

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): Arturo Cortés Sánchez

Grupo de prácticas y profesor de prácticas: C2

Fecha de entrega:

Fecha evaluación en clase:

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

Ejercicios basados en los ejemplos del seminario práctico

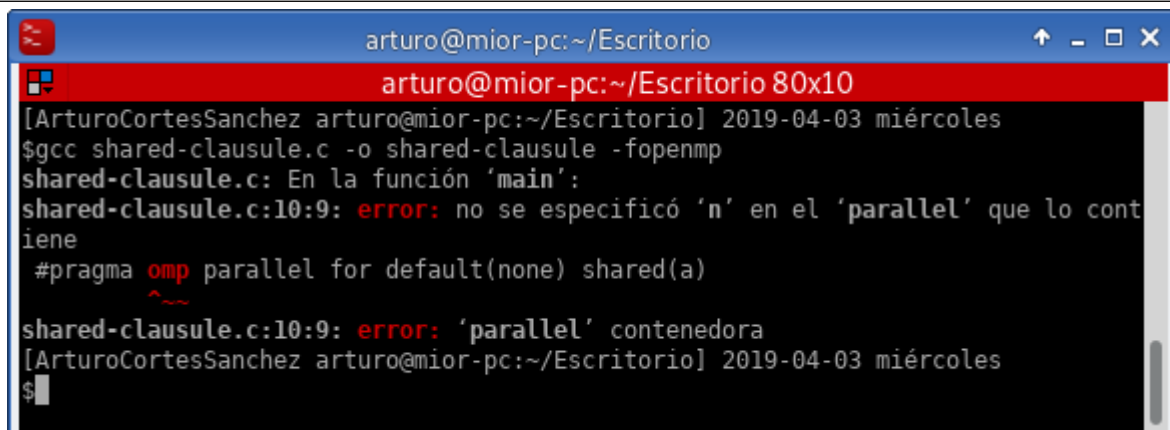
1. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? **(b)** Resuelva el problema generado sin eliminar `default(none)`. Añada el código con la modificación al cuaderno de prácticas. (Añada capturas de pantalla que muestren lo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

RESPUESTA:

Que el compilador da error ya que la variable `n` deja de estar compartida. Para evitar este error basta con añadir la variable `n` a la directiva `shared()`

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.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 #pragma omp parallel for default(none) shared(a, n)
11     for (i = 0; i < n; i++)
12         a[i] += i;
13     printf("Después de parallel for:\n");
14     for (i = 0; i < n; i++)
15         printf("a[%d] = %d\n", i, a[i]);
16 }
```



```
arturo@mior-pc: ~/Escritorio
arturo@mior-pc: ~/Escritorio 80x10
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$ gcc shared-clause.c -o shared-clause -fopenmp
shared-clause.c: En la función 'main':
shared-clause.c:10:9: error: no se especificó 'n' en el 'parallel' que lo contiene
#pragma omp parallel for default(none) shared(a)
      ~~~~~
shared-clause.c:10:9: error: 'parallel' contenedora
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$
```

2. Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel` e inicializar suma a un valor distinto de 0. Ejecute varias veces el código ¿Qué imprime el código fuera del `parallel`? (muéstrelo con una captura de pantalla) ¿Qué ocurre si en esta versión de `private-clause.c` se inicia la variable suma fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta (añada capturas de pantalla que muestren lo que ocurre). Añadir el código con las modificaciones al cuaderno de prácticas.

RESPUESTA:

Imprime una sola suma cuyo contenido es basura.

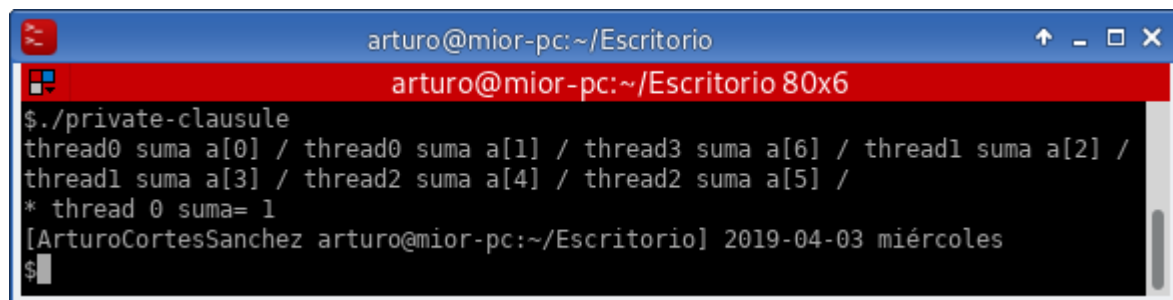
Si se inicializa fuera, al salir de la region `parallel`, suma conserva su valor inicial debido a que cada hebra tiene una copia privada de suma.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```

1  #include <stdio.h>
2  #ifdef _OPENMP
3  #include <omp.h>
4  #else
5  #define omp_get_thread_num() 0
6  #endif
7  int main() {
8      int i, n = 7;
9      int a[n], suma;
10     for (i = 0; i < n; i++)
11         a[i] = i;
12     suma = 1;
13     #pragma omp parallel private(suma)
14     {
15
16     #pragma omp for
17     for (i = 0; i < n; i++) {
18         suma = suma + a[i];
19         printf("thread%d suma a[%d] / ", omp_get_thread_num(), i);
20     }
21     }
22     printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
23     printf("\n");
24 }

```

CAPTURAS DE PANTALLA:


```

arturo@mior-pc:~/Escritorio
arturo@mior-pc:~/Escritorio 80x6
$ ./private-clausule
thread0 suma a[0] / thread0 suma a[1] / thread3 suma a[6] / thread1 suma a[2] /
thread1 suma a[3] / thread2 suma a[4] / thread2 suma a[5] /
* thread 0 suma= 1
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$

```

```

arturo@mior-pc:~/Escritorio
arturo@mior-pc:~/Escritorio 80x18
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./private-clausule
thread0 suma a[0] / thread0 suma a[1] / thread3 suma a[6] / thread2 suma a[4] /
thread2 suma a[5] / thread1 suma a[2] / thread1 suma a[3] /
* thread 0 suma= 32692
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./private-clausule
thread3 suma a[6] / thread0 suma a[0] / thread0 suma a[1] / thread2 suma a[4] /
thread2 suma a[5] / thread1 suma a[2] / thread1 suma a[3] /
* thread 0 suma= 32548
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./private-clausule
thread3 suma a[6] / thread0 suma a[0] / thread0 suma a[1] / thread2 suma a[4] /
thread2 suma a[5] / thread1 suma a[2] / thread1 suma a[3] /
* thread 0 suma= 32729
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$

```

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

RESPUESTA:

