

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ „ЛЬВІВСЬКА ПОЛІТЕХНІКА”

**МЕТОДИЧНІ ВКАЗІВКИ  
ДЛЯ ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ  
З ДИСЦИПЛІНИ:**

**ТЕХНОЛОГІЇ РОЗПОДІЛЕНИХ СИСТЕМ ТА ПАРАЛЕЛЬНИХ  
ОБЧИСЛЕНЬ**

Львів – 2013

*Пропоновані методичні вказівки призначені для виконання лабораторних робіт з дисципліни Технології розподілених систем та паралельних обчислень. Вказані завдання повинні сформувати в студентів навички з основних принципів побудови розподілених систем та здійснення паралельних обчислень.*

## **ВИМОГИ ДО ОФОРМЛЕННЯ ЗВІТІВ ПРО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ**

Звіти про виконання лабораторних робіт оформляються на зшитих аркушах формату А4.

Після захисту лабораторних робіт звіти здаються для зберігання на кафедрі.

Кожен звіт повинен містити такі розділи:

- Номер та **назва роботи;**
- **Мета виконання лабораторної роботи;**
- **Індивідуальне завдання** з детальним формулюванням розв'язуваної задачі, використовувані (**власні**) теоретичні відомості;
- **Відповіді на контрольні запитання**
- Текст програми реалізації.
- Результати кроків виконання алгоритму.
- Результати роботи програми.
- **Висновки.** Вказується призначення роботи, можливі варіанти вдосконалення та які знання отримано в ході виконання роботи.

Звіт повинен бути написаний українською мовою, акуратно та грамотно, з дотриманням правил оформлення ділової документації. Назви розділів звіту візуально виділити розміром, підкресленням.

## **Лабораторна робота №1**

**Тема: Освоєння технологій паралельного програмування засобами OpenMP.**

**Мета роботи:** ознайомитись та навчитись створювати програми у відповідності із специфікацією OPENMP.

### **ТЕОРЕТИЧНІ ВІДОМОСТІ**

У обчислювальних системах з розподіленою пам'яттю процесори працюють незалежно один від одного. Для організації паралельних обчислень в таких умовах необхідно мати можливість розподіляти обчислювальне навантаження і організувати інформаційну взаємодію (передачу даних) між процесорами. Одним з способів взаємодії між паралельними процесами є передача повідомлень між ними, що відображено в самій назві технології MPI (message passing interface) – інтерфейс передачі повідомлень.

Для розподілу обчислень між процесорами необхідно проаналізувати алгоритм розв'язку задачі, виділити інформаційно незалежні фрагменти обчислень, виконати їх програмну реалізацію і розмістити отримані частини програми на різних процесорах. В MPI використовуються простіший підхід - для виконання завдання розробляється одна програма, яка запускається одночасно на виконання на всіх наявних процесорах. При цьому для того, щоб уникнути ідентичності обчислень на різних процесорах, можна, по-перше, підставляти різні дані для програми на різних процесорах, а по-друге, використовувати наявні в MPI засоби для ідентифікації процесора, на якому виконується програма (тим самим надається можливість організувати відмінності в обчисленнях залежно від використовуваного програмою процесора).

Інтерфейс MPI підтримує реалізацію програм для MIMD систем (Multiple Instructions Multiple Data), проте відлагодження таких програм є не тривіальною задачею. Тому на практиці для написання програм в більшості випадків застосовується SPMD (single program multiple processes) модель паралельного програмування - "одна програма безліч процесів".

### **Загальні відомості про MPI**

MPI - це стандарт, якому повинні задовольняти засоби організації передачі повідомлень. Крім того MPI - це програмні засоби, які забезпечують можливість передачі повідомлень і при цьому відповідають всім вимогам стандарту MPI. Згідно стандарту, ці програмні засоби повинні бути організовані у вигляді бібліотек програмних функцій (бібліотеки MPI) і повинні бути доступні для найширше використовуваних алгоритмічних мов C і Fortran. Подібну "подвійність" MPI слід враховувати при використанні термінології. Як правило, аббревіатура MPI застосовується при згадці стандарту, а поєднання "бібліотека MPI" указує на ту або іншу програмну реалізацію стандарту. Оскільки достатньо часто

скорочено позначення MPI використовується і для бібліотек MPI, тому для правильної інтерпретації терміну, слід враховувати контекст.

**Поняття паралельної програми.** Під паралельною програмою в MPI розуміється множина одночасно виконуваних процесів. Процеси можуть виконуватися як на різних процесорах, так і на одному процесорі можуть виконуватися і декілька процесів (в цьому випадку їх виконання здійснюється в режимі розділення часу). У граничному випадку для виконання паралельної програми може використовуватися один процесор - як правило, такий спосіб застосовується для початкової перевірки правильності паралельної програми.

Кожен процес паралельної програми породжується на основі копії одного і того ж програмного коду (модель SPMP). Даний програмний код, представлений у вигляді виконуваної програми, повинен бути доступний у момент запуску паралельної програми на всіх використовуваних процесорах. Початковий програмний код для виконуваної програми розробляється на алгоритмічних мовах C або Fortran із застосуванням тієї або іншої реалізації бібліотеки MPI.

Кількість процесів і використовуваних процесорів визначається у момент запуску паралельної програми засобами середовища виконання MPI-програм і в ході обчислень не може змінюватися без застосування спеціальних засобів динамічного породження процесів і управління ними, згідно з стандартом MPI версії 2.0. Всі процеси програми послідовно перенумеровані від 0 до  $p-1$ , де  $p$  є загальна кількість процесів. Номер процесу іменується рангом процесу.

**Операції передачі даних.** Основу MPI складають операції передачі повідомлень. Серед передбачених у складі MPI функцій розрізняються парні (point-to-point) операції між двома процесами і колективні (collective) комунікаційні дії для одночасної взаємодії декількох процесів.

Для виконання парних операцій можуть використовуватися різні режими передачі (синхронний, блокуючий і ін.). Повний розгляд можливих режимів передачі розглядається нижче.

**Поняття комунікаторів.** Процеси паралельної програми об'єднуються в групи. Іншим важливим поняттям MPI, що описує набір процесів, є поняття комунікатора. Під комунікатором в MPI розуміється спеціально створений службовий об'єкт, який об'єднує групу процесів і ряд додаткових параметрів (контекст), використовуваних при виконанні операцій передачі даних.

Парні операції передачі даних можуть бути виконані між будь-якими процесами одного і того ж комунікатора, а в колективних операціях беруть участь всі процеси комунікатора. Як результат, вказання використовуваного комунікатора є обов'язковим для операцій передачі даних в MPI.

Логічна топологія ліній зв'язку між процесами має структуру повного графа (незалежно від наявності реальних фізичних каналів зв'язку між процесорами).

У MPI є можливість представлення множини процесів у вигляді ґраток довільної розмірності. Крім того, в MPI є засоби і для формування логічних (віртуальних) топологій будь-якого необхідного типу.

Під час обчислень можуть створюватися нові і видалятися існуючі групи процесів і комунікаторів. Один і той же процес може належати різним групам і комунікаторам. Всі наявні в паралельній програмі процеси входять до складу конструйованого за замовчуванням комунікатора з ідентифікатором `MPI_COMM_WORLD`.

У версії 2.0 стандарту з'явилася можливість створювати глобальні комунікатори (intercommunicator), об'єднуючи в одну структуру пару груп при необхідності виконання колективних операцій між процесами з різних груп.

**Типи даних.** При виконанні операцій передачі повідомлень для вказівки передаваних або отримуваних даних у функціях MPI необхідно вказувати тип даних, що пересилаються. MPI містить великий набір базових типів даних, багато в чому співпадаючих з типами даних в алгоритмічних мовах C і Fortran. Крім того, в MPI є можливості створення нових похідних типів даних для точнішого і коротшого опису вмісту повідомлень, що пересилаються.

## **Введення в розробку паралельних програм з використанням MPI**

Мінімально необхідний набір функцій MPI, достатній для розробки порівняно простих паралельних програм.

**Ініціалізація і завершення MPI-програм.** Першою функцією MPI, що викликається, повинна бути функція:

*int MPI\_Init(int \*argc, char \*\*\*argv), де*

*argc* - вказівник на кількість параметрів командного рядка;

*argv* - параметри командного рядка.

яка використовується для ініціалізації середовища виконання MPI-програми. Параметрами функції є кількість аргументів в командному рядку і адреса вказівника на масив параметрів командного рядка.

Останньою функцією MPI, що викликається, обов'язково повинна бути функція:

*int MPI\_Finalize(void).*

Структура паралельної програми з використанням MPI повинна мати такий вигляд:

```
#include "mpi.h"
int main(int argc, char *argv[])
{
    <програмний код без використання функцій MPI>
    MPI_Init(&argc, &argv);
    <програмний код з використанням функцій MPI>
    MPI_Finalize();
    <програмний код без використання функцій MPI>
    return 0;
}
```

Варто зазначити:

1. Файл *mpi.h* містить визначення іменованих констант, прототипів функцій і типів даних бібліотеки MPI.
2. Функції *MPI\_Init* і *MPI\_Finalize* є обов'язковими і повинні бути виконані (і лише один раз) кожним процесом паралельної програми.
3. Перед викликом *MPI\_Init* може бути використана функція *MPI\_Initialized* для визначення того, чи був раніше виконаний виклик *MPI\_Init*, а після виклику *MPI\_Finalize* - *MPI\_Finalized* аналогічного призначення.

Розглянуті приклади функцій дають представлення синтаксису іменування функцій в MPI. Імені функції передують префікс MPI; далі одне або декілька слів назви; перше слово в імені функції починається із заголовного символу; слова розділяються знаком підкреслення. Назви функцій MPI, як правило, пояснюють призначення виконуваних функцією дій.

**Визначення кількості і рангу процесів.** Визначення кількості процесів у виконуваний паралельній програмі здійснюється за допомогою функції:

*int MPI\_Comm\_size(MPI\_Comm comm, int \*size),* де

*comm* - комунікатор, розмір якого визначається;

*size* - визначена кількість процесів в комунікаторі.

Для визначення рангу процесу використовується функція:

*int MPI\_Comm\_Rank(MPI\_Comm comm, int \*rank),* де

*comm* - комунікатор, в якому визначається ранг процесу;

*rank* - ранг процесу в комунікаторі.

Як правило, виклик функцій *Mpi\_comm\_size* і *Mpi\_comm\_rank* виконується відразу після *Mpi\_init* для отримання загальної кількості процесів і рангу поточного процесу:

```
#include "mpi.h"

int main(int argc, char *argv[])
{
    int ProcNum, ProcRank;

    <програмний код без використання функцій MPI>
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    <програмний код з використанням функцій MPI>
    MPI_Finalize();

    <програмний код без використання функцій MPI>
    return 0;
}
```

Варто зазначити:

1. Комунікатор *MPI\_COMM\_WORLD*, як наголошувалося раніше, створюється за замовчуванням і представляє всі процеси виконуваної паралельної програми.
2. Ранг, що отримується за допомогою функції *MPI\_Comm\_rank*, є рангом процесу, що виконав виклик цієї функції, тобто змінна *ProcRank* прийме різні значення у різних процесів.

**2.3. Передача повідомлень.** Для передачі повідомлення процес-відправник повинен виконати функцію:

```
int MPI_Send(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm
comm),
```

де, *buf* - адреса буфера пам'яті, в якому розташовуються дані повідомлення, що відправляється;

*count* - кількість елементів даних в повідомленні;

*type* - тип елементів даних повідомлення, що пересилається;

*dest* - ранг процесу, якому відправляється повідомлення;

*tag* - значення-тег, що використовується для ідентифікації повідомлення;

*comm* - комунікатор, в рамках якого виконується передача даних.

Для вказання типу даних, що пересилаються, в MPI є ряд базових типів, повний список яких наведений в таблиці

Тип даних MPI	Тип даних C
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	Double
MPI_FLOAT	Float
MPI_INT	Int
MPI_LONG	Long
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_SHORT	Short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

Варто зазначити:

1. Повідомлення, що відправляється, визначається шляхом задання блоку пам'яті (буфера), в якому це повідомлення розташовується. Використовувана для цього тріада (*buf*, *count*, *type*) входить до складу параметрів практично всіх функцій передачі даних.
2. Процеси, між якими виконується передача даних обов'язково повинні належати комунікатору, що вказується у функції *MPI\_Send*.
3. Параметр *tag* використовується тільки при необхідності розрізнення передаваних повідомлень, інакше, як значення параметра, може бути використане довільне додатнє ціле число.

Відразу після завершення виконання функції *MPI\_Send* процес-відправник може почати повторно використовувати буфер пам'яті, в якому розташовувалося повідомлення, що відправлялося. Також необхідно пам'ятати, що у момент завершення функції *MPI\_Send* стан самого повідомлення, що пересилається, може бути абсолютно різним: повідомлення може розташовуватися в процесі-відправнику, може знаходитися в стані передачі, може зберігатися в процесі-одержувачі або ж може бути прийнято процесом-одержувачем за допомогою функції *MPI\_Recv*. Тим самим, завершення функції *MPI\_Send* лише означає, що операція передачі почала виконуватися і пересилка повідомлення рано чи пізно буде виконана.



**Прийом повідомлень.** Для прийому повідомлення процес-одержувач повинен виконати функцію:

*int MPI\_Recv(void \*buf, int count, MPI\_Datatype type, int source,  
int tag, MPI\_Comm comm, MPI\_Status \*status), де*

*buf, count, type* - буфер пам'яті для прийому повідомлення, призначення кожного окремого параметра відповідає опису в *MPI\_Send*.

*source* - ранг процесу, від якого повинен бути виконаний прийом повідомлення.

*tag* - тег повідомлення, яке повинне бути прийняте для процесу.

*comm* - комунікатор, в рамках якого виконується передача даних.

*status* - вказівник на структуру даних з інформацією про результат виконання операції прийому даних.

Варто зазначити:

1. Буфер пам'яті повинен бути достатнім для прийому повідомлення. При браку обсягу пам'яті частина повідомлення буде втрачена і в кодї завершення функції буде зафіксована помилка переповнення. З іншого боку, повідомлення, що приймається, може бути коротшим від розміру приймального буфера. У такому разі зміняться тільки області буфера, використані прийнятим повідомленням.

2. Типи елементів повідомлення, що передаються і приймаються, повинні співпадати.

3. При необхідності прийому повідомлення від будь-якого процесу-відправника для параметра *source* може бути вказане значення *MPI\_ANY\_SOURCE* (на відміну від функції передачі *MPI\_Send*, яка посилає повідомлення строго певного адресата).

4. При необхідності прийому повідомлення з будь-яким тегом для параметра *tag* може бути вказане значення *MPI\_ANY\_TAG* (знову-таки, при використанні функції *MPI\_Send* повинне бути вказане конкретне значення тега).

5. На відміну від параметрів "процес-одержувач" і "тег", параметр "комунікатор" не має значення, що означає "будь-який комунікатор".

6. Параметр *status* дозволяє визначити ряд характеристик прийнятого повідомлення.

7. *Status.MPI\_SOURCE* - ранг процесу - відправника прийнятого повідомлення.

8. *Status.MPI\_TAG* - тег прийнятого повідомлення.

Значення змінної *status* дозволяє визначити кількість елементів даних в прийнятому повідомленні за допомогою функції:

*int MPI\_Get\_count(MPI\_Status \*status, MPI\_Datatype type, int \*count), де*

*status* - статус операції *MPI\_Recv*;

*type* - тип прийнятих даних;

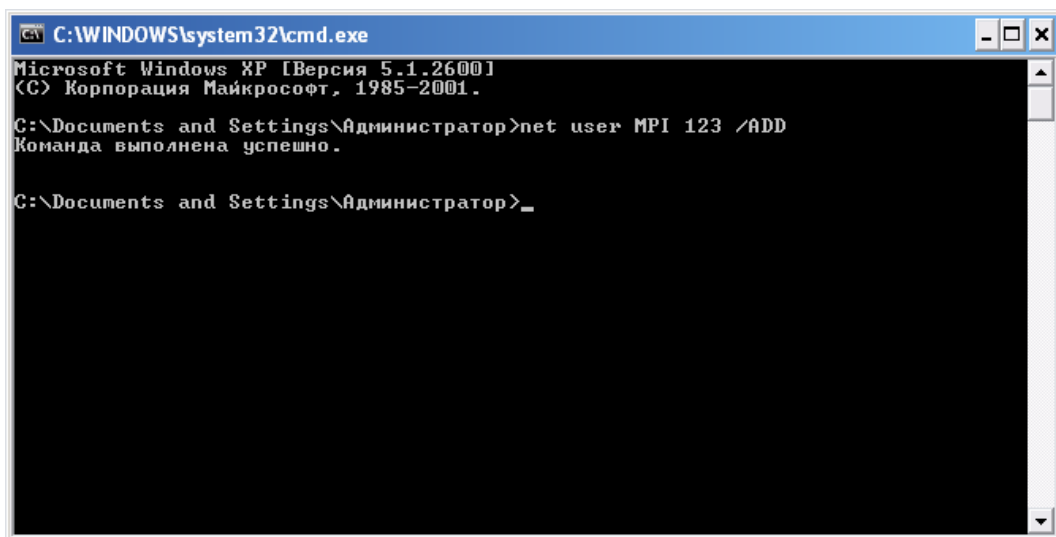
*count* - кількість елементів даних в повідомленні.

Виклик функції *Mpi\_Recv* не зобов'язаний бути узгодженим з часом виклику відповідної функції передачі повідомлення *MPI\_Send* - прийом повідомлення може бути ініційований до моменту, в момент чи після моменту початку відправки повідомлення.

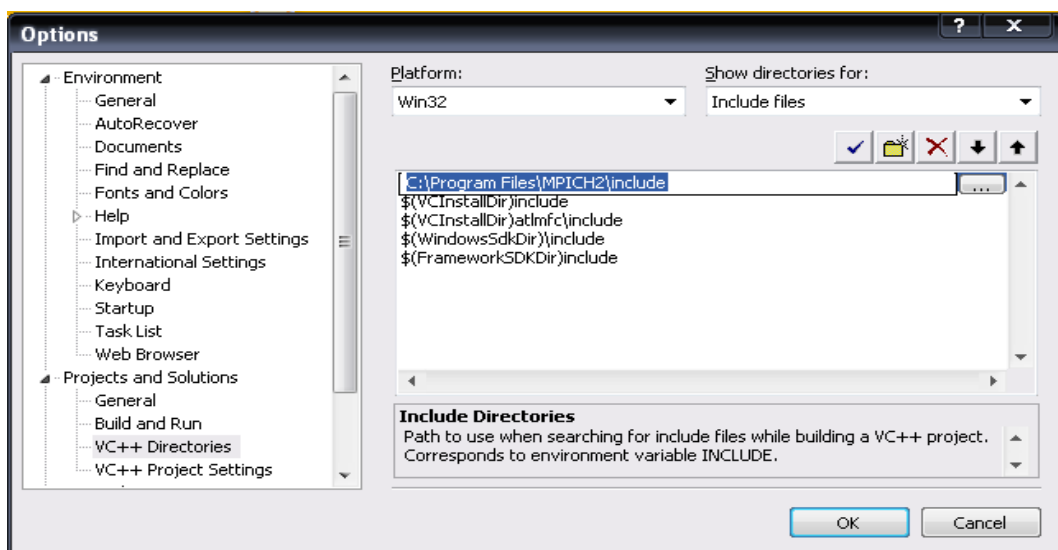
Після закінчення виконання функції *MPI\_Recv* в заданому буфері пам'яті буде розташоване прийняте повідомлення. Принциповий момент полягає в тому, що функція *MPI\_Recv* є блокуючою для процесу-одержувача, тобто його виконання припиняється до завершення роботи функції. Таким чином, якщо, по якихось причинах очікуване для прийому повідомлення буде відсутнє, виконання паралельної програми буде заблоковано.

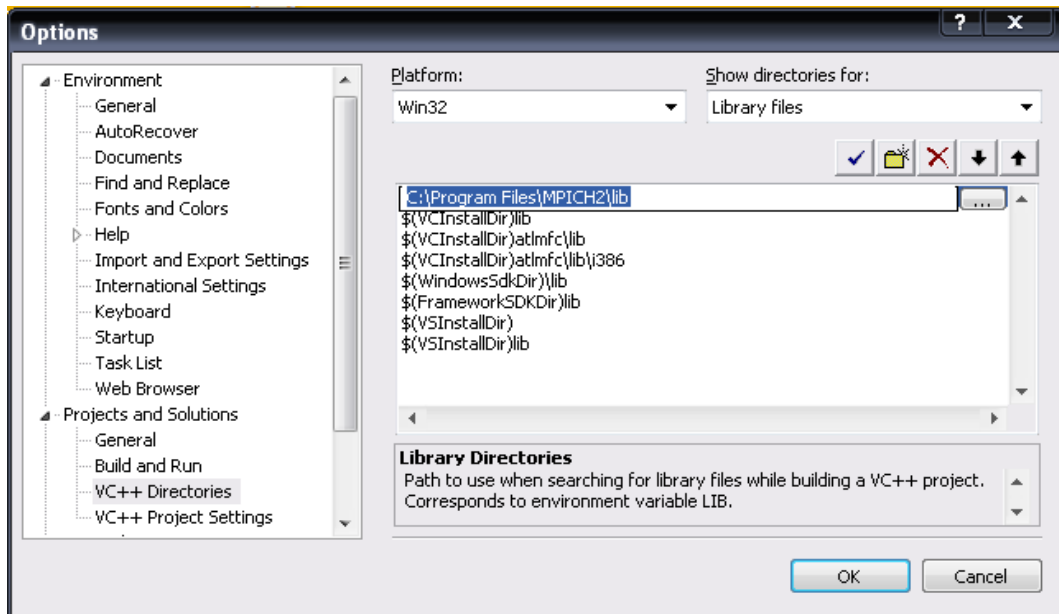
### Встановлення бібліотеки MPI

Запускаємо файл *mpich2-1.0.8-win-ia32.msi*, що додається до робіт та під керівництвом майстра встановлення інсталуємо *MPICH*. Далі командою `net user username password /add` прописати обліковий запис, під яким запускатимуться MPI- програми



Додати до середовища Visual Studio шляхи розміщення заголовних файлів та бібліотек





Після виконаних вказаних кроків бібліотека MPI готова до використання.

### КОНТРОЛЬНІ ЗАПИТАННЯ

1. Модель організації обчислень з використанням MPI?
2. Поняття паралельної програми в MPI.
3. Навести структуру паралельної програми з використанням MPI?
4. Як визначити кількість процесів у виконуваний паралельній програмі?
5. Які типи обміну повідомлень між процесами ви знаєте?

### ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Встановити та налаштувати бібліотеку MPI для розробки програм у середовищі програмування Microsoft Visual Studio 2005.
2. Написати програму обміну повідомлень між процесами з використанням MPI (див таблицю варіантів завдання – кількість процесів 1).
3. Кожен процес має визначати свій ранг та пересилати його значення головному процесу. Головний процес повинен виводити отримане значення на екран.
4. Виконати розроблену програму для різної кількості процесів (к-ть процесів 2).
5. Зробити висновок про порядок прийому повідомлень головним процесом та звернути увагу на його зміну від виклику до виклику розробленої програми.

Варіант №	Кількість процесів 1	Кількість процесів 2
1	4	26
2	8	22
3	3	27
4	7	23
5	5	25
6	2	28
7	6	24
8	12	18
9	10	22
10	30	10
11	15	5
12	22	8
13	11	19
14	13	17
15	16	14
16	9	21
17	18	12
18	23	7
19	25	5
20	27	3
21	14	16
22	19	11
23	6	24
24	8	22
25	11	19
26	14	18
27	16	22
28	18	8
29	8	12
30	6	22

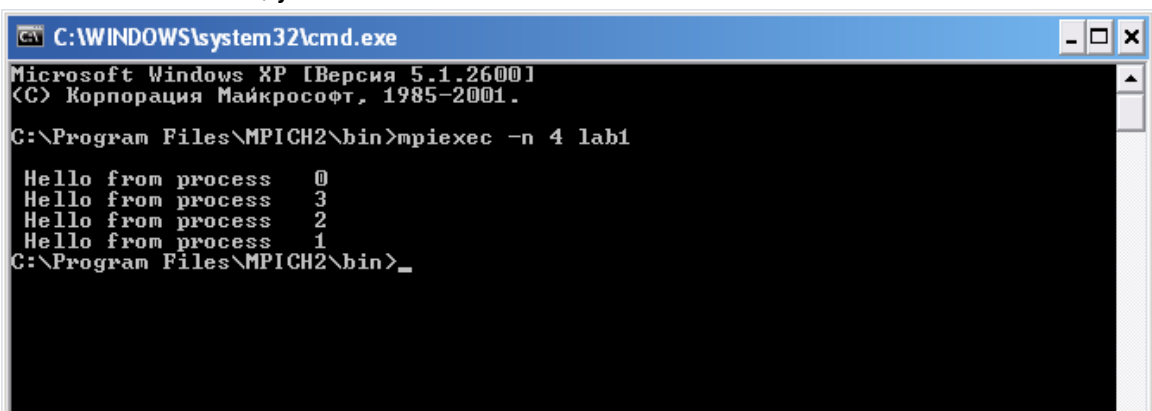
## ПРИКЛАД ВИКОНАННЯ

### Завдання:

Написати програму обміну повідомленнями між процесами з використанням MPI та запустити її для 4 процесів.

### Текст програми:

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[])
{
    int ProcNum, ProcRank, RecvRank;
    MPI_Status Status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    if ( ProcRank == 0 )
    {
        // Дії виконуються тільки процесом з рангом 0
        printf("\n Hello from process %3d", ProcRank);
        for (int i = 1; i < ProcNum; i++ )
        {
            MPI_Recv(&RecvRank,1, MPI_INT, MPI_ANY_SOURCE,
                    MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
            printf("\n Hello from process %3d", RecvRank);
        }
    }
    else // Повідомлення відправляється всіма процесами,
    // крім процесу з рангом 0
        MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
    MPI_Finalize();
    return 0; }
```



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Версія 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

C:\Program Files\MPICH2\bin>mpiexec -n 4 lab1

Hello from process 0
Hello from process 3
Hello from process 2
Hello from process 1
C:\Program Files\MPICH2\bin>_
```

Порядок прийому повідомлень заздалегідь не визначений і залежить від умов виконання паралельної програми (більш того, цей порядок може змінюватися від запуску до запуску). Так, можливий варіант результатів друку процесу 0 для паралельної програми з чотирьох процесів наведено на рисунку вище.

## Лабораторна робота №2

**Тема: Розпаралелювання програмного коду в OpenMP.**

**Мета роботи:** ознайомитись з засобами OpenMP для розпаралелювання коду та навчитись розробляти та виконувати відповідні програми.

### ТЕОРЕТИЧНІ ВІДОМОСТІ

Одним зі стандартів для програмування систем з загальною пам'яттю є інтерфейс (API) **OpenMP** (Open Multi-Processing). Цей стандарт реалізовано для мов програмування C, C++ та Fortran на великій кількості комп'ютерних архітектур, включаючи платформи Unix та Microsoft Windows.

OpenMP API складається з набору директив компілятора `pragma` (прагм), функцій та змінних середовища, що впливають на поведінку паралельної програми під час її виконання. Коли прагми OpenMP використовуються в програмі, вони дають вказівки компілятору створити виконуваний модуль, який буде виконуватись паралельно з використанням декількох потоків. При цьому в текст програми необхідно вносити лише невеликі зміни. В OpenMP застосовується модель паралельного виконання, що отримала назву «розгалуження-злиття». Така програма починається як один потік виконання, який називають начальним потоком. Коли потік зустрічає паралельну конструкцію, він створює нову групу потоків, що складається з цього потоку та невід'ємного числа допоміжних потоків, і стає головним в новій групі. Всі члени нової групи виконують код в межах паралельної конструкції. В кінці такої конструкції є невидимий бар'єр. Після її виконання код програми продовжує лише головний потік.

### Прагми OpenMP

Для активації підтримки прагм OpenMP в компіляторі необхідно застосовувати додаткові параметри або прапорці компіляції. Для Visual C++ таким параметром є `/openmp`, для gcc `-fopenmp`, для Sun Studio — `-xopenmp`. Також у Visual C++ необхідно увімкнути підтримку OpenMP можна за допомогою відповідного параметру в діалозі налаштування проекту: вибрати Configuration Properties, C/C++, Language і змінити значення параметру «OpenMP Support» на yes. Для використання функцій OpenMP необхідно підключити до проекту файл `vcomp.lib` (або `vcompd.lib` в режимі відлагодження). Цей файл з бібліотекою імпорту до файлу `vcomp.dll` (`vcompd.dll`).

Прагми OpenMP починаються зі слів **#pragma omp** і мають наступний формат:

`#pragma omp <директива> [список параметрів]`

OpenMP підтримує наступні директиви: `atomic`, `barrier`, `critical`, `flush`, `for`, `master`, `ordered`, `parallel`, `parallel for`, `section`, `sections` та `single`, що визначають або

механізми розділення коду, або конструкції синхронізації. Необов'язкові параметри директиви уточнюють її поведінку.

Найбільш уживана директива — **parallel**, яка має наступний синтаксис:

```
#pragma omp parallel [список параметрів]
```

Структурований блок

Вона інформує компілятор, що структурований блок (складений оператор) має виконуватись паралельно в кількох потоках. Як правило, кількість потоків дорівнює кількості процесорів в системі. В якості прикладу розглянемо варіант класичної програми «Hello, world!»:

```
#include <stdio.h>

int main()
{
#pragma omp parallel
{

    printf ("Hello , OpenMP!\n");
    } /* #pragma omp parallel */
    return 0;
} /* int main() */
```

На двопроцесорній системі результат роботи цієї програми може бути наступним:

Hello, OpenMP!

Hello, OpenMP!

### Функції та змінні середовища OpenMP

OpenMP передбачає також набір функцій, що дозволяють:

- під час виконання програми отримувати та встановлювати різноманітні параметри, що визначають її поведінку, наприклад, кількість потоків, можливість вкладеного паралелізму.
- застосовувати синхронізацію на основі замків (locks).

Прототипи функцій знаходяться в файлі `omp.h`. Розглянемо набір функцій OpenMP, що використовуються найчастіше. Функція **omp\_get\_thread\_num()** повертає номер потоку, в якому була викликана. Головний потік паралельного блоку має номер 0. Функція має наступний прототип:

```
int omp_get_thread_num (void )
```

Функція **omp\_set\_num\_threads()** має наступний прототип:

```
void omp_set_num_threads (int num_threads)
```

і використовується для встановлення кількості потоків, що будуть виконуватись в наступному паралельному блоці. Аргумент функції має бути позитивним числом.

Для визначення поточної кількості паралельних потоків застосовується функція **omp\_get\_num\_threads()** з наступним прототипом:

```
int omp_get_num_threads (void)
```

Для визначення чи виконується блок коду в паралельному режимі використовується функція **omp\_in\_parallel:**

```
int omp_in_parallel (void)
```

Якщо ця функція викликана з паралельного блоку, то вона повертає ненульове значення, в іншому випадку — повертає нуль.

В наступному лістингу наведено текст програми, що демонструє використання розглянутих функцій.

```
#include <stdio.h>
#include <omp.h>
int main()
{
    omp_set_num_threads (5) ;
    #pragma omp parallel
    {
        printf ("Hello , OpenMP! ( thread num=%d)\n",
            omp_get_thread_num () ) ;
    } /* #pragma omp parallel */
    return 0;
} /* int main () */
```

В стандарті OpenMP визначено ряд змінних середовища операційної системи, які контролюють поведінку OpenMP-програм. Зокрема, змінна **OMP\_NUM\_THREADS** визначає максимальну кількість потоків, що будуть виконуватись в паралельній програмі. Наприклад, в командному інтерпретаторі `bash` в консолі Linux кількість потоків OpenMP-програми можна встановити наступним чином:

```
export OMP_NUM_THREADS=4
```

В командному рядку Windows необхідно виконати наступну команду:

```
export OMP_NUM_THREADS=4
```

Таким чином, за допомогою функції `omp_set_num_threads()` або змінної середовища `OMP_NUM_THREADS` можна встановити довільну кількість потоків, що будуть створюватись під час виконання OpenMP-програм.



Більш детальну інформацію про OpenMP можна знайти на офіційному сайті цього стандарту, який містить великий перелік відповідних ресурсів.

### Директива **for**

Однією з директив, що застосовується досить часто, є директива **for**, яка належить до директив розподілення роботи (work-sharing directive). Ця директива інформує компілятор, що при виконанні циклу **for** в паралельному блоці ітерації циклу мають бути розподілені між потоками групи. Розглянемо роботу директиви **for** на прикладі програми, текст якої наведено в лістингу

```
#include <stdio.h>
#include <omp.h>
int main ()
{
    int i;
#pragma omp parallel
    {
#pragma omp for
        for (i = 0; i < 5; i++)
            printf ("thread N%d i=%d\n ",
                omp_get_thread_num (), i);
    } /* #pragma omp parallel */
    return 0;
}
```

Результат роботи програми наведено нижче:

```
thread N1 i=3
thread N1 i=4
thread N0 i=0
thread N0 i=1
thread N0 i=2
```

Порядок появи рядків на екрані може бути довільним і змінюватись при кожному запуску програми. Якщо з тексту програми видалити директиву **#pragma omp for**, то кожний потік буде виконувати повний цикл **for**.

У зв'язку з тим, що розпаралелювання коду найчастіше виконується саме в циклічних конструкціях, OpenMP має скорочений варіант запису комбінації директив **#pragma omp parallel** і **#pragma omp for**:

```
#pragma omp parallel for
for(i = 0 ; i < 5; ++i)
```

...

Слід підкреслити, що розпаралелюватись можуть лише такі цикли, в яких ітерації не мають залежностей одна від одної. Якщо цикл не має залежностей, компілятор має змогу виконати цикл в будь-якому порядку, навіть паралельно. Також цикли, які OpenMP може розпаралелювати, мають відповідати наступному формату:

```
For ( цілий тип i = інваріант циклу1;
      i {<,>,<=,>=} інваріант циклу2;
      i {+,-}= інваріант циклу3)
```

Ці вимоги введені для того, щоб OpenMP міг визначити кількість ітерацій циклу. При розробці паралельних програм необхідно враховувати, які змінні є спільними (shared), а які приватними (private). Спільні змінні доступні всім потокам в групі, тому зміна значень таких змінних в одному потоці стає видимою в інших потоках. Що стосується приватних змінних, то кожний потік має окремі їх копії, тому їх зміна в одному потоці не відображається на інших потоках.

За замовчуванням всі змінні в паралельному блоці є спільними. Лише в трьох випадках змінні є приватними. По-перше, приватними є індекси паралельних циклів. По-друге, приватними є локальні змінні паралельних блоків. По-третє, приватними стають змінні, що вказані в параметрах private, firstprivate, lastprivate и reduction прагми for.

Параметр **private** вказує, що для кожного потоку має бути створена локальна копія кожної змінної, що вказана в списку. Приватні копії будуть ініціалізуватись значенням за замовчуванням, при можливості буде застосовуватись конструктор за замовчуванням.

Параметр **firstprivate** виконує аналогічні дії, однак перед виконанням паралельного блоку він копіює значення приватної змінної в кожний потік, застосовуючи при необхідності конструктор копій.

Параметр **lastprivate** теж подібний до параметра private, однак після виконання останньої ітерації циклу або конструкції паралельного блоку значення змінних, що вказані в списку цього параметра, привласнюються змінним основного потоку. При цьому може застосовуватись оператор привласнювання копій.

Параметр **reduction** в якості аргументів приймає змінну й оператор. Оператори, що підтримуються reduction, наведені в табл. 1, а змінна має бути скалярною (наприклад, **m t** або float). Змінна параметру reduction ініціалізується значенням, що

вказане в табл. 1. Після завершення паралельного блоку вказаний оператор застосовується до кожної приватної копії і початкового значенні змінної.

В лістингу наведено приклад програми, що використовує розглянуті параметри.

```
#include <stdio.h>
#include <omp.h>
int main ()
{
    int i, j;
    int array [] = { 1 , 2, 3, 4, 5, 6, 7, 8, 9 };
    int sum;
    #pragma omp parallel
    {
        #pragma omp for firstprivate(j) \
                        lastprivate (i) \
                        reduction(+: sum)
        for (i = 0; i < sizeof (array)/sizeof (array[0]); i++)
        {
            long private = i ;
            sum += array [ i ] ;
        } /*    for(i = 0; i < 5; i++) */}
        /* #pragma omp parallel */
        printf ("%d\n" , sum);
        return 0;
    }
}
```

В цьому лістингу змінна **i** є приватною тому, що вона є змінною циклу **for**, а також тому, що вказана в параметрі **lastprivate**. Змінна **j** примусово зроблена приватною за допомогою **firstprivate**. Змінна **private** також є приватною через те, що вона оголошена в паралельному блоці. В кожному потоці екземпляр приватної змінної **sum** неявно ініціалізується нулем.

