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): Álvaro Vega Romero
Grupo de prácticas y profesor de prácticas: Niceto Rafael Luque Sola
Fecha de entrega: 22/04/21
Fecha evaluación en clase: 23/04/22

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con las normas de prácticas que se encuentra en SWAD

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:

Hay un fallo de compilación ya que no definimos el ámbito de las variables que usamos. (el de a si, ya que viene en el código que nos dan la clausula shared (a).

Para el apartado b, lo que deberíamos hacer es usar la clausula shared para hacer que todas las hebras compartiesen la variable n.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```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) shared(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:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer1] 2021-04-10 Saturday
$gcc -O2 -fopenmp -o shared-clauseModificado shared-clauseModificado.c
shared-clauseModificado.c:5:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
shared-clauseModificado.c: In function ‘main’:
shared-clauseModificado.c:12:9: error: ‘n’ not specified in enclosing ‘parallel’
 #pragma omp parallel for shared(a) default(none)
         ^~~
shared-clauseModificado.c:12:9: error: enclosing ‘parallel’
```

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer1] 2021-04-10 Saturday
$gcc -O2 -fopenmp -o shared-clauseModificado shared-clauseModificado.c
shared-clauseModificado.c:5:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer1] 2021-04-10 Saturday
$./shared-clauseModificado
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:

(Solo podemos ver el resultado de la suma parcial de la hebra master ya que el resto dejan de existir al salir de la región paralela, y para mostrar la suma de todas las sumas podríamos usar la clausula reduction o la directiva critical).

Vemos que el en la ejecución el resultado que siempre imprime la hebra master es 0 (al ser privada suma, cada hebra hace la suma guardando el resultado en “su suma” en vez de suma original) , dando igual el valor al que iniciemos a suma.

(Si hiciemos el print dentro la región paralella, mostraríamos el resultado de sumarle las sumas que hace cada hebra en el for pero, además, sumándole el valor al que hubiéramos inicializado suma)

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.c`

```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;

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

#pragma omp parallel private(suma)
{
        suma=5;
        #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:
(resultado con varias inicializaciones de suma=10 y suma=5)

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer2] 2021-04-10 Saturday
$./private-clauseModificado_a
thread 0 suma a[0] / thread 6 suma a[6] / thread 3 suma a[3] / thread 5 suma a[5] / thread 4 suma a[4] / thread 2 suma a[2] / thread 1 suma a[1] /
* thread 0 suma= 0
```

```
$./private-clauseModificado_a
thread 1 suma a[1] / thread 3 suma a[3] / thread 2 suma a[2] / thread 6 suma a[6] / thread 5 suma a[5] / thread 4 suma a[4] / thread 0 suma a[0] /
* thread 0 suma= 0
```

(b) RESPUESTA:
Suma tiene el valor al que la inicializamos ya que sigue sin guardarse el valor privado de la zona paralelizada.

Si mostrásemos el resultado en la zona paralelizada, veríamos que mostrarían basura ya que al hacer private(suma), se le daría un valor basura a cada suma de cada hebra.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_b.c`

```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 = 5;

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

#pragma omp parallel private(suma)
{
        #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:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer2] 2021-04-10 Saturday
$./private-clauseModificado_b
thread 4 suma a[4] / thread 0 suma a[0] / thread 2 suma a[2] / thread 1 suma a[1] / thread 6 suma a[6] / thread 5 suma a
[5] / thread 3 suma a[3] /
* thread 0 suma= 5
```

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

RESPUESTA:
Al quitar la cláusula private, suma pasaría a ser una variable compartida y mostraría el resultado producido por la última hebra que modificó 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
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:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer3] 2021-04-10 Saturday
$./private-clauseModificado3
thread 2 suma a[2] / thread 3 suma a[3] / thread 5 suma a[5] / thread 1 suma a[1] / thread 6 suma a[6] / thread 4 suma a
[4] / thread 0 suma a[0] /
* thread 2 suma= 5
* thread 3 suma= 5
* thread 5 suma= 5
* thread 0 suma= 5
* thread 6 suma= 5
* thread 1 suma= 5
* thread 7 suma= 5
* thread 4 suma= 5
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer3] 2021-04-10 Saturday
$./private-clauseModificado3
thread 1 suma a[1] / thread 3 suma a[3] / thread 6 suma a[6] / thread 5 suma a[5] / thread 2 suma a[2] / thread 0 suma a
[0] / thread 4 suma a[4] /
* thread 0 suma= 2
* thread 6 suma= 2
* thread 1 suma= 2
* thread 4 suma= 2
* thread 3 suma= 2
* thread 2 suma= 2
* thread 7 suma= 2
* thread 5 suma= 2
```

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:

Imprime 9 ya que la clausula lastprivate hace que suma coja la "última" suma parcial si se hiciese de forma secuencial (a[9] al ser n=10)

CAPTURAS DE PANTALLA:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer4] 2021-04-10 Saturday
$./firstlastprivate_a
 thread 5 suma a[7] suma=7
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 6 suma a[8] suma=8
 thread 7 suma a[9] suma=9
 thread 3 suma a[5] suma=5
 thread 4 suma a[6] suma=6
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 2 suma a[4] suma=4

Fuera de la construcción parallel suma=9
```

(b) RESPUESTA: Para mostrar otro valor, podríamos o bien cambiar el valor que tiene el último a[i] que se ejecuta o bien, modificando el número de hebras podemos hacer que la suma parcial de una hebra tenga varias sumas de varias componentes, y se muestre la suma parcial de la última hebra que ejecutó el for.

CAPTURAS DE PANTALLA:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer4] 2021-04-10 Saturday
$export OMP_NUM_THREADS=2
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer4] 2021-04-10 Saturday
$./firstlastprivate_b
 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
```

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: Se observa que solo una hebra tiene el valor que se ha escaneado.
Esto es así, ya que solo una hebra accede al single y al resto de hebras no se le difunde el valor de a que se ha obtenido desde la terminal

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```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("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:

```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("Depués de la región parallel:\n");

    for (i=0; i<n; i++)
        printf("b[%d] = %d\t",i,b[i]);
    printf("\n");
}
```

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 imprime el mismo resultado pero sumándole 10 (el valor inicial) a la suma.

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

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:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer6] 2021-04-10 Saturday
$./reduction-clauseModificado 20
Tras 'parallel' suma=200
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer6] 2021-04-10 Saturday
$./reduction-clauseModificado 10
Tras 'parallel' suma=55
```

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:

Ocurre que si no pusiésemos que es zona crítica o que la instrucción es atómica, podría haber varias posible interfoliaciones las cuales no serían válidas (ej: una hebra modifica el valor de suma y posteriormente la modifica otra sin tener en cuenta que había sido modificada)

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

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
    for (i=0; i<n; i++)
        #pragma omp critical //no es de trabajo compartido
        suma += a[i];

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

CAPTURAS DE PANTALLA:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer7] 2021-04-10 Saturday
$./reduction-clauseModificado7 10
Tras 'parallel' suma=45
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp2/ejer7] 2021-04-10 Saturday
$./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 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 (dos versiones pmv-secuencialGLOBAL.c y pmv-secuencialDINAMICO.c respectivamente)

pmv-secuencialGLOBAL.c

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

//gcc -O2 -fopenmp -o pmv-secuencialGLOBAL pmv-secuencialGLOBAL.c

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

#define MAX 33554432    //=2^25

int main(int argc, char const * argv[])
{
    if(argc != 2)
    {
        fprintf(stderr, "Introduce un único argumento: N \n");
        exit(EXIT_FAILURE);
    }

    int i, j;
    unsigned int n = atoi(argv[1]);
    double cgt1, cgt2, tiempo;
    double suma = 0;

    if(n>MAX)
            n=MAX;

    double A[n][n];  //Matriz
    double B[n];  //Vector
    double C[n];  //Vector resultante

    for(i = 0 ; i < n ; i++) //Rellenar matriz de datos "aleatorios" y Vectores
    {
        B[i]= 0.5*i;
        C[i]=0;
        for(j = 0 ; j < n ; j++)
            A[i][j] = 0.5*i - 0.5*j;
    }

    cgt1 = omp_get_wtime();

    for(i = 0 ; i < n ; i++) //Calculo del producto
    {
        suma = 0;
        for(j = 0 ; j < n ; j++)
        {
            suma += A[i][j] * B[j];

            /*
                printf("A[%d][%d] = %f con i=%d y j=%d ", i, j, A[i][j], i, j);
                printf("B[%d] = %f con j=%d ", j, B[j], j);
                printf("\n\n");
            */
        }

        C[i]=suma;
    }

    cgt2 = omp_get_wtime();
    tiempo = cgt2 - cgt1;

    printf("Tiempo de ejecución: %f \n", tiempo);
    printf("Valor de n: %d \n", n);

    if(n < 15) //Si es de tam pequeño
    {
        for(int i = 0 ; i < n ; i++)
            printf("C[%d] = %f \n", i, C[i]);
    }
    else
    {
            printf("C[0] = %f \n", C[0]);
            printf("C[ULTIMO] = %f \n", C[n-1]);
    }
}
```

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

//gcc -O2 -fopenmp -o pmv-secuencialGLOBAL pmv-secuencialGLOBAL.c

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

#define MAX 33554432    //=2^25

int main(int argc, char const * argv[])
{
    if(argc != 2)
    {
        fprintf(stderr, "Introduce un único argumento: N \n");
        exit(EXIT_FAILURE);
    }

    int i, j;
    unsigned int n = atoi(argv[1]);
    double cgt1, cgt2, tiempo;
    double suma = 0;

    if(n>MAX)
            n=MAX;


    double **A, *B, *C, t;

    A=(double**) malloc (n*sizeof(double*));
    B=(double*) malloc (n*sizeof(double));
    C=(double*) malloc (n*sizeof(double));

    if((A==NULL) || (B==NULL) || (C==NULL))
    {
        printf("Error en la reserva de memoria\n");
        exit(-2);
    }

    for(i = 0 ; i < n ; i++)
    {
        A[i] = (double*) malloc (n*sizeof(double));

        if(A[i] == NULL)
        {
            printf("Error en la reserva de memoria\n");
            exit(-3);
        }
    }

    for(i = 0 ; i < n ; i++) //Rellenar matriz de datos "aleatorios" y Vectores
    {
        B[i]= 0.5*i;
        C[i]=0;
        for(j = 0 ; j < n ; j++)
            A[i][j] = 0.5*i - 0.5*j;
    }

    cgt1 = omp_get_wtime();

    for(i = 0 ; i < n ; i++) //Calculo del producto
    {
        suma = 0;
        for(j = 0 ; j < n ; j++)
        {
            suma += A[i][j] * B[j];

            /*
                printf("A[%d][%d] = %f con i=%d y j=%d ", i, j, A[i][j], i, j);
                printf("B[%d] = %f con j=%d ", j, B[j], j);
                printf("\n\n");
            */
        }

        C[i]=suma;
    }

    cgt2 = omp_get_wtime();
    tiempo = cgt2 - cgt1;

    //Salida

    printf("Tiempo de ejecución: %f \n", tiempo);
    printf("Valor de n: %d \n", n);

    if(n < 15) //Si es de tam pequeño
    {
        for(int i = 0 ; i < n ; i++)
            printf("C[%d] = %f \n", i, C[i]);
    }
    else
    {
            printf("C[0] = %f \n", C[0]);
            printf("C[ULTIMO] = %f \n", C[n-1]);
    }

    //Liberar memoria

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

    free(A);
    free(B);
    free(C);
}
```

CAPTURAS DE PANTALLA:

```
[ÁlvaroVega b2estudiante24@atcgrid:~/bp2/ejer8] 2021-04-16 Friday
$srun -p ac pmv-secuencialDINAMICO 10
Tiempo de ejecución: 0.000001
Valor de n: 10
C[0] = -71.250000
C[1] = -60.000000
C[2] = -48.750000
C[3] = -37.500000
C[4] = -26.250000
C[5] = -15.000000
C[6] = -3.750000
C[7] = 7.500000
C[8] = 18.750000
C[9] = 30.000000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp2/ejer8] 2021-04-16 Friday
$srun -p ac pmv-secuencialGLOBAL 10
Tiempo de ejecución: 0.000002
Valor de n: 10
C[0] = -71.250000
C[1] = -60.000000
C[2] = -48.750000
C[3] = -37.500000
C[4] = -26.250000
C[5] = -15.000000
C[6] = -3.750000
C[7] = 7.500000
C[8] = 18.750000
C[9] = 30.000000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp2/ejer8] 2021-04-16 Friday
$
```

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 (mismo que dinámico del ejer8 pero con el cambio mostrado en la foto)`

```
#pragma omp parallel for private (j) //i ya es private
for(i = 0 ; i < n ; i++) //Calculo del producto
{
        suma = 0;
        for(j = 0 ; j < n ; j++)
        {
                suma += A[i][j] * B[j];

                /*
                        printf("A[%d][%d] = %f con i=%d y j=%d ", i, j, A[i][j], i, j);
                        printf("B[%d] = %f con j=%d ", j, B[j], j);
                        printf("\n\n");
                */
        }

        C[i]=suma;
}
```

CAPTURA CÓDIGO FUENTE: `pmv-OpenMP-b.c (mismo que dinámico del ejer8 pero con el cambio mostrado en la foto)`

```
#pragma omp parallel private(i,j)
for(i = 0 ; i < n ; i++) //Calculo del producto
{
    double tmp = 0;
    #pragma omp for
    for(j = 0 ; j < n ; j++)
    {
        tmp += A[i][j] * B[j];
    }

    #pragma omp atomic
        C[i]+=tmp;
}
```

RESPUESTA:

Sobre la solución, en el ejercicio a simplemente el bucle que recorre la matriz lo paralelizo y pongo como variable privada a cada hebra la j (la i por defecto ya es privada) para luego recorrer las columnas y vector también de forma privada.

En cambio, en el apartado b, tenemos una región paralelizada y dentro de esta un pragma omg for para las columnas y una directiva atomic para sumar los resultados de las distintas hebras que han entrado a la zona del bucle de las columnas y vector.

Solo he tenido un problema en la compilación y fue por en un error en el ejercicio de b de hacer una región paralelizada para un bucle (pragma omp parallel for) y dentro hacer otra región for.

CAPTURAS DE PANTALLA:

```
[ÁlvaroVega b2estudiante24@atcgrid:~/bp2/ejer9] 2021-04-16 Friday
$srun -p ac pmv-OpenMP-a 10
Tiempo de ejecución: 0.000053
Valor de n: 10
C[0] = -71.250000
C[1] = -60.000000
C[2] = -48.750000
C[3] = -37.500000
C[4] = -26.250000
C[5] = -15.000000
C[6] = -3.750000
C[7] = 7.500000
C[8] = 18.750000
C[9] = 30.000000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp2/ejer9] 2021-04-16 Friday
$srun -p ac pmv-OpenMP-b 10
Tiempo de ejecución: 0.000055
Valor de n: 10
C[0] = -71.250000
C[1] = -60.000000
C[2] = -48.750000
C[3] = -37.500000
C[4] = -26.250000
C[5] = -15.000000
C[6] = -3.750000
C[7] = 7.500000
C[8] = 18.750000
C[9] = 30.000000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp2/ejer9] 2021-04-16 Friday
```

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 (solo modifico la forma de calcular la multiplicación. Por eso la captura es solo eso)`

```
double tmp = 0;
#pragma omp parallel private(i,j)
for(i = 0 ; i < n ; i++) //Calculo del producto
{
    #pragma omp for reduction (+:tmp)
    for(j = 0 ; j < n ; j++)
    {
        tmp += A[i][j] * B[j];
    }

    #pragma omp single
    {
        C[i]=tmp;
        tmp = 0;
    }
}
```

RESPUESTA:

El procedimiento usado es el mismo usado en el ejercicio anterior en el apartado b, pero usando la clausula reduction que coge todas "las sumas parciales" de cada hebra y las suma (:+) y luego una hebra la asigna al vector resultado en la posición correspondiente y vuelve a poner la suma a 0. En este caso suma se llama tmp No he tenido ningún error.

CAPTURAS DE PANTALLA:

```
[ÁlvaroVega b2estudiante24@atcgrid:~/bp2/ejer10] 2021-04-16 Friday
$srun -p ac pmv-OpenmMP-reduction 10
Tiempo de ejecución: 0.000120
Valor de n: 10
C[0] = -71.250000
C[1] = -60.000000
C[2] = -48.750000
C[3] = -37.500000
C[4] = -26.250000
C[5] = -15.000000
C[6] = -3.750000
C[7] = 7.500000
C[8] = 18.750000
C[9] = 30.000000
[ÁlvaroVega b2estudiante24@atcgrid:~/bp2/ejer10] 2021-04-16 Friday
```

11. 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: