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): Emilio Chica Jiménez
Grupo de prácticas: B2
Fecha de entrega:
Fecha evaluación en clase:

---

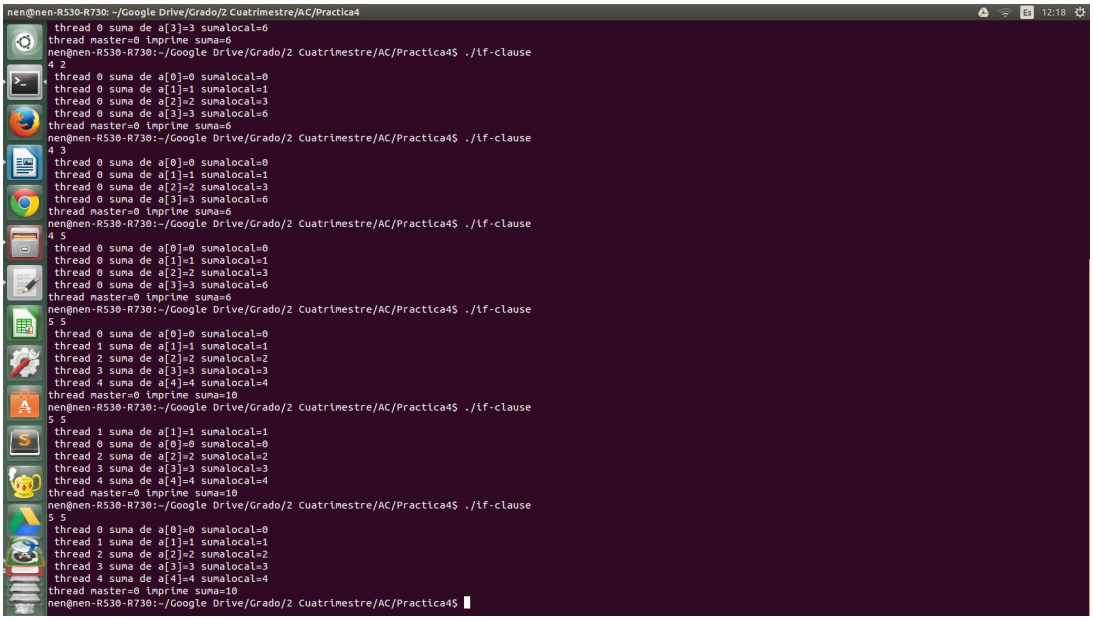
1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad 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;
              int a[n],suma=0,sumalocal,x;
              if(argc < 2) {
                            fprintf(stderr,"[ERROR]-Falta iteraciones y numero
de hebras\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:



RESPUESTA:

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

14	1	1	1	0	0	1	1	1	1
15	1	1	1	1	0	1	0	1	1

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

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

RESPUESTA: Con planificación estática el sistema operativo carga mediante asignación por round robin de las unidades de chunk en las que se han dividido las iteraciones a las hebras.
Con planificación dinámica el sistema operativo carga a cada hebra con las iteraciones divididas en tamaño de chunk. Si la hebra 0 ha terminado con sus unidades de chunk asignadas el SO le vuelve a asignar otras unidades de chunk dinámicamente. Por lo tanto las hebras más rápidas trabajarán más y el procesador se encontrará menos ocioso, pero al cambiar tanto de hebra y carecer de una planificación estática se sobrecarga el procesador debido al flush de TLB's.
Con la planificación guidada la asignación de iteraciones se hace dinámicamente por medio de bloques. Estos bloques de iteraciones van menguando a medida que el numero de interaciones restante a asignar es menor:

numero_iteraciones_restante / numero_hebras

El parámetro chunk define la última división posible de un bloque. Por defecto 1.

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`

```
/* 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:

RESPUESTA:

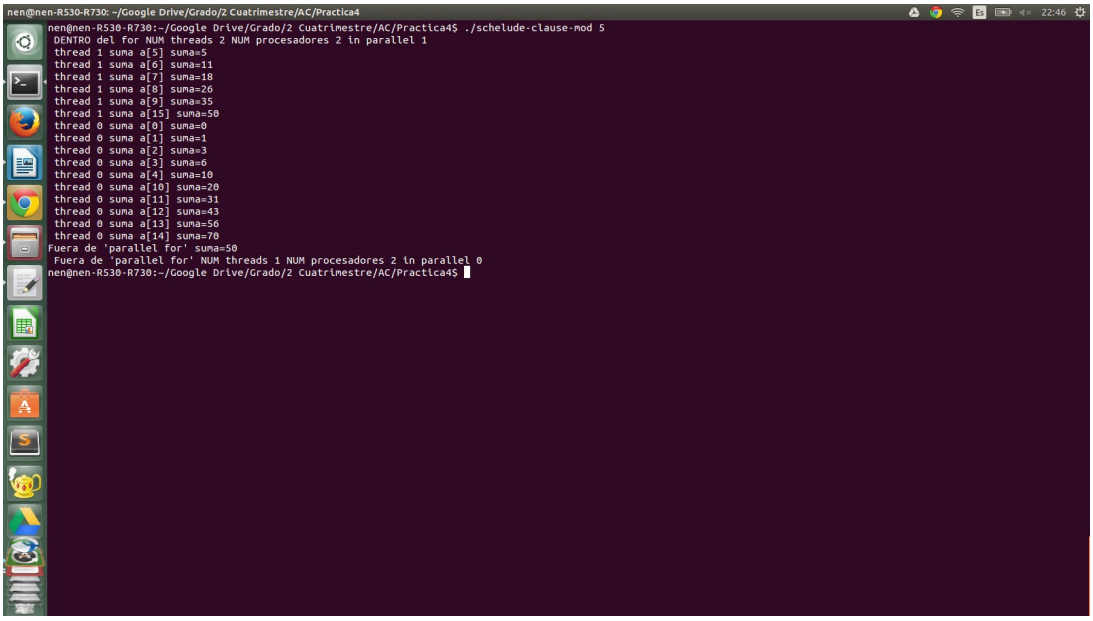
4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno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

CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
            #include <omp.h>
#else
            #define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
            int i, n = 16,chunk, a[n],suma=0,x;
            if(argc < 2) {
                        fprintf(stderr,"\nFalta chunk \n");
                        exit(-1);
            }
            chunk = atoi(argv[1]);
            for (i=0; i<n; i++) a[i] = i;
            #pragma omp parallel
            {
                        #pragma omp single
                        {
                                    printf(" DENTRO del for NUM threads %d
NUM procesadores %d in parallel %d \n",

            omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
                        }
                        #pragma omp for firstprivate(suma) \
```

```
                lastprivate(suma) schedule(static,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);

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

                printf(" Fuera de 'parallel for' NUM threads %d NUM procesos %d
in parallel %d \n",

                omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
}
```

CAPTURAS DE PANTALLA:



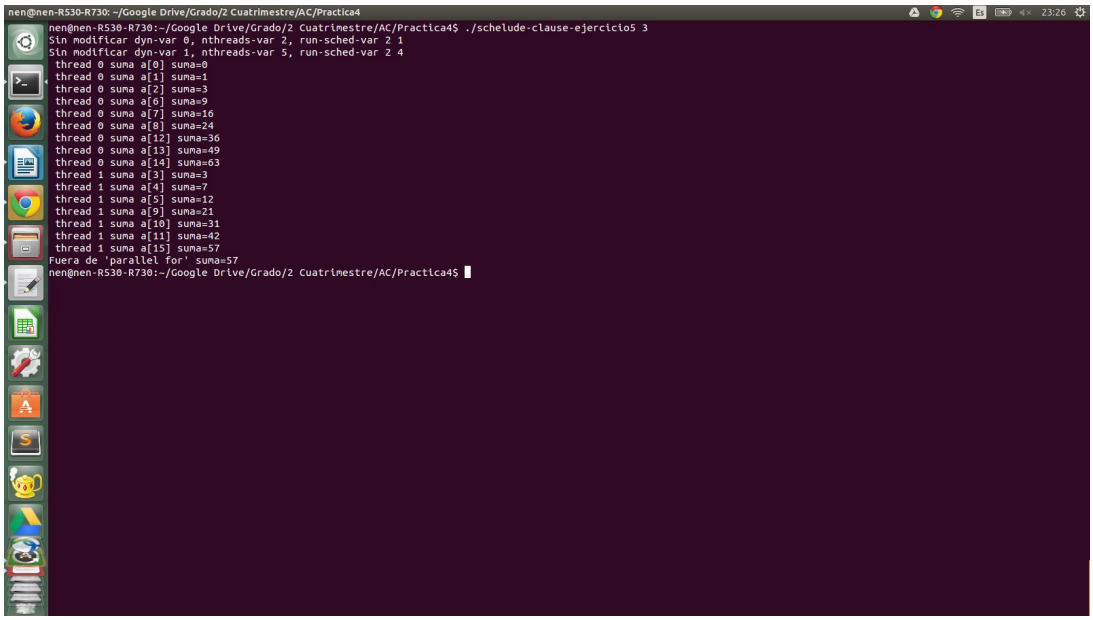
RESPUESTA: Da valores distintos para `omp_get_num_threads()` porque en la region paralela usamos 2 threads y en el código secuencial 1 porque no se está usando parallel. La función `omp_in_parallel()` también devuelve valores distintos porque en la primera se encuentra dentro de una región paralela por lo que devuelve true y en la segunda no.

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

CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
            #include <omp.h>
#else
            #define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
            int i, n = 16,chunk, a[n],suma=0,x;
            if(argc < 2) {
                        fprintf(stderr,"\nFalta chunk \n");
                        exit(-1);
            }
            omp_sched_t kind;int modifier;
            omp_get_schedule(&kind, &modifier);
            printf("Sin modificar dyn-var %d, nthreads-var %d, run-sched-var
%d %d\n",omp_get_dynamic(),omp_get_max_threads(),kind,modifier);

            //MODIFICO LAS VARIABLES
            omp_set_num_threads(5);
            omp_set_dynamic(1);
            kind = 2;
            modifier=4;
            omp_set_schedule(kind, modifier);
            omp_get_schedule(&kind, &modifier);
            printf("Sin modificar dyn-var %d, nthreads-var %d, run-sched-var
%d %d\n",omp_get_dynamic(),omp_get_max_threads(),kind,modifier);
            chunk = atoi(argv[1]);
            for (i=0; i<n; i++) a[i] = i;
            #pragma omp parallel for firstprivate(suma) \
            lastprivate(suma) schedule(static,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);
            }
            printf("Fuera de 'parallel for' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:



RESPUESTA:

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector.

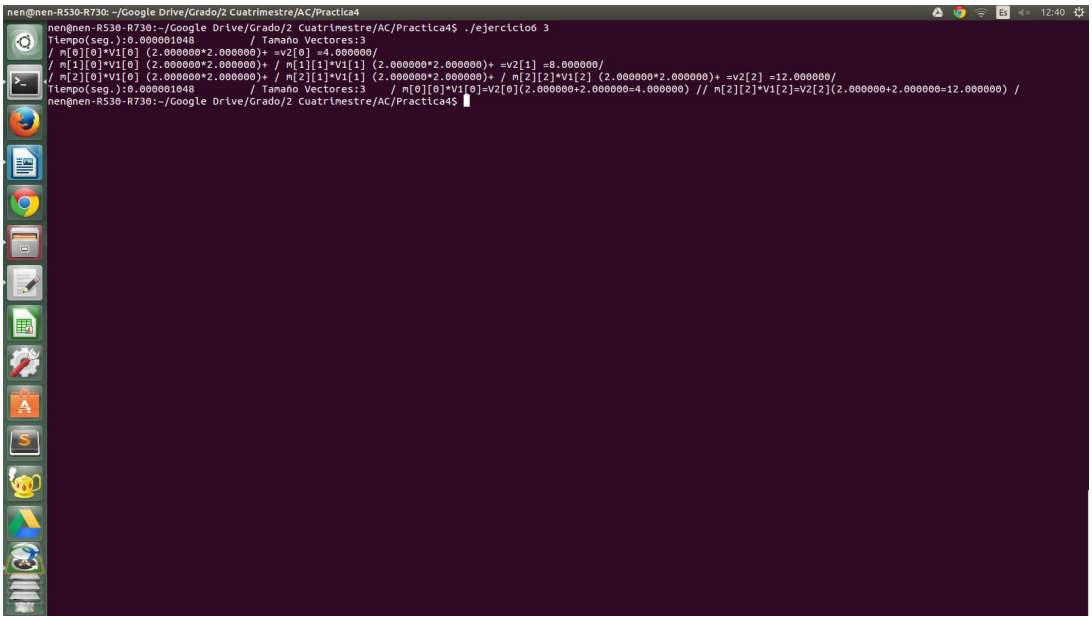
NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

CÓDIGO FUENTE: `pmtv-secuencial.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//#define PRINTF_ALL

main(int argc, char **argv)
{
		if(argc < 2) {
				fprintf(stderr,"Falta fila y columna\n");
				exit(-1);
		}

		struct timespec cgt1,cgt2; double ncgt; //para tiempo de
ejecución
		int i,k, n = atoi(argv[1]);
		double *v1,*v2;
		v1 = (double*)malloc(n*sizeof(double));
		v2 = (double*)malloc(n*sizeof(double));
		double sumalocal=0;
		double **m;
		m = (double**)malloc(n*sizeof(double*));
		//Inicializo v1 y reservo el espacio para la matriz
		for(i=0;i<n;++i){
				m[i]=(double*)malloc(n*sizeof(double));
				v1[i]=2;
		}
```

```
            //Inicializo la matriz
            for (i=0; i<n; i++)
                        for(k=0;k<n;++k)
                                    if(k<i+1)
                                                m[i][k]=2;
                                    else
                                                m[i][k]=0;

            //Calculo la multiplicacion de la matriz por el vector y obtengo
el tiempo
            clock_gettime(CLOCK_REALTIME,&cgt1);
            for (i=0; i<n; i++){
                        for(k=0;k<i+1;++k)
                                    sumalocal+=m[i][k]*v1[k];
                        v2[i]=sumalocal;
                        sumalocal=0;
            }
            clock_gettime(CLOCK_REALTIME,&cgt2);
            ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

            //Imprimo los resultados
            #ifdef PRINTF_ALL
            printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,n);
            for (i=0; i<n; i++){
                        for(k=0;k<i+1;++k){
                                     printf("/ m[%d][%d]*V1[%d] (%8.6f*
%8.6f)+ ",i,k,k,m[i][k],v1[k]);
                        }
                        printf("=v2[%i] =%8.6f/ \n",i,v2[i]);
            }
            #else

            printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ m[0]
[0]*V1[0]=V2[0](%8.6f+%8.6f=%8.6f) // m[%d][%d]*V1[%d]=V2[%d](%8.6f+%8.6f=
%8.6f) /\n", ncgt,n,m[0][0],v1[0],v2[0],n-1,n-1,n-1,n-1,m[n-1][n-1],v1[n-
1],v2[n-1]);
            #endif
            free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2

            for(i=0;i<n;++i){
                        free(m[i]);
            }
        free(m); // libera el espacio reservado para m

}
```

CAPTURAS DE PANTALLA:



(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno OMP_SCHEDULE. Obtener en atcgrid los tiempos de ejecución del código paralelo que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 2, 64, 128, 1024 y el chunk por defecto para la alternativa. No use vectores mayores de 32768 componentes ni menores de 4096 componentes. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 con los tiempos obtenidos, ponga en la tabla el número de threads que utilizan las ejecuciones. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. Rellenar la tabla y realizar la gráfica también para el PC local. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué.

RESPUESTA: Cuanto mayor es el chunk mejores prestacione, debido a que no tiene que cambiar de hebra continuamente y se producen menos flush de TLB's por lo tanto menos sobrecarga.

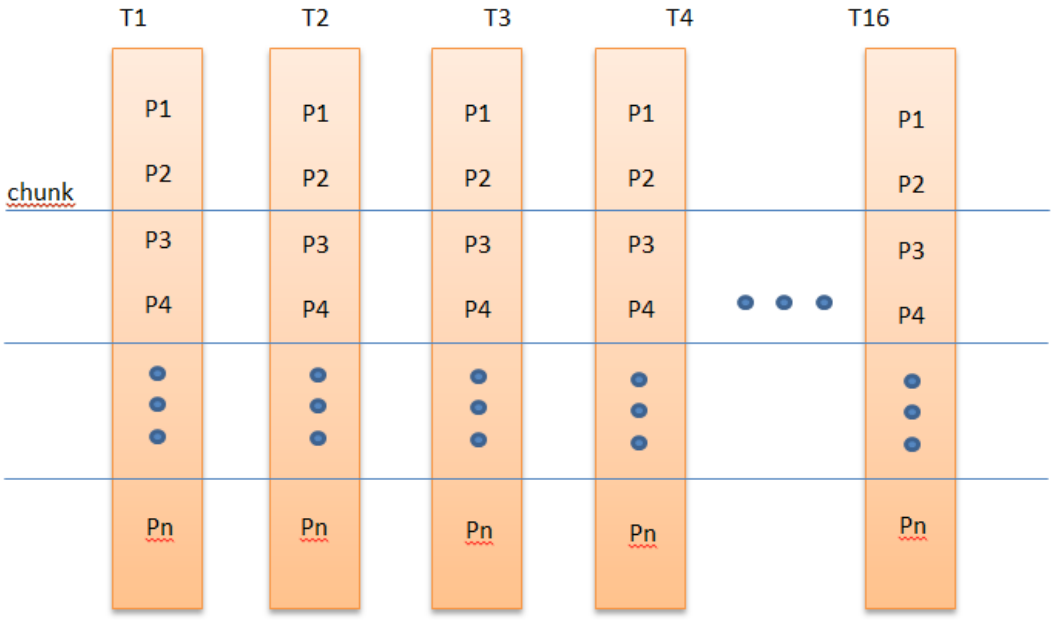
CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
//#define PRINTF_ALL

main(int argc, char **argv)
{
            if(argc < 4) {
                        fprintf(stderr,"Falta fila y columna\n");
                        exit(-1);
            }
            omp_set_num_threads(16);
            omp_set_dynamic(0);
```

```
            struct timespec cgt1,cgt2; double ncgt; //para tiempo de
ejecución
            int i,k, n = atoi(argv[1]);
            int kind = atoi(argv[2]);
            int chunk=atoi(argv[3]);
            omp_set_schedule(kind, chunk);
            double *v1,*v2;
            v1 = (double*)malloc(n*sizeof(double));
            v2 = (double*)malloc(n*sizeof(double));
            double sumalocal=0;
            double **m;
            m = (double**)malloc(n*sizeof(double*));

            //Inicializo v1 y reservo el espacio para la matriz
            #pragma omp parallel for
            for(i=0;i<n;++i){
                        m[i]=(double*)malloc(n*sizeof(double));
                        v1[i]=2;
            }

            //Inicializo la matriz
            #pragma omp parallel private(k)
            {
                        #pragma omp for
                        for (i=0; i<n; i++)

                                    for(k=0;k<n;++k)
                                    if(k<i+1)
                                                m[i][k]=2;
                                    else
                                                m[i][k]=0;

            }
            //Calculo la multiplicacion de la matriz por el vector y obtengo
el tiempo
            clock_gettime(CLOCK_REALTIME,&cgt1);
            #pragma omp parallel private(k,sumalocal)
            {
                        sumalocal=0;
                        #pragma omp for
                        for (i=0; i<n; i++){
                                    for(k=0;k<i+1;++k)
                                                sumalocal+=m[i]
[k]*v1[k];
                                    #pragma omp critical
                                    {
                                                v2[i]=sumalocal;
                                                sumalocal=0;
                                    }

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



            //Imprimo los resultados
            #ifdef PRINTF_ALL
            printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,n);
```

```
            for (i=0; i<n; i++){
                        for(k=0;k<i+1;++k){
                                    printf("/ m[%d][%d]*V1[%d] (%8.6f*
%8.6f)+ ",i,k,k,m[i][k],v1[k]);
                        }
                        printf("=v2[%i] =%8.6f/ \n",i,v2[i]);
            }

            #else
        printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ m[0]
[0]*V1[0]=V2[0](%8.6f+%8.6f=%8.6f) // m[%d][%d]*V1[%d]=V2[%d](%8.6f+%8.6f=
%8.6f) /\n", ncgt,n,m[0][0],v1[0],v2[0],n-1,n-1,n-1,n-1,m[n-1][n-1],v1[n-
1],v2[n-1]);
            #endif
            free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2

            for(i=0;i<n;++i){
                        free(m[i]);
            }
        free(m); // libera el espacio reservado para m

}
```

DESCOMPOSICIÓN DE DOMINIO:



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

TABLA RESULTADOS Y GRÁFICA ATCGRID

Tabla 3 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
por defecto	0.030968994	0.017932500	0.028880876
2	0.019498493	0.023215721	0.019131411
64	0.018862950	0.019848680	0.029042583
128	0.019958263	0.018820304	0.017901258
1024	0.019073429	0.018355125	0.018315786

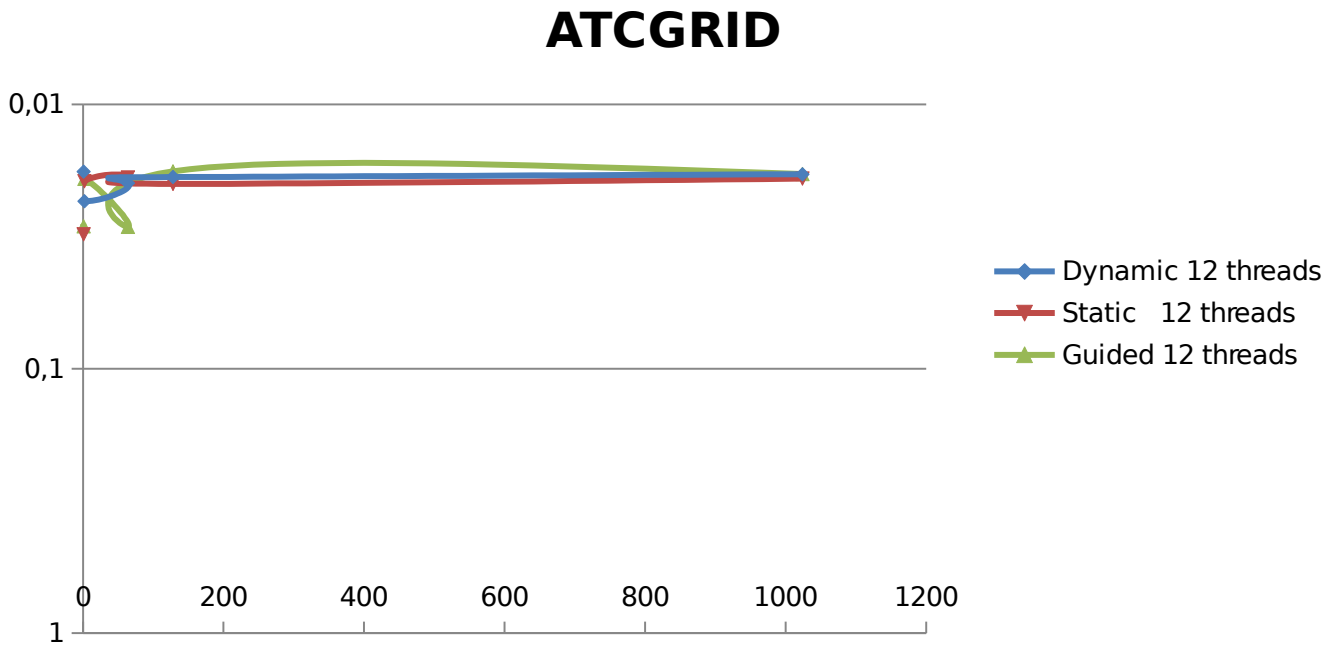
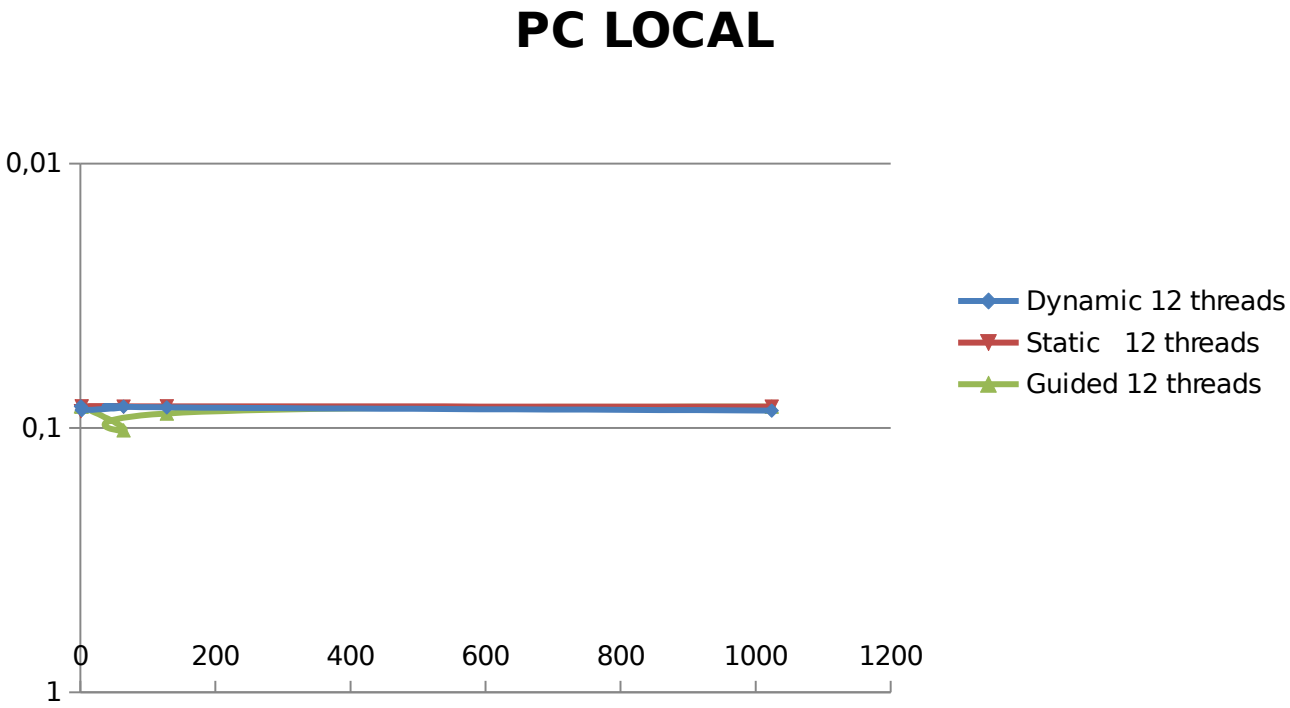


TABLA RESULTADOS Y GRÁFICA PC LOCAL

Tabla 4 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
por defecto	0.086442345	0.082992815	0.082956837
2	0.082941823	0.085615903	0.083026478
64	0.083011538	0.083117905	0.102225292
128	0.082915577	0.083611389	0.088215405
1024	0.083152761	0.085951086	0.083068600



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

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

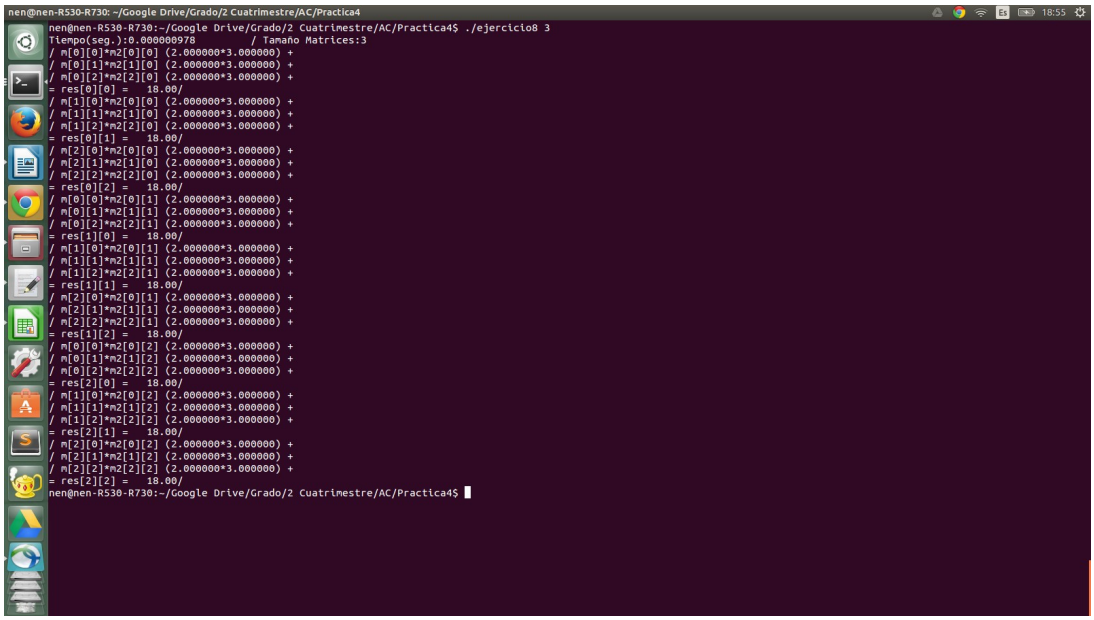
NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CÓDIGO FUENTE: `pmm-secuencial.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//#include <omp.h>
#define PRINTF_ALL

main(int argc, char **argv)
{
             if(argc < 2) {
                          fprintf(stderr,"Falta fila y columna\n");
                          exit(-1);
             }

             struct timespec cgt1,cgt2; double ncgt; //para tiempo de
ejecución
             int i,k, n = atoi(argv[1]),h;
             double sumalocal=0;
             double **m;
             double **m2;
             double **res;
             m = (double**)malloc(n*sizeof(double*));
             m2 = (double**)malloc(n*sizeof(double*));
             res = (double**)malloc(n*sizeof(double*));

             //Reservo el espacio para las matrices
             for(i=0;i<n;++i)
                          m[i]=(double*)malloc(n*sizeof(double));
             for(i=0;i<n;++i)
                          m2[i]=(double*)malloc(n*sizeof(double));
             for(i=0;i<n;++i)
                          res[i] = (double*)malloc(n*sizeof(double));

             //Inicializo la matriz m y m2
             for (i=0; i<n; i++){
                          for(k=0;k<n;++k){
                                       m[i][k]=2;
                                       m2[i][k]=3;
                                       res[i][k]=0;
                          }
             }


             //Calculo la multiplicacion de la matriz por la matriz y obtengo
el tiempo
             clock_gettime(CLOCK_REALTIME,&cgt1);
             for (h=0; h<n; h++){
                          for (i=0; i<n; i++){
                                       for(k=0;k<n;++k){
                                                    sumalocal+=m[i][k]*m2[k]
[h];
                                       }
```

```
                                                    res[h][i]=sumalocal;
                                                    sumalocal=0;
                            }
                }
                clock_gettime(CLOCK_REALTIME,&cgt2);
                ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
                //Imprimo los resultados
                #ifdef PRINTF_ALL
                printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\n",ncgt,n);
                for (h=0; h<n; h++){
                            for (i=0; i<n; i++){
                                        for(k=0;k<n;++k){
                                                    printf("/ m[%d]
[%d]*m2[%d][%d] (%8.6f*%8.6f) + \n",i,k,k,h,m[i][k],m2[k][h]);
                                        }
                                        printf("= res[%i][%i] =%8.2f/
\n",h,i,res[h][i]);
                            }


                }
                #else
        printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\t/ m[0][0]*m2[0]
[0]=res[0][0](%8.6f+%8.6f=%8.6f) // m[%d][%d]*m2[%d][%d]=res[%d][%d](%8.6f+
%8.6f=%8.6f) /\n", ncgt,n,m[0][0],m2[0][0],res[0][0],n-1,n-1,n-1,n-1,n-1,n-
1,m[n-1][n-1],m2[n-1][n-1],res[n-1][n-1]);
                #endif
                for(i=0;i<n;++i){
                            free(m[i]);
                }

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

                            free(m2[i]);
                }
                for(i=0;i<n;++i)
                            free(res[i]);
        free(m); // libera el espacio reservado para m
                free(m2); // libera el espacio reservado para m2
                free(res); // libera el espacio reservado para res

}
```

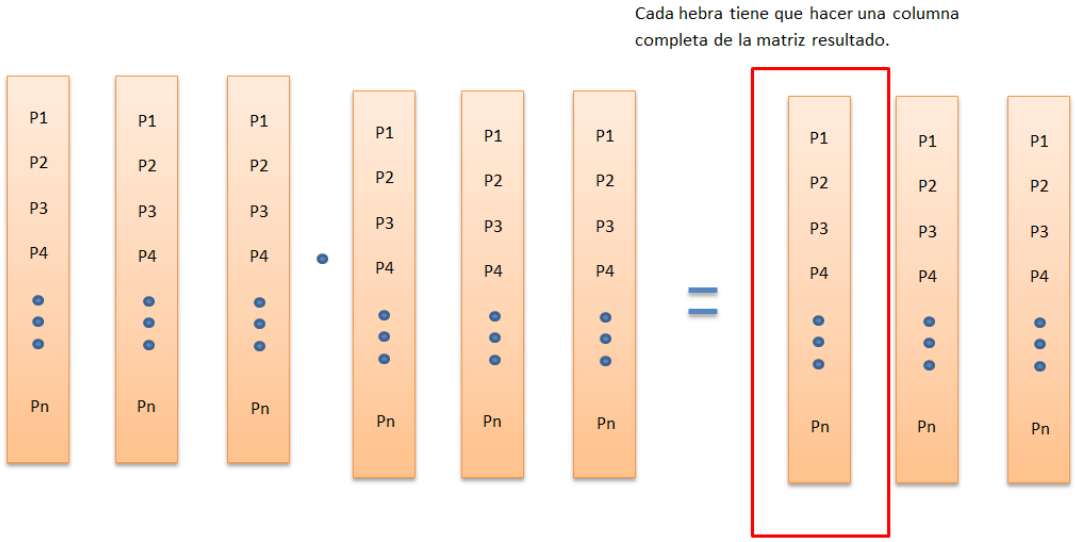
CAPTURAS DE PANTALLA:



(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: La descomposición se basa en división por iteraciones de columnas completas, es decir, que a cada hebra le toca obtener una columna de la matriz resultante, teniendo en cuenta que el chunk tiene el valor por defecto y que los threads son por defecto, y la

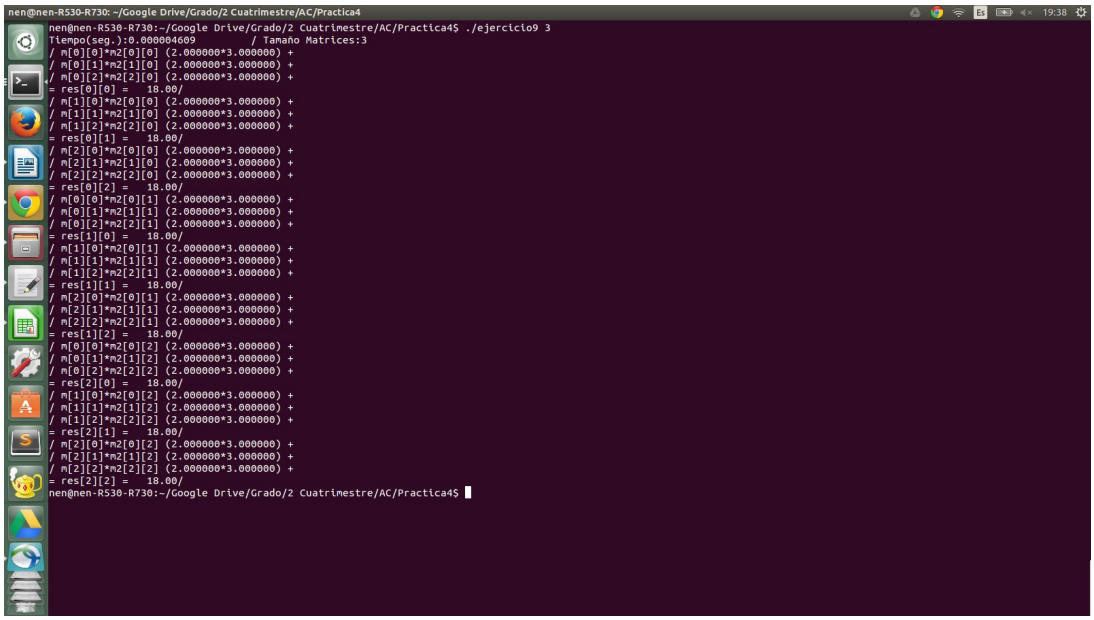


CÓDIGO FUENTE: pmm-OpenMP.c

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

main(int argc, char **argv)
{
		if(argc < 2) {
				fprintf(stderr,"Falta fila y columna\n");
				exit(-1);
		}

		struct timespec; double ncgt,cgt1,cgt2; //para tiempo de
ejecución
		int i,k, n = atoi(argv[1]),h;
		double sumalocal=0;
		double **m;
		double **m2;
		double **res;
		m = (double**)malloc(n*sizeof(double*));
		m2 = (double**)malloc(n*sizeof(double*));
		res = (double**)malloc(n*sizeof(double*));

		//Reservo el espacio para las matrices
		for(i=0;i<n;++i)
				m[i]=(double*)malloc(n*sizeof(double));
		for(i=0;i<n;++i)
				m2[i]=(double*)malloc(n*sizeof(double));
		for(i=0;i<n;++i)
				res[i] = (double*)malloc(n*sizeof(double));

		//Inicializo la matriz m y m2
		#pragma omp parallel for private(k)
		for (i=0; i<n; i++){
				for(k=0;k<n;++k){
						m[i][k]=2;
						m2[i][k]=3;
						res[i][k]=0;
				}
		}


		//Calculo la multiplicacion de la matriz por la matriz y obtengo
el tiempo
		#pragma omp parallel private(k,i,sumalocal)
		{
				#pragma omp single
				cgt1 = omp_get_wtime();
				#pragma omp for
				for (h=0; h<n; h++){
						for (i=0; i<n; i++){
								for(k=0;k<n;++k){

		sumalocal+=m[i][k]*m2[k][h];

								}
								#pragma omp critical
								{
										res[h]
[i]=sumalocal;
```

```
            sumalocal=0;
                                                    }
                                        }
                            }
                            #pragma omp single
                            cgt2 = omp_get_wtime();
            }
            ncgt=(cgt2-cgt1)/*/(1.e+9)*/;
            //Imprimo los resultados
            #ifdef PRINTF_ALL
            printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\n",ncgt,n);
            for (h=0; h<n; h++){
                            for (i=0; i<n; i++){
                                        for(k=0;k<n;++k){
                                                    printf("/ m[%d]
[%d]*m2[%d][%d] (%8.6f*%8.6f) + \n",i,k,k,h,m[i][k],m2[k][h]);
                                        }
                                        printf("= res[%i][%i] =%8.2f/
\n",h,i,res[h][i]);
                            }


            }
            #else
        printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\t/ m[0][0]*m2[0]
[0]=res[0][0](%8.6f+%8.6f=%8.6f) // m[%d][%d]*m2[%d][%d]=res[%d][%d](%8.6f+
%8.6f=%8.6f) /\n", ncgt,n,m[0][0],m2[0][0],res[0][0],n-1,n-1,n-1,n-1,n-1,n-
1,m[n-1][n-1],m2[n-1][n-1],res[n-1][n-1]);
            #endif
            for(i=0;i<n;++i){
                            free(m[i]);
            }

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

                            free(m2[i]);
            }
            for(i=0;i<n;++i)
                            free(res[i]);
        free(m); // libera el espacio reservado para m
            free(m2); // libera el espacio reservado para m2
            free(res); // libera el espacio reservado para res

}
```

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 tres tamaños de las matrices. Debe recordar usar –O2 al compilar. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas . Consulte la Lección 6/Tema 2.

ESTUDIO DE ESCALABILIDAD EN ATCGRID:

ATCGRID

procesadores	f	Tamaño Matriz	ley de amdahl
1	0,000155816	300	1
2	0,000155816		1,999844184
3	0,000155816		2,999688368
4	0,000155816		3,999532552
5	0,000155816		4,999376736
6	0,000155816		5,99922092
7	0,000155816		6,999065104
8	0,000155816		7,998909288
9	0,000155816		8,998753472
10	0,000155816		9,998597656
11	0,000155816		10,99844184
12	0,000155816		11,998286024
1	0,000710453	500	1
2	0,000710453		1,999289547
3	0,000710453		2,998579094
4	0,000710453		3,997868641
5	0,000710453		4,997158188
6	0,000710453		5,996447735
7	0,000710453		6,995737282
8	0,000710453		7,995026829
9	0,000710453		8,994316376
10	0,000710453		9,993605923
11	0,000710453		10,99289547
12	0,000710453		11,992185017
1	0,004322292	700	1
2	0,004322292		1,995677708
3	0,004322292		2,991355416
4	0,004322292		3,987033124
5	0,004322292		4,982710832
6	0,004322292		5,97838854
7	0,004322292		6,974066248
8	0,004322292		7,969743956
9	0,004322292		8,965421664
10	0,004322292		9,961099372
11	0,004322292		10,95677708
12	0,004322292		11,952454788

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

PCLOCAL

procesadores	f	Tamaño Matriz	ley de amdahl
1	0,000156165	300	1
2	0,000156165		1,999843835
1	0,001490276	500	1
2	0,001490276		1,998509724
1	0,003097046	700	1
2	0,003097046		1,996902954