**Табл. 1. Оператори параметру reduction**

Оператор reduction	Ініціалізаційне значення змінної
+	0
*	1
-	0
&	~0 (усі біти в змінній встановлені)
	0
^	0
&&	0
	0

## Директива sections

Як 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 директивою **#pragma omp section**. Кожній секції ставиться у відповідність один потік, і всі секції виконуються паралельно.

Фрагмент програми, що демонструє розпаралелювання на рівні функцій наведено в лістингу:

```
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) */
```

## КОНТРОЛЬНІ ЗАПИТАННЯ

1. Якими директивами виконується розпаралелювання циклів в OpenMP?
2. Опишіть спільні та приватні змінні в паралельних блоках?
3. Яким чином діє параметр private та його модифікації: firstprivate, lastprivate, reduction?
4. Які оператори можуть застосовуватись з параметром reduction?
5. Процес розпаралелювання коду на рівні функцій?

## ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Виконати програми, що наведені в лістингах та проєкспериментуйте з ними. Зокрема в програмі реалізуйте процедуру введення кількості потоків користувачем. Реалізуйте процедуру в якій парні потоки друкують «Hello, OpenMP!», а непарні – ім'я та прізвище студента.

2. Розробити програму згідно з варіантом та передбачити в ній використання двох потоків кожен з яких виконує інші операції. Для створення потоків використовувати механізм секцій.

№	Умова задачі
1.	Дано масив $X_i$ , $i=1, 2, \dots, 15$ . Знайти добуток додатніх та від'ємних елементів масиву P1 та P2. Визначити, що менше P1 чи P2.
2.	Дано масиви $A_i$ , $i=1, \dots, 15$ и $D_j$ , $j=1, \dots, 20$ . Знайти мінімальні елементи цих масивів $A_{\min}$ и $D_{\min}$ . Визначити який елемент менший.
3.	Дано масив $X_i$ , $i=1, \dots, 15$ . Знайти середнє арифметичне додатніх та від'ємних елементів масиву R1 та R2. Визначити, який із знайдених елементів більший.
4.	Дано масив $A_i$ , $i=1, 2, \dots, 15$ . Знайти суму додатніх елементів та суму від'ємних елементів масиву S1 та S2. Визначити, що більше S1 чи S2.
5.	Дано масиви $X_i$ , $i=1, \dots, 10$ . та $Y_j$ , $j=1, 2, \dots, 15$ . Знайти максимальні елементи масивів $X_{\max}$ та $Y_{\max}$ . Визначити, який елемент більше $X_{\max}$ чи $Y_{\max}$ .
6.	Дано масив $X_i$ , $i=1, 2, \dots, 15$ . Знайти добуток додатніх та від'ємних елементів масиву P1 та P2. Визначити, що більше по абсолютній величині P1 чи P2.
7.	Дано масив $X_i$ , $i=1, \dots, 15$ . Знайти кількість додатніх та від'ємних елементів масиву K1 та K2. Визначити, що більше K1 чи K2.
8.	Дано масив $X_i$ , $i=1, 2, \dots, 15$ . Знайти добуток від'ємних та додатніх елементів масиву P1 та P2. Визначити, що більше P1 чи P2.
9.	Дано масиви $X_i$ , $i=1, \dots, 10$ . и $Y_j$ , $j=1, 2, \dots, 15$ . Знайти максимальні елементи масивів $X_{\max}$ и $Y_{\max}$ . Визначити, який елемент менший $X_{\max}$ чи $Y_{\max}$ та на скільки.
10.	Обчислити $Z = 0.2X - Y^2$ , де X – максимальний елемент масиву $A_i$ ( $i=1, 25$ ), Y – мінімальний елемент масиву $B_j$ ( $j=1, 30$ ).
11.	Дано масиви $A_i$ , $i=1, \dots, 15$ и $D_j$ , $j=1, \dots, 20$ . Знайти мінімальні елементи цих масивів $A_{\min}$ та $D_{\min}$ . Визначити, який елемент більший.
12.	Дано масив $X_i$ , $i=1, \dots, 15$ . Знайти середнє арифметичне додатніх та від'ємних елементів масиву R1 та R2. Визначити, який із знайдених елементів менший.
13.	Дано масив $A_i$ , $i=1, 2, \dots, 15$ . Знайти суму додатніх та добуток від'ємних елементів S1 та S2.
14.	Дано масиви $X_i$ , $i=1, \dots, 10$ . и $Y_j$ , $j=1, 2, \dots, 15$ . Знайти максимальні елементи масивів

	X <sub>max</sub> та Y <sub>max</sub> . Визначити, який з X <sub>max</sub> чи Y <sub>max</sub> менший і на скільки.
15.	Дано масив X <sub>i</sub> , i=1, 2, ...,15. Знайти добуток додатніх та від'ємних елементів масиву P1 та P2. Визначити, що менше по абсолютній величині P1 чи P2.
16.	Дано масив X <sub>i</sub> , i=1, ..., 15. Знайти кількість додатніх та від'ємних елементів масиву K1 та K2 . Визначити, що більше K1 чи K2 і на скільки.
17.	Дано масив X <sub>i</sub> , i=1, 2, ...,15. Знайти добуток додатніх та від'ємних елементів масиву P1 та P2. Визначити, що менше P1 чи P2.
18.	Дано масиви X <sub>i</sub> , i=1,...,10. та Y <sub>j</sub> , j=1,2,...,15. Знайти максимальні елементи масивів X <sub>max</sub> та Y <sub>max</sub> . Визначити, який елемент більше X <sub>max</sub> чи Y <sub>max</sub> .
19.	Обчислити $Z = 0.3X - Y^3$ , де X – мінімальний елемент масиву A <sub>i</sub> (i=1,..., 25), Y – максимальний елемент масиву B <sub>j</sub> (j=1,..., 30).
20.	Дано масиви B <sub>i</sub> , i=1, ... , 15 та C <sub>j</sub> , j=1, ..., 20. Знайти мінімальні елементи цих масивів B <sub>min</sub> та C <sub>min</sub> . Визначити, який елемент менший.
21.	Дано масиви A <sub>i</sub> , i =1,...,15. Знайти середнє арифметичне додатніх та від'ємних елементів масиву R1 та R2. Визначити, який із знайдених елементів менший по абсолютній величині.
22.	Дано масив C <sub>i</sub> , i=1,2,...,15. Знайти суму додатніх та від'ємних елементів масиву S1 та S2. Визначити, що менше S1 чи S2 .
23.	Дано масиви X <sub>i</sub> , i=1,...,10. та Y <sub>j</sub> , j=1,2,...,15. Знайти максимальні елементи масивів X <sub>max</sub> та Y <sub>max</sub> . Визначити, який із X <sub>max</sub> чи Y <sub>max</sub> менший і на скільки.
24.	Дано масив A <sub>i</sub> , i=1, ...,20. Знайти добуток від'ємних та додатніх елементів масиву P1 та P2. Визначити, що менше по абсолютній величині P1 чи P2.
25.	Дано масив A <sub>i</sub> , i=1, ..., 30. Знайти кількість додатніх та від'ємних елементів масиву K1 та K2. Визначити, що більше K1 чи K2 і на скільки.
26.	Дано масив Y <sub>i</sub> , i=1, ...,25. Знайти добуток додатніх та від'ємних елементів масиву P1 та P2. Визначити, що менше P1 чи P2.
27.	Дано масив X <sub>i</sub> , i=1, 2, ...,15. Знайти добуток додатніх та від'ємних елементів масиву P1 та P2. Визначити, що менше по абсолютній величині P1 чи P2.
28.	Дано масив X <sub>i</sub> , i=1, ..., 15. Знайти кількість додатніх та від'ємних елементів масиву K1 та K2 . Визначити, що більше K1 чи K2 і на скільки.
29.	Дано масив X <sub>i</sub> , i=1, 2, ...,15. Знайти добуток додатніх та від'ємних елементів масиву P1 та P2. Визначити, що менше P1 чи P2.
30.	Дано масиви X <sub>i</sub> , i=1,...,10. та Y <sub>j</sub> , j=1,2,...,15. Знайти максимальні елементи масивів X <sub>max</sub> та Y <sub>max</sub> . Визначити, який елемент більше X <sub>max</sub> чи Y <sub>max</sub> .

3. Оформити звіт та сформулювати відповідні висновки.

### Лабораторна робота №3

**Тема:** Обчислення математичних виразів засобами OpenMP.

**Мета роботи:** навчитись розробляти та виконувати паралельні програми для вирішення інженерних завдань.

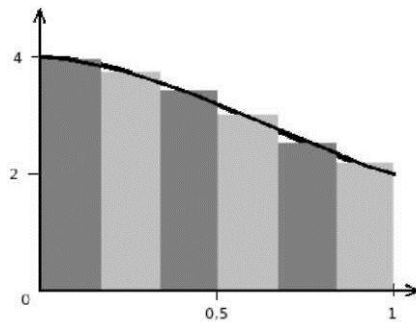
#### ТЕОРЕТИЧНІ ВІДОМОСТІ

Розглянемо задачу обчислення визначеного інтегралу:

$$I = \int_0^1 f(x) dx$$

Обчислення будемо вести методом прямокутників. З цією метою на відрізку  $[0,1]$  введемо рівномірну сітку  $\{x_j | x_j = j \cdot h; j=0, N; h \cdot N=1\}$  (рис. 1). Наближене значення інтеграла будемо обчислювати за наступною формулою:

$$I_N = \sum_{j=0}^{N-1} f(x_j + 0.5h) h$$



**Рис. 1. Обчислення інтегралу методом прямокутників**

Очевидно, що обчислення площ прямокутників можна виконувати паралельно кількома процесами. В якості прикладу розглянемо обчислення числа  $Pi$ , що обчислюється наступним чином:

$$\int_0^1 \frac{1}{1+x^2} dx = \arctg(1) - \arctg(0) = \pi/4$$

В лістингу наведено текст OpenMP-програми, що виконує обчислення числа  $Pi$ .

```
#include <stdio.h>
#include <omp.h>
int main( int argc , char **argv)
{
    int me, size ;
```

```

    unsigned intervals;
    int i, j;
    double h, Ik, x;
printf(" Enter intervals:");
scanf("%u", &intervals);
h = 1.0/intervals;
Ik = 0;
#pragma omp parallel
{
    #pragma omp for private(x) reduction( + :Ik)
    for (j = 0; j < intervals; j++)
    {
        printf("thread N%dj=%d\n", omp_get_thread_num(), j);
        x = (j + 0.5) * h;
        Ik += 4/(1 + x * x);
    }
}

Ik *= h;
printf("value = %.10lf\n", Ik);
return 0;
}

```

Іншим способом вирішення задачі є знаходження інтегралу використанням підходу Ньютона-Лейбніца.

**Основна теорема інтегрального числення.** Якщо  $F(x)$  є якою-небудь первісною від неперервної функції  $f(x)$ ,  $x \in [a; b]$ , то справедлива формула

$$\int_a^b f(x)dx = F(b) - F(a).$$

Різницю  $F(b) - F(a)$  значень функції  $F(x)$  часто записують у вигляді символу  $F(x)|_a^b$ . В цих позначеннях формула Ньютона-Лейбніца набуває вигляду

$$\boxed{\int_a^b f(x)dx = F(x)|_a^b.}$$

Ця формула встановлює взаємозв'язок між визначеним інтегралом та первісною для функції  $f(x)$  на відрізку  $[a; b]$ .



## Приклади розв'язування задач.

1.1. Обчислити інтеграли, застосовуючи формулу Ньютона-Лейбніца.

$$1) \int_0^1 \sqrt{1+x} \, dx = \int_0^1 (1+x)^{\frac{1}{2}} \, dx = \frac{(1+x)^{\frac{1}{2}+1}}{\frac{1}{2}+1} \bigg|_0^1 = \frac{2}{3} \cdot (1+x)^{\frac{3}{2}} \bigg|_0^1 = \frac{2}{3} \cdot \left( (1+1)^{\frac{3}{2}} - (1+0)^{\frac{3}{2}} \right) = \\ = \frac{2}{3} \left( 2^{\frac{3}{2}} - 1^{\frac{3}{2}} \right) = \frac{2}{3} (\sqrt{8} - 1).$$

$$2) \int_{-1}^7 \frac{dx}{\sqrt{3x+4}} = \frac{1}{3} \int_{-1}^7 \frac{d(3x+4)}{\sqrt{3x+4}} = \frac{2}{3} \int_{-1}^7 \frac{d(3x+4)}{2\sqrt{3x+4}} = \frac{2}{3} \sqrt{3x+4} \bigg|_{-1}^7 = \frac{2}{3} \sqrt{25} - \frac{2}{3} \sqrt{1} = \\ = \frac{10}{3} - \frac{2}{3} = \frac{8}{3}.$$

$$3) \int_0^4 (1 + e^{\frac{x}{4}}) \, dx = \int_0^4 dx + \int_0^4 e^{\frac{x}{4}} \, dx = x \bigg|_0^4 + 4 \cdot e^{\frac{x}{4}} \bigg|_0^4 = (4-0) + (4e - 4 \cdot 1) = 4e.$$

## ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Виконати програму, наведену в лістингу та переконатися у її ефективній роботі.  
Сформулювати висновки про можливе вдосконалення програмного коду.
2. Розробити OpenMP-програму для обчислення визначеного інтегралу (вибирається згідно з варіантом).
3. Оформити звіт згідно з вимогами.

*Варіанти завдань*

$$1 \quad \int_{-2}^{-1} \frac{dx}{(11+5x)^3}.$$

$$2 \quad \int_0^{16} \frac{dx}{\sqrt{x+9} - \sqrt{x}}.$$

$$3 \quad \int_0^1 (e^x - 1)^4 e^x \, dx.$$

$$4 \quad \int_0^1 \frac{x \, dx}{(x^2 + 1)^2}.$$

$$5 \quad \int_{-\pi/2}^{\pi/2} \frac{dx}{1 + \cos x}.$$

$$6 \quad \int_1^{e^2} \frac{dx}{x\sqrt{1 + \ln x}}.$$

$$7 \quad \int_0^{\frac{\pi}{4}} x \sqrt{\operatorname{tg} x} \, dx.$$

$$8 \quad \int_0^1 \frac{dx}{x^2 + 4x + 5}.$$

$$9 \quad \int_{-1/2}^1 \frac{dx}{\sqrt{8 + 2x - x^2}}.$$

$$10 \quad \int_0^4 \frac{x-1}{x+1} \, dx.$$

$$11 \quad \int_1^4 (x + 2\sqrt{x}) \, dx.$$

$$12 \quad \int_0^1 e^{-x^2} \, dx.$$

$$13 \quad \int_0^1 x^x \, dx.$$

$$14 \quad \int_{1/e}^e x^2 e^{-x^2} \, dx.$$

$$15 \quad \int_0^{2\pi} \frac{dx}{10 + 3\cos x}.$$

## Лабораторна робота №4

**Тема:** Застосування процесу декомпозиції в паралельному програмуванні.

**Мета роботи:** ознайомитись та вивчити на практиці методи декомпозиції.

### ТЕОРЕТИЧНІ ВІДОМОСТІ

Найважливішим та найважчим етапом при створенні програми є розробка алгоритму, особливо, якщо мова йде про паралельний алгоритм. Процес створення паралельного алгоритму можна розбити на чотири кроки.

**1. Декомпозиція.** На цьому етапі вихідна задача аналізується, оцінюється можливість її розпаралелювання. Іноді вигаиш від розпаралелення може бути незначним, а трудоемкість розробки паралельної програми велика. В цьому випадку перший крок розробки алгоритму виявляється і останнім. Якщо ж ситуація відмінна від описаної, то задача та пов'язані з нею дані розділяються на дрібніші частини – підзадачі і фрагменти структур даних. Особливості архітектури конкретної обчислювальної системи на цьому етапі можуть не враховуватися.

**2. Проектування комунікацій**(обміну даними) між задачами. На цьому етапі визначаються зв'язки, необхідні для пересилання вхідних даних, проміжних результатів виконання підзадач, а також комунікації, що необхідні для керування роботою під задач. Обираються методи та алгоритми комунікацій.

**3.Укрупнення.** Підзадачі можуть об'єднуватися у більші блоки, якщо це дозволяє підвищити ефективність алгоритму і знизити трудоемкість розробки. Основними критеріями на даному кроці є ефективність алгоритму (в першу чергу-продуктивність) та трудоемкість його реалізації.

**4. Планування обчислень.** На цьому кроці виконується розподіл під задач між процесорами. Основний критерій вибору способу розміщення під задач – ефективне використання процесорів з мінімальними затратами часу на обмін даними.

#### **Декомпозиція (сегментування).**

На цьому етапі визначається степінь можливого розпаралелення задачі. Іноді декомпозиція поставленої задачі природним чином впливає з самої задачі тому є очевидною, іноді – ні. Чим менший розмір підзадач, отриманих в результаті декомпозиції, чим більша їх кількість, тим гнучкішим виявляється паралельний алгоритм і тим легше забезпечити рівномірне завантаження процесорів обчислювальної системи. Надалі, можливо доведеться укрупнити алгоритм, оскільки слід прийняти до уваги інтенсивність обміну даними та інші фактори. Сегментувати можна як обчислювальний алгоритм, так і дані. Застосовуються різні варіанти декомпозиції.

В методі декомпозиції даних спочатку сегментуються дані, а потім алгоритм їх обробки. Дані розбиваються на сегменти приблизно однакового розміру. З фрагментами даних пов'язуються операції їх обробки, з яких і формуються підзадачі. Визначаються необхідні правила обміну даними. Перекриття частин, на які розбивається задача, повинне бути мінімальним. Це дозволяє уникнути дублювання обчислень. Схема розбиття може в подальшому уточнюватися. Іноді, якщо це необхідно для зменшення кількості обмінів, допускається збільшення степені перекриття фрагментів задачі.

При декомпозиції даних спочатку аналізуються структури даних найбільшого розміру, або ті, до яких найчастіше відбувається звертання. На різних стадіях розрахунків можуть використовуватися різні структури даних, тому можуть бути необхідними динамічні, тобто такі, що міняються в часі схеми декомпозиції цих структур.

В методі функціональної декомпозиції спочатку сегментується обчислювальний алгоритм, а потім під цю схему підбирається схема декомпозиції даних. Успіх у цьому випадку не завжди гарантовано, оскільки схема може вимагати багатьох додаткових пересилань даних. Цей метод декомпозиції може виявитися корисним у випадку, якщо немає структур даних, які б могли бути розпаралелені очевидним чином. Функціональна декомпозиція відіграє важливу роль і як метод структурування програм. Вона може виявитися корисною при моделюванні складних систем, що складаються з множини простих підсистем, зв'язаних між собою набором інтерфейсів.

Розмір блоків, з яких складається паралельна програма, може бути різним. В залежності від розміру блоків, алгоритм може мати різну “зернистість”. Її виміром в найпростішому випадку є кількість операцій у блоці. Виділяють три степені зернистості: дрібнозернистий, середньоблоковий та великоблоковий. Зернистість пов'язана з рівнем паралелізму.

Паралелізм на рівні команд найбільш дрібнозернистий. Його масштаб менше ніж 20 команд на блок. Кількість підзадач, що паралельно виконуються – від однієї до кількох тисяч, при чому середній масштаб паралелізму складає біля п'яти команд на блок.

Наступний рівень - паралелізм на рівні циклів. Переважно, цикл містить не більше 500 команд. Якщо ітерації циклу незалежні, вони можуть виконуватися, наприклад за допомогою конвеєра або на векторному процесорі. Це також дрібнозернистий паралелізм.

Паралелізм на рівні підзадач – середньоблоковий. Розмір блоку – до 2000 команд. Виконання такого паралелізму реалізувати важче, оскільки слід враховувати можливі міжпроцедурні залежності. Вимоги до комунікацій менші, ніж у випадку паралелізму на рівні команд. Паралелізм на рівні програм (задач) – великоблоковий. Він означає виконання незалежних програм на паралельному комп'ютері. Великоблоковий паралелізм повинен підтримуватися операційною системою.

Обмін даними через спільні/розподілені змінні використовується на рівнях дрібнозернистого і середньоблокового паралелізму, а на великоблоковому – засобами передачі повідомлень. Досягнути ефективності роботи паралельної програми можна, якщо збалансувати зернистість алгоритму і затрати часу на обмін даними.

Частини програми можуть виконуватися паралельно, лише якщо вони незалежні.

*Незалежність за даними* полягає в тому, що дані, які обробляються однією частиною програми не модифікуються іншою її частиною.

*Незалежність за керуванням* полягає в тому, що послідовність виконання частин програми може бути визначена лише під час виконання програми(наявність залежності по виконанню наперед визначає порядок виконання).

*Незалежність за ресурсами* забезпечується достатньою кількістю комп'ютерних ресурсів(об'єм пам'яті, кількість функціональних вузлів та ін.).

*Залежність за виводом* виникає, якщо дві підзадачі виконують запис в одну і ту ж змінну. *А залежність за вводом/виводом*, якщо оператори вводу/виводу двох чи декількох під задач звертаються до одного файлу(чи змінної).

Дві підзадачі можуть виконуватися паралельно, якщо вони незалежні за даними, за керуванням і за операціями вводу виводу.

## **КОНТРОЛЬНІ ЗАПИТАННЯ**

1. Опишіть метод декомпозиції та основні його стадії роботи
2. Особливості застосування паралелізму на рівні задач та команд
3. Опишіть основні етапи створення паралельних програм
4. Поняття функціональної декомпозиції та сфери її застосування.

## **ПОРЯДОК ВИКОНАННЯ РОБОТИ**

1. Використовуючи метод функціональної декомпозиції, розробити алгоритм обчислення запропонованого матрично-векторного виразу, який би враховував можливість паралельного виконання. ВСІ вхідні дані є цілими числами, більшими за нуль.

2. Визначити паралелізм якого рівня присутній в алгоритмі та зробити висновки щодо залежностей даних, керування, ресурсів, вводу/виводу;

3. На основі створеного алгоритму написати програму яка дозволяє обчислити вираз та ілюструє проведену декомпозицію та володіє такими характеристиками:

- 1) Введення розмірності даних (n);
- 2) Можливість вибору – ввід даних(тобто елементів матриці та векторів) з клавіатури чи генерування їх випадковим чином;
- 3) вивід на екран (або у файл) проміжних результатів за потребою користувача;

Правила знаходження елементів виразу.

1).Задати\* квадратну матрицю  $A$  порядку  $n$ . Отримати вектор(стовпець)  $y_1 = A * b$ , де  $b$  – вектор-стовпець, елементи якого обраховуються за формулою, згідно варіанту.

2).Задати квадратну матрицю  $A_1$  порядку  $n$  та вектори-стовпці  $b_1$  та  $c_1$  з  $n$  елементами кожен. Отримати вектор  $y_2$  згідно формули, що задається варіантом.

3).Задати квадратні матриці  $A_2$  та  $B_2$  порядку  $n$ . Отримати матрицю  $Y_3$ , яка залежить від  $A_2$ ,  $B_2$  та додатково визначеної матриці  $C_2$ , елементи якої знаходяться за формулою, вказаною варіантом.

Варіанти завдань

При чому:  $y'$  означає операцію транспонування;  $i, j = 1 \dots n$  ( $n$  – вхідна розмірність).

1	$x = Y_3^2 y_2 + Y_3 (y_1 + y_2)$ стовпець		
	$b_i = 1/(i^2 + 2)$ для парних $i$ $b_i = 1/i$ для непарних $i$	$A_1(b_1 + c_1)$	$A_2(B_2 - C_2)$ $C_{ij} = 1/(i + 2j)$
2	$x = Y_3^2 y_2 + Y_3 * (y_1 + y_2) + y_1 y_2' y_2 + Y_3^3 y_1$ стовпець		
	$b_i = 1/(i^2 + 2 + i)$ для парних $i$ $b_i = 1/i$ для непарних $i$	$A_1(b_1 + 2c_1)$	$A_2(C_2 - B_2)$ $C_{ij} = 1/(i + j)$
3	$x = Y_3 y_2 y_2' + Y_3^3 - Y_3 + y_2 y_1' + Y_3^2 y_1 y_1'$ матриця		
	$b_i = 3/(i^2 + 3)$ для парних $i$ $b_i = 3/i$ для непарних $i$	$A_1(3b_1 + c_1)$	$A_2(B_2 - C_2)$ $C_{ij} = 1/(i + j)^2$
4	$x = (Y_3 y_1 + Y_3^2 y_2 + y_2' y_1 Y_3 y_1)' * Y_3$ рядок		
	$b_i = 4/(i^3 + 3)$	$A_1(b_1 + 4c_1)$	$A_2(B_2 + C_2)$ $C_{ij} = 1/(i + j^2)$
5	$x = (y_1' Y_3 * y_1 + y_2') * (Y_3 * y_2 + y_1 + y_1 y_2' Y_3^2 y_2)$ число		
	$b_i = 5i^3$	$A_1(5b_1 - c_1)$	$A_2(B_2 + 10C_2)$ $C_{ij} = 1/(i^2 + j)$
6	$x = y_2' y_1 Y_3^2 + y_1' y_2 Y_3^2 + y_2' Y_3 y_1 Y_3 + Y_3$ матриця		
	$b_i = 6/i^2$	$A_1(6b_1 - c_1)$	$A_2(10B_2 + C_2)$ $C_{ij} = 1/(i + j)^3$
7	$x = y_2' * (y_1 y_2' + Y_3^3 + y_1' y_2 Y_3) * y_1$ число		
	$b_i = 7i$	$A_1(b_1 + c_1)$	$A_2(B_2 - C_2)$ $C_{ij} = 1/(i^3 + j^2)$

8	$(y_1' Y_3 y_1 y_2 y_2' + Y_3^2 + y_1 y_2') * (y_2' Y_3 y_2 y_1 + y_1)$ столбец		
	$b_i = 8/i$	$A_1(2b_1 + 3c_1)$	$A_2(B_2 - C_2)$ $C_{ij} = 1/(i+j+2)$
9	$x = y_2'(Y_3 + y_1' y_1 Y_3^2 + y_2 y_2') + y_1'(y_1' y_1 Y_3 + Y_3^3)$ рядок		
	$b_i = 9i$	$A_1(b_1 - c_1)$	$A_2(B_2 + C_2)$ $C_{ij} = 1/(i+j)$
10	$x = (y_2'(y_1' y_1 Y_3) + y_1' Y_3^3 + y_2' Y_3) * (Y_3 y_1 + y_2' y_2 y_1)$ число		
	$b_i = 10/(i^2 + 1)$	$A_1(b_1 + c_1)$	$A_2(C_2 + 2B_2)$ $C_{ij} = 1/(i+2j)$
11	$x = y_2 y_1' + y_2' y_2 Y_3 + y_1' Y_3^2 * y_2 + Y_3 + Y_3 y_1 y_1' Y_3$ матрица		
	$b_i = 11i^2$ для парних $i$ $b_i = 11/i$ для непарних $i$	$A_1(b_1 - 2c_1)$	$A_2(B_2 - 2C_2)$ $C_{ij} = 1/(i^2 + j)$
12	$x = y_2' + ((y_1'(Y_3^2 y_1 + y_2) Y_3 + y_1 y_2') y_2)$ рядок		
	$b_i = i^2/12$ для парних $i$ $b_i = i$ для непарних $i$	$A_1(12b_1 - c_1)$	$A_2(B_2 - C_2)$ $C_{ij} = 1/(i+j^2)$
13	$x = Y_3 y_1 + Y_3 y_2 + (13 y_1 + y_2) + y_1' y_2 y_1$ столбец		
	$b_i = 13/(i+2)$ для парних $i$ $b_i = 13/i^2$ для непарних $i$	$A_1 b_1 - c_1$	$A_2 - B_2 C_2$ $C_{ij} = 13/(i^2 + j^2)$
14	$x = Y_3^3 y_1 y_1' + y_2 y_1' + y_2' Y_3 y_1 Y_3$ матрица		
	$b_i = 14/(i^3)$ для парних $i$ $b_i = 1/(i+14)$ для непарних $i$	$A_1(14b_1 + 14c_1)$	$A_2 C_2 - B_2$ $C_{ij} = 14/(i+j^4)$
15	$x = y_2' + y_1' + (Y_3^2 y_1)' + y_2' y_2 y_1' Y_3$ рядок		
	$b_i = i$ для парних $i$ $b_i = 15/i$ для непарних $i$	$15A_1 b_1 + c_1$	$A_2 C_2 + B_2$ $C_{ij} = 15/(i^2 + j)$
16	$x = (Y_3 y_2 + y_2)' (Y_3^2 y_2 + Y_3 y_1) + y_1' y_2$ число		
	$b_i = 16/(i^3)$	$A_1(b_1 + 16c_1)$	$A_2(B_2 + 16C_2)$ $C_{ij} = 16/(i+j)^2$
17	$x = Y_3^3 + y_2' y_1 Y_3 + y_1 y_2' + Y_3 y_2 y_1'$ матрица		
	$b_i = 17/i^2$	$A_1(17b_1 + c_1)$	$A_2(B_2 + C_2)$ $C_{ij} = 17/(2i+j)$

18	$x = y_2 y_1' + y_1 y_2' + Y_3^2 + Y_3 - Y_3 y_2 y_1'$ матриця		
	$b_i = 18/(i+18)^2$	$A_1(b_1 - c_1)$	$A_2 B_2 - A_2 C_2$ $C_{ij} = 18/(i+2j)$
19	$x = y_1' * (y_2 y_1' + Y_3^2 + y_2' y_1 Y_3) * y_2$ число		
	$b_i = 19/(i^2+1)$ для парних $i$ $b_i = 19$ для непарних $i$	$A_1(b_1 + 19c_1)$	$A_2(B_2 + C_2)$ $C_{ij} = 19/(i+2j)^3$
20	$(y_1'(y_2' y_2 Y_3) + y_2' Y_3^3 + y_2' Y_3) * Y_3 y_1$ число		
	$b_i = 20/(i^3+20)$	$A_1(20b_1 - c_1)$	$A_2 C_2 - B_2$ $C_{ij} = 20/(i^3 - j^3 + 2)$
21	$x = (Y_3^2 y_2 + Y_3 y_1 + y_2' y_2 Y_3 y_1)' * Y_3^2$ рядок		
	$b_i = 21/i^4$	$A_1(b_1 + 20c_1)$	$A_2 B_2 - C_2$ $C_{ij} = 21/(i^2 + 2j)$
22	$x = y_2 + y_1 + (y_1'(Y_3 y_1 + y_2) Y_3 + y_2 y_2') y_2$ стовпець		
	$b_i = 22i$ для парних $i$ $b_i = 22$ для непарних $i$	$A_1(b_1 - 21c_1)$	$A_2(B_2 - C_2)$ $C_{ij} = 22/(i+j)$
23	$x = Y_3 y_1 y_2' + y_2 y_1' + Y_3^2 - Y_3 y_1 y_1' Y_3$ матриця		
	$b_i = 23/i$ для парних $i$ $b_i = 23/i^2$ для непарних $i$	$A_1(b_1 + c_1)$	$A_2(23B_2 + C_2)$ $C_{ij} = 23/(3i+j)^2$
24	$x = (y_2' Y_3^2 y_1 + y_1')(Y_3 y_1 + y_2)$ число		
	$b_i = 24/(i^2+4)$ для парних $i$ $b_i = 24$ для непарних $i$	$A_1(b_1 - 24c_1)$	$A_2(B_2 + 24C_2)$ $C_{ij} = 24/(i+3j^2)$
25	$x = (y_1' Y_3 y_2 + y_2')(Y_3^3 y_1 + y_1 + y_1 y_2' Y_3 y_1)$ число		
	$b_i = 25$ для парних $i$ $b_i = 25/i^3$ для непарних $i$	$A_1(b_1 + c_1)$	$A_2(B_2 + C_2)$ $C_{ij} = 25/(i+j)^3$
26	$x = (y_1' Y_3 * y_1 + y_2')(Y_3 * y_2 + y_1 + y_1 y_2' Y_3^2 y_2)$ число		
	$b_i = 26i^3$	$A_1(26b_1 - c_1)$	$A_2(B_2 + 26C_2)$ $C_{ij} = 1/(i^2 + j)$
27	$x = y_2' y_1 Y_3^2 + y_1' y_2 Y_3^2 + y_2' Y_3 y_1 Y_3 + Y_3$ матриця		
	$b_i = 27/i^2$	$A_1(27b_1 - 26c_1)$	$A_2(27B_2 + C_2)$ $C_{ij} = 1/(i+j)^3$

28	$x=y_2'*(y_1y_2'+Y_3^3+y_1'y_2Y_3)*y_1$ число		
	$b_i=28i$	$A_1(b_1+28c_1)$	$A_2(B_2-28C_2)$ $C_{ij}=1/(i^3+j^2)$
29	$(y_1'Y_3y_1y_2y_2'+Y_3^2+y_1y_2')*(y_2'Y_3y_2y_1+y_1)$ стовпець		
	$b_i=29/i$	$A_1(29b_1+3c_1)$	$A_2(29B_2-C_2)$ $C_{ij}=1/(i+j+2)$
30	$x=y_2'(Y_3+y_1'y_1Y_3^2+y_2y_2')+y_1'(y_1'y_1Y_3+Y_3^3)$ рядок		
	$b_i=30/i$ для парних $i$ $b_i=30/i^2$ для непарних $i$	$A_1(b_1-c_1)$	$A_2(B_2+C_2)$ $C_{ij}=30/(i+j)$

### **Приклад виконання роботи.**

*Завдання:*

Вираз, який слід обчислити, заданий наступним чином:

$$x = y_2' y_1 Y_3^2 + y_1' y_2 Y_3^3 + y_2' Y_3 y_1 Y_3 + Y_3$$

При чому елементи  $y_1, y_2, Y_3$  визначаються згідно правил:

$$y_1 = A * b, \text{ де } b_i = 1/(i^2), \quad i=1,2,\dots,n$$

$$y_2 = A_1(6b_1 - c_1)$$

$$Y_3 = A_2(10B_2 + C_2), \text{ де } C_{ij} = 1/(i+j)^3$$

*Послідовність виконання.*

#### **1. Аналіз завдання.**

Для заданого виразу вхідними даними є:

розмірність матриць –  $n$ ;

матриці  $A, A_1, A_2, B_2$ ;

вектори-стовпці  $b_1, c_1$ .

Ці параметри повинні вводитися з клавіатури, або генеруватися випадковим чином (крім розмірності). При чому, елементи всіх матриць та векторів є цілими додатними числами, більшими за нуль.

Вектор-стовпець  $b$  та матриця  $C_2$  обраховуються, виходячи з уведеної розмірності, зауважимо, що значення їх елементів завжди менші одиниці і різко спадають зі збільшенням розмірності.



Наприклад для  $n=3$ , значення вектора-стовпця  $b$  будуть становити:  $b_1 = 1/(1)^2 = 1$ ;  $b_2 = 1/(2)^2 = 0.25$ ;  $b_3 = 1/(3)^2 = 0.11111$ , а значення матриці  $C_2$ , відповідно:

$$C_{2_{11}} = 1/(1+1)^3 = 1 \quad C_{2_{12}} = 1/(1+2)^3 = 1/9$$

$$C_{2_{13}} = 1/(1+3)^3 = 1/64$$

$$C_{2_{21}} = 1/(2+1)^3 = 1/9 \quad C_{2_{22}} = 1/(2+2)^3 = 1/64$$

$$C_{2_{23}} = 1/(2+3)^3 = 1/125$$

$$C_{2_{31}} = 1/(3+1)^3 = 1/64 \quad C_{2_{32}} = 1/(3+2)^3 = 1/125$$

$$C_{2_{33}} = 1/(3+3)^3 = 1/216.$$

При утворенні  $y_1$  враховуємо, що результатом множення матриці  $A$  на вектор-стовпець  $b$  є вектор-стовпець, елементи якого будуть раціональними числами (тобто матимуть значущу дробову частину).

При утворенні  $y_2$  враховуємо, що результатом віднімання двох векторів-стовпців є вектор-стовпець, елементи якого можуть бути меншими за нуль цілими числами. Далі, при множенні цілочисельної додатної матриці  $A_1$  на результат віднімання, отримаємо вектор-стовпець з цілочисельними елементами довільного знаку.