El resultado varía entre ejecuciones debido a condiciones de carrera

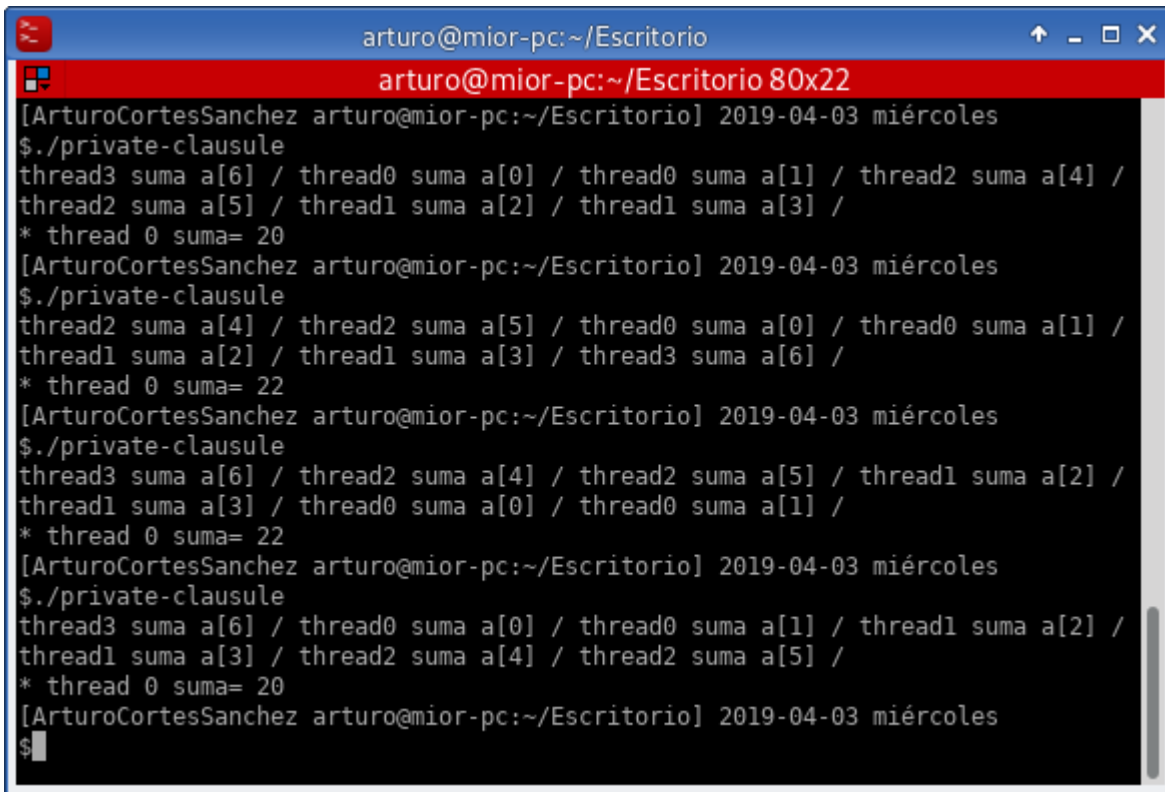
CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```

1  #include <stdio.h>
2  #ifdef _OPENMP
3  #include <omp.h>
4  #else
5  #define omp_get_thread_num() 0
6  #endif
7  int main() {
8      int i, n = 7;
9      int a[n], suma;
10     for (i = 0; i < n; i++)
11         a[i] = i;
12     suma = 1;
13     #pragma omp parallel
14     {
15
16     #pragma omp for
17     for (i = 0; i < n; i++) {
18         suma = suma + a[i];
19         printf("thread%d suma a[%d] / ", omp_get_thread_num(), i);
20     }
21     }
22     printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
23     printf("\n");
24 }

```

CAPTURAS DE PANTALLA:



```

arturo@mior-pc:~/Escritorio
arturo@mior-pc:~/Escritorio 80x22
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./private-clausule
thread3 suma a[6] / thread0 suma a[0] / thread0 suma a[1] / thread2 suma a[4] /
thread2 suma a[5] / thread1 suma a[2] / thread1 suma a[3] /
* thread 0 suma= 20
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./private-clausule
thread2 suma a[4] / thread2 suma a[5] / thread0 suma a[0] / thread0 suma a[1] /
thread1 suma a[2] / thread1 suma a[3] / thread3 suma a[6] /
* thread 0 suma= 22
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./private-clausule
thread3 suma a[6] / thread2 suma a[4] / thread2 suma a[5] / thread1 suma a[2] /
thread1 suma a[3] / thread0 suma a[0] / thread0 suma a[1] /
* thread 0 suma= 22
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./private-clausule
thread3 suma a[6] / thread0 suma a[0] / thread0 suma a[1] / thread1 suma a[2] /
thread1 suma a[3] / thread2 suma a[4] / thread2 suma a[5] /
* thread 0 suma= 20
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$

```

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 (añada capturas de pantalla que muestren lo que ocurre).

RESPUESTA:

Cambiando el número de hebras el resultado varía debido a que varía la suma parcial de cada hebra, y solo se devuelve la suma parcial de la última hebra.

