Міністерство освіти і науки України
Національний університет водного господарства та
природокористування

Кафедра прикладної математики

04-01-31

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисциплін ,,Оптимізація обчислень”,
,,Паралельні та розподілені обчислення”
для студентів спеціальностей
113,,Прикладна математика”,
121 ,,Інженерія програмного забезпечення”,
122 ,,Комп’ютерні науки”

Затверджено
науково-методичною
комісією зі спеціальності
122 ,,Комп’ютерні науки”
протокол № 3
від 02 листопада 2017р.

Рівне 2017

Методичні вказівки до виконання лабораторних робіт з дисциплін ,,Оптимізація обчислень", ,,Паралельні та розподілені обчислення" для студентів спеціальностей 113 ,,Прикладна математика", 121 ,,Інженерія програмного забезпечення", 122 ,,Комп'ютерні науки"/ Жуковський В.В., Жуковська Н.А, Харів Н.О. – Рівне : НУВГП, 2017. – 54 с.

Упорядники: Жуковський В.В. – ст. викладач кафедри прикладної математики,

Жуковська Н.А. – кандидат технічних наук, доцент кафедри прикладної математики,

Харів Н.О. – ст. викладач кафедри прикладної математики

Відповідальний за випуск: Мартинюк П.М., доктор технічних наук, доцент, завідувач кафедри прикладної математики

Зміст

Вступ

Методичні вказівки розроблені відповідно до навчальних і робочих програм. Структурно вони містять 4 лабораторні роботи, у кожній з яких надано теоретичний матеріал, варіанти завдань, пояснення до виконання і приклади виконання. Головна мета даних вказівок: набуття студентами практичних навичок у аналізі, розробці та реалізації алгоритмів паралельних та розподілених обчислень.

Лабораторна робота № 1

Тема: Робота з потоками за допомогою засобів мови програмування C#.

Мета: Вивчити основні оператори, класи та їх методи доступні у фреймворку Microsoft.Net для роботи з потоками.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Програма – це деяка послідовність машинних команд, що зберігається на диску, в разі необхідності завантажується у пам'ять і виконується. Можна сказати, що під час виконання програму представляє процес. Однозначна відповідність між програмою і процесом встановлюється тільки в конкретний момент часу: один процес у різний час може виконувати код декількох програм, код однієї програми можуть виконувати декілька процесів одночасно. Для успішного виконання програми потрібні певні ресурси. До них належать:

- ресурси, необхідні для послідовного виконання програмного коду (передусім процесорний час);
- ресурси, що дають можливість зберігати інформацію, яка забезпечує виконання програмного коду (регістри процесора, оперативна пам'ять тощо).

Ці групи ресурсів визначають дві складові частини процесу:

- послідовність виконуваних команд процесора;
- набір адрес пам'яті (адресний простір), у якому розташовані ці команди і дані для них.

Виділення цих частин виправдане ще й тим, що в рамках одного адресного простору може бути кілька паралельно виконуваних послідовностей команд, що спільно використовують одні й ті ж самі дані. Необхідність розмежування послідовності команд і адресного простору підводить до поняття потоку. ***Потоком*** (потік керування, нитка, thread) називають набір послідовно виконуваних команд процесора, які використовують загальний адресний простір процесу. Оскільки в системі може одночасно бути багато

потоків, завданням ОС є організація перемикання процесора між ними і планування їхнього виконання. У багатопроцесорних системах код окремих потоків може виконуватися на окремих процесорах. Тепер можна дати ще одне означення процесу. ***Процесом*** називають сукупність одного або декількох потоків і захищеного адресного простору, у якому ці потоки виконуються.

C# підтримує паралельне виконання коду через багатопоточність. Програма на C# запускається як процес з єдиним потоком, що автоматично створюється CLR і операційною системою („головний" потік), і стає багатопотоковою програмою за допомогою створення додаткових потоків. Ось простий приклад і його результат:

```
using System;
using System.Threading; // for threads
class ThreadTest
{
  static void Main()
  {
    Thread t = new Thread(WriteY);
    t.Start(); // Виконати WriteY в новому потоці
    while (true)
      Console.Write("x"); // Друкувати 'x'…
  }

  static void WriteY()
  {
    while (true)
      Console.Write("y"); // Друкувати 'y'…
  }
}
```

Результат

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
xxxxxxxxxxxxxxxxxxxxxxxyyy
yyyyyyyyyyyyyxxxxyyyyyyyyyyyyyyyyyyyyxxxxxxxxxxxxxyyyyyy
yyyyyyyyyxxxxxxxxxxxxxxxx
```

```
xxxxxxxxxyyyyyyyyyyyyyyyyyyyyyxxxxxxxxxxxxxxxxxxxyyyy
yyyyyyyyyyyyyyyxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyxxxxxxxxxxx
xxxxxxxxyyyyyyyyyyyyyyyyx
xxxxxxxxxxxxxxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyyyx
xxxxxxxxxxxxxxxxxxxxyyyy
yyyyyyyyyyyyxxxxy...
```

У головному потоці створюється новий потік t, що виконує метод, який безперервно друкує символ 'y'. Одночасно головний потік безперервно друкує символ 'x'.

ЗАВДАННЯ

1) Створіть проект консольного додатку в середовищі Microsoft Visual Studio C#.

2) Виконайте вищенаведений приклад зі створення програми з найпростішими потоками.

3) Протестуйте наступні приклади програм та проаналізуйте їх. Результати виконання та їх словесний аналіз запишіть у звіт.

УВАГА: кожну програму необхідно запускати декілька разів, для перевірки та аналізу роботи різних потоків!

ПРИКЛАДИ ПРОГРАМ ДЛЯ АНАЛІЗУ

CLR призначає кожному потоку свій стек, так що локальні змінні зберігаються окремо. У наступному прикладі визначається метод з локальною змінною, який потім виконується одночасно в головному і в новоствореному потоках:

```
static void Main()
{
  new Thread(Go).Start();
  Go();
}

static void Go()
{
  for (int cycles = 0; cycles < 5; cycles++)
```

```
    Console.Write('P');
  }
```

Разом з тим потоки розділяють дані, що відносяться до того ж екземпляру об’єкта, що і самі потоки:

```
class ThreadTest
{
  bool done;

  static void Main()
  {
    ThreadTest tt = new ThreadTest();
    new Thread(tt.Go).Start();
    tt.Go();
  }

  void Go()
  {
    if (!done) { done = true; Console.WriteLine("Done"); }
  }
}
```


Для статичних полів працює інший спосіб розподілу даних між потоками. Ось ще один приклад, але зі статичним полем:

```
class ThreadTest
{
  static bool done;    // Статичне поле, розділене потоками

  static void Main()
  {
    new Thread(Go).Start();
    Go();
  }

  static void Go()
  {
    if (!done) { done = true; Console.WriteLine("Done"); }
  }
}
```

4) Замініть останній метод на наступний та проаналізуйте роботу:

```
static void Go()
{
  if (!done)
  {
    Console.WriteLine("Done");
    done = true;
  }
}
```

C# забезпечує ексклюзивне блокування це за допомогою оператора **`lock`**:

```
class ThreadSafe
{
  static bool done;
  static object locker = new object();