При утворенні  $Y_3$  враховуємо, що присутні лише операції додавання та множення, а тому вихідний результат завжди буде додатним і завжди матиме значущу дробову частину.

Таким чином, згідно поставленої задачі, в обчисленні загального виразу приймають участь три різні елементи – два вектори стовпці  $y_1, y_2$  та матриця  $Y_3$ .

Перший доданок загального виразу містить три множники – транспонований вектор-стовпець  $y_2$  (тобто вектор-рядок), вектор-стовпець  $y_1$  та матрицю  $Y_3$  піднесену до квадрату. Оскільки, згідно правил матричних обчислень, добуток не є комутативною операцією, всі множення слід виконувати в тій послідовності, яка задана. Результатом множення рядка на стовпець є число, а матриці на матрицю – матриця. Тому, в загальному, перший доданок буде матрицею. Аналогічний аналіз можна застосувати до всіх решти доданків даного виразу. Другий доданок повністю повторює перший, третій доданок – рядок  $\times$  матрицю = рядок, рядок  $\times$  стовпець = число, число  $\times$  матрицю = матриця.

Таким чином, з попереднього випливає, що остаточний результат є матрицею, елементи якої можуть бути як додатними так і від'ємними і завжди мають дробову частину.

## 2. Декомпозиція задачі.

Однозначно, всі обчислення безпосередньо залежать від розмірності даних, тому найперше, слід забезпечити ввід змінної  $n$ , що визначає цю розмірність. Далі, можна паралельно виконувати обчислення значень вектора  $b$  та матриці  $C_2$ , оскільки вони незалежні від інших параметрів. Крім того, на тому ж рівні декомпозиції слід визначати вхідні дані, тобто вводити з клавіатури, або генерувати випадковим чином матриці  $A, A_1, A_2, B_2$  та вектори-стовпці  $b_1, c_1$ . Наступний рівень декомпозиції – це знаходження елементів виразу. Значення  $y_1$  залежить від введеної матриці  $A$  та обрахованого вектора  $b$ . Значення  $y_2$  залежить від введеної  $A_1$  та різниці векторів  $b_1$  і  $c_1$ , тому знайти його можна лише після обчислення  $(b_1 - c_1)$ . Зауважимо, що множення на константу не є окремою операцією, як і транспонування векторів. Аналогічно, знаходимо  $Y_3$ . Подальша декомпозиція відбувається згідно заданої послідовності операцій та врахування залежностей отриманих на кожному рівні даних. Повна схема декомпозиції обчислення заданого виразу приведена нижче.

3. Об'єднання частин виразу проведено безпосередньо у схемі декомпозиції, оскільки воно однозначно визначається порядком обчислень.

4. Для написання програми, що ілюструє процес обчислення виразу згідно розробленої схеми декомпозиції, слід врахувати наступні особливості:

Дані, що вводяться з клавіатури, або генеруються випадковим чином повинні бути цілими числами, більшими за нуль. Результати проміжних обрахунків будуть містити дробову частину, виняток складає лише елемент  $y_2$ . Тому слід обирати відповідний тип даних(багатобайтний).

Оскільки на окремо взятому рівні декомпозиції обраховуються незалежні частини підвиразів, функції що їх програмно реалізують повинні викликатися псевдо-одночасно.(Тобто перш ніж виконувати будь яку функцію третього рівня, слід виконати всі функції другого рівня і т.д.)

Для того, щоб результат загального виразу був співмірний з вхідними даними, слід нормувати отримані значення, визначивши тим самим порядок результату.

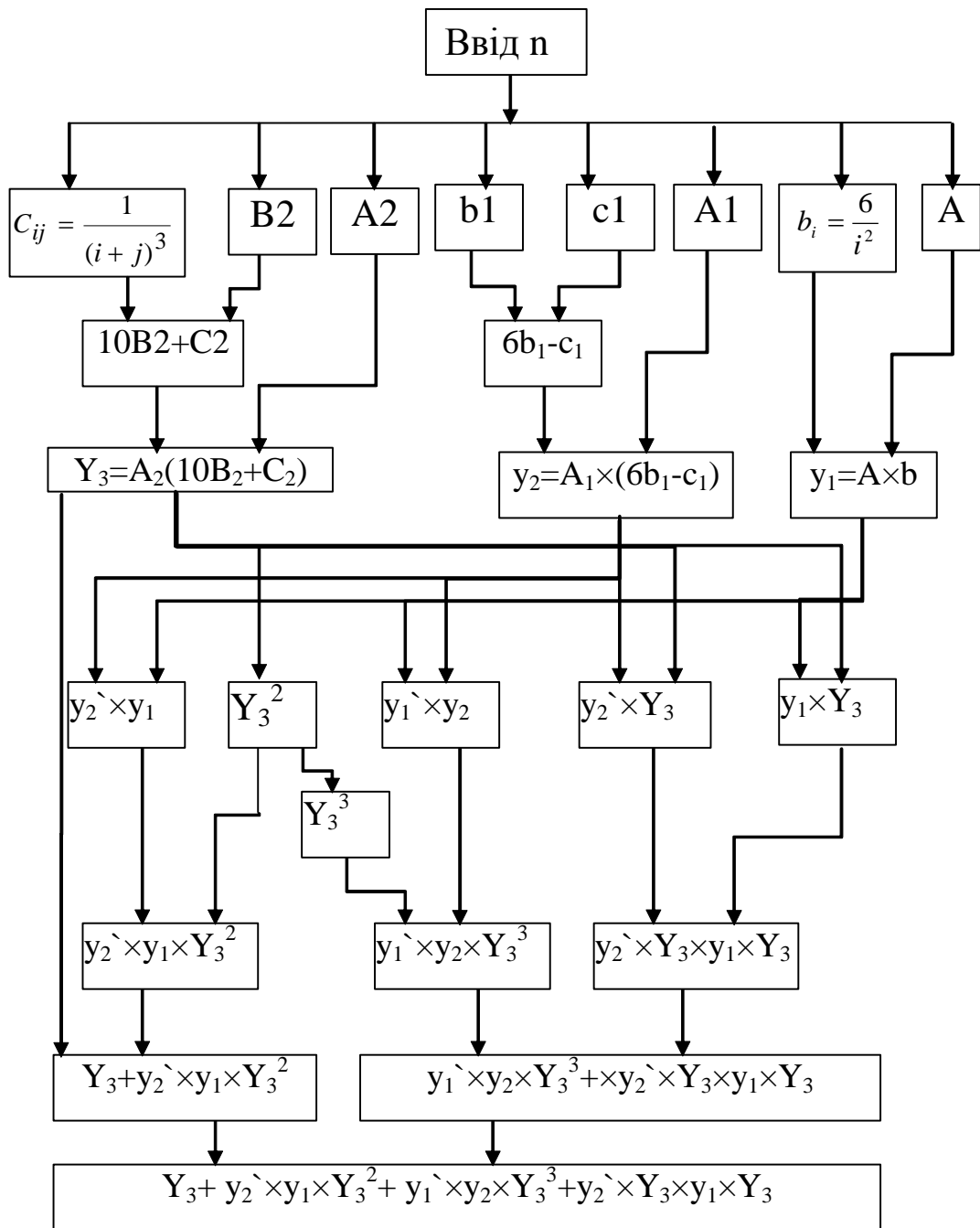


Схема декомпозиції обчислення виразу

## 5. Висновки

В роботі використано паралелізм на рівні підзадач, оскільки передбачається, що кожен блок зі схеми декомпозиції є реалізований у виді функції. Це є середньоблоковий паралелізм. Обмін даними відбувається через використання спільних змінних. Присутня залежність даних між різними рівнями декомпозиції, але в межах одного рівня її немає. Є залежність за керуванням, оскільки послідовність обчислювального процесу наперед однозначно відома.

## Лабораторна робота №5

**Тема:** Використання паралельних обчислень при розв'язанні інженерних завдань

**Мета роботи:** Дослідити можливості розв'язання різноманітних задач за допомогою паралельних алгоритмів.

### ТЕОРЕТИЧНІ ВІДОМОСТІ

#### *Паралелізм даних*

Основна ідея підходу – застосування однієї операції до кількох елементів масиву даних (векторна і паралельна обробка)

#### Основні особливості підходу

- обробкою даних керує одна програма.
- простір є глобальним (для програміста є одна єдина пам'ять а деталі структури даних доступу до пам'яті і міжпроцесорного обміну для програміста є приховані).
- слаба синхронізація обчислень на паралельних процесорах.
- паралельні операції над елементами масиву виконується на всіх допустимих даних програмі процесорах.

Найрозповсюдженішим підходом до розпаралелювання обчислень і обробки даних є підходом, що базується на моделях паралелізму даних і паралелізму задач.

В основі кожного підходу лежить розповсюдження обчислювальної роботи між доступними користувачу процесорами паралельного комп'ютера.

#### Проблеми:

- 1). Забезпечення рівномірного завантаження процесорів.
- 2). Ефективна організація обміну інформації між процесорами.

Базовою роботою операцій:

- операції керування даними.
- операції над масивами та їх фрагментами.
- умовні операції.
- операції зсуву.
- операції сканування.
- операції пересилання даних.

#### **Паралелізм задач**

Метод передбачає розбиття задачі на кілька відносно самостійних задач. Найпоширенішою структурою є MIMD структура при якій для кожного процесора створюється своя програма. Даний підхід є більш трудомісткий ніж паралелізм даних.

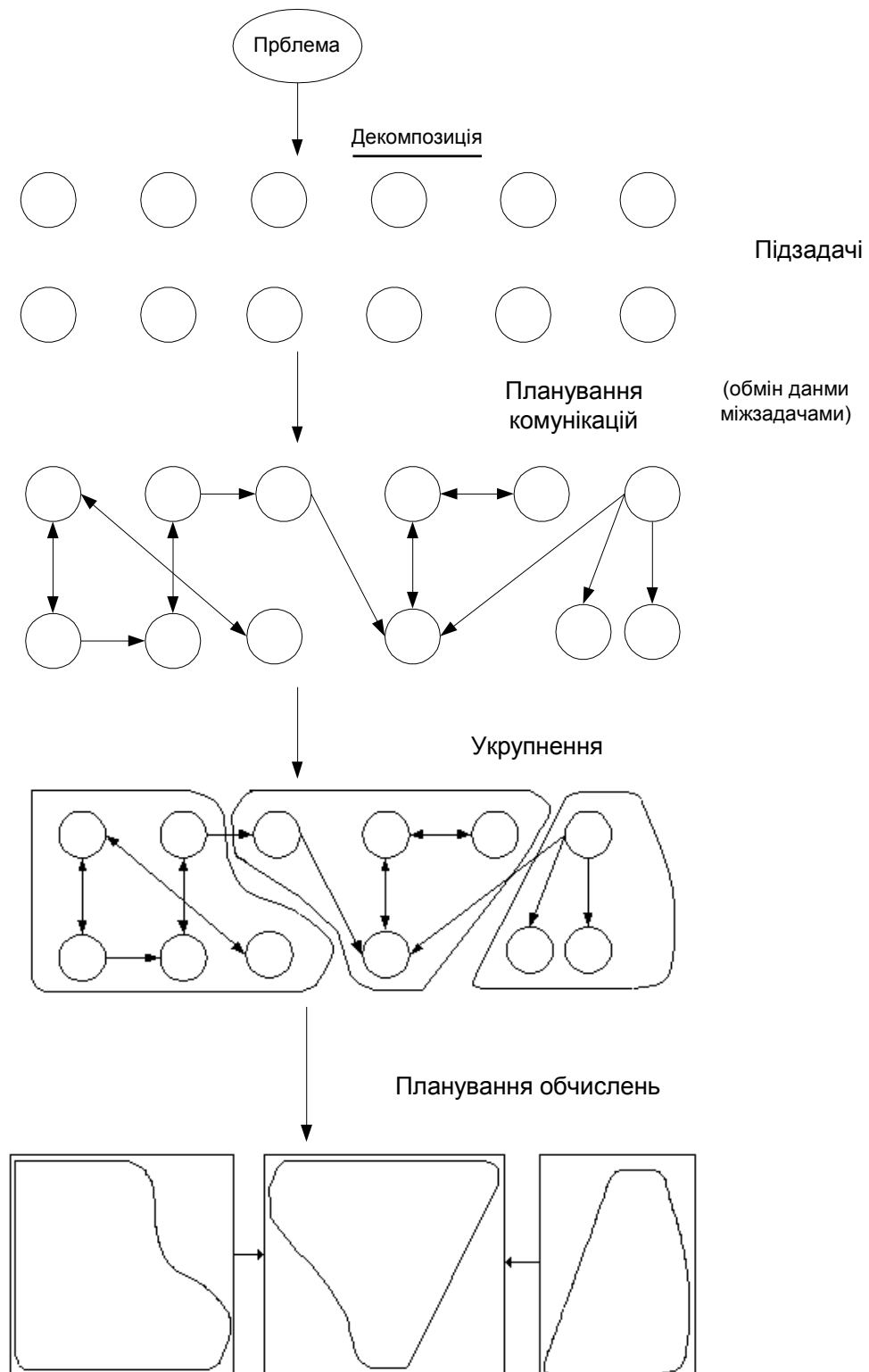
Проблеми:

1. підвищена трудоемність розробки і відлагодження програм.
2. необхідно мінімізувати обмін даними між процесорами.
3. підвищена небезпека виникнення виняткових ситуацій.

### Розробка паралельного алгоритму.

Алгоритм повинен забезпечити ефективність використати паралельної обчислювальні системи та має графічне представлення, яке складається з таких етапів:

Етапи розробки II алгоритму



## КОНТРОЛЬНІ ЗАПИТАННЯ

1. Поняття паралельного алгоритму та способи його завдання
2. Особливості застосування паралелізму задач
3. Опишіть основні особливості паралелізму даних
4. Особливості побудови незалежних гілок обчислень

## ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Проаналізувати завдання і виділити в ньому незалежні гілки обчислень. Використати метод розбиття задачі на підзадачі та виділення незалежних подій, як описано в попередній роботі.
2. Реалізувати кожен гілку окремо та привести їх графічне відображення
3. Об'єднати всі частини для вирішення поставленої задачі і переконатися у правильності реалізації.
4. Оцінити обчислювальні та часові затрати створеної програми (тобто як зростає час виконання при збільшенні розмірності задачі).
5. Зберегти результати та оформити звіт

№	Завдання
1	<p>Графом називається сукупність точок (вузлів), деякі з яких з'єднані між собою напрямленими ребрами. Граф, що складається з <math>n</math> вузлів можна описати двома матрицями порядку <math>n</math>: матрицею з'єднань та матрицею зв'язків. Елемент матриці з'єднань <math>a_{ij} = 1</math> якщо граф містить ребро направлене від вузла <math>i</math> до вузла <math>j</math> та <math>a_{ij} = 0</math> в іншому випадку. Елемент матриці зв'язків <math>b_{ij} = 1</math> якщо з вузла <math>i</math> можна потрапити у вузол <math>j</math>, рухаючись по ребрах і <math>b_{ij} = 0</math> в іншому випадку.</p> <p><u>Завдання:</u> ввести кількість вузлів деякого графу. Задавши довільним чином матрицю з'єднань(в інтерактивному режимі або випадковим чином), побудувати матрицю зв'язків для цього графу. Передбачити графічне відображення такого графу та числовий вивід обох матриць.</p>
2	<p>Лінія називається унікальною, якщо її можна накреслити не відриваючи перо від паперу та не проходячи два рази одне і теж ребро. (Зауважимо, що лінія є унікальною лише тоді, коли кількість тих вузлів з яких виходить непарна кількість ребер не більше двох). Лінію, що містить <math>n</math> вузлів можна задати квадратною матрицею з'єднань,(порядку <math>n</math>), в якій елемент <math>a_{ij} = 1</math> якщо вузол <math>i</math> з'єднаний з вузлом <math>j</math> ребром, що не містить інших вузлів.</p> <p><u>Завдання:</u> ввести кількість вузлів деякої лінії. Задавши довільним чином матрицю з'єднань(в інтерактивному режимі або випадковим чином), визначити чи є така лінія унікальною і якщо є, то отримати послідовність номерів вузлів, які будуть пройдені</p>

	для викреслювання лінії. Передбачити графічне відображення.
3	Ввести п'ять попарно різних цілих чисел $a, b, c, d, e$ . Впорядкувати їх по зростанню, використовуючи не більше 7 порівнянь. Запропонувати узагальнений алгоритм сортування таких послідовностей, зберігаючи пропорцію кількості порівнянь.
4	<p>“Ханойські вежі”. Є три стержні. На першому з них нанизано <math>m</math> дисків зростаючого вниз діаметру. Розташувати диски в тому ж порядку на іншому стержні. Диски можна переміщувати лише по одному, не можна класти більший диск на менший.</p> <p><u>Завдання:</u> запропонувати алгоритм вирішення поставленої задачі, забезпечивши ввід кількості дисків та відображення їх переміщення в процесі перестановки.</p>
5	Запропонувати та відобразити алгоритм реалізації гри “LIFE”. Гра моделює життя деякої колонії живих клітин, які виживають, розмножуються або гинуть згідно наступних умов. Клітина виживає, якщо вона має двох або трьох сусідів з восьми можливих. Якщо у клітини лише один сусід, або немає жодного, то вона гине (від ізоляції). Якщо клітина має більше трьох сусідів, то вона гине (від перенаселення). В будь-якій порожній позиції, яка має рівно трьох сусідів у наступному поколінні з'являється нова клітина. Гра повинна починатися з довільної кількості клітин, що розташовані в ігровому полі випадковим чином (інтерактивний режим задавання вхідних даних) – так звана початкова популяція. Програма повинна коректно завершувати роботу у таких випадках: а).загинула вся популяція, б) на вимогу користувача.
6	Задати (ввести з клавіатури, <u>або</u> генерувати випадковим чином) цілочисельну матрицю $A$ розмірності $n*m$ , кожен елемент якої дорівнює 0,1,2, або 3. Визначити кількість четвірок $a_{ij}, a_{i+1,j}, a_{i,j+1}, a_{i+1,j+1}$ в яких всі елементи різні.
7	Елемент матриці називається сідловою точкою, якщо він є одночасно найменшим у рядку та найбільшим у стовпці. Задати (ввести з клавіатури, <u>або</u> генерувати випадковим чином) дійсну матрицю розміру $n*m$ (вводиться з клавіатури). Визначити чи є в ній сідлові точки та вказати індекси першої та останньої з них.
8	В полі $8*8$ кліток зображено кілька прямокутників, кожен з яких складається з кліток, різні прямокутники не перетинаються і не доторкаються один до одного. Задана квадратна матриця порядку 8, в якій елемент рівний нулю, якщо відповідна клітина належить прямокутнику і відмінний від нуля, в іншому випадку. Визначити кількість прямокутників. Початковими даними вважати матрицю елементів, яка повинна вводитися під час виконання програми. Графічно відобразити вхідні дані.

9	<p>Задати натуральні числа <math>m, a_1, \dots, a_n</math>. В послідовності <math>a_1, \dots, a_n</math> вибрати підпослідовність <math>a_{i_1}, a_{i_2}, \dots, a_{i_k}</math> (<math>0 \leq i_1 &lt; i_2 &lt; \dots &lt; i_k \leq n</math>) таку, що <math>a_{i_1} + a_{i_2} + \dots + a_{i_k} = m</math>. Якщо таку підпослідовність вибрати неможливо, то слід вивести відповідне повідомлення.</p> <p>(При вирішенні даної задачі слід використати наступне міркування. Для того, щоб вибрати необхідну підпослідовність для кожного елемента вхідної послідовності слід визначити чи приймається він у шукану підпослідовність. Може виникнути наступна ситуація: відносно членів <math>a_1, \dots, a_i</math> (<math>i &lt; n</math>) прийняті які-небудь рішення, після чого виявилось, що, як би ми не розпоряджались іншими <math>n - i</math> членами, нам все одно не вдасться отримати підпослідовність, що задовольняє поставленій умові (наприклад, якщо сума декількох додатних чисел більше <math>m</math>, то неможливо додати до них ще декілька додатних чисел, так, щоб сума стала рівна <math>m</math>). В цьому випадку слід виключити з розгляду всі підпослідовності, перші елементи яких вибрані з <math>a_1, \dots, a_i</math>, згідно з прийнятими рішеннями.)</p>
10	<p>Задати натуральне число <math>m</math>. Отримати <math>m</math> розташувань 8 ферзів на шаховій дошці при яких жоден з ферзів не загрожує іншим (тобто жодні два ферзі не знаходяться на одній горизонталі, на одній вертикалі, на одній діагоналі). Якщо <math>m</math> більше ніж загальна кількість таких конфігурацій, то слід знайти всі комбінації.</p>
11	<p>Лабіринт задається матрицею з'єднань(див.1) в якій для кожної пари кімнат вказано чи з'єднані вони коридором. Задати (ввести з клавіатури, <u>або</u> генерувати випадковим чином): кількість кімнат лабіринту (<math>n</math>); матрицю з'єднань для них; номери кімнат <math>i</math> та <math>j</math> (<math>1 \leq i, j \leq n</math>). Побудувати шлях з кімнати з номером <math>i</math> в кімнату з номером <math>j</math>.</p>
12	<p>Задати (ввести з клавіатури, <u>або</u> генерувати випадковим чином) матрицю з'єднань деякої лінії, що містить 6 вузлів. З'ясувати чи існує замкнений шлях, що складається з декількох відрізків лінії, який проходить через кожні 6 вузлів рівно один раз. Якщо такий шлях існує, то побудувати відповідну послідовність номерів вузлів. Вивести лінію графічно.</p>
13	<p>Є деяка кількість міст, деякі з яких з'єднані дорогами відомої довжини. Вся система доріг описується квадратною матрицею порядку <math>n</math>, в якій елемент <math>a_{ij}</math> рівний будь-якому від'ємному числу, якщо місто <math>i</math> безпосередньо не з'єднано з містом <math>j</math> та рівний довжині дороги у протилежному випадку. Знайти для 1-го міста найкоротші дороги в інші міста. Вхідними даними вважати кількість міст та матрицю доріг.</p>
14	<p>Для попередньої задачі припустити, що кожне місто має пряму дорогу до кожного іншого, і знайти найкоротший шлях, що проходить через всі міста.</p>
15	<p>Задати (ввести з клавіатури, <u>або</u> генерувати випадковим чином) цілочисельну матрицю <math>A</math> розмірності <math>n \times m</math>, кожен елемент якої дорівнює 0, 1, 2, або 3. Визначити кількість четвірок <math>a_{ij}, a_{i+1,j}, a_{i,j+1}, a_{i+1,j+1}</math> в яких всі елементи однакові.</p>



## СПИСОК ВИКОРИСТАНОЇ ТА РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Аксак Н.Г. Паралельні та розподілені обчислення: підруч./ НГ.Аксак, О.Г. Руденко, А.М.Гуржій. – Х.:Компанія СМІТ, 2009. – 480с.
2. Богачев К.Ю. Основы параллельного программирования. /Богачев К.Ю. – М.: БИНОМ. Лаборатория знаний, 2003.
3. Ваврук Є, О. Акимішин Технологія паралельного програмування засобами MPI // Методичні вказівки до лабораторної роботи з курсу “Паралельні та розподілені обчислення” Вид-во Нац. ун-ту “Львівська політехніка”, 2010 р. – 10с.
4. Ваврук Є., О.Лашко Використання функціональної декомпозиції для розв’язання обчислювальних задач// Методичні вказівки до лабораторної роботи з курсу “Паралельні та розподілені обчислення” Вид-во Нац. ун-ту “Львівська політехніка”, 2010 р. –12с.
5. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. /еводин В.В. – СПб.: БХВ-Петербург, 2002.
6. Гергель В.П. Теория и практика параллельных вычислений/Гергель В.П. – М.:ИНТУИР.РУ Интернет-Университет Информационных технологий, 2007.
7. Рихтер Дж. Windows для профессионалов: создание эффективных Win32 приложений с учетом специфики 64-разрядной версии Windows/Рихтер Дж. – СПб.: Питер, 2001. – 752 с.
8. Таненбаум Э. Распределенные системы. Принципы и парадигмы – Спб.: Питер, 2003. – 877 с.
9. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования/Эндрюс Г.Р. – М.: «Вильямс», 2003. – 512 с.