



OPENMP

MEMORIA COMPARTIDA


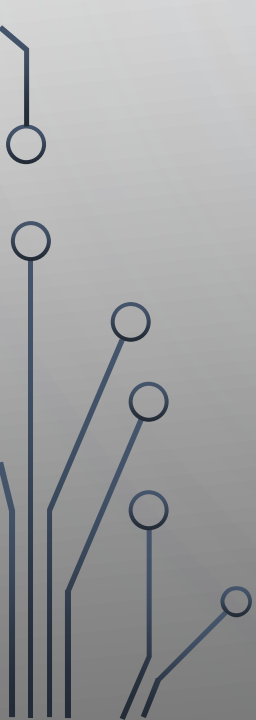


OPENMP

- API

OpenMP es una API (Application Programming Interface - Interfaz de Programación de Aplicaciones) que permite expresar y obtener paralelismo en programas escritos en C, C++, Fortran y Python.

Trabaja con un modelo de memoria compartida, es decir, todas las unidades de ejecución tienen acceso a un mismo espacio de memoria.



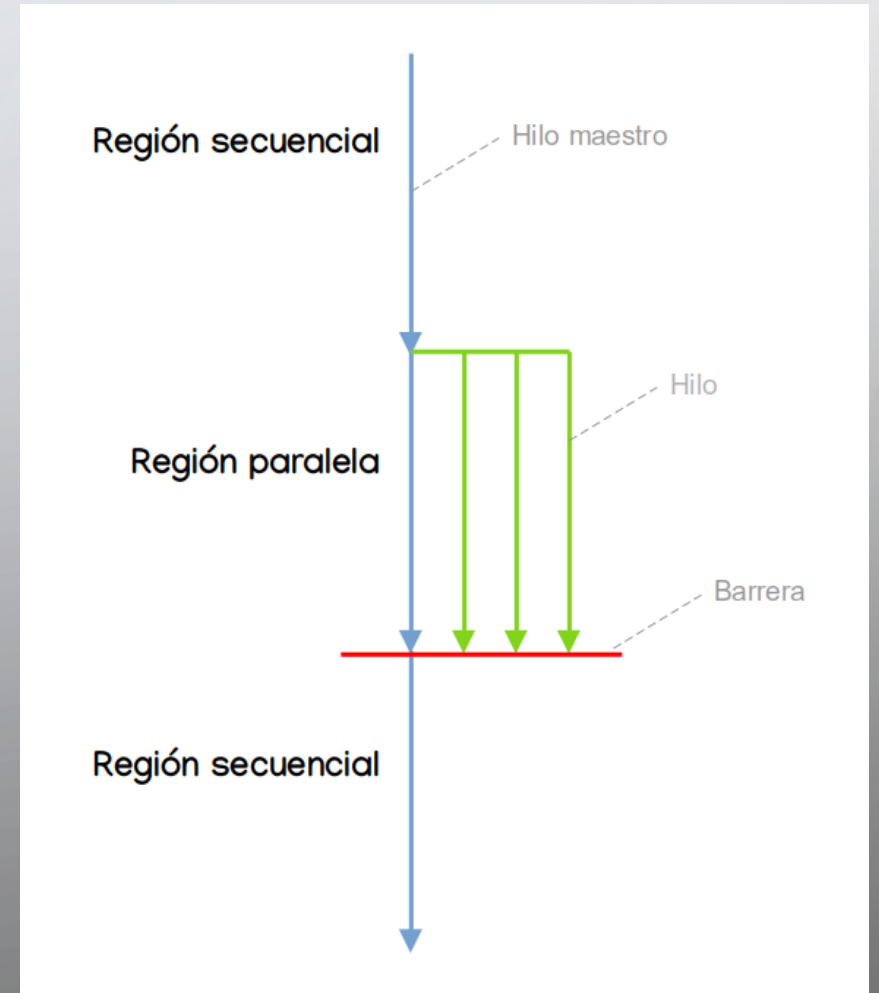
OPENMP

- Utiliza directivas o clausulas

`#pragma omp parallel`

Esta es la directiva que le indica al programa que se ha llegado a una región paralela.

Se crean nuevos hilos que ejecutarán el código contenido dentro de la región en paralelo.



OPENMP

Para obtener la cantidad de hilos de ejecución

```
int cantidad=omp_get_num_threads()
```

Para definir cantidad de hilos desde código

```
omp_set_num_threads(2)
```

Para saber que hilo es

```
int id=omp_get_thread_num()
```

OPENMPI

```
#include <stdio.h>
#include <omp.h>

int main(){
    //omp_set_num_threads(2);

    int num = omp_get_thread_num();
    printf("numero de hilo=%d en región secuencial\n", num);

    #pragma omp parallel
    {
        int thread = omp_get_thread_num();
        printf("numero de hilo=%d in región paralela\n", thread);

    } // barrera implicita

    num = omp_get_thread_num();
    printf("numero de hilo=%d en región secuencial\n", num);
}
```

OPENMP

- Compilar

```
gcc -fopenmp hola-omp.c -o hola-omp
```

- Ejecutar

```
./hola-mp
```

Se ejecutan la cantidad de hilos que
están definidos por defecto

OPENMP

```
#!/bin/bash
```

```
#SBATCH --nodes=1
```

```
#SBATCH --output=omp-out.%j
```

```
#SBATCH --error=omp-err.%j
```

```
#SBATCH --partition=cronos
```

```
export OMP_NUM_THREADS=4
```

```
./hola-mp
```

```
NOMBRAR omp-job.sh
```

```
EJECUTAR sbatch omp-job.sh
```

OPENMP

Directivas Sections y Section

Permite asignar distinto código a cada sección en una región paralela, cada sección será ejecutada por un hilo diferente.

```
#pragma omp sections
```

```
#pragma omp section
```


OPENMP

```
#include <stdio.h>
#include <omp.h>
int main(){
    int total = omp_get_thread_num();
    printf("hilo=%d en region secuencial\n", total);
```

OPENMP

```
#pragma omp parallel //inicio área paralela
{ #pragma omp sections //inicio secciones
  { #pragma omp section
    {
      int hilo = omp_get_thread_num();
      printf("Hola");
      printf("Section 0 ejecutada por hilo=%d\n", hilo);
    }
    #pragma omp section
    {
      int hilo = omp_get_thread_num();
      printf("Hello");
      printf("Section 1 ejecutada por hilo=%d\n", hilo);
    }
    #pragma omp section
    {
      int hilo = omp_get_thread_num();
      printf("Bonjour");
      printf("Section 2 ejecutada por hilo=%d\n", hilo);
    }
  }
} total = omp_get_thread_num();
printf("thread=%d in serial region\n", total);
return 0;}
```

OPENMP

Clausulas

shared, private, default

Permiten establecer de forma explicita las variables como compartidas o privadas.

```
#pragma omp parallel default(shared) private(var)
```

```
#pragam omp parallel shared(var1) private(var2)
```

OPENMP

```
#include <stdio.h>
#include <omp.h>
int main(){
    int hilo = -1;
    int master = omp_get_thread_num();
    printf("hilo mastro =%d en region secuencial\n", master);

    #pragma omp parallel default(none) shared(master) private(hilo)
    {
        hilo = omp_get_thread_num();
        if(hilo==master)
            printf("hilo maestro=%d en region paralela\n", master);
        else
            printf("hilo=%d en region paralela\n", hilo);
    }
    printf("hilo maestro=%d en region secuencial\n", master);
    printf("valor de variable hilo=%d\n", hilo);
}
```



OPENMP

Clausulas

reduction

reduction (operador:lista)

```
#pragma omp parallel for reduction(+:sum)
```



OPENMP

```
#include <stdio.h>
#include <omp.h>
static long num_steps=10;
int main(){
    int i;
    int sum=0;
    #pragma omp parallel for reduction(+:sum)
    for (i=0;i<=num_steps;i++)
    {
        sum=sum+i;
        int num = omp_get_thread_num();
        printf("numero de hilo=%d en región, suma %d \n", num, sum);
    }
    printf(" suma %d\n", sum);
    return 0;
}
```