Технології розподілених систем та паралельних обчислень

Лекція №7

Концепція необмеженого паралелізму

Поява перших паралельних обчислювальних систем в 60-х роках XX століття зумовила необхідність математичної концепції побудови *паралельних алгоритмів,* тобто алгоритмів, пристосованих до реалізації на таких системах. Тогочасний швидкий розвиток елементної бази підказував дослідникам, що кількість обчислювальних пристроїв у системі невдовзі може стати дуже великим. Відповідна концепція отримала назву *концепції необмеженого паралелізму*.

В основі концепції лежить припущення, що алгоритм реалізується на паралельній обчислювальній системі, яка не накладає на нього практично ніяких обмежень. Згідно до концепції вважається, що:

- процесорів може бути як завгодно багато;
- усі процесори системи є універсальними;
- процесори працюють у синхронному режимі;
- усі запам’ятовуючі пристрої системи спільні;
- передавання інформації у системі виконується миттєво і без конфліктів.

Концепція необмеженого паралелізму є ідеалізованою математичною моделлю паралельної обчислювальної системи. Вона має як свої переваги, так і недоліки. Розглянемо результати, отримані на її основі.

Для знаходження розв’язку однієї і тої самої задачі можуть використовуватися алгоритми різної паралельної складності. Серед них можуть бути і алгоритми найменшої висоти. Розглянемо класичний приклад, який демонструє принципи побудови алгоритмів "малої висоти".

Обчислення добутку елементів масиву

Нехай потрібно обчислити добуток n чисел $a_1, a_2, \ldots, a_n$.

Розглянемо випадок $n = 8$

Тоді звичайна схема, у якій реалізовується послідовний процес обчислень, має наступний вигляд:

Дані: $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$

Ярус 1: $a_1 a_2$

Ярус 2: $(a_1 a_2) a_3$

Ярус 3: $(a_1 a_2 a_3) a_4$

Ярус 4: $(a_1 a_2 a_3 a_4) a_5$

Ярус 5: $(a_1 a_2 a_3 a_4 a_5) a_6$

Ярус 6: $(a_1 a_2 a_3 a_4 a_5 a_6) a_7$

Ярус 7: $(a_1 a_2 a_3 a_4 a_5 a_6 a_7) a_8$

Висота паралельної форми рівна 7, ширина рівна 1. Якщо обчислювальна система має більше одного процесора, то за даної схеми усі вони крім одного будуть простоювати.

Наступна паралельна форма іншого алгоритму обчислення добутку використовує процесори більш ефективно:

Дані: $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$

Ярус 1: $a_1a_2 \quad a_3a_4 \quad a_5a_6 \quad a_7a_8$

Ярус 2: $(a_1a_2)(a_3a_4) \quad (a_5a_6)(a_7a_8)$

Ярус 3: $(a_1a_2a_3a_4) \quad (a_5a_6a_7a_8)$

Висота паралельної форми рівна 3, ширина рівна 4. Суттєве зменшення висоти зумовлене більшим завантаженням процесорів виконанням корисної роботи.

Відповідні графи описаних алгоритмів наведені на рис. 1 (початкові вершини символізують ввід даних).

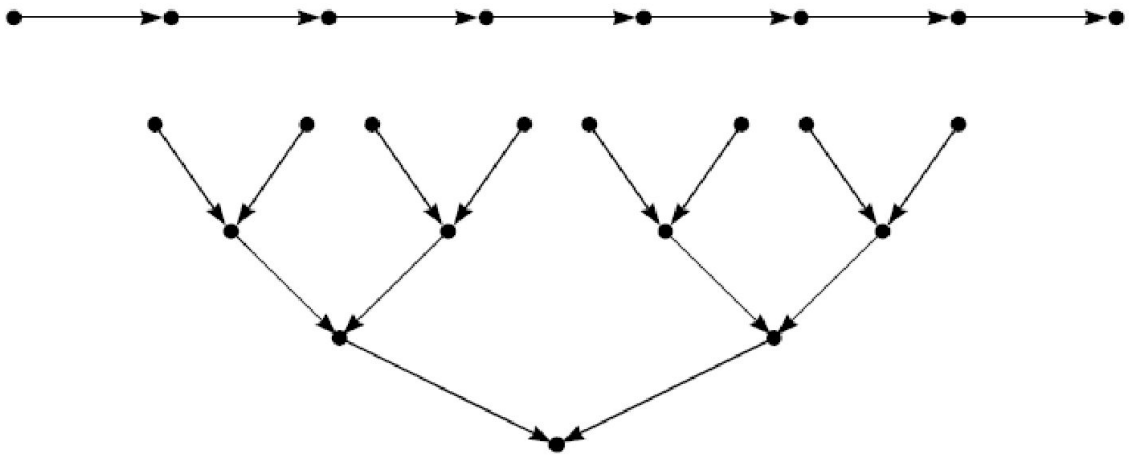

Рис. 1. Послідовний граф та граф алгоритму "здвоєння"

Остання схема очевидним чином розповсюджується на випадок довільного *n.* Для її реалізації потрібно на кожному ярусі виконувати максимально можливу кількість множень пар чисел, які отримані на попередньому ярусі (і не мають спільних множників). В загальному випадку висота паралельної форми рівна $\lceil \log_2 n \rceil$, означає "найближче зверху" до $\lceil \cdot \rceil$ ціле число. Ця паралельна форма $\left\lceil \frac{n}{2} \right\rceil$ може бути реалізована на процесорах, причому ступінь завантаженості процесорів зменшується від ярусу до ярусу.

Процес побудови елементів кожного ярусу називається процесом *здвоєння.*

З використанням процесу здвоєння можна будувати алгоритми логарифмічної висоти для обчислення значень довільної асоціативної операції, наприклад додавання векторів, множення матриць і т. п.

Твердження 3

Нехай функція суттєво залежить від n аргументів і зображується у вигляді суперпозиції скінченної кількості операцій, кожна з яких має не більше p аргументів. Припустимо, що значення функції обчислюється за допомогою деякого алгоритму з використанням тих самих операцій. Тоді якщо s — висота алгоритму (без урахуванням вводу даних), то $s \geq \log_p n$.

Якщо деяка задача має n вхідних даних, то можна в загальному випадку можна розраховувати на існування алгоритму її розв'язання із висотою не більше, ніж $\log n$. Якщо вдається розробити алгоритм висоти $\log^{\alpha} n$, де $\alpha > 0$, то такий алгоритм можна вважати достатньо ефективним з точки зору його реалізації на паралельній обчислювальній системі.

Обчислення добутку матриці на вектор

Нехай вектор $\mathbf{y} = (y_1, \ldots, y_n)$ обчислюється як добуток квадратної матриці $A = (a_{ij}), (i, j = 1, \ldots, n)$ та вектора $\mathbf{x} = (x_1, \ldots, x_n)$, тобто

$$y_i = \sum_{j=1}^{n} a_{ij} x_j .$$

Припустимо, що обчислювальна система має n^2 процесорів. Тоді на першому кроці можна паралельно обчислити усі n^2 добутків $a_{ij} x_j$.

Обчислення добутку матриці на вектор

Потім, з використанням схеми здвоєння для додавання, за $\lceil \log_2 n \rceil$ кроків обчислити паралельно усі n сум, які визначають координати вектора **y**. Тобто, ми отримали алгоритм обчислення добутку квадратної матриці та n-вимірного вектора, який має ширину n^2 та висоту $\lceil \log_2 n \rceil$. При цьому процесори використовуються нерівномірно. Усі процесори задіяні тільки на першому кроці. Далі кількість працюючих процесорів зменшується удвічі на кожному кроці.

За допомогою аналогічних міркувань будується паралельний алгоритм розв'язування задачі множення двох квадратних матриць. Цю задачу можна звести до *n* задач множення першої матриці на послідовні стовпці другої матриці. Якщо використати попередній алгоритм, то отримуємо алгоритм, який має висоту порядку $\log_2 n$ та ширину n^3 .

Слід звернути увагу на те, що у розглянутих паралельних версіях алгоритмів множення матриць та векторів виникає необхідність одночасного надсилання одним і тих самих даних на різні процесори. Такі операції не можна виконати дуже швидко. Тому на практиці процес обчислень може значно уповільнюватися.

Недоліки концепції необмеженого паралелізму

З використанням концепції необмеженого паралелізму розроблено велику кількість алгоритмів невеликої висоти. З деякими з них можна познайомитися в [1, 5].

Однак слід зазначити, що переважна більшість з цих алгоритмів виявилися практично непридатними на практиці. Основні причини цього — велика кількість необхідних процесорів, складні інформаційні зв'язки між операціями, катастрофічна обчислювальна нестійкість, велика кількість конфліктів пам'яті.

Докази практичної непридатності алгоритмів з використанням концепції необмеженого паралелізму можна також отримати проаналізувавши типові прикладні програмні пакети, які постачаються разом із популярними паралельними обчислювальними системами. По суті, усі вони складаються із програм, які реалізують ті самі методи, які добре себе зарекомендували на послідовних комп'ютерах. Реально в деякій мірі використовується лише принцип здвоєння для обчислення сум та добутків чисел.

Дякую за увагу!