

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): Francisco José González García

Grupo de prácticas: A2

Fecha de entrega: 16/04/18

Fecha evaluación en clase: 17/04/18

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:

```
/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-03 martes
> gcc -fopenmp -O2 -o shared-clause codigo/shared-clauseModificado.c
codigo/shared-clauseModificado.c: In function 'main':
codigo/shared-clauseModificado.c:9:9: error: 'n' not specified in enclosing parallel
#pragma omp parallel for shared(a) default(none)
      ^
codigo/shared-clauseModificado.c:9:9: error: enclosing parallel
```

Como se puede apreciar en la captura el código no llega a compilar porque no se ha especificado el alcance de la variable `n` usada dentro de la región.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.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) shared(n)
    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:

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-5726] 18-04-03 martes
> gcc -fopenmp -O2 -o shared-clause codigo/shared-clauseModificado.c

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-5726] 18-04-03 martes
> ./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. ¿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:

Lo que ocurre es que la variable tendrá un valor indefinido dentro de la region `parallel` y por tanto la suma no se realizará correctamente.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.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 = 0;
    for (i = 0; i < n; i++) a[i] = i;
    printf("valor suma fuera de parallel: %d\n", suma);
    #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:**Inicializado a 5 fuera de parallel, inicializado a 0 dentro:**

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-03 martes
> gcc -fopenmp -O2 -o private-clause codigo/private-clauseModificado.c

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-03 martes
> ./private-clause
valor suma fuera de parallel: 5
valor suma dentro de parallel: 0
thread 3 suma a[6] / valor suma dentro de parallel: 0
thread 1 suma a[2] / thread 1 suma a[3] / valor suma dentro de parallel: 0
thread 2 suma a[4] / thread 2 suma a[5] / valor suma dentro de parallel: 0
thread 0 suma a[0] / thread 0 suma a[1] /
* thread 1 suma= 5
* thread 3 suma= 6
* thread 2 suma= 9
* thread 0 suma= 1

```

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

RESPUESTA:

Se produce un cálculo erróneo de la suma que debería hacer cada hebra debido a que al eliminar la cláusula `private(suma)` la variable `suma` pasa a ser compartida por todas las hebras, por lo que se producen condiciones de carrera y el resultado final que tendrá la variable será indeterminado y diferente en cada ejecución.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.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 = 0;
    for (i = 0; i < n; i++) a[i] = i;
    #pragma omp parallel
    {
        #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:

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> g++ codigo/private-clauseModificado3.c -fopenmp -o private-clauseModificado3

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> ./private-clauseModificado3
thread 3 suma a[6] / thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma a[4] /
hread 2 suma a[5] / thread 1 suma a[2] / thread 1 suma a[3] /
* thread 0 suma= 15
* thread 1 suma= 15
* thread 3 suma= 15
* thread 2 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:

No, depende del número de hebras. Según la descripción de `lastprivate` la variable a la que afecta la cláusula mantiene el valor que se le da en la última iteración de un `for`, o en la última sección de una región `section`. Por tanto, al ser fijas el número de iteraciones el valor de suma dependerá del número de hebras que ejecuten el código.

CAPTURAS DE PANTALLA:

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> export OMP_NUM_THREADS=2

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> ./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:

Se puede observar que no todas las componentes del vector `b` están inicializadas al valor que introducimos desde teclado, esto se produce por la eliminación de la cláusula `copyprivate` provocando que solo las componentes del vector rellenas por la hebra que ha ejecutado la región `single` sean las que tengan el valor correcto.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```

#include <omp.h>
#include <stdio.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("Después de la región parallel:\n");
for (i = 0; i < n; i++) printf("b[%d] = %d\t", i, b[i]);
printf("\n");
}

```

CAPTURAS DE PANTALLA:

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> gcc -O2 codigo/copyprivate-clauseModificado.c -o copyprivate -fopenmp
/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> export OMP_NUM_THREADS=4
/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> ./copyprivate

Introduce valor de inicialización a : 5

Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 0      b[1] = 0      b[2] = 0      b[3] = 5      b[4] = 5      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:

Se imprime el resultado anterior más 10, debido a que hemos cambiado el valor de `suma=10`, cuyo valor no se machaca porque es una variable global a todas las hebras y al hacer la reducción obtenemos: `suma = suma + suma0 + suma1 + suma2 + sumaN`

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
int main(int argc, char **argv) {
    int i, n = 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:

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> gcc -O2 codigo/reduction-clauseModificado.c -o reduction -fopenmp

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> ./reduction 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 usar directivas de trabajo compartido.

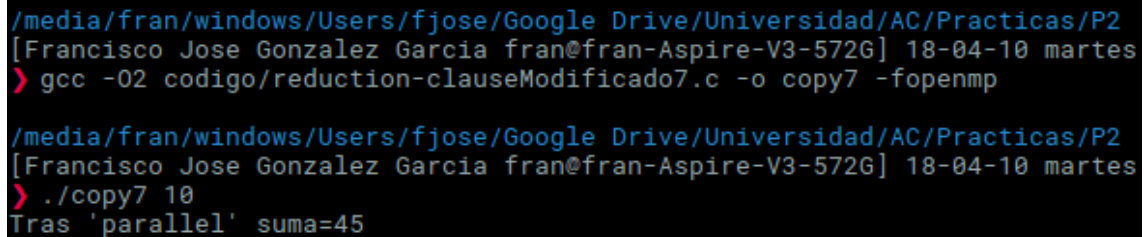
RESPUESTA:**CAPTURA CÓDIGO FUENTE:** `reduction-clauseModificado7.c`

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#define omp_set_num_threads() 1
#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;
    omp_set_num_threads(4);
    int aux = 0;
    #pragma omp parallel firstprivate(aux)
    {
        if (omp_get_thread_num() == 0)
            for (i = 0; i < n / 4; i++) aux += a[i];
        else if (omp_get_thread_num() == 1)
            for (i = n / 4; i < n / 2; i++) aux += a[i];
        else if (omp_get_thread_num() == 2)
            for (i = n / 2; i < 3 * n / 4; i++) aux += a[i];
        else if (omp_get_thread_num() == 3)
            for (i = 3 * n / 4; i < n; i++) aux += a[i];
    }
    suma = aux;
    printf("Tras 'parallel' suma=%d\n", suma);
}

```

```
#pragma omp atomic
    suma += aux;
}
printf("Tras 'parallel' suma=%d\n", suma);
}
```

CAPTURAS DE PANTALLA:


```
/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> gcc -O2 codigo/reduction-clauseModificado7.c -o copy7 -fopenmp

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-10 martes
> ./copy7 10
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):

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

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

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

int main(int argc, char** argv) {
    if (argc < 2) {
        perror("Falta tamaño del vector");
        exit(EXIT_FAILURE);
    }
    int N = atoi(argv[1]);
    int** matriz = malloc(N * sizeof(int*));
    for (int i = 0; i < N; i++) matriz[i] = malloc(N * sizeof(int));
    int* vector = malloc(N * sizeof(int));
    int* solucion = malloc(N * sizeof(int));

    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++) matriz[i][j] = i + j;
    for (int i = 0; i < N; i++) vector[i] = i;

    for (int i = 0; i < N; i++) {
        int aux = 0;
        for (int j = 0; j < N; j++) aux += matriz[i][j] * vector[j];
        solucion[i] = aux;
    }
    printf("resultado: {");
    if (N < 12) {
```

```

    for (int i = 0; i < N; i++) printf("%d, ", solucion[i]);
    printf("\b\b}");
} else{
    printf("%d,..., %d}", solucion[0], solucion[N-1]);
}

for (int i = 0; i < N; i++) free(matriz[i]);
free(matriz);
free(vector);
free(solucion);
}

```

CAPTURAS DE PANTALLA:

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-14 sábado
> gcc -O2 -o pmv-secuencial codigo/pmv-secuencial.c

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-14 sábado
> ./pmv-secuencial 10
resultado: {285, 330, 375, 420, 465, 510, 555, 600, 645, 690}%

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-14 sábado
> ./pmv-secuencial 12
resultado: {506,..., 1232}%

```

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):
- una primera que paralelice el bucle que recorre las filas de la matriz y
 - 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

```

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

```



```

int main(int argc, char** argv) {
    if (argc < 2) {
        perror("Falta tamaño del vector");
        exit(EXIT_FAILURE);
    }
    int N = atoi(argv[1]);
    int** matriz = malloc(N * sizeof(int*));
    for (int i = 0; i < N; i++) matriz[i] = malloc(N * sizeof(int));
    int* vector = malloc(N * sizeof(int));
    int* solucion = malloc(N * sizeof(int));

    #pragma omp parallel
    {
        #pragma omp for collapse(2)
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++) matriz[i][j] = i + j;
        #pragma omp for
        for (int i = 0; i < N; i++) vector[i] = i;

        int aux = 0;
        #pragma omp for
        for (int i = 0; i < N; i++) {
            aux = 0;
            for (int j = 0; j < N; j++) aux += matriz[i][j] * vector[j];
            solucion[i] = aux;
        }
    }
    printf("resultado: {");
    if (N < 12) {
        for (int i = 0; i < N; i++) printf("%d, ", solucion[i]);
        printf("\b\b}");
    } else {
        printf("%d,..., %d}", solucion[0], solucion[N - 1]);
    }

    for (int i = 0; i < N; i++) free(matriz[i]);
    free(matriz);
    free(vector);
    free(solucion);
}

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

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

int main(int argc, char** argv) {
    if (argc < 2) {
        perror("Falta tamaño del vector");
        exit(EXIT_FAILURE);
    }
    int N = atoi(argv[1]);
    int** matriz = malloc(N * sizeof(int*));
    for (int i = 0; i < N; i++) matriz[i] = malloc(N * sizeof(int));
    int* vector = malloc(N * sizeof(int));
    int* solucion = malloc(N * sizeof(int));

    #pragma omp parallel

```

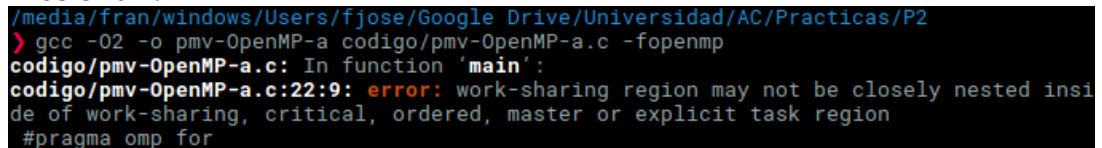
```

{
#pragma omp for collapse(2)
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++) matriz[i][j] = i + j;
#pragma omp for
    for (int i = 0; i < N; i++) vector[i] = i;
#pragma omp for
    for (int i = 0; i < N; i++) solucion[i] = 0;

    int aux = 0;
    for (int i = 0; i < N; i++) {
        aux = 0;
#pragma omp for
        for (int j = 0; j < N; j++) aux += matriz[i][j] * vector[j];
#pragma omp atomic
        solucion[i] += aux;
    }
}
printf("resultado: {");
if (N < 12) {
    for (int i = 0; i < N; i++) printf("%d, ", solucion[i]);
    printf("\b\b}");
} else {
    printf("%d,..., %d", solucion[0], solucion[N - 1]);
}

for (int i = 0; i < N; i++) free(matriz[i]);
free(matriz);
free(vector);
free(solucion);
}

```

RESPUESTA:**Problema 1:**


```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
> gcc -O2 -o pmv-OpenMP-a codigo/pmv-OpenMP-a.c -fopenmp
codigo/pmv-OpenMP-a.c: In function 'main':
codigo/pmv-OpenMP-a.c:22:9: error: work-sharing region may not be closely nested inside of work-sharing, critical, ordered, master or explicit task region
#pragma omp for

```

Al intentar paralelizar los dos bucles anidados de la inicialización de la matriz obtuve el problema que aparece en la imagen anterior. Esto se debe a que no se puede anidar una directiva for dentro de otra. Para resolver este problema me ayudé de una pregunta similar realizada en la página web 'Stack overflow' que respondía que para paralelizar dos bucles anidados se debe de hacer uso de la clausula *collapse*. A continuación el código corregido:

```

#pragma omp for collapse(2)
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++) matriz[i][j] = i + j;

```

Problema 2:

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
> ./pmv-secuencial 10
resultado: {285, 330, 375, 420, 465, 510, 555, 600, 645, 690}%

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
> gcc -O2 -o pmv-OpenMP-b codigo/pmv-OpenMP-b.c -fopenmp

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
> ./pmv-OpenMP-b 10
resultado: {145, 162, 179, 196, 213, 230, 247, 264, 281, 298}%

```

Para la versión b, se obtiene un problema de cálculo en una primera versión donde solo se ha incluido la directiva for para el bucle interior que recorre las columnas debido a que la variable *aux* donde se acumulan las sumas parciales es privada para cada hebra, por lo que el vector *solucion* almacenará el valor de la última suma parcial del bucle que no está completa. Para solucionar este problema se ha sumado(acumulado) de forma atómica cada suma parcial de *aux* en su posición correspondiente del vector *solucion*:

```

    int aux = 0;
    for (int i = 0; i < N; i++) {
        aux = 0;
        #pragma omp parallel firstprivate(aux)
        {
            #pragma omp for
            for (int j = 0; j < N; j++) aux += matriz[i][j] * vector[j];
            #pragma omp atomic
            solucion[i] += aux;
        }
    }

```

Compilación y comprobación de salida correcta del cálculo:

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-14 sábado
> gcc -O2 -o pmv-OpenMP-a codigo/pmv-OpenMP-a.c -fopenmp

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-14 sábado
> ./pmv-OpenMP-a 10
resultado: {285, 330, 375, 420, 465, 510, 555, 600, 645, 690}%

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-14 sábado
> ./pmv-OpenMP-a 12
resultado: {506,..., 1232}%

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-14 sábado
> gcc -O2 -o pmv-OpenMP-b codigo/pmv-OpenMP-b.c -fopenmp

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-14 sábado
> ./pmv-OpenMP-b 10
resultado: {285, 330, 375, 420, 465, 510, 555, 600, 645, 690}%

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-572G] 18-04-14 sábado
> ./pmv-OpenMP-b 12
resultado: {506,..., 1232}%

```

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-OpenMP-reduction.c`

```

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

int main(int argc, char** argv) {
    if (argc < 2) {
        perror("Falta tamaño del vector");
        exit(EXIT_FAILURE);
    }
    int N = atoi(argv[1]);
    int** matriz = malloc(N * sizeof(int*));
    for (int i = 0; i < N; i++) matriz[i] = malloc(N * sizeof(int));
    int* vector = malloc(N * sizeof(int));
    int* solucion = malloc(N * sizeof(int));

#pragma omp parallel

```

```

{
#pragma omp for collapse(2)
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++) matriz[i][j] = i + j;
#pragma omp for
    for (int i = 0; i < N; i++) vector[i] = i;
#pragma omp for
    for (int i = 0; i < N; i++) solucion[i] = 0;
}
int aux = 0;
for (int i = 0; i < N; i++) {
    aux = 0;
#pragma omp parallel for reduction(+ : aux)
    for (int j = 0; j < N; j++) aux += matriz[i][j] * vector[j];
    solucion[i] = aux;
}
printf("resultado: {");
if (N < 12) {
    for (int i = 0; i < N; i++) printf("%d, ", solucion[i]);
    printf("\b\b}");
} else {
    printf("%d, ..., %d", solucion[0], solucion[N - 1]);
}

for (int i = 0; i < N; i++) free(matriz[i]);
free(matriz);
free(vector);
free(solucion);
}

```

RESPUESTA:

No se ha tenido ningún problema de compilación y/ejecución durante la realización de este ejercicio. Aquí una captura del bucle for donde se hace uso de reduction:

```

int aux = 0;
for (int i = 0; i < N; i++) {
    aux = 0;
#pragma omp parallel for reduction(+ : aux)
    for (int j = 0; j < N; j++) aux += matriz[i][j] * vector[j];
    solucion[i] = aux;
}

```

CAPTURAS DE PANTALLA:**Compilación y comprobación de salida correcta del cálculo:**

```

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-5726] 18-04-14 sábado
> gcc -O2 -o pmv-OpenMP-reduction codigo/pmv-OpenMP-reduction.c -fopenmp

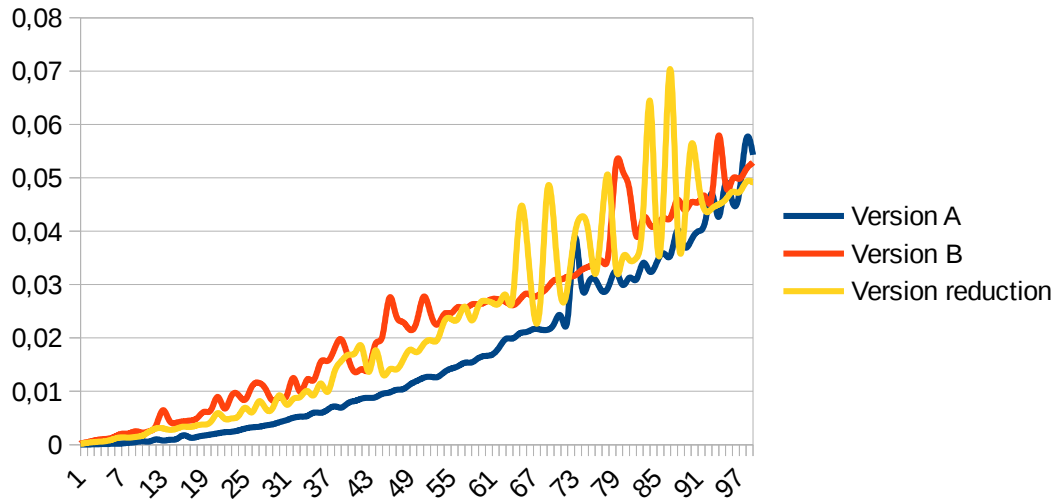
/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-5726] 18-04-14 sábado
> ./pmv-OpenMP-reduction 10
resultado: {285, 330, 375, 420, 465, 510, 555, 600, 645, 690}%

/media/fran/windows/Users/fjose/Google Drive/Universidad/AC/Practicas/P2
[Francisco Jose Gonzalez Garcia fran@fran-Aspire-V3-5726] 18-04-14 sábado
> ./pmv-OpenMP-reduction 12
resultado: {506, ..., 1232}%

```

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.

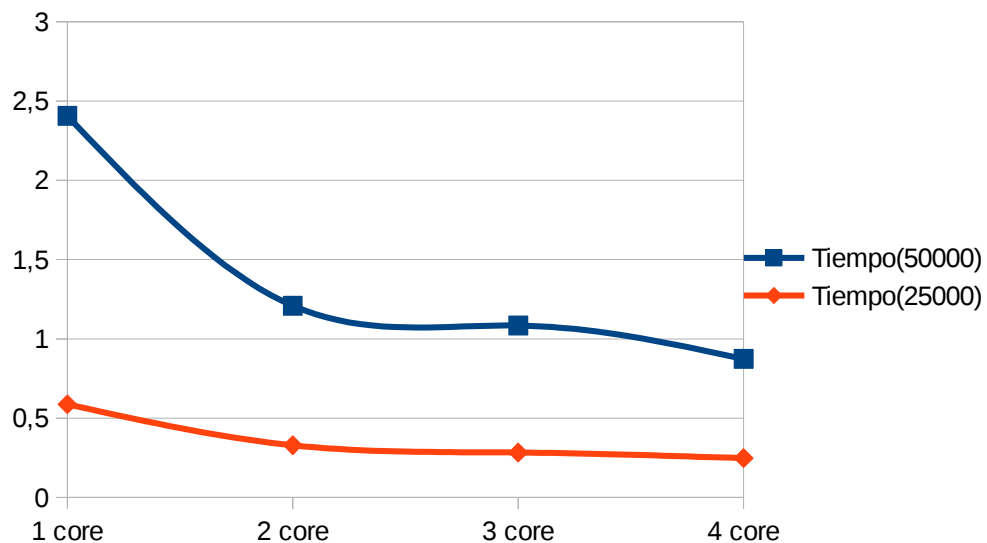
CAPTURAS DE PANTALLA (que justifique el código elegido):



Para elegir la mejor versión de entre los tres códigos paralelos implementados se ha realizado una medición de los tiempos que tarda en realizar el cálculo cada uno para diferentes tamaños de N y se ha obtenido la gráfica que vemos en la página anterior donde se aprecia que la mejor implementación es la versión A donde se paraleliza el bucle externo.

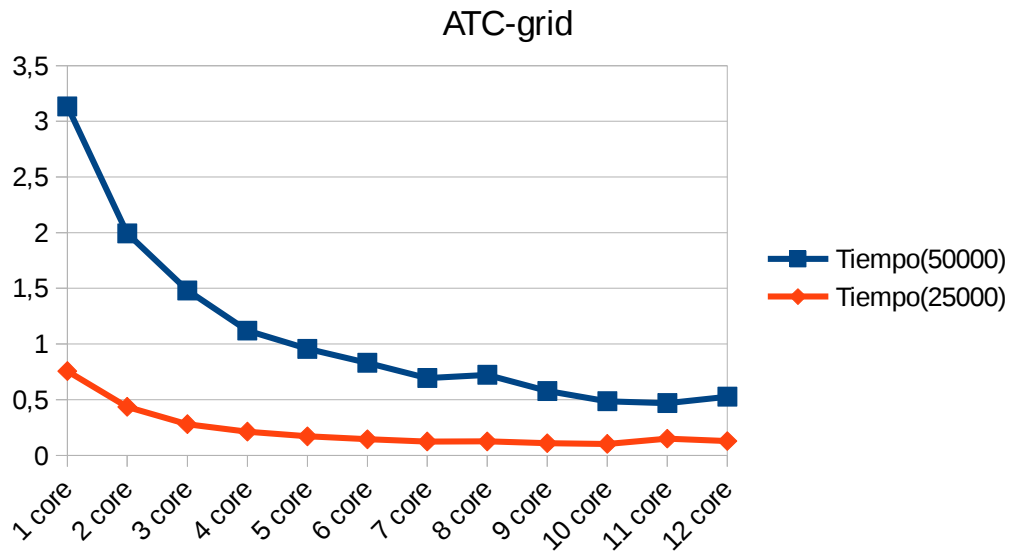
TABLA Y GRÁFICA (por ejemplo para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 30000 y 100000, y otro entre 5000 y 30000):

	Tiempo(50000)	Tiempo(25000)
1 core	2,40625	0,587564
2 core	1,209169	0,329023
3 core	1,084341	0,283121
4 core	0,873738	0,247508



	Tiempo(50000)	Tiempo(25000)
1 core	3,133532	0,755676
2 core	1,993206	0,434892
3 core	1,480595	0,280179
4 core	1,120122	0,212267
5 core	0,956637	0,170051
6 core	0,830405	0,144083
7 core	0,694175	0,123855
8 core	0,72257	0,125117
9 core	0,577803	0,10908
10 core	0,486023	0,102335
11 core	0,468502	0,149618
12 core	0,526604	0,128002

	Tiempo(50000)	Tiempo(25000)
1 core	3,133532	0,755676
2 core	1,993206	0,434892
3 core	1,480595	0,280179
4 core	1,120122	0,212267
5 core	0,956637	0,170051
6 core	0,830405	0,144083
7 core	0,694175	0,123855
8 core	0,72257	0,125117
9 core	0,577803	0,10908
10 core	0,486023	0,102335
11 core	0,468502	0,149618
12 core	0,526604	0,128002



COMENTARIOS SOBRE LOS RESULTADOS:

Como vemos en los resultados obtenidos en las tablas y reflejados en los gráficos se puede ver como la ganancia de prestaciones tiene un crecimiento logaritmico(en la gráfica se ha representado directamente el tiempo obtenido de ejecución, si se quisiera expresar en función de la ganancia respecto a la ejecución secuencial lo único que habría que hacer es dividir el tiempo obtenido para 1 core entre el tiempo obtenido para el resto de cores) y en el caso de *atc-grid*, a partir de los 10 cores la ganancia es practicamente nula o incluso decae un poco las prestaciones por el overhead asociado al uso de mayor número de cores y hebras.