  static void Main()
  {
    new Thread(Go).Start();
    Go();
  }

  static void Go()
  {
    lock (locker)
    {
      if (!done)
      {
        Console.WriteLine("Done");
        done = true;
      }
    }
  }
}
```


Коли два потоки одночасно змагаються за блокування (у нашому випадку об'єкт **locker**), один потік переходить до очікування (блокується), поки блокування не звільняється. У

цьому випадку це гарантує, що тільки один потік може одночасно виконувати критичну секцію коду.

Код, захищений таким чином від невизначеності в плані багатопоточного виконання, називається ***потокобезпечним***.

Тимчасова призупинка (блокування) – основна методика координації або синхронізації дій потоків.

5) Складіть звіт про виконану роботу:

ЗМІСТ ЗВІТУ

1. Тема, мета.
2. Текст програми з **коментарями незрозумілих частин**, результат її роботи та словесний аналіз/пояснення консольного виводу.
3. Висновки.

Лабораторна робота № 2

Тема: Використання функціональної декомпозиції для розв'язку обчислювальних задач.

Мета: Вивчити методи декомпозицій задач. Набути навиків розв'язування задач з використанням функціональної 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 виграш від розпаралелення може бути незначним, а трудоємкість 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 підзадач. Обираються методи та алгоритми комунікацій.

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

4. ***Планування обчислень***. На цьому кроці виконується розподіл підзадач між процесорами. Основний критерій вибору

способу розміщення підзадач – 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 випливає 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, причому 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 підзадач звертаються до одного файлу(чи змінної).

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

ЗАВДАННЯ

Використовуючи метод функціональної декомпозиції, розробити алгоритм обчислення запропонованого матрично-векторного виразу, який би враховував можливість паралельного виконання і був оптимальним з точки зору часових затрат.

На основі створеного алгоритму написати програму, яка дозволяє обчислити вираз та ілюструє проведену декомпозицію.

ПОРЯДОК ВИКОНАННЯ

1) Проаналізуйте наведені нижче правила обрахунку елементів виразу.

2) Проведіть декомпозицію задачі, виходячи з можливості паралельного виконання окремих підзадач.

3) Об'єднайте отримані проміжні результати і обчисліть кінцевий вираз.

4) Напишіть і продемонструйте програму обчислення виразу.

Окрім безпосередніх обчислень, програма повинна мати інтерфейс користувача, який забезпечує:

- ввід (з клавіатури) розмірності даних (n);
- можливість вибору – ввід даних (тобто елементів матриці та векторів) з клавіатури чи генерування їх випадковим чином;
- вивід на екран (або у файл) проміжних результатів за потребою користувача;
- обов'язковий вивід остаточних результатів на екран і у файл у зрозумілому вигляді.

Зауваження

Всі вхідні дані є цілими числами, більшими за нуль.

Необхідно знайти такі коефіцієнти K_1, K_2, які б забезпечили нормалізацію результатів (тобто пониження чи підвищення їх порядку).

5) Визначте паралелізм якого рівня присутній в алгоритмі та зробіть висновки щодо залежностей даних, керування, ресурсів, вводу/виводу.

6) Складіть звіт про виконану роботу:

ЗМІСТ ЗВІТУ

1. Тема, мета, завдання (варіант).
2. Схема декомпозиції задачі.
3. Текст програми та результат її роботи на довільному наборі вхідних даних.
4. Висновки.

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 згідно варіанту.

ВАРІАНТИ ЗАВДАНЬ

№	Для знаходження значень		
	b	y_2	Y_3
1	$x = Y_3^2 y_2 + Y_3(y_1 + K_1 * y_2)$		
	b_i=1/(i^2+2) для парних i b_i=1/i для непарних i i=1, 2, ..., n	$A_1(b_1+c_1)$	$A_2(B_2-C_2)$ C_{ij}=1/(i+2j)
2	$x = K_1 * Y_3^2 y_2 + Y_3 * K_1 * (y_1 + y_2) + y_1 y_2' y_2 + K_2 * Y_3^3 y_1$		
	b_i=1/(i^2+2+i) для парних i b_i=1/i для непарних i i=1, 2, ..., n	$A_1(b_1+2c_1)$	$A_2(C_2-B_2)$ C_{ij}=1/(i+j)
3	$x = K_1 Y_3 y_2 y_2' + Y_3^3 - Y_3 + y_2 y_1' + K_2 Y_3^2 y_1'$		
	b_i=3/(i^2+3) для парних i b_i=3/i для непарних i i=1, 2, ..., n	$A_1(3b_1+c_1)$	$A_2(B_2-C_2)$ C_{ij}=1/(i+j)2
4	$x = K_1 Y_3 y_2 + Y_3^3 - Y_3 y_1^2 + y_1 y_2 + K_2 Y_3^2 y_1$		
	b_i=4/(i^3+3) i=1, 2, ..., n	$A_1(b_1+4c_1)$	$A_2(B_2+C_2)$ C_{ij}=1/(i+j^2)
5	$x = K_1(y_1' Y_3 * y_1 + y_2') * (Y_3 * y_2 + y_1 + K_2 y_1 y_2' Y_3^2 y_2)$		
	b_i=5i^3 i=1, 2, ..., n	$A_1(5b_1-c_1)$	$A_2(B_2+10C_2)$ C_{ij}=1/(i^2+j)

6	$x = K_1 y_2' y_1 Y_3^2 + K_2 y_1' y_2 Y_2^3 + K_2 y_2' Y_3 y_1 Y_3 + Y_3$		
	b_i=6/i^2 i=1, 2, ..., n	$A_1(6b_1\text{-}c_1)$	$A_2(10B_2+C_2)$ $C_{ij}=1/(i+j)^3$
7	$x = y_2' K_1 (y_1 y_2' + K_2 Y_3^3 + y_1' Y_3) y_1$		
	y_1=A*b b_i=7i i=1, 2, ..., n	$y_2=A_1(b_1+c_1)$	$Y_3=A_2(B_2\text{-}C_2)$ $C_{ij}=1/(i^3+j^2)$
8	$(K_1 y_1' Y_3 y_1 y_2 y_2' + K_2 Y_3^2 + y_1 y_2') K_1 (y_2' Y_3 y_2 y_1 + y_1)$		
	b_i=8/i i=1, 2, ..., n	$A_1(2b_1+3c_1)$	$A_2(B_2\text{-}C_2)$ $C_{ij}=1/(i+j+2)$
9	$x = K_1 y_2' (Y_3 + y_1' y_1 Y_3^2 + y_2 y_2') + y_1' K_1 (y_1' y_1 Y_3 + K_2 Y_3^3)$		
	b_i=9i i=1, 2, ..., n	$A_1(b_1\text{-}c_1)$	$A_2(B_2+C_2)$ $C_{ij}=1/(i+j)$
10	$x = K_1 \left(y_2' (y_1' y_1 Y_3) + y_1' Y_3^3 + y_2' Y_3\right) * K_2 (Y_3 y_1 + y_2' y_2 y_1)$		
	b_i=10/(i^2+1) i=1, 2, ..., n	$A_1(b_1+c_1)$	$A_2(C_2+2B_2)$ $C_{ij}=1/(i+2j)$
11	$x = K_1 y_2 y_1' + y_2' K_1 y_2 Y_3 + y_1' K_2 Y_3^2 y_2 + Y_3 + K_2 Y_3 y_1 y_1' Y_3$		
	b_i=11i^2 для парних i b_i=11/i для непарних i i=1, 2, ..., n	$A_1(b_1\text{-}2c_1)$	$A_2(B_2\text{-}2C_2)$ $C_{ij}=1/(i^2+j)$
12	$x = y_2' + K_1 \left((y_1' K_2 (Y_3^2 y_1 + y_2) Y_3 + y_1 y') y_2\right)$		
	b_i=i^2/12 для парних i b_i=i для непарних i i=1, 2, ..., n	$A_1(12b_1\text{-}c_1)$	$A_2(B_2\text{-}C_2)$ $C_{ij}=1/(i+j^2)$

y' означає операцію транспонування.

K_1, K_2 – сталі коефіцієнти, значення яких обирається від порядку результату.

ПРИКЛАД ВИКОНАННЯ ЗАВДАННЯ

Варіант завдання

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)

При чому:

y' означає операцію транспонування; i, j=1, 2, …, n (n – вхідна розмірність).

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. 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=4, значення вектора-стовпця b будуть становити:

$$b = \begin{pmatrix} 7 \\ 3.5 \\ 3.148 \\ 3.063 \end{pmatrix}$$

а значення матриці C_2, відповідно:

$$C_2 = \begin{pmatrix} 0.5 & 0.2 & 0.1 & 0.059 \\ 0.333 & 0.167 & 0.091 & 0.056 \\ 0.25 & 0.143 & 0.083 & 0.053 \\ 0.2 & 0.125 & 0.077 & 0.05 \end{pmatrix}$$

Оскільки результатом множення матриці A на вектор-стовпець b є вектор-стовпець, а результатом віднімання/додавання двох векторів-стовпців є вектор-стовпець і т. д., то було вирішено, всі операції проводити над матрицями, а стовпці і рядки привести до матриць, заповнивши пусті елементи нулями. Кінцевий результат не зміниться в такому випадку. Таким чином, вектор b набув такого вигляду:

$$b = \begin{pmatrix} 7 & 0 & 0 & 0 \\ 3.5 & 0 & 0 & 0 \\ 3.148 & 0 & 0 & 0 \\ 3.063 & 0 & 0 & 0 \end{pmatrix}$$

Але існує один момент, при якому така заміна не допустима. Це виникає при множенні числа на матрицю.

Приклад:

$$y_2 = \begin{pmatrix} 640 \\ 1.536\times10^3 \\ 2.432\times10^3 \\ 3.328\times10^3 \end{pmatrix};\quad y_1 = \begin{pmatrix} 16.711 \\ 16.711 \\ 16.711 \\ 16.711 \end{pmatrix};\quad Y_3 \cdot y_1 = \begin{pmatrix} 3.653\times10^4 \\ 3.653\times10^4 \\ 3.653\times10^4 \\ 3.653\times10^4 \end{pmatrix};$$

$$y_2' \cdot y_1 = \left(1.326\times10^5\right)$$

$$\begin{pmatrix} 1.235\times10^4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot Y_3 \cdot y_1 = \begin{pmatrix} 4.511\times10^8 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

але

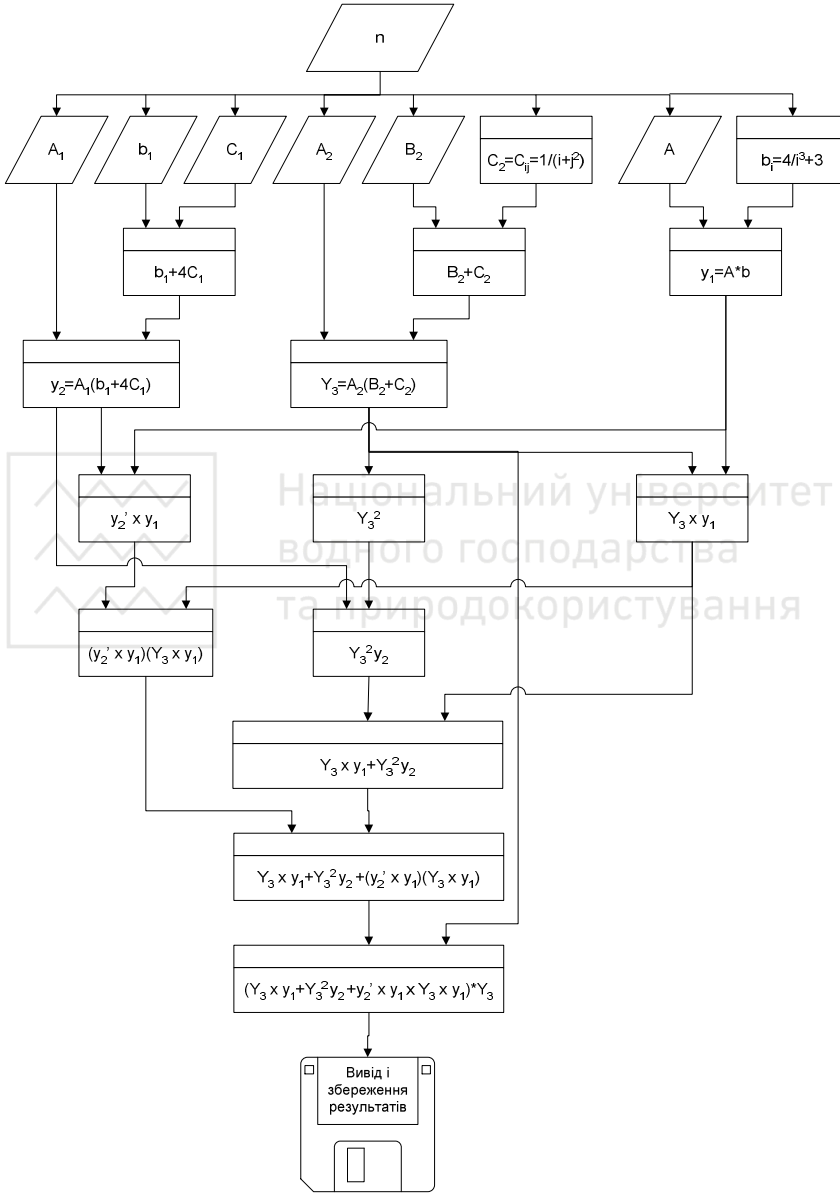
$$1.235\times10^4 \cdot \left(Y_3 \cdot y_1\right) = \begin{pmatrix} 4.511\times10^8 \\ 4.511\times10^8 \\ 4.511\times10^8 \\ 4.511\times10^8 \end{pmatrix}.$$

Тому в цьому випадку потрібно бути уважним. Так як, операція множення на константу не є окремою операцією, як і транспонування векторів, то її можна виконати окремо, без декомпозиції.

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 визначити 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 $(b_1\text{-}c_1)$. 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.

Схема декомпозиції обчислення виразу



Проектування комунікацій

При вирішенні цієї задачі було вибрано модель „сервер-клієнт”, де сервер виконує головну роль: отримує вхідні дані, посилає завдання на клієнти, зберігає проміжні дані, формує відповідь і т.д. Клієнт є простою програмою, яка отримує на вході дві матриці і операцію, яку необхідно виконати над ними.

Клієнт і сервер обмінюються інформацією через сокети і підтримують роботу в мережі. Це дозволяє досягти реального паралельного вирішення задачі.

Протокол обміну повідомленнями має вигляд:

(протокол)(відправник)(команда)(параметри команди)

де

(протокол) – це назва „ПРО”, що дозволяє відрізняти „свої” пакети від „чужих”;

(відправник) – сервер або клієнт(його ай пі + порт);

(команда) – множення, ділення, повернення результату і т.д.;

(параметри) – параметри команди, матриці і т.д.

Фрагменти тексту програми

zmatrix.h – бібліотека для роботи з матрицями

