Теорія оптимальних рішень

УДК 519.6

О.М. ХІМІЧ, О.В. ПОПОВ, А.Ю. БАРАНОВ, О.В. ЧИСТЯКОВ

ГІБРИДНИЙ АЛГОРИТМ РОЗВ'ЯЗУВАННЯ ЗАДАЧ НА ВЛАСНІ ЗНАЧЕННЯ ДЛЯ СТРІЧКОВИХ МАТРИЦЬ

Запропоновано алгоритм розв'язування часткової узагальненої алгебраїчної проблеми власних значень зі стрічковими симетричними матрицями на комп'ютерах гібридної архітектури. Приведено результати апробації алгоритму на персональному суперкомп'ютері гібридної архітектури Інпарком_рд.

Вступ. Велика кількість наукових та практичних задач, зокрема при дослідженні стійкості конструкцій, розрахунку динаміки напружено-деформованого стану об'єктів різної природи та інші, зводяться до розв'язання часткової узагальненої алгебраїчної проблеми власних значень з симетричними стрічковими додатно-означеними матрицями великих розмірів.

Важливою особливістю таких задач лінійної алгебри, які виникають при дискретизації, є те, що кількість ненульових елементів матриць складає kn, де $k << n$, а n – порядок матриці, тобто матриці є розрідженими [1, 2]. Структура розрідженої матриці визначається нумерацією невідомих задачі і часто є стрічковою, блочно-діагональною з обрамленням, профільною і тому подібне. В даній статті розглядаються симетричні додатно-означені стрічкові матриці.

Для розв'язування алгебраїчної проблеми власних значень зі стрічковою структурою матриць великих розмірів виникає необхідність створення нових алгоритмів, які забезпечують підвищення ефективності обчислень на комп'ютерах гібридної архітектури, тобто багатоядерних процесорах (CPU) з графічними прискорювачами (GPU).

Постановка задачі. Алгебраїчна проблема власних значень (АПВЗ) полягає у знаходженні таких чисел λ, при яких існують відмінні від нульового розв'язки системи лінійних алгебраїчних рівнянь (СЛАР):

$$Ax = \lambda Bx, \quad A, B \in Mn \times n, \quad x \in Cn, \quad \lambda \in C, \tag{1}$$

де $Mn \times n$ – множина квадратних матриць порядку n. Числа λ називаються власними значеннями задачі (1), а вектори x – власними векторами цієї задачі. Задача (1) називається узагальненою проблемою власних значень.

Якщо B – одинична матриця n-го порядку (тобто $B \equiv In$), то задача

$$Ax = \lambda x \tag{2}$$

називається стандартною проблемою власних значень, а числа λ та вектори x називаються відповідно власними значеннями та власними векторами матриці A.

Можуть розглядатися такі задачі на власні значення:

- повна проблема власних значень – знайти всі власні значення і всі власні вектори;
- часткова проблема власних значень – знайти одне або декілька власних значень та відповідних їм векторів або лише власні значення (всі або декілька).

Тут розглядається гібридний алгоритм методу ітерацій на підпросторі розв'язування часткової узагальненої проблеми власних значень (знаходження кількох найменших власних значень та відповідних власних векторів) для стрічкових симетричних додатно-означених матриць A та B. Також вважатимемо, що напівширина стрічки матриці A та матриці B рівні.

Гібридний алгоритм методу ітерацій на підпросторі. Метод ітерацій на підпросторі є узагальненням методу обернених ітерацій і полягає в побудові послідовності підпросторів E_t (t = 1, 2, ...), яка збігається до підпростору E_∞, що містить шукані власні вектори.

В методі ітерацій на підпросторі на t-й ітерації обчислюється ортонормований базис підпростору E_t та, якщо досягнута збіжність, визначаються шукані власні пари [1, 3].

Декомпозиція стрічкових симетричних матриць. Ефективність методів розв'язування великих АПВЗ (використання оперативної пам'яті та часу розв'язування) зі стрічковими матрицями у великій мірі залежить від ефективних способів зберігання, використання та обробки стрічкових матриць [4, 5]. Розподіл між процесами багатоядерного комп'ютера матриць A та B такий: елементи (головної діагоналі та піддіагональні) стрічкових симетричних матриць задачі повинні бути розподілені між процесами відповідно одномірної блочно-циклічної схеми, за якою процес з логічним номером r оперує з елементами рядків матриці з номерами $sI + 1,\ldots, sI + s$, а процес з номером $r + 1$ – з елементами рядків з номерами $s(I + 1) + 1,\ldots, s(I + 1) + s$.

Гібридний алгоритм методу ітерацій на підпросторі. Розв'язування АПВЗ зі стрічковими симетричними матрицями гібридним алгоритмом, що розглядається, зводиться до виконання для t = 1, 2, … таких підзадач [1, 3]:

- розв'язування системи рівнянь з матрицею A (розв'язується кожним процесом, використовуючи попередньо факторизовану матрицю):

$$AX_t = Y_{t-1}; \tag{3}$$

- обчислення проекції матриці A на підпростір E_t (виконується на GPU):

$$A_t = X_t^T Y_{t-1} \equiv X_t^T A X_t; \quad (4)$$

- обчислення прямокутної матриці (виконується з використанням GPU):

$$W_t = BX_t; \quad (5)$$

- обчислення проекції матриці B на підпростір E_t (виконується з використанням GPU):

$$B_t = X_t^T W_t \equiv X_t^T B X_t; \quad (6)$$

- розв'язування проблеми власних значень для проекцій (розв'язується кожним процесом незалежно):

$$A_t Z_t = B_t Z_t \Lambda_t; \quad (7)$$

- обчислення наступного наближення (за рахунок розподіленості даних між процесами, операції виконуються паралельно на CPU):

$$Y_t = W_t Z_t; \quad (8)$$

- перевірка умов закінчення ітераційного процесу:

$$\frac{\left|\lambda_i^{(t)} - \lambda_i^{(t-1)}\right|}{\lambda_i^{(t)}} \le \varepsilon \quad (i = 1, 2, \dots, r). \quad (9)$$

Якщо умови (9) виконуються після t ітерацій, то наближенням розв'язку задач (1) або (2) є:

$$\lambda_i^* = \lambda_i^{(t+1)}, \qquad X^* = X_{t+1} Z_{t+1} \quad (i = 1, 2, \dots, r).$$

Тут, як і в (9), мається на увазі, що власні значення упорядковані за зростанням $0 < \lambda_1 \le \lambda_2 \le \dots \le \lambda_r \le \dots$.

Як відомо (див., наприклад [1]), ітераційний процес збігається лінійно, причому швидкість збіжності λ_i визначається відношенням $\lambda_i / \lambda_q + 1$, де q – розмір підпростору E_t, що ітерується. Тому, чим більше значення q, тим за меншу кількість ітерацій буде досягнута точність наближень, але при цьому збільшується обсяг обчислень на кожній ітерації. З практики [1] рекомендується вибирати $q = \min(2r, r+8)$.

Результатом роботи гібридного алгоритму ітерацій на підпросторі є:

- обчислені власні значення λ_i, розташовані у кожному процесі в порядку зростання;
- матриця власних векторів, відповідних обчисленим власним значенням (елементи цієї матриці розподілені між процесами відповідно до одномірної блочно-циклічної схеми, яка співпадає зі схемою розподілу елементів матриць A та B).

Слід відзначити, що в підзадачах (3) та (5) беруть участь матриці вихідної задачі (1). Обсяг обчислень в кожній з цих підзадач на порядок або більше перевищує обсяг обчислень у інших підзадачах. Отже, ефективність гібридного алгоритму методу ітерацій на підпросторі визначається ефективним виконанням на гібридному комп'ютері підзадач (3) та (5). На ефективність гібридних алгоритмів лінійної алгебри у значній мірі впливає розподіл матриць між ядрами CPU та графічними процесорами, а також виконання обмінів даними між ними. Розглянемо детально як вирішуються ці проблеми в гібридному плитковому

алгоритмі розв'язування підзадачі (3) на основі LL^T-розвинення (факторизація методом Холецького) для комп'ютерів з одним та декількома графічними процесорами [6, 7].

Гібридний плитковий алгоритм LL^T-розвинення. Розв'язуємо систему лінійних рівнянь:

$$Ax = b, \tag{10}$$

де матриця A – стрічкова додатно-означена, n – порядок матриці, m – напівширина стрічки матриці.

Розв'язання системи (10) полягає в розв'язанні підзадач: трикутне розвинення матриці системи:

$$A = L \times L^T, \tag{11}$$

та розв'язання двох СЛАР з трикутними матрицями:

$$Ly = b, \tag{12}$$

$$L^T x = y. \tag{13}$$

Розіб'ємо стрічкову матрицю A на вертикальні прямокутні плитки [6]. Оскільки матриця симетрична, то доцільно зберігати тільки нижній її трикутник, що дає можливість оптимізувати обсяг пам'яті, який необхідний для реалізації алгоритму, а також зменшити кількість виконуваних арифметичних операцій. Крім того, таке розбиття даних дає можливість більш ефективно використовувати кеші CPU та GPU, а також високоефективні функції cuBLAS для виконання матрично-векторних операцій. Іншою важливою особливістю плиткового розбиття даних є те, що на кожному кроці блочного алгоритму Холецького потрібно мати в пам'яті тільки m/w плиток, де w – ширина плитки, які будуть розташовані послідовно за плиткою з провідним блоком. Таким чином, така декомпозиція даних дає можливість на кожному кроці алгоритму копіювати в пам'ять GPU лише одну додаткову плитку.

Введемо деякі позначення, які будуть потрібні для опису алгоритму трикутного розвинення (схематично показано на рисунку):

- Pb_i – діагональний блок плитки з номером i;
- $Pu_{i,j}$ – підматриця плитки i, що починається з рядка $j \times w$;
- $Ps_{i,j}$ – квадратна підматриця плитки i, що починається з рядка $j \times w$.

Гібридний плитковий алгоритм LL^T-розвинення для гібридних комп'ютерів з одним GPU. При реалізації цього алгоритму використовуються три основні операції:

1) факторизація щільної симетричної додатно-означеної матриці. В даному алгоритмі ця операція буде використовуватися для факторизації діагонального блоку плитки Pb_i;

2) розв'язування СЛАР з трикутною матрицею. Використовується для розв'язування рівняння $X \times Pb_i = Pu_{i,1}$;

3) обчислення добутку двох матриць. Використовується для оновлення плиток матриці системи, які розташовані за плиткою з провідним блоком:

$$Pu_{i+j,0} \leftarrow Pu_{i+j,0} - Pu_{i,j} \times (Ps_{i,j})^T. \tag{14}$$

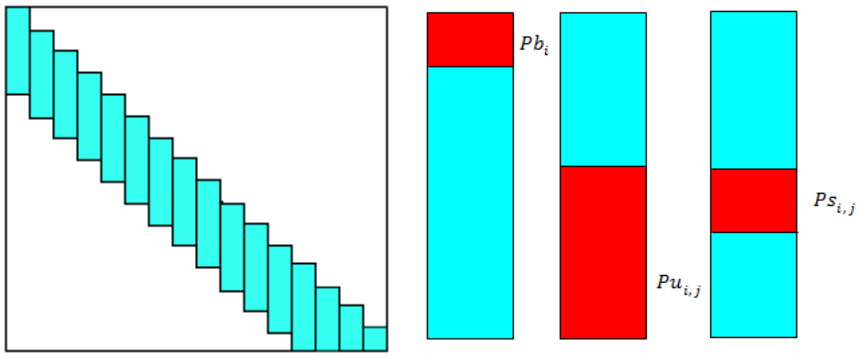

РИСУНОК. Декомпозиція даних

Для виконання перших двох операцій, на i-му кроці алгоритму, потрібно використовувати лише дані, які знаходяться в i-й плитці. Для операції добутку двох матриць, окрім i-ї плитки, також використовуються плитки, які розташовуються послідовно за i-ю плиткою. Отже, кількість плиток, які використовуються на i-му кроці алгоритму дорівнює m/w. На кожному наступному кроці в пам'ять GPU потрібно копіювати одну додаткову плитку.

Доцільно виконувати копіювання плитки, яка буде потрібна на наступному кроці, одночасно з виконанням операції добутку двох матриць. Таким чином, можна досягти більшої ефективності алгоритму. Для того, щоб на кожному кроці не виділяти додаткову пам'ять для наступної плитки, потрібно на початку роботи алгоритму виділити пам'ять для зберігання $m/w+1$ плитки. На кожному кроці m/w плиток будуть використовуватися для обчислень, а наступна плитка буде копіюватися у вільну ділянку пам'яті. Це дає можливість значно зменшити використання пам'яті GPU у порівнянні з алгоритмами, коли вся матриця зберігається на GPU.

Програмну реалізацію гібридного алгоритму розіб'ємо на два етапи: на першому (підготовчому) етапі виконується початкова ініціалізація, на другому – обчислюється трикутне розвинення матриці системи.

Підготовчий етап:

1) ініціалізація cuBLAS;

2) створення двох екземплярів cudaStream_t: cudaCopyStream, cudaCalculateStream;

3) потік cudaCalculateStream використовується як основний потік cuBLAS;

4) виділення пам'яті для $m/w+1$ плитки на GPU. Вказівник на цю пам'ять gpuTilesMemory;

5) копіювання перших m/w плиток в пам'ять GPU.

Варто зазначити, що доцільним є виділення однієї ділянки пам'яті для збереження $m/w+1$ плиток в GPU, що дає можливість швидше виділити всю необхідну пам'ять.

На другому етапі, для кожного $i = 1,\ldots, n/w$ виконуємо наступні кроки алгоритму:

1) факторизація діагонального блоку i-ї плитки Pb_i. Результат записується в Pb_i;

2) перетворення на GPU нижньої частини плитки $Pu_{i,1}$ за формулою $Pu_{i,1} \leftarrow Pu_{i,1} * (Pb_i)^{-1}$. Використовується функція cublasDtrsm;

3) асинхронне копіювання у потоці cudaCopyStream *i*-ї плитки в пам'ять CPU;

4) асинхронне копіювання плитки, котра буде потрібна на наступному кроці в gpuTilesMemory, використовуючи cudaCopyStream;

5) виконати цикл по j від 1 доки виконується нерівність $j \times w < len[i]$; на кожній ітерації виконати перетворення (14), використовуючи функцію cublasDgemm;

6) синхронізація потоку cudaCopyStream та збереження i-ї плитки на CPU;

7) синхронізація основного потоку cuBLAS – cudaCalculateStream.

Після виконання всіх кроків алгоритму отримаємо нижню трикутну матрицю, для якої справджується формула (6) другого етапу.

Реалізація гібридного плиткового алгоритму LL^T-розвинення для гібридних комп'ютерів з декількома GPU. Нехай кількість доступних GPU для реалізації алгоритму дорівнює numGPU. В цьому випадку i-ту плитку матриці буде обробляти GPU з номером $i \bmod \mathrm{numGPU}$ (нумерація GPU з 0). Нові версії CUDA дають можливість використовувати один CPU для роботи з усіма доступними GPU. Оскільки основна частина обчислень виконується на GPU, то будемо використовувати один CPU потік.

Як і в попередньому випадку, розіб'ємо реалізацію алгоритму на два етапи. На підготовчому етапі виконуються такі операції:

1) ініціалізація cuBLAS для кожного GPU;

2) створення екземплярів cudaStream_t для кожного GPU, а також їх запис в масиви: cudaCopyStreamArr[], cudaCalculateStreamArr[];

3) встановлення основних потоків для доступних GPU;

4) виділення пам'яті для $\dfrac{n}{w \times \mathrm{numGPU}} + 1$ плиток у кожному GPU;

5) копіювання перших m/w плиток у пам'ять відповідних GPU.

На другому етапі для кожного $i = 1 \ldots n/w$ продовжуємо виконувати такі кроки алгоритму:

1) виклик функції cudaSetDevice($i \bmod numGPU$). Таким чином поточним GPU стає той, в якому зберігається *i*-та плитка;

2) перетворення на GPU нижньої частини плитки $Pu_{i,1}$ за формулою $Pu_{i,1} \leftarrow Pu_{i,1} \times (Pb_i)^{-1}$. Використовується функція cublasDtrsm;

3) асинхронне копіювання у потоці cudaCopyStreamArr[$i \bmod numGPU$] i-ї плитки в пам'ять CPU;

4) встановлення поточним GPU, в якому має зберігатися плитка, що буде потрібна на наступному кроці. Копіювання цієї плитки у пам'ять відповідного GPU, використовуючи cudaCopyStreamArr;

5) копіювання i-ї плитки в пам'ять всіх інших GPU;

6) виконання циклу по j від 1 доки виконується нерівність $j \times w < len[i]$, на кожній ітерації виконати перетворення (14), використовуючи функцію cublasDgemm. Перед викликом cublasDgemm потрібно встановити GPU з номером $(i + j) \bmod numGPU$, як поточний;

7) синхронізація потоків cudaCopyStreamArr та збереження i-ї плитки на CPU;

8) синхронізація основних потоків cuBLAS за допомогою cudaCalculateStreamArr.

Після виконання всіх кроків алгоритму отримаємо нижню трикутну матрицю, для якої справджується формула (6) другого етапу.

Розв'язання трикутних систем. Кількість операцій при розв'язанні трикутних систем (12), (13) виконується значно менше, ніж при трикутному розвиненні матриці системи. Ці підзадачі ефективно можна реалізувати на CPU, використовуючи відповідні програми з бібліотеки Intel MKL [8].

Експериментальне дослідження ефективності гібридного алгоритму ітерацій на підпросторі. На основі запропонованого гібридного алгоритму ітерацій на підпросторі було створено програму для комп'ютера гібридної архітектури, використовуючи для розпаралелювання обчислень між ядрами CPU системи MPI, а для виконання обчислень на GPU – технологію CUDA.

Проведено апробацію розробленого гібридного алгоритму на інтелектуальному персональному суперкомп'ютері гібридної архітектури Інпарком_pg (1 обчислювальний вузол, 2 процесори Xeon 5606, 2 GPU Tesla K40) на різних задачах, наприклад, використовувалася узагальнена АПВЗ: $Ax = \lambda Bx$, матриці A і B отримано при дискретизації методом кінцевих елементів задачі на власні значення для оператора Лапласа в прямокутнику, одна сторона якого закріплена [9].

В таблиці приведено порівняльні часові характеристики розв'язування трьох задач на власні значення для оператора Лапласа паралельним блочно-циклічним та гібридним алгоритмами ітерацій на підпросторі на Інпарком_pg при використанні різної кількості процесів (ядер) на CPU та процесорів GPU, а саме:

задача 1 – n = 811 801, m = 901;

задача 2 – n = 1 002 001, m = 1 001;

задача 3 – n = 1 442 401, m = 1 201.

Тут n – порядок матриці, m – на півширина стрічки.

ТАБЛИЦЯ. Порівняння часу (сек) розв'язування АПВЗ паралельним та гібридним алгоритмами ітерацій на підпросторі

Задача	Паралельний алгоритм			Гібридний алгоритм	
	1 CPU	4 CPU	8 CPU	4CPU + 1 GPU	8 CPU + 2 GPU
1	1311,4	452,6	225,8	209,6	147,7
2	2420,8	1480,5	632,8	386,8	298,5
3	3224,1	1752,8	942,8	502,1	359,5

З таблиці ми бачимо, що час розв'язування задач за розробленим гібридним алгоритмом методу ітерацій на підпросторі значно менший у порівнянні з часом розв'язування цих задач за допомогою блочно-циклічного алгоритму. Зокрема, використовуючи один GPU було досягнуто прискорення в 6 – 8 раз у порівнянні з послідовною версією (1 CPU), а використовуючи 2 GPU отримано прискорення в 7 − 9 раз. При розв'язуванні задач гібридним алгоритмом з використанням одного GPU отримано прискорення в 1,25 раз у порівнянні з часом розв'язування паралельним блочно-циклічним алгоритмом на восьми процесах, а при використанні двох GPU – в 1,4 раз. Розроблений гібридний алгоритм добре масштабується та показує пропорційне зменшення часу розв'язування.

Висновки. Запропоновано гібридний алгоритм методу ітерацій на підпросторі для стрічкових матриць, який забезпечує високу ефективність розпаралелювання на GPU, враховує структуру стрічкової матриці, оптимізує використання пам'яті GPU.

Подальші дослідження доцільно направити на розробку алгоритму для графічних прискорювачів архітектури Kepler [10] та CUDA 7.5, зокрема використання можливості динамічного паралелізму, безпосереднього обміну даними між GPU без участі центрального процесора, а також забезпечити врахування архітектурних особливостей центрального процесора, використовуючи бібліотеку обчислювальної математики Intel MKL.

А.Н. Химич, А.В. Попов, А.Ю. Баранов, А.В. Чистяков

ГИБРИДНЫЙ АЛГОРИТМ РЕШЕНИЯ ЗАДАЧ НА СОБСТВЕННЫЕ ЗНАЧЕНИЯ ДЛЯ ЛЕНТОЧНЫХ МАТРИЦ

Предложен алгоритм решения частичной обобщенной алгебраической проблемы собственных значений с ленточными симметричными матрицами на компьютерах гибридной архитектуры. Приведены результаты апробации алгоритма на персональном суперкомпьютере гибридной архитектуры Инпарком_pg.

A.N. Khimich, A.V. Popov, A.Y. Baranov, A.V. Chystyakov

HYBRID ALGORITHM FOR SOLVING EIGENVALUES PROBLEM FOR BAND MATRICES

This paper proposes an algorithm for solving partial generalized algebraic eigenvalue problem with band symmetric matrices on hybrid computers. The results of testing of the algorithm on a personal hybrid supercomputer Inparkom_pg are shown.

1. *Бате К., Вилсон Е.Л.* Численные методы анализа и метод конечных элементов. – М.: Стройиздат, 1982. – 447 с.
2. *Химич А.Н., Молчанов И.Н., Попов А.В., Чистякова Т.В., Яковлев М.Ф.* Параллельные алгоритмы решения задач вычислительной математики. – Киев: Наук. думка, 2008. – 247 с.
3. *Парлет Б.* Симметричная проблема собственных значений. – М.: Мир, 1983. – 318 с.
4. *Попов А.В., Химич А.Н.* Параллельный алгоритм решения системы линейных алгебраических уравнений с ленточной симметричной матрицей // Компьютерная математика. – 2005. – № 2. – С. 52 − 59.
5. *Уилкинсон Дж.Х., Райнш К.* Справочник алгоритмов на языке Алгол. Линейная алгебра. – М.: Машиностроение, 1976. – 389 с.
6. *Buttari A., Langou J., Kurzak J., Dongarra J.* A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures. // Parallel Computing. – 2009. – Vol. 35, Issue 1. – P. 38. – 53.
7. *Хіміч О.М., Баранов А.Ю.* Гібридний алгоритм розв'язування лінійних систем зі стрічковими матрицями прямими методами // Компьютерная математика. – 2013. – Вып. 2. – С. 80 – 87.
8. *Math* Kernel Library (Intel (R) MKL) Documentation // https://software.intel.com/en-s/intel-mkl
9. *Молчанов И.Н., Попов А.В, Химич А.Н.* Алгоритм решения частичной проблемы собственных значений для больших профильных матриц // Кибернетика и системный анализ. – 1992. – № 2. – С. 141 – 147.
10. *Вычислительная* архитектура NVIDIA Kepler высокопроизводительные вычисления NVIDIA // http://www.nvidia.com/object/nvidia-kepler.html

Одержано 20.04.2016