

Grai2º curso / 2º
cuatr.
Grado Ing. Inform.
Doble Grado Ing.
Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos):

Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

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

RESPUESTA:

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,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:

```
[NombreApellidos antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-10 martes
$gcc -fopenmp shared-clause.c -o shared-clause.out
shared-clause.c: In function 'main':
shared-clause.c:13:13: error: 'n' not specified in enclosing parallel
    #pragma omp parallel for shared(a) default(none)
                        ^
shared-clause.c:13:13: error: enclosing parallel
```

Para solucionar el problema podremos usar la clausula shared n

```
#pragma omp parallel for shared(a,n) default(none)
    for (i=0; i<n; i++) a[i] += i;
```

Y ya nos deja compilar de forma adecuada

```
[NombreApellidos antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-10 martes
$gcc -fopenmp shared-clauseModificado.c -o shared-clauseModificado.out
[NombreApellidos antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-10 martes
$./shared-clauseModificado.out
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:

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;
    for (i=0; i<n; i++)
        a[i] = i;
    suma=1;

    #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");
    return 0;
}
```

CAPTURAS DE PANTALLA:

Antes de modificar el código

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-14 sábado
$gcc -fopenmp private-clauseModificado.c -o private-clauseModificado.out
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-14 sábado
$./private-clauseModificado.out
thread 0 suma a[0] /thread 6 suma a[6] /thread 5 suma a[5] /thread 1 suma a[1] /thread 2 suma a[2] /thread 3 suma a[3] /thread 4 suma a[4] /
* thread 3 suma= 4
* thread 5 suma= 6
* thread 1 suma= 2
* thread 2 suma= 3
* thread 7 suma= 1
* thread 4 suma= 5
* thread 6 suma= 7
* thread 0 suma= 1
```

Después de modificar el código

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-14 sábado
$gcc -fopenmp private-clauseModificado.c -o private-clauseModificado.out
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-14 sábado
$./private-clauseModificado.out
thread 0 suma a[0] /thread 2 suma a[2] /thread 3 suma a[3] /thread 6 suma a[6] /thread 1 suma a[1] /thread 4 suma a[4] /thread 5 suma a[5] /
* thread 4 suma= 4
* thread 3 suma= 3
* thread 0 suma= 0
* thread 7 suma= 0
* thread 2 suma= 2
* thread 5 suma= 5
* thread 1 suma= 1
* thread 6 suma= 6
```

Esto se debe a que al aparecer el `private(suma)` una vez que entra en esa parte, el valor de `suma` se inicia de nuevo a cero, perdiendo el valor que tenía antes y como esta inicializada a uno, si `suma` se inicializa a 1 dentro tiene +1 en todos los valores y si se inicializa a 0 no lo tendrá.

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

RESPUESTA:

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;
    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");
    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-14 sábado
$gcc -fopenmp private-clauseModificado3.c -o private-clauseModificado.out
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-14 sábado
$./private-clauseModificado.out
thread 6 suma a[6] /thread 1 suma a[1] /thread 0 suma a[0] /thread 2 suma a[2] /thread 5 suma a[5] /thread 3 suma a[3] /thread 4 suma a[4] /
* thread 0 suma= 14
* thread 6 suma= 14
* thread 7 suma= 14
* thread 1 suma= 14
* thread 5 suma= 14
* thread 4 suma= 14
* thread 2 suma= 14
* thread 3 suma= 14
```

Esto es debido a que todos los thread comparten la misma variable suma, entonces todos suman en esa variable compartida y a la hora de mostrarla todos muestran el mismo valor.

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:

Siempre que se ejecute va a imprimir por pantalla el ultimo valor calculado en la variable suma. En este caso siempre sera 6, ya que es el ultimo valor guardado en suma por el parallel for. Aunque podria darse el caso de ejecutarse en un unico thread y en ese caso no seria 6, por ejemplo.

CAPTURAS DE PANTALLA:

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-14 sábado
$./firstlastprivate.out
thread 0 suma a[0] suma=0
thread 6 suma a[6] suma=6
thread 1 suma a[1] suma=1
thread 4 suma a[4] suma=4
thread 3 suma a[3] suma=3
thread 5 suma a[5] suma=5
thread 2 suma a[2] suma=2
Fuera de la construcción parallel suma=6
```

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?

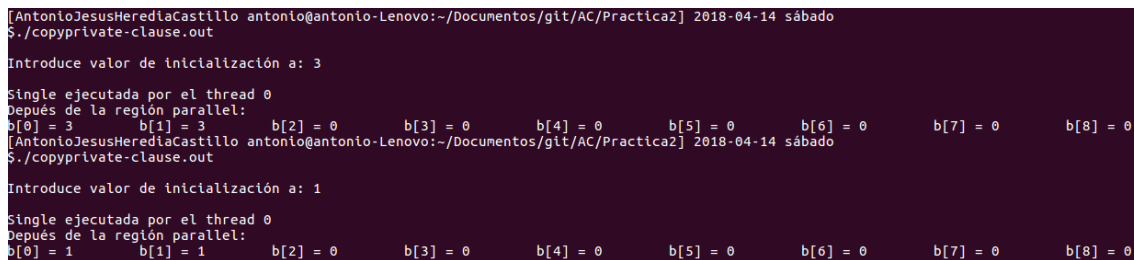
RESPUESTA:

Esto se debe a que como no se inicializa la variable a, y no se copia el valor a todas las variables "a" de todos los threads, estas van a tener valor basura.

CAPTURA CÓDIGO FUENTE: copyprivate-clauseModificado.c

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
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
{
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");
return 0;
```

CAPTURAS DE PANTALLA:


```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-14 sábado
$ ./copyprivate-clause.out
Introduce valor de inicialización a: 3
Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 3      b[1] = 3      b[2] = 0      b[3] = 0      b[4] = 0      b[5] = 0      b[6] = 0      b[7] = 0      b[8] = 0
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2] 2018-04-14 sábado
$ ./copyprivate-clause.out
Introduce valor de inicialización a: 1
Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 1      b[1] = 1      b[2] = 0      b[3] = 0      b[4] = 0      b[5] = 0      b[6] = 0      b[7] = 0      b[8] = 0
```

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

RESPUESTA:

Al estar inicializado a 10, se van añadiendo los demas datos, por lo tanto sera la suma de todos +10.

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);
return 0;
}
```

CAPTURAS DE PANTALLA:

```

Tras 'parallel' suma=16
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2/codigo] 2018-04-14 sábado
$./reduction-clauseModificado.out 4
Tras 'parallel' suma=16
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2/codigo] 2018-04-14 sábado
$./reduction-clauseModificado.out 5
Tras 'parallel' suma=20
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2/codigo] 2018-04-14 sábado
$./reduction-clauseModificado.out 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 usar directivas de trabajo compartido .

RESPUESTA:

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado7.c

```

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;
    int suma_parcial;
    #pragma omp parallel private(suma_parcial)
    {
        suma_parcial = 0;
        #pragma omp for
        for (i=0; i<n; i++)
            suma_parcial += a[i];
        #pragma omp atomic
        suma += suma_parcial;
    }
    printf("Tras 'parallel' suma=%d\n", suma);
    return 0;
}

```

CAPTURAS DE PANTALLA:

