Grai2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Juan Miguel Castro Guerrero
Grupo de prácticas: B3
Fecha de entrega:
Fecha evaluación en clase:

---

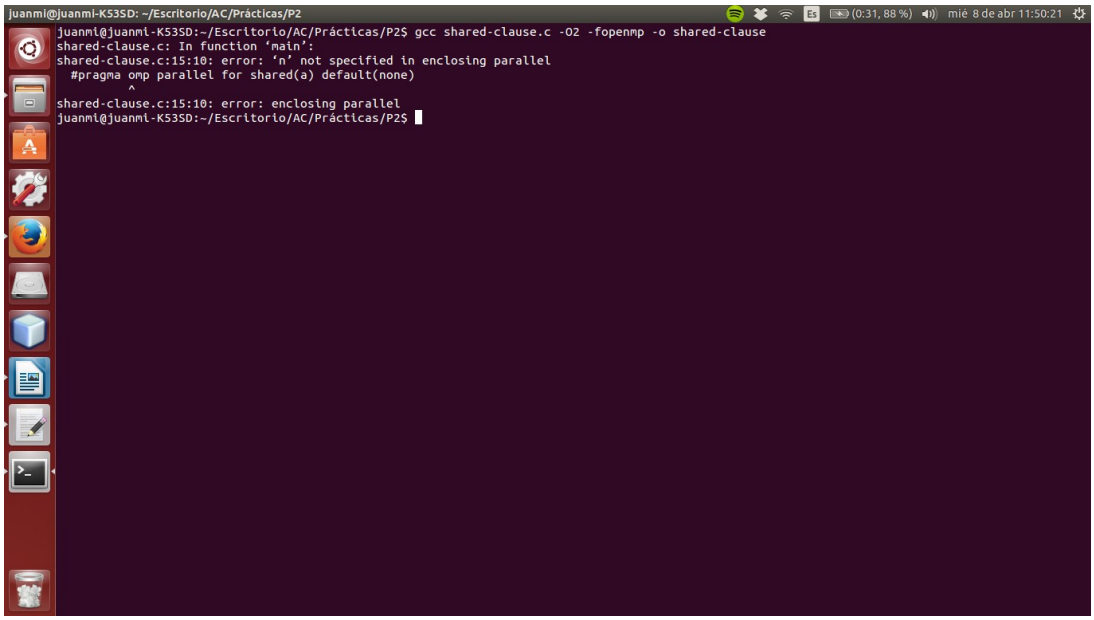
1. ¿Qué ocurre si en el ejemplo del seminario shared-clause.c se añade a la directiva parallel la cláusula default(none)? (añada una captura de pantalla que muestre lo que ocurre) **(b)** Resuelva el problema generado sin eliminar default(none).Añada el código con la modificación al cuaderno de prácticas.

   RESPUESTA: Da un error afirmando que el tipo de la variable n no está especificado en el parallel.

   CÓDIGO FUENTE: shared-clauseModificado.c

```
#include <stdio.h>
#ifdef _OPENMP
  #include <omp.h>
#endif

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

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

   #pragma omp parallel for shared(a,n) default(none)
   for (i=0; i<n; i++)    a[i] += i;

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

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

return 0;
}
```

   CAPTURAS DE PANTALLA:



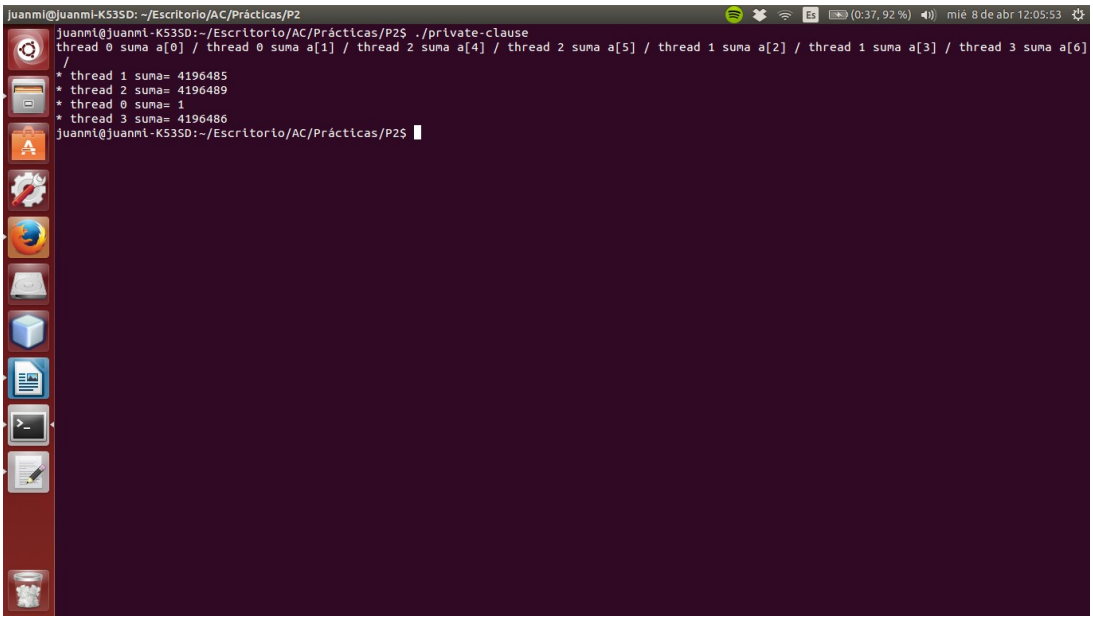
2. ¿Qué ocurre si en private-clause.c se inicializa la variable suma fuera de la construcción parallel en lugar de dentro? Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA: La hebra 0 realiza bien la suma ya que la variable suma ya estaba previamente inicializada a 0 y suma los valores del vector correctamente, las demás hebras también habrían realizado la suma correctamente sino fuera porque inicializaron la variable suma a 4196480.

