

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): Álvarez Ocete José Antonio

Grupo de prácticas: 2

Fecha de entrega: 10/05/2017

Fecha evaluación en clase:

#### 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.

**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, "Uso: \n %s <num-iteraciones> <num-hebras>\n",
argv[0]);
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;
    for (i=0; i<n; i++) {
        a[i] = i;
    }

    x = atoi(argv[2]); if (x < 1) x = 1;

    #pragma omp parallel if(n>4) default(none) private(sumalocal,tid) \
        shared(a,suma,n,x) num_threads(x)
    { 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);
}
}

```

**RESPUESTA:**

En la captura adjunta podemos ver el funcionamiento de la cláusula if. Ejecutaremos el código para 5 hebras (de ahí el segundo argumento proporcionado al programa). Si ejecutamos el código para un número de iteraciones mayor que 4, la región paralela se ejecuta efectivamente en paralelo, con tantas hebras como hemos especificado (5 en este caso).

En cambio si especificamos el número de iteraciones a  $n < 5$ , la región no se ejecuta en paralelo sino que la ejecuta únicamente la hebra 0, como podemos apreciar en la imagen.

**CAPTURAS DE PANTALLA:**

```

jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$ ./if-clau
se 10 4
thread 3 suma de a[8]=8 sumalocal=8
thread 3 suma de a[9]=9 sumalocal=17
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 2 suma de a[6]=6 sumalocal=6
thread 2 suma de a[7]=7 sumalocal=13
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 master=0 imprime suma=45
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$ ./if-clau
se 4 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 0 suma de a[3]=3 sumalocal=6
thread master=0 imprime suma=6
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$ ./if-clau
se
Uso:
./if-clause <num-iteraciones> <num-hebras>
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$

```

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-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	1	0	1
1	1	0	0	1	0	0	1	0	1
2	0	1	0	0	1	0	1	0	1
3	1	1	0	0	1	0	1	0	1
4	0	0	1	0	0	1	1	0	1
5	1	0	1	0	0	1	1	0	1
6	0	1	1	0	0	1	1	0	1
7	1	1	1	0	0	1	1	0	1
8	0	0	0	0	0	0	0	1	0
9	1	0	0	0	0	0	0	1	0
10	0	1	0	0	0	0	0	1	0
11	1	1	0	0	0	0	0	1	0
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	0	0	0
15	1	1	1	0	0	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-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	1	2	0	3	2	2
1	1	0	0	2	2	0	3	2	2
2	2	1	0	0	0	0	3	2	2
3	3	1	0	3	0	0	3	2	2
4	0	2	1	0	1	1	2	1	0
5	1	2	1	0	1	1	2	1	0
6	2	3	1	0	3	1	2	1	0
7	3	3	1	0	3	1	1	0	0
8	0	0	2	0	0	2	1	0	3
9	1	0	2	0	0	2	1	0	3
10	2	1	2	0	1	2	0	3	3

11	3	1	2	0	1	2	0	3	3
12	0	2	3	0	1	3	1	0	1
13	1	2	3	0	1	3	1	0	1
14	2	3	3	0	1	3	1	0	1
15	3	3	3	0	1	3	1	0	1

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

### RESPUESTA:

Con “static” se reparten las iteraciones de “chunk” en “chunk” hasta que se acaben. Esto es, a la primera hebra las primeras “chunk” iteraciones, a la segunda las “chunk” siguientes, y así sucesivamente. Cuando llegamos a la última hebra, volvemos a empezar por la hebra n.º 0. Repetimos el proceso hasta que se acaben las iteraciones.

Con “dynamic” el reparto es igual que para “static” durante la “primera ronda de reparto”, después se asignan todas las iteraciones restantes a la hebra 0.

Con “guided” dividimos el número de iteraciones entre el número de hebras y asignamos el resultado a la primera hebra. Dividimos ahora el número de iteraciones restantes entre el número de hebras y se lo asignamos a la siguiente hebra. Repetimos el proceso hasta que el resultado de la división sea menor que “chunk”.

COMPLETAR, ENTONCES QUE PASA?

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

char* printEnum (omp_sched_t type) {
    char * ret;
    if (type == omp_sched_static)
        ret = "Static";
    else if (type == omp_sched_dynamic)
        ret = "Dynamic";
    else if (type == omp_sched_guided)
        ret = "Guided";
    else if (type == omp_sched_auto)
        ret = "Auto";
    return ret;
}
```

```

int main(int argc, char **argv) {
    int i, n=16, chunk, a[n], suma=0, printed = 0, chunk_read;
    omp_sched_t sched_type;
    if(argc < 2) {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);

    for (i=0; i<n; i++)
        a[i] = i;
    omp_get_schedule(&sched_type, &chunk_read);
    printf("\n\tFuera del bucle:\n dyn var: %d\n nthreads-var: %d\n thread-
limit-var: %d\n run-shed-var: %s --- %d\n\n",
        omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
        printEnum(sched_type), chunk_read);

    #pragma omp parallel for firstprivate(suma) lastprivate(suma)
    schedule(dynamic, chunk)
    for (i=0; i<n; i++)
    {
        #pragma omp critical
        {
            if (printed == 0) {
                printed = 1;
                omp_get_schedule(&sched_type, &chunk_read);
                printf("\n\tDentro del bucle:\n dyn var: %d\n nthreads-var: %d\n thread-
limit-var: %d\n run-shed-var: %s --- %d\n\n\n",
                    omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
                    printEnum(sched_type), chunk_read);
            }
        }
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
            omp_get_thread_num(), i, a[i], suma);
    }

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

```

### CAPTURAS DE PANTALLA:

```

jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
export OMP_DYNAMIC=FALSE
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
export OMP_SCHEDULE="guided,5"
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
export OMP_THREAD_LIMIT=9
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
export OMP_NUM_THREADS=10
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
./scheduled-clauseModificado 4

    Dentro del bucle:
dyn var: 0
nthreads-var: 10
thread-limit-var: 9
run-shed-var: Guided --- 5

    Fuera del bucle:
dyn var: 0
nthreads-var: 10
thread-limit-var: 9
run-shed-var: Guided --- 5

```

```
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
export OMP_NUM_THREADS=3
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
export OMP_THREAD_LIMIT=8
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
export OMP_DYNAMIC=TRUE
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
export OMP_SCHEDULE="static,3"
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$
./scheduled-clauseModificado 4

    Dentro del bucle:
dyn var: 1
nthreads-var: 3
thread-limit-var: 8
run-shed-var: Static --- 3

    Fuera del bucle:
dyn var: 1
nthreads-var: 3
thread-limit-var: 8
run-shed-var: Static --- 3
```

**RESPUESTA:** Como podemos ver en las capturas de pantalla, los valores de las variables no cambian dentro y fuera del bucle.

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

char* printBool (int b) {
    char * ret;
    if (b == 0)
        ret = "False";
    else if (b == 1)
        ret = "True";
    else
        ret = "Error en printBool";
    return ret;
}

int main(int argc, char **argv) {
    int i, n=16, chunk, a[n], suma=0, printed = 0;
    if(argc < 2) {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);

    for (i=0; i<n; i++)
```

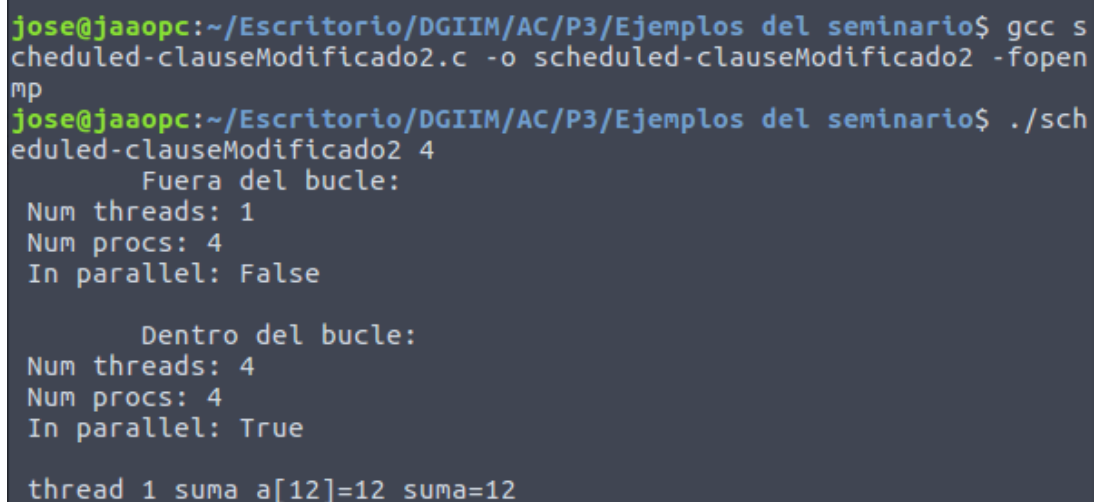
```

    a[i] = i;
    printf("\n\tFuera del bucle:\n Num threads: %d\n Num procs: %d\n In
parallel: %s\n",
        omp_get_num_threads(), omp_get_num_procs(), printBool(omp_in_parallel()));

    #pragma omp parallel for firstprivate(suma) lastprivate(suma)
    schedule(dynamic, chunk)
    for (i=0; i<n; i++)
    {
        #pragma omp critical
        {
            if (printed == 0) {
                printed = 1;
                printf("\n\tDentro del bucle:\n Num threads: %d\n Num procs: %d\n In
parallel: %s\n\n",
                    omp_get_num_threads(), omp_get_num_procs(),
                    printBool(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("Fuera de 'parallel for' suma=%d\n", suma);
}

```

**CAPTURAS DE PANTALLA:**


```

jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$ gcc s
cheduled-clauseModificado2.c -o scheduled-clauseModificado2 -fopen
mp
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$ ./sch
eduled-clauseModificado2 4
    Fuera del bucle:
    Num threads: 1
    Num procs: 4
    In parallel: False

    Dentro del bucle:
    Num threads: 4
    Num procs: 4
    In parallel: True

    thread 1 suma a[12]=12 suma=12

```

**RESPUESTA:**

Como podemos ver en el volcado de pantalla, el número de procesadores disponibles no varía mientras que el indicador de que estamos en una región paralela (`omp_in_parallel`) y el número de hebras actuales si que lo hacen.

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

char* printEnum (omp_sched_t type) {
    char * ret;
    if (type == omp_sched_static)
        ret = "Static";
    else if (type == omp_sched_dynamic)
        ret = "Dynamic";
    else if (type == omp_sched_guided)
        ret = "Guided";
    else if (type == omp_sched_auto)
        ret = "Auto";
    return ret;
}

char* printBool (int b) {
    char * ret;
    if (b == 0)
        ret = "False";
    else if (b == 1)
        ret = "True";
    else
        ret = "Error en printBool";
    return ret;
}

int main(int argc, char **argv) {
    int i, n=16, chunk, a[n], suma=0, printed = 0, chunk_read;
    omp_sched_t sched_type;
    if(argc < 2) {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);

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

    omp_get_schedule(&sched_type, &chunk_read);
    printf("\n\tAntes de la modificación:\n dyn var: %s\n nthreads-var: %d\n\n",
        printBool(omp_get_dynamic()), omp_get_max_threads(),
        printEnum(sched_type), chunk_read);

    omp_set_dynamic(1);
    omp_set_num_threads(2);
    omp_set_schedule(omp_sched_auto, chunk_read + 1);

    omp_get_schedule(&sched_type, &chunk_read);
    printf("\n\tDespués de la modificación:\n dyn var: %s\n nthreads-var: %d\n\n",
        printBool(omp_get_dynamic()), omp_get_max_threads(),
        printEnum(sched_type), chunk_read);

    #pragma omp parallel for firstprivate(suma) lastprivate(suma)

```

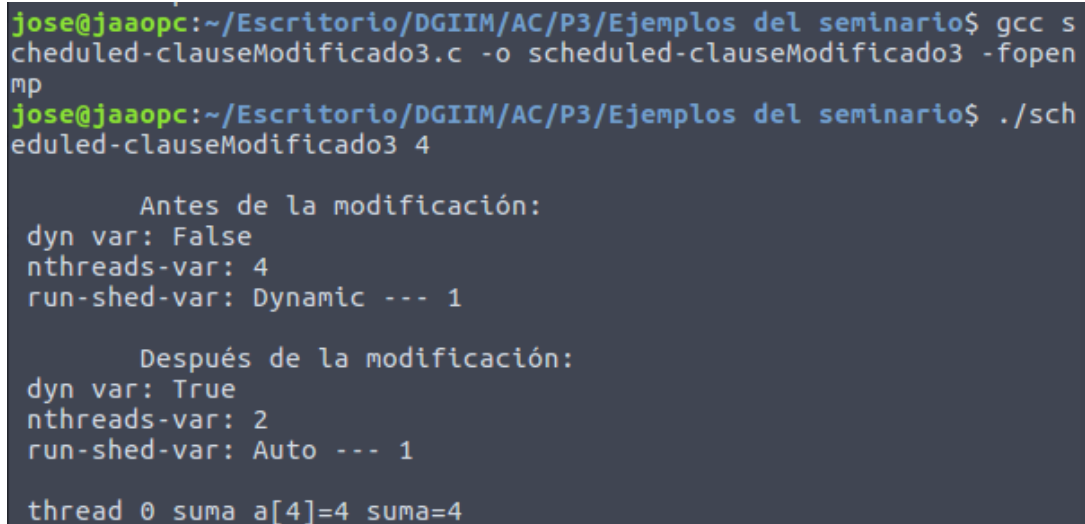


```

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

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

```

**CAPTURAS DE PANTALLA:**


```

jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$ gcc s
cheduled-clauseModificado3.c -o scheduled-clauseModificado3 -fopen
mp
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Ejemplos del seminario$ ./sch
eduled-clauseModificado3 4

    Antes de la modificación:
dyn var: False
nthreads-var: 4
run-shed-var: Dynamic --- 1

    Después de la modificación:
dyn var: True
nthreads-var: 2
run-shed-var: Auto --- 1

thread 0 suma a[4]=4 suma=4

```

**RESPUESTA:**

En este ejercicio no hay preguntar que responder.

**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.

**CÓDIGO FUENTE:** pmtv-secuencial.c

```

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

#define PRINT_ALL
// #define VECTOR_GLOBAL
#define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
#define MAX 32768 // = 2^10
double v[MAX], m[MAX][MAX], r[MAX];
#endif

```

```

int main(int argc, char** argv){

    if (argc<2){
        printf("Faltan nº componentes del vector \n");
        exit(-1);
    }

    struct timespec cgt1,cgt2;
    double ncgt;    //para tiempo de ejecución
    int i, j;
    unsigned int N = atoi(argv[1]);    // Máximo N =2^32 -1=4294967295
    (sizeof(unsigned int) = 4 B)

    #ifdef VECTOR_GLOBAL
    if (N>MAX)
        N=MAX;
    #endif

    #ifdef VECTOR_DYNAMIC
    double *v, **m, *r;
    v = (double*) malloc(N*sizeof(double));    // malloc necesita el tamaño en
bytes
    m = (double**) malloc(N*sizeof(double*));    //si no hay espacio suficiente
malloc devuelve NULL
    for (i=0; i<N; i++)
        m[i] = (double*) malloc(N*sizeof(double));
    r = (double*) malloc(N*sizeof(double));
    if ((v==NULL) || (m==NULL) || (r==NULL)) {
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
    #endif

    //Inicializar vector y matriz
    for (j=0; j<N; j++) {
        v[j] = 2.5;
        m[0][j] = 1.1;
        for (i=1; i<=j; i++)
            m[i][j] = -m[i-1][j];
        for (; i<N; i++)
            m[i][j] = 0;
    }

    //Comprobamos la incialización
    #ifdef PRINT_ALL
    printf("\n Vector:\n");
    for (i=0; i<N; i++) {
        printf("\t%0.1f", v[i]);
    }

    printf("\n\n Matriz: \n");

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++)
            printf("\t%0.1f", m[i][j]);
        printf("\n\n");
    }
    #endif

    clock_gettime(CLOCK_REALTIME,&cgt1);

```

```

//Calcular el producto
double sum;
for (j=0; j<N; j++) {
    sum = 0;
    for (i=0; i<=j; i++)
        sum += v[i]*m[i][j];
    r[j] = sum;
}

clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) +
        (double) ((cgt2.tv_nsec - cgt1.tv_nsec)/(1.e+9));

//Imprimir resultado del producto
printf("\n Resultado:\n");
#ifdef PRINT_ALL
for (i=0; i<N; i++) {
    printf("\t%0.2f", r[i]);
}
printf("\n");
#else
printf("Primer valor: %0.1f \t Último valor: %0.1f \n", r[0], r[N-1]);
#endif
printf("\n Tiempo de ejecución(s): %11.9f\n", ncgt);

#ifdef VECTOR_DYNAMIC
free(v);          // libera el espacio reservado para v
free(m);          // libera el espacio reservado para m
free(r);
#endif
return 0;
}

```

**CAPTURAS DE PANTALLA:**

```

jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Programas$ ./Ejer6 5

Vector:
    2.5    2.5    2.5    2.5    2.5

Matriz:
    1.1    1.1    1.1    1.1    1.1
    0.0   -1.1   -1.1   -1.1   -1.1
    0.0    0.0    1.1    1.1    1.1
    0.0    0.0    0.0   -1.1   -1.1
    0.0    0.0    0.0    0.0    1.1

Resultado:
    2.75    0.00    2.75    0.00    2.75

Tiempo de ejecución(s): 0.000000224

```

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:

Para ser el valor por defecto me he servido de internet, de una fuente segura:

<https://software.intel.com/en-us/articles/openmp-loop-scheduling>

Valores de `chunk` por defecto:

`Static` – el número de hebras.

`Dynamic` – 1.

`Guided` – aproximadamente el número de hebras.

Independientemente del valor de `N`, se realizan `i` multiplicaciones y sumas por fila, donde `i` es la fila actual. Si denominamos `a = omp_get_thread_num` y `b = omp_get_num_threads` (número de la hebra y número de hebras totales respectivamente), cada hebra tendrá exactamente las siguientes multiplicaciones y sumas: el sumatorio desde el número '`i = a`' hasta '`N / b`' de '`a*b*i`', asumiendo que `N` es múltiplo de `b`, como nos exige el enunciado.

Al ser el reparto estático, dicho número es fijo desde el momento en el que se compila el programa.

En contraposición, con los otros dos tipos de `scheduling` no podemos conocer de forma exacta el número de operaciones que realizará cada hebra porque el reparto es dinámico.

### CÓDIGO FUENTE: `pmtv-OpenMP.c`

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

```

// #define PRINT_ALL
// #define VECTOR_GLOBAL
#define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
#define MAX 32768 // = 2^10
double v[MAX], m[MAX][MAX], r[MAX];
#endif

omp_sched_t charToSchedType (char c) {
    omp_sched_t t;
    if (c == 'S')
        t = omp_sched_static;
    else if (c == 'D')
        t = omp_sched_dynamic;
    else if (c == 'G')
        t = omp_sched_guided;
    else if (c == 'A')
        t = omp_sched_auto;
    else {
        printf(" Error en tipo de schedule.\n"
            " Puede ser (S)tatic - (D)ynamic - (G)uided - (A)uto\n");
        exit(-1);
    }
    return t;
}

char* printEnum (omp_sched_t type) {
    char * ret;
    if (type == omp_sched_static)
        ret = "Static";
    else if (type == omp_sched_dynamic)
        ret = "Dynamic";
    else if (type == omp_sched_guided)
        ret = "Guided";
    else if (type == omp_sched_auto)
        ret = "Auto";
    return ret;
}

int main(int argc, char** argv){

    if (argc < 4){
        printf("Error en nº de parámetros. Ejecución:\n %s <Tipo de schedule>
<chunk> <iteraciones>\n", argv[0]);
        printf("El schedule puede ser (S)tatic - (D)ynamic - (G)uided -
(A)uto\n");
        printf("Introducir chunk = 0 para tomar el valor por defecto según el
schedueling.\n");
        exit(-1);
    }

    omp_sched_t sched_type = charToSchedType(argv[1][0]);

    int chunk = atoi(argv[2]);
    if (chunk == 0) {
        if (sched_type == 'S' || sched_type == 'G')
            chunk = omp_get_num_threads();
        else
            chunk = 1;
    } else if (chunk < 0) {
        chunk = 1;
    }
}

```

```

    printf("\n Valor de chunk negativo. Queda fijado a 1.\n");
}

unsigned int N = atoi(argv[3]); // Máximo N =2^32 -1=4294967295
(sizeof(unsigned int) = 4 B)
if (N < 1) {
    printf("Error - Número de iteraciones negativo.\n");
    exit(-1);
}

struct timespec cgt1,cgt2;
double ncgt; //para tiempo de ejecución
int i, j;

omp_set_schedule(sched_type, chunk);
/*
// Comprobamos que hemos fijado bien el schedueling y el chunk
omp_get_schedule(&sched_type, &chunk);
printf("\n run-shed-var: Schedule %s --- Chunk = %d\n",
    printEnum(sched_type), chunk);
*/

#ifdef VECTOR_GLOBAL
if (N>MAX)
    N=MAX;
printf("\n Número de iteraciones refijado al máximo posible: %d\n", N);
#endif
#ifdef VECTOR_DYNAMIC
double *v, **m, *r;
v = (double*) malloc(N*sizeof(double)); // malloc necesita el tamaño en
bytes
m = (double**) malloc(N*sizeof(double*)); //si no hay espacio suficiente
malloc devuelve NULL
for (i=0; i<N; i++)
    m[i] = (double*) malloc(N*sizeof(double));
r = (double*) malloc(N*sizeof(double));
if ((v==NULL) || (m==NULL) || (r==NULL)) {
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
}
#endif

//Inicializar vector y matriz
#pragma omp parallel for private(i)
for (j=0; j<N; j++) {
    v[j] = 2.5;
    m[0][j] = 1.1;
    for (i=1; i<=j; i++)
        m[i][j] = -m[i-1][j];
    for (; i<N; i++)
        m[i][j] = 0;
}

//Comprobamos la incialización
#ifdef PRINT_ALL
printf("\n Vector:\n");
for (i=0; i<N; i++) {
    printf("\t%0.1f", v[i]);
}

printf("\n\n Matriz: \n");

```

```

for (i=0; i<N; i++) {
    for (j=0; j<N; j++)
        printf("\t%0.1f", m[i][j]);
    printf("\n\n");
}
#endif

clock_gettime(CLOCK_REALTIME,&cgt1);
//Calcular el producto
double sum;
#pragma omp parallel for private(sum, i)
for (j=0; j<N; j++) {
    sum = 0;
    for (i=0; i<=j; i++)
        sum += v[i]*m[i][j];
    r[j] = sum;
}

clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) +
        (double) ((cgt2.tv_nsec - cgt1.tv_nsec)/(1.e+9));

//Imprimir resultado del producto
printf("\n Resultado:\n");
#ifdef PRINT_ALL
for (i=0; i<N; i++) {
    printf("\t%0.2f", r[i]);
}
printf("\n");
#else
printf("Primer valor: %0.1f \t Último valor: %0.1f \n", r[0], r[N-1]);
#endif
printf("\n Tiempo de ejecución(s): %11.9f\n\n", ncgt);

#ifdef VECTOR_DYNAMIC
free(v);          // libera el espacio reservado para v
free(m);          // libera el espacio reservado para m
free(r);
#endif
return 0;
}

```

**CAPTURAS DE PANTALLA:**

```

[E2estudiante2@atcgrid ~]$ qsub pvmt.sh
57143.atcgrid
[E2estudiante2@atcgrid ~]$ cat salida.txt
scheduling static -- Chunk por defecto -- 15360 iteraciones:

Resultado:
Primer valor: 2.8          Último valor: 0.0

Tiempo de ejecución(s): 0.380980752

scheduling dynamic -- Chunk por defecto -- 15360 iteraciones:

```

**TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID****SCRIPT:** pmvt-OpenMP\_atcgrid.sh

```
#!/bin/bash
# Se asigna al trabajo el nombre SumaVectores
#PBS -N pvmt
# Se asigna al trabajo la cola ac
#PBS -q ac
# Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. 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"
# Se ejecuta pvmt-OpenMP, que está en el directorio en el que se ha ejecutado
qsub,
# para distintos valores de scheduling y chunk, con 15360 iteraciones.
echo "scheduling static -- Chunk por defecto -- 15360 iteraciones:" >
salida.txt
./pvmt-OpenMP S 0 15360 >> salida.txt
echo "scheduling dynamic -- Chunk por defecto -- 15360 iteraciones:" >>
salida.txt
./pvmt-OpenMP D 0 15360 >> salida.txt
echo "scheduling guided -- Chunk por defecto -- 15360 iteraciones:" >>
salida.txt
./pvmt-OpenMP G 0 15360 >> salida.txt
echo "scheduling static -- Chunk 1 -- 15360 iteraciones:" >> salida.txt
./pvmt-OpenMP S 1 15360 >> salida.txt
echo "scheduling dynamic -- Chunk 1 -- 15360 iteraciones:" >> salida.txt
./pvmt-OpenMP D 1 15360 >> salida.txt
echo "scheduling guided -- Chunk 1 -- 15360 iteraciones:" >> salida.txt
./pvmt-OpenMP G 1 15360 >> salida.txt
echo "scheduling static -- Chunk 64 -- 15360 iteraciones:" >> salida.txt
./pvmt-OpenMP S 64 15360 >> salida.txt
echo "scheduling dynamic -- Chunk 64 -- 15360 iteraciones:" >> salida.txt
./pvmt-OpenMP D 64 15360 >> salida.txt
echo "scheduling guided -- Chunk 64 -- 15360 iteraciones:" >> salida.txt
./pvmt-OpenMP G 64 15360 >> salida.txt
echo "Fin del script - sin errores"
```

**Tabla 3.** Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector  $r$  para vectores de tamaño  $N = 15360$ , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.380980752	0.433201577	0.416379779
1	0.377752911	0.451625615	0.403091405
64	0.446546161	0.379777255	0.448751480
Chunk	Static	Dynamic	Guided
por defecto	0.453910425	0.395356335	0.443649250
1	0.417107388	0.423722947	0.429010721
64	0.524731778	0.395849186	0.381089460



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

#define PRINT_ALL
// #define VECTOR_GLOBAL
#define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
#define MAX 32768 // = 2^10
double m1[MAX][MAX], m2[MAX][MAX], r[MAX][MAX];
#endif

int main(int argc, char** argv){

    if (argc < 2){
        printf("Faltan nº componentes del vector \n");
        exit(-1);
    }

    struct timespec cgt1, cgt2;
    double ncgt; // para tiempo de ejecución
    int i, j, k;
    unsigned int N = atoi(argv[1]); // Máximo N = 2^32 - 1 = 4294967295
    (sizeof(unsigned int) = 4 B)

    #ifdef VECTOR_GLOBAL
    if (N > MAX)
        N = MAX;
    #endif

    #ifdef VECTOR_DYNAMIC
    double **m1, **m2, **r;
    m1 = (double**) malloc(N * sizeof(double*)); // malloc necesita el tamaño en bytes
    for (i=0; i<N; i++)
        m1[i] = (double*) malloc(N * sizeof(double));
    m2 = (double**) malloc(N * sizeof(double*)); // si no hay espacio suficiente
    malloc devuelve NULL
    for (i=0; i<N; i++)
        m2[i] = (double*) malloc(N * sizeof(double));
    r = (double**) malloc(N * sizeof(double*));
    for (i=0; i<N; i++)
        r[i] = (double*) malloc(N * sizeof(double));
    if ((m1==NULL) || (m2==NULL) || (r==NULL)) {
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
    #endif
}
```

```

}
#endif

// Inicializar matrices
for (i=0; i<N; i++) {
    m1[i][0] = 1.1;
    m2[i][0] = 1.1;
    for (j=1; j<N; j++) {
        m1[i][j] = - m1[i][j-1];
        m2[i][j] = - m2[i][j-1];
    }
}

//Comprobamos la inicialización
#ifdef PRINT_ALL
printf("\n\n Matriz 1: \n");
for (i=0; i<N; i++) {
    for (j=0; j<N; j++)
        printf("\t%.1f", m1[i][j]);
    printf("\n\n");
}
printf("\n\n Matriz 2: \n");
for (i=0; i<N; i++) {
    for (j=0; j<N; j++)
        printf("\t%.1f", m2[i][j]);
    printf("\n\n");
}
#endif

clock_gettime(CLOCK_REALTIME,&cgt1);
//Calcular el producto
double sum;
for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
        sum = 0;
        for (k=0; k<N; k++) {
            sum += m1[i][k] * m2[k][j];
        }
        r[i][j] = sum;
    }
}
clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) +
        (double) ((cgt2.tv_nsec - cgt1.tv_nsec)/(1.e+9));

//Imprimir resultado del producto
printf("\n Resultado:\n");
#ifdef PRINT_ALL
for (i=0; i<N; i++) {
    for (j=0; j<N; j++)
        printf("\t%.2f", r[i][j]);
    printf("\n\n");
}
printf("\n");
#else
printf("Primer valor: %.2f \t Último valor: %.2f \n", r[0][0], r[N-1][N-1]);
#endif
printf("\n Tiempo de ejecución(s): %11.9f\n", ncgt);

#ifdef VECTOR_DYNAMIC
for (i=0; i<N; i++)

```

```

    free(m1[i]);
    free(m1);
    for (i=0; i<N; i++)
        free(m2[i]);
    free(m2);
    for (i=0; i<N; i++)
        free(r[i]);
    free(r);
    #endif
    return 0;
}

```

**CAPTURAS DE PANTALLA:**

```

jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Programas$ gcc -O2 Ejer8.c -o Ejer8
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Programas$ ./Ejer8 5

Matriz 1:
  1.1   -1.1   1.1   -1.1   1.1
  1.1   -1.1   1.1   -1.1   1.1
  1.1   -1.1   1.1   -1.1   1.1
  1.1   -1.1   1.1   -1.1   1.1
  1.1   -1.1   1.1   -1.1   1.1

Matriz 2:
  1.1   -1.1   1.1   -1.1   1.1
  1.1   -1.1   1.1   -1.1   1.1
  1.1   -1.1   1.1   -1.1   1.1
  1.1   -1.1   1.1   -1.1   1.1
  1.1   -1.1   1.1   -1.1   1.1

Resultado:
  1.21  -1.21  1.21  -1.21  1.21
  1.21  -1.21  1.21  -1.21  1.21
  1.21  -1.21  1.21  -1.21  1.21
  1.21  -1.21  1.21  -1.21  1.21
  1.21  -1.21  1.21  -1.21  1.21

Tiempo de ejecución(s): 0.000000597
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Programas$

```

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).

**CÓDIGO FUENTE:** pmm-OpenMP.c

```

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

#define PRINT_ALL
// #define VECTOR_GLOBAL

```

```

#define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
#define MAX 32768 //2^10
double m1[MAX][MAX], m2[MAX][MAX], r[MAX][MAX];
#endif

int main(int argc, char** argv){

    if (argc<2){
        printf("Faltan nº componentes del vector \n");
        exit(-1);
    }

    struct timespec cgt1, cgt2;
    double ncgt; //para tiempo de ejecución
    int i, j, k;
    unsigned int N = atoi(argv[1]); // Máximo N =2^32 -1=4294967295
    (sizeof(unsigned int) = 4 B)

    #ifdef VECTOR_GLOBAL
    if (N>MAX)
        N=MAX;
    #endif

    #ifdef VECTOR_DYNAMIC
    double **m1, **m2, **r;
    m1 = (double**) malloc(N*sizeof(double)); // malloc necesita el tamaño en
bytes
    for (i=0; i<N; i++)
        m1[i] = (double*) malloc(N*sizeof(double));
    m2 = (double**) malloc(N*sizeof(double)); //si no hay espacio suficiente
malloc devuelve NULL
    for (i=0; i<N; i++)
        m2[i] = (double*) malloc(N*sizeof(double));
    r = (double**) malloc(N*sizeof(double));
    for (i=0; i<N; i++)
        r[i] = (double*) malloc(N*sizeof(double));
    if ((m1==NULL) || (m2==NULL) || (r==NULL)) {
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
    #endif

    // Inicializar matrices
    #pragma omp parallel for private(j)
    for (i=0; i<N; i++) {
        m1[i][0] = 1.1;
        m2[i][0] = 1.1;
        for (j=1; j<N; j++) {
            m1[i][j] = - m1[i][j-1];
            m2[i][j] = - m2[i][j-1];
        }
    }

    //Comprobamos la inicialización
    #ifdef PRINT_ALL
    printf("\n\n Matriz 1: \n");
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++)

```

```

        printf("\t%0.1f", m1[i][j]);
        printf("\n\n");
    }
    printf("\n\n Matriz 2: \n");
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++)
            printf("\t%0.1f", m2[i][j]);
        printf("\n\n");
    }
#endif

clock_gettime(CLOCK_REALTIME,&cgt1);
//Calcular el producto
double sum;

for (i=0; i<N; i++) {
    #pragma omp parallel for private(j,k,sum)
    for (j=0; j<N; j++) {
        sum = 0;
        for (k=0; k<N; k++) {
            sum += m1[i][k] * m2[k][j];
        }
        r[i][j] = sum;
    }
}

clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) +
        (double) ((cgt2.tv_nsec - cgt1.tv_nsec)/(1.e+9));

//Imprimir resultado del producto
printf("\n Resultado:\n");
#ifdef PRINT_ALL
for (i=0; i<N; i++) {
    for (j=0; j<N; j++)
        printf("\t%0.2f", r[i][j]);
    printf("\n\n");
}
printf("\n");
#else
printf("Primer valor: %0.2f \t Último valor: %0.2f \n", r[0][0], r[N-1][N-
1]);
#endif
printf("\n Tiempo de ejecución(s): %11.9f\n", ncgt);

#ifdef VECTOR_DYNAMIC
for (i=0; i<N; i++)
    free(m1[i]);
free(m1);
for (i=0; i<N; i++)
    free(m2[i]);
free(m2);
for (i=0; i<N; i++)
    free(r[i]);
free(r);
#endif
return 0;
}

```

**CAPTURAS DE PANTALLA:**

```
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Programas$ ./pmm-OpenMP 5

Matriz 1:
  1.1    -1.1    1.1    -1.1    1.1
  1.1    -1.1    1.1    -1.1    1.1
  1.1    -1.1    1.1    -1.1    1.1
  1.1    -1.1    1.1    -1.1    1.1
  1.1    -1.1    1.1    -1.1    1.1

Matriz 2:
  1.1    -1.1    1.1    -1.1    1.1
  1.1    -1.1    1.1    -1.1    1.1
  1.1    -1.1    1.1    -1.1    1.1
  1.1    -1.1    1.1    -1.1    1.1
  1.1    -1.1    1.1    -1.1    1.1

Resultado:
  1.21   -1.21   1.21   -1.21   1.21
  1.21   -1.21   1.21   -1.21   1.21
  1.21   -1.21   1.21   -1.21   1.21
  1.21   -1.21   1.21   -1.21   1.21
  1.21   -1.21   1.21   -1.21   1.21

Tiempo de ejecución(s): 0.000007318
jose@jaaopc:~/Escritorio/DGIIM/AC/P3/Programas$
```

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 dos 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 (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`

#### ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

**SCRIPT:** `pmm-OpenMP_pcllocal.sh`