Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Кваліфікаційна робота

Освітній ступінь – магістр

На тему «**АЛГОРИТМ ЗВЕДЕННЯ СИМЕТРИЧНОЇ МАТРИЦІ ДО ТРИДІАГОНАЛЬНОЇ ІЗ ЗАСТОСУВАННЯМ QR ТА QP РОЗКЛАДІВ»**

Виконав: студент 2-го року навчання
освітньої програми «Комп'ютерні науки»,
спеціальності 122 Комп'ютерні науки

Першута Павло Володимирович

Керівник: Малашонок Геннадій Іванович
доктор фіз-мат наук, професор.

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____

«____» _____ 2024 р.

Київ - 2024

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики
кандидат фіз-мат наук, доцент
Гороховський Семен Самуїлович
_____
(підпис)
“_____” _____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на кваліфікаційну роботу

студенту 2 р.н. магістерської програми Комп’ютерні науки
Першуті Павлу Володимировичу
Розробити Алгоритм зведення симетричної матриці до тридіагональної із застосуванням QR та QP розкладів.

Зміст текстової частини кваліфікаційної роботи:

Зміст
Анотація
Вступ
1. Рекурсивний алгоритм QR розкладу
2. Алгоритм приведення матриці до тридіагональної з використанням QR та QP розкладів
3. Середовище блоково-рекурсивних паралельних обчислень DAP
4. Оптимізація алгоритму QR розкладу
5. Результати тестування алгоритмів
Висновки
Список використаної літератури
Додатки

Дата видачі “_____” _____ 2023 р.

Керівник
Г.І. Малашонок, доктор фіз-мат наук, професор _____
(підпис)

Завдання отримав
П.В. Першута _____
(підпис)

Тема: Алгоритм зведення симетричної матриці до тридіагональної із застосуванням QR та QP розкладів

Календарний план виконання роботи:

№ п/п	Назва етапу дипломної роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу	31.10.2023	
2.	Огляд технічної літератури за темою роботи	20.11.2023	
3.	Відлагодження алгоритмів QR та QP розкладів у середовищі DAP02	16.01.2023	
4.	Оптимізація алгоритмів QR та QP розкладів	16.02.2023	
5.	Тестування алгоритмів QR та QP розкладів	05.03.2024	
6.	Формування алгоритмів Q3RP та Ribbon	13.03.2024	
7.	Програмна реалізація алгоритму Q3RP в DAP02	30.03.2024	
8.	Тестування алгоритму Q3RP	11.04.2024	
9.	Написання текстової частини роботи	12.05.2024	
10.	Попередній захист роботи	17.05.2024	
11.	Коригування роботи за результатами попереднього захисту	28.05.2024	
12.	Остаточне оформлення текстової частини роботи та презентації	5.06.2024	
13.	Захист магістерської роботи	12.06.2024	

Студент Першута Павло Володимирович

Керівник Малашонок Геннадій Іванович “____”_____ 2024 р.

Зміст

Анотація

У роботі розглянуто блоково-рекурсивні алгоритми *QR* та *QP* розкладів та запропоновано алгоритм зведення симетричної матриці до тридіагональної на основі *QR* та *QP*. Розроблено та протестовано паралельний блочний алгоритм *Q3RP*-розкладу. Проаналізовано отримані результати та зроблено висновки про подальший розвиток дослідження.

Ключові слова: *QR-розклад, QP-розклад, матриці Ґівенса, паралельні обчислення, тридіагональна матриця.*

Вступ

Збільшення об’єму даних, що потребують обробки та аналізу зростає кожного дня. Велика складність проведення матричних обрахунків, що є основним методом роботи з даними, є основною причиною пошуків оптимальних та стабільних способів обробки інформації та її структуризації.

Останні тенденції розвитку систем штучного інтелекту яскраво відображають їх корисність у всіх сферах людської діяльності. В свою чергу в основою більшості алгоритмів моделей є операція матричного множення. Оскільки складність самого множення $O(n^3)$ вже є доволі дорого вартісною, навіть розв’язання матричних рівнянь виду $Ax = B$ потребує значних обчислювальних потужностей для виконання обрахунків на матрицях великого розміру. Завдяки проведеним дослідженням степеневу складність множення вдалось зменшити. Метод запропонований Штрассеном [1] потребує виконання $4.7 * O(n^{log_2 7})$ чисельних операцій, що дає пришвидшення на великих щільних матрицях. На даний момент найшвидшим є метод Копперсміта - Винограда зі складністю $C * O(n^{2.38})$ [2], однак через велике значення константи C прискорення можна відчути лише на дуже великих щільних матрицях.

Для ефективнішої роботи з матрицями використовують підхід розкладу матриць для збільшення зручності та спрощення обрахунків. Ця робота побудована на одному з таких розкладів, а саме QR. Цей розклад широко використовується для розв’язку систем лінійних рівнянь, алгоритмів машинного навчання, рендерингу, обробки сигналів і т. д. QR-розклад є чисельно-стабільним, тобто, зберігає точність обчислень при використанні чисел з плаваючою точкою та наявності заокруглень, які можуть виникати в процесі обчислень на комп’ютері.

Паралельні обчислення відіграють важливу роль в розробці сучасних систем. За їх допомогою можна вирішувати завдання, які вимагають великих обчислювальних ресурсів або обробляють величезні обсяги даних, наприклад, у галузі наукових досліджень, біоінформатики, машинного навчання та аналізу великих даних. Активна розробка нових багатоядерних систем та кластерів спонукає до пошуків ефективніших методів використання наявних ресурсів для отримання найкращої продуктивності обчислювальних систем. Паралельне виконання операцій з матрицями є основним напрямком досліджень спрямованим на зменшення часу виконання чисельних операцій та раціонального використанням наявних процесорів системи.

Ця робота є логічним продовженням дослідження паралельного блоково-рекурсивного алгоритму *QR*-розкладу [6]. Метою дослідження є оптимізація запропонованого алгоритму використанням методу Штрассена для множення матриць та його застосування для зведення симетричної матриці до тридіагональної.

У першому розділі цієї роботи розглянемо алгоритм паралельного блочно-рекурсивного алгоритму *QR*-розкладу на основі матриць Ґівенса .

У другому розділі запропонуємо алгоритм застосування блоково - рекурсивного алгоритму *QR*-розкладу для зведення симетричної матриці до тридіагональної.

У третьому розділі розглянемо децентралізоване середовище блоково - рекурсивних паралельних обчислень DAP.

У четвертому розділі розглянемо спосіб оптимізації алгоритму *QR*-розкладу із застосуванням методу швидкого множення матриць Штрассена.

У п’ятому розділі наведено результати тестування оптимізованого блоково-рекурсивного алгоритму *QR*-розкладу та *Q3RP*-розкладу.

Розділ 1. Рекурсивний алгоритм QR-розкладу

1.1. Поняття QR-розкладу

QR-розкладом матриці називається представлення матриці A у вигляді добутку ортогональної матриці Q та верхньої трикутної матриці R: $A = QR$. Окрім того, пара (Q, R) є унікальною, тобто, не існує інших таких Q та R добутком яких є A. Основна ідея *QR*-факторизації полягає в тому, щоб звести лінійну систему до трикутного вигляду.

Розмір матриць Q та R залежить від розміру матриці A. Для матриці A розміром $m \times n$ матрицею Q буде квадратна ортогональна матриця розміром $m \times m$, тобто, буде справедливим вираз $Q^T Q = I$, де I - одинична матриця. Матрицею R розміром $m \times n$ є верхня трикутна матриця, це означає, що всі її елементи під головною діагоналлю є нулями. В роботі ми використовували лише квадратні матриці дійсних чисел розміру $2^N \times 2^N$.

Один із способів застосування *QR*-розкладу це розв'язання матричних рівнянь виду $Ax = B$. маючи *QR*-розклад матриці A потрібно виконати наступні дії:

$$(1)\ Ax = B$$

$$(2)\ QRx = B$$

$$(3)\ \begin{cases} Rx = y \\ Qy = B \end{cases}$$

$$(4)\ Q^T Qy = Q^T B, Q^T Q = I$$

$$(5)\ y = Q^T B$$

$$(6)\ Rx = Q^T B$$

Існує кілька способів знаходження *QR*-розкладу: метод поворотів Ґівенса [3], метод Грама-Шмідта [4], перетворення Хаусхолдера [5]. В роботі

ми використовували саме метод поворотів Ґівенса, зважаючи на його універсальність, ефективність, числову стабільність та відносну простоту паралелізації, що є важливим фактором в нашому дослідженні.

1.2. Матриця Ґівенса

Нехай $\theta \in R$ буде градусом, на який потрібно виконати поворот вектора, а p та q цілі числа та є індексами відповідних рядків та стовпчиків таких, що $q \neq p$ та $\in [1, n]$. Тоді матриця повороту Ґівенса $G(p, q, \theta)$ визначається значеннями $G_{p,q}(p, q, \theta)$:

$$G_{p,p}(p, q, \theta) = \cos\theta$$
$$G_{q,q}(p, q, \theta) = \cos\theta$$
$$G_{p,q}(p, q, \theta) = \sin\theta$$
$$G_{q,p}(p, q, \theta) = -\sin\theta$$
$$G_{i,i}(p, q, \theta) = 1, i \neq p, i \neq q$$

В загальному випадку матриця Ґівенса $G(p, q, \theta)$ матиме наступний вигляд:

$$G(p,q,\theta) = \begin{bmatrix} 1 & & & & \cdots & & & 0 \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & \cos\theta & & \sin\theta & & \\ & & & & 1 & & & \\ \vdots & & & & \ddots & & & \vdots \\ & & & & & 1 & & \\ & & & -\sin\theta & & \cos\theta & & \\ & & & & & & 1 & \\ & & & & & & & \ddots \\ 0 & & & & \cdots & & & 1 \end{bmatrix}$$

Розглянемо приклад застосування матриці повороту для QR розкладу матриці A розміром 2×2:

$$A = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}, \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} = \begin{pmatrix} a & b \\ 0 & d \end{pmatrix},$$

Де s та c мають задовільняти рівність $s^2 + c^2 = 1$. Якщо y = 0, то c = 1, s = 0. Якщо $\gamma \neq 0$, то $s = \frac{\alpha}{\sqrt{a^2+\gamma^2}}$, $c = \frac{\gamma}{\sqrt{a^2+\gamma^2}}$

Розглянемо частковий випадок матриці повороту Ґівенса для матриці *A*, в якій всі елементи ліворуч від *(p, q)* та *(p+1, q)* нульові. Визначимо матрицю повороту:

$$G(p,q,\theta) = \begin{bmatrix} 1 & & & & & & & & 0 \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & \cos\theta & \sin\theta & & & & \\ & & & -\sin\theta & \cos\theta & & & & \\ & & & & & 1 & & & \\ & & & & & & \ddots & & \\ 0 & & & & & & & & 1 \end{bmatrix}$$

Результат множення матриці $G_{p,q}A$ відрізнятиметься від початкової матриці *A* лише рядками *p* та *p+1*, водночас всі нульові елементи ліворуч стовпця *q* залишаться нульовими.

1.3. Послідовний алгоритм QR-розкладу методом Ґівенса

Ідея алгоритму полягає в постовпчиковому зануленні всіх елементів під основною діагоналлю використовуючи матрицю поворотів. Матрицею повороту, що зануляє k-й стовпчик, буде добуток відповідних матриць Ґівенса $G_{k,k}G_{k+1,k} \dots G_{n-1,k}$. Ітеративно обнуляючи елементи кожного стовпчика під основною діагоналлю матриці ми зведемо матрицю *A* до трикутної, яка є матрицею *R*, а добуток всіх матриць G_k є ортогональною матрицею *Q*. В загальному вигляді для матриці $A_{n\times n}$ можна записати:

(1) $G_{(k)} = G_{k,k}G_{k+1,k} \dots G_{n-1,k}$, для $k \in [1, n-1]$

(2) $A_{(k)} = G_{(k)}A_{(k-1)}$

Для обчислення *QR*-розкладу таким способом потрібно виконати *n – 1* ітерацій. Порахуємо кількість операцій, що буде виконана під час роботи

алгоритму. Для обрахунку коефіцієнтів матриці повороту потрібно виконати 6 операцій: два піднесення до квадрату, одне додавання, один корінь квадратний та два ділення. Таких матриць буде $\frac{n^2-n}{2}$. Загальна кількість множень матриці повороту на стовпець буде $O(n^3)$, а складність всього алгоритму $\approx 2n^3$ [3].

1.4. Рекурсивний QR-розклад

Ідея блоково-рекурсивного алгоритму *QR*-розкладу була запропонована Малашонком у статті «Recursive matrix algorithms, distributed dynamic control, scaling, stability» [7]. Оскільки множення матриці повороту на початкову матрицю змінює лише рядки де в матриці повороту знаходиться матриця удару, то алгоритм розкладу можна виконувати рекурсивно та паралельно. Схема блокового-рекурсивного алгоритму зображена на рисунку 1. Малашонком також було описано додатковий алгоритм *QP*-розкладу [7]. Далі розглядатимемо матриці лише розміру $2^N \times 2^N$.

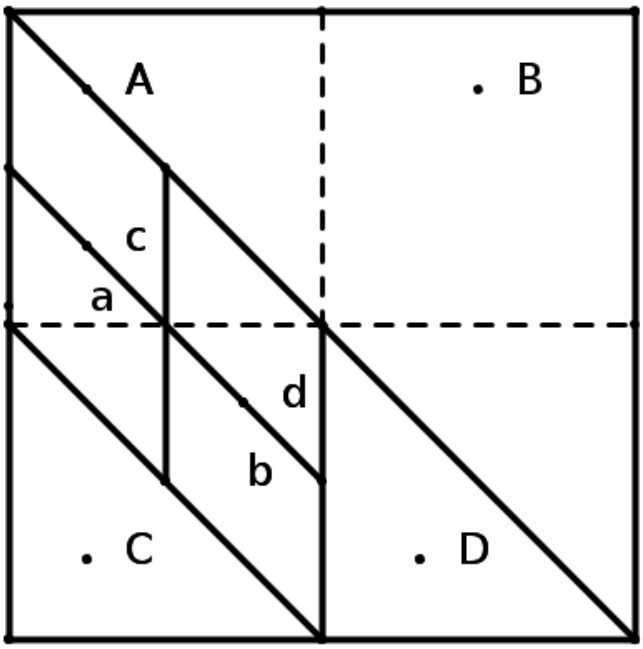

Рисунок 1. Схема блоково-рекурсивного QR-розкладу

Нехай матриця $M_{2^N \times 2^N} = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, тоді першим кроком є виконання *QR*-розкладу блоку *C*:

$$(1)\mathrm{C} = Q_1 C^U,\ M_1 = diag(I, Q_1)M = \begin{pmatrix} A & B \\ C^U & D_1 \end{pmatrix}$$

Другий крок – занулення паралелограму *abcd*. Оскільки паралелограм можна розбивати на менші, то його також можна обчислювати рекурсивно.

$$(2)\, Q_2 \begin{pmatrix} A \\ C^U \end{pmatrix} = \begin{pmatrix} A_1^U \\ 0 \end{pmatrix},\ M_2 = Q_2 M_1 = \begin{pmatrix} A_1^U & B_1 \\ 0 & D_2 \end{pmatrix}$$

Третій крок – *QR*-розклад блоку D_2:

$$(3)\, D_2 = Q_3 D^U, M_3 = diag(I, Q_3)M_2 = \begin{pmatrix} A_1^U & B_1 \\ 0 & D_3^U \end{pmatrix}$$

В результаті отримаємо:

$$(4)\, QM = R, M = Q^T R, \text{де } Q = diag(I, Q_3)Q_2 diag(I, Q_1)$$

1.5. Рекурсивний QP-розклад

Нехай матриця $P = \begin{pmatrix} A \\ C^U \end{pmatrix}$ та має розмір $2n \times n$ та сформована з лівих блоків початкової матриці *M*, тоді справедливим буде розбиття на наступні блоки:

$$P = \begin{pmatrix} A \\ C^U \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \\ e^U & f \\ 0 & h^U \end{pmatrix},$$

Метою розкладу є зведення матриці P до виду $P = \begin{pmatrix} A^U \\ 0 \end{pmatrix}$. Для цього розіб’ємо матрицю на 8 блоків. Оскільки ми розглядали лише матриці розміру

$2^N \times 2^N$, то всі блоки будуть однакового розміру, окрім того ми вже маємо один нульовий блок та два верхньо-трикутні блоки e^U та h^U.

Першим кроком є занулення блоку e^Uта зведення блоку c до верхньо трикутного виду:

$$(1)\ Q_{c,e^u}P = Q_{c,e^u}\begin{pmatrix} c \\ e^U \end{pmatrix} = \begin{pmatrix} c^u \\ 0 \end{pmatrix}$$

Далі можна одночасно зробити занулення блоків $\begin{pmatrix} a \\ c^u \end{pmatrix}$ та $\begin{pmatrix} f_1 \\ h^u \end{pmatrix}$, оскільки вони не перетинаються та їх матриці поворотів не будуть впливати на чужі елементи матриці:

$$(2)\, Q_{a,c^u}\ P_1 = Q_{a,c^u}\ \begin{pmatrix} a \\ c^U \end{pmatrix} = \begin{pmatrix} a^u \\ 0 \end{pmatrix}$$

$$(3)\, Q_{f_1,h^u}\ P_1 = Q_{f_1,h^u}\begin{pmatrix} f_1 \\ h^u \end{pmatrix} = \begin{pmatrix} f_2^U \\ 0 \end{pmatrix},$$

Наступним кроком є обнулення останніх двох блоків паралелограма $\begin{pmatrix} d_2 \\ f_2^U \end{pmatrix}$:

$$(4)\ Q_{d_2,f_2^U}P_2 = \ Q_{d_2,f_2^U}\begin{pmatrix} d_2 \\ f_2^U \end{pmatrix} = \begin{pmatrix} d_3^U \\ 0 \end{pmatrix}$$

Як результат отримаємо розклад $QP = P^U$, де $Q = \ Q_4 Q_{2,3} Q_1$, при чому $Q_1 \ = diag(I, Q_{c,e^u}, I)$, $Q_2 = diag(Q_{a,c^u}, Q_{f_1,h^u})$, $Q_3 = diag(I, Q_{d_2,f_2^U}, I)$.

1.6. Складність блоково-рекурсивного QP-розкладу

Потрібно виконати 28 множень для матричних блоків розміру *n/2 × n/2*. Звідси випливає, що загальна кількість операцій $Cp(2n) = 4Cp(n) + 28M\left(\frac{n}{2}\right)$. Нехай для того, щоб помножити дві матриці розміру *n × n* необхідно виконати γn^β операцій та $n = \ 2^k$, тоді отримаємо [6]:

$$Cp(2^{k+1}) = 4Cp(2^k) + 28M(2^{k-1}) = 4^k Cp(2) + 28\gamma \sum_{i=0}^{k-1} 4^{k-i-1} 2^{i\beta}$$

$$= 28\gamma \left(\frac{n^2}{4}\right) \frac{2^{k(\beta-2)} - 1}{2^{(\beta-2)} - 1} + 6n^2 = 7\gamma \frac{n^\beta - n^2}{2^b - 4} + 6n^2$$

$$Cp(n) = \frac{7\gamma n^b}{2^\beta (2^\beta - 4)} + \frac{3n^2}{2} \left(3 - \frac{7\gamma}{2^{\beta+1} - 8}\right)$$

1.7. Складність рекурсивного QR-розкладу

Для оцінки кількості операцій в блоково-рекурсивному алгоритму *QR*-розкладу припустимо, що складність матричного множення дорівнює γn^β, а складність *QP*-розкладу = αn^β, де α, β, γ константи та $n = 2^k$. Виразимо α з попередньо описаної складності *QP*. Звідси маємо, що $\alpha = \frac{7\gamma}{2^\beta (2^\beta - 4)}$ [6].

$$C(n) = 2C\left(\frac{n}{2}\right) + Cp(n) + 6M\left(\frac{n}{2}\right) = 2C(2^{k-1}) + Cp(2^k) + 6M(2^{k-1}) =$$

$$C(2^0)2^k + \sum_{i=0}^{k} 2^{k-i} Cp\left(2^i\right) + 6\sum_{i=0}^{k} 2^{k-i} M\left(2^{i-1}\right) = \alpha \sum_{i=0}^{k} 2^{k-i} 2^{i\beta} + 6\gamma \sum_{i=0}^{k} 2^{k-i} 2^{(i-1)\beta} =$$

$$\left(\alpha 2^k + 6\gamma 2^{k-\beta}\right) \sum_{i=0}^{k} 2^{i(\beta-1)} = \left(\alpha + 6\gamma 2^{-\beta}\right) \frac{2^\beta n^\beta - 2n}{2^\beta - 2} = \frac{\gamma (2^\beta 6 - 17)(n^\beta - \frac{2n}{2^\beta})}{(2^\beta - 4)(2^\beta - 2)}$$

Розділ 2. Застосування блоково-рекурсивних алгоритмів QR та QP розкладів для приведення симетричної матриці до тридіагонального виду

2.1. Ідея алгоритму та його етапи

Розглянемо алгоритм приведення симетричної матриці M розміру $2^N \times 2^N$ до тридіагонального виду застосовуючи вище описані алгоритми QR та QP розкладів. Основна ідея застосування цих розкладів для симетричних матриць полягає у відсутності необхідності перераховувати матрицю повороту для відповідного блоку, оскільки матриця повороту для симетричного блоку є транспонованою матрицею повороту.

Алгоритм зведення матриці до тридіагональної складається з двох етапів: занулення елементів нижнього лівого та верхнього правого трикутника залишивши лише смугу елементів шириною p за допомогою QR та QP розкладів та ітеративного звужування смуги до досягнення тридіагонального виду.

Для того щоб залишити лише смугу з ненульових елементів матриці необхідно порахувати три QR та три QP розклади. На рисунку 2, жирними лініями виділено трикутники та паралелограми, для яких необхідно знайти відповідні матриці повороту щоб занулити елементи, а пунктирними лініями у верхньому правому трикутнику позначено симетричне відображення тих самих трикутників та паралелограмів, оскільки за умовою матриця є симетричною. Для занулення симетричних елементів можемо використати транспоновані матриці поворотів обчислених при зануленні блоків $R_1, R_2, R_3, P_1, P_{11}, P_2$.

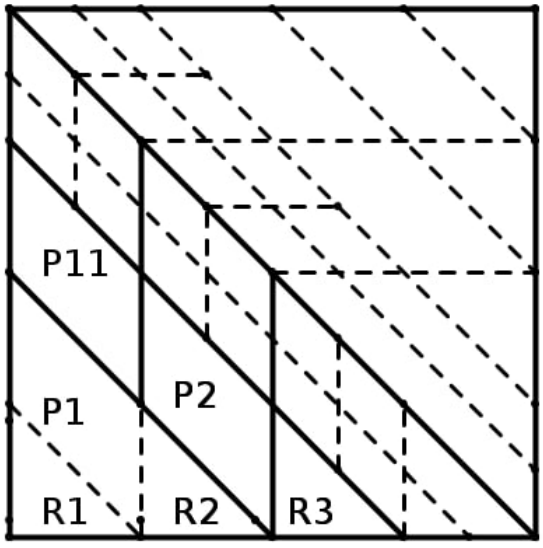

Рисунок 2. Схема розбиття симетричної матриці на блоки для Q3RP

Другим етапом алгоритму (рисунок 3) є ітеративне звуження смуги шириною p, отриманої після першого кроку. Звуження відбувається шляхом занулення зовнішньої половини смуги, яка складається з $2k-2$ блоків типу P (паралелограм), розміром $2p \times p$ та двох блоків типу R (трикутник). Значення змінної k дорівнює 4 під час першої ітерації і кожну наступну ітерацію збільшується вдвічі.

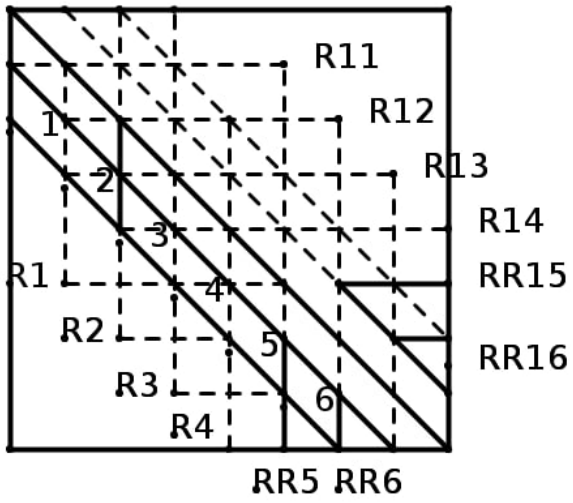

Рисунок 3. Розбиття симетричної матриці на блоки для алгоритму Ribbon

Множення вхідної матриці на отриману матрицю Q зліва та справа, породжуватиме нові не занулені елементи, тому виникає необхідність додаткового очищення цих елементів. Наприклад, множення матриці Q_{p1} на смугу зліва породжує новий не нульовий трикутник R_{11}, а множення смуги на Q_{p1}^T справа породжує не нульовий трикутник R_1, що є симетричним до R_{11}. Для занулення R_1 необхідно використати QR-розклад та домножити матрицю на отриману транспоновану матрицю повороту справа для занулення R_{11}.

Наступними кроками є аналогічне занулення блоків P та не нульових трикутників, породжених домноженням матриці повороту на смугу.

Передостанній та останній крок відрізняються від попередніх тим, що породжений трикутник має розмір $p \times p$ на відміну від попередніх, що мали розмір $2p \times 2p$.

2.2. Q3RP-розклад

Розіб'ємо вхідну матрицю M на шістнадцять блоків наступним чином, тоді можемо сформувати наступний алгоритм:

$$(1)\, M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & b_{1,1} & b_{1,2} \\ a_{2,1} & a_{2,2} & b_{2,1} & b_{2,2} \\ c_{1,1} & c_{1,2} & d_{1,1} & d_{1,2} \\ c_{2,1} & c_{2,2} & d_{2,1} & d_{2,2} \end{pmatrix}$$

Першим кроком є знаходження QR розкладу блоку $\mathrm{c}_{2,1}$, отримана матриця повороту Q_1 буде використана для занулення симетричних елементів блоку $b_{1,2}$:

$$(2)\ QR(c_{2,1}) = Q_1 c_{2,1}^U$$

Другим кроком є знаходження *QP*-розкладу для повного занулення блоку $c_{2,1}^{U}$:

$$(3)\, QP \begin{pmatrix} c_{1,1} \\ c^{U}{}_{2,1} \end{pmatrix} = Q_2 \begin{pmatrix} c^{U}{}_{1,1} \\ 0 \end{pmatrix}$$

Перемножимо отримані матриці поворотів щоб отримати загальну матрицю для занулення блоку $c_{2,1}^{U}$, Q_3 та домножимо оригінальну матрицю зліва на Q_3^T та справа на Q_3:

$$(4)\, Q_3 = Q_2 \times diag(I, Q_1)$$

$$(5)\, M_1 = diag(I, Q_3)^T \times M = \begin{pmatrix} a_{1,1} & a_{1,2} & b_{1,1} & b_{1,2} \\ a_{2,1} & a_{2,2} & b_{2,1} & b_{2,2} \\ c^{U}{}_{1,1} & c'{}_{1,2} & d'{}_{1,1} & d'{}_{1,2} \\ 0 & c'{}_{2,2} & d'{}_{2,1} & d'{}_{2,2} \end{pmatrix}$$

$$(6)\, M_2 = M_1 \times diag(I, Q_3) = \begin{pmatrix} a_{1,1} & a_{1,2} & (b_{1,1})_U & 0 \\ a_{2,1} & a_{2,2} & b'{}_{2,1} & b'{}_{2,2} \\ c^{U}{}_{1,1} & c'{}_{1,2} & d''{}_{1,1} & d''{}_{1,2} \\ 0 & c'{}_{2,2} & d''{}_{2,1} & d''{}_{2,2} \end{pmatrix}$$

Наступним кроком є повне занулення блоків $c^{U}{}_{1,1}$ та симетричного йому $(b_{1,1})_U$. Для цього необхідно застосувати *QP*-розклад та домножити матрицю M_2 на отриману матрицю повороту зліва ти справа:

$$(7)\, QP \begin{pmatrix} a_{2,1} \\ c^{U}{}_{1,1} \end{pmatrix} = Q_4 \begin{pmatrix} a^{U}{}_{2,1} \\ 0 \end{pmatrix}$$

$$(8)\, M_3 = diag(I, Q_4, I)^T \times M_2 = \begin{pmatrix} a_{1,1} & (a_{1,2})_U & (b_{1,1})_U & 0 \\ a^{U}{}_{2,1} & a'{}_{2,2} & b''{}_{2,1} & b''{}_{2,2} \\ 0 & c''{}_{1,2} & d'''{}_{1,1} & d'''{}_{1,2} \\ 0 & c'{}_{2,2} & d''{}_{2,1} & d''{}_{2,2} \end{pmatrix}$$

$$(9)\, M_4 = M_3 \times diag(I, Q_4, I) = \begin{pmatrix} a_{1,1} & (a_{1,2})_U & 0 & 0 \\ a^U{}_{2,1} & a''{}_{2,2} & b''{}_{2,1} & b''{}_{2,2} \\ 0 & c'''{}_{1,2} & d''''{}_{1,1} & d'''{}_{1,2} \\ 0 & c''{}_{2,2} & d'''{}_{2,1} & d''{}_{2,2} \end{pmatrix}$$

Аналогічно занулимо блок $c''{}_{2,2}$, а $c'''{}_{1,2}$ зведемо до верхньотрикутного вигляду використавши лише один *QR* та один *QP* розклади:

$$(10)\quad QR(c''{}_{2,2}) = Q_5 c^U_{2,2}$$

$$(11)\quad QP\begin{pmatrix} c'''{}_{1,2} \\ c^U_{2,2} \end{pmatrix} = Q_6 \begin{pmatrix} c^U{}_{1,2} \\ 0 \end{pmatrix}$$

Знайдемо загальну матрицю повороту для занулення блоку $c''{}_{2,2}$ перемноживши отримані матриці Q_5 та Q_6 :

$$(12)\quad Q_7 = Q_6 \times diag(I, Q_5)$$

Після множення матриці M_4 на отриману матрицю повороту зліва та справа отримаємо:

$$(13)\quad M_5 = diag(I, Q_7)^T \times M_4 = \begin{pmatrix} a_{1,1} & (a_{1,2})_U & 0 & 0 \\ a^U{}_{2,1} & a''{}_{2,2} & b''{}_{2,1} & b''{}_{2,2} \\ 0 & c^U{}_{1,2} & d'''''{}_{1,1} & d''''{}_{1,2} \\ 0 & 0 & d''''{}_{2,1} & d'''{}_{2,2} \end{pmatrix}$$

$$(14)\quad M_6 = M_5 \times diag(I, Q_7) = \begin{pmatrix} a_{1,1} & (a_{1,2})_U & 0 & 0 \\ a^U{}_{2,1} & a''{}_{2,2} & (b_{2,1})_U & 0 \\ 0 & c^U{}_{1,2} & d''''''{}_{1,1} & d'''''{}_{1,2} \\ 0 & 0 & d'''''{}_{2,1} & d''''{}_{2,2} \end{pmatrix}$$

Останнім етапом є *QR*-розклад блоку $d'''''{}_{2,1}$ та множення матриці M_3 на отриману матрицю повороту зліва та справа:

$$(15)\quad QR(d'''''{}_{2,1}) = Q_8 d^U_{2,1}$$

$$(16)\quad M_7 = diag(I, Q_8)^T \times M_6 = \begin{pmatrix} a_{1,1} & (a_{1,2})_U & 0 & 0 \\ a^U{}_{2,1} & a''{}_{2,2} & (b_{2,1})_U & 0 \\ 0 & c^U{}_{1,2} & d''''''{}_{1,1} & d'''''{}_{1,2} \\ 0 & 0 & d^U_{2,1} & d''''''{}_{2,2} \end{pmatrix}$$

$$(17)\quad M_8 = M_7 \times diag(I, Q_8) = \begin{pmatrix} a_{1,1} & (a_{1,2})_U & 0 & 0 \\ a^U{}_{2,1} & a''{}_{2,2} & (b_{2,1})_U & 0 \\ 0 & c^U{}_{1,2} & d''''''{}_{1,1} & (d_{1,2})_U \\ 0 & 0 & d^U_{2,1} & d''''''{}_{2,2} \end{pmatrix}$$

В результаті отримаємо розклад вхідної матриці на ліву матрицю повороту Q_L, матрицю із смугою шириною *p*, та праву матрицю повороту Q_R, де матриці поворотів є добутками матриць *Q,* отриманих в результаті *QR* та *QP* розкладів:

$$Q_L = diag(I, Q_8)^T \times diag(I, Q_7)^T \times diag(I, Q_4, I)^T \times diag(I, Q_3)^T$$

$$Q_R = diag(I, Q_3) \times diag(I, Q_4, I) \times diag(I, Q_7) \times diag(I, Q_8)$$

Використовуючи значення Q_L та Q_R, можемо виразити *Q3PR*-розклад наступним чином:

$$Q_L \times M \times Q_R = \begin{pmatrix} a_{1,1} & (a_{1,2})_U & 0 & 0 \\ a^U{}_{2,1} & a''{}_{2,2} & (b_{2,1})_U & 0 \\ 0 & c^U{}_{1,2} & d''''''{}_{1,1} & (d_{1,2})_U \\ 0 & 0 & d^U_{2,1} & d''''''{}_{2,2} \end{pmatrix},$$

$$M = Q_L^T \times \begin{pmatrix} a_{1,1} & (a_{1,2})_U & 0 & 0 \\ a^U{}_{2,1} & a''{}_{2,2} & (b_{2,1})_U & 0 \\ 0 & c^U{}_{1,2} & d''''''{}_{1,1} & (d_{1,2})_U \\ 0 & 0 & d^U_{2,1} & d''''''{}_{2,2} \end{pmatrix} \times Q_R^T$$

2.3. Складність Q3RP-розкладу

Нехай дано симетричну матрицю M розміру $2^k \times 2^k$. Для оцінки складності запропонованого алгоритму використаємо складності *QR* та *QP* розкладів описаних у розділах 1.6 та 1.7 та припустимо, що складність матричного множення дорівнює γn^β. Підрахуємо загальну кількість дій, яку необхідно виконати: три *QR*-розклади для матриці розміром $n = \frac{2^k}{4}$, три *QR*-розклади для матриці розміром $n = \frac{2^k}{4}$, та 14 матричних множень розміру $n = \frac{2^k}{2}$, та два матричних множення $n = \frac{2^k}{4}$.

$$C(n) = 3C_{qr}\left(\frac{n}{4}\right) + 3C_{qp}\left(\frac{n}{4}\right) + 14M\left(\frac{n}{2}\right) + 2M(\frac{n}{4})$$

$$C(n) = 3\,\frac{\gamma(2^\beta 6 - 17)((\frac{n}{4})^\beta - \frac{(\frac{n}{2})}{2^\beta})}{(2^\beta - 4)(2^\beta - 2)} + 3\frac{7\gamma(\frac{n}{4})^b}{2^\beta(2^\beta - 4)} + 14\gamma(\frac{n}{2})^\beta + 2y\left(\frac{n}{4}\right)^\beta$$

$$C(n) = \frac{3\gamma\left(2^\beta 6 - 17\right)\left(\left(\frac{n}{4}\right)^b - 2^{\beta-1}n\right)}{(2^\beta - 4)(2^\beta - 2)} + \frac{21\gamma(\frac{n}{4})^b}{2^\beta(2^\beta - 4)} + 14\gamma(\frac{n}{2})^\beta + 2y\left(\frac{n}{4}\right)^\beta$$

Нехай $\gamma = 1$, а $\beta = 3$, то:

$$C(n) = \frac{3(2^3 6 - 17)\left(\left(\frac{n}{4}\right)^3 - 2^2 n\right)}{(2^3 - 4)(2^3 - 2)} + \frac{21\left(\frac{n}{4}\right)^3}{2^3(2^3 - 4)} + 14\left(\frac{n}{2}\right)^3 + 2\left(\frac{n}{4}\right)^3 =$$

$$= \frac{31\left(\frac{n^3 - 256n}{64}\right)}{8} + \frac{21n^3}{2} + \frac{14n^3}{8} + \frac{2n^3}{64} \approx \frac{790n^3 - 1024n}{64} \approx 12.34n^3 - 16n$$

2.4. Алгоритм звуження стрічки Ribbon

Нехай M є симетричною матрицею розміром $2^n \times 2^n$, в якої всі елементи крім елементів діагональної смуги шириною p нульові. Розглянемо першу ітерацію алгоритму, тобто, $k = 4$ і для завершення основного алгоритму необхідно виконати шість QP та два QR розклади для матриці розміру 8 :

$$(1)\, M = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & & & & & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & & & & \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & & & \\ & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & & \\ & & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & \\ & & & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} \\ & & & & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} \\ 0 & & & & & a_{8,6} & a_{8,7} & a_{8,8} \end{pmatrix}$$

Першим кроком є занулення елемента $a_{3,1}$. Для цього використаємо QP-розклад:

$$(2)\ QP \begin{pmatrix} a_{2,1} \\ a_{3,1} \end{pmatrix} = Q_1 \begin{pmatrix} a'_{2,1} \\ 0 \end{pmatrix}$$

Домножимо початкову матрицю на Q_1 зліва та на Q_1^T справа. Очевидно, що в результаті множення зліва змін зазнають лише другий та третій рядки, а при множенні справа лише другий та третій стовпці, при чому нульові елементи $a_{2,5}$ та $a_{5,2}$, стануть не нульовими.

$$(3)\ Q_1 \times \begin{pmatrix} a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \end{pmatrix} = \begin{pmatrix} a'_{2,2} & a'_{2,3} \\ a'_{3,2} & a'_{3,3} \end{pmatrix}$$

$$(4)\, Q_1 \times \begin{pmatrix} a_{2,4} & 0 \\ a_{3,4} & a_{3,5} \end{pmatrix} = \begin{pmatrix} a'_{2,4} & a'_{2,5} \\ a'_{3,4} & a'_{3,5} \end{pmatrix}$$

$$(5)\ \begin{pmatrix} a'_{2,2} & a'_{2,3} \\ a'_{3,2} & a'_{3,3} \end{pmatrix} \times Q_1 = \begin{pmatrix} a''_{2,2} & a''_{2,3} \\ a''_{3,2} & a''_{3,3} \end{pmatrix}$$

$$(6)\begin{pmatrix} a_{4,2} & a_{4,3} \\ 0 & a_{5,3} \end{pmatrix} \times Q_1 = \begin{pmatrix} a'_{4,2} & a'_{4,3} \\ a'_{5,2} & a'_{5,3} \end{pmatrix}$$

