

## Arquitectura de Computadores (AC)

### Cuaderno de prácticas.

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

Estudiante (nombre y apellidos): Ivan Ortega Alba

Grupo de prácticas: C2

Fecha de entrega: 20/5/2015

Fecha evaluación en clase: 21/05/2015

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`

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

int main(int argc, char **argv)
{
    int i, n=20, tid, x;
    int a[n], suma=0, sumalocal;
    if(argc < 3) {
        fprintf(stderr, "[ERROR]-Faltan parametros: iteraciones nthreads\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;
    x = atoi(argv[2]);
    for (i=0; i<n; i++) {
        a[i] = i;
    }

#pragma omp parallel if(n>4) default(none) \
    private(sumalocal,tid) shared(a,suma,n) num_threads(x)
    //No se ejecutará en paralelo si el n < 4
{   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);
}

return(0);
}
```

### CAPTURAS DE PANTALLA:

```
MacBook-Pro-de-Ivan:SCRIPTS ivanortegaalba$ ./P3/1/nThreads 21 5
thread 0 suma de a[0]=0 sumalocal=0
thread 2 suma de a[8]=8 sumalocal=8
thread 3 suma de a[12]=12 sumalocal=12
thread 1 suma de a[4]=4 sumalocal=4
thread 4 suma de a[16]=16 sumalocal=16
thread 0 suma de a[1]=1 sumalocal=1
thread 2 suma de a[9]=9 sumalocal=17
thread 3 suma de a[13]=13 sumalocal=25
thread 1 suma de a[5]=5 sumalocal=9
thread 4 suma de a[17]=17 sumalocal=33
thread 0 suma de a[2]=2 sumalocal=3
thread 2 suma de a[10]=10 sumalocal=27
thread 3 suma de a[14]=14 sumalocal=39
thread 1 suma de a[6]=6 sumalocal=15
thread 4 suma de a[18]=18 sumalocal=51
thread 0 suma de a[3]=3 sumalocal=6
thread 2 suma de a[11]=11 sumalocal=38
thread 3 suma de a[15]=15 sumalocal=54
thread 1 suma de a[7]=7 sumalocal=22
thread 4 suma de a[19]=19 sumalocal=70
thread master=0 imprime suma=190

