Технології розподілених систем та паралельних обчислень

Лекція №8

Внутрішній паралелізм

У процесі тривалого використання послідовних комп’ютерів був накопичений значний багаж обчислювальних алгоритмів та програм. Поява паралельних комп’ютерів повинна була зумовити розробку нових ефективних паралельних методів. Але на практиці цього не відбулося. Тому постає питання, як тоді розв’язувати задачі на паралельних ком-п’ютерах?

Відповідь на поставлене питання у термінах, наведених у лекції 7, зводиться до наступного. Візьмемо довільний придатний алгоритм, записаний у вигляді математичних співвідношень, послідовних програм чи якимось іншим способом.

Припустимо, що для цієї форми запису побудовано граф алгоритму. Припустимо також, що для цього графа знайдено паралельну форму із достатньою шириною ярусів. Тоді розглянутий алгоритм, принаймні принципово, можна реалізувати на паралельній обчислювальній системі. Важливо зауважити, що згідно з твердженням 7, паралельна реалізація буде мати такі самі обчислювальні властивості, що й звичайна. Зокрема, чисельно стійкий початковий алгоритм зберігає цю властивість і у паралельній формі. Подібний паралелізм у алгоритмах отримав назву *внутрішнього паралелізму.*

Виявилося, що багато існуючих ефективних послідовних алгоритмів мають значний запас "внутрішнього паралелізму". Складність полягає лише у тому, як виявити цей паралелізм.

Паралелізм у алгоритмі множення матриць

Розглянемо наступний приклад. Нехай потрібно обчислити добуток $A = BC$ двох квадратних матриць *B* та *C* порядку *n*. Згідно визначення операції множення матриць

$$a_{ij} = \sum_{k=1}^{n} b_{ik} c_{kj}, \quad (i, j = 1, \ldots, n). \tag{1}$$

Самі ці формули не визначають алгоритм однозначно, оскільки не вказано порядок обчислення суми доданків $b_{ik} c_{kj}$. Однак відразу помітним є паралелізм обчислень. Він полягає у відсутності вказівок про якого-небудь порядку перебору індексів *i* та *j*.

Якщо операції множення та додавання виконуються точно, то усі порядки підсумування у (1) є еквівалентними і приводять до одного і того самого результату. Нехай вибрано наступний ε

$$a_{ij}^{(0)} = 0,$$
$$a_{ij}^{(k)} = a_{ij}^{(k-1)} + b_{ik} c_{kj}, \quad (i, j, k = 1, \ldots, n), \tag{2}$$
$$a_{ij} = a_{ij}^{(n)}.$$

Знову явно вказано паралелізм перебору індексів *i*, *j*. Однак по індексу *k* паралелізму вже нема, так як цей індекс має послідовно перебиратися від 1 до *n* (як це випливає із формули (2)).

Побудуємо тепер граф алгоритму (2). При побудові графа не будимо враховувати природу елементів матриць *A*, *B* та *C* (можна вважати їх елементами одного і того самого кільця). Вершини графа не можна розташовувати довільним чином. Прийнятний спосіб розташування підказує сам вигляд формул (7). Елементи графу будемо розташовувати у вузлах прямокутної ґратки у тривимірному просторі. Аналізуючи формулу (2) неважко помітити, що у вершину з координатами (i,j,k) буде передаватися результат операції, якій відповідає вершина $(i,j,k-1)$.

Граф алгоритму влаштований достатньо просто. Він розпадається на n^2 незв'язних компонент. Кожний підграф містить *n* вершин і являє собою ланцюг, розташований паралельно осі *k*. У випадку $n=2$ граф наведений на рис. 1, *а*.

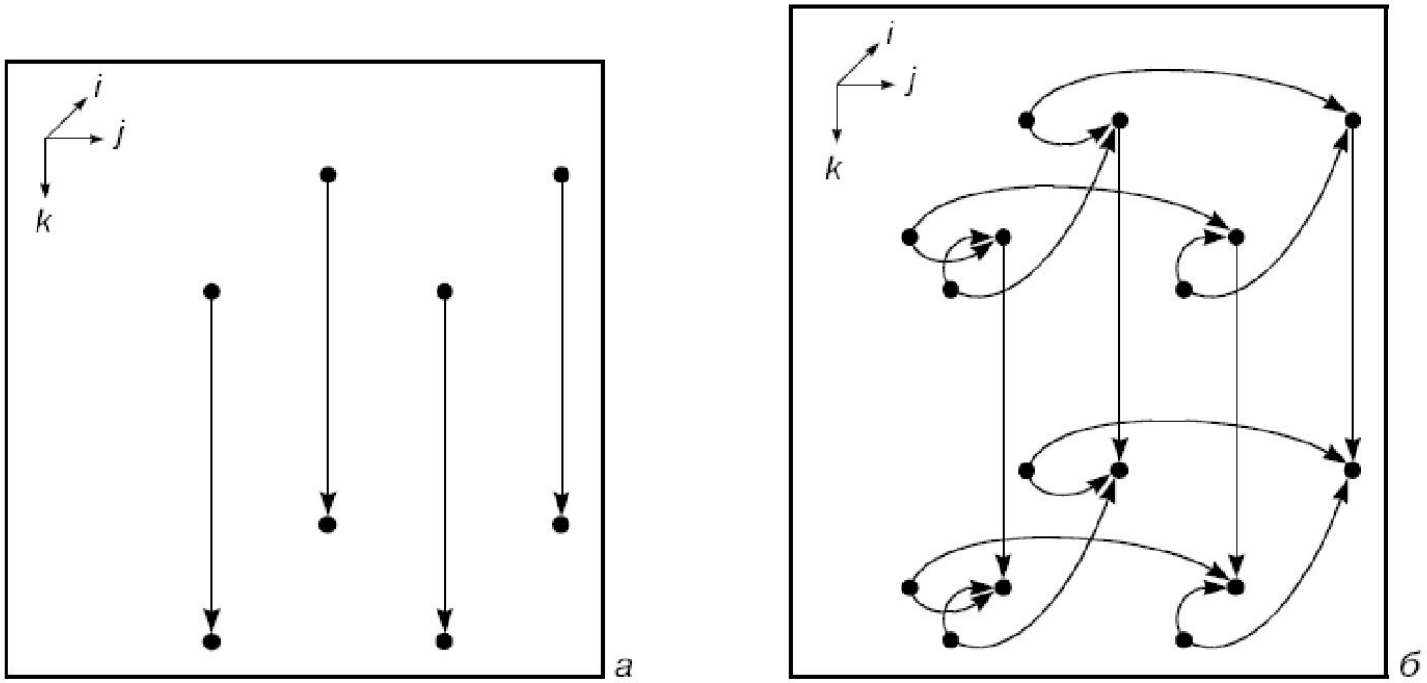

Рис. 1. Граф алгоритму множення матриць

У повному графі присутня множинна розсилка даних. Елемент b_{ik} розсилається по усім вершинам, які мають ті самі значення координат *i* та *k*. Для випадку $N = 2$ відповідні розсилки елементів матриць *B* та *C* наведені на рис. 31, *б*. Наведений приклад також демонструє, як важливо правильно розташовувати вершини графа.

Слід зауважити, що якщо додавання у (1) виконується за схемою здвоєння, то кожний вертикальний ланцюг у графі має бути замінений на дерево, наведене на рис. 1.

Паралелізм у алгоритмі розв’язування системи лінійних алгебраїчних рівнянь

Нехай потрібно знайти розв’язок системи лінійних алгебраїчних рівнянь $A\mathbf{x} = \mathbf{b}$ з невиродженою трикутною матрицею порядку *n* методом зворотної підстановки. Припустимо, що матриця *A* є лівою трикутною матрицею з одиничною діагоналлю. Тоді маємо

$$x_1 = b_1, \quad x_i = b_i - \sum_{j=1}^{i-1} a_{ij} x_j, \; (2 \le i \le n). \qquad (3)$$

Цей запис також не визначає алгоритм однозначно, бо не вказано порядок обчислення сум. Розглянемо наступне уточнення процесу (3):

$$\begin{gathered} x_i^{(0)} = b_i, \\ x_i^{(j)} = x_i^{(j-1)} - a_{ij} x_j^{(j-1)}, \quad i = 1, 2, \ldots, n, \; j = 1, 2, \ldots, i-1, \\ x_i = x_i^{(i-1)}. \end{gathered} \qquad (4)$$

Основна операція алгоритму має вигляд $a_{ij} = a_{ij} - a_{ik} b_{kj}$. Вона виконується для усіх допустимих значень індексів *i* та *j*. Для побудови графа алгоритму в декартовій системі координат з осями *i* та *j* побудуємо прямокутну сітку і розмістимо у вузлах при $2 \le i \le n, 1 \le j \le i-1$ вершини графа, які відповідають операціям $a_{ij} = a_{ij} - a_{ik} b_{kj}$. Також зобразимо на графі вершини, які відповідають вводу вхідних даних a_{ij} та b_i. Цей граф для випадку $n = 5$ зображено на рис. 2. Верхня кутова вершина відповідає b_1 знаходиться у точці (1, 0).

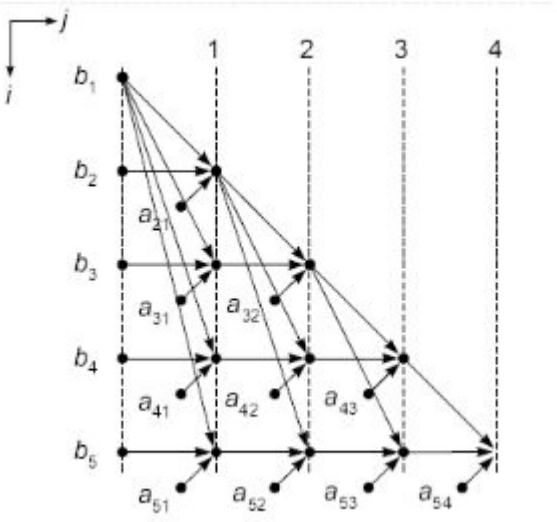

Рис. 2. Граф для алгоритму зворотної підстановки (4) для трикутної системи

На цьому рисунку зображена одна із максимальних паралельних форм. Її яруси помічені пунктиром. Ця паралельна форма стане канонічною, якщо вершини, відповідні за ввід елементів , розташувати у першому ярусі. Загальна кількість ярусів (без урахування вводу) рівна n - 1 .

Вибір зростаючого по *j* напрямку додавання у (3), який призвів до алгоритму (4), був зроблений, взагалі кажучи, випадково.

Аналогічно можна побудувати алгоритм зворотної підстановки з використанням додавання за спаданням індексу *j*

$$x_i^{(i)} = b_i,$$

$$x_i^{(j)} = x_i^{(j+1)} - a_{ij} x_j^{(j)}, \quad i = 1, 2, \ldots, n, \; j = 1, 2, \ldots, i-1, \qquad (5)$$

$$x_i = x_i^{(i-1)}.$$

Відповідний граф для випадку n = 5 наведено на рис. 3. Тепер верхня кутова вершина розташована у точці (1, 1).

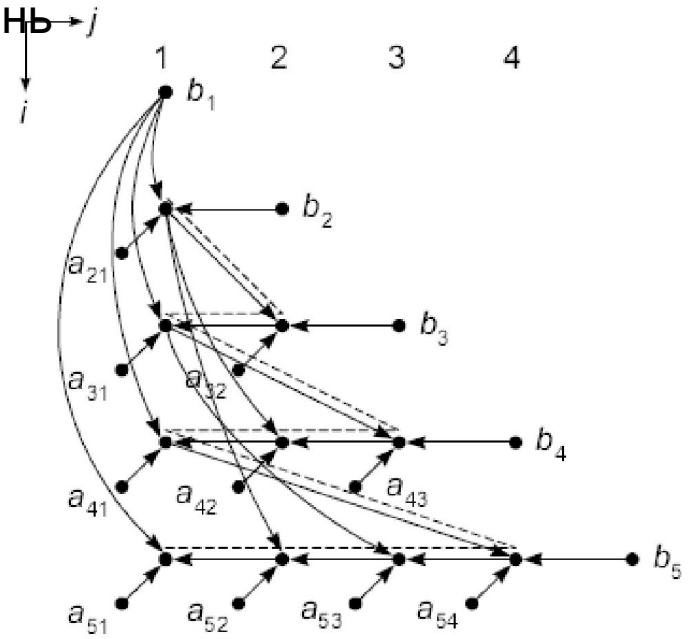

Рис. 3. Граф для алгоритму зворотної підстановки (5) для трикутної системи

Пробуючи розташувати вершини, які відповідають операціям $a - bc$, за ярусами хоча б однієї паралельної форми, приходимо до висновку, що тепер у кожному ярусі завжди може знаходитися тільки одна вершина. Цей факт пояснюється тим, що усі вершини графа на рис. 3 лежать на одному шляху, який позначений на рисунку пунктиром. Тому загальна кількість ярусів алгоритму (5), які містять операції вигляду $a - bc$, завжди рівна $(n^2 + n - 2)$, що набагато більше за число n - 1 — кількість ярусів для відповідних операцій у алгоритмі (4).

Отриманий результат є досить несподіваним. Обидва алгоритми (4) та (5) призначені для розв'язування тієї самої задачі і розроблені на основі формул (3). Обидва алгоритми абсолютно однакові з точки зору їх реалізації на багатопроцесорній системі, оскільки потребують виконання однакової кількості операцій множення та віднімання і однакового об'єму пам'яті і є еквівалентними з точки зору помилок заокруглення.

Тим не менш, паралельні графи алгоритмів принципово різні. Якщо ці алгоритми реалізувати на паралельній системі з n універсальними процесорами, то алгоритм (4) можна реалізувати за час, пропорційний n, а алгоритм (5) — лише за час пропорційний . У першому випадку завантаженість процесорів близька до 0, 5, а у другому — до 0.

Таким чином, алгоритми, цілком однакові при послідовній реалізації, можуть виявитися принципові відмінними при реалізацій на паралельній обчислювальній системі.

В цьому, взагалі кажучи, і полягає основна складність програмування програмного забезпечення для обчислень на паралельних комп'ютерах.

Дякую за увагу!