Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

Звіт
з лабораторної роботи №2
з дисципліни «Теорія паралельних обчислень»
на тему «Паралельні алгоритми розв'язування СЛАР з симетричними, блочно-діагональними та k-діагональними матрицями»

Виконали ст. гр. ІПЗм-22-6:
Миронюк С.А.,
Сєнічкін І.О.

Перевірив викладач:
доц. Кобзєв В.Г.

Харків, 2023

ПАРАЛЕЛЬНІ АЛГОРИТМИ РОЗВ'ЯЗУВАННЯ СЛАР З СИМЕТРИЧНИМИ, БЛОЧНО-ДІАГОНАЛЬНИМИ ТА k-ДІАГОНАЛЬНИМИ МАТРИЦЯМИ

Мета роботи – навчитися створювати і аналізувати паралельні алгоритми розв'язування систем лінійних рівнянь з симетричними матрицями (методи Холецького і LDL-розкладу), систем з розрідженими (блочно-діагональними і k-діагональними) матрицями та оцінювати показники їх прискорення у порівнянні з послідовними алгоритмами.

Індивідуальне завдання:

Система лінійних алгебраїчних рівнянь (СЛАР) задана матрицею коефіцієнтів A та вектором вільних членів B, які відповідають завданню до лабораторної роботи 1.

1. Розробити послідовний та паралельний алгоритми розв'язку СЛАР (у відповідності до номеру індивідуального завдання):

а) з використанням розкладу Холецького,

б) з використанням LDL-розкладу,

в) для блочно-діагональної матриці (розмір блоків отримати у викладача);

г) для *к*-діагональної матриці (значення *к* отримати у викладача);

д) для блочно-діагональної матриці з обрамленням (структуру матриці та розмір блоків отримати у викладача).

2. Реалізувати алгоритми програмно. Обчислити значення показників прискорення та ефективності розпаралелювання.

3. Порівняти отримані результати з результатами виконання лабораторної роботи 1.

Обрано варіант – г) для k-діагональної матриці (k=3).

Дані для лабораторної роботи: значення синіх пікселів зображення «Вхід до ХНУРЕ». Рядок – починаючи с 60-го пікселя, строка – з 1-го. Задані розміри матриць; 150×150, 300×300, 500×500, 1000×1000. Стовпчик вільних членів – 6-ій з кінця рисунку 1. У випадку недостатньої інформації на фотографії, нам потрібно додати ще одну копію цієї фотографії. Зазначена фотографія відображена на рисунку 1.

Рисунок 1 – Зображення для виконання завдання

Хід роботи:

k-діагональні матриці містять у собі ненульові елементи на головній діагоналі та на сусідніх з нею позиціях з кожного боку від неї. Таким чином додатково формується парна кількість субдіагоналей.

Структура k-діагональної матриці наведена на рис. 2.

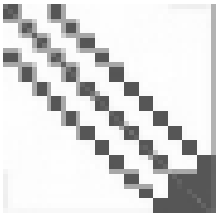


Рисунок 2 – Структура матриці з трьома діагоналями

Для розв'язування СЛАР, представлених трьох-діагональними матрицями, використовується метод прогонки – це модифікація методу Гаусса. Така система рівнянь може бути записана у вигляді:

$$\begin{cases} b_1x_1 + c_1x_2 = d_1 \\ a_2x_1 + b_2x_2 + c_2x_3 = d_2 \\ a_3x_2 + b_3x_3 + c_3x_4 = d_3 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n = d_{n-1} \\ a_nx_{n-1} + b_nx_n = d_n \end{cases} \quad (1)$$

Матрично-векторна форма ***Ax=d***, де

$$A = \begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & a_n & b_n \end{pmatrix}; d = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \dots \\ d_{n-1} \\ d_n \end{pmatrix} \quad (2)$$

Хід роботи за методом прогонки складається з двох етапів – прямої прогонки та оберненої прогонки. Ідея прямої прогонки полягає в тому, що кожне невідоме x_i виражається за допомогою прогоночних коефіцієнтів: α_i і β_i

$$x_i = \alpha_i x_{i+1} + \beta_i, i = 1,2,\dots,n-1 \quad (3)$$

З першого рівняння системи *(1)* знайдемо:

$$x_1 = -\frac{c_1}{b_1}x_2 + \frac{d_1}{b_1}$$

Використовуючи формулу (3) бачимо, що $x_1 = \alpha_1 x_2 + \beta_1$. Прирівнюючи коефіцієнти в обох виразах для x_1 отримаємо:

$$\alpha_1 = -\frac{c_1}{b_1}, \beta_1 = \frac{d_1}{b_1} \quad (4)$$

Підставимо в друге рівняння системи замість x_1 значення $\alpha_1 x_2 + \beta_1$ отримаємо:

$$a_2(\alpha_1 x_2 + \beta_1) + b_2x_2 + c_2x_3 = d_2$$

і звідси знаходимо x_2:

$$x_2 = \frac{-c_2 x_3 + d_2 - a_2\beta_1}{a_2\alpha_1 + b_2}$$

або $x_1 = \alpha_1 x_2 + \beta_1$, де

$$\alpha_2 = -\frac{c_2}{a_2\alpha_1 + b_2};\ \beta_2 = \frac{d_2 - a_2\beta_1}{a_2\alpha_1 + b_2}$$

Аналогічно можна знайти коефіцієнти для будь-якого i:

$$\alpha_i = -\frac{c_i}{a_i\alpha_{i-1} + b_i};\ \beta_i = \frac{d_i - a_i\beta_{i-1}}{a_i\alpha_{i-1} + b_i};\ i = 2,3,\dots,n-1 \quad (5)$$

Обернена прогонка полягає у послідовному обчисленні невідомих x_i. Для цього спочатку, з останнього рівняння системи, знаходимо x_n:

$$a_n x_{n-1} - b_n x_n = d_n$$

Підставимов останній вираз замість x_{n-1} значення $\alpha_{n-1}x_n + \beta_{n-1}$ отримаємо:

$$x_n = \frac{d_n - a_n\beta_{n-1}}{a_n\alpha_{n-1} + b_n}$$

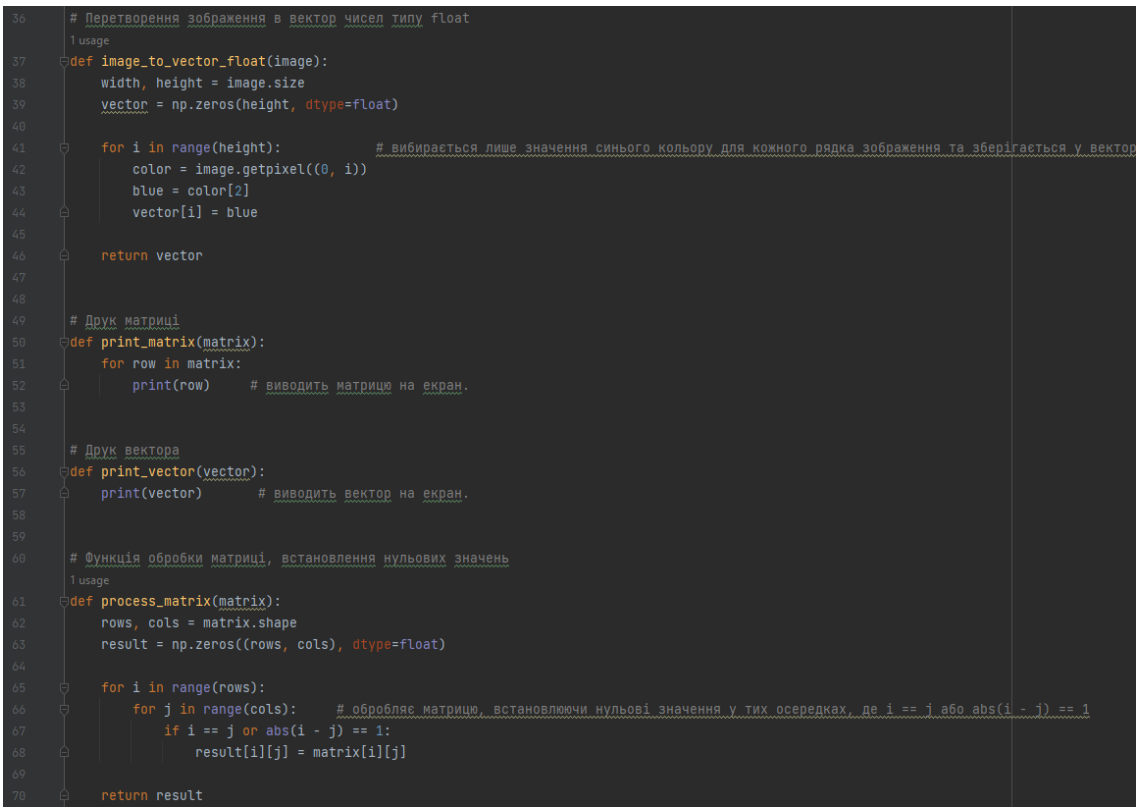
Далі, використовуючи формулу (1) і вирази для прогоночних коефіцієнтів (4) і (5), послідовно знаходимо всі невідомі $x_{n-1}, x_{n-2}, \dots, x_1$.

Дана програма була виконана на пристрої DESKTOP-IDO8GRU з процесором Intel(R) Core(TM) i5-7600 CPU, який включає 4 ядра, базовая тактовая частота 3500 МГц, объем кэш памяти 3 уровня (L1 – 256 КБ, L2 – 1,0 МБ, L3 – 6 МБ).

Імпортуються необхідні бібліотеки, PIL (Pillow) для роботи із зображеннями, NumPy для роботи з матрицями та векторами, ThreadPoolExecutor для паралельного виконання завдань.

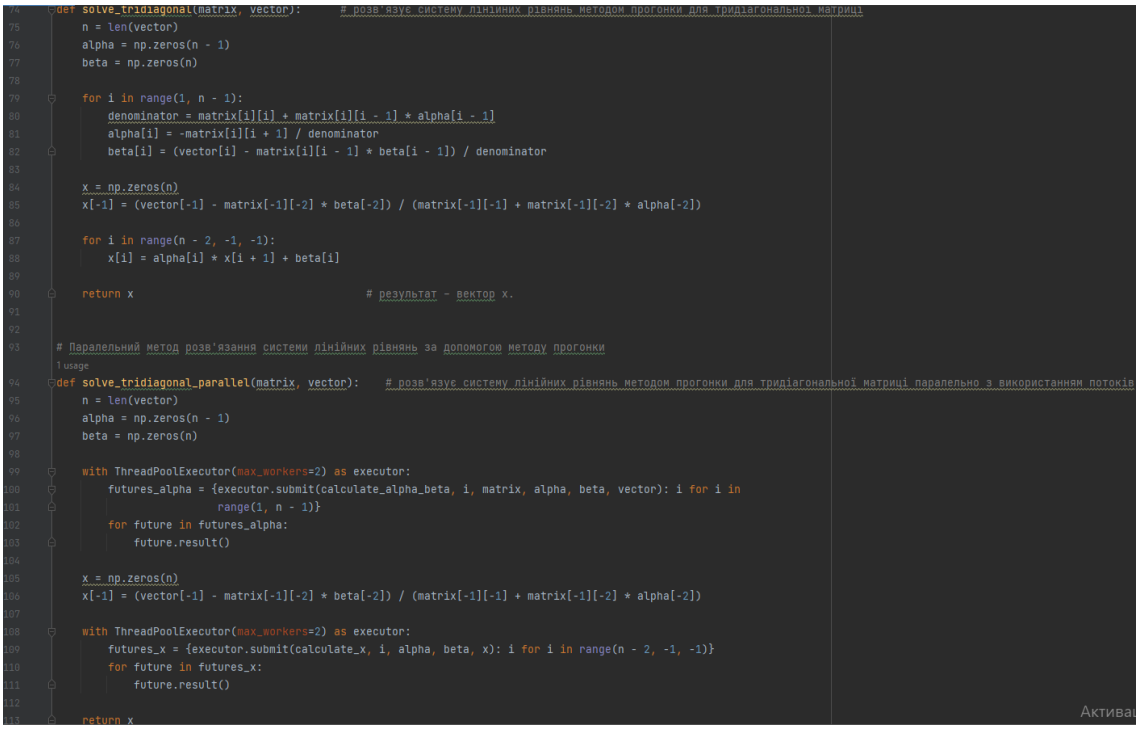
```python
from PIL import Image           # імпорт модуля Image з бібліотеки Python Imaging Library (PIL), який надає можливість працювати із зображеннями
import numpy as np              # Імпорт бібліотеки NumPy, яка надає функціональність для роботи з масивами та матрицями, що корисно при обробці зображень
import time                     # Імпорт модуля time, який надає функції для роботи з часом
from concurrent.futures import ThreadPoolExecutor
from concurrent.futures import ProcessPoolExecutor # імпорт класів, що надають абстракції для паралельного виконання коду з використанням потоків та процесів відповідно

# Функція об'єднання двох зображень вертикально
1 usage
def glue_images_vertically(image1, image2):    # об'єднує два зображення вертикально. Створюється нове зображення (result), яке вставляються image1 і image2 вертикально
    width = max(image1.width, image2.width)
    height = image1.height + image2.height
    result = Image.new('RGB', (width, height), 'white')
    result.paste(image1, (0, 0))               # вставляються image1
    result.paste(image2, (0, image1.height))   # вставляються image2 вертикально
    return result                              # створюється нове зображення


# Функція обрізки зображення за заданими координатами та розмірами
2 usages
def crop_image(image, x, y, width, height):
    return image.crop((x, y, x + width, y + height))   # обрізає зображення за заданими координатами (x, y) та розмірами (width, height)


# Перетворення зображення в матрицю чисел типу float
1 usage
def image_to_matrix_float(image):
    width, height = image.size
    matrix = np.zeros((height, width), dtype=float)   # кожен піксель зображення є значенням синього кольору (blue) і зберігається в матрицю

    for i in range(height):
        for j in range(width):
            color = image.getpixel((j, i))
            blue = color[2]
            matrix[i][j] = blue

    return matrix
```

Рисунок 3 – Частина коду для обробки зображення, перетворення їх у числові уявлення



```python
# Перетворення зображення в вектор чисел типу float
1 usage
def image_to_vector_float(image):
    width, height = image.size
    vector = np.zeros(height, dtype=float)

    for i in range(height):            # вибирається лише значення синього кольору для кожного рядка зображення та зберігається у вектор
        color = image.getpixel((0, i))
        blue = color[2]
        vector[i] = blue

    return vector


# Друк матриці
def print_matrix(matrix):
    for row in matrix:
        print(row)     # виводить матрицю на екран.


# Друк вектора
def print_vector(vector):
    print(vector)       # виводить вектор на екран.


# Функція обробки матриці, встановлення нульових значень
1 usage
def process_matrix(matrix):
    rows, cols = matrix.shape
    result = np.zeros((rows, cols), dtype=float)

    for i in range(rows):
        for j in range(cols):     # обробляє матрицю, встановлюючи нульові значення у тих осередках, де i == j або abs(i - j) == 1
            if i == j or abs(i - j) == 1:
                result[i][j] = matrix[i][j]

    return result
```

Рисунок 4 – Частина коду для перетворення зображення в числові уявлення



```python
def solve_tridiagonal(matrix, vector):      # розв'язує систему лінійних рівнянь методом прогонки для тридіагональної матриці
    n = len(vector)
    alpha = np.zeros(n - 1)
    beta = np.zeros(n)

    for i in range(1, n - 1):
        denominator = matrix[i][i] + matrix[i][i - 1] * alpha[i - 1]
        alpha[i] = -matrix[i][i + 1] / denominator
        beta[i] = (vector[i] - matrix[i][i - 1] * beta[i - 1]) / denominator

    x = np.zeros(n)
    x[-1] = (vector[-1] - matrix[-1][-2] * beta[-2]) / (matrix[-1][-1] + matrix[-1][-2] * alpha[-2])

    for i in range(n - 2, -1, -1):
        x[i] = alpha[i] * x[i + 1] + beta[i]

    return x                                # результат - вектор x.


# Паралельний метод розв'язання системи лінійних рівнянь за допомогою методу прогонки
1 usage
def solve_tridiagonal_parallel(matrix, vector):    # розв'язує систему лінійних рівнянь методом прогонки для тридіагональної матриці паралельно з використанням потоків
    n = len(vector)
    alpha = np.zeros(n - 1)
    beta = np.zeros(n)

    with ThreadPoolExecutor(max_workers=2) as executor:
        futures_alpha = {executor.submit(calculate_alpha_beta, i, matrix, alpha, beta, vector): i for i in
                         range(1, n - 1)}
        for future in futures_alpha:
            future.result()

    x = np.zeros(n)
    x[-1] = (vector[-1] - matrix[-1][-2] * beta[-2]) / (matrix[-1][-1] + matrix[-1][-2] * alpha[-2])

    with ThreadPoolExecutor(max_workers=2) as executor:
        futures_x = {executor.submit(calculate_x, i, alpha, beta, x): i for i in range(n - 2, -1, -1)}
        for future in futures_x:
            future.result()

    return x
```

Рисунок 5 – Частина коду розв'язки системи лінійних рівнянь послідовним та паралельним методами

Обробка зображень: завантажується зображення "knure.jpg" і виконується вертикальне поєднання зображення із самим собою (`glue_images_vertically`).

Виконується обрізка отриманого зображення для отримання матриці (`image_knure_matrix`) і вектора (`image_knure_vector`) числових даних.

```
# Завантаження зображення "knure.jpg"
image_knure = Image.open("images/knure.jpg")
start_time = time.time()

# Об'єднання та обрізка зображень
image_glued_knure = glue_images_vertically(image_knure, image_knure)
image_knure_matrix = crop_image(image_glued_knure, x: 60, y: 0, size, size)
image_knure_vector = crop_image(image_glued_knure, x: 1163, y: 0, width: 1, size)
```

Зображення перетворюються на числові уявлення з використанням функцій `image_to_matrix_float` та `image_to_vector_float`. Отримані числові уявлення матриці та вектора перетворюються на NumPy масиви.

```
# Перетворення зображень у числові представлення
matrix_float = image_to_matrix_float(image_knure_matrix)
vector_float = image_to_vector_float(image_knure_vector)

# Перетворення у NumPy масиви
matrix = np.array(matrix_float, dtype=float)
vector = np.array(vector_float, dtype=float)
```

Матриця обробляється функцією `process_matrix`.

```
# Обробка матриці
result_matrix = process_matrix(matrix)
```

Потім розв'язки системи лінійних рівнянь виконуються двома методами: послідовним (`solve_tridiagonal`).

```
# Розв'язання системи лінійних рівнянь (послідовний метод)
start_time_sequential = time.time()
result_vector = solve_tridiagonal(result_matrix, vector)
end_time_sequential = time.time()
elapsed_time_sequential = (end_time_sequential - start_time_sequential) * 1000
```

та паралельним (`solve_tridiagonal_parallel`) методами прогонки.

```
# Розв'язання системи лінійних рівнянь (паралельний метод)
start_time_parallel = time.time()
result_vector_parallel = solve_tridiagonal_parallel(result_matrix, vector)
end_time_parallel = time.time()
elapsed_time_parallel = (end_time_parallel - start_time_parallel) * 1000
```

Ця частина коду в циклі ітеративно обчислює значення alpha і beta для кожного i. Розпаралелювання полягає в тому, щоб кожна ітерація циклу виконувалася незалежно в окремому потоці. Тут використовується ThreadPoolExecutor для створення пулу потоків і кожен потік обчислює значення *alpha* і *beta* для свого індексу *i*. Різні ітерації можуть виконуватись паралельно.

Далі вимірюється час виконання для послідовного та паралельного методів

```
# Виведення часу виконання
print("Розмір матриці {}: ".format(size))
print("Час виконання послідовного методу: {} мілісекунд".format(float(elapsed_time_sequential)))
print("Час виконання паралельного методу: {} мілісекунд".format(float(elapsed_time_parallel)))
```

Результати відображаються на екрані для різних розмірів матриці.

```
Розмір матриці 150:
Час виконання послідовного методу: 0.9968280792236328 мілісекунд
Час виконання паралельного методу: 6.604671478271484 мілісекунд
Розмір матриці 300:
Час виконання послідовного методу: 0.9977817535400391 мілісекунд
Час виконання паралельного методу: 10.020732879638672 мілісекунд
Розмір матриці 500:
Час виконання послідовного методу: 0.9975433349609375 мілісекунд
Час виконання паралельного методу: 22.61042594909668 мілісекунд
Розмір матриці 1000:
Час виконання послідовного методу: 1.9946098327636719 мілісекунд
Час виконання паралельного методу: 30.782699584960938 мілісекунд

Process finished with exit code 0
```

Рисунок 6 – Результати виконання програми

Таблиця 1 – Час виконання алгоритмів розв’язання СЛАУ для k-діагональної матриці різних розмірів

Розмір матриці	Послідовний метод	Паралельний метод		
	Час виконання послідовного методу, мс	Час виконання паралельного методу, мс	Прискорення, $S = \frac{T_{sequential}}{T_{parallel}}$	Ефективність, $E = \frac{S}{число\ потоків}$
150×150	0,997	6,605	0,15	0,075
300×300	0,998	10,021	0,10	0,050
500×500	0,998	22,610	0,03	0,015
1000×1000	1,995	30,788	0,06	0,030

Отримані результати значення прискорення вказують на сповільнення роботи алгоритму при паралельній реалізації.

На рисунку 7 зображено графік порівняння часу виконання паралельного і послідовного алгоритмів для різних розмірів матриць.

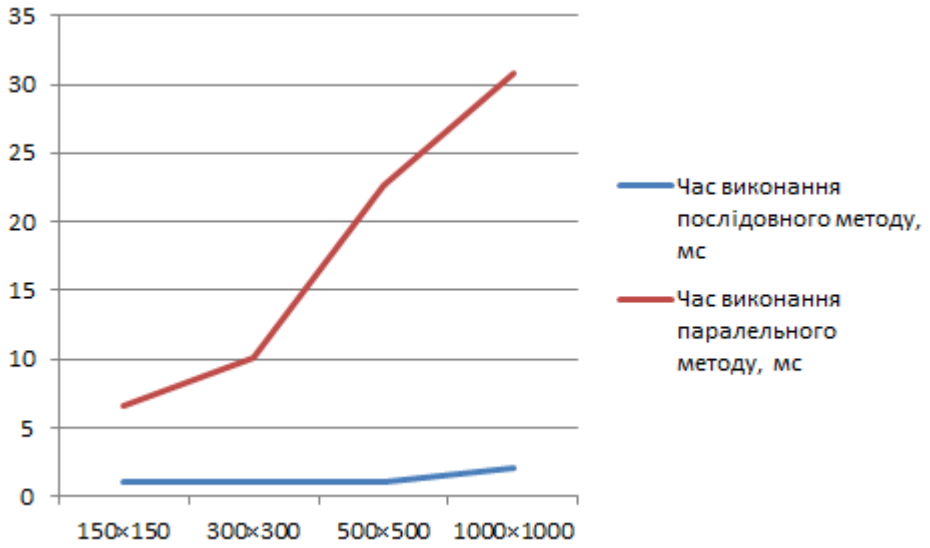


Рисунок 7 – Результати виконання програми

Висновки: В ході роботи були отримані результати, які вказують на те, що в даному випадку використання паралельних потоків призводить до надлишкових накладних витрат через управління потоками, і час виконання паралельного методу виявився більшим, ніж час виконання послідовного методу. При малих значеннях k такий підхід буде уповільнювати роботу алгоритму, тому що він потребує більше часових витрат на підтримку багатопоточності.

Також важливо врахувати, що в Python є Global Interpreter Lock (GIL), який дозволяє виконувати тільки одну інструкцію Python одночасно в одному процесі. Це може створювати обмеження на швидкість паралельного виконання в деяких випадках. Додатково, розмір матриці та обсяг роботи, яку потрібно виконати в кожному потоці, також можуть впливати на результати. Якщо розмір матриці невеликий, а обчислення невеликі, то накладки на створення та керування потоками можуть переважати вигоди від паралельного виконання.

З цього можна зробити висновок, що для розв'язку СЛАР тридіагональних матриц методом прогонки розміром до 1000×1000 і менше, більш ефективною буде послідовна реалізація алгоритму.

Відповіді на контрольні запитання:

1. Назвіть та охарактеризуйте відомі вам способи зберігання розріджених матриць.

Розріджені матриці – це матриці, у яких більшість елементів мають значення нуль, і тому немає необхідності зберігати їх всі. разрежённые матрицы могут быть легко сжаты путём записи только своих ненулевых элементов, что снижает требования к компьютерной памяти. Існує кілька способів ефективного зберігання розріджених матриць:

Компресія стовпців (CSC – Compressed Sparse Column): у цьому форматі дані зберігаються стовпцями. Три основні масиви: один для зберігання значень ненульових елементів (data), інший для зберігання номерів рядків цих елементів (row_indices), третій для зберігання позначки початку кожного стовпця в масиві значень (col_ptr). Використовується там, де матриця зберігається за стовпцями.

Компресія рядків (CSR – Compressed Sparse Row): у цьому форматі дані зберігаються рядками. Масиви для зберігання значень ненульових елементів (data),

номерів стовпців цих елементів (col_indices), та позначка початку кожного рядка в масиві значень (row_ptr). Використовується там, де матриця зберігається за рядками.

COO (Coordinate Format або triplet format): кожен ненульовий елемент представлений трійкою (рядок, стовпець, значення). Використовується для представлення розріджених матриць з невеликою кількістю ненульових елементів.

DOK (Dictionary of Keys) – словник за ключами: Зберігає лише ненульові елементи разом з їхніми індексами в словнику. Зручний для додавання нових елементів, але може бути менш ефективним для операцій над матрицями.

LIL (List of Lists) – список списків: кожен рядок зберігається як окремий список, і всі ці списки зберігаються в одному списку. Зручний для додавання нових елементів та видалення, але менш ефективний для операцій над матрицями.

Вибір конкретного формату залежить від конкретних операцій, які ви плануєте виконувати над матрицями (наприклад, додавання, множення, і так далі).

8. Назвіть та охарактеризуйте основні алгоритми оптимізації структури розріджених матриць.

Оптимізація структури розріджених матриць дуже важлива для забезпечення ефективності зберігання та операцій з такими матрицями. Основні алгоритми оптимізації структури розріджених матриць включають:

Пошук кращого формату зберігання: вибір найбільш підходящого формату зберігання, такого як CSR або CSC, залежно від операцій, які будуть виконуватися частіше. Наприклад, якщо вам частіше потрібно виконувати операції по стовпцях, CSC може бути ефективнішим.

Компактифікація матриці: видалення "зайвих" нульових елементів для зменшення розміру матриці. Це може бути корисним для зменшення обсягу пам'яті та прискорення операцій з матрицею.

Оптимізація додавання та видалення елементів: використання структур, які дозволяють ефективно додавати та видаляти елементи, таких як LIL (List of Lists) або DOK (Dictionary of Keys).

Оптимізація операцій додавання та множення: використання спеціалізованих алгоритмів для оптимізації операцій додавання та множення розріджених матриць, таких як алгоритми зіштовхування (sparse matrix multiplication algorithms).

Запаковка та розпаковка: деякі формати дозволяють ефективно згортати (pack) та розгортати (unpack) розріджені матриці для підвищення швидкодії операцій та зменшення розміру зберігання.

Оптимізація індексації: використання оптимізованих структур даних для доступу до елементів за їхніми індексами, таких як бінарний пошук або індексні структури даних.

Використання блочних структур: розгляд можливості використання блочних структур для зберігання розріджених матриць, що може сприяти оптимізації операцій.

Аналіз та визначення оптимальної структури: використання аналізу профілю використання матриці та оптимізація структури відповідно до конкретних операцій, які виконуються найчастіше.

Загальна стратегія оптимізації буде залежати від конкретного використання та характеристик матриці.

10. У чому полягає суть методу розв’язання СЛАР з розрідженою додатно-визначеною симетричною матрицею блочно-діагонального виду?

Суть методу розв'язання СЛАР з розрідженою додатно-визначеною симетричною матрицею блочно-діагонального виду полягає в застосуванні підходу, який дозволяє використовувати структурні властивості матриці для оптимізації процесу розв'язання СЛАР, враховуючи її розрідженість та блочну структуру. Цій метод може включати в себе використання специфічних алгоритмів для розв'язання систем з такими матрицями.

Основні кроки такого методу:

Структура матриці: матриця блочно-діагонального виду має блочну структуру, де кожен блок на діагоналі є матрицею. Блоки, які не є на діагоналі, можуть бути нульовими.

Розкладання матриці: Використовуючи властивості додатно-визначеної симетричної матриці, можна використовувати розкладання Холецького для матриці блочно-діагонального виду.

Розв'язання системи за допомогою розкладання: Після отримання розкладання Холецького можна використовувати його для розв'язання системи лінійних рівнянь за допомогою методів, таких як метод зведення до системи з нижче-трикутною матрицею (forward substitution) та метод зведення до системи з верхньою трикутною матрицею (backward substitution) для кожного блоку.

Використання рекурсивних методів: Застосування рекурсивних методів для розв'язання систем в кожному блоку. Це може бути ефективним, оскільки дозволяє розбити велику задачу на менші, легше розв'язувані підзадачі.

Застосування ітераційних методів: Ітераційні методи, такі як метод збільшення градієнта чи метод збільшення конюгованих градієнтів, також можуть бути використані для розв'язання СЛАР з розрідженою додатно-визначеною симетричною матрицею блочно-діагонального виду.

Конкретний метод може залежати від конкретних характеристик задачі та матриці.