CÓDIGO FUENTE: private-clauseModificado.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)
{
   //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:



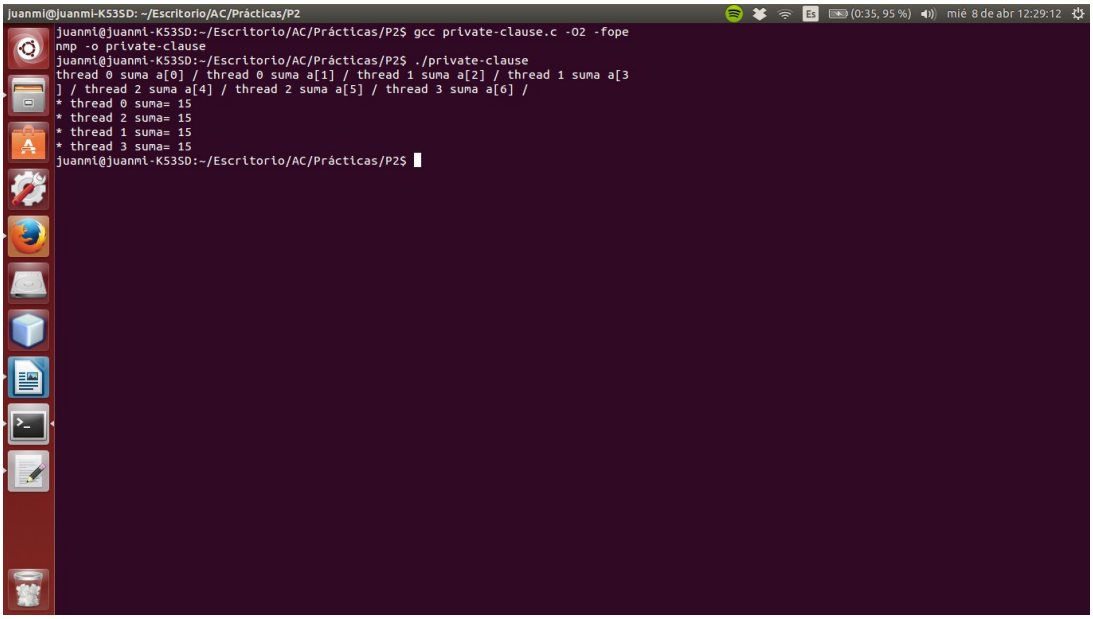
3. ¿Qué ocurre si en private-clause.c se elimina la cláusula private(suma)? ¿A qué cree que es debido?

RESPUESTA: Todas las hebras suman lo mismo, esto es debido a que si eliminamos la cláusula private(suma) todas las hebras acceden a la misma variable y no trabajan sobre una copia local de esa variable para cada hebra.

CÓDIGO FUENTE: private-clauseModificado3.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("\n");

   return 0;
}
```

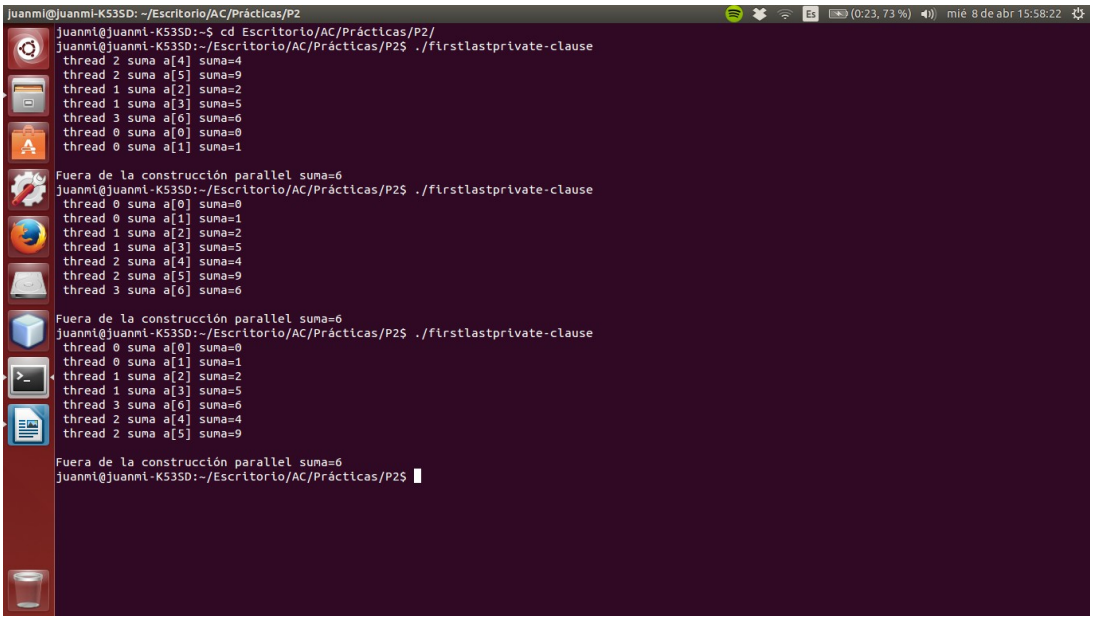
CAPTURAS DE PANTALLA:



4. En la ejecución de firstlastprivate.c de la página 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.

RESPUESTA: El código siempre imprime un 6 en la salida ya que se guarda en la suma el resultado de la última iteración debido a cláusula lastprivate.

CAPTURAS DE PANTALLA:



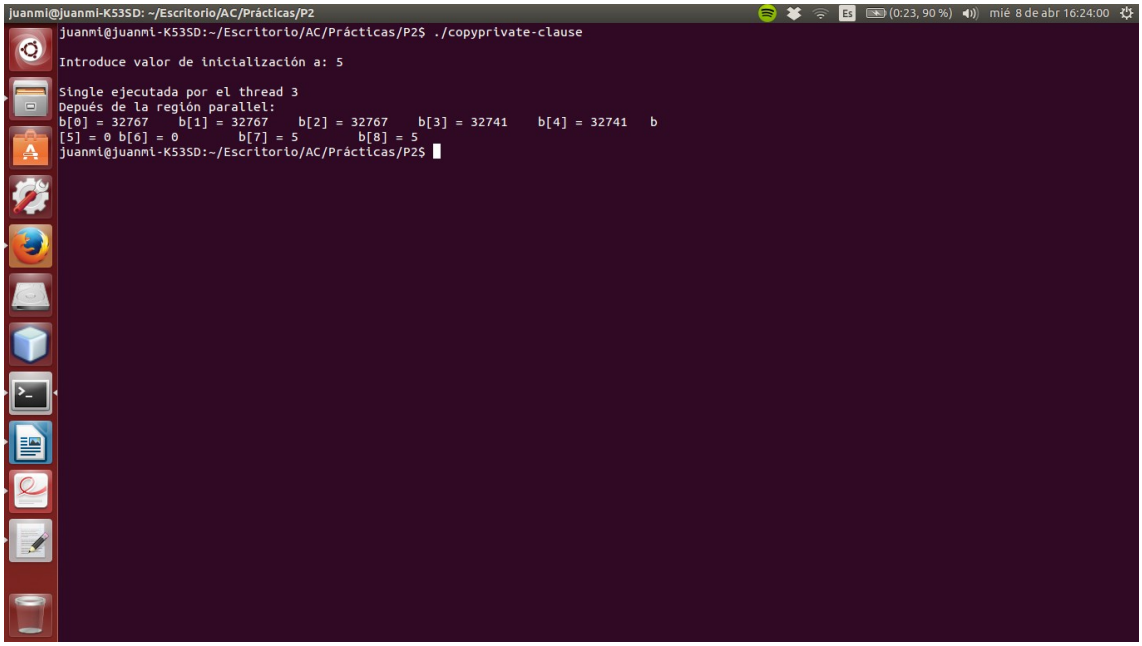
5. ¿Qué ocurre si en copyprivate-clause.c se elimina la cláusula copyprivate(a) en la directiva single? ¿A qué cree que es debido?

RESPUESTA: Al no utilizar copyprivate cada hebra que utilice la variable “a” no podrá utilizar la copia local de esa variable de otra hebra.

CÓDIGO FUENTE: copyprivate-clauseModificado.c

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

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

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

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

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

   return 0;
}
```

CAPTURAS DE PANTALLA:



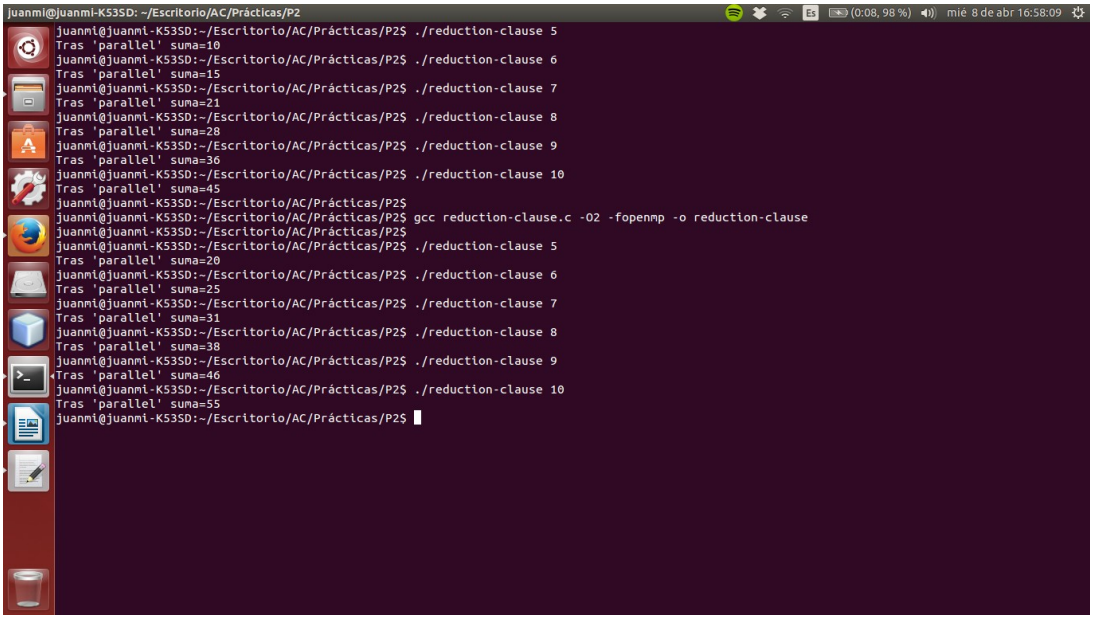
6. En el ejemplo reduction-clause.c sustituya suma=0 por suma=10. ¿Qué resultado se imprime ahora? Justifique el resultado

RESPUESTA: El resultado de cada iteración(cuando suma=10) es el resultado de cada iteración(cuando suma=0) más 10.

CÓDIGO FUENTE: reduction-clauseModificado.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:



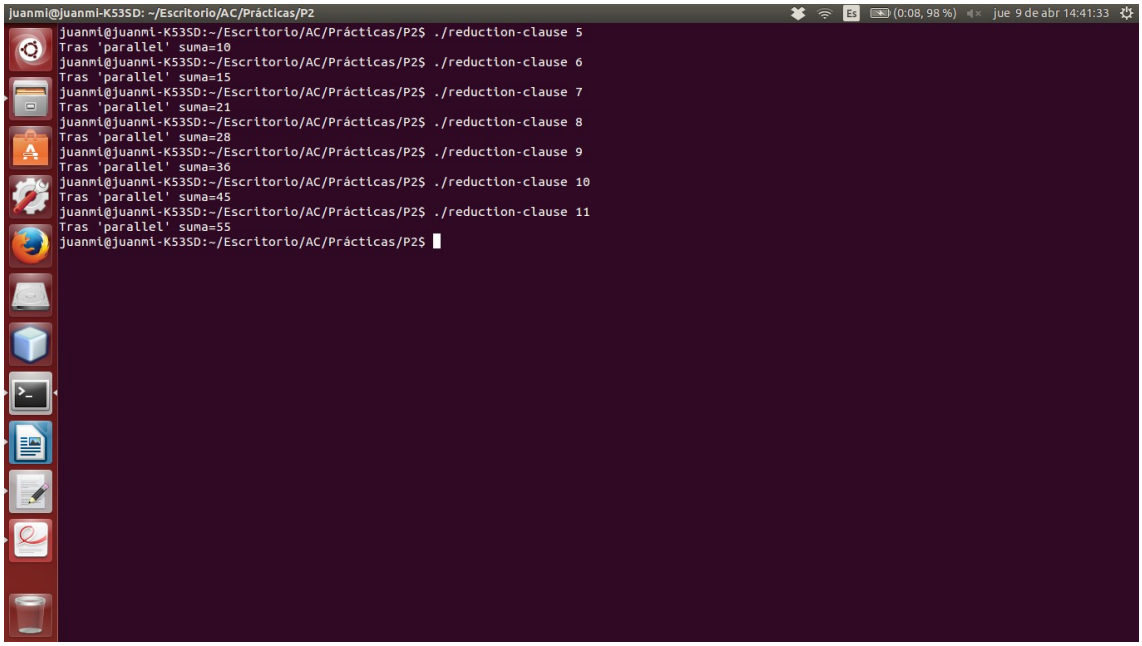
7. En el ejemplo reduction-clause.c, elimine reduction(+:suma) de #pragma omp parallel for reduction(+:suma) y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector a en paralelo.

RESPUESTA:

CÓDIGO FUENTE: reduction-clauseModificado7.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, sumalocal;

   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 private(sumalocal)
   {
             sumalocal=0;
             #pragma omp for
             for (i=0; i<n; i++)        sumalocal += a[i];
             #pragma omp critical
                          suma += sumalocal;
   }

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

CAPTURAS DE PANTALLA:



8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1:

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

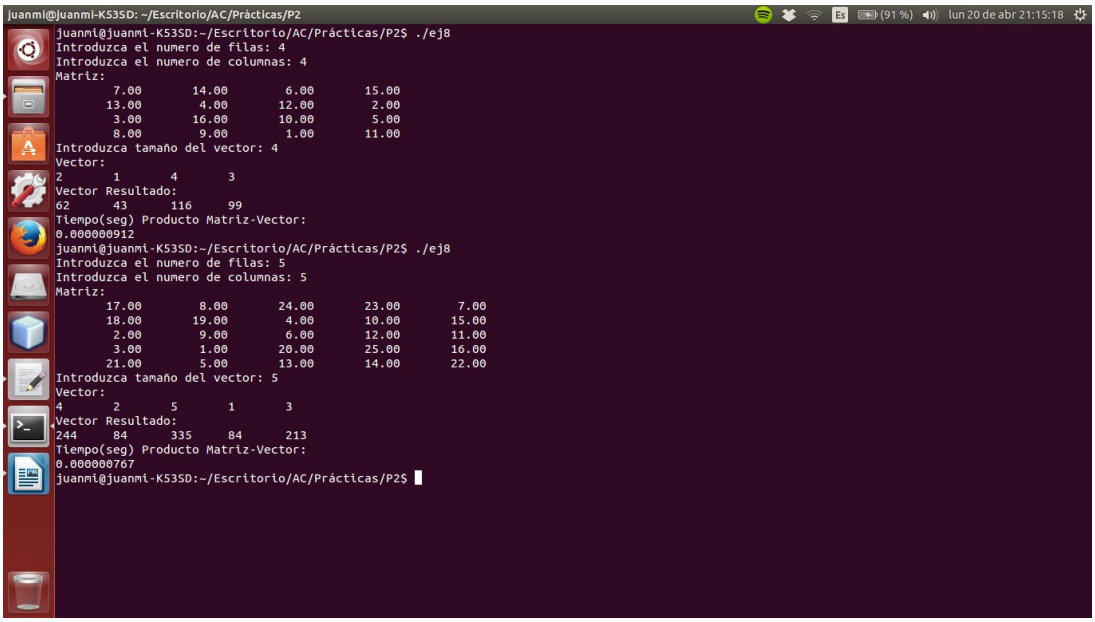
NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños 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).

CÓDIGO FUENTE: pmv-secuencial.c

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

int main(void)
{
    int z, y, x, i, g, temp, filas, columnas, tam;
              struct timespec cgt1,cgt2;
    double ncgt;

    printf ("Introduzca el numero de filas: ");
    scanf ("%i",&filas);
    printf ("Introduzca el numero de columnas: ");
    scanf ("%i",&columnas);

    double matriz[filas][columnas];
    temp = filas*columnas;

    for(i=0; i<filas; i++){
              for(g=0; g<columnas; g++)
                        matriz[i][g] = 0;
    }
```

```
    for(i=1; i<=temp; i++){
               do{
                              y = rand() % filas;
                              x = rand() % columnas;
               }while(matriz[y][x]);

               matriz[y][x] = i;
    }

    printf("Matriz:\n");
    for (i=0; i<filas; i++){
                              for(g=0; g<columnas; g++)
                                             printf("%12.2f",matriz[i][g]);
                              printf("\n");
    }

    printf ("Introduzca tamaño del vector: ");
    scanf ("%i",&tam);

    int v1[tam],v2[tam];

    for(i=0; i<tam; i++)
                              v1[i] = 0;

    for(i=0; i<tam; i++)
                              v2[i] = 0;

    for(i=1; i<=tam; i++){
               do{
                              z = rand() % tam;

               }while(v1[z]);

                              v1[z] = i;
    }

    printf("Vector:\n");
    for(i=0; i<tam; i++)
                              printf("%i\t",v1[i]);

    printf("\n");

               clock_gettime(CLOCK_REALTIME,&cgt1);
    for(i=0; i<filas; i++){
               for(g=0; g<columnas; g++)
                              v2[i]+=matriz[g][i]*v1[i];
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    printf("Vector Resultado:\n");
    for(i=0; i<tam; i++)
                              printf("%i\t",v2[i]);

               printf("\nTiempo(seg) Producto Matriz-Vector:\n");
               printf("%11.9f\t",ncgt);

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

CAPTURAS DE PANTALLA:



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).

