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): Mario Garcia Marquez
Grupo de prácticas y profesor de prácticas: Maribel, Grupo 1
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. **(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:
Al incluir default(none) se fuerza al programador a incluir que variables hay que compartir, entonces hay que compartir tambien la variable n pues no se compartiria de forma automatica como ocurria en el programa original.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```
BP2_GarciaMarquezMario_Y.odt  ejer1
ac/bp2/ejer1 on  main [!?]
❯ bat shared-clauseModificado.c
       File: shared-clauseModificado.c
   1   #include <stdio.h>
   2   #ifdef _OPENMP
   3   #include <omp.h>
   4   #endif
   5
   6   int main() {
   7     int i, n = 7;
   8     int a[n];
   9
  10     for (i = 0; i < n; i++)
  11       a[i] = i + 1;
  12
  13   #pragma omp parallel for default(none) shared(a, n)
  14     for (i = 0; i < n; i++)
  15       a[i] += i;
  16
  17     printf("Después de parallel for:\n");
  18
  19     for (i = 0; i < n; i++)
  20       printf("a[%d] = %d\n", i, a[i]);
  21   }
```

CAPTURAS DE PANTALLA:

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:

Fuera de parallel se imprime el valor 0 que es al que se inicializa en private, esto se debe a que la variable copiada se inicializa al valor especificado en la region.

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 = 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("\nSuma: %d", suma);

  printf("\n");

  return 0;
}
```

CAPTURAS DE PANTALLA:

(b) RESPUESTA:

Unicamente se suma bien en el caso de la hebra 0(master) dado que private genera una copia no inicializada para cada hilo, haciendo asi que se almacene codigo basura en la variable suma en los demas hilos. Finalmente se imprime 0 que es el valor inicializado.

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 = 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] / ", omp_get_thread_num(), i);
    }
    printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
  }

  printf("\nSuma: %d", suma);

  printf("\n");

  return 0;
}
```

CAPTURAS DE PANTALLA:

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

RESPUESTA:

Al dejar de ser privada, la variable pasa a ser compartida y por ello tenemos una condicion de carrera tanto en la inicializacion de la variable como en la suma. Es por esto que la suma no siempre dara el resultado correcto.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

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

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

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

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

  printf("\n");

  return 0;
}
```

CAPTURAS DE PANTALLA:

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:

El codigo pasa a imprimir 9 que es el valor que toma la variable suma tras la ultima iteracion de la ultima hebra. Esto se debe a la clausula lastprivate(suma)

CAPTURAS DE PANTALLA:

```
ac/bp2/ejer4 on  main [!?] took 9s
❯ gcc -O2 -fopenmp firstlastprivate-clauseModificado.c
ac/bp2/ejer4 on  main [!?]
❯ ls ..
2  BP2_GarciaMarquezMario_Y.odt  ejer1  ejer2  ejer3  ejer4
ac/bp2/ejer4 on  main [!?]
❯ ./a.out
 thread 5 suma a[5] suma=5
 thread 8 suma a[8] suma=8
 thread 2 suma a[2] suma=2
 thread 4 suma a[4] suma=4
 thread 6 suma a[6] suma=6
 thread 3 suma a[3] suma=3
 thread 1 suma a[1] suma=1
 thread 9 suma a[9] suma=9
 thread 7 suma a[7] suma=7
 thread 0 suma a[0] suma=0
```

(b) RESPUESTA:
Podria quitando la clausula lastprivate, si ponemos en su lugar por ejemplo lastprivate pasara a mostrar el valor de la primera iteracion, o poniendo la variable suma como shared se actuaria como en ejercicios anteriores. Dicho esto, sin cambiar las clausulas no es posible.
CAPTURAS DE PANTALLA:

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:
La variable a empieza a contener valores basura, esto se debe a que la inicializacion de la variable privada a se hacia por difusion gracias a la clausula copyprivate, al caracer de esta la variable solo es inicializada en la hebra que realiza el codigo single.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
ac/bp2/ejer5 on  main [!?]
❯ ls ..
   BP2_GarciaMarquezMario_Y.odt  ejer1  ejer2  ejer3  ejer4  ejer5
ac/bp2/ejer5 on  main [!?]
❯ bat copyprivate-clauseModificado.c
```

File: **copyprivate-clauseModificado.c**

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

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

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

#pragma omp parallel
  {
    int a;
#pragma omp single
    {
      printf("\nIntroduce valor de inicialización a: ");
      scanf("%d", &a);
      printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
    }
#pragma omp for
    for (i = 0; i < n; i++)
      b[i] = a;
  }

  printf("Depués de la región parallel:\n");
  for (i = 0; i < n; i++)
    printf("b[%d] = %d\t", i, b[i]);
  printf("\n");
```

