен елемент даного масиву являє собою структуру, що зберігає номер поточної і попередньої вершини, з якої був здійснений перехід. Вона необхідна для відновлення найкоротших шляхів з стартової вершини **startver** до решти вершин ***v є V*** графа ***G(V, E)***. На відміну від класичного методу реалізації алгоритму Дейкстри, данний метод дозволяє форматувати початкове значення шляхів не **0** і ∞, що накладає обмеження на поняття ∞ для конкретної реалізації мови програмування. У процесі роботи алгоритму Дейкстри підтримується безліч ***U с V***, що складаються з вершин ***v є V*** графа ***G(V, E)***, для яких найкоротшу відстань вже знайдено. Булевий масив **vertices** служить для зберігання інформації, чи знайдено найкоротшу відстань від стартової вершини **startver** до

вершини ***v***, чи ще ні. У наведеній нижче реалізації дані зчитуються і виводяться в консоль.

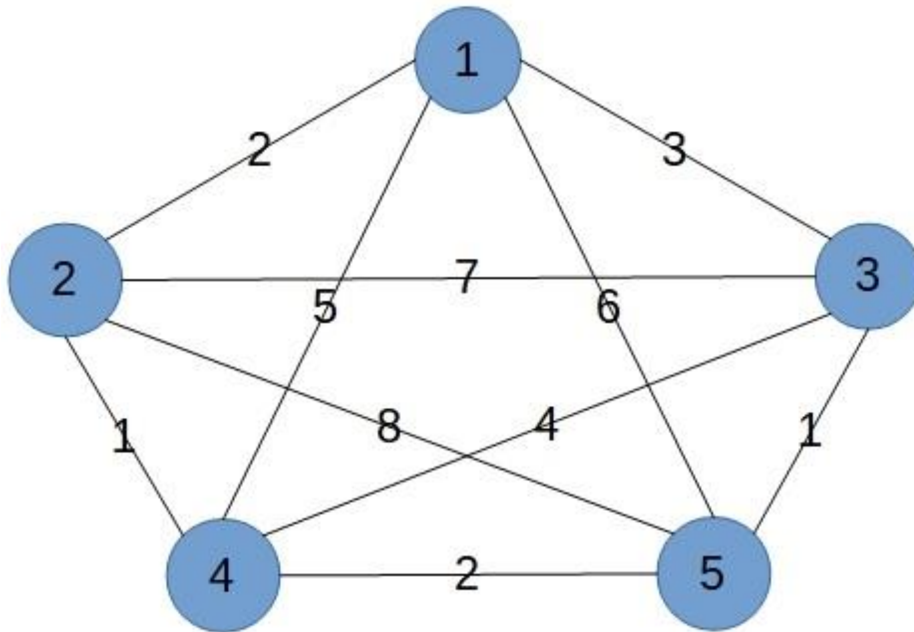

Рисунок 2.1 – Повнозв'язний граф

Таблиця 2.1 – Матриця вагових коэфіцієнтів

Vertices	1	2	3	4	5
1	0	2	3	5	6
2	2	0	7	1	8
3	3	7	0	4	1
4	5	1	4	0	2
5	6	8	1	2	0

Таблиця 2.2 – Реалізація алгоритму Декстри

Vertices					Rezult
3	3 – 1 (3)	3 — 5 — 1 (7) 3 – 1 (3)*	X	X	3 – 1 (3)*
	3 – 2 (7)	3 — 5 — 2 (9) 3 – 2 (7)*	3 – 1 – 2 (5)* 3 – 2 (7)	3 — 5 — 4 – 2 (4)* 3 – 1 – 2 (5)	3 — 5 — 4 – 2 (4)*
	3 – 4 (4)	3 — 5 — 4 (3)* 3 – 4 (4)	3 – 1 – 4 (8) 3 — 5 — 4 (3)*	X	3 — 5 — 4 (3)*
	3 – 5 (1)*	X	X	X	3 – 5 (1)
0	1	2	3	4	Rank

Вхідними даними для роботи програми є:
- кількість вершин;
- кількість процесорних ядер;
- номер стартової вершини;
- номер вершини, до якої розраховується найкоротший шлях.

Вихідні дані:
- зображаються вхідні дані;
- повний час виконання розрахункової частини програми;
- виведення найкоротшого шляху з стартової вершини startver в решту вершин графа ***G(V, E)***;
- виведення найкоротшого шляху з стартової вершини **startver** в задану **finishver**.

