ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ  
за курсом «Архітектура і програмне забезпечення»  
студента групи ПА-25-1  
Косинського Олександра Олександровича

кафедра комп’ютерних технологій, ДНУ  
2025/2026

1. Постановка задачі

Завдання: На мові С/C++ напишіть програму яка буде перемножувати квадратну матрицю розміром NxN на вектор розмірність N. Величину N виберіть таким чином, щоб максимально використати всю можливу оперативну пам’ять. Обчисліть затрати часу на дану дію. Перемножте також вектор на матрицю. Порівняйте затрачений час в обох випадках. Зробіть висновки. Запропонуйте зміну алгоритму таким чином, щоб затрачений час був приблизно однаковий.

Пояснення: Найшвидшим способом обходу є прямий послідовний. Це означає, що після звернення у програмі до деякого елемента відбувається звернення до елемента, що у пам'яті йде слідом за ним. Розглянемо розміщення у пам'яті двовимірного масиву у програмі мовою С/C++.

float A[N][N];

Відомо, що в мові С/C++ масиви розташовуються в пам'яті рядками (спочатку йдуть елементи першого рядка, потім елементи другого рядка і т.д.). Отже, у пам'яті він розміститься так:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **…** | A0,0 | A0,1 | A0,2 | … | A0,N-1 | A1,0 | A1,1 | A1,2 | … | A1,N-1 | … | AN-1,0 | AN-1,1 | AN-1,2 | … | AN-1,N-1 | **…** |

Отримуємо два варіанти перебору елементів масиву:

|  |  |
| --- | --- |
| **Швидко:**  **for** (i=0;i<N;i++)  **for** (j=0;j<N;j++) A[i][j]=x; | **Повільно:**  **for** (j=0;j<N;j++)  **for** (i=0;i<N;i++) A[i][j]=x; |
| **Швидко:**  **for** (i=0;i<N;i++)  **for** (j=0;j<N;j++) A[i][j]=x; | **Повільно:**  **for** (i=0;i<N;i++)  **for** (j=0;j<N;j++) A[j][i]=x; |

Розглянемо задачу перемноження двох квадратних матриць N×N. Якщо безпосередньо запрограмувати відому формулу: , наприклад, мовою С/C++, отримаємо наступний фрагмент програми:

for (i=0;i<N;i++)

for (k=0;k<N;k++)

for (j=0;j<N;j++) C[i][k]+=A[i][j]\*B[j][k];

Зауважимо, що у цьому випадку масив A перебирається рядками, а масив B – по стовпцям (дивимося на внутрішній цикл). Знаючи, що масиви в мові С/C++ зберігаються рядками, приходимо до висновку, що елементи масиву A перебираються послідовно, а елементи масиву B – ні. В даному випадку порядок обходу масиву C практично не важливий, оскільки між зверненнями до різних його елементів проходить багато часу. Щоб прискорити програму, потрібно, щоб принаймні у внутрішньому циклі елементи масивів перебиралися послідовно. Для цього необхідно або заздалегідь транспонувати масив B, або переставити цикли таким чином:

for (i=0;i<N;i++)

for (j=0;j<N;j++)

for (k=0;k<N;k++) C[i][k]+=A[i][j]\*B[j][k];

**За результатами виконання лабораторної роботи скласти звіт:**

**Виміряти час перемноження матриці на вектор, вектора на матрицю, а також час оптимізованого способу перемноження вектора на матрицю. Виміряти час для розмірів матриці від N=1000 до максимального розміру матриці, який вдалося розмістити в оперативній пам'яті з кроком 1000.**

**До звіту додати скріни або опис характеристик обчислювальної техніки на якій виконувався чисельний експеримент**

**Додати до звіту скріни про те, що перемноження відбувається вірно на прикладі невеликих матриць.**

**Виконати перемноження та засікти час для різних типів даних: int, float, double.**

**Створити таблиці, куди занести результати засічок часу.**

**Створити графіки, на яких відображати дані з таблиць.**

**Порівняти час перемноження різних типів даних та різних способів перемноження**

1. Опис розв’язку + блок-схема

Весь розв’язок завантажив на гитхаб https://github.com/Proggertopper/Sasha.git

1. Вихідний текст програми розв’язку задачі (основні фрагменти з коментарями)

У файлі results.txt та у вигляді таблиці Excel теж прикріпив та ось трішки з коду Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. Опис інтерфейсу програми (керівництво користувача)

Користвач чекає на результати програми, та заносить дані до таблиць

1. Опис текстових прикладів

Прикріпив у файлі results.txt

1. Аналіз помилок (опис усунення зауважень)

**bad\_alloc / аварійне завершення для великих N**

**при виділенні двовимірного масиву пам’ять закінчується → bad\_alloc або аварійна зупинка.  
Можна ловити bad\_alloc, зменшувати N або переходити на варіант без збереження A (генерація на льоту) або використовувати пам’ять на диску (memory-mapped files) — складніше.**

**Неправильні результати при вимірюванні часу**

**використання clock() повертає CPU time, ОС може надавати невелику точність; також інші процеси впливають на час.  
Тому використовувати більш точні таймери (у Windows — QueryPerformanceCounter) або запускати кожну операцію кілька разів і брати середнє. Вимикати зайві процеси**

**Використання rand() уповільнює ініціалізацію**

**генерація випадкових чисел займає час і впливає на виміри для малих N.  
тому при потребі використовувати fast deterministic generator (LCG), або генерувати значення по формулі (i+j) % K для тестів продуктивності.**

1. Висновки

**Локальність пам’яті має вирішальне значення для швидкості програм з великими масивами. Операції, що читають матрицю рядками (як вона лінійно зберігається в C/C++), працюють значно швидше, ніж ті, що читають стовпцями.**

**Транспонування матриці перед множенням або перестановка циклів — потужні прийоми для приведення доступу до послідовного і підвищення продуктивності. Транспонування вимагає додаткової пам’яті та часу, але у великих задачах швидкість множення після транспонування компенсує затрати.**

**Для великих розмірів N зберігання A[N][N] призводить до експоненціального росту пам’яті (O(N²)) та швидкого досягнення bad\_alloc. Щоб тестувати великі N (наприклад, 46000), потрібно або мати сервер з великим об’ємом RAM, або переходити до генерації значень на льоту або використовувати зовнішню пам’ять.**

**Тип даних впливає: int зазвичай дає найшвидші операції, double — найповільніші через більший об’єм пам’яті і операційні витрати. Вимірювання показують різницю, але головним фактором є кеш-локальність, а не тільки розмір типу.**

**Щоб забезпечити приблизно однаковий час для A\*x та x\*A, потрібно привести доступи до пам’яті до однакової локальності — наприклад, транспонувати матрицю для x\*A або переписати алгоритм (зміна порядку циклів). Тоді обидва множення можуть мати близьку продуктивність.**

1. **Характеристики пристрою, на якому проводилася лабораторна работа**

**Изображение выглядит как текст, снимок экрана, Шрифт, документ

Содержимое, созданное искусственным интеллектом, может быть неверным.**