```
#define MAT_TEMPLATE  template <class T>

#  include <iostream.h>
#  include <string.h>


#  define max(a,b)    (((a) > (b)) ? (a) : (b))
#  define min(a,b)    (((a) < (b)) ? (a) : (b))


#if !defined(_NO_TEMPLATE)
#  define MAT_TEMPLATE  template <class T>
#  define matrixT  matrix<T>
#else
#  define MAT_TEMPLATE
#  define matrixT  matrix
```

```
#  ifdef MATRIX_TYPE
    typedef MATRIX_TYPE T;
#  else
    typedef double T;
#  endif
#endif


//template <class T>
MAT_TEMPLATE
class matrix
{
public:
  // Constructors
  matrix (size_t row = 6, size_t col = 6);

  // Destructor
  ~matrix ();

  // Assignment operators
  matrixT& operator = (const matrixT& m);// _NO_THROW;

  // Value extraction method
  size_t RowNo () const { return _m->Row; }
  size_t ColNo () const { return _m->Col; }

  // Combined assignment - calculation operators
  matrixT& operator += (const matrixT& m);//
_THROW_MATRIX_ERROR;
  matrixT& operator -= (const matrixT& m);//
_THROW_MATRIX_ERROR;
  matrixT& operator *= (const matrixT&
m) ;//_THROW_MATRIX_ERROR;
  matrixT& operator *= (const T& c) ;//_NO_THROW;
  matrixT& operator /= (const T& c) ;//_NO_THROW;
  matrixT& operator ^= (const size_t& pow);//
_THROW_MATRIX_ERROR;

  void SetSize (size_t row, size_t col);// _NO_THROW;
  void SetElement (size_t row, size_t col,T value);
```

```
private:
   struct base_mat
   {
         T **Val;
         size_t Row, Col, RowSiz, ColSiz;
         int Refcnt;

         base_mat (size_t row, size_t col, T** v)
         {
            Row = row; RowSiz = row;
            Col = col; ColSiz = col;
            Refcnt = 1;

            Val = new T* [row];
            size_t rowlen = col * sizeof(T);

            for (size_t i=0; i < row; i++)
            {
                  Val[i] = new T [col];
                  if (v) memcpy( Val[i], v[i], rowlen);
            }
         }
         ~base_mat ()
         {
            for (size_t i=0; i < RowSiz; i++)
                  delete [] Val[i];
            delete [] Val;
         }
   };
   base_mat *_m;

   void clone ();
   void realloc (size_t row, size_t col);
   int pivot (size_t row);
};

#if defined(_MSC_VER) && _MSC_VER <= 1020
#  undef  _NO_THROW              // MSVC++ 4.0/4.2 does not
support
#  undef  _THROW_MATRIX_ERROR    // exception specification in
```

```
definition
#  define _NO_THROW
#  define _THROW_MATRIX_ERROR
#endif


// constructor
MAT_TEMPLATE inline
matrixT::matrix (size_t row, size_t col)
{
  _m = new base_mat( row, col, 0);
}

// combined power and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator ^= (const size_t& pow)
//_THROW_MATRIX_ERROR
{
        matrixT temp(*this);

        for (size_t i=2; i <= pow; i++)
    *this = *this * temp;

        return *this;
}


// binary addition operator
MAT_TEMPLATE inline matrixT
operator + (const matrixT& m1, const matrixT& m2)
//HROW_MATRIX_ERROR
{
   matrixT temp = m1;
   temp += m2;
   return temp;
}

// binary subtraction operator
MAT_TEMPLATE inline matrixT
operator - (const matrixT& m1, const matrixT& m2)
```

```
//HROW_MATRIX_ERROR
{
  matrixT temp = m1;
  temp -= m2;
  return temp;
}


// binary scalar multiplication operator
MAT_TEMPLATE inline matrixT
operator * (const matrixT& m, const T& no) //O_THROW
{
  matrixT temp = m;
  temp *= no;
  return temp;
}

// binary scalar multiplication operator
MAT_TEMPLATE inline matrixT
operator * (const T& no, const matrixT& m) //O_THROW
{
  return (m * no);
}


// binary matrix multiplication operator
MAT_TEMPLATE inline matrixT
operator * (const matrixT& m1, const matrixT& m2)
//HROW_MATRIX_ERROR
{
  matrixT temp = m1;
  temp *= m2;
  return temp;
}


// binary scalar division operator
MAT_TEMPLATE inline matrixT
operator / (const matrixT& m, const T& no) //O_THROW
```

```
{
   return (m * (T(1) / no));
}

// binary scalar division operator
MAT_TEMPLATE inline matrixT
operator / (const T& no, const matrixT& m)
//_THROW_MATRIX_ERROR
{
   return (!m * no);
}

// binary matrix division operator
MAT_TEMPLATE inline matrixT
operator / (const matrixT& m1, const matrixT& m2)
//_THROW_MATRIX_ERROR
{
   return (m1 * !m2);
}

// binary power operator
MAT_TEMPLATE inline matrixT
operator ^ (const matrixT& m, const size_t& pow)
//_THROW_MATRIX_ERROR
{
  matrixT temp = m;
  temp ^= pow;
  return temp;
}

// unary transpose operator
MAT_TEMPLATE inline matrixT
operator ~ (const matrixT& m) //_NO_THROW
{
  matrixT temp(m.ColNo(),m.RowNo());

  for (size_t i=0; i < m.RowNo(); i++)
    for (size_t j=0; j < m.ColNo(); j++)
    {
      T x = m(i,j);
```

```
                temp(j,i) = x;
        }
    return temp;
}

// unary inversion operator
MAT_TEMPLATE inline matrixT
operator ! (const matrixT m) //_THROW_MATRIX_ERROR
{
    matrixT temp = m;
    return temp.Inv();
}

// inversion function
/**MAT_TEMPLATE inline matrixT
matrixT::Inv () //_THROW_MATRIX_ERROR
{
    size_t i,j,k;
    T a1,a2,*rowptr;

    if (_m->Row != _m->Col)
        REPORT_ERROR( "matrixT::operator!: Inversion of a non-
square matrix");

    matrixT temp(_m->Row,_m->Col);
    if (_m->Refcnt > 1) clone();


    temp.Unit();
    for (k=0; k < _m->Row; k++)
    {
        int indx = pivot(k);
        if (indx == -1)
                REPORT_ERROR( "matrixT::operator!: Inversion of a
singular matrix");

        if (indx != 0)
        {
                rowptr = temp._m->Val[k];
                temp._m->Val[k] = temp._m->Val[indx];
```

```
                temp._m->Val[indx] = rowptr;
        }
        a1 = _m->Val[k][k];
        for (j=0; j < _m->Row; j++)
        {
                _m->Val[k][j] /= a1;
                temp._m->Val[k][j] /= a1;
        }
        for (i=0; i < _m->Row; i++)
                if (i != k)
                {
                  a2 = _m->Val[i][k];
                  for (j=0; j < _m->Row; j++)
                  {
                    _m->Val[i][j] -= a2 * _m->Val[k][j];
                    temp._m->Val[i][j] -= a2 * temp._m->Val[k][j];
                  }
                }
   }
   return temp;
}
```


Unit1.cpp – головний модуль сервера
Функція обробки вхідних повідомлень

```
void __fastcall TForm1::MyServerClientRead(TObject *Sender,
     TCustomWinSocket *Socket)
{
  AnsiString TextIn;
  int iIndex;

  TextIn = Socket->ReceiveText();
  TStringList *TMess = new TStringList();
  TMess->Text = TextIn;


  iIndex = ConnectedList->IndexOfObject(Socket);
  if(iIndex == -1)
    return;

   if (TMess->Strings[0]=="PRO")
```

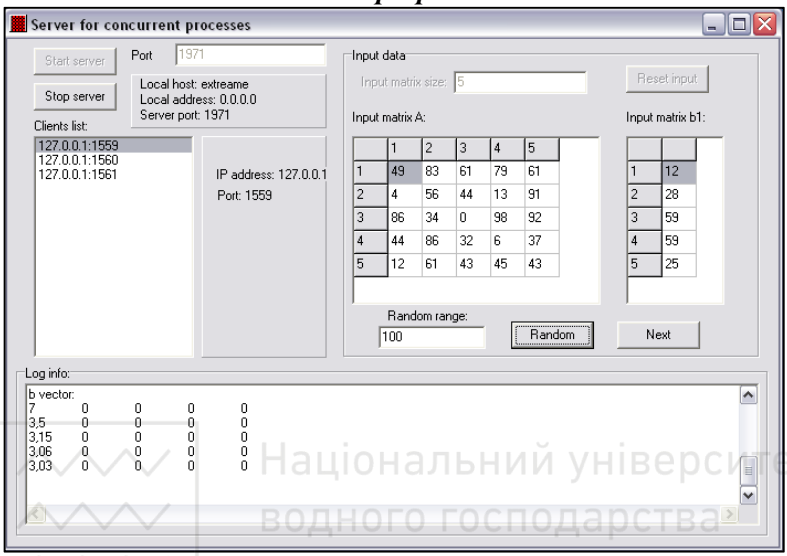
```
{
  LogMemo->Lines->Add("Received message from "+TMess->Strings[1]);

 if (TMess->Strings[2]=="_findb"){
   LogMemo->Lines->Add("b vector:");
   GetMatrixFromMSG(3,TMess, T2);
   LogMemo->Lines->Add(GetMatrixString (T2));
 }else
   if (TMess->Strings[2]=="_findC2"){
     LogMemo->Lines->Add("C2 matrix:");
     GetMatrixFromMSG(3,TMess, T1);
     LogMemo->Lines->Add(GetMatrixString (T1));
   }else
 if (TMess->Strings[2]=="_sum"){
  switch (InputStage) {
    case 6:{
      Done++;
      if (Done==2) {NextButton->Enabled=true; Done=0;}

      if (TMess->Strings[1]==ConnectedList->Strings[0]){
       LogMemo->Lines->Add("after b1+4*c1:");
       GetMatrixFromMSG(3,TMess, T1);
       LogMemo->Lines->Add(GetMatrixString (T1));
        }
       else if (TMess->Strings[1]==ConnectedList->Strings[1]){
         LogMemo->Lines->Add("after B2+C2:");
         GetMatrixFromMSG(3,TMess, T2);
         LogMemo->Lines->Add(GetMatrixString (T2));
       }
     }//case
    break;
    case 10:
      if (TMess->Strings[1]==ConnectedList->Strings[0]){
      LogMemo->Lines->Add("after Y3 x y1 + Y3^2 x y2:");
      GetMatrixFromMSG(3,TMess, TT2);
      LogMemo->Lines->Add(GetMatrixString (TT2));
       }
    break;
    case 11:
```

```
            if (TMess->Strings[1]==ConnectedList->Strings[0]){
            LogMemo->Lines->Add("after Y3 x y1 + Y3^2 x y2 + (y2~ x
y1)(Y3 x y1):");
            GetMatrixFromMSG(3,TMess, TT1);
            LogMemo->Lines->Add(GetMatrixString (TT1));
            TT1=~TT1;
            LogMemo->Lines->Add("transponovana matrix");
            LogMemo->Lines->Add(GetMatrixString (TT1));
             }
          break;
         } //switch
       }else
       {
       }//end sum
       if (TMess->Strings[2]=="_mul"){
        switch (InputStage) {
          case 6:
           if (TMess->Strings[1]==ConnectedList->Strings[2]){
            LogMemo->Lines->Add("after A*b:");
            GetMatrixFromMSG(3,TMess, T3);
            LogMemo->Lines->Add(GetMatrixString (T3));
           }
          break;
          case 7:
           if (TMess->Strings[1]==ConnectedList->Strings[0]){
            LogMemo->Lines->Add("after y2=A1*(b1+4*c1):");
            GetMatrixFromMSG(3,TMess, T1);
            LogMemo->Lines->Add(GetMatrixString (T1));
            TTT1=T1;
            T1=~T1;
            LogMemo->Lines->Add("transponovana y2");
            LogMemo->Lines->Add(GetMatrixString (T1));
           }
           if (TMess->Strings[1]==ConnectedList->Strings[1]){
            LogMemo->Lines->Add("after Y3=A2*(B2+C2):");
            GetMatrixFromMSG(3,TMess, T2);
            LogMemo->Lines->Add(GetMatrixString (T2));
           }
          break;
          case 8:
```

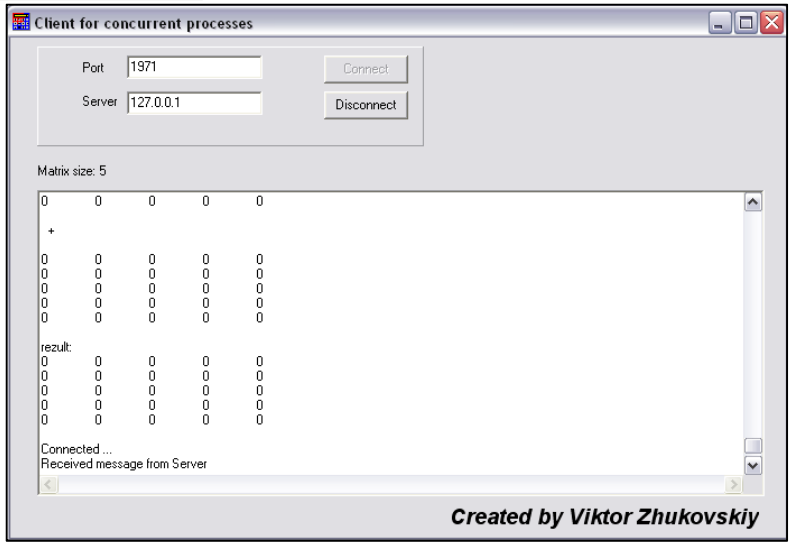
```
    if (TMess->Strings[1]==ConnectedList->Strings[0]){
     LogMemo->Lines->Add("after y2~ x y1:");
     GetMatrixFromMSG(3,TMess, TT1);
     LogMemo->Lines->Add(GetMatrixString (TT1));
    }
    if (TMess->Strings[1]==ConnectedList->Strings[1]){
     LogMemo->Lines->Add("after Y3^2");
     GetMatrixFromMSG(3,TMess, TT2);
     LogMemo->Lines->Add(GetMatrixString (TT2));
    }
    if (TMess->Strings[1]==ConnectedList->Strings[2]){
     LogMemo->Lines->Add("after Y3 x y1:");
     GetMatrixFromMSG(3,TMess, T3);
     LogMemo->Lines->Add(GetMatrixString (T3));
    }
  break;
   case 9:
    if (TMess->Strings[1]==ConnectedList->Strings[0]){
     LogMemo->Lines->Add("after (y2~ x y1)(Y3 x y1):");
     GetMatrixFromMSG(3,TMess, TT1);
     LogMemo->Lines->Add(GetMatrixString (TT1));
    }
    if (TMess->Strings[1]==ConnectedList->Strings[1]){
     LogMemo->Lines->Add("after Y3^2 x y2");
     GetMatrixFromMSG(3,TMess, TT2);
     LogMemo->Lines->Add(GetMatrixString (TT2));
    }
  break;
   case 12:
    if (TMess->Strings[1]==ConnectedList->Strings[0]){
     LogMemo->Lines->Add("Result:");
     GetMatrixFromMSG(3,TMess, TT1);
     LogMemo->Lines->Add(GetMatrixString (TT1));
    }
  break;


  } //switch
 }//case
}     }}
```

Скріншоти програм

Сервер:



Клієнт:



Висновок

В роботі використано паралелізм на рівні підзадач. Задачу було вирішено за допомогою клієнт серверної зв'язки. Контроль над даними (їх пересилку, залежність між рівнями і т.д.) виконує сервер, а клієнти є реалізаторами математичних дій. Обмін даними відбувається через сокети. 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. Залежність за ресурсами та вводом/виводом може бути визначена лише у відношенні до певної обчислювальної системи. В даному випадку помітно три своєрідних процеси, що є псевдонезалежні один від одного, тому найоптимальнішим варіантом роботи є 3 клієнта + сервер, що є аналогічним трьом процесорам + керуючий пристрій.

Лабораторна робота № 3

Тема: Паралельне представлення алгоритмів.

Мета: Вивчити можливості паралельного представлення алгоритмів. Набути навиків такого представлення.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Можливі два підходи до побудови паралельного представлення алгоритму:

1. Векторизація алгоритму представленого послідовно.
2. Безпосередньо паралельне представлення:
   2.1. Кадри.
   2.2. Програми з одноразовим присвоєнням.
   2.3. Рекурсивні рівняння.
   2.4. Графи залежностей

1. Векторизація

Векторизація – це процес генерації паралельних машинних кодів на основі послідовного алгоритму, записаного на деякій мові програмування. Вона виконується, як правило векторизуючим компілятором (автоматично) і полягає у виявленні та аналізі залежностей між операторами з метою паралельного виконання незалежних, невпорядкованих дій.

2. Пряме представлення паралельних алгоритмів

2.1. ***Кадр*** – опис обчислювальних дій в конкретний момент часу. Кадри є найбільш природнім засобом представлення алгоритму, за допомогою якого розробник може перевірити певний математичний алгоритм.

2.2. ***Програма з одноразовим присвоєнням*** – це форма, в якій кожній змінній присвоюється лише одне значення при виконанні алгоритму.

Приклад

Розглянемо задачу множення матриці на вектор, яка описується формулою:

$$c_i = \sum_{j=1}^{n} A_{ij} b_j \qquad (1)$$

Безпосередня реалізація на послідовній мові програмування (в даному випадку – на мові СІ) має вигляд:

```
for (i=0; i<n;i++)
{
  c[i]=0;
   for (j=0; j<n; j++)
c[i]=c[i]+A[i][j]*b[j];
}
```

В цій програмі $c[i]$ переписується багато разів з метою економії пам'яті. Таким чином, значення $c[i]$ присвоюється більше одного разу. При перетворенні цієї ж програми в програму з одноразовим присвоюванням кількість індексів вектора c – зросте:

```
for (i=0; i<n;i++)
{
  c[i][0]=0;
   for (j=0; j<n; j++)
c[i][j+1]=c[i][j]+A[i][j]*b[j];
}
```


Тепер кожному елементу вектора c буде присвоєно лише одне значення, а остаточні значення будуть отримані на останньому кроці ітерації.

2.3. ***Рекурсивний алгоритм*** – це алгоритм, який визначається за допомогою правила одноразового присвоювання і є стислим представленням багатьох алгоритмів. Побудова рекурсивного алгоритму зводиться до виведення рекурсивних рівнянь. Дії паралельних алгоритмів адекватно описуються в рекурсивних рівняннях з просторово-часовими індексами якщо один індекс використовується для часу, а інші – для простору (надалі – індексний простір).

Приклад

Для випадку множення матриці на вектор, рекурсивне рівняння буде мати вигляд:

$$c_i^{(j+1)} = c_i^{(j)} + A_i^{(j)} + b_i^{(j)}; \qquad (2)$$

$$a_i^{(j)} = A[i][j];$$

$$b_i^{(j)} = b[j]; \quad j - \text{індекс рекурсії.}$$

2.4. ***Граф залежностей*** – це граф, який описує залежність обчислень в алгоритмі. Він може розглядатися як графічне представлення алгоритму з одноразовим присвоєнням. Граф залежностей називається ***повним***, якщо він визначає всі залежності між всіма змінними в індексному просторі. Переважно, операції в вузлах графу не розкриваються, оскільки будуть виконуватися незалежними обчислювальними засобами (часто – процесорними елементами) і граф є ***локалізованим*** (***скороченим***).

Приклад

Для обчислення (1), виходячи з наведеного алгоритму з одноразовим присвоєнням очевидно, що $c[i][j+1]$ безпосередньо залежить від $c[i][j]$, $A[i][j]$, $b[j]$. Представивши кожну залежність у вигляді дуги між відповідними змінними, що розташовані в індексному просторі, можна отримати граф залежностей (рис. 1).

З нього видно, що значення $b[j]$ кожного елемента вектора b має бути розповсюджене у всі індексні точки, що мають однаковий індекс j. Цей тип даних називається „поширюваними" даними. Це означає, що має бути глобальний зв'язок, що не є завжди прийнятною умовою для обчислювальної системи.

Можна стверджувати, що алгоритм є зчисленним, якщо його повний граф не містить петель і циклів

Алгоритм є ***локалізованим***, якщо всі змінні безпосередньо залежать лише від змінних в сусідніх вузлах. Дані, що пересилаються незмінними до всіх вершин графу називаються ***передаваними***, в іншому випадку – це ***непередавані*** дані.

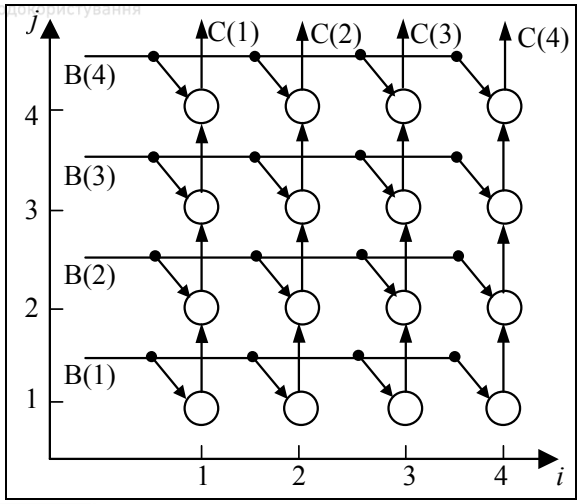


Рис. 1. Граф залежностей для множення матриці на вектор (для n=4) з глобальним зв'язком

Приклад

Програма для локалізованого алгоритму має вигляд ($b[0][j]$=b[j]):

```
for (i=0; i<n;i++)
{
  c[i][0]=0;
   for (j=0; j<n; j++)
b[i+1][j]=b[i][j]
c[i][j+1]=c[i][j]+A[i][j]*b[i][j];
}
```

В ній $b[i+1][j]$ безпосередньо залежить від $b[i][j]$, а $c[i][j+1]$ від $c[i][j]$, $A[i][j]$, $b[i][j]$.

Відповідний граф залежностей лише з локальними зв'язками зображений на рис. 2.

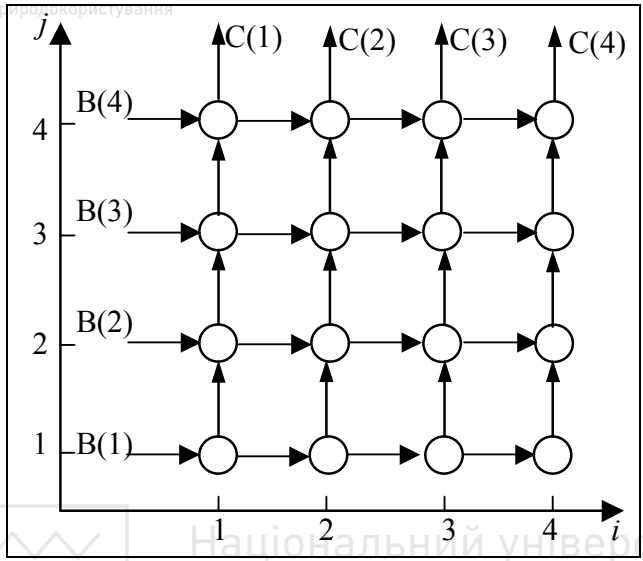


Рис. 2. Граф залежностей для множення матриці на вектор (для n=4) з локальним зв'язком

Виходячи з локалізованого графа залежностей можна дати означення:

Локально-рекурсивний алгоритм – це алгоритм, відповідний граф залежностей якого має лише локальні залежності, тобто розмір задачі не впливає на довжину кожної дуги і більшість вузлів графа складається з операцій одного типу.

ЗАВДАННЯ

Запропонувати та реалізувати локально-рекурсивний алгоритм обчислення виразу:

$$y_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \text{ ; для } j=1, 2, ..., n$$

Причому, тип вхідних послідовностей визначається згідно варіанта.

ПОРЯДОК ВИКОНАННЯ

1) Напишіть програму з одноразовим присвоєнням.

2) Знайдіть рекурсивні рівняння (тобто рекурсивний алгоритм).

3) Побудуйте граф залежностей та, виходячи з нього, локалізований граф залежностей.

4) Напишіть програму, що реалізовує локально-рекурсивний алгоритм.

5) Складіть звіт про виконану роботу:

ЗМІСТ ЗВІТУ

1. Тема, мета, завдання (варіант).
2. Результати виконання роботи (згідно наведених вище пунктів).
3. Висновки.

ВАРІАНТИ ЗАВДАНЬ

Варіант №	Тип матриці *A*	Тип матриці *B*
1.	*n* 0 .... 0 0 *n*-1...0 .... 0 .... *n*	
2.	1*2 0 ... 0 0 2*3 ... 0 .... 0 .... *n*(*n*+1)	
3.	11…11 1 … 1 . 0 . 1 … 1 11…11	

4.	111…111 011…110 … 011…110 111…111	
5.	100…001 110…011 …… 110…011 100…001	
6.	*n* 0 0 ... 0 *n*-1 *n* 0 ... 0 *n*-2 *n*-1 *n* .... .., 1 2 3 … *n*	
7.	1 2 3 … *n* …. *n*-2 *n*-1 *n* .... *n*-1 *n* 0 ... 0 *n* 0 0 ... 0	
8.	1 2 3 … *n*-1 n 2 1 2 … *n*-2 *n*-1 3 2 1 … *n*-3 *n*-2 … *n*-1 *n*-2 *n*-3 … 1 2 n *n*-1 *n*-2 … 2 1	
9.	111…111 222…220 333…300 … *n*000…00	
10.	123…*n* 123…*n* … 123…*n*	

ПРИКЛАД ВИКОНАННЯ ЗАВДАННЯ

Запропонувати та реалізувати локально-рекурсивний алгоритм обчислення виразу:

$$Y = A \times B,$$

де A та B матриці з елементами a_{ij} та b_{ij}, відповідно ($i, j = 1...n$), тобто:

$$y_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad (k = 1...n).$$

Варіант №	Тип матриці A	Тип матриці B
2	1*2 0 ... 0 0 2*3 ... 0 .... 0 .... $n(n+1)$	

Аналіз завдання

Оскільки матриця B у нижньому трикутнику містить нулі, то результуюча матриця також їх міститиме, тому не потрібно обраховувати значення цих елементів, це буде одним з етапів оптимізації алгоритму.

Рекурсивне рівняння для алгоритму множення матриць:

$$C[i][j]^{(k+1)} = C[i][j] + A[i]^{(k)} * B[j]^{(k)},$$

де k – індекс рекурсії, а

$$A[i]^{(k)} = A[i][k]$$
$$B[j]^{(k)} = B[k][j]$$

Аналітична оцінка кількості арифметичних операцій

Розмірність	Кількість операцій	
	Одноразове присвоєння	Локально-рекурсивно
2	16	4
3	54	7
4	128	80

Помітно, що локально-рекурсивний алгоритм вимагає менше арифметичних операцій порівняно з алгоритмом одноразового присвоєння. Це свідчить про вдало проведену оптимізацію. Але по часових витратах локально рекурсивний алгоритм займає більше процесорного часу і потребує більше оперативної пам'яті, так як використовується механізм рекурсії. Це означає, що його доцільніше використовувати при паралельній обробці інформації. В нашому випадку можна було розпаралелити операції при роботі з певними гранями. В комплексі це дало б значний виграш в часі.

Фрагменти тексту програми

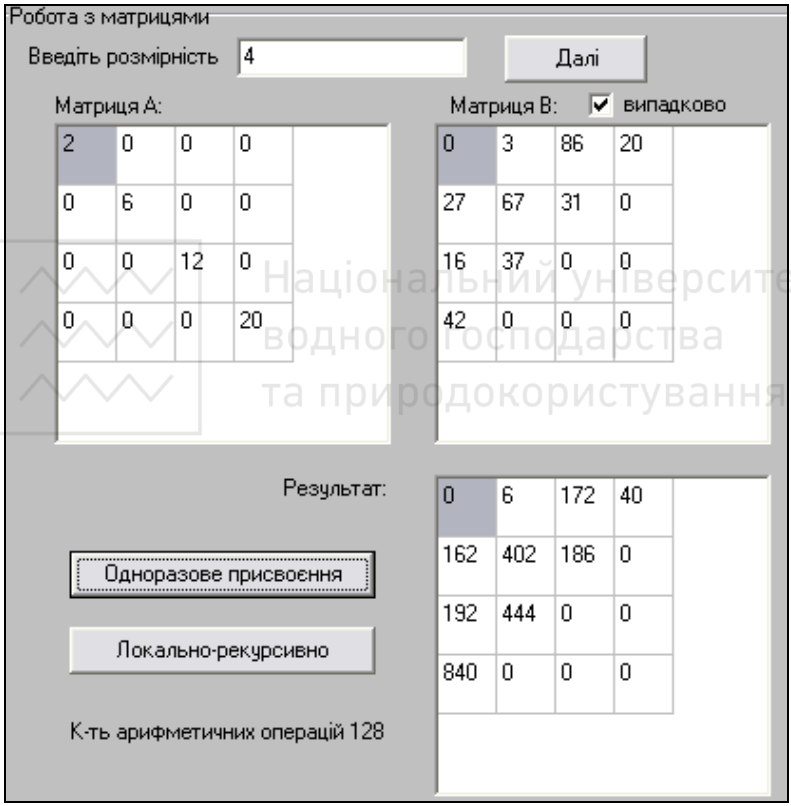
```
//Generate matrix A
procedure genA();
var i,j:integer;
begin
for i:=0 to sizem-1 do
for j:=0 to sizem-1 do
 begin
  if i=j then A[i,j]:=(i+1)*(i+2)
    else A[i,j]:=0;
 end;
end;


//Generate matrix B
procedure genB();
var i,j:integer;
begin
for i:=0 to sizem-1 do
for j:=0 to sizem-1 do
B[i,j]:=0;

for i:=0 to sizem-1 do
for j:=0 to sizem-1-i do
B[i,j]:=1;
end;
```

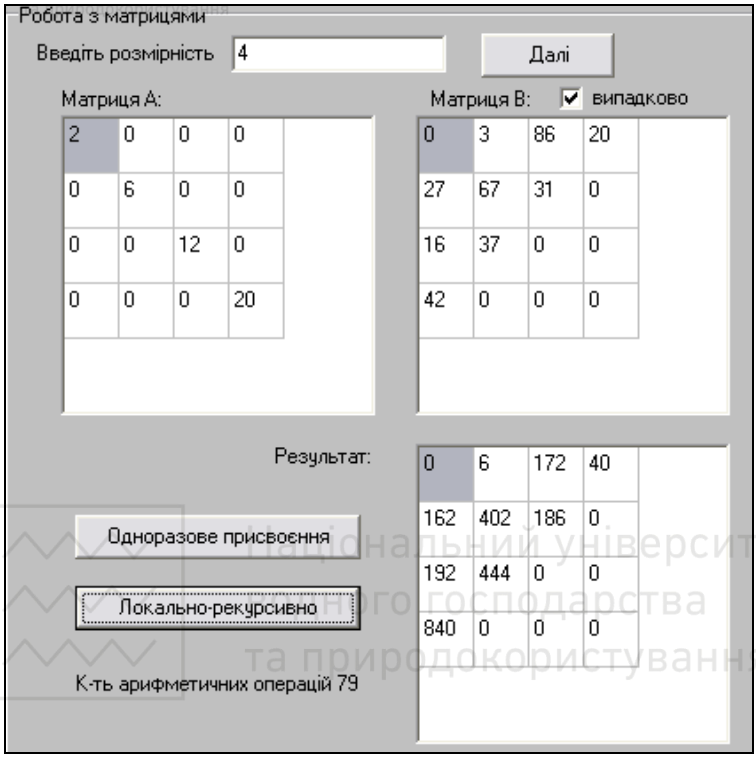
```
procedure mulmatrixes1(var
x,y:arint2d;
var z:arint3d);//Odnorazove
prusvojennja
var i,j,k:integer;
begin
for i:=0 to sizem-1 do
 begin
  for j:=0 to sizem-1 do
   begin
    z[i,j,0]:=0;
    for k:=0 to sizem-1 do
     begin

z[i,j,k+1]:=z[i,j,k]+X[i,k]*Y[k,j];
       inc(ko);  inc(ko);
     end;
   end;
 end;
end;
/
//local'no rekyrsuvnuj
procedure
mulmatrixes2(i,j,k:integer);
var l:integer;
begin
```

```
if (k<sizem) and (i<sizem) and
(j<sizem) then
begin
if((A[i,k]<>0) and (B[k,j]<>0))
then
  begin
    C2[i,j,sizem]:=A[i,k]*B[k,j];
    inc(ko);
  end;
    mulmatrixes2(i,j,k+1);//up
    mulmatrixes2(i+1,j,k);//right
    mulmatrixes2(i,j+1,k);//left
end; end;

procedure
TGame.NextClick(Sender:
TObject);
var i,j:integer;
begin

sizem:=StrToIntDef( Edit1.Text
,4);
  ko:=0;
  SetLength(A,sizem,sizem);
  SetLength(B,sizem,sizem);

SetLength(C1,sizem,sizem,siz
em+1);

SetLength(C2,sizem,sizem,siz
em+1);

  MatrixA.ColCount :=sizem;
  MatrixA.RowCount:=sizem;


  MatrixB.ColCount :=sizem;
  MatrixB.RowCount:=sizem;
  MatrixC.ColCount :=sizem;
  MatrixC.RowCount:=sizem;
   genA();  genB();
   for i:=0 to sizem-1 do
   for j:=0 to sizem-1-i do
     if      (CheckBox1.Checked)
then B[i,j]:=random(100);
   for i:=0 to sizem-1 do
   for j:=0 to sizem-1 do

MatrixA.Cells[j,i]:=inttostr(A[i,j]
);
   for i:=0 to sizem-1 do
   for j:=0 to sizem-1 do

MatrixB.Cells[j,i]:=inttostr(B[i,j]
);
end;

procedure
TGame.Button1Click(Sender:
TObject);
var i,j:integer;
begin
  mulmatrixes1(A,B,C1);
  for i:=0 to sizem-1 do
  for j:=0 to sizem-1 do

MatrixC.Cells[j,i]:=inttostr(C1[i,
j,sizem]);
  Label13.Caption:='К-ть
арифметичних      операцій
'+IntToStr(ko);
    ko:=0;
end;

procedure
TGame.Button3Click(Sender:
TObject);
var i,j:integer;
begin
for i:=0 to sizem-1 do
```

```
for j:=0 to sizem-1 do
  C2[i,j,0]:=0;
mulmatrixes2(0,0,0);
  for i:=0 to sizem-1 do  for
j:=0 to sizem-1 do
  MatrixC.Cells[j,i]:=inttostr(C2[i,
j,sizem]);
  Label13.Caption:='К-ть
арифметичних операцій
'+IntToStr(ko);  ko:=0; end;
```

Результати роботи:





Висновок

Під час виконання лабораторної роботи було вивчено можливості паралельного представлення алгоритмів і набуто навиків такого представлення. Зокрема на прикладі множення двох матриць було проаналізовано алгоритм з одноразовим присвоєнням і локально-рекурсивний алгоритм.

Лабораторна робота № 4

Тема: Можливості використання паралельних алгоритмів.

Мета: Дослідити можливості розв'язання різноманітних задач за допомогою паралельних алгоритмів. Навчитися виділяти незалежні гілки обчислень та виконувати їх паралельно.

ПОРЯДОК ВИКОНАННЯ

1) Проаналізуйте завдання і виділіть в ньому незалежні гілки обчислень (якщо це можливо). Використайте метод розбиття задачі на під задачі та виділення незалежних подій.

2) Реалізуйте кожну гілку окремо.

3) Об'єднайте всі частини для вирішення поставленої задачі і переконайтеся у правильності реалізації.

4) Оцініть обчислювальні та часові затрати створеної програми (тобто як зростає час виконання при збільшенні розмірності задачі)

5) Запропонуйте оптимальну архітектуру реалізації алгоритму.

6) Складіть звіт про виконану роботу:

ЗМІСТ ЗВІТУ

1. Тема, мета, завдання.
2. Аналіз задачі та опис незалежних подій.
3. Текст програми.
4. Висновки.

ВАРІАНТИ ЗАВДАНЬ

1. ***Графом*** називається сукупність точок (вузлів), деякі з яких з'єднані між собою направленими ребрами. Граф, що складається з n вузлів можна описати двома матрицями порядку n: матрицею з'єднань та матрицею зв'язків. Елемент матриці з'єднань a_{ij}=1, якщо граф містить ребро, направлене від вузла i

до вузла j та a_{ij}=0 в іншому випадку. Елемент матриці зв'язків b_{ij}=1, якщо з вузла i можна попасти у вузол j, рухаючись по ребрах і b_{ij}=1 в іншому випадку.

Завдання. 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.

2. Лінія називається ***унікурсальною***, якщо її можна накреслити не відриваючи перо від паперу та не проходячи два рази одне і теж ребро. (Зауважимо, що лінія є унікурсальною лише тоді, коли кількість тих вузлів, з яких виходить непарна кількість ребер, не більше двох). Лінію, що містить n вузлів можна задати квадратною матрицею з'єднань, (порядку n), в якій елемент a_{ij}=1, якщо вузол i з'єднаний з вузлом j ребром, що не містить інших вузлів.

Завдання. Ввести кількість вузлів деякої лінії. Задавши довільним чином матрицю з'єднань (в інтерактивному режимі або випадковим чином), визначити чи є така лінії унікурсальною і, якщо є, то отримати послідовність номерів вузлів, які будуть пройдені 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. „Ханойські вежі". Є три стержні. На першому з них нанизано m 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.

Завдання. Запропонувати алгоритм вирішення поставленої задачі, забезпечивши ввід кількості дисків та відображення їх переміщення в процесі перестановки.

5. Гра „LIFE” 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.

Завдання. Запропонувати та відобразити алгоритм реалізації гри.

6. Задати (ввести з клавіатури або 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.

Завдання. Задати (ввести з клавіатури або 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. Задати натуральні числа m, a_1, …, a_n. У послідовності a_1, …, a_n вибрати підпослідовність a_1, …, a_n $a_{i_1}, a_{i_2}, ..., a_{i_k}$ $(0 \le i_1 < i_2 < ... < i_k \le n)$ таку, що $a_{i_1} + a_{i_2} + ... + a_{i_k} = m$. Якщо це неможливо, то слід вивести відповідне повідомлення.

(При вирішенні даної задачі слід використати наступне міркування. Для того, щоб вибрати необхідну підпослідовність, для кожного елемента вхідної послідовності слід визначити, чи входить він у шукану підпослідовність. Може виникнути наступна ситуація: відносно членів a_1, ..., $a_i (i < n)$ прийняті які-небудь рішення, після чого виявилось, що, як би ми не розпоряджались іншими $n - i$ членами, нам все одно не вдасться отримати підпослідовність, що задовольняє поставленій умові (наприклад, якщо сума декількох додатних чисел більше m, то неможливо додати до них ще декілька додатних чисел, так, щоб сума стала рівна m). В цьому випадку слід виключити з розгляду всі підпослідовності, перші елементи яких вибрані з a_1, ..., a_i, згідно з прийнятими рішеннями.)

10. Задати натуральне число m. Отримати m 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 m більше ніж загальна кількість таких конфігурацій, то слід знайти всі комбінації.

11. Лабіринт задається матрицею з'єднань (див.1), в якій для кожної пари кімнат вказано, чи з'єднані вони коридором. Задати (ввести з клавіатури або генерувати випадковим чином): кількість кімнат лабіринту (n); матрицю з'єднань для них; номери кімнат i та j $(1 \le i, j \le n)$. Побудувати шлях з кімнати з номером i в кімнату з номером j.

12. Задати (ввести з клавіатури або 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.

13. Є деяка кількість міст, деякі з яких з'єднані дорогами відомої довжини. Вся система доріг описується квадратною матрицею порядку n, в якій елемент a_{ij} рівний будь-якому від'ємному числу, якщо місто i безпосередньо не з'єднано з містом j 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.

14. Для попередньої задачі припустити, що кожне місто має пряму дорогу до кожного іншого, і знайти найкоротший шлях, що проходить через всі міста.

Приклад виконання завдання

Варіант завдання 10

Задати натуральне число m. Отримати m 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 m більше, ніж загальна кількість таких конфігурацій, то слід знайти всі комбінації.

Аналіз задачі

Цю задачу розв'язав більше 200 років тому назад великий математик Леонард Ейлер. До речі, тоді в нього не було ніякого комп'ютера, але, тим не менше, він абсолютно правильно знайшов всі 92 такі розстановки.

Очевидно, на кожній з 8 вертикалей повинно стояти по ферзю. Кожну таку розстановку можна закодувати одномірним масивом:

$X[1], ..., X[8]$,

де $X[i]$ – номер горизонталі i-го ферзя. Оскільки ніякі два ферзі не можуть стояти на одній горизонталі (бо тоді вони б'ють один одного), то всі $X[i]$ різні, тобто становлять собою перестановку з

чисел 1..8. Можна, звичайно, перебрати всі 8! таких перестановок і вибрати серед них ті 92, що нас цікавлять. Але число 8!=40320 доволі велике.

Можна спробувати наступний алгоритм:

- Оскільки в кожній горизонталі стоїть рівно 1 ферзь (див. вище), то
- пробуємо перемістити першого ферзя на 1-шу горизонталь, ..., *i*-го на *i*-ту горизонталь так, щоб вони один одного не били;
- якщо *i*=8, то все добре, інакше повертаємося до попереднього ферзя і пробуємо його зсунути вправо так, щоб він не бив попередніх. Якщо не вийшло, то повертаємося ще назад ...
- як тільки вдалося зсунути ферзя, беремо наступного і пробуємо його помістити на горизонталь.

Для забезпечення паралельності потрібно дещо змінити код програми, передбачивши, що існують симетричні комбінації. Тобто одні процесори будуть працювати з дошкою зліва направо, інші справа наліво. Так потрібні комбінації будуть швидше знайдені. Але у випадку з симетричними позиціями можна обійтися і послідовним алгоритмом, шукаючи не 92, а 46 комбінацій і т.д.

Фрагменти тексту програми

```
class CQueen
{
public:
        int ncount;
        int calculate();
        void print_pos();
        bool testing (int vert1,
int vert2);
        bool find_next_quen();

        private:
        char buff [1024];
        DWORD     d;     //for
writing in file
                HANDLE hFile;
                int count;
                int cur_vert;
                int success;
                int pos[8];
};

bool
CQueen::find_next_quen()
 {
  int       old_quen_pos       =
pos[cur_vert];
```

```
  pos[cur_vert]++;
  for(; pos[cur_vert] <= 8; pos[cur_vert] ++ )
    if( cur_vert != 0 ) for( int vert = 0; vert < cur_vert; vert ++ ) {
      if( !testing( vert, cur_vert ) ) break;
      if( vert == ( cur_vert - 1 ) ) return true;
    }
    else {

          if( pos[cur_vert] != 9 ) { // changed
        return true;
      }
      else break;
    }
  pos[cur_vert] = old_quen_pos;
  return false;
}

bool CQueen::testing(int vert1, int vert2)
{
  if( pos[vert1] == pos[vert2] ) return false;
  if( abs( pos[vert1] - pos[vert2] ) == abs( vert1 - vert2 ) ) return false;
  return true;
}

void CQueen::print_pos()
{

  char v = 'a';
  int vert = 0;
  for( ; vert < 7; v++, vert++ )
  {
        sprintf(buff, "%c%d, ",v,pos[vert]);
        WriteFile(hFile, buff, strlen(buff), &d, 0);
  }
  sprintf(buff, "%c%d\r\n",v,pos[7]);
  WriteFile(hFile, buff, strlen(buff), &d, 0);
  sprintf(buff,"%s","");
  for (int i=0;i<8 ;i++)
  {
        for (int j=0;j<8;j++)
              if (pos[i]==j) strcat (buff,"x");
              else strcat (buff,"-");
        strcat (buff,"\r\n");
  }
  WriteFile(hFile, buff, strlen(buff), &d, 0);
}

int CQueen::calculate()
{
        hFile = CreateFile((LPCTSTR) "result.txt",
        GENERIC_WRITE,
        FILE_SHARE_WRITE,
        NULL,
        CREATE_ALWAYS,
        FILE_ATTRIBUTE_NORMAL | FILE_FLAG_SEQUENTIAL_SCAN,
        (HANDLE)NULL);
```

```
        if ((!hFile) || (hFile == (void*)0xffffffff)) return 0; // exit

        count=0;
        cur_vert=0;
        success=0;
        for (int i=0;i<8;i++)
          pos[i]=0;

for(;;) {
  count++;
  if( find_next_quen() ) {
    if( cur_vert == 7 ) {
      print_pos();
      success++;
      pos[cur_vert] = 0;
      cur_vert--;
    }
    else cur_vert++;
  }
  else {
    if( cur_vert == 0 ) break;
    pos[cur_vert] = 0;
    cur_vert--;
  }
}

        sprintf(buff, "count cycle\t %d \r\n", count);
        WriteFile(hFile, buff, strlen(buff), &d, 0);
        sprintf(buff, "success\t %d \r\n", success );
        WriteFile(hFile, buff, strlen(buff), &d, 0);
  CloseHandle(hFile);
}
```

Результати роботи:

Для 10 комбінацій

```
a1, b5, c8, d6, e3, f7, g2, h4
-x------
-----x--
--------
------x-
---x----
-------x
--x-----
----x---
a1, b6, c8, d3, e7, f4, g2, h5
-x------
------x-
--------
---x----
-------x
----x---
--x-----
-----x--
a1, b7, c4, d6, e8, f2, g5, h3
-x------
-------x
----x---
------x-
--------
--x-----
-----x--
---x----
a1, b7, c5, d8, e2, f4, g6, h3
-x------
-------x
-----x--
--------
--x-----
----x---
------x-
---x----
```

a2, b4, c6, d8, e3, f1, g7, h5
--x-----
----x---
------x-

---x----
-x------
-------x
-----x--

a2, b6, c1, d7, e4, f8, g3, h5
--x-----
------x-
-x------
-------x
----x---

---x----
-----x—

a2, b5, c7, d1, e3, f8, g6, h4
--x-----
-----x--
-------x
-x------
---x----

------x-
----x---

a2, b6, c8, d3, e1, f4, g7, h5
--x-----
------x-

---x----
-x------
----x---
-------x
----x--

a2, b5, c7, d4, e1, f8, g6, h3
--x-----
-----x--
-------x
----x---
-x------

------x-
---x----

a2, b7, c3, d6, e8, f5, g1, h4
--x-----
-------x
---x----
------x-

-----x--
-x------
----x---

Висновок

Під час виконання лабораторної роботи було вивчено можливості паралельного представлення алгоритмів і їх застосування до розв’язання конкретних задач. Зокрема для задачі знаходження комбінацій для 8 ферзів було проаналізовано можливість її паралельного виконання.