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): Raúl Castro Moreno
Grupo de prácticas y profesor de prácticas: B2-Cristián
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:

   Nos da error diciendo que n no está especificado en el parallel. Para solucionarlo basta con meter n dentro de la clausula de shared.

   CAPTURAS DE PANTALLA:

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer1] 2020-04-10 viernes
$g++ -fopenmp -o shared-clause shared-clause.c
shared-clause.c: In function ‘int main()’:
shared-clause.c:15:2: error: ‘n’ not specified in enclosing ‘parallel’
  for(i=0;i<n;i++)
  ^~~
shared-clause.c:14:10: error: enclosing ‘parallel’
  #pragma omp parallel for shared(a) default(none)
          ^~~
```

```
#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)
        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]);
}
```

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 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]);
}
```

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:

He usado el numero 7 para inicializar suma en los 2 casos.

Por los resultados que podemos observar, la razón por la que sucede lo de la ejecución con la inicialización de suma fuera del parallel, puede ser que no estamos inicializando de forma privada suma para todas las hebras, asi que solo la tiene inicializada correctamente la hebra master (la hebra 0).

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`
CAPTURAS DE PANTALLA:

```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 private(suma)
        {
        suma=7;
        #pragma omp for
        for(i=0;i<n;i++)
        {
           suma = suma + a[i];
           printf("Hebra %d suma a[%d]/",omp_get_thread_num(),i);
        }

        printf("\n* Hebra %d suma= %d", omp_get_thread_num(), suma);
        }
 printf("\n");
}
```

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer2] 2020-04-10 viernes
$g++ -O2 -fopenmp -o private-clauseModificado private-clauseModificado.c
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer2] 2020-04-10 viernes
$./private-clauseModificado
Hebra 6 suma a[6]/Hebra 0 suma a[0]/Hebra 1 suma a[1]/Hebra 3 suma a[3]/Hebra 4 suma a[4]/Hebra 5 suma a[5]/Hebra 2 suma a[2]/
* Hebra 1 suma= 8
* Hebra 5 suma= 12
* Hebra 6 suma= 13
* Hebra 0 suma= 7
* Hebra 2 suma= 9
* Hebra 3 suma= 10
* Hebra 7 suma= 7
* Hebra 4 suma= 11
```

```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 = 7;

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

        #pragma omp parallel private(suma)
        {
        #pragma omp for
        for(i=0;i<n;i++)
        {
           suma = suma + a[i];
           printf("Hebra %d suma a[%d]/",omp_get_thread_num(),i);
        }

        printf("\n* Hebra %d suma= %d", omp_get_thread_num(), suma);
        }
 printf("\n");
}
```

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer2] 2020-04-10 viernes
$g++ -O2 -fopenmp -o private-clauseModificado private-clauseModificado.c
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer2] 2020-04-10 viernes
$./private-clauseModificado
Hebra 0 suma a[0]/Hebra 4 suma a[4]/Hebra 2 suma a[2]/Hebra 1 suma a[1]/Hebra 5 suma a[5]/Hebra 6 suma a[6]/Hebra 3 suma a[3]/
* Hebra 2 suma= -1346137534
* Hebra 5 suma= -1346137531
* Hebra 0 suma= 8
* Hebra 3 suma= -1346137533
* Hebra 1 suma= -1346137535
* Hebra 7 suma= -1346137536
* Hebra 6 suma= -1346137530
* Hebra 4 suma= -1346137532
```

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

RESPUESTA:

En todos se imprime el mismo valor de thread. Esto es debido a que la variable suma se vuelve compartida y cada vez que se ejecuta un thread , se inicializa a 0 devolviendonos la suma dada por el ultimo thread que lo ha ejecutado.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

CAPTURAS DE PANTALLA:

```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 += a[i];
           printf("Hebra %d suma a[%d]/",omp_get_thread_num(),i);
        }
        printf("\n* Hebra %d suma= %d", omp_get_thread_num(), suma);
        }
 printf("\n");
}
```

