

BP3.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.



¡Buscamos las mejores fotos de tu Uni!

Envíanos la mejor foto de tu facultad y gana **un regalo seguro.**

 [Quiero participar](#)

 Sólo por participar, ¡llévate  10 Wuolah Coins!



1 año de Wuolah Pro

Descarga y visualiza todos tus apuntes sin anuncios durante un año.



Pack de 10 Wuolah Coins

Úsalas para quitar la publi de tus apuntes y participar en los sorteos.



WUOLAH

Seminario 3. Herramientas de programación paralela III: Interacción con el entorno en OpenMP y evaluación de prestaciones

Variables de control internas que afecta a una región parallel

Variable de control	Ámbito	Valor (valor inicial)	¿Qué controla?	Consultar/Modificar
dyn-var	entorno de datos	true/false (depende de la implementación)	Ajuste dinámico del numero de threads	sí(f) /sí(ve,f)
nthreads-var	entorno de datos	número (depende de la implementación)	threads en la siguiente ejecución paralela	sí(f) /sí(ve,f)
thread-limit-var	entorno de datos	número (depende de la implementación)	Máximo no de threads para todo el programa	sí(f) /sí(ve,-)
nest-var		true/false (false)	Paralelismo anidado	sí(f) /sí(ve,-)
run-sched-var	entorno de datos	(kind[,chunk]) (depende de la implementación)	Planificación de bucles para runtime	sí(f) /sí(ve,f)
def-sched-var	Dispositivo	(kind[,chunk]) (depende de la implementación)	Planificación de bucles por defecto. Ámbito el programa.	no /no

Variables de entorno

Variable de control	Variable de entorno	Ejemplos de modificación (shell bash/ksh)
dyn-var	OMP_DYNAMIC	export OMP_DYNAMIC=FALSE export OMP_DYNAMIC=TRUE
nthreads-var	OMP_NUM_THREADS	export OMP_NUM_THREADS=8
thread-limit-var	OMP_THREAD_LIMIT	export OMP_THREAD_LIMIT=8
nest-var	OMP_NESTED	export OMP_NESTED=TRUE export OMP_NESTED=FALSE

Variable de control	Variable de entorno	Ejemplos de modificación (shell bash/ksh)
run-sched-var	OMP_SCHEDULE	export OMP_SCHEDULE="static,4" export OMP_SCHEDULE="nonmonotonic:static,4" export OMP_SCHEDULE="dynamic" export OMP_SCHEDULE="monotonic:dynamic,4"
def-sched-var		

Funciones del entorno de ejecución

Variable de control	Rutina para consultar	Rutina para modificar
dyn-var	omp_get_dynamic()	omp_set_dynamic()
nthreads-var	omp_get_max_threads()	omp_set_num_threads()
thread-limit-var	omp_get_thread_limit()	
nest-var	omp_get_nested()	omp_set_nested()
run-sched-var	omp_get_schedule(&kind, &modifier)	omp_set_schedule(kind, modifier)
def-sched-var	No	No

omp_get_thread_num() -> Devuelve al thread su identificador dentro del grupo de thread

omp_get_num_threads() -> Obtiene el número de threads que se están usando en una región paralela. Devuelve 1 en código secuencial

omp_get_num_procs() -> Devuelve el número de procesadores disponibles para el programa en

omp_in_parallel() -> Devuelve true si se llama a la rutina dentro de una región parallel activa (puede estar dentro de varios parallel, basta que uno esté activo) y false en caso contrario.

Cláusula que interaccionan con el entorno

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	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		

¿Cuántos threads se usan?