Нижче подано текст програми реалізації алгоритму Дейкстри методом стягнених дерев:

```
#include <iostream>
#include <cstdlib>
#include <iomanip>
#include <climits>
#include <omp.h>
```

```
using namespace std;
struct dijminway {
	int ver;
	int backver;
};

int main(int argc, char **argv)
{
	if (argc != 5)
		cout << "!!!ERROR!!! Example of command: ./Dijkstra.bin
<number of vertices> <number of processor cores> <start vertix> <finish
vertix>!!!ERROR!!!" << endl, cin.get(), exit(1);
	cout <<
"*******************************************************
*********************" << endl;
	const int N = atoi(*++argv); // Enter number of vertices
	const int threads = atoi(*++argv); // Enter the number of
processor core
	const int startver = atoi(*++argv); // Enter start vertix
	const int finishver = atoi(*++argv); // Enter finish vertix
	cout << "\tNumber of vertices = " << N << "\tNumber of
processor cores = " << threads << endl;
	cout << "\tStart vertix = " << startver << "\tFinish vertix = " <<
finishver << endl;
	cout <<
"*******************************************************
*********************" << endl;

	int i, j, k, ver, minarc, minver = 0, minwaytemp;
	double t1, t2;

	int **arcs = new int *[N];
	for (i = 0; i< N; i++)
		arcs[i] = new int[N];
	dijminway *minvertices = new dijminway[N];

	bool vertices[N];
	int *minway = new int[N];
```

```
	//srand(time(NULL));
	for (i = 0; i < N; ++i) {
		minway[i] = arcs[i][i] = 0;
		for (j = i + 1; j < N; ++j) {
		    arcs[i][j] = rand() % 10;
			arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
			arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
			arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
			arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
			arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
			arcs[j][i] = arcs[i][j];
		}
	}

	cout << "\t\t\t\tEnd of rendom" << endl;
	cout <<
"*****************************************************
*********************" << endl;

	cout << "        ";
	for (i = 0; i < N; ++i)
	cout << setw(3) << i + 1;
	cout << endl;
	cout <<
"*****************************************************
*********************" << endl;
	for (i = 0; i < N; ++i) {
	cout << setw(5) << i + 1 << " * ";
	for (j = 0; j < N; ++j)
	cout << setw(3) << arcs[i][j];
	cout << endl;
	}
	cout <<
"*****************************************************
*********************" << endl;

	ver = startver - 1;
	omp_set_num_threads(threads);
```

```
	t1 = omp_get_wtime();

#pragma omp parallel for private(i) // shared(minvertices, arcs, vertices)
	for (i = 0; i < N; ++i) {
		minvertices[i].ver = arcs[ver][i];
		minvertices[i].backver = startver;
		vertices[i] = false;
	}

	vertices[ver] = true;

	for (i = 0, minarc = INT_MAX; i < N; ++i)
		if ((vertices[i] == false) && (minarc > minvertices[i].ver))
{
			minarc = minvertices[i].ver;
			minver = i;
		}

	vertices[minver] = true;

	for (i = 2; i < N; ++i) {
#pragma omp parallel for private(j) firstprivate(minarc, minver)
shared(minvertices, arcs, vertices)
		for (j = 0; j < N; ++j) {
			if ((vertices[j] == 0) && (minvertices[j].ver > minarc
+ arcs[minver][j])) {
				minvertices[j].ver = minarc + arcs[minver][j];
				minvertices[j].backver = minver + 1;
			}
		}

		for (j = 0, minarc = INT_MAX; j < N; ++j)
			if ((vertices[j] == false) && (minarc >
minvertices[j].ver)) {
			minarc = minvertices[j].ver;
				minver = j;
			}
		vertices[minver] = true;
```

```
    }

    t2 = omp_get_wtime();

    cout << endl << "\t\t\t Full RunTime = " << t2 - t1 << endl;
    cout << endl;
    cout <<
"*********************************************************
**********************" << endl;

    for(k = 1; k <= N; ++k) {
        for (minway[0] = k, minwaytemp = 0, i = 1; minwaytemp !=
startver; ++i)
            minwaytemp = minway[i] = minvertices[minway[i - 1] -
1].backver;

            cout << " Way from vertix " << startver << " in vertix " <<
k << ": ";
            for (i = N - 1; i > 0; --i)
                if (minway[i] > 0)
                    cout << minway[i] << " -> ";
              cout << minway[0] << " Weight -> " << minvertices[k -
1].ver << endl;

              for (j = 0; j < N; ++j)
                  minway[j] = 0;
    }

      cout <<
"*********************************************************
**********************" << endl;

      for (minway[0] = finishver, minwaytemp = 0, i = 1;
minwaytemp != startver; ++i)
            minwaytemp = minway[i] = minvertices[minway[i - 1] -
1].backver;

      cout << " Way from vertix " << startver << " in vertix " <<
```

```
finishver << ": ";
          for (i = N - 1; i > 0; --i)
               if (minway[i] > 0)
                  cout << minway[i] << " -> ";
          cout << minway[0] << " Weight -> " << minvertices[finishver
- 1].ver << endl;
          cout <<
"********************************************************
*********************" << endl;

       for (i = 0; i < N; i++)
               delete[] arcs[i];
       delete[] arcs;
       delete[] minvertices;
       delete[] minway;

       //cin.get();
       return 0;
}
```

Нижче наведено результат роботи програми при різних початкових умовах:

```
[student@cluster Dejkstra]$ ./dejkstra.bin 30000 1 3 1
*********************************************************
      Number of vertices = 30000      Number of processor cores = 1
      Start vertix = 3        Finish vertix = 1
*********************************************************
                         End of rendom
*********************************************************

                    Full RunTime = 2.42787

*********************************************************
 Way from vertix 3 in vertix 1: 3 -> 116 -> 1 Weight -> 2
*********************************************************

[student@cluster Dejkstra]$ ./dejkstra.bin 30000 2 3 1
```

****

Number of vertices = 30000   Number of processor cores = 2
Start vertix = 3   Finish vertix = 1

****

End of rendom

****

Full RunTime = 1.90965

****

Way from vertix 3 in vertix 1: 3 -> 116 -> 1 Weight -> 2

****

[student@cluster Dejkstra]$ ./dejkstra.bin 30000 4 3 1

****

Number of vertices = 30000   Number of processor cores = 4
Start vertix = 3   Finish vertix = 1

****

End of rendom

****

Full RunTime = 1.57598

****

Way from vertix 3 in vertix 1: 3 -> 116 -> 1 Weight -> 2

****

Хід виконання лабораторної роботи

1 Вивчити теоретичний розділ лабораторної роботи.

2 Підключитися до кластера. Для цього запустити програму ***PuTTy.exe***. Перевірити налаштування мовного кодування. Повинно бути **UTF-8**. Для підключення до кластера використовуйте IP-адресу 211.211.211.24.

У вікні введіть логін і пароль:

login: *studentX*

password: *studentX*

де *X* – номер курсу. Пароль при введенні на екрані не відображається.

3 Після входу на кластер перейдіть в каталог зі своїм прізвищем:

cd <*прізвище студента*>

4 Створіть всередині каталога ще один, з ім'ям **lab2**.

5 Перейдіть в нього: **cd lab2**

6 Виконайте команду **pbsnodes (qnodes)** для перевірки працездатності кластера.

7 Створіть файл з розширенням **.cpp**.

8 Наберіть в ньому текст програми-прикладу, наведеної вище. Розберіться з алгоритмом програми-прикладу. Вивчіть призначення змінних, директив і функцій.

9 Відкомпілюйте програму-приклад. При компіляції використовуйте ключ **-fopenmp**.

[student@cluster Dejkstra]$ g++ -O2 dejkstra.cpp -o dejkstra.bin –fopenmp

10 Створіть файл скрипта з розширенням **.pbs** для запуску файлу-прикладу на кластері.

11 Поставте завдання в чергу на кластер:

qsub <*ім'я файлу скрипта*>.pbs

12 Перевірте стан черги командою: **qstat**.

13 Перевірте, чи створюються файли звіту і помилок.

14 Перевірте коректність роботи програми-прикладу на малій кількості вершин (від 5 до 7), з виведенням результатів. Для цього необхідно прибрати коментарі у відповідних місцях програми-прикладу.

15 Після перевірки правильності роботи програми-прикладу поверніть коментарі назад.

16 За результатами роботи програми-прикладу заповніть таблицю 2.3, вказавши у відповідних полях повний час виконання програми-прикладу при різних параметрах кількості ядер і кількості вершин повнозв'язного неорієнтовного графа ***G(V, E)***:

Таблиця 2.3

Кіль. ядер, шт.	Кількість вершин повнозв'язного неорієнтовного графа ***G(V, E)***, шт.			
	1000	10000	30000	50000
1				
2				
3				
6				

Контрольні питання

1 Алгоритм Дейкстри.

2 Дайте визначення стягненого дерева.

3 Що таке локальний екстремум?

4 Який стандарт розпаралелювання використовується в програмі-прикладі?

5 Опишіть класичну реалізацію алгоритму Дейкстри.

6 Опишіть реалізацію алгоритму Дейкстри з використанням стягненого дерева.

7 Яка реалізація алгоритму Дейкстри використовується в програмі-прикладі?

8 Як реалізовано розпаралелювання в програмі-прикладі?

9 Порівняйте методи реалізації алгоритму Дейкстри.

10 Поясніть результати, отримані в таблиці 2.3.

11 Що таке директива препроцесора?

12 Який основний показник при виконанні програми-прикладу ми використовуємо і як він реалізований?

Індивідуальні завдання

Створіть програму розв’язання завдання знаходження найкоротшого шляху в повнозв'язному неорієнтованому графі, використовуючи класичний метод реалізації алгоритму Дейкстри. Виконайте програму з такими вхідними параметрами:

варіант 1:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 10;

- початкова вершина – 2;
- кінцева вершина — 9;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 10000;
- початкова вершина – 2;
- кінцева вершина — 9000;

варіант 2:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 11;
- початкова вершина – 3;
- кінцева вершина — 11;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 11000;
- початкова вершина – 3;
- кінцева вершина — 10000;

варіант 3:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 12;
- початкова вершина – 5;
- кінцева вершина — 7;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 12000;
- початкова вершина – 5;
- кінцева вершина — 7000;

варіант 4:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 13;
- початкова вершина – 1;
- кінцева вершина — 5;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 13000;
- початкова вершина – 1;
- кінцева вершина — 5000;

варіант 5:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 11;
- початкова вершина – 2;
- кінцева вершина — 10;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 11000;
- початкова вершина – 2;
- кінцева вершина — 10000;

