

# BP3MoralesEsturilloJorge.pdf



Jrg14



Arquitectura de Computadores



2º Grado en Ingeniería Informática



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

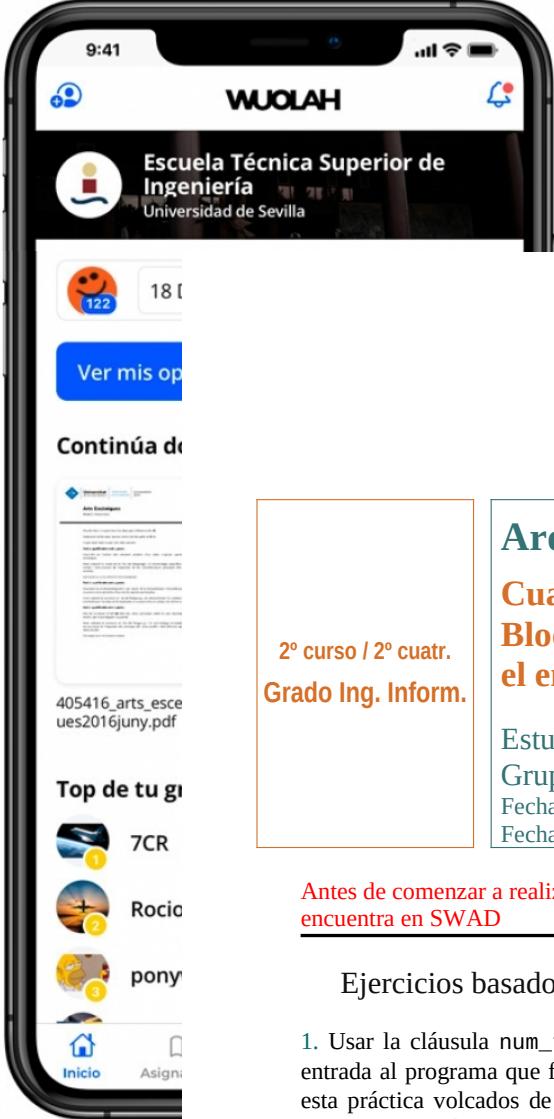


**Descarga la APP de Wuolah.**  
Ya disponible para el móvil y la tablet.





**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

## Arquitectura de Computadores (AC)

### Cuaderno de prácticas.

### Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP

Estudiante (nombre y apellidos):

Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con las normas de prácticas que se encuentra en SWAD

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la cláusula num\_threads(x) en el ejemplo del seminario if\_clause.c, y añadir un parámetro de entrada al programa que fije el valor x que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad de esta cláusula y explicar por qué lo ilustran.

#### CAPTURA CÓDIGO FUENTE: if-clauseModificado.c

```
#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,x;
    if (argc < 2)
    {
        fprintf(stderr, "[ERROR] Falta el número de iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]);
    x = atoi(argv[2]);
    if (n>20)
        n=20;
    for (i=0; i<n; i++)
        a[i]=i;
    #pragma omp parallel num_threads(x) 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);
    }
}
```

## CAPTURAS DE PANTALLA:

```
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer1] 2020-05-02 sábado
$ ./if-clauseModificado 10 1
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 0 suma de a[2]=2 sumalocal=3
thread 0 suma de a[3]=3 sumalocal=6
thread 0 suma de a[4]=4 sumalocal=10
thread 0 suma de a[5]=5 sumalocal=15
thread 0 suma de a[6]=6 sumalocal=21
thread 0 suma de a[7]=7 sumalocal=28
thread 0 suma de a[8]=8 sumalocal=36
thread 0 suma de a[9]=9 sumalocal=45
thread master=0 imprime suma=45
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer1] 2020-05-02 sábado
$ ./if-clauseModificado 10 4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 0 suma de a[2]=2 sumalocal=3
thread 2 suma de a[6]=6 sumalocal=6
thread 2 suma de a[7]=7 sumalocal=13
thread 1 suma de a[3]=3 sumalocal=3
thread 1 suma de a[4]=4 sumalocal=7
thread 1 suma de a[5]=5 sumalocal=12
thread 3 suma de a[8]=8 sumalocal=8
thread 3 suma de a[9]=9 sumalocal=17
thread master=0 imprime suma=45
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer1] 2020-05-02 sábado
$ ./if-clauseModificado 10 8
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 2 suma de a[4]=4 sumalocal=4
thread 6 suma de a[8]=8 sumalocal=8
thread 4 suma de a[6]=6 sumalocal=6
thread 3 suma de a[5]=5 sumalocal=5
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=3
thread 5 suma de a[7]=7 sumalocal=7
thread 7 suma de a[9]=9 sumalocal=9
thread master=0 imprime suma=45
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer1] 2020-05-02 sábado
$
```

**RESPUESTA:** Con esta cláusula lo que conseguimos es poder ejecutar el programa con las hebras que definamos según el parámetro de entrada que le demos. El primer parámetro son las iteraciones y el segundo las hebras que lo ejecutan. De esta manera como se muestra en la captura de la ejecución del programa, pues podemos hacer la suma con 1 hebra en la primera ejecución o 8 en la última por ejemplo.

**2. (a)** Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario *schedule-clause.c*, *scheduled-clause.c* y *scheduleg-clause.c* con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,...15)
- chunck= 1, 2 y 4

**Tabla 1 .** Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	1	0	0	1	0	0	0
6	0	1	1	0	0	1	0	0	0
7				0	0	1	0	0	0
8				0	0	0	1	1	1
9				0	0	0	1	1	1
10				0	0	0	1	1	1
11				1	0	0	1	1	1
12				1	0	0	1	0	0
13				1	0	0	1	0	0
14				1	1	0	1	0	0
15				1	1	0	0	0	0

**(b)** Rellenar otra tabla como la de la figura pero esta vez usando cuatro *threads* (0,1,2,3).

**Tabla 2 .** Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-clause.d.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	2	0	0	1	1	0
1	1	0	0	0	0	0	1	1	0
2	2	1	0	1	2	0	1	1	0
3	3	1	0	3	2	0	1	1	0
4	0	2	1	2	3	2	0	0	1
5	1	2	1	2	3	2	0	0	1
6	2	3	1	2	1	2	0	0	1
7				2	1	2	3	3	1
8				2	1	1	3	3	2
9				2	1	1	3	3	2
10				2	1	1	2	2	2
11				2	1	1	2	2	2
12				2	1	3	1	1	3
13				3	1	3	1	1	3
14				3	1	3	1	1	3
15				3	1	3	1	1	3

Escriba en el cuaderno de prácticas las diferencias en el comportamiento de `schedule()` con `static`, `dynamic` y `guided`.

**RESPUESTA:** La cláudula `schedule` distribuye y asigna el trabajo del bucle entre las distintas hebras. En `static` dicha distribución se hace en tiempo de compilación, dividiéndose las interacciones en chunks, de manera que el número de interacciones de cada hebra está ya definido y es equitativo como se puede ver (dependiendo de si el número de interacciones es múltiplo del número de hebras). En `dynamic` la distribución y asignación se hace en tiempo ejecución por lo que es desconocido qué hebra realizará qué interacciones. Las hebras más rápidas ejecutarán más trabajo, aunque como mínimo tienen que ejecutar el chunk definido con anterioridad. En `guided` la distribución y asignación es como en `dynamic`. Empieza con un bloque completo pero este va decreciendo dependiendo del número de interacciones que resten, aunque nunca más pequeño que el chunk. Aquí también las hebras más rápidas ejecutarán más

3. Añadir al programa `scheduled-clause.c` lo necesario para que imprima el valor de las variables de control `dyn-var`, `nthreads-var`, `thread-limit-var` y `run-sched-var` dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorporar en su cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

### CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado.c`

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

int main(int argc, char **argv)
{
    int i, n=200, chunk, modif, a[n], suma=0;
    omp_sched_t tipo;

    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]);

    int dyn_var = omp_get_dynamic();
    int nthreads_var = omp_get_max_threads();
    int thread_limit_var = omp_get_thread_limit();
    omp_get_schedule(&tipo, &modif);

    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);
}

#pragma omp single
printf("\nthread %d dyn-var %d \n", omp_get_thread_num(), dyn_var);
printf("thread %d nthreads-var %d \n", omp_get_thread_num(), nthreads_var);
printf("thread %d thread-limit-var %d \n", omp_get_thread_num(), thread_limit_var);
printf("thread %d run-sched-var %d \n", omp_get_thread_num(), modif);

printf("\nFuera de 'parallel for' suma %d\n dyn-var %d\n nthreads-var %d\n thread-limit-var %d\n run-sched-var mod %d\n", suma, dyn_var, nthreads_var, thread_limit_var, modif);
}
```

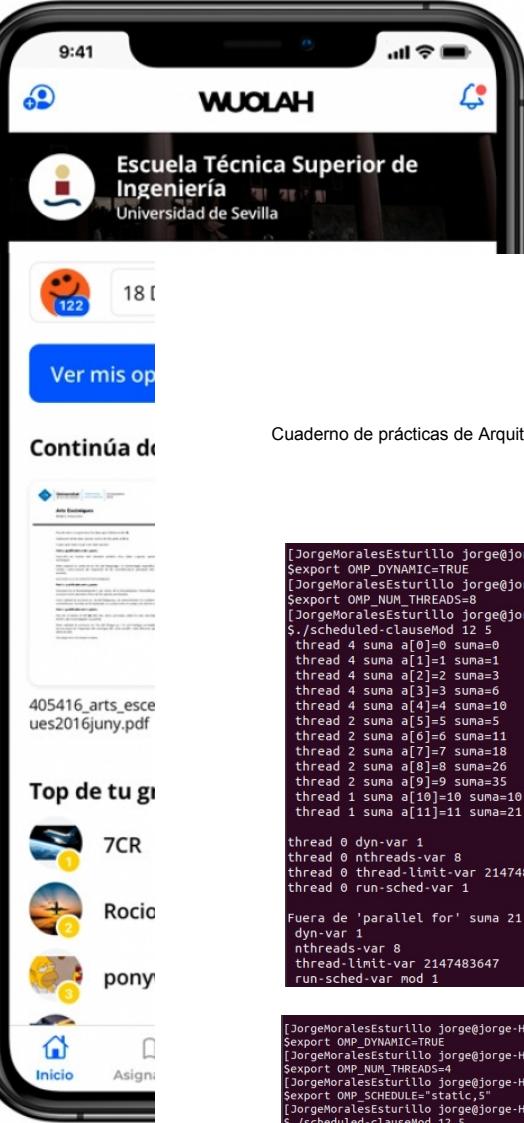
### CAPTURAS DE PANTALLA:

```
Fuera de 'parallel for' suma 5
dyn-var 0
nthreads-var 4
thread-limit-var 2147483647
run-sched-var mod 1
El tipo es dinámico
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:-/Escritorio/BP3/ejer3] 2020-05-02 sábado
$gcc -fopenmp -O2 -o scheduled-clauseMod scheduled-clauseMod.c
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:-/Escritorio/BP3/ejer3] 2020-05-02 sábado
$gcc -fopenmp -O2 -o scheduled-clauseMod scheduled-clauseMod.c
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:-/Escritorio/BP3/ejer3] 2020-05-02 sábado
$clear

[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:-/Escritorio/BP3/ejer3] 2020-05-02 sábado
$./scheduled-clauseMod 10
thread 2 suma a[0]=0 suma=0
thread 2 suma a[1]=1 suma=1
thread 2 suma a[2]=2 suma=3
thread 2 suma a[3]=3 suma=6
thread 1 suma a[4]=4 suma=4
thread 1 suma a[5]=5 suma=9
thread 1 suma a[6]=6 suma=15
thread 1 suma a[7]=7 suma=22
thread 3 suma a[8]=8 suma=8
thread 3 suma a[9]=9 suma=17

thread 0 dyn-var 0
thread 0 nthreads-var 4
thread 0 thread-limit-var 2147483647
thread 0 run-sched-var 1

Fuera de 'parallel for' suma 17
dyn-var 0
nthreads-var 4
thread-limit-var 2147483647
run-sched-var mod 1
```



# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

```
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer3] 2020-05-02 sábado
$export OMP_DYNAMIC=TRUE
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer3] 2020-05-02 sábado
$export OMP_NUM_THREADS=8
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer3] 2020-05-02 sábado
$./scheduled-clauseMod 12 5
thread 4 suma a[0]=0 suma=0
thread 4 suma a[1]=1 suma=1
thread 4 suma a[2]=2 suma=3
thread 4 suma a[3]=3 suma=
thread 4 suma a[4]=4 suma=10
thread 2 suma a[5]=5 suma=5
thread 2 suma a[6]=6 suma=11
thread 2 suma a[7]=7 suma=18
thread 2 suma a[8]=8 suma=26
thread 2 suma a[9]=9 suma=35
thread 1 suma a[10]=10 suma=10
thread 1 suma a[11]=11 suma=21
```

```
thread 0 dyn-var 1
thread 0 nthreads-var 8
thread 0 thread-limit-var 2147483647
thread 0 run-sched-var 1

Fuera de 'parallel for' suma 21
dyn-var 1
nthreads-var 8
thread-limit-var 2147483647
run-sched-var mod 1
```

```
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer3] 2020-05-02 sábado
$export OMP_DYNAMIC=TRUE
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer3] 2020-05-02 sábado
$export OMP_NUM_THREADS=4
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer3] 2020-05-02 sábado
$export OMP_SCHEDULE="static,5"
[JorgeMoralesEsturillo jorge@jorge-HP-Pavillion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer3] 2020-05-02 sábado
$./scheduled-clauseMod 12 5
thread 2 suma a[0]=0
thread 2 suma a[1]=1 suma=1
thread 2 suma a[2]=2 suma=3
thread 2 suma a[3]=3 suma=6
thread 2 suma a[4]=4 suma=10
thread 1 suma a[5]=5 suma=15
thread 1 suma a[6]=6 suma=21
thread 1 suma a[7]=7 suma=18
thread 1 suma a[8]=8 suma=26
thread 1 suma a[9]=9 suma=35
thread 0 suma a[10]=10 suma=10
thread 0 suma a[11]=11 suma=21

thread 0 dyn-var 1
thread 0 nthreads-var 4
thread 0 thread-limit-var 2147483647
thread 0 run-sched-var 5

Fuera de 'parallel for' suma 21
dyn-var 1
nthreads-var 4
thread-limit-var 2147483647
run-sched-var mod 5
```

**RESPUESTA:** En todas las ejecuciones imprime lo mismo dentro y fuera del parallel, que se corresponden a los valores establecidos anteriormente en las variables de entorno

4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

#### CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado4.c

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

#ifndef _OPENMP
#define _OPENMP
#endif
#include <omp.h>
#define _omp_get_thread_num() 0
#endif

int main(int argc, char **argv)
{
    int i, n=200, chunk, modif, a[n], suma=0;
    omp_sched_t tipo;
    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]);
    omp_get_schedule(&tipo, &modif);
    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++)
    {
        if( i == 0 )
            printf("Dentro del parallel; num_threads = %d, num_procs = %d, omp_in_parallel = %d\n",omp_get_num_threads(), omp_get_num_procs(),omp_in_parallel() );
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",omp_get_thread_num(),i,a[i],suma);
    }
    printf("\nFuera de 'parallel for' suma %d\n",suma);
    printf("omp_get_num_threads()=%d \n",omp_get_num_threads());
    printf("omp_get_num_procs()=%d \n",omp_get_num_procs());
    printf("omp_in_parallel()=%d \n",omp_in_parallel());
}
}
```

#### CAPTURAS DE PANTALLA:

```
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer4] 2020-05-02 sábado
$gcc -fopenmp -O2 -o scheduled-clauseMod2 scheduled-clauseMod2.c
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer4] 2020-05-02 sábado
$./scheduled-clauseMod2 4 4
Dentro del parallel; omp_get_num_threads()= 4, omp_get_num_procs() = 8, omp_in_parallel = 1
    thread 2 suma a[0]=0 suma=0
    thread 2 suma a[1]=1 suma=1
    thread 2 suma a[2]=2 suma=3
    thread 2 suma a[3]=3 suma=6

Fuera de 'parallel for' suma 6
omp_get_num_threads()= 1
omp_get_num_procs() = 8
omp_in_parallel() = 0
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer4] 2020-05-02 sábado
$]
```

**RESPUESTA:** Se obtienen diferentes el num\_threads y el omp\_parallel

**5.** Añadir al programa `scheduled-clause.c` lo necesario para modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

#### CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

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

int main(int argc, char ** argv){
    int t, n = 200, chunk,valor_chunk, a[n], suma = 0;
    omp_sched_t tipo;

    if(argc < 3) {
        fprintf(stderr, "\nFalta tteraciones 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);
        if(i==(n-2)){
            omp_set_dynamic(0);
            omp_set_num_threads(4);
            omp_set_schedule(1,2);
        }
        if(i==0 || i==(n-2)){
            if(i==0)
                printf("ANTES del cambio:\n");
            else
                printf("Después del cambio:\n");
            omp_get_schedule(&tipo, &valor_chunk);
            printf("dyn-var= %d, nthreads= %d, run-sched-var= %d \n",  omp_get_dynamic(), omp_get_max_threads(), tipo);
        }
    }
    printf("Fuerza de 'parallel for' suma=%d\n",suma);
    printf("dyn-var= %d, nthreads-var = %d, run-sched-var= %d\n",omp_get_dynamic(), omp_get_max_threads(), tipo);
}

return(0);
}
```

#### CAPTURAS DE PANTALLA:

```
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer5] 2020-05-02 sábado
$ ./scheduled-clauseMod3 8 8
thread 0 suma a[0]=0 suma=0
Antes del cambio:
dyn-var= 1, nthreads= 4, run-sched-var= 1
thread 0 suma a[1]=1 suma=1
thread 0 suma a[2]=2 suma=3
thread 0 suma a[3]=3 suma=6
thread 0 suma a[4]=4 suma=10
thread 0 suma a[5]=5 suma=15
thread 0 suma a[6]=6 suma=21
Después del cambio:
dyn-var= 0, nthreads= 4, run-sched-var= 1
thread 0 suma a[7]=7 suma=28
Fuerza de 'parallel for' suma=28
dyn-var = 1, nthreads-var = 4, run-sched-var= 1
```

**RESPUESTA:** Lo que hacemos es que los valores se dan por las variables de entorno, y esos son los valores antes de modificar. Tras modificarlos los valores están proporcionados por las funciones en tiempo de ejecución. Eso si, las variables `dyn-var` y `nthreads` fuera del `parallel` tienen los valores proporcionados por el entorno aunque `run-sched` si mantiene las modificaciones

## Resto de ejercicios

**6.** Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por vector.

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

### CAPTURA CÓDIGO FUENTE: pmtv-secuencial.c

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

int main(int argc, char** argv){
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    if (argc<2){
        printf("Faltan los componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    int i, j;
    int *v, *mv, **m;

    v = (int*) malloc(N*sizeof(int));
    mv = (int*) malloc(N*sizeof(int));
    m = (int**)malloc(N*sizeof(int *));
    // Máximo N=2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    for (i=0;i<N;i++)
        m[i] = (int *) malloc (N*sizeof(int));
    // si no hay suficiente espacio, malloc devuelve NULL
    // asignamos memoria a una matriz de punteros que
    // referenciarán a cada una de las filas
    // asignamos memoria para cada una de las filas

    // Comprobamos que no ha habido error en la reserva de espacio
    if (v==NULL || mv==NULL || m==NULL){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-2);
    }
    for(i=0; i<N; i++)
        if(m[i]==NULL){
            printf("Error en la reserva de espacio para la matriz y el vector\n");
            exit(-3);
        }
    // Inicializamos la matriz triangular superior(m), el vector(v) y el vector resultante(mv)
    for(i=0; i<N; i++){
        for(j=i; j<N; j++){
            m[i][j] = 9;
        }
    }
    v[i] = 2;
    mv[i] = 0;
    //los valores dependen de N
    // lo inicializo a 0 para que no contenga basura

    clock_gettime(CLOCK_REALTIME,&cg1);
    // Calculamos la multiplicación de la matriz por el vector
    for(i=0; i<N; i++){
        for(j=i; j<N; j++){
            mv[i] += m[i][j] * v[i];
        }
    }
    clock_gettime(CLOCK_REALTIME,&cg2);

    // vemos la matriz
    printf("Matriz:\n");
    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            if(j >= i)
                printf("%d ", m[i][j]);
            else
                printf("0 ");
        }
        printf("\n");
    }

    // vemos el vector
    printf("Vector:\n");
    for(i=0; i<N; i++)
        printf("%d ", v[i]);
    printf("\n");

    // vemos el vector resultante
    printf("Resultado:\n");
    for(i=0; i<N; i++)
        printf("%d ", mv[i]);
    printf("\n");

    ncgt=(double) (cg2.tv_sec-cg1.tv_sec)+(double) ((cg2.tv_nsec-cg1.tv_nsec)/(1.e+9));
    printf("Tamaño de la matriz y vector:%d\t / Tiempo(seg.) %11.9f\n", N, ncgt);

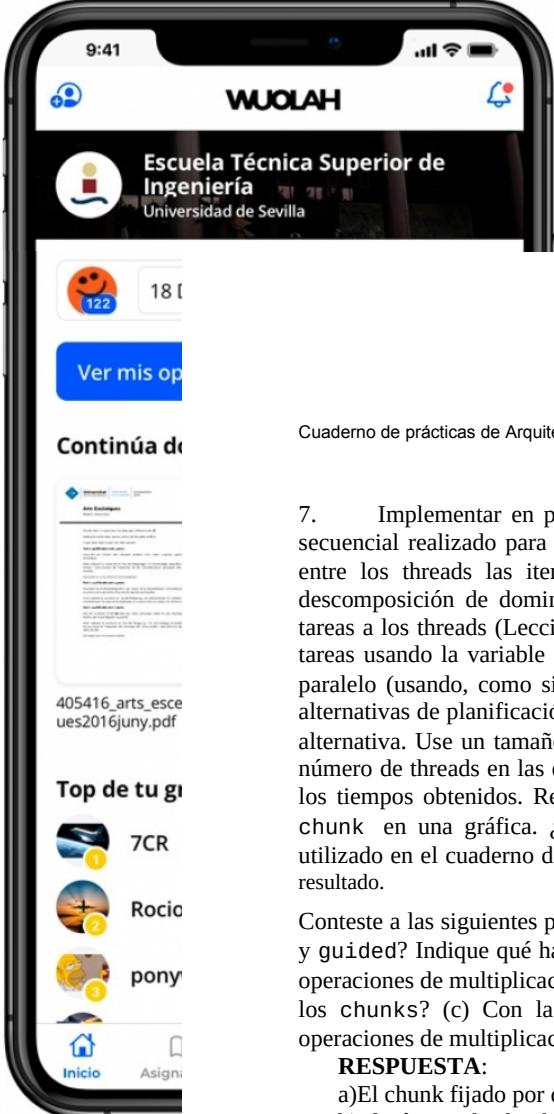
    for(i=0; i<N; i++)
        free(m[i]);
}

free(m); // liberamos el espacio de la matriz
free(v); // liberamos el espacio reservado para v
free(mv); // liberamos el espacio del vector mv resultado

return 0;
}
```

### CAPTURAS DE PANTALLA:

```
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer6] 2020-05-02 sábado
$./pmtv-secuencial 5
Matriz:
9 9 9 9 9
0 9 9 9 9
0 0 9 9 9
0 0 0 9 9
0 0 0 0 9
Vector:
2 2 2 2 2
Resultado:
90 72 54 36 18
Tamaño de la matriz y vector:5 / Tiempo(seg.) 0.000000796
```



# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Obtener en atcgrid los tiempos de ejecución del código paralelo (usando, como siempre, -O2 al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para chunk de 1, 64 y el chunk por defecto para la alternativa. Use un tamaño de vector N múltiplo del número de cores y de 64 que no sea inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del chunk en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para chunk con `static`, `dynamic` y `guided`? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación `static` para cada uno de los chunks? (c) Con la asignación `dynamic` y `guided`, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

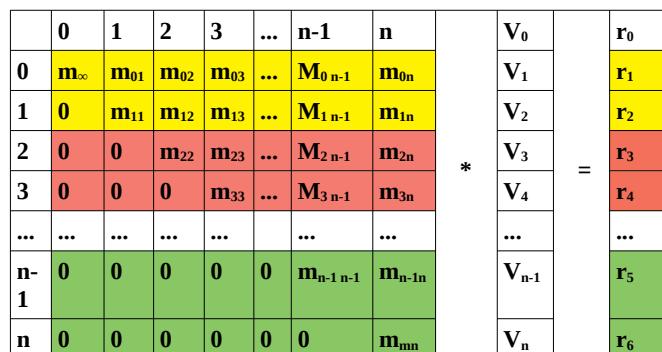
#### RESPUESTA:

- a) El chunk fijado por defecto de `dynamic` y `guided` es 1, pero `static` no tiene uno especificado
- b) El número de chunk por el número de fila
- c) En `dynamic` se ejecutarán las 64 operaciones a la vez o 1. En `guided` el último no será igual

**CAPTURA CÓDIGO FUENTE:** pmtv-OpenMP.c

```
// Calculamos la multiplicación de la matriz por el vector
#pragma omp parallel for schedule(runtime)
for(i=0; i<N; i++){
    for(j=i; j<N;j++){
        mv[i] += m[i][j] * v[j];
    }
}
t_final = omp_get_wtime();
t_total = t_final - t_init;
```

#### DESCOMPOSICIÓN DE DOMINIO:



Por ejemplo si hicieramos una asignación estática (RR) y fijamos el chunk a 2, para n hebras. Cada hebra multiplicaría por el vector v dos filas de la matriz de forma consecutiva. En la representación gráfica se muestra la descomposición por colores.

## CAPTURAS DE PANTALLA:

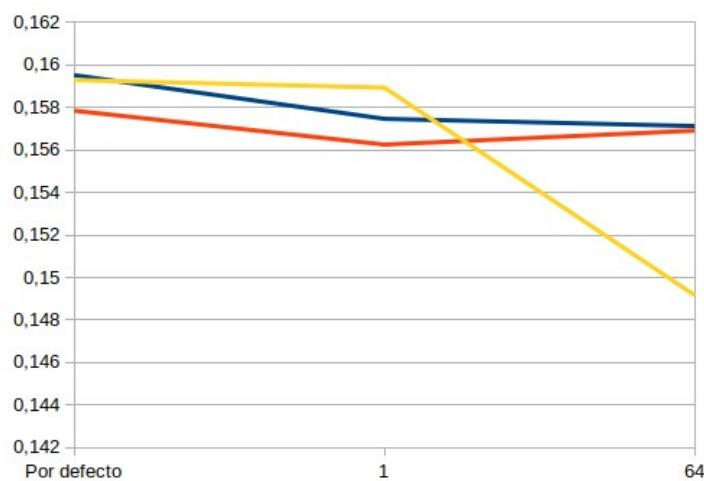
```
[JorgeMoralesEsturillo d2estudiante18@atcgrid:~] 2020-05-03 domingo
$srun -p ac ./pmvt-Open.sh
static y chunk por defecto
Tamaño de la matriz y vector:15360 / Tiempo(seg.) 0.159516856
static y chunk 1
Tamaño de la matriz y vector:15360 / Tiempo(seg.) 0.157462411
static y chunk 64
Tamaño de la matriz y vector:15360 / Tiempo(seg.) 0.157122519
dynamic y chunk por defecto
Tamaño de la matriz y vector:15360 / Tiempo(seg.) 0.157847788
dynamic y chunk 1
Tamaño de la matriz y vector:15360 / Tiempo(seg.) 0.156251792
dynamic y chunk 64
Tamaño de la matriz y vector:15360 / Tiempo(seg.) 0.156907581
guided y chunk por defecto
Tamaño de la matriz y vector:15360 / Tiempo(seg.) 0.159283459
guided y chunk 1
Tamaño de la matriz y vector:15360 / Tiempo(seg.) 0.158929277
guided y chunk 64
Tamaño de la matriz y vector:15360 / Tiempo(seg.) 0.149162970
```

## TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

**SCRIPT:** pmvt-OpenMP\_atcgrid.sh

**Tabla 3 .Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N= , 12 threads**

Chunk	Static	Dynamic	Guided
por defecto	0.159516856	0.157847788	0.159283459
1	0.157462411	0.156251792	0.158929277
64	0.157122519	0.156907581	0.149162970
Chunk	Static	Dynamic	Guided
por defecto	0.159501668	0.157358717	0.158832505
1	0.157429591	0.157280315	0.159252893
64	0.157919534	0.158094812	0.159371056



## GRÁFICA:

**8. Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:**

$$A = B \bullet C; A(i, j) = \sum_{k=0}^{N-1} B(i, k) \bullet C(k, j), i, j = 0 \dots N - 1$$

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

**CAPTURA CÓDIGO FUENTE:** pmm-secuencial.c

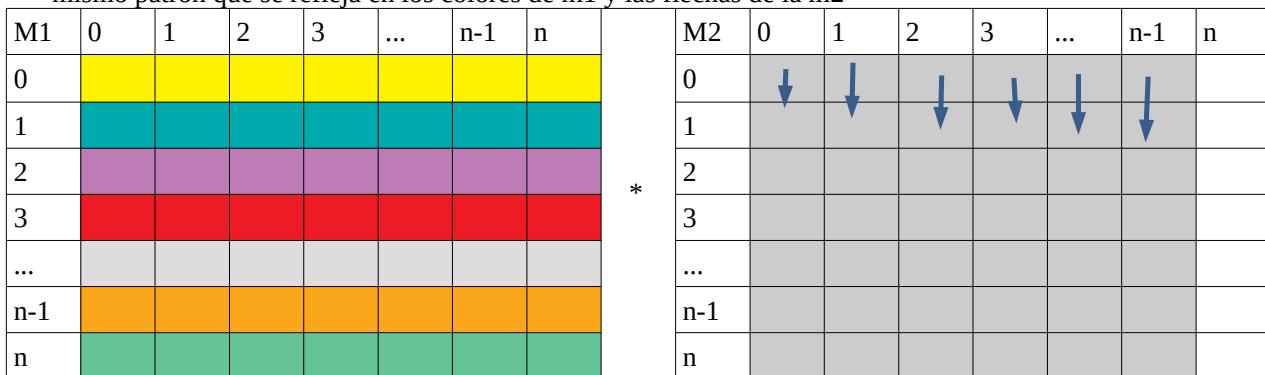
```
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#ifndef _OPENMP
#define _OPENMP
#endif
#include <omp.h>
int main(int argc, char** argv){
    int i, j, k;
    double t_in, t_fin, t_total;
    if (argc<2){
        printf("Faltan los argumentos\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); // Máximo N = 2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
    int **mr, **m1, **m2;
    mr = (int **) malloc(N*sizeof(int *));
    m1 = (int **) malloc(N*sizeof(int *));
    m2 = (int **) malloc(N*sizeof(int *));
    for (i=0;i<N;i++){
        mr[i] = (int *) malloc (N*sizeof(int));
        m1[i] = (int *) malloc (N*sizeof(int));
        m2[i] = (int *) malloc (N*sizeof(int));
    }
    // Comprobamos que no ha habido error en la reserva de espacio
    if ( mr==NULL || m1==NULL || m2==NULL ){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-2);
    }
    for(i=0; i<N; i++)
        if(mr[i]==NULL || m1[i]==NULL || m2[i]==NULL){
            printf("Error en la reserva de espacio para la matriz y el vector\n");
            exit(-3);
        }
    // Inicializamos las matrices
    for(i=0; i<N; i++)
        for(j=0; j<N; j++){
            mr[i][j] = 0;
            m1[i][j] = 2*i;
            m2[i][j] = 2*j;
        }
    t_in = omp_get_wtime();
    // Calculamos la multiplicación de la matriz por el vector
    for(i=0; i<N; i++)
        for(j=0; j<N; j++){
            for(k=0; k<N; k++){
                mr[i][j] += m1[i][k] * m2[k][j];
            }
        }
    t_fin = omp_get_wtime();
    t_total = t_fin - t_in;
    printf("Tamaño de la matriz y vector:%d / Tiempo(seg.) %11.9f\n", N, t_total);
    // Visualizamos la primera y última linea de la matriz resultante
    printf("Tiempo = %11.9f Primera linea= %d Última linea= %d\n", t_total, mr[0][0],mr[N-1][N-1]);
    return 0;
}
```

**CAPTURAS DE PANTALLA:**

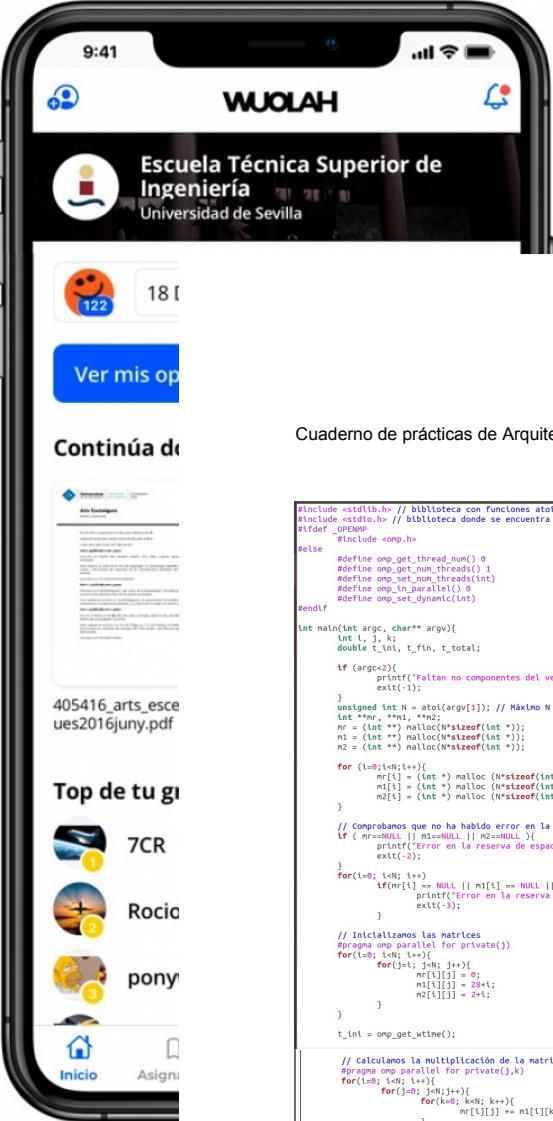
```
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer8] 2020-05-02 sábado
$./pmm-secuencial 50
Tamaño de la matriz y vector:50 / Tiempo(seg.) 0.000368102
Tiempo = 0.000368102 Primera linea= 56 Última linea= 3927
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer8] 2020-05-02 sábado
$./pmm-secuencial 100
Tamaño de la matriz y vector:100 / Tiempo(seg.) 0.003302876
Tiempo = 0.003302876 Primera linea= 56 Última linea= 12827
[JorgeMoralesEsturillo jorge@jorge-HP-Pavilion-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer8] 2020-05-02 sábado
$./pmm-secuencial 1000
Tamaño de la matriz y vector:1000 / Tiempo(seg.) 1.160321417
Tiempo = 1.160321417 Primera linea= 56 Última linea= 1028027
```

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe parallelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2, Lección 5/Tema 2).

**DESCOMPOSICIÓN DE DOMINIO:** Cada hebra recorre diferentes filas de la primera matriz, pero todas recorren sus columnas, y todas las filas y columnas de la m2. Es decir, cada hebra multiplicac cada casilla de cada fila de la matriz m1 por cada casilla de cada columna de la m2. De este modo cada hebra se correspondería con cada color de m1, que recorre dicha fila, además de todas las columnas de m<sup>2</sup> y escribiría los resultados en la misma fila de la matriz resultado, siguiendo el mismo patrón que se refleja en los colores de m1 y las flechas de la m2



**CAPTURA CÓDIGO FUENTE:** pmm-OpenMP.c



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

The image contains two promotional logos. On the left is the 'Available on the App Store' logo, which features the Apple logo icon followed by the text 'Available on the App Store'. On the right is the 'GET IT ON Google Play' logo, which features the Google Play logo icon followed by the text 'GET IT ON Google Play'.

The logo consists of the Apple logo followed by the text "Available on the App Store".

A black rectangular button with white text and a colorful play icon. The text "GET IT ON" is at the top, followed by "Google Play" below it, with a small play icon to the left of "Google".

18

[Ver mis op](#)

Continúa de

Cuaderno de prácticas de Arquitectura de Computadores. Grado en Ingeniería Informática

405416\_arts\_esce  
ues2016juniy.pdf

Top de tu gi

 7CR

 Rocio

pony

1

## CAPTURAS DE PANTALLA:

```
[JorgeMoralesEsturillo Jorge@Jorge-HP-Pavillion-Laptop-15-cs0xxx-]~/Escritorio/BP3/ejer9] 2020-05-02 sábado
S./ppm/OpenH 2
Tamaño de la matriz y vector:2 / Tiempo(seg.) 0.000008983
Tiempo = 0.000008983 Primera linea: 0168660792 Última linea: -535822185
[JorgeMoralesEsturillo Jorge@Jorge-HP-Pavillion-Laptop-15-cs0xxx-]~/Escritorio/BP3/ejer9] 2020-05-02 sábado
S./ppm/OpenH 4
Tamaño de la matriz y vector:4 / Tiempo(seg.) 0.000008983
Tiempo = 0.000008983 Primera linea: 0168660792 Última linea: -234266894
[JorgeMoralesEsturillo Jorge@Jorge-HP-Pavillion-Laptop-15-cs0xxx-]~/Escritorio/BP3/ejer9] 2020-05-02 sábado
S./ppm/OpenH 8
Tamaño de la matriz y vector:8 / Tiempo(seg.) 0.000008983
Tiempo = 0.000008983 Primera linea: 1682707456 Última linea: 315
[JorgeMoralesEsturillo Jorge@Jorge-HP-Pavillion-Laptop-15-cs0xxx-]~/Escritorio/BP3/ejer9] 2020-05-02 sábado
S./ppm/OpenH 16
Tamaño de la matriz y vector:16 / Tiempo(seg.) 0.000008983
Tiempo = 0.000008983 Primera linea: 56 Última linea: 731
```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar -O2 al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos del computador. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas (pruebe con valores de N entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

## ESTUDIO DE ESCALABILIDAD EN atcgrid:

**SCRIPT:** pmm-OpenMP\_atcgrid.sh

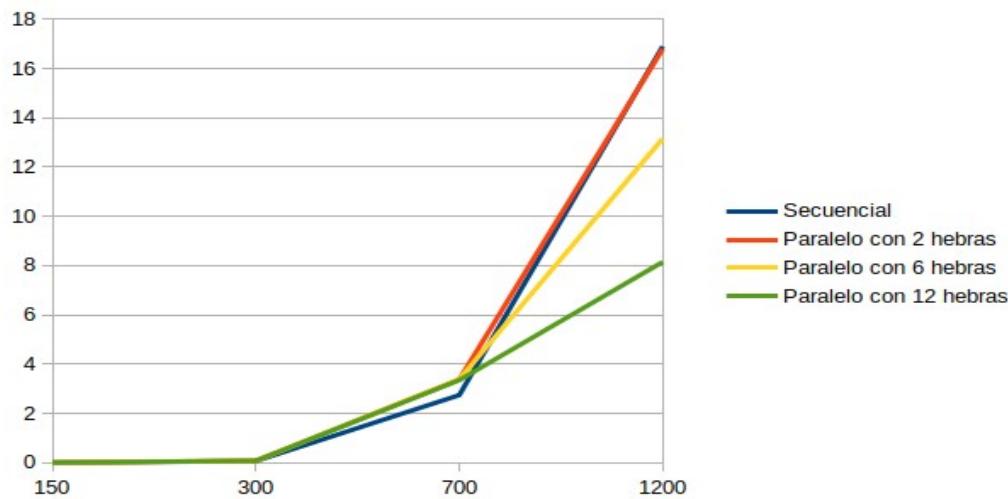
```
#!/bin/bash
#PBS -N ej7_atcgrid
#PBS -q ac
#echo "Id$PBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
#echo "Id$PBS_O_WORKDIR del trabajo: $PBS_JOBID"
#echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
#echo "Nodo que ejecuta qsub: $PBS_O_HOST"
#echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
#echo "Cola: $PBS_QUEUE"
#echo "Nodos asignados al trabajo:"
#cat $PBS_NODEFILE

echo "Paralelo"
./pmm-OpenMP 150
./pmm-OpenMP 300
./pmm-OpenMP 700
./pmm-OpenMP 1200
```

**Comentario:** El script del actgrid era similar al del localpc, pero no sé las causas no me dejaba realizar el export OMP\_NUM\_THREADS=threads que queramos. De forma que lo he hecho de manera manual

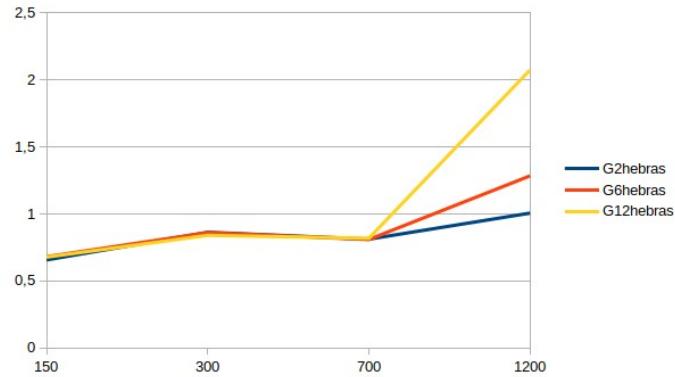
**Tabla:**

Tamaño	Secuencial	Paralelo con 2 hebras	Paralelo con 6 hebras	Paralelo con 12 hebras
150	0,005027078	0,007649172	0,007380974	0,007412121
300	0,067601498	0,078337748	0,078550808	0,080419362
700	2,744063843	3,379699446	3,389683731	3,352869093
1200	16,894426871	16,791755341	13,156893693	8,147155315



## Ganancia atcgrid

G2hebras	G6hebras	G12hebras
0,6572055119	0,6810859922	0,6782239524
0,8629492132	0,8606085631	0,8406122148
0,8119254055	0,809533886	0,8184226007
1,0061144013	1,2840741337	2,0736596048



## ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP\_pclocal.sh

```
echo "Secuencial"
./pmm-secuencial 150
./pmm-secuencial 300
./pmm-secuencial 700
./pmm-secuencial 1200

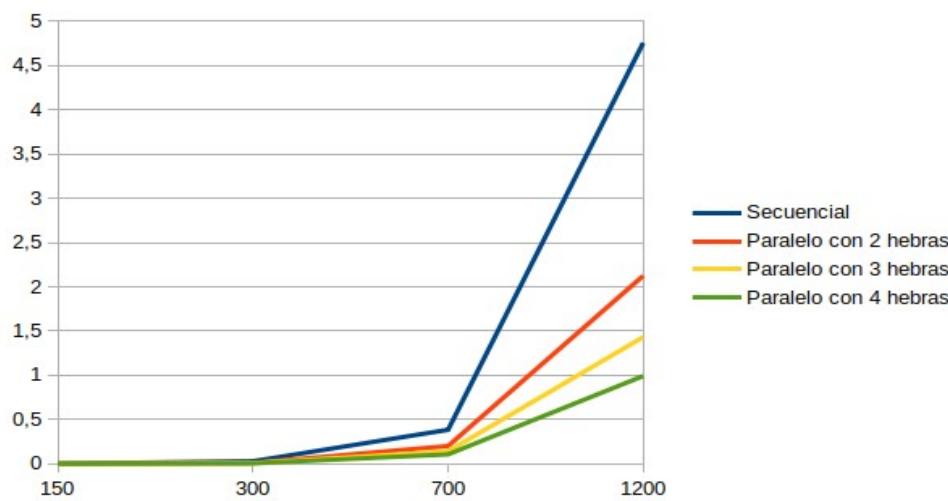
echo "\n"
echo "Paralelo con 2 threads"
export OMP_NUM_THREADS=2
./pmm-secuencial 150
./pmm-secuencial 300
./pmm-secuencial 700
./pmm-secuencial 1200

echo "\n"
echo "Paralelo con 3 threads"
export OMP_NUM_THREADS=3
./pmm-secuencial 150
./pmm-secuencial 300
./pmm-secuencial 700
./pmm-secuencial 1200

echo "\n"
echo "Paralelo con 4 threads"
export OMP_NUM_THREADS=4
./pmm-secuencial 150
[JorgeMoralesSturillo Jorge-HP-Pavillon-Laptop-15-cs0xxx:~/Escritorio/BP3/ejer10] 2020-05-03 domingo
$ ./pmm-OpenMP_pclocal.sh
Secuencial:
Tamaño de la matriz y vector:150      / Tiempo(seg.) 0.010739864
Tiempo = 0.010739864      Primera linea= 56      Última linea= 26727
Tamaño de la matriz y vector:300      / Tiempo(seg.) 0.043879417
Tiempo = 0.043879417      Primera linea= 56      Última linea= 98427
Tamaño de la matriz y vector:700      / Tiempo(seg.) 0.372901808
Tiempo = 0.372901808      Primera linea= 56      Última linea= 509627
Tamaño de la matriz y vector:1200     / Tiempo(seg.) 4.825917755
Tiempo = 4.825917755      Primera linea= 56      Última linea= 1473627
\n
Paralelo con 2 threads
Tamaño de la matriz y vector:150      / Tiempo(seg.) 0.001413899
Tiempo = 0.001413899      Primera linea= 56      Última linea= 26727
Tamaño de la matriz y vector:300      / Tiempo(seg.) 0.010988196
Tiempo = 0.010988196      Primera linea= 56      Última linea= 98427
Tamaño de la matriz y vector:700      / Tiempo(seg.) 0.200973409
Tiempo = 0.200973409      Primera linea= 56      Última linea= 509627
Tamaño de la matriz y vector:1200     / Tiempo(seg.) 2.123459612
Tiempo = 2.123459612      Primera linea= 56      Última linea= 1473627
\n
Paralelo con 3 threads
Tamaño de la matriz y vector:150      / Tiempo(seg.) 0.000793761
Tiempo = 0.000793761      Primera linea= 56      Última linea= 26727
Tamaño de la matriz y vector:300      / Tiempo(seg.) 0.008018171
Tiempo = 0.008018171      Primera linea= 56      Última linea= 98427
Tamaño de la matriz y vector:700      / Tiempo(seg.) 0.141338010
Tiempo = 0.141338010      Primera linea= 56      Última linea= 509627
Tamaño de la matriz y vector:1200     / Tiempo(seg.) 1.431296319
Tiempo = 1.431296319      Primera linea= 56      Última linea= 1473627
\n
Paralelo con 4 threads
Tamaño de la matriz y vector:150      / Tiempo(seg.) 0.001197582
Tiempo = 0.001197582      Primera linea= 56      Última linea= 26727
Tamaño de la matriz y vector:300      / Tiempo(seg.) 0.005303606
Tiempo = 0.005303606      Primera linea= 56      Última linea= 98427
Tamaño de la matriz y vector:700      / Tiempo(seg.) 0.106989759
Tiempo = 0.106989759      Primera linea= 56      Última linea= 509627
Tamaño de la matriz y vector:1200     / Tiempo(seg.) 0.992216089
Tiempo = 0.992216089      Primera linea= 56      Última linea= 1473627
```

Tabla:

Tamaño	Secuencial	Paralelo con 2 hebras	Paralelo con 3 hebras	Paralelo con 4 hebras
150	0.010739864	0.001413899	0.000793761	0.001197582
300	0.043879417	0.010988196	0.008018171	0.005303606
700	0.37290180	0.200973409	0.141338010	0.106989759
1200	4.825917755	2.123459612	1.431296319	0.992216089



### Ganancia en pc

Gcon2	Gcon3	Gcon4
7,5959202178	13,530349816	8,9679571002
3,9933231078	5,4724970321	8,2735061767
1,8554783036	2,6383688294	3,4853971397
2,2726675505	3,3717111481	4,863776962