В результаті матриця *M* матиме наступний вигляд:

$$(7)\, M_1 = Q_1 \times M \times Q_1^T$$

$$(8)\, M_1 = \begin{pmatrix} a_{1,1} & a'_{1,2} & 0 & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & a'_{2,4} & a'_{2,5} & & & \\ 0 & a''_{3,2} & a''_{3,3} & a'_{3,4} & a'_{3,5} & & & \\ & a'_{4,2} & a'_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & & \\ & a'_{5,2} & a'_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & \\ & & & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} \\ & & & & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} \\ 0 & & & & & a_{8,6} & a_{8,7} & a_{8,8} \end{pmatrix}$$

Використовуючи *QR*-розклад необхідно почистити елементи, що стали не нульовими після занулення першого шару.

$$(9)\, QR \begin{pmatrix} a'_{4,2} & a'_{4,3} \\ a'_{5,2} & a'_{5,3} \end{pmatrix} = Q_2 \times \begin{pmatrix} a''_{4,2} & a''_{4,3} \\ 0 & a''_{5,3} \end{pmatrix}$$

$$(10)\quad M_2 = Q_2 \times M_1 \times Q_2^T$$

$$(11)\quad M_2 = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & a'_{2,4} & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & a'_{3,5} & & & \\ & a''_{4,2} & a'''_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} & \\ & & a'_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & \\ & & & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} \\ & & & a_{7,4} & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} \\ 0 & & & & & a_{8,6} & a_{8,7} & a_{8,8} \end{pmatrix}$$

Занулення блоку R_{11} знову породить ненульові елементи $a_{4,7}$ та $a_{7,4}$, тому необхідно також почистити і їх також з використанням QR-розкладу:

$$(12)\quad QR\begin{pmatrix} a_{6,4} & a_{6,5} \\ a_{7,4} & a_{7,5} \end{pmatrix} = Q_3 \times \begin{pmatrix} a'_{6,4} & a'_{6,5} \\ 0 & a'_{7,5} \end{pmatrix}$$

$$(13)\quad M_3 = Q_1 \times M_2 \times Q_1^T$$

$$(14)\quad M_3 = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & a'_{2,4} & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & a'_{3,5} & & & \\ & a''_{4,2} & a'''_{4,3} & a'_{4,4} & a'_{4,5} & a'_{4,6} & & \\ & & a'_{5,3} & a'_{5,4} & a'_{5,5} & a'_{5,6} & a'_{5,7} & \\ & & & a'_{6,4} & a'_{6,5} & a''_{6,6} & a''_{6,7} & a'_{6,8} \\ & & & & a'_{7,5} & a''_{7,6} & a''_{7,7} & a'_{7,8} \\ 0 & & & & & a'_{8,6} & a'_{8,7} & a_{8,8} \end{pmatrix}$$

Занулення елементів $a_{4,7}$ та $a_{7,4}$ не породжує нові ненульові елементи, отже можна занулити наступний блок P:

$$(15)\quad QP\begin{pmatrix} a''_{3,2} \\ a''_{4,2} \end{pmatrix} = Q_4 \begin{pmatrix} a'''_{3,2} \\ 0 \end{pmatrix}$$

$$(16)\quad M_4 = Q_4 \times M_3 \times Q_4^T$$

$$(17)\quad M_4 = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & a'_{3,5} & & & \\ & & a'''_{4,3} & a'_{4,4} & a'_{4,5} & a'_{4,6} & a'_{4,7} & \\ & & a'_{5,3} & a'_{5,4} & a'_{5,5} & a'_{5,6} & a'_{5,7} & \\ & & & a'_{6,4} & a'_{6,5} & a''_{6,6} & a''_{6,7} & a'_{6,8} \\ & & & a'_{7,4} & a'_{7,5} & a''_{7,6} & a''_{7,7} & a'_{7,8} \\ 0 & & & & & a'_{8,6} & a'_{8,7} & a_{8,8} \end{pmatrix}$$

Оскільки занулення третього блоку P не створить нових ненульових елементів, то перед зануленням $a'_{7,4}$ та $a'_{4,7}$, породжених кроком 16 занулимо блоки $a'_{5,3}$ та $a'_{3,5}$:

(18) $$QP\begin{pmatrix} a'''_{4,3} \\ a'_{5,3} \end{pmatrix} = Q_5\begin{pmatrix} a''''_{4,3} \\ 0 \end{pmatrix}$$

(19) $$M_5 = Q_5 \times M_4 \times Q_5^T$$

(20) $$M_4 = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & & & & \\ & & a'''_{4,3} & a'_{4,4} & a'_{4,5} & a'_{4,6} & a''_{4,7} & \\ & & & a'_{5,4} & a'_{5,5} & a'_{5,6} & a'_{5,7} & \\ & & & a'_{6,4} & a'_{6,5} & a''_{6,6} & a''_{6,7} & a'_{6,8} \\ & & & a''_{7,4} & a'_{7,5} & a''_{7,6} & a''_{7,7} & a'_{7,8} \\ 0 & & & & & a'_{8,6} & a'_{8,7} & a_{8,8} \end{pmatrix}$$

Аналогічно крокам 9 - 11 необхідно занулити ненульові елементи $a'_{7,4}$ та $a'_{4,7}$ породжені кроком 16 та модифіковані кроком 19:

(21) $$QR\begin{pmatrix} a'_{6,4} & a'_{6,5} \\ a'_{7,4} & a'_{7,5} \end{pmatrix} = Q_6 \times \begin{pmatrix} a'_{6,4} & a'_{6,5} \\ 0 & a'_{7,5} \end{pmatrix}$$

(22) $$M_6 = Q_6 \times M_5 \times Q_6^T$$

(23) $$M_6 = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & & & & \\ & & a'''_{4,3} & a'_{4,4} & a'_{4,5} & a''_{4,6} & & \\ & & & a'_{5,4} & a'_{5,5} & a''_{5,6} & a''_{5,7} & \\ & & & a''_{6,4} & a''_{6,5} & a''''_{6,6} & a''''_{6,7} & a''_{6,8} \\ & & & & a''_{7,5} & a''''_{7,6} & a''''_{7,7} & a''_{7,8} \\ 0 & & & & & a''_{8,6} & a''_{8,7} & a_{8,8} \end{pmatrix}$$

Занулимо четвертий блок P:

(24) $$QP\begin{pmatrix} a'_{5,4} \\ a''_{6,4} \end{pmatrix} = Q_7 \begin{pmatrix} a''_{5,4} \\ 0 \end{pmatrix}$$

(25) $$M_7 = Q_7 \times M_6 \times Q_7^T$$

(26) $$M_7 = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & & & & \\ & & a'''_{4,3} & a'_{4,4} & a'_{4,5} & & & \\ & & & a'_{5,4} & a'_{5,5} & a''_{5,6} & a''_{5,7} & a'_{5,8} \\ & & & & a''_{6,5} & a''''_{6,6} & a''''_{6,7} & a''_{6,8} \\ & & & & a''_{7,5} & a''''_{7,6} & a''''_{7,7} & a''_{7,8} \\ 0 & & & & a'_{8,5} & a''_{8,6} & a''_{8,7} & a_{8,8} \end{pmatrix}$$

Занулимо породжені ненульові елементи $a'_{8,5}$ та $a'_{5,8}$ використовуючи *QR*-розклад:

(27) $$QR\begin{pmatrix} a''_{7,5} & a''''_{7,6} \\ a'_{8,5} & a''_{8,6} \end{pmatrix} = Q_8 \times \begin{pmatrix} a'''_{7,5} & a'''''_{7,6} \\ 0 & a'''_{8,6} \end{pmatrix}$$

(28) $$M_8 = Q_8 \times M_7 \times Q_8^T$$

(29) $$M_8 = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & & & & \\ & & a'''_{4,3} & a'_{4,4} & a'_{4,5} & & & \\ & & & a'_{5,4} & a'_{5,5} & a''_{5,6} & a''_{5,7} & \\ & & & & a''_{6,5} & a''''_{6,6} & a''''_{6,7} & a''_{6,8} \\ & & & & a''_{7,5} & a''''_{7,6} & a''''_{7,7} & a''_{7,8} \\ 0 & & & & & a''_{8,6} & a''_{8,7} & a_{8,8} \end{pmatrix}$$

Аналогічно до попередніх кроків занулимо п’ятий блок *P*, на відміну від занулень P_1-P_4 цей крок не породжує нових ненульових елементів, тому немає необхідності у додатковій чистці після множення матриці повороту на смугу:

(30) $$QP\begin{pmatrix} a''_{6,5} \\ a''_{7,5} \end{pmatrix} = Q_9 \begin{pmatrix} a'''_{6,5} \\ 0 \end{pmatrix}$$

(31) $$M_9 = Q_9 \times M_8 \times Q_9^T$$

$$M_8 = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & & & & \\ & & a'''_{4,3} & a'_{4,4} & a'_{4,5} & & & \\ & & & a'_{5,4} & a'_{5,5} & a''_{5,6} & & \\ & & & & a''_{6,5} & a''''_{6,6} & a''''_{6,7} & a''_{6,8} \\ & & & & & a''''_{7,6} & a''''_{7,7} & a''_{7,8} \\ 0 & & & & & a''_{8,6} & a''_{8,7} & a_{8,8} \end{pmatrix} \quad (32)$$

Перед зануленням останнього P блоку необхідно звести блоки $a''_{8,6}$ та $a''_{6,8}$ до верхньотрикутного вигляду використовуючи QR-розклад якщо розмір матриці більший 8:

$$QR\left(a''_{8,6}\right) = Q_{10} \times \left(a^U_{8,6}\right) \quad (33)$$

$$M_{10} = Q_{10} \times M_9 \times Q_{10}^T \quad (34)$$

$$M_{10} = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & & & & \\ & & a'''_{4,3} & a'_{4,4} & a'_{4,5} & & & \\ & & & a'_{5,4} & a'_{5,5} & a''_{5,6} & & \\ & & & & a''_{6,5} & a''''_{6,6} & a''''_{6,7} & (a_{6,8})_U \\ & & & & & a''''_{7,6} & a''''_{7,7} & a''_{7,8} \\ 0 & & & & & a^U_{8,6} & a''_{8,7} & a_{8,8} \end{pmatrix} \quad (35)$$

Далі можемо занулити останній блок P:

$$QP\begin{pmatrix} a''''_{7,6} \\ a^U_{8,6} \end{pmatrix} = Q_{11}\begin{pmatrix} a''''_{7,6} \\ 0 \end{pmatrix} \quad (36)$$

$$M_{11} = Q_{11} \times M_{10} \times Q_{11}^T \quad (37)$$

$$(38)\quad M_{11} = \begin{pmatrix} a_{1,1} & a'_{1,2} & & & & & & 0 \\ a'_{2,1} & ''a_{2,2} & a''_{2,3} & & & & & \\ & a''_{3,2} & a''_{3,3} & a'_{3,4} & & & & \\ & & a'''_{4,3} & a'_{4,4} & a'_{4,5} & & & \\ & & & a'_{5,4} & a'_{5,5} & a''_{5,6} & & \\ & & & & a''_{6,5} & a''''_{6,6} & a''''_{6,7} & \\ & & & & & a''''_{7,6} & a''''_{7,7} & a''_{7,8} \\ 0 & & & & & & a''_{8,7} & a_{8,8} \end{pmatrix}$$

Для матриці розміру 8 це є останній крок, адже отримана матриця є тридіагональною. Якщо матриця є матрицею блоків, то необхідно виконати останній крок це *QR*-розклад блоку $a''_{8,7}$ для приведення його до верхньотрикутного вигляду.

2.5. Складність алгоритму Ribbon

Нехай дано симетричну матрицю *M* розміру $2^n \times 2^n$, оцінимо кількість дій, яку необхідно виконати для першої ітерації алгоритму. Для основних занулень необхідно порахувати шість *QP*-розкладів блоків розміром $2^{n-2} \times 2^{n-3}$ та два *QR*-розклади блоків розміром $2^{n-3} \times 2^{n-3}$. Для занулень породжених ненульових елементів необхідно зробити чотири *QR*-розклади блоків розміром $2^{n-2} \times 2^{n-2}$. Також необхідно виконати тридцять шість матричних множень для блоків розміром $2^{n-2} \times 2^{n-2}$ та два матричних множення для блоків $2^{n-3} \times 2^{n-3}$.

$$C(n) = 6C_{qp}\left(\frac{n}{4}\right) + 2C_{qr}\left(\frac{n}{8}\right) + 4C_{qr}\left(\frac{n}{4}\right) + 36M\left(\frac{n}{4}\right) + 2M(\frac{n}{8})$$

Припустимо, що складність матричного множення дорівнює γn^{β}, де $\gamma = 1$, а $\beta = 3$. Визначимо складності *QR* та *QP* розкладів з такими коефіцієнтами:

$$C_{qp}(n) = \frac{7\gamma n^b}{2^{\beta}(2^{\beta}-4)} + \frac{3n^2}{2}\left(3 - \frac{7\gamma}{2^{\beta+1}-8}\right) = \frac{7n^3+2*23*3n^2}{32} \approx 0.22n^3 + 4.3n^2$$

$$C_{qr}(n) = \frac{\gamma(2^{\beta}6 - 17)(n^{\beta} - \frac{2n}{2^{\beta}})}{(2^{\beta} - 4)(2^{\beta} - 2)} = \frac{31(n^3 - \frac{n}{2^2})}{24} = 1.29n^3 - 0.32n$$

$$C(n) = 6\left(0.22\left(\frac{n}{4}\right)^3 + 4.3\left(\frac{n}{4}\right)^2\right) + 2\left(1.29\left(\frac{n}{8}\right)^3 - 0.32\left(\frac{n}{8}\right)\right)$$

$$+ 4\left(1.29\left(\frac{n}{4}\right)^3 - 0.32\left(\frac{n}{4}\right)\right) + 36\left(\frac{n}{4}\right)^3 + 2\left(\frac{n}{8}\right)^3$$

$$\approx 0.67n^3 + 1.61n^2 - 0.4n$$

Розділ 3. Середовище блоково-рекурсивних паралельних обчислень DAP

3.1. Середовище DAP

Середовище DAP (Drop – Amin – Pine) – децентралізована паралельна система керування завданнями. Програмний пакет реалізований мовою Java та використовує MPI для задач комунікації між потоками та транспортування даних.

Для того, щоб запустити алгоритм на виконання потрібно, сформувати обчислювальний граф відповідно до алгебраїчного алгоритму. Вершинами графу є обчислювальні блоки, що виконують певний крок алгоритму. Граф обов'язково містить вхідний блок та вихідний блок. Жоден з блоків не може бути викликаний перед вхідним або після вихідного блоку. Всі ребра графу направлені та визначають куди передаються матричні блоки, після виконання завдання.

Децентралізація управління полягає в тому, що кожен процесор може отримати будь-яку з вершин графу алгоритму та керувати процесами в цій вершині, крім того разом із вершиною процесор отримує список підлеглих йому вільних процесорів в середовищі[9].

3.2. Основні класи середовища DAP

Класи системи можна поділити на числові та функціональні. Числові – це ті, які є абстракцією над числовими елементами або алгебраїчними концепціями. Ці класи необхідні для покращення точності та розширення можливостей розробника, надаючи гнучкий функціонал для роботи з

числовими типами. Розглянемо кілька базових таких класів, які є основою всіх обчислень в DAP:

1. Element – базовий клас для всіх чисельних типів над якими виконуються обчислення. Похідними від класами від Element є NumberZ — клас цілих чисел, NumberR – клас дійсних чисел, Complex – клас комплексних чисел, NumberZ64 – обгортка над типом long, NumberR64 – обгортка над типом double.
2. Ring – це клас, що визначає алгебраїчний простір поточних змінних, тобто, кільце, в рамках якого виконуються поточні операції. Наприклад, “Z[]” — операції над кільцем цілих чисел, “R[]” — над кільцем дійсних чисел із заданою точністю.
3. MatrixS – клас розріджених матриць, що містить в собі елементи типу Element та має набір операцій для роботи з квадратними матрицями: add, subtractSquareMat, negate, multiplySquareMat, multiplyRecursive (паралельна процедура).

Функціональні класи – це ті, які є основними обчислювальними класами середовища DAP. До них відносяться: CalcThread, DispThread, Amin, Drop, Transport. Архітектура системи побудована на взаємодії цих класів між собою.

1. Drop – абстрактний клас завдання. Для того щоб створити завдання, користувачу потрібно визначити обчислювальний граф в класі-наслідникy та описати які саме обчислення потрібно виконати у вузлах перевизначивши метод doAmin().
2. CalcThread – обчислювальний потік, що виконує обрахунки визначені в завданні Drop. Може виконуватись на кожному процесорі.
3. DispThread – диспетчерський потік, що відповідає за розподіл завдань похідних від Drop між процесорами та обмін повідомлень між ними.

4. Amin – обчислювальний підграф, що необхідно розгорнути перед обчисленням дропу. В свою чергу Amin теж складається з дропів.

3.3. Конфігурація завдання Drop

Для того, щоб створити власний клас завдання похідного від Drop, необхідно ініціалізувати поля класу та перевизначити основні методи інтерфейсу:

1. *int [][] arcs* – двовимірний масив зв'язків обчислювального графу. Зв'язки для кожного вихідного блоку задаються трійками: номер залежного дропу, індекс блоку у вихідному масиві вузла, індекс блоку у вхідному масиві залежного дропу. Приклад такого масиву зображений в додатку А.
2. *int type* – унікальний тип завдання. Використовується класом Drop у функції doNewDrop(int type, byte[] config) для створення дропів цього ж типу при рекурсивному розгортанні обчислювального графу.
3. *int key* – унікальний ідентифікатор завдання дропу. Потрібен для збільшення гнучкості використання дропів. Залежно від значення ключа можна модифікувати самі обчислення використовуючи однакову структуру обчислювального графу. Хорошим прикладом буде дроп множення матриць *MatrSMult4*. Використовуючи різні значення ключа можна змінювати алгоритм обчислення. Наприклад: при key = 0 результатом обчислення буде добуток двох матриць, при key = 1 – добуток двох матриць з інвертованим знаком кожного елемента.
4. *int inputDataLength* – розмір вхідного масиву даних *inData*.
5. *int outputDataLength* – розмір вихідного масиву даних *outData*.

6. *int resultForOutFunctionLength* – розмір вхідного масиву даних необхідного для *outputFunction(Element[] input, Ring ring).*
7. *Element [] inData* – внутрішній масив, в який записуються вхідні блоки даних.
8. *Element [] outData* – внутрішній масив, в який записуються результати обчислень дропу.
9. *setVars()* – метод, що викликається при рекурсивному створенні завдань типу Drop для ініціалізації полів класу, таких як *inputDataLength, outputDataLength, resultForOutFunctionLength, inData, outData.*
10. *doAmin()* – метод для визначення типів завдань у вершинах обчислювального графу.
11. *inputFunction(Element[] input, Amin amin, Ring ring)* – перший блок, що виконується при обчисленні дропу. Найчастіше використовується для початкового розбиття вхідної матриці на блоки.
12. *outputFunction(Element[] input, Ring ring)* – останній блок дропу, який збирає та записує результати обчислень у масив outData.
13. *sequentialCalc (Ring ring)* – метод послідовного обрахунку завдання без подальшого рекурсивного розбиття.

3.4. Управління обчислювальним процесом в середовищі DAP

Ми вже розглянули основні чисельні та функціональні класи DAP, тому далі розглянемо як відбувається управління обчислювальним процесом в середовищі.

Всі аміни, що формуються на одному процесорі, зберігаються в загальному списку – Pine. Для того, щоб визначити куди повертати результат обчислення дропа, в кожному підзавданні є PAD ідентифікатор, який формується із трьох індексів: номер процесора, номер аміна в списку Pine

процесора та номером дропа у аміні. PAD = (np, na, nd). Оскільки дроп може бути виконаний на будь-якому процесорі і для нього буде розгорнуто Amin, то в адресному полі цього аміна потрібно вказати PAD для відправлення результатів обчислень. Комунікацію зі всіма процесорами забезпечують «аеропорт», «вокзал» та «термінал» [8], зв'язки між ними відображено на рисунку 4.

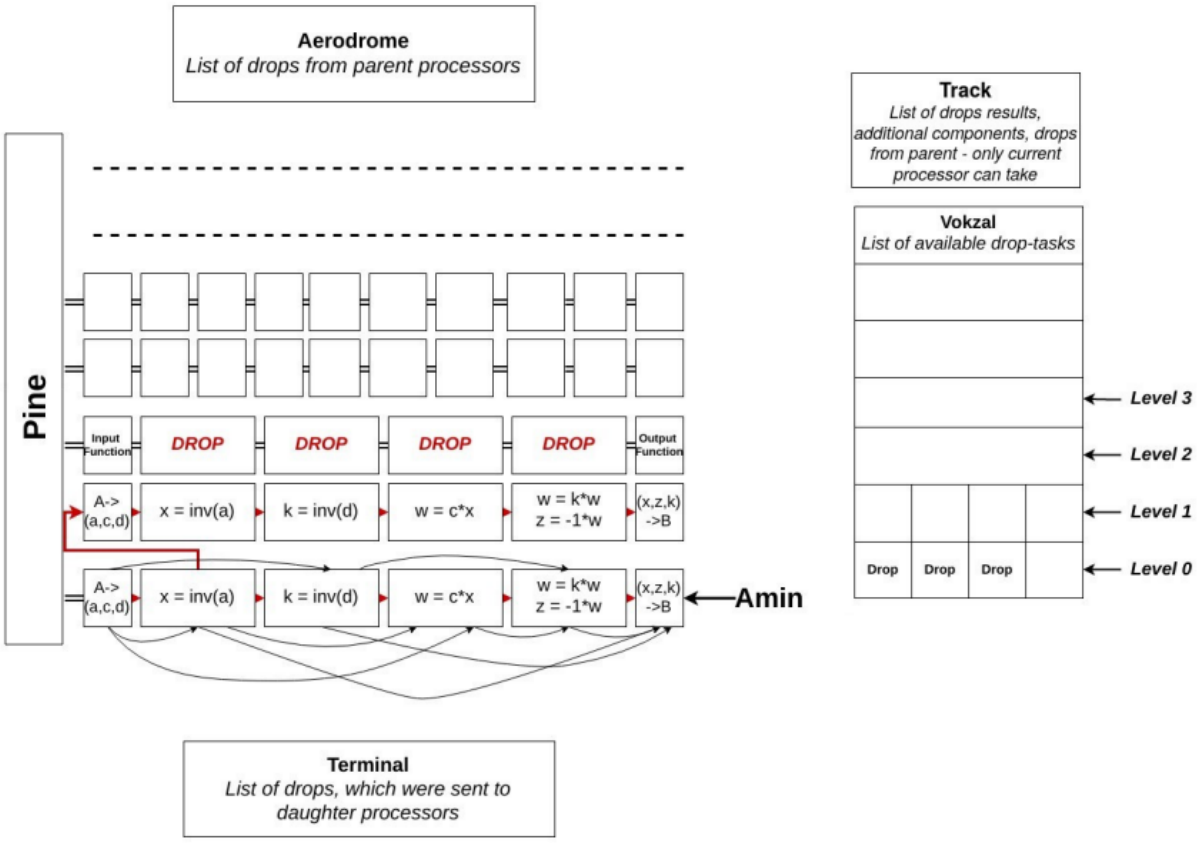

Рисунок 4. Схема управління обчисленнями в DAP

«Вокзал» є списком поточних дроп-завдань, що мають бути обчислені. Залежно від глибини рекурсії дропи розташовані на різних рівнях «вокзалу». На виконання спочатку відправляються дропи з найнижчого рівня «вокзалу». Нові завдання прибувають на «вокзал», коли при обчислені аміна з'являються завдання, що можуть бути виконані паралельно.

В «аеродромі» зберігаються номер батьківського процесора і список завдань від цього процесора. Батьківським називається процесор, який надіслав дроп-завдання. Коли всі завдання отримані від процесора будуть виконані, він видаляється із «аеродрому».

«Термінал» є симетричною сутністю за призначенням до «аеродрому». Він використовується для зв'язку з дочірніми процесорами, яким було надіслано список дроп-завданнь. Аналогічно «вокзалу», на терміналі дочірні процесори розташовуються на відповідних їм рівнях. «Рівнем терміналу» називається найменший рівень дочірнього процесу.

3.5. Приклад створення дропів для QR та QP розкладів

Дропи *QR* та *QP* розкладів вже були реалізовані в рамках DAP01, що має кілька відмінностей від актуальної версії DAP02. Для того, щоб налаштувати та запустити обчислення алгоритму на оновленій версії середовища було необхідно ініціалізувати всі поля та перевизначити всі методи класу Drop для дропів QRDecomposition та QPDecomposition, які були описані у розділі 2.3, однак структура обчислювального графу не змінювалась.

Обчислювальний граф *QR*-розкладу, зображеного на рисунку 5, складається з наступних вершин:

0. *Input* – розбиває вхідну матрицю на 4 блоки.
1. *QR (C)* – рекурсивний *QR*-розклад блоку *C*.
2. $QP\begin{pmatrix} A \\ C^U \end{pmatrix}$ – рекурсивний *QP*-розклад лівих блоків $\begin{pmatrix} A \\ C^U \end{pmatrix}$
3. $Multiply(Q_1, Q_2)$ – множення двох матриць Q, отриманих в результаті перших двох розкладів.

4. $Multiply\left(Q_{1,2}, B, D\right)$ – множення матриці Q отриманої на попередньому кроці на праві блоки $\binom{B}{D}$.
5. $QR\ (D_1)$ – рекурсивний QR-розклад блоку D_1
6. $Multiply(Q_{1,2}, Q_3)$ – знаходження загальної матриці Q.
7. $Output$ – збір результатів із блоків.

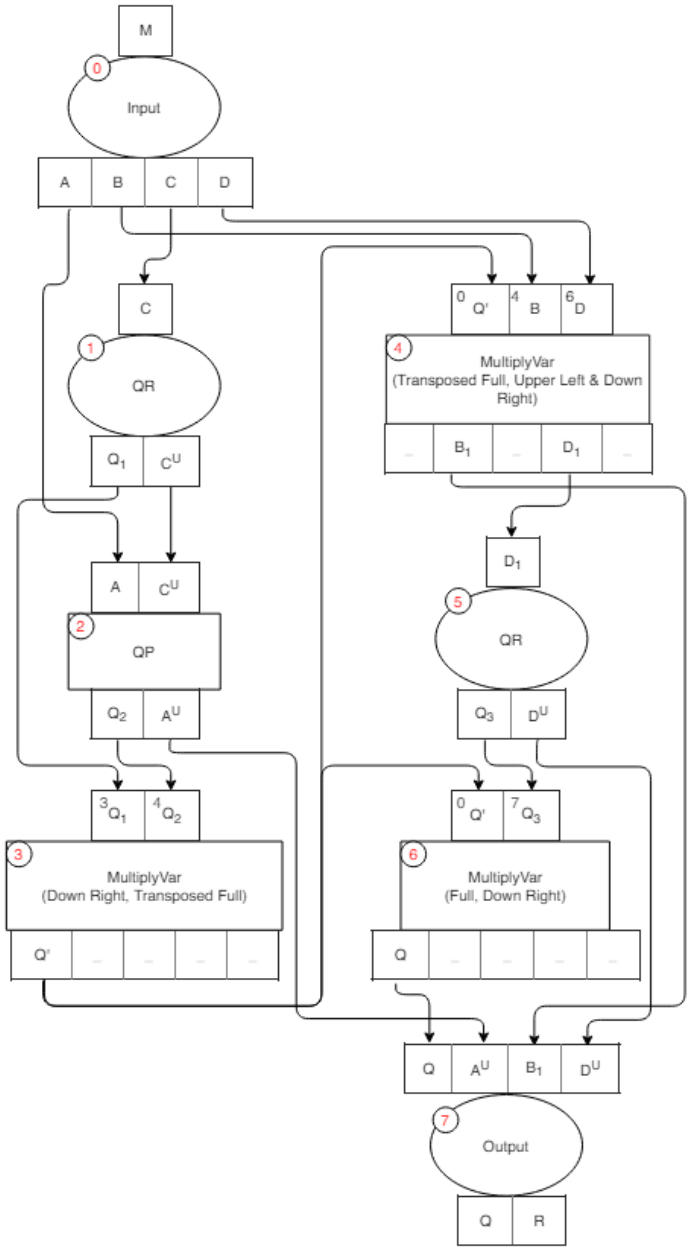

Рисунок 5. Граф блоково-рекурсивного алгоритму QR-розкладу

Розглянемо обчислювальний граф для *QP*-розкладу, зображеного на рисунку 6:

0. *Input* – початкове розбиття двох вхідних блоків на 8.
1. $QP\begin{pmatrix} c \\ e^U \end{pmatrix}$ – рекурсивний *QP*-розклад для блоків c та e^U
2. *Multiply* $(Q_1, \begin{pmatrix} d \\ f \end{pmatrix})$ – множення матриці Q отриманої після першого розкладу на блоки d та f.
3. $QP\begin{pmatrix} a \\ c^U \end{pmatrix}$ – рекурсивний *QP*-розклад для блоків a та c^U.
4. $QP\begin{pmatrix} f_1 \\ h^U \end{pmatrix}$ – рекурсивний *QP*-розклад для блоків f_1та h^U.
5. *Multiply* $(Q_2, \begin{pmatrix} b \\ d_1 \end{pmatrix})$ – множення матриці Q на блоки b та d_1, отриманої в результаті розкладу у вершині 3.
6. *Multiply* (Q_3, Q_1) – множення матриць Q_1 та Q_3 обрахованих на першій та четвертій вершинах.
7. $QP\begin{pmatrix} d_2 \\ f^U \end{pmatrix}$ – рекурсивний *QP*-розклад для блоків d_2та f^U.
8. *Multiply* $(Q_2, Q_{3,1})$– знаходження ортогональної матриці Q для правого нижнього блоку.
9. *Multiply* $(Q_4\,, Q_{2,3,1})$ – знаходження загальної ортогональної матриці Q.
10. *Output* – збір результатів обчислень.

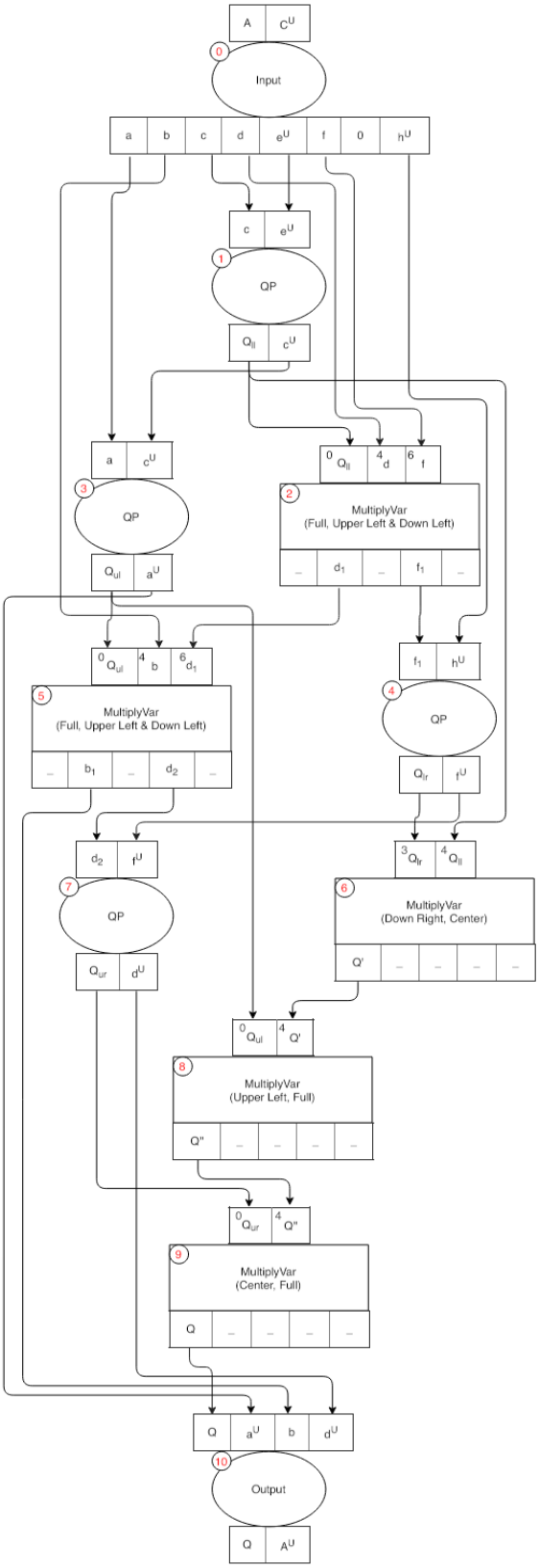

Рисунок 6. Граф блоково-рекурсивного алгоритму QP-розкладу

3.6. Обчислювальний граф алгоритму Q3RP

Алгоритм реалізований у DAP складається з 28 блоків частина з яких може бути обчислена одночасно. На рисунку 8 зображено схему зв'язків між дропами та зміну матричних блоків у процесі виконання операцій з ними всередині дропів. Весь алгоритм можна умовно поділити на чотири етапи: занулення першого шару (*QR*+2*QP*) , занулення другого шару (*QR*+*QP*), занулення третього шару (*QR*), множення матриць повороту для отримання загальних матриць повороту зліва та справа. На графічному зображені обчислювального графу алгоритму *Q3RP* (рисунок 7) вершини, що знаходяться на одному рівні можуть бути обчислені одночасно.

Розглянемо детальніше кожен блок дропу *Q3RP*:

0. *Input* – розбиття вхідної матриці на 10 блоків: спочатку розбиваємо на чотири рівні блоки *A, B, C* та *D,* далі блоки *A* та *C* розбиваються на менші рівні блоки.
1. $QR(c_{2,1})$ – розклад блоку $c_{2,1}$ та знаходження матриці Q_1.
2. $QP_1\begin{pmatrix} c_{1,1} \\ c'_{2,1} \end{pmatrix}$ – занулення блоку $c'_{2,1}$ та знаходження матриці Q_2.
3. $Multiply(Q_1, Q_2^T)$ – знаходження загальної матриці Q_3 для повного занулення блоку $c_{2,1}$.

Наступні чотири кроки це множення блоків матриці на отриману матрицю повороту Q_3 зліва та справа. Після перемноження блоки B та D розбиваються на рівні квадратні блоки. Розбиття необхідне для формування серединних блоків, які будуть змінені матрицею повороту Q_4, отриманої як результат обчислень кроку 8:

4. $Multiply\left(Q_3, \begin{pmatrix} c_{1,2} \\ c_{2,2} \end{pmatrix}\right)$.

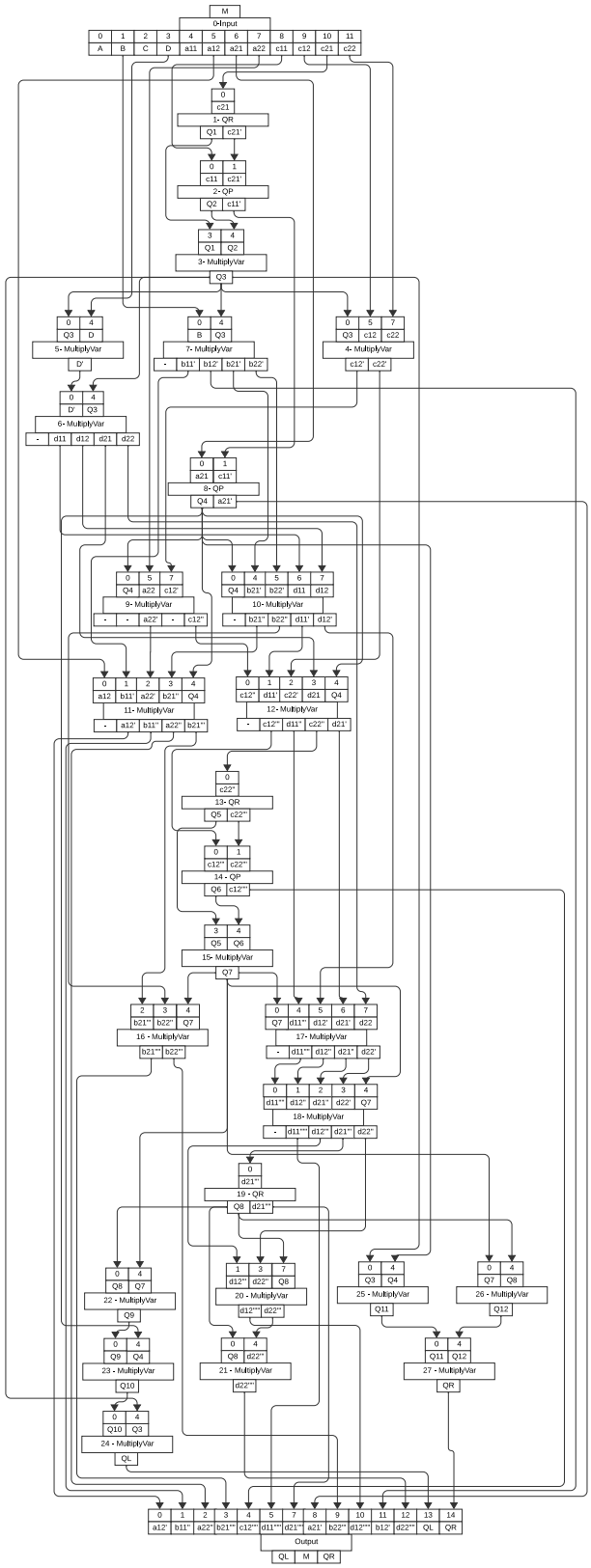

Рисунок 7. Обчислюваний граф Q3RP-розкладу

5. $Multiply(Q_3^T, D)$.
6. $Multiply(D', Q_3)$.
7. $Multiply\ (B, Q_3)$.
8. $QP_{11}\begin{pmatrix} a_{2,1} \\ c'_{1,1} \end{pmatrix}$ – знаходження матриці повороту Q_4 для повного занулення блоку $c'_{1,1}$.

Множення матриці повороту Q_4 на серединні блоки сформованих з другого та третього рядків вхідної матриці ліворуч:

9. $Multiply\left(Q_4, \begin{pmatrix} a_{2,2} \\ c_{1,2} \end{pmatrix}\right)$
10. $Multiply\ \left(Q_4, \begin{pmatrix} b_{2,1} & b_{2,2} \\ d_{1,1} & d_{1,2} \end{pmatrix}\right)$.

Множення матриці повороту Q_4 на серединні блоки сформованих з другого та третього стовпців вхідної матриці праворуч:

11. $Multiply\ \left(\begin{pmatrix} a_{1,2} & b_{1,1} \\ a_{2,2} & b_{2,1} \end{pmatrix}, Q_4^T\right)$.
12. $Multiply\ \left(\begin{pmatrix} c_{1,2} & d_{1,1} \\ c_{2,2} & d_{2,1} \end{pmatrix}, Q_4^T\right)$.

Знаходження загальної матриці повороту для занулення другого шару, що складається з блоків $c_{1,2}$ та $c_{2,2}$ аналогічно крокам 1-3:

13. $QR_2\left(c_{2,2}\right)$.
14. $QP_2\begin{pmatrix} c_{1,2} \\ c_{2,2} \end{pmatrix}$.
15. $Multiply\ (Q_5, Q_6^T)$

Множення блоків B та D на матрицю повороту Q_7 ліворуч та праворуч:

16. $Multiply\left((b_{2,1} \quad b_{2,2}), Q_7\right)$.

17. $Multiply\left(\begin{pmatrix} d_{1,1} & d_{1,2} \\ d_{2,1} & d_{2,2} \end{pmatrix}, Q_7^T\right)$.

18. $Multiply\left(Q_7, \begin{pmatrix} d_{1,1} & d_{1,2} \\ d_{2,1} & d_{2,2} \end{pmatrix}\right)$.

Занулення третього шару. На відміну від попередніх шарів під час цього етапу не використовується *QP*-розклад, тому елементи блоку *D* домножуються на матрицю повороту отриману в результаті третього *QR*-розкладу:

19. $QR_3\left(d_{2,1}\right)$

20. $Multiply\left(\begin{pmatrix} d_{1,2} \\ d_{2,2} \end{pmatrix}, Q_8\right)$.

21. $Multiply\left(Q_8^T, d_{2,2}\right)$.

Останнім етапом є перемноження матриць повороту для отримання загальних лівої та правої матриць повороту:

22. $Multiply(Q_8^T, Q_7^T) = Q_9$

23. $Multiply(Q_9, Q_4^T) = Q_{10}$

24. $Multiply(Q_{10}, Q_3^T) = Q_L$

25. $Multiply(Q_3, Q_4) = Q_{11}$

26. $Multiply(Q_7, Q_8) = Q_{12}$

27. $Multiply\ (Q_{11}, Q_{12}) = Q_R$

Вихідна функція збирає всі результати обчислень та формує результуючу матрицю, ліву та праву матриці повороту.

28. *Output.*

Розділ 4. Оптимізація алгоритму QR розкладу

4.1. Алгоритм Штрассена для швидкого множення матриць

У роботі [1] Штрассеном було запропоновано алгоритм, за допомогою якого можна зменшити кількість обчислювальних операцій для матриць розміру $n \times n$ до $4.7n^{\log 7}$, в той час коли при стандартному підході необхідно виконати $2n^3$ операцій.

Якщо необхідно помножити дві матриці A та B порядку $m2^{k+1}$, можемо записати:

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}, \qquad C = AB = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Де блоки $A_{i,j}, B_{i,j}, C_{i,j}$ будуть порядку $m2^k$. Обрахуємо наступні матриці:

$$(1)\, M1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}),$$
$$(2)\, M2 = (A_{2,1} + A_{2,2})B_{1,1},$$
$$(3)\, M3 = A_{1,1}(B_{1,2} - B_{2,2}),$$
$$(4)\, M4 = A_{2,2}(-B_{1,1} + B_{2,1}),$$
$$(5)\, M5 = (A_{1,1} + A_{1,2})\, B_{2,2},$$
$$(6)\, M6 = (-A_{1,1} + A_{2,1})(B_{1,1} + B_{1,2}),$$
$$(7)\, M7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2}),$$

Тепер можемо виразити блоки матриці *C* наступним чином :

$$C = \begin{pmatrix} M1 + M4 - M5 + M7 & M3 + M5 \\ M2 + M4 & M1 + M3 - M2 + M6 \end{pmatrix}$$

Для того щоб помножити дві матриці необхідно виконати 18 операцій додавання та лише 7 операцій множення замість 8 при стандартному підході.

Із застосуванням методу Винограда ми можемо зменшити кільки операцій додавання до 15. Складність алгоритму:

$$O(n) = n^{log7} + \frac{14}{3}\left(n^{\log 7} - n^2\right) = \frac{17}{3}n^{\log 7} - \frac{14}{3}n^2$$

4.2. Схема дропу швидкого множення матриць в DAP

Розглянемо обчислювальний граф швидкого матричного множення в DAP на рисунку 8:

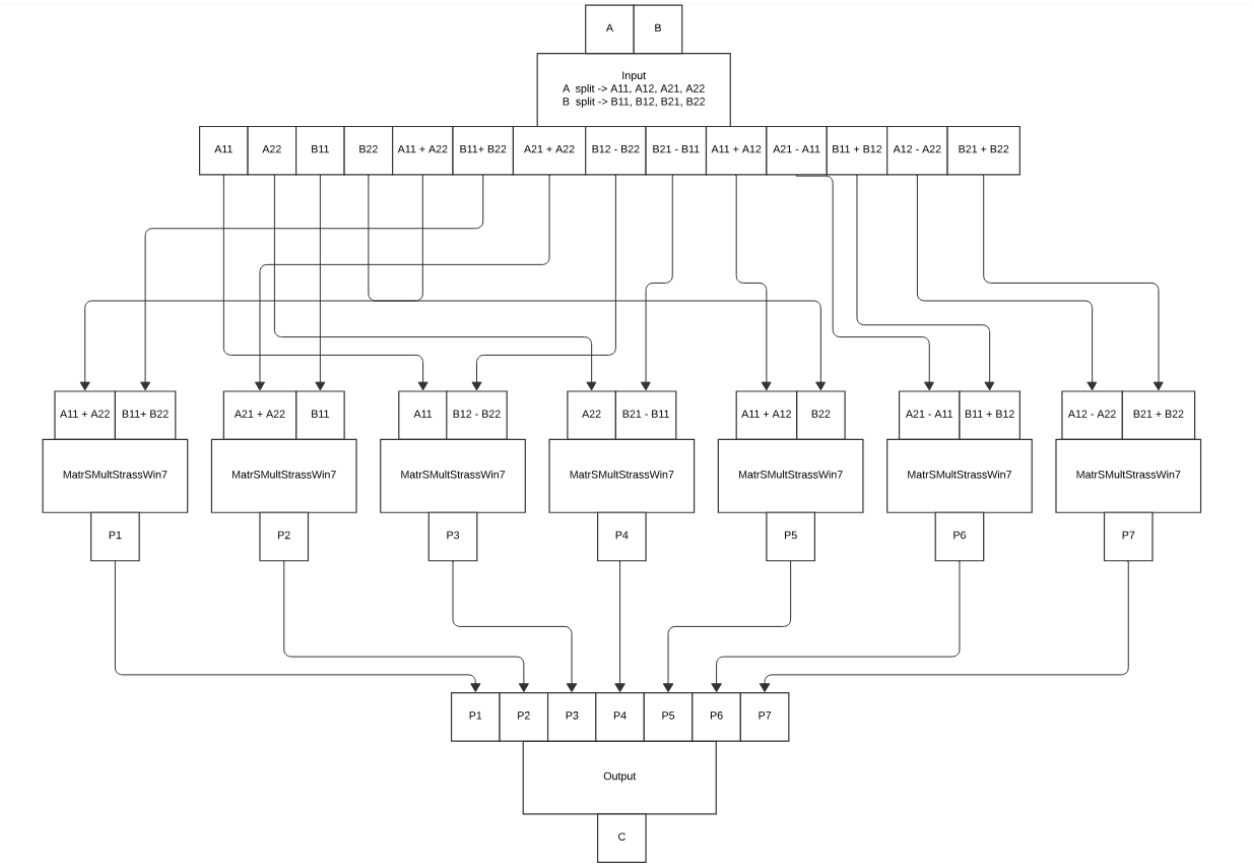

Рисунок 8. Обчислювальний граф алгоритму швидкого множення матриці методом Винограда - Штрассена

0. *Input* – початковий розбиття вхідних матриць на блоки та обрахунок сум та різниць, що використовуються для множення.

1. $Multiply(A_{1,1} + A_{2,2}, B_{1,1} + B_{2,2})$
2. $Multiply(A_{2,1} + A_{2,2}\ , B_{1,1})$
3. $Multiply(A_{1,1}\ , B_{1,2} - B_{2,2})$
4. $Multiply(A_{2,2}\ , B_{2,1} - B_{1,1})$
5. $Multiply(A_{1,1} + A_{1,2}\ , B_{2,2})$
6. $Multiply(\ A_{2,1} - A_{1,1}\ , B_{1,1} + B_{1,2})$
7. $Multiply(A_{1,2} - A_{2,2}\ , B_{2,1} + B_{2,2})$
8. *Output* – збір результатів обчислень та розрахунок блоків вихідної матриці. $C_{1,1} = P_1 + P_5 - P_7\ ,\ C_{1,2} = P_3 + P_5\ , C_{2,1} = P_2 + P_4\ , C_{2,2} = P_1 - P_2 + P_3 + P_6,$

4.3. Складність QR-розкладу із застосуванням методу Штрассена

Оскільки складність *QR*-розкладу була обрахована з використанням складності матричного множення γn^{β}, то для оцінки складності використаємо коефіцієнти: $\beta = \log_2 7, \gamma = 4.7$

$$C(n) = \frac{\gamma\left(2^{\beta}6 - 17\right)\left(n^{\beta} - \frac{2n}{2^{\beta}}\right)}{(2^{\beta} - 4)(2^{\beta} - 2)} = \frac{4.7(2^{2.8}6 - 17)\left(n^{2.8} - \frac{2n}{2^{2.8}}\right)}{(2^{2.8} - 4)(2^{2.8} - 2)} =$$

$$= \frac{4.7(6.96 * 6 - 17)(n^{2.8} - \frac{2n}{6.96})}{(6.96\ - 4)(6.96\ -\ 2)} = \frac{4.7 * 24.76 * (n^{2.8} - \frac{2n}{6.96})}{14.68} \approx 7.92n^{\log_2 7} - 2.28n$$

4.4. Визначення оптимального листкового розміру

Зважаючи на константу пришвидшеного алгоритму множення матриць, кількість необхідних операцій не завжди буде меншою за стандартну, тому необхідно знайти оптимальний розмір матриці, при якому буде відчутне

прискорення порівняно зі стандартним алгоритмом. Тобто, необхідно визначити мінімальний розмір матриці для якого виконується нерівність: $\frac{17}{3}n^{\log 7} - \frac{14}{3}n^2 < 2n^3$. Критичною точкою є $n = 228$. Це є мінімальний розмір матриці при якому алгоритм Винограда - Штрассена потребуватиме меншу кількість операцій ніж стандартний підхід.

Для того щоб знайти оптимальний розмір листка було проведено тестування з різними наборами параметрів: кількість процесорів, розмір даних, розмір листка. Результати тестування відображені на рисунку 9.

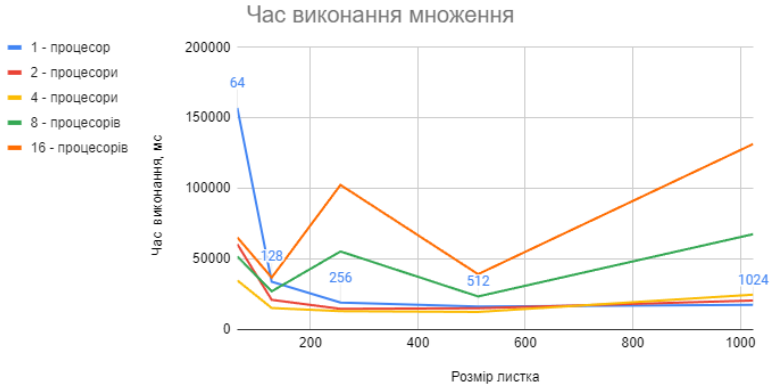

Рисунок 9. Залежність часу виконання алгоритму Винограда–Штрассена від кількості процесорів та розміру листка для матриці 1024x1024

Для методу швидкого множення матриць, було встановлено, що на персональному комп’ютері оптимальними є наступний набір параметрів: 4 процесори, розмір листка для послідовного обрахунку – 128, розмір матриці повинен бути більший за 1024. При збільшені кількості процесорів алгоритм відпрацьовував довше за рахунок того, що в системі є лише 4 фізичних ядра. Оскільки число процесорів більше за число фізичних ядер, то час обчислень та час очікування результату зростають. Малий розмір листка значно збільшував час виконання алгоритму через накладні витрати на розгортання аміну та

великій глибині рекурсії. Пришвидшення обчислень спостерігалось при зменшені листкового розміру до $n = 128$ (для 4-х процесорів).

Тобто, було експериментально підтверджено, що прискорення при використанні методу швидкого матричного множення є лише при розмірі матриці $n \geq 228$. Оскільки ми розглядаємо лише матриці з розміром $2^k \times 2^k$, то мінімальний розмір матриці, при якому рекурсивний обрахунок буде ефективним є 256, а отже шуканий розмір листка - 128.

Розділ 5. Результати тестування на персональному комп’ютері

У цьому розділі розглянемо результати тестування оптимізованого *QR*-розкладу та дропу *Q3RP* на особистому ПК. На момент написання роботи суперкомп’ютер ІК НАН України не доступний для проведення тестування у зв’язку з нестачею електроенергії. Всі тести запускались окремо, з різними конфігураціями для оцінки прискорення обчислень у багато поточному середовищі. Мінімальний розмір листка був використаний 256. Цей розмір був обґрунтований у розділі 4.4. Кількість фізичних ядер наявна у персональному комп’ютері - 4, тому це була максимальна кількість процесорів з якою запускались тести. В розділі 4.4 продемонстровано, що збільшення кількості віртуальних ядер не пришвидшує виконання алгоритму.

5.1. Результати тестування QR-розкладу

Оскільки *QR*-розклад є основою *Q3RP*, то спочатку протестуємо цей алгоритм та його прискорення з використанням пришвидченого матричного множення. Для оцінки прискорення будемо використовувати матрицю розміром 1024×1024. Результати тестування *QR*-розкладу з використанням одного ядра та різних листкових розмірів наведені у таблиці 5.1.

Таблиця 5.1 Час виконання QR на 1-му ядрі

Процесори	**Розмір даних**	**Розмір листка**	**Час виконання, мс**
1	1024	128	338450
1	1024	256	110238
1	1024	512	79619
1	1024	1024	78626

Зі збільшенням кількості процесорів час виконання зменшився, однак для отримання кращих результатів прискорення необхідно запустити

тестування на матриці більшого розміру. Результати тестування *QR*-розкладу на 4-х ядрах наведені у таблиці 5.2. Наявні ресурси персонального комп'ютера цього зробити не дозволяють, оскільки необхідно більше оперативної пам'яті для зберігання дерева рекурсії. Використання віртуальної пам'яті негативно вплинуло на результати тестів та не показало прискорення.

Таблиця 5.2 Результати тестування алгоритму QR-розкладу на 4-х ядрах

Процесори	**Розмір даних**	**Розмір листка**	**Час виконання, мс**
4	1024	128	217969
4	1024	256	86892
4	1024	512	73520
4	1024	1024	80010

QR-розклад з використанням методу матричного множення Винограда-Штрассена найкраще показав себе при послідовному виконанні алгоритму, адже він ефективний лише на матрицях великого розміру. Серед результатів, відображених у таблиці 5.3, також можна помітити велике збільшення часу виконання між послідовним виконанням та виконанням на чотирьох процесорах з листовим розміром 512. Це можна пояснити збільшенням кількості операції об'єднання матриць, що з'являються при використанні нового методу матричного множення.

Таблиця 5.3 Результати тестування алгоритму модифікованого QR-розкладу

Процесори	**Розмір даних**	**Розмір листка**	**Час виконання, мс**
4	1024	128	146503
4	1024	256	76277
4	1024	512	100318
4	1024	1024	73552

З використанням пришвидченого множення прискорення зростає зі збільшенням кількості рекурсивних кроків при обчисленні. Таким чином ми

отримали прискорення в 1,48 рази порівняно зі звичайним алгоритмом з використанням листкового розміру 128. Однак зменшення листкового розміру для збільшення кількості рекурсивних кроків не ефективне оскільки мінімальний розмір квадратної матриці з розміром $2^n \times 2^n$ при якому метод Винограда - Штрассена ефективний – 256.

5.2. Результати тестування дропу Q3RP

Спочатку визначимо час обчислення послідовного *Q3RP*-розкладу для того, щоб оцінити прискорення при збільшенні кількості потоків. Результати тестування наведені у таблиці 5.4 та відображені на рисунку 10.

Таблиця 5.4 Результати послідовного виконання алгоритму Q3RP

Розмір даних	**Час виконання, мс**
16	38
32	38
64	79
128	350
256	1622
512	6379
1024	67111
2048	505582

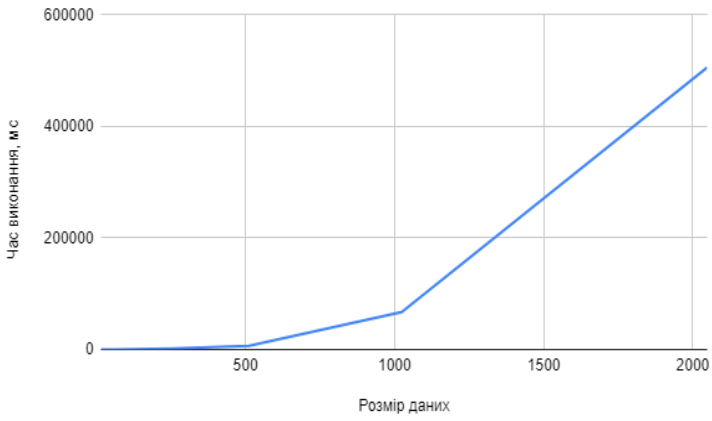

Рисунок 10. Залежність часу виконання Q3RP від розміру даних

Для оцінки прискорення алгоритму зі збільшенням процесорів будемо використовувати матрицю розміром 1024×1024. Найкращий час виконання на цьому розмірі матриці був досягнутий з листковим розміром 256 та прискоренням у 1.44 рази на двох процесорах. Збільшення глибини рекурсії як і в інших алгоритмах значно сповільнює виконання алгоритму, бо час необхідний на розгортання аміну значно більший ніж час його обчислення. В таблиці 5.5 наведені результати запусків алгоритму з різними параметрами листкового розміру та кількості процесорів:

Таблиця 5.5 Результати тестування Q3RP

Процесори	**Розмір даних**	**Розмір листка**	**Час виконання, мс**	**Прискорення**
1	1024	64	460137	0,14
1	1024	128	116387	0,57
1	1024	256	50480	1,33
1	1024	512	82148	0,82
2	1024	64	519403	0,12
2	1024	128	116680	0,57
2	1024	256	46463	1,44
2	1024	512	52904	1,26
4	1024	64	367683	0,18
4	1024	128	108088	0,62
4	1024	256	49355	1,35
4	1024	512	65634	1,02

Висновки

У ході роботи було досліджено застосування методу Ґівенса для знаходження *QR* та *QP* розкладів матриць, а також способи комбінування цих розкладів для задач зведення матриць до простішого виду.

У другому розділі роботи запропоновано алгоритм зведення симетричної квадратної матриці до тридіагональної зі складністю матричного множення застосовуючи комбінації *QR* та *QP* розкладів до окремих блоків матриці. Теоретично описано обидва етапи цього алгоритми та обґрунтовано складність кожного з них. Також було проведено покращення методу *QR*-розкладу з використанням методу пришвидшеного матричного множення Винограда-Штрассена.

На основі алгебраїчних рівнянь був створений обчислювальний граф паралельного блоково-рекурсивного алгоритму *Q3RP* та його програмну реалізацію в рамках середовища блоково-рекурсивних паралельних обчислень DAP02.

У п’ятому розділі наведено та проаналізовано результати тестування алгоритмів *QR* та *Q3RP* розкладів на персональному комп’ютері та обґрунтовано необхідність подальшого тестування алгоритмів на значно більших об’ємах даних. З використанням методу швидкого матричного множення вдалось отримати прискорення в 1.07 рази порівняно зі стандартним алгоритмом. Збільшення глибини рекурсії є ефективним лише тоді, коли час обчислення листка значно більший ніж час необхідний на розгортання аміну. При тестуванні дропу *Q3RP* було отримано прискорення у 1.44 рази, однак він потребує подальшого дослідження на значно більших матрицях.

Список використаної літератури

1. Strassen V. Gaussian elimination is not optimal. *Numerische Mathematik*. 1969. Vol. 13, no. 4. P. 354–356
2. Coppersmith D., Winograd S. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*. 1990. Vol. 9, no. 3. P. 251–280
3. Ford, W. Numerical Linear Algebra with Applications. Academic Press, 2015. pp 351-378. https://doi.org/10.1016/B978-0-12-394435-1.00017-X
4. Layton, W. and Sussman, M. (2014) Numerical Linear Algebra. pp 217-230.
5. Trefethen, L. and Bau, D. (1997). Numerical linear algebra. 1st ed. Philadelphia: SIAM. http://www.math.iit.edu/~fass/477577_Chapter_4.pdf
6. Сірош І. А. Алгоритм SVD для розподіленої пам’яті: маг. … магістра в галузі комп. наук : 121. Київ, 2020. 49 с.
7. Malashonok G. Recursive matrix algorithms, distributed dynamic control, scaling, stability. *2019 Computer Science and Information Technologies (CSIT),* Yerevan, 23–27 September 2019. P. 112–115.
8. Сідько А. А. Середовище виконання для блоково-рекурсивних матричних алгоритмів на суперкомп’ютері з розподіленою пам’яттю : дис. … д-ра філософії в галузі комп. наук : 12. Київ, 2024. 140 с.
9. Малашонок Г. І., Сідько А. А. Розподілені обчислення: ДАП-технологія розпаралелювання рекурсивних алгоритмів. *Наукові записки НаУКМА*. 2018. Т. 1. С. 25–32.

Додаток А

(Обов’язковий)

```
1 usage
private static int[][] _arcs = new int[][]{
        { 1,10,0, 2,8,0, 4,9,5, 4,11,7, 5,3,4, 7,1,0, 8,6,0, 9,7,5, 11,5,0 }, //0 - input
        { 2,1,1, 3,0,3 },                                                     //1 - QR c21
        { 3,0,4, 8,1,1 },                                                     //2 - QP1 c11 c21'
        { 4,0,0, 5,0,0, 6,0,4, 7,0,4, 24,0,7, 25,0,3 },                       //3 - Multiply Q1 * Q2 = Q3
        { 9,2,7, 12,4,2 },                                                    //4 - Multiply Q3 * (cS12 c22)
        { 6,0,0 },                                                            //5 - Multiply Q3 * D
        { 10,1,6, 10,2,7, 12,3,3, 17,4,7 },                                   //6 - Multiply D' * Q3
        { 10,3,4, 10,4,5, 11,1,1, 28,2,11 },                                  //7 - Multiply B * Q3
        { 9,0,0, 10,0,0, 11,0,4, 12,0,4, 23,0,4, 25,0,4, 28,1,8 },            //8 - QP11 (a21, c11)
        { 11,2,2, 12,4,0 },                                                   //9 - Multiply Q4 (a22, c12)
        { 11,1,3, 12,3,1, 16,2,3, 17,4,5 },                                   //10 - Multiply Q4, (b21, b22, d11, d12)
        { 28,1,0, 28,2,1, 28,3,2, 16,4,2 },                                   //11 - Multiply (a12, b11, a22, b21) Q4
        { 14,1,0, 17,2,4, 13,3,0, 17,4,6 },                                   //12 - Multiply (c12, d11, c22, d21) Q4
        { 14,1,1, 15,0,3 },                                                   //13 - QR2 c22
        { 15,0,4, 28,1,4 },                                                   //14 - QP (c12 c22)
        { 16,0,4, 17,0,0, 18,0,4, 22,0,4, 26,0,0, 28,0,6 },                   //15 - Multiply Q5 * Q6 = Q7
        { 28,3,3, 28,4,9 },                                                   //16 - Multiply (b21 b22) Q7
        { 18,1,0, 18,2,1, 18,3,2, 18,4,3 },                                   //17 - Multiply Q7 (d11, d12, d21, d22)
        { 28,1,5, 20,2,1, 19,3,0, 20,4,3 },                                   //18 - Multiply (d11, d12, d21, d22) Q7
        { 20,0,7, 21,0,0, 22,0,3, 26,0,7, 28,1,7 },                           //19 - QR3 d21
        { 28,2,10, 21,4,4 },                                                  //20 - Multiply (d12, d22) Q8
        { 28,0,12 },                                                          //21 - Multiply Q8 d22
        { 23,0,3 },                                                           //22 - Multiply Q8 Q7 = Q9
        { 24,0,0 },                                                           //23 - Multiply Q9 Q4 = Q10
        { 28,0,13 },                                                          //24 - Multiply Q10 Q3 = QL
        { 27,0,0 },                                                           //25 - Multiply Q3 Q4 = Q11
        { 27,0,7 },                                                           //26 - Multiply Q7 Q8 = Q12
        { 28,0,14 },                                                          //27 - Multiply Q11 Q12 = QR
        {}//28
};
```

Рисунок 11.Представлення обчислювального графу алгоритму Q3RP