варіант 6:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 12;
- початкова вершина – 4;
- кінцева вершина — 8;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 12000;
- початкова вершина – 4;
- кінцева вершина — 8000;

варіант 7:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 13;
- початкова вершина – 1;
- кінцева вершина — 9;

без виведення матриці вагових коефіцієнтів, але з виведенням

найкоротшого шляху:
- кількість вершин графа – 12000;
- початкова вершина – 1;
- кінцева вершина — 9000;

варіант 8:
з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:
- кількість вершин графа — 11;
- початкова вершина — 3;
- кінцева вершина — 10;
без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:
- кількість вершин графа — 11000;
- початкова вершина — 3;
- кінцева вершина — 10000;

варіант 9:
з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:
- кількість вершин графа – 10;
- початкова вершина – 5 ;
- кінцева вершина — 7;
без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:
- кількість вершин графа – 10000;
- початкова вершина – 5;
- кінцева вершина — 7000;

варіант 10:
з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:
- кількість вершин графа – 13;
- початкова вершина – 1;
- кінцева вершина — 8;
без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:
- кількість вершин графа – 13000;

- початкова вершина – 1;
- кінцева вершина — 8000.

Перевірити роботу програми на кількості ядер від 1 до 4. Результат оформити у вигляді таблиці, аналогічної таблиці 2.3.

ЛАБОРАТОРНА РОБОТА 3
Алгоритм Форда. Знаходження найкоротшого шляху від однієї з вершин графа до решти з використанням стандарту OpenMP

Мета: ознайомитися з алгоритмом Форда. Створити програму, що реалізовує знаходження найкоротшого шляху від однієї з вершин графа до решти, використовуючи стандарт OpenMP.

Теорія

Заданий простий неорієнтовний зважений граф ***G(V, E)*** (рисунок 3.1), що складається з **V** вершин і **E** дуг. Кожна дуга **(u, v)** має невід'ємну вагу **w (u, v), w (u, v) ≥ 0** (таблиця 3.1). Потрібно знайти найкоротші шляхи з деякою стартовою вершиною ***s*** до решти вершин графа ***G (V, E)***. Графічним еквівалентом простору можливих станів системи є стягнуте дерево ***D(i)*** всіх шляхів графа ***G(V, E)*** яке можна будувати від довільної вершини ***i***. $\overset{*}{d}(\mu^{r}_{sj})$ в графі ***D(s = i)***, від вершини ***s*** до вершини ***j*** рангу ***r***, локальних екстремумів рангу ***r***, а довжину найкоротшого шляху до вершини ***j*** на множині $\min_{r} \overset{*}{d}(\mu^{r}_{sj})$ локальних екстремумів на горизонтальній лінійці графа ***D (s = i)***. Найкоротший шлях $\overset{**}{d}(\mu^{r}_{sj})$ у графі ***D(s = i)*** від вершини ***s*** до деякої вершини ***j*** на горизонтальній лінійці графа назвемо глобальним екстремумом. Тоді задача визначення найкоротших шляхів еквівалентна визначенням глобального екстремуму на кожній горизонтальній лінійці в просторі можливих станів системи, що задається графом ***D(s = i)***. З вершини ***s*** формуємо множину шляхів наступного рангу до вершин ***j*** і визначаємо в кожній

множині локальні екстремуми. На основі локальних екстремумів на ярусі поточного рангу формуємо множину шляхів наступного рангу ***i*** в них знову виділяємо локальні екстремуми ***j***. Порівнюємо локальні екстремуми попереднього ***r - 1*** і поточного ***r*** рангів. Якщо всі значення поточного рангу більше попереднього, то подальші обчислення можна припиняти, оскільки найкоротші глобальні екстремуми на горизонтальних ярусах в графі ***D(s = i)*** вже визначені і алгоритм закінчує роботу, інакше формуємо шляхи наступного рангу r + 1 і продовжуємо обчислення. Перевіряємо, чи пройдені всі вершини ***v ϵ V***: ***r = n - 1***. Якщо ні, то повторюємо дії з формування нових шляхів, якщо так, то всі вершини ***j*** пройдені і алгоритм закінчує роботу (таблиця 3.2).

Вихідний граф ***G(V, E)*** зручно подавати в пам'яті обчислювальної системи у вигляді двовимірного масиву **arcs**, індекси якого є номери вершин вихідного графа, в які було здійснено перехід. Значення в полі матриці являють собою вагові коефіцієнти дуг, що з'єднують вершини і відповідні їх положенню в графі ***G(V, E)***. Для подання графа мовою ***C++*** використовується масив, кожен елемент якого є даними типу ***int***. Мінлива **startver** позначає стартову вершину, від якої шукається відстань до решти вершин ***v ϵ V***. Масив **minway** використовується для зберігання глобальних найкоротших шляхів від стартової вершини **startver** до решти ***v ϵ V*** вершин. Масив **minweight** містить ваги глобальних найкоротших шляхів **w(u, v)**. Також створюються масиви формування попередніх і наступних найкоротших шляхів **currentway**, **nextway,** що містять всі локальні найкоротші шляхи для всіх горизонтальних ярусів і масиви **currentweight i nextweight** для зберігання ваг цих шляхів. При формуванні наступних шляхів і обчисленні їх ваг використовується масив тимчасових шляхів **tempway** і тимчасова змінна їх ваги **tempweight**. У разі, якщо неможливо сформувати жоден локальний шлях на горизонтальному ярусі для якоїсь із вершин, така вершина повинна бути виключена з подальшого аналізу. Глобальний екстремум для неї вже визначено на попередніх рангах. У процесі роботи алгоритму Форда підтримується множина ***U ϵ V***, що складається з вершин ***v ϵ V*** графа ***G(V, E)***, для яких найкоротшу відстань вже знайдено.

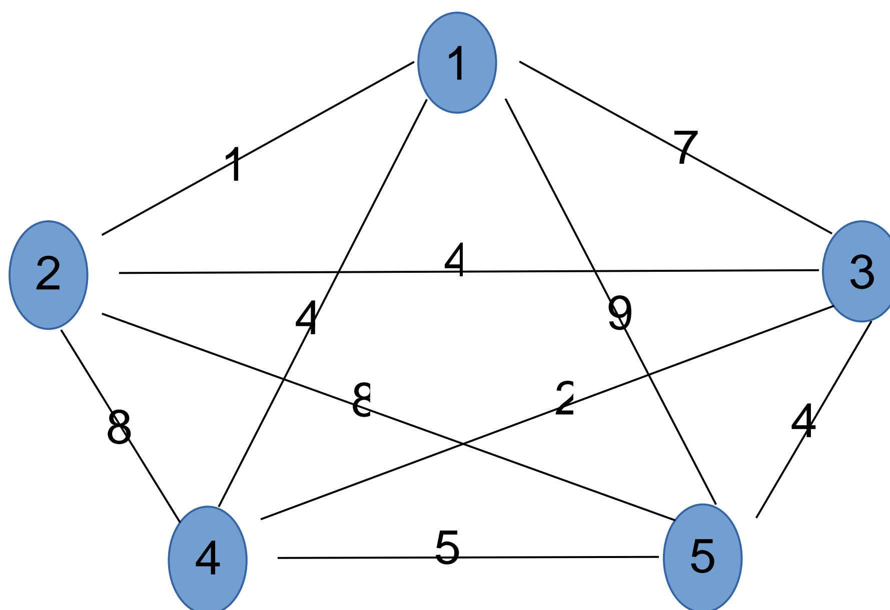

Рисунок 3.1 – Повнозв'язний граф

Таблиця 3.1 – Матриця вагових коефіцієнтів

Vertices	1	2	3	4	5
1	0	1	7	4	9
2	1	0	4	8	8
3	7	4	0	2	4
4	4	8	2	0	5
5	9	8	4	5	0

У підсумку в ньому буде сформована і збережена інформація про знайдені найкоротші відстані від стартової вершини **startver** до вершини ***v ϵ V*** графа ***G(V, E)***. У наведеній нижче реалізації дані зчитуються і виводяться в консоль.

Вхідними даними для роботи програми є:

- кількість вершин;
- кількість процесорних ядер;
- номер стартової вершини;
- номер вершини, до якої розраховується найкоротший шлях.

Таблиця 3.2 – Реалізація алгоритму Форда

Vertices					Rezult
3	3 – 1 (7)*	3 – 2 – 1 (5)* 3 – 4 – 1 (6) 3 – 5 – 1 (13)	3 – 5 – 4 – 1 (13)* 3 – 4 – 5 – 1 (16)	3 – 4 – 5 – 2 – 1 (16)*	3 – 2 – 1 (5)
	3 – 2 (4)*	3 – 1 – 2 (8)* 3 – 4 – 2 (10) 3 – 5 – 2 (12)	3 – 5 – 4 – 2 (17) 3 – 4 – 5 – 2 (15)*	3 – 5 – 4 – 1 – 2 (14)*	3 – 2 (4)*
	3 – 4 (2)*	3 – 1 – 4 (11) 3 – 2 – 4 (12) 3 – 5 – 4 (9)*	3 – 2 – 1 – 4 (9)* 3 – 1 – 2 – 4 (16)	3 – 2 – 1 – 5 – 4 (19)*	3 – 4 (2)*
	3 – 5 (4)*	3 – 1 – 5 (16) 3 – 2 – 5 (12) 3 – 4 – 5 (7)*	3 – 2 – 1 – 5 (14)* 3 – 1 – 2 – 5 (16)	3 – 2 – 1 – 4 – 5 (14)*	3 – 5 (4)*
0	1	2	3	4	Rank

Вихідні дані:

- відображаються вхідні дані;
- повний час виконання розрахункової частини програми;
- виведення найкоротшого шляху з стартовою вершиною **startver** в решту вершини графа $G(V, E)$.
- виведення найкоротшого шляху з стартової вершини **startver** в задану **finishver**.

