Grai2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos):
Grupo de prácticas y profesor de prácticas:

Ejercicios basados en los ejemplos del seminario práctico

1. **(a)** Añadir la cláusula `default(none)` a la directiva `parallel` del ejemplo del seminario `shared-clause.c`? ¿Qué ocurre? ¿A qué se debe? **(b)** Resolver el problema generado sin eliminar `default(none)`. Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla que muestren lo que ocurre)

RESPUESTA:
Lo que hacemos al añadir default(none) es cambiar el comportamiento por defecto de todas las variables. Por loq ue debemos especificar el ámbito de todas las variables explicitamente.
El problema de compilacion se basa en que el ambito de n no esta especificado, para arreglarlo debemos especificarlo en la clausula shared → shared(a,n)

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.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) 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]);
}
```

```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:

```
┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp shared-clause.c -o shared-clause
shared-clause.c: In function 'main':
shared-clause.c:14:13: error: 'n' not specified in enclosing 'parallel'
   14 |     #pragma omp parallel for shared(a) default (none)
      |             ^~
shared-clause.c:14:13: note: enclosing 'parallel'
```

```
┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp shared-clauseModificado.c -o shared-clause

┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ ./shared-clause
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
```

2. **(a)** Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel`. Inicializar `suma` dentro del `parallel` a un valor distinto de 0. Ejecutar varias veces el código ¿Qué imprime el código fuera del `parallel`? (mostrar lo que ocurre con una captura de pantalla) Razonar respuesta. **(b)** Modificar el código del apartado (a) para que se inicialice `suma` fuera del `parallel` en lugar de dentro ¿Qué ocurre? Comparar todo lo que imprime el código ahora con la salida en (a) (mostrar la salida con una captura de pantalla) Razonar respuesta.

(a) RESPUESTA:

Lo que he hecho fue desplazar la impresion de la suma fuera del parallel e inicializar la suma a 9, como podemos ver el resultado es siemrpe 0 no 9, es decir el valor queda indefinicdo el valor de cada hebra tmb es indefinido, eso es , si se inicia la variable fuera de la construccion del parallel el valor de entrada queda indefinido, de manera que se prodce error

Este ejercicio será para repasar las precauciones a la hora de tener en cuente para esta clausula. EL valor de salida esta indefinicdo autnque ka variable este declarada fuera de la funcion

CAPTURAS DE PANTALLA:

```
┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp private-clauseModificado.c -o private-clauseModificado

┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ ./private-clauseModificado
thread 2 suma a[2] / thread 1 suma a[1] / thread 3 suma a[3] / thread 5 suma a[5] / thread 0 suma a[0] / thread 6 s
uma a[6] / thread 4 suma a[4] /
* thread 0 suma= 0

┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ ./private-clauseModificado
thread 0 suma a[0] / thread 2 suma a[2] / thread 6 suma a[6] / thread 4 suma a[4] / thread 5 suma a[5] / thread 3 s
uma a[3] / thread 1 suma a[1] /
* thread 0 suma= 0

┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ ./private-clauseModificado
thread 0 suma a[0] / thread 3 suma a[3] / thread 2 suma a[2] / thread 6 suma a[6] / thread 5 suma a[5] / thread 1 s
uma a[1] / thread 4 suma a[4] /
* thread 0 suma= 0
```

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.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=0;
        suma = 9;
        #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* thread %d suma= %d",omp_get_thread_num(),suma);

    printf("\n");
}
```

(b) RESPUESTA:

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_b.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;

    suma = 9;
    #pragma omp parallel private(suma)
    {
        //suma=0;
        //suma = 9;
        #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* thread %d suma= %d",omp_get_thread_num(),suma);

    printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ ./private-clauseModificado
thread 0 suma a[0] / thread 3 suma a[3] / thread 5 suma a[5] / thread 4 suma a[4] / thread 1 suma a[1] / thread 6 s
uma a[6] / thread 2 suma a[2] /
* thread 6 suma= 1246704230
* thread 4 suma= 1246704228
* thread 2 suma= 1246704226
* thread 5 suma= 1246704229
* thread 1 suma= 1246704225
* thread 3 suma= 1246704227
* thread 0 suma= 8
* thread 7 suma= 1246704224
```

3. **(a)** Eliminar la cláusula `private(suma)` en `private-clause.c`. Ejecutar el código resultante. ¿Qué ocurre? **(b)** ¿A qué es debido?

RESPUESTA:

Si quitamos private(suma) esta variable sera accedida concurrentemente y se darán condiciones de carrera.Suma será compartida por todas las hebras y habrá condiciones de acceso. Muestra el resultado producido por la ultima hebra que modifico la variable suma.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.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;

    //suma = 9;
    #pragma omp parallel
    {
       suma=0;
        //suma = 9;
        #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* thread %d suma= %d",omp_get_thread_num(),suma);

    printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp private-clauseModificado3.c -o private-clauseModificado3

┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ ./private-clauseModificado3
thread 0 suma a[0] / thread 1 suma a[1] / thread 2 suma a[2] / thread 6 suma a[6] / thread 3 suma a[3] / thread 5 s
uma a[5] / thread 4 suma a[4] /
* thread 6 suma= 5
* thread 0 suma= 5
* thread 2 suma= 5
* thread 4 suma= 5
* thread 3 suma= 5
* thread 5 suma= 5
* thread 1 suma= 5
* thread 7 suma= 5
```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región `parallel`. **(a)** Cambiar el tamaño del vector a 10. Razonar lo que imprime el código en su PC con esta modificación. (añadir capturas de pantalla que muestren lo que ocurre). **(b)** Sin cambiar el tamaño del vector ¿podría imprimir el código otro valor? Razonar respuesta (añadir capturas de pantalla que muestren lo que ocurre).

(a) RESPUESTA:

Con firstprivate combina la accion de private y la inicializacion de las variables de la lista. Hay que inicializar suma a 0 dentro del parallel.(en la diapo, sin lastprivate, suma fuera del parallel quedaria a 0)
Con lastprivate actualiza la variable global suma con el ultimo valor que se dio en la ultima iteraciones
Dependiendo del numero de threads que le demos la suma de fuera dara un valor u otro.
Con 3 threads por ej, imprime 11 ya que lastprivate hace que suma coja la ultima suma parcial si se hiciese de forma secuencial (a[6] = 11)

CAPTURAS DE PANTALLA:

```c
/**
 * @file firstlastprivate.c
 */

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

int main() {
  //int i, n = 7;
  int i, n= 10;
  int a[n], suma=0;

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

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

  printf("\nFuera de la construcción parallel suma=%d\n",suma);
}
```

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp firstlastprivateModificado.c -o firstlastprivateModificado

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ export OMP_NUM_THREADS=5

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./firstlastprivateModificado
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 4 suma a[8] suma=8
 thread 4 suma a[9] suma=17
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 2 suma a[4] suma=4
 thread 2 suma a[5] suma=9
 thread 3 suma a[6] suma=6
 thread 3 suma a[7] suma=13

Fuera de la construcción parallel suma=17

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ export OMP_NUM_THREADS=2

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./firstlastprivateModificado
 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 0 suma a[4] suma=10
 thread 1 suma a[5] suma=5
 thread 1 suma a[6] suma=11
 thread 1 suma a[7] suma=18
 thread 1 suma a[8] suma=26
 thread 1 suma a[9] suma=35

Fuera de la construcción parallel suma=35
```

(b) RESPUESTA:

Como hemos dicho antes, podemos cambiar el valor si variamos el valor del ultimo a[i] o bien cambiando el numero de threads con el que opera el proceso ya que podemos hacer que la suma parcial de una hebra tenga varias sumas de varias componentes y se muestre una suma parcial de la ultima hebra que ejecuto el for

CAPTURAS DE PANTALLA:

```
┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp firstlastprivate.c -o firstlastprivate

┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ ./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

┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ export OMP_NUM_THREADS=3

┌──(icaro㉿kali)-[~/…/AC/Seminario2/Codigo/BP2]
└─$ ./firstlastprivate
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 0 suma a[2] suma=3
 thread 2 suma a[5] suma=5
 thread 2 suma a[6] suma=11
 thread 1 suma a[3] suma=3
 thread 1 suma a[4] suma=7

Fuera de la construcción parallel suma=11
```

5. **(a)** ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? **(b)** ¿A qué cree que es debido? (añadir una captura de pantalla que muestre lo que ocurre)

RESPUESTA:
Copyprivate se encarga de que una variable privada de un thread se copie a as variables privadas del mismo nombre del resto de threads (difusión), util para la lectura de variables, las otras variables estarian en un principio indefinidas
Si quitamos copyprivate(a) no se inicializaria el valor de “a” para el resto de hebras. El unico que tendrá inicializada la varable “a” sera aquel que el thres que ejecute el single

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)
        #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("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:

sin modificar:

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./copyprivate-clause

Introduce valor de inicialización a: 10

Single ejecutada por el thread 0
Depués de la región parallel:
b[0] = 10       b[1] = 10       b[2] = 10       b[3] = 10       b[4] = 10       b[5] = 10       b[6] = 10       b[7
] = 10  b[8] = 10
```

Modificado:

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp copyprivate-clauseModificado.c -o copyprivate-clauseModificado

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ export OMP_NUM_THREADS=5

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./copyprivate-clauseModificado

Introduce valor de inicialización a: 10

Single ejecutada por el thread 0
Depués de la región parallel:
b[0] = 10       b[1] = 10       b[2] = 0        b[3] = 0        b[4] = 0        b[5] = 0        b[6] = 0        b[7
] = 0   b[8] = 0

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./copyprivate-clauseModificado

Introduce valor de inicialización a: 5

Single ejecutada por el thread 1
Depués de la región parallel:
b[0] = 21902    b[1] = 21902    b[2] = 5        b[3] = 5        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 (añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA:

Se esta imprimiendo el mismo resultado pero sumandole 10 al resultado inicial.

La clausula reduction realiza lo mismo que atomic y critical pero mas eficiente. Reduction evita que tengamos que utilizar las secciones criticas

En este ejercicio el operador de reduccion es + y como esta en el seminario el valor de las variables locales para este operadores es 0

Cuando las hebras terminen de calcular su suma parcial correspondiente entonces de foorma segura se suma a la variable globla de suma con esto agrupamos los valores de varios threads en un unico valor

Con esto conseguimos que el valor inicial sea flexible y en vez de inicializarlo a 0 podamos inicializarlo a otros valores

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;
    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:

sin modificar

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp reduction-clause.c -o reduction-clause

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ export OMP_NUM_THREADS=3

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./reduction-clause 10
Tras 'parallel' suma=45

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./reduction-clause 20
Tras 'parallel' suma=190

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./reduction-clause 6
Tras 'parallel' suma=15
```

modificado

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp reduction-clauseModificado.c -o reduction-clauseModificado

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ export OMP_NUM_THREADS=3

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./reduction-clauseModificado 10
Tras 'parallel' suma=55

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./reduction-clauseModificado 20
Tras 'parallel' suma=200

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./reduction-clauseModificado 6
Tras 'parallel' suma=25
```

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 añadir más directivas de trabajo compartido (añada capturas de pantalla que muestren lo que ocurre).

RESPUESTA:

Como con reduction estamos aplicando lo mismo que atomic o critical, lo que tenemos que hace es poner una seccion critica para el acceso de la suma, usando atomic o critical es lo mismo

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);
    }

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

    //#pragma omp parallel for reduction(+:suma)
    #pragma omp parallel for
    for (i=0; i<n; i++) {
        #pragma omp critical
            suma += a[i];
    }

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

CAPTURAS DE PANTALLA:

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ gcc -O2 -fopenmp reduction-clauseModificado7.c -o reduction-clauseModificado7

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ export OMP_NUM_THREADS=3

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./reduction-clauseModificado7 10
Tras 'parallel' suma=45

┌──(icaro㉿kali)-[~/…/Practica/Seminario2/Codigo/BP2]
└─$ ./reduction-clauseModificado7 20
Tras 'parallel' suma=190
```

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

8. Implementar en paralelo el producto matriz por vector con OpenMP usando la directiva `for`. Partir del código secuencial disponible en SWAD. 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, v2, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (4) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código que calcula el producto matriz vector, el número de hilos que usa 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`

```
#ifdef VECTOR_DYNAMIC
double *v1, *v2, **m;
v1 = (double*)malloc(N*sizeof(double));
v2 = (double*)malloc(N*sizeof(double));
m = (double**)malloc(N*sizeof(double*));
if ((v1 == NULL) || (v2 == NULL) || (m == NULL)) {
    printf("No hay suficiente espacio para v1, v2 y m \n");
    exit(EXIT_FAILURE);
}
for (i = 0; i < N; i++) {
    m[i] = (double*)malloc(N*sizeof(double));
    if (m[i] == NULL) {
        printf("No hay suficiente espacio para m \n");
        exit(EXIT_FAILURE);
    }
}
#endif

// Inicializar vector y matriz
#pragma omp parallel for private(j)
for (i = 0; i < N; i++){
    v1[i] = 0.1*i;
    v2[i] = 0;
    for (j = 0; j < N; j++)
        m[i][j] = i*N+j;
}

// Calcular v2 = m * v1
//clock_gettime(CLOCK_REALTIME,&cgt1);
#pragma omp single
cgt1= omp_get_wtime();

#pragma omp parallel for private(j)
for(i = 0; i < N; i++){
    for (j = 0; j < N; j++)
        v2[i] += m[i][j] * v1[j];
}

#pragma omp single
cgt2 = omp_get_wtime();

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

RESPUESTA:

para pmv-OpenMP-a.c

Lo primero que he hecho es cambiar como se calculaban los tiempos cgt1 y cgt2 ngct por las funciones de omp_get_wtime(), les he asignado un single para que no se interfolien, esto es igual a otros ejercicios de la relacion anterior .

Además para paralelizar el bucle que recorre para las filas he paralelizado el bucle que inicializa la matriz y la variable que lo recorre hacer un private(j) y tmb a la hora del calculo de la matriz hacer lo mismo de la inicializacion, un pragma omp parallel for y poner la variable de la iteracion a privada

Supongo que esta bien por que da los mismos valores que la version secuencial y ademas con valores altos los tiempos son menores que el secuencial

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

```
// Inicializar vector y matriz
#pragma omp parallel  private(i)
for (i = 0; i < N; i++){
    v1[i] = 0.1*i;
    v2[i] = 0;

    #pragma omp for
    for (j = 0; j < N; j++)
        m[i][j] = i*N+j;
}

// Calcular v2 = m * v1
//clock_gettime(CLOCK_REALTIME,&cgt1);
#pragma omp single
cgt1= omp_get_wtime();

//#pragma omp parallel for private(j)

double suma;

for(i = 0; i < N; i++){
    #pragma omp parallel firstprivate(suma)
    {
            #pragma omp for
            for (j = 0; j < N; j++)
                    suma += m[i][j] * v1[j];

            #pragma omp critical
                    v2[i] += suma;
    }
}

#pragma omp single
cgt2 = omp_get_wtime();
```

El error es que tenia un for arriba de otro, lo que no puedo anidad, solo necesitaba poner el private

```
ac274@atcgrid:~/bp2
Archivo  Acciones  Editar  Vista  Ayuda
[ac274@atcgrid bp2]$ gcc -O2 -fopenmp pmv-OpenMP-b.c -o pmvOpenMP-b
pmv-OpenMP-b.c: En la función 'main':
pmv-OpenMP-b.c:85:10: error: la región de trabajo compartido puede no estar bien anidada dentro de la región de trab
ajo compartido, 'critical', 'ordered', 'master', 'task' explícita o región 'taskloop'
   85 |  #pragma omp for
      |          ^~~
```

CAPTURAS DE PANTALLA:

```
[ac274@atcgrid bp2]$ srun -pac -Aac -n1 -c12 --hint=nomultithread pmv-secuencial 11
Tiempo: 0.000000440      Tamaño: 11      v2[0]: 38.500000        v2[N-1]: 643.500000
[ac274@atcgrid bp2]$ srun -pac -Aac -n1 -c12 --hint=nomultithread pmvOpenMP-a 11
Tiempo: 0.000008210      Tamaño: 11      v2[0]: 38.500000        v2[N-1]: 643.500000
[ac274@atcgrid bp2]$ srun -pac -Aac -n1 -c12 --hint=nomultithread pmvOpenMP-a 110
Tiempo: 0.000013016      Tamaño: 110     v2[0]: 43763.500000     v2[N-1]: 7231768.500000
[ac274@atcgrid bp2]$ srun -pac -Aac -n1 -c12 --hint=nomultithread pmv-secuencial 110
Tiempo: 0.000024529      Tamaño: 110     v2[0]: 43763.500000     v2[N-1]: 7231768.500000
```

```
[ac274@atcgrid bp2]$ srun -pac -Aac -n1 -c12 --hint=nomultithread pmvOpenMP-b 110
Tiempo: 0.001418510      Tamaño: 110     v2[0]: 43763.500000     v2[N-1]: 7231768.500000
```

9. 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`

```
// Inicializar vector y matriz
#pragma omp parallel  private(i)
for (i = 0; i < N; i++){
    v1[i] = 0.1*i;
    v2[i] = 0;

    #pragma omp for
    for (j = 0; j < N; j++)
        m[i][j] = i*N+j;
}

// Calcular v2 = m * v1
//clock_gettime(CLOCK_REALTIME,&cgt1);
#pragma omp single
cgt1= omp_get_wtime();

//#pragma omp parallel for private(j)

double suma;

for(i = 0; i < N; i++){
    //#pragma omp parallel firstprivate(suma)
    #pragma omp parallel reduction(+:suma)
    {
            #pragma omp for
            for (j = 0; j < N; j++)
                    suma += m[i][j] * v1[j];

            #pragma omp critical
                    v2[i] += suma;
    }
}

#pragma omp single
cgt2 = omp_get_wtime();
```

RESPUESTA:

He implementado reduction(+:suma) en vez del first private, con el reduction he inicializado las copias privadas de la suma a 0, y + por que estamos con una suma

CAPTURAS DE PANTALLA:

```
[ac274@atcgrid bp2]$ gcc -O2 -fopenmp pmv-OpenMP-reducion.c -o pmvOpenMP-reducion
[ac274@atcgrid bp2]$ srun -pac -Aac -n1 -c12 --hint=nomultithread pmvOpenMP-reducion 4
Tiempo: 0.000037078      Tamaño: 4
Matriz:
        0.000000        1.000000        2.000000        3.000000
        4.000000        5.000000        6.000000        7.000000
        8.000000        9.000000        10.000000       11.000000
        12.000000       13.000000       14.000000       15.000000

Vector:
        0.000000 0.100000 0.200000 0.300000

Vector resultado:
        1.400000 3.800000 6.200000 8.600000
[ac274@atcgrid bp2]$ srun -pac -Aac -n1 -c12 --hint=nomultithread pmvOpenMP-reducion 110
Tiempo: 0.000972627      Tamaño: 110     v2[0]: 43763.500000     v2[N-1]: 7231768.500000
```

10. Realizar una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid4, en uno de los nodos de la cola ac 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):

JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:

CAPTURA DE PANTALLA del script `pmv-OpenmMP-script.sh`

CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):

TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia):

Tabla 1. Tiempos de ejecución del código secuencial y de la versión paralela para atcgrid y para el PC personal

	atcgrid1, atcgrid2 o atcgrid3				**atcgrid4**				**PC**			
	Tamaño= entre 5000 y 10000		**Tamaño= entre 10000 y 100000**		**Tamaño= entre 5000 y 10000**		**Tamaño= entre 10000 y 100000**		**Tamaño= entre 5000 y 10000**		**Tamaño= entre 10000 y 100000**	
Nº de núcleos (p)	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**
Código Secuencial		----		----		----		----		----		----
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
32												

COMENTARIOS SOBRE LOS RESULTADOS: