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

Ejercicios basados en los ejemplos del seminario práctico

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:
default(none) hace que no quede claro el ámbito de las variables dentro de la directiva parallel luego da error al no saber si considerarla privada o compartida.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

shared-clause.c

```
1  #include <stdio.h>
2  #ifdef _OPENMP
3      #include <omp.h>
4  #endif
5  int main(){
6      int i, n = 7;
7      int a[n];
8      for (i=0; i<n; i++)
9          a[i] = i+1;
10
11     #pragma omp parallel for shared(a) default(none)
12     for (i=0; i<n; i++) a[i] += i;
13
14     printf("Después de parallel for:\n");
15
16     for (i=0; i<n; i++)
17         printf("a[%d] = %d\n",i,a[i]);
18 }
19
```

```
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º
 Semestre/AC/SEMINARIOS/Seminario_2/ejer1] 2018-04-06 viernes
$cd gcc -fopenmp -O2 shared-clause.c -o shared-clause
shared-clause.c: In function ‘main’:
shared-clause.c:11:10: error: ‘n’ not specified in enclosing ‘parallel’
  #pragma omp parallel for shared(a) default(none)
          ^~~
shared-clause.c:11:10: error: enclosing ‘parallel’
```

shared-clause.c

```c
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#endif
int 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? (inicialice suma a un valor distinto de 0 dentro y fuera de `parallel`) Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA:
Si se inicializa fuera de la directiva *parallel* la variable adquiere un valor basura o indeterminado.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

private-clause.c

```c
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(){
    int i, n = 7;
    int a[n], suma;

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

    #pragma omp parallel private(suma)
    {
        suma=7;
        #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", omp_get_thread_num(), suma);
    }
    printf("\n");
}
```

```
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2
º Semestre/AC/SEMINARIOS/Seminario_2/ejer2] 2018-04-07 sábado
$gcc -O2 -fopenmp private-clause.c -o private-clause
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2
º Semestre/AC/SEMINARIOS/Seminario_2/ejer2] 2018-04-07 sábado
$./private-clause
thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma a[4] / thread 2 suma a
[5] / thread 1 suma a[2] / thread 1 suma a[3] / thread 3 suma a[6] /
* thread 0 suma= 8
* thread 1 suma= 12
* thread 2 suma= 16
* thread 3 suma= 13
```

CAPTURAS DE PANTALLA:

private-clause.c

```c
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(){
    int i, n = 7;
    int a[n], suma=7;

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

    #pragma omp parallel private(suma)
    {
        //suma=7;
        #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", omp_get_thread_num(), suma);
    }
    printf("\n");
}
```

```
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/
2º Semestre/AC/SEMINARIOS/Seminario_2/ejer2] 2018-04-07 sábado
$gcc -O2 -fopenmp private-clause.c -o private-clause
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/
2º Semestre/AC/SEMINARIOS/Seminario_2/ejer2] 2018-04-07 sábado
$./private-clause
thread 3 suma a[6] / thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma
a[4] / thread 2 suma a[5] / thread 1 suma a[2] / thread 1 suma a[3] /
* thread 2 suma= -1376058535
* thread 0 suma= 59848529
* thread 3 suma= -1376058538
* thread 1 suma= -1376058539
```

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

RESPUESTA:

Los resultados de todas las sumas son el mismo porque la variable pasa a ser compartida. Además al ser compartida y no presentar una sección de exclusión mutua se pueden producir condiciones de carrera y provocar que el resultado no sea siempre el mismo. También importa si inicializamos *suma* dentro o fuera del *parallel*.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

private-clause.c

```c
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(){
    int i, n = 7;
    int a[n], suma;

    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", omp_get_thread_num(), suma);
    }
    printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer3] 2018-04-07 sábado
$./private-clause
thread 1 suma a[2] / thread 1 suma a[3] / thread 3 suma a[6] / thread 0 suma
 a[0] / thread 0 suma a[1] / thread 2 suma a[4] / thread 2 suma a[5] /
* thread 3 suma= 13
* thread 2 suma= 13
* thread 1 suma= 13
* thread 0 suma= 13
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º
/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer3] 2018-04-07 sábado
$./private-clause
thread 0 suma a[0] / thread 0 suma a[1] / thread 1 suma a[2] / thread 1 suma
 a[3] / thread 2 suma a[4] / thread 2 suma a[5] / thread 3 suma a[6] /
* thread 0 suma= 15
* thread 2 suma= 15
* thread 1 suma= 15
* thread 3 suma= 15
```

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:

Imprime siempre el mismo resultado a no ser que variemos el número de hebras. Con *OMP_NUMTHREADS* fijado a 4, la última hebra siempre asignará el valor 6 a la variable *suma* (la directiva *lastprivate* mantiene en la variable *suma* que se encuentra fuera del parallel el último valor asignado a la misma por la hebra que realice la última iteracion del bucle) por como reparten las iteraciones del bucle las directivas *parallel for*. En cambio si variamos el número de hebras también variará el número mostrado fuera de la directiva *parallel*.

CAPTURAS DE PANTALLA:

```
 thread 0 suma a[1] suma=1
 thread 2 suma a[4] suma=4
 thread 2 suma a[5] suma=9
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 3 suma a[6] suma=6

Fuera de la construcción parallel suma=6
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/G
II/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer4] 2018-04-07 sábado
$export OMP_NUM_THREADS=3
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/G
II/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer4] 2018-04-07 sábado
$./firstlastprivate
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 0 suma a[2] suma=3
 thread 1 suma a[3] suma=3
 thread 1 suma a[4] suma=7
 thread 2 suma a[5] suma=5
 thread 2 suma a[6] suma=11

Fuera de la construcción parallel suma=11
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/G
II/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer4] 2018-04-07 sábado
$export OMP_NUM_THREADS=2
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/G
II/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer4] 2018-04-07 sábado
$./firstlastprivate
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 0 suma a[2] suma=3
 thread 0 suma a[3] suma=6
 thread 1 suma a[4] suma=4
 thread 1 suma a[5] suma=9
 thread 1 suma a[6] suma=15

Fuera de la construcción parallel suma=15
```

5. ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?

RESPUESTA:

Al quitar *copyprivate* la variable *a* solo obtiene el valor leído por pantalla en la propia hebra que ha leído el valor. Las demás hebras obtienen un valor que parece que OMP asigna por defecto (0). Para comprobar esto se ha modificado el código para mostrar que la hebra que lee el valor de *a* es la única que asigna ese valor a las componentes del vector.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

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

int main() {
    int n = 9, i, b[n];

    for (i=0; i<n; i++)
        b[i] = -1;

    #pragma omp parallel
    {
        int a;
        #pragma omp single //copyprivate(a)
        {
            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("\nasignación indice: %d\t hebra: %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");
}
```

```
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer5] 2018-04-13 viernes
$./cgcc -fopenmp -O2 copyprivate-clause.c -o copyprivate-clause
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer5] 2018-04-13 viernes
$gcc./copyprivate-clause

Introduce valor de inicialización a: 5

Single ejecutada por el thread 0

asignación indice: 0     hebra: 0

asignación indice: 1     hebra: 0

asignación indice: 2     hebra: 0

asignación indice: 3     hebra: 1

asignación indice: 4     hebra: 1

asignación indice: 7     hebra: 3

asignación indice: 8     hebra: 3

asignación indice: 5     hebra: 2

asignación indice: 6     hebra: 2
Depués de la región parallel:
b[0] = 5        b[1] = 5        b[2] = 5        b[3] = 0        b[4] = 0        b[5] = 0        b[6] = 0        b[7] = 0        b[8] = 0
```

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

RESPUESTA:
Si con *suma=0* obtenemos un resultado X, con *suma=10* obtenemos X+10.
La directiva *reduction* inicializa la variable acumulativa a 0 pero solo dentro del dominio de la directiva. Tras realizar la suma de todos los valores se añade el valor calculado al valor que tenía la variable antes de la directiva *reduction*.

Se ha modificado el código para mostrar lo cómo funciona (más o menos) la directiva *reduction*.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(int argc, char **argv)
{
    int i, n=20, a[n],suma=0;

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

    n = atoi(argv[1]);
    if (n>20)
    {
        n=20;
        printf("n=%d\n",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("suma: %d i: %d hebra: %d\n", suma, i, omp_get_thread_num());
        }

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

CAPTURAS DE PANTALLA:

suma=0

```
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer6] 2018-04-13 viernes
$gcc./reduction-clause 30
n=20
suma: 5 i: 5 hebra: 1
suma: 11 i: 6 hebra: 1
suma: 18 i: 7 hebra: 1
suma: 26 i: 8 hebra: 1
suma: 35 i: 9 hebra: 1
suma: 10 i: 10 hebra: 2
suma: 21 i: 11 hebra: 2
suma: 33 i: 12 hebra: 2
suma: 46 i: 13 hebra: 2
suma: 60 i: 14 hebra: 2
suma: 15 i: 15 hebra: 3
suma: 31 i: 16 hebra: 3
suma: 48 i: 17 hebra: 3
suma: 66 i: 18 hebra: 3
suma: 85 i: 19 hebra: 3
suma: 0 i: 0 hebra: 0
suma: 1 i: 1 hebra: 0
suma: 3 i: 2 hebra: 0
suma: 6 i: 3 hebra: 0
suma: 10 i: 4 hebra: 0
Tras 'parallel' suma=190
```

suma=20

```
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer6] 2018-04-13 viernes
$gcc./reduction-clause 30
n=20
suma: 5 i: 5 hebra: 1
suma: 11 i: 6 hebra: 1
suma: 18 i: 7 hebra: 1
suma: 26 i: 8 hebra: 1
suma: 35 i: 9 hebra: 1
suma: 15 i: 15 hebra: 3
suma: 31 i: 16 hebra: 3
suma: 48 i: 17 hebra: 3
suma: 66 i: 18 hebra: 3
suma: 85 i: 19 hebra: 3
suma: 0 i: 0 hebra: 0
suma: 1 i: 1 hebra: 0
suma: 3 i: 2 hebra: 0
suma: 6 i: 3 hebra: 0
suma: 10 i: 4 hebra: 0
suma: 10 i: 10 hebra: 2
suma: 21 i: 11 hebra: 2
suma: 33 i: 12 hebra: 2
suma: 46 i: 13 hebra: 2
suma: 60 i: 14 hebra: 2
Tras 'parallel' suma=210
```

7. En el ejemplo `reduction-clause.c`, elimine `reduction()` 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 sin usar directivas de trabajo compartido .

RESPUESTA:

He optado por la solución explicada en la transparencia 8 del seminario a la hora de realizar la acumulacion en la variable *suma*, es decir, he utilizado la directiva *critical* para crear una sección crítica y así evitar que haya condiciones de carrera y dos hebras accedan a la variable a la vez.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif
int main(int argc, char **argv)
{
    int i, n=20, a[n],suma=0;
    if(argc < 2)
    {
        fprintf(stderr,"Falta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]);
    if (n>20)
    {
        n=20;
        printf("n=%d\n",n);
    }
    for (i=0; i<n; i++)
        a[i] = i;
    #pragma omp parallel
    {
        int sumalocal = 0;
        #pragma omp for
        for (i=0; i<n; i++)
        {
            sumalocal += a[i];
            printf(" thread %d suma de a[%d]=%d sumalocal=%d \n", omp_get_thread_num(),i,a[i],sumalocal);
        }
        #pragma omp atomic
        suma += sumalocal;
    }
    printf("Tras 'parallel' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

```
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer7] 2018-04-13 viernes
$./reduction-clause 10
suma: 8 i: 8 hebra: 3
suma: 20 i: 9 hebra: 3
suma: 11 i: 0 hebra: 0
suma: 21 i: 1 hebra: 0
suma: 23 i: 2 hebra: 0
suma: 11 i: 3 hebra: 1
suma: 27 i: 4 hebra: 1
suma: 32 i: 5 hebra: 1
suma: 38 i: 6 hebra: 2
suma: 45 i: 7 hebra: 2
Tras 'parallel' suma=45
```

Resto de ejercicios

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1 (implemente una versión para variables globales y otra para variables dinámicas, use una de estas versiones en los siguientes ejercicios):

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

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 paralelo 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).

CAPTURA CÓDIGO FUENTE: `pmv-secuencial.c`

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

int main(int argc, char const *argv[])
{
    if (argc<2)
    {
        printf("Faltan tamaño de la matriz\n");
        exit(-1);
    }
    struct timespec cgt1,
                    cgt2;
    double          ncgt; //para tiempo de ejecución
    int             N   = atoi(argv[1]),
                    i,
                    j;

    double          **m;
    double          *v, *v_res;

        m = (double**) malloc(N*sizeof(double*));
        v = (double*) malloc(N*sizeof(double));
    v_res = (double*) malloc(N*sizeof(double));

    srand(time(NULL));
    for(i = 0 ; i < N ; ++i)
    {
        m[i] = (double*) malloc(N*sizeof(double));
        v[i] = rand();
        v_res[i] = 0;
        for(j=0 ; j < N ; ++j)
        {
            m[i][j] = rand();
        }
    }

```

```
38      double tmp;
39      clock_gettime(CLOCK_REALTIME,&cgt1);
40
41      for(i=0; i<N; i++){
42          tmp=0;
43          for(j=0; j<N; j++){
44              tmp += m[i][j]*v[j];
45          }
46          v_res[i] = tmp;
47      }
48
49      clock_gettime(CLOCK_REALTIME,&cgt2);
50      ncgt =  (double) (cgt2.tv_sec-cgt1.tv_sec)+
51              (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
52
53      if(N <= 15)
54      {
55          for(i = 0 ; i < N ; ++i)
56          {
57              printf("v[%d]: %f\n", i, v_res[i]);
58          }
59      }
60      else
61      {
62          printf("v[0]: %f\n", v_res[0]);
63          printf("v[%d]: %f\n", N-1, v_res[N-1]);
64      }
65
66      for(i = 0 ; i < N ; ++i)
67          free(m[i]);
68      free(m);
69
70      printf("tiempo: %11.9f\n", ncgt);
71
72      return 0;
73  }
```

CAPTURAS DE PANTALLA:

```
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer8] 2018-04-13 viernes
$gcc./productoMxV 8
v[0]: 7511777565097878528.000000
v[1]: 9637029009821458432.000000
v[2]: 7884431746710715392.000000
v[3]: 8498491404854037504.000000
v[4]: 10570734743346941952.000000
v[5]: 9677014628886810624.000000
v[6]: 7358404080365035520.000000
v[7]: 6373174270428522496.000000
tiempo: 0.000001155
[JoseJavierAlonsoRamos jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/SEMINARIOS/Seminario_2/ejer8] 2018-04-13 viernes
$./productoMxV 8000
v[0]: 9132892690919976861696.000000
v[7999]: 9197659055249662410752.000000
tiempo: 0.103283471
```

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).

CAPTURA CÓDIGO FUENTE : `pmv-OpenMP-a.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(int argc, char const *argv[])
{
    if (argc<2)
    {
        printf("Faltan tamaño de la matriz\n");
        exit(-1);
    }
    struct timespec cgt1,
                    cgt2;
    double          ncgt; //para tiempo de ejecución
    int             N   = atoi(argv[1]),
                    i,
                    j;

    double          **m;
    double          *v, *v_res;

        m = (double**) malloc(N*sizeof(double*));
        v = (double*) malloc(N*sizeof(double));
    v_res = (double*) malloc(N*sizeof(double));

    srand(time(NULL));
    #pragma omp parallel for //shared(N, m, v, v_res)
    for(i = 0 ; i < N ; ++i)
    {
        m[i] = (double*) malloc(N*sizeof(double));
        v[i] = 2;//rand();
        v_res[i] = 0;

        for(j=0 ; j < N ; ++j)
        {
            m[i][j] = 2;//rand();
        }
```

```
43      }
44
45      double tmp;
46      clock_gettime(CLOCK_REALTIME,&cgt1);
47
48      #pragma omp parallel for
49      for(i=0; i<N; i++){
50          tmp=0;
51          for(j=0; j<N; j++){
52              tmp += m[i][j]*v[j];
53          }
54          v_res[i] = tmp;
55      }
56
57      clock_gettime(CLOCK_REALTIME,&cgt2);
58      ncgt =  (double) (cgt2.tv_sec-cgt1.tv_sec)+
59              (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
60
61      if(N <= 15)
62      {
63          for(i = 0 ; i < N ; ++i)
64          {
65              printf("v[%d]: %f\n", i, v_res[i]);
66          }
67      }
68      else
69      {
70          printf("v[0]: %f\n", v_res[0]);
71          printf("v[%d]: %f\n", N-1, v_res[N-1]);
72      }
73
74      for(i = 0 ; i < N ; ++i)
75          free(m[i]);
76      free(m);
77
78      printf("tiempo: %11.9f\n", ncgt);
79
80      return 0;
81  }
```

CAPTURA CÓDIGO FUENTE: `pmv-OpenMP-b.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(int argc, char const *argv[])
{
    if (argc<2)
    {
        printf("Faltan tamaño de la matriz\n");
        exit(-1);
    }
    struct timespec cgt1,
                    cgt2;
    double          ncgt; //para tiempo de ejecución
    int             N   = atoi(argv[1]),
                    i,
                    j;
    double          **m;
    double          *v, *v_res;

        m = (double**) malloc(N*sizeof(double*));
        v = (double*) malloc(N*sizeof(double));
    v_res = (double*) malloc(N*sizeof(double));

    srand(time(NULL));
    #pragma omp parallel for
    for(i = 0 ; i < N ; ++i)
    {
        m[i] = (double*) malloc(N*sizeof(double));
        v[i] = 2;//rand();
        v_res[i] = 0;

        for(j=0 ; j < N ; ++j)
        {
            m[i][j] = 2;//rand();
        }
    }
```

```
43
44      clock_gettime(CLOCK_REALTIME,&cgt1);
45      for(i=0; i<N; i++){
46          #pragma omp parallel
47          {
48              int sumalocal = 0;
49              #pragma omp for
50              for(j=0; j<N; j++){
51                  sumalocal += m[i][j]*v[j];
52              }
53              #pragma omp atomic
54              v_res[i] += sumalocal;
55          }
56      }
57
58      clock_gettime(CLOCK_REALTIME,&cgt2);
59      ncgt =  (double) (cgt2.tv_sec-cgt1.tv_sec)+
60              (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
61
62      if(N <= 15)
63      {
64          for(i = 0 ; i < N ; ++i)
65          {
66              printf("v[%d]: %f\n", i, v_res[i]);
67          }
68      }
69      else
70      {
71          printf("v[0]: %f\n", v_res[0]);
72          printf("v[%d]: %f\n", N-1, v_res[N-1]);
73      }
74
75      for(i = 0 ; i < N ; ++i)
76          free(m[i]);
77      free(m);
78
79      printf("tiempo: %11.9f\n", ncgt);
80
81      return 0;
82  }
```

RESPUESTA:
He obtenido eror de ejecución en el apartado b porque en la directiva atomic me faltaba la palabra omp.

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).

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

```
for(i=0; i<N; i++){
    #pragma omp parallel for reduction(+:v_res[i])
    for(j=0; j<N; j++){
        v_res[i] += m[i][j]*v[j];
    }
}
```

    RESPUESTA:
    Obtenia error de compilación porque tenía una llave '}' sin pareja.

    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 su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar –O2 al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

    CAPTURAS DE PANTALLA (que justifique el código elegido):

    TABLA Y GRÁFICA (por *ejemplo* para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 30000 y 100000, y otro entre 5000 y 30000):

    COMENTARIOS SOBRE LOS RESULTADOS:

```
jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/SEMINARIOS/
minario_2/ejer11$ ./../ejer9/productoMxV-a 24000
v[0]: 96000.000000
v[23999]: 96000.000000
tiempo: 95.518771449
jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/SEMINARIOS/
minario_2/ejer11$ ./../ejer9/productoMxV-a 25000
Terminado (killed)
```

    Los programas abortan para tamaños superiores a 24000 por lo que no puedo hacer un estudio con la suficiente veracidad.