1. **Загальні відомості про OpenMP**
2. **Конструкції OpenMP**
3. **Особливості реалізації директив OpenMP**

Переваги OpenMP:

1. Інкрементальне розпаралелювання для програми з великими паралельними циклами - послідовно додаються в текст послідовної програми OpenMP-директиви.
2. OpenMP - надає можливості контролю над поведінкою паралельного додатку.
3. OpenMP-программа на однопроцесорній платформі може бути використана як послідовна програма.
4. Підтримка директиви синхронізації і розподілу роботи, що можуть не входити безпосередньо в лексичний контекст паралельної області.

Основні конструкції OpenMP - це директиви компілятора або директиви препроцесора (#pragma) мови C/C++.
Вигляд директиви OpenMP директиви:

#pragma omp конструкція [Речення [Речення] .]

Загальний вигляд програми OpenMP

```
#include <omp.h>
int main()
{   // послідовний код

#pragma omp parallel
   {    // паралельний код
                …
   }

// послідовний код
   return 0;
}
```

Загальний вигляд основних директив OpenMP у C++.

```
# pragma omp parallel    \
                private (var1, var2, .) \
                shared (var1, var2, .) \
                firstprivate (var1, var2, .) \
                lastprivate (var1, var2, .) \
                copyin (var1, var2, .) \
                reduction (operator: var1, var 2, .)\
                if (expression)                    \
                default (shared | none)            \
{

[ Структурний блок програми]

}
```

Директиви shared, private і default
використовуються для опису типів змінних усередині паралельних потоків.

shared(var1, var2, ., varN)
визначає змінні var1, var2, ., varN як загальні змінні для усіх потоків. Вони розміщуються в одній і тій же області пам'яті для усіх потоків.

private(var1, var2, ., varN)
визначає змінні var1, var2, ., varN як локальні змінні для кожного з паралельних потоків. У кожному з потоків ці змінні мають власні значення і відносяться до різних областей пам'яті: локальним областям пам'яті кожного конкретного паралельного потоку.

default (shared | private | none)
задає тип усіх змінних, визначених за замовченням в наступному паралельному структурному блоці як shared, private або none.

Приклад програми на мові Із з використанням OpenMP

```
#include "omp.h"
#include <stdio.h>
double f(double x)
{
return 4.0  / (1 + x * x);
}
main()
{
const long N = 100000;
long i;
double h, sum, x; sum = 0;
h = 1.0  / N;
#pragma omp parallel shared(h)\
                              private(x)
reduction(+: sum)
for  (i = 0;  i < N;  i++)
{ x = h * (i + 0.5);
sum = sum + f(x);
}
}
printf("PI = %f\n",  sum / N);
}
```

Директиви firstprivate і lastprivate
використовуються для опису локальних змінних, що ініціалізувалися усередині паралельних потоків.

firstprivate(var1, var2, .., varN)
визначає змінні var1, var2, .., varN як локальні змінні для кожного з паралельних потоків, причому ініціалізація значень цих змінних відбувається на самому початку паралельного структурного блоку по значеннях з передування послідовного структурного блоку програми.

lastprivate(var1, var2, .., varN)
визначає змінні var1, var2, .., varN як локальні змінні для кожного з паралельних потоків, причому значення цих змінних зберігаються при виході з паралельного структурного блоку за умови, що паралельні потоки виконувалися в послідовному порядку.

```
void main()
{ int myid, c;
       c=98;
       #pragma omp parallel private (myid)
  {
    #pragma omp firstprivate (c)
       { myid = omp_get_thread_num ( )
         printf("T: %i c=%i", myid, c)
       }
   }
 }
```

```
{
#pragma omp parallel shared ( x)\
lastprivate ( i)
 {
   for( i = 1, i<N, i++)
     x (i) = a
   }
  N = i;
}
```

Директива if

використовується для організації умовного виконання потоків в паралельному структурному блоці.

Директива reduction

дозволяє зібрати разом в головному потоці результати обчислень часткових сум, різниць і тому подібне з паралельних потоків наступного паралельного структурного блоку.

if(expression)

визначає умова виконання паралельних потоків в наступному паралельному структурному блоці. Якщо вираження expression набуває значення TRUE, то потоки в наступному паралельному структурному блоці виконуються. Інакше, коли вираження expression набуває значення FALSE, потоки в наступному паралельному структурному блоці не виконуються.

reduction(operator | intrinsic: var1 [, var2,.., varN])

- визначається operator - операції (+-, *, / тощо) або функції, для яких обчислюватимуться відповідні часткові значення в паралельних потоках наступного паралельного структурного блоку.
- визначається список локальних змінних var1, var2, .., varN, в якому зберігатимуться відповідні часткові значення.
- після завершення усіх паралельних процесів часткові значення складаються (віднімаються, перемножуються і тому подібне), і результат зберігається в однойменній загальній змінній.

```
#pragma omp parallel shared (x) private (i)\
                            reduction(+: sum)
for  (i = 0;  i < N;  i++)
{ x = h * (i + 0.5);
sum = sum + f(x);
}
```

```
#pragma omp parallel if ( n>2000 )
{
    for(i=1; i<n i = 1; i++)
        a (i) = b (i)*c + d (i)
}
```

Директива copyin

використовується для передачі даних з головного потоку в паралельні потоки, званою міграцією даних.

Директива for

означає, що оператор for, що йде за цією пропозицією, виконуватиметься в паралельному режимі, тобто кожній петлі цього циклу відповідатиме власний паралельний потік.

copyin(var1 [, var2[, ..[, varN]]])

визначає список локальних змінних var1, var2, .., varN, яким привласнюються значення з однойменних загальних змінних, заданих в глобальному потоці.

```
#pragma omp parallel shared (a, b) private (j)
{
            #pragma omp  for
              for (j=0; j<N; j++)
        a [j] = a [j] + b [j];
}
```

Директива sections

використовується для виділення ділянок програми в області паралельних структурних блоків, що виконуються в окремих

#pragma omp sections

виконується в окремому паралельному потоці. Загальне ж число паралельних потоків дорівнює кількості структурних блоків (section), перерахованих прагм

#pragma omp sections [Речення [Речення.]]

{#pragma omp section
structured block
[#pragma omp section
structured block…]}

```
#pragma omp parallel
  {#pragma omp sections
    {#pragma omp section
      {Func1( )}
      #pragma omp section
       {Func2( )}
      #pragma omp section
      {Func3( )}}.
```

Директива single

використовується для виділення ділянок програми в області паралельних структурних блоків, що виконуються тільки в одному з паралельних потоків.

У усіх інших паралельних потоках виділена директивою single ділянка програми не виконується, проте паралельні процеси, що виконуються в інших потоках, чекають завершення виконання виділеної ділянки програми, тобто неявно реалізується процедура синхронізації.

Для запобігання цьому очікуванню у разі потреби можна використовувати директиву nowait.

Структурний блок (structured block), що йде за #pragma omp single виконується тільки в одному з потоків. Допускаются наступні OpenMP директиви:

private(list),

firstprivate(list).

```
#pragma omp single [Речення [Речення .] ]
        structured block
```

У OpenMP існують різні режими виконання (Execution Mode) паралельних структурних блоків. Можливі наступні режими:

•Динамічний режим (Dynamic Mode). Цей режим встановлений за умовчанням і визначається завданням змінної оточення OMP_DYNAMIC в операційній системі. Існує можливість задати цей режим і саму змінну усередині програми

•Статичний режим (Static Mode). Цей режим визначається завданням змінної оточення OMP_STATIC в операційній системі. В цьому випадку кількість потоків визначається програмістом. Задати змінну оточення OMP_STATIC можна за допомогою команди. Існує можливість задати цей режим і саму змінну усередині програми.

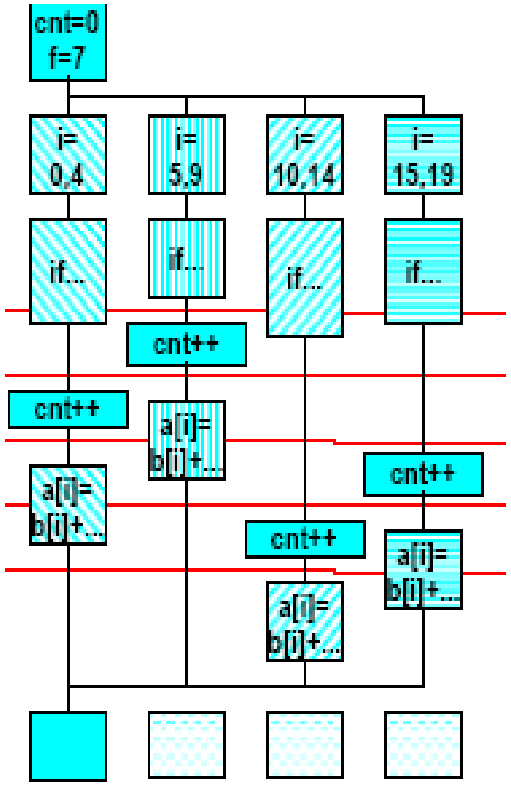
Всього в OpenMP існує шість типів синхронізації :
critical
atomic
barrier
master
ordered
flush.

Синхронізація типу atomic
визначає змінну в лівій частині оператора присвоєння, яка повинна коректно оновлюватися декількома потоками. В цьому випадку відбувається запобігання перериванню доступу, читання і запису даних, таких, що знаходяться в загальній пам'яті, з боку інших потоків.
Синхронізація типу critical
використовується для опису структурних блоків, що виконуються тільки в одному потоці з усього набору паралельних потоків.
Синхронізація типу barrier
встановлює режим очікування завершення роботи усіх запущених в програмі паралельних потоків досягши точки barrier.
Синхронізація типу master
використовується для визначення структурного блоку програми, який виконуватиметься виключно в головному потоці (паралельному потоці з нульовим номером) з усього набору паралельних потоків.

Синхронізація типу ordered
використовується для визначення потоків в паралельній області програми, які виконуються в порядку, відповідному послідовній версії програми.
Синхронізація типу flush
використовується для оновлення значень локальних змінних, перерахованих як аргументи цієї команди, в оперативній пам'яті. Після виконання цієї директиви усі змінні, перераховані в цій директиві, мають одне і те ж значення для усіх паралельних потоків.

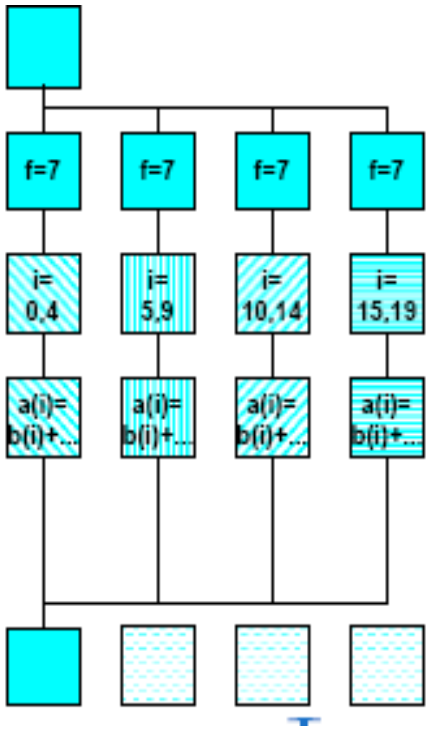
Помилки при роботі з системами із загальною пам'яттю:
1. **Помилки, пов'язані з умовами змагальності**. При непередбачуваному часі завершення паралельних потоків можливі непередбачувані результати обчислень. Автоматичне розпаралелювання в таких ситуаціях ускладнене, оскільки компілятору або важко, або взагалі неможливо розпізнати взаємозалежності за даними, що виникають в процесі паралельних обчислень при непередбачуваному часі завершення роботи паралельних потоків. У таких ситуаціях важливо правильно визначити послідовність виконання паралельних потоків, щоб уникнути непередбачуваних результатів.
2. **Мертве блокуванням**. В процесі виконання програм із загальною пам'яттю можлива ситуація, коли один з паралельних потоків чекає звільнення доступу до об'єкту, який ніколи не буде відкритий.

```
#pragma omp parallel private(f)
{    f=7;
#pragma omp for
      for (i=0; i<20; i++)
      a[i] = b[i] + f * (i+1);
} /* omp end parallel */
```




```
cnt = 0;
f=7;
#pragma omp parallel
{#pragma omp for

for (i=0; i<20; i++)
{ if (b[i] == 0)
{#pragma omp critical
                    cnt ++; }
 /* endif */
a[i] = b[i] + f * (i+1);}
/* end for */
}
/*omp end parallel */
```




```
#include <stdio.h>
#include <omp.h>
int main ()
{
      int n =100000, i;
      double pi, h, sum, x;
      h   = 1.0 / (double) n;
      sum = 0.0;
#pragma omp parallel default (none) private (i, x) shared (n, h, sum)
      {
            int id = omp_get_thread_num();
            int numt = omp_get_num_threads();
            for (i = id + 1; i <= n;  i=i+numt)
            {
                        x = h * ((double)i - 0.5);
             #pragma omp critical
                              sum += (4.0 / (1.0 + x*x));
              }
       }
       pi = h * sum;
       printf("pi is approximately %.16f", pi);
       return 0;
}
```