

AC-BP1.pdf



patrivc



Arquitectura de Computadores



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.





**KEEP
CALM
AND
ESTUDIA
UN POQUITO**



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Seminario 1. Herramientas de programación paralela I: Directivas OpenMP

¿De dónde viene el acrónimo OpenMP?

- Versión corta: Open Multi-Processing
- Versión larga: Especificaciones abiertas (Open) para multiprocesamiento (Multi-Processing) generadas mediante trabajo colaborativo de diferentes partes interesadas de la industria del hardware y del software, y también del mundo académico y del gobierno.
Miembros permanentes (vendedores): AMD, Cray, Fujitsu, HP, IBM, Intel, NEC, Oracle, Microsoft, etc.
Miembros auxiliares (académico, gobierno): NASA, RWTH Aachen University, etc.

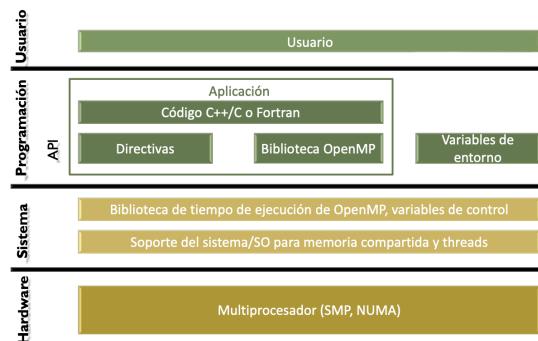
¿Qué es OpenMP v.3?

Es una API para escribir código paralelo con el **paradigma/estilo de programación de variables compartidas** para ejecutar aplicaciones en paralelo en varios threads.

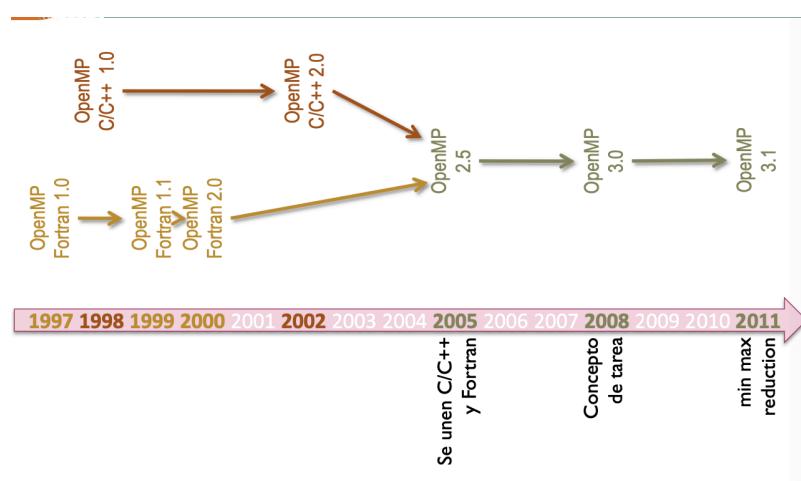
Las API (Application Programming Interface) tienen una **capa de abstracción** que permite al programador acceder cómodamente a través de una interfaz a un conjunto de funcionalidades.

La API OpenMP comprende o define las **directivas del compilador, funciones de biblioteca, y variables de entorno**.

Es una **herramienta para programación paralela: no automática** (no extrae paralelismo implícito), con un **modelo de programación basado en el** paradigma/estilo de variables compartidas, multithread y directivas del compilador y funciones. El código paralelo OpenMP es código descrito con un lenguaje secuencial (C, C++ o Fortran), directivas y funciones de la interfaz OpenMP. Es **portable** (API especificada para C/C++ y Fortran (77, 90 y 95) y la mayor parte de las plataformas (SO/hardware) tienen implementaciones de OpenMP. Se podría considerar **estándar** en cuanto que lo han definido un conjunto de vendedores de hardware y software destacados.



Evolución de OpenMP



Componentes de OpenMP

-Directivas: El preprocesador del compilador las sustituye por código.

-Funciones: Por ejemplo, para fijar parámetros y preguntar por parámetros (p. ej.: no de threads) en tiempo de ejecución.

-Variables de entorno: Para fijar parámetros antes de la ejecución (p. ej.: no de threads): `export OMP_NUM_THREADS=4`

```
C/C++
#include <omp.h>
...
omp_set_num_threads(nthread)
#pragma omp parallel
{
    #pragma omp for
    for (i=0; i<n; i++) {
        for (j=0; j<m; j++) {
            .
        }
    }
}
```

Fortran

```
use omp_lib
...
call omp_set_num_threads(nthread)
!$OMP PARALLEL
!$OMP DO
DO i=1,N
    DO j=1,M
        .
    END DO
END DO
!$OMP END PARALLEL DO
```

Sintaxis Directivas C/C++

#pragma omp	nombre de la directiva	[cláusula [,]...]	newline
Necesario en todas las directivas C/C++ OpenMP	Necesario. Coincidirán los nombres en Fortran y C/C++ (salvo <code>for</code>)	Opcional. Pueden aparecer en cualquier orden. Coincidirán los nombres con los de Fortran	Necesario. Precede al bloque estructurado que engloba la directiva

El **nombre** define y **controla la acción que se realiza**. -> **Ejemplo:** parallel,for,section...

Las **cláusulas** especifican adicionalmente la **acción o comportamiento, la ajustan**. -> **Ejemplo:** private,schedule,reduction...

Las comas separando cláusulas son opcionales

Se distingue entre mayúsculas y minúsculas

Ejemplo: `#pragma omp parallel num_threads (8) if (N>20)`

Directivas

Las directivas en C/C++ son los pragmas.

```
#pragma omp <directive> [<clause>, <clause> ...]
```

Para dividirlas en varias filas o líneas de código se usa: “\”

```
#pragma omp parallel private (...) \ shared (...)
```

Portabilidad

La compilación en C/C++ es -> `gcc -fopenmp` ó `g++ -fopenmp`

Las **directivas** no se tendrán en cuenta si no se compilan usando OpenMP (-fopenmp, -openmp, etc).

Las funciones se evitan usando compilación condicional. Para C/C++ es usando: `_OPENMP` (esto se define cuando se compila usando OpenMP) y `#ifdef ... #endif`

```
#ifdef _OPENMP
omp_set_num_threads(nthread)
#endif
#pragma omp parallel for
    for (i=0; i<n; i++) {
        for (j=0; j<m; j++) {
            .
        }
    }
}
```

Algunas definiciones

Directiva ejecutable (executable directive) -> aparece en código ejecutable

Bloque estructurado (structured block) -> un conjunto de sentencias con una única entrada al principio del mismo y una única salida al final. No tiene saltos para entrar o salir. Se permite exit() en C/C++

```
omp_set_num_threads(nthread)
#pragma omp parallel
{
    for (i=0; i<n; i++) {
        ...
    }
    ....
    c=funcion();
    ...
}
```

Construcción (construct) (extensión estática o léxica) -> Directiva ejecutable + [sentencia, bucle o bloque estructurado]

```
omp_set_num_threads(nthread)
#pragma omp parallel
{
    for (i=0; i<n; i++) {
        ...
    }
    ....
    c=funcion();
    ...
}
```

Región (extensión dinámica) -> Código encontrado en una instancia concreta de la ejecución de una directiva o subrutina de la biblioteca OpenMP. Una construcción puede originar varias regiones durante la ejecución. Incluye: código de subrutinas y código implícito introducido por OpenMP

Extensión dinámica de parallel

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

int VE[4096],VR[4096],A[4096*4096];

int prodesc(int *x, int *y, int N)
{ int j,z;
z=0;
for (j=0 ; j<N; j++) z += x[j]*y[j];
return(z);
}

void prodmv(int* z,int* x, int* y, int M, int N)
{ int i;
#pragma omp for
for (i=0 ; i<M; i++) z[i]=prodesc(&x[i*N],y,N);
}

main()
{ int j,N=4096,i,M=4096;
    for (j=0 ; j<N; j++) VE[j]=j;
    for (i=0 ; i<M; i++)
        for (j=0 ; j<N; j++) A[i*N+j]=i+j;
    #pragma omp parallel
    prodmv(VR,A,VE,M,N);
}
```

Extensión estática de parallel

Directivas/constructores (v2.5)

Directiva	Ejecutable	declarativa	
Con bloque estructurado	parallel sections , worksharing , single master critical ordered		Con sentencia

Directiva	Ejecutable	declarativa	
Bucle	<u>DO/for</u>		Con sentencia
Simple (una sentencia)	atomic		Con sentencia
autónoma (sin código asociado)	barrier, flush	threadprivate	Sin sentencia

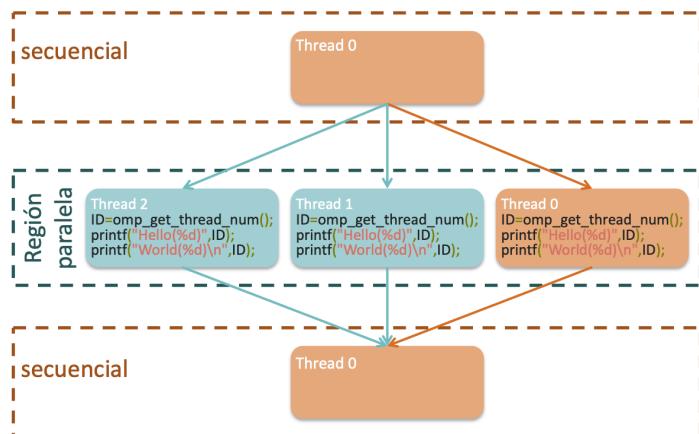
La directiva define la acción que se realiza. Las directivas en **color naranja** son las que vamos a estudiar. Las directivas subrayadas son las que tienen barrera implícita al final.

Directiva parallel

DIRECTIVA	ejecutable	declarativa	
Con bucle estructurado	<u>parallel</u> <u>sections, worksharing,</u> <u>single</u> master critical ordered		Con sentencia
bucle	<u>DO/for</u>		Con sentencia
simple (una sentencia)	atomic		Con sentencia
autónoma (sin código asociado)	barrier , flush	threadprivate	Sin sentencia

- Especifica qué cálculos se ejecutarán en paralelo.
- Un **thread (master)** crea un conjunto de threads cuando alcanza una Directiva parallel.
- Cada thread ejecuta el código incluido en la región que conforma el bloque estructurado.
- No reparte tareas entre threads.**
- Barrera implícita al final.**
- Se pueden usar de forma anidada.

Ejemplo: Hello World -> Imprime en pantalla con dos printf distintos "Hello World" y el identificador del thread que imprime.



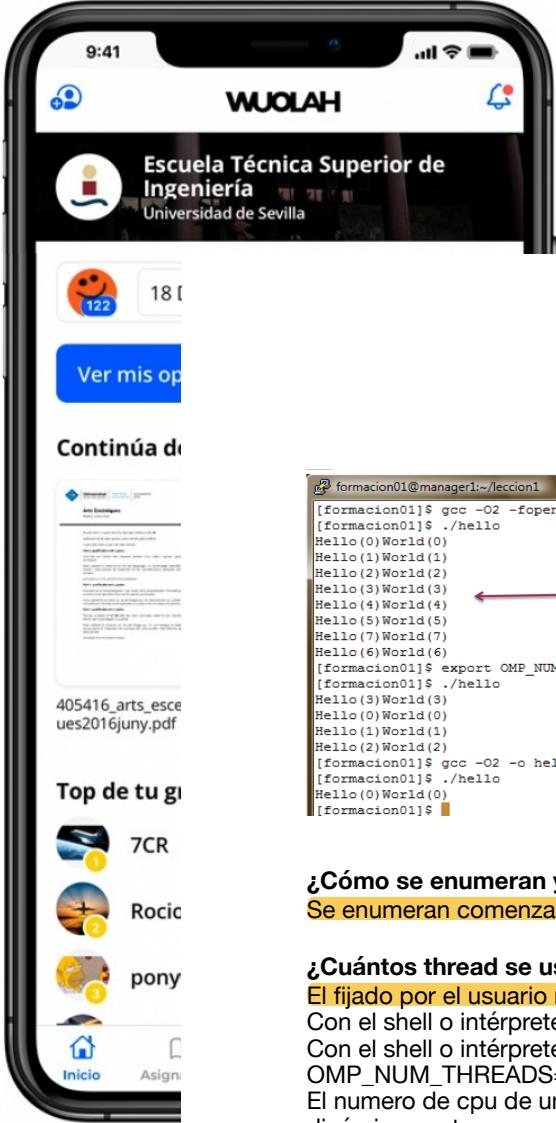
```
#include <stdio.h>
#ifndef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif
main() {

    int ID;
    #pragma omp parallel private(ID)
    {

        ID = omp_get_thread_num();

        printf("Hello(%d)",ID);
        printf("World(%d)\n",ID);

    }
}
```



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play

```

formacion01@manager1:~/lección1
[formacion01]$ gcc -O2 -fopenmp -o hello hello.c
[formacion01]$ ./hello
Hello(0)World(0)
Hello(1)World(1)
Hello(2)World(2)
Hello(3)World(3)
Hello(4)World(4)
Hello(5)World(5)
Hello(7)World(7)
Hello(6)World(6)
[formacion01]$ export OMP_NUM_THREADS=4
[formacion01]$ ./hello
Hello(3)World(3)
Hello(0)World(0)
Hello(1)World(1)
Hello(2)World(2)
[formacion01]$ gcc -O2 -o hello hello.c
[formacion01]$ ./hello
Hello(0)World(0)
[formacion01]$
```

Compilación: con -fopenmp

Plataforma: nodos de 8 cores

Cambio número de threads: usamos variables de entorno

Código portable: no hay errores de compilación sin -fopenmp

¿Cómo se enumeran y cuántos threads se usan?

Se enumeran comenzando desde 0 (0... no threads-1). El **master** es la 0.

¿Cuántos thread se usan en las ejecuciones anteriores?

El fijado por el usuario modificando la variable de entorno **OMP_NUM_THREADS**.

Con el shell o intérprete de comandos Unix csh (C shell): setenv OMP_NUM_THREADS 4

Con el shell o intérprete de comandos Unix ksh (Korn shell) o bash (Bourne-again shell): export OMP_NUM_THREADS=4

El numero de cpu de un nodo normalmente esta fijado por defecto, aunque puede variar dinámicamente.

Ejemplo: parallel.c

```

#include <stdio.h>
#include <stdlib.h>
#ifndef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(int argc, char **argv) {
    int thread;

    if(argc < 2) {
        fprintf(stderr, "\nFalta no de thread \n");
        exit(-1);
    }
    thread = atoi(argv[1]);
    #pragma omp parallel
    {
        if ( omp_get_thread_num() < thread )
            printf(" thread %d realiza la tarea 1\n",
                   omp_get_thread_num());
        else
            printf(" thread %d realiza la tarea 2\n",
                   omp_get_thread_num());
    }
    return(0);
}
```

Entrada programa: numero de thread (variable thread) -> parallel 4

Los threads con identificador menor que thread imprimen un mensaje y los que tienen identificador mayor imprimen otro distinto

```
mancia@mancia-ubuntu: ~/docencia/Organización de datos
Archivo Editar Ver Terminal Ayuda
mancia$ gcc -O2 -fopenmp -o parallel parallel.c
mancia$ export OMP_DYNAMIC=FALSE
mancia$ export OMP_NUM_THREADS=8
mancia$ parallel 3
  Hebra 1 realiza la tarea 1
  Hebra 2 realiza la tarea 1
  Hebra 3 realiza la tarea 2
  Hebra 5 realiza la tarea 2
  Hebra 0 realiza la tarea 1
  Hebra 6 realiza la tarea 2
  Hebra 4 realiza la tarea 2
  Hebra 7 realiza la tarea 2
mancia$ export OMP_NUM_THREADS=4
mancia$ parallel 3
  Hebra 2 realiza la tarea 1
  Hebra 1 realiza la tarea 1
  Hebra 3 realiza la tarea 2
  Hebra 0 realiza la tarea 1
mancia$ parallel 1
  Hebra 1 realiza la tarea 2
  Hebra 3 realiza la tarea 2
  Hebra 2 realiza la tarea 2
  Hebra 0 realiza la tarea 1
mancia$
```

Compilación con gcc.

Fija variables de entorno con export (ksh, bash).

Para fijar variables con setenv (csh):

setenv OMP_DYNAMIC FALSE

setenv OMP_NUM_THREADS 8

Como fijar variables en DOS:

set OMP_DYNAMIC=FALSE

set OMP_NUM_THREADS=8

Ejecuciones con diferentes parámetro de entrada

Directivas de trabajo compartido

DIRECTIVA	ejecutable	declarativa	
Con bucle estructurado	parallel sections, worksharing, single master critical ordered		Con sentencia
bucle	DO/for		Con sentencia
simple (una sentencia)	atomic		Con sentencia
autónoma (sin código asociado)	barrier , flush	threadprivate	Sin sentencia

Fortran: worksharing y DO

Para distribuir las iteraciones de un bucle entre las threads (paralelismo de datos) -> C/C++:
#pragma omp for

Para distribuir trozos de código independientes entre las threads (paralelismo de tareas) -> C/C++:
+: #pragma omp sections

Para que uno de los threads ejecute un trozo de código secuencial -> C/C++: #pragma omp single

Directiva bucle (DO/for)

```
#pragma omp for [clause[,clause]...]
for-loop
```

Tipos de paralelismo -> paralelismo de datos o paralelismo a nivel de bucle

Tipos de estructuras de procesos/tareas -> Implícita: descomposición de dominio, divide y vencerás

Sincronización -> Barrera implícita al final, no al principio

Características de los bucles -> Se tiene que conocer el nº de iteraciones, la variable de iteración debe ser un entero. No pueden ser de tipo do ... while. Formato usual C: for(var=lb ; var relational-op b ; var += incr). Las iteraciones se deben poder parallelizar (la herramienta no extrae paralelismo)

Asignación de tareas a threads -> La herramienta de programación decide cómo hacer la asignación a no ser que se use la cláusula `schedule`

Directiva for

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

int main(int argc, char **argv) {
    int i, n = 9;

    if(argc < 2) {
        fprintf(stderr, "\n[ERROR] - Falta no iteraciones \n");
        exit(-1);
    }
    n = atoi(argv[1]);
    #pragma omp parallel
    {
        #pragma omp for
        for (i=0; i<n; i++)
            printf("thread %d ejecuta la iteración %d del
bucle\n", omp_get_thread_num(),i);
    }
    return 0;
}
```

Entrada programa: número de iteraciones (n) -> bucle-for 8

Los threads imprimen las iteraciones que ejecutan

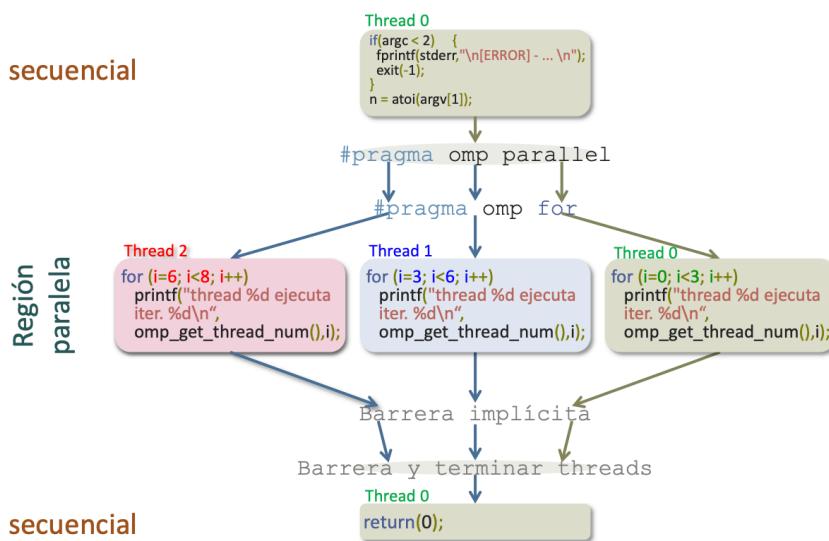
Compilación con gac

Ejemplos de variables de control interno con experto

-Ejecuciones con parámetro de entrada igual a 8 (no de iteraciones del bucle)

```
mancia@mancia-ubuntu: ~/docencia/Oper_ - 
Archivo Editar Ver Terminal Ayuda
mancia $gcc -O2 -fopenmp -o bucle-for bucle-for.c
mancia $export OMP_DYNAMIC=FALSE
mancia $export OMP_NUM_THREADS=8
mancia $bucle-for 8
Hebra 7 ejecuta la iteración 7 del bucle
Hebra 1 ejecuta la iteración 1 del bucle
Hebra 2 ejecuta la iteración 2 del bucle
Hebra 3 ejecuta la iteración 3 del bucle
Hebra 4 ejecuta la iteración 4 del bucle
Hebra 5 ejecuta la iteración 5 del bucle
Hebra 6 ejecuta la iteración 6 del bucle
Hebra 0 ejecuta la iteración 0 del bucle
mancia $export OMP_NUM_THREADS=4
mancia $bucle-for 8
Hebra 3 ejecuta la iteración 6 del bucle
Hebra 3 ejecuta la iteración 7 del bucle
Hebra 1 ejecuta la iteración 2 del bucle
Hebra 1 ejecuta la iteración 3 del bucle
Hebra 2 ejecuta la iteración 4 del bucle
Hebra 2 ejecuta la iteración 5 del bucle
Hebra 0 ejecuta la iteración 0 del bucle
Hebra 0 ejecuta la iteración 1 del bucle
mancia $export OMP_NUM_THREADS=2
mancia $bucle-for 8
Hebra 1 ejecuta la iteración 4 del bucle
Hebra 1 ejecuta la iteración 5 del bucle
Hebra 1 ejecuta la iteración 6 del bucle
Hebra 1 ejecuta la iteración 7 del bucle
Hebra 0 ejecuta la iteración 0 del bucle
Hebra 0 ejecuta la iteración 1 del bucle
Hebra 0 ejecuta la iteración 2 del bucle
Hebra 0 ejecuta la iteración 3 del bucle
mancia $
```

Directiva for. Reparto de trabajo.



Directiva sections

```
#pragma omp sections [clause[.,]clause]...
{
    [#pragma omp section]
    structured block
    [#pragma omp section]
    structured block
    ...
}
```

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

void funcA() {
    printf("En funcA: esta sección la ejecuta el thread
    %d\n",
    omp_get_thread_num());
}

void funcB() {
    printf("En funcB: esta sección la ejecuta el thread
    %d\n",
    omp_get_thread_num());
}

main() {
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        (void) funcA();
        #pragma omp section
        (void) funcB();
    }
}
}
```

Tipo de parallelismo: de tareas o a nivel de función

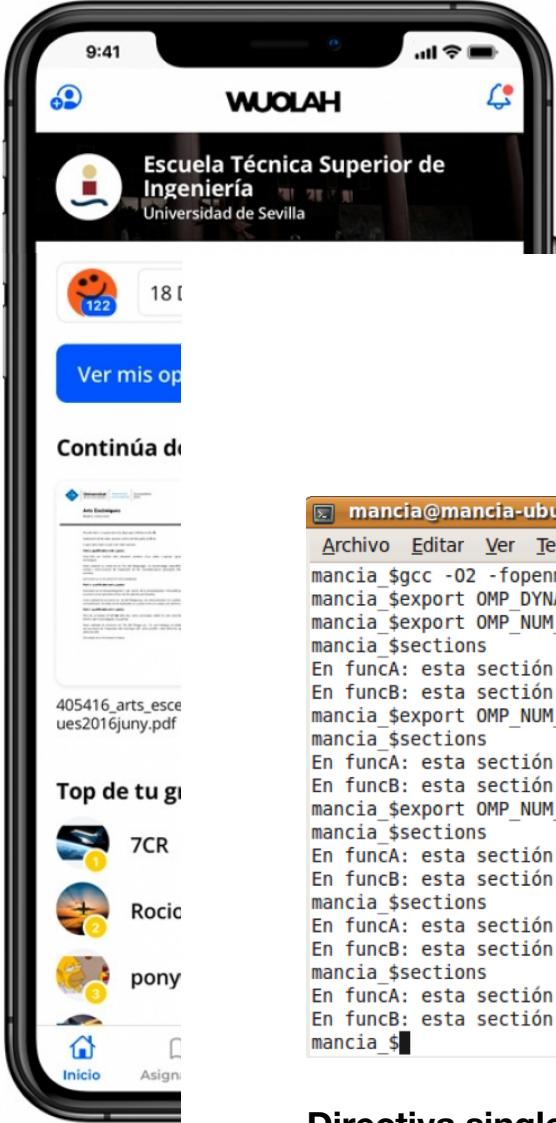
Tipos de estructuras de procesos/tareas: Explícita: Mater/slave, cliente/servidor, flujo de datos, descomposición de dominio, divide y vencerás

Sincronización: Barrera implícita al final, no al principio

Asignación de tareas a threads concretos: No la hace el programador, lo hace la herramienta

Un thread ejecuta funcA y otro funcB

El número de thread que ejecutan trabajo dentro de un sections coincide con el número de section

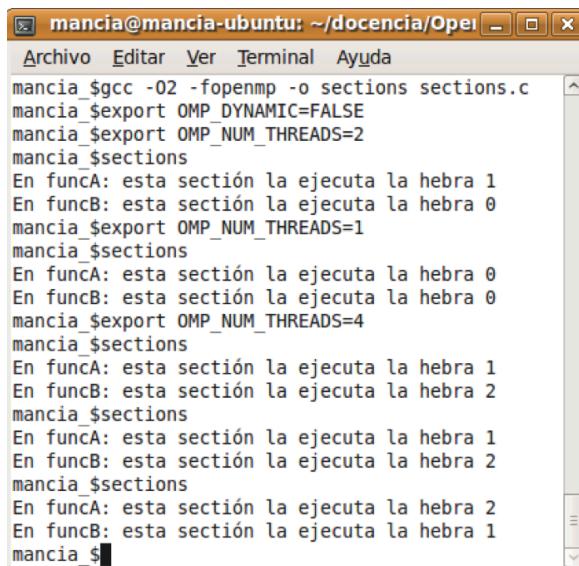


Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store  GET IT ON Google Play 

Continúa d



```
mancia@mancia-ubuntu: ~/docencia/Operadores/parallel/sections
Archivo  Editar  Ver  Terminal  Ayuda
mancia$ gcc -O2 -fopenmp -o sections sections.c
mancia$ export OMP_DYNAMIC=FALSE
mancia$ export OMP_NUM_THREADS=2
mancia$ sections
En funcA: esta sección la ejecuta la hebra 1
En funcB: esta sección la ejecuta la hebra 0
mancia$ export OMP_NUM_THREADS=1
mancia$ sections
En funcA: esta sección la ejecuta la hebra 0
En funcB: esta sección la ejecuta la hebra 0
mancia$ export OMP_NUM_THREADS=4
mancia$ sections
En funcA: esta sección la ejecuta la hebra 1
En funcB: esta sección la ejecuta la hebra 2
mancia$ sections
En funcA: esta sección la ejecuta la hebra 1
En funcB: esta sección la ejecuta la hebra 2
mancia$ sections
En funcA: esta sección la ejecuta la hebra 2
En funcB: esta sección la ejecuta la hebra 1
mancia$
```

Compilación con gcc

Fijar variables de entorno con export (ksh, bash)

Ejecuciones con diferente número de threads

Directiva single

```
#pragma omp sections [clause[.,]clause]...
{
    structured block
```

Ejecución de un trozo secuencial por un thread, útil cuando el código a ejecutar no es thread-safe (E/S)

Sincronización: barrera implícita al final

Asignación de tareas a threads:

Cualquiera de los threads puede ejecutar el trabajo del bloque estructurado (no lo controla el programador)

Inicializa un vector a un valor solicitado al usuario

```
#include <stdio.h>
#include <omp.h>
main()
{
    int n = 9, i, a, b[n];
    for (i=0; i<n; i++) b[i] = -1;
    #pragma omp parallel
    {
        #pragma omp single
        { printf("Introduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("Single ejecutada por el thread %d\n",
                   omp_get_thread_num());
        }

        #pragma omp for
        for (i=0; i<n; i++)
        b[i] = a;
    }

    printf("Después de la región parallel:\n");
    for (i=0; i<n; i++) printf("b[%d] = %d\n", i, b[i]);
    printf("\n");
}
```

```

formacion01@manager2:~/lección2/i
Archivo Editar Ver Terminal Ayuda
$ gcc -O2 -fopenmp single.c -o single
$ export OMP_DYNAMIC=FALSE
$ export OMP_NUM_THREADS=8
$ single
Introduce valor de inicialización a: 23
Single ejecutada por la hebra 1
Después de la región parallel:
b[0] = 23      b[1] = 23      b[2] = 23
b[3] = 23      b[4] = 23      b[5] = 23
b[6] = 23      b[7] = 23      b[8] = 23
$ export OMP_NUM_THREADS=3
$ single
Introduce valor de inicialización a: 12
Single ejecutada por la hebra 0
Después de la región parallel:
b[0] = 12      b[1] = 12      b[2] = 12
b[3] = 12      b[4] = 12      b[5] = 12
b[6] = 12      b[7] = 12      b[8] = 12
$ 

```

Compilación con gcc

Fijar variables de entorno con export

Varias ejecuciones

Combinar parallel con directivas de trabajo compartido

```
#pragma omp parallel [clauses]
  #pragma omp for [clauses]
  for-loop
```

```
#pragma omp parallel for [clauses]
  for-loop
```

```
#pragma omp parallel [clauses]
  #pragma omp sections [clauses]
{
  [#pragma omp section ]
  structured block
  [#pragma omp section
  structured block]
...
}
```

```
#pragma omp parallel sections [clauses]
{
  [#pragma omp section ]
  structured block
  [#pragma omp section
  structured block]
  ...
}
```

Cláusulas: la directiva combinada admite las cláusulas de las dos directivas, excepto **nowait**
 Diferencias con la alternativa que no combina: legibilidad, prestaciones

Directivas básicas para comunicación y sincronización

Directivas/construcciones

DIRECTIVA	ejecutable	declarativa	
Con bucle estructurado	parallel sections, worksharing, single master critical ordered		Con sentencia
bucle	DO/for		Con sentencia
simple (una sentencia)	atomic		Con sentencia

DIRECTIVA	ejecutable	declarativa	
autónoma (sin código asociado)	barrier , flush	threadprivate	Sin sentencia

Directiva barrier

```
#pragma omp barrier
```

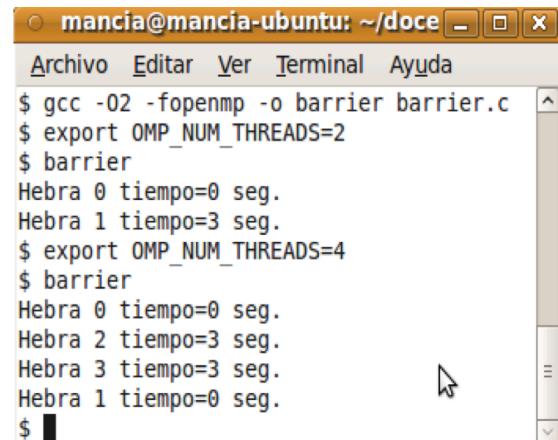
Barrera: punto en el código en el que los threads se esperan entre sí.

```
#include <stdlib.h>
#include <time.h>
#ifndef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif
main() { int tid;
time_t t1,t2;

#pragma omp parallel private(tid,t1,t2)
{
    tid = omp_get_thread_num();
    if (tid < omp_get_num_threads()/2) system("sleep 3");
    t1= time(NULL);
    #pragma omp barrier
    t2= time(NULL)-t1;

    printf("Tiempo=%d seg.\n", t2);
}
}
```

Al final de parallel y de las construcciones de trabajo compartido hay una barrera implícita



```
mancia@mancia-ubuntu: ~/doce
Archivo Editar Ver Terminal Ayuda
$ gcc -O2 -fopenmp -o barrier barrier.c
$ export OMP_NUM_THREADS=2
$ barrier
Hebra 0 tiempo=0 seg.
Hebra 1 tiempo=3 seg.
$ export OMP_NUM_THREADS=4
$ barrier
Hebra 0 tiempo=0 seg.
Hebra 2 tiempo=3 seg.
Hebra 3 tiempo=3 seg.
Hebra 1 tiempo=0 seg.
$
```

Algunos threads tienen que esperar 3 segundos en la barrera

Directiva critical

```
#pragma omp critical [(name)]
bloque estructurado
```

Evita que varios threads accedan a variables compartidas a la vez (evita race conditions)

“name” permite evitar deadlock

Sección crítica: código que accede a variables compartidas

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n=20, a[n], suma=0,sumalocal;
    if(argc < 2) {
        fprintf(stderr, "\nFalta iteraciones\n"); exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;

    for (i=0; i<n; i++) a[i] = i;

#pragma omp parallel private(sumalocal)
{ sumalocal=0;
#pragma omp for schedule(static)
    for (i=0; i<n; i++)
    { sumalocal += a[i];
        printf(" thread %d suma de a[%d]=%d sumalocal=%d\n",
            omp_get_thread_num(),i,a[i],sumalocal);
    }
#pragma omp critical
    suma = suma + sumalocal;
}
    printf("Fuera de 'parallel' suma=%d\n",suma); return(0);
}
```

```

mancia@mancia-ubuntu: ~/docen - Terminal Ayuda
$ gcc -O2 -fopenmp -o critical critical.c
$ export OMP_NUM_THREADS=2
$ critical 6
Hebra 1 suma de a[3]=3 sumalocal=3
Hebra 1 suma de a[4]=4 sumalocal=7
Hebra 1 suma de a[5]=5 sumalocal=12
Hebra 0 suma de a[0]=0 sumalocal=0
Hebra 0 suma de a[1]=1 sumalocal=1
Hebra 0 suma de a[2]=2 sumalocal=3
Fuera de 'parallel' suma=15
$ export OMP_NUM_THREADS=3
$ critical 6
Hebra 2 suma de a[4]=4 sumalocal=4
Hebra 2 suma de a[5]=5 sumalocal=9
Hebra 1 suma de a[2]=2 sumalocal=2
Hebra 1 suma de a[3]=3 sumalocal=5
Hebra 0 suma de a[0]=0 sumalocal=0
Hebra 0 suma de a[1]=1 sumalocal=1
Fuera de 'parallel' suma=15
$ 

```

Ilustra que en la sección crítica entra un thread detrás de otro

Directiva atomic +,*,/,&,^,|,<>

Puede ser una alternativa a critical más eficiente.

```

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

main(int argc, char **argv) {
    int i, n=20, a[n], suma=0, sumalocal;
    if(argc < 2) {
        fprintf(stderr, "\nFalta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;

    for (i=0; i<n; i++) a[i] = i;

    #pragma omp parallel private(sumalocal)
    { sumalocal=0;
        #pragma omp for schedule(static)
        for (i=0; i<n; i++)
        { sumalocal += a[i];
            printf(" thread %d suma de a[%d]=%d sumalocal=%d
\n", omp_get_thread_num(), i, a[i], sumalocal);
        }
        #pragma omp atomic
        suma += sumalocal;
    }
    printf("Fuera de 'parallel' suma=%d\n", suma); return(0);
}

```

```
#pragma omp atomic
```

```
x <binop> = expre.
```

...

```
#pragma omp atomic
```

```
x++, ++x, x-- o --x
```

```

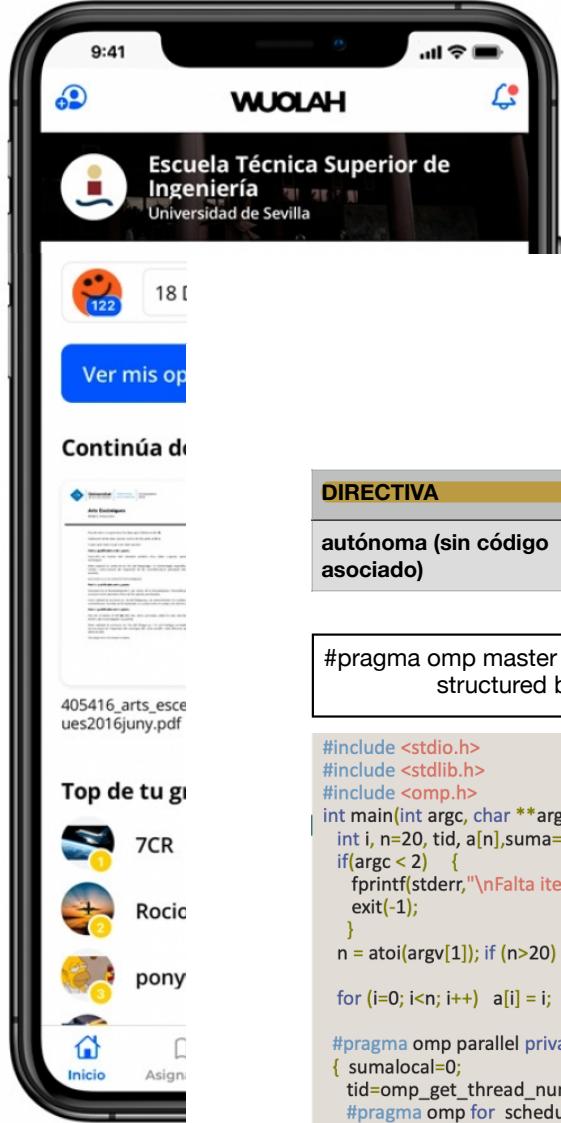
mancia@mancia-ubuntu: ~/docen - Terminal Ayuda
$ gcc -O2 -fopenmp -o atomic atomic.c
$ export OMP_NUM_THREADS=2
$ atomic 6
Hebra 1 suma de a[3]=3 sumalocal=3
Hebra 1 suma de a[4]=4 sumalocal=7
Hebra 1 suma de a[5]=5 sumalocal=12
Hebra 0 suma de a[0]=0 sumalocal=0
Hebra 0 suma de a[1]=1 sumalocal=1
Hebra 0 suma de a[2]=2 sumalocal=3
Fuera de 'parallel' suma=15
$ export OMP_NUM_THREADS=3
$ atomic 6
Hebra 2 suma de a[4]=4 sumalocal=4
Hebra 2 suma de a[5]=5 sumalocal=9
Hebra 1 suma de a[2]=2 sumalocal=2
Hebra 1 suma de a[3]=3 sumalocal=5
Hebra 0 suma de a[0]=0 sumalocal=0
Hebra 0 suma de a[1]=1 sumalocal=1
Fuera de 'parallel' suma=15
$ 

```

En la sección crítica entra un thread detrás de otro

Directiva master

DIRECTIVA	ejecutable	declarativa	
Con bucle estructurado	<u>parallel</u> <u>sections</u> , <u>worksharing</u> , <u>single</u> master critical ordered		Con sentencia
bucle	<u>DO/for</u>		Con sentencia
simple (una sentencia)	atomic		Con sentencia



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play



18

Ver mis op

Continúa d

DIRECTIVA	ejecutable	declarativa	
autónoma (sin código asociado)	barrier , flush	threadprivate	Sin sentencia

#pragma omp master
structured block

No es una directiva de trabajo compartido.
No tiene barreras implícitas

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv) {
    int i, n=20, tid, a[n], suma=0,sumalocal;
    if(argc < 2) {
        fprintf(stderr, "\nFalta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;
    for (i=0; i<n; i++) a[i] = i;

    #pragma omp parallel private(sumalocal,tid)
    { sumalocal=0;
        tid=omp_get_thread_num();
        #pragma omp for schedule(static)
        for (i=0; i<n; i++)
        { sumalocal += a[i];
            printf(" thread %d suma de a[%d]=%d sumalocal=%d
\n", tid,i,a[i],sumalocal);
        }
        #pragma omp atomic
        suma += sumalocal;
        #pragma omp barrier
        #pragma omp master
        printf("thread master=%d imprime suma=%d\n",
tid,suma);
    }
}
```

```
$ gcc -O2 -fopenmp -o master master.c
$ export OMP_NUM_THREADS=3
$ master 6
Hebra 0 suma de a[0]=0 sumalocal=0
Hebra 0 suma de a[1]=1 sumalocal=1
Hebra 1 suma de a[2]=2 sumalocal=2
Hebra 1 suma de a[3]=3 sumalocal=5
Hebra 2 suma de a[4]=4 sumalocal=4
Hebra 2 suma de a[5]=5 sumalocal=9
Hebra master=0 imprime suma=1
$ master 6
Hebra 2 suma de a[4]=4 sumalocal=4
Hebra 2 suma de a[5]=5 sumalocal=9
Hebra 1 suma de a[2]=2 sumalocal=2
Hebra 1 suma de a[3]=3 sumalocal=5
Hebra 0 suma de a[0]=0 sumalocal=0
Hebra 0 suma de a[1]=1 sumalocal=1
Hebra master=0 imprime suma=15
```

Problema: Se ha quitado la directiva barrier antes de la directiva master

Consecuencia: la suma no siempre es correcta

¿Cómo calcular los MFLOPS y MIPS?

MIPS -> (N) / (tiempo * 10^6inst/s)

MFLOPS -> (N en coma flotante) / (tiempo * 10^6inst/s)

¿Cuántos threads y cores podría utilizar como máximo con parallel for?

Lo ideal sería tener tantas hebras como número de iteraciones tengamos (el for divide N tareas entre N hebras)

¿Cuántos threads y cores podría utilizar como máximo con parallel sections?

Para sections lo ideal es tener tantas hebras como sections tengamos (es decir, depende de los sections que hayamos puesto)

¿Cómo se calcula el tiempo real del sistema?

El **real time** de un sistema es igual a **user time + sys time**, pero a veces, la suma de **user time + sys time** puede ser menor que el **tiempo real** del sistema, esto puede estar provocado porque el proceso puede haber estado suspendido. La suma de tiempos es solo el tiempo de ejecución mientras que el **tiempo real** mide desde que se lanza la ejecución hasta que finaliza, teniendo en cuenta el tiempo que estuvo suspendido el proceso. La suma **user+sys** no lo tiene en cuenta. Si la suma de **user time + sys time** fuese mayor que el **tiempo real**, es porque se usa paralelismo.

Pues pondria algo mas simle rollo asi La suma es menor ya que al ejecutar un programa secuencial se cumple: $T_{real} \geq T_{user} + T_{system}$.

En el caso anterior seria: $0,59 \geq 0,52 + 0,04$

Cuando es un programa paralelo (creo) que no siempre se cumple, ya que el tiempo de user + t de system que son los tiempos de cpu como usuario y como kernel, pues estos tiempos realmente son la suma de los tiempos de las distintas hebras que lo hayan ejecutado.

Cuando se paraleliza no se cumple ya que el t de usuario, time hace la suma del tiempo de ejecucion de cada cpu.

¿Por qué si se elimina directiva barrier en el ejemplo master.c la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

Porque la directiva master no dispone de barre implícita, por lo que es necesario añadir una directiva barrier antes para el correcto funcionamiento. El código master.c realiza la suma de las componentes de un vector en paralelo, entonces se puede dar el caso en el que la hebra maestra llegue y ejecute la estructura de la cabecera master y todavía las otras hebras no hayan terminado de ejecutar la suma. Básicamente la barrera se pone para esperar a que todas las hebras hayan acabado de realizar las sumas y así no imprimir el valor antes de tiempo. Es por eso por lo que es necesario la directiva barrier antes de master, para evitar condición de carrera y que todas las hebras se sincronicen adecuadamente.

Con la directiva master obligamos a que las secciones de código sean ejecutadas siempre por la hebra 0.

¿Qué directivas tienen barrera implícita al final?

Parallel, sections, worksharing, single, DO/for

¿Cómo se compila hello.c?

gcc -O2 -fopenmp -o hello hello.c

Hebra 0 = master

Variable de entorno para fijar hebras -> OMP_NUM_THREADS

También se pueden fijar las variables de entorno con **export** -> export OMP_NUM_THREADS=8

¿Cómo se distribuyen las iteraciones de un bucle entre las threads (parallelismo de datos) en directivas de trabajo compartido?

con #pragma omp for

¿Cómo se distribuyen los trozos de código independientes entre las threads (parallelismo de tareas) en directivas de trabajo compartido?

#pragma omp sections

¿Qué se debe hacer para que uno de los threads ejecute un trozo de código secuencial en directivas de trabajo compartido?

#pragma omp single

¿Qué tipo de paralelismo tiene la directiva bucle (DO/for)?

paralelismo de datos o paralelismo a nivel de bucle

¿Qué tipos de estructuras de procesos/tareas tiene la directiva bucle (DO/for)?

Implícita: descomposición de dominio, divide y vencerás

¿Cómo se hace la asignación de tareas a threads en la directiva bucle (DO/for)?

La herramienta de programación decide cómo hacer la asignación a no ser que se use la cláusula **schedule**

¿Qué tipo de paralelismo tiene la directiva sections?

parallelismo de tareas o a nivel de función

¿Qué tipos de estructuras de procesos/tareas tiene la directiva sections?

Explícita: Mater/slave, cliente/servidor, flujo de datos, descomposición de dominio, divide y vencerás

¿Cómo se hace la asignación de tareas a threads en la directiva sections?

La asignación la realiza la herramienta de programación

La directiva single es la ejecución de un trozo secuencial por un thread, es útil cuando el código a ejecutar no es thread-safe

¿Cómo se hace la asignación de tareas a threads en la directiva single?

Cualquiera de los threads puede ejecutar el trabajo del bloque estructurado (no lo controla el programador)

¿Qué es una barrera?

Un punto en el código en el que los threads se esperan entre sí

La directiva critical evita que varios threads accedan a variables compartidas a la vez (evita race conditions)

En las directivas critical, "name" permite evitar deadlock.

En las directivas critical y atomic, en la sección crítica entre un thread detrás de otro

La directiva atomic es una directiva critical pero más eficiente

¿Tiene la directiva master barreras implícitas?

No

¿Es la directiva master una directiva de trabajo compartido?

No

¿Cuál de las siguientes no es una directiva de OpenMP?

- a) exclusive
- b) for
- c) single
- d) master

La API de programación de OpenMP está formada por...

- a) Ninguna de las otras respuestas
- b) **Directivas del compilador, funciones y variables de entorno**
- c) Variables del compilador y funciones para medición de tiempo en programas con varias hebras
- d) Variables de entorno definidas en consola y funciones de paso de mensajes

¿Para qué sirve la directiva barrier?

- a) Para que todas las hebras vayan a la misma velocidad
- b) **Para fijar un punto en el código que ninguna hebra podrá sobrepasar hasta que lo hayan alcanzado todas las demás**
- c) Para evitar las condiciones de carrera
- d) Para proteger el acceso a una variable compartida

¿Cuántas hebras ejecutarán una directiva con 4 secciones (section) en una plataforma con 4 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2? (considere que no se usan en el código funciones OpenMP)

- a) 2

- b) 3
- c) 1
- d) 4

¿Cuál de las siguientes afirmaciones acerca de sections es cierta?

- a) La sincronización se realiza al final de la directiva, pero no al principio**
- b) La sincronización se realiza al principio de la directiva, pero no al final
- c) La sincronización se realiza al principio y al final de la directiva
- d) No se realiza sincronización

¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva critical en una plataforma con 3 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2?

- a) 4
- b) 1**
- c) 2
- d) 3

¿Cuál de las siguientes afirmaciones acerca de la directiva master es cierta?

- a) Las hebras se sincronizan al principio de la directiva, pero no al final.
- b) Las hebras se sincronizan al final de la directiva, pero no al principio.
- c) Las hebras no se sincronizan ni al principio ni al final de la directiva**
- d) Las hebras se sincronizan al principio y al final de la directiva.

Cuando se mide el tiempo de ejecución de un programa mediante la orden del sistema time , la suma de los tiempos de usuario y sistema ..

- a) Es siempre menor o igual que el tiempo de ejecución para programas paralelos
- b) Es siempre igual que el tiempo de ejecución
- c) Es siempre menor o igual que el tiempo de ejecución
- d) Es siempre menor o igual que el tiempo de ejecución para programas secuenciales**

¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva critical en una plataforma con 4 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2?

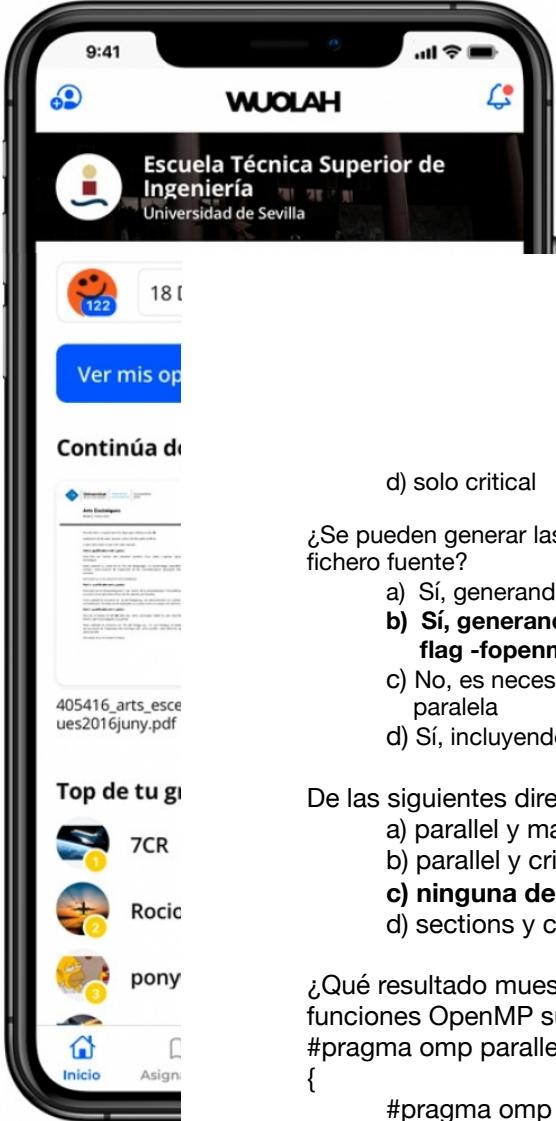
- a) 4
- b) 1**
- c) 2
- d) 3

¿Cuántas hebras ejecutarán una directiva con 4 secciones (section) en una plataforma con 8 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 3? (considere que no se usan en el código funciones OpenMP)

- a) 2
- b) 1
- c) Ninguna
- d) 3**

¿Qué directiva usarías para que las hebras ejecuten el siguiente código en exclusión mutua? {b -= 2}

- a) solo atomic
- b) critical o atomic**
- c) exclusive



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Continúa d

d) solo critical

¿Se pueden generar las versiones secuencial y paralela de un programa a partir de un mismo fichero fuente?

- a) Sí, generando código condicionado a la existencia del símbolo _OPENMP
- b) Sí, generando código condicionado al símbolo _OPENMP e incluyendo o no el flag -fopenmp a la hora de compilar**
- c) No, es necesario tener dos ficheros fuente, uno para la versión secuencial y otro para la paralela
- d) Sí, incluyendo o no el flag -fopenmp a la hora de compilar

405416_arts_esce
ues2016juniy.pdf

Top de tu g

7CR

Rocic

pony

Inicio Asign.

De las siguientes directivas de OpenMP, ¿cuáles incorporan una barrera implícita al final?

- a) parallel y master
- b) parallel y critical
- c) ninguna de las otras respuestas es correcta**
- d) sections y critical

¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=3?

#pragma omp parallel

```
{  
    #pragma omp single  
    {  
        printf("x");  
    }  
}
```

- a) indeterminado, existe condición de carrera
- b) x**
- c) xxx
- d) xx

¿Qué directivas admiten una forma combinada?

- a) Todas las respuestas anteriores son correctas
- b) parallel y schedule
- c) parallel y reduction
- d) parallel y for**

¿Qué identificador tiene la hebra master en la ejecución de un programa paralelo?

- a) El 0**
- b) El valor que defina el usuario mediante la función OpenMP
omp_set_master_num()
- c) Un valor numérico arbitrario, dependiente del entorno de ejecución
- d) Un valor numérico arbitrario, dependiente de los identificadores que usaron en la ejecución anterior

¿Cuándo existe condición de carrera?

- a) Solamente si se garantiza el acceso en exclusión mutua a un recurso compartido
- b) Cuando el resultado de un programa paralelo depende del orden de ejecución de sus hebras**
- c) Siempre que haya mas de una hebra en ejecución

- d) Siempre que diferentes hebras comparten una variable

¿Cuál de las siguientes directivas no incorpora barrera implícita al final?

- a) for
- b) atomic**
- c) parallel
- d) sections

¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=3?

```
#pragma omp parallel
{
    #pragma omp critical
    {
        printf("x");
    }
}
```

a) xx
b) xxxx
c) xxx
d) x

¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva critical en una plataforma con 4 cores en las que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2?

- a) 3
- b) 4
- c) 1**
- d) 2

¿Cuántas hebras ejecutarán una directiva sections con 4 secciones (section) en una plataforma con 8 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 3? (considere que no se usan en el código funciones OpenMP)

- a) 2
- b) 1
- c) 3**
- d) ninguna