CÓDIGO FUENTE : pmv-OpenMP-a.c

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
              #include <omp.h>
#else
```

```
            #define omp_get_thread_num() 0
#endif

int main(void)
{
    int z, y, x, i, g, temp, filas, columnas, tam;
            double inicio,final,diferencia;

    printf ("Introduzca el numero de filas: ");
    scanf ("%i",&filas);
    printf ("Introduzca el numero de columnas: ");
    scanf ("%i",&columnas);

    double matriz[filas][columnas];
    temp = filas*columnas;

    for(i=0; i<filas; i++){
            for(g=0; g<columnas; g++)
                        matriz[i][g] = 0;
    }

    for(i=1; i<=temp; i++){
            do{
                        y = rand() % filas;
                        x = rand() % columnas;
            }while(matriz[y][x]);

            matriz[y][x] = i;
    }

    printf("Matriz:\n");
    for (i=0; i<filas; i++){
                        for(g=0; g<columnas; g++)
                                    printf("%12.2f",matriz[i][g]);
                        printf("\n");
    }

    printf ("Introduzca tamaño del vector: ");
    scanf ("%i",&tam);

    int v1[tam],v2[tam];

    for(i=0; i<tam; i++)
                        v1[i] = 0;

    for(i=0; i<tam; i++)
                        v2[i] = 0;

    for(i=1; i<=tam; i++){
            do{
                        z = rand() % tam;

            }while(v1[z]);

                        v1[z] = i;
    }

    printf("Vector:\n");
    for(i=0; i<tam; i++)
                        printf("%i\t",v1[i]);

    printf("\n");
```

```
            inicio = omp_get_wtime();
            #pragma omp parallel for private(g)
    for(i=0; i<filas; i++)
            for(g=0; g<columnas; g++)
                        v2[i] += matriz[g][i]*v1[i];

    final = omp_get_wtime();

    diferencia = final - inicio;

    printf("Vector Resultado:\n");
    for(i=0; i<tam; i++)
                        printf("%i\t",v2[i]);

            printf("\nTiempo(seg) Producto Matriz-Vector:\n");
            printf("%f\t",diferencia);

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

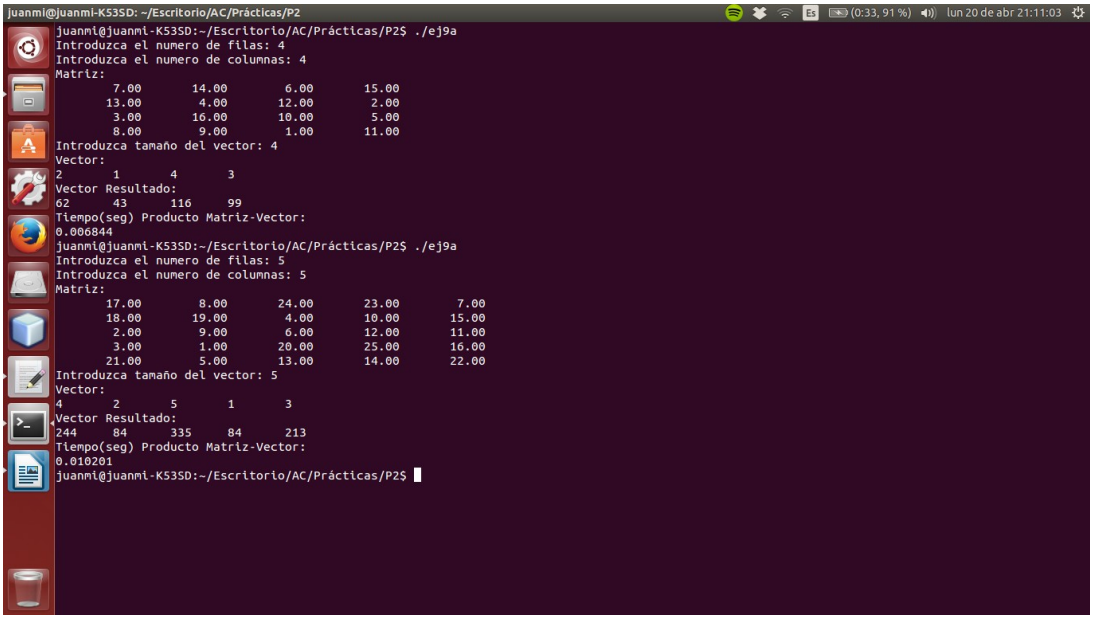
CÓDIGO FUENTE: pmv-OpenMP-b.c

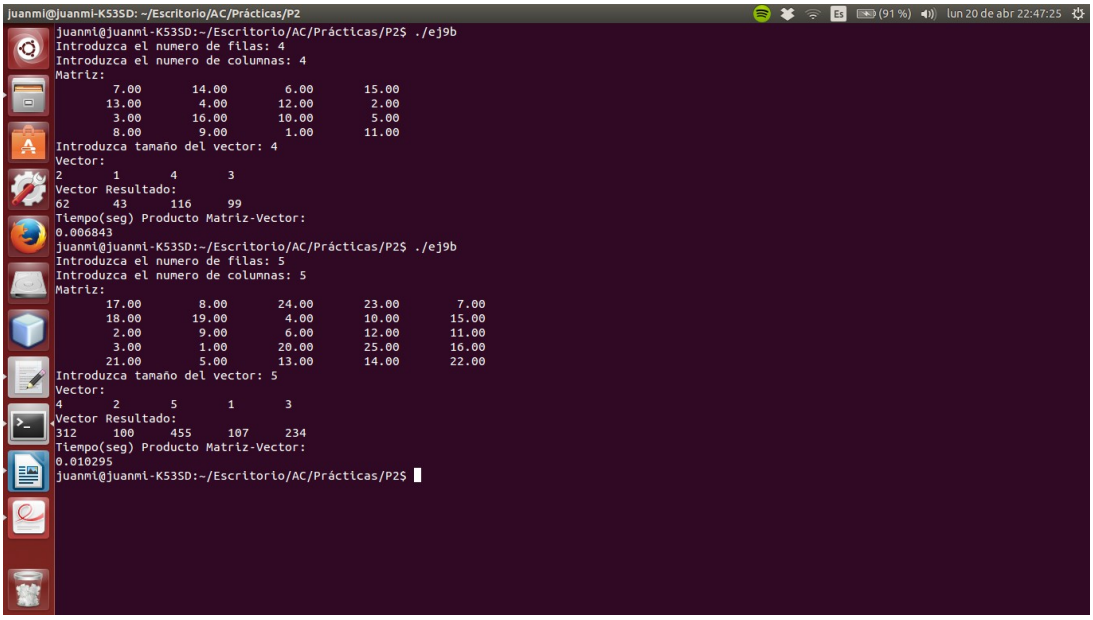
```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
            #include <omp.h>
#else
            #define omp_get_thread_num() 0
#endif

int main(void)
{
    int z, y, x, i, g, temp, filas, columnas, tam,suma=0;
            double inicio,final,diferencia;

    printf ("Introduzca el numero de filas: ");
    scanf ("%i",&filas);
    printf ("Introduzca el numero de columnas: ");
    scanf ("%i",&columnas);

    double matriz[filas][columnas];
    temp = filas*columnas;

    for(i=0; i<filas; i++){
            for(g=0; g<columnas; g++)
                        matriz[i][g] = 0;
    }

    for(i=1; i<=temp; i++){
            do{
                        y = rand() % filas;
                        x = rand() % columnas;
            }while(matriz[y][x]);

            matriz[y][x] = i;
    }

    printf("Matriz:\n");
    for (i=0; i<filas; i++){
                        for(g=0; g<columnas; g++)
                                    printf("%12.2f",matriz[i][g]);
                        printf("\n");
```

```
    }

    printf ("Introduzca tamaño del vector: ");
    scanf ("%i",&tam);

    int v1[tam],v2[tam];

    for(i=0; i<tam; i++)
                            v1[i] = 0;

    for(i=0; i<tam; i++)
                            v2[i] = 0;

    for(i=1; i<=tam; i++){
              do{
                            z = rand() % tam;

              }while(v1[z]);

                            v1[z] = i;
    }

    printf("Vector:\n");
    for(i=0; i<tam; i++)
                            printf("%i\t",v1[i]);

    printf("\n");

              inicio = omp_get_wtime();
    for(i=0; i<filas; i++){
              #pragma omp parallel private(suma)
              {
                            suma=0;
                            #pragma omp for
                            for(g=0; g<columnas; g++){
                                          suma += matriz[g][i]*v1[i];
                                          #pragma omp atomic
                                          v2[i] += suma;
                            }
              }
    }
    final = omp_get_wtime();

    diferencia = final - inicio;

    printf("Vector Resultado:\n");
    for(i=0; i<tam; i++)
                            printf("%i\t",v2[i]);

              printf("\nTiempo(seg) Producto Matriz-Vector:\n");
              printf("%f\t",diferencia);

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

RESPUESTA: No se han dado errores en compilación o ejecución, tan solo tres warnings con respecto a la función scanf().

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).

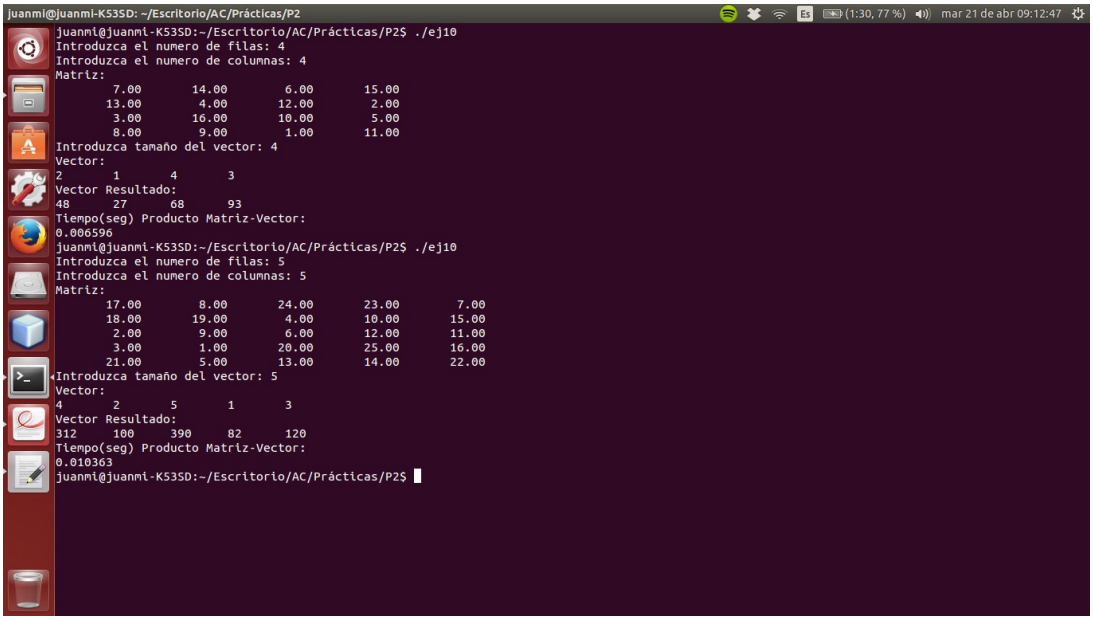
CÓDIGO FUENTE: pmv-OpenmMP-reduction.c

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

int main(void)
{
    int z, y, x, i, g, temp, filas, columnas, tam,suma=0;
            double inicio,final,diferencia;

    printf ("Introduzca el numero de filas: ");
    scanf ("%i",&filas);
    printf ("Introduzca el numero de columnas: ");
    scanf ("%i",&columnas);

    double matriz[filas][columnas];
    temp = filas*columnas;

    for(i=0; i<filas; i++){
            for(g=0; g<columnas; g++)
                        matriz[i][g] = 0;
    }

    for(i=1; i<=temp; i++){
            do{
                        y = rand() % filas;
                        x = rand() % columnas;
            }while(matriz[y][x]);

            matriz[y][x] = i;
    }

    printf("Matriz:\n");
    for (i=0; i<filas; i++){
                        for(g=0; g<columnas; g++)
                                    printf("%12.2f",matriz[i][g]);
                        printf("\n");
    }

    printf ("Introduzca tamaño del vector: ");
    scanf ("%i",&tam);

    int v1[tam],v2[tam];

    for(i=0; i<tam; i++)
                        v1[i] = 0;

    for(i=0; i<tam; i++)
                        v2[i] = 0;

    for(i=1; i<=tam; i++){
            do{
                        z = rand() % tam;

            }while(v1[z]);

                        v1[z] = i;
    }
```

```
    printf("Vector:\n");
    for(i=0; i<tam; i++)
                            printf("%i\t",v1[i]);

    printf("\n");

              inicio = omp_get_wtime();
    for(i=0; i<filas; i++){
              #pragma omp parallel for reduction(+:suma)
              for(g=0; g<columnas; g++){
                            suma += matriz[g][i]*v1[i];
                            v2[i] += suma;
              }
    }
    final = omp_get_wtime();

    diferencia = final - inicio;

    printf("Vector Resultado:\n");
    for(i=0; i<tam; i++)
                            printf("%i\t",v2[i]);

              printf("\nTiempo(seg) Producto Matriz-Vector:\n");
              printf("%f\t",diferencia);

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

RESPUESTA: No se han dado errores en compilación o ejecución, tan solo tres warnings con respecto a la función scanf().

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 el PC del aula de prácticas de los tres códigos implementados en los ejercicios anteriores para tres tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar –O2 al compilar.

    TABLA Y GRÁFICA (por *ejemplo* para 1-4 threads PC aula, y para 1-12 threads en atcgrid, tamaños-N-: 1.000, 10.000, 100.000):

    COMENTARIOS SOBRE LOS RESULTADOS: