

# bp2.pdf



patrivc



Arquitectura de Computadores



2º Grado en Ingeniería Informática



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

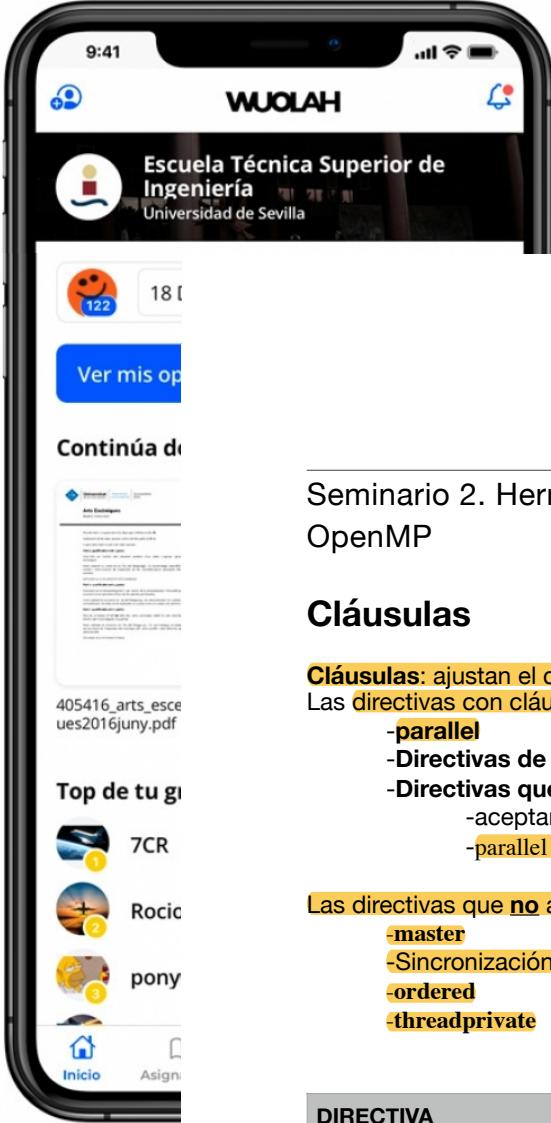


**Descarga la APP de Wuolah.**  
Ya disponible para el móvil y la tablet.





**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

Continúa d...

## Seminario 2. Herramientas de programación paralela II: Cláusulas OpenMP

### Cláusulas

**Cláusulas:** ajustan el comportamiento de las directivas.

Las directivas con cláusulas son:

-parallel

-Directivas de trabajo compartido: DO/for, sections, single y workshare

-Directivas que combinan parallel y directivas de trabajo compartido:

-aceptan cláusulas de las dos directivas que combinan excepto nowait

-parallel DO/for, parallel sections, parallel workshare

NO LA USAREMOS

Las directivas que no aceptan cláusulas son:

-master

-Sincronización / consistencia: critical, barrier, atomic, flush

-ordered

-threadprivate

DIRECTIVA	Ejecutable	Declarativa	
Con bloque estructurado	<i>parallel</i> <i>sections</i> , <i>worksharing</i> , <i>single</i> master critical ordered		Con sentencias
Bucle	<i>DO/for</i>		Con sentencias
Simple (una sentencia)	atomic		Con sentencias
Autónoma (sin código asociado)	barrier, flush	threadprivate	Sin sentencias

\*Las directivas que aceptan cláusulas aparecen en cursiva

### Cláusulas relacionadas con compartición de datos

TIPO	Cláusula	Directivas					
		parallel	DO/for	sections	single	parallel DO/for	parallel sections
Control nº threads	if (1)	X				X	X
	num_threads (1)	X				X	X
Control ámbito de las variables	shared	X				X	X
	private	X	X	X	X	X	X
	lastprivate		X	X		X	X
	firstprivate	X	X	X	X	X	X
	default (1)	X				X	X
	reduction	X	X	X		X	X
Copia de valores	copyin	X				X	X
	copyprivate				X		
Planifica. iteraciones bucle	schedule (1)		X			X	
	ordered (1)		X			X	
No espera	nowait		X	X	X		

Las cláusulas en color son las que vamos a estudiar este seminario.

## Ámbito de los datos por defecto. Cláusulas relacionadas con la compartición de datos

**Regla general para regiones paralelas** (para variables que no se usan en cláusulas o directivas de ámbito): Las variables declaradas fuera de una región y las dinámicas son compartidas por las threads de la región. Las variables declaradas dentro son privadas.

**Excepciones:** índice de bucles for: ámbito predeterminado de privado y variables declaradas static

TIPO	Cláusula	Directivas					
		parallel	DO/for	sections	single	parallel DO/for	parallel sections
Control nº threads	If (1)	X				X	X
	num_threads (1)	X				X	X
Control ámbito de las variables	shared	X				X	X
	private	X	X	X	X	X	X
	lastprivate		X	X		X	X
	firstprivate	X	X	X	X	X	X
	default (1)	X				X	X
	reduction	X	X	X		X	X
Copia de valores	copyin	X				X	X
	copyprivate				X		
Planifica. iteraciones bucle	schedule (1)		X			X	
	ordered (1)		X			X	
No espera	nowait		X	X	X		

```
#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;

#pragma omp parallel
{ int 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);
}
```

## Cláusula shared

**Sintaxis:** shared (list)

Se comparten las variables de list por todas los threads. (List es la vale que se vale a compartir)

Hay que tener precaución cuando al menos un thread lee lo que otro escribe en alguna variable de la lista.

Todas las tareas ven su contenido, pueden cambiarlos y ven los cambios realizados.

TIPO	Cláusula	Directivas					
		parallel	DO/for	sections	single	parallel DO/for	parallel sections
Control nº threads	If (1)	X				X	X
	num_threads (1)	X				X	X
Control ámbito de las variables	shared	X				X	X
	private	X	X	X	X	X	X
	lastprivate		X	X		X	X
	firstprivate	X	X	X	X	X	X
	default (1)	X				X	X
	reduction	X	X	X		X	X
Copia de valores	copyin	X				X	X
	copyprivate				X		
Planifica. iteraciones bucle	schedule (1)		X			X	
	ordered (1)		X			X	
No espera	nowait		X	X	X		

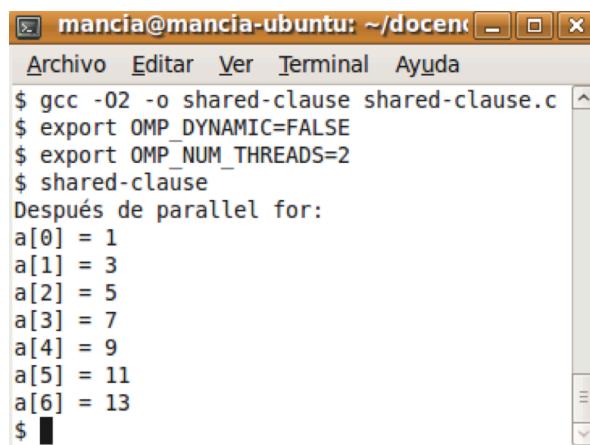
```
#include <stdio.h>
#ifndef _OPENMP
#include <omp.h>
#endif

main()
{
    int i, n = 7;
    int a[n];

    for (i=0; i<n; i++)
        a[i] = i+1;

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

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



```
mancia@mancia-ubuntu: ~/docenc
Archivo Editar Ver Terminal Ayuda
$ gcc -O2 -o shared-clause shared-clause.c
$ export OMP_DYNAMIC=FALSE
$ export OMP_NUM_THREADS=2
$ shared-clause
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
$
```

En la salida podemos observar que lo compila de forma secuencial al no usar -fopenmp.

## Cláusula private

Especifica que cada subprocesso debe tener su propia instancia de una variable.

**Sintaxis:** private(list)

Hay que tener **precaución** cuando el valor de entrada y de salida está indefinido aunque la variable esté declarada fuera de la construcción.

Su valor inicial no esta definido y tras la región paralela tampoco esta definido.

TIPO	Cláusula	Directivas					
		parallel	DO/for	sections	single	parallel DO/for	parallel sections
Control nº threads	If (1)	X				X	X
	num_threads (1)	X				X	X
	shared	X				X	X
Control ámbito de las variables	private	X	X	X	X	X	X
	lastprivate		X	X		X	X
	firstprivate	X	X	X	X	X	X
	default (1)	X				X	X
	reduction	X	X	X		X	X
Copia de valores	copyin	X				X	X
	copyprivate				X		
Planifica. iteraciones bucle	schedule (1)		X			X	
	ordered (1)		X			X	
No espera	nowait		X	X	X		

Esta **predeterminado**: los índices de un bucle tienen un ámbito predeterminado de privado si se usa para ese bucle la directiva for

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

main()
{
    int i, n = 7;
    int a[n], suma;
    for (i=0; i<n; i++)
        a[i] = i;
#pragma omp parallel private(suma)
{
    suma=0;
#pragma omp for
    for (i=0; i<n; i++)
    {
        suma = suma + a[i];
        printf(
            "thread %d suma a[%d] / ", omp_get_thread_num(), i);
    }
    printf(
        "\n* thread %d suma= %d", omp_get_thread_num(), suma);
}
printf("\n");
}
```

```
mancia@mancia-ubuntu: ~/docencia/OpenMP/ - □ □ □
Archivo Editar Ver Terminal Ayuda
$ gcc -O2 -fopenmp -o private-clause private-clause.c
$ export OMP_DYNAMIC=FALSE
$ export OMP_NUM_THREADS=4
$ private-clause
Hebra 3 suma a[6] / Hebra 1 suma a[2] / Hebra 1 suma a[3] / Hebra 2 suma a[4] / Hebra 2 suma a[5] / Hebra 2 suma a[5] / Hebra 0 suma a[0] / Hebra 0 suma a[1] /
* Hebra 2 suma= 9
* Hebra 1 suma= 5
* Hebra 3 suma= 6
* Hebra 0 suma= 1
$ export OMP_NUM_THREADS=3
$ private-clause
Hebra 2 suma a[6] / Hebra 1 suma a[3] / Hebra 1 suma a[4] / Hebra 1 suma a[5] / Hebra 0 suma a[0] / Hebra 0 suma a[1] / Hebra 0 suma a[2] /
* Hebra 1 suma= 12
* Hebra 2 suma= 6
* Hebra 0 suma= 3
$ private-clause
Hebra 2 suma a[6] / Hebra 1 suma a[3] / Hebra 1 suma a[4] / Hebra 1 suma a[5] / Hebra 0 suma a[0] / Hebra 0 suma a[1] / Hebra 0 suma a[2] /
* Hebra 1 suma= 12
* Hebra 2 suma= 6
* Hebra 0 suma= 3
```

## Cláusula lastprivate

Especifica que la versión del contexto envolvente de la variable se establece igual que la versión privada de cualquier subprocesso que ejecute la iteración final (construcción for-Loop) o la última sección (#pragma secciones).

**Sintaxis:** lastprivate(list)

Esta cláusula **combina**: la acción de private (con) la copia (al salir de región paralela) del último valor (en una ejecución secuencial) de las variables de la lista:

**-Construcción bucle:** el valor en la última iteración

**-Construcción sections:** el valor que tenga tras la última sección

TIPO	Cláusula	Directivas					
		parallel	DO/for	sections	single	parallel DO/for	parallel sections
Control nº threads	If (1)	X				X	X
	num_threads (1)	X				X	X
	shared	X				X	X
Control ámbito de las variables	private	X	X	X	X	X	X
	lastprivate		X	X		X	X
	firstprivate	X	X	X	X	X	X
	default (1)	X				X	X
	reduction	X	X	X		X	X
Copia de valores	copyin	X				X	X
	copyprivate				X		
Planifica. iteraciones bucle	schedule (1)		X			X	
	ordered (1)		X			X	
No espera	nowait		X	X	X		

# 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



Top de tu gr



7CR



Rocic



pony



Inicio



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

main() {
    int i, n = 7;
    int a[n], v;

    for (i=0; i<n; i++) a[i] = i+1;

#pragma omp parallel for lastprivate(v)
    for (i=0; i<n; i++) {
        v = a[i];
        printf("thread %d v=%d\n",
            omp_get_thread_num(), v);
    }

    printf("\nFuera de la construcción 'parallel for' v=%d\n",
        v);
}

```

## Cláusula firstprivate

Se usa para crear una variable privada que tiene su valor inicial igual al valor de la variable controlada por el hilo maestro cuando se entra el bucle. Si un hilo cambia el valor de una variable en alguna iteración, entonces este valor será el valor de la variable en iteraciones subsecuentes.

Las otras tareas no pueden ver su contenido. Su valor es inicializado con la variable original.

**Peculiaridad:** tras la región paralela su valor no está definido

**Sintaxis:** firstprivate(list)

Esta cláusula **combina**: la acción de private (con) la inicialización de las variables de la lista (al entrar en región paralela) (difusión)

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

main() {
    int i, n = 7;
    int a[n], suma=0;

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

#pragma omp parallel for firstprivate(suma)
    for (i=0; i<n; i++) {
        suma = suma + a[i];
        printf(" thread %d suma a[%d] suma=%d\n",
            omp_get_thread_num(), i, suma);
    }

    printf("\nFuera de la construcción parallel suma=%d\n",
        suma);
}

```

```
mancia@mancia-ubuntu: ~/docencia/OpenMP/lección ->
$ gcc -O2 -fopenmp -o lastprivate-clause lastprivate-clause.c
$ export OMP_DYNAMIC=FALSE
$ export OMP_NUM_THREADS=3
$ lastprivate-clause
Hebra 2 v=7 / Hebra 1 v=4 / Hebra 1 v=5 / Hebra 1 v=6 / Hebra 0 v=1 / Hebra 0 v=2 / Hebra 0 v=3 /
Fuera de la construcción 'parallel for' v=7
$ lastprivate-clause
Hebra 0 v=1 / Hebra 0 v=2 / Hebra 0 v=3 / Hebra 1 v=4 / Hebra 1 v=5 / Hebra 1 v=6 / Hebra 2 v=7 /
Fuera de la construcción 'parallel for' v=7
$ lastprivate-clause
Hebra 0 v=1 / Hebra 0 v=2 / Hebra 0 v=3 / Hebra 1 v=4 / Hebra 1 v=5 / Hebra 1 v=6 / Hebra 2 v=7 /
Fuera de la construcción 'parallel for' v=7
$
```

TIPO	Cláusula	Directivas					
		parallel	DO/for	sections	single	parallel DO/for	parallel sections
Control nº threads	if (1) num_threads (1)	X				X	X
Control ámbito de las variables	shared private lastprivate firstprivate default (1) reduction	X X X X X X	X X X X X X	X X X X X X	X X X X X X	X X X X X X	X X X X X X
Copia de valores	copyin copyprivate					X	X
Planifica. iteraciones bucle	schedule (1) ordered (1)			X		X	
No espera	nowait			X	X	X	

```
mancia@mancia-ubuntu: ~/docencia/OpenMP/lección2/cláusulas ->
$ gcc -O2 -fopenmp -o firstlastprivate-clause firstlastprivate-clause.c
$ firstlastprivate-clause
Hebra 1 suma a[3] suma=3
Hebra 1 suma a[4] suma=7
Hebra 1 suma a[5] suma=12
Hebra 0 suma a[0] suma=0
Hebra 0 suma a[1] suma=1
Hebra 0 suma a[2] suma=3
Hebra 2 suma a[6] suma=6
Fuera de la construcción parallel suma=0
$ firstlastprivate-clause
Hebra 2 suma a[6] suma=6
Hebra 1 suma a[3] suma=3
Hebra 1 suma a[4] suma=7
Hebra 1 suma a[5] suma=12
Hebra 0 suma a[0] suma=0
Hebra 0 suma a[1] suma=1
Hebra 0 suma a[2] suma=3
Fuera de la construcción parallel suma=0
$
```

El segundo código ejecutado usa también lastprivate(suma)

## Cláusula default

Especifica el comportamiento de las variables sin ámbito en una región paralela.

default (none | shared)

Con **none** el programador debe especificar el alcance de todas las variables usadas en la construcción. **Excepción:** variables threadprivate y los índices en Directivas con for

Se pueden excluir variables del ámbito especificado en **default** con: shared, private, firstprivate, lastprivate, reduction

**Restricción:** sólo puede haber una cláusula default (solo puede aparecer una vez).

Admite tres opciones: private, shared y none. Con **none** todas las variables deben ser definidas.

TIPO	Cláusula	Directivas					
		parallel	DO/for	sections	single	parallel DO/for	parallel sections
Control nº threads	If (1)	X				X	X
	num_threads (1)	X				X	X
	shared	X				X	X
	private	X	X	X	X	X	X
	lastprivate			X	X	X	X
	firstprivate	X	X	X	X	X	X
Control ámbito de las variables	default (1)	X				X	X
	reduction	X	X	X	X	X	X
	copyin	X				X	X
	copyprivate			X			
	schedule (1)			X			X
	ordered (1)			X			X
No espera	nowait			X	X	X	

## Cláusulas relacionadas con comunicación/sincronización

TIPO	Cláusula	Directivas					
		parallel	DO/for	sections	single	parallel DO/for	parallel sections
Control nº threads	If (1)	X				X	X
	num_threads (1)	X				X	X
	shared	X				X	X
	private	X	X	X	X	X	X
	lastprivate		X	X		X	X
	firstprivate	X	X	X	X	X	X
Control ámbito de las variables	default (1)	X				X	X
	reduction	X	X	X		X	X
	copyin	X				X	X
	copyprivate			X			
	schedule (1)		X			X	
	ordered (1)		X			X	
No espera	nowait		X	X	X		

## Cláusula reduction

**Sintaxis:** reduction (operator: list)

Operator -> Operador de la operación que se va a realizar en las variables *list* al final de la región paralela.

List -> Una o más variables en las que se va a realizar la reducción escalar. Si se especifica más de una variable, separe los nombres de las variables con una coma.  
Se obtiene por la aplicación del operador.

Especifica qué una o varias variables que son privadas para cada subproceso son el sujeto de una operación de reducción al final de la región paralela.

Comunicación colectiva todos a uno.

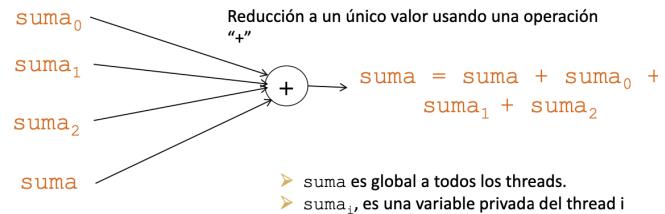
### Operadores de reducción:

C/C++	
Tipo	Valor inicial variables locales
+	0
-	0
*	1
&	~0 (bits a 1)
	0
^	0
&&	1
	0

La cláusula reduction mantiene el contenido de la variable antes de realizar la operación

Reducción a un único valor usando una operación

suma es global a todos los threads.  
sumai, es una variable privada del thread i



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

main(int argc, char **argv) {
    int i, n=20, a[n], suma=0;

    if(argc < 2) {
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d",n);}

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

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

    printf("Tras 'parallel' suma=%d\n",suma);
}
```

```
mancia@mancia-ubuntu: ~/docencia/OpenMP/lec
Archivo Editar Ver Terminal Ayuda
$ gcc -O2 -fopenmp -o reduction-clause reduction-clause.c
$ export OMP_NUM_THREADS=3
$ reduction-clause 10
Tras 'parallel' suma=45
$ reduction-clause 20
Tras 'parallel' suma=190
$ reduction-clause 6
Tras 'parallel' suma=15
$
```

## Cláusula copyprivate y directiva single

Especifica que una o más variables se deben compartir entre todos los subprocessos.

**Sintaxis:** copyprivate(list)

List es la variable que se va a compartir.

**Sólo se puede usar con single.**

Permite que una variable privada de un thread se copie a las variables privadas del mismo nombre del resto de threads (difusión).

Útil para lectura (entrada) de variables.

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

main() {
    int n = 9, i, b[n];
    for (i=0; i<n; i++) b[i] = -1;

    #pragma omp parallel
    { int a;
    #pragma omp single copyprivate(a)
    {
        printf("\nIntroduce valor de inicialización a: ");
        scanf("%d", &a);
        printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
    }
    #pragma omp for
    for (i=0; i<n; i++) b[i] = a;
    }

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

Algunos ejemplos:

**¿Qué ocurre si inicializamos la variable suma fuera y dentro de parallel?**

Si en private-clause.c inicializamos suma dentro de la región **parallel**, vemos que siempre se obtiene como resultado final 0. Podemos ver que aún así la suma se realiza correctamente.

Cuando se inicializa la variable suma fuera de la región **parallel**, lo que sucede es que se muestra el valor al que hemos inicializado dicha variable.



# 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...

Esto se produce porque la cláusula **private(suma)** crea la variable pero no la inicializa a ningún valor, es indefinido, de ahí a que se produzcan estos fenómenos.

### ¿Función de la cláusula **private(suma)**?

**private(suma)** pone la variable **suma** en privado, si la quitásemos, **suma** dejaría de ser privada, por lo que cada hebra modifica la **suma total** en vez de la copia de la variable que tenían reservada antes.

### ¿Para qué sirve **nowait**?

Invalida la barrera implícita en una directiva. **Nowait** se aplica con las directivas: **for**, **sección**, **single**

### ¿Para qué sirve **threadprivate**?

Especifica que una variable es privada en un subproceso.

### ¿Para qué sirve **none**?

**none** significa que cualquier variable que se use en una región paralela que no tenga el ámbito con la cláusula **Private**, **Shared**, **Reduction**, **firstprivate** o **lastprivate** producirá un error del compilador.

### ¿Con qué directivas se puede aplicar **copyprivate**?

Sólo con **single**

### ¿Con qué directivas se puede aplicar **shared**?

Con **parallel**, **for**, **section**

### ¿Con qué directivas se puede aplicar **private**?

Con **for**, **parallel**, **section**, **single**

### ¿Con qué directivas se puede aplicar **lastprivate**?

Con **for** y **section**

### ¿Con qué directivas se puede aplicar **firstprivate**?

Con **for**, **parallel**, **section**, **single**

### ¿Con qué directivas se puede aplicar **default**?

Parallel, **for**, **section**

### ¿Con qué directivas se puede aplicar **reduction**?

Con **parallel**, **for**, **section**

### ¿Qué imprime la ejecución del código **firstlastprivate**?

Lo que obtenemos por pantalla es el resultado de la suma de la hebra que ejecuta la última iteración del bucle **for** (si tenemos un vector de tamaño 10, obtendremos por pantalla un 9, si tenemos un vector de tamaño 7, obtendremos por pantalla un 6, etc)

### ***¿Qué se observa en los resultados de ejecución de copyprivate-clause.c cuando se elimina la cláusula copyprivate(a) en la directiva single?***

El copyprivate hace que el valor de la variable se comparta con las demás hebras. Si eliminamos la cláusula el valor que lee la hebra que ejecuta el single no se comparte a las demás hebras por lo que solo se inicializarán con el mismo valor que aquellos valores que se encargue de inicializar la hebra que ha leído el valor de a en el single. Lo que sucede es que la hebra que ha ejecutado la directiva single ha inicializado las componentes del vector que se le han asignado, mientras el resto se mantienen a 0.

### ***¿Qué ocurre si en el ejemplo del seminario shared-clause.c se añade a la directiva parallel la cláusula default(None)?***

Al poner default(None) estamos especificando que las variables que estén fuera del parallel no se compartan, dando un error de compilación. Para solucionarlo hay que decirle que las variables que se usan en parallel que se inicializan fuera del parallel se compartan.

\*Tanto en **PC-LOCAL** como en **ATCGRID**, a medida que aumentamos el número de hebras, se produce una reducción del tiempo de ejecución considerable (especialmente si lo comparamos con una ejecución secuencial).