Grai2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Emilio Chica Jiménez
Grupo de prácticas: B2
Fecha de entrega:
Fecha evaluación en clase:

---

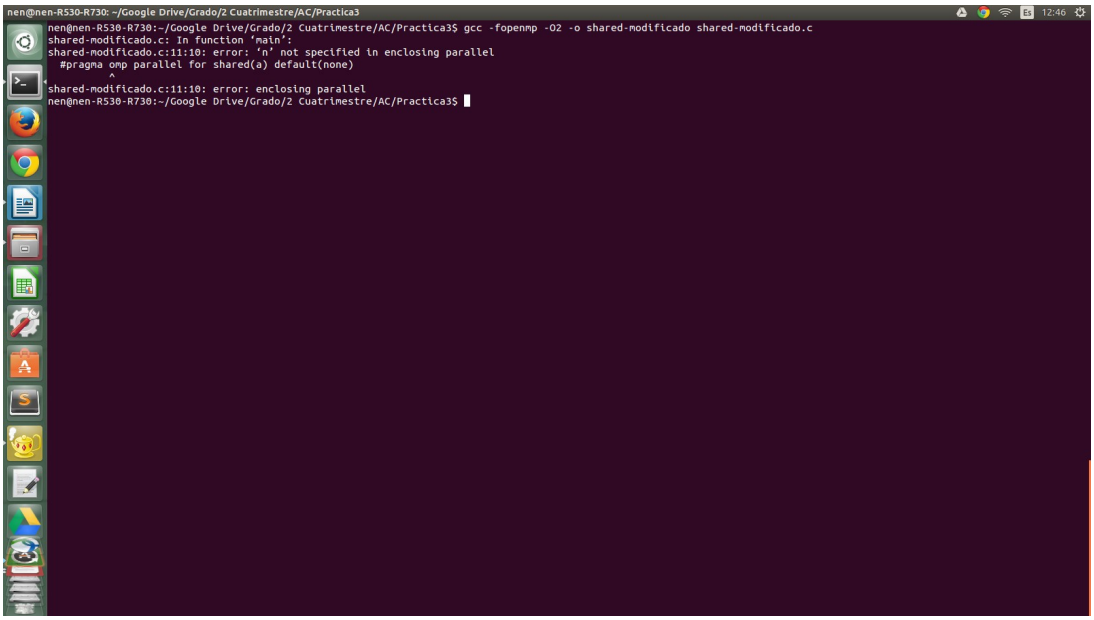
1. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? (añada una captura de pantalla que muestre lo que ocurre) **(b)** Resuelva el problema generado sin eliminar `default(none)`.Añada el código con la modificación al cuaderno de prácticas.

   RESPUESTA: Obtenemos un error debido a que no especificamos si la variable n, es privada o es compartida. Para solucionarlo tenemos que usarla como compartida para que todas las hebras puedan usar la variable.

   CÓDIGO FUENTE: `shared-clauseModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
            #include <omp.h>
#endif
main()
{
            int i, n = 7;
            int a[n];
            for (i=0; i<n; i++)
                        a[i] = i+1;
            #pragma omp parallel for shared(a,n),default(none)
                        for (i=0; i<n; i++) a[i] += i;

            printf("Después de parallel for:\n");
            for (i=0; i<n; i++)
                        printf("a[%d] = %d\n",i,a[i]);
}
```

   CAPTURAS DE PANTALLA:



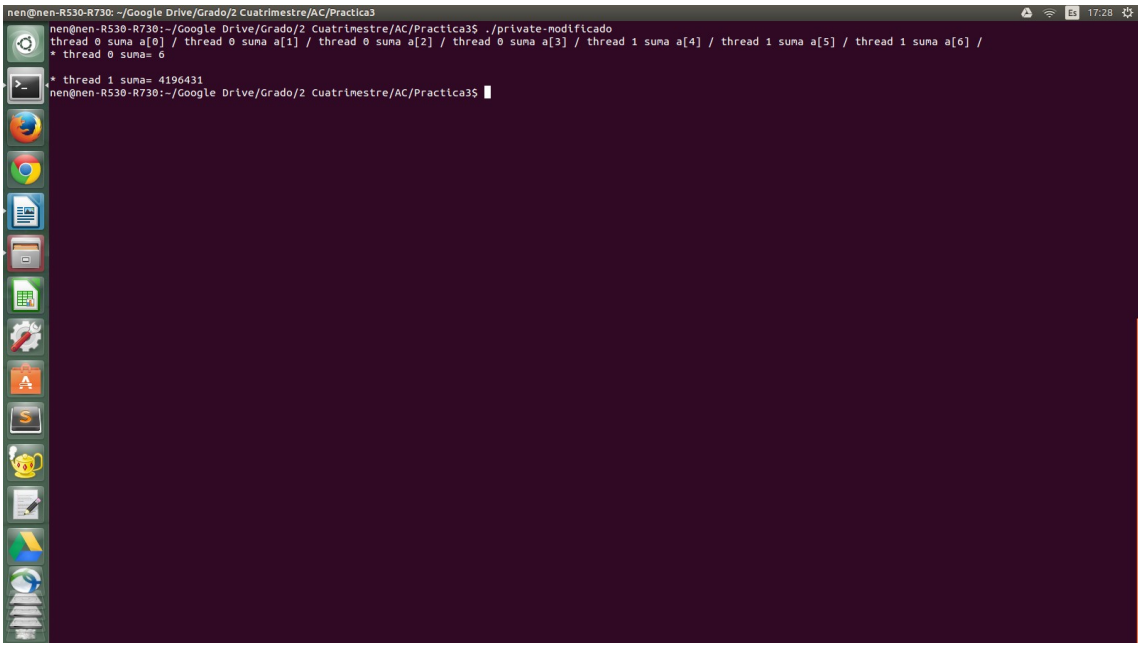
2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA: El resultado es indeterminado de todas las hebras excepto de la primera, que puede usar el valor que se le ha asignado por primera vez.

CÓDIGO FUENTE: `private-clauseModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
main()
{
            int i, n = 7;
            int a[n], suma=0;
            for (i=0; i<n; i++)
                        a[i] = i;
            #pragma omp parallel private(suma)
            {
                        //suma=0;
                        #pragma omp for
                        for (i=0; i<n; i++)
                        {
                                    suma = suma + a[i];
                                    printf("thread %d suma a[%d] / ",
omp_get_thread_num(), i);
                        }
                        printf("\n* thread %d suma= %d\n",
omp_get_thread_num(), suma);
            }
}
```

CAPTURAS DE PANTALLA:



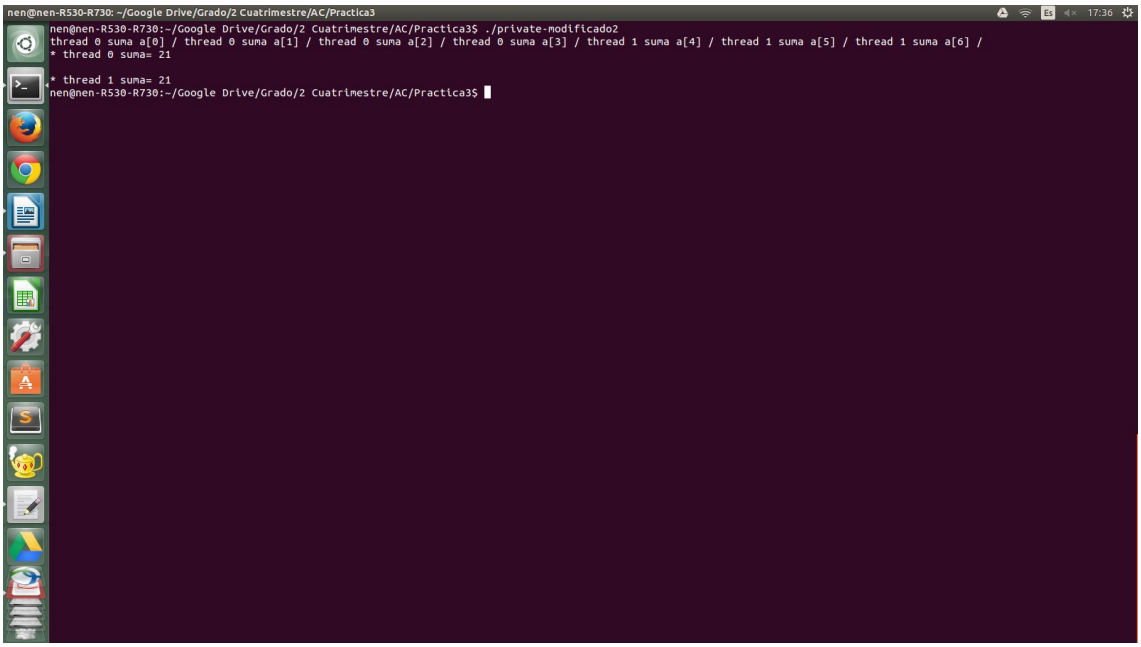
3. ¿Qué ocurre si en `private-clause.c` se elimina la cláusula `private(suma)`? ¿A qué cree que es debido?

RESPUESTA: Al eleminar la clausula private la variable suma se comparte entre las hebras y ambas hebras hacen la misma suma total de las iteraciones totales.

CÓDIGO FUENTE: `private-clauseModificado3.c`

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
main()
{
              int i, n = 7;
              int a[n], suma=0;
              for (i=0; i<n; i++)
                            a[i] = i;
              #pragma omp parallel
              {
                            //suma=0;
                            #pragma omp for
                            for (i=0; i<n; i++)
                            {
                                          suma = suma + a[i];
                                          printf("thread %d suma a[%d] / ",
omp_get_thread_num(), i);
                            }
                            printf("\n* thread %d suma= %d\n",
omp_get_thread_num(), suma);
              }
}
```

CAPTURAS DE PANTALLA:



4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región `parallel`. ¿El código imprime siempre 6 fuera de la región `parallel`? Razone su respuesta.

RESPUESTA: El código imprimirá el último elemento a procesar del bucle que se le asigne a la última hebra, en este caso se le ha asignado el elemento a[6] por lo que imprimirá un 6, pero si controlaramos la asignación de los datos en el bucle for imprimirá un numero distinto.

CAPTURAS DE PANTALLA:

5. ¿Qué ocurre si en `copyprivate-clause.c` se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?

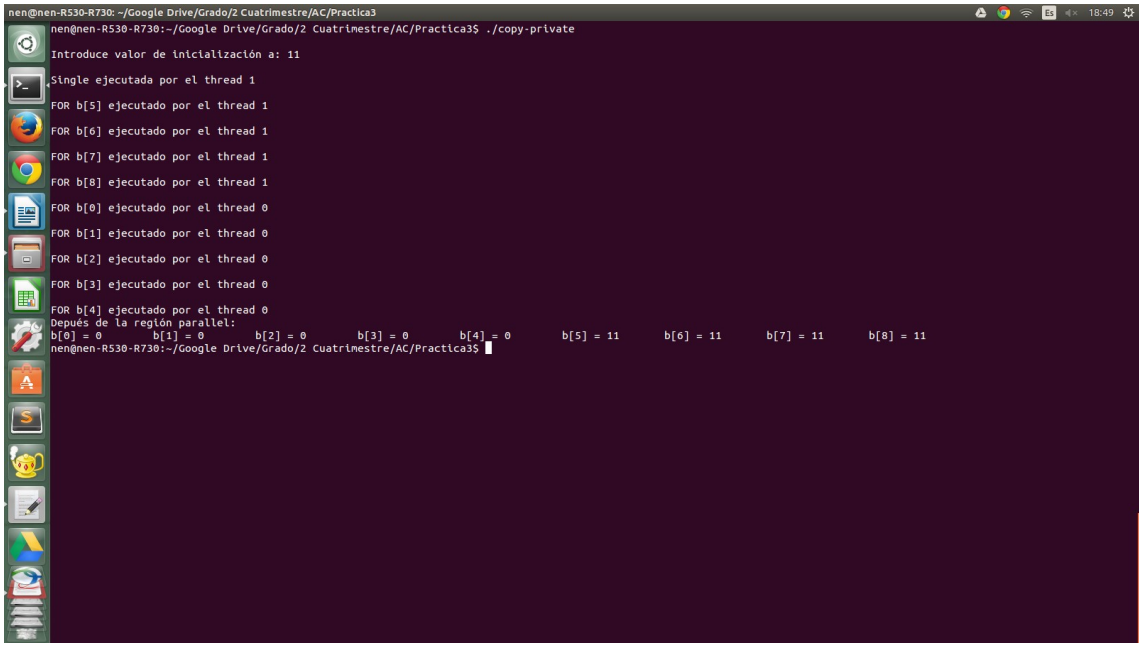
RESPUESTA: el valor de "a" unicamente lo tendrá la hebra que haya pedido como input con el scanf el valor de la variable. Las demás pueden tener un valor indeterminado.

CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
#include <stdio.h>
#include <omp.h>
main() {

              int n = 9, i, b[n];
              for (i=0; i<n; i++)
              b[i] = -1;
              #pragma omp parallel
              {
                            int a;
                            #pragma omp single
                            {
                                          printf("\nIntroduce valor de
inicialización a: ");
                                          scanf("%d", &a );
                                          printf("\nSingle ejecutada por el
thread %d\n", omp_get_thread_num());
                            }
                            #pragma omp for
                            for (i=0; i<n; i++){
                                          b[i] = a;
                                          printf("\nFOR b[%d] ejecutado por el
```

```
thread %d\n",i, omp_get_thread_num());
                        }
            }
            printf("Depués de la región parallel:\n");
            for (i=0; i<n; i++)
                        printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
}
```

CAPTURAS DE PANTALLA:



6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado

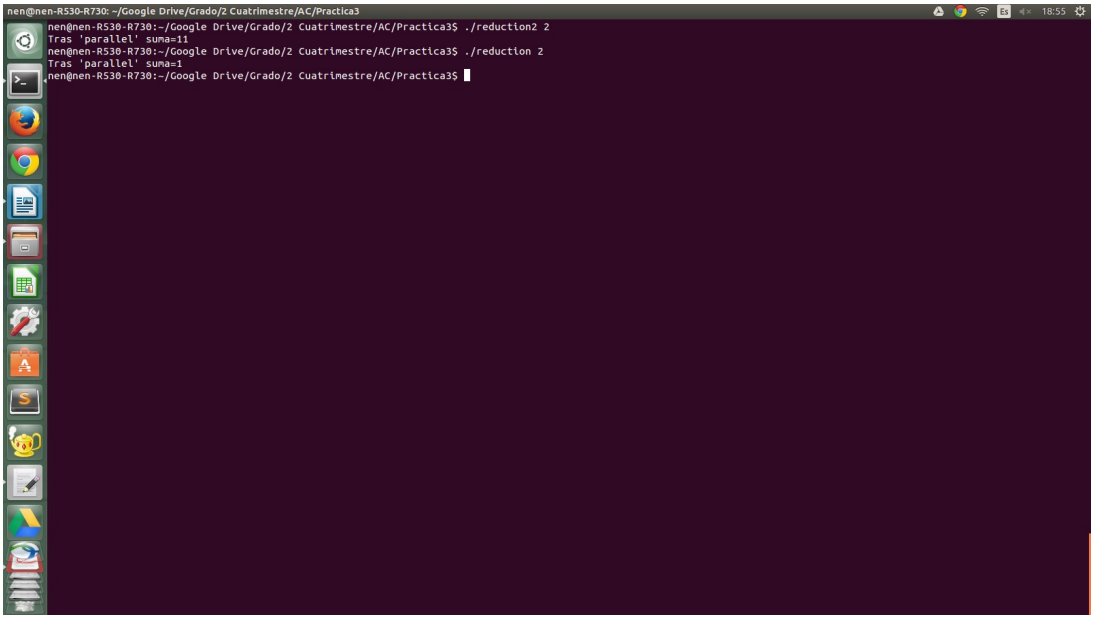
RESPUESTA: Dependiendo de las iteraciones elegidas, sumará a partir del valor inicial 10 hasta el numero de iteraciones desde i=0 hasta numero de iteraciones. Por lo tanto comparten la variable “suma” y ambas hebras NO empiezan con suma=10 sino con el resultado de la anterior y van acumulando sobre ese resultado.

CÓDIGO FUENTE: `reduction-clauseModificado.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=20, a[n],suma=10;
            if(argc < 2) {
                        fprintf(stderr,"Falta iteraciones\n");
```

```
                        exit(-1);
                }
                n = atoi(argv[1]);
                if (n>20) {n=20; printf("n=%d",n);}
                for (i=0; i<n; i++) a[i] = i;
                #pragma omp parallel for reduction(+:suma)
                        for (i=0; i<n; i++) suma += a[i];

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

CAPTURAS DE PANTALLA:



7. En el ejemplo `reduction-clause.c`, elimine `reduction(+:suma)` de `#pragma omp parallel for reduction(+:suma)` y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector a en paralelo.

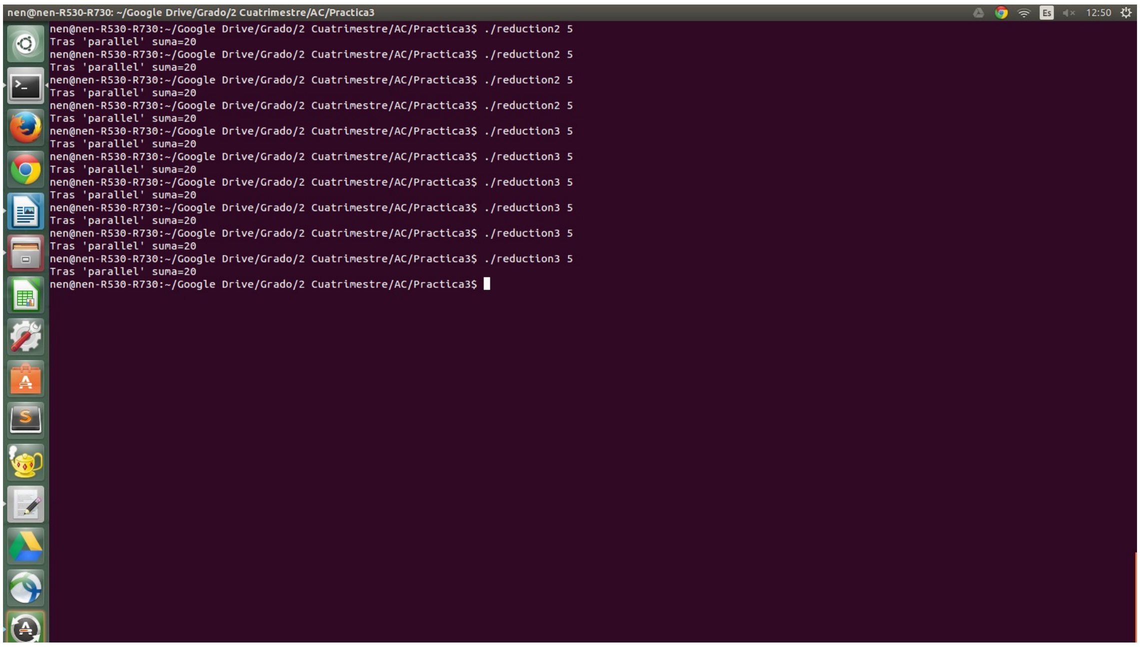
RESPUESTA: La suma sale errónea porque se produce una condición de carrera al quitar el reduction, para evitar esa condición de carrera usamos critical una variable local para que cada thread vaya acumulando su suma.

CÓDIGO FUENTE: `reduction-clauseModificado7.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=20, a[n],sumalocal,suma=10;
                if(argc < 2) {
                        fprintf(stderr,"Falta iteraciones\n");
                        exit(-1);
                }
                n = atoi(argv[1]);
                if (n>20) {n=20; printf("n=%d",n);}
                for (i=0; i<n; i++) a[i] = i;
                #pragma omp parallel private(sumalocal)
                {
                sumalocal=0;
```

```
                        #pragma omp for
                                    for (i=0; i<n; i++)

                                                sumalocal += a[i];

            #pragma omp critical
                        suma = suma+sumalocal;

            }

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

CAPTURAS DE PANTALLA:



8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1:

$$v2 = M \bullet v1;\ \ v2(i) = \sum_{k=0}^{N-1} M(i,k) \bullet v(k),\ \ i = 0,\ldots N-1$$

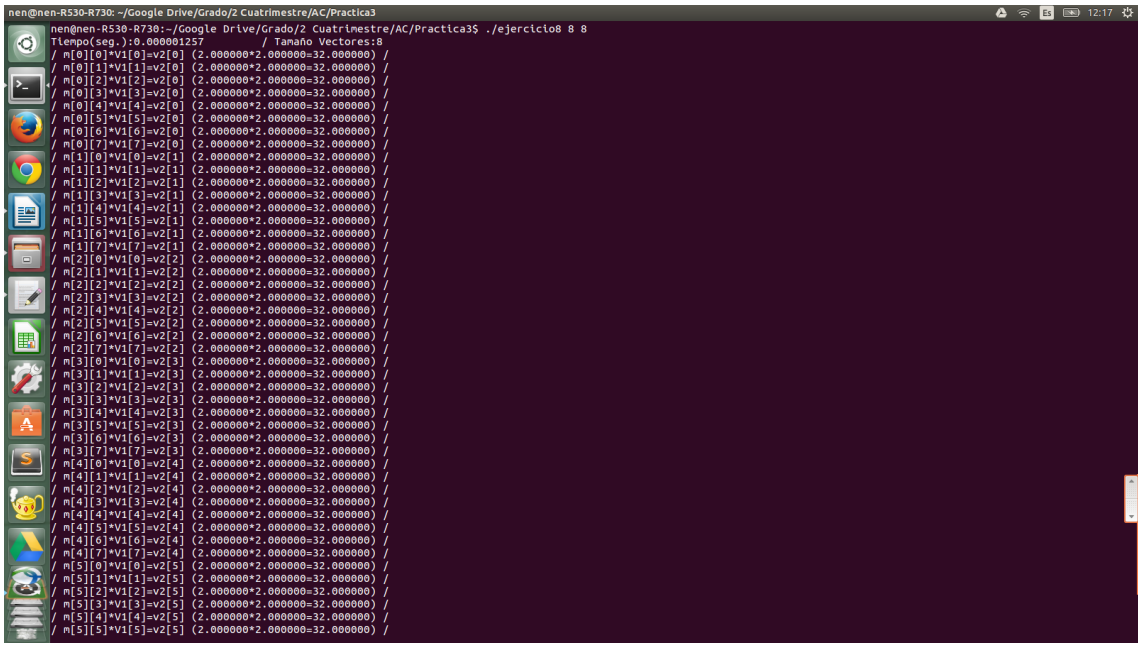
NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código secuencial que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

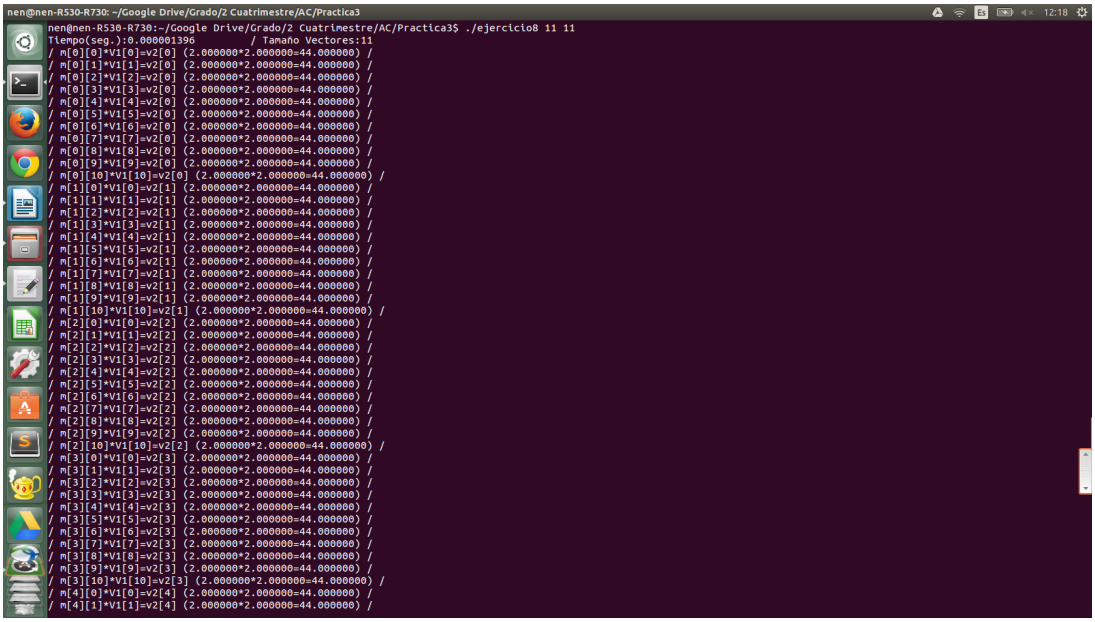
CÓDIGO FUENTE: `pmv-secuencial.c`

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

main(int argc, char **argv)
{
            if(argc < 3) {
                        fprintf(stderr,"Falta fila y columna\n");
                        exit(-1);
            }
```

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

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

            //Inicializo la matriz
            for (i=0; i<f; i++)
                        for(k=0;k<c;++k)
                                    m[i][k]=2;

            //Calculo la multiplicacion de la matriz por el vector y obtengo
el tiempo
            clock_gettime(CLOCK_REALTIME,&cgt1);
            for (i=0; i<f; i++){
                        for(k=0;k<c;++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
            printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,f);
            for (i=0; i<f; i++){
                        for(k=0;k<c;++k){
                                     printf("/ m[%d][%d]*V1[%d]=v2[%i]
(%8.6f*%8.6f=%8.6f) /\n",i,k,k,i,m[i][k],v1[k],v2[i]);
                        }
            }
            free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2

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

}
```

CAPTURAS DE PANTALLA:





9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior usando la directiva `for` . Debe implementar dos versiones del código (consulte la lección 5/Tema 2):

    a. una primera que paralelice el bucle que recorre las filas de la matriz y

    b. una segunda que paralelice el bucle que recorre las columnas.

    Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula `reduction`**. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CÓDIGO FUENTE : `pmv-OpenMP-a.c`

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

main(int argc, char **argv)
{
             if(argc < 3) {
                          fprintf(stderr,"Falta fila y columna\n");
                          exit(-1);
             }

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

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

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

                                       for(k=0;k<c;++k)
                                                    m[i][k]=2;
             }
             //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<f; i++){
                                    for(k=0;k<c;++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
            printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,f);
            for (i=0; i<f; i++){
                        for(k=0;k<c;++k){
                                     printf("/ m[%d][%d]*V1[%d]=v2[%i]
(%8.6f*%8.6f=%8.6f) /\n",i,k,k,i,m[i][k],v1[k],v2[i]);
                        }
            }
            free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2

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

}
```

CÓDIGO FUENTE: `pmv-OpenMP-b.c`

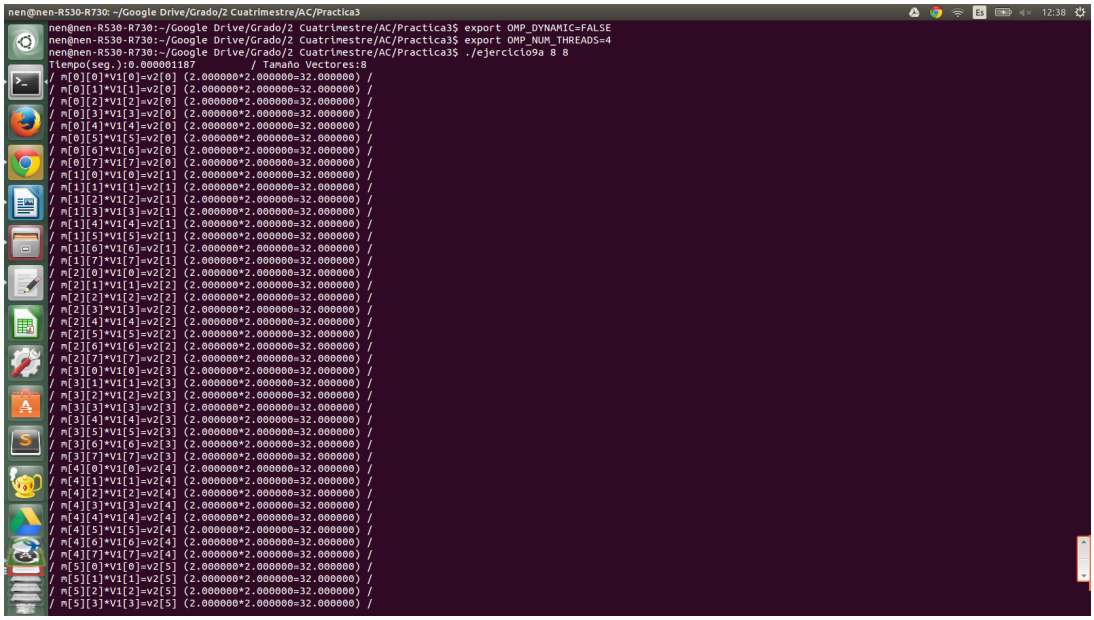
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

main(int argc, char **argv)
{
            if(argc < 3) {
                        fprintf(stderr,"Falta fila y columna\n");
                        exit(-1);
            }

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

            //Inicializo v1 y reservo el espacio para la matriz
            #pragma omp parallel for
```

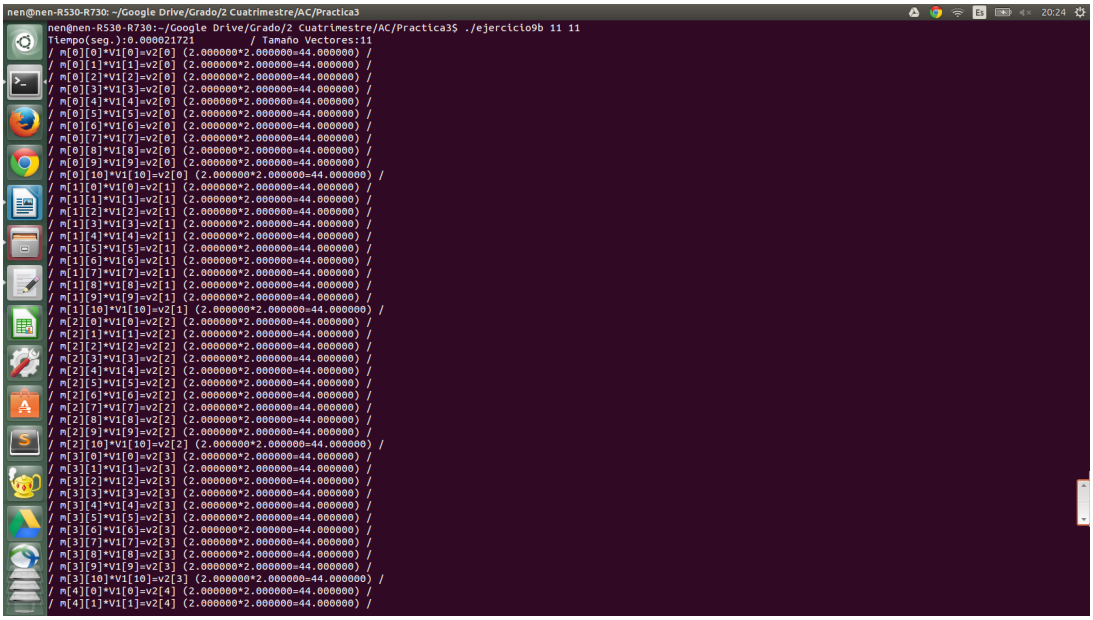
```
		for(i=0;i<c;++i){
				m[i]=(double*)malloc(c*sizeof(double));
				v1[i]=2;
		}

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

						for(k=0;k<c;++k)
								m[i][k]=2;
		}
		//Calculo la multiplicacion de la matriz por el vector y obtengo
el tiempo
		for(i=0;i<c;++i)
				v2[i]=0;


		#pragma omp parallel private(i)
		{
				#pragma omp single
				cgt1 = omp_get_wtime();

				for (i=0; i<f; i++){
						#pragma omp single
						sumalocal=0;
						#pragma omp for
						for(k=0;k<c;++k){
								#pragma omp atomic
								sumalocal+=m[i]
[k]*v1[k];
						}
						#pragma omp single
						{
								v2[i]=sumalocal;
								sumalocal=0;
						}
				}
				#pragma omp single
				cgt2 = omp_get_wtime();

		}
		ncgt=(cgt2-cgt1)/*/(1.e+9)*/;

		//Imprimo los resultados
		printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,f);
		for (i=0; i<f; i++){
				for(k=0;k<c;++k){
					 printf("/ m[%d][%d]*V1[%d]=v2[%i]
(%8.6f*%8.6f=%8.6f) /\n",i,k,k,i,m[i][k],v1[k],v2[i]);
				}
		}
		free(v1); // libera el espacio reservado para v1
	free(v2); // libera el espacio reservado para v2

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

```
}
```

RESPUESTA: En el 9A no he tenido ningún error de compilación ni en tiempo de ejecución.
El 9b en tiempo de ejecución no da errores pero muestra un resultado incorrecto. He tenido que utilizar varios single y varias funciones atomicas y single.

CAPTURAS DE PANTALLA:





10. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula `reduction`. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

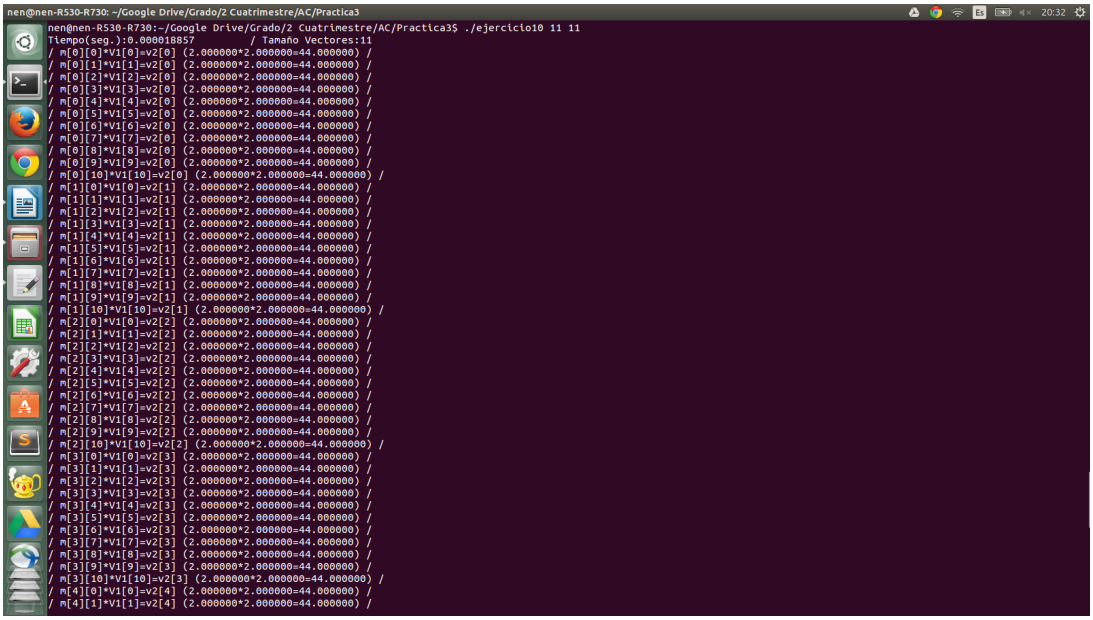
CÓDIGO FUENTE: `pmv-OpenmMP-reduction.c`

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

main(int argc, char **argv)
{
		if(argc < 3) {
				fprintf(stderr,"Falta fila y columna\n");
				exit(-1);
		}

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

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

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

						for(k=0;k<c;++k)
								m[i][k]=2;
		}
		//Calculo la multiplicacion de la matriz por el vector y obtengo
el tiempo
		for(i=0;i<c;++i)
				v2[i]=0;


		#pragma omp parallel private(i)
		{
				#pragma omp single
				cgt1 = omp_get_wtime();

				for (i=0; i<f; i++){
						#pragma omp single
						sumalocal=0;
```

```
                                        #pragma omp for reduction(+:sumalocal)
                                        for(k=0;k<c;++k){
                                                sumalocal+=m[i]
[k]*v1[k];
                                        }
                                        #pragma omp single
                                        {
                                                v2[i]=sumalocal;
                                                sumalocal=0;
                                        }
                                }
                                #pragma omp single
                                cgt2 = omp_get_wtime();

                }
                ncgt=(cgt2-cgt1)/*/(1.e+9)*/;

                //Imprimo los resultados
                printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,f);
                for (i=0; i<f; i++){
                                for(k=0;k<c;++k){
                                        printf("/ m[%d][%d]*V1[%d]=v2[%i]
(%8.6f*%8.6f=%8.6f) /\n",i,k,k,i,m[i][k],v1[k],v2[i]);
                                }
                }
                free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2

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

}
```

RESPUESTA: No he tenido problemas

CAPTURAS DE PANTALLA:



11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC del aula de prácticas de los tres códigos implementados en los ejercicios anteriores para tres tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar –O2 al compilar.

TABLA Y GRÁFICA (por *ejemplo* para 1-4 threads PC aula, y para 1-12 threads en atcgrid, tamaños-N-: 1.000, 10.000, 100.000):

COMENTARIOS SOBRE LOS RESULTADOS: A falta de tiempo he incluido los resultados de mi PC y los de el ATGRID porque no he tenido tiempo para añadir más tablas y crear las graficas

ATCGRID TABLAS

Threads 12

Threads	Tiempo(seg.)	Tamaño vectores
12	0.000488560	1000
12	0.001707022	2000
12	0.004385375	3000
12	0.007284288	4000
12	0.013763162	5000

Threads 11

Threads	Tiempo(seg.)	Tamaño vectores
11	0.000417329	1000
11	0.001803975	2000
11	0.003677014	3000
11	0.007051732	4000
11	0.010192787	5000

10

Threads	Tiempo(seg.)	Tamaño vectores
10	0.000313671	1000
10	0.001547152	2000
10	0.004006497	3000
10	0.005402170	4000
10	0.007930457	5000

9

Threads	Tiempo(seg.)	Tamaño vectores
9	0.000264743	1000
9	0.001914393	2000
9	0.004147696	3000
9	0.005888828	4000
9	0.008632730	5000

8

Threads	Tiempo(seg.)	Tamaño vectores
8	0.000306497	1000
8	0.003027951	2000
8	0.004066422	3000
8	0.005658224	4000
8	0.008624993	5000