

2º curso / 2º
cuatr.

Grado Ing.
Inform.

Doble Grado
Ing. Inform.
y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

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

Estudiante (nombre y apellidos): Marta Gómez Macías

Grupo de prácticas: C1

Fecha de entrega: 19/05/2015 - 23:59

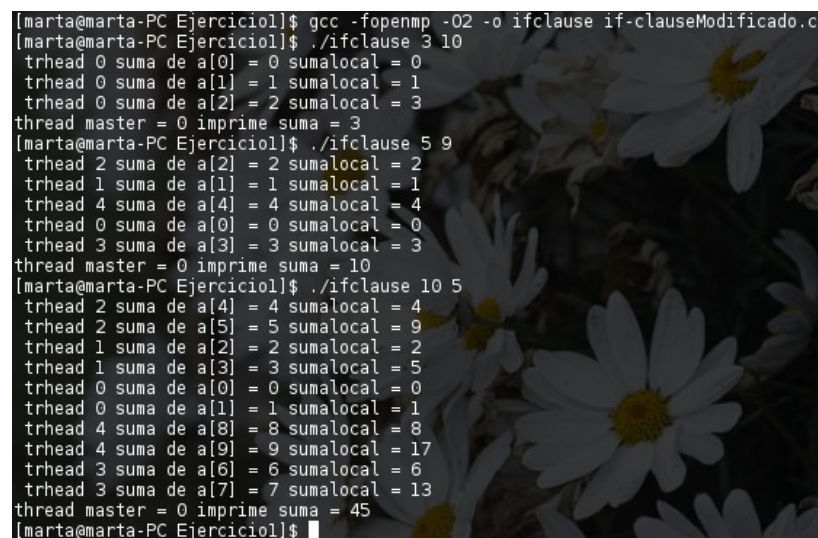
Fecha evaluación en clase: 20/05/2015 - 15:30

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.

CÓDIGO FUENTE: `if-clauseModificado.c`

```
#pragma omp parallel if(n>4) num_threads(num_hebras) 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(" trhead %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:



```
[marta@marta-PC Ejercicio1]$ gcc -fopenmp -O2 -o ifclause if-clauseModificado.c
[marta@marta-PC Ejercicio1]$ ./ifclause 3 10
trhead 0 suma de a[0] = 0 sumalocal = 0
trhead 0 suma de a[1] = 1 sumalocal = 1
trhead 0 suma de a[2] = 2 sumalocal = 3
thread master = 0 imprime suma = 3
[marta@marta-PC Ejercicio1]$ ./ifclause 5 9
trhead 2 suma de a[2] = 2 sumalocal = 2
trhead 1 suma de a[1] = 1 sumalocal = 1
trhead 4 suma de a[4] = 4 sumalocal = 4
trhead 0 suma de a[0] = 0 sumalocal = 0
trhead 3 suma de a[3] = 3 sumalocal = 3
thread master = 0 imprime suma = 10
[marta@marta-PC Ejercicio1]$ ./ifclause 10 5
trhead 2 suma de a[4] = 4 sumalocal = 4
trhead 2 suma de a[5] = 5 sumalocal = 9
trhead 1 suma de a[2] = 2 sumalocal = 2
trhead 1 suma de a[3] = 3 sumalocal = 5
trhead 0 suma de a[0] = 0 sumalocal = 0
trhead 0 suma de a[1] = 1 sumalocal = 1
trhead 4 suma de a[8] = 8 sumalocal = 8
trhead 4 suma de a[9] = 9 sumalocal = 17
trhead 3 suma de a[6] = 6 sumalocal = 6
trhead 3 suma de a[7] = 7 sumalocal = 13
thread master = 0 imprime suma = 45
[marta@marta-PC Ejercicio1]$
```

RESPUESTA: **if** ahorra la creación innecesaria de hebras cuando tenemos pocas iteraciones y, no sale rentable perder tiempo creando, sincronizando y destruyendo hebras. Por ejemplo, como se ve en la captura de pantalla, cuando sólo tenemos 3 iteraciones, la hebra master se encarga de ellas. Cuando ya tenemos un número mayor, sí que se hace de forma paralela. Con **num_threads** podemos decidir las hebras que se crearán, si pasamos este número por consola nos ahorramos tener que recompilar el programa cada vez que queramos cambiar dicho número.

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

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

RESPUESTA: La principal diferencia se ve en el reparto de las iteraciones: con `static` todas las hebras hacen prácticamente las mismas iteraciones en `round-robin`. Con `dynamic` el orden y el reparto no se puede saber, lo único que se puede saber es que como mínimo cada hebra hará *chunk* iteraciones. Lo mismo pasa con `guided`, con la diferencia de que las iteraciones están más “equilibradas” entre las hebras y el número de iteraciones que hace cada hebra no es múltiplo de *chunk*.

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?

CÓDIGO FUENTE: `scheduled-clauseModificado.c`

```
#pragma omp parallel
{
    #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic, 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, a[i], suma);
    }
    if(omp_get_thread_num() == 0) {
        printf("Dentro de 'parallel':\n");
        printf("dyn-var: %d\n", omp_get_dynamic());
        printf("nthreads-var: %d\n", omp_get_max_threads());
        printf("thread-limit-var: %d\n", omp_get_thread_limit());
        omp_sched_t schedule_type;
        int chunk_size;
        omp_get_schedule(&schedule_type, &chunk_size);
        printf("run-sched-var:\n");
        if (schedule_type == omp_sched_static) printf("\tomp_sched_static\n");
        else if (schedule_type == omp_sched_dynamic) printf("\tomp_sched_dynamic\n");
        else if (schedule_type == omp_sched_guided) printf("\tomp_sched_guided\n");
        else /*if (schedule_type == omp_sched_auto)*/ printf("\tomp_sched_auto\n");
        printf("\tchunk: %d\n", chunk_size);
    }
}
printf("Fuera de 'parallel' suma = %d\n", suma);
printf("dyn-var: %d\n", omp_get_dynamic());
printf("nthreads-var: %d\n", omp_get_max_threads());
printf("thread-limit-var: %d\n", omp_get_thread_limit());
omp_sched_t schedule_type;
int chunk_size;
omp_get_schedule(&schedule_type, &chunk_size);
printf("run-sched-var:\n");
if (schedule_type == omp_sched_static) printf("\tomp_sched_static\n");
else if (schedule_type == omp_sched_dynamic) printf("\tomp_sched_dynamic\n");
else if (schedule_type == omp_sched_guided) printf("\tomp_sched_guided\n");
else /*if (schedule_type == omp_sched_auto)*/ printf("\tomp_sched_auto\n");
printf("\tchunk: %d\n", chunk_size);

```

CAPTURAS DE PANTALLA:

Valores iniciales:

```

Dentro de 'parallel':
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_dynamic
    chunk: 1
Fuera de 'parallel' suma = 99
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_dynamic
    chunk: 1

```

Cambiando variables desde el shell:

```

[marta@marta-PC Ejercicio3]$ export OMP_SCHEDULE="static,2"
[marta@marta-PC Ejercicio3]$ ./sched_modificado 5 1
thread 2 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 3 suma a[0] suma=0
thread 1 suma a[2] suma=2
thread 0 suma a[3] suma=3
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 2
Fuera de 'parallel' suma = 5
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 2
[marta@marta-PC Ejercicio3]$

[marta@marta-PC Ejercicio3]$ export OMP_NUM_THREADS=17
[marta@marta-PC Ejercicio3]$ ./sched_modificado 5 1
thread 4 suma a[1] suma=1
thread 1 suma a[0] suma=0
thread 11 suma a[4] suma=4
thread 8 suma a[2] suma=2
thread 3 suma a[3] suma=3
Dentro de 'parallel':
dyn-var: 0
nthreads-var: 17
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 2
Fuera de 'parallel' suma = 4
dyn-var: 0
nthreads-var: 17
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 2
[marta@marta-PC Ejercicio3]$

```

```
[marta@marta-PC Ejercicio3]$ export OMP_DYNAMIC=TRUE
[marta@marta-PC Ejercicio3]$ ./sched_modificado 5 1
thread 0 suma a[1] suma=1
thread 0 suma a[4] suma=4
thread 3 suma a[2] suma=2
thread 1 suma a[3] suma=3
thread 2 suma a[0] suma=0
Dentro de 'parallel':
dyn-var: 1
nthreads-var: 17
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 2
Fuera de 'parallel' suma = 5
dyn-var: 1
nthreads-var: 17
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_static
    chunk: 2
[marta@marta-PC Ejercicio3]$
```

```
[marta@marta-PC Ejercicio3]$ export OMP_THREAD_LIMIT=2
[marta@marta-PC Ejercicio3]$ ./sched_modificado 5 1
thread 0 suma a[0] suma=0
thread 1 suma a[1] suma=1
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=4
thread 0 suma a[2] suma=2
Dentro de 'parallel':
dyn-var: 1
nthreads-var: 17
thread-limit-var: 2
run-sched-var:
    omp_sched_static
    chunk: 2
Fuera de 'parallel' suma = 8
dyn-var: 1
nthreads-var: 17
thread-limit-var: 2
run-sched-var:
    omp_sched_static
    chunk: 2
[marta@marta-PC Ejercicio3]$
```

RESPUESTA: Se imprimen los mismos valores tanto dentro como fuera de parallel: los valores que hemos dicho modificando las variables de entorno desde el shell.

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.

CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

```
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(): ");
if (omp_in_parallel()) printf("True\n");
else printf("False\n");
```

CAPTURAS DE PANTALLA:

```
[marta@marta-PC Ejercicio4]$ g++ -fopenmp -O2 -o sched_mod1 scheduled_clauseModificado1.c
[marta@marta-PC Ejercicio4]$ ./sched_mod1 5 3
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=2
thread 2 suma a[3] suma=3
thread 2 suma a[4] suma=4
Dentro de 'parallel':
omp_get_num_threads(): 4
omp_get_num_procs(): 4
omp_in_parallel(): True
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_dynamic
    chunk: 1
Fuera de 'parallel' suma = 7
omp_get_num_threads(): 1
omp_get_num_procs(): 4
omp_in_parallel(): False
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var:
    omp_sched_dynamic
    chunk: 1
[marta@marta-PC Ejercicio4]$
```


RESPUESTA: De las nuevas variables de este ejercicio, sólo se mantiene constante `omp_get_num_procs()`. El resto (`omp_get_num_threads()`, `omp_in_parallel()`) varían según estemos dentro o fuera de la región paralela.

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.

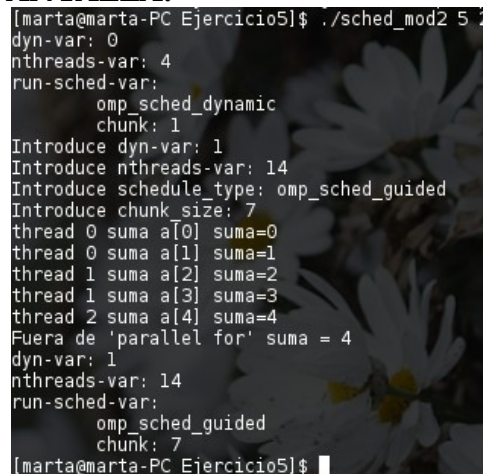
CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```
int set_dyn;
do {
    printf("Introduce dyn-var: ");
    scanf("%i",&set_dyn);
    omp_set_dynamic(set_dyn);
} while(set_dyn > 1);

int n_threads;
printf("Introduce nthreads-var: ");
scanf("%i",&n_threads);
omp_set_num_threads(n_threads);

char sched_t[20];
printf("Introduce schedule_type: ");
scanf("%s",sched_t);
printf("Introduce chunk_size: ");
scanf("%i",&chunk_size);
if (strcmp(sched_t, "omp_sched_static") == 0) schedule_type = omp_sched_static;
else if (strcmp(sched_t, "omp_sched_dynamic") == 0) schedule_type = omp_sched_dynamic;
else if (strcmp(sched_t, "omp_sched_guided") == 0) schedule_type = omp_sched_guided;
else /*if (sched_t == "omp_sched_auto")*/ schedule_type = omp_sched_auto;
omp_set_schedule(schedule_type, chunk_size);
```

CAPTURAS DE PANTALLA:



```
[marta@marta-PC Ejercicio5]$ ./sched_mod2 5 2
dyn-var: 0
nthreads-var: 4
run-sched-var:
    omp_sched_dynamic
    chunk: 1
Introduce dyn-var: 1
Introduce nthreads-var: 14
Introduce schedule_type: omp_sched_guided
Introduce chunk_size: 7
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=3
thread 2 suma a[4] suma=4
Fuera de 'parallel for' suma = 4
dyn-var: 1
nthreads-var: 14
run-sched-var:
    omp_sched_guided
    chunk: 7
[marta@marta-PC Ejercicio5]$
```

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector.

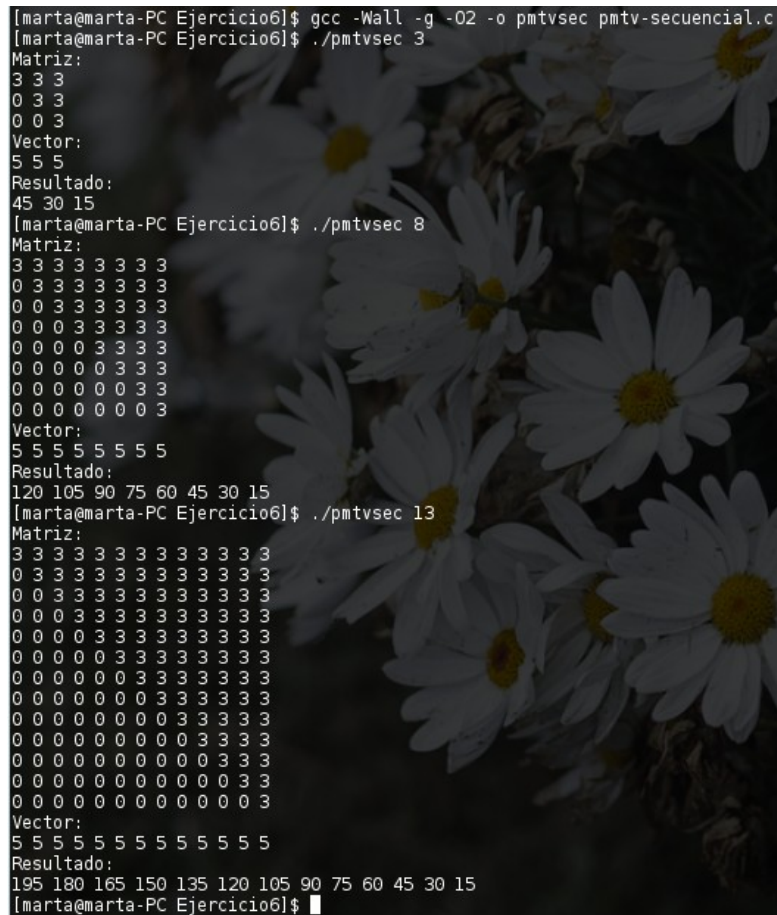
NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre

la primera y última componente del resultado antes de que termine el programa.

CÓDIGO FUENTE: pmtv-secuencial.c

```
// OBTENCION DEL VECTOR SOLUCION
for (i=0; i<N; i++)
    for (j=i; j<N; j++)
        sol[i] += matriz[i][j] * vector[j];
// cogemos la ultima componente del vector
sol[N-1] = matriz[N-1][N-1] * vector[N-1];
```

CAPTURAS DE PANTALLA:



```
[marta@marta-PC Ejercicio6]$ gcc -Wall -g -O2 -o pmtvsec pmtv-secuencial.c
[marta@marta-PC Ejercicio6]$ ./pmtvsec 3
Matriz:
3 3 3
0 3 3
0 0 3
Vector:
5 5 5
Resultado:
45 30 15
[marta@marta-PC Ejercicio6]$ ./pmtvsec 8
Matriz:
3 3 3 3 3 3 3 3
0 3 3 3 3 3 3 3
0 0 3 3 3 3 3 3
0 0 0 3 3 3 3 3
0 0 0 0 3 3 3 3
0 0 0 0 0 3 3 3
0 0 0 0 0 0 3 3
0 0 0 0 0 0 0 3
Vector:
5 5 5 5 5 5 5 5
Resultado:
120 105 90 75 60 45 30 15
[marta@marta-PC Ejercicio6]$ ./pmtvsec 13
Matriz:
3 3 3 3 3 3 3 3 3 3 3 3 3
0 3 3 3 3 3 3 3 3 3 3 3
0 0 3 3 3 3 3 3 3 3 3 3
0 0 0 3 3 3 3 3 3 3 3 3
0 0 0 0 3 3 3 3 3 3 3 3
0 0 0 0 0 3 3 3 3 3 3 3
0 0 0 0 0 0 3 3 3 3 3 3
0 0 0 0 0 0 0 3 3 3 3 3
0 0 0 0 0 0 0 0 3 3 3 3
0 0 0 0 0 0 0 0 0 3 3 3
0 0 0 0 0 0 0 0 0 0 3 3
0 0 0 0 0 0 0 0 0 0 0 3
Vector:
5 5 5 5 5 5 5 5 5 5 5 5
Resultado:
195 180 165 150 135 120 105 90 75 60 45 30 15
[marta@marta-PC Ejercicio6]$
```

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. 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 que multiplica una matriz triangular por un vector con las alternativas de planificación static, dynamic y guided para chunk de 2, 32, 64, 1024 y el chunk por defecto para la alternativa. No use vectores mayores de 32768 componentes ni menores de 4096 componentes. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 con los tiempos obtenidos, ponga en la tabla el número de threads que utilizan las ejecuciones. Representar el tiempo para static, dynamic y guided en

función del tamaño del chunk en una gráfica. Rellenar la tabla y realizar la gráfica también para el PC local. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué.

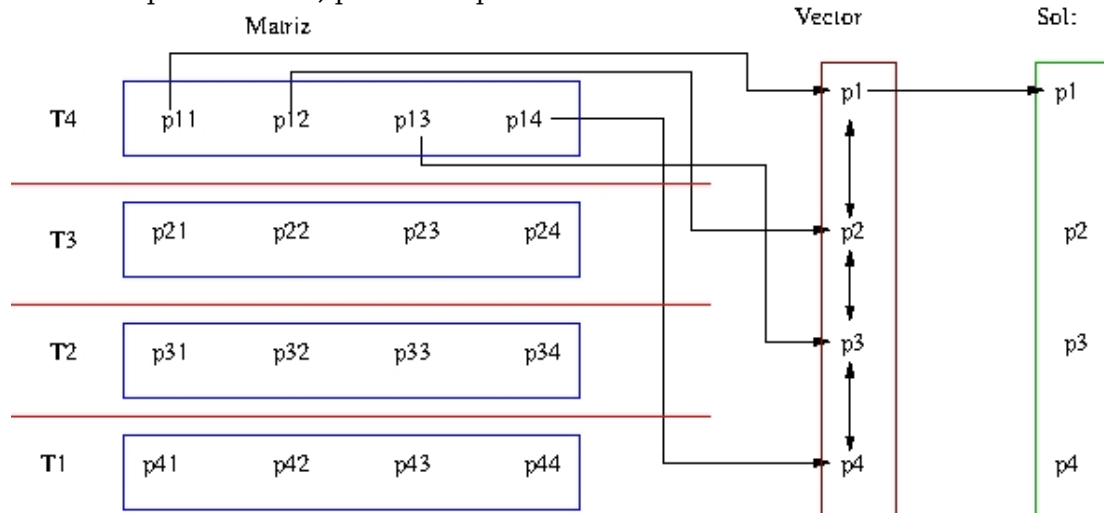
RESPUESTA: La dinámica, porque es con la que obtenemos tiempos de ejecución más bajos. La carga adicional de hacer el reparto del trabajo en tiempo de ejecución se compensa al hacer que cada hebra vaya cogiendo iteraciones conforme va terminando.

CÓDIGO FUENTE: pmtv-OpenMP.c

```
// OBTENCION DEL VECTOR SOLUCION
// con schedule(runtime) el usuario podra escoger la
// planificacion que quiera con OMP_SCHEDULE
#pragma omp parallel for private(j) schedule(runtime)
for (i=0; i<N; i++)
    for (j=i; j<N; j++)
        sol[i] += matriz[i][j] * vector[j];
// cogemos la ultima componente del vector
sol[N-1] = matriz[N-1][N-1] * vector[N-1];
```

DESCOMPOSICIÓN DE DOMINIO:

Descomponemos la matriz en filas. Para mayor claridad sólo se han puesto flechas para una fila, pero son aplicables a todas las demás.

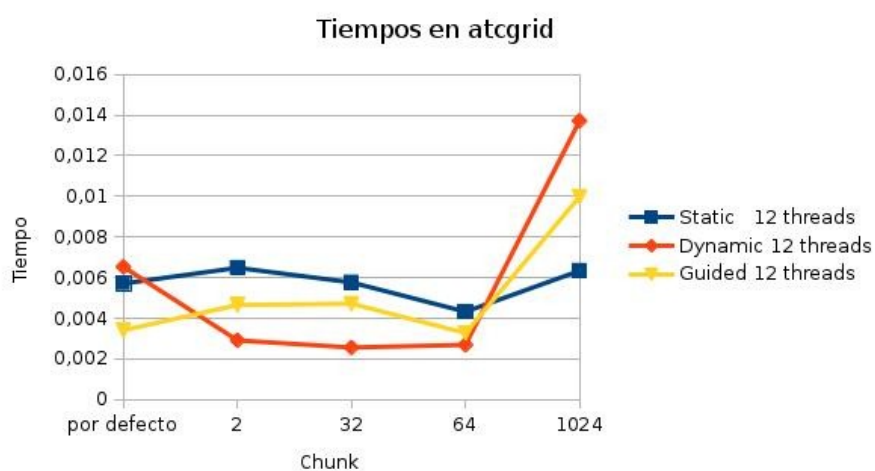


CAPTURAS DE PANTALLA:

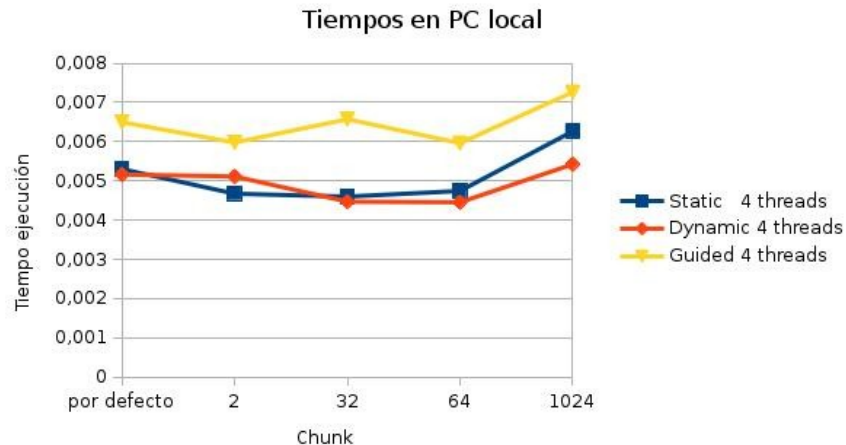
```
[marta@marta-PC Ejercicio7]$ ./pmtvomp 3
Matriz:
3 3 3
0 3 3
0 0 3
Vector:
5 5 5
Resultado:
45 30 15
[marta@marta-PC Ejercicio7]$ ./pmtvomp 10
Matriz:
3 3 3 3 3 3 3 3 3
0 3 3 3 3 3 3 3 3
0 0 3 3 3 3 3 3 3
0 0 0 3 3 3 3 3 3
0 0 0 0 3 3 3 3 3
0 0 0 0 0 3 3 3 3
0 0 0 0 0 0 3 3 3
0 0 0 0 0 0 0 3 3
0 0 0 0 0 0 0 0 3
0 0 0 0 0 0 0 0 0
Vector:
5 5 5 5 5 5 5 5 5
Resultado:
150 135 120 105 90 75 60 45 30 15
[marta@marta-PC Ejercicio7]$
```


TABLA RESULTADOS Y GRÁFICA ATCGRID**Tabla 3** .Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
por defecto	0,005707981	0,006523161	0,003408467
2	0,006481320	0,002911142	0,004651800
32	0,005762273	0,002558213	0,004715245
64	0,004319906	0,002688871	0,003270651
1024	0,006335141	0,013714543	0,009985434

**TABLA RESULTADOS Y GRÁFICA PC LOCAL****Tabla 4** .Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static 4 threads	Dynamic 4 threads	Guided 4 threads
por defecto	0,005300160	0,005162950	0,006496364
2	0,004673978	0,005112459	0,005977970
32	0,004599149	0,004470883	0,006573172
64	0,004744181	0,004453998	0,005962719
1024	0,006265302	0,005424586	0,007264495



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

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

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.

CÓDIGO FUENTE: pmm-secuencial.c

```
// Multiplicacion
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        for (k=0; k<n; k++)
            a[i][j] += b[i][k] * c[k][j];
```

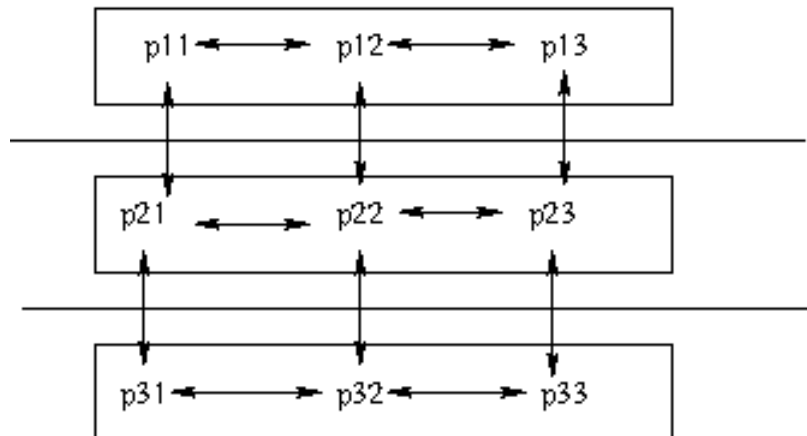
CAPTURAS DE PANTALLA:

```
[marta@marta-PC Ejercicio8]$ ./pmmsec 3
M1:
1 1 1
2 2 2
3 3 3
M2:
1 2 3
1 2 3
1 2 3
Sol:
3 6 9
6 12 18
9 18 27
```

```
[marta@marta-PC Ejercicio8]$ ./pmmsec 8
M1:
1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8
M2:
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
Sol:
8 16 24 32 40 48 56 64
16 32 48 64 80 96 112 128
24 48 72 96 120 144 168 192
32 64 96 128 160 192 224 256
40 80 120 160 200 240 280 320
48 96 144 192 240 288 336 384
56 112 168 224 280 336 392 448
64 128 192 256 320 384 448 512
[marta@marta-PC Ejercicio8]$
```

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 paralelizar 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: la descomposición se hará por filas, es decir, cada thread se encargará de una fila de la matriz. La comunicación entre threads se hará por columnas, pues para calcular la multiplicación necesitamos los datos de una misma columna.



CÓDIGO FUENTE: pmm-OpenMP.c

```
// Inicializacion
#pragma omp parallel for private(j)
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        a[i][j] = 0;
        b[i][j] = i+1;
        c[i][j] = j+1;
    }
}
// Multiplicacion
#pragma omp parallel for private(k,j)
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        for (k=0; k<n; k++)
            a[i][j] += b[i][k] * c[k][j];
```

CAPTURAS DE PANTALLA:

```
[marta@marta-PC Ejercicio9]$ ./pmmomp 3
M1:
1 1 1
2 2 2
3 3 3
M2:
1 2 3
1 2 3
1 2 3
Sol:
3 6 9
6 12 18
9 18 27
[marta@marta-PC Ejercicio9]$ ./pmmomp 7
M1:
1 1 1 1 1 1 1
2 2 2 2 2 2 2
3 3 3 3 3 3 3
4 4 4 4 4 4 4
5 5 5 5 5 5 5
6 6 6 6 6 6 6
7 7 7 7 7 7 7
M2:
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
Sol:
7 14 21 28 35 42 49
14 28 42 56 70 84 98
21 42 63 84 105 126 147
28 56 84 112 140 168 196
35 70 105 140 175 210 245
42 84 126 168 210 252 294
49 98 147 196 245 294 343
[marta@marta-PC Ejercicio9]$
```

```
[marta@marta-PC Ejercicio8]$ ./pmmsec 3
M1:
1 1 1
2 2 2
3 3 3
M2:
1 2 3
1 2 3
1 2 3
Sol:
3 6 9
6 12 18
9 18 27
[marta@marta-PC Ejercicio8]$ ./pmmsec 7
M1:
1 1 1 1 1 1 1
2 2 2 2 2 2 2
3 3 3 3 3 3 3
4 4 4 4 4 4 4
5 5 5 5 5 5 5
6 6 6 6 6 6 6
7 7 7 7 7 7 7
M2:
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
Sol:
7 14 21 28 35 42 49
14 28 42 56 70 84 98
21 42 63 84 105 126 147
28 56 84 112 140 168 196
35 70 105 140 175 210 245
42 84 126 168 210 252 294
49 98 147 196 245 294 343
[marta@marta-PC Ejercicio8]$
```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para tres tamaños de las matrices. Debe recordar usar -O2 al compilar. 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. Consulte la Lección 6/Tema 2.

ESTUDIO DE ESCALABILIDAD EN ATCGRID:



	secuencial	dos cores	cuatro cores	ocho cores	doce cores
100	0,001745939	0,001545519	0,00088594	0,000605452	0,000269204
500	0,355614865	0,199652872	0,103354527	0,051759665	0,0339903
1000	9,53643889	4,925662981	2,682087484	1,364372716	0,910306485
1500	34,002600375	17,389563134	9,036692597	4,722099363	3,151529519

La ganancia en prestaciones, se nota más para tamaños grandes que para pequeños. Por tanto vamos a estudiarla para 500, 1000 y 1500.

Para un tamaño de matriz 500, la ganancia en prestaciones sería:

1. En dos cores:

$$S(2) = \frac{0,355614865}{0,199652872} = 1.7811657875875685$$

es decir, usando dos procesadores conseguimos una ganancia de casi el doble.

2. En cuatro cores:

$$S(4) = \frac{0,355614865}{0,103354527} = 3.4407284840072849$$

usando cuatro procesadores, conseguimos un tiempo de ejecución casi cuatro veces mayor.

3. En ocho cores:

$$S(8) = \frac{0,355614865}{0,051759665} = 6.8705016734555759$$

usando ocho cores conseguimos una ganancia casi siete veces mayor. Al aumentar el número de cores, la sobrecarga de comunicación y sincronización se hace mayor.

4. En doce cores:

$$S(12) = \frac{0,355614865}{0,033990300} = 10.4622455524075986$$

en este caso, la ganancia es de 10 veces mayor. Como ya dije antes, cuanto mayor es el número de cores, más cuesta conseguir una ganancia igual a dicho número.

Para un tamaño de matriz 1000, la ganancia en prestaciones sería:

1. En dos cores:

$$S(2) = \frac{9,536438890}{4,925662981} = 1.9360721443560736$$

la ganancia es de casi el doble en este caso.

2. Con cuatro cores:

$$S(4) = \frac{9,536438890}{2,682087484} = 3.555603218347519$$

en este caso, se queda a medio camino entre ser tres veces mayor y cuatro veces mayor.

3. Con ocho cores:

$$S(8) = \frac{9,536438890}{1,364372716} = 6.9896141854539988$$

en este caso, obtenemos un tiempo de ejecución casi siete veces menor.

4. Con doce cores:

$$S(12) = \frac{9,536438890}{0,910306485} = 10.4760748683450278$$

obtenemos en este caso una ganancia bastante parecida a la que obteníamos con tamaño 500. Pero si nos fijamos en la ganancia con menor número de procesadores, ésta era más alta con tamaño 1000 que con tamaño 500.

Por último, para un tamaño de matriz 1500, la ganancia en prestaciones sería:

1. Con dos cores:

$$S(2) = \frac{34,002600375}{17,389563134} = 1.9553452903321223$$

En este caso obtenemos una ganancia muy parecida a la obtenida con tamaño 1000

2. Con cuatro cores:

$$S(4) = \frac{34,002600375}{9,036692597} = 3.7627262419314981$$

En este caso obtenemos una ganancia ligeramente superior a la obtenida con tamaño 1000

3. Con ocho cores:

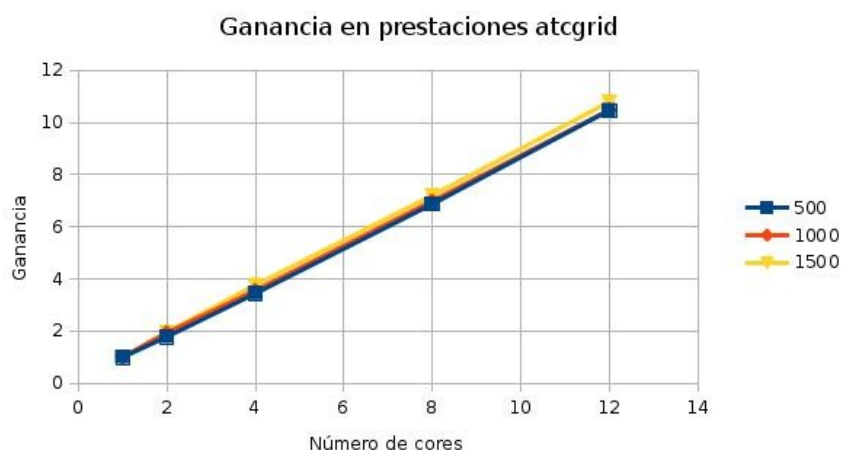
$$S(8) = \frac{34,002600375}{4,722099363} = 7.2007380110268976$$

Esta vez, si podemos decir que el tiempo de ejecución con 8 cores es 7 veces menor.

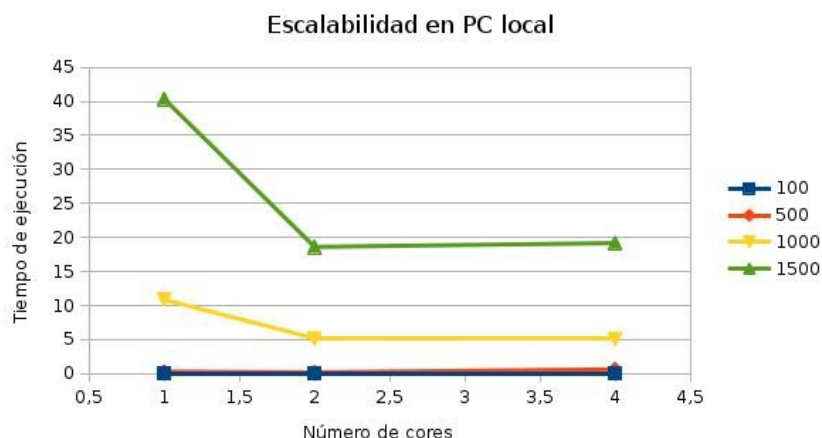
4. Con doce cores:

$$S(12) = \frac{34,002600375}{3,151529519} = 10.7892374702519802$$

La ganancia en prestaciones es casi once veces mayor.



Cuanto mayor es el número de cores, mayor es la ganancia que obtenemos.

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

	secuencial	dos cores	cuatro cores
100	0,003610504	0,002065647	0,001486125
500	0,277564385	0,178290718	0,601877455
1000	10,855511043	5,116787663	5,084231102
1500	40,350126244	18,610337987	19,171313684

En este caso, la escalabilidad para dos y cuatro cores es muy parecida pero hay una gran diferencia con la ejecución en secuencial.

Escalabilidad para tamaño 500:

1. Con dos cores:

$$S(2) = \frac{0,277564385}{0,178290718} = 1.5568078255201148$$

La velocidad con dos procesadores es 1,5 veces mayor. Hemos conseguido una ganancia menor que en atcgrid.

2. Con cuatro cores:

$$S(4) = \frac{0,277564385}{0,601877455} = 0.4611642830183762$$

Esta medida ha debido de ser errónea (puede deberse a una interrupción del sistema operativo) pues es imposible obtener un tiempo de ejecución el doble de lento con cuatro procesadores.

Escalabilidad para tamaño 1000:

1. Con dos cores:

$$S(2) = \frac{10,855511043}{5,116787663} = 2.121548080155305$$

La velocidad con dos procesadores es mayor al doble. Esta cifra es también mayor a la obtenida en atcgrid.

2. Con cuatro cores:

$$S(4) = \frac{10,855511043}{5,084231102} = 2.1351332827356596$$

Si la anterior medida me parecía un poco excesiva, esta al contrario. Lo normal hubiese sido acercarse más a tres y no quedarse en una ganancia de dos. Puede ser, al igual que antes por una interrupción del sistema operativo.

Escalabilidad para tamaño 1500:

1. Con dos cores:

$$S(2) = \frac{40,350126244}{18,610337987} = 2.168156552137099$$

La velocidad con dos procesadores es mayor al doble. Esta cifra es también mayor a la obtenida en atcgrid.

2. Con cuatro cores:

$$S(4) = \frac{40,350126244}{19,171313684} = 2.1047136836363707$$

Volvemos al mismo caso anterior.

El hecho de que con cuatro cores salgan resultados malos, o no tan buenos, puede deberse también a la **reducción debida a la sobrecarga**. Lo que no me termina de cuadrar es por qué en atcgrid no se nota esta reducción mientras que en mi pc sí.

