УДК 004.7+519.6
DOI: 10.18523/2617-3808.2020.3.114-120

Малашонок Г. І., Іваськевич А. Я.

СТАТИЧНИЙ БЛОЧНО-РЕКУРСИВНИЙ АЛГОРИТМ ХОЛЕЦЬКОГО ДЛЯ КЛАСТЕРА З РОЗПОДІЛЕНОЮ ПАМ'ЯТТЮ

У цій статті досліджено блочно-рекурсивний паралельний алгоритм розкладання Холецького для суперкомп'ютера з розподіленою пам'яттю. Для зручності масштабування, кількість ядер – це деяка ступінь двійки. Розглянуто стійкість алгоритму до накопичення помилок обчислень і ефективність масштабування статичного блочно-рекурсивного алгоритму. Показано, що для щільних матриць, починаючи з розміру 64, використання подвійної точності (стандартна арифметика з плаваючою точкою і 64-розрядним машинним словом), немає змоги отримувати похибку, меншу ніж 1, навіть у простих випадках. Зі збільшенням розмірів коефіцієнтів похибка тільки зростає, тому використання такої арифметики для матриць ще більшого розміру втрачає сенс. Щоб розв'язати цю проблему, для матриць великого розміру ми використовували арифметику BigDecimal. Вона дає змогу програмно задавати точність, яка вказується як кількість десяткових знаків після коми. Спочатку ми визначаємо необхідну кількість знаків для BigDecimal так, щоб похибка обчислень цієї матриці не перевищувала одиницю, а потім робимо експерименти з різною кількістю ядер для такої матриці. Ми даємо рекомендації для щільних матриць, елементи яких задавались випадково з рівномірним розподілом. Для таких матриць, зважаючи на їхній розмір і кількість десяткових знаків у їхніх елементів, ми рекомендуємо вибір точності для машинної арифметики і кількість ядер для обчислень.

Ключові слова: паралельне програмування, алгоритм Холецького, блочно-рекурсивний алгоритм, факторизація матриць, обчислення на кластері з розподіленою пам'яттю, накопичення похибки.

Вступ

Дві головні задачі, які треба розв'язати при переході від матричних алгоритмів для невеликих матриць до алгоритмів із великими матрицями, – це отримання задовільної похибки і розумного часу рішення. Ми досліджуємо задачу декомпозиції позитивно-визначених симетричних матриць – алгоритм розкладання Холецького. Це матриця A, і відомо, що існує розкладання A = B * B', де B – нижня трикутна матриця, а B' – транспонована до неї матриця. Потрібно знайти ці невідомі співмножники.

Зазначимо, що завдання не має точного розв'язання в раціональних числах, як, наприклад, LU-розкладання, тому необхідно використовувати наближені обчислення, і немає інших підходів, крім обчислень із деякою точністю.

Нам не вдалося знайти систематичні дослідження з накопичення похибки в алгоритмі Холецького, тому ми проводимо такі дослідження в цій роботі. Ми використовуємо тільки щільні випадкові матриці, у яких елементами є цілі числа з фіксованою кількістю двійкових розрядів і рівномірним розподілом. Так формується матриця B. Потім ми обчислюємо матрицю A = B * B' і використовуємо її як вхідну матрицю. При цьому, ми ще обчислюємо найбільший за абсолютною величиною елемент A і вказуємо його також. Наведені нами експерименти допускають просту перевірку.

Головним завданням роботи є дослідження масштабованості статичного блочно-рекурсивного алгоритму Холецького для щільних матриць із використанням стандартного блокового матричного множення [1; 2; 3]. Зазначимо, що можна використовувати алгоритм множення Штрассена, і в цьому випадку можна мати менша кількість операцій. Але таке дослідження виходить за рамки цієї роботи.

1. Блочно-рекурсивний алгоритм розкладання Холецького

Нехай задано позитивно-визначену симетричну матрицю A, і потрібно знайти нижню трикутну матрицю L, таку, що $A = L * L^T$.

Розіб'ємо кожну з цих матриць на чотири блоки:

$$L = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix}, A = \begin{pmatrix} \alpha & \beta \\ \beta^T & \gamma \end{pmatrix}$$

Блоки α, β і γ відомі. Блоки a, b, c потрібно знайти. Перемножимо L і L^T та прирівняємо до A.

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

Випишемо рівності для блоків, які призводять до рекурсивного алгоритму.

(1) $a*a^T = \alpha$ (рекурсивний крок за *a*)

(2) $a*b^T = \beta$; $b^T = a^{-1} * \beta$; $b = (b^T)^T$

(3) $b*b^T + c*c^T = \gamma$; $c*c^T = \gamma - b*b^T$ (рекурсивний крок за *c*)

Ми розширимо алгоритм і будемо додатково знаходити матрицю, зворотну до матриці *L*.

Алгоритм:

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)

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

(c, c_1) = choleskyDecomp (δ)

$z = - c_1 \bullet b \bullet a_1$;

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

2. Схема блочно-рекурсивного паралельного алгоритму розкладання Холецького

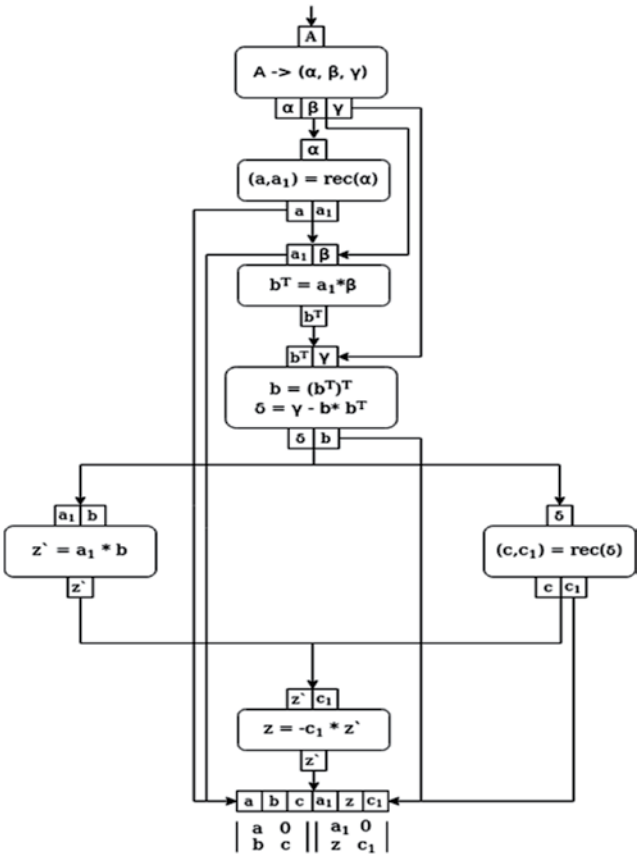

Рис. 1. Схема блочно-рекурсивного паралельного алгоритму розкладання Холецького

На рис. 1 наведено паралельну схему блочно-рекурсивного алгоритму розкладання Холецького. Нехай є обчислювальний кластер, який містить *n* процесорів. На вхід надходить матриця *A* розміру *n*m*. Числа *n* і *m* – це деякі степені двійки. Потрібно організувати обчислювальний процес відповідно до цього алгоритму.

Як видно зі схеми на рис. 1, нам потрібен тільки алгоритм множення матриць, а також його модифікація – множення зі складанням і додатковим транспонуванням, тому ми написали кілька варіантів паралельних процедур для блочного статичного алгоритму множення, залежно від доступної кількості ядер: одного ядра, двох ядер, чотирьох ядер і для випадку $8 * p$ ядер. У разі $8 * p$ ядер, процесори діляться на вісім рівних груп по *p* ядер. На кожну групу з *p* процесорів припадає одне з восьми блокових множень, тому тут рекурсивно застосовується один із чотирьох наведених вище варіантів.

Як видно з рисунку, всі блоки одного рівня рекурсії можуть використовувати однакову кількість ядер, крім передостаннього етапу. На цьому етапі всі доступні ядра діляться на дві рівні частини. Одна половина використовується у матричному множенні, а друга – в рекурсивному алгоритмі Холецького.

Головний принцип паралельного програмування – це принцип великозернистого розпаралелювання, тому ми використовуємо термін «розмір листа» для зазначення найменшого розміру блока, для якого застосування розпаралелювання є корисним. Такий розмір підбирається кожного разу емпірично, відповідно до фізичних особливостей кластера. Якщо розмір блока матриці менший, ніж листовий, то обчислення виконують на одному ядрі.

3. Головні програмні компоненти

Програму написано на мові Java з допомогою бібліотеки OpenMPI [ref] і пакета матричних процедур проекту MathPartner [ref]. Можна виділити такі основні методи:

- MatrixMul.multiplyMatrix;
- MatrixMul.multiplyMatrixWithAdd;
- CholeskyDecomposition.choleskyDecomposition.

Кожен із методів на вході отримує одну або кілька квадратних матриць типу MatrixS, комутатор для транспорту типу Intracomm і параметр Ring, який дає змогу визначати формат – кількість десяткових знаків після коми, та задавати алгебраїчний простір, надаючи вибір чисел типу Double, BigDecimal тощо.

Щоб отримати результат паралельного алгоритму Холецького, досить викликати метод choleskyDecomposition (MatrixS A, Ring ring, Intracomm intracomm, int leaf) із переданням йому відповідної матриці для розкладання і параметрів для коректного результату, згідно з рекомендаціями.

4. Етапи виконання паралельного алгоритму

Розглянемо етапи виконання паралельного алгоритму (див. рис. 1). Якщо кількість отриманих на вході процесорів дорівнює 1, то алгоритм виконується послідовно. В іншому разі першим кроком алгоритму буде розбиття вхідної матриці на чотири рівні блоки, розміром $m/2 * m/2 : A \rightarrow (\alpha, \beta, \gamma)$.

Наступним кроком буде рекурсивне виконання цього алгоритму для блока α: (a, a_1) = choleskyDecomposition(α) на n процесорах. Результатом буде пара матриць (a, a_1), де a – шукана нижньотрикутна матриця, a_1 – матриця, обернена до матриці a.

Наступними двома кроками будуть паралельне множення матриць mutrixMul $(a_1, \beta) \rightarrow b^T$ і паралельне множення матриць із додаванням mutrixMulWithAdd $(-b^T, b, \gamma) \rightarrow \delta$. Кількість процесорів на кожному з цих кроків дорівнює n.

На цьому етапі, отримавши два незалежні послідовні кроки алгоритму, всі процесори діляться на дві групи по $n/2$ у кожній, перша з яких виконує розкладання Холецького choleskyDecomposition(δ) $\rightarrow (c, c_1)$, а друга – множення матриць mutrixMul$(a_1, b) \rightarrow z`$.

Останнім кроком обчислень буде ще одне множення матриць mutrixMul $(-c_1, z`) \rightarrow z$ на n процесорах.

На виході алгоритму, шляхом об'єднання блоків, отримаємо матриці $L = ((a, 0), (b, c))$ і $L^{inv} = ((a_1, 0), (z, c_1))$, де L – результуюча нижньотрикутна матриця, а L^{inv} – обернена до неї.

5. Накопичення похибки в алгоритмі Холецького

Ми провели експерименти, в яких оцінили похибку обчислень в алгоритмі Холецького. Для цього ми використали щільні випадкові матриці, елементами яких є цілі числа з фіксованою кількістю двійкових розрядів і рівномірним розподілом. Так формується матриця B.

За матрицею B ми обчислюємо матрицю $A = B * B'$ і використовуємо її як вхідну матрицю. При цьому, ми ще обчислюємо найбільший за абсолютною величиною елемент A і вказуємо його також. Оскільки матриця B відома заздалегідь, то, віднімаючи від неї знайдену за алгоритмом матрицю, ми знаходимо матрицю помилок. Знаходимо найбільший за абсолютною величиною елемент цієї матриці і наводимо його в таблиці.

Експерименти проводяться у форматі чисел BigDecimal. У цьому форматі можна встановлювати кількість десяткових знаків після коми, за що відповідає параметр Accuracy, який вказаний у всіх таблицях нижче.

У табл. 1 наведено дві серії експериментів: на одному ядрі й на чотирьох ядрах. Максимальна похибка в обчисленні залежить від розміру матриці і є приблизно однаковою в кожній серії.

Крім похибки, тут ще наведено час обчислень. Із результатів у таблиці видно, що, починаючи з матриць розміром 64×64, паралельний алгоритм, за часом виконання, працює швидше, ніж послідовний.

Таблиця 1

Результати порівняльного тестування послідовної та паралельної версій алгоритму

Matrix size	Bits in B	Accuracy	Exec time		Max mistake	Max number in matrix A
			Sequential (1 proc)	Parallel (4 proc)		
8×8	8	300	0.69 sec	1 sec	8.045E-297	133000
16×16	8	300	1 sec	2 sec	3.90E-295	377000
32×32	7	300	4 sec	7 sec	1.50E-287	206000
64×64	7	300	17 sec	13 sec	4.65E-272	373000
128×128	7	300	151 sec	63 sec	8.61E-249	736000
256×256	6	300	1001 sec	477 sec	5.31E-198	383000

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

Тестування програм проводили на матрицях різних розмірів і з різними елементами за кількістю десяткових цифр у коефіцієнтах. Було вирішено провести три серії тестів із різною кількістю десяткових цифр у коефіцієнтах в елементах матриці, а саме: 3, 6 і 12. Було застосовано генератор випадкових чисел із рівномірною щільністю заповнення, що дало змогу отримати щільно заповнену вхідну матрицю.

Результати серій наведено в табл. 2.

Таблиця 2

Результати тестування алгоритму з різною кількістю десяткових цифр у коефіцієнтах матриці

		Кількість десяткових цифр у коефіцієнтів матриці		
		3	6	12
Matrix size	**Accuracy**	**Max mistake**		
16×16	230	1E-224	2.2E-225	1.10E-219
32×32	230	1.2E-225	6.114E-222	4.40E-210
64×64	230	5.8E-216	1.82E-197	1.31E-192
128×128	230	6.7E-198	6.49E-177	8.4E-173
256×256	230	2.6E-171	5.50E-136	5.10E-117
512×512	230	6.9E-116	1.18E-34	1.61E-7

Як видно з табл. 2, накопичення похибки, яка вимірюється у кількості десяткових знаків, зростає лінійно зі зростанням розміру матриці. Цей факт можна використовувати для екстраполяції. Продовжуючи відповідні прямі, можна знайти, скільки необхідно ставити десяткових знаків після коми в числах типу BigInteger, щоб похибка результату не перевищувала 1. Отримані таким способом оцінки для різних розмірів матриць і різних розмірів чисел наведено в табл. 3.

Таблиця 3

Рекомендації для вибору точності обчислень в алгоритмі Холецького

	Кількість десяткових цифр у коефіцієнтах матриці					
	3	6	12	16	20	24
Matrix size	*Accuracy*					
8×8	2	5	5	5	5	5
16×16	5	10	10	10	10	10
32×32	10	20	20	20	20	20
64×64	20	30	30	30	40	40
128×128	30	50	60	60	70	70
256×256	60	90	110	120	130	140
512×512	110	170	220	240	250	270
1024×1024	220	340	440	480	500	540
2048×2048	440	780	870	950	1000	1080
4096×4096	870	1550	1750	1890	2000	2150

Зазначимо, що перші три рядки належать до матриць розміру 8, 16 і 32. Для таких матриць можна застосовувати числа типу Double. Але, починаючи з четвертого рядка, тобто для матриць порядку 64 і вище, ми рекомендуємо користуватися числами типу BigDecimal і наводимо в таблиці значення параметра точності – Accuracy.

Оскільки доступні нам публікації про алгоритм Холецького не звертають увагу на проблему накопичення похибки, то проведені нами експерименти й отримані висновки мають спонукати тих, хто застосовує цей алгоритм або тільки має намір його застосовувати, зважити на цю проблему.

5.2. Рекомендації для користувачів щодо кількості процесорів та розміру блока для переходу на послідовний алгоритм, на основі аналізу результатів експериментів на кластері

У таблицях 4–6 зібрано результати тестів. Експерименти проводили на кластері MVS-10P Joint Supercomputer Center the Russian Academy of Sciences {5}. Параметр Leaf визначає розмір блока матриці, за якого рекурсивний процес розпаралелювання зупиняється і відбувається обрахунок результату на одному ядрі. Рядки з найкращим часом виконання виділено напівжирним шрифтом. Кількість десяткових цифр у коефіцієнтах матриць у всіх експериментах дорівнює 3.

Таблиця 4

Серія тестів для матриці розміром 128

Matrix size	Proc numb	Accuracy	Leaf	Exec time	Max mistake
128	1	30	8	10 sec	1.3E-10
128	**1**	30	**16**	**9 sec**	**1.3E-10**
128	2	30	8	4 sec	1.1E-8
128	**2**	30	**16**	**4 sec**	**1.1E-8**
128	4	30	8	2.1 sec	3.8E-9
128	**4**	30	**16**	**2.1 sec**	**3.8E-9**
128	**8**	30	**8**	**1.2 sec**	**4E-8**
128	8	30	16	1.3 sec	4E-8
128	16	30	8	0.9 sec	7.8E-7
128	**16**	30	**16**	**0.8 sec**	**8.1E-7**

До чотирьох ядер ефективність розпаралелення є досить високою. При збільшенні кількості ядер знижується; за кількості ядер більшій, ніж 32, прискорення при розпаралелюванні доволі незначне.

Таблиця 5

Серія тестів для матриці розміром 256

Matrix size	Proc numb	Accu-racy	Leaf	Exec time	Max mistake
256×256	1	60	8	82 sec	3E-7
256×256	**1**	60	**16**	**81 sec**	**3E-7**
256×256	2	60	8	45 sec	2.1E-8
256×256	**2**	60	**16**	**43 sec**	**2.1E-8**
256×256	4	60	8	27 sec	1.3E-9
256×256	**4**	60	**16**	**27 sec**	**1.3E-9**
256×256	8	60	8	15 sec	2E-7
256×256	**8**	60	**16**	**16 sec**	**2E-7**
256×256	16	60	8	14 sec	2.4E-7
256×256	**16**	60	**16**	**13 sec**	**2.4E-7**

До восьми ядер ефективність розпаралелення є досить високою. За кількості ядер більшій, ніж 32, прискорення при розпаралелюванні доволі незначне, порівняно з попередніми результатами, як і в минулій серії.

Таблиця 6

Серія тестів для матриці розміром 512

Matrix size	Proc numb	Accu-racy	Leaf	Exec time	Max mistake
512×512	16	120	8	204 sec	4.1E-26
512×512	**16**	120	**16**	**199 sec**	**4.1E-26**
512×512	16	120	32	200 sec	4.1E-26
512×512	32	120	8	115 sec	4.6E-13
512×512	**32**	120	**16**	**108 sec**	**4.5E-13**
512×512	32	120	32	115 sec	4.5E-13
512×512	64	120	8	73 sec	3.3E-9
512×512	**64**	120	**16**	**63 sec**	**3.3E-9**
512×512	64	120	32	82 sec	3.3E-9
512×512	128	120	8	49 sec	5.2E-25
512×512	**128**	120	**16**	**41 sec**	**4.8E-25**
512×512	128	120	32	64 sec	4.8E-25
512×512	256	120	8	39 sec	5E-25
512×512	**256**	120	**16**	**31 sec**	**2.1E-25**
512×512	256	120	32	58 sec	2.1E-25

За результатами таблиць 4–6 ми будуємо графіки залежності часу виконання від кількості ядер на кластері, що використовувались в експерименті (див. рисунки 2 і 3).

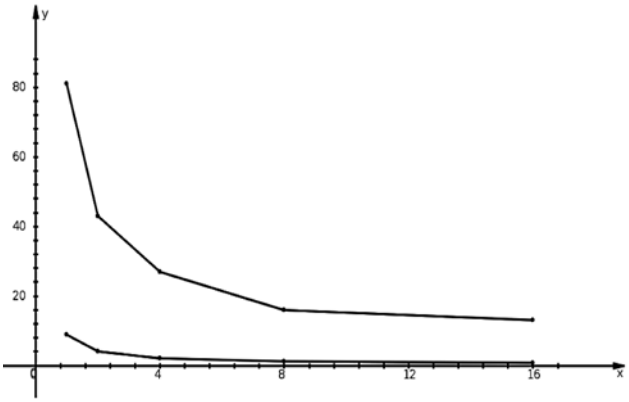

Рис. 2. Графіки залежності часу виконання від кількості ядер у діапазоні 1–16

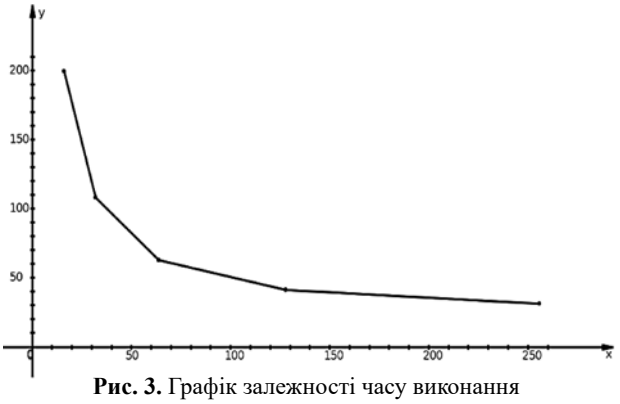

Рис. 3. Графік залежності часу виконання від кількості ядер у діапазоні 16–256

Для оцінювання масштабованості обчислень ми будуємо відповідні графіки для ефективності. Ефективність вважається рівною 100 %, якщо при збільшенні кількості ядер у n разів час виконання зменшується у n разів. Ефективність обчислень на n ядрах щодо до обчислень на k ядрах обчислюють за формулою:

$$Eff_k = (N_k * T_k / N_n * T_n) * 100\ \% .$$

За $k = 1$ отримаємо ефективність щодо однопроцесорного алгоритму.

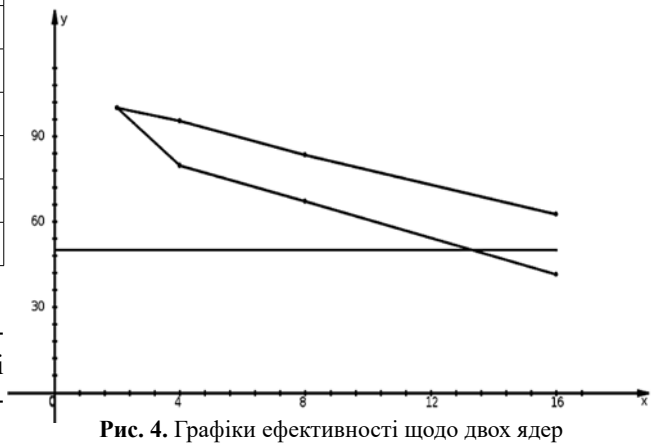

Рис. 4. Графіки ефективності щодо двох ядер

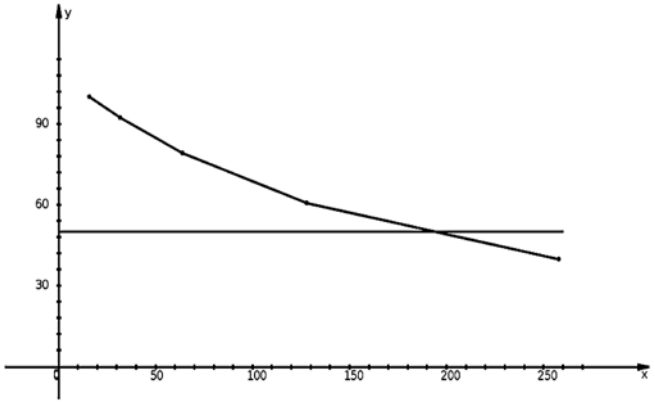

Рис. 5. Графік ефективності щодо 16 ядер

На рисунках 4–5 побудовано графіки ефективності. Перші два графіки – ефективність щодо двох ядер, а третій графік – ефективність щодо 16 ядер.

Прийнято вважати, що хорошої масштабованості можна досягти, якщо ефективність вища ніж 50 %, тому горизонтальною лінією ми позначаємо діапазон хорошої масштабованості.

Починаючи з розміру матриць, що дорівнює 512, розпаралелення призводить до прискорення обрахунків. До 32 ядер ефективність розпаралелення дуже висока – 92 %.

При збільшенні числа ядер уже за кількості 128 ядер ефективність знижується до 77 %.

Із розміром блока Leaf, за якого відбувається перехід на послідовний алгоритм, все досить стабільно на множині всіх тестів, що проводились упродовж налагодження програми. Можна стверджувати, що розміри листів 8 і 16 є найкращими.

Загалом, підсумовуючи, можна надати рекомендації для користувачів (табл. 7).

Таблиця 7

Рекомендації щодо вибору числа ядер і розміру листа

Matrix size	Accuracy	Maxn num	Proc numb	Leaf
128×128	60	756830	32	16
256×256	120	388676	64	8
512×512	240	711258	128	4
1024×1024	480	843451	256	4
2048×2048	960	641113	512	4
4096×4096	1920	789905	1024	4

Висновки

Ми надали опис паралельного блочно-рекурсивного варіанта алгоритму Холецького і експериментів із паралельної програми для щільних матриць. Усі експерименти проводили на кластері з розподіленою пам'яттю. Кількість ядер і розмір матриць обирали зі степенів числа 2. У таблицях зазначено рекомендовані для використання кількість ядер кластера залежно від розміру матриці.

Ми експериментально досліджували, який розмір має бути у блока матриці, щоб перейти до виконання обчислення на одному ядрі, замість розпаралелювання на кілька ядер.

Для всіх експериментів ми брали щільні випадкові матриці з рівномірним розподілом і з певною кількістю десяткових знаків у елементів матриц. Ми навели детальний опис алгоритму, тому експерименти легко можна повторити незалежно.

Окремий інтерес становлять результати, які отримані при дослідженні стійкості алгоритму Холецького. Ми показали, що для випадкових матриць із рівномірним розподілом похибка обчислень накопичується дуже швидко, тому треба відмовитися від формату чисел Double і використовувати формат чисел, для яких точність встановлюється програмно. Для мови Java це формат BigDecimal, без використання якого обчислення з матрицями великого розміру втрачають сенс через втрату точності.

Слушне твердження про залежність максимальної похибки від розміру вхідних даних було підтверджено результатами тестування, на основі яких ми отримали рекомендації для користувачів. Для матриці розміром 4096 рекомендовано таку відповідність кількості розрядів у елементів та кількості знаків дробової частини числа: 3 → 870, 6 → 1550, 12 → 1750, 16 → 1890, 20 → 2000, 24 → 2150. Для матриць меншого розміру при зменшенні розміру матриці вдвічі кількість знаків варто також зменшити вдвічі (наприклад, для матриці розміром 2048 отримаємо: 3 → 435, 6 → 775, 12 → 875, ...).

Алгоритм засвідчив хорошу масштабованість.

Слова подяки

Ми висловлюємо подяку Joint Supercomputer Center of the Russian Academy of Sciences і співробітникам лабораторії алгебраїчних обчислень Тамбовського ДУ за організацію обчислювальних експериментів на суперкомп'ютері. Дякуємо

за підтримку проекту «Створення е-платформи відкритих даних для центрів колективного користування приладами НАН України» Київського академічного університету.

Список літератури

1. Ильченко Е. А. Об эффективном методе распараллеливания блочных рекурсивных алгоритмов / Е. А. Ильченко // Вестник Тамбовского университета. Серия : Естественные и технические науки. – 2015. – Т. 20, вып. 5. – С. 1173–1186.
2. Малашонок Г. И. Организация параллельных вычислений в рекурсивных символьно-численных алгоритмах / Г. И. Малашонок, Ю. Д. Валеев // Труды конференции ПаВТ'2008 (Санкт-Петербург). – Челябинск : Изд-во ЮУрГУ, 2008. – С. 153–165.
3. Малашонок Г. И. Управление параллельным вычислительным процессом / Г. И. Малашонок // Вестник Тамбовского университета. Серия: Естественные и технические науки. – 2009. – Т. 14, вып. 1. – С. 269–274.

References

Ilchenko, E. A. (2015). Ob effektivnom metode rasparallelivania blochnykh rekurivnykh algoritmov. *Vestnik Tambovskogo universiteta. Ser. Estestvennye i tehnichekie nauki*, *20* (5), 1173–1186 [in Russian].

Malashonok, G. I. (2009). Upravlenie parallelnym vychislitelnym processom. *Vestnik Tambovskogo universiteta. Ser. Estestvennye i tehnichekie nauki*, *14* (1), 269–274 [in Russian].

Malashonok, G. I., & Valeev, Ju. D. (2008). Organizacia parallelnykh vychislenii v rekursivnykh simvolno-chislennykh algoritmakh. In *Trudy konferencii PaVT'2008 (Sankt-Peterburg)* (pp. 153–165). Cheliabinsk: JuUrGU [in Russian].

G. Malaschonok, A. Ivaskevych

STATIC BLOCK-RECURSIVE CHOLESKY ALGORITHM FOR A DISTRIBUTED MEMORY CLUSTER

This paper investigates the block-recursive parallel algorithm for Cholesky decomposition for a supercomputer with distributed memory. For ease of scaling, the number of cores is a degree of the number two. We investigate the resistance of the algorithm to the accumulation of computational errors and the scaling efficiency of the static block-recursive algorithm.

Note that the problem does not have an exact solution in rational numbers (for example, LU-decomposition has an exact solution), so it is necessary to use approximate calculations and there are no other approaches than calculations with some accuracy. We have not been able to find systematic studies on the accumulation of error in the Cholesky algorithm, so we conduct such studies in this paper. elements are integers with a fixed number of binary digits and a uniform distribution.

We have shown that for dense matrices, starting with matrix size 64, the use of double precision (standard floating-point arithmetic and 64-bit machine word) does not allow to obtain an error less than 1, even in simple cases. As the size of the coefficients increases, the error only increases, so the use of such arithmetic for matrices of even larger size loses its meaning. To solve this problem, we used BigDecimal arithmetic for large matrices. It allows you to programmatically specify the accuracy, which is specified as the number of decimal places. First, we determine the required number of characters for BigDecimal so that the calculation error of this matrix does not exceed one, and then we experiment with different numbers of cores for such a matrix. We give recommendations for dense matrices, the elements of which were given randomly with a uniform distribution. For such matrices, based on their size and the number of decimal places in their elements, we recommend the choice of accuracy for machine arithmetic and the number of cores for calculations.

Keywords: parallel programming, Cholesky algorithm, block-recursive algorithm, matrix factorization, computation on a cluster with distributed memory, error accumulation.

Матеріал надійшов 09.06.2020