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): David Gómez Hernández
Grupo de prácticas y profesor de prácticas: Niceto Luque Sola
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:
   Ocurre que da un error ya que toma n como una valor del for la cual no esta especificada debido a la cláusula default.

   CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

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

printf("Después de parallel for:\n");

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

   CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/bp2/ejer1] 03/24/20 09:56:50
$ gcc -O2 -fopenmp shared-clause.c -o shared-clause
shared-clause.c: In function 'main':
shared-clause.c:14:12: error: 'n' not specified in enclosing 'parallel'
    #pragma omp parallel for shared(a) default(none)
            ^~~
shared-clause.c:14:12: error: enclosing 'parallel'
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/bp2/ejer1] 03/24/20 09:56:53
$ gcc -O2 -fopenmp shared-clause-modificado.c -o shared-clause
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/bp2/ejer1] 03/24/20 09:56:56
$ ./shared-clause
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/AC/bp2/ejer1] 03/24/20 09:57:00
```

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:

Lo que pasa es que el valor de suma fuera del paralllel es el mismo que cuando lo iniciamos, ya que al ser privada en la suma no se imprime el resultado, es decir, el valor pasa a ser el que inicializamos (en mi caso 1) en vez de 0 como estaba antes. Además, sustituimos la clausula private por una de firstprivate, de forma que todas las hebras a la hora de que inicien la operación apunten a la misma dirección de memoria desde la que empezar, ya que si dejamos private el resultado de las sumas da un valor basura.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```
#pragma omp parallel firstprivate(suma)
{
   #pragma omp for
   for (int 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\n", suma);
```

CAPTURAS DE PANTALLA:

Ejecución con suma declarada fuera del parallel.

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/A
C/bp2/ejer2] 03/24/20 10:12:52
$ gcc -O2 -fopenmp private-clause-modificado.c -o private-clause
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/A
C/bp2/ejer2] 03/24/20 10:17:34
$ ./private-clause
thread 0 suma a[0] / thread 4 suma a[4] / thread 3 suma a[3] / thread 6 suma a[6] / th
read 5 suma a[5] / thread 2 suma a[2] / thread 1 suma a[1] /
* thread 6 suma= 7
* thread 5 suma= 6
* thread 2 suma= 3
* thread 4 suma= 5
* thread 3 suma= 4
* thread 0 suma= 1
* thread 7 suma= 1
* thread 1 suma= 2
Suma=1
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Carrera/A
C/bp2/ejer2] 03/24/20 10:17:38
```

Ejecución con suma dentro del parallel

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer2] 04/08/20 16:51:06
$ ./private-clause
thread 4 suma a[4] / thread 2 suma a[2] / thread 5 suma a[5] / thread 0 suma a[0
] / thread 3 suma a[3] / thread 1 suma a[1] / thread 6 suma a[6] /
* thread 1 suma= 2
* thread 0 suma= 1
* thread 2 suma= 3
* thread 3 suma= 4
* thread 5 suma= 6
* thread 4 suma= 5
* thread 7 suma= 1
* thread 6 suma= 7
Suma=0
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
```

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

   RESPUESTA:
   Debido a que todas las hebras comparten en mismo valor de la suma, la suma de todas las sumas parciales son las mismas. Por ello, el resultado siempre es el mismo.

   CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```c
#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);
}
```

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer3] 04/08/20 16:51:50
$ ./private-clause
thread 0 suma a[0] / thread 4 suma a[4] / thread 5 suma a[5] / thread 3 suma a[3
] / thread 1 suma a[1] / thread 2 suma a[2] / thread 6 suma a[6] /
* thread 0 suma= 6
* thread 2 suma= 6
* thread 5 suma= 6
* thread 7 suma= 6
* thread 4 suma= 6
* thread 3 suma= 6
* thread 6 suma= 6
* thread 1 suma= 6
```

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:

   No porque depende del reparto que se haga del número de hebras, ya que la última hebra puede realizar alguna interacción extra, dando como resultado un valor distinto a 6.

   CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer4] 04/08/20 16:57:23
$ ./firstlastprivate-clause
 thread 0 suma a[0] suma=0
 thread 7 suma a[7] suma=7
 thread 3 suma a[3] suma=3
 thread 4 suma a[4] suma=4
 thread 1 suma a[1] suma=1
 thread 6 suma a[6] suma=6
 thread 2 suma a[2] suma=2
 thread 5 suma a[5] suma=5

Fuera de la construcción parallel suma=7
```

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:

   Sin copyprivate, el valor de la variable a no se difunde y copia al resto de hebras. Por tanto, sin la cláusula, solamente se inicializan las componentes del vector cuyas hebras hayan podido acceder a la cláusula single.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
#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:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer5] 04/08/20 17:42:57
$ ./copyprivate-clause

Introduce valor de inicialización a: 5

Single ejecutada por el thread 1
Depués de la región parallel:
b[0] = 0        b[1] = 0        b[2] = 5        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:

Hace lo mismo que con suma=0, pero va sumándole 10 al resultado debido a que la cláusula reduction mantiene el valor incial.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

```
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:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer6] 04/08/20 17:50:46
$ ./reduction-clause 5
Tras 'parallel' suma=20
```

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:

La única forma de hacer eso es tener una variable que sea la suma de todas las sumas parciales, sumada en un atomic/critical para que se respete la exclusión mutua.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

```c
int main(int argc, char **argv)  {
   int i, n=20, a[n],suma=0, total=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
   {
    #pragma omp for
      for (i=0; i<n; i++){
        suma += a[i];
      }

      #pragma omp atomic
      total = total + suma;
   }

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

CAPTURAS DE PANTALLA:

```
Falta iteraciones
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer7] 04/08/20 18:00:16
$ ./reduction-clause 5
Tras 'parallel' suma=8
```

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 <omp.h>

int main(int argc, char **argv)  {
   double inicio, final;

   if(argc < 2){
     printf(stderr,"Error de introducción de parámetros: Falta el tamaño de la matriz (tiene que ser cuadrada)\n");
     exit(-1);
   }

   int tamanio = atoi(argv[1]);

   if(tamanio < 2){
     printf(stderr, "El tamaño tiene que ser igual o superior a 2");
     exit(-1);
   }

   double *v1, *v2, **m;

   v1 = (double*) malloc(tamanio*sizeof(double));
   v2 = (double*) malloc(tamanio*sizeof(double));
   m = (double**) malloc(tamanio*sizeof(double*));

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

   for(int i=0; i<tamanio; i++){
     for(int j=0; j<tamanio; j++){
        m[i][j] = 2;
     }
   }

   for(int i=0; i<tamanio; i++){
     v1[i] = 1;
     v2[i] = 0;
   }

   inicio = omp_get_wtime();
```

```
    inicio = omp_get_wtime();

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

    final = omp_get_wtime() - inicio;

    if(tamanio <= 8){
      printf("Tamaño: %i\n Tiempo de ejecución: %f\n", tamanio, final);

      for(int i=0; i<tamanio; i++) printf("v2[%i] = %f\n", i, v2[i]);
    }

    else{
      printf("Tamaño: %i\n Tiempo de ejecución: %f\n Primer elemento: %f\n Último elemento: %f\n", tamanio, final,
       v2[0], v2[tamanio-1]);
    }

    free(v1);
    free(v2);

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

    free(m);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer8] 04/09/20 16:42:49
$ ./pmv-secuencial 8
Tamaño: 8
 Tiempo de ejecución: 0.000001
v2[0] = 16.000000
v2[1] = 16.000000
v2[2] = 16.000000
v2[3] = 16.000000
v2[4] = 16.000000
v2[5] = 16.000000
v2[6] = 16.000000
v2[7] = 16.000000
```

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer8] 04/09/20 16:42:47
$ ./pmv-secuencial 2000
Tamaño: 2000
 Tiempo de ejecución: 0.007217
 Primer elemento: 4000.000000
 Último elemento: 4000.000000
```

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`

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

int main(int argc, char **argv)  {
   double inicio, final;
   int j;

   if(argc < 2){
      printf(stderr,"Error de introducción de parametros: Falta el tamaño de la matriz (tiene que ser cuadrada)\n");
      exit(-1);
    }

    int tamanio = atoi(argv[1]);

    if(tamanio < 2){
        printf(stderr, "El tamaño tiene que ser igual o superior a 2");
        exit(-1);
    }

    double *v1, *v2, **m;

    v1 = (double*) malloc(tamanio*sizeof(double));
    v2 = (double*) malloc(tamanio*sizeof(double));
    m = (double**) malloc(tamanio*sizeof(double*));

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

```
#pragma omp parallel
{
    #pragma omp for private(j)
    for(int i=0; i<tamanio; i++){
        for(j=0; j<tamanio; j++){
            m[i][j] = 2;
        }
    }
    #pragma omp for
    for(int i=0; i<tamanio; i++){
        v1[i] = 1;
        v2[i] = 0;
    }


    #pragma omp critical
        inicio = omp_get_wtime();

    #pragma omp for private(j)
    for(int i=0; i<tamanio; i++){
        for(j=0; j<tamanio; j++){
            v2[i] = v2[i] + (m[i][j]*v1[j]);
        }
    }
}

    final = omp_get_wtime() - inicio;
```

```
    final = omp_get_wtime() - inicio;



    if(tamanio <= 8){
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n", tamanio, final);

        for(int i=0; i<tamanio; i++) printf("v2[%i] = %f\n", i, v2[i]);
    }

    else{
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n Primer elemento: %f\n Último elemento: %f\n", tamanio, final,
         v2[0], v2[tamanio-1]);
    }

    free(v1);
    free(v2);

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

    free(m);

    return 0;
}
```

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

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

int main(int argc, char **argv)  {
   double inicio, final;
   int i,j;

   if(argc < 2){
     printf(stderr,"Error de introducción de parámetros: Falta el tamaño de la matriz (tiene que ser cuadrada)\n");
     exit(-1);
    }

    int tamanio = atoi(argv[1]);

    if(tamanio < 2){
        printf(stderr, "El tamaño tiene que ser igual o superior a 2");
        exit(-1);
    }

    double *v1, *v2, **m;

    v1 = (double*) malloc(tamanio*sizeof(double));
    v2 = (double*) malloc(tamanio*sizeof(double));
    m = (double**) malloc(tamanio*sizeof(double*));

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

```c
#pragma omp parallel private(i)
{
    #pragma omp for private(j)
    for(i=0; i<tamanio; i++){
        for(j=0; j<tamanio; j++){
            m[i][j] = 2;
        }
    }
    #pragma omp for
    for(i=0; i<tamanio; i++){
        v1[i] = 1;
        v2[i] = 0;
    }


    #pragma omp critical
        inicio = omp_get_wtime();

    #pragma omp for private(j)
    for(i=0; i<tamanio; i++){
        for(j=0; j<tamanio; j++){
            v2[i] = v2[i] + (m[i][j]*v1[j]);
        }
    }
}
```

```
    final = omp_get_wtime() - inicio;



    if(tamanio <= 8){
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n", tamanio, final);

        for(i=0; i<tamanio; i++) printf("v2[%i] = %f\n", i, v2[i]);
    }

    else{
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n Primer elemento: %f\n Último elemento: %f\n", tamanio, final,
         v2[0], v2[tamanio-1]);
    }

    free(v1);
    free(v2);

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

    free(m);

    return 0;
}
```

RESPUESTA:
A la hora de parametrizar el bucle anidado, he tenido que buscar en intenet maneras de hacerlo. He encontrado soluciones que utilizaban la cláusula schedule, pero no queriéndola usar ya que es de un bloque práctico posterior. EN segundo lugar he encontrado soluciones con la cláusula collapse, pero por la que me he decantado ha sido por la cláusula private, ya que es la que está incluida en este bloque práctico.
CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer9] 04/10/20 12:11:31
$ ./pmv-OpenMP-a 2000
Tamaño: 2000
 Tiempo de ejecución: 0.002062
 Primer elemento: 4000.000000
 Último elemento: 4000.000000
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer9] 04/10/20 12:11:35
$ ./pmv-OpenMP-b 2000
Tamaño: 2000
 Tiempo de ejecución: 0.002227
 Primer elemento: 4000.000000
 Último elemento: 4000.000000
```

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`

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

int main(int argc, char **argv)  {
   double inicio, final;
   int i,j;
   double local = 0;

   if(argc < 2){
      printf(stderr,"Error de introducción de parámetros: Falta el tamaño de la matriz (tiene que ser cuadrada)\n");
      exit(-1);
    }

    int tamanio = atoi(argv[1]);

    if(tamanio < 2){
        printf(stderr, "El tamaño tiene que ser igual o superior a 2");
        exit(-1);
    }

    double *v1, *v2, **m;

    v1 = (double*) malloc(tamanio*sizeof(double));
    v2 = (double*) malloc(tamanio*sizeof(double));
    m = (double**) malloc(tamanio*sizeof(double*));

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

```c
#pragma omp parallel private(i)
{
    #pragma omp for private(j)
    for(i=0; i<tamanio; i++){
        for(j=0; j<tamanio; j++){
            m[i][j] = 2;
        }
    }
    #pragma omp for
    for(i=0; i<tamanio; i++){
        v1[i] = 1;
        v2[i] = 0;
    }


    #pragma omp critical
        inicio = omp_get_wtime();


    for(i=0; i<tamanio; i++){
        #pragma omp for reduction(+:local)
        for(j=0; j<tamanio; j++){
            local = local + (m[i][j]*v1[j]);
        }

        #pragma omp single
        {
            v2[i] = local;
            local = 0;
        }
    }
}
```

```
    final = omp_get_wtime() - inicio;



    if(tamanio <= 8){
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n", tamanio, final);

        for(i=0; i<tamanio; i++) printf("v2[%i] = %f\n", i, v2[i]);
    }

    else{
        printf("Tamaño: %i\n Tiempo de ejecución: %f\n Primer elemento: %f\n Último elemento: %f\n", tamanio, final,
         v2[0], v2[tamanio-1]);
    }

    free(v1);
    free(v2);

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

    free(m);

    return 0;
}
```

RESPUESTA:

```
pmv-OpenMP-reduction.c:56:17: error: work-sharing region may not be closely nest
ed inside of work-sharing, ‘critical’, ‘ordered’, ‘master’, explicit ‘task’ or ‘
taskloop’ region
          #pragma omp single
```

Al poner la cláusula reduction en el primer for me daba error para cuando ese valor del local fuera asignado a elemento i del vector resultado. Por ello, he tenido que poner la cláusula reduction en el segundo for.

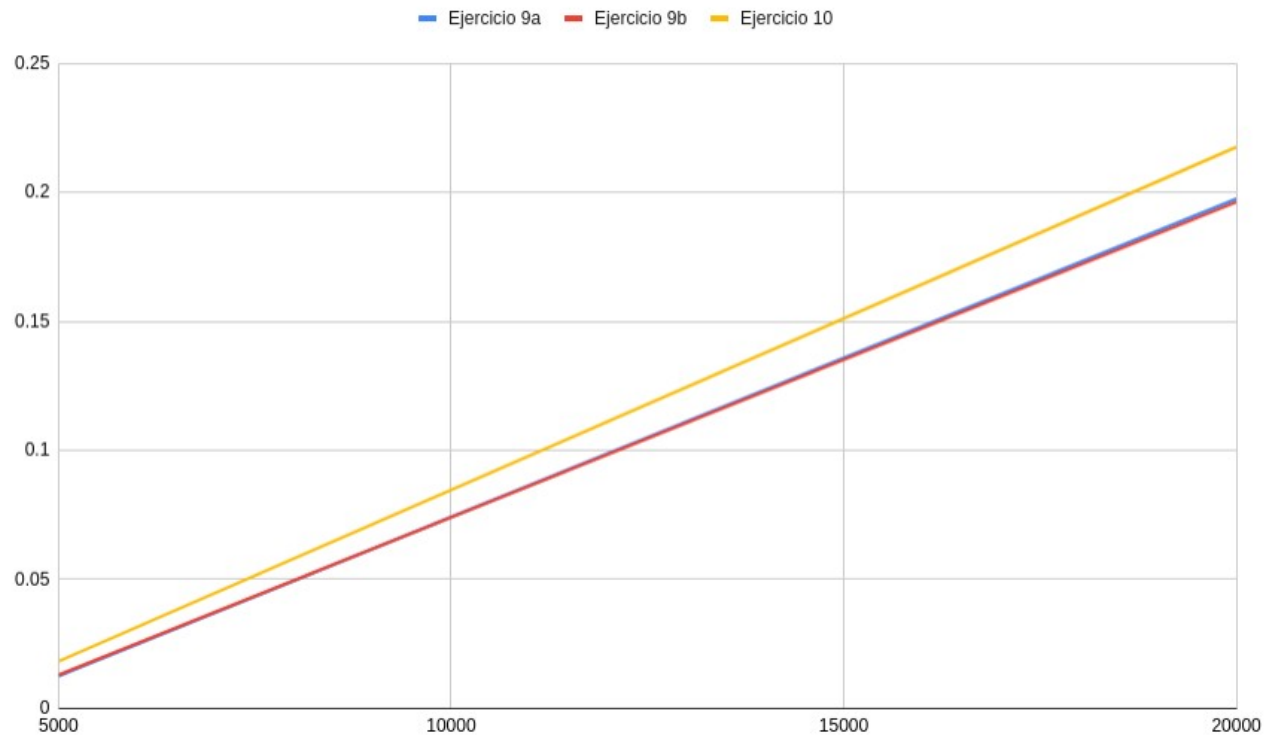
CAPTURAS DE PANTALLA:

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Car
rera/AC/bp2/ejer10] 04/10/20 12:23:41
$ ./pmv-OpenMP-reduction 2000
Tamaño: 2000
 Tiempo de ejecución: 0.008842
 Primer elemento: 4000.000000
 Último elemento: 4000.000000
```

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):

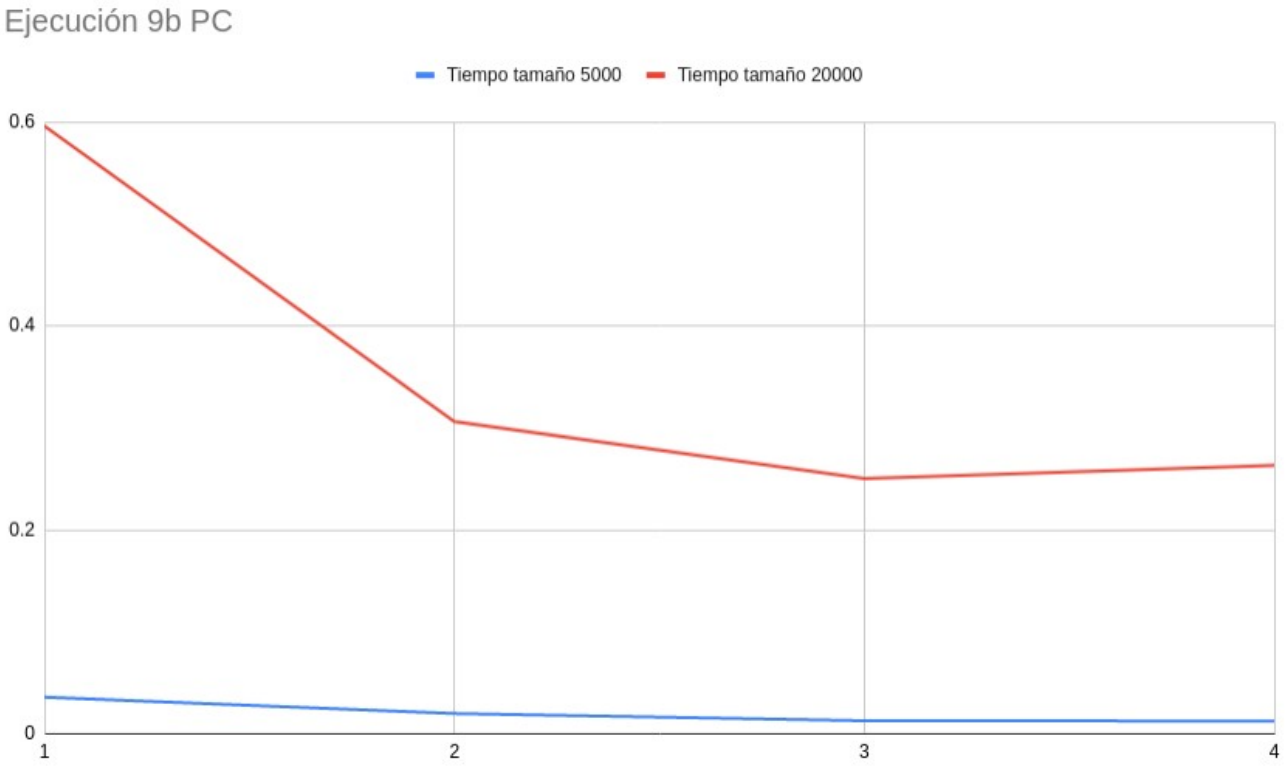
Tamaño	Ejercicio 9a	Ejercicio 9b	Ejercicio 10
5000	0.012584	0.012989	0.018246
20000	0.197413	0.196209	0.217528



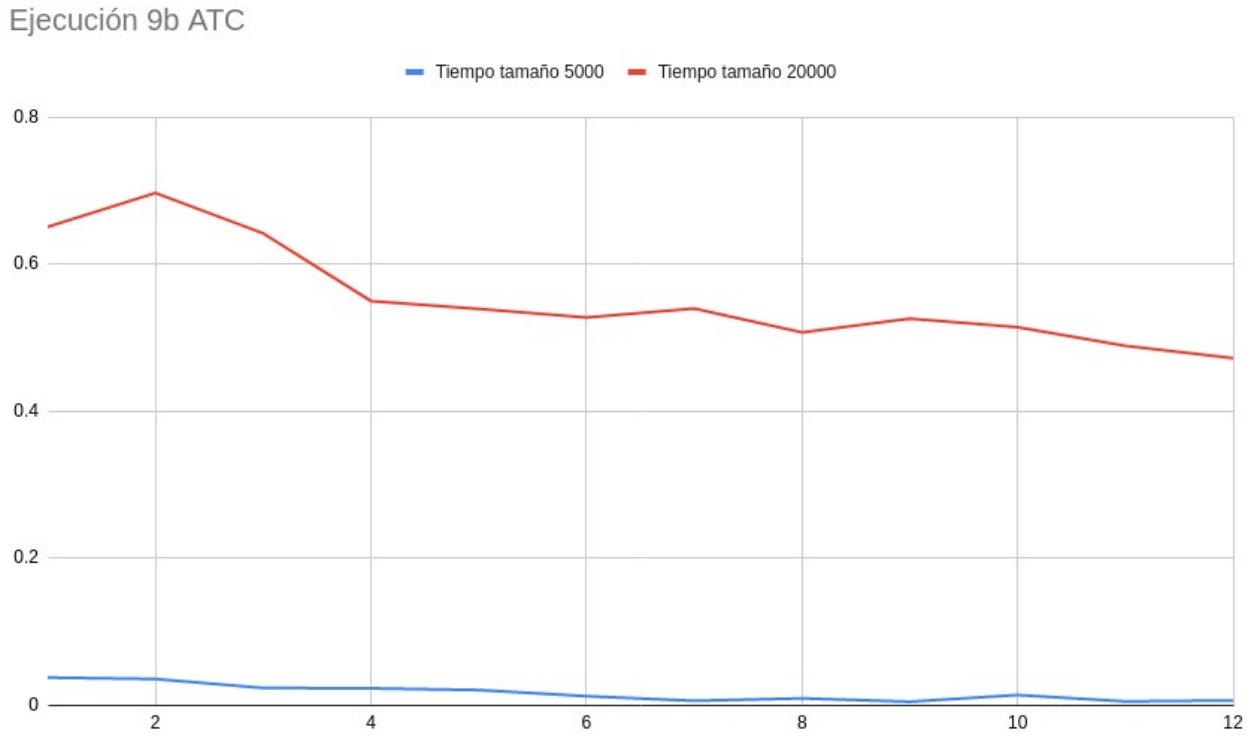
Tanto el 9a como el 9b tienen tiempos muy parecidos. Sin embargo el 9b aunque empiece siendo más lento termina siendo más rápido que el 9a, por lo tanto es mejor elegir el 9b para los ejercicios siguientes.

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):

Número Hebras	Tiempo tamaño 5000	Tiempo tamaño 20000
1	0.035973	0.595755
2	0.019966	0.305999
3	0.012874	0.25004
4	0.012597	0.263079



Número hebras	Tiempo tamaño 5000	Tiempo tamaño 20000
1	0.036994	0.650207
2	0.034996	0.696506
3	0.022793	0.641477
4	0.022285	0.549332
5	0.019973	0.538747
6	0.011673	0.526701
7	0.005316	0.539233
8	0.008715	0.506564
9	0.004123	0.525415
10	0.013237	0.513763
11	0.00454	0.488558
12	0.005822	0.471435



COMENTARIOS SOBRE LOS RESULTADOS:

He elegido los valores mínimos de ambos intervalos, ya que si aumento y me acerco a 100000 el ordenador va bastante mal.

Dicho esto, vemos que las gráficas muestran un comportamiento irregular, llegando a ser incluso peor en el caso del PC con 4 hebras, ya que el tiempo aumenta. Esto nos lleva a la conclusión de que usar una mayor cantidad de hebras no implica una mejora en el tiempo de ejecución.