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) : Álvaro Fernández García
Grupo de prácticas: A2
Fecha de entrega:
Fecha evaluación en clase:

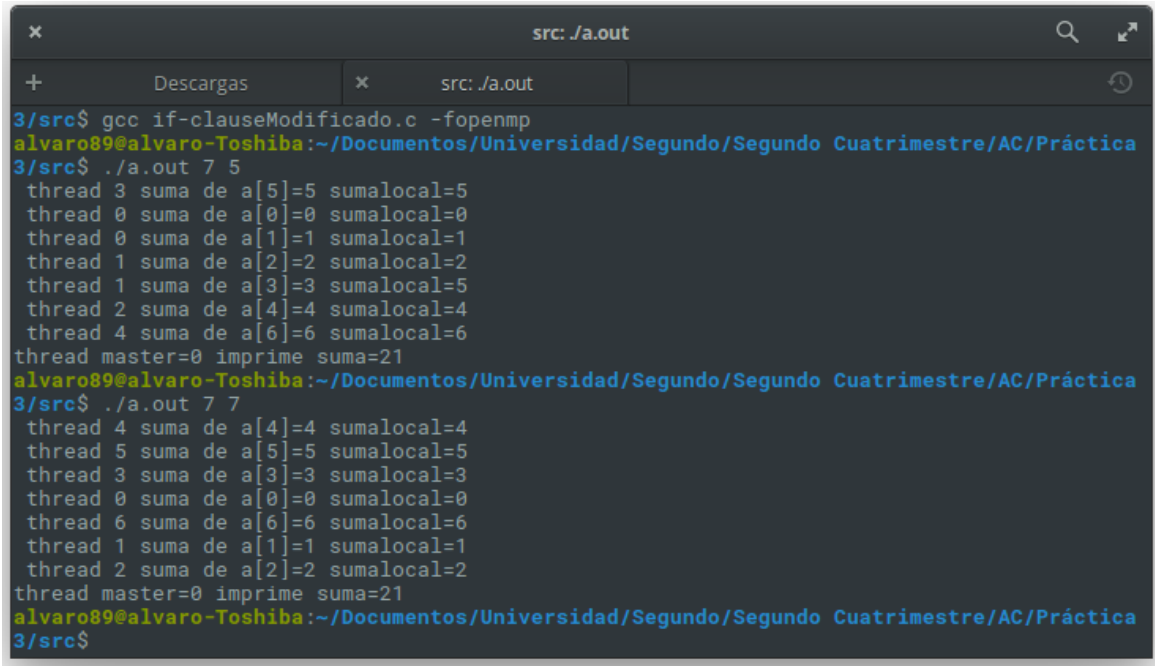
Ejercicios basados en los ejemplos del seminario práctico

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 <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]-Falta iteraciones o Número de
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) num_threads(x) 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: Como podemos ver en la captura anterior, efectivamente la cláusula está realizando correctamente su trabajo (modificar el número de threads que ejecutan la región parallel), en el primer caso se ha especificado que el número de hebras sea 5 y como vemos los identificadores de hebra son 0, 1, 2, 3 y 4 (5 en total). En la segunda ejecución por su parte se ha especificado que el número de threads sea 7, y efectivamente los identificadores de hebra son 0, 1, 2, 3, 4, 5 y 6.

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			scheduled-clause.c			scheduleg-clause.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	1	0
1	1	0	0	1	0	0	0	1	0
2	0	1	0	0	1	0	0	1	0
3	1	1	0	0	1	0	0	1	0
4	0	0	1	0	0	1	0	1	0
5	1	0	1	0	0	1	0	1	0
6	0	1	1	0	0	1	0	1	0
7	1	1	1	0	0	1	0	1	0
8	0	0	0	0	1	0	1	0	1
9	1	0	0	0	1	0	1	0	1
10	0	1	0	1	1	0	1	0	1
11	1	1	0	1	1	0	1	0	1

12	0	0	1	1	1	0	0	1	1
13	1	0	1	1	1	0	0	1	1
14	0	1	1	1	1	0	0	1	1
15	1	1	1	1	1	0	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			scheduled-clause.c			scheduleg-clause.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	3	2	2	1	0	1
1	1	0	0	2	2	2	1	0	1
2	2	1	0	0	0	2	1	0	1
3	3	1	0	1	0	2	1	0	1
4	0	2	1	3	1	1	2	1	2
5	1	2	1	3	1	1	2	1	2
6	2	3	1	3	3	1	2	1	2
7	3	3	1	3	3	1	3	3	2
8	0	0	2	3	2	3	3	3	0
9	1	0	2	2	2	3	3	3	0
10	2	1	2	2	2	3	0	2	0
11	3	1	2	2	2	3	0	2	0
12	0	2	3	2	2	0	1	0	3
13	1	2	3	2	2	0	1	0	3
14	2	3	3	2	2	0	1	0	3
15	3	3	3	2	2	0	1	0	3

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

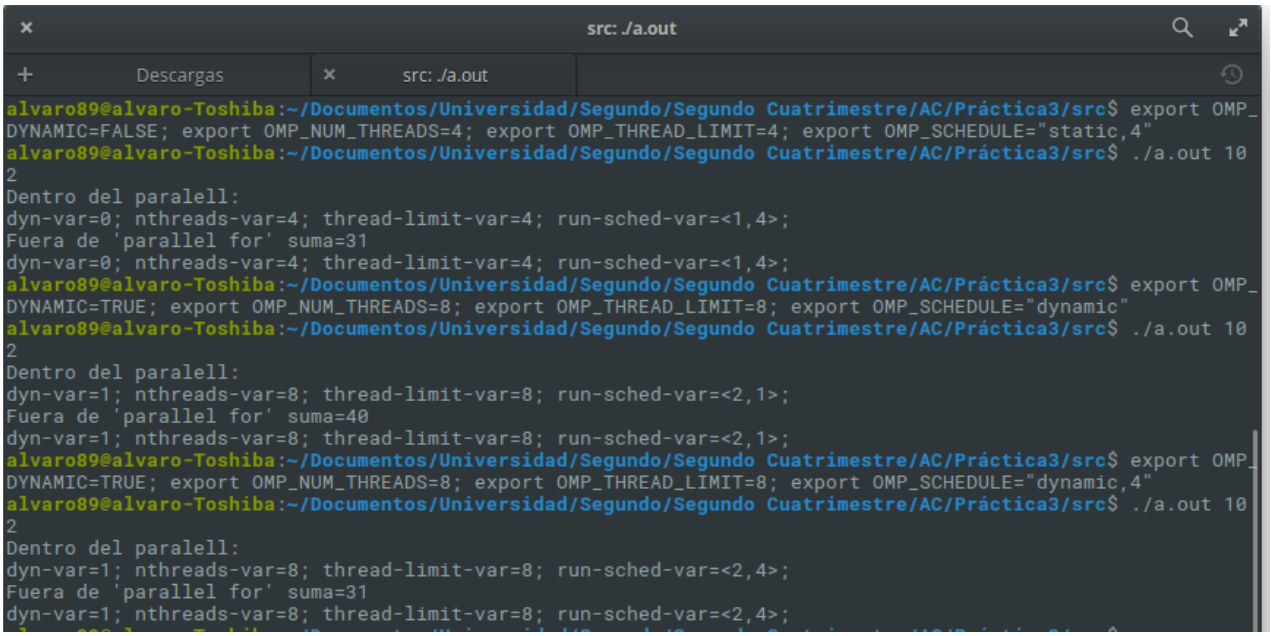
RESPUESTA: La principal diferencia entre estas tres opciones reside en que por un lado static realiza la distribución de las iteraciones en tiempo de compilación mientras que dynamic y guided lo hacen en tiempo de ejecución. Static y Dynamic comparten que calculan las unidades dividiendo el número de iteraciones entre el chunk. Por su parte, static reparte las unidades obtenidas en round robin como se puede ver en la tabla. Dynamic por el contrario las va repartiendo según la ejecución, por tanto las hebras más rápidas ejecutarán más unidades (de ahí que en las tablas se vean hebras que ejecutan muchas iteraciones seguidas). Por último, guided calcula las unidades dividiendo el número de iteraciones restantes entre el número de threads (pero nunca más pequeño que chunk) de ahí que las hebras vayan ejecutando cada vez menos iteraciones hasta llegar al chunk impuesto (esto se aprecia mejor en la segunda tabla con chunk=4).

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;
               omp_sched_t kind;
               int modifier;

               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;

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

                              }

                              #pragma omp single
                              {
                                             omp_get_schedule(&kind, &modifier);
                                             printf("Dentro del paralell:\n");
                                             printf("dyn-var=%d; nthreads-var=%d;
thread-limit-var=%d; run-sched-var=<%d,%d>;\n", omp_get_dynamic(),
omp_get_max_threads(),omp_get_thread_limit(),kind, modifier);
                              }
               }

               printf("Fuera de 'parallel for' suma=%d\n",suma);
               omp_get_schedule(&kind, &modifier);
               printf("dyn-var=%d; nthreads-var=%d; thread-limit-var=%d; run-sched-
var=<%d,%d>;\n", omp_get_dynamic(), omp_get_max_threads(),omp_get_thread_limit(),kind,
modifier);
}
```

CAPTURAS DE PANTALLA:



RESPUESTA: Se muestran los mismos valores dentro y fuera del parallel.

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;

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

```
                                }

                                #pragma omp single
                                {
                                                printf("Dentro del paralell:\n");
                                                printf("Threads=%d; Procs=%d; In paralell=
%d\n", omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
                                }
                }

                printf("Fuera de 'parallel for' suma=%d\n",suma);
                printf("Threads=%d; Procs=%d; In paralell=%d\n", omp_get_num_threads(),
omp_get_num_procs(), omp_in_parallel());
}
```

CAPTURAS DE PANTALLA:

RESPUESTA: Como es lógico el número de procesadores disponibles para el programa en el momento de la ejecución es el mismo antes y después. Sin embargo, el número de threads no es el mismo, dentro del parallel hay cuatro hebras ejecutando el código mientras que fuera de la misma existe un único flujo de control. Por otra parte también se modifica el valor de In parallel, ya que este devuelve True cuando se está dentro de una construcción parallel y False cuando no.

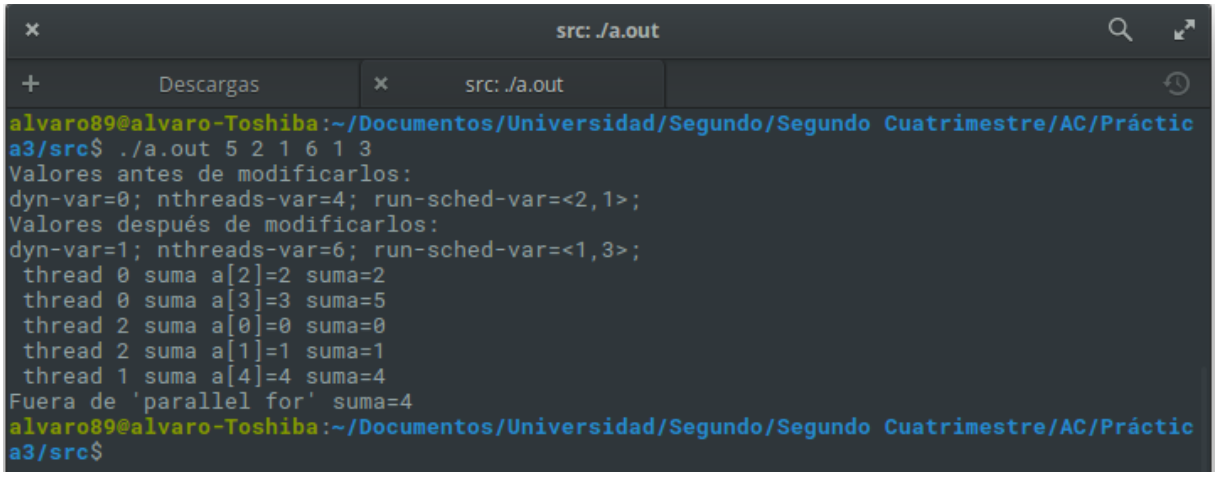
5. Añadir al programa `scheduled-clause.c` lo necesario para modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados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

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

                int i, n=200,chunk,a[n],suma=0, dv, ntv, m;
                omp_sched_t kind, knd;
                int modifier;

                if(argc < 7) {
                                fprintf(stderr,"\nUso: <Iteraciones> <chunk> <dyn-var>
<nthreads-var> <kind,modifier>\n");
                                exit(-1);
                }
```

```
                n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
                dv = atoi(argv[3]);
                ntv = atoi(argv[4]);
                knd = atoi(argv[5]);
                m = atoi(argv[6]);

                printf("Valores antes de modificarlos:\n");
                omp_get_schedule(&kind, &modifier);
                printf("dyn-var=%d; nthreads-var=%d; run-sched-var=<%d,%d>;\n",
omp_get_dynamic(), omp_get_max_threads(),kind, modifier);

                omp_set_dynamic(dv);
                omp_set_num_threads(ntv);
                omp_set_schedule(knd,m);

                printf("Valores después de modificarlos:\n");
                omp_get_schedule(&kind, &modifier);
                printf("dyn-var=%d; nthreads-var=%d; run-sched-var=<%d,%d>;\n",
omp_get_dynamic(), omp_get_max_threads(),kind, modifier);

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

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

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

CAPTURAS DE PANTALLA:



RESPUESTA: Basta simplemente con leer de la línea de comandos las opciones correspondientes y cambiarlas con las funciones apropiadas.

Resto de ejercicios

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

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

CÓDIGO FUENTE: pmtv-secuencial.c

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

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

		struct timespec cgt1,cgt2;
		double ncgt;

		if(argc < 2){
				printf("Modo de empleo: %s, N\n", argv[0]);
				exit(-1);
		}

		int N = atoi(argv[1]);

		//Reservar memoria dinámica:

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

		double* V = (double*) malloc(N*sizeof(double));

		double* R = (double*) malloc(N*sizeof(double));

		if(M == NULL || V == NULL || R == NULL){
				printf("No se ha podido reservar memoria\n");
				exit(-1);
		}

		//Inicializar elementos:

		for(int i = 0; i<N; ++i)
				for(int j = 0; j<N; ++j){
						if(j >= i)
								M[i][j] = i+j+1;
						else
								M[i][j] = 0.0;
				}

		for(int i = 0; i<N; ++i){
				V[i] = N*0.1 + i*0.1;
				R[i] = 0;
		}

		//Realizar R = M * V y medir tiempo:

		clock_gettime(CLOCK_REALTIME, &cgt1);

		for(int i = 0; i<N; ++i)
				for(int j = i; j<N; ++j)
						R[i] += (M[i][j] * V[j]);

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

		//Imprimir resultados:

		if(N<12){
				printf("Tiempo transcurrido (seg): %11.9f \n\nMatriz
generada:\n", ncgt);
				for(int i = 0; i<N; ++i){
						for(int j = 0; j<N; ++j)
								printf("[%.0f]", M[i][j]);
						printf("\n");
				}
				printf("\n\nVector generado:\n");
```

```
                                for(int i = 0; i<N; ++i)
                                        printf("[%.1f]", V[i]);

                                printf("\n\nResultado:\n");
                                for(int i = 0; i<N; ++i)
                                        printf("[%.1f]", R[i]);


                                printf("\n\n");
                }
                else
                                printf("Tiempo(seg.): %11.9f \nTamaño: %u \nM[0]
[-]*V=R[0] = %.1f \nM[%d][-]*V=R[%d] = %.1f \n",ncgt,N, R[0], N-1, N-1, R[N-1]);

                //Liberar memoria:
                for(int i = 0; i<N; ++i)
                                free(M[i]);

                free(M);
                free(V);
                free(R);

                exit(0);
}
```

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

El orden de complejidad del algoritmo sigue siendo cuadrático, igual que el anterior, ya que el algoritmo es esencialmente el mismo, dos bucles anidados. Lo único que varía, es que en este implementación, el bucle que itera las columnas empieza desde i en lugar de 0.

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: Como se puede observar en la gráfica (en la cual se representa la media de las dos ejecuciones realizadas) realmente no existe una alternativa que sea mejor o peor, dependiendo del caso se obtienen mejores tiempos o peores, aunque si tenemos en cuenta los aspectos teóricos creo que la mejor alternativa sería guided, ya que esta funciona bien cuando se desconoce el número de iteraciones (como en este caso) y es la que presenta unos tiempos más uniformes conforme varía el chunk. Además tiene menos sobrecarga que dynamic.

a) Para static utiliza el número de threads, y para dynamic y guided utiliza 1, (de ahí que la gráfica solo presente dos puntos para estos). Para obtener los valores de dynamic y guided he utilizado omp_get_schedule(&kind, &modifier) dentro del programa y he imprimido el modifier. Para static (como daba cero lo anterior lo cual es imposible) he realizado una prueba en la que se imprimía la iteración y el id de hebra con tamaños manejables.

b) En cada iteración del bucle externo se realizan N sumas y N multiplicaciones, por tanto, basta con calcular el número de iteraciones que realiza cada thread para cada chunk. Las unidades se calculan como el número de iteraciones del bucle entre el chunk (N/chunk) Dichas unidades se repartirán en round robin. En definitiva cada hebra hará (N/chunk)/Num_trheads unidades. Como cada unidad tiene chunk iteraciones del bucle externo la fórmula quedaría como ((N/chunk)/Num_trheads)*chunk*N

Chunk	Defecto (12 en este caso)	1	64
Unidades	1280	15360	240
Operaciones	19660800	19660800	19660800

c) En dynamic cada hebra realizará un número distinto, las unidades a repartir son las mismas pero al no ser en round robin, si no que al hacerse en tiempo de ejecución, las hebras más rápidas recibirán más unidades y en consecuencia ejecutarán más operaciones. Con guided pasa más de lo mismo, solo que en este caso las unidades se calculan como el número de iteraciones restantes entre el número de threads, en consecuencia cada vez irán recibiendo unidades más pequeñas y el número de operaciones irá disminuyendo.

CÓDIGO FUENTE: `pmtv-OpenMP.c`

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

#ifdef _OPENMP
               #include <omp.h>
#endif

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

               struct timespec cgt1,cgt2;
               double ncgt;
```

```
                int i,j;

                if(argc < 2){
                                printf("Modo de empleo: %s, N\n", argv[0]);
                                exit(-1);
                }

                int N = atoi(argv[1]);

                //Reservar memoria dinámica:

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

                double* V = (double*) malloc(N*sizeof(double));

                double* R = (double*) malloc(N*sizeof(double));

                if(M == NULL || V == NULL || R == NULL){
                                printf("No se ha podido reservar memoria\n");
                                exit(-1);
                }

                //Inicializar elementos:

                #pragma omp parallel
                {
                                #pragma omp for private(j)
                                for(i = 0; i<N; ++i)
                                                for(j = 0; j<N; ++j){
                                                                if(j >= i)
                                                                                M[i][j] =
i+j+1;
                                                                else
                                                                                M[i][j] =
0.0;
                                                }

                                #pragma omp for
                                for(i = 0; i<N; ++i){
                                                V[i] = N*0.1 + i*0.1;
                                                R[i] = 0;
                                }
                }

                //Realizar R = M * V y medir tiempo:

                clock_gettime(CLOCK_REALTIME, &cgt1);

                #pragma omp parallel for private(j) schedule(runtime)
                for(i = 0; i<N; ++i)
                                for(j = i; j<N; ++j)
                                                R[i] += (M[i][j] * V[j]);

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

                //Imprimir resultados:
                if(N<12){
                                printf("Tiempo transcurrido (seg): %11.9f \n\nMatriz
generada:\n", ncgt);
                                for(i = 0; i<N; ++i){
                                                for(j = 0; j<N; ++j)
                                                                printf("[%.0f]", M[i][j]);
                                                printf("\n");
                                }
                                printf("\n\nVector generado:\n");
                                for(i = 0; i<N; ++i)
```

```
                                        printf("[%.1f]", V[i]);

                                printf("\n\nResultado:\n");
                                for(i = 0; i<N; ++i)
                                        printf("[%.1f]", R[i]);


                                printf("\n\n");
                }
                else
                                printf("Tiempo(seg.): %11.9f \nTamaño: %u \nM[0]
[-]*V=R[0] = %.1f \nM[%d][-]*V=R[%d] = %.1f \n",ncgt,N, R[0], N-1, N-1, R[N-1]);



                //Liberar memoria:
                for(i = 0; i<N; ++i)
                                free(M[i]);

                free(M);
                free(V);
                free(R);

                exit(0);
}
```

DESCOMPOSICIÓN DE DOMINIO:

$$\begin{matrix}T1\\T2\\T3\\T4\end{matrix}\begin{pmatrix}a_{11} & a_{12} & a_{13} & a_{14}\\ a_{21} & a_{22} & a_{23} & a_{24}\\ a_{31} & a_{32} & a_{33} & a_{34}\\ a_{41} & a_{42} & a_{43} & a_{44}\end{pmatrix} * \begin{pmatrix}b_1\\b_2\\b_3\\b_4\end{pmatrix} = \begin{pmatrix}a_{11}*b_1+a_{12}*b_2+a_{13}*b_3+a_{14}*b_4\\ a_{21}*b_1+a_{22}*b_2+a_{23}*b_3+a_{24}*b_4\\ a_{31}*b_1+a_{32}*b_2+a_{33}*b_3+a_{34}*b_4\\ a_{41}*b_1+a_{42}*b_2+a_{43}*b_3+a_{44}*b_4\end{pmatrix}$$

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

```
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctic
a3/src$ gcc pmtv-OpenMP.c -o pmtv -O2 -fopenmp
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctic
a3/src$ ./pmtv 5
Tiempo transcurrido (seg): 0.000031518

Matriz generada:
[1][2][3][4][5]
[0][3][4][5][6]
[0][0][5][6][7]
[0][0][0][7][8]
[0][0][0][0][9]


Vector generado:
[0.5][0.6][0.7][0.8][0.9]

Resultado:
[11.5][14.0][14.6][12.8][8.1]

alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctic
a3/src$
```

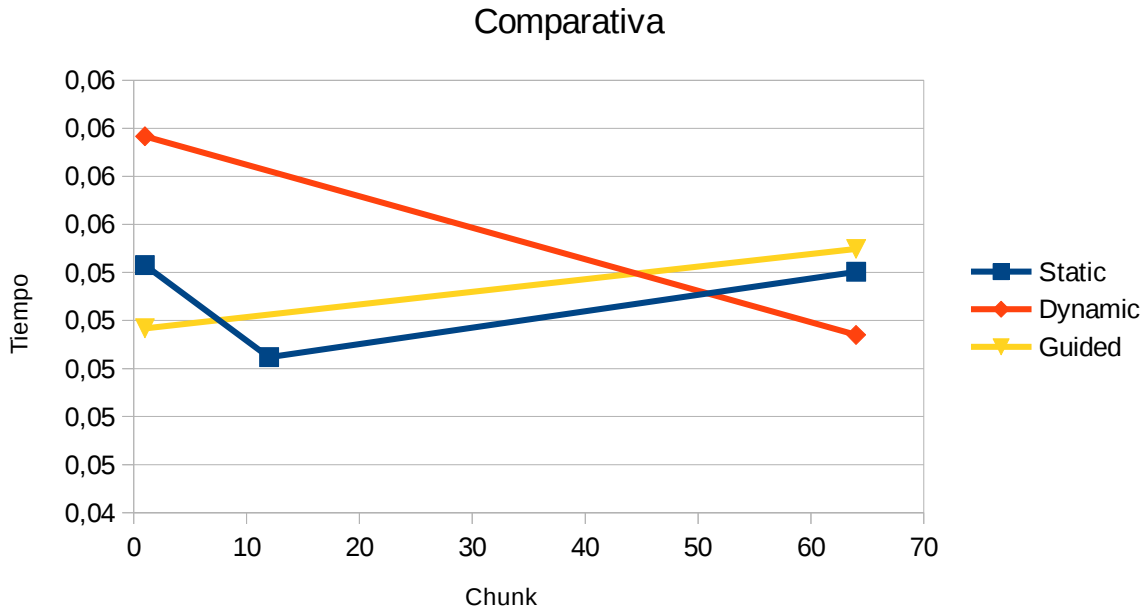
TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash

#PBS -N pmtv-OpenMP
#PBS -q ac

#Alguna información:
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"

#Asegurarnos de que el número de threads concide con el número de cores:
export OMP_NUM_THREADS=12

#STATIC
                #Chunk = default
                export OMP_SCHEDULE="static"
                $PBS_O_WORKDIR/pmtv 15360
                #Chunk = 1
                export OMP_SCHEDULE="static,1"
                $PBS_O_WORKDIR/pmtv 15360
                #Chunk = 64
                export OMP_SCHEDULE="static,64"
                $PBS_O_WORKDIR/pmtv 15360
#DYNAMIC
                #Chunk = default
                export OMP_SCHEDULE="dynamic"
                $PBS_O_WORKDIR/pmtv 15360
                #Chunk = 1
                export OMP_SCHEDULE="dynamic,1"
                $PBS_O_WORKDIR/pmtv 15360
                #Chunk = 64
                export OMP_SCHEDULE="dynamic,64"
                $PBS_O_WORKDIR/pmtv 15360
#GUIDED
                #Chunk = default
                export OMP_SCHEDULE="guided"
                $PBS_O_WORKDIR/pmtv 15360
                #Chunk = 1
                export OMP_SCHEDULE="guided,1"
                $PBS_O_WORKDIR/pmtv 15360
                #Chunk = 64
                export OMP_SCHEDULE="guided,64"
                $PBS_O_WORKDIR/pmtv 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= 15360, 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.053418337	0.057794493	0.054353235
1	0.054816725	0.059750396	0.044318154
64	0.048033619	0.049972824	0.055358408
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.047529733	0.057949659	0.049093953
1	0.053795515	0.059588058	0.059033433
64	0.060024330	0.052829562	0.054603271



(Nota: En la gráfica dynamic y guided solo tienen dos puntos ya que el defecto también es 1, por su parte en static el defecto es el número de threads, 12 en nuestro caso. La gráfica representa la media de las dos mediciones realizadas)

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.

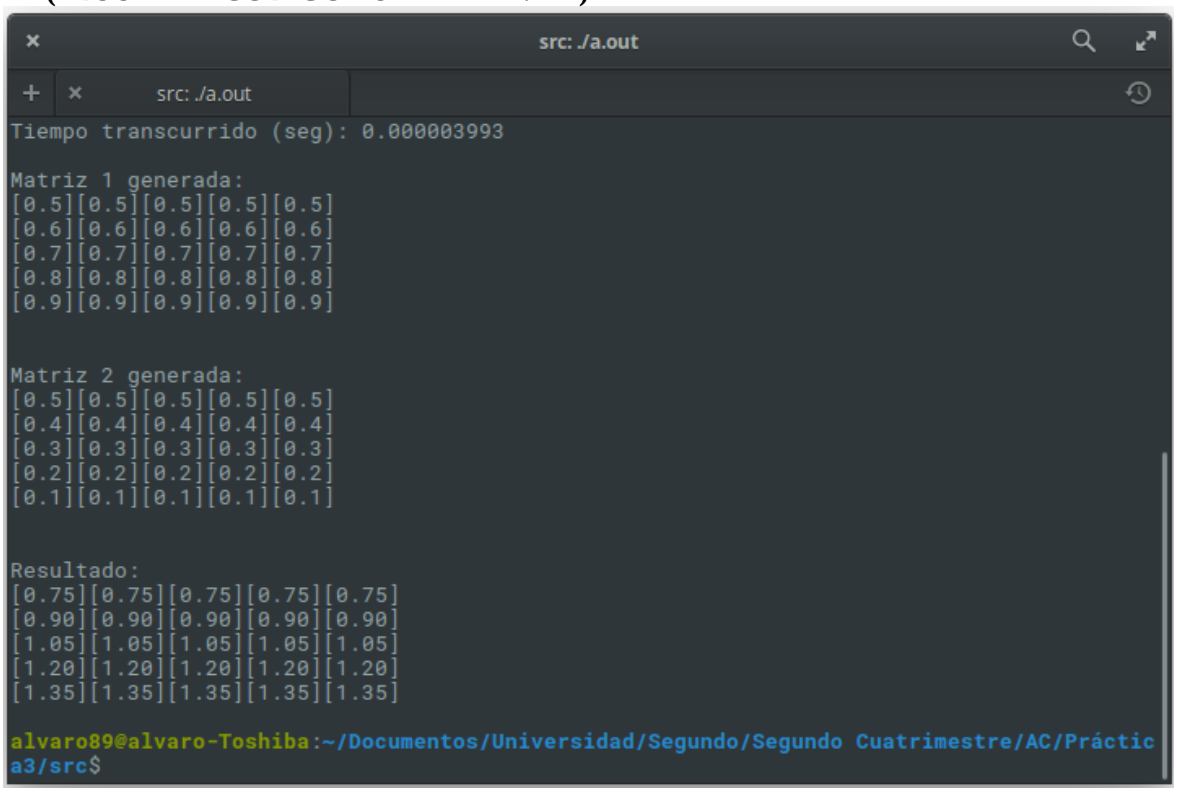
CÓDIGO FUENTE: `pmm-secuencial.c`

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

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

                struct timespec cgt1,cgt2;
                double ncgt;

                if(argc < 2){
                                printf("Modo de empleo: %s, N\n", argv[0]);
                                exit(-1);
                }

                int N = atoi(argv[1]);

                //Reservar memoria dinámica:

                double** A = (double**) malloc(N*sizeof(double*));
                double** B = (double**) malloc(N*sizeof(double*));
                double** C = (double**) malloc(N*sizeof(double*));
                for(int i = 0; i<N; ++i){
                                A[i] = (double*) malloc(N*sizeof(double));
                                B[i] = (double*) malloc(N*sizeof(double));
                                C[i] = (double*) malloc(N*sizeof(double));
```

```
                }

                if(A == NULL || B == NULL || C == NULL){
                                printf("No se ha podido reservar memoria\n");
                                exit(-1);
                }

                //Inicializar elementos:

                for(int i = 0; i<N; ++i)
                                for(int j = 0; j<N; ++j){
                                                A[i][j] = 0.0;
                                                B[i][j] = N*0.1 + i*0.1;
                                                C[i][j] = N*0.1 - i*0.1;
                                }

                //Realizar R = M * V y medir tiempo:

                clock_gettime(CLOCK_REALTIME, &cgt1);

                for(int i = 0; i<N; ++i)
                                for(int j = 0; j<N; ++j)
                                                for(int k = 0; k<N; ++k)
                                                                A[i][j] += B[i][k] * C[k]
[j];

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

                //Imprimir resultados:

                if(N<12){
                                printf("Tiempo transcurrido (seg): %11.9f \n\nMatriz 1
generada:\n", ncgt);
                                for(int i = 0; i<N; ++i){
                                                for(int j = 0; j<N; ++j)
                                                                printf("[%.1f]", B[i][j]);
                                                printf("\n");
                                }
                                printf("\n\nMatriz 2 generada:\n");
                                for(int i = 0; i<N; ++i){
                                                for(int j = 0; j<N; ++j)
                                                                printf("[%.1f]", C[i][j]);
                                                printf("\n");
                                }

                                printf("\n\nResultado:\n");
                                for(int i = 0; i<N; ++i){
                                                for(int j = 0; j<N; ++j)
                                                                printf("[%.2f]", A[i][j]);
                                                printf("\n");
                                }

                                printf("\n");
                }
                else
                                printf("Tiempo(seg.): %11.9f \nTamaño: %u \nB[0][0]*C[0]
[0]=A[0][0] = %.1f \nB[%d][%d]*C[%d][%d]=A[%d][%d] = %.2f \n",ncgt,N, A[0][0], N-1, N-
1,N-1, N-1,N-1, N-1, A[N-1][N-1]);

                //Liberar memoria:
                for(int i = 0; i<N; ++i){
                                free(A[i]);
                                free(B[i]);
                                free(C[i]);
                }

                free(A);
                free(B);
```

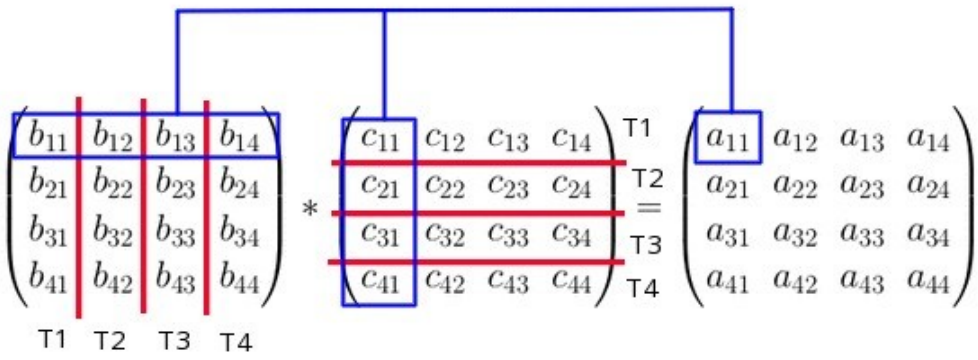
```
                free(C);

                exit(0);
}
```

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`

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

```
#ifdef _OPENMP
                #include <omp.h>
#endif

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

                struct timespec cgt1,cgt2;
                double ncgt;
                int i,j,k;

                if(argc < 2){
                                printf("Modo de empleo: %s, N\n", argv[0]);
                                exit(-1);
                }

                int N = atoi(argv[1]);

                //Reservar memoria dinámica:

                double** A = (double**) malloc(N*sizeof(double*));
                double** B = (double**) malloc(N*sizeof(double*));
                double** C = (double**) malloc(N*sizeof(double*));
                for(i = 0; i<N; ++i){
                                A[i] = (double*) malloc(N*sizeof(double));
                                B[i] = (double*) malloc(N*sizeof(double));
                                C[i] = (double*) malloc(N*sizeof(double));
                }

                if(A == NULL || B == NULL || C == NULL){
                                printf("No se ha podido reservar memoria\n");
                                exit(-1);
                }

                //Inicializar elementos:

                #pragma omp parallel for private(j)
                for(i = 0; i<N; ++i)
                                for(j = 0; j<N; ++j){
                                                A[i][j] = 0.0;
                                                B[i][j] = N*0.1 + i*0.1;
                                                C[i][j] = N*0.1 - i*0.1;
                                }

                //Realizar R = M * V y medir tiempo:

                clock_gettime(CLOCK_REALTIME, &cgt1);

                double res = 0.0;
                #pragma omp parallel private(j,i)
                {
                                for(i =0 ; i<N; ++i)
                                                for(j=0; j<N; ++j){
                                                                #pragma omp for
reduction(+:res)
                                                                for(k=0; k<N; ++k)
                                                                                res += B[i]
[k] * C[k][j];
                                                                #pragma omp single
                                                                {
                                                                                A[i][j] =
res;
                                                                                res = 0;
                                                                }
                                                }
                }

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

```
                //Imprimir resultados:

                if(N<12){
                                printf("Tiempo transcurrido (seg): %11.9f \n\nMatriz 1
generada:\n", ncgt);
                                for(i = 0; i<N; ++i){
                                                for(j = 0; j<N; ++j)
                                                                printf("[%.1f]", B[i][j]);
                                                printf("\n");
                                }
                                printf("\n\nMatriz 2 generada:\n");
                                for(i = 0; i<N; ++i){
                                                for(j = 0; j<N; ++j)
                                                                printf("[%.1f]", C[i][j]);
                                                printf("\n");
                                }

                                printf("\n\nResultado:\n");
                                for(i = 0; i<N; ++i){
                                                for(j = 0; j<N; ++j)
                                                                printf("[%.2f]", A[i][j]);
                                                printf("\n");
                                }

                                printf("\n");
                }
                else
                                printf("Tiempo(seg.): %11.9f \nTamaño: %u \nB[0][0]*C[0]
[0]=A[0][0] = %.1f \nB[%d][%d]*C[%d][%d]=A[%d][%d] = %.2f \n",ncgt,N, A[0][0], N-1, N-
1,N-1, N-1,N-1, N-1, A[N-1][N-1]);

                //Liberar memoria:
                for(i = 0; i<N; ++i){
                                free(A[i]);
                                free(B[i]);
                                free(C[i]);
                }

                free(A);
                free(B);
                free(C);

                exit(0);
}
```

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

```
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctic
a3/src$ gcc pmm-OpenMP.c -fopenmp
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctic
a3/src$ ./a.out 3
Tiempo transcurrido (seg): 0.014389771

Matriz 1 generada:
[0.3][0.3][0.3]
[0.4][0.4][0.4]
[0.5][0.5][0.5]


Matriz 2 generada:
[0.3][0.3][0.3]
[0.2][0.2][0.2]
[0.1][0.1][0.1]


Resultado:
[0.18][0.18][0.18]
[0.24][0.24][0.24]
[0.30][0.30][0.30]

alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctic
a3/src$ ./a.out 1000
Tiempo(seg.): 12.106910794
Tamaño: 1000
B[0][0]*C[0][0]=A[0][0] = 5005000.0
B[999][999]*C[999][999]=A[999][999] = 10004995.00
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctic
a3/src$
```

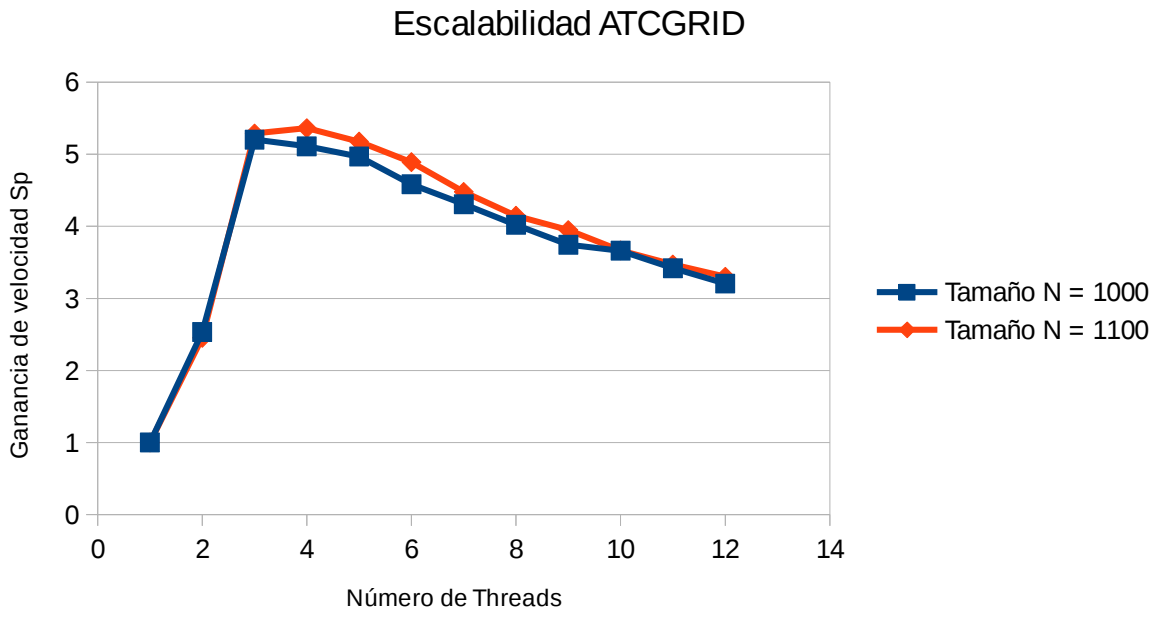
10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger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

ESTUDIO DE ESCALABILIDAD EN ATCGRID:

SCRIPT: pmm-OpenMP_atcgrid.sh

```
#!/bin/bash

#PBS -N pmm-OpenMP
#PBS -q ac

#Alguna información:
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"

echo "N=1000"
for((T=1;T<13;T=T+1))
do
                export OMP_NUM_THREADS=$T
                $PBS_O_WORKDIR/pmm 1000
done

echo "N=1100"
for((T=1;T<13;T=T+1))
do
                export OMP_NUM_THREADS=$T
                $PBS_O_WORKDIR/pmm 1100
done
```

Nº de Hebras (p)	Tiempo secuencial Ts	Tiempo paralelo para p hebras Tp(p)	Ganancia de velocidad S(p)
Tamaño N = 1000			
1	12,158947179	12,158947179	1
2	12,158947179	4,801690025	2,5322224291
3	12,158947179	2,337470277	5,2017547768
4	12,158947179	2,378852149	5,1112664501
5	12,158947179	2,447800528	4,9672949409
6	12,158947179	2,653292913	4,5825875912
7	12,158947179	2,822821191	4,3073742034
8	12,158947179	3,024579273	4,020045792
9	12,158947179	3,247710592	3,7438518102
10	12,158947179	3,320491248	3,661791666
11	12,158947179	3,556072408	3,4192068619
12	12,158947179	3,793776336	3,2049720653
Tamaño N = 1100			
1	16,647128082	16,647128082	1
2	16,647128082	6,785340497	2,453396125
3	16,647128082	3,148571702	5,2871999299
4	16,647128082	3,105079242	5,3612570838
5	16,647128082	3,216639679	5,1753163995
6	16,647128082	3,404592036	4,8896102399
7	16,647128082	3,71709716	4,4785291762
8	16,647128082	4,014075365	4,1471887218
9	16,647128082	4,216315832	3,9482640166
10	16,647128082	4,541037429	3,6659306034
11	16,647128082	4,796080909	3,4709856647
12	16,647128082	5,043220101	3,3008926338



ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```
#!/bin/bash

echo "N=1000" >> local_1000.txt
for((T=1;T<5;T=T+1))
do
                export OMP_NUM_THREADS=$T
                ./pmm 1000 >> local_1000.txt
done

echo "N=1100" >> local_1100.txt
for((T=1;T<5;T=T+1))
do
                export OMP_NUM_THREADS=$T
                ./pmm 1100 >> local_1100.txt
done
```

Nº de Hebras (p)	Tiempo secuencial Ts	Tiempo paralelo para p hebras Tp(p)	Ganancia de velocidad S(p)
Tamaño N = 1000			
1	10,251070992	10,251070992	1
2	10,251070992	5,561841517	1,8431073522
3	10,251070992	4,770149883	2,1490039608
4	10,251070992	5,456823746	1,8785783579
Tamaño N = 1100			
1	13,543738403	13,543738403	1
2	13,543738403	7,80966636	1,7342275302
3	13,543738403	6,199017416	2,1848201891
4	13,543738403	6,460663058	2,096338763