

BP3-AC.pdf



varo12ff_490302



Arquitectura de Computadores



2º Grado en Ingeniería Informática



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

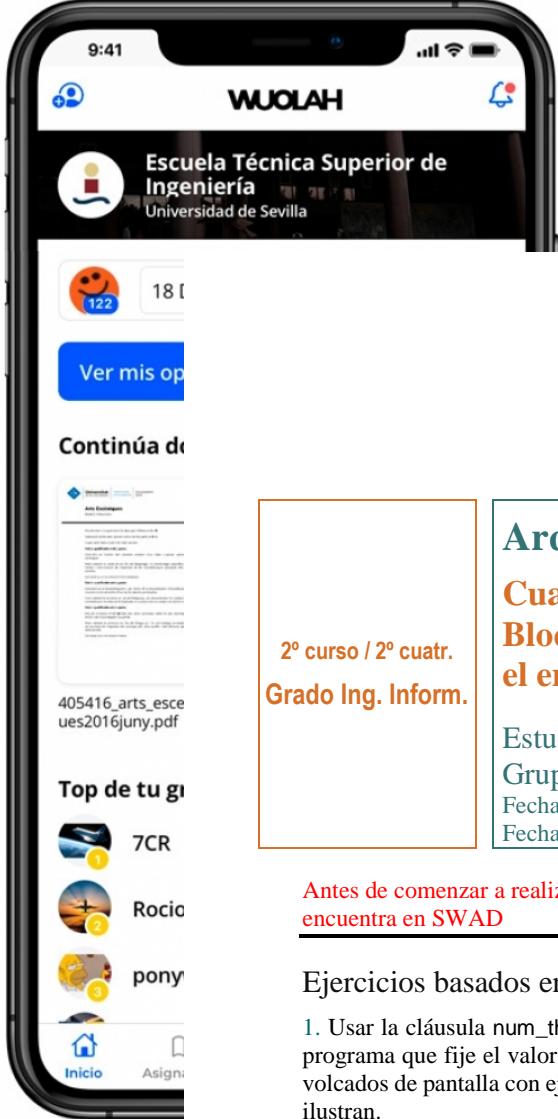


Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.





**KEEP
CALM
AND
ESTUDIA
UN POQUITO**



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

Available on the
App Store

GET IT ON
Google Play

Continúa d

2º curso / 2º cuatr.
Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

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

Estudiante (nombre y apellidos):

Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

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

Ejercicios basados en los ejemplos del seminario práctico

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

CAPTURA CÓDIGO FUENTE: if-clauseModificado.c

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

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

    int i, n = 20, tid;
    int a[n], suma = 0, sumalocal, x;

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

    if(argc < 3){
        fprintf(stderr, "[ERROR]-Falta número de hebras\n");
        exit(-2);
    }

    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:

```
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer1$ ./if-clauseModificado 2 5
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=1
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer1$ ./if-clauseModificado 5 7
thread 0 suma de a[0]=0 sumalocal=0
thread 4 suma de a[4]=4 sumalocal=4
thread 3 suma de a[3]=3 sumalocal=3
thread 2 suma de a[2]=2 sumalocal=2
thread 1 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=10
```

RESPUESTA:

Cuando le pasamos un tamaño de vector menor que 4 (en este caso 2) vemos que la hebra que realiza el trabajo es la hebra master mientras que en caso de usar un vector mayor no ocurre esto.

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	1	0	0	0

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

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

Tabla 2 .

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

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

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

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

RESPUESTA:

Las diferencias entre `static`, `dynamic` y `guided` es que `static` y `dynamic` utilizan el valor de `chunk` como el número de iteraciones que forman el tamaño del bloque mientras que `guided` lo utiliza como el valor del tamaño mínimo del bloque.

Las diferencias que aquí se observan es que `static`, a diferencia de `dynamic` y `guided`, las iteraciones se realizan según el identificador de la hebra y va en orden. Cuando el `chunk` era 1, la hebra x ejecutaba 1 vez la iteración; cuando es 2, la hebra ejecuta 2 iteraciones seguidas. En `dynamic` y `guided`, vemos que las iteraciones no siguen un orden de identificador, sino que se asigna a la iteración una hebra según vayan llegando.

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

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado.c`



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

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

```
#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, modif;

    omp_sched_t tipo; // omp_sched_t es realmente una estructura que devuelve un entero

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

    n = atoi(argv[1]);

    if(n>200)
        n=200;

    chunk = atoi(argv[2]);

    int dyn_var = omp_get_dynamic(); // al ser un booleano en C, devuelve un entero
    int nthreads_var = omp_get_max_threads();
    int thread_limit_var = omp_get_thread_limit();

    omp_get_schedule(&tipo, &modif); //void omp_get_schedule(omp_sched_t *kind, int *chunk_size);

    // omp_sched_t devuelve un entero

    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] suma=%d \n", omp_get_thread_num(), i, suma);
}

#pragma omp single

printf("\nthread %d imprime dyn-var %d \n", omp_get_thread_num(), dyn_var);
printf("thread %d imprime nthreads-var %d \n", omp_get_thread_num(), nthreads_var);
printf("thread %d imprime thread-limit-var %d \n", omp_get_thread_num(),
thread_limit_var);

printf("thread %d imprime run-sched-var mod %d \n", omp_get_thread_num(), modif);

if(tipo == omp_sched_static)
    printf("El tipo es estático \n");

if(tipo == omp_sched_dynamic)
    printf("El tipo es dinámico \n");

if(tipo == omp_sched_guided)
    printf("El tipo es guided \n");

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

if(tipo == omp_sched_static) printf("El tipo es estático \n");
if(tipo == omp_sched_dynamic)printf("El tipo es dinámico \n");

if(tipo == omp_sched_guided) printf("El tipo es guided \n");
}
```

CAPTURAS DE PANTALLA:

```
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer3$ ./scheduled-clauseModificado 8 4
thread 2 suma a[0] suma=0
thread 2 suma a[1] suma=1
thread 2 suma a[2] suma=3
thread 2 suma a[3] suma=6
thread 4 suma a[4] suma=4
thread 4 suma a[5] suma=9
thread 4 suma a[6] suma=15
thread 4 suma a[7] suma=22

thread 0 imprime dyn-var 0
thread 0 imprime nthreads-var 8
thread 0 imprime thread-limit-var 2147483647
thread 0 imprime run-sched-var mod 1
El tipo es dinámico

Fuera de 'parallel for' suma 22
dyn-var 0
nthreads-var 8
thread-limit-var 2147483647
run-sched-var mod 1
El tipo es dinámico
```

```
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer3$ export OMP_DYNAMIC=TRUE
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer3$ export OMP_NUM_THREADS=8
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer3$ export OMP_NUM_THREADS=4
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer3$ export OMP_THREAD_LIMIT=24
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer3$ export OMP_SCHEDULE="static,4"
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer3$ ./scheduled-clauseModificado 15 3

thread 3 suma a[12] suma=12
thread 3 suma a[13] suma=25
thread 3 suma a[14] suma=39
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 7 suma a[3] suma=3
thread 7 suma a[4] suma=7
thread 7 suma a[5] suma=12
thread 2 suma a[6] suma=6
thread 2 suma a[7] suma=13
thread 2 suma a[8] suma=21
thread 5 suma a[9] suma=9
thread 5 suma a[10] suma=19
thread 5 suma a[11] suma=30

thread 0 imprime dyn-var 1
thread 0 imprime nthreads-var 8
thread 0 imprime thread-limit-var 24
thread 0 imprime run-sched-var mod 4
El tipo es estático

Fuera de 'parallel for' suma 39
dyn-var 1
nthreads-var 8
thread-limit-var 24
run-sched-var mod 4
El tipo es estático
```

RESPUESTA:

No influye el hecho de que sea fuera o dentro de la directiva parallel para el resultado.

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

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado4.c

```
#include <stdio.h>
#include <stdlib.h>
#ifndef _OPENMP
#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, modif;

    omp_sched_t tipo; // omp_sched_t es realmente una estructura que devuelve un entero

    if(argc < 3){

        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    n = atoi(argv[1]);

    if(n>200)
        n=200;

    chunk = atoi(argv[2]);

    int dyn_var = omp_get_dynamic(); // al ser un booleano en C, devuelve un entero
    int nthreads_var = omp_get_max_threads();
    int thread_limit_var = omp_get_thread_limit();

    omp_get_schedule(&tipo, &modif); //void omp_get_schedule(omp_sched_t *kind, int *chunk_size);
    // omp_sched_t devuelve un entero

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



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

```
#pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic, chunk)

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

    printf("\nthread %d suma a[%d] suma=%d \n", omp_get_thread_num(), i, suma);
    printf("omp_get_num_threads() = %d \n", omp_get_num_threads());
}

printf("\nDentro de la región parallel:\n");
printf("omp_get_num_threads() = %d \n", omp_get_num_threads());
printf("omp_get_num_procs() = %d \n", omp_get_num_procs());
printf("omp_in_parallel() = %d \n", omp_in_parallel());
printf("\nFuera de la región parallel: \n");
printf("omp_get_num_threads() = %d \n", omp_get_num_threads());
printf("omp_get_num_procs() = %d \n", omp_get_num_procs());
printf("omp_in_parallel() = %d \n", omp_in_parallel());

}
```

CAPTURAS DE PANTALLA:

```
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer4$ ./scheduled-clauseModificado4 7 2

thread 6 suma a[0] suma=0
omp_get_num_threads() = 8

thread 6 suma a[1] suma=1
omp_get_num_threads() = 8

thread 1 suma a[4] suma=4
omp_get_num_threads() = 8

thread 1 suma a[5] suma=9
omp_get_num_threads() = 8

thread 3 suma a[2] suma=2
omp_get_num_threads() = 8

thread 3 suma a[3] suma=5
omp_get_num_threads() = 8

thread 2 suma a[6] suma=6
omp_get_num_threads() = 8

Dentro de la región parallel:
omp_get_num_threads() = 1
omp_get_num_procs() = 8
omp_in_parallel() = 0

Fuera de la región parallel:
omp_get_num_threads() = 1
omp_get_num_procs() = 8
omp_in_parallel() = 0
```

RESPUESTA:

El `omp_get_num_threads` es 8 en el bucle `for` pero dentro de la directiva `parallel` es 1, aunque las demás directivas se mantienen igual.

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

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado5.c

```
#include <stdio.h>
#include <stdlib.h>
#ifndef _OPENMP
#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, modif;

    omp_sched_t tipo; // omp_sched_t es realmente una estructura que devuelve un entero

    if(argc < 3){

        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    n = atoi(argv[1]);

    if(n>200)
        n=200;

    chunk = atoi(argv[2]);

    int dyn_var = omp_get_dynamic(); // al ser un booleano en C, devuelve un entero
    int nthreads_var = omp_get_max_threads();
    int thread_limit_var = omp_get_thread_limit();

    omp_get_schedule(&tipo, &modif); //void omp_get_schedule(omp_sched_t *kind, int *chunk_size);

    // omp_sched_t devuelve un entero

    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] suma=%d \n", omp_get_thread_num(), i, suma);
    }

#pragma omp single

    omp_get_schedule(&tipo, &modif);

    printf("\nValores actuales: dyn-var: %d, nthreads-var: %d, run-sched-var[tipo: %d,modif: %d]\n", omp_get_dynamic(), omp_get_max_threads(), tipo, modif);

    omp_set_dynamic(9);

    omp_set_num_threads(2);

    omp_set_schedule(omp_sched_guided, chunk + 3);

    omp_get_schedule(&tipo, &modif);

    printf("\nValores modificados: dyn-var: %d, nthreads-var: %d, run-sched-var[tipo: %d, modif: %d]\n", omp_get_dynamic(), omp_get_max_threads(), tipo, modif);

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

CAPTURAS DE PANTALLA:

```
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer5$ ./scheduled-clauseModificado5 7 4
thread 2 suma a[0] suma=0
thread 2 suma a[1] suma=1
thread 2 suma a[2] suma=3
thread 2 suma a[3] suma=6
thread 3 suma a[4] suma=4
thread 3 suma a[5] suma=9
thread 3 suma a[6] suma=15

Valores actuales: dyn-var: 1, nthreads-var: 8, run-sched-var[tipo: 1,modif: 4]

Valores modificados: dyn-var: 1, nthreads-var: 2, run-sched-var[tipo: 3, modif: 7]
Fuera de 'parallel for' suma=15
```

Resto de ejercicios

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

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

CAPTURA CÓDIGO FUENTE: pmtv-secuencial.c

```
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()

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

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

    //Leer argumento de entrada (nº de componentes del vector)

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

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

    int i, j;
    int *v, *mv, **m;

    v = (int*) malloc(N*sizeof(int)); // si no hay suficiente espacio, malloc devuelve NULL
    mv = (int*) malloc(N*sizeof(int));
    m = (int **)malloc(N*sizeof(int *)); //asignamos memoria a una matriz de punteros que

    //referenciarán a cada una de las filas

    for (i=0;i<N;i++) // asignamos memoria para cada una de las filas
        m[i] = (int *) malloc (N*sizeof(int));

    // Comprobamos que no ha habido error en la reserva de espacio

    if ( v==NULL || mv==NULL || m==NULL ){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-2);
    }
}
```



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

Available on the App Store GET IT ON Google Play

```
for (i=0; i<N; i++){
    for(j=0; j<N; j++){
        if(j>i)
            printf("%d",m[i][j]);

        else
            printf("0");

    }

    printf("\n");
}

//Vemos el vector

printf("Vector: \n");

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

printf("\n");

//Guardamos en ncgt el tiempo que ha tardado

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

//Imprimir resultados de la suma y el tiempo de ejecucion

printf("Tamaño de la matriz y vector: %d\t / Tiempo(seg) %11.9f\n", N, ncgt);
```

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

free(m);
free(v);
free(mv);

return 0;

}
```

CAPTURAS DE PANTALLA:

```
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer6$ ./pmtv-secuencial 6
Matriz:
099999
009999
000999
000099
000009
000000
Vector:
1089072543618
Tamaño de la matriz y vector: 6 / Tiempo(seg) 0.000001700
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer6$ ./pmtv-secuencial 12
Matriz:
099999999999
009999999999
000999999999
000099999999
000009999999
000000999999
000000099999
000000009999
000000000999
000000000099
000000000009
0000000000009
0000000000000
Vector:
2161981801621441261089072543618
Tamaño de la matriz y vector: 12 / Tiempo(seg) 0.000002900
```

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:

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

```
1 #include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
2 #include <stdio.h> // biblioteca donde se encuentra la función printf()
3 #ifndef _OPENMP
4     #include <omp.h>
5 #else
6     #define omp_get_thread_num() 0
7     #define omp_get_num_threads() 1
8     #define omp_set_num_threads(int)
9     #define omp_in_parallel() 0
10    #define omp_set_dynamic(int)
11 #endif
12
13 int main(int argc, char** argv){
14
15     double t_ini, t_final, t_total;
16
17     if (argc<2){
18         printf("Faltan no componentes del vector\n");
19         exit(-1);
20     }
21
22     unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int)=4 B)
23     int i, j;
24     int *v, **mv, **m;
25     v = (int*) malloc(N*sizeof(int)); // si no hay suficiente espacio, malloc devuelve NULL
26     mv = (int**) malloc(N*sizeof(int));
27     m = (int **)malloc(N*sizeof(int *)); //asignamos memoria a una matriz de punteros que referenciaran a cada una de las filas
28
29     for (i=0;i<N;i++) // asignamos memoria para cada una de las filas
30         m[i] = (int *) malloc (N*sizeof(int));
31
32     // Comprobamos que no ha habido error en la reserva de espacio
33     if ( v==NULL || mv==NULL || m==NULL ){
34         printf("Error en la reserva de espacio para la matriz y el vector\n");
35         exit(-2);
36     }
37
38     printf("Tamaño de la matriz y vector:%d\t / Tiempo(seg.) %11.9f\n", N, t_total);
39     for(i=0; i<N; i++)
40         free(m[i]);
41     free(m); // libera el espacio de la matriz
42     free(v); // libera el espacio reservado para v
43     free(mv); // libera el espacio del vector mv resultado
44
45     return 0;
46 }
```



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

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

```
for(i=0; i<N; i++){
    if(m[i] == NULL){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-3);
    }

    // Inicializamos la matriz triangular superior(m), el vector(v) y el vector resultante(mv)
#pragma omp parallel for schedule(runtime)
for(i=0; i<N; i++){
    for(j=i; j<N; j++){
        m[i][j] = 9;
    }
    v[i] = 2;           //los valores dependen de N
    mv[i] = 0;          // lo inicializo a 0 para que no contenga basura
}

t_ini = omp_get_wtime();

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

t_final = omp_get_wtime();

t_total = t_final - t_ini;
```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

```
[a2estudiante4@atcgrid ejer7]$ ./pmtv-Open.sh
static y chunk por defecto
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.029147699
static y chunk 1
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.018891394
static y chunk 64
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.018194538
dynamic y chunk por defecto
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.020757992
dynamic y chunk 1
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.020757794
dynamic y chunk 64
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.017860174
guided y chunk por defecto
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.028177660
guided y chunk 1
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.031257056
guided y chunk 64
Tamaño de la matriz y vector:15360      / Tiempo(seg.) 0.028288417
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

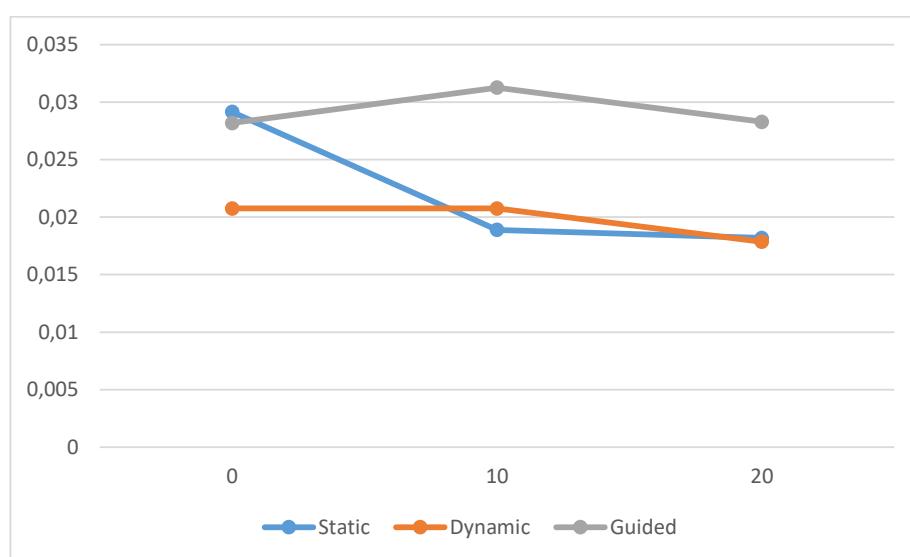
```
#!/bin/bash

#PBS -N ej7_atcgrid
#PBS -q ac
#echo "Id$PBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
#echo "Id$PBS_O_WORKDIR del trabajo: $PBS_JOBID"
#echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
#echo "Nodo que ejecuta qsub: $PBS_O_HOST"
#echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
#echo "Cola: $PBS_QUEUE"
#echo "Nodos asignados al trabajo:"
#cat $PBS_NODEFILE
export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
./pmvt-OpenMP 15360
export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
./pmvt-OpenMP 15360
export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
./pmvt-OpenMP 15360
export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
./pmvt-OpenMP 15360
export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
./pmvt-OpenMP 15360
export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk 64"
./pmvt-OpenMP 15360
export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
./pmvt-OpenMP 15360
export OMP_SCHEDULE="guided,1"
echo "guided y chunk 1"
./pmvt-OpenMP 15360
export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
./pmvt-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	0,029147699	0,020757992	0,02817766
1	0,018891394	0,020757794	0,031257056
64	0,018194538	0,017860174	0,028288417

Chunk	Static	Dynamic	Guided
por defecto	0,023948964	0,020753045	0,028171934
1	0,018505722	0,02078658	0,028041657
64	0,018168621	0,017857421	0,030882817



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

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

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

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

```
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#ifndef _OPENMP
    #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, k;
    double t_ini, t_fin, t_total;

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

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

    int **mr, **m1, **m2;
    mr = (int **) malloc(N*sizeof(int *));
    m1 = (int **) malloc(N*sizeof(int *));
    m2 = (int **) malloc(N*sizeof(int *));

    for (i=0;i<N;i++){
        mr[i] = (int *) malloc (N*sizeof(int));
        m1[i] = (int *) malloc (N*sizeof(int));
        m2[i] = (int *) malloc (N*sizeof(int));
    }
}
```



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play

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

```
// Comprobamos que no ha habido error en la reserva de espacio
if ( mr==NULL || m1==NULL || m2==NULL ){
    printf("Error en la reserva de espacio para la matriz y el vector\n");
    exit(-2);
}

for(i=0; i<N; i++)
    if(mr[i] == NULL || m1[i] == NULL || m2[i] == NULL){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-3);
    }

// Inicializamos las matrices
for(i=0; i<N; i++){
    for(j=i; j<N; j++){
        mr[i][j] = 0;
        m1[i][j] = 28+i;
        m2[i][j] = 2+i;
    }
}

t_ini = omp_get_wtime();

// calculamos la multiplicación de la matriz por el vector
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        for(k=0; k<N; k++){
            mr[i][j] += m1[i][k] * m2[k][j];
        }
    }
}
t_fin = omp_get_wtime();

t_total = t_fin - t_ini;

printf("Tamaño de la matriz y vector:%d\t / Tiempo(seg.) %11.9f\n", N, t_total);

// visualizamos la primera y última línea de la matriz resultante
printf("Tiempo = %11.9f\t Primera linea= %d\t Última linea= %d\n", t_total, mr[0][0],mr[N-1][N-1]);
return 0;
}
```

CAPTURAS DE PANTALLA:

```
alvaro@DESKTOP-S91G5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer8$ ./pmm-secuencial 5
Tamaño de la matriz y vector:5      / Tiempo(seg.) 0.000005000
Tiempo = 0.000005000      Primera linea= 1088422840          Última linea= 192
alvaro@DESKTOP-S91G5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer8$ ./pmm-secuencial 3
Tamaño de la matriz y vector:3      / Tiempo(seg.) 0.000005300
Tiempo = 0.000005300      Primera linea= 812      Última linea= 178
alvaro@DESKTOP-S91G5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer8$ ./pmm-secuencial 9
Tamaño de la matriz y vector:9      / Tiempo(seg.) 0.000005400
Tiempo = 0.000005400      Primera linea= 56      Última linea= 360
alvaro@DESKTOP-S91G5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer8$ ./pmm-secuencial 12
Tamaño de la matriz y vector:12     / Tiempo(seg.) 0.000006700
Tiempo = 0.000006700      Primera linea= 56      Última linea= 507
```

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

10. DESCOMPOSICIÓN DE DOMINIO:

CAPTURA CÓDIGO FUENTE: pmm-OpenMP.c

```
1 #include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
2 #include <stdio.h> // biblioteca donde se encuentra la función printf()
3 #ifdef _OPENMP
4     #include <omp.h>
5 #else
6     #define omp_get_thread_num() 0
7     #define omp_get_num_threads() 1
8     #define omp_set_num_threads(int)
9     #define omp_in_parallel() 0
10    #define omp_set_dynamic(int)
11 #endif
12
13 int main(int argc, char** argv){
14     int i, j, k;
15     double t_ini, t_fin, t_total;
16
17     if (argc<2){
18         printf("Faltan no componentes del vector\n");
19         exit(-1);
20     }
21     unsigned int N = atoi(argv[1]); // Máximo N = 2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
22     int **mr, **m1, **m2;
23
24     mr = (int **) malloc(N*sizeof(int *));
25     m1 = (int **) malloc(N*sizeof(int *));
26     m2 = (int **) malloc(N*sizeof(int *));
27
28     for (i=0;i<N;i++){
29         mr[i] = (int *) malloc (N*sizeof(int));
30         m1[i] = (int *) malloc (N*sizeof(int));
31         m2[i] = (int *) malloc (N*sizeof(int));
32     }
33 }
```

```

// Comprobamos que no ha habido error en la reserva de espacio
if ( mr==NULL || m1==NULL || m2==NULL ){
    printf("Error en la reserva de espacio para la matriz y el vector\n");
    exit(-2);
}
for(i=0; i<N; i++)
    if(mr[i] == NULL || m1[i] == NULL || m2[i] == NULL){
        printf("Error en la reserva de espacio para la matriz y el vector\n");
        exit(-3);
    }

// Inicializamos las matrices
#pragma omp parallel for private(j)
for(i=0; i<N; i++){
    for(j=i; j<N; j++){
        mr[i][j] = 0;
        m1[i][j] = 28+i;
        m2[i][j] = 2+i;
    }
}

t_ini = omp_get_wtime();

// Calculamos la multiplicación de la matriz por el vector
#pragma omp parallel for private(j,k)
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        for(k=0; k<N; k++){
            mr[i][j] += m1[i][k] * m2[k][j];
        }
    }
}

t_fin = omp_get_wtime();

t_total = t_fin-t_ini;

printf("Tamaño de la matriz y vector:%d\t / Tiempo(seg.) %11.9f\n", N, t_total);

// Visualizamos la primera y última línea de la matriz resultante
printf("Tiempo = %11.9f\t Primera línea= %d\t Última línea= %d\n", t_total, mr[0][0], mr[N-1][N-1]);
return 0;
}

```

CAPTURAS DE PANTALLA:

```
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer9$ ./pmm-OpenMP 4
Tamaño de la matriz y vector:4 / Tiempo(seg.) 0.000011600
Tiempo = 0.000011600 Primera línea= 1088422840 Última línea= 155
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer9$ ./pmm-OpenMP 8
Tamaño de la matriz y vector:8 / Tiempo(seg.) 0.000011200
Tiempo = 0.000011200 Primera línea= 56 Última línea= 315
alvaro@DESKTOP-S91GC5N:/mnt/c/Users/Alvaro/Desktop/Uni/Prácticas/2º Año/AC/bp3/ejer9$ ./pmm-OpenMP 6
Tamaño de la matriz y vector:6 / Tiempo(seg.) 0.000010900
Tiempo = 0.000010900 Primera línea= 56 Última línea= 231
```

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

ESTUDIO DE ESCALABILIDAD EN atcgrid:

SCRIPT: pmm-OpenMP_atcgrid.sh

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

echo "Secuencial:"
./pmm-secuencial 100
./pmm-secuencial 500
./pmm-secuencial 800
./pmm-secuencial 1500
```

Tamaño	Secuencial	2 threads	6 threads	11 threads
100	0,001286156	0,00074026	0,000473671	0,000538345
500	0,094253514	0,05324408	0,034702905	0,028946891
800	0,414102711	0,22275667	0,359373529	0,259472474
1500	6,471019685	3,072363801	2,417482667	3,525543764



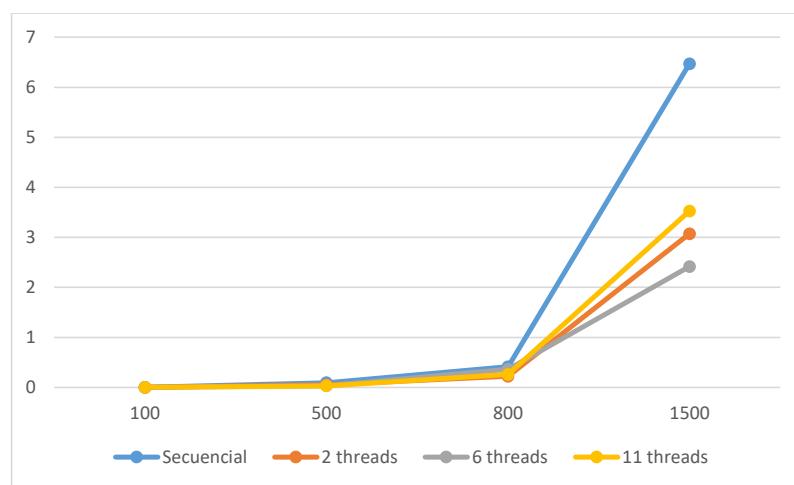
Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

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



ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```

echo "Secuencial:"
./pmm-secuencial 100
./pmm-secuencial 500
./pmm-secuencial 800
./pmm-secuencial 1500

echo "\n"
echo "Paralelo con 2 threads"
export OMP_NUM_THREADS=2
./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 800
./pmm-OpenMP 1500

echo "\n"
echo "Paralelo con 3 threads"
export OMP_NUM_THREADS=3
./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 800
./pmm-OpenMP 1500

echo "\n"
echo "Paralelo con 4 threads"
export OMP_NUM_THREADS=4
./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 800
./pmm-OpenMP 1500

```

Tamaño	Secuencial	2 threads	3 threads	4 threads
100	0,0009117	0,0004247	0,0003677	0,000211
500	0,1687144	0,0677819	0,0735047	0,053556
800	0,7793759	0,4004166	0,3361919	0,391254
1500	8,4592296	5,7423776	5,2671958	4,1985596

