



---

# PRÁCTICA 3

---

Tarea 1: Estudio del API OpenMP



9 DE DICIEMBRE DE 2018

GUILLERMO JIMENEZ

## Índice

<b>#PRAGMA OMP PARALLEL .....</b>	<b>- 3 -</b>
<b>#pragma omp parallel sections .....</b>	<b>- 4 -</b>

## Introducción

Se deberá estudiar el API de OpenMP y su uso con gcc comprobando el correcto funcionamiento de algunos ejemplos disponibles en Internet.

La página en la que me he apoyado ha sido la que nos habéis proporcionado y he escogido entre todos algunos pocos como “pragma omp parallel”, “pragma omp for” y “pragma omp sections” por que han sido los que he usado para la práctica a la hora de aplicar OpenMP.

## #PRAGMA OMP PARALLEL

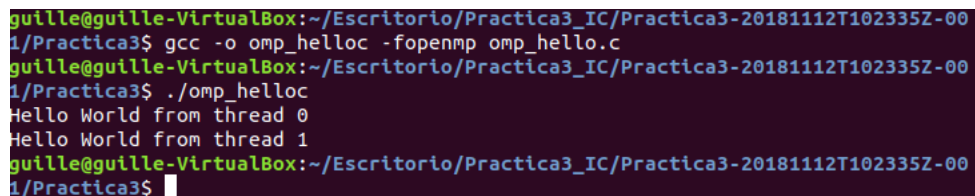
La directiva parallel forma un conjunto de hebras e inicia la ejecución paralela, en algunos casos es recomendable insertar el número de threads como una expresión de enteros. Dentro de la directiva parallel existen algunas cláusulas como private(LIST) o como shared(LIST)

Un ejemplo de OpenMP con #pragma omp parallel private(numThreads), donde numThreads será el número de cores disponibles.

```
int main (int argc, char *argv[])
{
    int p,th_id;
    p = omp_get_num_procs();
    omp_set_num_threads(p);
    #pragma omp parallel private(th_id);
    {
        th_id = omp_get_thread_num();
        Printf ("Hello World from thread %d\n", th_id);
    }
    return 0;
}
```

La manera de compilarlos mediante GNU GCC sería:

Para el primero → gcc -o omp\_helloc -fopenmp omp\_hello.c



```
guille@guille-VirtualBox:~/Escritorio/Practica3_IC/Practica3-20181112T102335Z-00
1/Practica3$ gcc -o omp_helloc -fopenmp omp_hello.c
guille@guille-VirtualBox:~/Escritorio/Practica3_IC/Practica3-20181112T102335Z-00
1/Practica3$ ./omp_helloc
Hello World from thread 0
Hello World from thread 1
guille@guille-VirtualBox:~/Escritorio/Practica3_IC/Practica3-20181112T102335Z-00
1/Practica3$
```

Y esta sería la salida, podemos comprobar como usa distintos cores para ejecutar el código, por lo tanto será mucho más óptimo el computado, de tiempo no he mostrado nada, pero es lógico que al usar 2 cores vaya más rápido que con uno.

```
#include <omp.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
    int p, th_id;
    p = omp_get_num_procs();
    omp_set_num_threads(p);
    #pragma omp parallel private(th_id)
    {
        th_id = omp_get_thread_num();
        printf ("Hello World from thread %d\n", th_id);
    }
    return 0;
}
```

## #PRAGMA OMP PARALLEL SECTIONS

```
#include <omp.h>
#include <stdio.h>
#include <time.h>
#include <sys/time.h>

int main (int argc, char *argv[])
{
    double area,x;

    int i,n;
    area = 0.0;

    printf ("n=");

    scanf ("%d",&n);
```

```

    for (i = 0; i < n ; i++)
    {
        #pragma omp parallel sections
        {
            #pragma omp section
            x=(i+0.5)/n;
            #pragma omp section
            area += 4.0/(1.0+x*x);
        }
    }

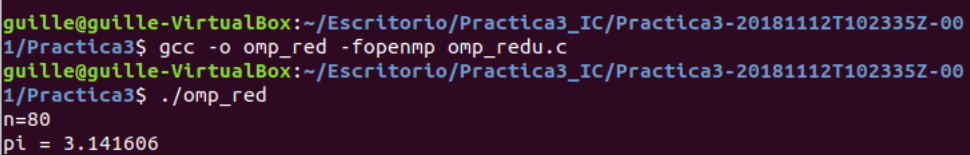
    printf ("pi = %lf\n", area/n);

    return 0;
}

```

La manera de compilarlo sería la misma:

Gcc -o omp\_red -fopenmp omp\_redu.c



```

guille@guille-VirtualBox:~/Escritorio/Practica3_IC/Practica3-2018112T102335Z-00
1/Practica3$ gcc -o omp_red -fopenmp omp_redu.c
guille@guille-VirtualBox:~/Escritorio/Practica3_IC/Practica3-2018112T102335Z-00
1/Practica3$ ./omp_red
n=80
pi = 3.141606

```

Y el código ejecutado te devuelve el valor del número pi que es igual al área partido de la n que introduces tu

```

#include <omp.h>
#include <stdio.h>
#include <time.h>
#include <sys/time.h>

int main (int argc, char *argv[])
{
    double area,x;

    int i,n;area = 0.0;

    printf ("n=");
    scanf ("%d",&n);

    for (i = 0; i < n ; i++)
    {
        #pragma omp parallel sections
        {
            #pragma omp section
            x=(i+0.5)/n;
            #pragma omp section
            area += 4.0/(1.0+x*x);
        }
    }

    printf ("pi = %lf\n", area/n);
    return 0;
}

```