

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): Jose Miguel Hernández García

Grupo de prácticas: C3

Fecha de entrega: 10-05-17

Fecha evaluación en clase: 11-05-17

### 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 < 2){
        fprintf(stderr, "[ERROR]-Falta iteraciones\n");
        exit(-1);
    }

    if(argc < 3){
        fprintf(stderr, "[ERROR]-Falta numero threads\n");
        exit(-1);
    }

    n = atoi(argv[1]);
    x = atoi(argv[2]);

    if(n>20) n=20;

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

    #pragma omp parallel num_threads(x) if(n>4) default(none)
    private(sumalocal, tid) \
        shared(a, suma,n)
    {
        sumalocal = 0;
        tid = omp_get_thread_num();

        #pragma omp for private(i) schedule(static) nowait
        for(i=0; i<n; i++){
```

```

        sumalocal+=a[i];
        printf("thread %d suma de a[%d]=%d sumalocal=%d\n", tid, i, a[i],
sumalocal);
    }

    #pragma omp atomic
    suma+=sumalocal;

    #pragma omp barrier
    #pragma omp master
    printf("thread master=%d imprime suma=%d\n", tid, suma);
}
}

```

**CAPTURAS DE PANTALLA:**

```

ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./if-clauseModificado 2 4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=1
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./if-clauseModificado 2 7
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=1
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./if-clauseModificado 5 7
thread 4 suma de a[4]=4 sumalocal=4
thread 2 suma de a[2]=2 sumalocal=2
thread 3 suma de a[3]=3 sumalocal=3
thread 1 suma de a[1]=1 sumalocal=1
thread 0 suma de a[0]=0 sumalocal=0
thread master=0 imprime suma=10
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./if-clauseModificado 5 3
thread 2 suma de a[4]=4 sumalocal=4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread master=0 imprime suma=10
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./if-clauseModificado 5 2

```

**RESPUESTA:** En el código es posible apreciar que únicamente se va a paralelizar cuando el numero de iteraciones sea mayor que 4.

Por otro lado vemos que con `num_threads()` podemos indicar el numero de hebras que se crean, sin necesidad de recompilar el programa, pero esto solo ocurrirá cuando hayan mas de 4 iteraciones.

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.c			schedule-clause.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	1	0	0	1	0	0	0
6	0	1	1	0	0	1	0	0	0
7	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	1	0	1	1	1
11	1	1	0	0	1	0	1	1	1
12	0	0	1	0	1	0	0	0	0
13	1	0	1	0	1	0	0	0	0
14	0	1	1	0	1	0	0	0	0
15	1	1	1	0	1	0	0	0	0

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

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

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

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

**RESPUESTA:** Cuando usamos `static`, todas las tareas se reparten de forma equitativa mediante `round-robin`, mientras que con `dynamic` y con `guided` no se sabe como ocurrirá el reparto, únicamente se sabe que el tamaño de las tareas viene definido por `chunk`(cada hebra hace `chunk` iteraciones).

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
    #define omp_get_num_threads() 1
    #define omp_set_num_threads(int)
#endif

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

    int i, n=200, chunk, a[n], suma=0;
    omp_sched_t schedule_type;
    int chunk_value;

    if(argc < 3){
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    // omp_set_num_threads(4);

    n = atoi(argv[1]);
    chunk = atoi(argv[2]);

    if (n>200) n=200;

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

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

        if(omp_get_thread_num() == 0){
            printf("static = 1, dynamic = 2, guided = 3, auto = 4\n");
            omp_get_schedule(&schedule_type, &chunk_value);
            printf("dyn-var: %d, nthreads-var:%d, thread-limit-var:%d, run-
            sched-var: %d, chunk: %d\n", \
                omp_get_dynamic(), omp_get_max_threads(),
            omp_get_thread_limit(), \
                schedule_type, chunk_value);
        }
    }

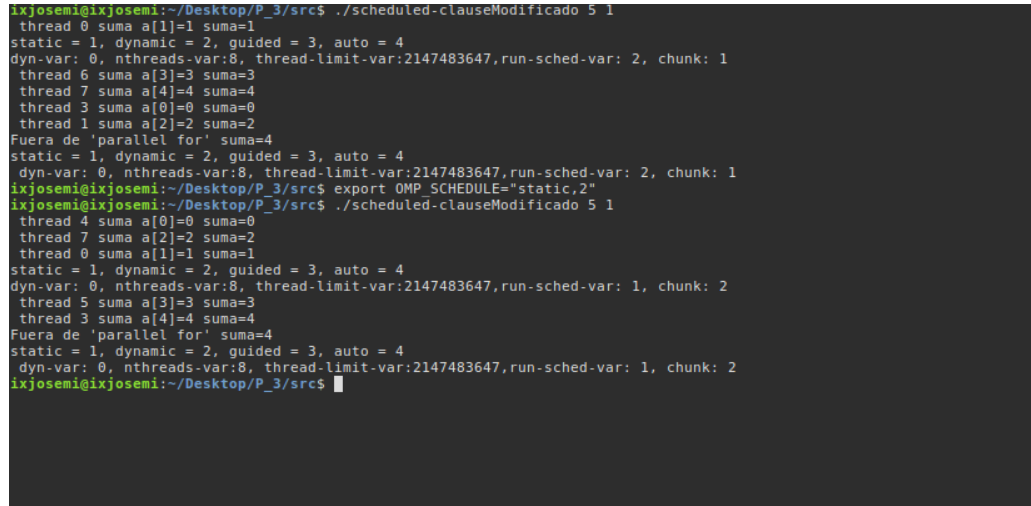
    printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("static = 1, dynamic = 2, guided = 3, auto = 4\n");
    omp_get_schedule(&schedule_type, &chunk_value);
}
```

```

printf(" dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var:
%d, chunk: %d\n", \
omp_get_dynamic(),omp_get_max_threads(), omp_get_thread_limit(), \
schedule_type, chunk_value);
}

```

### CAPTURAS DE PANTALLA:



```

ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./scheduled-clauseModificado 5 1
thread 0 suma a[1]=1 suma=1
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var:8, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
thread 6 suma a[3]=3 suma=3
thread 7 suma a[4]=4 suma=4
thread 3 suma a[0]=0 suma=0
thread 1 suma a[2]=2 suma=2
Fuera de 'parallel for' suma=4
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var:8, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
ixjosemi@ixjosemi:~/Desktop/P_3/src$ export OMP_SCHEDULE="static,2"
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./scheduled-clauseModificado 5 1
thread 4 suma a[0]=0 suma=0
thread 7 suma a[2]=2 suma=2
thread 0 suma a[1]=1 suma=1
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var:8, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
thread 5 suma a[3]=3 suma=3
thread 3 suma a[4]=4 suma=4
Fuera de 'parallel for' suma=4
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var:8, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
ixjosemi@ixjosemi:~/Desktop/P_3/src$

```

**RESPUESTA:** Obtenemos los mismos resultados fuera y dentro de la región paralela

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
#define omp_get_num_threads() 1
#define omp_set_num_threads(int)
#endif

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

    int i, n=200, chunk, a[n], suma=0;
    omp_sched_t schedule_type;
    int chunk_value;

    if(argc < 3){
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    // omp_set_num_threads(4);

```

```

n = atoi(argv[1]);
chunk = atoi(argv[2]);

if (n>200) n=200;

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

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

    if(omp_get_thread_num() == 0){
        printf("static = 1, dynamic = 2, guided = 3, auto = 4\n");
        omp_get_schedule(&schedule_type, &chunk_value);
        printf("dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-
sched-var: %d, chunk: %d\n", \
            omp_get_dynamic(),omp_get_max_threads(),
omp_get_thread_limit(),\
            schedule_type, chunk_value);
        printf(" get_num_threads: %d,get_num_procs: %d,in_parallel():%d
\n", \
            omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
    }
}

printf("Fuera de 'parallel for' suma=%d\n",suma);
printf("static = 1, dynamic = 2, guided = 3, auto = 4\n");
omp_get_schedule(&schedule_type, &chunk_value);
printf(" dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var:
%d, chunk: %d\n", \
    omp_get_dynamic(),omp_get_max_threads(), omp_get_thread_limit(), \
    schedule_type, chunk_value);
}

```

**CAPTURAS DE PANTALLA:**

```

ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./scheduled-clauseModificado4 5 3
thread 5 suma a[0]=0 suma=0
thread 5 suma a[1]=1 suma=1
thread 5 suma a[2]=2 suma=3
thread 3 suma a[3]=3 suma=3
thread 3 suma a[4]=4 suma=7
Fuera de 'parallel for' suma=7
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var:8, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
ixjosemi@ixjosemi:~/Desktop/P_3/src$

```

**RESPUESTA:** La única que se mantiene tanto dentro como fuera es `omp_get_num_procs()` las otras dos varían.

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

```

/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 .*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

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

```

**CAPTURAS DE PANTALLA:**

```

ixjosemi@ixjosemi:~/Desktop/P_3/src$ gcc -O2 -fopenmp -o scheduled-clauseModificado5 scheduled-clauseModificado5.c
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./scheduled-clauseModificado5 5 3
Antes del cambio
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 0, nthreads-var:8, thread-limit-var:2147483647,run-sched-var: 1, chunk: 2
get_num_threads: 1,get_num_procs: 8,in_parallel():0
thread 0 suma a[0]=0 suma=0
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
get_num_threads: 2,get_num_procs: 8,in_parallel():1
thread 0 suma a[1]=1 suma=1
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
get_num_threads: 2,get_num_procs: 8,in_parallel():1
thread 0 suma a[2]=2 suma=3
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
thread 1 suma a[3]=3 suma=3
thread 1 suma a[4]=4 suma=7
get_num_threads: 2,get_num_procs: 8,in_parallel():1
Fuera de 'parallel for' suma=7
static = 1, dynamic = 2, guided = 3, auto = 4
dyn-var: 1, nthreads-var:2, thread-limit-var:2147483647,run-sched-var: 2, chunk: 1
get_num_threads: 1,get_num_procs: 8,in_parallel():0
ixjosemi@ixjosemi:~/Desktop/P_3/src$

```

**RESPUESTA:** Al cambiar los valores, las funciones devuelven los valores que hemos establecido.

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

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

    int i, j;

    if(argc<2){

```



```

        fprintf(stderr, "Falta tamaño\n");
        exit(-1);
    }

    unsigned int N;
    N=atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4
B)

    // Inicializamos la matriz triangular (superior)
    int *vector, *resultado, **matriz;
    vector=(int *) malloc(N*sizeof(int)); // malloc necesita el tamaño en
bytes
    resultado=(int *) malloc(N*sizeof(int)); // si no hay espacio suficiente
malloc devuelve NULL
    matriz=(int **) malloc(N*sizeof(int*));

    for (i=0; i<N; i++)
        matriz[i] = (int*) malloc(N*sizeof(int));

    for (i=0; i<N; i++){
        for (j=i; j<N; j++)
            matriz[i][j]=3;

        vector[i]=5;
        resultado[i]=0;
    }

    // Pintamos la matriz
    printf("Matriz:\n");

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

    // Pintamos el vector
    printf("Vector:\n");

    for (i=0; i<N; i++)
        printf("%d ", vector[i]);
    printf("\n");

    // Obtenemos los resultados
    for (i=0; i<N; i++)
        for (j=i; j<N; j++)
            resultado[i] += matriz[i][j] * vector[j];

    // Pintamos los resultados
    printf("Resultado:\n");

    for (i=0; i<N; i++)
        printf("%d ", resultado[i]);
    printf("\n");

    // Liberamos la memoria
    for (i=0; i<N; i++)
        free(matriz[i]);

```

```

    free(matriz);
    free(vector);
    free(resultado);

    return 0;
}

```

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

```

ixjosemi@ixjosemi:~/Desktop/P_3/src$ gcc -O2 -fopenmp -o pmtv-secuencial pmtv-secuencial.c
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmtv-secuencial 3
Matriz:
3 3 3
0 3 3
0 0 3
Vector:
5 5 5
Resultado:
45 30 15
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmtv-secuencial 4
Matriz:
3 3 3 3
0 3 3 3
0 0 3 3
0 0 0 3
Vector:
5 5 5 5
Resultado:
60 45 30 15
ixjosemi@ixjosemi:~/Desktop/P_3/src$

```

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:** Las tres opciones ofrecen un rendimiento similar.

a) `static` no tiene  
`dynamic` → 1  
`guided` → 1

Lo he extraído de aquí: <https://computing.llnl.gov/tutorials/openMP/>

b) hara el numero de operaciones indicaod por `chunk * num_fila`

c) en guide, el último valor no será igual mientras que en dynamic se ejecutarán o 64 operaciones o 1.

### CÓDIGO FUENTE: pmtv-OpenMP.c

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
    #define omp_set_num_threads(int)
    #define omp_in_parallel() 0
    #define omp_set_dynamic(int)
#endif

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

    int i, j;

    //Leer argumento de entrada
    if(argc < 2){
        fprintf(stderr, "Falta size [optional debug]\n");
        exit(-1);
    }

    unsigned int N;
    N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) =
4 B)

    // Inicializamos la matriz triangular (superior)
    int *vector, *resultado, **matriz;
    vector = (int *) malloc(N*sizeof(int)); // malloc necesita el tamaño en
bytes
    resultado = (int *) malloc(N*sizeof(int)); //si no hay espacio suficiente
malloc devuelve NULL
    matriz = (int **) malloc(N*sizeof(int*));

    for (i=0; i<N; i++)
        matriz[i] = (int*) malloc(N*sizeof(int));

    for (i=0; i<N; i++){
        for (j=i; j<N; j++){
            matriz[i][j] = 3;

            vector[i] = 5;
            resultado[i]=0;
        }

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

```

    }

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

    double start, end, total;
    start = omp_get_wtime();

    // Obtenemos los resultados

    // Usamos runtime para poder variarlo luego con la variable OMP_SCHEDULE
    #pragma omp parallel for private(j) schedule(runtime)
    for (i=0; i<N; i++)
        for (j=i; j<N; j++)
            result[i] += matriz[i][j] * vector[j];

    end = omp_get_wtime();
    total = end - start;

    // Pintamos los resultados
    printf("Resultado:\n");
    for (i=0; i<N; i++)
        printf("%d ", resultado[i]);

    printf("\n");

    printf("Tiempo = %11.9f\t Primera = %d\t Ultima= %d\n",total,resultado[0],resultado[N-1]);

    // Liberamos la memoria
    for (i=0; i<N; i++)
        free(matriz[i]);

    free(matriz);
    free(vector);
    free(resultado);

    return 0;
}

```

**DESCOMPOSICIÓN DE DOMINIO:** Las filas de la matriz se reparten y cada thread calcula un valor del vector final, recorriendo una fila distinta de la matriz.

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

```

ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmtv-OpenMP 5
Matriz:
3 3 3 3 3
0 3 3 3 3
0 0 3 3 3
0 0 0 3 3
0 0 0 0 3
Vector:
5 5 5 5 5
Resultado:
75 60 45 30 15
Tiempo = 0.003644674      Primera = 75      Ultima=15
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmtv-OpenMP 6
Matriz:
3 3 3 3 3 3
0 3 3 3 3 3
0 0 3 3 3 3
0 0 0 3 3 3
0 0 0 0 3 3
0 0 0 0 0 3
Vector:
5 5 5 5 5 5
Resultado:
90 75 60 45 30 15
Tiempo = 0.004445894      Primera = 90      Ultima=15
ixjosemi@ixjosemi:~/Desktop/P_3/src$

```

## TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

### SCRIPT: pmtv-OpenMP\_atcgrid.sh

```

#!/bin/bash

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

export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="dynamic,64"

```

```

echo "dynamic y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

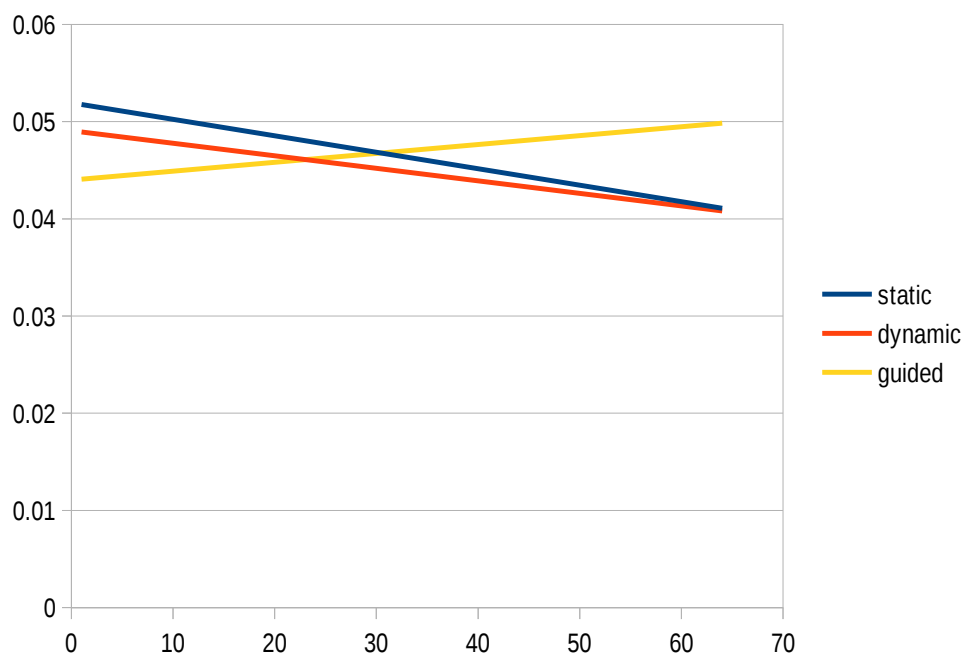
export OMP_SCHEDULE="guided,10"
echo "guided y chunk 1"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

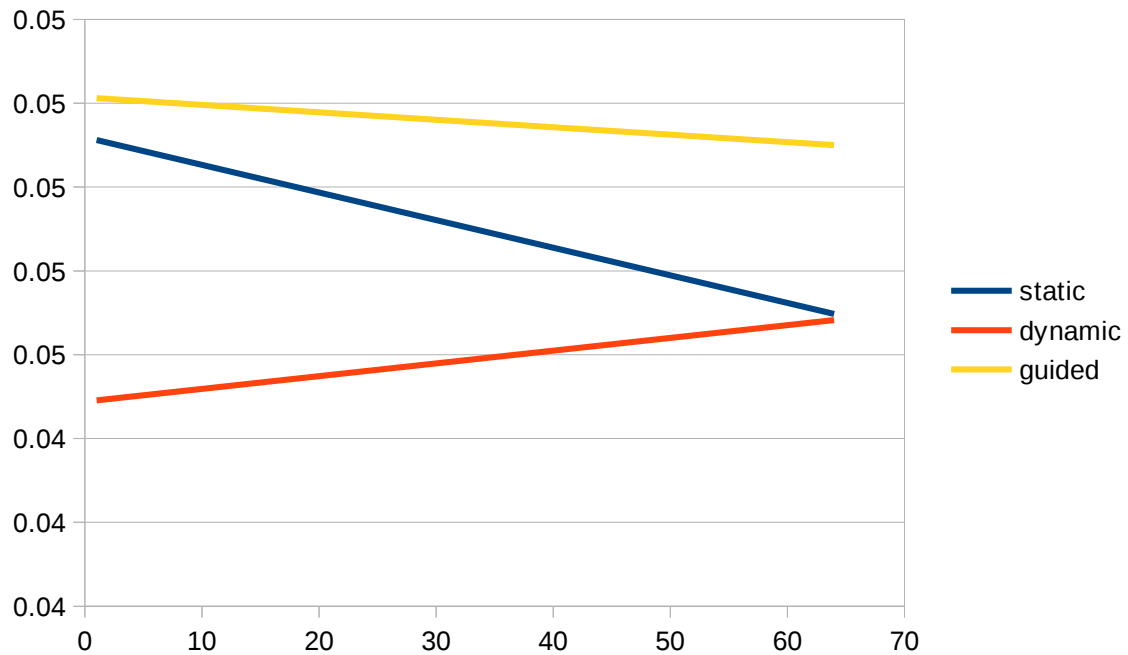
export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
$PBS_O_WORKDIR/pmtv-OpenMP 15360

```

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

Chunk	Static	Dynamic	Guided
por defecto	0.045925681	0.043508332	0.049312804
1	0.051764175	0.048922013	0.044071957
64	0.041075576	0.040813267	0.049844872
Chunk	Static	Dynamic	Guided
por defecto	0.051873066	0.055214081	0.043490164
1	0.047561202	0.044456493	0.048060302
64	0.045487382	0.045412093	0.047499359





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

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

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

    unsigned i, j, k;

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

    unsigned int N;
    M = atoi(argv[1]); // Máximo N = 2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    int **a, **b, **c;
    a = (int **) malloc(N*sizeof(int*));
    b = (int **) malloc(N*sizeof(int*));
    c = (int **) malloc(N*sizeof(int*));

    for (i=0; i<N; i++){
        a[i] = (int *) malloc(N*sizeof(int));
        b[i] = (int *) malloc(N*sizeof(int));
        c[i] = (int *) malloc(N*sizeof(int));
    }
```

```

    }

    // Inicializamos las matrices
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            a[i][j] = 0;
            b[i][j] = 2;
            c[i][j] = 2;
        }
    }

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    // 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];

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

    // Pintamos la primera y la ultima linea de la matriz resultante
    printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,a[0][0],a[N-1]
[N-1]);

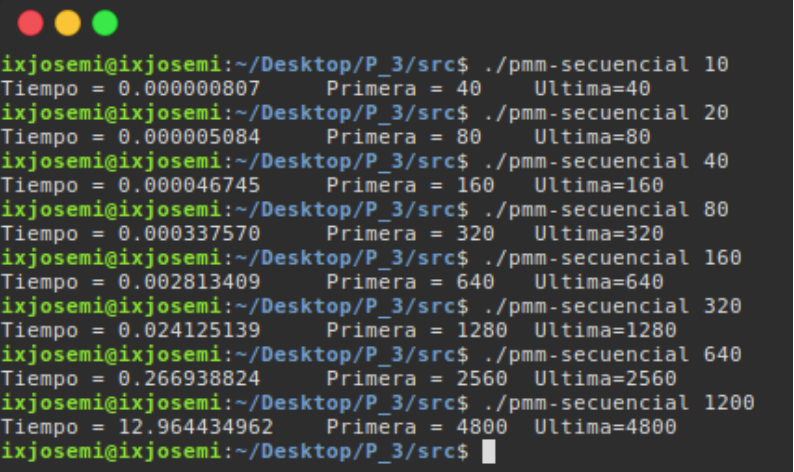
    // Liberamos la memoria
    for (i=0; i<N; i++){
        free(a[i]);
        free(b[i]);
        free(c[i]);
    }

    free(a);
    free(b);
    free(c);

    return 0;
}

```

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



```

ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-secuencial 10
Tiempo = 0.000000807    Primera = 40    Ultima=40
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-secuencial 20
Tiempo = 0.000005084    Primera = 80    Ultima=80
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-secuencial 40
Tiempo = 0.000046745    Primera = 160   Ultima=160
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-secuencial 80
Tiempo = 0.000337570    Primera = 320   Ultima=320
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-secuencial 160
Tiempo = 0.002813409    Primera = 640   Ultima=640
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-secuencial 320
Tiempo = 0.024125139    Primera = 1280  Ultima=1280
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-secuencial 640
Tiempo = 0.266938824    Primera = 2560  Ultima=2560
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-secuencial 1200
Tiempo = 12.964434962   Primera = 4800  Ultima=4800
ixjosemi@ixjosemi:~/Desktop/P_3/src$

```



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:** Las filas de la matriz resultado se reparten y cada thread recorre las filas de la primera matriz, pero todos los threads recorren sus columnas y todas las filas y columnas de la segunda matriz

**CÓDIGO FUENTE:** pmm-OpenMP.c

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
    #define omp_set_num_threads(int)
    #define omp_in_parallel() 0
    #define omp_set_dynamic(int)
#endif

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

    unsigned i, j, k;

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

    unsigned int N;
    N = atoi(argv[1]); // Máximo N = 2^32-1=4294967295 (sizeof(unsigned int) =
4 B)

    int **a, **b, **c;
    a = (int **) malloc(N*sizeof(int*));
    b = (int **) malloc(N*sizeof(int*));
    c = (int **) malloc(N*sizeof(int*));

    for (i=0; i<N; i++){
        a[i] = (int *) malloc(N*sizeof(int));
        b[i] = (int *) malloc(N*sizeof(int));
        c[i] = (int *) malloc(N*sizeof(int));
    }

    // Inicializamos las matrices
    #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] = 2;
            c[i][j] = 2;
        }
    }

    double start, end, total;
    start = omp_get_wtime();
```

```

// 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];

end = omp_get_wtime();
total = end - start;

// Pintamos la primera y la ultima linea de la matriz resultante
printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",total,a[0][0],a[N-1]
[N-1]);

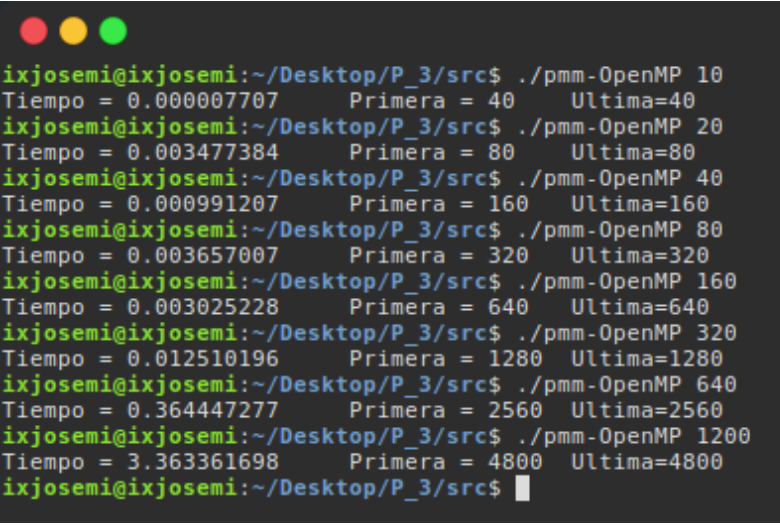
// Liberamos la memoria
for (i=0; i<N; i++){
    free(a[i]);
    free(b[i]);
    free(c[i]);
}

free(a);
free(b);
free(c);

return 0;
}

```

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



```

ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-OpenMP 10
Tiempo = 0.000007707    Primera = 40    Ultima=40
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-OpenMP 20
Tiempo = 0.003477384    Primera = 80    Ultima=80
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-OpenMP 40
Tiempo = 0.000991207    Primera = 160   Ultima=160
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-OpenMP 80
Tiempo = 0.003657007    Primera = 320   Ultima=320
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-OpenMP 160
Tiempo = 0.003025228    Primera = 640   Ultima=640
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-OpenMP 320
Tiempo = 0.012510196    Primera = 1280  Ultima=1280
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-OpenMP 640
Tiempo = 0.364447277    Primera = 2560  Ultima=2560
ixjosemi@ixjosemi:~/Desktop/P_3/src$ ./pmm-OpenMP 1200
Tiempo = 3.363361698    Primera = 4800  Ultima=4800
ixjosemi@ixjosemi:~/Desktop/P_3/src$

```

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**

```
#!/bin/bash

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

export OMP_DYNAMIC=FALSE

echo "Tiempo secuencial con 100"
$PBS_O_WORKDIR/pmm-secuencial 100

echo "Tiempo secuencial con 250"
$PBS_O_WORKDIR/pmm-secuencial 250

echo "Tiempo secuencial con 500"
$PBS_O_WORKDIR/pmm-secuencial 500

echo "Tiempo secuencial con 1000"
$PBS_O_WORKDIR/pmm-secuencial 1000

echo "Tiempo secuencial con 1500"
$PBS_O_WORKDIR/pmm-secuencial 1500

export OMP_NUM_THREADS=4
echo "Tiempo con cuatro threads con 100"
$PBS_O_WORKDIR/pmm-OpenMP 100

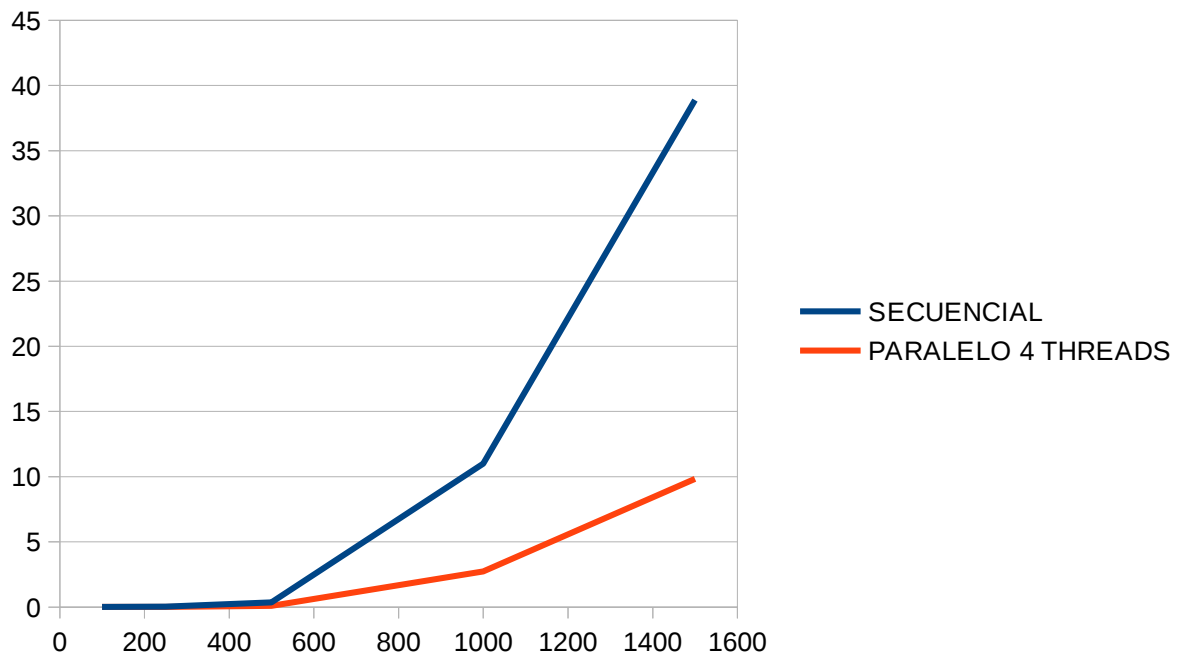
echo "Tiempo con cuatro threads con 250"
$PBS_O_WORKDIR/pmm-OpenMP 250

echo "Tiempo con cuatro threads con 500"
$PBS_O_WORKDIR/pmm-OpenMP 500

echo "Tiempo con cuatro threads con 1000"
$PBS_O_WORKDIR/pmm-OpenMP 1000

echo "Tiempo con cuatro threads con 1500"
$PBS_O_WORKDIR/pmm-OpenMP 1500
```

	SECUENCIAL	PARALELO 4 THREADS
100	0.002077294	0.000804752
250	0.030015221	0.011042278
500	0.350686498	0.102463845
1000	10.993756350	2.740818869
1500	38.881811189	9.838202674



### ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

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

```
#!/bin/bash

echo "secuencial"
./pmm-secuencial 100
./pmm-secuencial 250
./pmm-secuencial 500
./pmm-secuencial 1000
./pmm-secuencial 1500

echo "paralelo con 4 threads"
export OMP_NUM_THREADS=4
./pmm-OpenMP 100
./pmm-OpenMP 250
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500
```

	SECUENCIAL	PARALELO 4 THREADS
100	0.000725635	0.000224663
250	0.011494127	0.003980882
500	0.121298168	0.038716550
1000	3.196096145	0.837176082
1500	27.852006557	7.412566671

