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

Розкладання Холецького: блочно-рекурсивний алгоритм та програма на кількості процесорів 2^n

Текстова частина до курсової роботи за спеціальністю „Комп'ютерні науки та інформаційні технології" - 122

Керівник курсової роботи

проф., д. ф.-м. н. Малашонок Г. І.

_____

"____" _____ 2020 р.

Виконав студент 4 курсу бакалаврської програми

Комп'ютерні науки та інформаційні технології

Іваськевич А.Я. "____" _____ 2020 р.

Київ 2020

Тема: Розклад Холецького: блочно-рекурсивний алгоритм на кількості процесорів 2^n та програма

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	24.12.2019	
2.	Огляд технічної літератури за темою роботи.	25.12.2019	
3.	Аналіз паралельного алгоритму розкладання матриць за методом Холецького.	25.12.2019	
3.	Розробка паралельної програми розкладання матриць за методом Холецького.	12.03.2020	
4.	Тестування розробленої програми.	16.03.2020	
5.	Аналіз тестування та виправлення помилок.	20.03.2020	
6.	Написання пояснювальної роботи.	25.03.2020	
7.	Створення слайдів для доповіді та написання доповіді.	10.04.2020	
8.	Аналіз отриманих результатів з керівником.	10.04.2020	
9.	Коригування роботи за результатами попереднього захисту.	15.04.2020	
10.	Захист курсової роботи.	24.04.2020	

Студент Іваськевич А. Я.

Керівник Малашонок Г.І.

“_____” _____

Зміст

Анотація

У роботі розглянуто блочно-рекурсивний алгоритм розкладу Холецького, для знаходження нижньотрикутної матриці. Основну увагу приділено розробці універсальної паралельної програми, для розв’язання такого типу задач на кількості процесорів 2 в степені n, де n - натуральне число. В результаті дослідження було розроблено алгоритм управління паралельним обчислювальним процесом на суперкомп’ютері з розподіленою пам’яттю, для блочно-рекурсивних алгоритмів.

Ключові слова: алгоритм, паралельне програмування, процесор, матриця, блочний алгоритм, множення, обернення, транспонування, рекурсія, вхідні, вихідні дані , результат.

Вступ

Ціллю даної курсової роботи була розробка статичної програми, для блочно-рекурсивного алгоритму розкладу Холецького, та отримання результатів тестування в якості користувацьких інструкцій для запуску. Під час розробки статичної програми міжпроцесорні обміни плануються завчасно до виконання програми.

Під час дослідження теми курсової роботи, було прийнято рішення розробити блочно-рекурсивний алгоритм множення матриць та множення матриць з додаванням на кількості процесорів 2 в степені n, де n - натуральне число.

Основна задача полягала в створенні універсальної паралельної програми для кластера з розподіленою пам’яттю, яка , для будь-якої додатньо-визначеної симетричної матриці A, знайде нижню трикутну матрицю B таку , що A = B * B’ , де B’ - транспонована матриця з вказаною точністю розрахунків.

Розділ 1: Опис алгоритму розкладання матриць за методом Холецького

1.1 Означення та теореми

ОЗНАЧЕННЯ. Кожен мінор матриці, що знаходиться у верхньому лівому кутку, називається діагональним мінором.

ОЗНАЧЕННЯ. Матриця називається позитивно-визначеною, якщо всі її діагональні мінори строго позитивні.

ОЗНАЧЕННЯ. Квадратна матриця називається симетричною, якщо її елементи симетричні відносно головної діагоналі, це означає, що $A = A^T$.

ТЕОРЕМА 1.

Нехай L невироджена нижньотрикутна матриця над дійсними числами. Тоді матриця LL^T є позитивно-визначеною.

ДОВЕДЕННЯ.

Нехай $L = (a_{i,j})$, $i,j = 1,..,n$, $a_{i,j} = 0$ при $i < j$.. В матриці LL^T діагональний мінор n-го порядку дорівнює $\prod_{i=1}^{n} a_{i,i}^2$ і є, очевидно, позитивним.

ТЕОРЕМА 1А.

Нехай L невироджена нижньо-трикутна матриця над комплексними числами. Тоді матриця LL^* є позитивно-визначеною.

ДОВЕДЕННЯ. Аналогічно Теоремі 1.

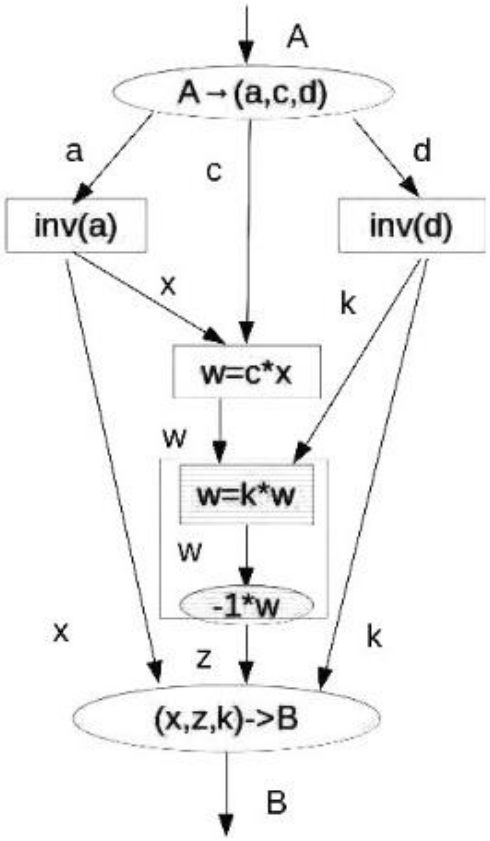
1.2 Знаходження оберненої матриці для нижньо-трикутної матриці.

Розглянемо, як знаходиться обернена матриця для трикутної матриці. Розіб'ємо матриці на блоки і обчислимо вираз:

$$\begin{pmatrix} a & 0 \\ b & c \end{pmatrix} \begin{pmatrix} x & 0 \\ z & y \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}.$$

Звідси випливає, що: $\begin{pmatrix} x & 0 \\ z & y \end{pmatrix} = \begin{pmatrix} a^{-1} & 0 \\ -c^{-1}ba^{-1} & c^{-1} \end{pmatrix}$

1.3 Граф рекурсивного знаходження оберненої матриці для нижньо-трикутної матриці.



1.4 Загальне визначення розкладу Холецького

Розклад Холецького - це представлення симетричної позитивно-визначеної матриці A у вигляді $A = LL^T$, де L нижньо-трикутна матриця з додатними елементами на діагоналі. Розклад Холецького завжди існує і єдиний для будь-якої симетричної позитивно-визначеної матриці.

1.4.1 Короткі відомості про автора алгоритму

Розклад названо на честь французького математика польського походження Андре-Луї Холецького (1875 - 1918).

Андре-Луї служив офіцером в французькій армії і був убитий в кінці першої світової війни. Алгоритм, відкритий Холецьким, був опублікований його колегою, офіцером Бенотою, через 6 років в журналі . Стаття отримала назву: «Зауваження про метод вирішення нормальних рівнянь, отриманому при застосуванні методу найменших квадратів до системи лінійних рівнянь, в кількості, меншій ніж число невідомих (алгоритм офіцера Холецького)».

1.4.2 Використання алгоритму

Цей розклад може використовуватись для розв'язання системи лінійних рівнянь $Ax = b$, якщо матриця A симетрична і позитивно-визначена. Такі матриці часто виникають, при використанні методу найменших квадратів і чисельному розв'язку диференціальних рівнянь. Основна ідея методу Холицького полягає у розкладі матриці коефіцієнтів на добуток двох матриць — нижньої трикутної та транспонованої до неї матриці і в подальшому рішення системи зводиться до послідовного вирішення двох систем рівнянь з трикутними матрицями. Виконавши розклад $A = LL^T$, розв'язок x можна отримати послідовним вирішенням двох трикутних систем рівняння: $Ly = b$ та $L^T x = y$. Такий спосіб вирішення іноді називають методом квадратних коренів.

Розклад Холецького також використовується в методах Монте-Карло для генерації корельованих випадкових величин та в інших багатьох випадках.

1.4.3 Блочно-рекурсивний розклад Холецького

Нехай матриця A - позитивно-визначена симетрична матриця і $A = LL^T$.

Нехай: $L = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix}, A = \begin{pmatrix} \alpha & \beta \\ \beta & \gamma \end{pmatrix}.$

Тоді: $A = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix} \begin{pmatrix} a^T & b^T \\ 0 & c^T \end{pmatrix} = \begin{pmatrix} aa^T & ab^T \\ ba^T & bb^T + cc^T \end{pmatrix}$

Знайдемо блоки матриці *L*:

1. $aa^T = \alpha$ (рекурсивний крок по a)
2. $ab^T = \beta; b^T = a^{-1} * \beta; b = \beta^T (a^{-1})^T$
3. $bb^T + cc^T = \gamma; cc^T = \gamma - bb^T$ (рекурсивний крок по c)

1.5 Блочно-рекурсивний алгоритм розкладу Холецького

Дано матрицю: $A = \begin{pmatrix} \alpha & \beta \\ \beta & \gamma \end{pmatrix}$.

Необхідно знайти матрицю: $L = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix}$

.

Ми будемо знаходити матрицю L і обернену до неї L^{-1}.

CholeskyDecomp(A) = (L,L^{-1})

if(size(A) == 1 & A = [α]) then return ([sqrt(α)],[1/sqrt(α)])

else A --> (α, β, γ) - ``we create three blocks``

(a,a_1) = choleskyDecomp(α)

$b^T = a_1 * \beta$; $b = (b^T)^T$; $\delta = \gamma - b * b^T$;

(c,c_1) = choleskyDecomp(δ)

$z = - c_1 * b * a_1$;

return $\begin{pmatrix} a & 0 \\ b & c \end{pmatrix}, \begin{pmatrix} a_1 & 0 \\ z & c_1 \end{pmatrix}$

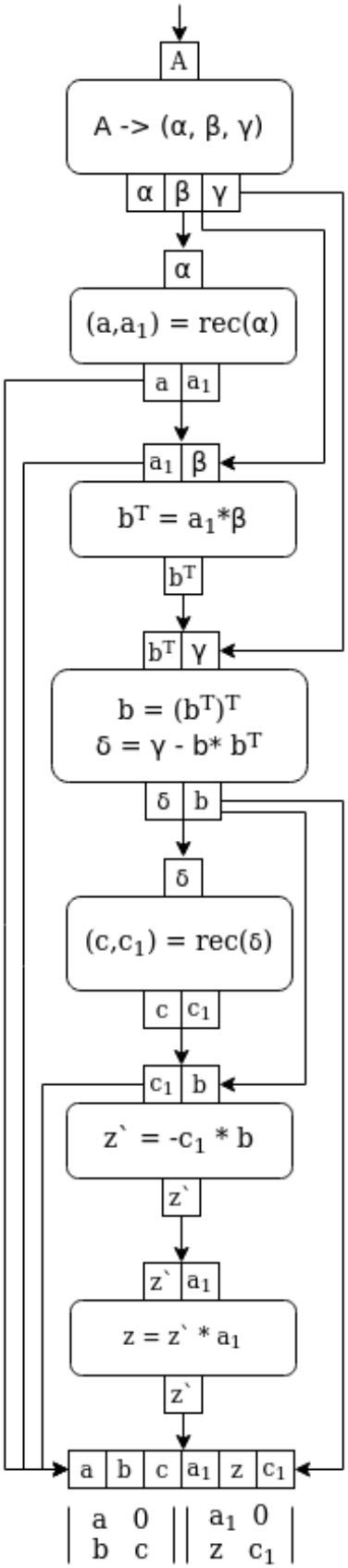
1.6 Приклад розкладу Холецького

$$A = \begin{pmatrix} 16 & 24 & 28 & 4 \\ 24 & 72 & 42 & 42 \\ 28 & 42 & 85 & 13 \\ 4 & 42 & 13 & 74 \end{pmatrix},\ \alpha = \begin{pmatrix} 16 & 24 \\ 24 & 72 \end{pmatrix},\ \beta = \begin{pmatrix} 28 & 4 \\ 42 & 42 \end{pmatrix},\ \gamma = \begin{pmatrix} 85 & 13 \\ 13 & 74 \end{pmatrix}.$$

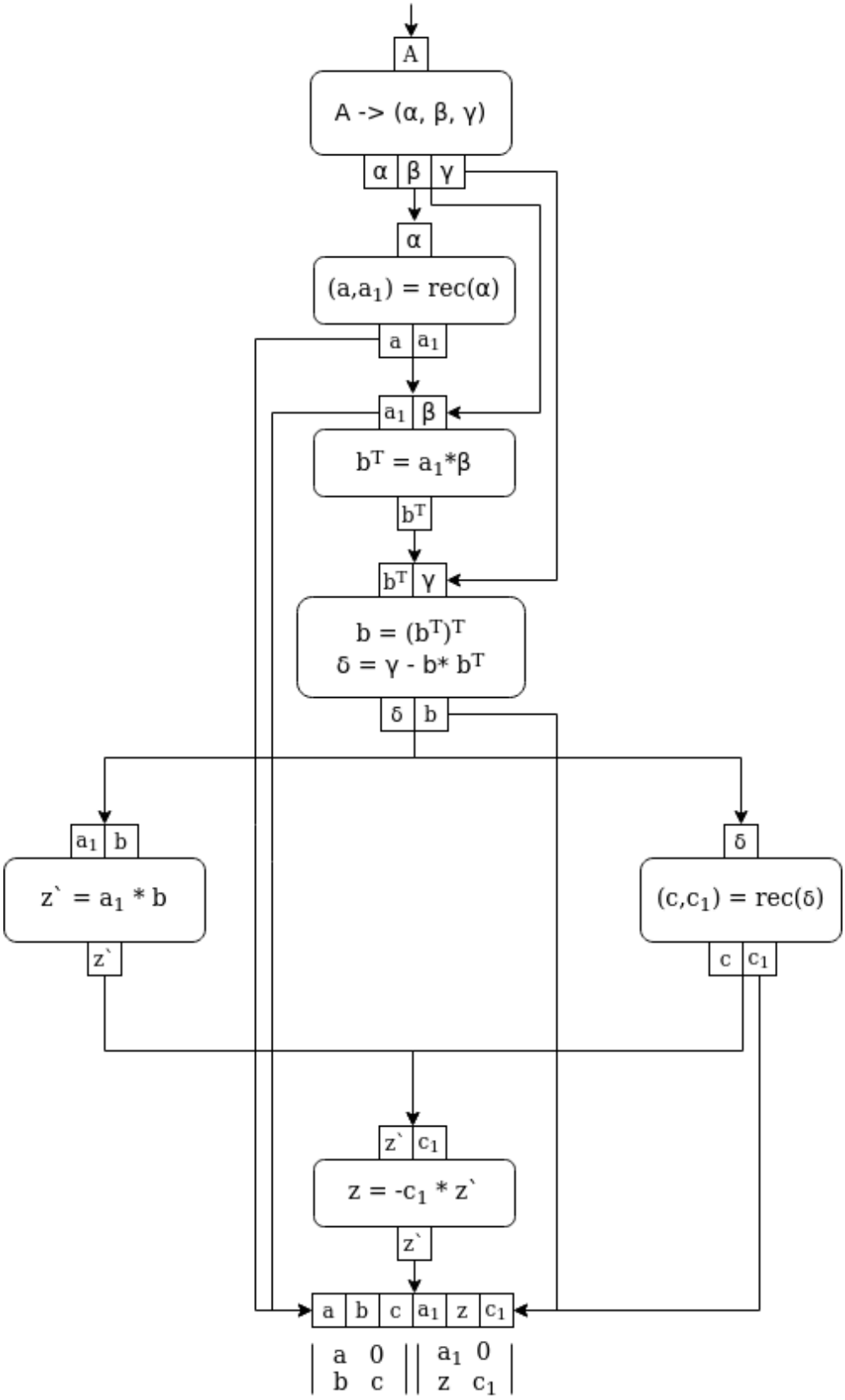
$1.(a, a_1) = [\begin{pmatrix} 4 & 0 \\ 6 & 6 \end{pmatrix},\ \begin{pmatrix} 1/4 & 0 \\ -1/4 & 1/6 \end{pmatrix}] = Cholesky(\alpha)$
with $\alpha_1 = 16, \beta_1 = 24, \gamma_1 = 72,\ (a', a_1') = [4, 1/4] = Cholesky(\alpha_1)$,
$b' = \beta_1/a' = 6,\ \delta_1 = \gamma_1 - (b')^2 = 36,$
$(c', c_1') = [6, 1/6] = Cholesky(\delta_1),\ z' = -c_1' b' a_1' = -1/4.$
$2. b^T = a_1 \cdot\ \beta = \begin{pmatrix} 1/4 & 0 \\ -1/4 & 1/6 \end{pmatrix} \begin{pmatrix} 28 & 4 \\ 42 & 42 \end{pmatrix} = \begin{pmatrix} 7 & 1 \\ 0 & 6 \end{pmatrix};\ b = \begin{pmatrix} 7 & 0 \\ 1 & 6 \end{pmatrix}$
$3. \delta = \gamma - b \cdot\ b^T = \begin{pmatrix} 85 & 13 \\ 13 & 74 \end{pmatrix} - \begin{pmatrix} 7 & 0 \\ 1 & 6 \end{pmatrix} \begin{pmatrix} 7 & 1 \\ 0 & 6 \end{pmatrix} = \begin{pmatrix} 36 & 6 \\ 6 & 37 \end{pmatrix}$
$4. (c, c_1) = [\begin{pmatrix} 6 & 0 \\ 1 & 6 \end{pmatrix},\ \begin{pmatrix} 1/6 & 0 \\ -1/36 & 1/6 \end{pmatrix}] = Cholesky(\delta)$
with $\alpha_2 = 36, \beta_2 = 6, \gamma_2 = 37,\ (a'', a_1'') = [6, 1/6] = Cholesky(\alpha_2)$,
$b'' = \beta_2/a'' = 1,\ \delta_2 = \gamma_1 - (b'')^2 = 36,\ (c'', c_1'') = [6, 1/6] = Cholesky(\delta_2),$
$z'' = -c_2'' b'' a_2'' = -1/6 \cdot 1 \cdot\ 1/6 = -1/36.$
$5. z = -\begin{pmatrix} 1/6 & 0 \\ -1/36 & 1/6 \end{pmatrix} \begin{pmatrix} 7 & 0 \\ 1 & 6 \end{pmatrix} \begin{pmatrix} 1/4 & 0 \\ -1/4 & 1/6 \end{pmatrix} = \begin{pmatrix} -7/24 & 0 \\ 37/144 & -1/6 \end{pmatrix}$

$$L = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix} = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 6 & 6 & 0 & 0 \\ 7 & 0 & 6 & 0 \\ 1 & 6 & 1 & 6 \end{pmatrix},\ L^{-1} = \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ -1/4 & 1/6 & 0 & 0 \\ -7/24 & 0 & 1/6 & 0 \\ 37/144 & -1/6 & -1/36 & 1/6 \end{pmatrix}.$$

1.7 Граф блочно-рекурсивного послідовного алгоритму розкладу Холецького



1.8 Граф блочно-рекурсивного паралельного алгоритму розкладу Холецького



1.9 Блочний алгоритм множення матриць

Паралельне обчислення розкладу Холецького відбувається завдяки блочному множенню матриць.

Нехай дано матриці:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \text{ і } B = \begin{pmatrix} x & y \\ z & k \end{pmatrix},$$

де a, b, c, d, x, y, z, k - блоки матриць A і B відповідно.

Необхідно знайти матрицю C = *A∗B.*

Знайдемо блоки матриці :

1). C_{11} =a*x + b*z

2). C_{12} =a*y + b*k

3). C_{21} =c*x + d*z

4). C_{22} =c*y + d*k

Як бачимо, всі множення блоків незалежно один від одного і можуть виконуватись паралельно.

1.10 Блочний алгоритм множення матриць на 4 процесорах

*if(size(A) == 1) then return (C =A*B) else*

1) A−>(a,b,c,d),

B−> (x,y,z,k)

2) $C_{11} = b * z + a * x$

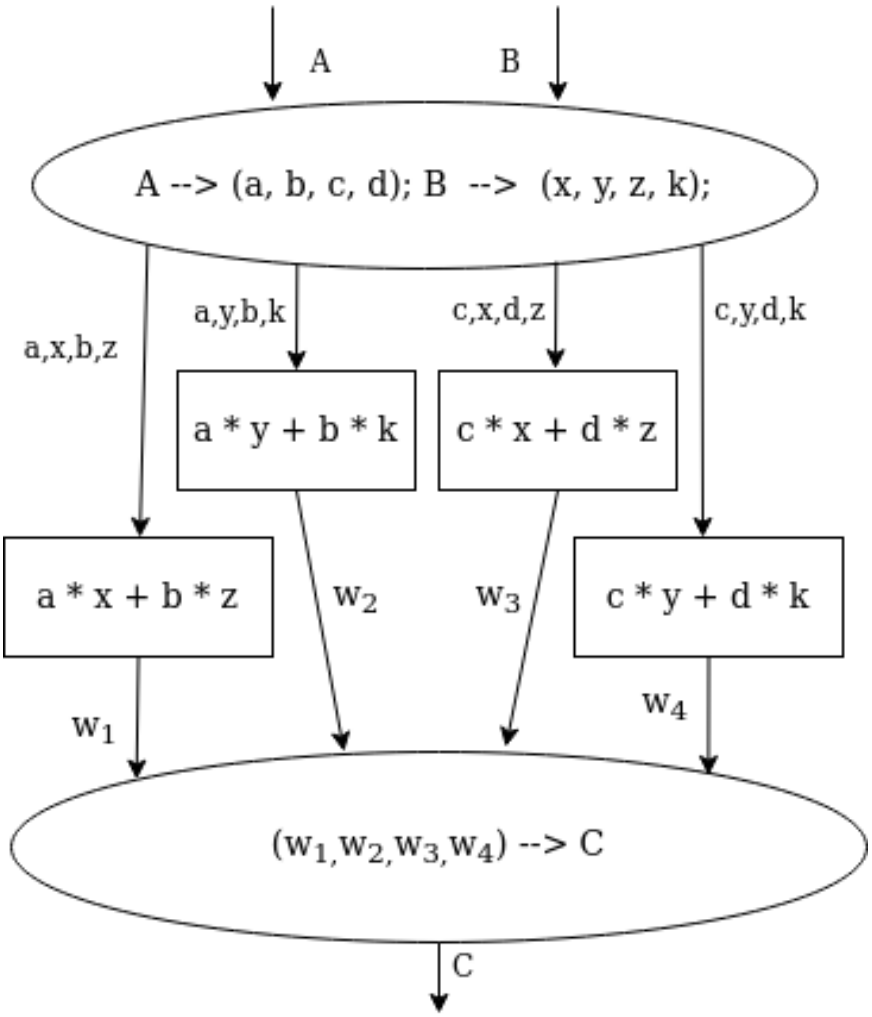
3) $C_{12} = b * k + a * y$

4) $C_{21} = d * z + c * x$

5) $C_{22} = d * k + c * y$

6) return $\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

1.11 Граф блочного алгоритму множення матриць на 4 процесорах



1.12 Блочний алгоритм множення матриць на 8 процесорах

*if(size(A) == 1) then return (C =A*B) else*

1) *A−>(a,b,c,d),*

   B−> (x,y,z,k)

2) $\alpha = a * x$

3) $C_{11} = b * z + \alpha$

4) $\beta = a * y$

5) $C_{12} = b * k + \beta$

6) $\gamma = c * x$

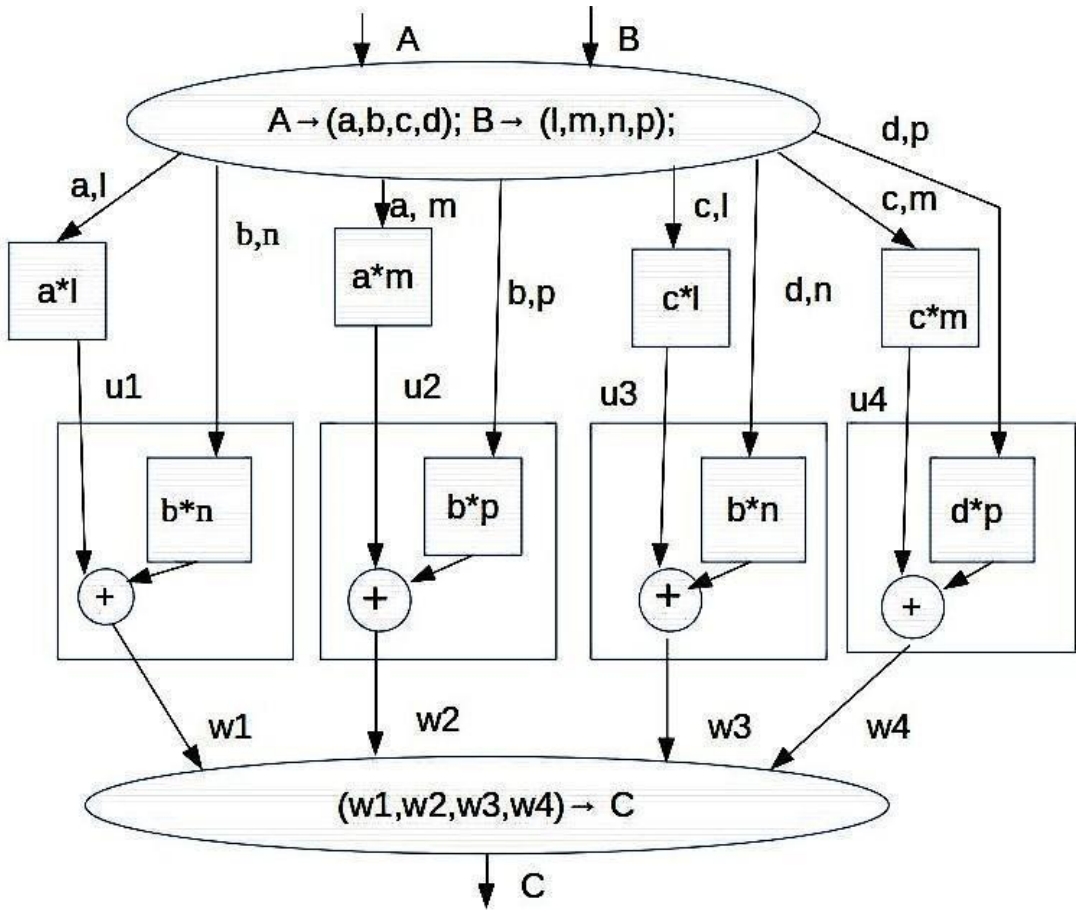
7) $C_{21} = d * z + \gamma$

8) $\delta = c * y$

9) $C_{22} = d * k + \delta$

10) return $\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

1.13 Граф блочного алгоритму множення матриць на 8 процесорах



Розділ 2: Структура програмної реалізації алгоритму.

2.1 Блочно-рекурсивне множення матриць.

MatrixS MatrixMul.**multiplyMatrix**(MatrixS A, MatrixS B, Ring ring, Intracomm intracomm, int leaf)

A - перша матриця для множення,

B - друга матриця для множення,

ring - алгебраїчний простір змінних,

intracomm - комутатор для транспорту,

leaf - розмір листа переходу на послідовний алгоритм.

MatrixMul.**multiplyMatrix**(MatrixS A, MatrixS B, Ring ring, Intracomm intracomm, int leaf) - множення двох матриць з результатом на процесорі , ранк якого доріанює нулю. Даний метод, з таким набором параметрів, може бути використаний в якості послідовного алгоритму множення матриць, а, також, в якості паралельного алгоритму для процесора , ранк якого дорівнює нулю.

MatrixS MatrixMul.**multiplyMatrix**(Ring ring, Intracomm intracomm, int leaf)

ring - алгебраїчний простір змінних,

intracomm - комутатор для транспорту,

leaf - розмір листа переходу на послідовний алгоритм.

MatrixMul.**multiplyMatrix**(Ring ring, Intracomm intracomm, int leaf) - даний метод, з таким набором параметрів, може бути використаний в якості паралельного алгоритму для процесорів , ранк яких більший нуля.

В підсумку, щоб помножити дві матриці послідовно, використовуючи методи даного класу, достатньо буде виклику першого, з передачею йому, в якості агрумента, комутатора, що містить один процесор. В разі використання методів даного класу для множення двох матриць паралельно, процесор з ранком 0 повинен отримати вхід у перший з них, в той час, як всі інші - у другий. Таким чином, як вхідними, так і вихідними даними володітиме тільки процесор , ранк якого рівний нулю.

2.2 Блочно-рекурсивне множення матриць з додаванням.

MatrixS MatrixMul.**multiplyMatrixWithAdd**(MatrixS A, MatrixS B, MatrixS C, Ring ring, Intracomm intracomm, int leaf)

A - перша матриця для множення,

B - друга матриця для множення,

C - матриця для додавання,

ring - алгебраїчний простір змінних,

intracomm - комутатор для транспорту,

leaf - розмір листа переходу на послідовний алгоритм.

MatrixMul.**multiplyMatrixWithAdd**(MatrixS A, MatrixS B, MatrixS C, Ring ring, Intracomm intracomm, int leaf) - множення двох матриць з додаванням результату множення до третьої матриці. Результат множення з додаванням буде отримано на процесорі , ранк якого доріанює нулю. Даний метод, з таким набором параметрів, може бути використаний в якості послідовного алгоритму множення матриць з додаванням, а, також, в якості паралельного алгоритму для процесора , ранк якого дорівнює нулю.

MatrixS MatrixMul.**multiplyMatrixWithAdd**(Ring ring, Intracomm intracomm, int leaf)

ring - алгебраїчний простір змінних,

intracomm - комутатор для транспорту,

leaf - розмір листа переходу на послідовний алгоритм.

MatrixMul.**multiplyMatrixWithAdd**(Ring ring, Intracomm intracomm, int leaf) - множення двох матриць з додаванням результату множення до третьої матриці. Даний метод з таким набором параметрів може бути

використаний в якості паралельного алгоритму для процесорів , ранк яких більший нуля.

В підсумку, щоб помножити дві матриці з додаванням третьої послідовно, використовуючи методи даного класу, достатньо буде виклику першого, з передачею йому, в якості агрумента, комутатора, що містить один процесор. В разі використання методів даного класу для множення двох матриць з додаванням третьої паралельно, процесор з ранком 0 повинен отримати вхід у перший з них, в той час, як всі інші - у другий. Таким чином, як вхідними, так і вихідними даними володітиме тільки процесор , ранк якого рівний нулю.

2.3 Блочно-рекурсивний розклад Холецького.

CholeskyDecomposition.**choleskyDecomposition**(MatrixS A, Ring ring, Intracomm intracomm, int leaf)

A - матриця для розкладу,

ring - алгебраїчний простір змінних,

intracomm - комутатор для транспорту,

leaf - розмір листа переходу на послідовний алгоритм.

CholeskyDecomposition.**choleskyDecomposition**(MatrixS A, Ring ring, Intracomm intracomm, int leaf) - розкладу Холецького з результатом на процесорі , ранк якого рівний нулю. Даний метод, з таким набором параметрів, може бути використаний в якості послідовного алгоритму розкладу Холецького а, також, в якості паралельного алгоритму для процесора , ранк якого дорівнює нулю.

CholeskyDecomposition.**choleskyDecomposition**(Ring ring, Intracomm intracomm, int leaf)

ring - алгебраїчний простір змінних,

intracomm - комутатор для транспорту,

leaf - розмір листа переходу на послідовний алгоритм.

CholeskyDecomposition.**choleskyDecomposition**(MatrixS A, Ring ring, Intracomm intracomm, int leaf) - даний метод, з таким набором параметрів, може бути використаний в якості паралельного алгоритму розкладу Холецького для процесорів з ранком більшим за нуль.

В підсумку, щоб отримати результат розкладу Холецького послідовно, використовуючи методи даного класу, достатньо буде виклику першого, з передачею йому, в якості аргумента, комутатора, що містить один процесор. В разі використання методів даного класу для розкладу Холецького паралельно, процесор з ранком 0 повинен отримати вхід у перший з них, в той час, як всі інші - у другий. Таким чином, як вхідними, так і вихідними даними володітиме тільки процесор , ранк якого рівний нулю, що властиво і методам всередині алгоритму.

Розділ 3: Тестування програмної реалізації алгоритму. Рекомендації до запуску на основі результатів тестування.

3.1 Порівняльне тестування паралельної та послідовної реалізації алгоритму на локальному комп'ютері.

Щоб порівняти результати паралельної та послідовної реалізації, було проведено по дві серії тестів на матрицях розміром від 8 до 256, з кількістю знаків після коми в елементах матриць 300 та 400 відповідно. Послідовний алгоритм використовував один процесор, в той час, як паралельний - 4 , що пов'язано з характеристиками локального пристрою, в наявності якого саме така кількість процесорів.

Послідовний алгоритм (1 процесор)					
Matrix size	Bit	Accuracy	Exec time	Max mistake	Max number
8x8	8	300	0.69 sec	8.045E-297	133369
16x16	8	300	1 sec	3.90E-295	376904
32x32	7	300	4 sec	1.50E-287	206094
64x64	7	300	17 sec	4.65E-272	372868
128x128	7	300	151 sec	8.61E-249	736155
256x256	6	300	1001 sec	5.31E-198	383073
8x8	8	400	0.96 sec	3.487E-396	272873
16x16	8	400	1 sec	8.61E-391	413391
32x32	7	400	6 sec	1.95E-388	183305
64x64	7	400	29 sec	2E-382	388930

128x128	7	400	216 sec	3.11E-352	762478
256x256	6	400	1697 sec	2.66E-300	362334

Паралельний алгоритм (4 процесори)					
Matrix size	Bit	Accuracy	Exec time	Max mistake	Max number
8x8	8	300	1 sec	5.71E-293	192445
16x16	8	300	2 sec	1.435E-294	425675
32x32	7	300	7 sec	1.26E-283	182067
64x64	7	300	13 sec	1.38E-275	416966
128x128	7	300	63 sec	1.52E-259	786675
256x256	6	300	477 sec	7.38E-196	374351
8x8	8	400	2 sec	7.35E-397	256376
16x16	8	400	3 sec	2.77E-391	395205
32x32	7	400	10 sec	2.64E-391	206751
64x64	7	400	19 sec	2.31E-370	420591
128x128	7	400	99 sec	3.99E-349	750172
256x256	6	400	762 sec	3.99E-303	375102

За результатами тестів можна чітко побачити, що , починаючи з матриць розміром 64x64, паралельний алгоритм, за часом виконання, працює швидше послідовного. Дана тенденція зберігатиметься зі збільшенням розміру матриці, причому, відношення часу виконання послідовного алгоритму до часу виконання паралельного ставатиме більшим. Щодо максимальної похибки в обчисленні , тут результати в двох

реалізаціях алгоритму майже однакові, з невеликою різницею , що пов’язано з різними вхідними даними.

3.2 Рекомендації для користувача при виборі точності обчислень, на основі аналізу результатів тестів на кластері.

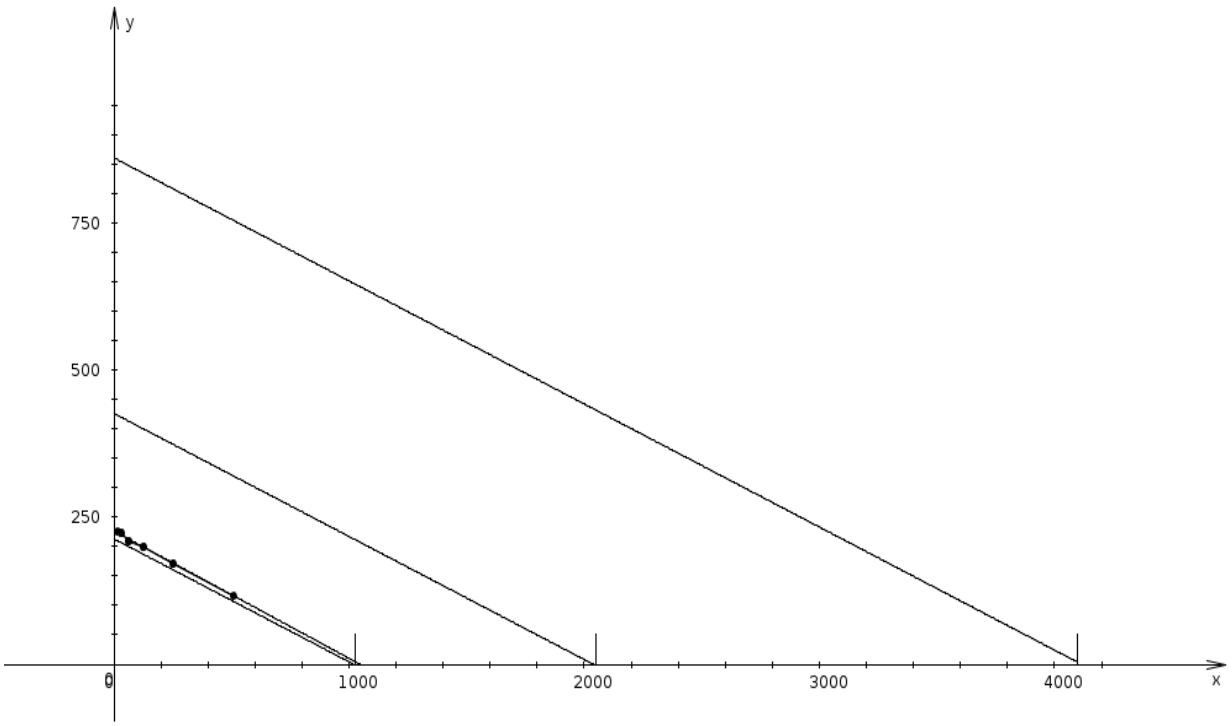
Тестування програм проводились на матрицях різних розмірів та з різними елементами за кількістю десяткових цифр у коефіцієнтів. Таким чином, було вирішено провести три серії тестів з різною кількістю десяткових цифр у коефіцієнтів в елементах матриці , а саме : три , шість, та дванадцять. Було застосовано генератор випадкових чисел з рівномірною щільністю заповнення , що дозволило отримати щільно заповнену вхідну матрицю.

Результати першої серії тестів:

Кількість десяткових цифр коефіцієнтів матриці - 3(4)						
Matrix size	Proc number	Leaf	Accuracy	Exec time	Max mistake	Max number
16x16	16	4	20	0.13 sec	1E-14	254
16x16	16	8	20	0.35 sec	5.52E-14	254
32x32	16	4	30	0.42 sec	1.26E-15	628
32x32	16	8	30	0.27 sec	1.35E-15	628
64x64	16	4	50	0.40 sec	5.81E-36	250

64x64	16	8	50	0.29 sec	5.74E-36	250
128x128	16	4	60	0.93 sec	6.71E-28	501
128x128	16	8	60	0.90 sec	8.14E-28	501
256x256	64	4	120	6 sec	2.69E-61	948
256x256	64	8	120	6 sec	1.43E-61	948
512x512	128	4	230	52 sec	6.90E-116	1855
512x512	128	8	230	56 sec	9.80E-116	1855

За результатами першої серії тестів було побудовано наступний графік, де вісь абсцис - розмір матриці, вісь ординат - величина похибки :



Виходячи з результатів графіку, отримано наступні рекомендації запуску алгоритму, щодо точності обчислень :

Рекомендації до запуску ***Кількість десяткових цифр коефіцієнтів матриці - 3(4)***				
Matrix size	Accuracy		Matrix size	Accuracy
8x8	2		256x256	60
16x16	5		512x512	110
32x32	10		1024x1024	220
64x64	20		2048x2048	440
128x128	30		4096x4096	870

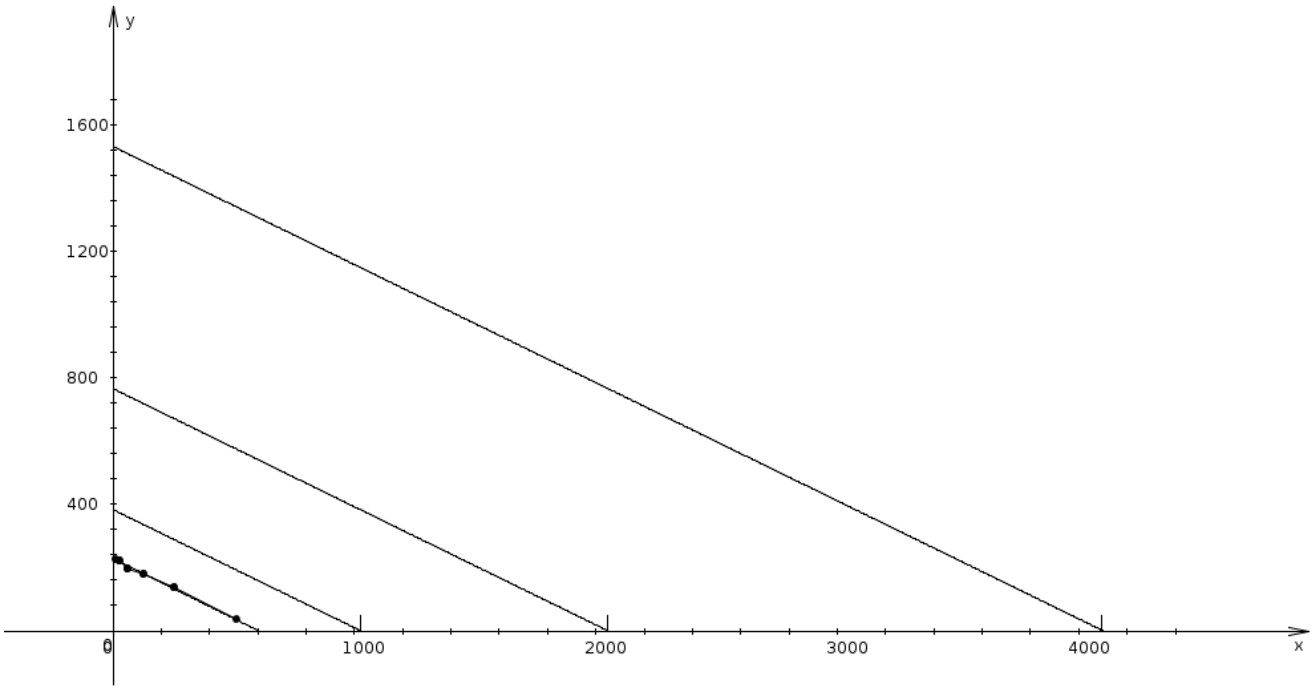
Друга серія тестів проводилась з матрицями різних розмірів, елементи яких містять шість коефіцієнтів.

Результати другої серії тестів:

Кількість десяткових цифр коефіцієнтів матриці - 6						
Matrix size	Proc number	Leaf	Accuracy	Exec time	Max mistake	Max number
16x16	16	4	20	0.16 sec	2.2E-15	470276
16x16	16	8	20	0.57 sec	4.9E-15	470276
32x32	16	4	30	0.40 sec	6.114E-22	806453

32x32	16	8	30	0.23 sec	3.204E-22	806453
64x64	16	4	50	0.42 sec	1.82E-17	352222
64x64	16	8	50	0.31 sec	2.92E-17	352222
128x128	16	4	60	1 sec	6.49E-7	756830
128x128	16	8	60	1 sec	6.49E-7	756830
256x256	64	4	120	6 sec	5.50E-26	388676
256x256	64	8	120	6 sec	6.44E-26	388676
512x512	128	4	230	56 sec	1.18E-34	731960
512x512	128	8	230	58 sec	5.20E-35	731960

За результатами другої серії тестів було побудовано наступний графік, де вісь абсцис - розмір матриці, вісь ординат - величина похибки :



Виходячи з результатів графіку, отримано наступні рекомендації запуску алгоритму, щодо точності обчислень :

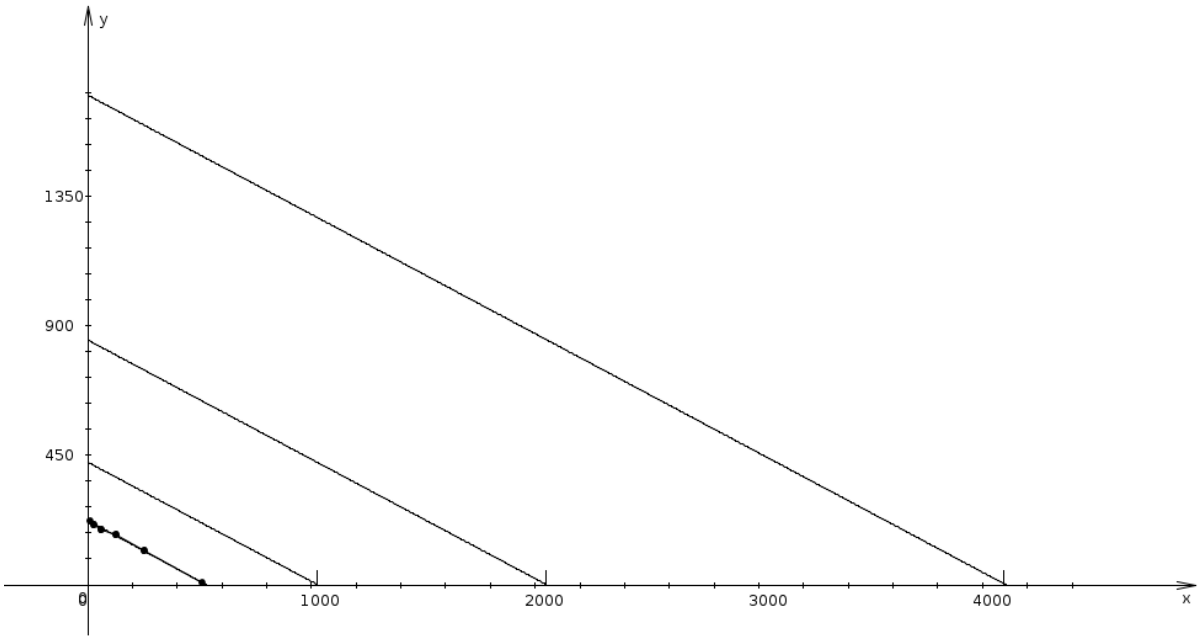
Рекомендації до запуску ***Кількість десяткових цифр коефіцієнтів матриці - 6***				
Matrix size	Accuracy		Matrix size	Accuracy
8x8	5		256x256	90
16x16	10		512x512	170
32x32	20		1024x1024	340
64x64	30		2048x2048	780
128x128	50		4096x4096	1550

Третя серія тестів проводилась з матрицями різних розмірів, елементи яких містять дванадцять коефіцієнтів.

Результати третьої серії тестів:

Кількість десяткових цифр коефіцієнтів матриці - 12						
Matrix size	Proc number	Leaf	Accuracy	Exec time	Max mistake	Max number
16x16	16	4	20	0.15 sec	1.10E-9	397934320088
16x16	16	8	20	0.10 sec	1.11E-9	397934320088
32x32	16	4	30	0.39 sec	4.40E-10	803152697905
32x32	16	8	30	0.21 sec	1.72E-11	803152697905
64x64	16	4	50	0.35 sec	1.31E-12	410535019626
64x64	16	8	50	0.32 sec	1.30E-12	410535019626
128x128	16	4	60	1 sec	8.4E-3	782476652357
128x128	16	8	60	1 sec	8.3E-3	782476652357
256x256	64	4	120	6 sec	5.10E-7	404453753083
256x256	64	8	120	6 sec	1.10E-7	404453753083
512x512	128	4	230	58 sec	1.61E-7	194871084180
512x512	128	8	230	60 sec	3.47E-8	194871084180

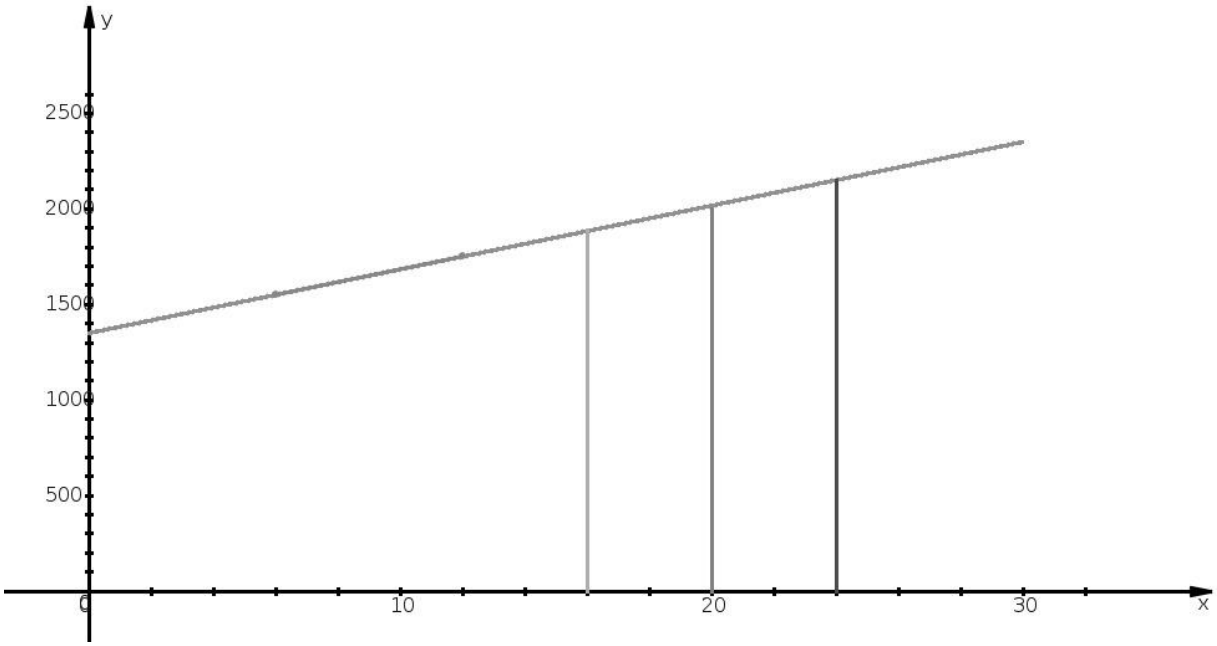
За результатами третьої серії тестів було побудовано наступний графік, де вісь абсцис - розмір матриці, вісь ординат - величина похибки :



Виходячи з результатів графіку, отримано наступні рекомендації запуску алгоритму, щодо точності обчислень :

Рекомендації до запуску ***Кількість десяткових цифр коефіцієнтів матриці - 12***				
Matrix size	Accuracy		Matrix size	Accuracy
8x8	5		256x256	110
16x16	10		512x512	220
32x32	20		1024x1024	440
64x64	30		2048x2048	870
128x128	60		4096x4096	1750

За результатами трьох серій було побудовано ще один графік, з прогнозом для точності серед матриць з кількістю десяткових цифр у коефіцієнтах : 16, 20, 24.



Точки перетину : ***(16, 1890)*** , ***(20, 2000)*** , ***(24, 2150)***.

Проаналізувавши графік можна зпрогнозувати наступні рекомендації до запуску :

Рекомендації до запуску ***Кількість десяткових цифр коефіцієнтів матриці - 16***				
Matrix size	Accuracy		Matrix size	Accuracy
8x8	5		256x256	120
16x16	10		512x512	240
32x32	20		1024x1024	480
64x64	30		2048x2048	950

128x128	60		4096x4096	1890

Рекомендації до запуску Кількість десяткових цифр коефіцієнтів матриці - 20				
Matrix size	Accuracy		Matrix size	Accuracy
8x8	5		256x256	130
16x16	10		512x512	250
32x32	20		1024x1024	500
64x64	40		2048x2048	1000
128x128	70		4096x4096	2000

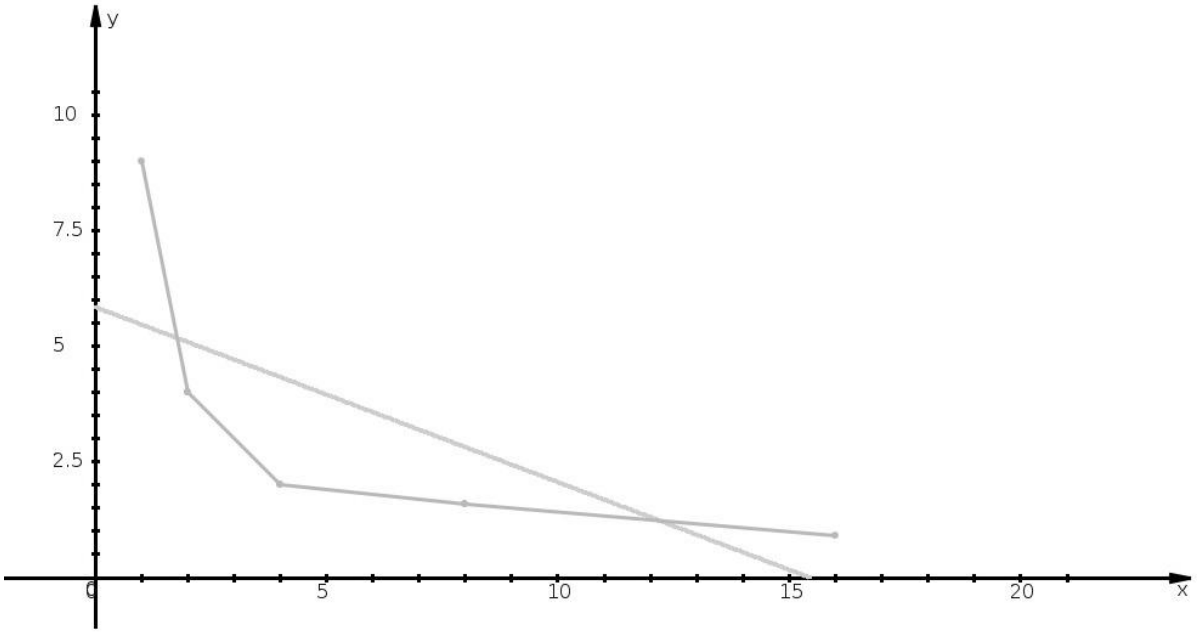
Рекомендації до запуску Кількість десяткових цифр коефіцієнтів матриці - 24				
Matrix size	Accuracy		Matrix size	Accuracy
8x8	5		256x256	140
16x16	10		512x512	270
32x32	20		1024x1024	540
64x64	40		2048x2048	1080
128x128	70		4096x4096	2150

3.3 Рекомендації для запуску, щодо кількості процесорів та розміру листа переходу на послідовний алгоритм, на основі аналізу результатів тестів на кластері.

В таблицях зібрані результати тестів з листами переходу на послідовний алгоритм 8, 16, та 32, в зв’язку з найкращими часовими результатами незалежно від розміру вхідної матриці, починаючи з 64.

Серія тестів на матриці розміром 128				
Matrix size	Proc numb	Leaf	Exec time	Max mistake
128x128	1	8	10 sec	1.3E-10
128x128	1	16	9 sec	1.3E-10
128x128	2	8	4 sec	1.1E-8
128x128	2	16	4 sec	1.1E-8
128x128	4	8	2.1 sec	3.8E-9
128x128	4	16	2.1 sec	3.8E-9
128x128	8	8	1.2 sec	4E-8
128x128	8	16	1.3 sec	4E-8
128x128	16	8	0.9 sec	7.8E-7
128x128	16	16	0.8 sec	8.1E-7

За результатами серії було побудовано наступний графік , де вісь абсцис - кількість процесорів, вісь ординат - час виконання :



До 4 ядер ефективність розпаралелення достатньо висока : 100% (4/2.1) (2/4) = 95%.

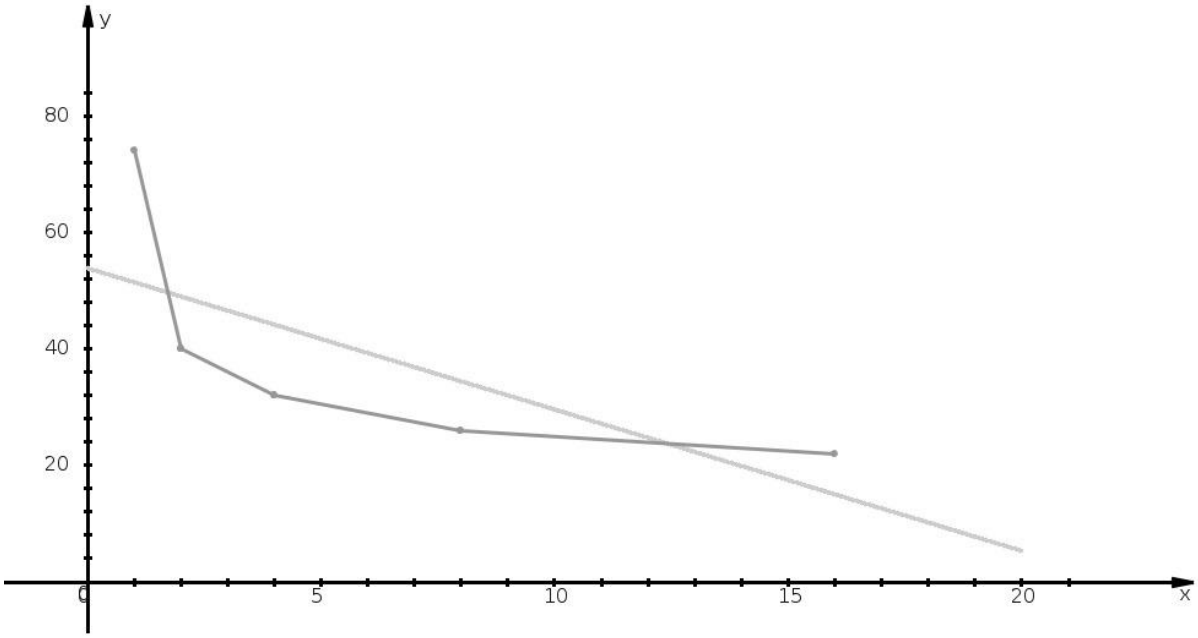
При збільшенні числа ядер, та вже при кількості 16 ядер понижується до величини: 100%(1.3 / 0.8)(8/16)= 81%.

При кількості ядер більшій ніж 32 , прискорення при розпаралелюванні достатньо незначне , в порівнянні з попередніми результатами.

Серія тестів на матриці розміром 256				
Matrix size	Proc numb	Leaf	Exec time	Max mistake
256x256	1	8	82 sec	3E-7
256x256	1	16	81 sec	3E-7
256x256	2	8	45 sec	2.1E-8
256x256	2	16	43 sec	2.1E-8
256x256	4	8	27 sec	1.3E-9
256x256	4	16	27 sec	1.3E-9

256x256	8	8	15 sec	2E-7
256x256	8	16	16 sec	2E-7
256x256	16	8	14 sec	2.4E-7
256x256	16	16	13 sec	2.4E-7

За результатами серії було побудовано наступний графік , де вісь абсцис - кількість процесорів, вісь ординат - час виконання :



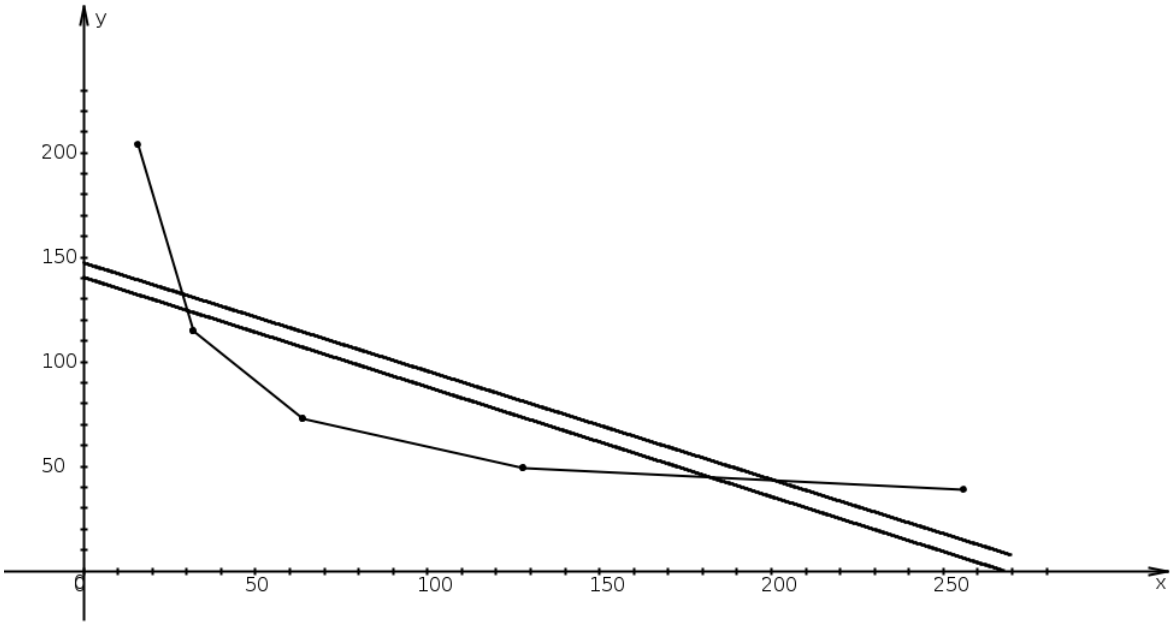
До 8 ядер ефективність розпаралелення достатньо висока : 100% (27/15) (4/8) = 90%.

При збільшенні числа ядер, та вже при кількості 16 ядер понижується до величини: 100% (16/ 13) (8/16) = 61%.

При кількості ядер більшій ніж 32 , прискорення при розпаралелюванні достатньо незначне , в порівнянні з попередніми результатами, як і в минулій серії.

Серія тестів на матриці розміром 512				
Matrix size	Proc numb	Leaf	Exec time	Max mistake
512x512	16	8	204 sec	4.1E-26
512x512	16	16	199 sec	4.1E-26
512x512	16	32	200 sec	4.1E-26
512x512	32	8	115 sec	4.6E-13
512x512	32	16	108 sec	4.5E-13
512x512	32	32	115 sec	4.5E-13
512x512	64	8	73 sec	3.3E-9
512x512	64	16	63 sec	3.3E-9
512x512	64	32	82 sec	3.3E-9
512x512	128	8	49 sec	5.2E-25
512x512	128	16	41 sec	4.8E-25
512x512	128	32	64 sec	4.8E-25
512x512	256	8	39 sec	5E-25
512x512	256	16	31 sec	2.1E-25
512x512	256	32	58 sec	2.1E-25

За результатами серії було побудовано наступний графік , де вісь абсцис - кількість процесорів, вісь ординат - час виконання :



З графіку можна помітити, що , для матриці розміром 512, зі зменшенням кількості процесорів вдвічі, час виконання збільшується приблизно в 1,4 раза. Починаючи з розміру матриць рівною 512 розпаралелення призводить до прискорення обрахунків. До 32 ядер ефективність розпаралелення дуже висока : 100% (199/108) (16/32) = 92%.

При збільшенні числа ядер, та вже при кількості 128 ядер понижується до величини: 100%(63 / 41)(64/128)= 77%

Тобто, матриці, розміром від 512, уже чітко виконуються швидше зі збільшенням кількості процесорів (не більше , ніж розмір матриці), що не сможна сказати про матриці з меншими розмірами.

Виходячи з результатів тестів, можна стверджувати, що немає необхідності використовувати кількість процесорів , що дорівнює розміру вхідної матриці, оскільки, час виконання на менших кількостях процесорів кращий. Також, можна відмітити, що, беручи кількість процесорів рівну половині розміру вхідної матриці, ми отримаємо один з найшвидших результатів часу виконання, але не завжди вигідний , беручи до уваги відношення часу виконання до кількості процесорів.

З розміром листа переходу на послідовний алгоритм все достатньо стабільно на множині всіх тестів, що проводились впродовж налагодження програми. Можна стверджувати, що розміри листів 8 та 16 є найбільш рекомендованими , про що свідчать результати часу та похибок.

Загалом, якщо підсумувати, отримаємо наступні рекомендації запуску:

Рекомендації до запуску		
Matrix size	Proc numb , p	Leaf , l
128x128	4 <= p <= 32	4<= l <= 32
256x256	4 <= p <= 32	4<= l <= 32
512x512	64 <= p <= 256	4<= l <= 32
1024x1024	128 <= p <= 512	4<= l <= 32
2048x2048	256 <= p <= 1024	4<= l <= 32
4096x4096	512 <= p <= 2048	4<= l <= 32

Висновки

Під час виконання роботи вдалось реалізувати послідовний та паралельний алгоритми розкладу Холецького , готові до запуску на суперкомп'ютері з розподіленою пам'яттю. Паралельність алгоритму забезпечена завдяки пакету OpenMPI, та мові програмування Java, з використанням пакету mathpar.

В результаті налагодження та тестування програми було отримано рекомендації стосовно параметрів запуску, що допоможе користувачам економити час та отримувати, максимально близькі до очікуваних , результати роботи алгоритму.

Серед рекомендацій стосовно точності обчислень можна спростити всі отримані результати до наступної відповідності кількості цифр коефіцієнтів матриці та кількості знаків після коми в елементах матриці розміром 4096 : 3 - 870, 6 - 1550, 12 - 1750, 16 - 1890, 20 - 2000, 24 - 2150. При зменшенні розміру матриці вдвічі, кількість знаків варто теж зменшити вдвічі.

Стосовно кількості процесорів , матриці розміром до 64 рекомендується обраховувати послідовно. Прискорення в розрахунках в матрицях розміром від 64 до 256 відбувається до 32 ядер. Після 32 прискорення не таке значне , в порівнянні з збільшенням кількості процесорів. Прискорення в розрахунках в матрицях розміром від 512 відбувається до кількості ядер, рівній половині розміру матриці. Щодо листа переходу на послідовний алгоритм в матрицях , розміром до 512, рекомендується обирати значення від 4 до 16, та , відповідно , від 8 до 64 в матрицях розміром від 512.

Стосовно часу виконання , існують рекомендації очікувати його збільшення до 8 разів при збільшенні розміру матриці вдвічі. Також , при збільшенні точності обчислень вдвічі, очікується збільшення часу виконання також вдвічі.

Список використаних джерел

1. *Gennadi Malaschonok, Evgeni Ilchenko*. Recursive Algorithms for Sparse Matrices over Domain in Distributed Memory. [Електронний ресурс]. – режим доступу: http://parca.tsutmb.ru/publicationsMalaschonok.html
2. *Евгений Ильченко*. Об одном алгоритме управления параллельными вычислениями с децентрализованным вычислением. [Електронний ресурс]. – режим доступу: https://cyberleninka.ru/article/n/ob-odnom-algoritme-upravleniya-parallelnymi-vychisleniyami-s-detsentralizovannym-upravleniem
3. *Евгений Ильченко*. Об эффективном методе распараллеливания блочных рекурсивных алгоритмов. [Електронний ресурс]. – режим доступу: https://cyberleninka.ru/article/n/ob-effektivnom-metode-rasparallelivaniya-blochnyh-rekursivnyh-algoritmov
4. *Малашонок Г.И., Переславцева О.Н., Рыбаков М.А.* Параллельное программирование на OpenMPI Java с приложением в Math Parner. Часть 1. [Електронний ресурс]. – режим доступу: https://search.rsl.ru/ru/record/01007878356
5. *Малашонок Г.И., Переславцева О.Н., Рыбаков М.А.* Параллельное программирование на OpenMPI Java с приложением в Math Parner. Часть 2. [Електронний ресурс]. – режим доступу: https://search.rsl.ru/ru/record/01007878356
6. *Gennadi Malaschonok., Alla Sidko.* Control of matrix computations on distributed memory. International Conference Polynomial Computer Algebra '2019.