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): David Martínez Díaz
Grupo de prácticas y profesor de prácticas:Grupo 2 - Juan José Escobar Pérez
Fecha de entrega: 20/04/2021
Fecha evaluación en clase: 22/04/2021

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. **(a)** Añadir la cláusula `default(none)` a la directiva `parallel` del ejemplo del seminario `shared-clause.c`? ¿Qué ocurre? ¿A qué se debe? **(b)** Resolver el problema generado sin eliminar `default(none)`. Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla que muestren lo que ocurre)

RESPUESTA:

Lo que pasa es que con la cláusula default(none), tiene que especificarse que el alcance de todas las variables usadas en su construcción, y que las variables que están fuera de este no se compartan dando el error de compilar. En este caso, se debe especificar que el vector "a" es compartido y la variable "n".

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

    printf("Después de parallel for:\n");

    for (i=0; i<n; i++)
        printf("a[%d] = %d\n",i,a[i]);
}
```

CAPTURAS DE PANTALLA:

```
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp2/ejer1     ls
shared-clause.c  shared-clauseModificado.c
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp2/ejer1     gcc -O2 -fopenmp
shared-clauseModificado.c -o shared-clauseModificado
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:13:13: error: 'n' not specified
in enclosing 'parallel'
   13 |     #pragma omp parallel for default(none) shared(
a)
      |             ^~~
shared-clauseModificado.c:13:13: error: enclosing 'paralle
l'
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp2/ejer1
```

```
C | /mnt/d/U/De/I/2/2/AC/A/bp2/ejer1     gcc -O2 -fopenmp
shared-clauseModificado.c -o shared-clauseModificado
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp2/ejer1     ./shared-clauseMo
dificado
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
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp2/ejer1
```

2. **(a)** Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel`. Inicializar `suma` dentro del `parallel` a un valor distinto de 0. Ejecutar varias veces el código ¿Qué imprime el código fuera del `parallel`? (mostrar lo que ocurre con una captura de pantalla) Razonar respuesta. **(b)** Modificar el código del apartado (a) para que se inicialice `suma` fuera del `parallel` en lugar de dentro ¿Qué ocurre? Comparar todo lo que imprime el código ahora con la salida en (a) (mostrar la salida con una captura de pantalla) Razonar respuesta.

(a) RESPUESTA:

El código fuera del parallel imprime por pantalla el valor de suma que es 0, como hemos puesto que la variable suma es privada con la clausula private, cada hebra calcula su suma independiente al ser privado, y al salir de la zona privada no se guarda.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.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 private(suma)
    {
        suma=3;
        #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:

```
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp2/ejer2      gcc -O2 -fopenmp private-clauseModificado_a.c -o priv
ate-clauseModificado_a
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp2/ejer2      ./private-clauseModificado_a
thread 0 suma a[0] / thread 1 suma a[1] / thread 3 suma a[3] / thread 6 suma
 a[6] / thread 2 suma a[2] / thread 5 suma a[5] / thread 4 suma a[4] /
* thread 0 suma= 0
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp2/ejer2      ./private-clauseModificado_a
thread 3 suma a[3] / thread 1 suma a[1] / thread 2 suma a[2] / thread 5 suma
 a[5] / thread 0 suma a[0] / thread 4 suma a[4] / thread 6 suma a[6] /
* thread 0 suma= 0
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp2/ejer2                                          ✓ | 17:04:01
```

(b) RESPUESTA:

Si inicializamos la variable suma fuera del parallel, de la construcción ocurre lo mismo, no se guarda el resultado en la suma original, y entonces no imprime el trozo secuencial.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_b.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 = 3;

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

    #pragma omp parallel private(suma)
    {
        //suma=3;
        #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:

```
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp2/ejer2     gcc -O2 -fopenmp private-clauseModificado_b.c -o priv
ate-clauseModificado_b
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp2/ejer2     ./private-clauseModificado_b            ✓ | 17:09:35
thread 1 suma a[1] / thread 3 suma a[3] / thread 0 suma a[0] / thread 4 suma
 a[4] / thread 5 suma a[5] / thread 2 suma a[2] / thread 6 suma a[6] /
* thread 0 suma= 3
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp2/ejer2     ./private-clauseModificado_b            ✓ | 17:09:50
thread 4 suma a[4] / thread 1 suma a[1] / thread 3 suma a[3] / thread 6 suma
 a[6] / thread 0 suma a[0] / thread 2 suma a[2] / thread 5 suma a[5] /
* thread 0 suma= 3
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp2/ejer2                                           ✓ | 17:09:51
```

3. **(a)** Eliminar la cláusula `private(suma)` en `private-clause.c`. Ejecutar el código resultante. ¿Qué ocurre? **(b)** ¿A qué es debido?

RESPUESTA:
Lo que ocurre que al quitarle el private a la variable suma, por lo que se ha convertido otra vez en común para todas las hebras, obteniendo la suma del total de las hebras.

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++)s
        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:

```
[e2estudiante14@atcgrid ejer3]$ gcc -O2 -fopenmp private-clauseModificado3.c
-o private-clauseModificado3
[e2estudiante14@atcgrid ejer3]$ sbatch -p ac --wrap "./private-clauseModifica
do3"
Submitted batch job 87943
[e2estudiante14@atcgrid ejer3]$ sbatch -p ac --wrap "./private-clauseModifica
do3"
Submitted batch job 87944
[e2estudiante14@atcgrid ejer3]$ sbatch -p ac --wrap "./private-clauseModifica
do3"
Submitted batch job 87945
[e2estudiante14@atcgrid ejer3]$ ls
private-clause.c             slurm-87943.out
private-clauseModificado3    slurm-87944.out
private-clauseModificado3.c  slurm-87945.out
[e2estudiante14@atcgrid ejer3]$ cat slurm-87943.out slurm-87944.out slurm-879
45.out
thread 1 suma a[4] / thread 1 suma a[5] / thread 1 suma a[6] / thread 0 suma
a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3] /
* thread 1 suma= 21
* thread 0 suma= 21
thread 1 suma a[4] / thread 1 suma a[5] / thread 1 suma a[6] / thread 0 suma
a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3] /
* thread 1 suma= 21
* thread 0 suma= 21
thread 1 suma a[4] / thread 1 suma a[5] / thread 1 suma a[6] / thread 0 suma
a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3] /
* thread 1 suma= 21
* thread 0 suma= 21
[e2estudiante14@atcgrid ejer3]$
```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región `parallel`. **(a)** Cambiar el tamaño del vector a 10. Razonar lo que imprime el código en su PC con esta modificación. (añadir capturas de pantalla que muestren lo que ocurre). **(b)** Sin cambiar el tamaño del vector ¿podría imprimir el código otro valor? Razonar respuesta (añadir capturas de pantalla que muestren lo que ocurre).

(a) RESPUESTA:
Lo que ocurre es que guarda el ultimo valor de la ultima hebra que ha accedido a la sección parallel machacando el valor del fistprivate. De tal forma que el lastprivate obtendría suma el valor de la primera hebra que ha accedido a la región parallalel.

CAPTURAS DE PANTALLA:

```
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp2/ejer4     gcc -O2 -fopenmp
firstlastprivate.c -o firstlastprivate
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp2/ejer4     ./firstlastprivat
e
 thread 0 suma a[0] suma=0
 thread 3 suma a[3] suma=3
 thread 8 suma a[8] suma=8
 thread 2 suma a[2] suma=2
 thread 5 suma a[5] suma=5
 thread 9 suma a[9] suma=9
 thread 1 suma a[1] suma=1
 thread 4 suma a[4] suma=4
 thread 6 suma a[6] suma=6
 thread 7 suma a[7] suma=7

Fuera de la construcción parallel suma=9
 DavidMartinezDiaz - 21-04-12 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp2/ejer4
```

(b) RESPUESTA:
Pienso que no se podría cambiar el valor, ya que con el “lastprivate”, este le asigna a la variable el valor de la última iteración del bucle y por eso siempre sale 9.

CAPTURAS DE PANTALLA:

```
 DavidMartinezDiaz - 21-04-20 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/
U/De/I/2/2/AC/A/bp2/ejer4     ./firstlastprivate
 thread 0 suma a[0] suma=0
 thread 5 suma a[5] suma=5
 thread 3 suma a[3] suma=3
 thread 4 suma a[4] suma=4
 thread 7 suma a[7] suma=7
 thread 1 suma a[1] suma=1
 thread 6 suma a[6] suma=6
 thread 8 suma a[8] suma=8
 thread 2 suma a[2] suma=2
 thread 9 suma a[9] suma=9

Fuera de la construcción parallel suma=9
 DavidMartinezDiaz - 21-04-20 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/
U/De/I/2/2/AC/A/bp2/ejer4     ./firstlastprivate
 thread 3 suma a[3] suma=3
 thread 0 suma a[0] suma=0
 thread 6 suma a[6] suma=6
 thread 8 suma a[8] suma=8
 thread 4 suma a[4] suma=4
 thread 9 suma a[9] suma=9
 thread 1 suma a[1] suma=1
 thread 7 suma a[7] suma=7
 thread 2 suma a[2] suma=2
 thread 5 suma a[5] suma=5

Fuera de la construcción parallel suma=9
 DavidMartinezDiaz - 21-04-20 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/
U/De/I/2/2/AC/A/bp2/ejer4                          | 18:48:55
```

5. **(a)** ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? **(b)** ¿A qué cree que es debido? (añadir una captura de pantalla que muestre lo que ocurre)

RESPUESTA:

Se imprime el mismo valor para la hebra que realiza esa serie de sumas en el vector, es decir, los valores del vector que tengan un numero son los que están siendo ejecutados por la mima hebra.

Lo que ocurre es que al eliminar el copyprivate(a), la hebra que se mete dentro del parallel single, no va a copiar la variable privada a las demás hebras en sus respectivas variables privadas del mismo nombre de estas, por lo que si lo quitamos esta clausula, las variables privadas no tendrán el mismo valor que la hebra que se mete.

CAPTURA CÓDIGO FUENTE: `copyprivate-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 n = 9, i, b[n];

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

    #pragma omp parallel
    {
        int a;
        #pragma omp single //copyprivate(a)
        {
            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:

```
[e2estudiante14@atcgrid ejer5]$ srun -p ac ./copyprivate-clauseModificado
4

Introduce valor de inicialización a:
Single ejecutada por el thread 1
Depués de la región parallel:
b[0] = 0        b[1] = 0        b[2] = 0        b[3] = 0        b[4] = 0   b
[5] = 4 b[6] = 4        b[7] = 4        b[8] = 4
[e2estudiante14@atcgrid ejer5]$
```

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:
En el resultado nos va a salir el valor de nuestro bucle mas nuestro valor inicial, en este caso es 10, por lo que a la hora de especificar la clausula reduction(+:suma), hemos definido que el valor inicial es 10, al cual le iremos sumando el numero de iteraciones que vayamos haciendo, en mi caso, con la iteración 4, le sumamos a 10, el 0, 1, 2, 3, dando lugar el numero 16, que seria en este caso correcto.

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:

```
[e2estudiante14@atcgrid ejer6]$ ls
reduction-clause.c  reduction-clauseModificado.c
[e2estudiante14@atcgrid ejer6]$ gcc -O2 -fopenmp reduction-clauseModificado.
c -o reduction-clauseModificado
[e2estudiante14@atcgrid ejer6]$ srun -p ac ./reduction-clauseModificado 4
Tras 'parallel' suma=16
[e2estudiante14@atcgrid ejer6]$
```

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:

En este caso lo que es más eficaz es utilizar la directiva atomic, para poder sincronizar las distintas threads, y poco a poco ir actualizando su valor en la variable, forzando al menos a una hebra a que lo vaya cambiando a la vez, para que no se produzcan errores.

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

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

CAPTURAS DE PANTALLA:

```
[e2estudiante14@atcgrid ejer7]$ gcc -O2 -fopenmp reduction
-clauseModificado7.c -o reduction-clauseModificado7
[e2estudiante14@atcgrid ejer7]$ srun -p ac ./reduction-cla
useModificado7 5
Tras 'parallel' suma=10
[e2estudiante14@atcgrid ejer7]$ srun -p ac ./reduction-cla
useModificado7 5
Tras 'parallel' suma=10
[e2estudiante14@atcgrid ejer7]$
```

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

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`

```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){
        printf("Error de argumentos %s", argv[0]);
        return(EXIT_FAILURE);
    }

    struct timespec cgt1, cgt2;
    double ncgt;

    int N = atoi(argv[1]);

//------------------------------------------------------------------//

    #ifdef GLOBAL

        if(N > MAX) N = MAX;

        int matriz[N][N];
        int vector[N];
        int vector_resultado[N];

        printf("Ejecutado GLOBALMENTE\n");

    #endif

    #ifdef DINAMIC

        int **matriz, *vector, *vector_resultado;

        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));
        vector_resultado = (int*) malloc(N * sizeof(int));

        printf("Ejecutado DINAMICAMENTE\n");

    #endif

//------------------------------------------------------------------//

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

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

    clock_gettime(CLOCK_REALTIME, &cgt1);

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

        vector_resultado[i] = suma;
    }

```

```
    clock_gettime(CLOCK_REALTIME, &cgt2);
    ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) + (double) (cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9);

    printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);

    if(N < 15)
        for(int i = 0; i < N; i++){
            printf("VECTOR_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
            printf("\n");
        }
    else{S
        printf("VECTOR_RESULTADO[0] = %d  ", vector_resultado[0]);
        printf("VECTOR_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
        printf("\n");
    }

    #ifdef DINAMIC
    for(int i = 0; i < N; i++)
        free(matriz[i]);

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

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[e2estudiante14@atcgrid ejer8]$ srun -p ac ./pmv-secuencia
l 8
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000000290        / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
VECTOR_RESULTADO[2] = 196
VECTOR_RESULTADO[3] = 224
VECTOR_RESULTADO[4] = 252
VECTOR_RESULTADO[5] = 280
VECTOR_RESULTADO[6] = 308
VECTOR_RESULTADO[7] = 336
[e2estudiante14@atcgrid ejer8]$ srun -p ac ./pmv-secuencia
l 11
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000000400        / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385
VECTOR_RESULTADO[1] = 440
VECTOR_RESULTADO[2] = 495
VECTOR_RESULTADO[3] = 550
VECTOR_RESULTADO[4] = 605
VECTOR_RESULTADO[5] = 660
VECTOR_RESULTADO[6] = 715
VECTOR_RESULTADO[7] = 770
VECTOR_RESULTADO[8] = 825
VECTOR_RESULTADO[9] = 880
VECTOR_RESULTADO[10] = 935
[e2estudiante14@atcgrid ejer8]$
```

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 <malloc.h>
#include <omp.h>

#define GLOBAL
//#define DINAMIC

#ifdef GLOBAL
    #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
    if(argc != 2){
        printf("Error de argumentos %s", argv[0]);
        return(EXIT_FAILURE);
    }

    struct timespec cgt1, cgt2;
    double ncgt;

    int N = atoi(argv[1]);

    #ifdef GLOBAL
        if(N > MAX) N = MAX;
        int matriz[N][N];
        int vector[N];
        int vector_resultado[N];
        printf("Ejecutado GLOBALMENTE\n");
    #endif

    #ifdef DINAMIC
        int **matriz, *vector, *vector_resultado;
        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));
        vector_resultado = (int*) malloc(N * sizeof(int));
        printf("Ejecutado DINAMICAMENTE\n");
    #endif

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


    clock_gettime(CLOCK_REALTIME, &cgt1);

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

        vector_resultado[i] = suma;
    }

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

    printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);
    if(N < 15)
        for(int i = 0; i < N; i++){
            printf("VECTOR_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
            printf("\n");
        }
    else{
        printf("VECTOR_RESULTADO[0] = %d  ", vector_resultado[0]);
        printf("VECTOR_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
        printf("\n");
    }

    #ifdef DINAMIC
    for(int i = 0; i < N; i++)
        free(matriz[i]);

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

    return 0;
}
```

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

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

#define GLOBAL
//#define DINAMIC

#ifdef GLOBAL
    #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
    if(argc != 2){
        printf("Error de argumentos %s", argv[0]);
        return(EXIT_FAILURE);
    }

    struct timespec cgt1, cgt2;
    double ncgt;

    int N = atoi(argv[1]);

    #ifdef GLOBAL
        if(N > MAX) N = MAX;
        int matriz[N][N];
        int vector[N];
        int vector_resultado[N];
        printf("Ejecutado GLOBALMENTE\n");
    #endif

    #ifdef DINAMIC
        int **matriz, *vector, *vector_resultado;
        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));
        vector_resultado = (int*) malloc(N * sizeof(int));
        printf("Ejecutado DINAMICAMENTE\n");
    #endif

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

    clock_gettime(CLOCK_REALTIME, &cgt1);

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

        vector_resultado[i] = suma;
    }

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

    printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);
    if(N < 15)
        for(int i = 0; i < N; i++){
            printf("VECTOR_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
            printf("\n");
        }
    else{
```

```
72      else{
73          printf("VECTOR_RESULTADO[0] = %d  ", vector_resultado[0]);
74          printf("VECTOR_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
75          printf("\n");
76      }
77
78      #ifdef DINAMIC
79      for(int i = 0; i < N; i++)
80          free(matriz[i]);
81
82      free(matriz); free(vector); free(vector_resultado);
83      #endif
84
85      return 0;
86  }
87
```

RESPUESTA:

Este código es parecido al ejercicio anterior, por lo que he no he tenido problemas de compilación, ya que me he ido orientando, además le he añadido el "omp.h".
Aunque si he tenido un fallo en la suma, ya que un resultado no me coincidia y al final puse la directiva atomic para que funcionase.

CAPTURAS DE PANTALLA:

```
[e2estudiante14@atcgrid ejer9]$ srun -p ac ./pmv-OpenMP-a
8
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000003352       / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
VECTOR_RESULTADO[2] = 196
VECTOR_RESULTADO[3] = 224
VECTOR_RESULTADO[4] = 252
VECTOR_RESULTADO[5] = 280
VECTOR_RESULTADO[6] = 308
VECTOR_RESULTADO[7] = 336
[e2estudiante14@atcgrid ejer9]$ srun -p ac ./pmv-OpenMP-b
8
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000093373       / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
VECTOR_RESULTADO[2] = 196
VECTOR_RESULTADO[3] = 224
VECTOR_RESULTADO[4] = 252
VECTOR_RESULTADO[5] = 280
VECTOR_RESULTADO[6] = 308
VECTOR_RESULTADO[7] = 336
[e2estudiante14@atcgrid ejer9]$
```

```
[e2estudiante14@atcgrid ejer9]$ srun -p ac ./pmv-OpenMP-a
11
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000003335       / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385
VECTOR_RESULTADO[1] = 440
VECTOR_RESULTADO[2] = 495
VECTOR_RESULTADO[3] = 550
VECTOR_RESULTADO[4] = 605
VECTOR_RESULTADO[5] = 660
VECTOR_RESULTADO[6] = 715
VECTOR_RESULTADO[7] = 770
VECTOR_RESULTADO[8] = 825
VECTOR_RESULTADO[9] = 880
VECTOR_RESULTADO[10] = 935
[e2estudiante14@atcgrid ejer9]$ srun -p ac ./pmv-OpenMP-b
11
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000025743       / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385
VECTOR_RESULTADO[1] = 440
VECTOR_RESULTADO[2] = 495
VECTOR_RESULTADO[3] = 550
VECTOR_RESULTADO[4] = 605
VECTOR_RESULTADO[5] = 660
VECTOR_RESULTADO[6] = 715
VECTOR_RESULTADO[7] = 770
VECTOR_RESULTADO[8] = 825
VECTOR_RESULTADO[9] = 880
[e2estudiante14@atcgrid ejer9]$
[e2estudiante14@atcgrid ejer9]$
```

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`

```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){
        printf("Error de argumentos %s", argv[0]);
        return(EXIT_FAILURE);
    }

    struct timespec cgt1, cgt2;
    double ncgt;

    int N = atoi(argv[1]);

    #ifdef GLOBAL
        if(N > MAX) N = MAX;
        int matriz[N][N];
        int vector[N];
        int vector_resultado[N];
        printf("Ejecutado GLOBALMENTE\n");
    #endif

    #ifdef DINAMIC
        int **matriz, *vector, *vector_resultado;
        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));
        vector_resultado = (int*) malloc(N * sizeof(int));
        printf("Ejecutado DINAMICAMENTE\n");
    #endif

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

    clock_gettime(CLOCK_REALTIME, &cgt1);
    //Calculamos el vector resultado
    for(int i = 0; i < N; i++){
        int suma = 0;
        #pragma omp parallel for reduction(+:suma)
        for(int j = 0; j < N; j++)
            suma += matriz[i][j] * vector[j];

        vector_resultado[i] = suma;
    }

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

    printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);
    if(N < 15)
        for(int i = 0; i < N; i++){
            printf("VECTOR_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
            printf("\n");
        }
    else{
        printf("VECTOR_RESULTADO[0] = %d  ", vector_resultado[0]);
        printf("VECTOR_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
        printf("\n");
    }

    #ifdef DINAMIC
    for(int i = 0; i < N; i++)
        free(matriz[i]);

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

    return 0;
}
```

RESPUESTA:
En ejercicio no he tenido ningún problema ya que venía experimentado del ejercicio anterior y la implementación del reduction ha sido mas fácil.

CAPTURAS DE PANTALLA:

```
[e2estudiante14@atcgrid ejer10]$ gcc -O2 -fopenmp pmv-OpenmMP-reducti
on.c  -o pmv-OpenmMP-reduction
[e2estudiante14@atcgrid ejer10]$ srun -p ac ./pmv-OpenmMP-reduction 8
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000018819       / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
VECTOR_RESULTADO[2] = 196
VECTOR_RESULTADO[3] = 224
VECTOR_RESULTADO[4] = 252
VECTOR_RESULTADO[5] = 280
VECTOR_RESULTADO[6] = 308
VECTOR_RESULTADO[7] = 336
[e2estudiante14@atcgrid ejer10]$ srun -p ac ./pmv-OpenmMP-reduction 1
1
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000025109       / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385
VECTOR_RESULTADO[1] = 440
VECTOR_RESULTADO[2] = 495
VECTOR_RESULTADO[3] = 550
VECTOR_RESULTADO[4] = 605
VECTOR_RESULTADO[5] = 660
VECTOR_RESULTADO[6] = 715
VECTOR_RESULTADO[7] = 770
VECTOR_RESULTADO[8] = 825
VECTOR_RESULTADO[9] = 880
VECTOR_RESULTADO[10] = 935
[e2estudiante14@atcgrid ejer10]$
```

11. Realizar una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid4, en uno de los nodos de la cola ac 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):

```
[e2estudiante14@atcgrid ejer10]$ srun -p ac ./pmv-OpenmMP-reduction 8
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000018819       / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
```

```
[e2estudiante14@atcgrid ejer9]$ srun -p ac ./pmv-OpenMP-a
8
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000003352       / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
```

```
[e2estudiante14@atcgrid ejer9]$ srun -p ac ./pmv-OpenMP-b
8
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000093373       / Tamaño vectores: 8
```

JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:

He decidido coger el código del ejercicio 9 apartado a, ya que me parece que es el más eficaz de los 3, teniendo un tiempo de 0.000003352 segundos.

CAPTURA DE PANTALLA del script `pmv-OpenmMP-script.sh`

```
#!/bin/bash

echo "Secuencial"
./pmv-secuencial 15000
./pmv-secuencial 24000

echo "Paralelo"

for ((N=0; N<=16; N++)); do

        echo " $N threads "
        export OMP_NUM_THREADS=$N
        ./pmv-OpenMP-a 15000
        ./pmv-OpenMP-a 24000
done

echo " 32 threads "
export OMP_NUM_THREADS=32
./pmv-OpenMP-a 15000
./pmv-OpenMP-a 24000
```

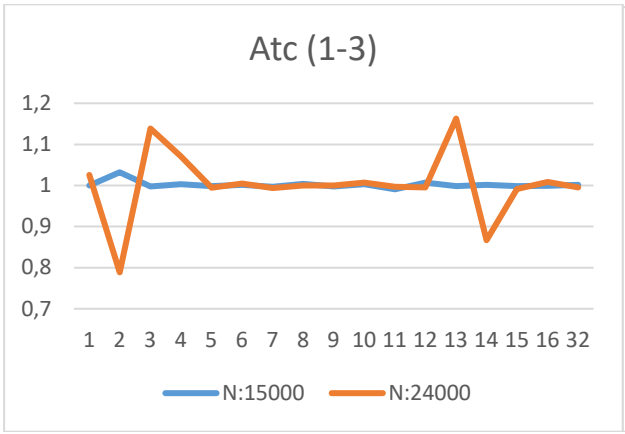
CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):

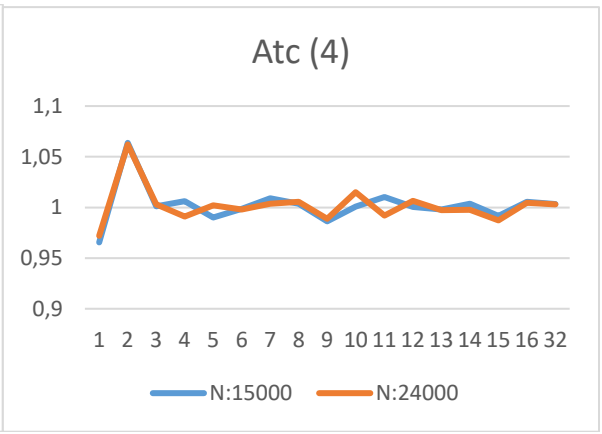
```
[e2estudiante14@atcgrid ejer11]$ srun -p ac ./pmv-OpenmMP-script.sh
Secuencial
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.301239406        / Tamaño vectores: 15000
VECTOR_RESULTADO[0] = -393929052  VECTOR_RESULTADO[14999] = -10410688
80
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.736281200        / Tamaño vectores: 24000
VECTOR_RESULTADO[0] = -787904608  VECTOR_RESULTADO[23999] = 33728128
Paralelo
 0 threads

libgomp: Invalid value for environment variable OMP_NUM_THREADS
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.292065003        / Tamaño vectores: 15000
VECTOR_RESULTADO[0] = -393929052  VECTOR_RESULTADO[14999] = -10410688
80

libgomp: Invalid value for environment variable OMP_NUM_THREADS
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.912405329        / Tamaño vectores: 24000
VECTOR_RESULTADO[0] = -787904608  VECTOR_RESULTADO[23999] = 33728128
 1 threads
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.301204101        / Tamaño vectores: 15000
VECTOR_RESULTADO[0] = -393929052  VECTOR_RESULTADO[14999] = -10410688
80
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.717691012        / Tamaño vectores: 24000
VECTOR_RESULTADO[0] = -787904608  VECTOR_RESULTADO[23999] = 33728128
 2 threads
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.291849454        / Tamaño vectores: 15000
VECTOR_RESULTADO[0] = -393929052  VECTOR_RESULTADO[14999] = -10410688
80
```

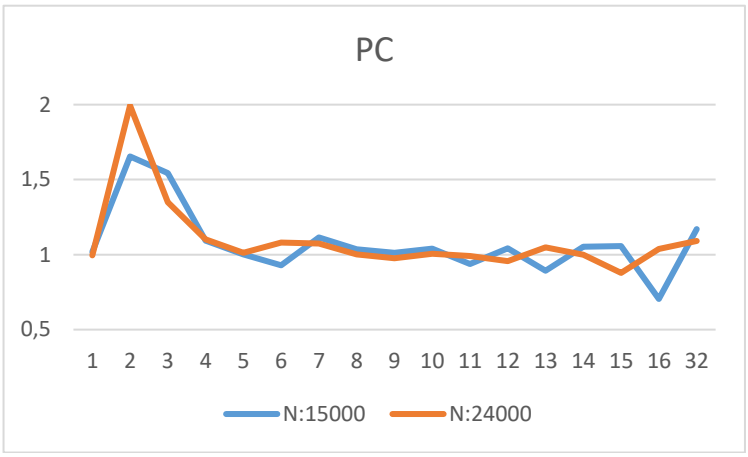
TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia):

Tabla 1. Tiempos de ejecución del código secuencial y de la versión paralela para atcgrid y para el PC personal

	atcgrid1, atcgrid2 o atcgrid3				**atcgrid4**				**PC**			
	Tamaño= entre 5000 y 10000		**Tamaño= entre 10000 y 100000**		**Tamaño= entre 5000 y 10000**		**Tamaño= entre 10000 y 100000**		**Tamaño= entre 5000 y 10000**		**Tamaño= entre 10000 y 100000**	
Nº de núcleos (p)	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**
Código Secuencial	0.301239406	----	0.736281200	----	0,120315	----	0,308643024	----	0.106698000	----	0.291956800	----
1	0.301204101	1,0001172	0.717691012	1,025903	0,124615	0,9654992	0,317536542	0,971992143	0.108738600	1,018707525	0.293305100	0,995403
2	0.291849454	1,032053	0.910098055	0,788586	0,117151585	1,0637041	0,298956793	1,062148 61	0.063302400	1,654575498	0.147085300	1,994116
3	0.292439017	0,997984	0.799454332	1,138399	0,117030718	1,0010328	0,298073236	1,002964228	0.041010400	1,543569436	0.108948200	1,350048
4	0.291634769	1,0027577	0.746827682	1,070467	0,116302887	1,0062581	0,300772939	0,991024116	0.037542500	1,092372644	0.098824700	1,102439
5	0.292154413	0,9982213	0.750895050	0,994583	0,117479015	0,9899886	0,300151296	1,002071099	0.037526100	1,000437029	0.097644800	1,012084
6	0.291605715	1,0018816	0.747247174	1,004882	0,117656041	0,9984954	0,300727134	0,998085181	0.040425000	0,928289425	0.090339400	1,080866
7	0.292574154	0,9966899	0.751867351	0,993855	0,116609757	1,0089725	0,299683546	1,0034823	0.036298100	1,113694656	0.084093700	1,074271
8	0.291527659	1,0035897	0.751724092	1,000191	0,116191446	1,00360019	0,297997684	1,005657299	0.035017700	1,036564366	0.083997500	1,001145
9	0.292158108	0,9978421	0.751676304	1,000064	0,117805352	0,98630023	0,301371329	0,988805687	0.034567700	1,013017933	0.086071000	0,975909
10	0.291236707	1,0031638	0.746302757	1,0072	0,117717263	1,00074831	0,296879338	1,015130696	0.033261700	1,039264379	0.085495800	1,006728
11	0.293972528	0,9906936	0.748431074	0,997156	0,116517031	1,01030091	0,299255723	0,992059016	0.035446500	0,938363449	0.086357200	0,990025
12	0.291851636	1,007267	0.752011673	0,995239	0,116445878	1,00061104	0,297294764	1,006596009	0.034042400	1,041245623	0.090201100	0,957385
13	0.292229469	0,9987071	0.646769000	1,162721	0,116666748	0,99810683	0,298077 71	0,997373349	0.038172800	1,041245623	0.085972800	1,049182
14	0.291720770	1,0017438	0.746042819	0,866933	0,116248849	1,00359487	0,298788 74	0,997620292	0.036238600	0,891797301	0.086110100	0,998406
15	0.292170797	0,9984597	0.752436165	0,991503	0,117243832	0,99151356	0,302700665	0,987076589	0.034271000	1,053374027	0.098053200	0,878198
16	0.292321233	0,9994854	0.746102667	1,008489	0,116587256	1,00563163	0,301321 4	1,004577388	0.048651800	0,704413814	0.094564200	1,036896
32	0.291787870	1,0018279	0.749731626	0,99516	0,116217 52	1,00318141	0,300465924	1,002847165	0.041567300	1,170434452	0.086655900	1,091261







COMENTARIOS SOBRE LOS RESULTADOS:

En la gráfica de la ganancia del PC se observa que si que es significativa la ganancia del programa en paralelo respecto a la secuencial alcanzando un valor de 2, siendo la mitad de tiempo que en secuencial.

Respecto a la grafica del atc, se nota la diferencia entre el programa secuencial y el paralelo, donde a medida que avanza el numero de hebras se van notando menos las diferencias siendo los tiempos muy parecidos entre unos y otros.