Лабораторна робота №1

Знайомство з технологією паралельного програмування OpenMP. Модель данних.

Мета: Вивчити основи програмування і виробити практичні навики роботи з технологією OpenMP.

1.Вступ

Одним з найбільш популярних засобів програмування для комп'ютерів із загальною пам'яттю, що базуються на традиційних мовах програмування й використанні спеціальних коментарів, у цей час є технологія OpenMP. Цю технологію можна розглядати як високорівневу надбудову над Pthreads (або аналогічними бібліотеками потоків).

Распараллелювання в OpenMP виконується явно за допомогою вставки в текст програми спеціальних директив, а також виклику допоміжних функцій. При використанні OpenMP передбачається SPMD - Модель (Single Program Multiple Data) паралельного програмування, у рамках якої для всіх паралельних потоків використовується той самий код.

2. Директиви та функції

Значна частина функціональності OpenMP реалізується за допомогою директив компілятору. Вони повинні бути явно вставлені користувачем, що дозволить виконувати програму в паралельному режимі. Директиви OpenMP в програмах на мові C починаються з **# pragma omp**:

#pragma omp directive-name [опція**[[,]**опція**]...]**

Об'єктом дії більшості директив є один оператор або блок перед яким розташована директива у вихідному тексті програми. У OpenMP такі оператори або блоки називаються асоційованими з директивою. Асоційований блок повинен мати одну точку входу на початку і одну точку виходу в кінці. Порядок опцій в описі директиви неістотний, в одній директиві більшість опцій можуть зустрічатися кілька разів. Після деяких опцій може слідувати список змінних, що розділяються комами.

Всі директиви OpenMP можна розділити на 3 категорії: визначення паралельної області, розподіл роботи, синхронізація. Кожна директива може мати декілька додаткових атрибутів – опцій (clause). Окремо специфікуються опції для призначення класів змінних, які можуть бути атрибутами різних директив.

У момент запуску програми породжується єдиний потік-майстер або «основний» потік, який починає виконання програми з першого оператора. Основний потік і лише він виконує всі послідовні області програми. При вході в паралельну область породжуються додаткові потоки.

2.1.Директива parallel

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

#pragma omp parallel [опція **[[,]**опція**]...]**

Можливі опції:

- **if**(умова) – виконання паралельної області по умові. Входження в паралельну область здійснюється лише при виконанні деякої умови. Якщо умова не виконана, то директива не спрацьовує і продовжується обробка програми в колишньому режимі;
- **num_threads** (цілочисельне вираження) – явне завдання кількості потоків, які виконуватимуть паралельну область; за умовчанням вибирається останнє значення, встановлене за допомогою функції **omp_set_num_threads()** або значення змінної **Omp_num_threads**;
- **default(private|firstprivate|shared|none)** – всім змінним в паралельної області, яким явно не призначений клас, буде призначений клас **private**, **firstprivate** або **shared** відповідно; **none** означає, що всім змінним в паралельної області клас має бути призначений явно;
- **private** (список) – задає список змінних, для яких породжується локальна копія в кожному потоці; початкове значення локальних копій змінних із списку не визначене;
- **firstprivate** (список) – задає список змінних, для яких породжується локальна копія в кожному потоці; локальні копії змінних ініціалізувалися значеннями цих змінних в потоці-майстрові;
- **shared** (список) – задає список змінних, загальних для всіх потоків;
- **copyin**(список) – задає список змінних, оголошених як **readprivate**, які при вході в паралельну область ініціалізувалися значеннями відповідних змінних в потоці-майстрові;
- **reduction**(оператор:список) – задає оператор і список загальних змінних; для кожної змінної створюються локальні копії в кожному потоці; над локальними копіями змінних після виконання всіх операторів паралельної області виконується заданий оператор; оператор для мови С: – +, *, - &, |, ^, &&, ||;

При вході в паралельну область породжуються нові **Omp_num_threads**-1 потоки, кожен потік отримує свій унікальний номер, причому потік, що породжує, отримує номер 0 і стає основним потоком групи («майстром»). Інші потоки отримують номер з 1 до **Omp_num_threads**-1. Кількість потоків, що виконують дану паралельну область, залишається незмінною до моменту виходу з області. При виході з паралельної області виробляється неявна синхронізація і знищуються всі потоки, окрім того, що породив.

Нижче наведений приклад використання директиви **parallel**:

```
#include <omp.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
     printf("Послідовна область 1\n");
     #pragma omp parallel
     {
          printf("Паралельна область \n");
     }
     printf("Послідовна область 2\n");
}
```

Приклад демонструє вживання опції **reduction**:

```
#include <omp.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
```

```
    int count = 0;
    #pragma omp parallel reduction (+: count)
    {
        count++;
        printf("Поточне значення count: %d\n", count);
    }
   printf("Число ниток: %d\n", count);
}
```

Перед запуском програми кількість потоків, що виконують паралельну область, можна задати, визначивши значення змінної середовища **omp_num_threads**. Значення за умовчуванням змінної **omp_num_threads** залежить від реалізації. З програми її можна змінити за допомогою виклику функції **omp_set_num_threads ():**

void omp_set_num_threads(int num);

Приклад демонструє вживання функції **omp_set_num_threads()** і опції **num_threads**. Перед першою паралельною областю викликом функції **omp_set_num_threads(2)** виставляється кількість потоків, рівна 2. Але до першої паралельної області застосовується опція **num_threads(3),** яка вказує, що дану область слід виконувати трьома потоками. Отже, повідомлення "Паралельна область 1" буде виведено трьома потоками. До другої паралельної області опція **num_threads** не застосовується, тому діє значення встановлене функцією **omp_set_num_threads(2),** і повідомлення "Паралельна область 2" буде виведено двома потоками.

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
    omp_set_num_threads(2);
#pragma omp parallel num_threads(3)
    {
        printf("Паралельна область 1\n");
    }
#pragma omp parallel
    {
        printf("Паралельна область 2\n");
    }
}
```

В деяких випадках система може динамічно змінювати кількість потоків, використовуваних для виконання паралельної області, наприклад, для оптимізації використання ресурсів системи. Це дозволено робити, якщо змінна середовища **omp_dynamic** встановлена в true. Змінну **omp_dynamic** можна встановити за допомогою функції **omp_set_dynamic():**

void omp_set_dynamic(int num);

Взнати значення змінної **Omp_dynamic** можна за допомогою **omp_get_dynamic():**

int omp_get_dynamic(void);

Функція **omp_get_max_threads()** повертає максимально допустиме число потоків для використання в наступній паралельній області:

int omp_get_max_threads(void);

Паралельні області можуть бути вкладеними; за умовчанням вкладена паралельна область виконується одним потоком. Змінити значення змінної **Omp_nested** можна за допомогою виклику функції **omp_set_nested()**:

void omp_set_nested(int nested)

Функція **omp_set_nested()** дозволяє або забороняє вкладений паралелізм. Якщо вкладений паралелізм дозволений, то кожен потік, в якому зустрінеться опис паралельної області, породить для її виконання нову групу потоків. На мові С значення параметра задається 0 або 1. Сам потік, що породив, стане в новій групі потоком-майстром. Якщо система не підтримує вкладений паралелізм, дана функція не матиме ефекту. Взнати значення змінної **omp_nested** можна за допомогою **omp_get_nested()**:

int omp_get_nested(void);

Функція **omp_in_parallel()** повертає 1, якщо вона була викликана з активної паралельної області програми:

int omp_in_parallel(void);

2.2.Директива single

Якщо в паралельній області яка-небудь ділянка коду має бути виконаний лише один раз, то його потрібно виділити директивами **single**:

#pragma omp single [опція **[[,]**опція**]...]**

Можливі опції:

- **private** (список) – задає список змінних, для яких породжується локальна копія в кожному потоці; початкове значення локальних копій змінних із списку не визначене;
- **firstprivate** (список) – задає список змінних, для яких породжується локальна копія в кожному потоці; локальні копії змінних ініціалізувалися значеннями цих змінних в потоці-майстрові;
- **copyprivate**(список) – після виконання потока, що містить конструкцію **single**, нові значення змінних списку будуть доступні всім однойменним приватним змінним (**private** і **firstprivate**), описаним на початку паралельної; опція не може використовуватися спільно з опцією **nowait**;
- **nowait** – після виконання виділеної ділянки відбувається неявна бар'єрна синхронізація паралельно працюючих потоків: їх подальше виконання відбувається лише тоді, коли всі вони досягнуть даної точки; якщо в подібній затримці немає необхідності, опція **nowait** дозволяє потокам, що вже дійшли до кінця ділянки, продовжити виконання без синхронізації з останніми.

Який саме потік виконуватиме виділену ділянку програми, не специфікується. Один потік виконуватиме даний фрагмент, а всі інші потоки чекатимуть завершення її роботи, якщо лише не вказана опція **nowait**. Необхідність використання директиви **single** часто виникає при роботі із загальними змінними.

Приклад ілюструє вживання директиви **single** разом з опцією **nowait**. Спочатку всі потоки надрукують текст "Повідомлення 1", при цьому один потік (не обов'язково потік-майстер) додатково надрукує текст "Один потік". Інші потоки, не чекаючи завершення виконання області **single**, надрукують текст "Повідомлення 2". Таким чином, перша поява "Повідомлення 2" у виводі може зустрітися як до тексту "Один потік", так і після нього. Якщо прибрати опцію **nowait**, то після закінчення області **single** станеться бар'єрна синхронізація, і жодна видача "Повідомлення 2" не може з'явитися до видачі "Один потік".

```
#include <stdio.h>
int main(int argc, char *argv[])
{
#pragma omp parallel
      {
            printf("Повідомлення 1\n");
            #pragma omp single nowait
            {
              printf("Один потік \n");
            }
            printf("Повідомлення 2\n");
      }
}
```

2.3.Директива master

Директива виділяє ділянку коду, яка буде виконана лише потоком-майстром. Інші потоки просто пропускають дану ділянку і продовжують роботу з оператора, розташованого слідом за ним. Неявної синхронізації дана директива не передбачає:

#pragma omp master

Приклад демонструє вживання директиви **master**. Змінна n є локальною, тобто кожен потік працює зі своїм екземпляром. Спочатку всі потоки привласнять змінній n значення 1. Потім потік-майстер привласнить змінній n значення 2, і всі потоки надрукують значення n. Потім потік-майстер привласнить змінній n значення 3, і знову всі потоки надрукують значення n. Видно, що директиву **master** завжди виконує один і той же потік. У даному прикладі всі потоки виведуть значення 1, а потік-майстер спочатку виведе значення 2, а потім - значення 3.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
      int n;
      #pragma omp parallel
      {
            n=1;
```

```
    #pragma omp master
        {
            n=2;
        }
      printf("Перше значення n: %d\n", n);
    #pragma omp barrier
    #pragma omp master
        {
            n=3;
        }
    printf("Друге значення n: %d\n", n);
     }
}
```

3.Модель даних

Модель даних в OpenMP передбачає наявність як загальної для всіх потоків області пам'яті, так і локальної області пам'яті для кожного потоку. У OpenMP змінні в паралельних областях програми розділяються на два основні класи:

shared (загальні; всі потоки бачать одну і ту ж змінну);

private (локальні, приватні; кожен потік бачить свій екземпляр даної змінної).

Загальна змінна завжди існує лише в одному екземплярі для всієї зони дії і доступна всім потокам під одним і тим же ім'ям. Оголошення локальної змінної викликає породження свого екземпляра даною змінною (того ж типа і розміру) для кожного потоку. Зміна потоком значення своєї локальної змінної ніяк не впливає на зміну значення цій же локальній змінній в інших потоках.

Якщо декілька потоків одночасно записують значення загальної змінної без виконання синхронізації або якщо як мінімум один потік читає значення загальної змінної і як мінімум один потік записує значення цієї змінної без виконання синхронізації, то виникає ситуація так званої «гонки даних» (data race), при якій результат виконання програми непередбачуваний.

За умовчанням, всі змінні, породжені поза паралельною областю, при вході в цю область залишаються загальними (**shared**). Виняток становлять змінні, що є лічильниками ітерацій в циклі, по очевидних причинах. Змінні, породжені усередині паралельної області, за умовчанням є локальними (**private**). Явно призначити клас змінних за умовчанням можна за допомогою опції **default**. Не рекомендується постійно покладатися на правила за умовчанням, для більшої надійності краще завжди явно описувати класи використовуваних змінних, вказуючи в директивах OpenMP опції **private**, **shared**, **firstprivate**, **lastprivate, reduction**.

Приклад демонструє використання опції **private**. У даному прикладі змінна n оголошена як локальна змінна в паралельній області. Це означає, що кожен потік працюватиме зі своєю копією змінної n, при цьому на початку паралельної області на кожному потоці змінна n не буде ініціалізована. В ході виконання програми значення змінної n буде виведено в чотирьох різних місцях. Перший раз значення n буде виведено в послідовній області, відразу після привласнення змінній n значення 1. Другий раз всі потоки виведуть значення своєї копії змінною n на початку паралельної області. Далі всі

потоки виведуть свій порядковий номер, отриманий за допомогою функції **omp_get_thread_num()** і привласнений змінній n. Після завершення паралельної області буде ще раз виведено значення змінної n, яке виявиться рівним 1 (не змінилося під час виконання паралельної області).

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
      int n=1;
      printf("n у послідовній області (початок): %d\n", n);

     #pragma omp parallel private(n)
      {
        printf("Значення n на потоці (на вході): %d\n", n);
        n=omp_get_thread_num();
        printf("Значення n на потоці (на виході): %d\n", n);
      }

      printf("n у послідовній області (кінець): %d\n", n);
}
```

Приклад демонструє використання опції **shared**. Масив m оголошений загальним для всіх потоків. На початку послідовної області масив m заповнюється нулями і виводиться на друк. У паралельної області кожен потік знаходить елемент, номер якого збігається з порядковим номером потоку в загальному масиві, і привласнює цьому елементу значення 1. Далі, в послідовній області друкується змінений масив m.

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
      int i, m[10];
      printf("Масив m на початку:\n");
      for (i=0; i<10; i++){
            m[i]=0;
            printf("%d\n", m[i]);
      }
#pragma omp parallel shared(m)
      {
            m[omp_get_thread_num()]=1;
      }
      printf("Масив m в кінці:\n");
      for (i=0; i<10; i++) printf("%d\n", m[i]);
}
```

Приклад демонструє використання опції **firstprivate**. Змінна n оголошена як **firstprivate** в паралельній області. Значення n буде виведено в чотирьох різних місцях. Перший раз значення n буде виведено в послідовній області відразу після ініціалізації. Другий раз всі потоки виведуть значення своєї копії змінною n на початку паралельної області, і це значення дорівнюватиме 1. Далі, за допомогою функції **omp_get_thread_num()** всі потоки

привласнять змінній n свій порядковий номер і ще раз виведуть значення n. У послідовній області буде ще раз виведено значення n, яке знову виявиться рівним 1.

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
      int n=1;
      printf("Значення n на початку: %d\n", n);
      #pragma omp parallel firstprivate(n)
      {
      printf("Значення n на потоці (на вході): %d\n", n);
      n=omp_get_thread_num();
      printf("Значення n на потоці (на виході): %d\n", n);
}
printf("Значення n в кінці: %d\n", n);
}
```

Директива **threadprivate** вказує, що змінні із списку мають бути розмножені з тим, щоб кожен потік мав свою локальну копію:

#pragma omp threadprivate(список**)**

Приклад демонструє використання директиви **threadprivate**. Глобальна змінна n оголошена як **threadprivate** змінна. Значення змінної n виводиться в чотирьох різних місцях. Перший раз всі потоки виведуть значення своєї копії змінної n на початку паралельної області, і це значення дорівнюватиме 1 на потоці-майстрові і 0 на інших потоках. Далі за допомогою функції **omp_get_thread_num()** всі потоки привласнять змінній n свій порядковий номер і виведуть це значення. Потім в послідовній області буде ще раз виведено значення змінної n, яке виявиться рівним порядковому номеру потока-майстра, тобто 0. Востаннє значення змінної n виводиться в новій паралельній області, причому значення кожної локальної копії повинне зберегтися.

```
#include <stdio.h>
#include <omp.h>
int n;
#pragma omp threadprivate(n)
int main(int argc, char *argv[])
{
      int num;
      n=1;
      #pragma omp parallel private (num)
            {
            num=omp_get_thread_num();
            printf("Значення n на потоці %d (на вході): %d\n", num, n);
            n=omp_get_thread_num();
            printf("Значення n на потоці %d (на виході): %d\n", num, n);
            }
      printf("Значення n (середина): %d\n", n);
      #pragma omp parallel private (num)
            {
            num=omp_get_thread_num();
```

```
            printf("Значення n на потоці %d (ещё раз): %d\n", num, n);
            }
}
```

Якщо необхідно змінну, оголошену як **threadprivate**, ініціалізувати значенням розмножуваної змінної з потоку-майстра, то на вході в паралельну область можна використати опцію **copyin**. Якщо значення локальної змінної або змінної, оголошеної як **threadprivate**, необхідно переслати від одного потоку всім, що працюють в даній паралельній області, для цього можна використовувати опцію **copyprivate** директиви **single**.

Приклад демонструє використання опції **copyin**. Глобальна змінна n визначена як **threadprivate**. Вживання опції **copyin** дозволяє ініціалізувати локальні копії змінної n початковим значенням потоку-майстра. Всі потоки виведуть значення n, рівне 1.

```
#include <stdio.h>
int n;
#pragma omp threadprivate(n)
int main(int argc, char *argv[])
{
      n=1;
      #pragma omp parallel copyin(n)
      {
            printf("Значение n: %d\n", n);
      }
}
```

Хід роботи

Створити програму яка повинна реалізувати наступні дії:

1.Створити матрицю A розміром m * n, елементи якої заповнюються рандомно, m задає кількість рядків і кількість потоків, які виконуватимуть паралельну область програми, n задає кількість стовпців. Змінні можуть задаватися в програмному коді або вводитися з клавіатури.

2. У паралельній області за допомогою директиви single або master вивести наступні дані: номер лабораторної роботи; назва лабораторної роботи; групу студента; ФІО студента; номер варіанту; завдання.

3.Обробити паралельним способом матрицю відповідно до свого варіанту. Кожен потік повинен обробляти свій рядок матриці. Результати обробки записати в масив B.

4.Вивести результат обробки масиву паралельним способом. При роботі потоку на екран повинна виводитися інформація про номер потоку та номер рядка матриці, яку обробляє потік.

5.Послідовно обробити матрицю відповідно до свого варіанту. Результати обробки записати в масив C.

6.Вивести результат обробки матриці послідовним способом. Визначити час, який був затрачений на обробку паралельним та послідовним способом.

Порівняти отримані результати на наявність ідентичності.

№ Варіанта	Завдання
1	Знайти мінімальне значення кожного рядка матриці
2	Знайти максимальне значення кожного рядка матриці
3	Порахувати кількість додатних елементів в кожному рядку матриці
4	Порахувати кількість від'ємних елементів в кожному рядку матриці
5	Порахувати кількість нульових елементів в кожному рядку матриці
6	Порахувати суму елементів в кожному рядку матриці
7	Порахувати суму додатних елементів в кожному рядку матриці
8	Порахувати суму від'ємних елементів в кожному рядку матриці

Контрольні питання

1. У яких випадках може бути необхідне використання опції **if** директиви **parallel**?
2. Чим відрізняються директиви **single** і **master**?
3. Чи може нитка-майстер виконати область, що асоціюється з директивою **single**?
4. Чи може нитка з номером 1 виконати область, що асоціюється з директивою **master**?
5. Чи може одна і та ж змінна виступати в одній частині програми як загальна, а в іншій частині – як локальна?
6. Що станеться, якщо декілька ниток одночасно звернуться до загальної змінної?
7. Чи може статися конфлікт, якщо декілька ниток одночасно звернуться до однієї і тієї ж локальної змінної?
8. Яким чином при вході в паралельну область розіслати всім по народжуваних нитках значення деякої змінної?
9. Чи можна зберегти значення локальних копій загальних змінних після завершення паралельної області?
10. У чому відмінність опції **copyin** від опції **firstprivate**?