Нижче подано текст програми реалізації алгоритму Форда методом стягненого дерева:

```
#include <iostream>
#include <cstdlib>
#include <iomanip>
#include <climits>
#include <ctime>
#include <omp.h>

using namespace std;

int main(int argc, char **argv)
{
	if (argc != 5)
		cout << "!!!ERROR!!! Example of command:
./FordListrovoy.bin <number of vertices> <number of processor
cores> <start vertix> <finish vertix>!!!ERROR!!!" << endl,
/*cin.get(),*/ exit(1);
	cout <<
"*********************************************************
**********************" << endl;
	const int N = atoi(*++argv); // Enter number of vertices
	const int threads = atoi(*++argv); // Enter the number of
processor core
	const int startver = atoi(*++argv); // Enter start vertix
	const int finishver = atoi(*++argv); // Enter finish vertix
	cout << "\tNumber of vertices = " << N << "\tNumber of
processor cores = " << threads << endl;
	cout << "\tStart vertix = " << startver << "\tFinish vertix = " <<
finishver << endl;
	cout <<
"*********************************************************
**********************" << endl;

	int i, j, ver, rank;
	bool countend;
	double t1, t2;

	omp_set_num_threads(threads);
```

```
int **arcs = new int *[N];
for (i = 0; i< N; i++)
        arcs[i] = new int[N];

int **minway = new int *[N];
for (i = 0; i< N; i++)
      minway[i] = new int[N];

int *minweight = new int[N];

int **currentway = new int *[N];
for (i = 0; i< N; i++)
      currentway[i] = new int[N];

int *currentweight = new int[N];

int **nextway = new int *[N];
for (i = 0; i< N; i++)
      nextway[i] = new int[N];

int *nextweight = new int[N];

int tempway[N];
int tempweight;

for (i = 0; i < N; ++i) {
      currentweight[i] = nextweight[i] = minweight[i] =
tempway[i] = 0;
      for (j = 0; j < N; ++j)
            currentway[i][j] = nextway[i][j] = minway[i][j] = 0;
}

//srand(time(NULL));
for (i = 0; i < N; ++i) {
        arcs[i][i] = 0;
        for (j = i + 1; j < N; ++j) {
              arcs[i][j] = rand() % 10;
              arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
```

```
                arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
                arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
                arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
                arcs[i][j] == 0 ? arcs[i][j] = rand() % 10 : arcs[i][j];
                arcs[j][i] = arcs[i][j];
            }
        }
        cout << "\t\t\t\tEnd of rendom" << endl;
        cout <<
"*******************************************************
*********************" << endl;
/*
        cout << "       ";
            for (i = 0; i < N; ++i)
                cout << setw(3) << i + 1;
            cout << endl;
            cout <<
"*******************************************************
*********************" << endl;
            for (i = 0; i < N; ++i) {
                cout << setw(5) << i + 1 << " * ";
                for (j = 0; j < N; ++j)
                    cout << setw(3) << arcs[i][j];
                cout << endl;
            }
            cout <<
"*******************************************************
*********************" << endl;
*/
        t1 = omp_get_wtime();
        int sver = startver - 1;

#pragma omp parallel for private (i)
   for (i = 0; i < N; ++i) {
        currentway[i][0] = startver;
        currentway[i][1] = i + 1;
        currentweight[i] = arcs[sver][i];
        minway[i][0] = currentway[i][0];
```

```
        minway[i][1] = currentway[i][1];
        minweight[i] = currentweight[i];
    }

    for (rank = 2; rank < N; ++rank) {
        // cout << " rank = " << rank << endl;
#pragma omp parallel for private (i, j, ver, tempweight, tempway)
        for (ver = 1; ver <= N; ++ver) {
                nextweight[ver - 1] = INT_MAX;
                if (ver != startver) {
                        for (i = 0; i < N; ++i) {
                                if (i + 1 != startver) {
                                    for (j = 0; j < rank; ++j)
                                        if (!(currentway[i][j] == ver ||
currentway[i][0] == -1)) {
                                            for (j = 0; j < rank; ++j)
                                                tempway[j] = currentway[i][j];
                                            tempway[rank] = ver;
                                            tempweight = 0;
                                            for (j = 0; j < rank; ++j)
                                                tempweight += arcs[tempway[j] -
1][tempway[j + 1] - 1];

                                        /*  cout << "ver: " << ver << "
tempweight = " << tempweight << " tempway";
                                          for (j = 0; j <= rank; ++j)
                                              cout << " -> " << tempway[j];
                                          cout << endl; */

                                            if (tempweight < nextweight[ver - 1])
{
                                                nextweight[ver - 1] = tempweight;
                                                for (j = 0; j <= rank; ++j)
                                                    nextway[ver - 1][j] =
tempway[j];
                                            }
                                        }
                                }
```

```
                    }
                }
            }

#pragma omp parallel for private (i)
                for (i = 0; i < N; ++i)
              if(nextweight[i] == INT_MAX && (i + 1 != startver))
                  nextway[i][0] = -1;


        /* cout << " nextweight:";
         for (i = 0; i < N; ++i)
              cout << " -> " << nextweight[i];
              cout << endl; */

                countend = 0;
            for (i = 0; i < N; ++i)
              if(currentweight[i] > nextweight[i])
                  countend = 1;

              if (countend) {
#pragma omp parallel for private (i,j)
              for (i = 0; i < N; ++i) {
                 currentweight[i] = nextweight[i];
                for (j = 0; j <= rank; ++j)
                    currentway[i][j] = nextway[i][j];
                if (currentweight[i] < minweight[i]) {
                  minweight[i] = currentweight[i];
                  for (j = 0; j <= rank; ++j)
                      minway[i][j] = currentway[i][j];
                }
              }
            }
            else
              break;
        }

        t2 = omp_get_wtime();
```

```
	cout << endl << "\t\t\t Full RunTime = " << t2 - t1 << endl;
	cout << endl;
	cout <<
"********************************************************
*********************" << endl;

	for (i = 0; i < N; ++i) {
		cout << " Way from vertix " << startver << " in vertix " <<
i + 1 << ": ";
		for (j = 0; j < N; ++j) {
			if (minway[i][j])
			    cout << minway[i][j] << " -> ";
		}
		cout << "\b\b\bWeight -> " << minweight[i] << endl;
	}
	cout <<
"********************************************************
*********************" << endl;

	cout << " Way from vertix " << startver << " in vertix " <<
finishver << ": ";
			for (j = 0; j < N; ++j) {
				if (minway[finishver - 1][j])
				    cout << minway[finishver - 1][j] << " -> ";
			}
			cout << "\b\b\bWeight -> " << minweight[finishver -
1] << endl;

	cout <<
"********************************************************
*********************" << endl;

	for (i = 0; i < N; i++)
		delete[] arcs[i];
	delete[] arcs;

	for (i = 0; i < N; i++)
		delete[] minway[i];
```

```
	delete[] minway;

	for (i = 0; i < N; i++)
	    delete[] currentway[i];
	delete[] currentway;

	for (i = 0; i < N; i++)
	    delete[] nextway[i];
	delete[] nextway;

	delete[] minweight;
	delete[] currentweight;
	delete[] nextweight;

	//cin.get();
	return 0;
}
```

Нижче наведено результат роботи програми при різних початкових умовах:

```
[student@cluster Ford]$ ./ford.bin 30000 1 3 1
*******************************************************
    Number of vertices = 30000     Number of processor cores = 1
    Start vertix = 3       Finish vertix = 1
*******************************************************
                    End of rendom
*******************************************************

                 Full RunTime = 174.063

*******************************************************
 Way from vertix 3 in vertix 1: 3 -> 116 -> 1 Weight -> 2
*******************************************************
[student@cluster Ford]$ ./ford.bin 30000 2 3 1
*******************************************************
    Number of vertices = 30000     Number of processor cores = 2
    Start vertix = 3       Finish vertix = 1
```

```
**********************************************************
                 End of rendom
**********************************************************

             Full RunTime = 101.208

**********************************************************
 Way from vertix 3 in vertix 1: 3 -> 116 -> 1 Weight -> 2
**********************************************************

[student@cluster Ford]$ ./ford.bin 30000 4 3 1
**********************************************************
    Number of vertices = 30000     Number of processor cores = 4
    Start vertix = 3       Finish vertix = 1
**********************************************************
                 End of rendom
**********************************************************

             Full RunTime = 55.1371

**********************************************************
 Way from vertix 3 in vertix 1: 3 -> 116 -> 1 Weight -> 2
**********************************************************
```

Хід виконання лабораторної роботи

1 Вивчіть теоретичний розділ лабораторної роботи.

2 Підключіться до кластера. Для цього запустіть програму ***PuTTy.exe***. Перевірте налаштування мовного кодування. Має бути **UTF-8**. Для підключення до кластера використовуйте IP-адресу 211.211.211.24.

У вікні введіть логін і пароль:

login: *studentX*

password: *studentX*

де *X* – номер курсу. Пароль при введенні на екрані не відображається.

3 Після входу на кластер перейдіть в каталог зі своїм прізвищем:

cd *<прізвище студента>*

4 Створіть всередині каталогу ще один з ім'ям **lab3**.

5 Перейдіть в нього: **cd lab3**

6 Виконайте команду **pbsnodes (qnodes)** для перевірки працездатності кластера.

7 Створіть файл з розширенням **.cpp**.

8 Наберіть в ньому текст програми-прикладу, наведеної вище. Розберіться з алгоритмом програми-прикладу. Вивчіть призначення змінних, директив і функцій

9 Відкомпілюйте програму-приклад. При компіляції використовуйте ключ **-fopenmp**.

[student@cluster Ford]$ g++ -O2 ford.cpp -o ford.bin -fopenmp

10 Створіть файл скрипта з розширенням **.pbs** для запуску файлу-прикладу на кластері.

11 Поставте завдання в чергу на кластер:

qsub <*ім'я файлу скрипта*> .pbs.

12 Перевірте стан черги командою: **qstat**.

13 Перевірте, чи створюються файли звіту і помилок.

14 Перевірте коректність роботи програми-прикладу на малій кількості вершин (від 5 до 7) з виведенням результатів. Для цього необхідно прибрати коментарі у відповідних місцях програми-прикладу.

15 Після перевірки правильності роботи програми-прикладу поверніть коментарі назад.

16 За результатами роботи програми-прикладу заповніть таблицю 3.3, вказавши у відповідних полях повний час виконання програми-прикладу при різних параметрах кількості ядер і кількості вершин повнозв'язного неорієнтовного графа ***G(V, E)***:

Таблиця 3.3

Кіль. ядер, шт.	Кількість вершин повнозв'язного неорієнтовного графа ***G(V, E)***, шт.			
	1000	10000	30000	50000
1				
2				
3				
6				

Контрольні питання

1 Алгоритм Форда.

2 Дайте визначення стягненого дерева.

3 Що таке глобальний екстремум?

4 Який стандарт розпаралелювання використовується в програмі-прикладі?

5 Опишіть класичну реалізацію алгоритму Форда.

6 Опишіть реалізацію алгоритму Форда з використанням стягненого дерева.

7 Яка реалізація алгоритму Форда використовується в програмі-прикладі?

8 Як реалізовано розпаралелювання в програмі-прикладі?

9 Порівняйте методи реалізації алгоритму Форда.

10 Поясніть результати отримані в таблиці 3.3.

11 Що таке директива препроцесора?

12 Який основний показник при виконанні програми-прикладу ми використовуємо і як він реалізований?

Індивідуальні завдання

Створіть програму розв’язання завдання знаходження найкоротшого шляху в повнозв'язному неорієнтованому графі, використовуючи класичний метод реалізації алгоритму Форда. Виконайте програму з такими вхідними параметрами:

варіант 1:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 10;
- початкова вершина – 2;
- кінцева вершина — 9;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 10000;
- початкова вершина – 2;
- кінцева вершина — 9000;

варіант 2:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 11;
- початкова вершина – 3;
- кінцева вершина — 11;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 11000;
- початкова вершина – 3;
- кінцева вершина — 10000;

варіант 3:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 12;
- початкова вершина – 5; - кінцева вершина — 7;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 12000;
- початкова вершина – 5;
- кінцева вершина — 7000;

варіант 4:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 13;
- початкова вершина – 1;
- кінцева вершина — 5;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 13000;
- початкова вершина – 1;
- кінцева вершина — 5000;

варіант 5:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 11;
- початкова вершина – 2;
- кінцева вершина — 10;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:
- кількість вершин графа – 11000;
- початкова вершина – 2;
- кінцева вершина — 10000;

варіант 6:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:
- кількість вершин графа – 12;
- початкова вершина – 4;
- кінцева вершина — 8;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:
- кількість вершин графа – 12000;
- початкова вершина – 4;
- кінцева вершина — 8000;

варіант 7:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:
- кількість вершин графа – 13;
- початкова вершина – 1;
- кінцева вершина — 9;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:
- кількість вершин графа – 12000;
- початкова вершина – 1;
- кінцева вершина — 9000;

варіант 8:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:
- кількість вершин графа – 11;

- початкова вершина – 3;
- кінцева вершина — 10;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 11000;
- початкова вершина – 3;
- кінцева вершина — 10000;

варіант 9:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 10;
- початкова вершина – 5 ;
- кінцева вершина — 7;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 10000;
- початкова вершина – 5;
- кінцева вершина — 7000;

варіант 10:

з виведенням матриці вагових коефіцієнтів і найкоротшого шляху:

- кількість вершин графа – 13;
- початкова вершина – 1;
- кінцева вершина — 8;

без виведення матриці вагових коефіцієнтів, але з виведенням найкоротшого шляху:

- кількість вершин графа – 13000;
- початкова вершина – 1;
- кінцева вершина — 8000.

Перевірити роботу програми на кількості ядер від 1 до 4. Результат оформити у вигляді таблиці, аналогічної таблиці 3.3.

СПИСОК ЛІТЕРАТУРИ

1 OpenMP Architecture Review Board [Электронный ресурс]. – Режим доступа : http://www.openmp.org/

2 The Community of OpenMP Users, Researchers, Tool Developers and Providers [Электронный ресурс]. – Режим доступа : http://www.compunity.org/.

3 OpenMP Application Program Interface Version 4.5 November 2015 [Электронный ресурс]. – Режим доступа : http://www.openmp.org/mp-documents/spec45.pdf

4 Воеводин, В. В. Параллельные вычисления [Текст] / В. В. Воеводин, Вл. В. Воеводин. – СПб.: БХВ-Петербург, 2002. – 608 с.

5 Gebauer, H. Finding and enumerating hamilton cycles in 4-regular graphs [Text] / H. Gebauer // Theoretical Computer Science. – 2011. – Vol. 412, Issue 35. – P. 4579–4591. doi: 10.1016/j.tcs.2011.04.038

6 Евстигнеев, В. А. Применение теории графов в программировании. [Текст] / В. А. Евстигнеев; под. ред. А. П. Ершова. – М. : Наука. Главная редакция физико-математической литературы, 1985. – 352 с.