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): Brian Sena Simons
Grupo de prácticas y profesor de prácticas: 2ºB
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: ***Default(none)*** es una especificación que indica que no se comparta aquellas variables que hayan sido declaradas anteriormente al bucle ***for*** que procede. De esta manera, como hacemos referencias a ellas en nuestro bucle, no indica un error por "falta de asignación" en las variables mencionadas. Una manera de solventar el problema manteniendo la clausula ***default(none)*** es una nueva especificación más detallada sobre que variables compartir tras la declaraciones con la clausula ***shared(…);***
(No indicamos que la variable de índice en el "shared" ya que es privada).

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```c
#include <stdio.h>

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

//gcc -O2 -fopenmp -o shared_clave shared_clauseModificado.c

int main(int argc, char const* argv[]){
  int i, n=7;
  int a[n];

#pragma omp parallel for default(none) shared(a,n)
  for(i=0;i<n;++i) a[i] = i + 1;

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

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

  return 0;

}
```

CAPTURAS DE PANTALLA:

```
2021-04-13-04:13 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer1$ nvim shared-clauseModificado.c
2021-04-13-04:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer1$ ./shared_clause
Despues del parallel for:
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 7
2021-04-13-04:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer1$
```

Error al escribir apenas default(none):

```
shared-clauseModificado.c: In function ‘main’:
shared-clauseModificado.c:13:9: error: ‘n’ not specified in enclosing ‘parallel’
   13 | #pragma omp parallel for default(none)
      |         ^~~
shared-clauseModificado.c:13:9: error: enclosing ‘parallel’
shared-clauseModificado.c:14:21: error: ‘a’ not specified in enclosing ‘parallel’
   14 |   for(i=0;i<n;++i) a[i] = i + 1;
      |                     ^~~
shared-clauseModificado.c:13:9: error: enclosing ‘parallel’
   13 | #pragma omp parallel for default(none)
      |         ^~~
2021-04-13-04:17 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer1$ nvim shared-clauseModificado.c
```

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 clausula “**private(…)**” lo que hace es que cada hebra posee la variable para ella sola y entonces al salir del bucle se pierde el valor que haya adquirido tras finalizado los cálculos computacionales dentro de la zona paralela y luego se devuelve el valor que había antes. En nuestro caso, por alguna razón devuelve 0 en la variable suma que no se ha inicializado con ningún valor.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.c`

```c
#include <stdio.h>

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

//gcc -O2 -fopenmp -o private_clause private-clauseModificado.c

int main(int argc, char const *argv[]){
  int i, n=7;
  int a[n], suma;

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

#pragma omp parallel private(suma)
  {
    suma=7;
#pragma omp for
    for(i=0;i<n;++i){
      suma = suma + a[i];
      printf("thread %d, suma a[%d] = %d / ", omp_get_thread_num(), i, a[i]);
    }
  }

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

  return 0;

}
```

CAPTURAS DE PANTALLA:

```
2021-04-13-04:30 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer2$ ./private-clauseModificado_a
thread 5, suma a[5] = 5 / thread 0, suma a[0] = 0 / thread 4, suma a[4] = 4 / thread 1, suma a[1] = 1 /
thread 6, suma a[6] = 6 / thread 2, suma a[2] = 2 / thread 3, suma a[3] = 3 /
thread 0 suma = 0
2021-04-13-04:30 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer2$ ./private-clauseModificado_a
thread 3, suma a[3] = 3 / thread 4, suma a[4] = 4 / thread 2, suma a[2] = 2 / thread 1, suma a[1] = 1 /
thread 0, suma a[0] = 0 / thread 5, suma a[5] = 5 / thread 6, suma a[6] = 6 /
thread 0 suma = 0
2021-04-13-04:30 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer2$ ./private-clauseModificado_a
thread 5, suma a[5] = 5 / thread 0, suma a[0] = 0 / thread 2, suma a[2] = 2 / thread 6, suma a[6] = 6 /
thread 4, suma a[4] = 4 / thread 1, suma a[1] = 1 / thread 3, suma a[3] = 3 /
thread 0 suma = 0
```

(b) RESPUESTA: Siguiendo el mismo hilo lógico de la respuesta del apartado a, podemos afirmar incluso antes de ejecutar el código que el resultado final será el asignado antes de entrar en la zona paralela, en nuestro caso ahora, 7.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_b.c`

```c
#include <stdio.h>

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

//gcc -O2 -fopenmp -o private_clause private-clauseModificado.c

int main(int argc, char const *argv[]){
  int i, n=7;
  int a[n], suma;

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

  suma = 7;
#pragma omp parallel private(suma)
  {
#pragma omp for
    for(i=0;i<n;++i){
      suma = suma + a[i];
      printf("thread %d, suma a[%d] = %d / ", omp_get_thread_num(), i, a[i]);
    }
  }

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

  return 0;

}
```

CAPTURAS DE PANTALLA:

```
2021-04-13-04:31 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer2$ ./private-clauseModificado_b
thread 0, suma a[0] = 0 / thread 1, suma a[1] = 1 / thread 6, suma a[6] = 6 / thread 4, suma a[4] = 4 /
thread 3, suma a[3] = 3 / thread 2, suma a[2] = 2 / thread 5, suma a[5] = 5 /
thread 0 suma = 7
2021-04-13-04:31 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer2$ ./private-clauseModificado_b
thread 1, suma a[1] = 1 / thread 3, suma a[3] = 3 / thread 0, suma a[0] = 0 / thread 2, suma a[2] = 2 /
thread 4, suma a[4] = 4 / thread 5, suma a[5] = 5 / thread 6, suma a[6] = 6 /
thread 0 suma = 7
2021-04-13-04:31 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer2$ ./private-clauseModificado_b
thread 0, suma a[0] = 0 / thread 1, suma a[1] = 1 / thread 2, suma a[2] = 2 / thread 4, suma a[4] = 4 /
thread 5, suma a[5] = 5 / thread 6, suma a[6] = 6 / thread 3, suma a[3] = 3 /
thread 0 suma = 7
```

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

RESPUESTA: Al quitar la cláusula **"private(..)"** lo que estamos permitiendo es que haya una paralelización implícita del código automáticamente. Por ende, siempre vamos a tener el resultado de sumarle los índices del vector a la variable anteriormente inicializada **"suma"**. (El resultado empezando en 0 es 21).

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```c
#include <stdio.h>

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

//gcc -O2 -fopenmp -o private_clause private-clauseModificado.c

int main(int argc, char const *argv[]){
  int i, n=7;
  int a[n], suma;

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

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

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

  return 0;

}
```

CAPTURAS DE PANTALLA:

```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer3] 2021-04-13 martes
$cat slurm-89256.out slurm-89265.out
thread 0, suma a[0] = 0 / thread 0, suma a[1] = 1 / thread 0, suma a[2] = 2 / thread 0, suma a[3] = 3 /
thread 1, suma a[4] = 4 / thread 1, suma a[5] = 5 / thread 1, suma a[6] = 6 /
thread 0 suma = 21
thread 0, suma a[0] = 0 / thread 0, suma a[1] = 1 / thread 0, suma a[2] = 2 / thread 0, suma a[3] = 3 /
thread 1, suma a[4] = 4 / thread 1, suma a[5] = 5 / thread 1, suma a[6] = 6 /
thread 0 suma = 21
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer3] 2021-04-13 martes
$
```

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: Podemos razonar entonces dado los resultados que **"lastprivate"** machaca el valor de **"firstprivate"** al final de la ejecución y por ende el **resultado final siempre será el tamaño del vector n-1.**

CAPTURAS DE PANTALLA:

```
2021-04-13-05:04 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer4$ ./firstlastprivate
thread 2, suma a[4] SUMA = 4
thread 5, suma a[7] SUMA = 7
thread 4, suma a[6] SUMA = 6
thread 7, suma a[9] SUMA = 9
thread 0, suma a[0] SUMA = 0
thread 0, suma a[1] SUMA = 1
thread 3, suma a[5] SUMA = 5
thread 1, suma a[2] SUMA = 2
thread 1, suma a[3] SUMA = 5
thread 6, suma a[8] SUMA = 8

Fuera de la region parallel suma = 9
2021-04-13-05:04 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer4$ ./firstlastprivate
thread 1, suma a[2] SUMA = 2
thread 1, suma a[3] SUMA = 5
thread 5, suma a[7] SUMA = 7
thread 4, suma a[6] SUMA = 6
thread 3, suma a[5] SUMA = 5
thread 6, suma a[8] SUMA = 8
thread 2, suma a[4] SUMA = 4
thread 7, suma a[9] SUMA = 9
thread 0, suma a[0] SUMA = 0
thread 0, suma a[1] SUMA = 1

Fuera de la region parallel suma = 9
2021-04-13-05:04 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer4$ ./firstlastprivate
thread 6, suma a[8] SUMA = 8
thread 2, suma a[4] SUMA = 4
thread 3, suma a[5] SUMA = 5
thread 7, suma a[9] SUMA = 9
thread 0, suma a[0] SUMA = 0
thread 0, suma a[1] SUMA = 1
thread 5, suma a[7] SUMA = 7
thread 4, suma a[6] SUMA = 6
thread 1, suma a[2] SUMA = 2
thread 1, suma a[3] SUMA = 5

Fuera de la region parallel suma = 9
```

(b) RESPUESTA: Como he dicho anteriormente la única solución sería quitar la cláusula "**lastprivate**" para poder así mantener correctamente el valor de la primera vez que se accede a la variable privada "**suma**"

CAPTURAS DE PANTALLA:

```
2021-04-13-05:11 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer4$ ./firstprivate
thread 7, suma a[9] SUMA = 9
thread 6, suma a[8] SUMA = 8
thread 3, suma a[5] SUMA = 5
thread 4, suma a[6] SUMA = 6
thread 2, suma a[4] SUMA = 4
thread 5, suma a[7] SUMA = 7
thread 1, suma a[2] SUMA = 2
thread 1, suma a[3] SUMA = 5
thread 0, suma a[0] SUMA = 0
thread 0, suma a[1] SUMA = 1

Fuera de la region parallel suma = 0
2021-04-13-05:11 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer4$ ./firstprivate
thread 2, suma a[4] SUMA = 4
thread 5, suma a[7] SUMA = 7
thread 7, suma a[9] SUMA = 9
thread 4, suma a[6] SUMA = 6
thread 6, suma a[8] SUMA = 8
thread 1, suma a[2] SUMA = 2
thread 1, suma a[3] SUMA = 5
thread 0, suma a[0] SUMA = 0
thread 0, suma a[1] SUMA = 1
thread 3, suma a[5] SUMA = 5

Fuera de la region parallel suma = 0
2021-04-13-05:11 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer4$ ./firstprivate
thread 1, suma a[2] SUMA = 2
thread 1, suma a[3] SUMA = 5
thread 7, suma a[9] SUMA = 9
thread 0, suma a[0] SUMA = 0
thread 0, suma a[1] SUMA = 1
thread 4, suma a[6] SUMA = 6
thread 6, suma a[8] SUMA = 8
thread 2, suma a[4] SUMA = 4
thread 3, suma a[5] SUMA = 5
thread 5, suma a[7] SUMA = 7

Fuera de la region 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: Debido a que la variable "**a**" se declarado dentro del bucle de ejecución en paralelo es por ende una variable privada a cada hebra. Y ya que no utilizamos la cláusula "**copyprivate**" apenas aquella hebra que llegue a ejecutar le cláusula "**single**" será la que va a poseer el valor introducido por el usuario, las demás se quedaran con el valor de inicialización de la variable. Entonces el resultado, como así se puede ver en la captura de pantalla adjunta abajo, apenas 1 hebra tendrá su valor cambiado a los demás.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```c
#include <stdio.h>

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

int main(int argc, char const *argv[]){
  int i, n=9, b[n];

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

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

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

  return 0;

}
```

CAPTURAS DE PANTALLA:

```
2021-04-13-05:27 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2$ ./copyprivate-clauseModificado
Introduce valor de a: 2

 Single ejecutada por la hebra: 0

Fuera de la reigon parallel:
b[0] = 2
b[1] = 2
b[2] = 0
b[3] = 0
b[4] = 0
b[5] = 0
b[6] = 0
b[7] = 0
b[8] = 0
2021-04-13-05:27 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2$ ./copyprivate-clauseModificado
Introduce valor de a: 3

 Single ejecutada por la hebra: 0

Fuera de la reigon parallel:
b[0] = 3
b[1] = 3
b[2] = 0
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: Para el siguiente conjunto de números: 1,5,10,30; Vemos que realiza la suma bien, a igual que anteriormente, pero ahora con la única diferencia de empezar por el valor de inicialización de la variable que en este caso es 10.

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 const *argv[])
{
  int i, n=20, a[n], suma=10;

  if(argc<2){
    fprintf(stderr, "Error falta recibir por paramentro un numero.\n");
    exit(-1);
  }

  n = atoi(argv[1]);
  if(n>20){
    n=20;
    printf("\n n = %d\n", 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("\nFuera de la region parallel: suma = %d\n",suma);
  return 0;

}
```

CAPTURAS DE PANTALLA:

```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer6] 2021-04-13 martes
$sbatch -p ac --wrap "./reduction-clauseModificado 1" -o result
Submitted batch job 89321
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer6] 2021-04-13 martes
$sbatch -p ac --wrap "./reduction-clauseModificado 5" -o result2
Submitted batch job 89322
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer6] 2021-04-13 martes
$sbatch -p ac --wrap "./reduction-clauseModificado 10" -o result3
Submitted batch job 89323
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer6] 2021-04-13 martes
$sbatch -p ac --wrap "./reduction-clauseModificado 30" -o result4
Submitted batch job 89324
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer6] 2021-04-13 martes
$ls
reduction-clauseModificado  reduction-clauseModificado.c  result  result2  result3  result4
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer6] 2021-04-13 martes
$cat result result2 result3 result4

Fuera de la region parallel: suma = 10

Fuera de la region parallel: suma = 20

Fuera de la region parallel: suma = 55

 n = 20

Fuera de la region parallel: suma = 200
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer6] 2021-04-13 martes
```

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: Pues 1 de las maneras de realizar este ejercicio sería apoyándonos en la cláusula "**atomic**" que básicamente transforma el código a continuación en una **sentencia atómica** (Una sentencia atómica es aquella que solo puede ser ejecutada de inicio a fin por una hebra sin interrupciones). Y como vemos, realiza correctamente la suma.

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 const *argv[])
{
  int i, n=20, a[n], suma=10;

  if(argc<2){
    fprintf(stderr, "Error falta recibir por paramentro un numero.\n");
    exit(-1);
  }

  n = atoi(argv[1]);
  if(n>20){
    n=20;
    printf("\n n = %d\n", 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("\nFuera de la region parallel: suma = %d\n",suma);
  return 0;

}
```

CAPTURAS DE PANTALLA:

```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer7] 2021-04-13 martes
$sbatch -p ac --wrap "./reduction-clauseModificado7 1" -o result
Submitted batch job 89340
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer7] 2021-04-13 martes
$sbatch -p ac --wrap "./reduction-clauseModificado7 5" -o result2
Submitted batch job 89341
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer7] 2021-04-13 martes
$sbatch -p ac --wrap "./reduction-clauseModificado7 10" -o result3
Submitted batch job 89342
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer7] 2021-04-13 martes
$sbatch -p ac --wrap "./reduction-clauseModificado7 30" -o result4
Submitted batch job 89343
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer7] 2021-04-13 martes
$cat result result2 result3 result4

Fuera de la region parallel: suma = 10

Fuera de la region parallel: suma = 20

Fuera de la region parallel: suma = 55

 n = 20

Fuera de la region parallel: suma = 200
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer7] 2021-04-13 martes
```

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>

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

//#define GLOBAL //Para vectores y matriz globales
#define DYNAMIC //Para vectores y matriz dinamicos

//gcc -O2 -fopenmp -o Pvm_secuencial Pmv_secuencial.c

#ifdef GLOBAL
    #define MAX 25 //2^25
#endif

int main(int argc, char const* argv[])
{
  if(argc<2){
    fprintf(stderr,"[ERROR]: Execution: ./{this} [int size]");
    exit(-1);
  }

  unsigned int n = atoi(argv[1]);

#ifdef GLOBAL
  if(n>MAX){
      n=MAX;
      printf("n=%d\n",n);
  }
  int m[n][n], v1[n], v2[n];
#endif

#ifdef DYNAMIC
  int **m, *v1, *v2;
  v1 = (int*) malloc(n*sizeof(int));
  v2 = (int*) malloc(n*sizeof(int));
  m = (int**) malloc(n*sizeof(int*));

  if((v1 == NULL) || (v2 == NULL) || (m == NULL)){
    printf("[MEMORY ERROR] Unable to reserve memory, malloc error;");
    exit(-2);
  }
```

```
  for(int i=0;i<n;++i){
    m[i] = (int*)malloc(n*sizeof(int));
    if(m[i]==NULL){
      printf("[MEMORY ERROR 2] Unable to reserve memory for row %d", i);
      exit(-2);
    }
  }

#endif

  int half = n / 2;

  for(int i=0; i<n;++i){
    for(int j=0;j<n;++j){
      m[i][j]=2;
    }
    v1[i]=3;
    v2[i]=0;
  }

  double time_start = omp_get_wtime();

  for(int i=0;i<n;++i){
    for(int j=0;j<n;++j){
      v2[i]+=m[i][j] * v1[j];
    }
  }

  double total_time = (double)(omp_get_wtime() - time_start);

  printf("Tiempo (seg.) %11.9f\n", total_time);
```

```
39    if(n>25){
38      printf("(");
37      for(int i=0;i<6;++i){
36        if(i+1==6)
35          printf("...,%d)\n", v2[n-1]);
34        else if(i<5)
33          printf("%d, ", v2[i]);
32      }
31    }else{
30      for(int i=0;i<n;i++){
29        printf(" |");
28        for(int j=0;j<n;++j){
27          if(j+1==n)
26            printf("%d| ",m[i][j]);
25          else
24            printf("%d ", m[i][j]);
23        }
22        if(half==i)
21          printf("x |%d| = |%d| \n", v1[i], v2[i]);
20        else
19          printf("  |%d|   |%d| \n", v1[i], v2[i]);
18      }
17    }
16
15    printf("V2[%d]=%d\n", 0, v2[0]);
14    printf("V2[%d]=%d\n", n-1, v2[n-1]);
13
12  #ifdef VECTOR_DYNAMIC
11    free(v1);
10    free(v2);
 9
 8    for(int i=0;i<n;++i)
 7      free(m[i]);
 6
 5    free(m);
 4
 3  #endif
 2
 1    return 0;
118  }
```

CAPTURAS DE PANTALLA:

```
2021-04-16-01:04 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2$ ./Pmv_secuencial 8
Tiempo (seg.) 0.000000700
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2| x |3| = |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
V2[0]=48
V2[7]=48
2021-04-16-01:04 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2$ ./Pmv_secuencial 11
Tiempo (seg.) 0.000000800
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2| x |3| = |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
V2[0]=66
V2[10]=66
2021-04-16-01:05 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2$
```

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`

```
Pd: Aquí he puesto la captura de los cambios realizados en relación al ejercicio anterior.
```

```
32 int i;
31 #pragma omp for
30   for(i=0;i<n;++i){
29     m[i] = (int*)malloc(n*sizeof(int));
28     if(m[i]==NULL){
27       printf("[MEMORY ERROR 2] Unable to reserve memory for row %d", i);
26       exit(-2);
25     }
24   }
23
22 #endif
21
20   int half = n / 2;
19
18 #pragma omp for
17   for(i=0; i<n;++i){
16     for(int j=0;j<n;++j){
15       m[i][j]=2;
14     }
13     v1[i]=3;
12     v2[i]=0;
11   }
10
 9   double time_start = omp_get_wtime();
 8
 7 #pragma omp for
 6   for(i=0;i<n;++i){
 5     for(int j=0;j<n;++j){
 4       v2[i]+=m[i][j] * v1[j];
 3     }
 2   }
 1
78   double total_time = (double)(omp_get_wtime() - time_start);
```

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

Pd: Aquí he puesto la captura de los cambios realizados en relación al apartador anterior.

```
25
26   double time_start = omp_get_wtime();
27
28   double temp;
29   for(i=0;i<n;++i){
30     temp = 0;
31 #pragma omp for
32     for(j=0;j<n;++j){
33       temp+=(m[i][j] * v1[j]);
34     }
35 #pragma omp atomic
36     v2[i]+=temp;
37   }
38
39   double total_time = (double)(omp_get_wtime() - time_start);
```

RESPUESTA: He tenido un par de errores sobre todo en la ejecución en paralelo por columnas, en los cuáles no me salía el resultado correcto. Tras un par de minutos de investigación, me he acordado de lo dicho en clase de que haría falta una variable temporal para la suma y acceder apenas atómicamente al vector suma una vez que se haya calculado la respuesta final ya que si no tendríamos una "race condition" que causaba los resultados erróneos. Además de estos, pequeños fallos de sintaxis por la prisa;

CAPTURAS DE PANTALLA:

```
2021-04-17-10:25 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer9$ ./pmv_OpenMP-a 8
Tiempo (seg.) 0.000000800
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2| x |3| = |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
V2[0]=48
V2[7]=48
2021-04-17-10:25 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer9$ ./pmv_OpenMP-a 11
Tiempo (seg.) 0.000000900
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2| x |3| = |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
V2[0]=66
V2[10]=66
2021-04-17-10:25 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer9$ ./pmv_OpenMP-b 8
Tiempo (seg.) 0.000001000
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2| x |3| = |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
V2[0]=48
V2[7]=48
2021-04-17-10:25 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP2/ejer9$ ./pmv_OpenMP-b 11
Tiempo (seg.) 0.000001200
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2| x |3| = |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
```

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`

```
  double time_start = omp_get_wtime();

#pragma omp parallel private(i,j)
{
  for(i=0;i<n;++i){

#pragma omp for reduction(+:temp)
    for(j=0;j<n;++j){
      temp+=m[i][j] * v1[j];
    }
#pragma omp single
  {
    v2[i]+=temp;
    temp = 0;
    }
  }
}
  double total_time = (double)(omp_get_wtime() - time_start);
```

RESPUESTA: Como es de esperar, tuve varios problemas de “race conditions” dónde las hebras no están sincronizadas causando que hubiera fallos en los cálculos. Para arreglarlo me he intentado ayudar de las cláusulas de sincronización, “barrier, single, atomic”, quedándome finalmente con “single.”

CAPTURAS DE PANTALLA:

```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer10] 2021-04-17 sábado
$./pmv_OpemMP-reduction 8
Tiempo (seg.) 0.002835806
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2| x |3| = |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
 |2 2 2 2 2 2 2 2|   |3|   |48|
V2[0]=48
V2[7]=48
[BrianSenaSimons b2estudiante23@atcgrid:~/BP2/ejer10] 2021-04-17 sábado
$./pmv_OpemMP-reduction 11
Tiempo (seg.) 0.008276023
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2| x |3| = |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
 |2 2 2 2 2 2 2 2 2 2 2|   |3|   |66|
V2[0]=66
```

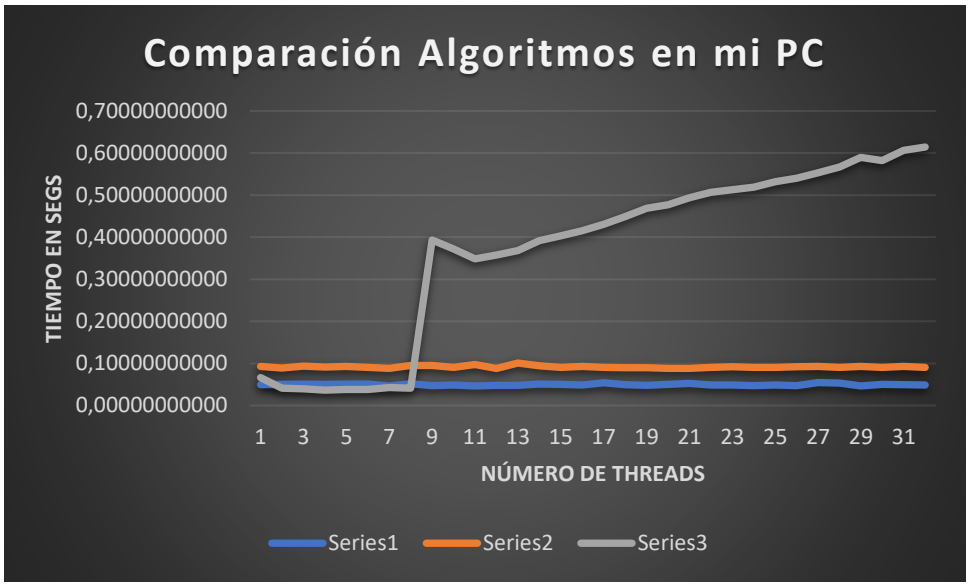
11. Realizar una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid4, en uno de los nodos de la cola ac y en su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (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):

Antes de proceder a ejecutar el script para comparar la ejecución secuencial vs la en paralelo he decido hacer un test en mi PC para elegir el "mejor código". Para ello, he ejecutado un script para ejecutar los 3 códigos, imprimir sus tiempos para n:1-32 threads. Luego he utilzado Excel y estos son los resultados:

Pmv_OpenMP-a (Ejer9): Average-Time: 0,04961611250
Pmv_OpenMP-b (Ejer9): Average-Time: 0,09183636250
Pmv_OpenMP-reduction (Ejer10): Average-Time: 0,36981974375



Cómo vemos el hecho de tener que crear/destruir hebras influye mucho en la ejecución del programa, ya que estas acciones conllevan un tiempo predefinido. Lo que me sorprendió fue que a priori el ***"Pmv_OpenMP-reduction"*** me parecía que iba a ser el mejor código, sin embargo no parece serlo. El resultado es que el mejor código es la paralelización por filas.

JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:

A mi parecer, el hecho de estar creando y repartiendo el trabajo de las columnas a las hebras en cada fila esta empeorando en cierta medida la ejecución del código. Sin embargo, no llego a explicar el gran incremento

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

El script de selección en mi PC:

```
#!/bin/bash

echo "Tiempo en ejecución paralela código A (10000):"
for i in $(seq 1 32); do
    export OMP_NUM_THREADS=$i
    ./pmv_OpenMP-a 10000
done

echo "Tiempo en ejecución paralela código B (10000):"
for i in $(seq 1 32); do
    export OMP_NUM_THREADS=$i
    ./pmv_OpenMP-b 10000
done

echo "Tiempo en ejecución paralela código C (10000):"
for i in $(seq 1 32); do
    export OMP_NUM_THREADS=$i
    ./pmv_OpenMP-reduction 10000
done
```

Script de Ejecución y comparación en atc-grid:
Pd:Al final he utilizado **20000** en vez de **50000**;

```
#!/bin/bash

echo "Tiempo en ejecución secuencial atcgrid[1-3] (10000-100000):"
srun -pac -Aac ./pmv_secuencial 10000
srun -pac -Aac ./pmv_secuencial 50000

echo "Tiempo en ejecución secuencial atcgrid4 (10000-100000):"
srun -pac4 -Aac ./pmv_secuencial 10000
srun -pac4 -Aac ./pmv_secuencial 50000

echo "Tiempo en ejecución paralela atcgrid[1-3] (10000):"
for i in $(seq 1 32); do
    export OMP_NUM_THREADS=$i
    srun -pac -Aac ./pmv_OpenMP-a 10000
done

echo "Tiempo en ejecución paralela atcgrid4 (10000):"
for i in $(seq 1 32); do
    export OMP_NUM_THREADS=$i
    srun -pac4 -Aac ./pmv_OpenMP-a 10000
done

echo "Tiempo en ejecución paralela atcgrid[1-3] (10000):"
for i in $(seq 1 32); do
    export OMP_NUM_THREADS=$i
    srun -pac -Aac ./pmv_OpenMP-a 50000
done

echo "Tiempo en ejecución paralela atcgrid[1-3] (10000):"
for i in $(seq 1 32); do
    export OMP_NUM_THREADS=$i
    srun -pac4 -Aac ./pmv_OpenMP-a 50000
done
```

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	0.134028103	----	0.534848060	----	0.062053190	----	0.206405535	----	0.048091100	----	0.352024400	----
1	0.133924529		0.534800384		0.053725885		0.214497506		0.048445000		0.189102900	
2	0.133988671		0.534813680		0.051654949		0.214013012		0.052432200		0.209464700	
3	0.133961637		0.534829278		0.051648590		0.213334145		0.047025100		0.228840800	
4	0.133961912		0.534830932		0.051813784		0.213930325		0.062589700		0.234527500	
5	0.133980207		0.534816675		0.051655816		0.214364617		0.045265000		0.187586700	
6	0.133986421		0.534902565		0.079717967		0.213335878		0.078852300		0.206198300	
7	0.133983810		0.534974899		0.051690726		0.213106765		0.045946600		0.238747900	
8	0.133966375		0.534888126		0.051698396		0.214570874		0.065152500		0.231493700	
9	0.133999668		0.534864545		0.051674965		0.213408289		0.046446400		0.181493800	
10	0.133957382		0.534811374		0.051694754		0.214217531		0.079980500		0.190481600	
11	0.133983351		0.534847394		0.056580195		0.212202891		0.045536900		0.246651000	
12	0.133964848		0.534888320		0.051686382		0.212790748		0.057980100		0.199237700	
13	0.133986495		0.534798291		0.051680434		0.214012898		0.048688400		0.191647800	
14	0.133892611		0.534895431		0.051683542		0.212258061		0.059565800		0.207941500	
15	0.134068191		0.534827039		0.051737901		0.213998207		0.046414200		0.200921900	
16	0.133943278		0.534861248		0.051696270		0.214722642		0.048075600		0.283166800	
32	0.133966163		0.534867804		0.051967874		0.213026675		0.070563300		0.205327800	

COMENTARIOS SOBRE LOS RESULTADOS: