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):
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

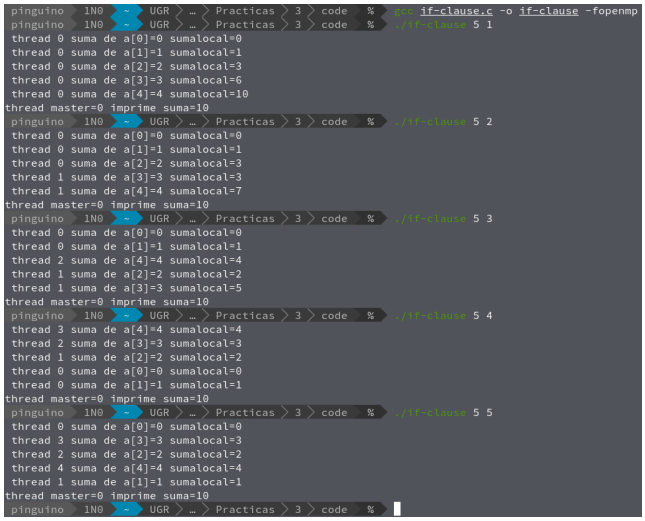
1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad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

CÓDIGO FUENTE: `if-clauseModificado.c`

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

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

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

#pragma omp parallel if(n>4) default(none) num_threads(x) \
   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);
   }

   return(0);
}
```

RESPUESTA: En la captura de pantalla siguiente se ejecuta el mismo programa varias veces pasando como primer argumento (número de iteraciones) siempre el número 5 y variando el segundo argumento (que será el 'x' del enunciado y representa el número de threads con que se ejecuta el programa). Se puede apreciar en la captura que el número de threads que ejecuta el código varía al variar el argumento, basta con ver que en la primera ejecución todas las iteraciones las realiza el thread 0 y en la última cada thread, desde el 0 hasta el 4, ejecuta una iteración.

CAPTURAS DE PANTALLA:



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	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	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	0	0	1	1	1
11	1	1	0	0	0	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	0	0	1	0	0
15	1	1	1	0	0	0	1	0	0

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

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

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

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

RESPUESTA:

En static las iteraciones se dividen en unidades de chunk iteraciones, que son repartidas a cada thread usando round-robin. En dynamic, la distribución se hace en tiempo de ejecución (la unidad de división sigue siendo de chunk iteraciones) y en Guided, la distribución se hace en tiempo de ejecución (como en dynamic) pero las unidades no son de chunk iteraciones sino que van variando (empiezan tomando valores grandes y menguan hasta repartir todas las iteraciones, el tamaño de bloque se calcula como las iteraciones que faltan partido el número de threads).

Así, podemos decir que dynamic y guided provocan una mayor sobrecarga que static en el reparto de iteraciones (guided menos que dynamic) pero a cambio permiten que los threads no queden ociosos cuando hay mucha diferencia entre el tiempo de ejecución de unas iteraciones y otras.

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

int main(int argc, char **argv) {
    int i, n=200,chunk,a[n],suma=0;
    if(argc < 3)     {
        fprintf(stderr,"\nFalta iteraciones o chunk \n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);

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

    omp_sched_t schedule_type;
    int chunk_size;


    printf("\nAntes de la región parallel\n----------------------\n") ;

    printf("dyn-var: ") ;
    if (omp_get_dynamic()) printf("True\n");
    else printf("False\n");
    printf("nthreads-var: %d\n", omp_get_max_threads());
    printf("thread-limit-var: %d",omp_get_thread_limit());

    omp_get_schedule(&schedule_type, &chunk_size);

    printf("\nrun-sched-var: (modifier) %d (type)",
           chunk_size);

    switch(schedule_type){
    case 1:
        printf("static");
        break;
    case 2:
        printf("dynamic");
        break;
    case 3:
        printf("guided");
        break;
    case 4:
        printf("auto");
        break;
    }

    printf("\n") ;

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

    for (i=0; i<n; i++)
    {
        if(i == 0){
            {
                printf("\nDentro de la región parallel\n----------------------\n") ;

                printf("dyn-var: ") ;
                if (omp_get_dynamic()) printf("True\n");
                else printf("False\n");
                printf("nthreads-var: %d\n", omp_get_max_threads());
                printf("thread-limit-var: %d",omp_get_thread_limit());

                omp_get_schedule(&schedule_type, &chunk_size);

                printf("\nrun-sched-var: (modifier) %d (type)",
                       chunk_size);

                switch(schedule_type){
                case 1:
                    printf("static");
                    break;
                case 2:
                    printf("dynamic");
                    break;
                case 3:
                    printf("guided");
```

```
                    break;
                case 4:
                    printf("auto");
                    break;
                }

                printf("\n") ;
            }
        }

        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);

    return(0);
}
```

CAPTURAS DE PANTALLA:

```
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  export OMP_SCHEDULE="static,4"
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  export OMP_THREAD_LIMIT=8
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  export OMP_NUM_THREADS=4
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  export OMP_DYNAMIC=FALSE
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  ./scheduledM 2 1

Antes de la región parallel
----------------------
dyn-var: False
nthreads-var: 4
thread-limit-var: 8
run-sched-var: (modifier) 4 (type)static

Dentro de la región parallel
----------------------
dyn-var: False
nthreads-var: 4
thread-limit-var: 8
run-sched-var: (modifier) 4 (type)static thread 0 suma a[1]=1 suma=1

 thread 3 suma a[0]=0 suma=0
Fuera de 'parallel for' suma=1
```

```
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  export OMP_SCHEDULE="dynamic,4"
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  export OMP_THREAD_LIMIT=4
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  export OMP_NUM_THREADS=2
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  export OMP_DYNAMIC=TRUE
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  ./scheduledM 2 1

Antes de la región parallel
----------------------
dyn-var: True
nthreads-var: 2
thread-limit-var: 4
run-sched-var: (modifier) 4 (type)dynamic

Dentro de la región parallel
----------------------
dyn-var: True
 thread 1 suma a[1]=1 suma=1
nthreads-var: 2
thread-limit-var: 4
run-sched-var: (modifier) 4 (type)dynamic
 thread 0 suma a[0]=0 suma=0
Fuera de 'parallel for' suma=1
```

RESPUESTA: Tanto dentro como fuera de la región parallel for se imprimen los mismos resultados. Esto se debe a que, aunque en la directiva parallel se especifica un cambio en el comportamiento del paralelismo (por ejemplo, que establece el reparto como dinámico) dicho cambio en el comportamiento solo afecta a esa región parallel (y no a posteriores), por eso no se modifica la variable de control, sino que se modifica directamente el comportamiento paralelo de esa y solo esa región.

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

int main(int argc, char **argv) {
    int i, n=200,chunk,a[n],suma=0;
    if(argc < 3)     {
        fprintf(stderr,"\nFalta iteraciones o chunk \n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);

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

    omp_sched_t schedule_type;
    int chunk_size;


    printf("\nAntes de la región parallel\n----------------------\n") ;

    printf("\nomp_get_num_threads(): %d",omp_get_num_threads()) ;
    printf("\nomp_get_num_procs(): %d",omp_get_num_procs()) ;
    printf("\nomp_in_parallel(): ") ;
    if(omp_in_parallel())
        printf(" true\n");
    else
        printf(" false\n");

    printf("\n") ;

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

    for (i=0; i<n; i++)
    {
        if(i == 0){
            {
                printf("\nAntes de la región parallel\n----------------------\n") ;

                printf("\nomp_get_num_threads(): %d",omp_get_num_threads()) ;
                printf("\nomp_get_num_procs(): %d",omp_get_num_procs()) ;
                printf("\nomp_in_parallel(): ") ;
                if(omp_in_parallel())
                    printf(" true\n");
                else
                    printf(" false\n");

                printf("\n") ;
            }
        }

        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);

    return(0);
}
```

CAPTURAS DE PANTALLA:

```
pinguino 1N0 ~ UGR … Practicas 3 code % ./scM4 3 3

Antes de la región parallel
----------------------

omp_get_num_threads(): 1
omp_get_num_procs(): 8
omp_in_parallel():  false


Antes de la región parallel
----------------------

omp_get_num_threads(): 8
omp_get_num_procs(): 8
omp_in_parallel():  true

 thread 2 suma a[0]=0 suma=0
 thread 2 suma a[1]=1 suma=1
 thread 2 suma a[2]=2 suma=3
Fuera de 'parallel for' suma=3
```

RESPUESTA: omp_in_parallel cambia porque nos informa de si se está ejecutando una región paralela o no, luego devuelve false antes de la región parallel y true dentro de la misma. Omp_get_num_procs no varía. Omp_get_num_threads pasa de valer uno antes de la región parallel (ejecución secuencial) a valer 8 (pues se paraleliza en tantos threads como procesos tiene asignados la ejecución).

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`

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _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;
    if(argc < 3)     {
        fprintf(stderr,"\nFalta iteraciones o chunk \n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);

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

    omp_sched_t schedule_type;
    int chunk_size;


    printf("\nAntes de la modificación\n----------------------\n") ;

    printf("dyn-var: ") ;
    if (omp_get_dynamic()) printf("True\n");
    else printf("False\n");
    printf("nthreads-var: %d\n", omp_get_max_threads());
    printf("thread-limit-var: %d",omp_get_thread_limit());

    omp_get_schedule(&schedule_type, &chunk_size);

    printf("\nrun-sched-var: (modifier) %d (type)",
           chunk_size);

    switch(schedule_type){
    case 1:
        printf("static");
        break;
    case 2:
        printf("dynamic");
        break;
    case 3:
        printf("guided");
        break;
    case 4:
        printf("auto");
        break;
    }

    printf("\n") ;
```

```
    omp_set_dynamic(1) ;
    omp_set_num_threads(10) ;
    omp_set_schedule(1,10) ;

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

    for (i=0; i<n; i++)
    {
        if(i == 0){
            {
                printf("\nDespués de la modificación\n----------------------\n") ;

                printf("dyn-var: ") ;
                if (omp_get_dynamic()) printf("True\n");
                else printf("False\n");
                printf("nthreads-var: %d\n", omp_get_max_threads());
                printf("thread-limit-var: %d",omp_get_thread_limit());

                omp_get_schedule(&schedule_type, &chunk_size);

                printf("\nrun-sched-var: (modifier) %d (type)",
                       chunk_size);

                switch(schedule_type){
                case 1:
                    printf("static");
                    break;
                case 2:
                    printf("dynamic");
                    break;
                case 3:
                    printf("guided");
                    break;
                case 4:
                    printf("auto");
                    break;
                }

                printf("\n") ;
            }
        }

        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);

    return(0);
}
```

CAPTURAS DE PANTALLA:

```
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  ./scM5 5 5

Antes de la modificación
----------------------
dyn-var: False
nthreads-var: 4
thread-limit-var: 6
run-sched-var: (modifier) 8 (type)dynamic

Después de la modificación
----------------------
dyn-var: True
nthreads-var: 10
thread-limit-var: 6
run-sched-var: (modifier) 10 (type)static
 thread 3 suma a[0]=0 suma=0
 thread 3 suma a[1]=1 suma=1
 thread 3 suma a[2]=2 suma=3
 thread 3 suma a[3]=3 suma=6
 thread 3 suma a[4]=4 suma=10
Fuera de 'parallel for' suma=10
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  10:11  pinguino@1N0
```

RESPUESTA:

Como era de esperar, los valores finales son los establecidos en la parte del código correspondiente a la modificación:

```
    omp_set_dynamic(1) ;
    omp_set_num_threads(10) ;
    omp_set_schedule(1,10) ;
```

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.

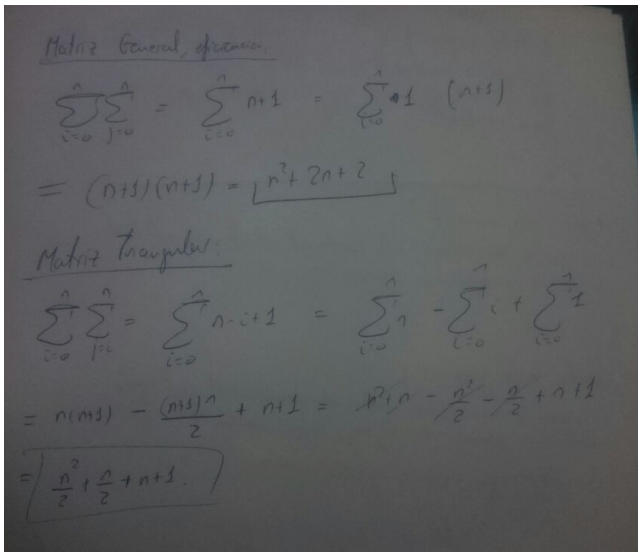
CÓDIGO FUENTE: `pmtv-secuencial.c`

```c
/* pmtv-secuencial.c
   Producto de una matriz cuadrada M triangular por un vector v1
   Para compilar usar (-lrt: real time library)
   gcc  -O2 pmv-secuencial.c -o pmv -lrt
*/

#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){

    int i;

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

    //Leer argumento de entrada (nº de componentes del vector)
    if (argc<2){
        printf("Error: Falta el número de filas y columnas.\n");
        exit(-1);
    }
    unsigned int f = atoi(argv[1]);// Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    double **M = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) M[i] = (double*)malloc(f * sizeof(double)) ;
    double *v1, *v2 ;
    v1 = (double*) malloc(f*sizeof(double));// malloc necesita el tamaño en bytes
    v2 = (double*) malloc(f*sizeof(double)); //si no hay espacio suficiente malloc devuelve NULL
    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }


    //Inicializar vector
    for(i=0; i<f; i++){
        v1[i] = 1.0;
    }
    //Inicializar matriz
    int j ;
    for(i=0 ; i < f ; i++)
        for(j = 0 ; j < f ; j++){
            if(i>j)
                M[i][j] = 0;
            else
                M[i][j] = 2.0;
        }

    int h ;
    int VECES = 100 ;
    for(h = 0 ; h < VECES ; h++){
    clock_gettime(CLOCK_REALTIME,&cgt1);

    //Calcular multiplicación.
    for(i=0; i<f; i++){
        v2[i] = 0 ;
        for(j=i ; j<f ; j++){
            v2[i] += M[i][j]*v1[j] ;
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt+=(double) (cgt2.tv_sec-cgt1.tv_sec)+
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    int k ;

    if(h%100==0){

        printf("\n%d\n",h) ;
    for(k = 0 ; k < f ; k=k+f-1){
        printf("\nv1[%d]=  %11.3f",k,v1[k]);
    }

    for(k=0 ; k < f ; k=k+f-1)
        printf("\nM[1][%d]=%11.3f",k,M[1][k]);
```

```
    for(k=0 ; k < f ; k=k+f-1)
        printf("\nv2[%d]=  %11.3f",k,v2[k]);
        }
    }
    ncgt = ncgt/VECES ;
    printf("\nTiempo(seg.): %11.9f\t / filas: %u\t / columnas: %u\t",
           ncgt,f,f) ;

    FILE * fp ;
    fp = fopen("salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\n%d\t%f",f,ncgt);
    fclose(fp);


#ifdef VECTOR_DYNAMIC
            for(i = 0 ; i < f ; i++)
                free(M[i]) ;
            free(M) ;
    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
#endif
    return 0;
}
```

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

```
pinguino  1N0  ~  UGR  …  Practicas  3  code  %  ./pmv-secuencial 123

0

v1[0]=        1.000
v1[122]=        1.000
M[1][0]=      0.000
M[1][122]=      2.000
v2[0]=      246.000
v2[122]=        2.000
Tiempo(seg.): 0.000015180        / filas: 123    / columnas: 123        %
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  ./pmtv-secuencial 123

0

v1[0]=        1.000
v1[122]=        1.000
M[1][0]=      0.000
M[1][122]=      2.000
v2[0]=      246.000
v2[122]=        2.000
Tiempo(seg.): 0.000008277        / filas: 123    / columnas: 123        %
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %
```



RESPUESTA:

El codigo de multiplicación de una matriz cualquiera por un vector ejecuta dos bucles anidados. Siendo n el número de filas y columnas, el bucle exterior tiene n iteraciones y el interior otras n. Por tanto el orden de eficiencia es cuadrático (n^2+n). En la versión para matrices triangulares tienes n iteraciones en el bucle externo y el número de iteraciones del bucle interior varía en cada iteración. Haciendo los cálculos pertinentes podemos comprobar que ambas versiones tienen un orden de eficiencia cuadrático (o grande n al cuadrado) pero que la versión para matrices triangulares es aproximadamente el doble de rápida (es lógico pues hace casi la mitad de iteraciones). Aún así, para tamaños de matriz muy grandes (que tienden a infinito) la diferencia entre los algoritmos es casi despreciable. En la captura de pantalla se ve una ejecución de cada programa para un tamaño de 123; la diferencia de tiempo es de unos 0.000006903 segundos…

La principal diferencia entre el código para una matriz general y el código para matrices triangulares es que en las segundas solo se ejecutan las sumas de las posiciones de la matriz que cumplan que la columna es igual o mayor que la fila (**IMPORTANTE:** ESTAMOS SUPONIENDO EN ESTE PROGRAMA QUE LA MATRIZ ES TRIANGULAR INFERIOR, PARA MATRICES TRIANGULARES SUPERIORES EL CÓDIGO SERÍA AL CONTRARIO)

```
for(i=0; i<f; i++){
        v2[i] = 0 ;
        for(j=i ; j<f ; j++){
            v2[i] += M[i][j]*v1[j]
         }
}
```

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 que el reparto se realice de acuerdo a la variable de entorno OMP_SCHEDULE hace falta establecer el parámetro “schedule” de las secciones paralelas como “runtime”.
Los valores por defecto son:
static: 0 (no tiene sentido el chunck en este caso porque se divide en partes iguales)
dynamic: 1
guided: 1
Los he obtenido utilizando la función omp_get_schedule(&schedule_type, &chunk_size); y modificando la variable de entorno sin especificar el chunck (por ejemplo export OMP_SCHEDULE="dynamic")

(B) El trabajo se reparte por filas, siendo el total de estas 30720. En cada fila se hace una suma menos que multiplicaciones y se hacen i multiplicaciones en cada fila (desde i = 0 hasta 30720) como las iteraciones por filas se reparten de forma “equitativa” entre los 12 threads y se da a cada thread un conjunto de filas consecutivas, tendremos que el primer thread tiene las ejecuciones desde i=0 hasta i=x y el siguiente desde i=x+1 hasta i=2x ocurrirá que habrá threads que ejecuten muchas operaciones (aquellos a los que le toquen las iteraciones a las que corresponden más operaciones, las del final) y threads que ejecuten muy pocas instrucciones (aquellos que ejecuten las primeras iteraciones, más cortas. Por otro lado, si la asignación se hace dinámicamente o “guided”, el número de operaciones de multiplicación y suma debería estabilizarse y, por tanto, debería reducirse el tiempo de ejecución.
Por lo general tanto dynamic como guided tendrán repartos distintos para distintas ejecuciones y darán lugar a tiempos muy diferentes de una ejecución a otra. Sin embargo, para este caso y, dado que la duración de las iteraciones a paralelizar es heterogenea, tanto guided como dynamic darán mejores resultados que static, siendo guided en principio algo mejor que dynamic como se aprecia en la gráfica.

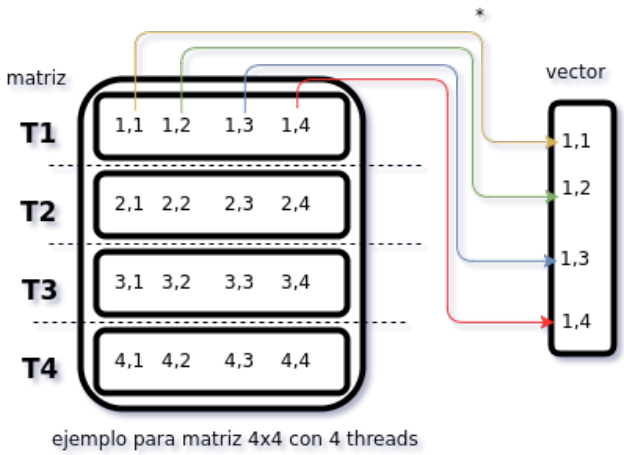
```
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  export OMP_SCHEDULE="
static"
 pinguino  1N0  ~  UGR  …  Practicas  3  code  %  ./pmtvOMP 1000

Antes de la modificación
----------------------
dyn-var: False
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: (modifier) 0 (type)static

0
```

CÓDIGO FUENTE: `pmtv-OpenMP.c`

```c
/* pmtv-secuencial.c
   Producto de una matriz cuadrada M triangular por un vector v1
   Para compilar usar (-lrt: real time library)
   gcc  -O2 pmv-secuencial.c -o pmv -lrt
*/

#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()
#include<omp.h>

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

    int i;

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

    //Leer argumento de entrada (nº de componentes del vector)
    if (argc<2){
        printf("Error: Falta el número de filas y columnas.\n");
        exit(-1);
    }
    unsigned int f = atoi(argv[1]);// Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    double **M = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) M[i] = (double*)malloc(f * sizeof(double)) ;
    double *v1, *v2 ;
    v1 = (double*) malloc(f*sizeof(double));// malloc necesita el tamaño en bytes
    v2 = (double*) malloc(f*sizeof(double)); //si no hay espacio suficiente malloc devuelve NULL
    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }


    //Inicializar vector
#pragma omp for schedule(runtime)
    for(i=0; i<f; i++){
        v1[i] = 1.0;
    }
    //Inicializar matriz
    int j ;
#pragma omp for schedule(runtime)
    for(i=0 ; i < f ; i++)
        for(j = 0 ; j < f ; j++){
            if(i>j)
                M[i][j] = 0;
            else
                M[i][j] = 2.0;
        }

    int h ;
    int VECES = 100 ;
    for(h = 0 ; h < VECES ; h++){
    clock_gettime(CLOCK_REALTIME,&cgt1);

    //Calcular multiplicación.
#pragma omp for schedule(runtime)
    for(i=0; i<f; i++){
        /**
        if(i==0 && h == 0){

            omp_sched_t schedule_type;
            int chunk_size;

            printf("dyn-var: ") ;
            if (omp_get_dynamic()) printf("True\n");
            else printf("False\n");
            printf("nthreads-var: %d\n", omp_get_max_threads());
            printf("thread-limit-var: %d",omp_get_thread_limit());

            omp_get_schedule(&schedule_type, &chunk_size);

            printf("\nrun-sched-var: (modifier) %d (type)",
                   chunk_size);

            switch(schedule_type){
            case 1:
                printf("static");
                break;
            case 2:
                printf("dynamic");
```

```
                break;
            case 3:
                printf("guided");
                break;
            case 4:
                printf("auto");
                break;
            }

            printf("\n") ;
        }
        **/
        v2[i] = 0 ;
        for(j=i ; j<f ; j++){
            v2[i] += M[i][j]*v1[j] ;
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt+=(double) (cgt2.tv_sec-cgt1.tv_sec)+
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    int k ;

    if(h%100==0){

        printf("\n%d\n",h) ;
    for(k = 0 ; k < f ; k=k+f-1){
        printf("\nv1[%d]=  %11.3f",k,v1[k]);
    }

    for(k=0 ; k < f ; k=k+f-1)
        printf("\nM[1][%d]=%11.3f",k,M[1][k]);
    for(k=0 ; k < f ; k=k+f-1)
        printf("\nv2[%d]=  %11.3f",k,v2[k]);
        }
    }
    ncgt = ncgt/VECES ;
    printf("\nTiempo(seg.): %11.9f\t / filas: %u\t / columnas: %u\t",
           ncgt,f,f) ;

    FILE * fp ;
    fp = fopen("salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\n%d\t%f",f,ncgt);
    fclose(fp);


#ifdef VECTOR_DYNAMIC
            for(i = 0 ; i < f ; i++)
                free(M[i]) ;
            free(M) ;
    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
#endif
    return 0;
}
```

DESCOMPOSICIÓN DE DOMINIO:



ejemplo para matriz 4x4 con 4 threads

v2[i]=M[i][j]*v[j]
Cada thread ejecuta una (o varias) filas, es decir, unos i determinados. Para cada fila que ejecuta suma M[i][j] con todos los j que le corresponden (desde j=i hasta el final).

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

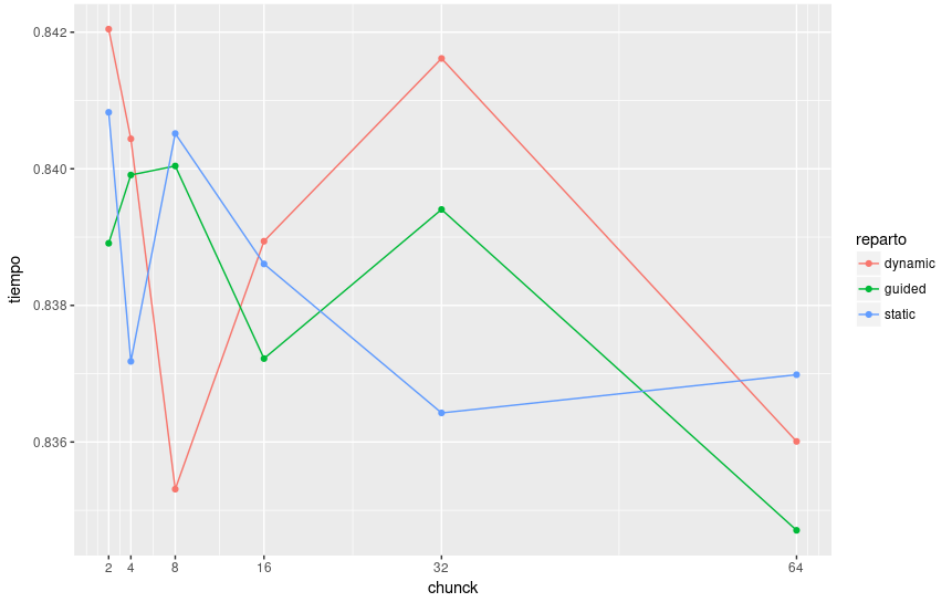
TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash
export OMP_NUM_THREADS=12
mkdir resultados
for i in {2,4,8,16,64}
do
    export OMP_SCHEDULE="static, $i"
    #echo $OMP_SCHEDULE
    #echo
    ./pmtvOMP 30720 >> ./resultados/static
done
for i in {2,4,8,16,64}
do
    export OMP_SCHEDULE="dynamic, $i"
    #echo $OMP_SCHEDULE
    #echo
    ./pmtvOMP 30720 >> ./resultados/dynamic
done
for i in {2,4,8,16,64}
do
    export OMP_SCHEDULE="guided, $i"
    #echo $OMP_SCHEDULE
    #echo
    ./pmtvOMP 30720 >> ./resultados/guided
done
```

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N=30720 , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.847434625	0.842044787	0.838909499
1	0.840826915	0.842044787	0.838909499
64	0.836985550	0.836009105	0.834707954
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.862334455	0.842044787	0.844246419
1	0.842824125	0.842044787	0.8489494439
64	0.839344555	0.836009105	0.8334471924



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

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

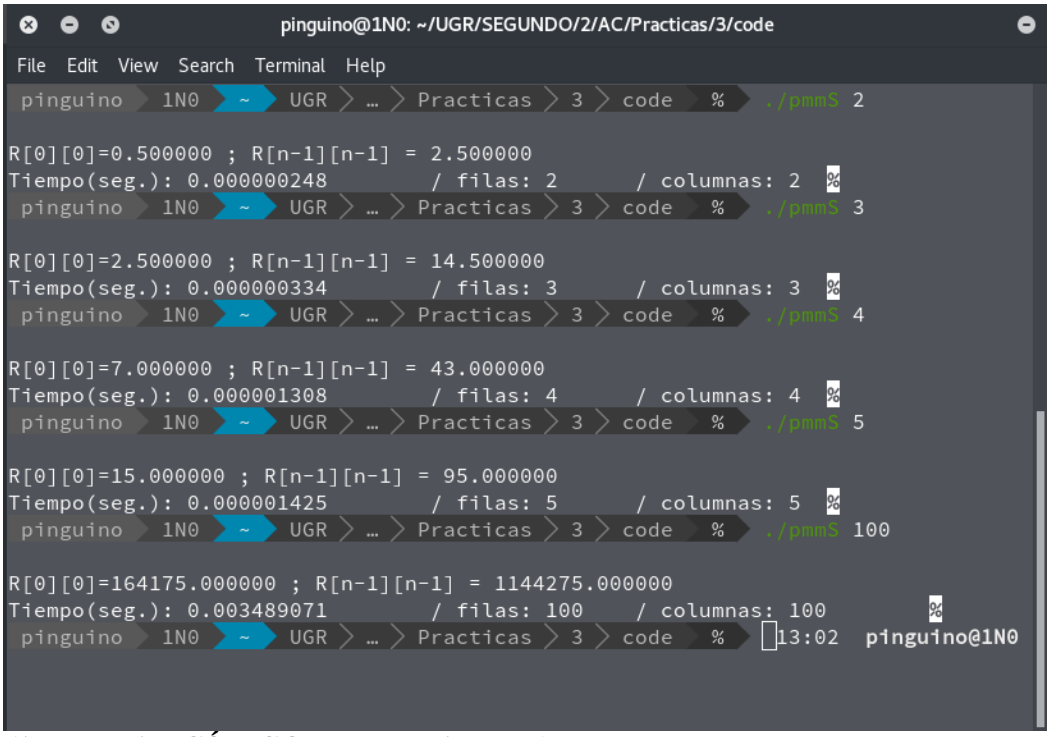
NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CÓDIGO FUENTE: `pmm-secuencial.c`

```c
/* pmm-secuencial.c
   Producto de dos matrices cuadradas, M y L.
   Para compilar usar (-lrt: real time library)
   gcc  -O2 pmv-secuencial.c -o pmv -lrt
*/

#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){

    int i;

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

    //Leer argumento de entrada (nº de filas y columnas)
    if (argc<2){
        printf("Error: Falta el número de filas y columnas.\n");
        exit(-1);
    }
    unsigned int f = atoi(argv[1]);// Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    double **M = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) M[i] = (double*)malloc(f * sizeof(double)) ;
    double **L = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) L[i] = (double*)malloc(f * sizeof(double)) ;
    // matriz resultado
    double **R = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) R[i] = (double*)malloc(f * sizeof(double)) ;

    if ( (L==NULL) ||  (M==NULL) || (R==NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }

    int j ;

    //Inicializar matrices

    for(int i = 0 ; i < f ; i++){
        for(int j = 0 ; j < f ; j++){
            M[i][j] = 0.5*(i+j);
            L[i][j] = (i+j) ;
            R[i][j] = 0 ;
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    // transposición del a matriz L para eliminar fallos de caché.

    for(i = 0 ; i < f ; i++)
        for(j = 0 ; j < f ; j++){
            if(i != j){
                int aux = L[i][j] ;
                L[i][j] = L[j][i] ;
                L[j][i] = aux ;
            }
        }

    int k ;
    //Calcular multiplicación.
    for(i=0; i<f; i++){
        for(j=0 ; j<f ; j++){
            for(k=0 ; k<f ; k++){
                R[i][j] += M[i][k]*L[j][k];
            }
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt+=(double) (cgt2.tv_sec-cgt1.tv_sec)+
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));



    printf("\nR[0][0]=%f ; R[n-1][n-1] = %f", R[0][0], R[f-1][f-1]) ;

    ncgt = ncgt ;
```

```
    printf("\nTiempo(seg.): %11.9f\t / filas: %u\t / columnas: %u\t",
           ncgt,f,f) ;

    FILE * fp ;
    fp = fopen("salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\n%d\t%f",f,ncgt);
    fclose(fp);

}
```

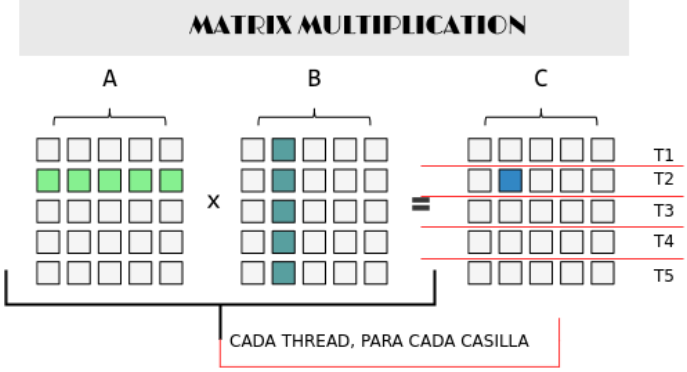
CAPTURAS DE PANTALLA:



(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

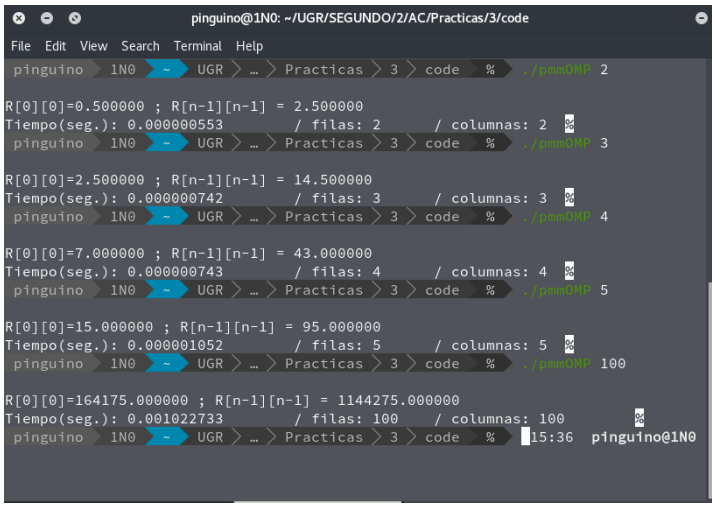
DESCOMPOSICIÓN DE DOMINIO:



CÓDIGO FUENTE: pmm-OpenMP.c

```c
/* pmm-secuencial.c
   Producto de dos matrices cuadradas, M y L.
   Para compilar usar (-lrt: real time library)
   gcc  -O2 pmv-secuencial.c -o pmv -lrt
*/

#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){

    int i;

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

    //Leer argumento de entrada (nº de filas y columnas)
    if (argc<2){
        printf("Error: Falta el número de filas y columnas.\n");
        exit(-1);
    }
    unsigned int f = atoi(argv[1]);// Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    double **M = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) M[i] = (double*)malloc(f * sizeof(double)) ;
    double **L = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) L[i] = (double*)malloc(f * sizeof(double)) ;
    // matriz resultado
    double **R = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) R[i] = (double*)malloc(f * sizeof(double)) ;

    if ( (L==NULL) ||  (M==NULL) || (R==NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }

    int j ;

    //Inicializar matrices
#pragma omp for schedule(runtime)
    for(int i = 0 ; i < f ; i++){
        for(int j = 0 ; j < f ; j++){
            M[i][j] = 0.5*(i+j);
            L[i][j] = (i+j) ;
            R[i][j] = 0 ;
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    // transposición del a matriz L para eliminar fallos de caché.

#pragma omp for schedule(runtime)
    for(i = 0 ; i < f ; i++)
        for(j = 0 ; j < f ; j++){
            if(i != j){
                int aux = L[i][j] ;
                L[i][j] = L[j][i] ;
                L[j][i] = aux ;
            }
        }

    int k ;
    //Calcular multiplicación.
#pragma omp for schedule(runtime)
    for(i=0; i<f; i++){
        for(j=0 ; j<f ; j++){
            for(k=0 ; k<f ; k++){
                R[i][j] += M[i][k]*L[j][k];
            }
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt+=(double) (cgt2.tv_sec-cgt1.tv_sec)+
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));


    printf("\nR[0][0]=%f ; R[n-1][n-1] = %f", R[0][0], R[f-1][f-1]) ;

    ncgt = ncgt ;

    printf("\nTiempo(seg.): %11.9f\t / filas: %u\t / columnas: %u\t",
           ncgt,f,f) ;

    FILE * fp ;
    fp = fopen("salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\n%d\t%f",f,ncgt);
    fclose(fp);
}
```

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



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 –02 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.

He cambiado un poco el código modificando la escritura del final:

```
FILE * fp ;
    fp = fopen("data","a");
    if(fp==NULL){perror("Error opening file.");}
    // tamaño cores tiempo pc
    fprintf(fp,"\n%d\t%d\t%f\t%s",f,cores,ncgt,argv[3]);    fclose(fp);
```

ESTUDIO DE ESCALABILIDAD EN ATCGRID:

SCRIPT: pmm-OpenMP_atcgrid.sh

```
#!/bin/bash

for i in {1..14..1}
do
    echo cores: $i
    export OMP_NUM_THREADS=$i
    ./pmmOMP 100 $i atcgrid
    ./pmmOMP 1500 $i atcgrid
done
```

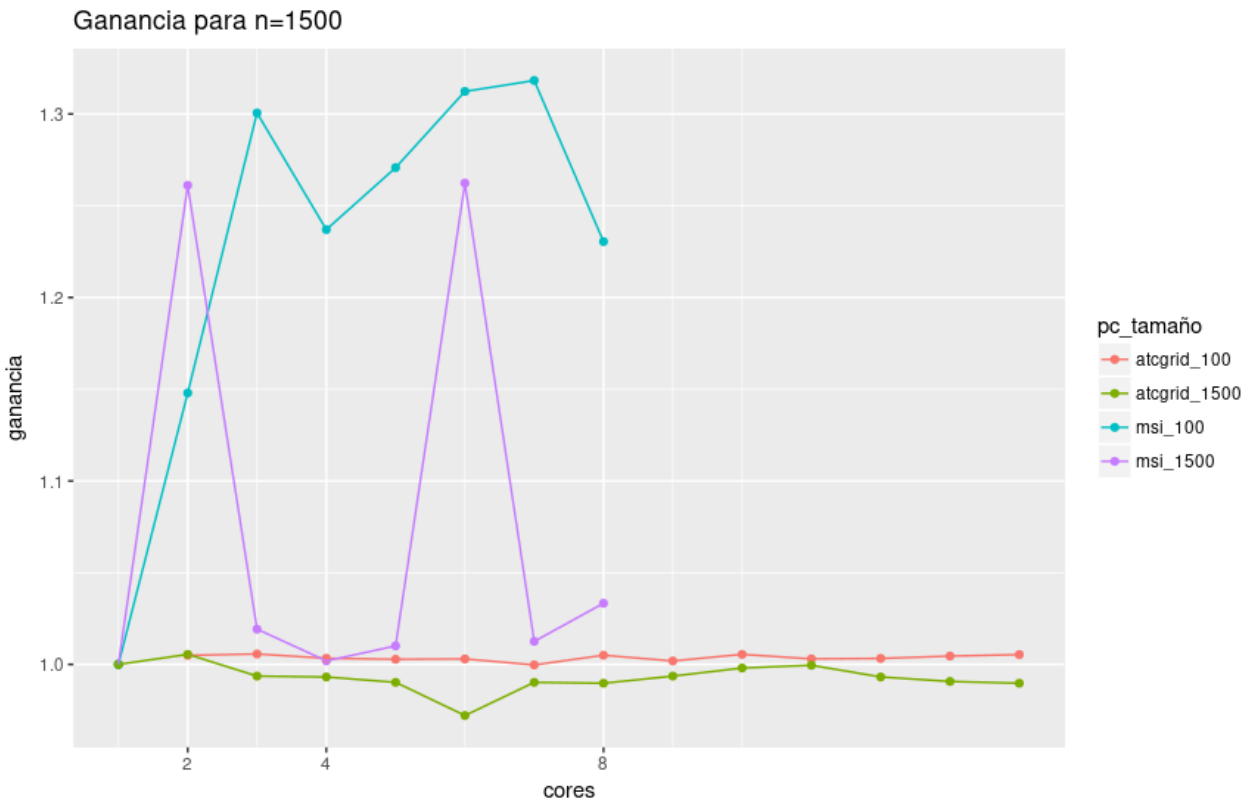
ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```
#!/bin/bash

for i in {1..8..1}
do
    echo cores: $i
    export OMP_NUM_THREADS=$i
    ./pmmOMP 100 $i msi
    ./pmmOMP 1500 $i msi
done
```

RESULTADOS:

tamaño	cores	tiempo	pc_tamaño	ganancia	
1500	1	4.526868	msi_1500	1	
1500	2	3.589576	msi_1500	1.26111	
1500	3	4.441662	msi_1500	1.01918	
1500	4	4.518262	msi_1500	1.00190	
1500	5	4.481663	msi_1500	1.01008	
1500	6	3.586205	msi_1500	1.26230	
1500	7	4.470846	msi_1500	1.01253	
1500	8	4.38093	msi_1500	1.03331	
1500	1	4.84959	atcgrid_1500		1
1500	2	4.825818	atcgrid_100	1.00492	
1500	3	4.822239	atcgrid_100	1.0056	
1500	4	4.833547	atcgrid_100	1.00331	
1500	5	4.836195	atcgrid_100	1.00276	
1500	6	4.835287	atcgrid_100	1.00295	
1500	7	4.850954	atcgrid_100	0.999718	
1500	8	4.825612	atcgrid_100	1.00496	
1500	9	4.840575	atcgrid_100	1.00186	
1500	10	4.823275	atcgrid_100	1.0054	
1500	11	4.835074	atcgrid_100	1.00300	
1500	12	4.833911	atcgrid_100	1.00324	
1500	13	4.827779	atcgrid_100	1.00451	
1500	14	4.823886	atcgrid_100	1.00532	
100	1	0.001164	msi_100	1	
100	2	0.001014	msi_100	1.14792	
100	3	0.000895	msi_100	1.30055	
100	4	0.000941	msi_100	1.23698	
100	5	0.000916	msi_100	1.2707	
100	6	0.000887	msi_100	1.31228	
100	7	0.000883	msi_100	1.31823	
100	8	0.000946	msi_100	1.23044	
100	1	0.002023	atcgrid_1500		1
100	2	0.002012	atcgrid_1500		1.00546
100	3	0.002036	atcgrid_1500		0.993614
100	4	0.002037	atcgrid_1500		0.993127
100	5	0.002043	atcgrid_1500		0.990210
100	6	0.002081	atcgrid_1500		0.972128
100	7	0.002043	atcgrid_1500		0.990210
100	8	0.002044	atcgrid_1500		0.98972
100	9	0.002036	atcgrid_1500		0.993614
100	10	0.002027	atcgrid_1500		0.998026
100	11	0.002024	atcgrid_1500		0.999505
100	12	0.002037	atcgrid_1500		0.993127
100	13	0.002042	atcgrid_1500		0.990695
100	14	0.002044	atcgrid_1500		0.98972


Ganancia para n=1500
ganancia
1.3
1.2
1.1
1.0
2
4
8
cores
pc_tamaño
atcgrid_100
atcgrid_1500
msi_100
msi_1500