CAPTURAS DE PANTALLA:

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:
La suma se sigue realizando correctamente solo que al resultado final se ha sumado 10, esto se debe a que se guarda el valor inicial de suma antes de reducion

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

File: **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:

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:

Se ha agregado el codigo atomic para poder realizar la suma de forma correcta si que haya ningun tipo de condicion de carrera.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

File: **reduction-clauseModificado2.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
  for (i = 0; i < n; i++)
#pragma omp atomic
    suma += a[i];

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

CAPTURAS DE PANTALLA:

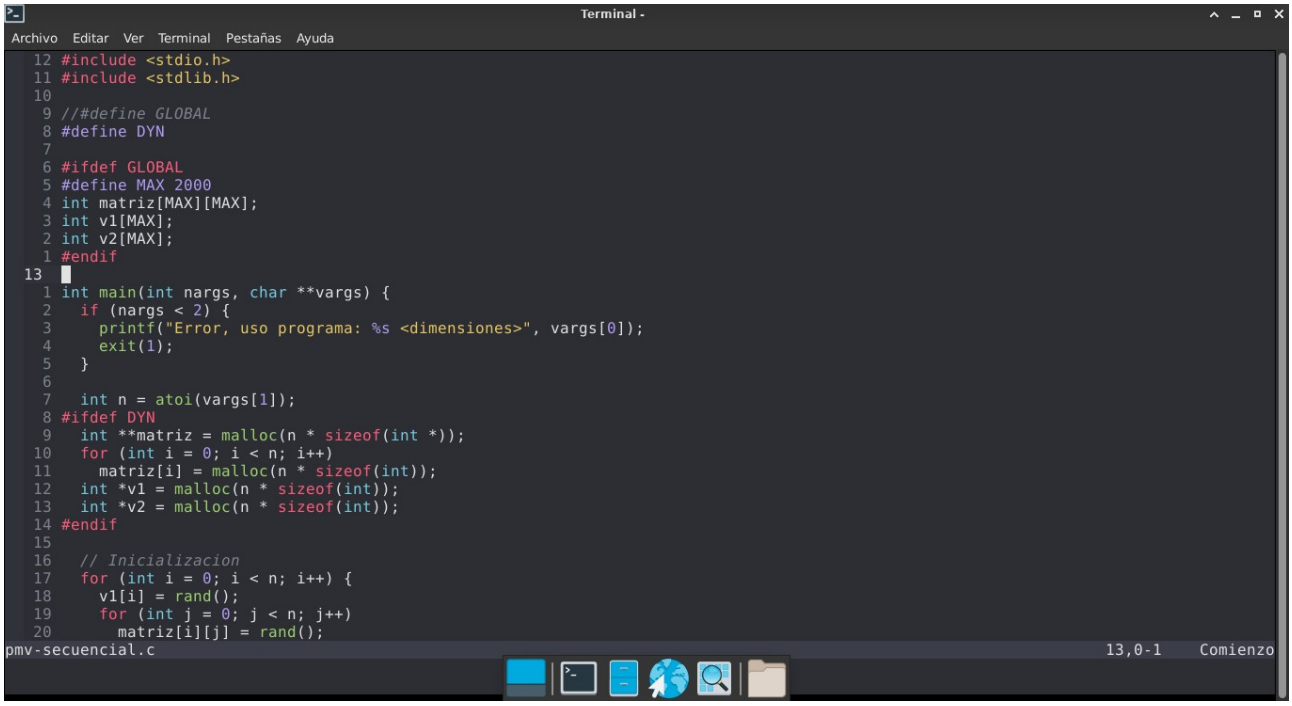
Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

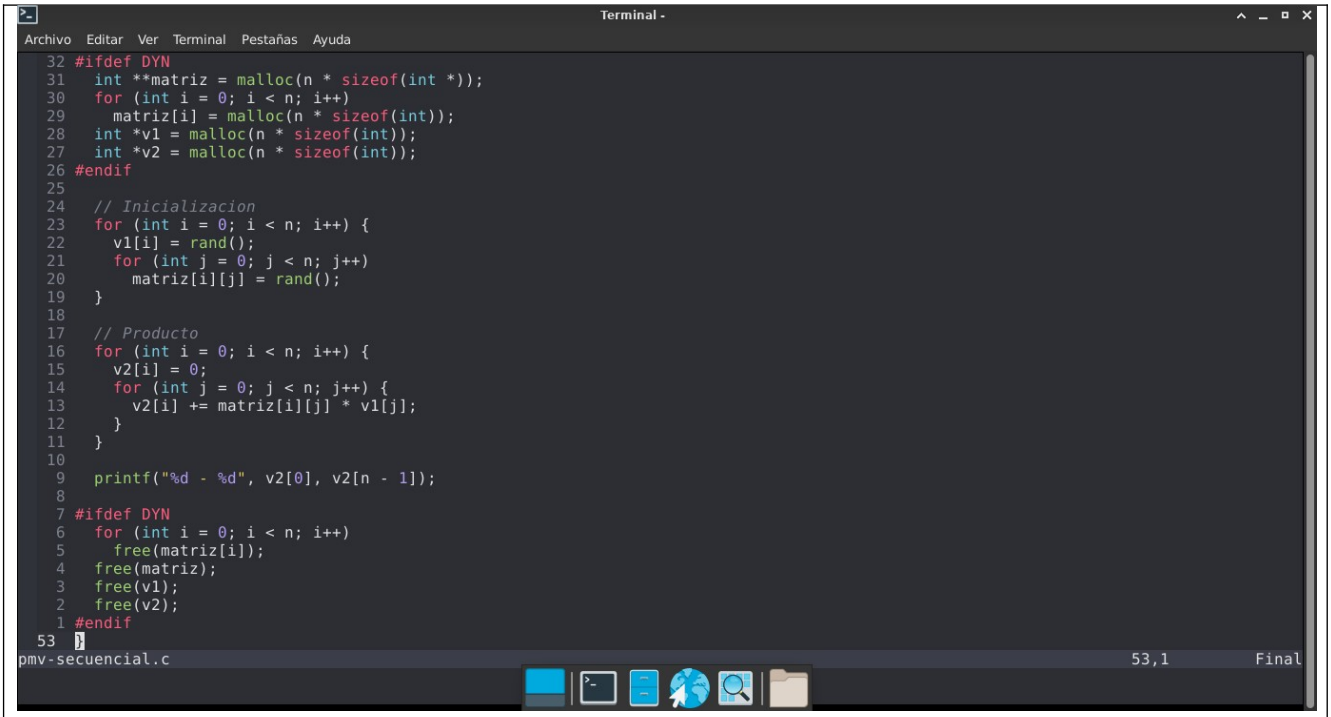
8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1 (implemente una versión para variables globales y otra para variables dinámicas, use una de estas versiones en los siguientes ejercicios):

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

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`





CAPTURAS DE PANTALLA:

```
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp2] 2021-04-24 sábado
$cd ejer8/
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp2/ejer8] 2021-04-24 sábado
$ls
codigo-1.png  codigo-2.png  pmv-secuencial.c
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp2/ejer8] 2021-04-24 sábado
$gcc pmv-secuencial.c -O2 -fopenmp
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp2/ejer8] 2021-04-24 sábado
$./a.out 11
-1237894986 - 566842365[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp2/ejer8] 2021-04-24 sábado
$
```

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

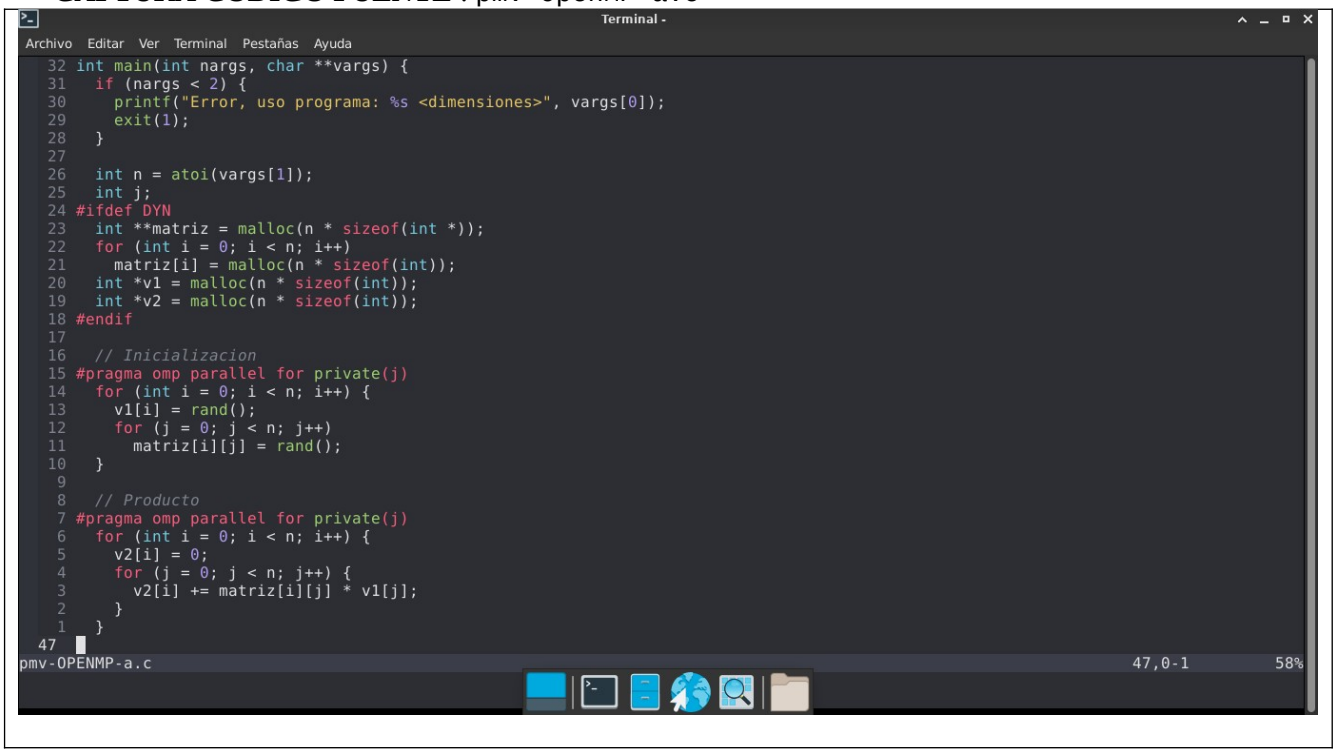
b. una segunda que paralelice el bucle que recorre las columnas.

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:

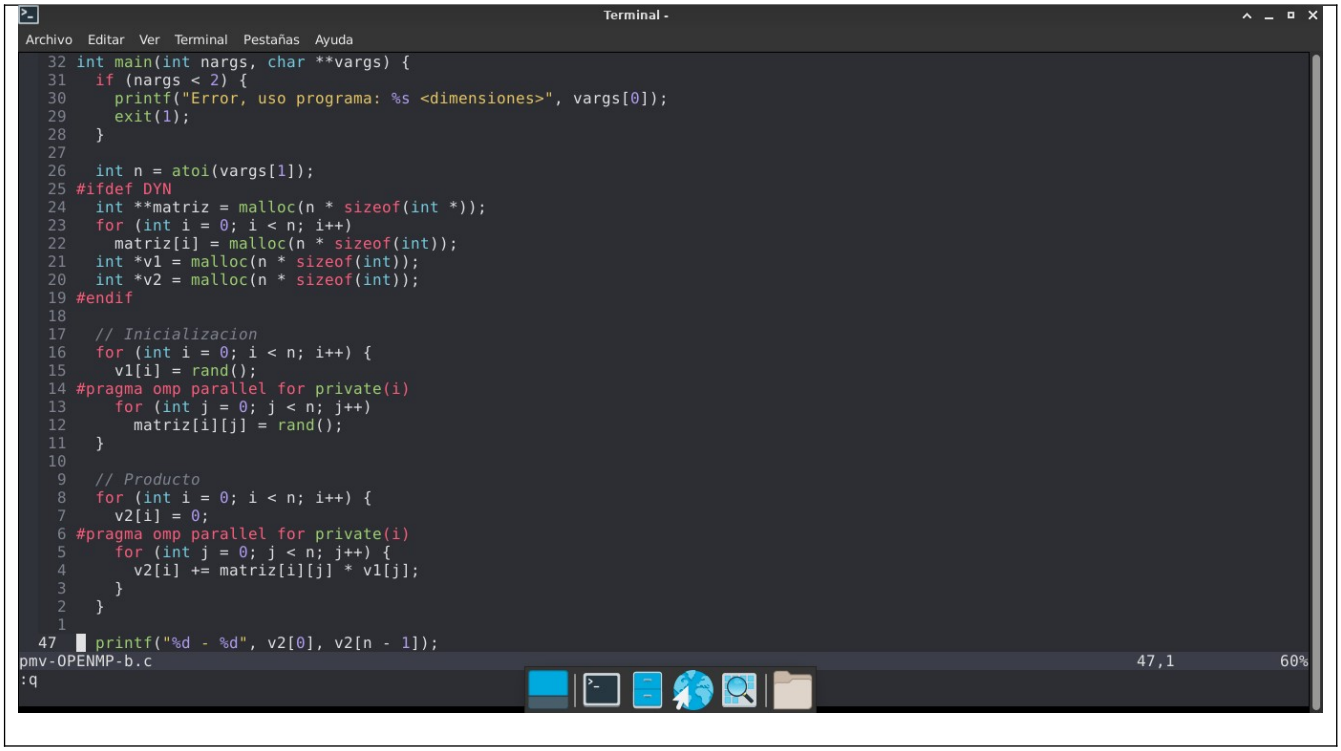
- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización 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`

```
29
30      // Inicializacion
31      for (int i = 0; i < n; i++) {
32        v1[i] = rand();
33        v2[i] = 0;
34    #pragma omp parallel for private(i)
35        for (int j = 0; j < n; j++)
36          matriz[i][j] = rand();
37      }
38
39      // Producto
40      int i;
41 -    int j = 0;
42      for (i = 0; i < n; i++) {
43 +      int suma = 0;
44 +  #pragma omp parallel reduction(+ : suma)
45        for (j = 0; j < n; j++) {
46 ~        suma += matriz[i][j] * v1[j];
47        }
48 +      v2[i] = suma;
49      }
50
51      printf("%d - %d", v2[0], v2[n - 1]);
52
53    #ifdef DYN
54      for (int i = 0; i < n; i++)
55        free(matriz[i]);
56      free(matriz);
57      free(v1);
58      free(v2);
59    #endif
60    }
```

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):

```
ac/bp2/ejer11 on  main [✗!?]
❯ ls
pmv-OPENMP-reduction.c  script.sh
ac/bp2/ejer11 on  main [✗!?]
❯ gcc -fopenmp -O2 pmv-OPENMP-reduction.c
ac/bp2/ejer11 on  main [✗!?]
❯ ./a.out 5000
-2136197408 - 0
time: 1.299113ac/bp2/ejer11 on  main [✗!?] took 5s
❯
ac/bp2/ejer11 on  main [✗!?] took 5s
❯ ls
a.out  pmv-OPENMP-reduction.c  script.sh
ac/bp2/ejer11 on  main [✗!?]
❯ cp ../ejer
ejer1/  ejer11/ ejer3/  ejer5/  ejer7/  ejer9/
ejer10/ ejer2/  ejer4/  ejer6/  ejer8/
ac/bp2/ejer11 on  main [✗!?]
❯ cp ../ejer9/
a.png           b.png           pmv-OPENMP-a.c  pmv-OPENMP-b.c
ac/bp2/ejer11 on  main [✗!?]
❯ cp ../ejer9/pmv-OPENMP-a.c .
ac/bp2/ejer11 on  main [✗!?]
❯ v .
ac/bp2/ejer11 on  main [✗!?] took 36s
❯ gcc -fopenmp -O2 pmv-OPENMP-.c
pmv-OPENMP-a.c          pmv-OPENMP-reduction.c
ac/bp2/ejer11 on  main [✗!?] took 36s
❯ gcc -fopenmp -O2 pmv-OPENMP-a.c.c
gcc: error: pmv-OPENMP-a.c.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
ac/bp2/ejer11 on  main [✗!?]
❯ gcc -fopenmp -O2 pmv-OPENMP-a.c
ac/bp2/ejer11 on  main [✗!?]
❯
ac/bp2/ejer11 on  main [✗!?]
❯ ./a.out 5000
-1660946476 - -1461017535
time: 0.003520ac/bp2/ejer11 on  main [✗!?] took 3s
```

JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:
Debido a que reduction crea una copia de suma para cada iteracion lo hace mas ineficaz que las otras dos versiones pues estas comparten todos los datos de forma simultanea. Ademas se prefiere iteracion por filas ya que a los vectores se accede por columnas lo que favorece la localidad espacial.

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

Alacritty

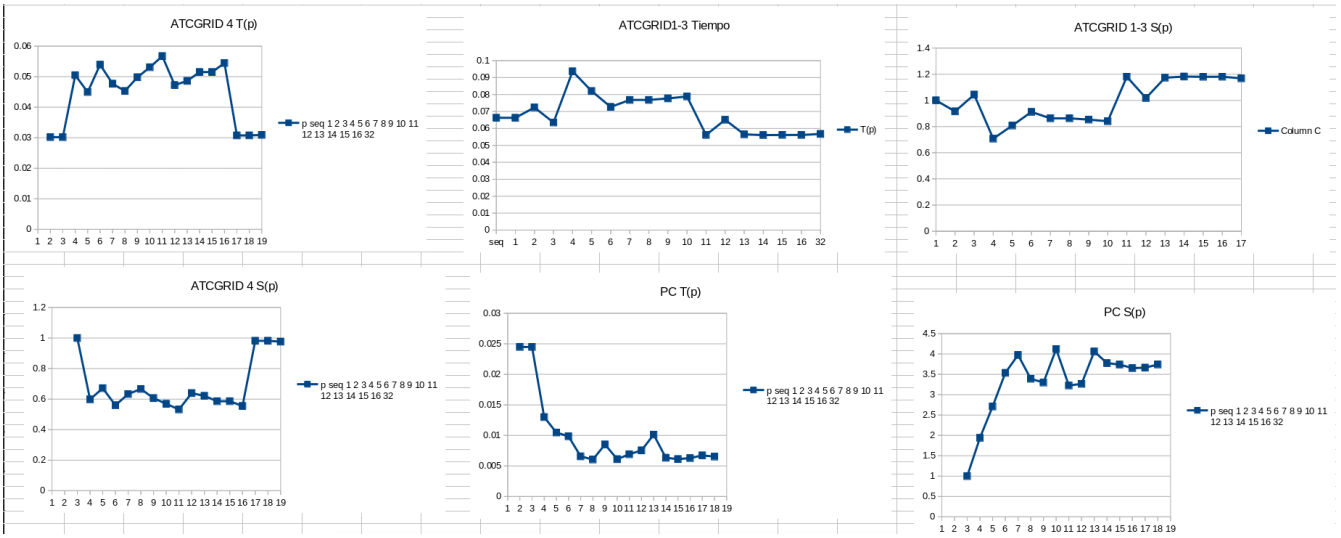
```
#!/bin/bash
#SBATCH --ntasks=1

# Script para atcgrid

vec=(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 32)
for i in "${vec[@]}"; do
    export OMP_NUM_THREADS=$i
    echo "$i 7k"
    ./a.out 7000
    echo "$i 50k"
    ./a.out 50000
done
```

```
pmv-OpenMP-script.sh                                    8,27-29       All
:!scrot -s
```

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

```
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp2/ejer11] 2021-04-24 sábado
$sbatch -p ac pmv-OpenMP-script.sh
Submitted batch job 97022
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp2/ejer11] 2021-04-24 sábado
$squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             97022        ac pmv-Open e1estudi  R       0:08      1 atcgrid1
```

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



COMENTARIOS SOBRE LOS RESULTADOS:

Para atcgrid solo se han ejecutado para 7000 componentes ya que el slurm rechazaba el script debido al alto tiempo de computo que implica inicializar la matriz con valores aleatorios, siendo este el 90% del tiempo de ejecucion del programa.

Se observa que en la cola AC1 solo se ve mejora a partir de usar 11 nucleos sobre la que se estabiliza el tiempo mientras que en la cola AC4 el rendimiento en general es peor que el caso secuencial. Sin embargo en mi pc la ganancia mejora sin importar los cores usados. Cabe tener en cuenta que a partir de los 6 hilos se obtiene la eficiencia maxima de 4. A partir de aquí el rendimiento se mantiene estable.