Лекція 6_дод. Деякі базові паралельні алгоритми обчислень (І ч.).

План лекції.

1. Розпаралелювання матрично-векторного добутку на підставі використання:
   - скалярних добутків;
   - лінійних комбінацій стовпців матриці.
2. Оцінка ефективності побудованих паралельних алгоритмів.
3. Розпаралелювання матричного множення на підставі блочного подання матриць.
4. Розпаралелювання ітераційного методу Якобі для розв'язування СЛАР.

1. Припустимо, що як обчислювальну систему ми використовуємо MIMD-систему з розподіленою пам'яттю, кількість процесорів у якій дорівнює $N>1$. Нехай A– матриця розмірності $m\times n$, а $\vec{x}$, $\vec{b}$ – вектори розмірності *n* та *m* відповідно. У даному разі вважаємо, що m – це кількість рядків, а n – це кількість стовпців матриці A. **Матрично-векторний добуток** $\vec{b}=A\cdot\vec{x}$ можна обчислити, подаючи $A\cdot\vec{x}$ у вигляді сукупності **скалярних добутків**:

$$\vec{b}=A\cdot\vec{x}=\begin{pmatrix}\vec{a}_1\cdot\vec{x}\\ \vec{a}_2\cdot\vec{x}\\ \cdots\cdots\\ \vec{a}_m\cdot\vec{x}\end{pmatrix},$$

де $\vec{a}_i$ – вектор, елементами якого є *i*-й рядок матриці *A*.

Для **простоти запису** будемо вважати, що кількість рядків m матриці A є кратною до кількості процесорів N в обчислювальній системі і до того ж $p=m/N$. Тоді **схему паралельного алгоритму** виконання матрично-векторного добутку можна подати у такому вигляді:

1) розподіляємо по процесорах рядки матриці A і компоненти вектора $\vec{x}$ так, як це зображено на рис. 1;

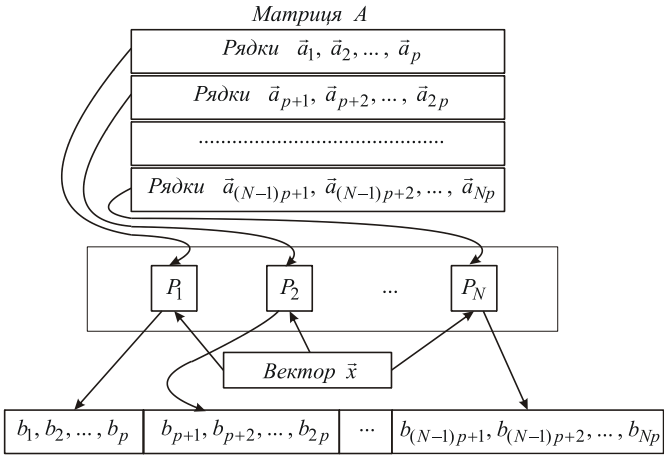

Рис. 1.

2) паралельно на всіх процесорах системи обчислюємо скалярні добутки відповідних рядків матриці A на вектор $\vec{x}$:
   - на процесорі P_1 – скалярні добутки:
   $$\vec{a}_1 \cdot \vec{x} = b_1,\ \vec{a}_2 \cdot \vec{x} = b_2, \ldots, \vec{a}_p \cdot \vec{x} = b_p;$$
   - на процесорі P_2 – скалярні добутки:
   $$\vec{a}_{p+1} \cdot \vec{x} = b_{p+1},\ \vec{a}_{p+2} \cdot \vec{x} = b_{p+2}, \ldots, \vec{a}_{2p} \cdot \vec{x} = b_{2p};$$
   ……
   - на процесорі P_N – скалярні добутки:
   $$\vec{a}_{(N-1)p+1} \cdot \vec{x} = b_{(N-1)p+1},\ \vec{a}_{(N-1)p+2} \cdot \vec{x} = b_{(N-1)p+2}, \ldots, \vec{a}_{Np} \cdot \vec{x} = b_{Np};$$
3) передаємо на один із процесорів усі обчислені скалярні добутки і формуємо вектор $\vec{b}$.

Матрично-векторний добуток можна також обчислити, використовуючи **лінійні комбінації стовпців** матриці A, тобто $\vec{b} = A \cdot \vec{x} = \sum_{j=1}^{n} x_j \cdot \vec{a}_j$, де $\vec{a}_j$ – j-й стовпець матриці A, а x_j – j-та компонента вектора $\vec{x}$. Отже, результатом добутку $x_j \cdot \vec{a}_j$ є вектор розмірності m.

Для **простоти покладемо**, що кількість стовпців n матриці A є кратною до N (кількість процесорів у системі) і при цьому $q = n/N$. Тоді **схему паралельного алгоритму** матрично-векторного добутку можна подати так:

1) розподіляємо по процесорах по q стовпців матриці і q елементів вектора $\vec{x}$ так, як зображено на рис. 2;

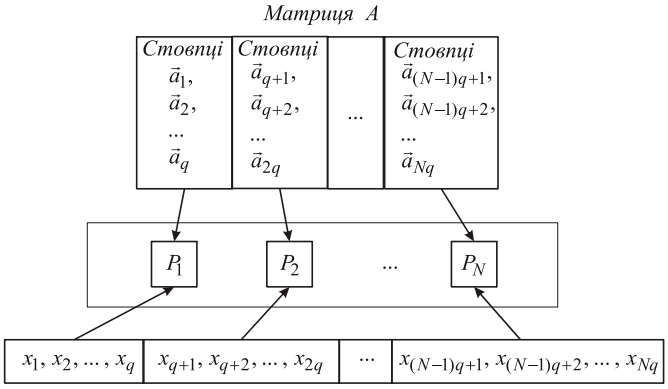

Рис. 2

2) паралельно на всіх процесорах системи виконуємо наступні обчислення:
   а) обчислюємо добутки відповідних стовпців матриці A на відповідні компоненти вектора $\vec{x}$:
   - на процесорі P_1 добутки $x_1\vec{a}_1, x_2\vec{a}_2, \ldots, x_q\vec{a}_q$;
   - на процесорі P_2 добутки $x_{q+1}\vec{a}_{q+1}, x_{q+2}\vec{a}_{q+2}, \ldots, x_{2q}\vec{a}_{2q}$;
   ……
   - на P_N добутки $x_{(N-1)q+1}\vec{a}_{(N-1)q+1}, x_{(N-1)q+2}\vec{a}_{(N-1)q+2}, \ldots, x_{Nq}\vec{a}_{Nq}$;

б) обчислюємо суми одержаних векторів:

- на процесорі P_1 суму $\sum_{i=1}^{q} x_i \vec{a}_i$;

- на процесорі P_2 суму $\sum_{i=q+1}^{2q} x_i \vec{a}_i$;

……

- на процесорі P_N суму $\sum_{i=(N-1)q+1}^{Nq} x_i \vec{a}_i$;

3) сумуємо за схемою повного двійкового дерева вектори, одержані на кожному із процесорів або передаємо їх всіх на один із процесорів, на якому виконується це сумування.

2. Оцінимо **прискорення** розглянутого вище **алгоритму** обчислення **матрично-векторного добутку** з використанням скалярних добутків. Для простоти викладу вважатимемо, що час виконання операції множення і додавання двох чисел на процесорах $P_i (i = \overline{1, N})$ є однаковим і дорівнює *t*. Нехай $n = m$. Тоді час обчислення одного скалярного добутку $\vec{a}_i \cdot \vec{x} = b_i$ складає $(n + (n-1))t$, а **обчислювальні витрати** T_{cal} кожного із процесорів дорівнюють:

$$T_{cal} = (n + (n-1))\, pt = (2n-1)\, pt \approx 2n^2 t / N .$$

Комунікаціні затрати T_{com} кожного із процесорів складаються із затрат на прийом np компонент відповідних векторів $\vec{a}_i$, прийом *n* компонент вектора $\vec{x}$ і передачу *p* компонент результуючого вектора $\vec{b}$. Отже,

$$T_{com} = 2S + (np + n + p)l / R \approx 2S + (n^2 / N + n)l / R ,$$

де *S* – це латентність комунікаційної мережі;

l – довжина дійсного числа в байтах;

R – пропускна здатність комунікаційної мережі в байт/с.

Діаметр комунікаційної мережі тут дорівнює 1.

Тобто **час виконання** розглядуваного **паралельного алгоритму** на *N* процесорах можна приблизно оцінити величиною:

$$T_N \approx 2n^2 t / N + 2S + (n^2 / N + n)l / R ,$$

а час виконання алгоритму на одному процесорі – величиною:

$$T_1 \approx (n + (n-1))nt \approx 2n^2 t .$$

Покладемо, що $t = 10^{-8}\text{с}, l = 32, S = 5 \cdot 10^{-5}\text{с}, R = 8 \cdot 10^7$ байт/с. Слід зазначити, що приблизно таким параметрам відповідає мережа, побудована за технологією SCI. Отже, для **прискорення алгоритму** маємо вираз:

$$S_N = \frac{T_1}{T_N} \approx \frac{20n^2 N}{20n^2 + 400(n^2 + nN) + 10^5 N} .$$

Із наведеної формули легко одержати, що на 4-х, 8-ми і 16-ти процесорних системах обчислення уповільнюються (прискорення є **меншим за 1**), а на 64-х процесорах прискорення обчислень **не перевищує 3**. Тобто у даному випадку ефективність розглянутого алгоритму є надзвичайно низькою. Зауважимо, що на прискорення впливає пропускна здатність комунікаційної мережі. Так, у разі її збільшення в 5 разів, тобто коли $R = 4 \cdot 10^8$ байт/с, прискорення обчислень **зростає приблизно у 4 рази**.

Слід зазначити, що наведені вище **оцінки** прискорення є доволі **спрощеними**. **Точну оцінку** прискорення можна отримати експериментально – на **конкретній** обчислювальній **системі**, у конкретному **середовищі програмування** і для конкретного **набору** вхідних **даних**.

Аналогічно можна оцінити прискорення для розглянутого паралельного алгоритму матрично-векторного добутку з використанням лінійних комбінацій стовпців матриці.

3. Матричне множення. Розглянемо A – матрицю розмірності $l \times m$ (l рядків, m стовпців) і B – матрицю розмірності $m \times n$. Тоді добутком цих матриць буде деяка матриця C розмірності $l \times n$: $C = A \cdot B$.

Для простоти будемо вважати, що розміри матриць l, m, n є кратними до кількості процесорів N у заданій обчислювальній MIMD-системі. Тоді матрицю C можна подати у вигляді:

$$C = A \cdot B = \begin{pmatrix} A_{1,1} & \dots & A_{1,N} \\ \dots & \dots & \dots \\ A_{N,1} & \dots & A_{N,N} \end{pmatrix} \begin{pmatrix} B_{1,1} & \dots & B_{1,N} \\ \dots & \dots & \dots \\ B_{N,1} & \dots & B_{N,N} \end{pmatrix} = \begin{pmatrix} C_{1,1} & \dots & C_{1,N} \\ \dots & \dots & \dots \\ C_{N,1} & \dots & C_{N,N} \end{pmatrix},$$

де розмірність кожного із блоків матриць A, B, C дорівнює відповідно $p \times q$, $q \times r$, $p \times r$. Тут $p = l/N$, $q = m/N$, $r = n/N$. Насправді блок $C_{j,k}$ матриці C є сумою N матричних добутків j-го рядка блоків матриці A на k-й стовпець блоків матриці B:

$$C_{j,k} = \sum_{i=1}^{N} A_{j,i} \cdot B_{i,k}, \text{ де } 1 \le j \le N, 1 \le k \le N.$$

В основі паралельного алгоритму матричного множення лежить блочне подання матриць. А сам алгоритм має таку схему:

1) розподіляємо по процесорах блоки матриці A та матрицю B так, як це показано на рис. 3;

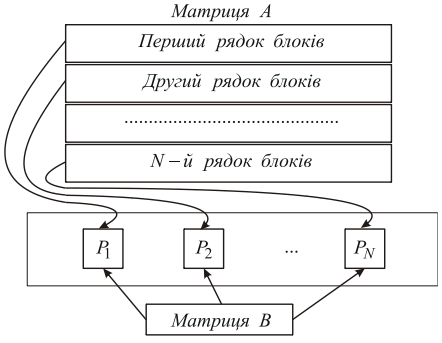

Рис. 3

2) паралельно на всіх процесорах системи здійснюємо обчислення компонент відповідних блоків матриці *C*:

- на процесорі P_1 – блоків $C_{1,k}$; $k = \overline{1, N}$, які утворюють перший рядок блоків матриці *C*;
- на процесорі P_2 – блоків $C_{2,k}$; $k = \overline{1, N}$, що утворюють другий рядок блоків матриці *C*;

…….

- на процесорі P_N – блоків $C_{N,k}$; $k = \overline{1, N}$, що утворюють *N*-й рядок блоків матриці *C*;

3) передаємо на один із процесорів всі обчислені блоки та формуємо матрицю *C*.

Досить легко можна запропонувати низку модифікацій даного алгоритму, наприклад:

- замість рядкового розподілу блоків матриці *A* по процесорах, можна використати розподіл матриці *B* за стовпцями;
- можна використовувати розбиття матриць A, B на блоки так, щоб загальна кількість блоків матриці дорівнювала кількості процесорів у системі, і для кожного процесора призначати обчислення одного блоку матриці *C*.

Розглянуті **задачі** обчислення **матрично-векторного добутку** і задача **множення матриць** мають чітко виражений векторний характер і для їх розв'язання більш доцільним є використання векторноконвеєрних і векторнопаралельних обчислювальних систем.

4. Метод Якобі для розв'язання СЛАР. Нехай *A* – невироджена матриця розмірності $n \times n$ з відмінними від нуля діагональними елементами. Якщо цю матрицю подати у вигляді різниці її діагональної частини:

$$D = diag(A) = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$$

і позадіагональної частини з протилежним знаком:

$$B = D - A = \begin{pmatrix} 0 & -a_{12} & \dots & -a_{1n} \\ -a_{21} & 0 & \dots & -a_{2n} \\ \dots & \dots & \dots & \dots \\ -a_{n1} & -a_{n2} & \dots & 0 \end{pmatrix},$$

то ітераційна формула методу Якобі для розв'язання СЛАР

$$A\vec{x} = \vec{b}, \qquad (1)$$

де $\vec{x}, \vec{b}$ – вектори розмірності n, має вигляд:

$$\vec{x}^{(r+1)} = H\vec{x}^r + \vec{d}, \quad r = 0, 1, 2, \ldots, \tag{2}$$

де $H = D^{-1}B$ – матриця $n \times n$, а $\vec{d} = D^{-1}\vec{b}$ – вектор розміру n, $\vec{x}^0$ – початкове наближення до розв'язку системи (1).

Нам буде потрібне подання (2) у вигляді

$$x_i^{r+1} = \vec{h}_i \cdot \vec{x}^r + d_i, \quad r = 0, 1, 2, \ldots,$$

де x_i^{r+1}, d_i – i-ті компоненти відповідно векторів $\vec{x}^{(r+1)}, \vec{d}$; $\vec{h}_i \cdot \vec{x}^r$ – скалярний добуток рядка $\vec{h}_i$ матриці H на вектор $\vec{x}^r$.

Умовою закінчення ітерації в (2) зазвичай є:

$$\left\| \vec{b} - A\vec{x}^r \right\| \le \varepsilon, \tag{3}$$

де $\vec{b} - A\vec{x}^r$ – нев'язка для розв'язку СЛАР (1) на r-й ітерації, ε – точність розв'язання, що вимагається. Будемо вважати, що тут використовується евклідова норма:

$$\left\| \vec{b} - A\vec{x}^r \right\| = \sqrt{\sum_{i=1}^{n} (b_i - \vec{a}_i \vec{x}^r)^2},$$

де $\vec{a}_i$ – i-й рядок матриці A.

Для простоти будемо вважати, що кількість рядків і стовпців n матриці A є кратною до кількості процесорів N у системі та $p = n/N$. Оскільки обчислення компонент матриці $H = D^{-1}B$ і вектора $\vec{d} = D^{-1}\vec{b}$ необхідно виконати лише один раз, то основною операцією в (2) є операція матрично-векторного добутку $H\vec{x}^r$.

Розглянемо схему розпаралелювання методу Якобі, використовуючи скалярні добутки:

1) розподіляємо по процесорах елементи матриці A і компоненти векторів $\vec{b}, \vec{x}^0$ так, як показано на рис. 4;

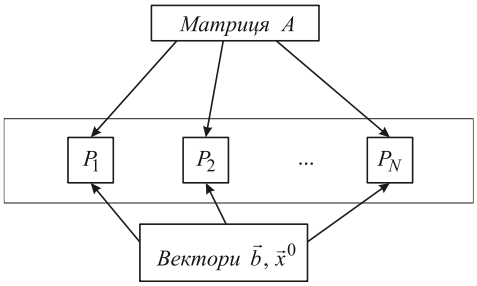

Рис. 4

2) паралельно на всіх процесорах системи обчислюємо відповідні рядки $\vec{h}_i$ матриці H, а також відповідні елементи d_i вектора $\vec{d}$:
   - на процесорі P_1 рядки $\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_p$ і елементи $d_1, d_2, \ldots, d_p$;
   - на процесорі P_2 рядки $\vec{h}_{p+1}, \vec{h}_{p+2}, \ldots, \vec{h}_{2p}$ і елементи $d_{p+1}, d_{p+2}, \ldots, d_{2p}$;

   ……

- на процесорі P_N рядки $\vec{h}_{(N-1)p+1}, \vec{h}_{(N-1)p+2}, \ldots, \vec{h}_{Np}$ і елементи $d_{(N-1)p+1}, d_{(N-1)p+2}, \ldots, d_{Np}$;

3) паралельно на всіх процесорах системи виконуємо наступну ітерацію:

а) обчислюємо відповідні компоненти $\vec{x}^{(r+1)}$:

- на процесорі P_1 компоненти $x_1^{r+1}, x_2^{r+1}, \ldots, x_p^{r+1}$;
- на процесорі P_2 компоненти $x_{p+1}^{r+1}, x_{p+2}^{r+1}, \ldots, x_{2p}^{r+1}$;

……

- на процесорі P_N компоненти $x_{(N-1)p+1}^{r+1}, x_{(N-1)p+2}^{r+1}, \ldots, x_{Np}^{r+1}$;

б) передаємо з кожного із процесорів $P_1, P_2, \ldots, P_N$ одержані компоненти вектора $\vec{x}^{(r+1)}$ кожному із решти процесорів (потрібно мати весь вектор попереднього наближення на кожному із процесорних елементів);

в) обчислюємо відповідні суми в нормі (перевірка точності):

- на процесорі P_1 суму $\sum_{i=1}^{p}(b_i - \vec{a}_i\vec{x}^{(r+1)})^2$;
- на процесорі P_2 суму $\sum_{i=p+1}^{2p}(b_i - \vec{a}_i\vec{x}^{(r+1)})^2$;

……

- на процесорі P_N суму $\sum_{i=(N-1)p+1}^{Np}(b_i - \vec{a}_i\vec{x}^{(r+1)})^2$;

г) передаємо з кожного із процесорів $P_1, P_2, \ldots, P_N$ обчислені на в) суми на один із процесорів системи;

4) обчислюємо на вказаному процесорі значення норми (виконуємо додавання та беремо корінь квадратний) і перевіряємо виконання умови (3);
5) якщо умова (3) виконана, то закінчуємо обчислення, інакше, переходимо на пункт 3) для обчислення наступної ітерації.

У розглянутій паралельній схемі **методу Якобі** на кожній ітерації процесор P_i $(i = \overline{1, N})$ повинен, **по-перше**, передати всім решті процесорів системи свою частину вектора $\vec{x}^{(r+1)}$ і, **по-друге**, до одержання від цих процесорів їх частин вектора $\vec{x}^{(r+1)}$ не може продовжити ітерації. Тобто усі процесори повинні виконувати синхронні ітерації. Метод Якобі може збігатися і під час виконання асинхронних ітерацій, коли процесор P_i починає наступну ітерацію до одержання всіх нових компонент вектора $\vec{x}$. Тобто замість не одержаних компонент вектора $\vec{x}^{(r+1)}$ можна використовувати значення цих компонент, одержані на попередній ітерації.