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): Carmen Garcia Romero Grupo de prácticas y profesor de prácticas: 2C-C1 Christian Morillas Fecha de entrega:26/04/21 Fecha evaluación en clase: 27/04/21

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:

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

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("DespuÃ©s de parallel for:\n");
    for (i=0; i<n; i++)
        printf("a[%d] = %d\n",i,a[i]);
}
```

CAPTURAS DE PANTALLA:

- **Error**

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp2/ejer1  2021-04-24 sábado
$gcc -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
’
   14 |      #pragma omp parallel for shared(a) default(none)
      |              ^~~
shared-clauseModificado.c:14:13: error: enclosing ‘parallel’
```

- **Añadiendo n a la clausula shared(s)**

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp2/ejer1  2021-04-24 sábado
$gcc -O2 -fopenmp -o shared-clauseModificado shared-clauseModificado.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp2/ejer1  2021-04-24 sábado
$./shared-clauseModificado
DespuÃ©s de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
```

Respuesta:

Al principio da error de compilación porque usando la cláusula default(none) el programador debe especificar el alcance de las variables usadas en la construcción de la región paralela, a excepción de las variable threadprivate y los índices dentro de una cláusula for.

Por tanto, da error en la variable “n”, ya que no no se ha especificado su alcance.

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: La suma total de las hebras, 22.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.c`

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

    #pragma omp parallel
    {
        suma=1;
        #pragma omp for reduction(+:suma)
        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:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2/ejer2
 2021-04-26 lunes
$gcc -O2 -fopenmp -o private-clauseModificado private-clauseModificado.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2/ejer2
 2021-04-26 lunes
$./private-clauseModificado
thread 0 suma a[0] / thread 4 suma a[4] / thread 2 suma a[2] / thread 3 suma a[3] / thread 5 sum
a a[5] / thread 6 suma a[6] / thread 1 suma a[1] /
* thread 0 suma= 22
```

(b) RESPUESTA: Si se inicializa fuera suma tendrá un valor indeterminado, o en este caso al haberla inicializado con valor 1, se quedará con dicho valor, independientemente de lo que ocurra dentro de parallel.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_b.c`

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

    suma=1;

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

    printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2/ejer2
 2021-04-26 lunes
$gcc -O2 -fopenmp -o private-clauseModificado_b private-clauseModificado_b.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2/ejer2
 2021-04-26 lunes
$./private-clauseModificado_b
thread 0 suma a[0] / thread 6 suma a[6] / thread 3 suma a[3] / thread 1 suma a[1] / thread 5 sum
a a[5] / thread 4 suma a[4] / thread 2 suma a[2] /
* thread 0 suma= 1
```

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

RESPUESTA: Nos da en todo momento el mismo resultado, esto se debe a que al quitar la cláusula private, suma pasa a ser compartida por todas las hebras y por tanto su valor se está sobreescribiendo constantemente, lo que es erróneo.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

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=1;
        #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:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2
/ejer3  2021-04-24 sábado
$gcc -O2 -fopenmp -o private-clause private-clause.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2
/ejer3  2021-04-24 sábado
$./private-clause
thread 1 suma a[1] / thread 2 suma a[2] / thread 4 suma a[4] / thread 0 suma a[0] / threa
d 5 suma a[5] / thread 3 suma a[3] / thread 6 suma a[6] /
* thread 0 suma= 1
* thread 4 suma= 1
* thread 5 suma= 1
* thread 11 suma= 1
* thread 9 suma= 1
* thread 7 suma= 1
* thread 3 suma= 1
* thread 1 suma= 1
* thread 10 suma= 1
* thread 6 suma= 1
* thread 8 suma= 1
* thread 2 suma= 1
```

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: Al usar la cláusula lastprivate, el resultado que se muestra fuera de la región parallel va a ser el valor que asigne la última hebra a la variable suma, en este caso, 9.