```

Tras 'parallel' suma=16
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2/codigo] 2018-04-14 sábado
$gcc -fopenmp reduction-clauseModificado7.c -o reduction-clauseModificado7.out
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica2/codigo] 2018-04-14 sábado
$./reduction-clauseModificado7.out 6
Tras 'parallel' suma=15

```

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 <time.h> // biblioteca donde se encuentra la función clock_gettime()
//#define PRUEBAS
int main(int argc, char **argv) {
    int fil;
    printf("\nIntroduce el tamaño de la matriz cuadrada: ");
    scanf("%d", &fil);
    //Reservamos memoria
    int *v = (int*) malloc(fil*sizeof(int));
    int *resultado = (int*) malloc(fil*sizeof(int));
    int **matriz = (int **)malloc(fil*sizeof(int*));
    for(int i = 0; i < fil; i++){
        matriz[i] = (int*)malloc(fil*sizeof(int));
        if(matriz[i] == NULL) perror("Error: ");
    }
    //Inicializamos la matriz
    for(int f = 0; f < fil; f++){
        for(int c = 0; c < fil; c++){
            matriz[f][c] = 2.0;
        }
    }
    //Inicializamos los vectores tanto v como donde se guardara el resultado
    for(int c = 0; c < fil; c++){
        v[c] = 3.0;
        resultado[c] = 0;
    }
    struct timespec cgt1, cgt2;
    double ncgt; //para tiempo de ejecución
    clock_gettime(CLOCK_REALTIME, &cgt1);
    for(int f = 0; f < fil; f++){
        for(int c = 0; c < fil; c++){
            resultado[f] += v[f]*matriz[f][c];
        }
    }
    clock_gettime(CLOCK_REALTIME, &cgt2);
```

```

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
//imprimimos los datos
#ifdef PRUEBAS
printf("La matriz inicializada es:\n");
for(int f = 0; f < fil; f++){
printf("[");
for(int c = 0; c < fil-1; c++){
printf("%d,",matriz[f][c]);
printf("%d]\n",matriz[f][fil-1]);
}

printf("El vector inicializado es:\n");
for(int c = 0; c < fil; c++){
printf("[%d]\n",v[c]);
}

printf("El resultado de multiplicarlo es:\n");
for(int c = 0; c < fil; c++){
printf("[%d]\n",resultado[c]);
}

printf("El tiempo usado es%f:\n",ncgt);
#else
printf("Primero:%d Ultimo:%d %f \n",resultado[0],resultado[fil-1],ncgt);
#endif
free(matriz);
free(v);
free(resultado);
return 0;
}

```

CAPTURAS DE PANTALLA:

```

$ ./pmv-secuencial.out
Introduce el tamaño de la matriz cuadrada: 8
La matriz inicializada es:
[2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2]
El vector inicializado es:
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
El resultado de multiplicarlo es:
[48]
[48]
[48]
[48]
[48]
[48]
[48]
[48]

```

```

[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos
$ ./pmv-secuencial.out
Introduce el tamaño de la matriz cuadrada: 11
La matriz inicializada es:
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
El vector inicializado es:
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
El resultado de multiplicarlo es:
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
El tiempo usado es0.000002:

```


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>
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>
#define PRUEBAS
int main(int argc, char **argv) {
    int fil;
    printf("\nIntroduce el tamaño de la matriz cuadrada: ");
    scanf("%d", &fil);
    //Reservamos memoria
    int *v= (int*) malloc(fil*sizeof(int));
    int *resultado = (int*) malloc(fil*sizeof(int));
    int **matriz = (int **)malloc(fil*sizeof(int*));
    for(int i = 0; i < fil; i++){
        matriz[i] = (int*)malloc(fil*sizeof(int));
        if(matriz[i] == NULL) perror("Error: ");
    }
    double start, end, diferencia;
    #pragma omp parallel
    {
        //Inicializamos los datos
        #pragma omp for
        for(int f = 0; f < fil; f++){
            v[f] = 3.0;
            resultado[f] = 0;
            for(int c = 0; c < fil; c++)
                matriz[f][c] = 2.0;
        }
        #pragma single
        start = omp_get_wtime();
        #pragma omp for
        for(int f = 0; f < fil; f++){
            for(int c = 0; c < fil; c++)
                resultado[f] += v[f]*matriz[f][c];
        }
        #pragma single
        end = omp_get_wtime();
    }
}
```