```

arturo@mior-pc: ~/Escritorio
arturo@mior-pc:~/Escritorio 80x39
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./firstlastprivate
thread 3 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
thread 2 suma a[5] suma=9
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5

Fuerade la construcción parallel suma=6
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$export OMP_NUM_THREADS=3
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./firstlastprivate
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
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

Fuerade la construcción parallel suma=11
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$export OMP_NUM_THREADS=2
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./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

Fuerade la construcción parallel suma=15
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$

```

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? (añada una captura de pantalla que muestre lo que ocurre)

RESPUESTA:

CAPTURA CÓDIGO FUENTE: copyprivate-clauseModificado.c

```

1  #include <omp.h>
2  #include <stdio.h>
3  int main() {
4      int n = 9, i, b[n];
5      for (i = 0; i < n; i++)
6          b[i] = -1;
7      #pragma omp parallel
8      {
9          int a;
10     #pragma omp single
11     {
12         printf("\nIntroduce valor de inicialización a : ");
13         scanf("%d", &a);
14         printf("\nSingle ejecutada por el thread%d\n", omp_get_thread_num());
15     }
16     #pragma omp for
17     for (i = 0; i < n; i++)
18         b[i] = a;
19 }
20 printf("Después de la región parallel:\n");
21 for (i = 0; i < n; i++)
22     printf("b[%d] = %d\t", i, b[i]);
23 printf("\n");
24 }

```

CAPTURAS DE PANTALLA:

```

arturo@mior-pc:~/Escritorio
arturo@mior-pc:~/Escritorio 80x23
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$gcc copyprivate.c -o copyprivate -fopenmp
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./copyprivate

Introduce valor de inicialización a : 4

Single ejecutada por el thread0
Después de la región parallel:
b[0] = 4      b[1] = 4      b[2] = 4      b[3] = 4      b[4] = 4      b
[5] = 4 b[6] = 4      b[7] = 4      b[8] = 4
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$gcc copyprivate.c -o copyprivate -fopenmp
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles
$./copyprivate

Introduce valor de inicialización a : 4

Single ejecutada por el thread0
Después de la región parallel:
b[0] = 4      b[1] = 4      b[2] = 4      b[3] = 4      b[4] = 4      b
[5] = 0 b[6] = 0      b[7] = 0      b[8] = 0
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio] 2019-04-03 miércoles

```

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:

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

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:

```

Escritorio : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc Escritorio]$ gcc reduction.c -o reduction -fopenmp
reduction.c:10:1: aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(int argc, char **argv) {
^~~~
[arturo@arturo-pc Escritorio]$ ./reduction 10
Tras parallel suma = 45
[arturo@arturo-pc Escritorio]$ gcc reduction.c -o reduction -fopenmp
reduction.c:10:1: aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(int argc, char **argv) {
^~~~
[arturo@arturo-pc Escritorio]$ ./reduction 10
Tras parallel suma = 55
[arturo@arturo-pc Escritorio]$

```

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:

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

```

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

main(int argc, char **argv) {
    int i, n = 20, a[n], 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
for (i = 0; i < n; i++) {
#pragma omp atomic
suma += a[i];
}
printf("Tras parallel suma = %d\n", suma);
}

```

CAPTURAS DE PANTALLA:

```

Escritorio : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc Escritorio]$ gcc reduction.c -o reduction -fopenmp
reduction.c:10:1: aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(int argc, char **argv) {
^~~~
[arturo@arturo-pc Escritorio]$ ./reduction 10
Tras parallel suma = 55
[arturo@arturo-pc Escritorio]$

```

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"
#include <omp.h>

#define dynamic

#ifndef dynamic
#define n 10
double matriz[n][n];
double v1[n], v2[n];
#endif

double **GenerarMatriz(double **M, int N) {
    M = (double **)malloc(N * sizeof(double *));
    for (int i = 0; i < N; i++)
        M[i] = (double *)malloc(N * sizeof(double));
    return M;
}

void BorrarMatriz(double **M, int N) {
    for (int i = 0; i < N; i++)
        free(M[i]);
    free(M);
}

int main(int argc, char *argv[]) {

#ifdef dynamic
    int n = argc > 1 ? atoi(argv[1]) : 10;
    double **matriz = NULL;
    double *v1, *v2;
    matriz = GenerarMatriz(matriz, n);
    v1 = (double *)malloc(n * sizeof(double));
    v2 = (double *)malloc(n * sizeof(double));
#endif

    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++) {  
    v1[i] = i % 2 ? i * 3 : i * 4;  
    v2[i] = 0;  
}  
double start_time = omp_get_wtime();  
for (int i = 0; i < n; i++)  
    for (int j = 0; j < n; j++)  
        v2[i] += matriz[i][j] * v1[j];  
double ncgt = omp_get_wtime() - start_time;  
  
if (n <= 11) {  
    printf("[ ");  
    for (int i = 0; i < n; i++)  
        printf("%f ", v2[i]);  
    puts("]");  
} else {  
    printf("[ %f ... %f ]\n", v2[0], v2[n - 1]);  
}  
  
printf("tiempo: %f\n", ncgt);  
  
#ifdef dynamic  
    BorrarMatriz(matriz, n);  
    free(v1);  
    free(v2);  
#endif  
}
```

CAPTURAS DE PANTALLA:

```

Practica 3 : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc Practica 3]$ gcc matriz.c -o matriz -fopenmp
[arturo@arturo-pc Practica 3]$ ./matriz 30000
[ 0.000000 ... 944914502550052352.000000 ]
tiempo: 2.621564
[arturo@arturo-pc Practica 3]$

```

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`

```

#include "stdio.h"
#include "stdlib.h"
#include <omp.h>

#define dynamic

#ifndef dynamic

```

```

#define n 10
double matriz[n][n];
double v1[n], v2[n];
#endif

double **GenerarMatriz(double **M, int N) {
M = (double **)malloc(N * sizeof(double *));
for (int i = 0; i < N; i++)
M[i] = (double *)malloc(N * sizeof(double));

return M;
}

void BorrarMatriz(double **M, int N) {
for (int i = 0; i < N; i++)
free(M[i]);
free(M);
}

int main(int argc, char *argv[]) {

#ifdef dynamic
int n = argc > 1 ? atoi(argv[1]) : 10;
double **matriz = NULL;
double *v1, *v2;
matriz = GenerarMatriz(matriz, n);
v1 = (double *)malloc(n * sizeof(double));
v2 = (double *)malloc(n * sizeof(double));
#endif

for (int i = 0; i < n; i++) {
#pragma omp parallel for
for (int j = 0; j < n; j++) {
matriz[i][j] = i * j;
}
}

for (int i = 0; i < n; i++) {
v1[i] = i % 2 ? i * 3 : i * 4;
v2[i] = 0;
}
double start_time = omp_get_wtime();

for (int i = 0; i < n; i++) {

```

```

#pragma omp parallel for
for (int j = 0; j < n; j++) {
#pragma omp atomic
v2[i] += matriz[i][j] * v1[j];
}
}

double ncgt = omp_get_wtime() - start_time;
if (n <= 11) {
printf("[ ");
for (int i = 0; i < n; i++)
printf("%f ", v2[i]);
puts("]");
} else {
printf("[ %f ... %f ]\n", v2[0], v2[n - 1]);
}

printf("tiempo: %f\n", ncgt);

#ifdef dynamic
BorrarMatriz(matriz, n);
free(v1);
free(v2);
#endif
}

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

#include "stdio.h"
#include "stdlib.h"
#include <omp.h>

#define dynamic

#ifndef dynamic
#define n 10
double matriz[n][n];
double v1[n], v2[n];
#endif

double **GenerarMatriz(double **M, int N) {
M = (double **)malloc(N * sizeof(double *));
for (int i = 0; i < N; i++)
M[i] = (double *)malloc(N * sizeof(double));

return M;

```

```

}

void BorrarMatriz(double **M, int N) {
    for (int i = 0; i < N; i++)
        free(M[i]);
    free(M);
}

int main(int argc, char *argv[]) {

#ifdef dynamic
    int n = argc > 1 ? atoi(argv[1]) : 10;
    double **matriz = NULL;
    double *v1, *v2;
    matriz = GenerarMatriz(matriz, n);
    v1 = (double *)malloc(n * sizeof(double));
    v2 = (double *)malloc(n * sizeof(double));
#endif

#pragma omp parallel for
    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++) {
        v1[i] = i % 2 ? i * 3 : i * 4;
        v2[i] = 0;
    }
    double start_time = omp_get_wtime();
#pragma omp parallel for
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
#pragma omp atomic
            v2[i] += matriz[i][j] * v1[j];
        }
    }
    double ncgt = omp_get_wtime() - start_time;
    if (n <= 11) {
        printf("[ ");
        for (int i = 0; i < n; i++)
            printf("%f ", v2[i]);
        puts("]");
    } else {
        printf("[ %f ... %f ]\n", v2[0], v2[n - 1]);
    }
}

```

```

}

printf("tiempo: %f\n", ncgt);

#ifdef dynamic
BorrarMatriz(matriz, n);
free(v1);
free(v2);
#endif
}

```

RESPUESTA:

Para evitar condiciones de carrera he tenido que añadir un `#pragma omp atomic` antes de la suma

CAPTURAS DE PANTALLA:

```

Practica 3: bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[arturo@arturo-pc Practica 3]$ gcc matriz_columnas.c -o matriz_columnas -fopenmp
[arturo@arturo-pc Practica 3]$ ./matriz_columnas 10
[ 0.000000 975.000000 1950.000000 2925.000000 3900.000000 4875.000000 5850.000000 6825.000000 7800.000000 8775.000000 ]
tiempo: 0.000201
[arturo@arturo-pc Practica 3]$ gcc matriz_filas.c -o matriz_filas -fopenmp
[arturo@arturo-pc Practica 3]$ ./matriz_filas 10
[ 0.000000 975.000000 1950.000000 2925.000000 3900.000000 4875.000000 5850.000000 6825.000000 7800.000000 8775.000000 ]
tiempo: 0.000087
[arturo@arturo-pc Practica 3]$

```

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

```
#include "stdio.h"
```



```

#include "stdlib.h"
#include <omp.h>

#define dynamic

#ifndef dynamic
#define n 10
double matriz[n][n];
double v1[n], v2[n];
#endif

double **GenerarMatriz(double **M, int N) {
M = (double **)malloc(N * sizeof(double *));
for (int i = 0; i < N; i++)
M[i] = (double *)malloc(N * sizeof(double));

return M;
}

void BorrarMatriz(double **M, int N) {
for (int i = 0; i < N; i++)
free(M[i]);
free(M);
}

int main(int argc, char *argv[]) {

#ifdef dynamic
int n = argc > 1 ? atoi(argv[1]) : 10;
double **matriz = NULL;
double *v1, *v2;
matriz = GenerarMatriz(matriz, n);
v1 = (double *)malloc(n * sizeof(double));
v2 = (double *)malloc(n * sizeof(double));
#endif

#pragma omp parallel for
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++) {
v1[i] = i % 2 ? i * 3 : i * 4;
v2[i] = 0;
}
double start_time = omp_get_wtime();

for (int i = 0; i < n; i++) {
#pragma omp parallel for reduction(+ : v2[i])
for (int j = 0; j < n; j++) {
v2[i] += matriz[i][j] * v1[j];
}
}
double ncgt = omp_get_wtime() - start_time;
if (n <= 11) {
printf("[ ");
for (int i = 0; i < n; i++)
printf("%f ", v2[i]);
puts("]");
} else {
printf("[ %f ... %f ]\n", v2[0], v2[n - 1]);
}

printf("tiempo: %f\n", ncgt);

#ifdef dynamic
BorrarMatriz(matriz, n);
free(v1);
free(v2);
#endif
}

```

RESPUESTA:

Lo ideal sería que el `#pragma omp parallel for reduction(+ : v2[i])` estuviera en el bucle externo en lugar del interno para así reducir el overhead. Pero por alguna razón que no entiendo al hacerlo el programa da segmentation fault.

CAPTURAS DE PANTALLA:

```

Practica 3: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc Practica 3]$ gcc matriz-reduction.c -o matriz-reduction -fopenmp
[arturo@arturo-pc Practica 3]$ ./matriz_filas 30000
[ 0.000000 ... 944914502550052352.000000 ]
tiempo: 1.504647
[arturo@arturo-pc Practica 3]$

```

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):

```

Practica 3: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc Practica 3]$ ./matriz_filas 30000
[ 0.000000 ... 944914502550052352.000000 ]
tiempo: 1.493784
[arturo@arturo-pc Practica 3]$ ./matriz_columnas 30000
[ 0.000000 ... 944914502550048000.000000 ]
tiempo: 90.451976
[arturo@arturo-pc Practica 3]$ ./matriz-reduction 30000
[ 0.000000 ... 944914502549981952.000000 ]
tiempo: 0.655515
[arturo@arturo-pc Practica 3]$ ./matriz_filas 10
[ 0.000000 975.000000 1950.000000 2925.000000 3900.000000 4875.000000 5850.000000 6825.000000 7800.000000 8775.000000 ]
tiempo: 0.000142
[arturo@arturo-pc Practica 3]$ ./matriz_columnas 10
[ 0.000000 975.000000 1950.000000 2925.000000 3900.000000 4875.000000 5850.000000 6825.000000 7800.000000 8775.000000 ]
tiempo: 0.000303
[arturo@arturo-pc Practica 3]$ ./matriz-reduction 10
[ 0.000000 975.000000 1950.000000 2925.000000 3900.000000 4875.000000 5850.000000 6825.000000 7800.000000 8775.000000 ]
tiempo: 0.000127
[arturo@arturo-pc Practica 3]$

```

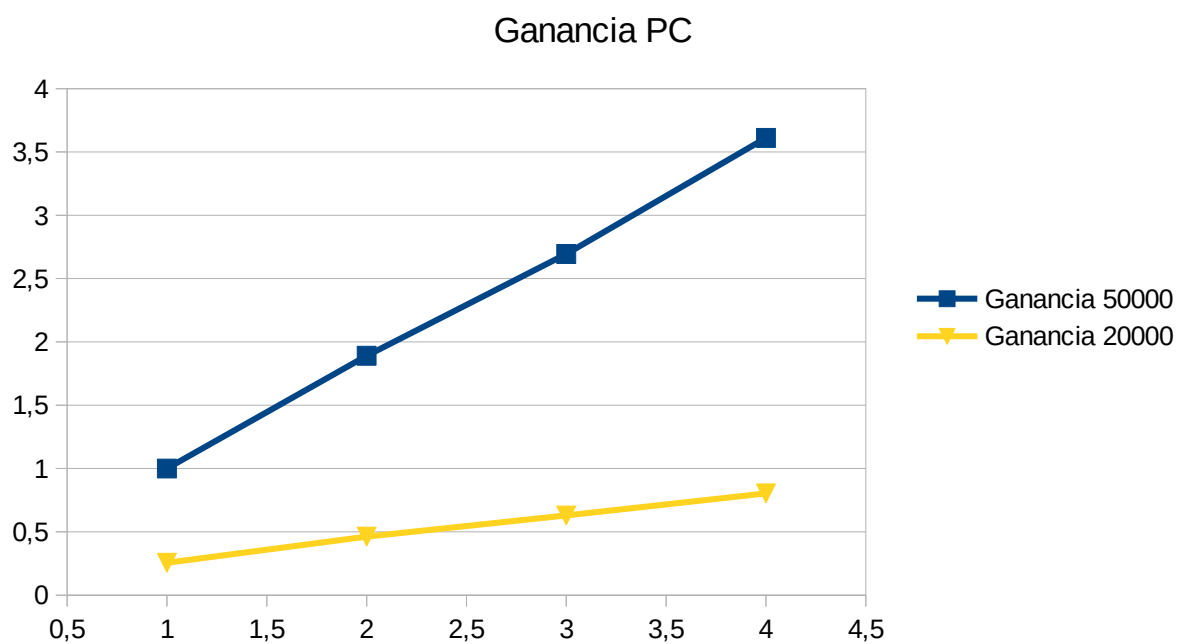
TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia) (para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N:- un N entre 20000 y 100000, y otro entre 5000 y 20000):

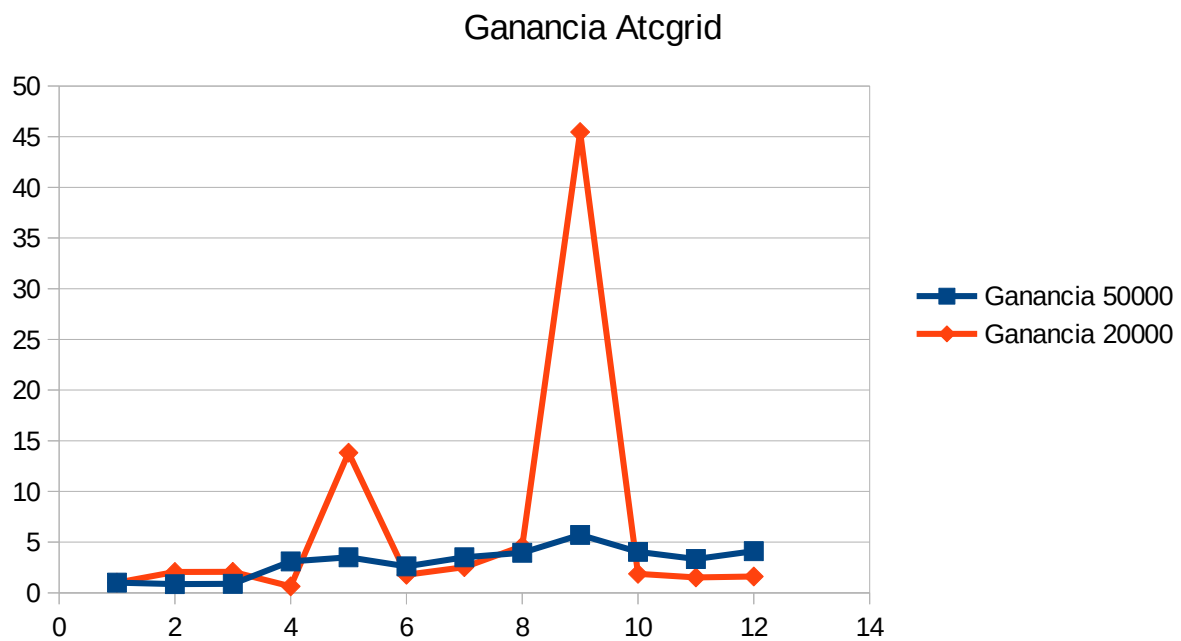
PC N=50000		
Núcleos	Tiempo	Ganancia 50000
1	1,809073	1
2	0,956946	1,89046508371423
3	0,671472	2,69418978006529
4	0,501054	3,61053499223637

PC N=20000		
Núcleos	Tiempo	Ganancia 20000
1	0,291372	0,255958019301786
2	0,161269	0,462450936013741
3	0,118511	0,629300233733577
4	0,092799	0,803661677388765

Atcgrid N=50000		
Núcleos	Tiempo	Ganancia 50000
1	4,022839	1
2	4,710007	0,854104675428296
3	4,550411	0,884060582659456
4	1,297483	3,10049457295394
5	1,145288	3,5125130098281
6	1,537879	2,61583583623939
7	1,149285	3,50029714126609
8	1,019205	3,94703617034846
9	0,704729	5,70834888304582
10	0,995787	4,03985892565378
11	1,206786	3,33351480709919
12	0,979571	4,10673549951969

Atcgrid N=20000		
Núcleos	Tiempo	Ganancia 20000
1	0,74561	1
2	0,361683	2,06150136998421
3	0,360513	2,06819171569403
4	1,169549	0,637519248872856
5	0,053918	13,828591564969
6	0,41949	1,77742020071992
7	0,294989	2,52758577438481
8	0,16235	4,59260856174931
9	0,016403	45,4557093214656
10	0,398886	1,86923080779972
11	0,491225	1,51785841518652
12	0,464218	1,60616348353575





COMENTARIOS SOBRE LOS RESULTADOS: