

# Introducción a Open Multiprocessing (OpenMP)

# ¿Qué es OpenMP?

- **API** diseñada para la programación multiproceso en sistemas con arquitectura de memoria compartida, compatible con múltiples plataformas. Se compone de:
  - Directivas del compilador.
  - Biblioteca de funciones.
  - Variables de entorno.

**Nota:** No es un lenguaje de programación, sino que se basa en el modelo de hilos.

# OpenMP

Definido por un consorcio de proveedores de hardware y software, encabezado por AMD y conocido como OpenMP Architecture Review Board (ARB) desde 1997.

- BCS - Barcelona Supercomputing Center
- CAPS-Entreprise
- Convey Computer
- Cray
- Fujitsu
- HP
- IBM
- Intel
- Microsoft
- NEC
- NVIDIA
- Oracle Corporation
- Signalogic
- The Portland Group, Inc.
- Texas

Instruments Mas

información en

[www.openmp.org](http://www.openmp.org)

# ¿Qué es OpenMP?

- Modelo de programación paralela.
- Paralelismo basado en memoria compartida.
- Extensiones para lenguajes de programación existentes (C, C++, Fortran).
- Combina código serial y paralelo en un solo archivo fuente.

# Compiladores que soportan OpenMP

...

**GNU GCC** (a partir de la versión 4.2).

...

# Compilando con GNU GCC

El soporte para OpenMP ha de ser activado:

- Versión serial de un programa:
  - `gcc ejemplo.c -o ejemplo`
- Versión paralela de un programa:
  - `gcc -fopenmp ejemplo.c -o ejemplo`
- En la mayoría de los compiladores la opción es **openmp**

# Modelo de programación en OpenMP

- El paralelismo en OpenMP se especifica a través de directivas que se insertan en el código de C.
- Básicamente un programa en OpenMP es un programa secuencias que se añaden directivas OpenMP en el punto adecuado.
- En **lenguaje C** una directiva OpenMP tiene la forma:
  - **#pragma omp** <directiva OpenMP especificada>
- Todos los programas OpenMP deben incluir el archivo cabecera `#include <omp.h>`

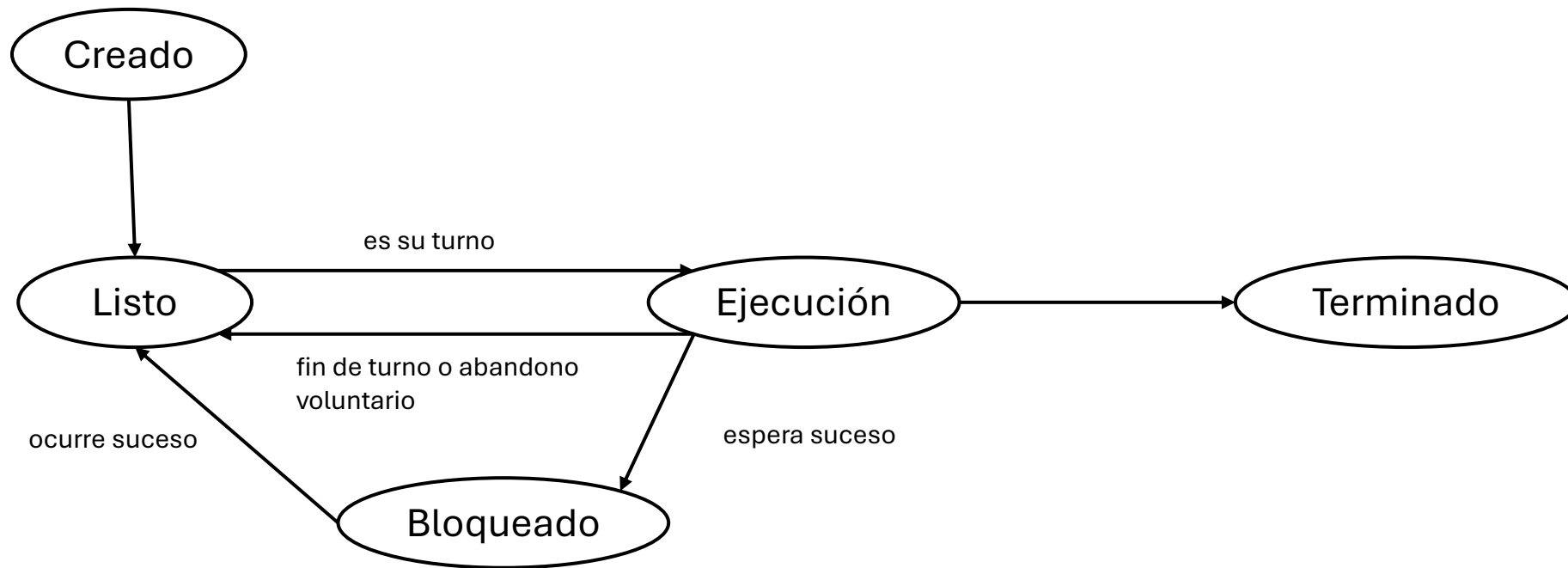
# Antes de iniciar....

1. ¿Qué es un proceso?
2. ¿Qué es un hilo?
3. ¿Qué se comparte entre los hilos?
4. ¿Cuál es la diferencia entre hilo y proceso?



# Proceso

Programa en ejecución que cuenta con su propio espacio de memoria y recursos, como CPU y memoria, para realizar tareas específicas.



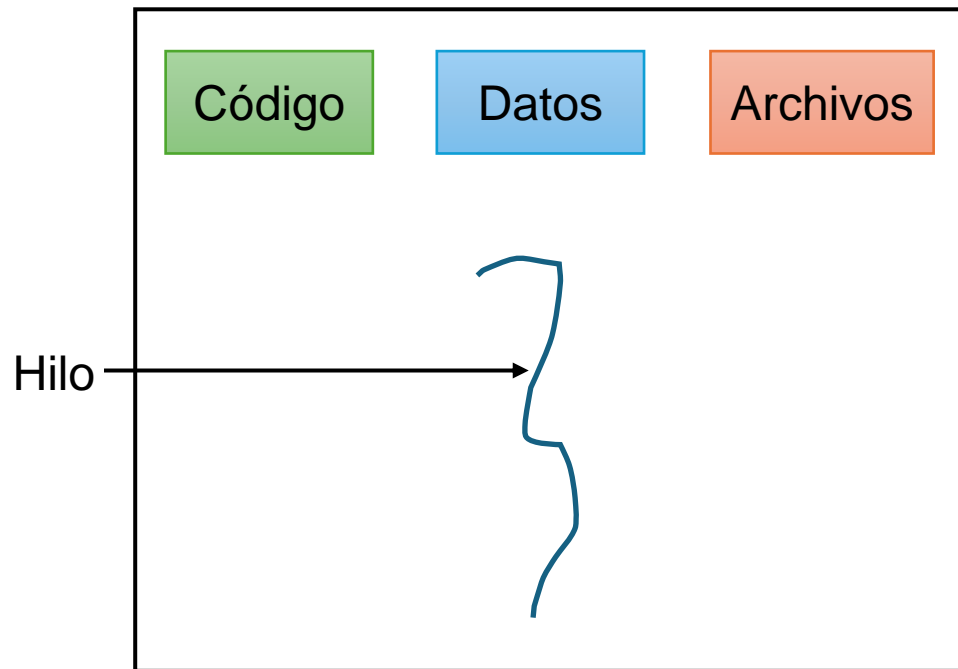
# Estados de un proceso

- En un principio, un proceso no existe.
- Una vez creado, el proceso pasa al estado denominado Listo (está en condiciones de usar la CPU tan pronto se le dé oportunidad).
- Un proceso puede pasar de Ejecución a Bloqueado cuando debe esperar a que ocurra un determinado evento o suceso, como:
  - La terminación de una operación de entrada/salida (E/S).
  - La finalización de otra tarea de otro proceso.

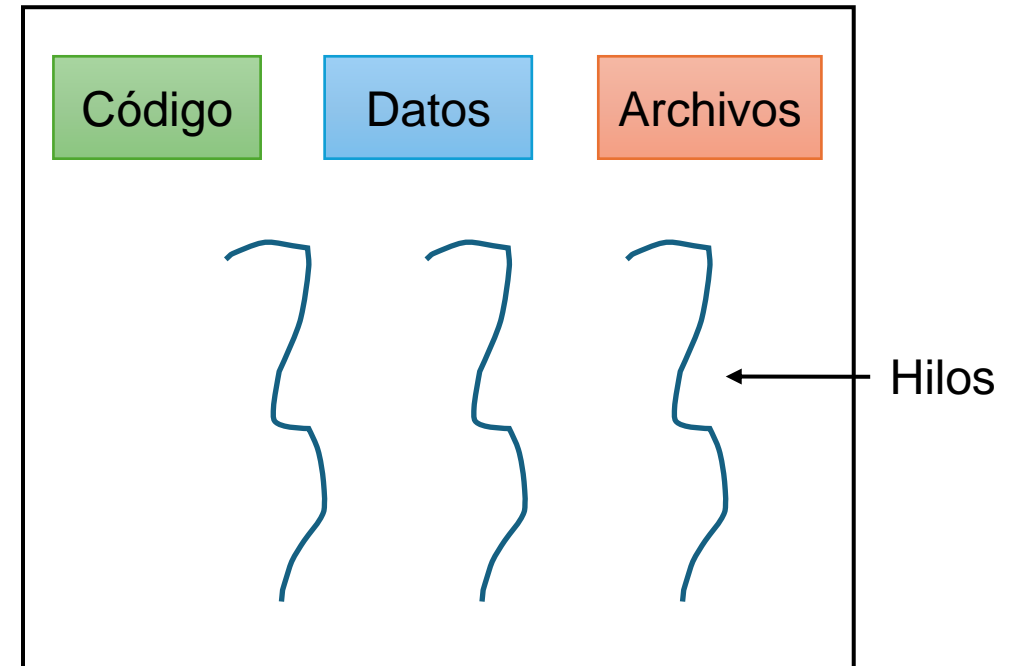
# Hilos (Thread)

- Dentro de un proceso puede haber varios hilos de ejecución.
- Un hilo, también conocido como hebra, proceso ligero, flujo, subprocesso o “thread”, es un programa en ejecución que comparte la imagen de la memoria y otros recursos del proceso con otros hilos.
- Por tanto, un hilo puede definirse como cada secuencia de control dentro de un proceso que ejecuta sus instrucciones de forma independiente.

# Procesos con un solo hilo y con múltiples hilos

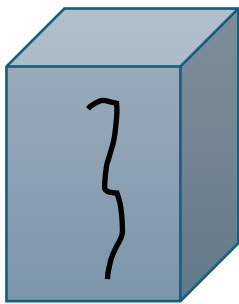


Mono-hilo

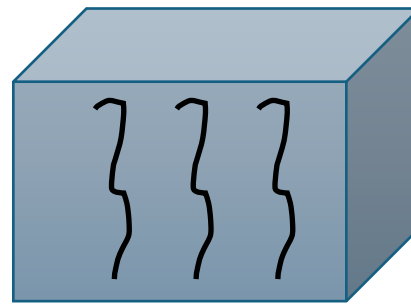


Multi-hilo

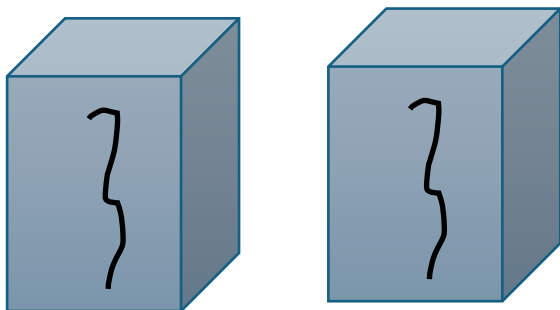
# Hilos y procesos



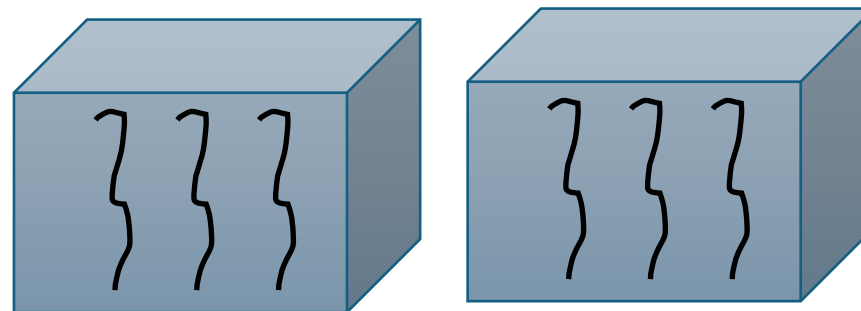
Un proceso, un hilo



Un proceso, varios hilos



Varios procesos, un  
hilo por proceso

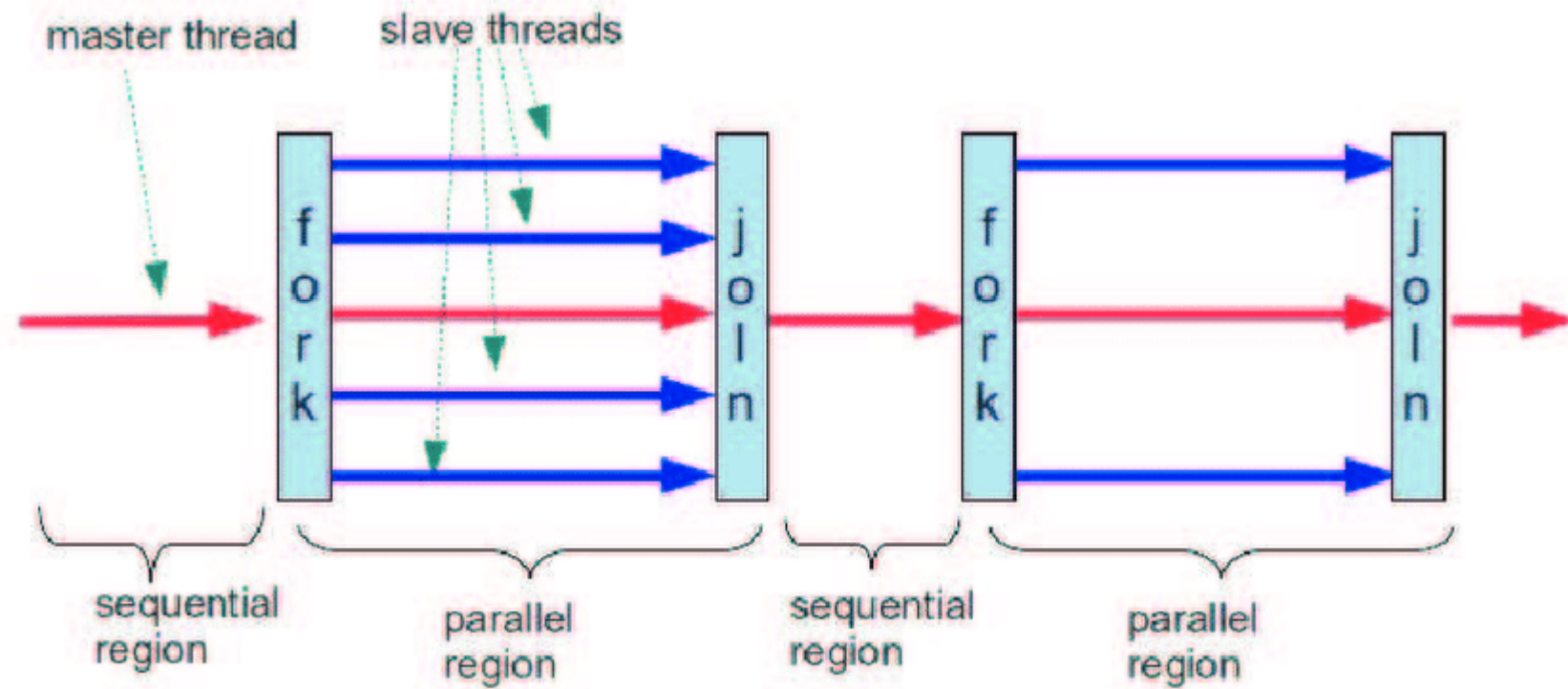


Varios procesos, varios hilos  
por proceso

# Arquitectura de OpenMP

- Fork / join (maestro - esclavo).
- Trabajo y datos compartidos entre hilos.
- Maneja sincronización (barreras, otras).

# Fork / Join



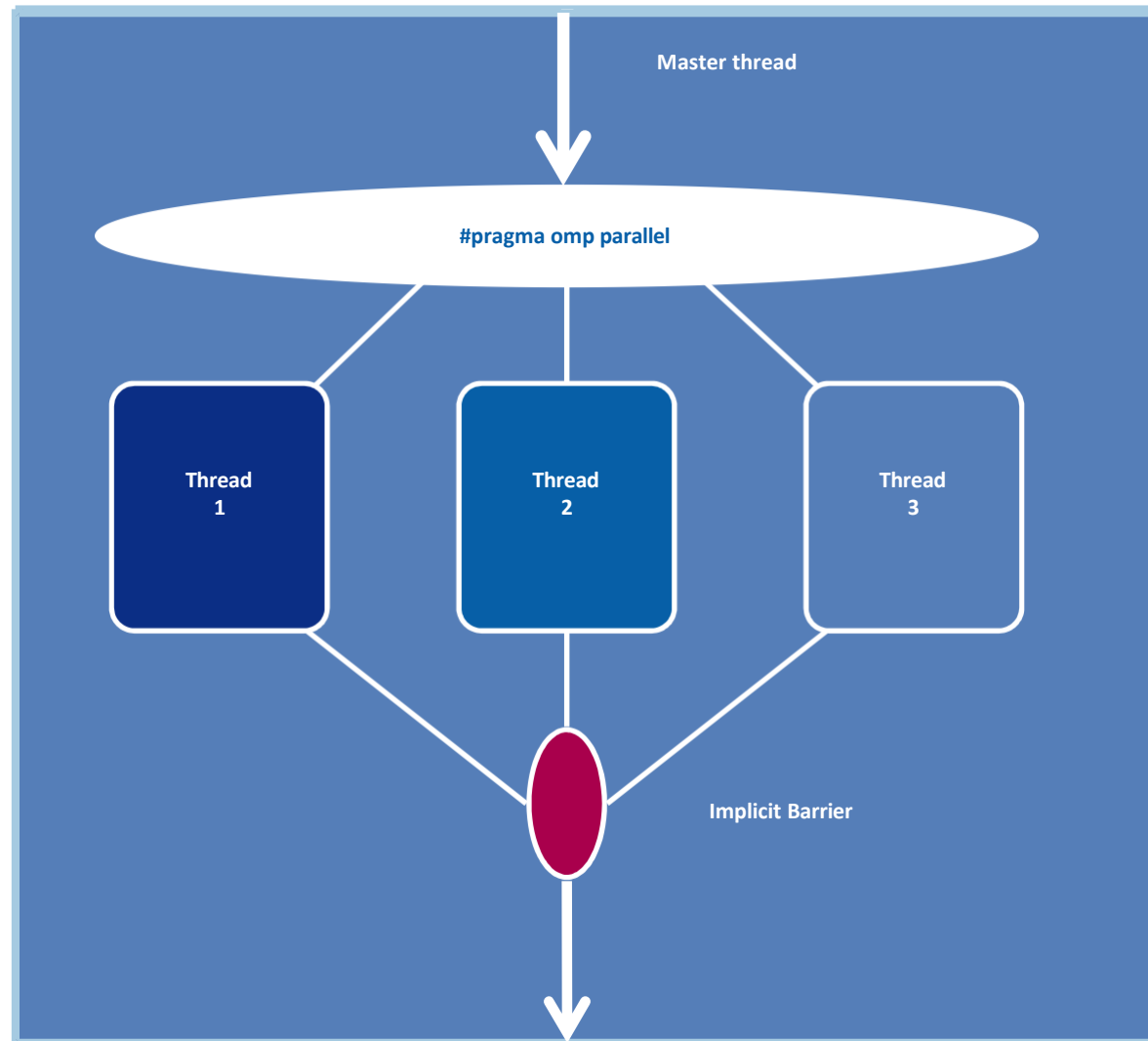
# Sintaxis

- Directivas o pragmas

- `#pragma omp construct [clause[clause]...]`
- Cláusulas: especifican atributos para compartir datos y calendarización.
- Una pragma en C o C++ es una directiva al compilador.



# Regiones paralelas



# Regiones paralelas

- Los hilos son creados desde el pragma *parallel*.
- Los datos son compartidos entre hilos.
- **C/C++:**

```
#pragma omp parallel
{
    bloque código
}
```

# Regiones paralelas

- Conceptualmente, la programación de aplicaciones con OpenMP consiste en definir zonas del código que serán ejecutadas por un número determinado de hilos.
- Estas zonas de código reciben el nombre de regiones paralelas.
- Todos los programas OpenMP comienzan con una única hebra, llamada **maestra**.

# Ejemplo

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

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

```
int main( int argc, char * argv[] ) {
```

```
    printf("Hola mundo \n");
```

```
    #pragma omp parallel
```

```
    {
```

```
        printf("Hola desde el hilo \n");
```

```
    }
```

```
    return 0;
```

```
}
```

# ¿Cuántos hilos?

- Número de hilos = número de procesadores o núcleos.
- Intel lo usa de esta forma.
- Se definen más hilos con la variable de ambiente
  - OMP\_NUM\_THREADS.

# Actividad 1

- Compilar la siguiente versión serial:

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

```
int main (){
```

```
    int i;
```

```
    printf("Hola Mundo\n");
```

```
    for(i=0;i<6;i++)
```

```
        printf("Iteración: %d\n",i);
```

```
    printf("Bye");
```

```
}
```

# Actividad 1

- Agregar la directiva para ejecutar las primeras cuatro líneas del *main* en paralelo.
- Compilar con la opción `-fopenmp`
- ¿Qué sucede?

# Biblioteca de funciones

- Establece el número de hilos a utilizar en la siguiente región paralela.
  - `void omp_set_num_threads(int num_hilos);`
- Obtiene el identificador del hilo actual.
  - `int omp_get_thread_num();`
- Obtiene el número de CPUs/Cores disponibles.
  - `int omp_get_num_threads();`
- Obtiene el número total de hilos requeridos.
  - `int omp_get_num_procs();`



# Ejemplo

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

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

```
int main( int argc, char * argv[] ) {
```

```
    omp_set_num_threads(4);
```

```
    #pragma omp parallel
```

```
    {
```

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

```
        printf(" hilo (%d) \n", id)
```

```
    }
```

```
    return 0;
```

```
}
```

*Runtime function*  
para solicitar un  
determinado número  
de hilos

*Runtime function*  
retorna el id de hilo (0  
al 3)

# Ejemplo

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

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

```
int main( int argc, char * argv[] ) {
```

```
#pragma omp parallel num_threads(4)
```

```
{
```

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

```
    printf(" hilo (%d) \n", id)
```

```
}
```

```
return 0;
```

```
}
```

*Cláusula*  
para solicitar un  
determinado número  
de hilos

*Runtime function*  
retorna el id de hilo (0  
al 3)

# Número de hilos en la región paralela

- El número de hilos que se generan para ejecutar una región paralela se controla:
- Estáticamente, mediante una variable de entorno:
  - > export OMP\_NUM\_THREADS=4
- En ejecución, mediante una función :
  - omp\_set\_num\_threads(4);
- En ejecución, mediante una cláusula del “pragma parallel”:
  - num\_threads(4)

# Ejemplo

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[]){
#pragma omp parallel num_threads(16)
{
    int idHilo,numeroHilos, numerocpus;
    /* obtiene el id de la hilo actual */
    idHilo = omp_get_thread_num();
    numeroHilos = omp_get_num_threads();
    numerocpus = omp_get_num_procs();
    printf("Hola yo soy el hilo %d \n", idHilo);
    if (idHilo == 0) {
        printf("Número de CPUs = %d \n", numerocpus);
        printf("Número de Hilos Totales = %d \n", numeroHilos);
    }
}
return(0);
}
```

# Actividad

- Desarrolle un programa que modifique los valores de un vector de 30 elementos utilizando programación paralela con OpenMP. Cada hilo será responsable de modificar una única posición del vector, asignando su propio identificador de hilo como valor. Al finalizar la ejecución, el programa deberá mostrar el vector actualizado con los valores asignados por cada hilo.

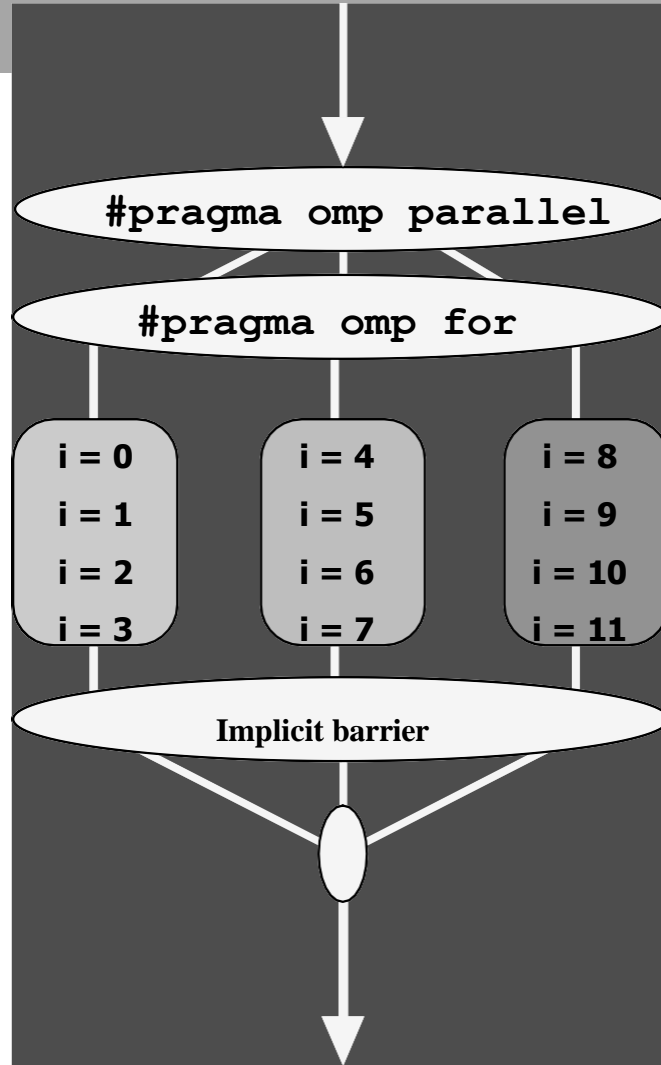
# Directiva For

```
#pragma omp paralell
#pragma omp for
    for(i=0;i<100;i++) {
        Realizar Trabajo();
    }
```

- Divide las iteraciones entre los hilos.
- Debe estar especificada en una región paralela

# Directiva For

```
#pragma omp parallel
#pragma omp for
  for(i = 0; i < 12; i++)
```



# Directiva For

- Estos dos segmentos de código son equivalentes.

```
#pragma omp parallel
{
    #pragma omp for
    for (i=0; i< SIZE; i++) {
        arreglo[i] = i * 2;
    }
}
```

```
#pragma omp parallel for
    for (i=0; i< SIZE; i++) {
        arreglo[i] = i * 2;
    }
```



# Gestión de datos

- La compartición de variables es el punto clave en un sistema paralelo de memoria compartida, por lo que es necesario controlar correctamente el **ámbito** de cada variable.
- Las variables **globales** son compartidas por todos los *hilos*. Sin embargo, algunas variables deberán ser propias de cada *hilo*, **privadas**.
- Para poder especificar adecuadamente el **ámbito** de validez de cada variable, se añaden una serie de **cláusulas** a la directiva **parallel**, en las que se indica el carácter de las variables que se utilizan en dicha región paralela.

# Gestión de datos

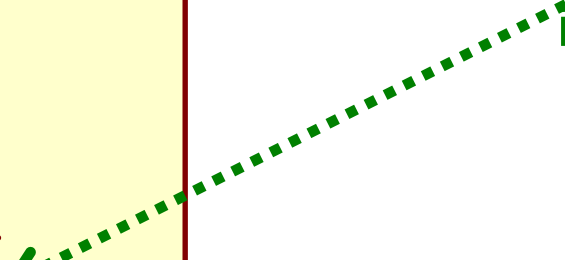
- Para la gestión de datos en las directivas paralelas se dispone de las siguientes cláusulas.
- **private (lista\_variables)**. Indica que la lista de variables es local a cada hilo sin inicializar.
- **shared (lista\_variables)**. Indica que las variables de la lista serán compartidas por todos los hilos de ejecución. Sólo existe una copia, y todos los *hilos* acceden y modifican dicha copia.

# Gestión de datos


## Ejemplo

```
x = 2;  
y = 1;  
z = 3;  
  
#pragma omp parallel  
  shared(y) private(x,z)  
  {  
    z = x * x + 3;  
    x = y * 3 + z;  
  }  
  
printf("x = %d \n", x);
```

x no está  
inicializada



x no mantiene  
el nuevo valor



# Ejemplo

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int numThreads, threadId;
    #pragma omp parallel private(numThreads, threadId)
    {
        threadId = omp_get_thread_num();
        printf("Hello World from thread = %d\n", threadId);

        if (threadId == 0)
        {
            numThreads = omp_get_num_threads();
            printf("Number of threads = %d\n", numThreads);
        }
    }
}
```



Variables locales al hilo

# Gestión de datos

- Se declaran **objetos completos**: no se puede declarar un elemento de un arreglo como compartido y el resto como privado.
- Por **defecto**, las variables son **shared**.
- Cada *hilo* utiliza su propia pila, por lo que las variables declaradas en la propia región paralela son privadas.

# Gestión de datos

- Para la gestión de datos en las directivas paralelas se dispone de las siguientes cláusulas.
- **firstprivate (lista\_variables)**. Es similar a **private** solo que las variables se inicializan al entrar a los hilos paralelos con los valores que contenía antes de la directiva paralela.

# Gestión de datos

Ejemplo:

```
x = y = z = 2;  
#pragma omp parallel  
  private(Y) firstprivate(Z)  
{  
    ...  
    x = y = z = 1;  
}  
...
```

valores **dentro** de  
la región  
paralela?

x = 2  
y = ?  
z = 2

valores **fuera** de la  
región paralela?

x = 1  
y = 2  
z = 2

# Gestión de datos

**default (none / shared)**

**none**: obliga a declarar explícitamente el ámbito de todas las variables. Útil para no olvidarse de declarar ninguna variable (provoca error al compilar).

**shared**: las variables sin declarar son **shared** (por **defecto**).



# Referencias

- Basado en: Sáenz García, E. K., & Valdez Casillas, O. R. (s. f.). *Programación paralela en OpenMP* [Diapositivas]. <http://lcomp89.fi-b.unam.mx/licad/assets/ProgramacionOpenMP/Programaci%C3%83%C2%B3nParalelaOpenMP.pdf>.