MacBook-Pro-de-Ivan:SCRIPTS ivanortegaalba$ ./P3/1/nThreads 21 2
thread 0 suma de a[0]=0 sumalocal=0
thread 1 suma de a[10]=10 sumalocal=10
thread 0 suma de a[1]=1 sumalocal=1
thread 1 suma de a[11]=11 sumalocal=21
thread 0 suma de a[2]=2 sumalocal=3
thread 1 suma de a[12]=12 sumalocal=33
thread 0 suma de a[3]=3 sumalocal=6
thread 1 suma de a[13]=13 sumalocal=46
thread 0 suma de a[4]=4 sumalocal=10
thread 1 suma de a[14]=14 sumalocal=60
thread 0 suma de a[5]=5 sumalocal=15
thread 1 suma de a[15]=15 sumalocal=75
thread 0 suma de a[6]=6 sumalocal=21
thread 1 suma de a[16]=16 sumalocal=91
thread 0 suma de a[7]=7 sumalocal=28
thread 1 suma de a[17]=17 sumalocal=108
thread 0 suma de a[8]=8 sumalocal=36
thread 1 suma de a[18]=18 sumalocal=126
thread 0 suma de a[9]=9 sumalocal=45
thread 1 suma de a[19]=19 sumalocal=145
thread master=0 imprime suma=190
```

### RESPUESTA:

En las capturas puede verse que solo se usan las hebras que se le pasan como segundo parámetro.

El identificador demuestra el número de threads que se están usando.

También podemos verlo por el reparto del vector que se reparte por partes iguales en función al numero de hebras.

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	<i>schedule-clause.c</i>			<i>schedule-clause.c</i>			<i>schedule-clauseg.c</i>		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	1	0	0
1	1	0	0	1	0	0	1	0	0
2	0	1	0	0	1	0	1	0	0
3	1	1	0	1	1	0	1	0	0

<b>4</b>	0	0	1	0	1	1	1	0	0
<b>5</b>	1	0	1	1	1	1	1	0	0
<b>6</b>	0	1	1	0	0	1	1	0	0
<b>7</b>	1	1	1	0	0	1	1	0	0
<b>8</b>	0	0	0	0	1	0	1	1	1
<b>9</b>	1	0	0	0	1	0	0	1	1
<b>10</b>	0	1	0	1	1	0	0	1	1
<b>11</b>	1	1	0	0	1	0	0	1	1
<b>12</b>	0	0	1	0	1	0	0	1	0
<b>13</b>	1	0	1	0	1	0	1	1	0
<b>14</b>	0	1	1	0	0	0	1	1	0
<b>15</b>	1	1	1	0	0	0	0	1	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
<b>0</b>	0	0	0	1	0	0	0	0	2
<b>1</b>	1	0	0	3	0	0	0	0	2
<b>2</b>	2	1	0	2	1	0	0	0	2
<b>3</b>	3	1	0	0	1	0	0	0	2
<b>4</b>	0	2	1	1	3	1	2	3	0
<b>5</b>	1	2	1	2	3	1	2	3	0
<b>6</b>	2	3	1	3	2	1	2	3	0
<b>7</b>	3	3	1	2	2	1	1	2	0
<b>8</b>	0	0	2	2	1	3	1	2	3
<b>9</b>	1	0	2	2	1	3	1	2	3
<b>10</b>	2	1	2	2	2	3	3	1	3
<b>11</b>	3	1	2	2	2	3	3	1	3
<b>12</b>	0	2	3	2	3	2	2	3	1
<b>13</b>	1	2	3	2	3	2	1	3	1
<b>14</b>	2	3	3	2	2	2	2	0	1

15	3	3	3	2	2	2	1	0	1
----	---	---	---	---	---	---	---	---	---

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

**RESPUESTA:**

**Static:** El schedule organiza la forma de repartir las iteraciones del bucle. Con static, establece un Round Robin, y departiendo de n en n, siendo n el chunk. Es similar a un reparto de cartas, se reparte por jugador tantas cartas como determine el chunk.

**Dynamic:** No se asignan los chunk previamente, sino que se van asignando cuando van terminando las hebras. Se hace una primera asignación a las hebras disponibles, en paquetes de tamaño chunk, y la primer que termine se le asigna otro paquete de tamaño chunk. Así hasta realizarse todas las iteraciones.

**Guided:** Empezamos por el bloque más largo posible, hacemos una distribución previa igual que el static. Posteriormente, si una hebra queda esperando, se reasigna el trabajo restante previamente asignado a las otras hebras, y ahora sí que se tiene en cuenta el chunk para reasignar.

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`

```
#include <stdio.h>
#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,a[n],suma=0;
    int dyn_var_value, nthreads_var_value, thread_limit_var;
    omp_sched_t kind;
    int modifier = 0;
    int primera = 1;

    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;
    printf(" Fuera del parallel: \n");
    printf(" dyn-var-> %d ;nthreads-var->%d; thread-limit-var->%d; run-
    sched-var-kind->%d; run-sched-var-modifier->%d; \n",
    dyn_var_value, nthreads_var_value, thread_limit_var, kind, modifier);
}
```

```

    omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),
kind, modifier);
#pragma omp parallel for firstprivate(suma) \
lastprivate(suma) schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
    #pragma omp critical
    if(primeria == 1){
        primera = 0;
        omp_get_schedule(&kind, &modifier);
        printf(" Fuera del parallel: \n");
        printf(" dyn-var-> %d ;nthreads-var->%d; thread-limit-var-
>%d; run-sched-var-kind->%d; run-sched-var-modifier->%d; \n",
            omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(), kind,
            modifier);
    }
    #pragma omp atomic
    suma = suma + a[i];
    #pragma omp critical
    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);

return(0);
}

```

#### CAPTURAS DE PANTALLA:

```

MacBook-Pro-de-Ivan:3 ivanortegaalba$ ./ScheduledPrint 10 1
Fuera del parallel:
dyn-var-> 0 ;nthreads-var->4; thread-limit-var->2147483647; run-sched-var-kind-
>1546947424; run-sched-var-modifier->0;
Fuera del parallel:
dyn-var-> 0 ;nthreads-var->4; thread-limit-var->2147483647; run-sched-var-kind-
>2; run-sched-var-modifier->1;
thread 1 suma a[1]=1 suma=1
thread 3 suma a[0]=0 suma=0
thread 2 suma a[2]=2 suma=2
thread 0 suma a[3]=3 suma=3
thread 1 suma a[4]=4 suma=5
thread 3 suma a[5]=5 suma=5
thread 2 suma a[6]=6 suma=8
thread 0 suma a[7]=7 suma=10
thread 1 suma a[8]=8 suma=13
thread 3 suma a[9]=9 suma=14
Fuera de 'parallel for' suma=14

```

#### RESPUESTA:

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

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

```
#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,a[n],suma=0;
    int primera = 1;
    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;

    printf(" Fuera del parallel: \n");
    printf(" in-parallel> %s ;num-proc->%d; num-thread->%d; \n",
    omp_in_parallel(),omp_get_num_procs(),omp_get_num_threads());
    #pragma omp parallel for firstprivate(suma) \
            lastprivate(suma) schedule(dynamic,chunk)
    for (i=0; i<n; i++)
    {
        #pragma omp critical
        if (primera == 1){
            primera = 0;
            printf(" Dentro del parallel: \n");
            printf(" in-parallel-> %d ;num-proc->%d; num-thread->%d;
\n",
            omp_in_parallel(),omp_get_num_procs(),omp_get_num_threads());
        }
        #pragma omp atomic
        suma = suma + a[i];
        #pragma omp critical
        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);

    return(0);
}
```

**CAPTURAS DE PANTALLA:**

```
MacBook-Pro-de-Ivan:4 ivanortegaalba$ ./ScheduledPrint 10 2
Fuera del parallel:
in-parallel> (null) ;num-proc->4; num-thread->1;
Dentro del parallel:
in-parallel> 1 ;num-proc->4; num-thread->4;
thread 2 suma a[0]=0 suma=0
thread 1 suma a[2]=2 suma=2
thread 0 suma a[4]=4 suma=4
thread 3 suma a[6]=6 suma=6
thread 2 suma a[1]=1 suma=1
thread 1 suma a[3]=3 suma=5
thread 0 suma a[5]=5 suma=9
thread 3 suma a[7]=7 suma=13
thread 2 suma a[8]=8 suma=9
thread 2 suma a[9]=9 suma=18
Fuera de 'parallel for' suma=18
```

**RESPUESTA:**

Se obtienen valores distintos para `omp_in_parallel()`, puesto que fuera nos indica que no estamos dentro de una región paralela, y dentro que sí; para `omp_get_num_threads()`, puesto que dentro nos indica que se están usando 16 threads en la región paralela, y fuera, puesto que es código secuencial, nos indica que se están usando 1.

NOTA: `omp_get_num_threads()` devuelve el valor de la variable de entorno `OMP_THREAD_LIMIT`

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`

```
#include <stdio.h>
#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,a[n],suma=0;
    int dyn_var_value, nthreads_var_value, thread_limit_var;
    omp_sched_t kind;
    int modifier = 0;
    int primera = 1;

    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;

    printf(" Antes de la modificación: \n");
}
```

```

    printf(" dyn-var-> %d ;nthreads-var->%d; thread-limit-var->%d; run-
sched-var-kind->%d; run-sched-var-modifier->%d; \n",
    omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),
kind, modifier);

//Modificación de las variables
omp_set_num_threads(16);
omp_set_schedule((omp_sched_t) 3,1);
omp_set_dynamic(8);

#pragma omp parallel for firstprivate(suma) \
    lastprivate(suma) schedule(static,chunk)
for (i=0; i<n; i++)
{
    #pragma omp critical
    if(primer == 1){
        primera = 0;
        omp_get_schedule(&kind, &modifier);
        printf(" Dentro del parallel: \n");
        printf(" dyn-var-> %d ;nthreads-var->%d; thread-limit-var-
>%d; run-sched-var-kind->%d; run-sched-var-modifier->%d; \n",
        omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(), kind,
modifier);
    }
    #pragma omp atomic
    suma = suma + a[i];
    #pragma omp critical
    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);

return(0);
}

```

#### CAPTURAS DE PANTALLA:

```

MacBook-Pro-de-Ivan:5 ivanortegaalba$ ./ScheduledPrint 50 8
Antes de la modificación:
dyn-var-> 0 ;nthreads-var->4; thread-limit-var->2147483647; run-sched-var-kind->156
4343136; run-sched-var-modifier->0;
Dentro del parallel:
dyn-var-> 1 ;nthreads-var->16; thread-limit-var->2147483647; run-sched-var-kind->3;
run-sched-var-modifier->1;

```

#### RESPUESTA:

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

```

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

```

```

main(int argc, char **argv) {
    int N = atoi(argv[1]);
    int i,j;

    double **m, *v1, *v2;
    m = (double **)malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        m[i] = (double *) malloc (N*sizeof(double));

    v1 = (double *) malloc (N*sizeof(double));
    v2 = (double *) malloc (N*sizeof(double));

    double start,end,elapsed;
    if(argc < 2) {
        fprintf(stderr,"Faltan argumentos\n");
        exit(-1);
    }

    // Inicialización del vector y la matriz
    for (i = 0; i < N; ++i){
        v1[i] = i + 1;
        // Matriz triangular superior
        for(j = 0; j < i; ++j){
            m[i][j] = 0;
        }
        for(j = i; j < N; ++j){
            m[i][j] = j + 1;
        }
    }
    if(N<20){
        printf("Matriz:\n");
        for(i = 0; i < N;i++){
            for(j = 0; j < N; j++)
                printf(" %g ",m[i][j]);
            printf("\n");
        }
        printf("Vector:\n");
        for(i = 0; i < N;i++){
            printf(" %g ",v1[i]);
        }
        printf("\n");
    }

    start = omp_get_wtime();

    //Multiplicamos
    for (i=0; i<N; ++i){
        for (j=i; j<N; ++j)
            v2[i] = (m[i][j] * v1[j])+v2[i];
    }

    end = omp_get_wtime();
    elapsed = end - start;
    //Imprimimos
}

```

```

printf("Vector Resultante:\n");
if(N<20)
for(i = 0; i < N;i++)
    printf("v[%d] = %g\n",i,v2[i]);
else{
    printf("v[%d] = %g\n",0,v2[0]);
    printf("v[%d] = %g\n",N-1,v2[N-1]);
}

printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",elapsed,N);
}

```

**CAPTURAS DE PANTALLA:**  
**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

```

MacBook-Pro-de-Ivan:6 ivanortegaalba$ ./PvmSecuencial 5
Matriz:
 1 2 3 4 5
 0 2 3 4 5
 0 0 3 4 5
 0 0 0 4 5
 0 0 0 0 5
Vector:
 1 2 3 4 5
Vector Resultante:
v[0] = 55
v[1] = 54
v[2] = 50
v[3] = 41
v[4] = 25
Tiempo(seg.):0.000000954      / Tamaño Vectores:5
MacBook-Pro-de-Ivan:6 ivanortegaalba$ █

```

```

MacBook-Pro-de-Ivan:6 ivanortegaalba$ ./PvmSecuencial 10000
Vector Resultante:
v[0] = 3.33383e+11
v[9999] = 1e+08
Tiempo(seg.):0.058166981      / Tamaño Vectores:10000
MacBook-Pro-de-Ivan:6 ivanortegaalba$ ./PvmSecuencial 15000
Vector Resultante:
v[0] = 1.12511e+12
v[14999] = 2.25e+08
Tiempo(seg.):0.195763111      / Tamaño Vectores:15000
MacBook-Pro-de-Ivan:6 ivanortegaalba$ █

```

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, 64, 128, 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:**

**CÓDIGO FUENTE:** pmtv-OpenMP.c

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

main(int argc, char **argv) {
    int N = atoi(argv[1]);
    int i,j;
    omp_sched_t kind;
    int modifier;
    double **m, *v1, *v2;
    m = (double **)malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        m[i] = (double *) malloc (N*sizeof(double));

    v1 = (double *) malloc (N*sizeof(double));
    v2 = (double *) malloc (N*sizeof(double));

    double start,end,elapsed;
    if(argc < 2) {
        fprintf(stderr,"Faltan argumentos\n");
        exit(-1);
    }

    // Inicialización del vector y la matriz
    for (i = 0; i < N; ++i){
        v1[i] = i + 1;
        // Matriz triangular superior
        for(j = 0; j < i; ++j){
            m[i][j] = 0;
        }
        for(j = i; j < N; ++j){
            m[i][j] = j + 1;
        }
    }
    if(N<20){
        printf("Matriz:\n");
        for(i = 0; i < N;i++){
            for(j = 0; j < N; j++)
                printf(" %g ",m[i][j]);
            printf("\n");
        }
        printf("Vector:\n");
        for(i = 0; i < N;i++){
            printf(" %g ",v1[i]);
        }
        printf("\n");
    }
    //Modificaci&on de las variables
    omp_set_num_threads(omp_get_num_procs());
    omp_get_schedule(&kind, &modifier);
    printf(" Dentro del parallel: \n");
}
```

```

        printf(" dyn-var-> %d ;nthreads-var->%d; thread-limit-var->%d;
run-sched-var-kind->%d; run-sched-var-modifier->%d; \n",
omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(), kind,
modifier);
    start = omp_get_wtime();

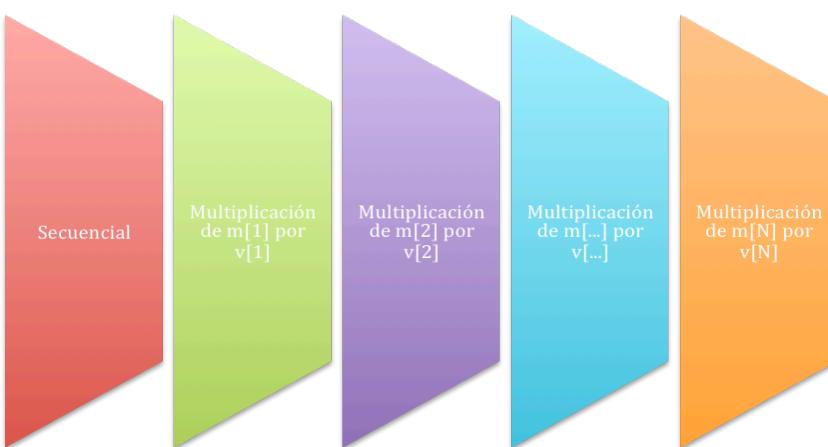
//Multiplicamos
#pragma omp parallel for schedule(runtime) private (j)
for (i=0; i<N; ++i){
    for (j=i; j<N; ++j)
    #pragma omp atomic
    v2[i] = (m[i][j] * v1[j])+v2[i];
}

end = omp_get_wtime();
elapsed = end - start;
//Imprimimos
printf("Vector Resultante:\n");
if(N<20)
for(i = 0; i < N;i++)
    printf("v[%d] = %g\n",i,v2[i]);
else{
    printf("v[%d] = %g\n",0,v2[0]);
    printf("v[%d] = %g\n",N-1,v2[N-1]);
}

printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",elapsed,N);
}

```

### DESCOMPOSICIÓN DE DOMINIO:



Se asignaran x filas a una hebra en función del Chunk

Se le asignarán de una forma un otra en función al algoritmo.

- Static: N/nhebras filas por hebra
- Dynamic: Chuck filas por hebra en función a la que termine primero.
- Guided: N/nhebras filas a cada hebra, y si termina otra antes, se le asignan chuck hebras.

### CAPTURAS DE PANTALLA:

**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

```

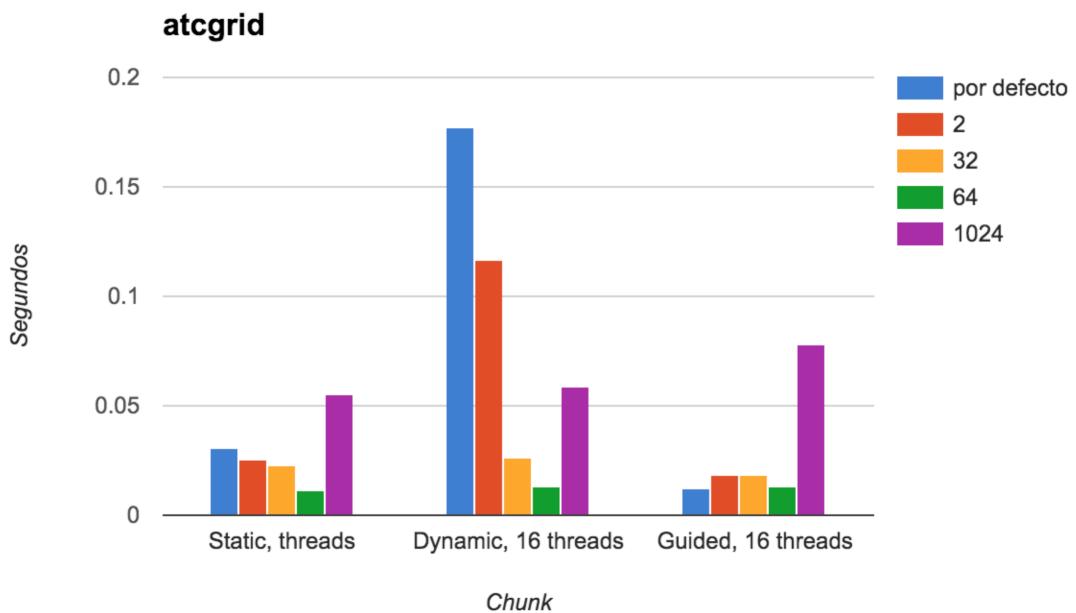
MacBook-Pro-de-Ivan:7 ivanortegaalba$ ./PvmParaleloSchedule 20
Dentro del parallel:
dyn-var-> 0 ;nthreads-var->4; thread-limit-var->2147483647; run-sched-var-kind->2;
run-sched-var-modifier->4;
Vector Resultante:
v[0] = 2870
v[19] = 400
Tiempo(seg.):0.000148058      / Tamaño Vectores:20
MacBook-Pro-de-Ivan:7 ivanortegaalba$ export OMP_SCHEDULE="static,2"
MacBook-Pro-de-Ivan:7 ivanortegaalba$ ./PvmParaleloSchedule 20
Dentro del parallel:
dyn-var-> 0 ;nthreads-var->4; thread-limit-var->2147483647; run-sched-var-kind->1;
run-sched-var-modifier->2;
Vector Resultante:
v[0] = 2870
v[19] = 400
Tiempo(seg.):0.000209093      / Tamaño Vectores:20

```

**TABLA RESULTADOS Y GRÁFICA ATCGRID:**

**Tabla 3 .** Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector

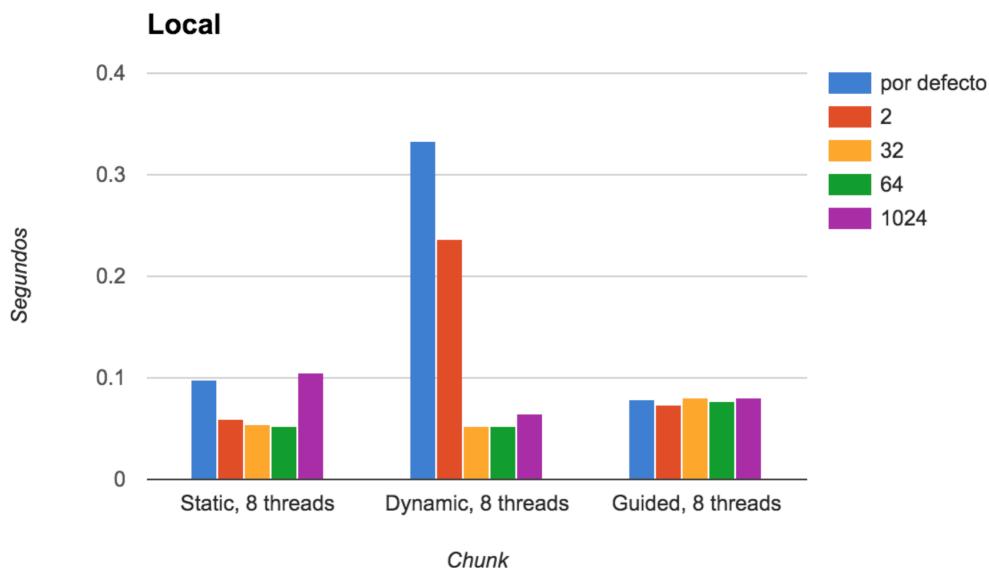
Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
por defecto	0.03067	0.177093	0.012655
2	0.02516	0.116978	0.01813
32	0.023036	0.026304	0.01807
64	0.011588	0.013049	0.012881
1024	0.055436	0.058654	0.077844



**TABLA RESULTADOS Y GRÁFICA PC LOCAL:**

**Tabla 4 .** Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector

Chunk	Static 8 threads	Dynamic 8 threads	Guided 8 threads
por defecto	0.097698927	0.33392787	0.078240156
2	0.060452938	0.236817837	0.072952986
32	0.055069923	0.053452015	0.080199003
64	0.05338192	0.053275108	0.077713013
1024	0.10497117	0.065028906	0.080174923



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.

**CÓDIGO FUENTE:** pmm-secuencial.c

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

main(int argc, char **argv) {
    int N = atoi(argv[1]);
    int i,j,k;
    omp_sched_t kind;
    int modifier;
    double start=0,end=0,elapsed = 0;

    //Matriz 1
    double **a;
    a = (double **) malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        a[i] = (double *) malloc (N*sizeof(double));
    //Matriz 2
    double **b;
    b = (double **)malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        b[i] = (double *) malloc (N*sizeof(double));
    //Matriz Resultado
    double **resultado;
    resultado = (double **)malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        resultado[i] = (double *) malloc (N*sizeof(double));
    for (i=0;i<N;i++)
        for (j=0;j<N;j++)
            for (k=0;k<N;k++)
                resultado[i][j] += a[i][k]*b[k][j];
    printf("Componente (0,0) = %f\n", resultado[0][0]);
    printf("Componente (%d,%d) = %f\n", N-1, N-1, resultado[N-1][N-1]);
}
```

```

resultado[i] = (double *) malloc (N*sizeof(double));

if(argc < 2) {
    fprintf(stderr,"Faltan argumentos\n");
    exit(-1);
}

for (i=0; i<N; ++i)
    for (j=0; j<N; ++j){
        resultado[i][j] = 0;
        a[i][j] = j+1;
        b[i][j] = j+1;
    }

//Multiplicamos
start = omp_get_wtime();
for (i=0; i<N; ++i){
    for (j=0; j<N; ++j)
        for (k=0; k<N; ++k)
            resultado[i][j] += a[i][k]*b[k][j];
}
end = omp_get_wtime();
elapsed = end - start;

//Imprimimos
printf("resultado[0][0] = %g\n",resultado[0][0]);
printf("resultado[N-1][N-1] = %g\n",resultado[N-1][N-1]);
printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\n",elapsed,N);

}

```

**CAPTURAS DE PANTALLA:**  
**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

```

cvi067065:8 ivanortegaalba$ ./PmmSecuencial 500
resultado[0][0] = 125250
resultado[N-1][N-1] = 6.2625e+07
Tiempo(seg.):0.322519064      / Tamaño Matrices:500
cvi067065:8 ivanortegaalba$ ./PmmSecuencial 800
resultado[0][0] = 320400
resultado[N-1][N-1] = 2.5632e+08
Tiempo(seg.):1.574472904      / Tamaño Matrices:800
cvi067065:8 ivanortegaalba$ █

```

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:**

**CÓDIGO FUENTE:** pmm-OpenMP.c

```

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

```

```
#include <omp.h>

main(int argc, char **argv) {
    int N = atoi(argv[1]);
    int i,j,k;
    omp_sched_t kind;
    int modifier;
    double start=0,end=0,elapsed = 0;

    //Matriz 1
    double **a;
    a = (double **) malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        a[i] = (double *) malloc (N*sizeof(double));
    //Matriz 2
    double **b;
    b = (double **)malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        b[i] = (double *) malloc (N*sizeof(double));
    //Matriz Resulado
    double **resultado;
    resultado = (double **)malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        resultado[i] = (double *) malloc (N*sizeof(double));

    if(argc < 2) {
        fprintf(stderr,"Faltan argumentos\n");
        exit(-1);
    }

    for (i=0; i<N; ++i)
        for (j=0; j<N; ++j){
            resultado[i][j] = 0;
            a[i][j] = j+1;
            b[i][j] = j+1;
        }

    //Multiplicamos
    start = omp_get_wtime();
    #pragma omp parallel for schedule(dynamic) private(j,k)
    for (i=0; i<N; ++i){
        #pragma omp parallel for schedule(dynamic) private(k)
        for (j=0; j<N; ++j)
            for (k=0; k<N; ++k)
                #pragma omp atomic
                resultado[i][j] += a[i][k]*b[k][j];
    }
    end = omp_get_wtime();
    elapsed = end - start;

    //Imprimimos
    printf("resultado[0][0] = %g\n",resultado[0][0]);
    printf("resultado[N-1][N-1] = %g\n",resultado[N-1][N-1]);
    printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\n",elapsed,N);
}
```

}

**CAPTURAS DE PANTALLA:**  
**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

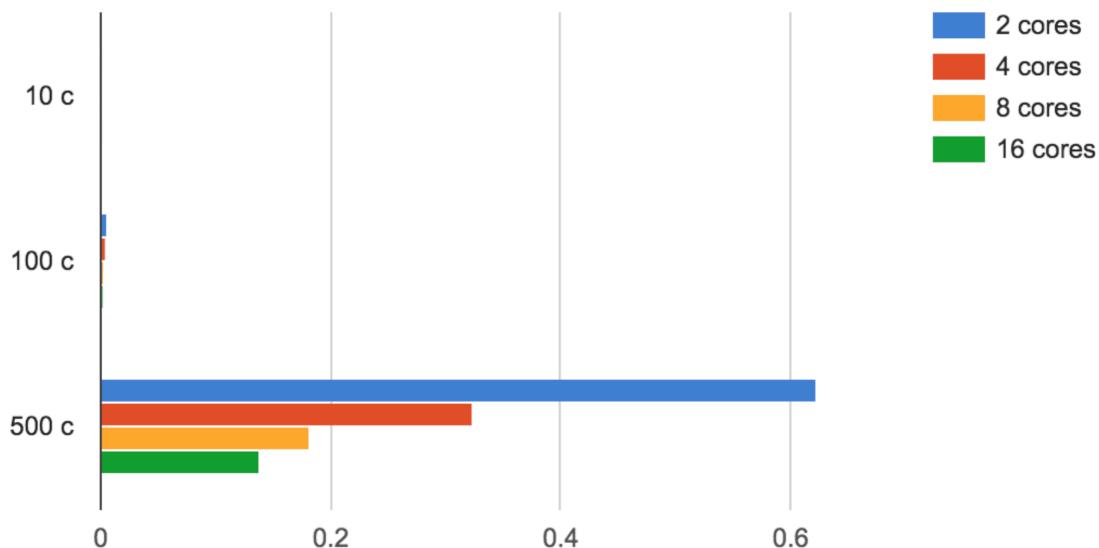
```
cvi067065:9 ivanortegaalba$ ./PmmOpenMP 800
resultado[0][0] = 320400
resultado[N-1][N-1] = 2.5632e+08
Tiempo(seg.):2.328666210 / Tamaño Matrices:800
cvi067065:9 ivanortegaalba$
```

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 ESCALABILIDAD ATCGRID:**

	2 cores	4 cores	8 cores	16 cores
10 c	0.000066	0.000059	0.000048	0.000197
100 c	0.006592	0.003539	0.001832	0.001521
500 c	0.621625	0.32333	0.181763	0.137531

**atcgrid**



**ESTUDIO ESCALABILIDAD PC LOCAL:**

	2 cores	4 cores	8 cores
10 c	0.000071007	0.000023635	0.000030829
100 c	0.012585181	0.002695289	0.002377048
500 c	0.687282269	0.348488738	0.323487649

