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

Лабораторна робота №5
«Бібліотека OpenMP. Бар’єри, критичні секції»

з дисципліни
«Програмне забезпечення високопродуктивних комп’ютерних систем»

Виконала:
Студентка групи ІМ-13
Дубчак Аліна Євгеніївна
номер у списку групи: 5

Перевірив:
Корочкін О. В.

Київ 2024

ПОСТАНОВКА ЗАДАЧІ

- розробити паралельний алгоритм рішення математичної задачі;
- виявити спільні ресурси;
- описати алгоритм кожного потоку (T1 – Tp) з визначенням критичних ділянок (КД) і точок синхронізації (Wij , Sij);
- розробити структурну схему взаємодії задач;
- використати як засоби організації взаємодії потоків - монітори
- розробити програму (обов'язкові "шапка", коментарі)
- виконати налагодження програми;
- отримати правильні результати обчислень.
- за допомогою Диспетчеру задач Windows проконтролювати завантаження ядер процесору.
- введення даних повинно бути через забивання всіх елементів 1

Варіант завдання:

A = (R*MC)*MD*p + (B*Z)*E*d

1 – MC, E
2 – MD, d
3 – A,B,p
4 – R,Z

Мова програмування: C++

ОПИС ПРОГРАМИ

І етап. Побудова паралельного математичного алгоритму

Варіант 6

A = (R*MC)*MD*p + (B*Z)*E*d

1.	ai = (Bн*Zн)	i = 1...P
2.	a = a + ai	CP: a
3.	Sн = R*MCн	CP: R
4.	Aн = S*MDн*p + a*Eн*d	CP: S, a, d, p

N - розмірність вектора/матриці.
P - кількість потоків, які виконують обчислення.
H = N / P

ІІ етап. Розробка алгоритмів потоків.

Т1

Точки синхронізації

1. Ведення MC, E
2. Сигнал задачам Т2, Т3, Т4 про введення MC, E -- S_{2-1}, S_{3-1}, S_{4-1}
3. Чекати на введення даних в задачах Т2, Т3, Т4 -- W_{2-1}, W_{3-1} W_{4-1}
4. Обчислення 1: a1 = (B_H*Z_H)
5. Обчислення 2: a = a + ai -- КД1
6. Сигнал задачам Т2, Т3, Т4 про завершення обчислення 2 -- S_{2-2}, S_{3-2}, S_{4-2}
7. Чекати на завершення обчислення 2 в задачах Т2, Т3, Т4 -- W_{2-2}, W_{3-2}, W_{4-2}
8. Обчислення 3 Sн = R*MCн
9. Сигнал задачам Т2, Т3, Т4 про завершення обчислення 3 -- S_{2-3}, S_{3-3}, S_{4-3}
10. Чекати на завершення обчислення 3 в задачах Т2, Т3, Т4 -- W_{2-3}, W_{3-3}, W_{4-3}
11. Копія a1 = a -- КД2
12. Копія d1 = d -- КД3
13. Копія p1 = p -- КД4
14. Обчислення 4: Ан = S*MDн*p1 + a1*Ен*d1
15. Сигнал задачі Т3 про завершення обчислення 4 -- S_{3-4}

Т2

1. Ведення MD, d
2. Сигнал задачам Т1, Т3, Т4 про введення MD, d -- S_{1-1}, S_{3-1}, S_{4-1}
3. Чекати на введення даних в задачах Т1, Т3, Т4 -- W_{1-1}, $W_{3-1}W_{4-1}$
4. Обчислення 1: a2 = (B_H*Z_H)
5. Обчислення 2: a = a + ai -- КД1
6. Сигнал задачам Т1, Т3, Т4 про завершення обчислення 2 -- S_{1-2}, S_{3-2}, S_{4-2}
7. Чекати на завершення обчислення 2 в задачах Т1, Т3, Т4 -- W_{1-2}, W_{3-2}, W_{4-2}
8. Обчислення 3 Sн = R*MCн
9. Сигнал задачам Т1, Т3, Т4 про завершення обчислення 3 -- S_{1-3}, S_{3-3}, S_{4-3}

10. Чекати на завершення обчислення 3 в задачах Т1, Т3, Т4 -- W_{1-3}, W_{3-3}, W_{4-3}

11. Копія a2 = a -- КД2

12. Копія d2 = d -- КД3

13. Копія p2 = p -- КД4

14. Обчислення 4: Ан = S*MDн*p2 + a2*Ен*d2

15. Сигнал задачі Т3 про завершення обчислення 4 -- S_{3-4}

Т3

1. Ведення A, B, p

2. Сигнал задачам Т1, Т2, Т4 про введення A, B, p -- S_{1-1}, S_{2-1}, S_{4-1}

3. Чекати на введення даних в задачах Т1, Т2, Т4 -- W_{1-1}, $W_{2-1}W_{4-1}$

4. Обчислення 1: a3 = (B_H*Z_H)

5. Обчислення 2: a = a + ai -- КД1

6. Сигнал задачам Т1, Т2, Т4 про завершення обчислення 2 -- S_{1-2}, S_{2-2}, S_{4-2}

7. Чекати на завершення обчислення 2 в задачах Т1, Т2, Т4 -- W_{1-2}, W_{2-2}, W_{4-2}

8. Обчислення 3 Sн = R*MCн

9. Сигнал задачам Т1, Т2, Т4 про завершення обчислення 3 -- S_{1-3}, S_{2-3}, S_{4-3}

10. Чекати на завершення обчислення 3 в задачах Т1, Т2, Т4 -- W_{1-3}, W_{2-3}, W_{4-3}

11. Копія a3 = a -- КД2

12. Копія d3 = d -- КД3

13. Копія p3 = p -- КД4

14. Обчислення 4: Ан = S*MDн*p3 + a3*Ен*d3

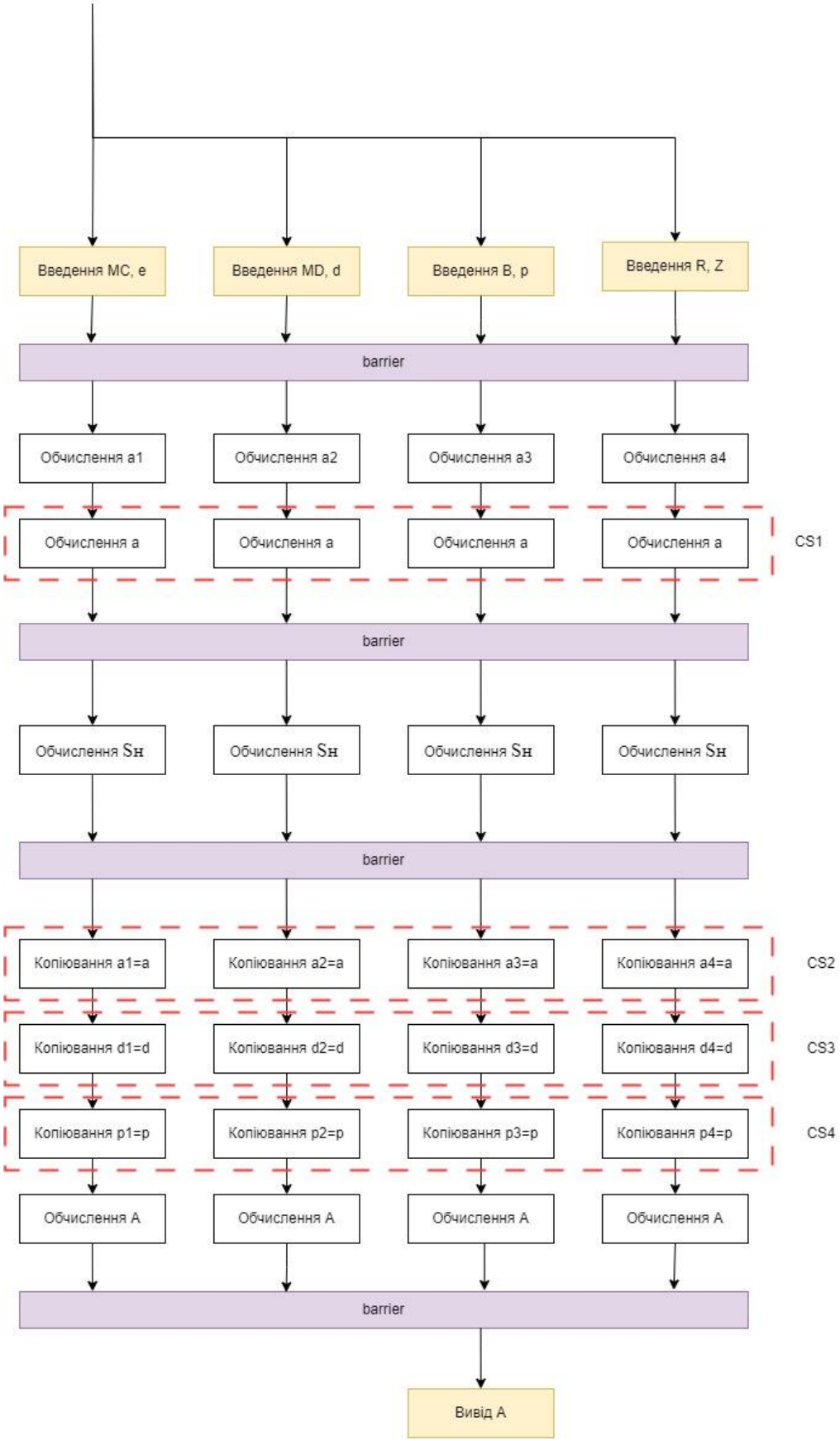
15. Чекати на завершення обчислення 4 в задачах Т1, Т2, Т4 -- W_{1-4}, W_{2-4}, W_{4-4}

16. Виведення результату A

T4

1. Ведення R, Z
2. Сигнал задачам T1, T2, T4 про введення R, Z -- S_{1-1}, S_{2-1}, S_{4-1}
3. Чекати на введення даних в задачах T1, T2, T3 -- W_{1-1}, $W_{2-1}W_{3-1}$
4. Обчислення 1: a4 = (B_H*Z_H)
5. Обчислення 2: a = a + ai -- КД1
6. Сигнал задачам T1, T2, T3 про завершення обчислення 2 -- S_{1-2}, S_{2-2}, S_{3-2}
7. Чекати на завершення обчислення 2 в задачах T1, T2, T3 -- W_{1-2}, W_{2-2}, W_{3-2}
8. Обчислення 3 Sн =R*MCн
9. Сигнал задачам T1, T2, T3 про завершення обчислення 3 -- S_{1-3}, S_{2-3}, S_{3-3}
10. Чекати на завершення обчислення 3 в задачах T1, T2, T3 -- W_{1-3}, W_{2-3}, W_{3-3}
11. Копія a4 = a -- КД2
12. Копія d4 = d -- КД3
13. Копія p4 = p -- КД4
14. Обчислення 4: Ан = S*MDн*p4 + a4*Ен*d4
15. Сигнал задачі T3 про завершення обчислення 4 -- S_{3-4}

III етап. Розробка схеми взаємодії задач



Мал 1. Схема взаємодії задач

CS1 – критична секція для захисту CP a, під час обчислення;
CS2 – критична секція для захисту CP a, під час копіювання;
CS3 – критична секція для захисту CP d, під час копіювання;
CS3 – критична секція для захисту CP p, під час копіювання;

Бар’єри використано для синхронізації: при введені, по обчисленню a та Sн, перед виведенням.

IV Етап. Розробка програми

Лістинг програми без використання pragma for

Main.cpp

```
// Лабороторна робота ЛР5 Варіант 6
// A = (R*MC)*MD*p + (B*Z)*E*d
//
// T1: X, e
// T2: C, MA
// T3: R, MD
// T4: B, p
//
// Дубчак Аліна Євгеніївна, група ІМ-13
// Дата 14 05 2024

using namespace std;

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

int *readVector(int size);

int **readMatrix(int n);

void printVector(int *v, int size);

void printMatrix(int **matrix, int rows, int cols);

void insertSubVectorIntoVector(int *subVector, int startIndex, int endIndex, int *S, int statusVector);

int *multiplyVectorAndMatrix(int *vector, int **matrix);

int *getSubVector(int *v, int start, int size);

int vectorMultiplication(int *v1, int *v2, int size);

int **getSubMatrix(int **matrix, int startRow, int startCol, int size);

int *calculateRes(int *S, int** MDh, int p_i, int a_i, int *Eh, int d_i);

const int P = 4;
const int N = 16;
const int H = N / P;


int main() {
    cout << "Lab5 started!" << endl;
    auto start = chrono::high_resolution_clock::now();

    //глобальні змінні
    int d;
    int p;
    int a = 0;
    int *Z;
    int *R;
```

```
  int *B;
  int *E;
  int *S = new int[N];
  int *A = new int[N];
  int *S_MDh;
  int **MC;
  int **MD;

  //локальні змінні
  int T_id;
  int p_i;
  int a_i;
  int d_i;

  omp_set_num_threads(P);

#pragma omp parallel private(T_id, p_i, a_i, d_i) shared(d, a, p, Z, R, B, E, A, MC, MD)
  {
    T_id = omp_get_thread_num() + 1;
#pragma omp critical
    cout << "T" << T_id << " started" << endl;

    switch (T_id) {
      case 1: //T1
        E = readVector(N); //Введення E
        MC = readMatrix(N); //Введення MC
        break;
      case 2: //T2
        d = 1; //Введення d
        MD = readMatrix(N); //Введення MD
        break;
      case 3: //T3
        B = readVector(N); //Введення B
        p = 1; //Введення p
        break;
      case 4: //T4
        R = readVector(N); //Введення R
        Z = readVector(N); //Введення Z

        break;
      default:
        break;
    }
#pragma omp barrier                  //Синхронізація по введенню

    int *Bh = getSubVector(B, (T_id - 1) * H, H); // Отримати субвектор Bh
    int *Zh = getSubVector(Z, (T_id - 1) * H, H); // Отримати субвектор Zh

    a_i = vectorMultiplication(Bh, Zh, H);


#pragma omp critical(CS)             //CS1 - Обчислення 2  a = a + ai
    {
      a += a_i;
    }

#pragma omp barrier            //Синхронізація по Обчисленню 2  a = a + ai
```

```
    int **MCh = getSubMatrix(MC, N, (T_id - 1) * H, T_id * H); // Отримати суматрицю MCh
    int *Sh = multiplyVectorAndMatrix(R, MCh);
    insertSubVectorIntoVector(Sh, (T_id - 1) * H, T_id * H, S, 1);

#pragma omp barrier             //Синхронізація по Обчисленню 3  S =R*MCн

#pragma omp critical(CS) // Копія ai = a
    {
      a_i = a; // CS2
    }

#pragma omp critical(CS)  // Копія di = d
    {
      d_i = d; // CS3
    }

#pragma omp critical(CS) // Копія pi = p
    {
      p_i = p; // CS4
    }

    int **MDh = getSubMatrix(MD, N, (T_id - 1) * H, T_id * H); // Отримати суматрицю MDh
    int *Eh = getSubVector(E, (T_id - 1) * H, H); // Отримати субвектор Eh

    int *Ah = calculateRes(S, MDh, p_i, a_i, Eh, d_i); //??

#pragma omp barrier             //Синхронізація по Обчисленню 4  A = S*MDн*pi + ai*Eн*di

    insertSubVectorIntoVector(Ah, (T_id - 1) * H, T_id * H, A, 1);

    if (T_id == 1) {
#pragma omp critical
      {
        cout << "A: " << endl;
        printVector(A, N);
      }
    }

    // Звільнення пам'яті
    delete [] Bh;
    delete [] Zh;
    delete [] Sh;
    delete [] Eh;
    delete [] Ah;

    for (int i = 0; i < N; ++i) {
      delete[] MCh[i];
      delete[] MDh[i];
    }
#pragma omp critical

    cout << "T" << T_id << " finished" << endl;
  }

  // Звільнення пам'яті
  delete[] S;
  delete[] A;
  delete [] S_MDh;
```

```
    delete[] Z;
    delete[] R;
    delete[] B;
    delete[] E;

    for (int i = 0; i < N; ++i) {
        delete[] MC[i];
        delete[] MD[i];
    }

    auto stop = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::milliseconds>(stop - start);
    cout << "Time:  " << duration.count() << " ms" << endl;
    return 0;
}

int *readVector(int size) {
    int *v = new int[size];
    for (int i = 0; i < size; ++i)
        v[i] = 1;
    return v;
}

int **readMatrix(int n) {
    int **m = new int *[n];
    for (int i = 0; i < n; ++i)
        m[i] = readVector(n);
    return m;
}

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

void printMatrix(int **matrix, int rows, int cols) {
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}

int **getSubMatrix(int **matrix, int rowCount, int startColumn, int endColumn) {
    int columnCount = endColumn - startColumn;
    int **submatrix = new int *[rowCount];
    for (int i = 0; i < rowCount; ++i) {
        submatrix[i] = new int[columnCount];
        for (int j = startColumn; j < endColumn; ++j) {
            submatrix[i][j - startColumn] = matrix[i][j];
        }
    }
    return submatrix;
}

int *getSubVector(int *v, int start, int size) {
    int *subVector = new int[size];
```

```
    for (int i = 0; i < size; ++i) {
        subVector[i] = v[start + i];
    }
    return subVector;
}

int vectorMultiplication(int *v1, int *v2, int size) {
    int result = 0;
    for (int i = 0; i < size; ++i) {
        result += v1[i] * v2[i];
    }
    return result;
}

int *multiplyVectorAndMatrix(int *vector, int **matrix) {
    int *result = new int[H];

    for (int i = 0; i < H; i++) {
        int sum = 0;
        for (int j = 0; j < N; j++) {
            sum += matrix[j][i] * vector[j];
        }
        result[i] = sum;
    }

    return result;
}

void insertSubVectorIntoVector(int *subVector, int startIndex, int endIndex, int *S, int statusVector) {
    for (int i = startIndex; i < endIndex; i++) {
        if (statusVector == 0) {
            subVector[i] = subVector[i - startIndex];
        } else {
            S[i] = subVector[i - startIndex];
        }
    }
}

int *calculateRes(int *S, int** MDh, int p_i, int a_i, int *Eh, int d_i) {
    int *S_MD = multiplyVectorAndMatrix(S, MDh);
    int *result = new int[N];
    for (int i = 0; i < N; ++i) {
        result[i] = S_MD[i] * p_i + a_i * Eh[i] * d_i;
    }
    delete[] S_MD;
    return result;
}
```

Результат виконання програми

```
Lab5 started!
T2 started
T1 started
T3 started
T4 started
A:
272 272 272 272 272 272 272 272 272 272 272 272 272 272 272 272
T3 finished
T2 finished
T4 finished
T1 finished
```

Результат виконання програми при N=16 без використання pragma for

Лістинг програми з використання pragma for

Main.cpp

```
// Лабороторна робота ЛР5 Варіант 6
// A = (R*MC)*MD*p + (B*Z)*E*d
//
// T1: X, e
// T2: C, MA
// T3: R, MD
// T4: B, p
//
// Дубчак Аліна Євгеніївна, група ІМ-13
// Дата 14 05 2024

using namespace std;

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

int *readVector(int size);

int **readMatrix(int n);

void printVector(int *v, int size);

void printMatrix(int **matrix, int rows, int cols);

void insertSubVectorIntoVector(int *subVector, int startIndex, int endIndex, int *S, int statusVector);

int *multiplyVectorAndMatrix(int *vector, int **matrix);

int *getSubVector(int *v, int start, int size);

int vectorMultiplication(int *v1, int *v2, int size);

int **getSubMatrix(int **matrix, int startRow, int startCol, int size);

int *calculateRes(int *S_MDh, int p_i, int a_i, int *Eh, int d_i);

const int P = 4;
const int N = 16;
```

```
const int H = N / P;


int main() {
   cout << "Lab5 started!" << endl;
   auto start = chrono::high_resolution_clock::now();

   //глобальні змінні
   int d;
   int p;
   int a = 0;
   int *Z;
   int *R;
   int *B;
   int *E;
   int *S = new int[N];
   int *A = new int[N];
   int *S_MDh;
   int *SDM = new int[N];
   int **MC;
   int **MD;

   //локальні змінні
   int T_id;
   int p_i;
   int a_i;
   int d_i;

   omp_set_num_threads(P);

#pragma omp parallel private(T_id, p_i, a_i, d_i) shared(d, a, p, Z, R, B, E, A, MC, MD)
   {
      T_id = omp_get_thread_num() + 1;
#pragma omp critical
      cout << "T" << T_id << " started" << endl;

      switch (T_id) {
         case 1: //T1
            E = readVector(N); //Введення E
            MC = readMatrix(N); //Введення MC
            break;
         case 2: //T2
            d = 1; //Введення d
            MD = readMatrix(N); //Введення MD
            break;
         case 3: //T3
            B = readVector(N); //Введення B
            p = 1; //Введення p
            break;
         case 4: //T4
            R = readVector(N); //Введення R
            Z = readVector(N); //Введення Z

            break;
         default:
            break;
      }
#pragma omp barrier                    //Синхронізація по введенню
```

```
    int *Bh = getSubVector(B, (T_id - 1) * H, H); // Отримати субвектор Bh
    int *Zh = getSubVector(Z, (T_id - 1) * H, H); // Отримати субвектор Zh

    a_i = vectorMultiplication(Bh, Zh, H);


#pragma omp critical(CS)              //CS1 - Обчислення 2  a = a + ai
    {
      a += a_i;
    }

#pragma omp barrier             //Синхронізація по Обчисленню 2  a = a + ai

    int **MCh = getSubMatrix(MC, N, (T_id - 1) * H, T_id * H); // Отримати суматрицю MCh
    int *Sh;

    Sh = multiplyVectorAndMatrix(R, MCh);
    insertSubVectorIntoVector(Sh, (T_id - 1) * H, T_id * H, S, 1);

#pragma omp barrier             //Синхронізація по Обчисленню 3  S =R*MCн

#pragma omp critical(CS) // Копія ai = a
    {
      a_i = a; // CS2
    }

#pragma omp critical(CS)  // Копія di = d
    {
      d_i = d; // CS3
    }

#pragma omp critical(CS) // Копія pi = p
    {
      p_i = p; // CS4
    }

    int **MDh = getSubMatrix(MD, N, (T_id - 1) * H, T_id * H); // Отримати суматрицю MDh
    int *Eh = getSubVector(E, (T_id - 1) * H, H); // Отримати субвектор Eh

    S_MDh = multiplyVectorAndMatrix(S, MDh);
    insertSubVectorIntoVector(S_MDh, (T_id - 1) * H, T_id * H, SDM, 1);

#pragma omp parallel for
    for (int i = 0; i < N; i++) {
      A[i] = (SDM[i] * p_i) + (a_i * E[i] * d_i);
    }

#pragma omp barrier             //Синхронізація по Обчисленню 4  A = S*MDн*pi + ai*Eн*di

    if (T_id == 1) {
#pragma omp critical
      {
        cout << "A: " << endl;
        printVector(A, N);
      }
    }

// Звільнення пам'яті
    delete [] Bh;
```

```
        delete [] Zh;
        delete [] Sh;
        delete [] Eh;

        for (int i = 0; i < N; ++i) {
            delete[] MCh[i];
            delete[] MDh[i];
        }

#pragma omp critical
        cout << "T" << T_id << " finished" << endl;
    }

    // Звільнення пам'яті
    delete[] S;
    delete[] A;
    delete [] S_MDh;
    delete [] SDM;
    delete[] Z;
    delete[] R;
    delete[] B;
    delete[] E;

    for (int i = 0; i < N; ++i) {
        delete[] MC[i];
        delete[] MD[i];
    }

    auto stop = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::milliseconds>(stop - start);
    cout << "Time:  " << duration.count() << " ms" << endl;
    return 0;
}

int *readVector(int size) {
    int *v = new int[size];
    for (int i = 0; i < size; ++i)
        v[i] = 1;
    return v;
}

int **readMatrix(int n) {
    int **m = new int *[n];
    for (int i = 0; i < n; ++i)
        m[i] = readVector(n);
    return m;
}

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

void printMatrix(int **matrix, int rows, int cols) {
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            cout << matrix[i][j] << " ";
        }
```

```
        cout << endl;
    }
}

int **getSubMatrix(int **matrix, int rowCount, int startColumn, int endColumn) {
    int columnCount = endColumn - startColumn;
    int **submatrix = new int *[rowCount];
    for (int i = 0; i < rowCount; ++i) {
        submatrix[i] = new int[columnCount];
        for (int j = startColumn; j < endColumn; ++j) {
            submatrix[i][j - startColumn] = matrix[i][j];
        }
    }
    return submatrix;
}

int *getSubVector(int *v, int start, int size) {
    int *subVector = new int[size];
    for (int i = 0; i < size; ++i) {
        subVector[i] = v[start + i];
    }
    return subVector;
}

int vectorMultiplication(int *v1, int *v2, int size) {
    int result = 0;
    for (int i = 0; i < size; ++i) {
        result += v1[i] * v2[i];
    }
    return result;
}

int *multiplyVectorAndMatrix(int *vector, int **matrix) {
    int *result = new int[H];

    for (int i = 0; i < H; i++) {
        int sum = 0;
        for (int j = 0; j < N; j++) {
            sum += matrix[j][i] * vector[j];
        }
        result[i] = sum;
    }

    return result;
}

void insertSubVectorIntoVector(int *subVector, int startIndex, int endIndex, int *S, int statusVector) {
    for (int i = startIndex; i < endIndex; i++) {
        if (statusVector == 0) {
            subVector[i] = subVector[i - startIndex];
        } else {
            S[i] = subVector[i - startIndex];
        }
    }
}

int *calculateRes(int *S_MDh, int p_i, int a_i, int *Eh, int d_i) {
    int *result = new int[N];
    for (int i = 0; i < N; ++i) {
```

```
        result[i] = S_MDh[i] * p_i + a_i * Eh[i] * d_i;
    }
    return result;
}
```

Результат виконання програми

```
T3 started
A:
272 272 272 272 272 272 272 272 272 272 272 272 272 272 272 272
T1 finished
T3 finished
T2 finished
T4 finished
```

Результат виконання програми при N=16 з використанням pragma for

Тестування

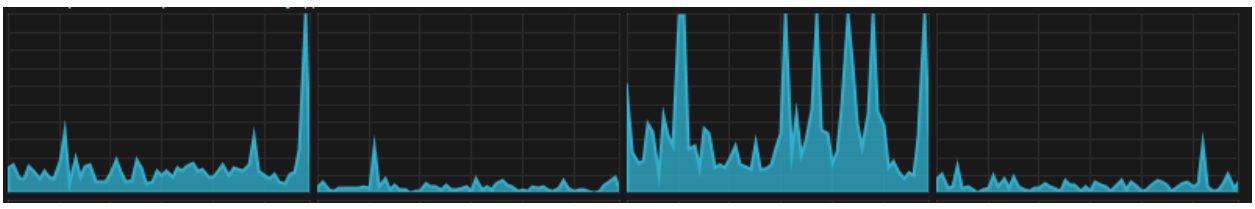
Опис комп’ютера:
Процесор: AMD Ryzen 5 5500U
Кількість ядер: 6
Кількість логічних ядер: 12

Для N=1000, було проаналізовано процес завантаження програмою ядер багатоядерного процесора за допомогою **Диспетчеру задач ОС Windows** та вираховано середню швидкість програми

Програма, в якій використовуються pragma for для:

1 ядра:
- 250 ms
- 288 ms
- 307 ms
- 381 ms
- 299 ms

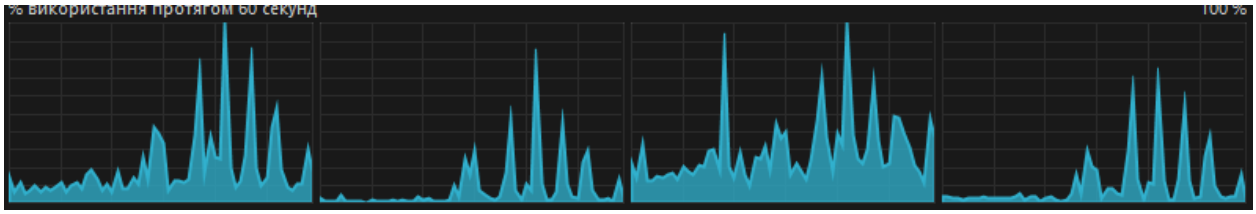
$C_{знач}$ =**(250+288+307+381+299)/5 = 305ms**



Навантаження на 1 ядро

4 ядра
- 201 ms
- 239 ms
- 170 ms
- 193 ms
- 202 ms

$C_{знач}$ =**(201+239+170+193+202)/5 = 201ms**



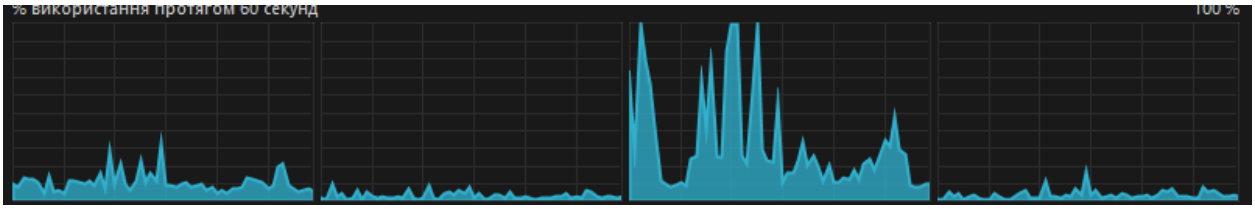
Навантаження на 4 ядра

Кп = **305ms/ 201ms ~ 1,52**

Програма, в якій не використовуються pragma for для:

1 ядра:
- 350 ms
- 291 ms
- 321 ms
- 381 ms
- 287 ms

$C_{знач}$ =**(350+291+321+381+287)/5 = 326ms**

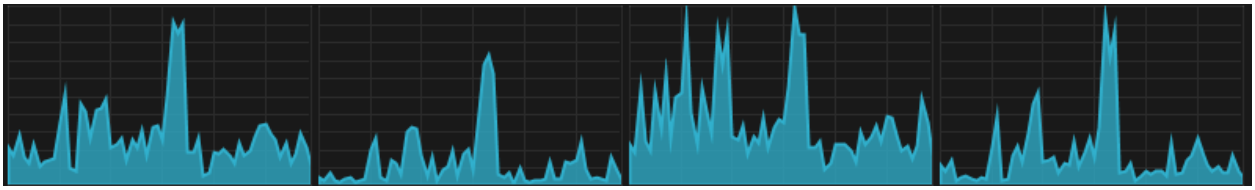


Навантаження на 1 ядро

4 ядра
- 222 ms
- 208 ms
- 171 ms
- 195 ms
- 217 ms

$C_{знач}$ =**(222+208+171+195+217)/5 = 213ms**

Кп = **326ms/ 213ms ~ 1,53**



Навантаження на 4 ядра

ВИСНОВОК

1. У процесі розробки було паралельний математичний алгоритм, який дає змогу виконати обчислення заданої формули та визначити спільні ресурси: R, S, a, d, p
2. Був розроблений алгоритм потоків, в яких визначені відповідні точки синхронізації – введення даних, виконання обчислень, виведення. Визначені критичні ділянки(КД1-4).
3. Було створено схему взаємодії задач, де визначено та позначено засоби організації взаємодії потоків: критичні секції для захисту спільних ресурсів та бар'єри для синхронізації взаємодії потоків.
4. Розробка програми виконувалася на мові C++. Для роботи з потоками було використано бібліотеку OpenMP. У процесі розробки було використано: #pragma omp parallel – для створення потоків. private – для задання локальних змінних – для кожного потоку буде створена копія цих змінних, omp_set_num_threads(P) – задання кількості потоків, omp_get_thread_num() – отримаеня номеру потоку(1-4), #pragma omp critical – критичні секції, #pragma omp barrier – бар'єри та #pragma omp parallel for – щоб автоматично розділити ітерації виконання циклу паралельно між потоками
5. Було проведено тестування ефективності багатопотокової програми для значення N = 1000. Результати показали, що коефіцієнт прискорення програми без використання pragma for становить 1,52, а з його використанням - 1,53. Середня швидкість виконання програми з pragma for та без нього на чотирьох ядрах складає відповідно 201 мс та 213 мс. Це свідчить про те, що різниця в швидкості виконання незначна, і обидві програми працюють майже з однаковою ефективністю.