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:
Lo que ocurre es que al poner default(none), tenemos que definir los tipos de todas las variables declaradas antes de la clausula en cuestión.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```c
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif

int main()
{
    int i, n = 7;
    int a[n];

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

    #pragma omp parallel for default(none) private(n) shared(a)
        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:

```
$ gcc-7 -O2 -fopenmp -o shared-clauseModificado shared-clauseModificado.c
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:14:13: error: 'n' not specified in enclosing 'parallel'
     #pragma omp parallel for default(none) shared(a)
             ^~~
shared-clauseModificado.c:14:13: error: enclosing 'parallel'
```

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 como la clausula private define a la variable suma como privada, la inicialización de fuera de la directiva parallel no la toma como valida.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

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

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

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

    #pragma omp parallel private(suma)
    {
        suma=100;

        #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:

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2ºCuatrimestre/Arquitectura de computadores/AC---Practicas/Practica 2] 2018-04-13 viernes
$./private-clauseModificado
thread 3 suma a[6] / thread 2 suma a[4] / thread 0 suma a[0] / thread 1 suma a[2] / thread 2 suma a[5] / thread 0 suma a[1] / thread 1 suma a[3] /
* thread 3 suma= 106
* thread 2 suma= 109
* thread 0 suma= 101
* thread 1 suma= 105
```

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

RESPUESTA:
Como la variable suma esta declarada fuera de la directiva parallel, es compartida para todas las hebras. Esto significa que todas las hebras están escribiendo sobre la misma variable y por ello el resultado se esta sobrescribiendo.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

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

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

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

    #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:

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2ºCuatrimestre/Arquitectura de computadores/AC---Practicas/Practica 2] 2018-04-13 viernes
$./private-clauseModificado
thread 2 suma a[4] / thread 3 suma a[6] / thread 0 suma a[0] / thread 1 suma a[2] / thread 2 suma a[5] / thread 0 suma a[1] / thread 1 suma a[3] /
* thread 2 suma= 11
* thread 3 suma= 11
* thread 0 suma= 11
* thread 1 suma= 11
```

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:
La razón por la que siempre imprime 6 es por que coge el último valor del for de forma secuencial, y no paralela.

CAPTURAS DE PANTALLA:

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2ºCuatrimestre/Arquitectura de computadores/AC---Practicas/Practica 2] 2018-04-13 viernes
$./firstlastprivate
 thread 3 suma a[6] suma=6
 thread 0 suma a[0] suma=0
 thread 2 suma a[4] suma=4
 thread 1 suma a[2] suma=2
 thread 0 suma a[1] suma=1
 thread 2 suma a[5] suma=9
 thread 1 suma a[3] suma=5

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:
Lo que observamos es que solo se inicializa la variable *a* para el thread que ejecuta la directiva single, y para los demás contiene basura.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```c
#include <stdio.h>
#include <omp.h>

int main() {
    int n = 9, i, b[n];

    for (i=0; i<n; i++) b[i] = -1;

    #pragma omp parallel
    {
        int a;

        #pragma omp single
        {
            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:

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC---Practicas/Practica 2] 2018-04-14 sábado
$./copyprivate-clauseModificado

Introduce valor de inicialización a: 7

Single ejecutada por el thread 3
Depués de la región parallel:
b[0] = 1        b[1] = 1        b[2] = 1        b[3] = 0        b[4] = 0        b[5] = 32652    b[6] = 32652    b[7] = 7        b[8] = 7
```

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

RESPUESTA:
Imprime el mismo resultado sumándole 10, puesto que es el valor que le damos ahora al inicializarlo.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

int main(int argc, char **argv) {
    int i, n=20, a[n],suma=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:

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC---Practicas/Practica 2] 2018-04-14 sábado
$./reduction-clause 5
Tras 'parallel' suma=10
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC---Practicas/Practica 2] 2018-04-14 sábado
$./reduction-clauseModificado 5
Tras 'parallel' suma=20
```

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:

Usamos una variable local para cada uno de los threads y una variable donde cada una ira guardando su suma local al pasar por la directiva atomic.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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

int main(int argc, char **argv) {
    int i, n=20, a[n],suma=0;

    if(argc < 2) {
        fprintf(stderr,"Falta iteraciones\n");
        exit(-1);
    }

    n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d",n);}

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

    #pragma omp parallel
    {
        int sumaLocal=0;

        #pragma omp for
            for (i=0; i<n; i++) sumaLocal += a[i];

        #pragma omp atomic
            suma += sumaLocal;
    }

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

CAPTURAS DE PANTALLA:

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC---Practicas/Practica 2] 2018-04-14 sábado
$./reduction-clauseModificado7 5
Tras 'parallel' suma=10
```

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):

$$v2 = M \,!\, v1;\ \ v2(i) = \mathop{!}_{k\,!\,0}^{N\,!\,1} M(i,k) \,!\, v(k),\ \ i = 0,\ldots N-1$$

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`

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

//#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
                       // globales (su longitud no estará limitada por el ...
                       // tamaño de la pila del programa)
#define VECTOR_DYNAMIC  // descomentar para que los vectores sean variables ...
                          // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 15000
double v1[MAX], v2[MAX], m[MAX][MAX];
#endif

int main(int argc, char ** argv)
{
    if(argc < 2){
        printf("Falta el número de filas/columnas. \n");
        exit(-1);
    }

    int n = atoi(argv[1]);

    #ifdef VECTOR_GLOBAL
    if (n>MAX) n=MAX;
    #endif
    #ifdef VECTOR_DYNAMIC
    double *v1, *v2, **m;
    v1 = (double*) malloc(n*sizeof(double));
    v2 = (double*) malloc(n*sizeof(double));
    m = (double**) malloc(n*sizeof(double*));
    for(int i = 0; i < n; i++) m[i] = (double*)malloc(n*sizeof(double));
    if ( (v1==NULL) || (v2==NULL) || (m==NULL) ){
        printf("Error en la reserva de espacio para los vectores y matriz\n");
        exit(-2);
    }
    #endif

    int i, j;
    struct timespec cgt1, cgt2; double ncgt; //para tiempo de ejecución
```

```
    for(i=0; i<n; i++){
        v1[i] = i;
        v2[i] = 0;
    }

    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            m[i][j] = i*j;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            v2[i] += m[i][j]*v1[j];
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

    ncgt=(double)(cgt2.tv_sec-cgt1.tv_sec)+ (double)((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    if(n < 16){
        for(i=0; i<n; i++)
            printf("La suma de la componente %d es: %f \n",i, v2[i]);
    }else{
        printf("El componente 0 del vector resultante es: %f \n",v2[0]);
        printf("El componente %d del vector resultante es: %f \n", n-1, v2[n-1]);
    }

    printf("Tiempo(seg.):%11.9f \n", ncgt);

    #ifdef VECTOR_DYNAMIC
    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
    free(m); // libera el espacio reservado para m
    #endif

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi072223:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 2] 2018-04-17 martes
$./pmv-secuencial 8
La suma de la componente 0 es: 0.000000
La suma de la componente 1 es: 140.000000
La suma de la componente 2 es: 280.000000
La suma de la componente 3 es: 420.000000
La suma de la componente 4 es: 560.000000
La suma de la componente 5 es: 700.000000
La suma de la componente 6 es: 840.000000
La suma de la componente 7 es: 980.000000
Tiempo(seg.):0.000000000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi072223:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 2] 2018-04-17 martes
$./pmv-secuencial 11
La suma de la componente 0 es: 0.000000
La suma de la componente 1 es: 385.000000
La suma de la componente 2 es: 770.000000
La suma de la componente 3 es: 1155.000000
La suma de la componente 4 es: 1540.000000
La suma de la componente 5 es: 1925.000000
La suma de la componente 6 es: 2310.000000
La suma de la componente 7 es: 2695.000000
La suma de la componente 8 es: 3080.000000
La suma de la componente 9 es: 3465.000000
La suma de la componente 10 es: 3850.000000
Tiempo(seg.):0.000000000
```

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`

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

//#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
                       // globales (su longitud no estará limitada por el ...
                       // tamaño de la pila del programa)
#define VECTOR_DYNAMIC  // descomentar para que los vectores sean variables ...
                          // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 15000
double v1[MAX], v2[MAX], m[MAX][MAX];
#endif

int main(int argc, char ** argv)
{
    if(argc < 2){
        printf("Falta el número de filas/columnas. \n");
        exit(-1);
    }

    int n = atoi(argv[1]);

    #ifdef VECTOR_GLOBAL
    if (n>MAX) n=MAX;
    #endif
    #ifdef VECTOR_DYNAMIC
    double *v1, *v2, **m;
    v1 = (double*) malloc(n*sizeof(double));
    v2 = (double*) malloc(n*sizeof(double));
    m = (double**) malloc(n*sizeof(double*));
    for(int i = 0; i < n; i++) m[i] = (double*)malloc(n*sizeof(double));
    if ( (v1==NULL) || (v2==NULL) || (m==NULL) ){
        printf("Error en la reserva de espacio para los vectores y matriz\n");
        exit(-2);
    }
    #endif
```

```
    int i, j;
    double start, end, tTotal; //para tiempo de ejecución

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

        #pragma omp single
        start = omp_get_wtime();

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

        #pragma omp single
        end = omp_get_wtime( );
    }

    tTotal = end - start;

    if(n < 16){
        for(i=0; i<n; i++)
            printf("La suma de la componente %d es: %f \n",i, v2[i]);
    }else{
        printf("El componente 0 del vector resultante es: %f \n",v2[0]);
        printf("El componente %d del vector resultante es: %f \n", n-1, v2[n-1]);
    }
    printf("Tiempo(seg.):%11.9f \n", tTotal);

    #ifdef VECTOR_DYNAMIC
    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
    for (i=0; i<n; i++)
        free(m[i]);
    free(m);
    #endif

    return 0;
}
```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

No pongo captura de la creación de variables y demás por encima de la siguiente captura para ahorrar espacio, ya que es igual que en pmv-OpenMP-a.c .

```
    int i, j;
    double tTotal; //para tiempo de ejecución

    omp_set_num_threads(n);

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

    double start = omp_get_wtime();

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

    double end = omp_get_wtime( );
    tTotal = end - start;

    if(n < 16){
        for(i=0; i<n; i++)
            printf("La suma de la componente %d es: %f \n",i, v2[i]);
    }else{
        printf("El componente 0 del vector resultante es: %f \n",v2[0]);
        printf("El componente %d del vector resultante es: %f \n", n-1, v2[n-1]);
    }
    printf("Tiempo(seg.):%11.9f \n", tTotal);

    #ifdef VECTOR_DYNAMIC
    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
    for (i=0; i<n; i++)
        free(m[i]);
    free(m);
    #endif

    return 0;
}
```

RESPUESTA:

CAPTURAS DE PANTALLA:

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi072223:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-a 8
La suma de la componente 0 es: 0.000000
La suma de la componente 1 es: 140.000000
La suma de la componente 2 es: 280.000000
La suma de la componente 3 es: 420.000000
La suma de la componente 4 es: 560.000000
La suma de la componente 5 es: 700.000000
La suma de la componente 6 es: 840.000000
La suma de la componente 7 es: 980.000000
Tiempo(seg.):0.000008000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi072223:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-a 11
La suma de la componente 0 es: 0.000000
La suma de la componente 1 es: 385.000000
La suma de la componente 2 es: 770.000000
La suma de la componente 3 es: 1155.000000
La suma de la componente 4 es: 1540.000000
La suma de la componente 5 es: 1925.000000
La suma de la componente 6 es: 2310.000000
La suma de la componente 7 es: 2695.000000
La suma de la componente 8 es: 3080.000000
La suma de la componente 9 es: 3465.000000
La suma de la componente 10 es: 3850.000000
Tiempo(seg.):0.000007000
```

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi072223:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-b 8
La suma de la componente 0 es: 0.000000
La suma de la componente 1 es: 140.000000
La suma de la componente 2 es: 280.000000
La suma de la componente 3 es: 420.000000
La suma de la componente 4 es: 560.000000
La suma de la componente 5 es: 700.000000
La suma de la componente 6 es: 840.000000
La suma de la componente 7 es: 980.000000
Tiempo(seg.):0.000008000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi072223:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-b 11
La suma de la componente 0 es: 0.000000
La suma de la componente 1 es: 385.000000
La suma de la componente 2 es: 770.000000
La suma de la componente 3 es: 1155.000000
La suma de la componente 4 es: 1540.000000
La suma de la componente 5 es: 1925.000000
La suma de la componente 6 es: 2310.000000
La suma de la componente 7 es: 2695.000000
La suma de la componente 8 es: 3080.000000
La suma de la componente 9 es: 3465.000000
La suma de la componente 10 es: 3850.000000
Tiempo(seg.):0.000009000
```

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`

```
    int i, j;
    double tTotal; //para tiempo de ejecución

    omp_set_num_threads(n);

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

    double start = omp_get_wtime();

    for(i=0; i<n; i++){
        double suma = 0.0;
        #pragma omp parallel for reduction(+:suma)
            for(j=0; j<n; j++){
                suma += m[i][j]*v1[j];
            }
        v2[i] = suma;
    }

    double end = omp_get_wtime( );
    tTotal = end - start;

    if(n < 16){
        for(i=0; i<n; i++)
            printf("La suma de la componente %d es: %f \n",i, v2[i]);
    }else{
        printf("El componente 0 del vector resultante es: %f \n",v2[0]);
        printf("El componente %d del vector resultante es: %f \n", n-1, v2[n-1]);
    }
    printf("Tiempo(seg.):%11.9f \n", tTotal);


    #ifdef VECTOR_DYNAMIC
    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
    for (i=0; i<n; i++)
        free(m[i]);
    free(m);
    #endif

    return 0;
}
```

RESPUESTA:
Tuve el siguiente problema que solucione quitarndo la directiva parallel de delante del primer for.

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi072223:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 2] 2018-04-17 martes
$gcc-7 -O2 -fopenmp -o pmv-OpenMP-reduction pmv-OpenMP-reduction.c
pmv-OpenMP-reduction.c: In function 'main':
pmv-OpenMP-reduction.c:60:17: error: reduction variable 'suma' is private in outer context
        #pragma omp for reduction(+:suma)
                ^~~
```

CAPTURAS DE PANTALLA:

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi072223:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-reduction 8
La suma de la componente 0 es: 0.000000
La suma de la componente 1 es: 140.000000
La suma de la componente 2 es: 280.000000
La suma de la componente 3 es: 420.000000
La suma de la componente 4 es: 560.000000
La suma de la componente 5 es: 700.000000
La suma de la componente 6 es: 840.000000
La suma de la componente 7 es: 980.000000
Tiempo(seg.):0.000940000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi072223:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-reduction 11
La suma de la componente 0 es: 0.000000
La suma de la componente 1 es: 385.000000
La suma de la componente 2 es: 770.000000
La suma de la componente 3 es: 1155.000000
La suma de la componente 4 es: 1540.000000
La suma de la componente 5 es: 1925.000000
La suma de la componente 6 es: 2310.000000
La suma de la componente 7 es: 2695.000000
La suma de la componente 8 es: 3080.000000
La suma de la componente 9 es: 3465.000000
La suma de la componente 10 es: 3850.000000
Tiempo(seg.):0.001403000
```

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.

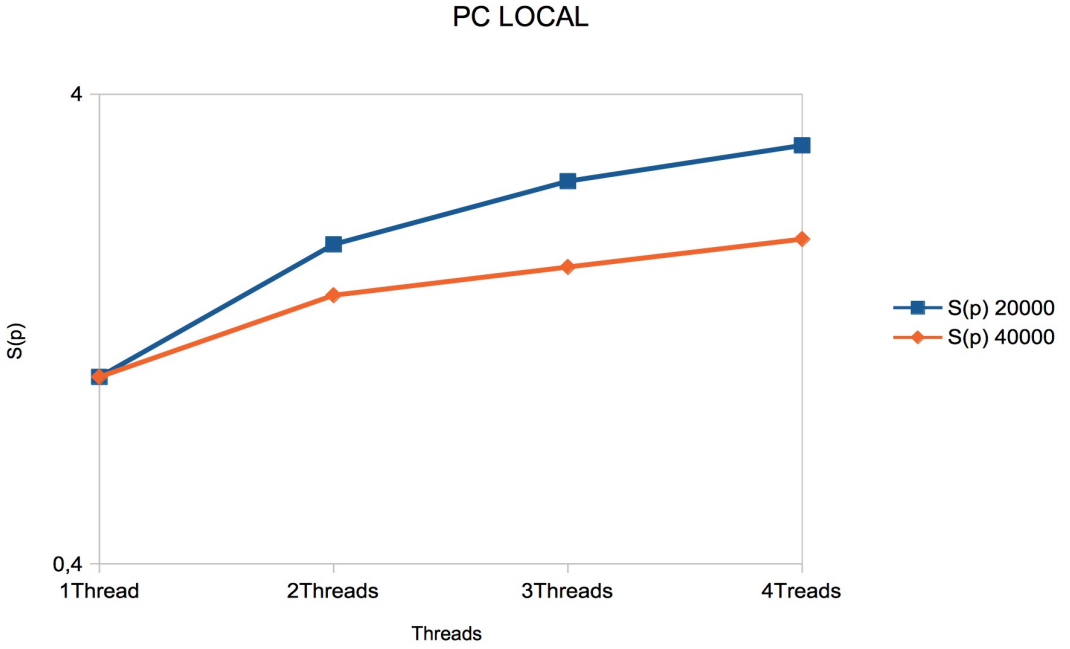
CAPTURAS DE PANTALLA (que justifique el código elegido):

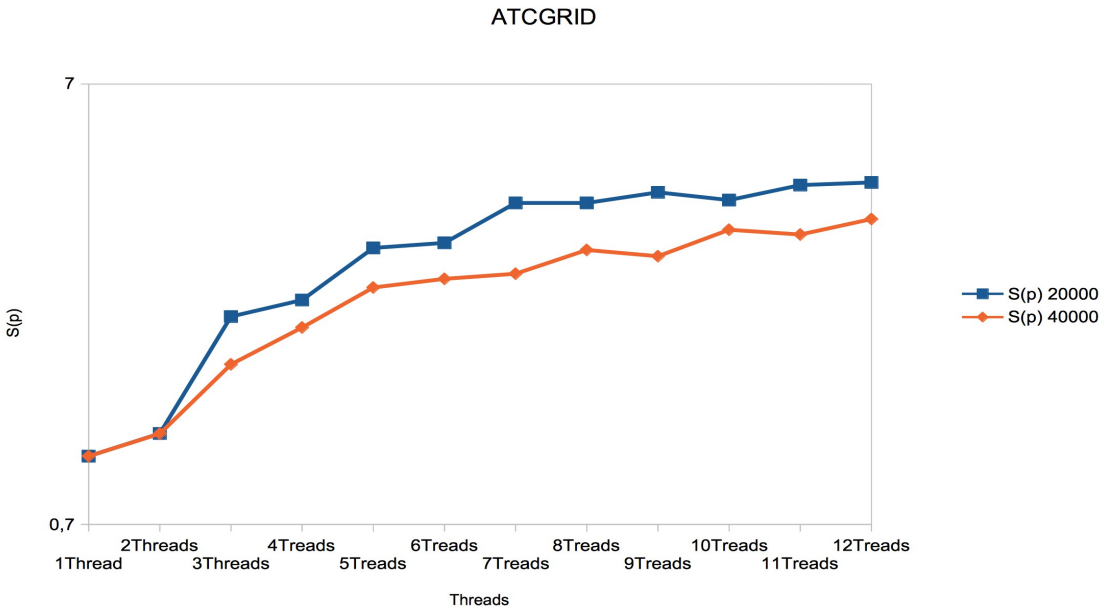
```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@vpn-s241225:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC--
-Practicas/Practica 2] 2018-04-17 martes
$export OMP_NUM_THREADS=1
[AlejandroGarciaVallecillo alejandrogarciavallecillo@vpn-s241225:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC--
-Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-a 20000
El componente 0 del vector resultante es: 0.000000
El componente 19999 del vector resultante es: 53326666933336912.000000
Tiempo(seg.):4.264168000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@vpn-s241225:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC--
-Practicas/Practica 2] 2018-04-17 martes
$export OMP_NUM_THREADS=2
[AlejandroGarciaVallecillo alejandrogarciavallecillo@vpn-s241225:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC--
-Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-a 20000
El componente 0 del vector resultante es: 0.000000
El componente 19999 del vector resultante es: 53326666933336912.000000
Tiempo(seg.):0.278482000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@vpn-s241225:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC--
-Practicas/Practica 2] 2018-04-17 martes
$export OMP_NUM_THREADS=3
[AlejandroGarciaVallecillo alejandrogarciavallecillo@vpn-s241225:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC--
-Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-a 20000
El componente 0 del vector resultante es: 0.000000
El componente 19999 del vector resultante es: 53326666933336912.000000
Tiempo(seg.):0.204325000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@vpn-s241225:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC--
-Practicas/Practica 2] 2018-04-17 martes
$export OMP_NUM_THREADS=4
[AlejandroGarciaVallecillo alejandrogarciavallecillo@vpn-s241225:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC--
-Practicas/Practica 2] 2018-04-17 martes
$./pmv-OpenMP-a 20000
El componente 0 del vector resultante es: 0.000000
El componente 19999 del vector resultante es: 53326666933336912.000000
Tiempo(seg.):0.171337000
```

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 / threads	**1Thread**	**2Threads**	**3Threads**	**4Treads**
20000	0,533444	0,278482	0,204325	0,171337
40000	57,042128	38,217346	33,278868	29,016692
S(p) 20000	1	1,915542118	2,610762266	3,113419752
S(p) 40000	1	1,492571672	1,714064553	1,965838422

ATCGRID												
N / threads	1Thread	2Threads	3Threads	4Treads	5Treads	6Treads	7Treads	8Treads	9Treads	10Treads	11Treads	12Treads
20000	0,554365863	0,492307264	0,267103055	0,244960045	0,186574972	0,181699349	0,147462713	0,147462713	0,13949461	0,145303163	0,134352415	0,132479058
40000	2,164932563	1,924703809	1,339424147	1,104606121	0,896186617	0,856357194	0,833865422	0,736242577	0,760819582	0,662622147	0,679319681	0,626610882
S(p) 20000	1	1,126056639	2,075475561	2,263086876	2,971276678	3,051006325	3,759362972	3,759362972	3,97410239	3,815236032	4,126206909	4,184554686
S(p) 40000	1	1,124813362	1,616315913	1,959913603	2,415716238	2,528071905	2,596261346	2,940515301	2,845526869	3,26722035	3,186912765	3,454987178





COMENTARIOS SOBRE LOS RESULTADOS:

Según he ido comprobando al hacer pruebas en mi pc local, el código más rápido es el que paraleliza el bucle que recorre las filas. Lo más lógico sería que el más rápido fuera el que tiene la directiva reductión, pero no es así. Supongo que sucede por el hecho de tener que crear la variable suma y tener que hacer todos los pasos al compilar.