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer3] 2020-04-10 viernes
$g++ -O2 -fopenmp -o private-clauseModificado3 private-clauseModificado3.c
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer3] 2020-04-10 viernes
$./private-clauseModificado3
Hebra 6 suma a[6]/Hebra 5 suma a[5]/Hebra 0 suma a[0]/Hebra 1 suma a[1]/Hebra 2 suma a[2]/Hebra 3 suma a[3]/Hebra 4 suma a[4]/
* Hebra 1 suma= 4
* Hebra 5 suma= 4
* Hebra 6 suma= 4
* Hebra 0 suma= 4
* Hebra 7 suma= 4
* Hebra 3 suma= 4
* Hebra 2 suma= 4
* Hebra 4 suma= 4
```

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:

Sí lo imprime siempre, esto es debido a que por la directiva lastprivate, imprime siempre fuera del parallel el valor de la suma que ha ejecutado la ultima hebra en la ultima iteración del for.

CAPTURAS DE PANTALLA:

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer4] 2020-04-10 viernes
$export OMP_NUM_THREADS=4
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer4] 2020-04-10 viernes
$g++ -O2 -fopenmp -o firstlastprivate firstlastprivate.c
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer4] 2020-04-10 viernes
$./firstlastprivate
 thread 2 suma a[4] suma=4
 thread 2 suma a[5] suma=9
 thread 3 suma a[6] suma=6
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1

Fuera de la construcción parallel suma=6
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer4] 2020-04-10 viernes
$./firstlastprivate
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 3 suma a[6] suma=6
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 2 suma a[4] suma=4
 thread 2 suma a[5] suma=9

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

RESPUESTA:

Cuando se elimina la clausula, solo se copia en algunos el valor de inicialización, y los demás tienen 0 inicializado.

Esto es debido a que solo se inicializan al valor que establecemos solo los valores que han sido inicializados por la hebra que ejecuta el single.(Si la ejecuta la hebra 2, todos los valores que ejecute esa hebra estarán inicializados al valor que establezcamos).

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

CAPTURAS DE PANTALLA:

```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");
}
```

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer5] 2020-04-10 viernes
$./copyprivate-clause

Introduce valor de inicialización a: 7

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

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer5] 2020-04-10 viernes
$./copyprivate-clauseModificado

Introduce valor de inicialización a: 7

Single ejecutada por el thread 0
Depués de la región parallel:
b[0] = 7        b[1] = 7        b[2] = 7        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 (añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA:

Se imprime el mismo resultado de antes +10 . Esto se debe a que no se ha inicializado 0 en la clausula reduction donde se vaya a almacenar el resultado.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

CAPTURAS DE PANTALLA:

```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);
}
```

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer6] 2020-04-10 viernes
$./reduction-clause 10
Tras 'parallel' suma=45
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer6] 2020-04-10 viernes
$./reduction-clause 20
Tras 'parallel' suma=190
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer6] 2020-04-10 viernes
$./reduction-clause 30
n=20Tras 'parallel' suma=190
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer6] 2020-04-10 viernes
$./reduction-clauseModificado 10
Tras 'parallel' suma=55
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer6] 2020-04-10 viernes
$./reduction-clauseModificado 20
Tras 'parallel' suma=200
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer6] 2020-04-10 viernes
$./reduction-clauseModificado 30
n=20Tras 'parallel' suma=200
```

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:

Solo hay que añadir una directiva atomic para poder acceder a suma y modificarla, en caso contrario podrian modificarla varias a la vez de manera que sea erróneo.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

CAPTURAS DE PANTALLA:

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

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

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer7] 2020-04-10 viernes
$./reduction-clauseModificado7 10
Tras 'parallel' suma=45
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer7] 2020-04-10 viernes
$./reduction-clauseModificado7 20
Tras 'parallel' suma=190
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer7] 2020-04-10 viernes
$./reduction-clauseModificado7 30
n=20Tras 'parallel' suma=190
```

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):

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

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CAPTURA CÓDIGO FUENTE: `pmv-secuencial.c`
CAPTURAS DE PANTALLA:

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

//#define GLOBAL
#define DINAMIC

#ifdef GLOBAL
        #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
        if(argc != 2){                                  //Esto es para aceptar el argumento N(numero de filas/columnas)
        printf("Error de argumentos %s", argv[0]);
        return(EXIT_FAILURE);
    }

    struct timespec cgt1, cgt2;                 //Definimos estructuras para medir el tiempo
    double ncgt;

    int N = atoi(argv[1]);

    #ifdef GLOBAL                               //Se inicializan los vectores y la matriz de forma global
        if(N > MAX) N = MAX;
        int matriz[N][N];
        int vector[N];
        int vr[N];
        printf("Con Vectores Globales\n");
    #endif

    #ifdef DINAMIC
        int **matriz, *vector, *vr;                     //Inicializacion dinámica de la matriz y los vectores
        matriz = (int**) malloc(N * sizeof(int*));
        for(int i = 0; i < N; ++i)
            matriz[i] = (int*) malloc(N * sizeof(int));

        vector = (int*) malloc(N * sizeof(int));
        vr = (int*) malloc(N * sizeof(int));
        printf("Con Vectores Dinámicos\n");
    #endif

    for(int i = 0; i < N; ++i){                         //Se rellenan el vector y la matriz con valores
        vector[i] = i;
        for(int j = 0; j < N; ++j)
                matriz[i][j] = i + j;
        }
```

```
        clock_gettime(CLOCK_REALTIME, &cgt1);           //Cojemos este instante de tiempo

        for(int i = 0; i < N; i++){                     //Calculamos el vector resultante
                int suma = 0;
                for(int j = 0; j < N; j++)
                        suma += matriz[i][j] * vector[j];

                vr[i] = suma;
        }

        clock_gettime(CLOCK_REALTIME, &cgt2);           //Cojemos el instante de tiempo de justo cuando termina el algoritmo
        ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) + (double) (cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9); //Obtenemos el tiempo de ejecucion

        printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);
        if(N < 15)                                                      //Con este if/else mostramos el vector con valores pequeños
                for(int i = 0; i < N; i++){
                        printf("Vector Resultante[%d] = %d  ", i, vr[i]);
                        printf("\n");
                }
        else{
                printf("Vector Resultante[0] = %d  ", vr[0]);
                printf("Vector Resultante[%d] = %d  ", N - 1, vr[N - 1]);
                printf("\n");
        }

        #ifdef DINAMIC
        for(int i = 0; i < N; i++)                      //Liberación del espacio cuando usamos matriz y vectores dinamicos
                free(matriz[i]);

        free(matriz); free(vector); free(vr);
        #endif

    return 0;
}
```

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`

```
#pragma omp parallel for
for(int i = 0; i < N; ++i){                         //Se rellenan el vector y la matriz con valores
    vector[i] = i;
    for(int j = 0; j < N; ++j)
            matriz[i][j] = i + j;
    }

    clock_gettime(CLOCK_REALTIME, &cgt1);           //Cojemos este instante de tiempo

    #pragma omp parallel for
    for(int i = 0; i < N; i++){                     //Calculamos el vector resultante
            int suma = 0;
            for(int j = 0; j < N; j++)
                    suma += matriz[i][j] * vector[j];

            vr[i] = suma;
    }
```

CAPTURA CÓDIGO FUENTE: `pmv-OpenMP-b.c`

```
#pragma omp parallel for
for(int i = 0; i < N; ++i){                         //Se rellenan el vector y la matriz con valores
    vector[i] = i;
    for(int j = 0; j < N; ++j)
            matriz[i][j] = i + j;
    }

    clock_gettime(CLOCK_REALTIME, &cgt1);           //Cojemos este instante de tiempo


    for(int i = 0; i < N; i++){                     //Calculamos el vector resultante
            int suma = 0;
            #pragma omp parallel for
            for(int j = 0; j < N; j++)
                    #pragma omp atomic
                    suma += matriz[i][j] * vector[j];

            vr[i] = suma;
    }
```

RESPUESTA:

Mi único error ha sido en el de paralelizar las columnas, no había puesto el pragma omp atomic, y daba resultados diferentes en cada ejecución.

CAPTURAS DE PANTALLA:

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer9] 2020-04-11 sábado
$./pmv-OpenMP-a 11
Con Vectores Dinámicos
Tiempo(seg.): 0.002325412       / Tamaño vectores: 11
Vector Resultante[0] = 385
Vector Resultante[1] = 440
Vector Resultante[2] = 495
Vector Resultante[3] = 550
Vector Resultante[4] = 605
Vector Resultante[5] = 660
Vector Resultante[6] = 715
Vector Resultante[7] = 770
Vector Resultante[8] = 825
Vector Resultante[9] = 880
Vector Resultante[10] = 935
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer9] 2020-04-11 sábado
$./pmv-OpenMP-b 11
Con Vectores Dinámicos
Tiempo(seg.): 0.030735740       / Tamaño vectores: 11
Vector Resultante[0] = 385
Vector Resultante[1] = 440
Vector Resultante[2] = 495
Vector Resultante[3] = 550
Vector Resultante[4] = 605
Vector Resultante[5] = 660
Vector Resultante[6] = 715
Vector Resultante[7] = 770
Vector Resultante[8] = 825
Vector Resultante[9] = 880
Vector Resultante[10] = 935
```

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`

```
for(int i = 0; i < N; i++){                    //Calculamos el vector resultante
        int suma = 0;
        #pragma omp parallel for reduction(+:suma)
        for(int j = 0; j < N; j++)
                suma += matriz[i][j] * vector[j];

        vr[i] = suma;
}
```

RESPUESTA: **No he tenido ningún fallo ni complicación en este ejercicio.**

CAPTURAS DE PANTALLA:

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer10] 2020-04-11 sábado
$./pmv-OpenMP-reduction 11
Con Vectores Dinámicos
Tiempo(seg.): 0.027413497        / Tamaño vectores: 11
Vector Resultante[0] = 385
Vector Resultante[1] = 440
Vector Resultante[2] = 495
Vector Resultante[3] = 550
Vector Resultante[4] = 605
Vector Resultante[5] = 660
Vector Resultante[6] = 715
Vector Resultante[7] = 770
Vector Resultante[8] = 825
Vector Resultante[9] = 880
Vector Resultante[10] = 935
```

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):

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer11] 2020-04-11 sábado
$gcc -O2 -fopenmp -o pmv-OpenMP-a pmv-OpenMP-a.c
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer11] 2020-04-11 sábado
$gcc -O2 -fopenmp -o pmv-OpenMP-b pmv-OpenMP-b.c
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer11] 2020-04-11 sábado
$gcc -O2 -fopenmp -o pmv-OpenMP-reduction pmv-OpenMP-reduction.c
```

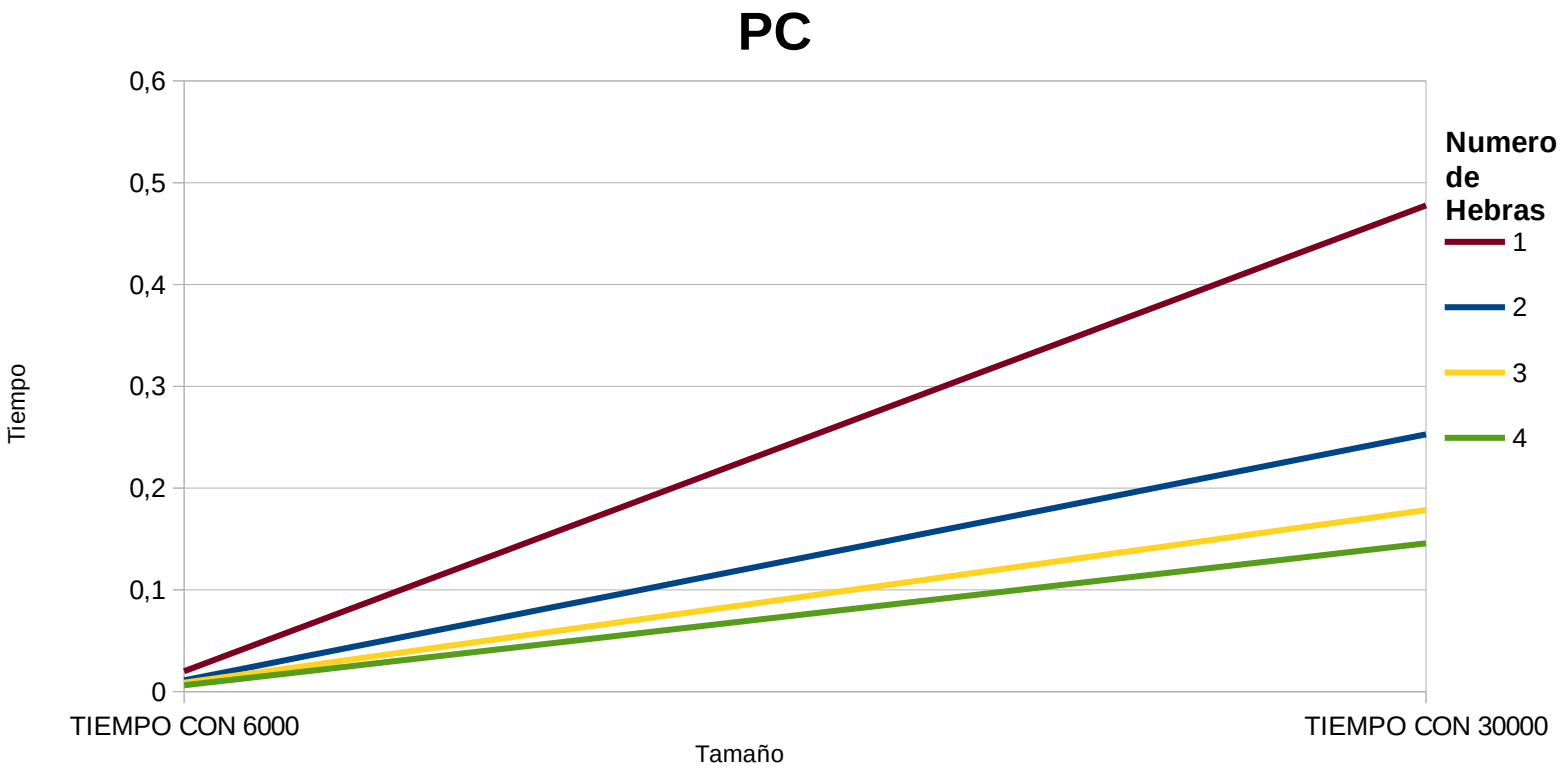
Elegimos el pmv-OpenMP-a ya que es el más rápido.

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer11] 2020-04-11 sábado
$./pmv-OpenMP-a 6000 ; ./pmv-OpenMP-b 6000 ; ./pmv-OpenMP-reduction 6000
Con Vectores Dinámicos
Tiempo(seg.): 0.005379123        / Tamaño vectores: 6000
Con Vectores Dinámicos
Tiempo(seg.): 1.185934304        / Tamaño vectores: 6000
Con Vectores Dinámicos
Tiempo(seg.): 0.014366084        / Tamaño vectores: 6000
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer11] 2020-04-11 sábado
$./pmv-OpenMP-a 20000 ; ./pmv-OpenMP-b 20000 ; ./pmv-OpenMP-reduction 20000
Con Vectores Dinámicos
Tiempo(seg.): 0.093705606        / Tamaño vectores: 20000
Con Vectores Dinámicos
Tiempo(seg.): 16.081198984       / Tamaño vectores: 20000
Con Vectores Dinámicos
Tiempo(seg.): 0.084352629        / Tamaño vectores: 20000
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp2/PC/ejer11] 2020-04-11 sábado
$./pmv-OpenMP-a 30000 ; ./pmv-OpenMP-reduction 30000
Con Vectores Dinámicos
Tiempo(seg.): 0.140077453        / Tamaño vectores: 30000
Con Vectores Dinámicos
Tiempo(seg.): 0.174043860        / Tamaño vectores: 30000
```

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):

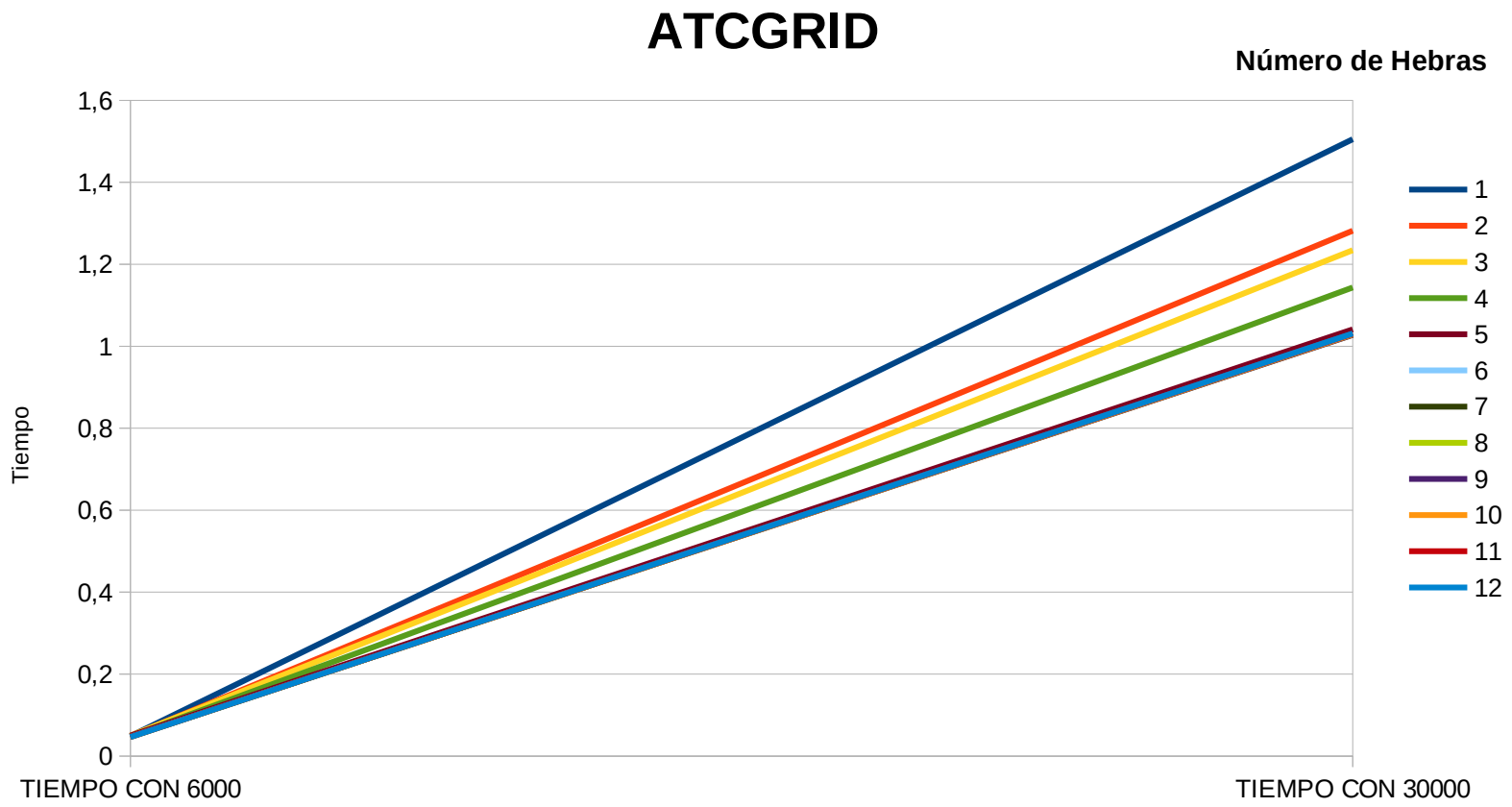
PC

HEBRAS	TIEMPO CON 6000	TIEMPO CON 30000
1	0,020126529	0,477595772
2	0,011158813	0,252843944
3	0,008809056	0,178341523
4	0,006321325	0,145673456



ATCGRID

HEBRAS	TIEMPO CON 6000	TIEMPO CON 30000
1	0,048708266	1,504930558
2	0,049240056	1,281789676
3	0,049925194	1,234123487
4	0,048311947	1,143258956
5	0,049964567	1,041106336
6	0,046872006	1,030164723
7	0,046748347	1,030515732
8	0,047124018	1,027874272
9	0,047351995	1,027770846
10	0,046784739	1,028857375
11	0,046961605	1,030765734
12	0,046746536	1,030942886



COMENTARIOS SOBRE LOS RESULTADOS:

En mi PC se observa que hay un gran salto de tiempo entre el secuencial y el primer paralelo con 2 hilos y luego de eso, sigue bajando el tiempo pero cada vez en un porcentaje menor.

En ATCGRID se observa lo mismo con el secuencial y el primer paralelo, y luego siguen bajando los tiempos de forma indefinida hasta los 5/6 hilos, donde a partir de ahi se obtienen tiempos super similares dando a entender que a partir de 5/6 hilos se va a tener la misma velocidad de procesamiento.