CAPTURAS DE PANTALLA:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp2/ejer4  2021-04-24 sábado
$gcc -O2 -fopenmp -o firstlastprivate firstlastprivate.c.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp2/ejer4  2021-04-24 sábado
$./firstlastprivate
 thread 0 suma a[0] suma=0
 thread 3 suma a[3] suma=3
 thread 9 suma a[9] suma=9
 thread 6 suma a[6] suma=6
 thread 2 suma a[2] suma=2
 thread 8 suma a[8] suma=8
 thread 4 suma a[4] suma=4
 thread 5 suma a[5] suma=5
 thread 7 suma a[7] suma=7
 thread 1 suma a[1] suma=1

Fuera de la construcciÃ³n parallel suma=9
```

(b) RESPUESTA: Si eliminamos la cláusula lastprivate, quedando solo firstprivate, hará que el valor que el valor mostrado por pantalla sea el que le asigne la primera hebra a la variable suma, en este caso, 0.

CAPTURAS DE PANTALLA:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp2/ejer4  2021-04-26 lunes
$gcc -O2 -fopenmp -o firstlastprivate firstlastprivate.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp2/ejer4  2021-04-26 lunes
$./firstlastprivate
 thread 4 suma a[4] suma=4
 thread 2 suma a[2] suma=2
 thread 1 suma a[1] suma=1
 thread 5 suma a[5] suma=5
 thread 8 suma a[8] suma=8
 thread 9 suma a[9] suma=9
 thread 3 suma a[3] suma=3
 thread 7 suma a[7] suma=7
 thread 0 suma a[0] suma=0
 thread 6 suma a[6] suma=6

Fuera de la construcciÃ³n parallel suma=0
```

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: Que en a solo se guarda el valor de la hebra que ejecute el single, esto se debe a que al eliminar la cláusula copyprivate, la hebra que ejecuta el código ya no comparte el valor con el resto de hebras.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

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 inicializaciÃ¡n 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("DepuÃŠs de la regiÃ¡n parallel:\n");

    for (i=0; i<n; i++)
        printf("b[%d] = %d\t",i,b[i]);

    printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2/ejer5  2021-04-26 lunes
$gcc -O2 -fopenmp -o copyprivate_clause copyprivate_clause.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2/ejer5  2021-04-26 lunes
$./copyprivate_clause

Introduce valor de inicializaciÃ¡n a: 7

Single ejecutada por el thread 2
DepuÃŠs de la regiÃ¡n parallel:
b[0] = 22047    b[1] = 0        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:

Imprime el resultado anterior más los 10 de la inicialización. Esto se debe a que reduction(+:suma) hace que, además de acumular el resultado en la variable suma, se le añada las 10 unidades con las que se inicializa la variable.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

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:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2/ejer6  2021-04-26 lunes
$./reduction-clause 12
Tras 'parallel' suma=66
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp2/ejer6  2021-04-26 lunes
$./reduction-clauseModificado 12
Tras 'parallel' suma=76
```

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: Una posible alternativa es usar firstprivate() con la variable suma y después usar la directiva atomic, de forma que en suma_final se almacene el resultado de todas las sumas parciales realizadas por las hebras, sincronizándolas y asegurando la exclusión mutua.
(Otra alternativa, menos eficiente, sería critical)

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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, suma_final=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 firstprivate(suma)
    {
      #pragma omp for
        for (i=0; i<n; i++)
            suma += a[i];

        #pragma omp atomic
          suma_final += suma;
    }

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

CAPTURAS DE PANTALLA:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp2/ejer7  2021-04-26 lunes
$gcc -O2 -fopenmp -o reduction-clauseModificado7 reduction-clauseModificado7.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp2/ejer7  2021-04-26 lunes
$./reduction-clauseModificado7 12
Tras 'parallel' suma_final=66
```

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`

pmv-secuencial.c

```
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h>  // biblioteca donde se encuentra la funciÃ³n printf()
#include <time.h> // biblioteca donde se encuentra la funciÃ³n clock_gettime()

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

//#define VECTOR_GLOBAL
#define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
//#define MAX 33554432  //=2^25
#define MAX 4294967295  //=2^32 -1
int  M[MAX][MAX], v1[MAX], v2[MAX];
#endif

int main(int argc, char** argv){
  int i;
  double ncgt, cgt1,cgt2; //para tiempo de ejecuciÃ³n
  int suma=0;
  unsigned int N = atoi(argv[1]); // MÃ¡ximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
  printf("TamaÃ±o Vectores:%u (%u B)\n",N, sizeof(unsigned int));


  if (argc<2){
    printf("Falta el tamanio\n");
    exit(-1);
  }

  #ifdef VECTOR_GLOBAL
  if (N>MAX) N=MAX;
  #endif

  #ifdef VECTOR_DYNAMIC
  int *v1, *v2, **M;
    v1 = (int*) malloc(N*sizeof(int));// malloc necesita el tamaño en bytes
    v2 = (int*) malloc(N*sizeof(int)); //si no hay espacio suficiente malloc devuelve NULL
    M = (int**) malloc(N*sizeof(int *));

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

   if ( (M==NULL) || (v1==NULL) || (v2==NULL) ){
   printf("Error en la reserva de espacio para los vectores\n");
   exit(-2);
 }
  #endif
```

```
48
49      //Inicializacion de la matriz y vectores
50      for(int fil=0 ; fil<N ; fil++){
51        for(int col=0 ; col<N ; col++)
52           M[fil][col]=fil+col;
53
54        v1[fil]=fil;
55        v2[fil]=0;
56       }
57
58       //Medimos el tiempo
59       cgt1 = omp_get_wtime();
60
61       // Cálculo de v2
62       for (int fil=0 ; fil<N; fil++){
63         for(int col=0 ; col<N ; col++)
64            v2[fil] += M[fil][col] * v1[col];
65       }
66
67       cgt2= omp_get_wtime();
68       ncgt=cgt2-cgt1;
69
70       //Imprimimos resultado y tiempo de ejecucion
71       printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
72
73       //Imprimir todos los componentes de v2 (hasta N < 20)
74       if (N < 20)
75          for (int i = 0; i < N; i++)
76             printf(" V2[%d]=%d ", i, v2[i]);
77       else{
78          printf("V2[0]=%d ", v2[0]);
79          printf("V2[%d]=%d ", N-1, v2[N-1]);
80       }
81       printf("\n");
82
83       for(int i=0 ; i<N ; i++)
84        free(M[i]);
85
86       free(M);
87       free(v1); //libera el espacio reservado para v1
88       free(v2); //libera el espacio reservado para v2
89
90       return 0;
91    }
```

CAPTURAS DE PANTALLA:

- **N = 8**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp2/ejer8  2021-04-26 lunes
$sbatch -p ac --wrap "./pmv-secuencial 8"
Submitted batch job 98368
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp2/ejer8  2021-04-26 lunes
$cat slurm-98368.out
TamaÃ±o Vectores:8 (4 B)
Tiempo(seg.):0.000000402          / Tamaño Vectores:8
 V2[0]=140  V2[1]=168  V2[2]=196  V2[3]=224  V2[4]=252  V2[5]=280  V2[6]=308
 V2[7]=336
```

- **N = 11**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp2/ejer8  2021-04-26 lunes
$sbatch -p ac --wrap "./pmv-secuencial 11"
Submitted batch job 98376
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp2/ejer8  2021-04-26 lunes
$cat slurm-98376.out
TamaÃ±o Vectores:11 (4 B)
Tiempo(seg.):0.000000592          / Tamaño Vectores:11
 V2[0]=385  V2[1]=440  V2[2]=495  V2[3]=550  V2[4]=605  V2[5]=660  V2[6]=715
 V2[7]=770  V2[8]=825  V2[9]=880  V2[10]=935
```

****Nota: el resto de ejercicios no me ha dado tiempo a hacerlos**

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`

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

RESPUESTA:

CAPTURAS DE PANTALLA:

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`

RESPUESTA:

CAPTURAS DE PANTALLA:

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):

JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:

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

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

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		----		----		----		----		----		----
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
32												

COMENTARIOS SOBRE LOS RESULTADOS: