Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Звіт до лабораторної роботи №5

«Бібліотека OpenMP. Бар’єри, критичні секції»

з дисципліни **«Програмне забезпечення високопродуктивних комп’ютерних систем»**

Виконала:
студентка групи ІМ-13
Мартинюк Марія Павлівна
номер у списку групи 13

Перевірив:
Корочкін О. В.

Київ 2024

Мета роботи: розробка програми для ПКС зі СП
Мова програмування: C++
Засоби організації взаємодії процесів: бар'єри, критичні секції OpenMP

Вхідні дані:

- комп'ютерна система зі спільною пам'яттю включає чотири процесори і три пристрої введення-виведення (рис. 1.1);
- математичне завдання згідно **варіанту №4**:
  A = sort(D * (ME * MM)) * p + (B * C) * E * x,
  де A, D, B, C, E - вектори розміру N; ME, MM - матриці розміру N; p, x - скаляри;
- введення значень для D, MM, p виконується у процесорі 2;
  введення значень для ME, B, E виконується у процесорі 3;
  введення значень для C, x виконується у процесорі 4;
  виведення результату A виконується у процесорі 4.

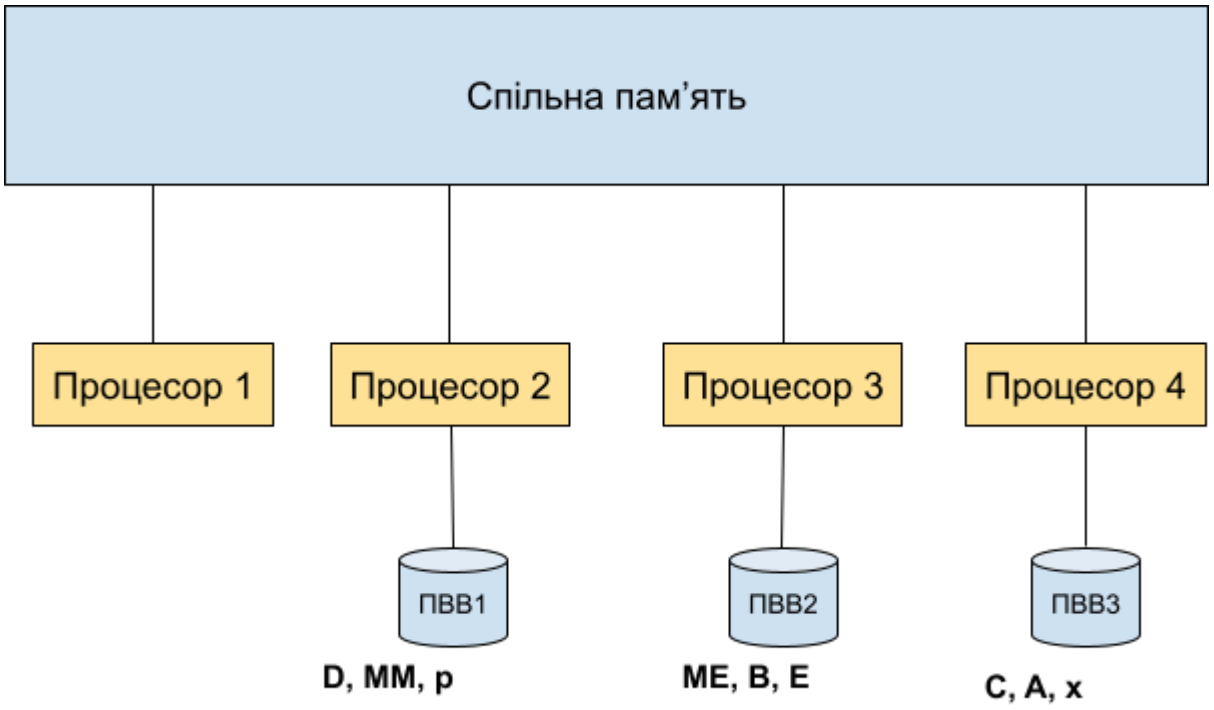


рис. 1.1 Структурна схема 4 процесорної системи

Етап 1. Побудова паралельного алгоритму

A = sort(D * (ME * MM)) * p + (B * C) * E * x,

1. Xн = sort(D * (ME * MMн)) Потоки T1-T4
2. X2н = sortzl(Xн, Xн) Потоки T1 і T3
3. X = sortzl(X2н, X2н) Потік T1
4. ai = (Bн * Cн), i = 1...P
2. a = a + ai
4. Aн = Xн * p + a * Eн * x CP: p, x, a - копії

,де н = N/P частина вектору або матриці, N - розмір вектору/матриці, P = 4 - кількість процесорів (потоків).

Етап 2. Розробка алгоритмів потоків

Задача T1	**КД**
1. Чекати сигналу від задачі T2 про введення D, MM, p; від задачі T3 про введення ME, B, E; від задачі T4 про введення C, x	
2. Обчислення1 Xн = sort(D * (ME * MMн))	
3. Чекати на завершення обчислень Xн в T2	
4. Обчислення2 X2н = sortzl(Xн, Xн)	
5. Чекати на завершення обчислень X2н в T3	
6. Обчислення3 X = sortzl(X2н, X2н)	
7. Сигнал T2, T3, T4 про завершення обчислень X	
8. Обчислення4 a1 = (Bн * Cн)	
9. Обчислення5 a = a + a1	КД1

10. Сигнал T2, T3, T4 про завершення обчислень a	
11. Чекати на завершення обчислень a в T2, T3, T4	
12. Копія a1 = a	КД2
13. Копія p1 = p	КД3
14. Копія x1 = x	КД4
15. Обчислення5 Ан = Хн * p1 + a1 * Ен * x1	
16. Сигнал T4 про завершення обчислень Ан	

Задача T2	**КД**
1. Введення D, ММ	
2. Сигнал задачі T1, T3, T4 про введення D, ММ	
3. Чекати сигналу від задачі T3 про введення МЕ, В, Е; від задачі T4 про введення С, х	
4. Обчислення1 Хн = sort(D * (МЕ * ММн))	
5. Сигнал T1 про завершення обчислень Хн	
6. Чекати на завершення обчислень Х в T1	
7. Обчислення2 a2 = (Вн * Сн)	
8. Обчислення3 a = a + a2	КД1
9. Сигнал T1, T3, T4 про завершення обчислень a	
10. Чекати на завершення обчислень a в T1, T3, T4	

11. Копія a2 = a	КД2
12. Копія p2 = p	КД3
13. Копія x2 = x	КД4
14. Обчислення4 Ан = Хн * p2 + a2 * Ен * x2	
15. Сигнал Т4 про завершення обчислень Ан	

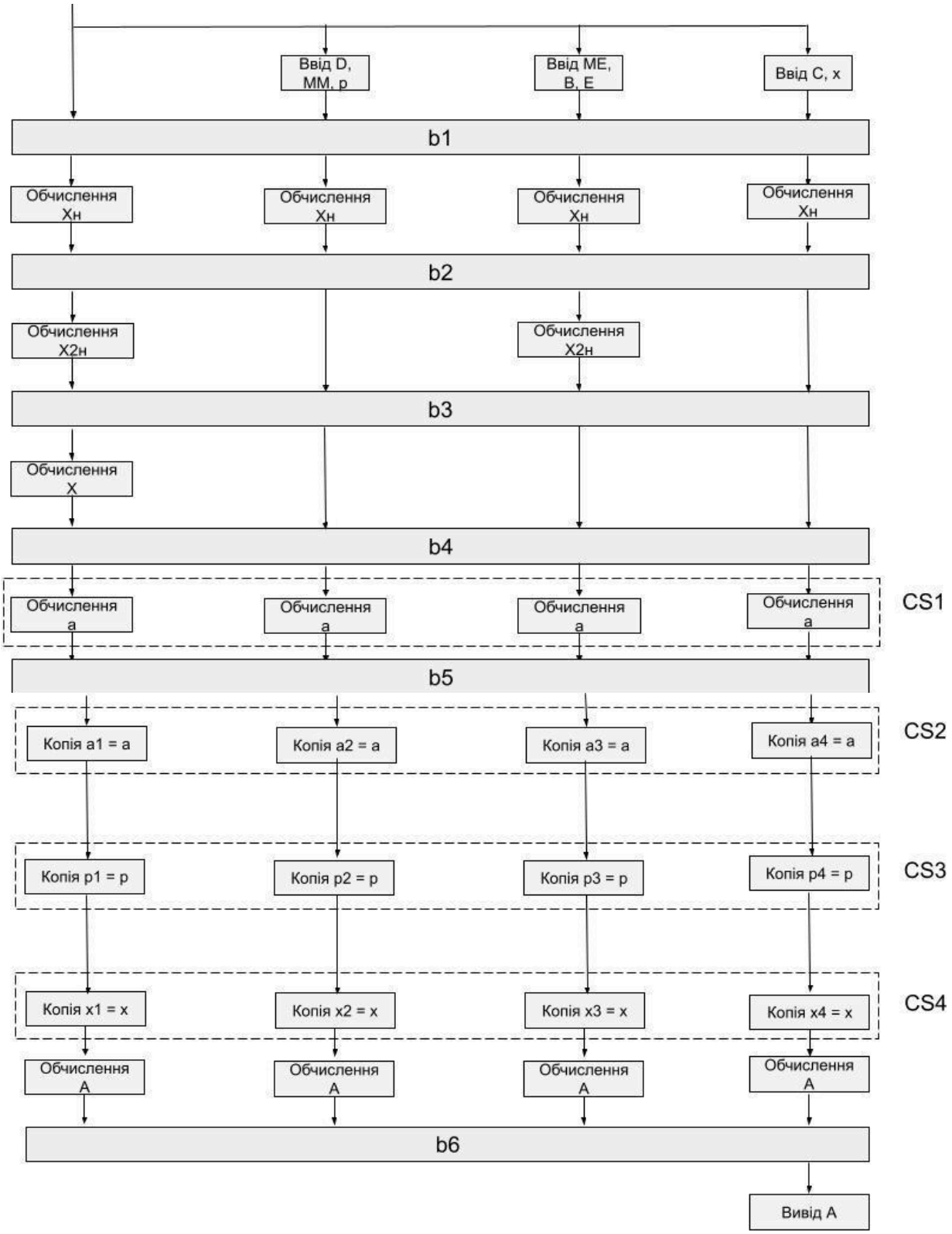
Задача Т3	**КД**
1. Введення МЕ, В, Е	
2. Сигнал задачі Т1, Т2, Т4 про введення МЕ, В, Е	
3. Чекати сигналу від задачі Т2 про введення D, ММ, p; від задачі Т4 про введення С, x	
4. Обчислення1 Хн = sort(D * (МЕ * ММн))	
5. Чекати на завершення обчислень Хн в Т4	
6. Обчислення2 Х2н = sortzl(Хн, Хн)	
7. Сигнал Т1 про завершення обчислень Х2н	
8. Чекати на завершення обчислень Х в Т1	
9. Обчислення3 a3 = (Вн * Сн)	
10. Обчислення4 a = a + a3	КД1
11. Сигнал Т1, Т2, Т4 про завершення обчислень a	
12. Чекати на завершення обчислень a в Т1, Т2, Т4	
13. Копія a3 = a	КД2

14. Копія p3 = p	КД3
15. Копія x3 = x	КД4
16. Обчислення5 Ан = Хн * p3 + a3 * Ен * x3	
17. Сигнал Т4 про завершення обчислень Ан	

Задача Т4	**КД**
1. Введення С, x	
2. Сигнал задачі Т1, Т2, Т3 про введення С, x	
3. Чекати сигналу від задачі Т2 про введення D, ММ, p; від задачі Т3 про введення МЕ, В, Е;	
4. Обчислення1 Хн = sort(D * (МЕ * ММн))	
5. Сигнал Т3 про завершення обчислень Хн	
6. Чекати на завершення обчислень Х в Т1	
7. Обчислення2 a4 = (Вн * Сн)	
8. Обчислення3 a = a + a4	КД1
9. Сигнал Т1, Т2, Т3 про завершення обчислень a	
10. Чекати на завершення обчислень a в Т1, Т2, Т3	
11. Копія a4 = a	КД2
12. Копія p4 = p	КД3
13. Копія x4 = x	КД4
14. Обчислення5 Ан = Хн * p4 + a4 * Ен * x4	

15. Чекати на завершення обчислень Ан в Т1, Т2, Т3	
16. Вивід А	

Етап 3. Розробка схеми взаємодії задач



Бар’єри:

- **b1 -** для синхронізації введення даних в потоках T2, T3, T4
- **b2 -** для синхронізації обчислень Xн
- **b3 -** для синхронізації обчислень X2н

- **b4 -** для синхронізації обчислень X
- **b5 -** для синхронізації обчислень a
- **b6 -** для синхронізації обчислень A

Критичні секції:

- **CS1 -** для захисту КД1 при обчисленні значення a
- **CS2 -** для захисту КД2 при копіюванні значення a
- **CS3 -** для захисту КД3 при копіюванні значення p
- **CS4 -** для захисту КД4 при копіюванні значення x

Структурна схема взаємодії базується на розробленому в етапі 2 алгоритмі взаємодії потоків та дозволяє наочно проконтролювати зв'язок у багатопотоковій програмі, використовуючи при цьому вище описані засоби синхронізації.

Етап 4. Розробка програми

Програма складається з головного файлу Lab5.cpp, в якому з використанням бібліотеки OpenMP відбувається розподілення обчислення наданого за варіантом математичного виразу між доступними потоками. Синхронізація роботи потоків здійснюється за допомогою бар'єрів, доступ до спільних ресурсів регулюється за допомогою критичних секцій. Окрім цього у розробці програми була використана pragma for, яка дозволяє автоматично розподілити елементи в циклі між потоками.

Lab5.cpp

```
//Лабораторна робота ЛР5 Варіант 4
//A = sort(D * (ME * MM)) * p + (B * C) * E * x
//Мартинюк Марія Павлівна
//Дата 08.05.2024

#include <iostream>
#include <omp.h>
#include <chrono>
#include <algorithm>

const int N = 8;
const int P = 4;
const int H = N / P;
```

```
int* A = new int[N];
int* D;
int* B;
int* C;
int* E;
int* X;

int** ME;
int** MM;
int **MExMM = new int* [N];

int a, p, x;

int pi, xi, ai;
int thread_num;

void createArrays();
void vectorInput(int* v);
void matrixInput(int** m);
void XnCalculate();
int vectorMultiplication(int* v1, int* v2);
void printVector(int* vector, int size);

int main() {
   auto start = std::chrono::high_resolution_clock::now();
   createArrays();

#pragma omp parallel num_threads(P) private(thread_num, ai, pi, xi)
shared(a, p, x)
   {
       thread_num = omp_get_thread_num() + 1;

       switch (thread_num) {
           case 1:                  //T1
               std::cout << "T1 is started.\n";
               break;
           case 2:                  //T2
               std::cout << "T2 is started.\n";
               vectorInput(D);      //Введення D
               matrixInput(MM);     //Введення MM
               p = 1;               //Введення p
               break;
           case 3:                  //T3
               std::cout << "T3 is started.\n";
               matrixInput(ME);     //Введення ME
               vectorInput(B);      //Введення B
               vectorInput(E);      //Введення E
               break;
```

```
            case 4:                     //T4
                std::cout << "T4 is started.\n";
                vectorInput(C);         //Введення C
                x = 1;                  //Введення x
                break;
            default:
                break;
        }

#pragma omp barrier                 //b1 Синхронізація по введенню

        XnCalculate();              //Обчислення1 Xн = sort(D * (ME * MMн))

#pragma omp barrier                 //b2 Синхронізація по обчисленню Xn

        // Обчислення2 X2н = sortzl(Xн, Xн)
        switch (thread_num) {
            case 1:                 //T1
                std::inplace_merge(X, X + H, X + H * 2);
                break;
            case 3:                 //T3
                std::inplace_merge(X + H * 2, X + H * 3, X + H * 4);
                break;
        }


#pragma omp barrier                 //b3 Синхронізація по обчисленню X2n

        //Обчислення3 X = sortzl(X2н, X2н)
        if(thread_num == 1) {
            std::inplace_merge(X, X + H * 2, X + H * 4);
        }

#pragma omp barrier                 //b4 Синхронізація по обчисленню X

        ai = vectorMultiplication(B, C);    //Обчислення4 a1 = (Bн * Cн)

#pragma omp critical(CS1)            //КД1
        {
            a = a + ai;             //Обчислення5 a = a + a1
        }

#pragma omp barrier                 //b5 Синхронізація по обчисленню a

#pragma omp critical(CS2)           //КД2
        {
            ai = a;                 //Копія ai = a
        }
```

```
#pragma omp critical(CS3)      //КД3
        {
             pi = p;           //Копія pi = p
        }

#pragma omp critical(CS4)      //КД4
        {
            xi = x;            //Копія xi = x
        }

#pragma parralel for schedule(guided)
        for (int i = 0; i < N; i++) {
            A[i]  = (X[i] * pi) + (a * E[i] * xi);  //Обчислення5 Ан =
Хн * pi + ai * Ен * xi
        }

#pragma omp barrier              //b6 Синхронізація по виведенню

        if(thread_num == 4) {
#pragma omp critical
            {
                std::cout << "A: ";
                printVector(A, N);     //Вивід A
            }
        }

#pragma omp critical
        std::cout << "T" << thread_num << " is finished.\n";
    }

    auto stop = std::chrono::high_resolution_clock::now();
    auto duration =
std::chrono::duration_cast<std::chrono::milliseconds>(stop - start);
    std::cout << "Time: " << duration.count() << " ms" << std::endl;

}

// допоміжні методи для виконання обчислень
void createArrays() {
    A = new int[N];
    B = new int[N];
    C = new int[N];
    D = new int[N];
    E = new int[N];
    X = new int[N];
```

```
    ME = new int *[N];
    MM = new int *[N];
    MExMM = new int *[N];


#pragma omp parallel for schedule(guided)
    for (int i = 0; i < N; ++i) {
        ME[i] = new int[N];
        MM[i] = new int[N];
        MExMM[i] = new int[N];

        A[i] = 0;
        X[i] = 0;
    }
}

void vectorInput(int* v) {
    for (int i = 0; i < N; ++i)
        v[i] = 1;
}

void matrixInput(int** m) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            m[i][j] = 1;
        }
    }
}

int vectorMultiplication(int* v1, int* v2) {
    int sum = 0;
#pragma omp for schedule(static)
    for (int i = 0; i < N; ++i) {
        sum += v1[i] * v2[i];
    }
    return sum;
}

void printVector(int* vector, int size) {
    for (int i = 0; i < size; ++i)
        std::cout << vector[i] << " ";
    std::cout << std::endl;
}

void XnCalculate() {
    for (int i = 0; i < N; i++) {
#pragma omp for schedule(static)
        for (int j = 0; j < N; j++) {
```

```
            MExMM[i][j] = 0;
            for (int k = 0; k < N; k++) {
                MExMM[i][j] += ME[i][k] * MM[k][j];
            }
        }
    }

    for (int i = 0; i < N; i++) {
#pragma omp for schedule(static)
        for (int j = 0; j < N; j++) {
            int res = MExMM[j][i] * D[j];
            X[i] += res;
        }
        X[2] = 3;
    }

    int start = thread_num - 1 * H;
    int end = thread_num * H;

    std::sort(X + start, X + end);
}
```

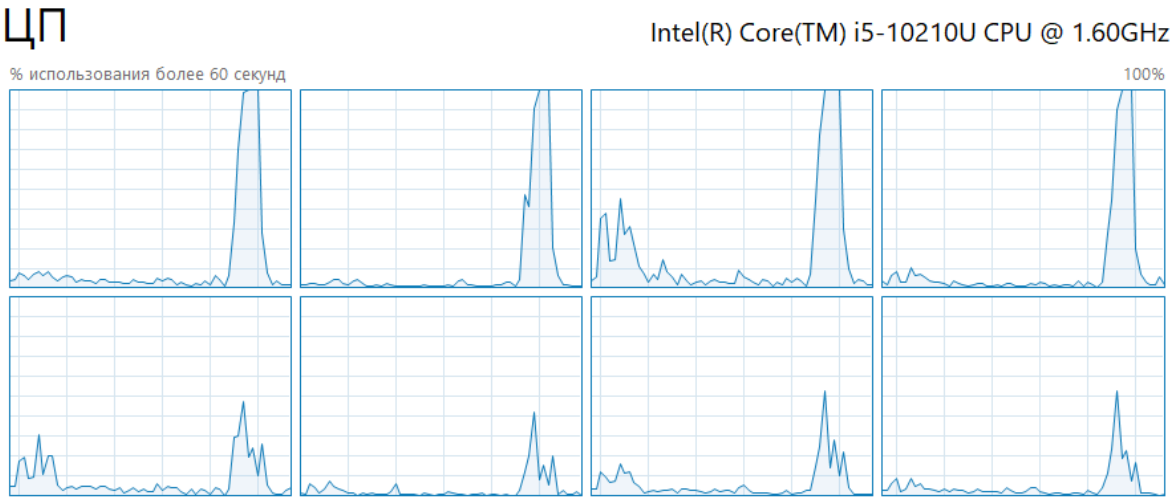
Результат виконання:

На рис. нижче наведений результат виконання багатопотокової програми при N = 8, P = 4. Задля впевненості у правильності розрахунків через використання у математичній формулі операції сортування, було змінено значення одного з елементів результуючого вектора X на 3. Усі інші елементи використовуваних векторів та матриць мали значення 1. Як видно з рис. нижче разрахунок частин та об’єднання їх у остаточний результат відбувається коректно.

```
T3 is started.
T2 is started.
T4 is started.
T1 is started.
T1 is finished.
T3 is finished.
T2 is finished.
A: 11 72 72 72 72 72 72 72
T4 is finished.
Time: 3 ms
```

Дослідження завантаженості ядер процесору:

Варто зазначити, що конкретно мій комп'ютер містить 8 логічних ядер та 4 фізичних. Нижче наведений рис. демонструє, що при використанні 4 ядер процесору розподілення обчислювальних можливостей відбувається рівномірно між кожним з них. Поєднуючи цю інформацію із раніше отриманими остаточними значеннями обрахунків, можна зробити висновок, що чотирьох потокова програма написана коректно.



Додатково було обраховано коефіцієнт прискорення при запуску програми на 4 ядрах та на 1 ядрі при N = 2000: Кп = 70609 ms/ 32330 ms = 2,18. Тобто при використанні 4 ядер замість 1, програма завершується у 2,18 рази швидше.

Висновок:

1. Побудовано та розроблено паралельний алгоритм для обчислення математичної задачі. За отриманим алгоритмом було визначено критичні ділянки з використання спільних ресурсів a, x, p, a.
2. Було розроблено схему взаємодії потоків, де наочно продемонстровано такі засоби синхронізації: бар'єри b1 - b6 та критичні секції cs1 -cs4. Завдяки наочності схеми було досягнуто успішної реалізації синхронізації паралельних обчислень.

3. На основі вище зазначених етапів розробки паралельного алгоритму було створено багатопоточну програму з використанням мови C++ та бібліотеки OpenMP. Для розгалуження циклів була використана pragma for. Отриманий вивід результату обчислень демонструє, що розрахунок остаточного значення виразу відбувається коректно.
4. Аналіз завантаженості ядер процесору при запуску програми підтвердив рівномірне розподілення навантаження між задіяними ядрами, що свідчить про успішну оптимізацію паралельних обчислень.
5. Обчислений коефіцієнт прискорення при N = 2000: Кп = 2,18, що також сповіщає про пришвидшення виконання програми у разі використання 4 ядер процесора замість 1.