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): Antonio David Villegas Yeguas
Grupo de prácticas y profesor de prácticas: B1
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: Vemos como el compilador nos avisa de que no hemos declarado el ámbito de la variable n, y no sabe si ponerla como compartida o privada, debido a que hemos eliminado si valor por defecto

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

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

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

}
```

CAPTURAS DE PANTALLA:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer1] 2019-04-11 jueves
$gcc -O2 shared-clauseModificado.c -o shared-clauseModificado -fopenmp
shared-clauseModificado.c: En la función 'main':
shared-clauseModificado.c:15:12: error: no se especificó 'n' en el 'parallel' que lo contiene
    #pragma omp parallel for default(none) shared(a)
            ^~~
shared-clauseModificado.c:15:12: error: 'parallel' contenedora
```

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer1] 2019-04-11 jueves
$gcc -O2 shared-clauseModificado.c -o shared-clauseModificado -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer1] 2019-04-11 jueves
$export OMP_DYNAMIC=FALSE
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer1] 2019-04-11 jueves
$export OMP_NUM_THREADS=4
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer1] 2019-04-11 jueves
$./shared-clauseModificado
Despues de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer1] 2019-04-11 jueves
$
```

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: Vemos como si mostramos la suma fuera del parallel, inicializando suma en el parallel, nos muestra el valor de cada una de la suma en los distintos threads, mientras que fuera nos muestra 0, debido a que la directiva private no nos garantiza el valor de entrada y salida al usar la clausula, es decir, la variable puede tener valores de entrada y salida indeterminados. Esto lo vemos si inicializamos suma fuera del parallel, que vemos que nos da valores indeterminados.

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)
   {

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

   }

   printf("\n* suma = %d", suma);

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

CAPTURAS DE PANTALLA:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
2] 2019-04-22 lunes
$gcc -O2 private-clauseModificado.c -o private-clauseModificado -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
2] 2019-04-22 lunes
$export OMP_DYNAMIC=FALSE
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
2] 2019-04-22 lunes
$export OMP_NUM_THREADS=4
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
2] 2019-04-22 lunes
$./private-clauseModificado
thread 0 suma a[0] /
thread 0 suma a[1] /
thread 3 suma a[6] /
thread 1 suma a[2] /
thread 1 suma a[3] /
thread 2 suma a[4] /
thread 2 suma a[5] /

* thread 3 suma = 6
* thread 1 suma = 5
* thread 0 suma = 1
* thread 2 suma = 9
* suma = 0
```

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
2] 2019-04-22 lunes
$gcc -O2 private-clauseModificado.c -o private-clauseModificado -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
2] 2019-04-22 lunes
$./private-clauseModificado
thread 0 suma a[0] /
thread 0 suma a[1] /
thread 1 suma a[2] /
thread 1 suma a[3] /
thread 3 suma a[6] /
thread 2 suma a[4] /
thread 2 suma a[5] /

* thread 1 suma = -224906667
* thread 3 suma = -224906666
* thread 2 suma = -224906663
* thread 0 suma = 5
* suma = 0
```

3. ¿Qué ocurre si en `private-clause.c` se elimina la cláusula `private(suma)`? ¿A qué cree que es debido?
   RESPUESTA: Por defecto se vuelve una variable compartida, luego todos los threads comparten el mismo valor

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

   }

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

CAPTURAS DE PANTALLA:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
3] 2019-04-22 lunes
$gcc -O2 private-clauseModificado3.c -o private-clauseModificado3 -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
3] 2019-04-22 lunes
$./private-clauseModificado3
thread 0 suma a[0] /
thread 0 suma a[1] /
thread 1 suma a[2] /
thread 1 suma a[3] /
thread 2 suma a[4] /
thread 2 suma a[5] /
thread 3 suma a[6] /

* thread 2 suma = 11
* thread 0 suma = 11
* thread 1 suma = 11
* thread 3 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 (añada capturas de pantalla que muestren lo que ocurre).
   RESPUESTA: Si, ya que realiza una copia de la ultima iteración que se habría ejecutado en secuencial sobre la variable a la que se aplica la clausula lastprivate.

CAPTURAS DE PANTALLA:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
4] 2019-04-22 lunes
$./firstprivate-clause
thread 0 suma a[0] suma = 0
thread 0 suma a[1] suma = 1
thread 1 suma a[2] suma = 2
thread 1 suma a[3] suma = 5
thread 3 suma a[6] suma = 6
thread 2 suma a[4] suma = 4
thread 2 suma a[5] suma = 9

Fuera de la construccion parallel suma = 6
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
4] 2019-04-22 lunes
$./firstprivate-clause
thread 0 suma a[0] suma = 0
thread 0 suma a[1] suma = 1
thread 2 suma a[4] suma = 4
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[5] suma = 9

Fuera de la construccion parallel suma = 6
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
4] 2019-04-22 lunes
$./firstprivate-clause
thread 0 suma a[0] suma = 0
thread 0 suma a[1] suma = 1
thread 3 suma a[6] suma = 6
thread 2 suma a[4] suma = 4
thread 2 suma a[5] suma = 9
thread 1 suma a[2] suma = 2
thread 1 suma a[3] suma = 5

Fuera de la construccion 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: El nuevo valor de la variable a solo lo tendrá la copia privada de la hebra que ejecuta la sección single, luego las otras hebras no modificaran el valor de a por el leído por teclado.

   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// copyprivate(a)
      {
         printf("\nIntroduce valor de inicializacion 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("Despues de la region parallel\n");

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



}
```

CAPTURAS DE PANTALLA:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
5] 2019-04-22 lunes
$gcc -O2 copyprivate-clause.c -o copyprivate-clause -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
5] 2019-04-22 lunes
$export OMP_NUM_THREADS=4
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
5] 2019-04-22 lunes
$export OMP_DYNAMIC=FALSE
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
5] 2019-04-22 lunes
$./copyprivate-clause

Introduce valor de inicializacion a: 1

Single ejecutada por el thread 0
Despues de la region parallel
b[0] = 1
b[1] = 1
b[2] = 1
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: Vemos como al realizar la modificación, el valor final de suma se incrementa en 10, porque la reducción suma tanto las copias privadas como el valor que tenia la variable antes del parallel

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:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
6] 2019-04-22 lunes
$gcc -O2 reduction-clause.c -o reduction-clause -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
6] 2019-04-22 lunes
$export OMP_DYNAMIC=FALSE
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
6] 2019-04-22 lunes
$export OMP_NUM_THREADS=4
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
6] 2019-04-22 lunes
$./reduction-clause 5
Tras parallel suma = 10
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
6] 2019-04-22 lunes
$gcc -O2 reduction-clause.c -o reduction-clause -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
6] 2019-04-22 lunes
$./reduction-clause 5
Tras parallel suma = 20
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
6] 2019-04-22 lunes
$
```

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: He añadido una directiva para que la suma sea una variable compartida, pero se realice la operación de forma atómica, evitando errores de lectura/escritura simultanea

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 //reduction(+:suma)
   for (i = suma = 0; i < n; i++){
      #pragma omp atomic
      suma += a[i];
   }
   printf("Tras parallel suma = %d\n", suma);

}
```

CAPTURAS DE PANTALLA:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
7] 2019-04-24 miércoles
$gcc -O2 reduction-clauseModificado7.c -o reduction-clauseModificado7 -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
7] 2019-04-24 miércoles
$export OMP_DYNAMIC=FALSE
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
7] 2019-04-24 miércoles
$export OMP_NUM_THREADS=4
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
7] 2019-04-24 miércoles
$./reduction-clauseModificado7 5
Tras parallel suma = 10
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
7] 2019-04-24 miércoles
$./reduction-clauseModificado7 5
Tras parallel suma = 10
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
7] 2019-04-24 miércoles
$
```

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`

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

int main(int argc, char ** argv){

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

   int n = atoi(argv[1]);

   double ** m = (double **) malloc (n*sizeof(double*));

   for (int i = 0; i < n ; i++){
      m[i] = (double *) malloc (n * sizeof(double) );
   }


   double * v = (double *) malloc (n * sizeof(double));

   double * r = (double *) malloc (n * sizeof(double));

   //velores iniciales
   for (int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
         m[i][j] = i + j;
      }
      v[i] = i;
   }


   struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecuci♦n

   clock_gettime(CLOCK_REALTIME,&cgt1);

   // multiplicacion
   for (int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
         r[i] += m[i][j] * v[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 < 10){
      for (int i = 0; i < n; i++)
         printf("r[%d]=%f\n", i, r[i]);
    }

    else printf("\nTiempo (seg): %11.9f\t Tamaño: %d\tv2[0]=%f\tv[%d]=%f\n", ncgt, n, r[0], n -









   for (int i = 0; i < n ; i++){
      free(m[i]);
   }

   free(m);



}
```

CAPTURAS DE PANTALLA:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
8] 2019-04-24 miércoles
$gcc pmv-secuencial.c -o pmv-secuencial -O2
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
8] 2019-04-24 miércoles
$./pmv-secuencial 3
r[0]=5.000000
r[1]=8.000000
r[2]=11.000000
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
8] 2019-04-24 miércoles
$./pmv-secuencial 11

Tiempo (seg): 0.000064570        Tamaño: 11     v2[0]=385.000000        v[10]=935.000000
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP2/ejer
8] 2019-04-24 miércoles
$
```

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

CAPTURAS DE PANTALLA (que justifique el código elegido):

TABLA (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):

COMENTARIOS SOBRE LOS RESULTADOS: