Курс «Паралельні алгоритми: побудова та аналіз».

Лекція 5. ***Методи синтезу паралельних алгоритмів. Паралельні алгоритми обчислення рекурсій.***

План лекції.

1. Поняття дрібнозернистого та крупнозернистого алгоритмів; сильнозв'язані та слабозв'язані задачі.
2. Деякі загальні методи синтезу паралельних алгоритмів:
   - декомпозиція за даними;
   - функціональна декомпозиція;
   - декомпозиція області розв'язання задачі;
   - реструктуризація даних.
3. Поняття рекурсії.
4. Паралельні алгоритми обчислення рекурсій:
   - алгоритм паралельного каскадного сумування;
   - алгоритм циклічної редукції.

1. Зернистість алгоритму або ступінь його **грануляції** розглядають як міру ступеня паралелізму цього алгоритму. **Дрібнозернистий** алгоритм дозволяє виділити в ньому паралельні гілки лише малого обсягу. А **крупнозернистий** алгоритм складається із незалежних або слабозв'язаних гілок значного обсягу, які можуть оброблятися паралельно. Крупнозернистий алгоритм часто ще називають **крупноблочним** алгоритмом.

Задача, для якої відомі лише дрібнозернисті алгоритми її розв'язання, називається **сильнозв'язаною** задачею. А задача, для якої відомий хоча б один крупнозернистий алгоритм її розв'язання, називається **слабозв'язаною** задачею.

Очевидно, що паралельні алгоритми, орієнтовані на *SIMD*- та *MIMD*-системи, суттєво **відрізняються** своєю **зернистістю**.

2. Паралелізмом за даними володіють задачі, які включають у себе неодноразове виконання одного і того ж фрагменту обчислень з різними вхідними даними. Такі обчислення, очевидно, можуть виконуватися паралельно. Якщо задача володіє паралелізмом за даними, то відповідну паралельну програму доцільно розробляти у вигляді сукупності однакових процесів, кожен з яких виконується на своєму підпорядкованому процесорі, та головного процесу, який виконується на керуючому процесорі. Алгоритм, який реалізує така паралельна **програма**, є зазвичай **крупнозернистим**. Схематично це можна подати так, як показано на рис.1.

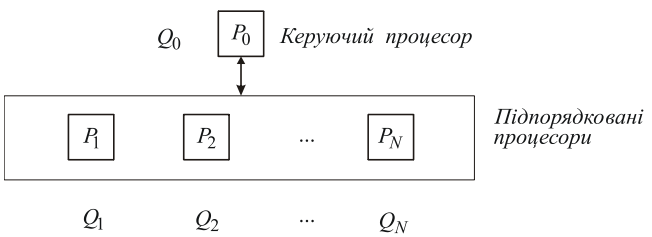

Рис. 1

Тут Q_0 – головний процес, а $Q_1, Q_2, \ldots, Q_N$ – однакові підпорядковані процеси.

Метод розпаралелювання на підставі паралелізму даних називається **декомпозицією за даними**. Розглянемо простенький приклад застосування цього методу. Нехай на *N*-процесорній *MIMD*-обчислювальній системі необхідно виконати додавання двох векторів $\vec{a}, \vec{b}$ з кількістю компонент n кожний і одержати унаслідок цього вектор $\vec{c}$ з такою ж кількістю компонент. Тоді **декомпозиція за даними** полягає в розподіленні n/N елементів векторів $\vec{a}, \vec{b}$ по процесорах $P_1, P_2, \ldots, P_N$ і обчисленні відповідних елементів результуючого вектора $\vec{c}$.

Декомпозиція за даними може бути **статичною і динамічною**. У разі статичної декомпозиції за даними кожному процесорові наперед призначається його частка даних. У випадку ж динамічної декомпозиції блоки даних розподіляються по процесорах під час розв'язання задачі в порядку надходження даних і звільнення відповідних процесорів.

Функціональний паралелізм – це паралелізм груп операцій, об'єднаних за функціональною ознакою. Метод розпаралелювання на підставі функціонального паралелізму називається **функціональною декомпозицією**. Тривіальним прикладом функціональної декомпозиції є декомпозиція деякої задачі на **чотири такі підзадачі**: введення початкових даних, оброблення їх, виведення результатів, візуалізація результатів. Паралелізм при цьому досягається унаслідок одночасного виконання чотирьох вказаних підзадач і створення «конвеєра» між ними. Зазначимо, що кожна із згаданих підзадач може володіти паралелізмом за даними. У разі функціональної декомпозиції задачі кількість використовуваних процесорів визначається кількістю підзадач. Збільшити кількість процесорів з метою збільшення швидкості розв'язання задачі за такого підходу є досить проблематично.

Геометричним паралелізмом володіють, наприклад, фізичні задачі, що описуються диференціальними рівняннями в частинних похідних (задачі механіки суцільного середовища, теорії поля тощо). Такі задачі зазвичай розв'язуються з допомогою методів скінченних різниць або скінченних елементів. Для дискретних аналогів таких задач характерними є локально-обмежені взаємодії між вузлами сітки, що покриває область розв'язання. А це дозволяє розбити цю область на «підобласті» і обчислення в кожній із них «доручити» окремому процесору. Схематично це виглядає так, як зображено на рис. 2.

Тут $\Omega = \Omega_1 \cup \Omega_2 \cup \ldots \cup \Omega_N$. **Відмінність** геометричного паралелізму від паралелізму за даними полягає в тому, що підзадачі оброблення кожної із підобластей Ω_i $(i = \overline{1, N})$ взаємозв'язані між собою (потрібен обмін даними між цими підзадачами).

Метод декомпозиції області розв'язання є ефективним за умови, що обчислювальна складність кожної із підзадач є приблизно однаковою. Крім цього, для ефективності даного методу програма, що виконується деяким процесором P_i, повинна використовувати лише невеликий обсяг даних, розташованих на інших процесорах. Бажано, щоб ці, не локальні дані були розташовані на невеликій кількості сусідніх процесорів.

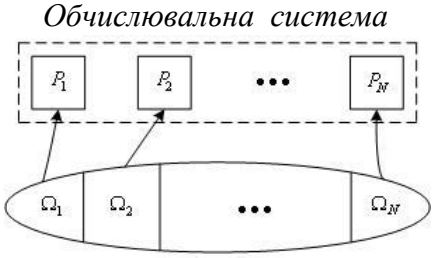

Рис. 2

Можна виділити **статичну декомпозицію** області розв'язання **і динамічну**. Якщо обчислювальні складності підзадач оброблення кожної із підобластей Ω_i $(i = \overline{1, N})$ змінюються в процесі обчислень, то **статична декомпозиція** області може виявитися малоефективною. У цьому випадку може бути більш доцільною **динамічна декомпозиція** області, за якої межі між підобластями змінюються під час обчислень. Така ситуація є можливою, наприклад, під час розв'язування задач механіки суцільного середовища на **адаптивних сітках**.

Метод реструктуризації даних продемонструємо на відомому вже прикладі. Розглянемо задачу обчислення добутку n = 8 дійсних чисел $a_1, a_2, \ldots, a_8$. У найпростішому випадку обчислення цього добутку можна організувати у відповідності з графом алгоритму, тобто послідовно. Якщо для обчислень використовується $N > n$ процесорів, то у даному разі в кожний момент часу простоюють усі процесори, крім одного. Більш ефективним для розв'язання сформульованої задачі є алгоритм, що реалізує схему повного двійкового дерева, який можна застосувати для будь-якої асоціативної операції. У цьому випадку висота ЯПФ дорівнює 3, а її ширина дорівнює 4. Однак, якщо у нас є хоча б 4 процесори, то на першому кроці будуть залучені усі з них, на другому – лише два, а на третьому – один. Якщо ж виявляться потрібними значення деяких проміжних добутків, то на всіх кроках можна залучити по 4 процесори.

3. У чисельному аналізі **рекурсії** використовують досить широко (розв'язання СЛАР методом виключення Гаусса і будь-яким із ітераційних методів, більшість методів інтегрування звичайних диференціальних рівнянь тощо).

Рекурсія за своєю суттю, задає послідовність обчислень і тому є певною проблемою для розпаралелювання. Обмежимося розглядом **лінійних рекурсій** вигляду:

$$x_j = a_j x_{j-1} + d_j, \quad j = \overline{1, n}, \tag{1}$$

де $x_0; a_1, a_2, \ldots, a_n; d_1, d_2, \ldots, d_n$ – задані константи. Нехай $x_0 = a_1 = 0$ (цього легко досягти, якщо прийняти $d_1 = d_1 + a_1 x_0$). У даному випадку граф алгоритму для обчислення (1) при $n = 4$ зображено на рис. 3.

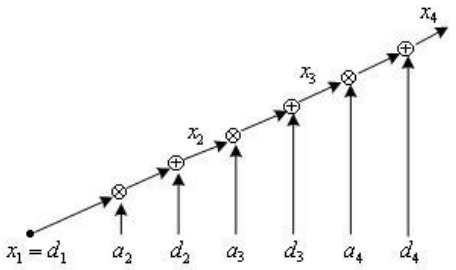

Рис. 3

4. Розглянемо **алгоритм паралельного каскадного сумування**. Покладемо $a_2 = a_3 = ... = a_n = 1$. У цьому випадку формула (1) є рекурентним записом алгоритму сумування чисел $d_1, d_2, ..., d_n$.

Структура інформаційних зв'язків алгоритму паралельного каскадного сумування для випадку n = 8 подана на рис. 4.

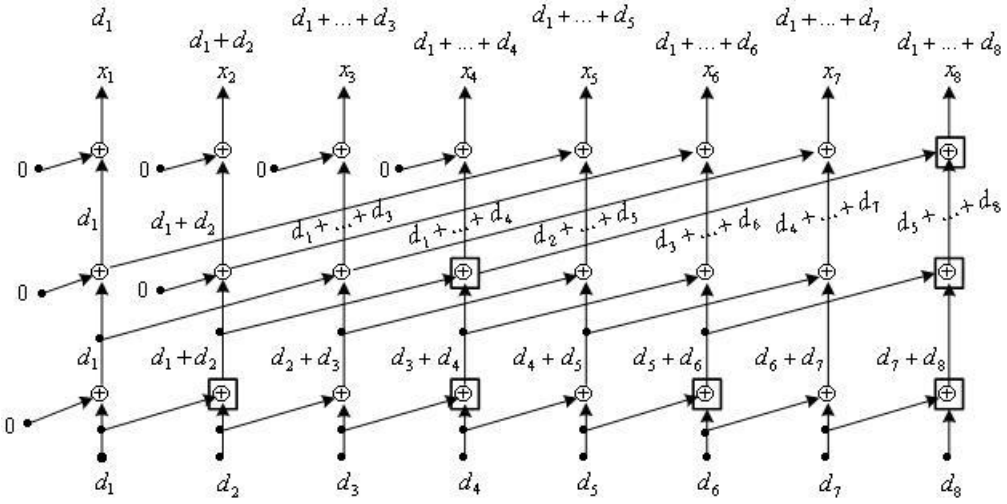

Рис. 4

Наведений алгоритм дозволяє одержати разом з величиною x_n усі проміжні величини $x_1, x_2, ..., x_{n-1}$. Якщо потрібне лише значення величини x_n, то достатньо виконати лише ті додавання, які виділені на рисунку квадратиками. У цьому випадку алгоритм перетворюється в алгоритм сумування за схемою **повного двійкового дерева**.

Легко побачити, що висота ярусно-паралельної форми (ЯПФ) алгоритму каскадного сумування дорівнює $\lceil \log_2 n \rceil$, а ширина кожного із ярусів дорівнює n. Зазначимо також, що наведений алгоритм збільшує загальну кількість операцій з (n-1) до $n\lceil \log_2 n \rceil$, де $\lceil M \rceil$– це найближче ціле число, що є більшим або дорівнює M.

Розглянемо **алгоритм циклічної редукції** (з латині «редукція» означає зведення, в логіці та математиці – це прийом зведення складного до простого). Покладемо тепер, що має місце $a_2 \neq 1, a_3 \neq 1, ..., a_n \neq 1$. **Головна ідея** алгоритму циклічної редукції полягає в об'єднанні суміжних членів рекурсії так, щоб одержати співвідношення між членами рекурсії x_j, x_{j-2}. Унаслідок цього одержуємо нову лінійну рекурсію з кількістю членів $n/2$, яка зв'язує кожну другу змінну вихідної

рекурсії. У цій новій рекурсії знову об'єднаємо суміжні члени і одержимо рекурсію з кількістю членів $n/4$, яка зв'язує кожну четверту змінну вихідної рекурсії. І т.д. Після $\log_2 n$ таких редукцій одержимо формулу для обчислення x_n. Розглянемо далі описану схему більш детально.

Редукція 1.

Із (1) випливає, що $x_{j-1} = a_{j-1}x_{j-2} + d_{j-1}$. Підставивши цей вираз в (1), одержимо лінійну рекурсію між кожним другим членом вихідної послідовності:

$$x_j = a_j(a_{j-1}x_{j-2} + d_{j-1}) + d_j = a_j a_{j-1} x_{j-2} + a_j d_{j-1} + d_j = a_j^{(1)} x_{j-2} + d_j^{(1)},$$

де $a_j^{(1)} = a_j a_{j-1}$; $d_j^{(1)} = a_j d_{j-1} + d_j$.

Редукція 2.

За тією ж схемою із попередньої рекурсії одержимо лінійну рекурсію між кожним четвертим членом вихідної рекурсії:

$$x_{j-2} = a_{j-2}^{(1)} x_{j-4} + d_{j-2}^{(1)};$$

$$x_j = a_j^{(1)} a_{j-2}^{(1)} x_{j-4} + a_j^{(1)} d_{j-2}^{(1)} + d_j^{(1)} = a_j^{(2)} x_{j-4} + d_j^{(2)},$$

де $a_j^{(2)} = a_j^{(1)} a_{j-2}^{(1)}$; $d_j^{(2)} = a_j^{(1)} d_{j-2}^{(1)} + d_j^{(1)}$.

- -

Редукція k.

За такою ж схемою із попередньої рекурсії одержимо наступну лінійну рекурсію:

$$x_j = a_j^{(k)} x_{j-2^k} + d_j^{(k)}, \qquad (2)$$

де $a_j^{(k)} = a_j^{(k-1)} a_{j-2^{k-1}}^{(k-1)}$; $d_j^{(k)} = a_j^{(k-1)} d_{j-2^{k-1}}^{(k-1)} + d_j^{(k-1)}$; $a_j^{(0)} = a_j$; $d_j^{(0)} = d_j$.

Якщо у формулі (2) нижні індекси виходять за межі відрізка [1, *n*], то відповідну компоненту рекурсії прирівнюємо до нуля.

- -

Редукція $\log_2 n$.

За розглянутою схемою одержуємо $x_j = a_j^{(\log_2 n)} x_{j-n} + d_j^{(\log_2 n)}$, а звідси маємо

$$x_n = d_n^{(\log_2 n)},$$

де $d_n^{(\log_2 n)} = a_n^{(\log_2 n-1)} \cdot d_{n-2^{\log_2 n-1}}^{(\log_2 n-1)} + d_n^{(\log_2 n-1)} = a_n^{(\log_2 n-1)} \cdot d_{n/2}^{(\log_2 n-1)} + d_n^{(\log_2 n-1)}$.

Отже, у відповідності до алгоритму циклічної редукції обчислення рекурсії зводиться до обчислення коефіцієнтів $a_j^{(k)}, d_j^{(k)}$, де $k = 1, 2, \dots, \log_2 n$, при цьому слід мати на увазі, що $a_j^{(0)} = a_j$; $d_j^{(0)} = d_j$. Обчислення згаданих коефіцієнтів бу-

демо виконувати **за схемою**, близькою **до паралельного каскадного сумування**. На наступному рисунку (див. рис. 5) наведена така схема для обчислення $a_j^{(k)}$ у разі, коли $n=8$.

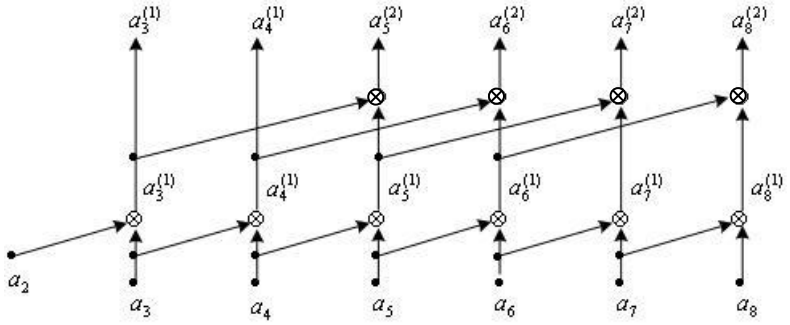

Рис. 5

Зауважимо, що на цьому рисунку показані лише фактично використовувані коефіцієнти. Легко побачити, що висота ЯПФ наведеного алгоритму обчислення коефіцієнтів $a_j^{(k)}$ дорівнює $\log_2 n-1$, тобто 2, а ширина ярусів змінюється **приблизно** від *n*-2 до *n*/2, тобто відповідно від 6 до 4.

Розглянемо схему обчислення коефіцієнтів $d_j^{(k)}$ у разі, коли $n=8$ (див. рис. 6).

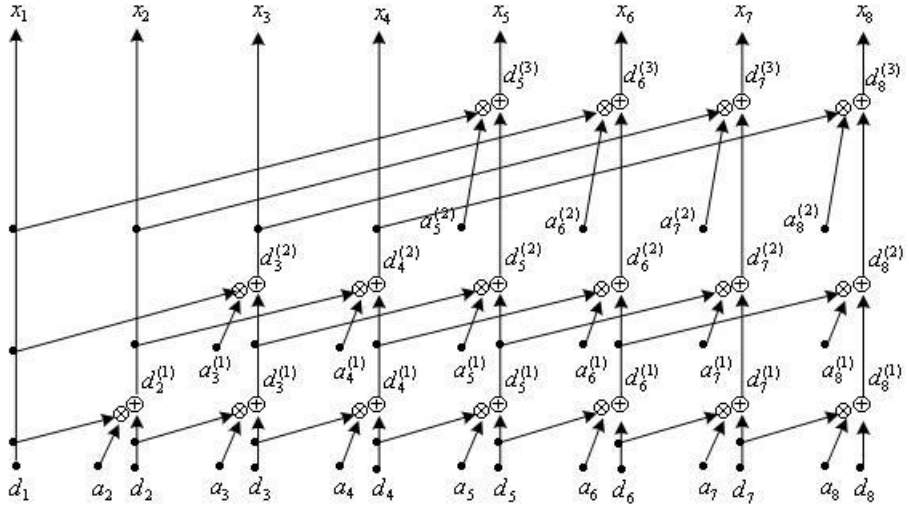

Рис. 6

Із наведеного рисунку бачимо, що висота ЯПФ такого алгоритму обчислення коефіцієнтів $d_j^{(k)}$ дорівнює $\log_2 n$, тобто 3, а ширина ярусів змінюється **приблизно** від *n*-1 до *n*/2, тобто відповідно від 7 до 4.