МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Волинський національний університет імені Лесі Українки

Кафедра комп'ютерних наук та кібербезпеки

Гришанович Т. О.

ЛАБОРАТОРНИЙ ПРАКТИКУМ ІЗ ДИСЦИПЛІНИ "ПАРАЛЕЛЬНІ ТА РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ"

для студентів спеціальності 122 Комп'ютерні науки
першого (бакалаврського) рівня

Луцьк 2022

УДК 004(076)

Рекомендовано до видання науково-методичною радою
Волинського національного університету імені Лесі Українки
(протокол №3 від 16 листопада 2022 р.)

Рецензенти:
Хомяк М. Я. – кандидат фізико-математичних наук, доцент, завідувач кафедри загальної математики та методики навчання інформатики Волинського національного університету імені Лесі Українки;
Багнюк Н. В. – кандидат технічних наук, доцент кафедри комп'ютерної інженерії та кібербезпеки Луцького національного технічного університету.

Г 85 Гришанович Т. О. Лабораторний практикум із дисципліни "Паралельні та розподілені обчислення" [Електронний ресурс]; ВНУ імені Лесі Українки. Електронні текстові дані (1 файл: 728 КБ). Луцьк : ВНУ імені Лесі Українки, 2022. 50 с.

Лабораторний практикум містить інструкції до виконання лабораторних робіт із дисципліни «Паралельні та розподілені обчислення». Для кожної роботи наведено тему та мету, на прикладі продемонстровано хід її виконання, наведено зразки вхідних/вихідних даних.

Практикум також містить ряд завдань для індивідуального виконання. Лабораторний практикум призначений для здобувачів першого (бакалаврського) ступеня, що навчаються за спеціальністю 122 Комп'ютерні науки.

Зміст

Лабораторна робота №1

“Визначення характеристик систем функціональних пристроїв”

Мета: вироблення умінь та навичок оцінювати характеристики систем обчислювальних пристроїв, які працюють у різних режимах роботи.

Хід роботи

Приклад. Граф системи функціональних пристроїв (ФП) наведений на рис. 1. Відомі пікові продуктивності пристроїв системи: π_1 =10, π_2 = 5, π_3 =8, π_4 = 6, π_5 = 7, π_6 =9, π_7 =12, π_8 =8, π_9 =10, π_{10} = 4, π_{11} = 6, π_{12} = 4, π_{13} = 6.

Знайти:

1) завантаженості усіх пристроїв системи;
2) реальну продуктивність системи;
3) завантаженість системи;
4) прискорення системи.

Розв’язування. Як видно з рис. 1, система складається з трьох незалежних підсистем. Згідно до 1-го закону Амдала реальна продуктивність кожної із підсистем визначається продуктивністю найменш продуктивного пристрою цієї підсистеми.

1) Нехай $\pi^{(i)}$ — реальні продуктивності, з якими працюють усі пристрої i-системи, i =1, 2, 3. Тоді

   $\pi^{(1)} = \min \{\pi_1, ..., \pi_5\} = 5$, $\pi^{(2)} = \min \{\pi_6, ..., \pi_9\} = 8$,

   $\pi^{(3)} = \min \{\pi_{10}, ..., \pi_{13}\} = 4$.

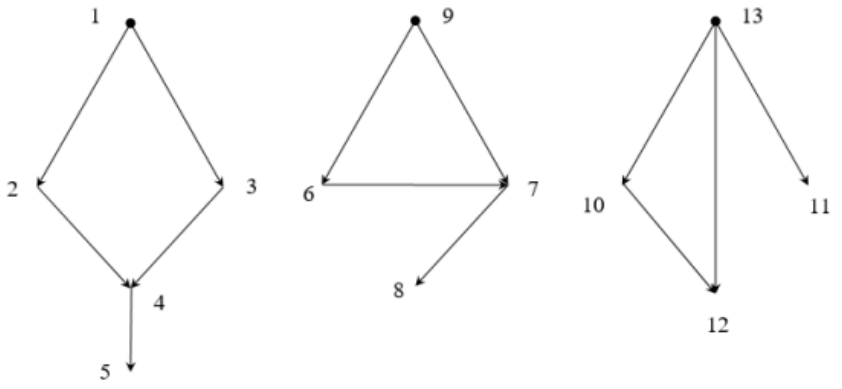

Рисунок 1 - Граф системи функціональних пристроїв”

Завантаженості p_i пристроїв першої підсистеми рівні $\pi^{(1)}/\pi_i$, (i = 1,...5):

$$p_1 = \frac{5}{10},\ p_2 = 1,\ p_3 = \frac{5}{8},\ p_4 = \frac{5}{6},\ p_5 = \frac{5}{7}.$$

Завантаженості p_i пристроїв другої підсистеми рівні $\pi^{(2)}/\pi_i$, (i = 6, ... 9):

$$p_6 = \frac{8}{9},\ p_7 = \frac{2}{3},\ p_8 = 1,\ p_9 = \frac{4}{5}.$$

Завантаженості p_i пристроїв третьої підсистеми рівні $\pi^{(3)}/\pi_i$, (i = 10, ... 13):

$$p_{10} = 1,\ p_{11} = \frac{2}{3},\ p_{12} = 1,\ p_{13} = \frac{2}{3}.$$

2) $r = r^{(1)} + r^{(2)} + r^{(3)}$, де $r^{(i)}$ - реальна продуктивність i-ї підсистеми, i =1, 2, 3. За першим законом Амдала $r^{(i)} = l_i \cdot \pi^{(i)}$, де l_i кількість пристроїв i-ї підсистеми. Тому $r^{(1)} = 5 \cdot 5 = 25$, $r^{(2)} = 4 \cdot 8 = 32$, $r^{(3)} = 4 \cdot 4 = 16$. Отже, $r = 25 + 32 + 16 = 73$.

3) Для знаходження завантаженості системи використаємо формулу $r = \pi \cdot p$, де p - завантаженість, π - пікова продуктивність системи.

Тоді $\pi = \pi_1 + ... + \pi_{13} = 10+5+8+6+7+9+12+8+10+4+6+4+6 = 95$. Тому $p = r/\pi = 73 / 95 \approx 0,77$.

Скористаємося формулою

$$S = r / \max_{1 \le i \le l} \pi_i .$$

Тоді $S = 73 / 12 \approx 6,08$.

Завдання для індивідуального виконання

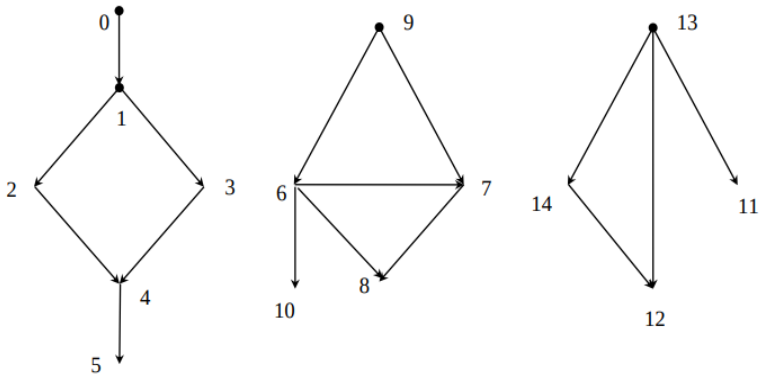

Рисунок 2 - Граф системи функціональних пристроїв 1

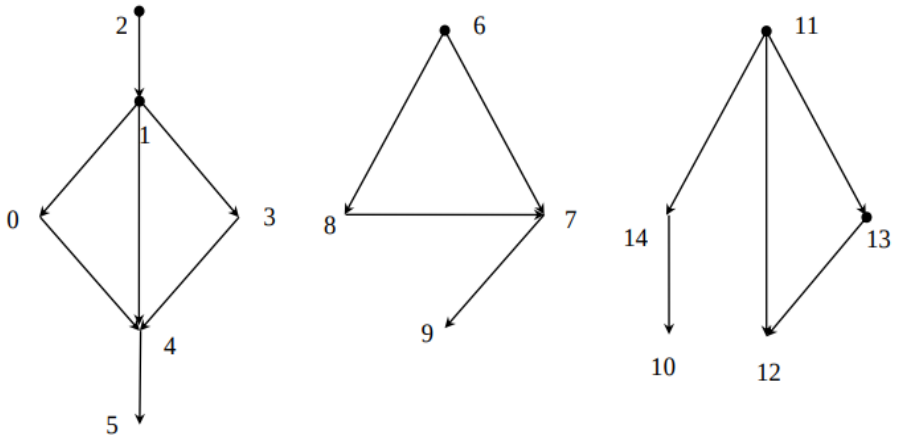

Рисунок 3 - Граф системи функціональних пристроїв 2

Варіант 1. Граф системи ФП 1 наведений на рис. 2. Відомі продуктивності пристроїв системи:

$\pi_0 = 5$, $\pi_1 = 8$, $\pi_2 = 4$, $\pi_3 = 8$, $\pi_4 = 7$, $\pi_5 = 6$, $\pi_6 = 9$, $\pi_7 = 12$, $\pi_8 = 9$, $\pi_9 = 15$, $\pi_{10} = 4$, $\pi_{11} = 8$, $\pi_{12} = 10$, $\pi_{13} = 8$, $\pi_{14} = 11$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Варіант 2. Граф системи ФП 2 наведений на рис. 3. Відомі продуктивності пристроїв системи:

$\pi_0 = 7$, $\pi_1 = 5$, $\pi_2 = 10$, $\pi_3 = 8$, $\pi_4 = 6$, $\pi_5 = 6$, $\pi_6 = 9$, $\pi_7 = 12$, $\pi_8 = 8$, $\pi_9 = 5$, $\pi_{10} = 10$, $\pi_{11} = 8$, $\pi_{12} = 7$, $\pi_{13} = \pi_{14} = 8$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Варіант 3. Граф системи ФП 1 наведений на рис. 2. Відомі продуктивності пристроїв системи:

$\pi_0 = 15$, $\pi_1 = 9$, $\pi_2 = 4$, $\pi_3 = 8$, $\pi_4 = 7$, $\pi_5 = 6$, $\pi_6 = 9$, $\pi_7 = 12$, $\pi_8 = 8$, $\pi_9 = 12$, $\pi_{10} = 7$, $\pi_{11} = 8$, $\pi_{12} = 10$, $\pi_{13} = 8$, $\pi_{14} = 11$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Варіант 4. Граф системи ФП 2 наведений на рис. 3. Відомі продуктивності пристроїв системи:

$\pi_0 = 9$, $\pi_1 = 6$, $\pi_2 = 12$, $\pi_3 = 7$, $\pi_4 = 5$, $\pi_5 = 8$, $\pi_6 = 3$, $\pi_7 = 10$, $\pi_8 = 9$, $\pi_9 = 7$, $\pi_{10} = 11$, $\pi_{11} = 9$, $\pi_{12} = 7$, $\pi_{13} = \pi_{14} = 8$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Варіант 5. Граф системи ФП 2 наведений на рис. 3. Відомі продуктивності пристроїв системи:

$\pi_0 = 9$, $\pi_1 = 8$, $\pi_2 = 5$, $\pi_3 = 8$, $\pi_4 = 7$, $\pi_5 = 6$, $\pi_6 = 9$, $\pi_7 = 12$, $\pi_8 = 9$, $\pi_9 = 15$, $\pi_{10} = 7$, $\pi_{11} = 8$, $\pi_{12} = 10$, $\pi_{13} = 8$, $\pi_{14} = 11$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Варіант 6. Граф системи ФП 1 наведений на рис. 2. Відомі продуктивності пристроїв системи:

$\pi_0 = 4$, $\pi_1 = 3$, $\pi_2 = 10$, $\pi_3 = 8$, $\pi_4 = 2$, $\pi_5 = 4$, $\pi_6 = 9$, $\pi_7 = 3$, $\pi_8 = 8$, $\pi_9 = 9$, $\pi_{10} = 11$, $\pi_{11} = 8$, $\pi_{12} = 6$, $\pi_{13} = \pi_{14} = 8$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Варіант 7. Граф системи ФП 1 наведений на рис. 2. Відомі продуктивності пристроїв системи:

$\pi_0 = 7$, $\pi_1 = 5$, $\pi_2 = 4$, $\pi_3 = 7$, $\pi_4 = 5$, $\pi_5 = 8$, $\pi_6 = 7$, $\pi_7 = 12$, $\pi_8 = 9$, $\pi_9 = 5$, $\pi_{10} = 14$, $\pi_{11} = 8$, $\pi_{12} = 10$, $\pi_{13} = 8$, $\pi_{14} = 11$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Варіант 8. Граф системи ФП наведений на рис. 3. Відомі продуктивності пристроїв системи:

$\pi_0 = 7$, $\pi_1 = 5$, $\pi_2 = 10$, $\pi_3 = 8$, $\pi_4 = 6$, $\pi_5 = 6$, $\pi_6 = 9$, $\pi_7 = 12$, $\pi_8 = 8$, $\pi_9 = 5$, $\pi_{10} = 11$, $\pi_{11} = 8$, $\pi_{12} = 7$, $\pi_{13} = \pi_{14} = 8$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Варіант 9. Граф системи ФП наведений на рис. 3. Відомі продуктивності пристроїв системи:

$\pi_0 = 5$, $\pi_1 = 8$, $\pi_2 = 4$, $\pi_3 = 8$, $\pi_4 = 7$, $\pi_5 = 6$, $\pi_6 = 9$, $\pi_7 = 12$, $\pi_8 = 9$, $\pi_9 = 15$, $\pi_{10} = 4$, $\pi_{11} = 8$, $\pi_{12} = 10$, $\pi_{13} = 8$, $\pi_{14} = 11$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Варіант 10. Граф системи ФП наведений на рис. 3. Відомі продуктивності пристроїв системи:

$\pi_0 = 5$, $\pi_1 = 10$, $\pi_2 = 8$, $\pi_3 = 6$, $\pi_4 = 7$, $\pi_5 = 6$, $\pi_6 = 9$, $\pi_7 = 12$, $\pi_8 = 8$, $\pi_9 = 5$, $\pi_{10} = 11$, $\pi_{11} = 8$, $\pi_{12} = 7$, $\pi_{13} = \pi_{14} = 8$.

Визначити:

1. завантаженості усіх пристроїв системи;
2. завантаженість системи;
3. реальну продуктивність системи;
4. прискорення системи.

Лабораторна робота №2

“Визначення максимально можливого прискорення і ефективності системи”

Мета: навчитись обчислювати максимально можливе прискорення для системи та її ефективність, використовуючи закони Амдаля.

Приклад. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 4 (вершини графу відповідають операціям).

Рисунок 4 - Граф системи

Розв’язок. Використаємо 2-й закон Амдала. З рис. 4 можна зробити висновок, що $N = 15$, $n = 6$. Звідси $\beta = 6 / 15 = 2 / 5$. Ширина алгоритму - 3. Тому

$$S_3 = \frac{3}{3 \cdot 2/5 + (1 - 2/5)} = \frac{3 \cdot 5}{9} = \frac{5}{3}, \quad E_3 = \frac{S_3}{3} = \frac{5}{9} \approx 55{,}56\%.$$

Завдання для індивідуального виконання

Рисунок 5 - Граф системи 1

Рисунок 6 - Граф системи 2

Варіант 1. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 5

Варіант 2. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 6

Варіант 3. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 5

Варіант 4. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 6

Варіант 5. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 5

Варіант 6. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 6

Варіант 7. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 5

Варіант 8. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 6

Варіант 9. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 5

Варіант 10. Визначити максимальне можливе прискорення і ефективність системи, яка складається з однакових пристроїв і призначена для реалізації алгоритму, граф якого наведений на рис. 6

Лабораторна робота №3

“Визначення максимального прискорення для системи”

Мета: Визначити максимальне прискорення для системи за умови використання заданої кількості процесорів та задовольнити умови мінімального граничного прискорення.

Приклад. Нехай у алгоритмі паралельні обчислення складають 5/6. Визначити:

1) Максимальне можливе прискорення у випадку використання 5 однакових універсальних процесорів;
2) Мінімальну кількість процесорів, використання яких може забезпечити 85% граничного прискорення.

Розв’язок. Визначимо частку послідовних обчислень: $\beta = 1 - 5/6 = 1/6$.

1) Скористаємося 2-м законом Амдала:

$$S_5 = \frac{5}{5 \cdot 1/6 + 1 - 1/6} = \frac{5 \cdot 6}{10} = 3.$$

2) Використаємо 3-й закон Амдала:

$$S_l \geq 0{,}8 \cdot S_{\max},$$

$$\frac{l}{l/6 + 5/6} \geq \frac{1}{1/6},$$

$$\frac{6l}{l+5} \geq 0{,}85 \cdot 6,$$

$$l \geq 0{,}85 \cdot (l+5),$$

$$0{,}15l \geq 4{,}25,$$

$$l \geq 85/3 \approx 28{,}33$$

Відповідь: 29.

Завдання для індивідуального виконання

Варіант 1. Нехай у деякому алгоритмі послідовні обчислення складають 1/4. Визначити:

1. максимальне можливе прискорення у випадку використання l = 6 однакових універсальних процесорів;

2. кількість процесорів, використання яких може забезпечити від 60 до 95% від максимально можливого прискорення.

Варіант 2. Нехай у деякому алгоритмі паралельні обчислення складають 5/6.

Визначити:

1. максимальне можливе прискорення у випадку використання $l = 10$ однакових універсальних процесорів;
2. найменшу кількість процесорів, використання яких може забезпечити 90% максимально можливого прискорення.

Варіант 3. Нехай у деякому алгоритмі послідовні обчислення складають 30%.

Визначити:

1. максимальне можливе прискорення у випадку використання $l = 7$ однакових універсальних процесорів;
2. найбільшу кількість процесорів, при використанні яких ефективність системи не менша за 15%.

Варіант 4. Нехай у деякому алгоритмі послідовні обчислення складають 25%.

Визначити:

1. максимальне можливе прискорення у випадку використання $l = 8$ однакових універсальних процесорів;
2. найменшу кількість процесорів, використання яких може забезпечити 90% максимально можливого прискорення.

Варіант № 5. Нехай у деякому алгоритмі паралельні обчислення складають 60%.

Визначити:

1. максимальне можливе прискорення у випадку використання $l = 20$ однакових універсальних процесорів;
2. кількість процесорів, використання яких може забезпечити від 70 до 85% від максимально можливого прискорення.

Варіант № 6. Нехай у деякому алгоритмі послідовні обчислення складають 1/5.

Визначити:

1. максимальне можливе прискорення у випадку використання $l = 9$ однакових універсальних процесорів;
2. наскільки відсотків прискориться алгоритм, якщо кількість процесорів зросте з 5 до 30.

Варіант № 7. Нехай у деякому алгоритмі паралельні обчислення складають 80%.

Визначити:

1. максимальне можливе прискорення у випадку використання l =10 однакових універсальних процесорів;
2. кількість процесорів, використання яких може забезпечити від 50% до 80% від максимально можливого прискорення.

Варіант № 8. Нехай для деякого алгоритму максимальне прискорення рівне 4.

Визначити:

1. максимальне можливе прискорення у випадку використання s = 3 однакових універсальних процесорів;
2. кількість процесорів, використання яких може забезпечити від 90% до 95% максимально можливого прискорення.

Варіант № 9. Нехай максимальне можливе прискорення алгоритму у випадку 10 універсальних процесорів рівне 3. Визначити:

1. максимальне можливе прискорення у випадку використання 20 однакових універсальних процесорів;
2. кількість процесорів, використання яких може забезпечити від 50 до 60% від максимально можливого прискорення.

Варіант № 10. Нехай у деякому алгоритмі паралельні обчислення складають 3/4.

Визначити:

1. максимальне можливе прискорення у випадку використання l = 11 однакових універсальних процесорів;
2. найменшу кількість процесорів, використання яких може забезпечити 92% максимально можливого прискорення.

Лабораторна робота №4
“Концепції паралелізму”

Мета: засвоєння основних концепцій паралелізму, вироблення навичок побудови та аналізу паралельних форм обчислювальних алгоритмів.

Зауваження: у кожному варіанті потрібно зобразити 3 графи алгоритму (для кожної системи окремо). У кожному випадку вказати висоту і ширину паралельної форми алгоритму та обчислити прискорення і ефективність реалізації алгоритму за формулами

$$S_l(n)=\frac{T_1(n)}{T_l(n)},\quad E_l(n)=\frac{S_l(n)}{l},$$

де $T_1(n)$ - час, за який можна реалізувати алгоритм на одному процесорі, $T_l(n)$ - час реалізації алгоритму в системі з l процесорів (висота алгоритму), n - розмірність задачі, $S_l(n)$ - прискорення реалізації алгоритму на цій системі, $E_l(n)$ - ефективність реалізації алгоритму у паралельній системі (завантаженість системи). У третьому завданні визначити мінімальну кількість процесорів, які забезпечують досягнення максималь-ного можливого прискорення при паралельній реалізації алгоритму.

Приклад. Зобразити граф алгоритму паралельного обчислення значення виразу

$$a_1a_2 + a_2\,a_3 + a_3\,a_4 + a_4\,a_5 + a_5\,a_6 + a_6\,a_7 + a_7\,a_1$$

на обчислювальному пристрої

1) із трьома універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Для кожної паралельної форми обчислити її висоту, ширину, прискорення та ефективність реалізації алгоритму.

Розв’язок. Будемо вважати, що у алгоритмі спочатку обчислюються зліва направо усі 7 добутків, а потім - 6 сум (у такому самому порядку). Розглянемо спочатку реалізацію алгоритму у послідовній системі. Відповідний граф наведено на рис 7. Запишемо відповідні характеристики: $n = 7$, $l = 1$, $T_1(7) = 13$, $S_1(7) =1$, $E_1(7) = 1$.

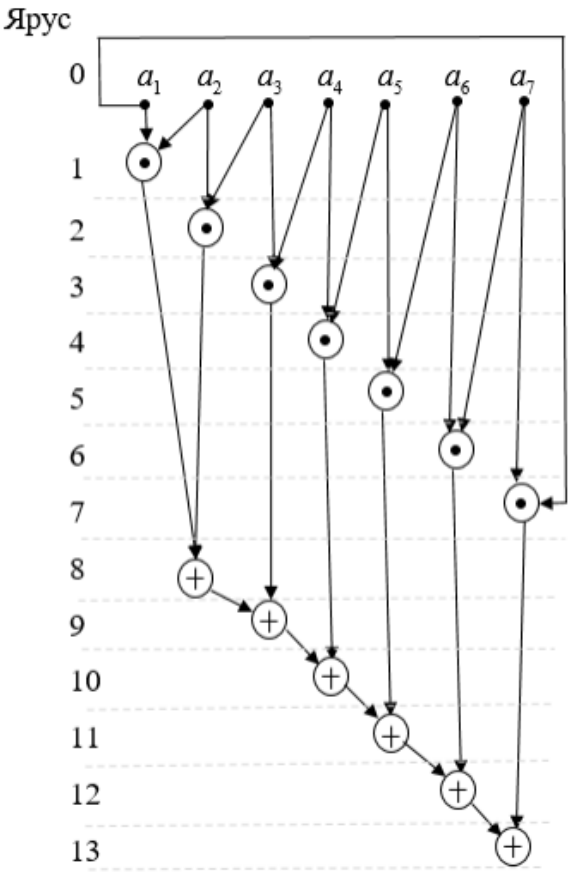

Рис. 7. - Граф алгоритму у випадку $l = 11$

Розглянемо випадок системи з трьома процесорами. Відповідний граф алгоритму ширини 3 наведено на рис. 8.

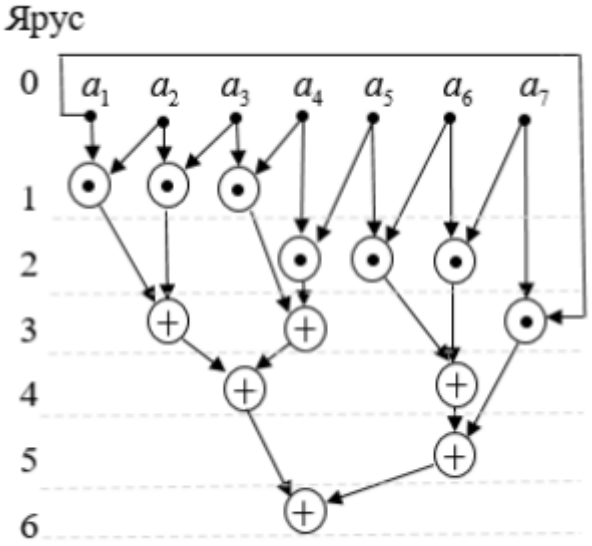

Рис. 8. - Граф алгоритму у випадку $l = 3$

Тоді

$$n=7,\ l=3,\ T_3(7)=6,\ S_3(7)=\frac{13}{6}\approx 2{,}17,\ E_3(7)=\frac{13}{6}:3=\frac{13}{18}\approx 0{,}72.$$

Розглянемо реалізацію алгоритму у випадку системи, кількість процесорів якої необмежена. Оскільки у алгоритмі операції додавання не можуть виконуватися раніше, ніж відповідні доданки-множники будуть обчисленні, то для висоти алгоритму $T_l(7)$ справджується оцінка

$$T_l(7)\le 1+\lceil \log_2 7\rceil = 4$$

У випадку $l = 6$ можна вказати алгоритм висоти 4, граф якого наведено на рис. 9.

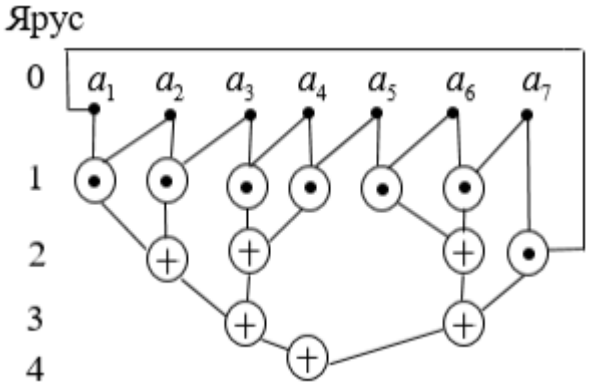

Рис. 9 - Граф алгоритму у випадку $l = 6$

Для цієї паралельної форми

$$l=6,\ T_6(7)=4,\ S_6(7)=\frac{13}{4}=3{,}25,\ E_6(7)=\frac{13}{4}:6=\frac{13}{24}\approx \quad 0{,}54\ .$$

Можна показати, що якщо $l < 6$, то $T_1(7) > 4$. Дійсно, із специфіки операцій алгоритму випливає, що на останньому ярусі виконується тільки одна операція, на передстанньому - не більше двох операцій, на третьому знизу - не більше чотирьох. Тому загалом на трьох нижніх ярусах може виконуватися не більше 7 операцій. Тому для попередніх ярусів залишається не менше, ніж 13 − 7 = 6 операцій, які у випадку $l < 6$ не можуть бути виконані на одному ярусі. Тому кількість ярусів має бути не меншою за 5.

Завдання для індивідуального виконання

Варіант 1. Зобразити граф алгоритму знаходження скалярного добутку двох 10-вимірних векторів в обчислювальній системі з:

1) чотирма універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Варіант 2. Зобразити граф алгоритму паралельного обчислення значення виразу $a_1 + a_1 a_2 + + a_1 a_2 a_3 + ... + a_1 a_2 a_7 a_8$ в обчислювальній системі з:

1) трьома універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Варіант 3. Задана прямокутна числова матриця 3×4. Зобразити граф алгоритму знаходження суми $s_{12} + s_{13} + s_{23}$, де s_{ij} - скалярний добуток i-го та j-го рядків матриці, в обчислювальній системі із:

1) п’ятьма універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Варіант 4. Зобразити граф алгоритму паралельного обчислення значення виразу $a_1 (a_1 + a_2)\times(a_1 + a_2 + a_3)(a_1 + a_2 + + a_{10})$ в обчислювальній системі із:

1) чотирма універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Варіант 5. Побудувати граф алгоритму паралельного обчислення значення виразу

$$a_1 + a_1^2 a_2 + a_1^4 a_2^2 a_3 + ... + a_1^{32} a_2^{16} \ldots a_5^2 a_6$$

в обчислювальній системі із:

1) трьома універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Варіант 6. Побудувати граф алгоритму обчислення значення многочлена шостого степеня у точці за схемою Горнера в обчислювальній системі із:

1) двома універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Варіант 7. Побудувати граф алгоритму паралельного обчислення суми квадратів елементів 16-вимірного вектора в обчислювальній системі із:

1) чотирма універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Варіант 8. Зобразити граф алгоритму паралельного обчислення значення виразу $a_1 + a_1 \times \times (a_1 + a_2) + ... + a_1 (a_1 + a_2) \cdot \cdot (a_1 + a_2 + + a_6)$ в обчислювальній системі із:

1) трьома універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Варіант 9. Зобразити граф алгоритму знаходження добутку двох 3×3-матриць в обчислювальній системі із:

1) п’ятьма універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Варіант 10. Задана прямокутна числова матриця розмірності 3×5. Зобразити граф алгоритму знаходження суми $p_1 + p_2 + p_3$, де p_i — добуток елементів i-го рядка матриці, в обчислювальній системі із:

1) чотирма універсальними процесорами;
2) в умовах концепції необмеженого паралелізму.

Лабораторна робота №5
“Паралельні обчислення з використанням багатопотокового програмування”

Мета та завдання роботи: вироблення навичок реалізації обчислювальних алгоритмів з використанням засобів багатопотокового програмування.

Зауваження: У кожному варіанті потрібно написати багатопотокову версію програми на одній із сучасних мов програмування (Java, C++, C#, Scala) та виміряти її прискорення на кількох тестових прикладах.

Приклад. Написати програму обчислення квадрату довжини *n*-вимірного вектора із використанням розпаралелення обчислень.

Розв’язок. Нижче наведено багатопотокову програму на мові Java. Для багатопотокових обчислень використовуються об’єкти класу Adder, які реалізовують інтерфейс Runnable шляхом перевизначення методу Run().

```
class Adder implements Runnable {
    double[] array;
    int from;

    int to;
    private double result;

    public Adder(double[] array, int from,
        int to) { this.array = array;
        this.from = from;
        this.to = to;

    }

  public void run() {

       result = 0;
        for (int i = from; i < to; i++) {
            result += array[i]*array[i];
        }

    }

    public double getResult(){

        return result;
```

```
    }
}

public class MyClass {

    public static void main(String[]
        args){ final int N =
        60_000_000;
        final int ThreadCount = 4;

        double[] array = new double[N];
        for (int i = 0; i < N; i++) {
            array[i] = Math.random();

        }

        int portionSize = N / ThreadCount;
        Adder[] adders = new
        Adder[ThreadCount]; Thread[] threads
        = new Thread[ThreadCount]; long
        start = System.nanoTime();

        for (int i = 0; i < ThreadCount; i++) {

            adders[i] = new Adder(array, i*portionSize,
            (i+1)*portionSize); threads[i] = new
            Thread(adders[i]); threads[i].start();
        }

        double result = 0;

        for(int i = ThreadCount*portionSize; i <
            N; i++) { result += array[i];
        }

        for(int i = 0; i < ThreadCount;
            i++) { try{
                threads[i].join();

            }

            catch (InterruptedException e){
                System.out.println("an error
                occured");
            }
```

```
        }

        for(int i = 0; i < ThreadCount;
            i++) { result +=
            adders[i].getResult();

        }

        long duration1 = System.nanoTime() - start;
        System.out.printf("Length is %g. Duration in
        parallel mode is
%d", result, duration1);

        start = System.nanoTime();
        result = 0;
        for (int i = 0; i < N; i++) {
            result += array[i]*array[i];

        }

        long duration2 = System.nanoTime() - start;
        System.out.printf("\n Length is %g. Duration in
        serial mode is

%d\n", result, duration2);

        System.out.printf("Speed-up is %g", (double)duration2
        / duration1);
    }
}
```

Завдання для індивідуального виконання

Варіант 1. Задана прямокутна числова матриця $m \times n$. Написати паралельну програму знаходження пари рядків, скалярний добуток яких найменший

Варіант 2. Написати програму для паралельного пошуку моди варіаційного ряду довжини n (значення, яке найчастіше зустрічається серед елементів числового масиву).

Варіант 3. Задана квадратна числова $n \times n$ -матриця $A = (a_{ij})$. Написати паралельну програму знаходження кількості пар індексів i та j, для яких величина

$S(i,j) = \sum_{k=0}^{n-1} a_{ki} a_{j,n-k-1}, \left(i, j \in \overline{0, n-1}\right)$ приймає від'ємне значення.

Варіант 4. Написати програму для паралельного множення послідовності перестановок.

Варіант 5. Написати паралельну програму для знаходження кількості спільних елементів двох числових масивів (значень, які зустрічаються в обох масивах).

Варіант 6. Задана прямокутна числова матриця $m \times n$. Написати паралельну програму знаходження пар стовпців матриці, манхетенська відстань між якими є найбільшою.

Варіант 7. Вагою елемента матриці A назвемо суму відмінних від нього елементів матриці, які містяться у одному рядку чи одному стовпці з ним. Написати паралельну програму для знаходження елементів найбільшої ваги.

Варіант 8. Задана квадратна числова матриця A. Назвемо порядком елемента кількість відмінних від нуля елементів, які розташовані на діагоналях матриці, на перетині яких міститься цей елемент. Написати паралельну програму для знаходження суми елементів найбільшого порядку.

Варіант 9. Написати паралельну версію програми знаходження кількості входжень однієї послідовності у іншу послідовність (вважати, що послідовності - одновимірні масиви).

Варіант 10. Написати паралельну версію програми знаходження такої $k \times k$ підматриці B заданої числової $n \times n$ матриці A, сума усіх елементів якої найбільша (підматриця має бути розташована на перетині k послідовних рядків та стовпчиків матриці A).

Лабораторна робота №6

“Технологія OpenMP”

Мета та завдання роботи: ознайомлення із можливостями технології OpenMP та використання OpenMP в межах концепції внутрішнього паралелізму.

Зауваження: У кожному варіанті потрібно виконати два завдання із використанням технології OpenMP.

Перше завдання - програмна реалізація паралельного алгоритму із 2-го завдання відповідного варіанта лабораторної роботи №4.

Друге завдання - розробка паралельної програми. Потрібно побудувати графіки залежності прискорення реалізації алгоритму $S_m(n)$ від розмірності задачі у випадку використання m потоків ($m \in \{2, 4, 6,8,10, 20\}$).

Приклад. З використанням OpenMP написати програму обчислення матричної норми Нижче наведено розв’язок, розроблений з використанням парадигми ітеративного паралелізму.

```
#include "stdafx.h"
#include<stdio.h>
#include<omp.h>

#include<iostream>
#include<fstream>
using namespace std;

double norm_1(double** matrix, int m,
    int n) { double norm = -1;
    for (int j = 0; j < n; j++) {

        double sum = 0;
        for (int i = 0; i < m; i++)

            sum += matrix[i][j]>=0 ? matrix[i][j] :
        -matrix[i][j]; if (sum > norm) norm = sum;
    }
    return norm;
}
```

```
double norm_1_parallel(double** matrix, int m, int n,
   int thread_count) { double norm = -1;
#pragma omp parallel for num_threads(thread_count)

   for (int j = 0; j < n; j++) {
      double sum = 0;
    for (int i = 0; i < m; i++)

         sum += matrix[i][j] >= 0 ? matrix[i][j] :
      -matrix[i][j]; if (sum > norm)

#pragma omp critical

      {
         if(sum > norm) norm = sum;
      }
   }

   return norm;
}

int main()
{
   const int DIM_COUNT = 7;

   int dims[DIM_COUNT] = { 100, 500, 1000, 2000, 3000,
   5000, 10000}; const int THREADS_COUNT = 6;

   int threads[THREADS_COUNT] = { 2, 4, 6, 8,
   10, 20 }; double** matrix = new
   double*[dims[DIM_COUNT-1]]; cout <<
   "Initialization...\n";

   for (int i = 0; i < dims[DIM_COUNT - 1];
      i++) { matrix[i] = new
      double[dims[DIM_COUNT - 1]]; for (size_t
      j = 0; j < dims[DIM_COUNT - 1]; j++)

         matrix[i][j] = 2*(double)rand() / (RAND_MAX + 1) -
         1;
   }

   fstream fs = fstream("d:\\result.txt",
   ios::out); for (int d = 0; d < DIM_COUNT;
   d++) {
      int m = dims[d], n = dims[d];
```

```
        double start = omp_get_wtime();
        double norm = norm_1(matrix, m, n);

        double duration = omp_get_wtime() -
        start; cout << "Dimension: " << m
        << '\n';

        printf("\tIn serial mode ||A||_1 = %e; duration is
        %e\n", norm, duration); for (int t = 0; t <
        THREADS_COUNT; t++) {

            start = omp_get_wtime();

            norm = norm_1_parallel(matrix, m, n, threads[t]);
            double speedup = duration / (omp_get_wtime() -
            start); printf("\tFor %d threads ||A||_1 = %e;
            speedup is %e\n",
threads[t], norm, speedup);
            fs << speedup << '\t';
        }

        fs << '\n';
    }
    fs.close();

    for (size_t i = 0; i < dims[DIM_COUNT - 1]; i++)
        delete[] matrix[i];
    delete[] matrix;
}
```

Завдання для індивідуального виконання

Варіант 1. Написати програму для обчислення матричних норм $||\mathrm{A}||_{\infty}$ та $||A||_F$.

Варіант 2. Написати програму для обчислення відстаней між *n*-вимірними дійсними векторами з формулами:

$$d_p(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p},\ p \in \mathbb{N},\ d_\infty(\mathbf{x},\mathbf{y}) = \max_{1 \le i \le n} |x_i - y_i|.$$

Варіант 3. Задана прямокутна числова матриця $m \times n$. Написати програму знаходження пари рядків, скалярний добуток яких найменший.

Варіант 4. Написати програму для паралельного швидкого сортування.

Варіант 5. Написати програму для паралельного сортування методом вибору.

Варіант 6. Написати паралельну реалізацію алгоритму сортування злиттям.

Варіант 7. Написати паралельну програму знаходження розв'язку системи лінійних алгебраїчних рівнянь.

Варіант 8. Написати паралельну програму обчислення детермінанта матриці.

Варіант 9. Написати паралельну версію решета Ератосфена.

Варіант 10. Написати паралельну версію програми наближеного обчислення визначеного ін-тегралу за формулою $\int_a^b f(x)dx$ методами прямокутників, трапецій та Сімпсона.

Лабораторна робота №7
“Паралельне багатопотокове програмування на базі сучасних фреймворків”

Мета та завдання роботи: ознайомлення із підтримкою парадигми паралельного програмування в сучасних фреймворках та забезпечення синхронізації даних за допомогою взаємного виключення та умовної синхронізації.

Зауваження: Завдання потрібно виконати з використанням багатопотокового програмування (на мовах C#, C++, Scala, Java). Написати дві версії класів програми. У першій реалізувати синхронізацію потоків за допомогою *семафорів*, у другій - *моніторів*. Написати програму, яка моделює паралельне функціонування об’єктів предметної області.

Приклад. На автозаправній станції є кілька колонок. Час від часу відбувається дозаправка резервуарів станції. До станції під’їжджають автомобілі для того, щоб заправитися потрібним їм об’ємом палива. Якщо немає вільних колонок, то автомобілі очікують, поки якась із колонок не звільниться. Якщо запас пального на станції достатній, то відбувається заправка, інакше запит на заправку ігнорується і автомобіль звільняє колонку. Написати паралельну програму для моделювання функціонування предметної області. Написати дві версії класів програми. У першій реалізувати синхронізацію потоків за допомогою *семафорів*, у другій — *моніторів*.

Розв’язок. Використаємо наступний абстрактний клас-предок автозаправної станції:

```
abstract class AbstractFuelStation
{

    public int Reserve { get; protected
    set; } public int Capacity { get; }
    = 1000; public int ColumnCount {
    get; }

    public AbstractFuelStation(int columnCount, int
    reserve = 1000) {

        if (columnCount <= 0) throw new Exception("Invalid
        column count");
```

```
        if (reserve > Capacity || reserve < 0) throw new
        Exception("Invalid reserve");

        ColumnCount = columnCount;
        Reserve = reserve;

    }

    public abstract void Fill(int amount);

    public abstract bool TryRefuel(Car car, int volume);
}
```

Використаємо таку реалізацію класу Car:

```
class Car
{

    private static Random random = new
    Random(); public string Name { get;
    set; } public int TankVolume { get; }

    public AbstractFuelStation Station {
    get; } public int RefuelCount { get;
    }

   public Car(string name, AbstractFuelStation station, int
    tankVolume, int refuelCount) {

        Name = name;
        TankVolume = tankVolume;
        Station = station;

        RefuelCount = refuelCount;
    }

    public void Run()

    {

        Console.WriteLine($"{Name} STARTED; refuel count is
        {RefuelCount}"); for (int i = 0; i < RefuelCount;
        i++) {

            var volume = random.Next(TankVolume) + 1;
```

```
            Console.WriteLine($"{Name} tries to get {volume}
            lit. for {i + 1} times"); while
            (!Station.TryRefuel(this, volume)) {

                Console.WriteLine($"{Name} is
                waiting");
                Thread.Sleep(random.Next(1, 5) *
                1000);

            }
            Thread.Sleep(random.Next(1, 3) * 1000);

            Console.WriteLine($"{Name} left the station for {i
            + 1} times");
        }

        Console.WriteLine($"{Name} SAID GOOD-BYE");
    }
}
```

Розглянемо такий клас-нащадок, розроблений з використанням семафорів:

```
class FuelStationSemaphore :
AbstractFuelStation {

    private Mutex mutex = new Mutex();
    private SemaphoreSlim semaphore;

    public  FuelStationSemaphore(int  columnCount,  int
    reserve  =  1000)  :
base(columnCount, reserve)

    {
        semaphore = new SemaphoreSlim(ColumnCount,
        ColumnCount);
    }

    public override void Fill(int amount)
    {mutex.WaitOne();

        if (amount + Reserve > Capacity)
            Reserve = Capacity;
        else

            Reserve += Capacity;
```

```
        Console.WriteLine("STATION TANK REFUELED");
        mutex.ReleaseMutex();
    }

    public override bool TryRefuel(Car car, int volume)
    {

        semaphore.Wait();
        mutex.WaitOne();
        bool result = false;
        if (volume <= Reserve)

        {

            Console.WriteLine($"Fuels reserve is {Reserve}.
{volume} liters fueling began for {car.Name}");

            Reserve -= volume;
            mutex.ReleaseMutex();
            Thread.Sleep(100 * volume);
            Console.WriteLine($"{car.Name} fueling finished");

            result = true;
        }
        else

        {

            Console.WriteLine($"Fuel reserve {Reserve} is
            insufficient for {car.Name}");
            mutex.ReleaseMutex();

        }

        semaphore.Release();
        return result;
    }
}
```

У наведеному класі взаємне виключення реалізується з використанням м’ютекса, а синхронізація при звертанні до заправних колонок — за допомогою семафорів.

Розглянемо реалізацію, яка заснована на використанні моніторів:

```
class FuelStationMonitor :
AbstractFuelStation {

    private object lockObject;
    private int freeColummCount;

    public FuelStationMonitor(int columnCount, int reserve) :
base(columnCount, reserve)
    {

        lockObject = new object();
        freeColummCount = ColumnCount;
    }

    public override void Fill(int amount){

        lock (lockObject)
        {
            if (amount + Reserve > Capacity)

                Reserve = Capacity;
            else
                Reserve += Capacity;
            Console.WriteLine("STATION TANK REFUELED");

            Monitor.PulseAll(lockObject);
        }
    }

    public override bool TryRefuel(Car car, int volume)
    {
        bool result = false;

        lock (lockObject)
        {
            while (freeColummCount <= 0)

                Monitor.Wait(lockObject);
            if (volume <= Reserve)
            {
                freeColummCount--;

                Console.WriteLine($"Fuels reserve is {Reserve}.
{volume} liters fueling began for {car.Name}");
                Reserve -= volume;
```

```
                result = true;
            }
            else

            {

    Console.WriteLine($"Fuel reserve {Reserve} is insufficient
                                                for {car.Name}");
            }
        }
        if (result)

        {
            Thread.Sleep(100 * volume);
            Console.WriteLine($"{car.Name} fueling finished");

            lock (lockObject)
            {
                freeColummCount++;
                Monitor.PulseAll(lockObject);

            }
        }
        return result;

    }
}
```

Розглянемо наступне комп’ютерне моделювання взаємодії об’єктів предметної області:

```
class Program

{
    static void Main(string[] args)
    {

        int carCount = 5;
        Random random = new Random();

        Console.WriteLine("\tGO!!!");

        var fuelStation = new
        FuelStationMonitor(2, 300); var tasks =
        new Task[carCount]; for (int i = 0; i <
        carCount; i++)
```

```
        {

            var car = new Car("car"+(i+1), fuelStation, 100,
            random.Next(1, 5)); tasks[i] = new Task(car.Run);

        }

        tasks.ToList().ForEach(x =>
        x.Start()); Thread.Sleep(30000);
        fuelStation.Fill(2000);
        Task.WaitAll(tasks);
        Console.WriteLine("\tFINISH!!!")
        ;
    }

}
```

Завдання для індивідуального виконання

Варіант 1. Розв'язати задачу про виробників-споживачів у випадку одного споживача та n виробників, які можуть використовувати буфер розміру k. Споживач має зчитати усі n_i значень, які йому має передати i-й виробник ($i = 1, \ldots, m$).

Варіант 2. *Трамвай*. Трамвай, у який може вміститися n пасажирів, рухається по циклічному маршруту з k зупинками, на яких входять та виходять пасажири. Реалізувати об'єкти «трамвай» та «пасажир» і змоделювати процес руху.

Варіант 3. *Принтери*. Нехай n користувачів спільно та багаторазово використовують два принтери. Перед використанням принтера користувачі викликають функцію request. Ця функція чекає, поки один з двох принтерів не звільниться, і повертає ідентифікатор вільного принтера. Після використання принтера користувач звільняє його, викликаючи функцію release.

Варіант 4. *Курник*. Є n пташенят та мама-квочка. Пташенята їдять із загальної миски, яка вміщує F порцій їжі. Кожне пташеня з'їдає порцію їжі, спить деякий час, а потім знову їсть. Коли закінчується їжа, то повідомляється квочка, яка наповнює миску F

новими порціями їжі. Далі ці дії повторюються. Реалізувати класи «курник», «курча» та ««квочка»» і змоделювати функціонування курника.

Варіант 5. *Ліфт*. Ліфт, у який може вміститися *m* пасажирів, стоїть на 1-му поверсі. Пасажири викликають ліфт та входять (якщо є вільні місця) і виходять з нього на потрібному їм поверсі. Реалізувати класи «ліфт» та «пасажир» і змоделювати процес руху та посадки/висадки пасажирів.

Варіант 6. *Машини на мосту.* До вузького мосту під'їжджають машини з півночі та півдня. Машини, які рухаються у одному напрямку, можуть долати міст одночасно, а в протилежних — ні. Змоделювати процес руху.

Варіант 7. *Обід філософів*. П'ять філософів сидять біля круглого столу. Вони проводять життя, чергуючи прийоми їжі та роздуми. У центрі столу знаходиться велике блюдо спагетті. процесі їжі філософи повинні користуватися двома виделками. На жаль, їм дали всього п'ять виделок. Між кожною парою філософів лежить одна виделка і вони домовилися, що кожен буде користуватися тільки тими виделками, які лежать поруч з ним (зліва і справа). Завдання — написати програму, що моделює поведінку філософів. Програма повинна уникати ситуації, в якій всі філософи голодні, але жоден з них не може взяти обидві виделки — наприклад, коли кожен з них тримає по одній вилці і не хоче віддавати її.

Варіант 8. *Вулик*. Є n бджіл та 1 ведмідь. Вони використовують один горщик, який уміщує H порцій меду. Спочатку горщик порожній. Поки горщик не наповниться, ведмідь спить, потім з'їдає увесь мед та засипає. Кожна бджола (багаторазово) збирає одну порцію меду та кладе її у горщик. Бджола, яка приносить останню порцію меду та заповнює горщик, будить ведмедя.

Варіант 9. *Банківський рахунок*. Кілька людей (потоків) використовують спільний рахунок. Можна розміщати або знімати кошти з рахунку. Поточний баланс рівний сумі усіх вкладених грошей мінус сума знятих коштів. Баланс не може бути від'ємним.

Розміщати кошти можна без затримки, при вилучені коштів можливою є пауза, поки на рахунку не буде достатньої суми. Реалізувати два методи: deposit(amount) та withdraw(amount).

Варіант 10. *Американські гірки*. Є n потоків-пасажирів і один потік-вагончик. Пасажири чекають черги проїхати в вагончику, який вміщує C людей, C < n. Вагончик може їхати тільки заповненим. Напишіть коди потоків-пасажирів і потоку-вагончика і розробіть засоби для їх синхронізації. Реалізувати три операції: takeRide, яку викликають пасажири, load і unload, які викликає потік-вагончик.

Лабораторна робота №8

“Використання функцій блокування та синхронізації технології OpenMP”

Мета: ознайомити студентів із засобами блокування та синхронізації доступу до даних технології OpenMP і навчитися їх застосовувати.

Порядок виконання роботи

Розробити програму, яка має реалізувати такі дії:

1) створити матрицю A розміром m n, елементи якої визначаються рандомно, m задає кількість рядків і кількість потоків, які виконуватимуть паралельну область програми, n задає кількість стовпців. Змінні задають у в програмному коді;

2) у паралельній області за допомогою директиви single або master вивести такі дані: номер лабораторної роботи; назву лабораторної роботи; групу студента; ПІБ студента; номер варіанта і номер завдання;

3) обробити паралельним способом матрицю відповідно до свого варіанту.

Кожен потік має обробляти свій рядок матриці. Результати обробки вивести у текстовий файл. При цьому перед виводом кожен потік має виставити блокування за допомогою механізму замків;

4) у текстовий файл усі потоки почергово мають вивести повідомлення «Початок закритої секції...» і «Кінець закритої секції...». Якщо при цьому між двома повідомленнями від одного потоку зустрінуться повідомлення від інших потоків про невдалу спробу увійти до закритої секції, вони також мають бути записані у файл.

Завдання для індивідуального виконання

Варіант 1. Знайти мінімальне значення кожного рядка матриці.

Варіант 2. Знайти максимальне значення кожного рядка матриці.

Варіант 3. Порахувати кількість додатних елементів у кожному рядку матриці.

Варіант 4. Порахувати кількість від’ємних елементів у кожному рядку матриці.

Варіант 5. Порахувати кількість нульових елементів у кожному рядку матриці.

Варіант 6. Порахувати суму елементів у кожному рядку матриці.

Варіант 7. Порахувати суму додатних елементів у кожному рядку матриці.

Варіант 8. Порахувати суму від’ємних елементів у кожному рядку матриці.

Варіант 9. Порахувати суму додатних елементів у кожному стовпці матриці.

Варіант 10. Порахувати суму від’ємних елементів у кожному стовпці матриці.

Лабораторна робота №9

“Алгоритми планування паралельного виконання циклів технології OpenMP”

Мета: ознайомитись із засобами планування паралельного виконання циклів технології OpenMP і навчитися їх застосовувати.

Порядок виконання роботи: Створити програму яка має реалізувати наступні дії:

1) створити квадратні матриці A та B розміром n n, елементи яких визначені рандомно, n задає кількість рядків і кількість потоків, які виконуватимуть паралельну область програми. Змінні можуть задаватися у програмному коді або вводитися з клавіатури;

2) у паралельній області за допомогою директиви single/master вивести такі дані: номер лабораторної роботи; назву лабораторної роботи; групу студента; ПІБ студента; номер варіанта і номер завдання;

3) у кожній матриці окремо обчислити паралельним способом вираз відповідно до свого варіанта. Розподіл ітерацій між потоками виконати за допомогою директиви for з використанням опції schedule. Для кожної з матриць використати різні значення параметра type та розміри блоку – chunk. Результати обробки записати в масиви C та D;

4) вивести результати обробки матриць паралельним способом, вказавши під час виводу розподіл ітерацій за потоками для різних значень опції schedule.

Порівняти результати, отримані за різних значень параметра type та розміром блоку – chunk.

Завдання для індивідуального виконання

Варіант 1. Знайти мінімальне значення кожного рядка матриці.

Варіант 2. Знайти максимальне значення кожного рядка матриці.

Варіант 3. Порахувати кількість додатних елементів у кожному рядку матриці.

Варіант 4. Порахувати кількість від’ємних елементів у кожному рядку матриці.

Варіант 5. Порахувати кількість нульових елементів у кожному рядку матриці.

Варіант 6. Порахувати суму елементів у кожному рядку матриці.

Варіант 7. Порахувати суму додатних елементів у кожному рядку матриці.

Варіант 8. Порахувати суму від'ємних елементів у кожному рядку матриці.

Варіант 9. Порахувати суму додатних елементів у кожному стовпці матриці.

Варіант 10. Порахувати суму від'ємних елементів у кожному стовпці матриці.

Лабораторна робота №10

“Програмування паралельних процесів, використовуючи різні схеми розділення даних”

Мета: ознайомитись зі створенням паралельних програм, використовуючи різні схеми розподілення даних та ресурси технології OpenMP та MPI.

Порядок виконання роботи: Лабораторна робота виконується у 2 етапи.

Перший етап. Опис задачі. Передбачається, що вихідні матриці A (MxN) та B (NxL) і результат множення (матриця C(M L) цілком розташовані в пам’яті кожного процесора.

1. У нульовому процесорі матриці A і B заповнюються будь-якими числами. Для подальшого контролю правильності роботи програми на нульовому процесорі обчислити їх добуток – матрицю D = AxB. У MPI-програмі за допомогою процедури MPI_Bcast значення матриці A та B розсилають на всі інші процесори. Матриця A поділяється на M/n горизонтальних смуг (n – кількість процесорів, M/n – ціле число). Схема поділу матриці A на горизонтальні смуги представлена на рис 10.

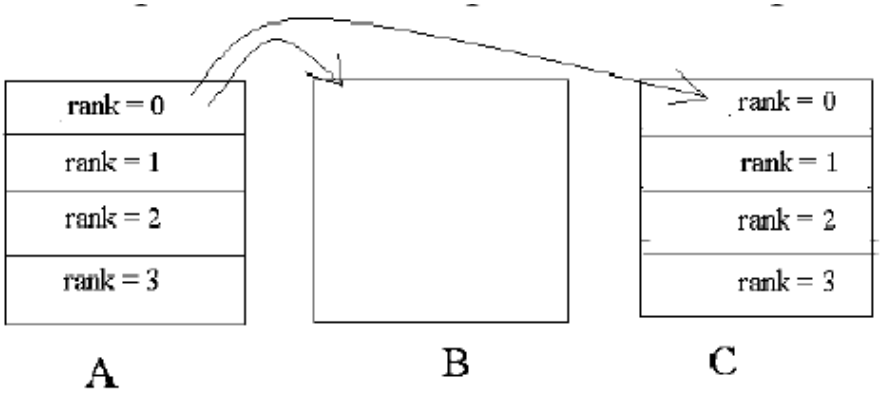

Рисунок 10 - Схема поділу матриці A на горизонтальні смуги

На кожному процесорі виконується множення своєї смуги на стовпчики матриці B так, щоб матриця C у кожному процесорі теж являла собою набір із M/n горизонтальних смуг. Після їх обчислення усі смуги матриці C збираються на нульовому процесорі за допомогою команди MPI_Gather. Для контролю правильності роботи програми матриця C поелементно порівнюється з матрицею Рис. 10 – Схема поділу матриці A на горизонтальні смуги D. Визначити час виконання програми за допомогою процедури MPI_Wtime як різницю часу між часом звернення до неї на

початку роботи програми (після розсилки) і останнім на завершення роботи програми. В OpenMP-програмі немає необхідності в поділу матриць на смуги. Використовуючи відповідні директиви OpenMP, розпаралелити самий зовнішний цикл алгоритмів множення матриць.

Для реалізації завдання першого етапу необхідно:

1) розробити програму для розв'язання задачі, яка описана вище, на MPI. Кількість рядків і стовпчиків всіх матриць однакова і дорівнює 1600 (для зручності рахування на 2, 4, 8 та 16 процесорах). Спосіб заповнення матриць числами – рандомне. Додайте до програми процедуру MPI_Wtime для визначення загального часу її роботи (t_c) і часу, який витрачено на обмін у процедурі MPI_Gather («накладні витрати» t_mpi);

2) проведіть розрахунки на 2, 4, 8 та 16 процесорах, побудуйте залежність коефіцієнта прискорення від числа процесорів. Побудуйте таблицю залежності t_c і t_mpi від кількості процесорів і визначте «оптимальну» кількість процесорів, які підтримують баланс між витратами часу на виконання програми та «накладними витратами»;

3) напишіть програму на OpenMP для розв'язання цієї задачі, визначте коефіцієнт прискорення обчислень залежно від кількості паралельних потоків, які використовуються (від кількості ядер на одному обчислювальному вузлі);

4) порівняйте ефективність роботи програм на MPI та OpenMP. Другий етап. Мета роботи – розробка паралельної програми на MPI та OpenMP для множення двох матриць A та B.

Опис задачі. Написати MPI-програму множення двох квадратних матриць A(N N) та B(N N), у якій матриця A поділяється на n горизонтальних смуг, а матриця B – на n вертикальних за кількістю процесорів n так, що на кожному процесорі міститься K = N/n (K – ціле) рядків матриці A і K стовпчиків матриці B. Матриця C (результат обчислень) у цьому випадку буде поділена на аналогичні горизонтальні смуги і також буде розподілена по процесорах. Для організації паралельних обчислень необхідно почергово із кожного процесора за допомогою топології «кільце» розсилати вертикальну смугу матриці B усім процесорам, що залишились. Після обчислення усіх смуг матриці C (NxN) вона збирається на нульовому процесорі та поелементно

порівнюється з елементами контрольної матриці, так визначаються всі витрати часу за аналогією з першим етапом. У кожному процесорі описуються масиви A[N][N], B[N][N], C_t[N][N]. Масиви A і B на нульовому процесорі заповнюються числовими константами рандомно. На нульовому процесорі обчислюється контрольна матриця C_t(A B). За допомогою процедури MPI_Bcast виконайте розсилання із нульового процесору всім іншим горизонтальні смуги матриці A і вертикальні смуги матриці B. Після завершення усіх обмінів за «кільцем» необхідно зібрати отриману матрицю на нульовому процесорі та порівняйте її поелементно з контрольною матрицею C_t. Схема поділу матриці A на горизонтальні смуги, матриці B на вертикальні смуги, формування матриці C і структура обмінів по «кільцю» для 4 процесорів показана на рис. 11.

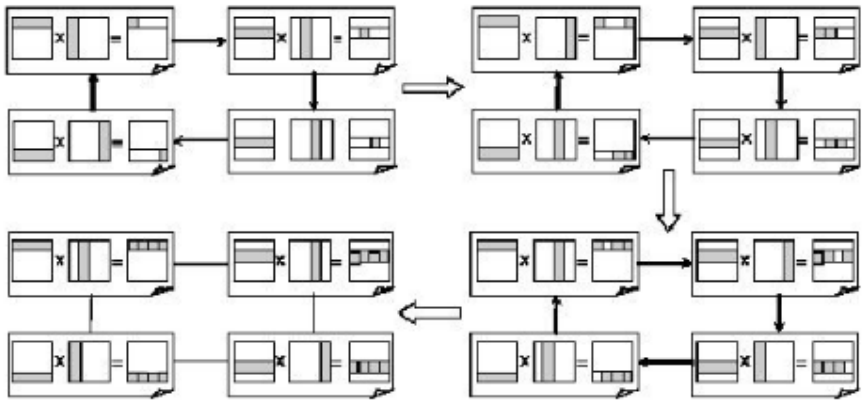

Рисунок 11 - Схема обмінів у топології «кільце»

Для реалізації завдання другого етапу необхідно:

1) розробити програму для розв’язання задачі, яка описана вище, на MPI з використанням топології «кільце». Задайте N = 1600 (для для зручності рахування на 2, 4, 8 та 16 процесорах) і заповнити матриці довільними числами. Додайте до програми процедуру MPI_Wtime для визначення загального часу її роботи, для цього необхідно зафіксувати час у момент закінчення виконання розсилання і час після завершення всіх обмінів за «кільцем»;

2) проведіть розрахунки на 2, 4, 8 та 16 процесорах, побудуйте залежність коефіцієнта прискорення від числа процесорів;

3) порівняйте ефективність роботи програм першого та другого етапів.

Лабораторна робота №11

“Розв’язування системи алгебраїчних рівнянь засобами паралельного програмування”

Мета: ознайомитись зі створенням паралельних програм для розв’язання СЛАР методом Гауса, використовуючи ресурси технології OpenMP та MPI.

Порядок виконання роботи

1. Напишіть паралельну програму для розв’язання СЛАР методом Гауса використовуючи технологію OpenMP та MPI. Реалізувати програму розв’язання СЛАР на 2, 4, 6 і 12 процесорах.

2. Додайте до програми виміри часу лічильної її частини (t_c) у кожному процесорі та часу (t_mpi), для визначення часу на обмін даними з сусідніми процесорами («накладні витрати»). Проведіть обчислення на 2, 4, 6 і 12 процесорах, побудуйте залежність часу на обмін даними від кількості процесорів.

Побудуйте у вигляді таблиці залежності t_c і t_mpi від кількості процесорів, визначте «оптимальну» кількість процесорів, яке підтримує баланс між тимчасовими витратами на рахунок і «накладними витратами».

Завдання для індивідуального виконання

1.

$$\begin{cases} 2x_1 - 2x_2 + 3x_3 = 1, \\ 3x_1 - 5x_2 + 4x_3 = 5, \\ 4x_1 + 2x_2 + 15x_3 = -1. \end{cases}$$

2.

$$\begin{cases} 3x_1 + 2x_2 - x_3 = -4, \\ 4x_1 - 3x_2 - 4x_3 = -14, \\ -2x_1 + x_2 + 3x_3 = 5. \end{cases}$$

3.
$$\begin{cases} 2x_1 + 3x_2 + 2x_3 = -8, \\ 3x_1 + 3x_2 + 2x_3 = -5, \\ 4x_1 + x_2 + 2x_3 = 2. \end{cases}$$

4.
$$\begin{cases} 2x_1 + 2x_2 + 3x_3 = 10, \\ -3x_1 + 4x_2 + 5x_3 = -3, \\ x_1 - 2x_2 - 2x_3 = 1. \end{cases}$$

5.
$$\begin{cases} 2x_1 + 3x_2 + 4x_3 = 6, \\ x_1 + 2x_2 + x_3 = 7, \\ 5x_1 + x_2 + x_3 = 7. \end{cases}$$

6.
$$\begin{cases} 2x_1 - 3x_2 + 3x_3 = -1, \\ 7x_1 + 3x_2 - 2x_3 = 7, \\ 4x_1 + x_2 + 3x_3 = -7. \end{cases}$$

7.
$$\begin{cases} -2x_1 - 5x_2 + x_3 = 12, \\ 2x_1 + 4x_2 + 3x_3 = 6, \\ 3x_1 + x_2 + x_3 = 5. \end{cases}$$

8.
$$\begin{cases} 2x_1 + 3x_2 + x_3 = -5, \\ 3x_1 - 4x_2 - 5x_3 = 5 \\ 2x_1 + 2x_2 + 3x_3 = 2. \end{cases}$$

9.

$$\begin{cases} 3x_1 + x_2 - x_3 = -1, \\ 2x_1 - 3x_2 - 3x_3 = 1, \\ -3x_1 + 2x_2 + 2x_3 = -4. \end{cases}$$

10.

$$\begin{cases} x_1 + 2x_2 - x_3 = 11, \\ 2x_1 - x_2 - 3x_3 = 4, \\ 7x_1 - 2x_2 + x_3 = -3. \end{cases}$$

Лабораторна робота №12

“Програмування паралельних процесів, використовуючи різні методи сортування”

Мета: ознайомитись зі створенням паралельних програм для використання різних методів сортування, використовуючи ресурси технології OpenMP та MPI.

Порядок виконання роботи

1. Реалізувати алгоритм сортування «бульбашкою» у його послідовній версії для однопроцесорних систем, паралельної версії для багатопроцесорних систем, використовуючи стандарт MPI та OpenMP і модифікацію методу сортування «бульбашкою», відомий в літературі як метод парної-непарної перестановки (the odd-even transposition method). Сутність модифікації полягає в тому, що в алгоритм сортування вводять два різні правила виконання ітерацій методу: залежно від парності або непарності номера ітерації сортування для обробки вибираються елементи з парними або непарними індексами відповідно, порівняння виділених значень завжди здійснюється з їх правими сусідніми елементами. Отже, на всіх непарних ітераціях порівнюються пари:

(a1, a2), (a3, a4), ..., (an-1,an) (при парному n),

а на парних ітераціях обробляються елементи:

(a2, a3), (a4, a5), ..., (an-2,an-1).

Після n-кратного повторення ітерацій сортування вихідний набір даних виявляється впорядкованим.

2. Обчислити час роботи кожної із програм у всіх випадках, протестувати і порівняти прискорення паралельного та послідовного алгоритму.

Лабораторна робота №12

“Розпаралелювання програми обчислення визначеного інтеграла за допомогою OpenMP”

Мета: ознайомити студентів зі створенням паралельних программ, використовуючи ресурси технології OpenMP.

Порядок виконання роботи

1. Застосовуючи технологію паралельного програмування OpenMP, написати програму для обчислення значення визначеного інтеграла, використовуючи метод трапеції. Завдання згідно з індивідуальним варіантом.

2. Скласти порівняльну таблицю часу виконання програми за різними характеристиками.

Завдання для індивідуального виконання

1. $\int_{-2}^{0}(x^2+5x+6)\cos 2x dx.$
2. $\int_{-2}^{0}(x^2-4)\cos 3x dx.$
3. $\int_{-1}^{0}(x^2+4x+3)\cos x dx.$
4. $\int_{-2}^{0}(x+2)^2\cos 3x dx.$
5. $\int_{-4}^{0}(x^2+7x+12)\cos x dx.$
6. $\int_{0}^{\pi}(2x^2+4x+7)\cos 2x dx.$
7. $\int_{0}^{\pi}(9x^2+9x+11)\cos 3x dx.$
8. $\int_{0}^{\pi}(8x^2+16x+17)\cos 4x dx.$
9. $\int_{0}^{2\pi}(3x^2+5)\cos 2x dx.$
10. $\int_{0}^{2\pi}(2x^2-15)\cos 3x dx.$
11. $\int_{0}^{2\pi}(3-7x^2)\cos 2x dx.$
12. $\int_{0}^{2\pi}(1-8x^2)\cos 4x dx.$
13. $\int_{-1}^{0}(x^2+2x+1)\sin 3x dx.$
14. $\int_{0}^{3}(x^2-2x)\sin 2x dx.$
15. $\int_{0}^{\pi}(x^2-3x+2)\sin x dx.$
16. $\int_{0}^{\pi/2}(x^2-5x+6)\sin 3x dx.$

Список використаних джерел

1. Коцовський В. М. Частина II: Методичний посібник. Ужгород: Видавництво УжНУ "Говерла", 2017. 76 с.
2. Коцовський В. М. Основи дискретної математики: навчальний посібник. Ужгород: ПП «АУТДОР-ШАРК», 2020. 128 с.
3. Коцовський В. М. Дискретна математика та теорія алгоритмів. Частина I: Конспект лекцій для студентів спеціальностей: 6.122 - "Комп’ютерні науки", 6.121 - "Інженерія програмного забезпечення". Ужгород: Видавництво УжНУ "Говерла", 2019. 52 с.
4. Коцовський В. М. Дискретна математика та теорія алгоритмів. Частина II: Конспект лекцій для студентів спеціальностей: 6.122 - "Комп’ютерні науки", 6.121 - "Інженерія програмного забезпечення". Ужгород: Видавництво УжНУ "Говерла", 2019. 52 с.
5. Коцовський В. М. Теорія паралельних обчислень Методичні матеріали до лабораторних робіт. Ужгород, 2020. 32 с.
6. Максимова Л. П. Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Паралельне програмування та розподілені обчислення». Кременчук, 2019. 55 с.

УДК 004(076)

Електронне мережне навчальне видання

ЛАБОРАТОРНИЙ ПРАКТИКУМ ІЗ ДИСЦИПЛІНИ “ПАРАЛЕЛЬНІ ТА РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ”

для студентів спеціальності 122 Комп’ютерні науки
першого (бакалаврського) рівня

Друкується в авторській редакції