ЛАБОРАТОРНА РОБОТА 2
ОСНОВИ OpenMP. Робота з матрицями

Завдання 1

1. Визначте, яку версію OpenMP підтримує компілятор на доступній системі.

Для перевірки того, що компілятор підтримує якусь версію OpenMP, достатньо написати директиви умов компіляції **#ifdef** або **#ifndef**.

Приклад такої компіляції на мові С:

```
#include
int main(){
#ifdef _OPENMP
printf("OpenMP is supported!\n");
#endif
}
```

2. За допомогою функцій OpenMP визначте час, потрібній системі для роботу функції вимірбвання час. Визначити точність системного таймера.

Якщо деякий участок системи оточити викликами функції **omp_get_wtime()**, то різниця повертаємих значень покаже час роботи даного участку коду. Функція **omp_get_wtick()** повертає в деякому участку коду час у секундах на момент виклику. Цей час можна розглядати як точність системного таймеру.

```
int main(int argc, char *argv[])
{
double start_time, end_time, tick;
start_time = omp_get_wtime();
end_time = omp_get_wtime();
```

```
tick = omp_get_wtick();
printf("Час на вимірювання часу %lf\n", end_time-start_time);
printf("Точність таймеру %lf\n", tick);
}
```

3. Напишіть просту програму, яка виводить «Hello World» у паралельній області.

```
#include <omp.h>
main () {
/* Выделение параллельного фрагмента*/
#pragma omp parallel
{
printf("Hello World !\n");
}/* Завершение параллельного фрагмента */ }
```

Завдання 2

Реалізувати множення квадратних матриць з використанням OpenMP.

В цьому завданні найскладніше – зрозуміти яку саме область програми потрібно розпаралелити.

Алгоритм множення матриць:

$$A \times B = C$$

Множення матриць $(A \times B)$ – є операція обчислення матриці C, кожен елемент якої дорівнює сумі добутків елементів у відповідному рядку першого множника і стовпці другого.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Кількість стовпців в матриці A має збігатися з кількістю рядків у матриці B, іншими словами, матриця A обов'язково має бути ***узгодженою*** з матрицею B.

Якщо матриця A має розмірність $m \times n$, B – $n \times k$, то розмірність їхнього добутку $A \times B = C$ є $m \times k$.

В запропонованому прикладі будемо обмежуватися квадратними матрицями.

```
int n ; // розмір матриці
  double sum;
  int i, j, k;
...
  double *MatrixA=new double [n*n];
  double *MatrixB=new double [n*n];
  double *MatrixC=new double [n*n];
...
#pragma omp parallel for private(j,k,sum)
 for(i=0;i<n;i++)
 {
      for(k=0;k<n;k++)
      {
              sum=0;
              for(j=0;j<n;j++)
             {
                sum+=MatrixA[i*n+j]*MatrixB[j*n+k];

             }
              MatrixC[i*n+k]=sum;
      }

 }
```

Ще один приклад коду:

```
#include <stdio.h>
#include <omp.h>
#define N 4096
double a[N][N], b[N][N], c[N][N];
int main()
{
int i, j, k;
double t1, t2;
// инициализация матриц
   for (i=0; i<N; i++)
      for (j=0; j<N; j++)
         a[i][j]=b[i][j]=i*j;
```

```
   t1=omp_get_wtime();
// основной вычислительный блок
#pragma omp parallel for shared(a, b, c) private(i, j, k)
   for(i=0; i<N; i++){
      for(j=0; j<N; j++){
         c[i][j] = 0.0;
         for(k=0; k<N; k++) c[i][j]+=a[i][k]*b[k][j];
      }
   }
   t2=omp_get_wtime();
   printf("Time=%lf\n", t2-t1);
}
```

Завдання 3

1. Вивести суму елементів строки матриці.

```
#include <omp.h>
#define CHUNK 100
#define NMAX 1000
main () {
int i, j, sum;
float a[NMAX][NMAX];
<инициализация данных>
#pragma omp parallel shared(a) private(i,j,sum) {
#pragma omp for
for (i=0; i < NMAX; i++) {
    sum = 0;
    for (j=0; j < NMAX; j++)
sum += a[i][j];
printf ("Сумма элементов строки %d равна %f\n",i,sum);
} /* Завершение параллельного фрагмента */ }
```

2. Вивести суму всіх елементів всіх строк матриці з використанням редукції.

Директива ***parallel***

Паралельна область задається за допомогою директиви ***parallel***

```
#pragma omp parallel [опция[[,] опция]...]
```

- **Private (**список**)-** задає список змінних, для яких робиться локальна копія для кожної нитки.
- **Shared(**список**)-** Задає список змінних, що спільний для всіх ниток.
- **Reduction**(оператор:список) – задає оператори та список спільних змінних, для кожної змінної створюється локальна копія в кожній нитці. Локальні копії ініціалізуються згідно типу оператору (адитивний-0, мультиплікативний-1).

```
total = 0;
#pragma omp parallel for shared(a) private(i,j,sum) reduction (+:total)
{
for (i=0; i < NMAX; i++) {
    sum = 0;
    for (j=i; j < NMAX; j++)
sum += a[i][j];
printf ("Сумма элементов строки %d равна %f\n",i,sum); total = total + sum;
} /* Завершение параллельного фрагмента */
printf ("Общая сумма элементов матрицы равна %f\n",total);
```

Порядок виконання лабораторної роботи:

1) Написати програму для одного потоку.
2) Реалізувати багатопоточність.
3) Порівняти час затрачений на обчислення в однопоточній та багатопоточній реалізаціях завдань 1 та 2.