El orden de precedencia para fijar el número de threads de mayor a menor prioridad es el siguiente:

- El número que resulte de evaluar la cláusula if
- El número que fija la cláusula num_threads
- El número que fija la función omp_set_num_threads()
- El contenido de la variable de entorno OMP_NUM_THREADS
- Fijado por defecto por la implementación: normalmente el no de cores de un nodo

```
#pragma omp <directive> [<clause> <clause> ...]
#pragma omp parallel num_threads(8) if(N>20)
```

Cláusula if

Sintaxis: if(scalar-exp) (C/C++)

No hay ejecución paralela si no se cumple la condición

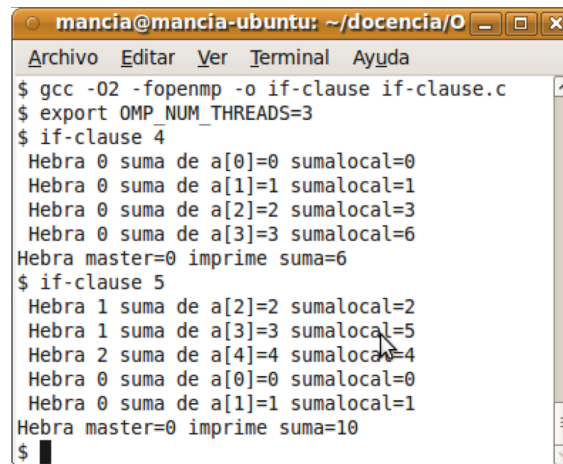
Precaución: usar sólo en construcciones con parallel

En la salida de esta cláusula podemos ver que en la primera ejecución sólo trabaja el thread 0 porque no hay más de 4 iteraciones

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

int main(int argc, char **argv)
{
    int i, n=20, tid;
    int a[n], suma=0, sumalocal;
    if(argc < 2) {
        fprintf(stderr, "[ERROR]-Falta 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 if(n>4) default(none) \
        private(sumalocal,tid) shared(a,suma,n)
    {
        sumalocal=0;
        tid=omp_get_thread_num();
        #pragma omp for private(i) schedule(static) nowait
        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);
    }
}
```



```
mancia@mancia-ubuntu: ~/docencia/O
Archivo Editar Ver Terminal Ayuda
$ gcc -O2 -fopenmp -o if-clause.c
$ export OMP_NUM_THREADS=3
$ if-clause 4
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
Hebra 0 suma de a[3]=3 sumalocal=6
Hebra master=0 imprime suma=6
$ if-clause 5
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 0 suma de a[0]=0 sumalocal=0
Hebra 0 suma de a[1]=1 sumalocal=1
Hebra master=0 imprime suma=10
$
```

Cláusula schedule

Sintaxis: schedule (kind,[chunk])

Kind es la forma de asignación y puede ser del tipo: static, dynamic, guided, auto o runtime

Chunk es la granularidad de la distribución

Las precauciones a tener en cuenta con esta cláusula son: que sólo se usa en bucles, por defecto es tipo static (distribución en tiempo de compilación) en la mayor parte de las implementaciones y es mejor no asumir una granularidad de distribución por defecto.

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

main(int argc, char **argv) {
    int i, n = 7, chunk, a[n], suma=0;

    if(argc < 2) {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);
    for (i=0; i<n; i++) a[i] = i;

    #pragma omp parallel for firstprivate(suma) \
        lastprivate(suma) schedule(static,chunk)
    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("Fuera de 'parallel for' suma=%d\n",suma);
}
```

Cláusula **schedule.static**

Esta cláusula usa: `schedule (static, chunk)`. Las iteraciones se dividen en unidades de chunk iteraciones. Las unidades se asignan en **round-robin**.

La entrada es el número de iteraciones de la unidad de distribución (chunk). Usando `schedule(static)` se asigna un único chunk a cada thread (comportamiento usual por defecto).

Si se elige static con `schedule ->` menos coste / mejor localidad datos

```
mancia@mancia-ubuntu: ~/docencia/OpenMP/lec
Archivo Editar Ver Terminal Ayuda
$ gcc -O2 -fopenmp -o scheduled-clause scheduled-clause.c
$ scheduled-clause 1
Hebra 1 suma a[1] suma=1
Hebra 1 suma a[4] suma=5
Hebra 2 suma a[2] suma=2
Hebra 2 suma a[5] suma=7
Hebra 0 suma a[0] suma=0
Hebra 0 suma a[3] suma=3
Hebra 0 suma a[6] suma=9
Fuera de 'parallel for' suma=9
$ scheduled-clause 2
Hebra 1 suma a[2] suma=2
Hebra 1 suma a[3] suma=5
Hebra 2 suma a[4] suma=4
Hebra 2 suma a[5] suma=9
Hebra 0 suma a[0] suma=0
Hebra 0 suma a[1] suma=1
Hebra 0 suma a[6] suma=7
Fuera de 'parallel for' suma=7
$ scheduled-clause 3
Hebra 2 suma a[6] suma=6
Hebra 0 suma a[0] suma=0
Hebra 0 suma a[1] suma=1
Hebra 0 suma a[2] suma=3
Hebra 1 suma a[3] suma=3
Hebra 1 suma a[4] suma=7
Hebra 1 suma a[5] suma=12
Fuera de 'parallel for' suma=6
```

Cláusula **schedule.dynamic**

El kind que vamos a usar va ser `dynamic`.

Si se elige `dynamic` con `schedule ->` mas coste / carga mas equilibrada

Distribución en tiempo de ejecución. **Apropiado si se desconoce el tiempo de ejecución de las iteraciones**. La unidad de distribución tiene chunk iteraciones. **Número unidades $O(n/\text{chunk})$** .

Precauciones: añade sobrecarga adicional.

Usa: `schedule (dynamic, chunk)`. Las iteraciones se dividen en unidades de chunk iteraciones. Las unidades se asignan en tiempo de ejecución. Los threads más rápidos ejecutan más unidades. Si no se especifica chunk se usan unidades de una iteración.

Cada thread coge "chunk" iteraciones hasta terminar con todas.

Entradas: número de iteraciones y tamaño de la unidad de distribución

```
mancia@mancia-ubuntu: ~/docencia/OpenMP/lecci
Archivo Editar Ver Terminal Ayuda
$ export OMP_NUM_THREADS=2
$ gcc -O2 -fopenmp -o scheduled-clause scheduled-clause.c
$ scheduled-clause 7 3
Hebra 0 suma a[0]=0 suma=0
Hebra 0 suma a[1]=1 suma=1
Hebra 0 suma a[2]=2 suma=3
Hebra 0 suma a[6]=6 suma=9
Hebra 1 suma a[3]=3 suma=3
Hebra 1 suma a[4]=4 suma=7
Hebra 1 suma a[5]=5 suma=12
Fuera de 'parallel for' suma=9
$ scheduled-clause 7 2
Hebra 0 suma a[0]=0 suma=0
Hebra 0 suma a[1]=1 suma=1
Hebra 0 suma a[4]=4 suma=5
Hebra 0 suma a[5]=5 suma=10
Hebra 0 suma a[6]=6 suma=16
Hebra 1 suma a[2]=2 suma=2
Hebra 1 suma a[3]=3 suma=5
Fuera de 'parallel for' suma=16
$
```

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

main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);

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

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

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

Cláusula **schedule.guided**

El kind que vamos a usar va ser `guided`.

Distribución en tiempo de ejecución. Es apropiado si se desconoce el tiempo de ejecución de las iteraciones o su número. **Comienza con un bloque largo. El tamaño del bloque va menguando (número de iteraciones que restan dividido por número de threads), no más pequeño que chunk (excepto la última).**

Precauciones: Sobrecarga extra, pero menos que `dynamic` para el mismo chunk.

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

main(int argc, char **argv) {
    int i, n=20, chunk, a[n], suma=0;
    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones y/o chunk \n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20; chunk = atoi(argv[2]);
    for (i=0; i<n; i++) a[i] = i;

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

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


Cláusula schedule runtime

El kind que vamos a usar va a ser runtime. El tipo de distribución (static, dynamic o guided) se fija en tiempo de ejecución. El tipo de distribución depende de la variable de control run- sched-var.

Se usa: schedule (runtime) y OMP_SCHEDULE para fijar el tipo de distribución (ejemplo: OMP_SCHEDULE="static"). En la entrada se pone el número de iteraciones.

```
mancia@mancia-ubuntu: ~/docencia/OpenMP/lecci
Archivo Editar Ver Terminal Ayuda
$ gcc -O2 -fopenmp -o scheduler-clause scheduler-clause.c
$ export OMP_SCHEDULE="static"
$ scheduler-clause 8
Hebra 0 suma a[0]=0 suma=0
Hebra 0 suma a[2]=2 suma=2
Hebra 0 suma a[4]=4 suma=6
Hebra 0 suma a[6]=6 suma=12
Hebra 1 suma a[1]=1 suma=1
Hebra 1 suma a[3]=3 suma=4
Hebra 1 suma a[5]=5 suma=9
Hebra 1 suma a[7]=7 suma=16
Fuera de 'parallel for' suma=16
$ export OMP_SCHEDULE="static,2"
$ scheduler-clause 8
Hebra 0 suma a[0]=0 suma=0
Hebra 0 suma a[1]=1 suma=1
Hebra 0 suma a[4]=4 suma=5
Hebra 0 suma a[5]=5 suma=10
Hebra 1 suma a[2]=2 suma=2
Hebra 1 suma a[3]=3 suma=5
Hebra 1 suma a[6]=6 suma=11
Hebra 1 suma a[7]=7 suma=18
Fuera de 'parallel for' suma=18
```

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _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, "\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 for firstprivate(suma) \
        lastprivate(suma) schedule(runtime)
    for (i=0; i<n; i++)
    {
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
            omp_get_thread_num(), i, a[i], suma);
    }

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

La planificación y tamaño del bloque vendrán determinados por la variable de entorno OMP_SCHEDULE.

Funciones de la biblioteca OpenMP

Son funciones para acceder al entorno de ejecución de OpenMP.

Las funciones para usar sincronización con cerrojos son: (v2.5) omp_init_lock(), omp_destroy_lock(), omp_set_lock(), omp_unset_lock(), omp_test_lock() y (v3.0) omp_destroy_nest_lock, omp_set_nest_lock, omp_unset_nest_lock, omp_test_nest_lock

Y las funciones para obtener tiempos de ejecución son: omp_get_wtime (), omp_get_wtick()

Información extra

Un **chunk** es un conjunto de datos que se envía a un procesador, o bien, cada una de las secciones en que se descompone el problema para su paralelización. En definitiva es un segmento de información que se ubica dentro de muchos formatos multimedia.

Kind -> tipo que vamos a usar, es la forma de asignación (por ejemplo: static, dynamic, guided...)

Con schedule para saber que planificación usar tenemos que tener en cuenta lo siguiente:

Cláusula schedule	Cuando utilizar
Static	Predecible y trabajo similar por iteración
Dynamic	Impredecible, trabajo altamente variable por iteración

Ejemplo de la cláusula schedule static:

```
#pragma omp parallel for schedule (static, 8)
for(int i= start; i<= end; i+=2){
    if (TestForPrime(i) ) gPrimesFound++;
}
```

Las iteraciones se dividen en bloques de 8. Si Start = 3, el primer bloque es $i = \{3, 5, 7, 9, 11, 13, 15, 17\}$

En auto la elección de la planificación la realiza el compilador (o el runtime system). Es dependiente de la implementación.

Planificación con la cláusula schedule:

```
#pragma omp for schedule(..)
```

schedule (runtime)

Planificación y tamaño de bloque determinado por la variable de entorno OMP_SCHEDULE

Schedule (static)

Divide el total de iteraciones del ciclo entre los procesos paralelos, asignando grupos aproximadamente iguales de índices contiguos a cada proceso.

Schedule (static, chunk)

Divide el total de iteraciones del ciclo en tamaños dados por chunk y los va entregando a cada proceso en forma ordenada, al inicio del ciclo (estáticamente).

Schedule (dynamic, chunk)

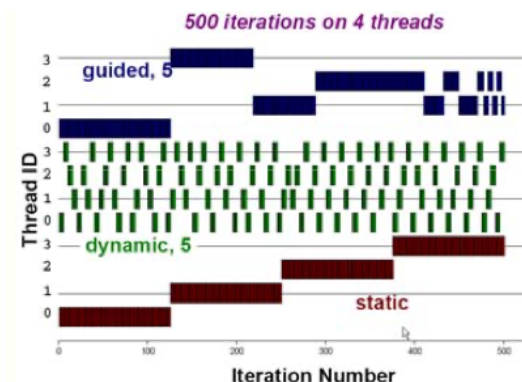
Divide el total de iteraciones del ciclo en tamaños dados por chunk y los va entregando uno a cada proceso. A medida que cada proceso termina su tarea, les va entregando nuevos grupos de índices. Por default chunk= 1.

Schedule (guided, chunk)

Similar al dynamic, pero asignando grupos de tamaño mayor al principio y menores luego, hasta terminar con grupos de tamaño chunk.

Schedule (runtime)

El diagramado se define al momento de ejecución. Lo toma de una variable de control.



Algunas preguntas

1. Dado el código que se tiene a continuación, que tipo de iteraciones a hebras seria el más optimo en tiempo de ejecución?

```
#pragma omp parallel for
for(int i=0; i<100; i++)
    for(int j=0; j<i; j++)
        a[i][j] = 0;
```

- A) Static
- B) El que indique la variable de control interno def-sched-var
- C) Runtime
- D) Dynamic**

2. Cual de las siguientes formas es la correcta para fijar a 4 el numero de hebras para un programa OpenMP?

- A) En la consola del sistema, usando la variable de entorno export OMP_THREAD_LIMIT=4
- B) En un programa OpenMP, usando la función omp_max_threads(4) al principio de la función main
- C) En un programa OpenMP, usando la función omp_num_threads(4) al principio de la función main
- D) En un programa OpenMP, usando la función omp_set_num_threads(4) al principio de la función main**

3. Indica que reparto de iteraciones a hebras es correcto suponiendo dos hebras y la cláusula schedule(guided, 3)

C

- a)

iteración	0	1	2	3	4	5	6	7	8	9	10	11
hebra	0	0	0	0	0	0	0	1	1	0	0	0
- b)

iteración	0	1	2	3	4	5	6	7	8	9	10	11
hebra	0	0	0	1	1	1	0	0	0	1	1	1
- c)

iteración	0	1	2	3	4	5	6	7	8	9	10	11
hebra	0	0	0	0	0	0	1	1	1	0	0	0
- d)

iteración	0	1	2	3	4	5	6	7	8	9	10	11
hebra	0	0	0	0	0	0	1	1	1	0	0	1

4. Indica que reparto de iteraciones a hebras es correcto suponiendo 3 hebras y la cláusula schedule(dynamic, 2)

D

- a)

iteración	0	1	2	3	4	5	6	7	8	9
hebra	0	0	1	1	2	2	2	2	0	1
- b)

iteración	0	1	2	3	4	5	6	7	8	9
hebra	0	0	1	1	2	2	0	0	0	2
- c)

iteración	0	1	2	3	4	5	6	7	8	9
hebra	0	0	0	1	1	1	2	2	2	0
- d)

iteración	0	1	2	3	4	5	6	7	8	9
hebra	0	0	1	1	2	2	1	1	0	0

5. Como se puede modificar el reparto de iteraciones del bucle de una directiva #pragma omp for entre las hebras si usamos la cláusula schedule(runtime)?

- A) Usando solo la función omp_set_schedule()
- B) Usando la variable de entorno OMP_SCHEDULE y la función omp_set_schedule()
- C) Usando solo la variable de entorno OMP_SCHEDULE
- D) Usando la variable de entorno OMP_SCHEDULE o la función omp_set_schedule()**

6. En una maquina con 8 cores y tras ejecutar export OMP_NUM_THREADS=4, cuantas iteraciones ejecuta la hebra master en la región parallel?

```
int N = omp_get_max_threads();
omp_set_num_threads(2);
#pragma omp parallel for num_threads(6) if(N>=4) schedule(static)
for(int i=0; i<12; i++)
    printf("thread: %d iteracion: %d \n", omp_get_thread_num(), i);
```

- A) 6
- B) 4
- C) 12
- D) 2**

7. Las variables de control internas de OpenMP

- A) Pueden ser accedidas directamente por el programador
- B) Las otras respuestas son incorrectas**
- C) Solo se pueden modificar mediante el uso de las funciones que proporciona el API de OpenMP
- D) Solo se pueden modificar mediante el uso de variables de entorno en la consola del sistema

8. Cuando se usa una planificación dynamic de un bucle for en OpenMP, el tamaño del Chunk...

- A) Es siempre constante**
- B) Vale decreciendo conforme se va ejecutando las iteraciones del bucle
- C) Se adapta dinámicamente en función de la velocidad de cada hebra
- D) Siempre debe ser mayor que 1

9. Que muestra la ejecución del siguiente programa por pantalla suponiendo que se ejecuta un nodo de actgrid?

```
int n = (int) (omp_get_max_threads() / 4);
#pragma omp parallel for num_threads(6) if (n>6)
for (int i=0; i<n; i++)
    printf("h: %d ", omp_get_thread_num());
```

- A) h:0
- B) Las otras respuestas no son correctas
- C) h:0 h:0 h:0 h:0 h:0 h:0**
- D) h:0 h:1 h:2 h:3 h:4 h:5

10. Con cuántas hebras se ejecuta este código si previamente se ha fijado la variable de entorno OMP_NUM_THREADS=8?

```
omp_set_num_threads(4);
#pragma omp parallel num_threads(2);
printf("Hello\n");
```

- A) 8
- B) 1
- C) 4
- D) 2**

11. Qué código cree mejor para conseguir multiplicar una matriz triangular superior por un vector?

```
int m[N][N], v[N], r[N] = {0};
```

A) `for (int i=0; i<N; i++)
 for(int j=0; j<N; j++)
 r[i] += m[i][j] * v[j]`

**B) `for (int i=0; i<N; i++)
 for(int j=i; j<N; j++)
 r[i] += m[i][j] * v[j]`**

C) `for (int j=0; j<N; j++)
 for(int i=0; i<N; i++)
 r[i] += m[i][j] * v[j]`

D) `for (int i=0; i<N; i++)
 for(int j=0; j<=i; j++)
 r[i] += m[i][j] * v[j]`

12. Si le piden que realice un estudio de escalabilidad de un código que calcula el producto de dos matrices

A) Representaría en una gráfica el tiempo de ejecución en función del tamaño de las matrices

B) Representaría en una gráfica la ganancia en velocidad (o ganancia en prestaciones) en función del número de núcleos ?

C) No haría nada de lo indicado en el resto de esquemas

D) Representa en una gráfica el tiempo de ejecución en función del número de núcleos

13. Qué código cree que es mejor para conseguir multiplicar una matriz triangular inferior por un vector?

A) `for (int i=0; i<N; i++) for(int j=i; j<=i; j++) r[i] += m[i][j] * v[j]`

B) `for (int j=0; j<N; j++) for(int i=0; i<N; i++) r[i] += m[i][j] * v[j]`

C) `for (int i=0; i<N; i++) for(int j=0; j<N; j++) r[i] += m[i][j] * v[j]`

D) `for (int i=0; i<N; i++) for(int j=0; j<=i; j++) r[i] += m[i][j] * v[j]`