

2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores

Seminario 1. Herramientas de programación paralela I: Directivas OpenMP

Material elaborado por los profesores responsables de la asignatura:

Mancia Anguita – Julio Ortega

Licencia Creative Commons



ugr

Universidad
de Granada

ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



ATC

Departamento de Arquitectura
y Tecnología de Computadores
UNIVERSIDAD DE GRANADA



¿Qué es 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?

- Es una API para escribir **código paralelo** con el paradigma/estilo de programación de **variables compartidas** para ejecutar aplicaciones en paralelo en varios **threads**.
- API (*Application Programming Interface*):
 - **Capa de abstracción** que permite al programador acceder cómodamente a través de una interfaz a un conjunto de funcionalidades.
- La API OpenMP define/comprende:
 - **Directivas** del compilador, **funciones** de biblioteca, y **variables** de entorno.

API OpenMP

Usuario

Usuario

Programación

API

Aplicación

Código C++/C o Fortran

Directivas

Biblioteca OpenMP

Variables de
entorno

Sistema

Biblioteca de tiempo de ejecución de OpenMP, variables de control

Soporte del sistema/SO para memoria compartida y threads

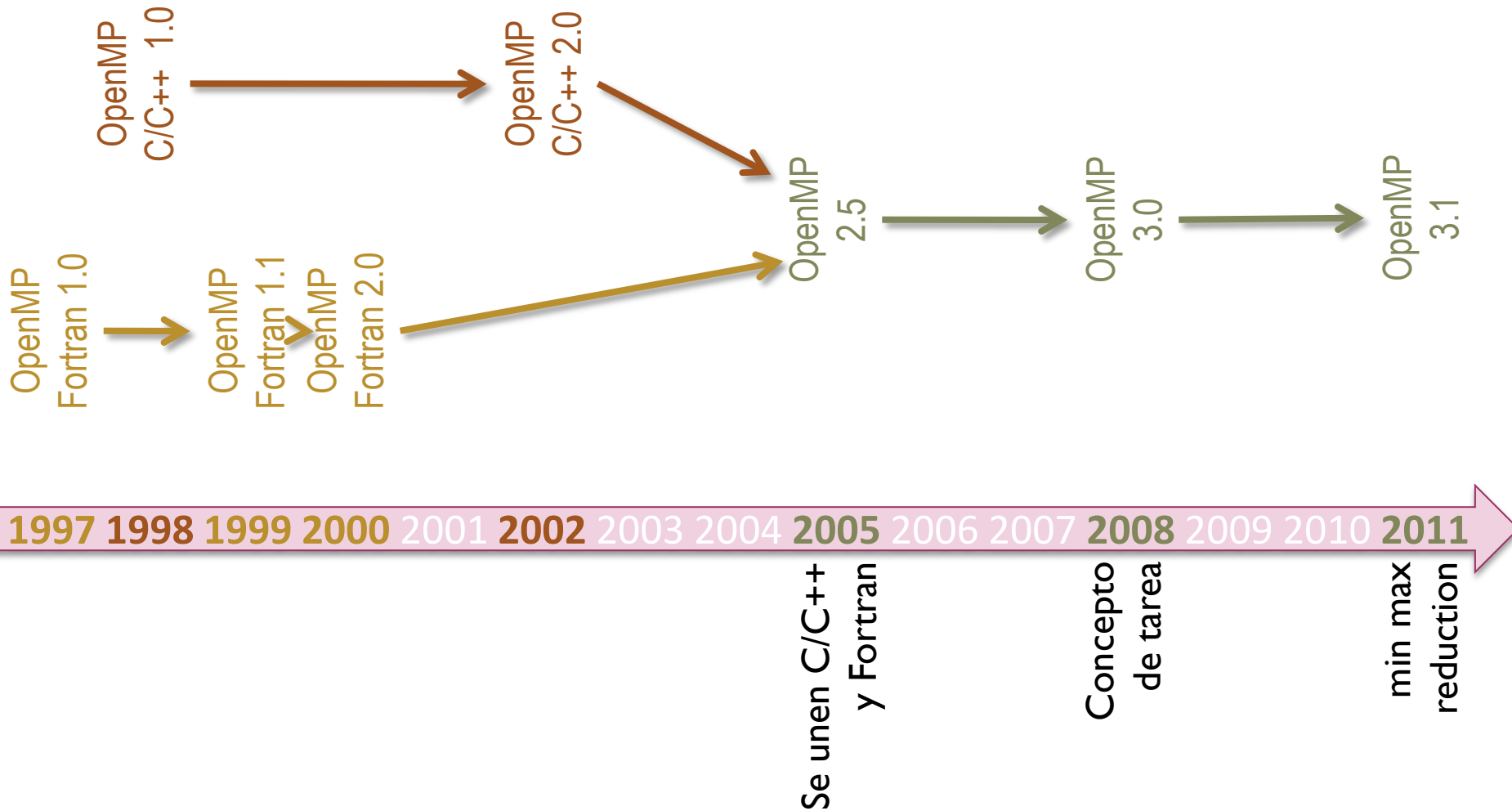
Hardware

Multiprocesador (SMP, NUMA)

¿Qué es OpenMP?

- Es una herramienta para programación paralela:
 - No automática (no extrae paralelismo implícito)
 - Con un modelo de programación:
 - Basado el paradigma/estilo de variables compartidas (Lección 4/Tema 2)
 - Multithread
 - Basada en directivas del compilador y funciones (Lección 4/Tema2):
 - El código paralelo OpenMP es código escrito con un lenguaje secuencial (C, C++ o Fortran) + directivas y funciones de la interfaz OpenMP
 - Portable:
 - API especificada para C/C++ y Fortran (77, 90 y 95)
 - La mayor parte de las plataformas (SO/hardware) tienen implementaciones de OpenMP
 - ¿Estándar?
 - Se podría considerar en cuanto que lo han definido un conjunto de vendedores de hardware y software destacados

Evolución de OpenMP



Contenidos

- Componentes de OpenMP
- Directivas
- Directiva `parallel`
- Directivas para trabajo compartido (*worksharing*)
- Combinar `parallel` con directivas de trabajo compartido
- Directivas básicas para comunicación y sincronización
- Directiva `master`

Contenidos

- Componentes de OpenMP
- Directivas
- Directiva `parallel`
- Directivas para trabajo compartido (*worksharing*)
- Combinar `parallel` con directivas de trabajo compartido
- Directivas básicas para comunicación y sincronización
- Directiva `master`

Componentes de OpenMP

➤ Directivas

- El preprocesador del compilador las sustituye por código.

➤ Funciones

- P. ej. 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. <i>Coinciden los nombres en Fortran y C/C++ (salvo <code>for</code>)</i>	Opcional. <i>Pueden aparecer en cualquier orden. Coinciden los nombres con los de Fortran</i>	Necesario. <i>Precede al bloque estructurado que engloba la directiva</i>

- El **nombre** define y controla la acción que se realiza
 - Ej.: `parallel, for, section ...`
- Las **cláusulas** especifican adicionalmente la acción o comportamiento, la ajustan
 - Ej.: `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

- Directivas C/C++ (pragmas):
 - `#pragma omp <directive> [<clause>, <clause> ...]`
 - Para dividir en varias filas o líneas de código: “\”

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

Portabilidad

➤ Compilación C/C++

- gcc -fopenmp
- g++ -fopenmp

➤ Directivas

- Las directivas no se tendrán en cuenta si no se compila usando OpenMP:
 - -fopenmp, -openmp, etc.

➤ Funciones

- Se evitan usando compilación condicional. Para C/C++:
 - Usando `_OPENMP` y `#ifdef ... #endif`
 - `_OPENMP` se define cuando se compila usando OpenMP

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

C/C++

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

    . . .
    c=funcion();
    . . .
}
```

C/C++

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

    . . .
    c=funcion();
    . . .
}
```

Algunas definiciones

➤ 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`

C/C++

```
#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`

Contenidos

- Componentes de OpenMP
- Directivas
- Directiva `parallel`
- Directivas para trabajo compartido (*worksharing*)
- Combinar `parallel` con directivas de trabajo compartido
- Directivas básicas para comunicación y sincronización
- Directiva `master`

Directivas/constructores (v2.5)

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

- La directiva define la acción que se realiza
- Se han destacado en **color** las directivas que se van a comentar en este seminario
- Se han subrayado las directivas con barrera implícita al final

Contenidos

- Componentes de OpenMP. Portabilidad
- Directivas
- Directiva `parallel`
- Directivas para trabajo compartido (*worksharing*)
- Combinar `parallel` con directivas de trabajo compartido
- Directivas básicas para comunicación y sincronización
- Directiva `master`

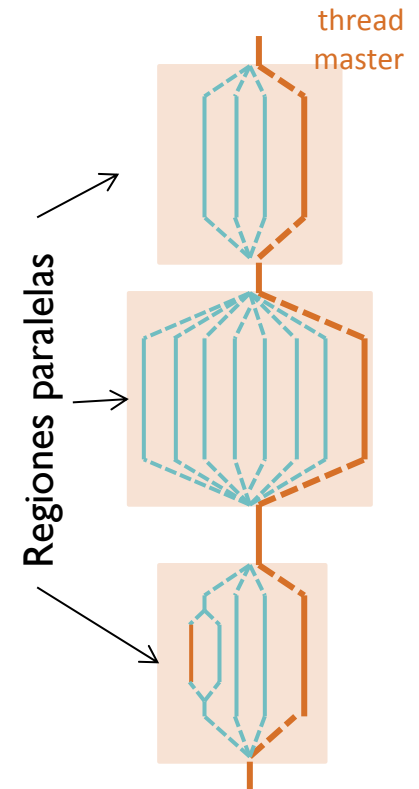
Directiva parallel

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

Directiva `parallel`

```
#pragma omp parallel [clause[:,clause]...]
    bloque estructurado
```

- 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

hello.c

```
#include <stdio.h>

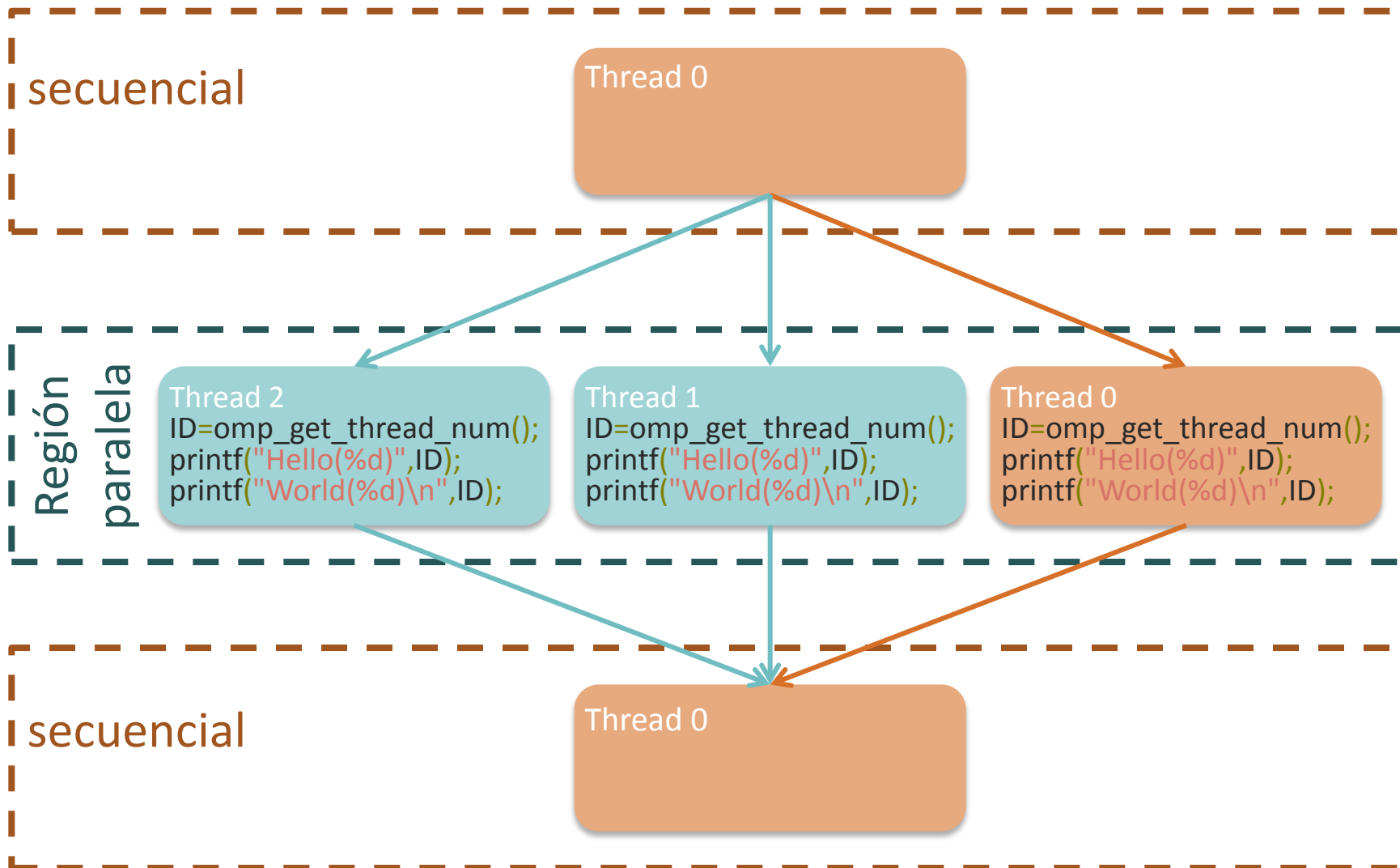
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

main() {

    int ID;
    #pragma omp parallel private(ID)
    {
        ID = omp_get_thread_num();
        printf("Hello(%d)", ID);
        printf("World(%d)\n", ID);
    }

}
```

Región paralela en Hello Word



Resultados hello.c

```
formacion01@manager1:~/leccion1
[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 nº de threads
 - Usamos variable 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`
- Fijado por defecto por la implementación: normalmente el **nº de cpu** de un nodo, aunque puede variar dinámicamente

parallel



- Entrada programa: nº 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

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _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);
}
```


Directiva Parallel. Salida

```
mancia@mancia-ubuntu: ~/docencia/Op
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`
- Cómo fijar variables en DOS:
 - `set OMP_DYNAMIC=FALSE`
 - `set OMP_NUM_THREADS=8`
- Ejecuciones con diferentes parámetro de entrada

Contenidos

- Componentes de OpenMP
- Directivas
- Directiva `parallel`
- Directivas para trabajo compartido (*worksharing*)
- Combinar `parallel` con directivas de trabajo compartido
- Directivas básicas para comunicación y sincronización
- Directiva `master`

Directivas de trabajo compartido

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

➤ Fortran: worksharing y DO

Directivas para trabajo compartido para C/C++

- 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 (Lección 1/Tema1):
 - paralelismo de datos o paralelismo a nivel de bucle
- Tipos de estructuras de procesos/tareas (Lección 4/Tema 2):
 - Implícita: descomposición de dominio, divide y vencerás
- Sincronización (Lección 4/Tema 2, Lección 10/Tema 3):
 - 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 paralelizar (la herramienta no extrae paralelismo)
- Asignación de tareas a threads (Lección 5/Tema 2):
 - La herramienta de programación decide cómo hacer la asignación a no ser que se use la cláusula `schedule`

Directiva `for`

bucle-for.c [Chapman 2008]



- Entrada programa: nº de iteraciones (n)
 - bucle-for 8
- Los threads imprimen las iteraciones que ejecutan

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

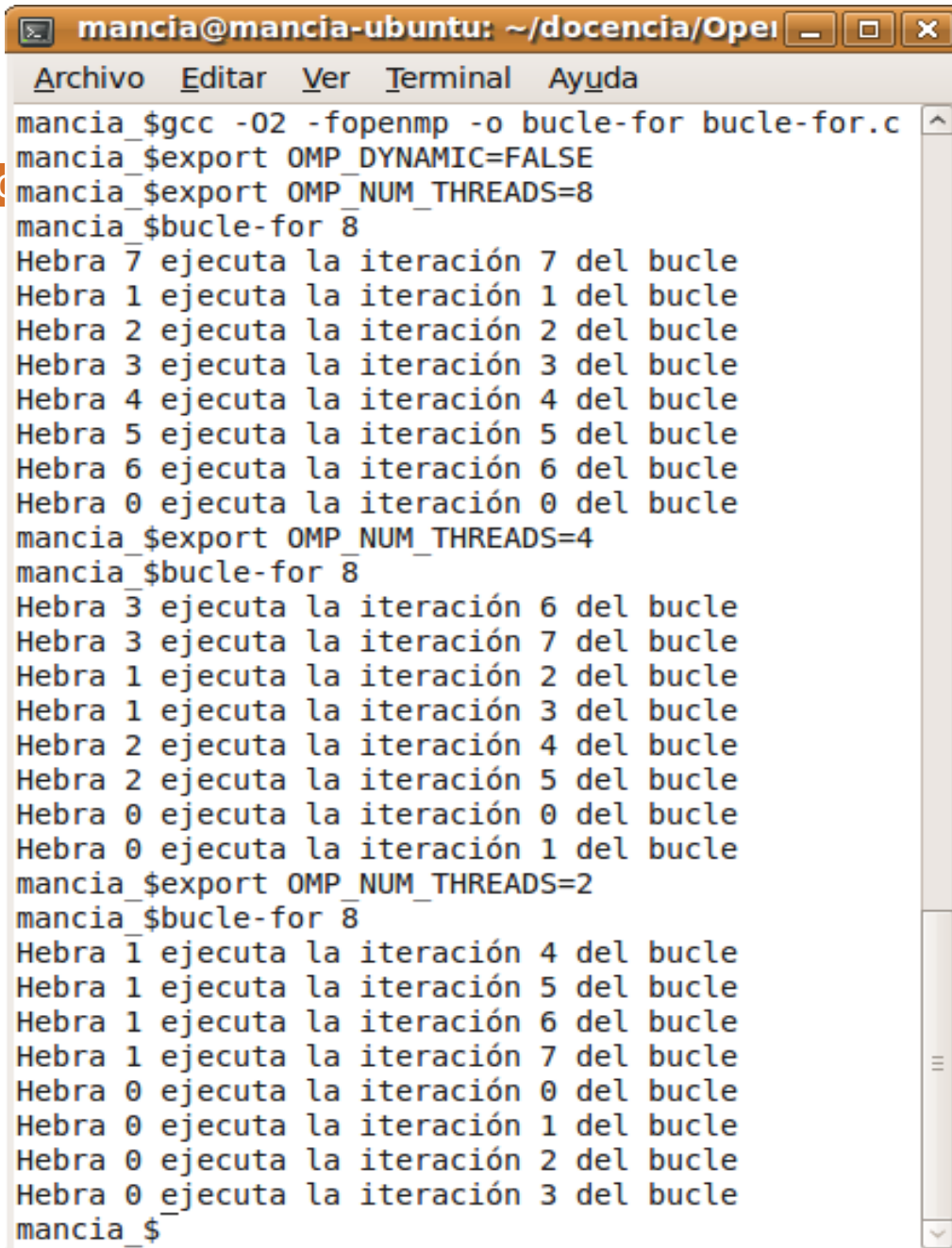
int main(int argc, char **argv) {

    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);
}
```

Directiva `for`. Salida



The screenshot shows a terminal window titled 'mancia@mancia-ubuntu: ~/docencia/OpenMP'. The menu bar includes 'Archivo', 'Editar', 'Ver', 'Terminal', and 'Ayuda'. The terminal content shows the compilation and execution of a program named 'bucle-for.c'. The first execution uses 8 threads, and the second uses 4 threads. The output shows threads (Hebra) executing iterations of a loop. The third execution uses 2 threads.

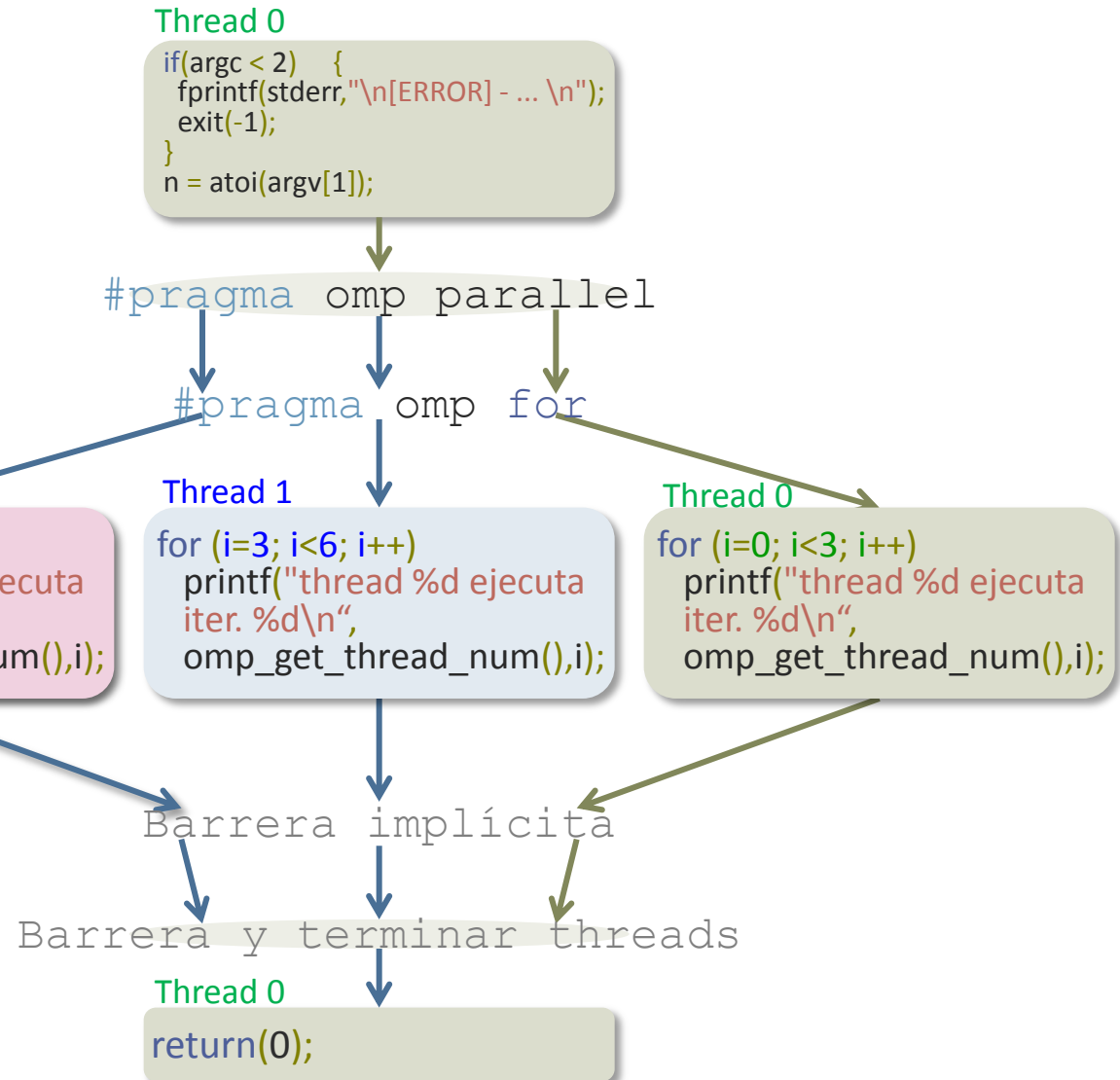
```
mancia_$gcc -O2 -fopenmp -o bucle-for bucle-for.c
mancia_$export OMP_DYNAMIC=FALSE
mancia_$export OMP_NUM_THREADS=8
mancia_$bucle-for 8
Hebra 7 ejecuta la iteración 7 del bucle
Hebra 1 ejecuta la iteración 1 del bucle
Hebra 2 ejecuta la iteración 2 del bucle
Hebra 3 ejecuta la iteración 3 del bucle
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_$
```

- Compilación con gcc
- Fija variables de entorno con `export`
- Ejecuciones con parámetro de entrada igual a 8 (nº de iteraciones del bucle)

Directiva `for`. Reparto de trabajo

secuencial

Región
paralela



secuencial

Directiva sections

C/C++

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

- Tipos de paralelismo (Lección 1/Tema1):
 - paralelismo de tareas o paralelismo a nivel de función
- Tipos de estructuras de procesos/tareas (Lección 4/Tema 2):
 - Explícita: Mater/slave, cliente/servidor, flujo de datos, descomposición de dominio, divide y vencerás
- Sincronización (Lección 4/Tema 2, Lección 10/Tema 3):
 - Barrera implícita al final, no al principio
- Asignación de tareas a threads concretos (Lección 5/Tema 2):
 - No la hace el programador, lo hace la herramienta

Directiva sections

sections.c [Chapman 2008]



- Un thread ejecuta funcA y otro funcB
- El nº de thread que ejecutan trabajo dentro de un sections coincide con el nº de section

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

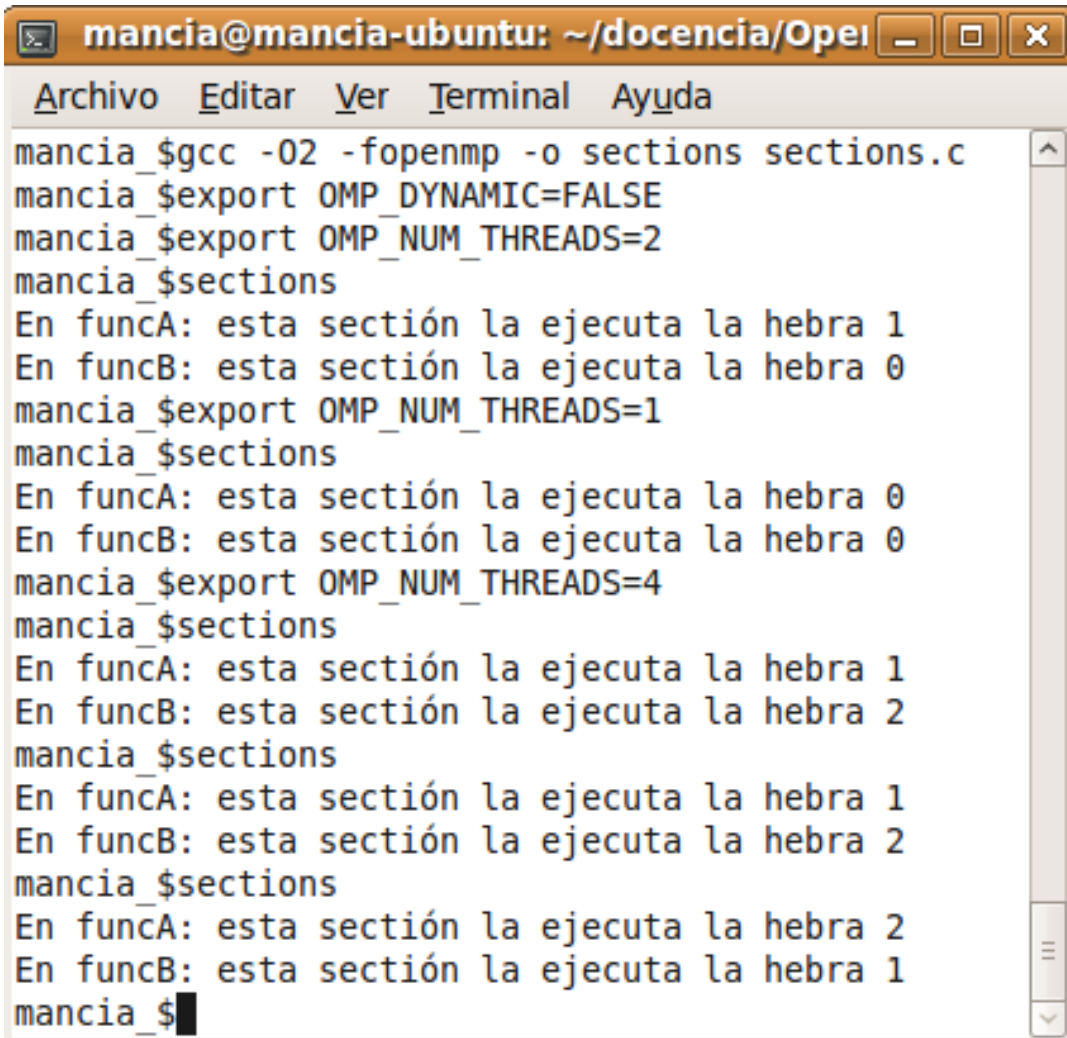
void funcA() {
    printf("En funcA: esta sección la ejecuta el thread
%d\n",
        omp_get_thread_num());
}

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();
        }
    }
}
```

Directiva `sections`. Salida



```
mancia@mancia-ubuntu: ~/docencia/OpenMP
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 single [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)

Directiva single



- Inicializa un vector a un valor solicitado al usuario

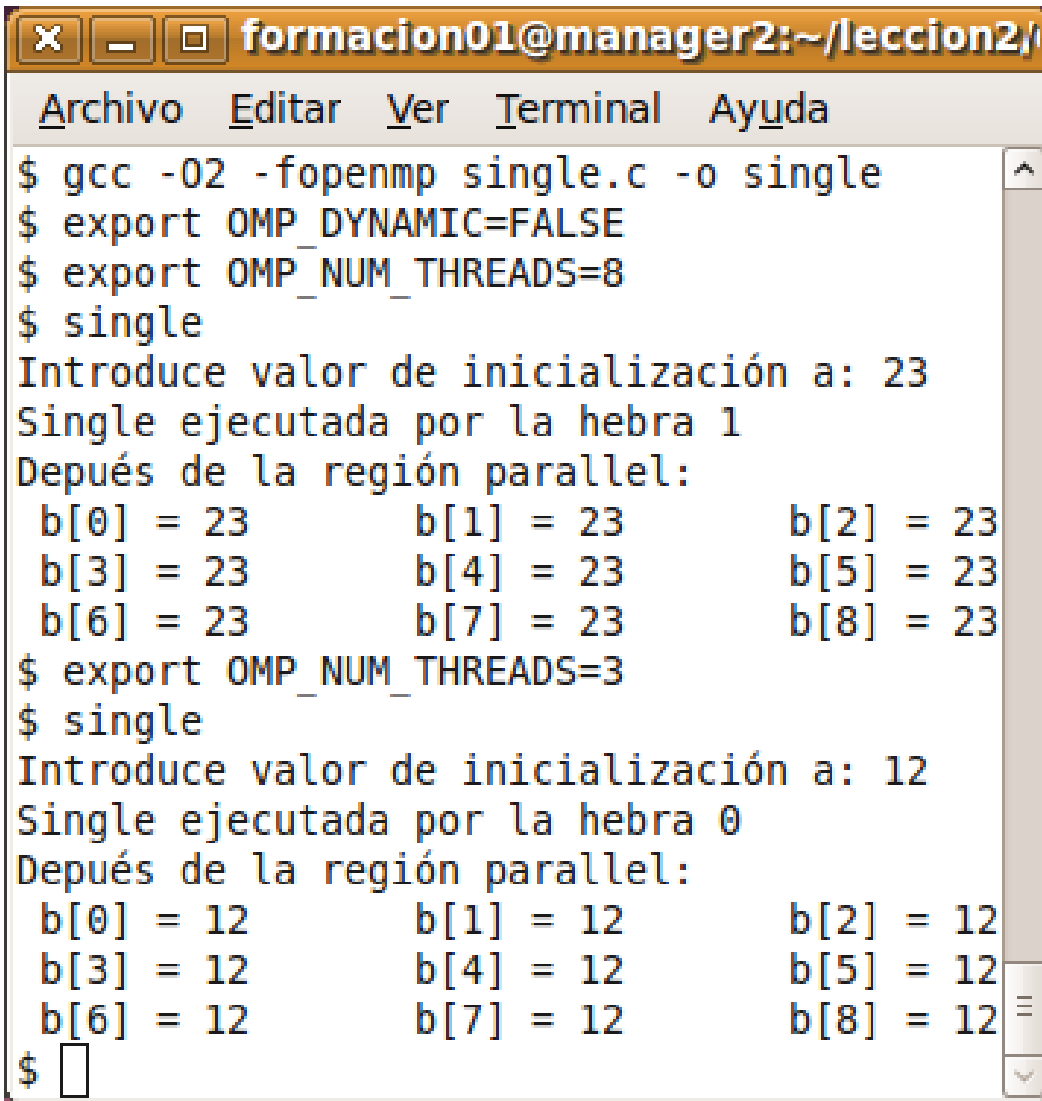
```
#include <stdio.h>
#include <omp.h>

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\t", i, b[i]);
    printf("\n");
}
```

Directiva single. Salida



The screenshot shows a terminal window titled 'formacion01@manager2:~/leccion2/'. The terminal contains the following commands and output:

```
$ 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

Contenidos

- Componentes de OpenMP
- Directivas
- Directiva `parallel`
- Directivas para trabajo compartido (*worksharing*)
- Combinar `parallel` con directivas de trabajo compartido
- Directivas básicas para comunicación y sincronización
- Directiva `master`

Directivas `parallel` y de trabajo compartido combinadas

C/C++ versión completa

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

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

C/C++ versión combinada

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

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

- Cláusulas (Seminario 2):
 - La directiva combinada admite las cláusulas de las dos directivas, excepto `nowait`
- Diferencias con la alternativa que no combina:
 - Legibilidad, prestaciones

Contenido

- Componentes de OpenMP
- Directivas
- Directiva `parallel`
- Directivas para trabajo compartido (*worksharing*)
- Combinar `parallel` con directivas de trabajo compartido
- Directivas básicas para comunicación y sincronización
- Directiva `master`

Directivas/construcciones

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

- En Tema 3/Lección 10 se estudia como se pueden implementar estas directivas para comunicación/sincronización a bajo nivel; es decir, como las puede implementar el programador de OpenMP

Comunicación colectiva en multiprocesadores

Secuencial	Paralela (sum=0)
<pre>for (i=0 ; i<n ; i++) { sum = sum + a[i]; }</pre>	<pre>for (<i>i=ithread</i> ; <i>i</i><n ; <i>i=i+nthread</i>) { <i>sump</i> = <i>sump</i> + a[<i>i</i>]; } sum = sum + <i>sump</i>; /* SC, sum compart. */ if (<i>ithread</i>==0) printf(sum);</pre>

- Ejemplo de comunicación colectiva: suma de n números:
 - La lectura-modificación-escritura (R-W) de `sum` se debería hacer en exclusión mutua (secuencial) => **directivas critical, atomic**
 - **Sección crítica:** Secuencia de instrucciones con una o varias direcciones compartidas (variables) que se deben acceder en exclusión mutua
 - El proceso 0 no debería imprimir hasta que no hayan acumulado `sump` en `sum` todos los procesos => **directiva barrier**

Comunicación colectiva en multiprocesadores (carrera)

Sistema de memoria



$R_0(\text{suma})$	1.0	2.0	$R_2(\text{suma})$	3.1	1.0
$W_0(\text{suma})$	2.1	3.1	$W_2(\text{suma})$	4.4	4.3
$R_1(\text{suma})$	5.4	7.7	$R_3(\text{suma})$	7.6	5.3
$W_1(\text{suma})$	6.6	8.9	$W_3(\text{suma})$	8.10	6.7

Thread 0

Thread 1

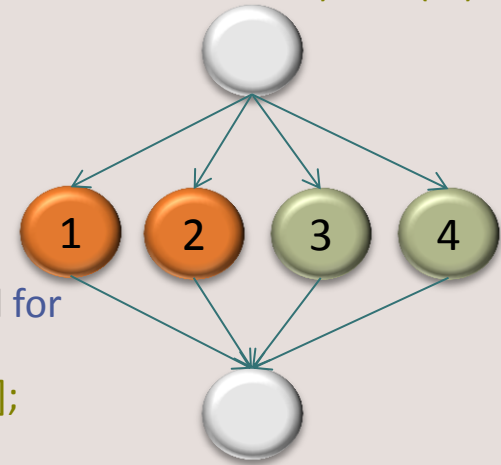
- Ej. para $n=4$, el compilador no optimiza
- $a=\{1,2,3,4\}$
- $R_i(\text{suma})$: Lectura de suma en la iteración i

sin exclusión mutua en el acceso a suma

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n=20, a[n], suma=0;
    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+1;

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

    printf("Fuera de 'parallel' suma=%d\n", suma);
    return(0);
}
```



Directiva barrier

barrier.c [Chapman 2008]

AC  ATC

C/C++

```
#pragma omp barrier
```

- Barrera: punto en el código en el que los threads se esperan entre sí.
- Al final de `parallel` y de las construcciones de trabajo compartido hay una barrera implícita

```
#include <stdlib.h>
#include <time.h>
#ifdef _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");

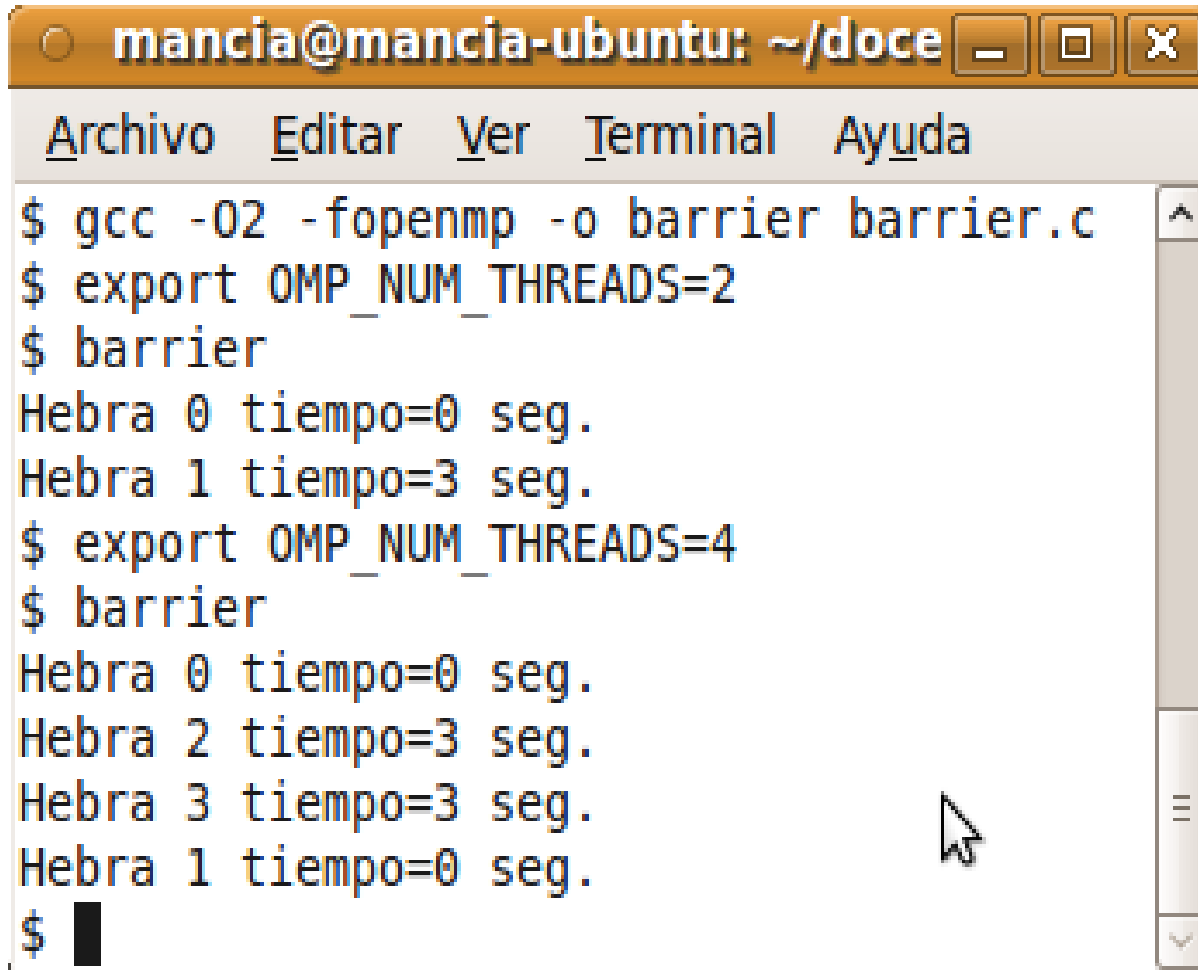
        t1= time(NULL);

        #pragma omp barrier

        t2= time(NULL)-t1;

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

Directiva `barrier`. Salida

A terminal window titled 'mancia@mancia-ubuntu: ~/doce' with standard window controls. The menu bar includes 'Archivo', 'Editar', 'Ver', 'Terminal', and 'Ayuda'. The terminal shows two runs of the 'barrier' program. The first run sets 'OMP_NUM_THREADS=2' and shows threads 0 and 1, with thread 1 waiting 3 seconds. The second run sets 'OMP_NUM_THREADS=4' and shows threads 0, 2, 3, and 1, with threads 2 and 3 waiting 3 seconds. A mouse cursor is visible over the terminal output.

```
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

AC  ATC

C/C++

```
#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

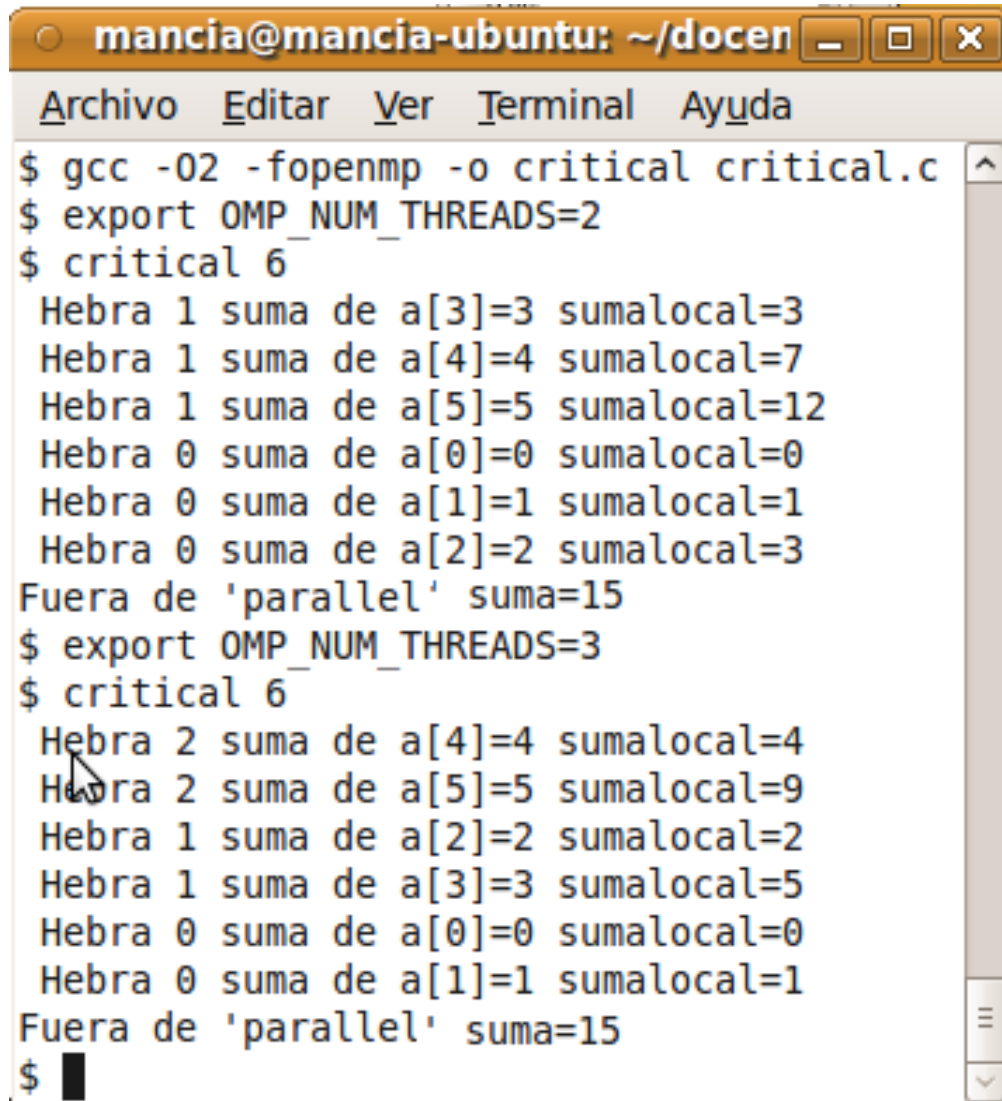
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

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);
}
```

Directiva `critical`. Salida



```
mancia@mancia-ubuntu: ~/docen
Archivo  Editar  Ver  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



C/C++ (v3.0):

`+, *, -, /, &, ^, |, <<, >>`

```
#pragma omp atomic
    x <binop> = expre.
```

...

```
#pragma omp atomic
    x ++, ++x, x-- o -- x.
```

- Puede ser una alternativa a `critical` más eficiente.

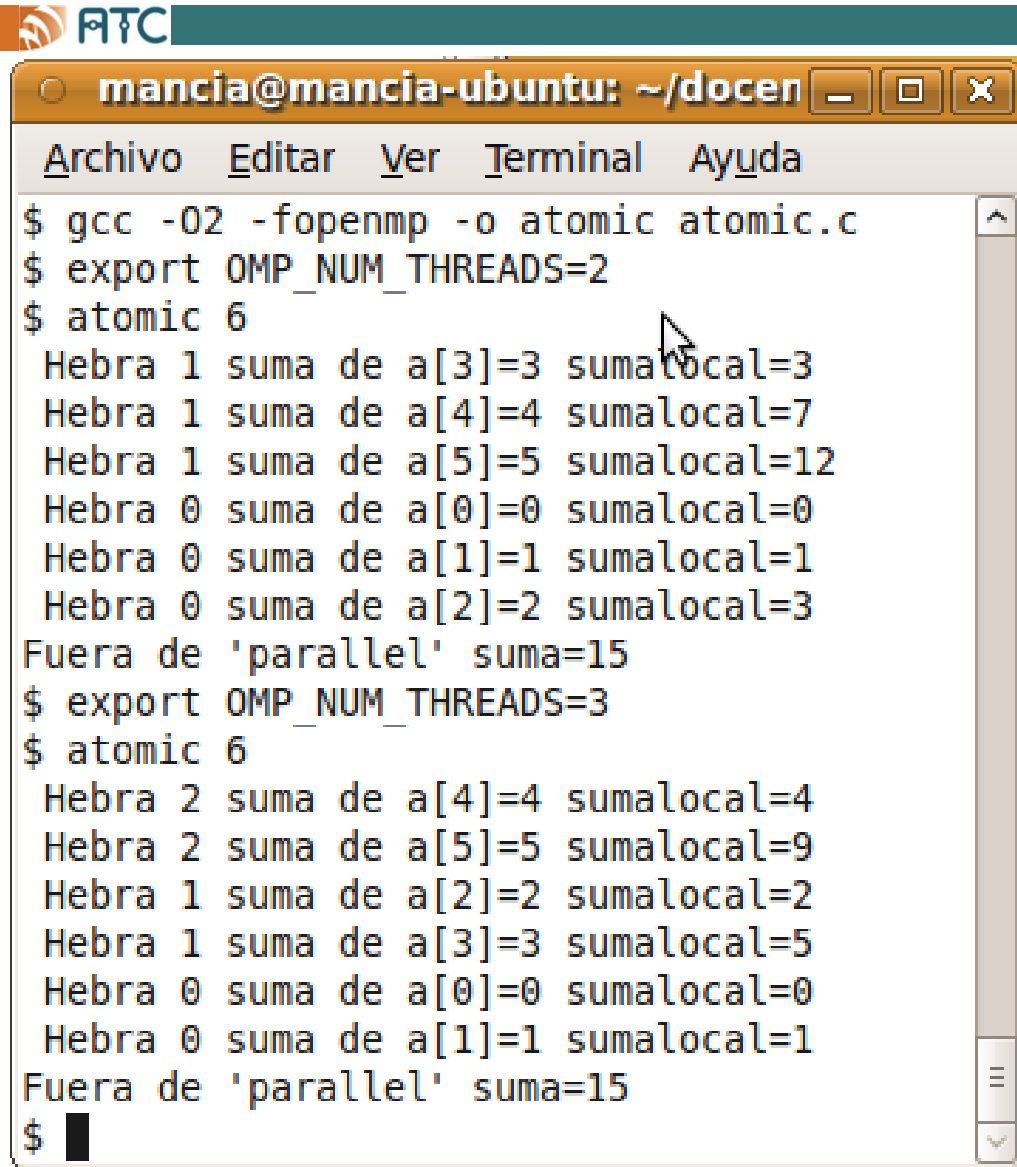
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

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);
}
```

Directiva `atomic`. Salida



A terminal window titled 'mancia@mancia-ubuntu: ~/docen' with a menu bar (Archivo, Editar, Ver, Terminal, Ayuda). The terminal shows the compilation and execution of a program using the `atomic` directive. The first run uses 2 threads, and the second run uses 3 threads. The output shows the state of a shared array `a` and a local sum for each thread, followed by the final sum after the parallel region.

```
$ 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

Contenido

- Componentes de OpenMP
- Directivas
- Directiva `parallel`
- Directivas para trabajo compartido (*worksharing*)
- Combinar `parallel` con directivas de trabajo compartido
- Directivas básicas para comunicación y sincronización
- Directiva `master`

Directivas/construcciones

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

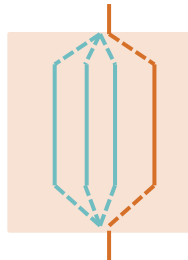
Directiva master



C/C++

```
#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);
    }
}
```

Directiva master. Ejemplo master.c con la barrera eliminada. Salida

```
mancia@mancia-ubuntu: ~/docencia/O
Archivo  Editar  Ver  Terminal  Ayuda
$ 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 es siempre correcta