```

        diferencia=end-start;
//imprimimos los datos
#ifdef PRUEBAS
printf("La matriz inicializada es:\n");
for(int f = 0; f < fil; f++){
printf("[");
for(int c = 0; c < fil-1; c++)
printf("%d,",matriz[f][c]);
printf("%d\n",matriz[fil-1][fil-1]);
}

printf("El vector inicializado es:\n");
for(int c = 0; c < fil; c++){
printf("[%d]\n",v[c]);
}
printf("El resultado de multiplicarlo es:\n");
for(int c = 0; c < fil; c++){
printf("[%d]\n",resultado[c]);
}
printf("El tiempo usado es%f:\n",diferencia);
#else
printf("Primero:%d Ultimo:%d %f \n",resultado[0],resultado[fil-1],diferencia);
#endif
free(matriz);
free(v);
free(resultado);
return 0;
}

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>
#define PRUEBAS
int main(int argc, char **argv) {
int fil;
printf("\nIntroduce el tamaño de la matriz cuadrada: ");
scanf("%d", &fil );
//Reservamos memoria
int *v= (int*) malloc(fil*sizeof(int));
int *resultado = (int*) malloc(fil*sizeof(int));
int **matriz = (int **)malloc(fil*sizeof(int*));
for(int i = 0; i < fil; i++){
matriz[i] = (int*)malloc(fil*sizeof(int));
if(matriz[i] == NULL) perror("Error: ");
}

double start, end , diferencia;
int suma_parcial;
#pragma omp parallel private(suma_parcial)
{
//Inicializamos los datos
for(int f = 0; f < fil; f++){
v[f] = 3.0;
resultado[f] = 0;
#pragma omp for
for(int c = 0; c < fil; c++)

```

```

matriz[f][c] = 2.0;
}

#pragma single
start = omp_get_wtime();
for(int f = 0; f < fil; f++){
    suma_parcial = 0;
    #pragma omp for
    for(int c = 0; c < fil; c++){
        suma_parcial += v[f]*matriz[f][c];
    }
    #pragma omp atomic
    resultado[f] += suma_parcial;
}
#pragma single
end = omp_get_wtime();
}

diferencia=end-start;
//imprimimos los datos
#ifdef PRUEBAS
printf("La matriz inicializada es:\n");
for(int f = 0; f < fil; f++){
    printf("(");
    for(int c = 0; c < fil-1; c++){
        printf("%d,",matriz[f][c]);
    }
    printf("%d]\n",matriz[fil-1][fil-1]);
}
printf("El vector inicializado es:\n");
for(int c = 0; c < fil; c++){
    printf("[%d]\n",v[c]);
}
printf("El resultado de multiplicarlo es:\n");
for(int c = 0; c < fil; c++){
    printf("[%d]\n",resultado[c]);
}
printf("El tiempo usado es%f:\n",diferencia);
#else
printf("Primero:%d Ultimo:%d %f \n",resultado[0],resultado[fil-1],diferencia);
#endif
free(matriz);
free(v);
free(resultado);
return 0;
}

```

RESPUESTA:

Solo he tenido un error durante la realización de esta practica y es que al hacer la suma por columnas no caí en que se iba a hacer mal. Luego me di cuenta que se podría solucionar de

la misma forma que solucione el ejercicio 7. Cuando puse la misma estructura que en el ejercicio 7 inicializaba a 0 la suma parcial fuera del for de las filas y se me iba acumulando las sumas, lo puse dentro del for y ya funciono correctamente. Se podría haber solucionado todo mas rápido usando reduction.

CAPTURAS DE PANTALLA:

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Prac
$. /pmv-OpenMP-a.out

Introduce el tamaño de la matriz cuadrada: 11
La matriz inicializada es:
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
El vector inicializado es:
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
El resultado de multiplicarlo es:
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
El tiempo usado es0.000001s:

[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Docume
$. /pmv-OpenMP-b.out

Introduce el tamaño de la matriz cuadrada: 11
La matriz inicializada es:
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
El vector inicializado es:
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
El resultado de multiplicarlo es:
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
El tiempo usado es0.000020:
```

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 <time.h> // biblioteca donde se encuentra la función clock_gettime()
#define PRUEBAS
int main(int argc, char **argv) {
    int fil;
    printf("\nIntroduce el tamaño de la matriz cuadrada: ");
    scanf("%d", &fil);
    //Reservamos memoria
    int *v= (int*) malloc(fil*sizeof(int));
    int *resultado = (int*) malloc(fil*sizeof(int));
    int **matriz = (int **)malloc(fil*sizeof(int*));
```

```

for(int i = 0; i < fil; i++){
    matriz[i] = (int*)malloc(fil*sizeof(int));
    if(matriz[i] == NULL) perror("Error: ");
}

    struct timespec cgt1,cgt2;
double ncgt; //para tiempo de ejecución
    //Inicializamos los datos
    for(int f = 0; f < fil; f++){
        v[f] = 3.0;
        resultado[f] = 0;
        #pragma omp parallel for
            for(int c = 0; c < fil; c++)
                matriz[f][c] = 2.0;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for(int f = 0; f < fil; f++){
        int suma = 0;
        #pragma omp parallel for reduction(+:suma)
            for(int c = 0; c < fil; c++)
                suma += v[f]*matriz[f][c];
        resultado[f] = suma;
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
//imprimimos los datos
#ifdef PRUEBAS
printf("La matriz inicializada es:\n");
for(int f = 0; f < fil; f++){
    printf("[");
    for(int c = 0; c < fil-1; c++)
        printf("%d,",matriz[f][c]);
    printf("%d]\n",matriz[f][fil-1]);
}
printf("El vector inicializado es:\n");
for(int c = 0; c < fil; c++){
    printf("[%d]\n",v[c]);
}
printf("El resultado de multiplicarlo es:\n");
for(int c = 0; c < fil; c++){
    printf("[%d]\n",resultado[c]);
}
printf("El tiempo usado es%f:\n",ncgt);
#else
printf("Primero:%d Ultimo:%d %f \n",resultado[0],resultado[fil-1],ncgt);
#endif
free(matriz);
free(v);

```

```
free(resultado);
return 0;
}
```

RESPUESTA:

El problema que tenía era que tenía el parallel como un bloque para todo. Es decir un solo parallel, entonces si declaraba el int suma me decía que tenía que estar fuera, porque dentro lo marcaba como privado, entonces tuve que poner la region parallel solo para el for. Y ya funcionaba.

CAPTURAS DE PANTALLA:

```
[AntonioJesusHerediaCastillo antonio@Linux-Antonio:
~/git/AC/Practica2/codigo] 2018-04-16 lunes
$./pmv-OpenMP-reduction.out

Introduce el tamaño de la matriz cuadrada: 11
La matriz inicializada es:
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
[2,2,2,2,2,2,2,2,2,2,2]
El vector inicializado es:
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
[3]
El resultado de multiplicarlo es:
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
[66]
El tiempo usado es0.000010:
```

11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar -O2 al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

CAPTURAS DE PANTALLA (que justifique el código elegido):

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

PC local							
N=15000							
	A	Tiempo a		B	Tiempo b		Reduction
							Tiempo c
	1	0,124584	0,1107822982	0,093166	0,0852258486	0,09482	0,0866078442
	2	0,063529	0,0615731594	0,053269	0,0518870153	0,060229	0,0584682577
	3	0,046444	0,0457359466	0,048386	0,0476179854	0,054027	0,05307124
	4	0,042749	0,0422969618	0,04709	0,0465420833	0,05734	0,0565296475
	5	0,037512	0,0372326656	0,15581	0,1511013788	0,260509	0,2476081687
	6	0,036039	0,0358238242	0,235407	0,2265196161	0,430665	0,4018231396
	7	0,031493	0,0313519476	0,302765	0,2902126797	0,508701	0,4742374214
	8	0,027877	0,0277801964	0,138605	0,1362444792	0,294643	0,2841766668
ATCGRID							
N=15000							
	A			B	Reduction		
	1	0,304418	0,2333745778	0,245246	0,1969458244	0,225315	0,183883328
	2	0,155587	0,1443569663	0,149139	0,1387895339	0,224123	0,2015383142
	3	0,102462	0,099078087	0,108359	0,1045815493	0,23282	0,2160528579
	4	0,077506	0,0760327514	0,086069	0,0842560417	0,2254	0,2133762484
	5	0,062198	0,0614337882	0,072419	0,0713850729	0,22536	0,2156406449
	6	0,052875	0,0524131095	0,066658	0,0659255887	0,225371	0,2172121147
	7	0,046662	0,0463530108	0,06358	0,06300771	0,224232	0,217272092
	8	0,041666	0,0414501174	0,061077	0,0606142331	0,225342	0,219168516
	9	0,04058	0,0403978506	0,059556	0,0591644889	0,225392	0,2198852905
	10	0,036928	0,036792134	0,060602	0,060236952	0,225351	0,2203846108
	11	0,060001	0,0596754919	0,067917	0,0675002351	0,225393	0,2208673674
	12	0,05099	0,0507742517	0,067053	0,0666804066	0,22536	0,2212057559

PC local

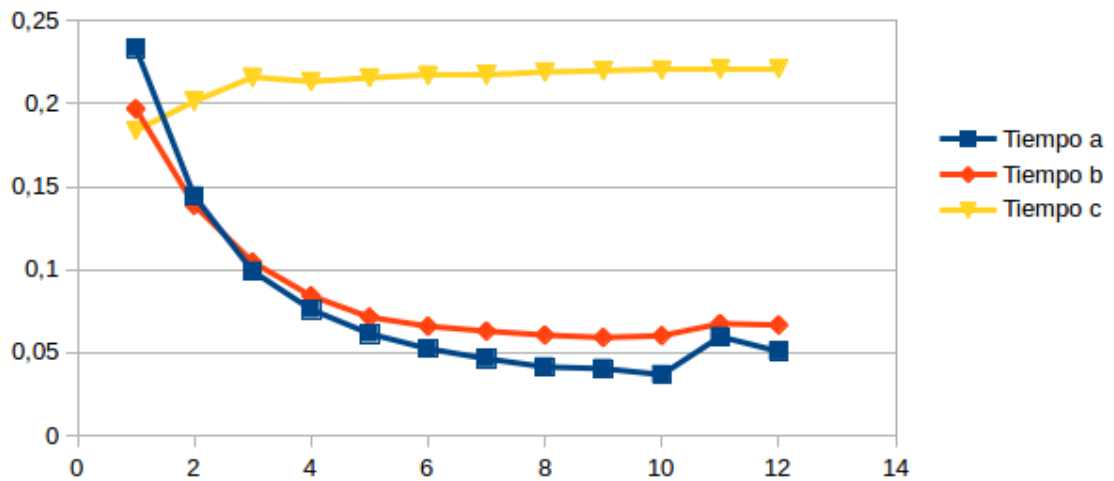
N=50000	A	Tiempo a	B	Tiempo b	Reduction	Tiempo c
1	3,136375	0,7582424224	1,027478	0,436281849	1,035668	0,5087607606
2	1,563141	0,8773949726	0,564981	0,343677029	0,584281	0,4521807033
3	1,066194	0,7866279868	0,502307	0,3065544408	0,527357	0,4485145677
4	0,816528	0,6781050582	0,488137	0,2838245855	0,523886	0,463217685
5	0,957094	0,8033228954	0,936054	0,4323120604	1,284629	1,0220404418
6	0,820251	0,7216018883	1,197565	0,4502814063	1,857695	1,4185037724
7	0,825164	0,7381504081	0,786397	0,3807551431	1,362696	1,1406455526
8	0,798413	0,7259609205	0,764533	0,3723739311	1,217567	1,0567361214

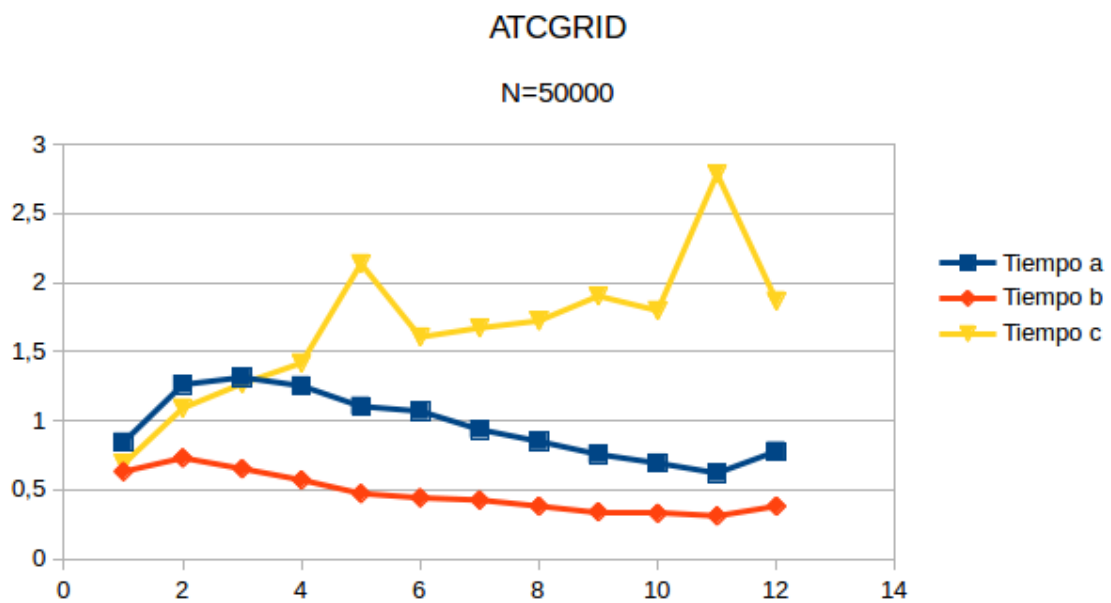
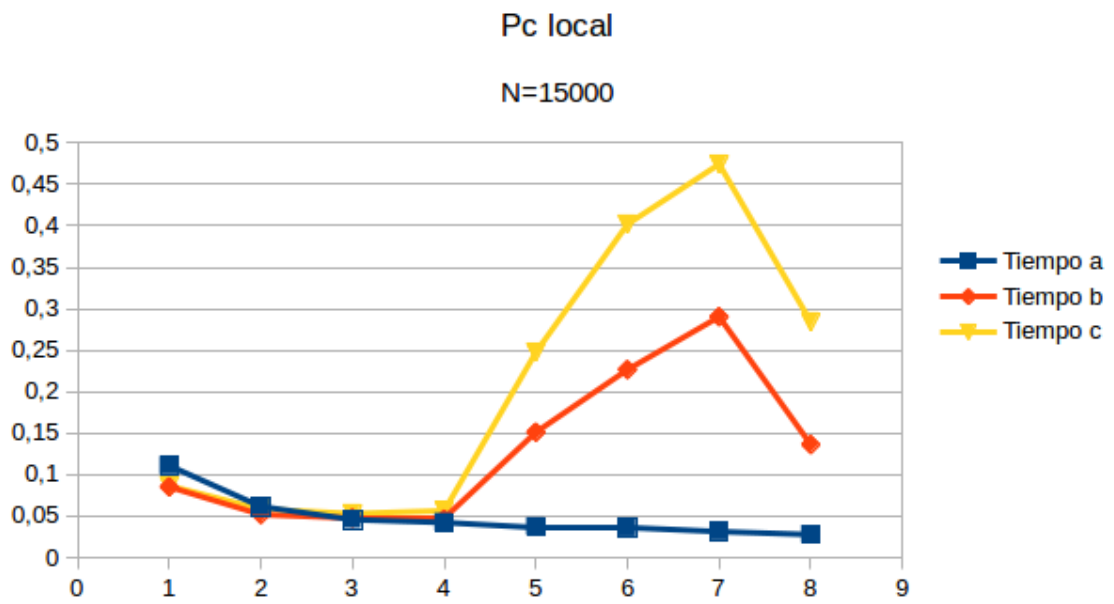
ATCGRID

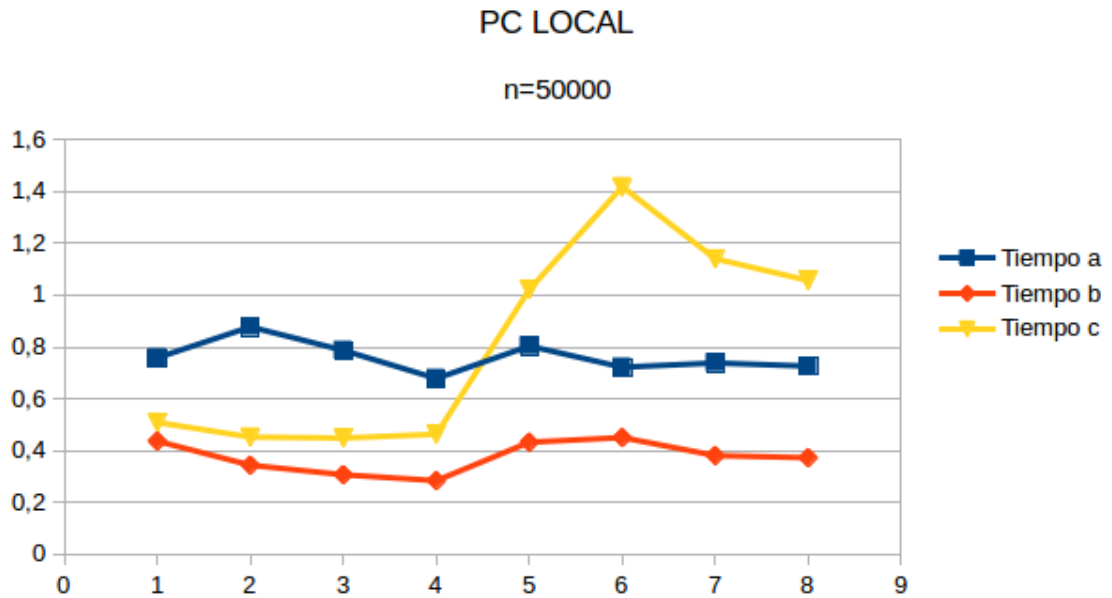
N=50000	A	B	Reduction
1	5,428371	0,8444395944	2,512668
2	3,421325	1,2621729928	1,737136
3	2,336499	1,3135010425	1,297963
4	1,824752	1,2531019346	1,051639
5	1,416496	1,1037924749	0,828508
6	1,30118	1,0692901695	0,754309
7	1,080497	0,9360165594	0,781675
8	0,951894	0,8506749521	0,692074
9	0,825874	0,7564585094	0,621615
10	0,744845	0,6932114889	0,629352
11	0,658933	0,6216917963	0,624317
12	0,833322	0,77921087	0,748635

ATCGRID

15000







COMENTARIOS SOBRE LOS RESULTADOS:

Pues según el tamaño del problema elegiría un programa u otro ya que para tamaños algo mas pequeños parece funcionar mejor la opcion a en cambio para tamaños mas grandes parece funcionar mejor la opcion b. En cualquier caso no usaria reduction.