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): Antonio Gámiz Delgado
Grupo de prácticas: B
Fecha de entrega: 13/04/2018
Fecha evaluación en clase:

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)`? (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:
   Al añadir default(none) a la cláusula parallel e intentar compilar, vemos que nos da un error. Nos dice que la variable "n" no está especificada dentro del bloque paralelo. Esto se debe a que default(none) indica que el programador debe ser el que indique al bloque paralelo mediante código a qué variables pueden tener acceso los threads. Esto se puede arreglar haciendo lo dicho anteriormente, añadiendo la cláusula shared(n).

   CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

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

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

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

    #pragma omp parallel for shared(a) default(none) shared(n)
    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]);

}
```

CAPTURAS DE PANTALLA:

```
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/shared-clauseModificado.c -o ./bin/shared-clauseModificado
./source/shared-clauseModificado.c: In function 'main':
./source/shared-clauseModificado.c:17:13: error: 'n' not specified in enclosing parallel
     #pragma omp parallel for shared(a) default(none)
             ^
./source/shared-clauseModificado.c:17:13: error: enclosing parallel
```

```
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/shared-clauseModificado
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
```

2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? (inicialice suma a un valor distinto de 0 dentro y fuera de `parallel`) Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA:

Vemos que al inicializar la variable suma a dos valores distintos entre sí (-2 y -1), uno fuera de la región paralalela (-2) y el otro dentro (-1), el primero es ignorado y sobreescrito por el segundo, producienco un resultado erróneo de la suma. Esto se debe a la cláusula private, que hace como si redeclaramos la variable suma dentro de la región paralela para cada thread, es decir, cada thread tiene su propia variable suma independiente (si dentro de la región no inicializamos suma entonces tendrá un valor basura).

```
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/private-clauseModificado
thread 1 suma a[2]
thread 1 suma a[3]
thread 3 suma a[6]
thread 2 suma a[4]
thread 2 suma a[5]
thread 0 suma a[0]
thread 0 suma a[1]

* thread 2 suma=8
* thread 1 suma=4
* thread 0 suma=0
* thread 3 suma=5
```

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

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

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

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

    suma=-2;
    #pragma omp parallel private(suma)
    {
        suma=-1;
        #pragma omp for
        for(i=0; i<n; i++)
        {
            suma+=a[i];
            printf("thread %d suma a[%d]\n", omp_get_thread_num(), i);
        }
        printf("\n* thread %d suma=%d",omp_get_thread_num(), suma);
    }
    printf("\n");
}
```

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

RESPUESTA:

Ocurre que, como vemos. La suma siempre sale igual a 15.

Esto es debido a que, como he dicho en el ejercicio anterior, private hace que cada thread tenga su propia variable suma, pero en este caso, todos los thread están actuando sobre la misma variable suma, por lo que además de realizar la suma mal, todos muestran lo mismo.El fallo en la suma se debe a un fallo de dependencia de datos y el de que muestren lo mismo al haber solo una variable suma todos muestran la misma (la del proceso).

```
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/private-clauseModificado2
thread 0 suma a[0]
thread 0 suma a[1]
thread 3 suma a[6]
thread 2 suma a[4]
thread 2 suma a[5]
thread 1 suma a[2]
thread 1 suma a[3]

* thread 0 suma=15
* thread 2 suma=15
* thread 1 suma=15
* thread 3 suma=15
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
```

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+=a[i];
            printf("thread %d suma a[%d]\n", omp_get_thread_num(), i);
        }
        printf("\n* thread %d suma=%d",omp_get_thread_num(), suma);
    }
    printf("\n");
}
```

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.

RESPUESTA:
Sí, ya que aunque esté puesto firstprivate, el último que se ejecuta es lastprivate, por lo que firstprivate es como si no estuviera.
(En la captura en lugar de 6 aparece 11 porque la carga de trabajo se ha repartido difenrente en mis threads que en los de la transparencia).

CAPTURAS DE PANTALLA:

```
└─ $ ▶ gcc -fopenmp -O2 ./source/firstlastprivate-clause.c -o ./bin/firstlastprivate-clause
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/firstlastprivate-clause
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 2 suma a[5] suma=5
thread 2 suma a[6] suma=11
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7

 Fuera de la construccion 'parallel for' suma=11
```

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?

RESPUESTA:

En la captura vemos que solo algunos valores del vector han sido correctamente inicializados, en concreto siempre valores consecutivos. Esto se debe a que al eliminar la directiva, el valor de la variable "a" no ha sido copiada a los demás threads, por lo que el único thread que tiene el valor correcto de "a" es el que ha ejecutado la sentencia single, en este caso el thread 1, que es el que ha ejecutado las iteraciones 3, 4 y 5.

```
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/copyprivate-clauseModificado.c -o ./bin/copyprivate-clauseModificado
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/copyprivate-clauseModificado

Introduce el valor de inicializacion a:1

Single ejecutada por el thread 1
Despues de la region parallel:
b[0]=0  b[1]=0  b[2]=0  b[3]=1  b[4]=1  b[5]=1  b[6]=0  b[7]=0  b[8]=0
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶
```

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

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

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

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

    #pragma omp parallel
    {
        int a;
        #pragma omp single
        {
            printf("\nIntroduce el valor de inicializacion a:");
            scanf("%d", &a);
            printf("\nSingle ejecutada por el thread %d \n", omp_get_thread_num());
        }

        #pragma omp for
        for(i=0; i<n; i++) b[i]=a;
    }

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

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

RESPUESTA:
En el caso de ejecutarlo con n=6, se obtiene 25, es decir, la suma correcta (15) más el valor al que lo hemos inicializado (10). Esto se debe a que al aplicar la directiva reduction, la variable suma pasa a ser local en cada uno de los threads, es decir, cada thread tiene su variable suma_i, que al finalizar se suman todas en la variable suma definida arriba, que es global. Por eso, siempre saldrá el resultado correcto de la suma más el valor al que hayamos inicializado la variable suma.

```
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/reduction-clauseModificado.c -o ./bin/reduction-clauseModificado
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/reduction-clauseModificado 6
Tras 'parallel' suma=25
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
```

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

int main(int argc, char **argv){
    int i, n=20, a[n], suma=10;

    if( argc<2 ) {
        fprintf(stderr, "Faltan 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);

}
```

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 usar directivas de trabajo compartido .

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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

int main(int argc, char **argv){
    int i, n=20, a[n], suma=0, suma_local;

    if( argc<2 ) {
        fprintf(stderr, "Faltan 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;

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

    #pragma omp parallel private(suma_local)
    {
        suma_local=0;
        #pragma omp for
        for(i=0; i<n; i++) suma_local+=a[i];

        #pragma omp atomic
        suma+=suma_local;
    }

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

}
```

CAPTURAS DE PANTALLA:

```
antoniogamizdelgado 2018-04-18 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/reduction-clauseModificado7.c -o ./bin/reduction-clauseModificado7
antoniogamizdelgado 2018-04-18 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/reduction-clauseModificado7 8
a[0]=0 a[1]=1 a[2]=2 a[3]=3 a[4]=4 a[5]=5 a[6]=6 a[7]=7
Tras 'parallel' suma=28
antoniogamizdelgado 2018-04-18 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/reduction-clauseModificado7 11
a[0]=0 a[1]=1 a[2]=2 a[3]=3 a[4]=4 a[5]=5 a[6]=6 a[7]=7 a[8]=8 a[9]=9 a[10]=10
Tras 'parallel' suma=55
antoniogamizdelgado 2018-04-18 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶
```

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):

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

CAPTURA CÓDIGO FUENTE: `pmv-secuencial.c`

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

#define MOSTRAR_MATRIZ 1
#define MOSTRAR_V 1
#define MOSTRAR_V2 1
#define MOSTRAR_PRIMERO_ULTIMO 0

int main(int argc, char ** argv)
{
    if( argc<2 ) {
        fprintf(stderr, "Formato: %s <N>\n", argv[0]);
        exit(-1);
    }

    int N=atoi(argv[1]);

    int m[N][N];
    int v[N], v2[N];

    //Inicializamos la matriz m.
    for(int i=0; i<N; i++) for(int j=0; j<N; j++) m[i][j]=i+j;

    //Inicializamos el vector v.
    for(int i=0; i<N; i++) v[i]=1;

    //Inicializamos el vector v2.
    for(int i=0; i<N; i++) v2[i]=0;

    //Calculamos la multiplicacion m*v en v2.
    double start = omp_get_wtime();
    for(int i=0; i<N; i++) for(int j=0; j<N; j++) v2[i]+=m[i][j]*v[j];
    double end = omp_get_wtime();

    //Mostramos matriz/vectores.
    #if MOSTRAR_MATRIZ
        printf("Matrix m: \n");
        for(int i=0; i<N; i++) {for(int j=0; j<N; j++) printf("%d\t", m[i][j]); printf("\n"); }
    #endif
    #if MOSTRAR_V
        printf("Vector v: \n");
        for(int i=0; i<N; i++) printf("%d ", v[i]); printf("\n");
    #endif
    #if MOSTRAR_V2
        printf("Vector v2: \n");
        for(int i=0; i<N; i++) printf("%d ", v2[i]); printf("\n");
    #endif

    #if MOSTRAR_PRIMERO_ULTIMO
        printf("Tiempo(seg): %11.9f \t N=%u\n", (float)(end-start), N);
    #else
        printf("Tiempo(seg): %11.9f \t N=%u \t (v2[0]=%u v2[%u]=%u)\n", (float)(end-start), N, v2[0], N-1, v2[N-1]);
    #endif
}
```

CAPTURAS DE PANTALLA:

```
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/pmv-secuencial.c -o ./bin/pmv-secuencial
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/pmv-secuencial  8
Matrix m:
0       1       2       3       4       5       6       7
1       2       3       4       5       6       7       8
2       3       4       5       6       7       8       9
3       4       5       6       7       8       9       10
4       5       6       7       8       9       10      11
5       6       7       8       9       10      11      12
6       7       8       9       10      11      12      13
7       8       9       10      11      12      13      14
Vector v:
1 1 1 1 1 1 1 1
Vector v2:
28 36 44 52 60 68 76 84
Tiempo(seg): 0.000000901         N=8     (v2[0]=28 v2[7]=84)
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/pmv-secuencial  11
Matrix m:
0       1       2       3       4       5       6       7       8       9       10
1       2       3       4       5       6       7       8       9       10      11
2       3       4       5       6       7       8       9       10      11      12
3       4       5       6       7       8       9       10      11      12      13
4       5       6       7       8       9       10      11      12      13      14
5       6       7       8       9       10      11      12      13      14      15
6       7       8       9       10      11      12      13      14      15      16
7       8       9       10      11      12      13      14      15      16      17
8       9       10      11      12      13      14      15      16      17      18
9       10      11      12      13      14      15      16      17      18      19
10      11      12      13      14      15      16      17      18      19      20
Vector v:
1 1 1 1 1 1 1 1 1 1 1
Vector v2:
55 66 77 88 99 110 121 132 143 154 165
Tiempo(seg): 0.000000935         N=11    (v2[0]=55 v2[10]=165)
antoniogamizdelgado 2018-04-11 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶
```

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`

```
//Inicializamos la matriz m.
for(int i=0; i<N; i++)
{
    #pragma omp parallel for
    for(int j=0; j<N; j++) m[i][j]=i+j;
}
//Inicializamos el vector v.
#pragma omp parallel for
for(int i=0; i<N; i++) v[i]=1;

//Inicializamos el vector v2.
#pragma omp parallel for
for(int i=0; i<N; i++) v2[i]=0;

//Calculamos la multiplicacion m*v en v2.
double start = omp_get_wtime();

for(int i=0; i<N; i++)
{
    #pragma omp parallel for
    for(int j=0; j<N; j++) v2[i]+=m[i][j]*v[j];
}
double end = omp_get_wtime();
```

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

```
//Inicializamos la matriz m.
#pragma omp parallel for
for(int i=0; i<N; i++)
    for(int j=0; j<N; j++) m[i][j]=i+j;

//Inicializamos el vector v.
#pragma omp parallel for
for(int i=0; i<N; i++) v[i]=1;

//Inicializamos el vector v2.
#pragma omp parallel for
for(int i=0; i<N; i++) v2[i]=0;

//Calculamos la multiplicacion m*v en v2.
double start = omp_get_wtime();

#pragma omp parallel for
for(int i=0; i<N; i++)
    for(int j=0; j<N; j++) v2[i]+=m[i][j]*v[j];

double end = omp_get_wtime();
```

RESPUESTA:
Adjunto solo la parte del código que cambia ya que el programa entero es un poco extenso.
No he tenido problemas al realizar este ejercicio.
CAPTURAS DE PANTALLA:

```
antoniogamizdelgado 2018-04-12 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/pmv-OpenMP-a 8
Matrix m:
0       1       2       3       4       5       6       7
1       2       3       4       5       6       7       8
2       3       4       5       6       7       8       9
3       4       5       6       7       8       9       10
4       5       6       7       8       9       10      11
5       6       7       8       9       10      11      12
6       7       8       9       10      11      12      13
7       8       9       10      11      12      13      14
Vector v:
1 1 1 1 1 1 1 1
Vector v2:
28 36 44 52 60 68 76 84
Tiempo(seg): 0.000356228         N=8     (v2[0]=28 v2[7]=84)
antoniogamizdelgado 2018-04-12 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/pmv-OpenMP-a 11
Matrix m:
0       1       2       3       4       5       6       7       8       9       10
1       2       3       4       5       6       7       8       9       10      11
2       3       4       5       6       7       8       9       10      11      12
3       4       5       6       7       8       9       10      11      12      13
4       5       6       7       8       9       10      11      12      13      14
5       6       7       8       9       10      11      12      13      14      15
6       7       8       9       10      11      12      13      14      15      16
7       8       9       10      11      12      13      14      15      16      17
8       9       10      11      12      13      14      15      16      17      18
9       10      11      12      13      14      15      16      17      18      19
10      11      12      13      14      15      16      17      18      19      20
Vector v:
1 1 1 1 1 1 1 1 1 1 1
Vector v2:
55 66 77 88 99 110 121 132 143 154 165
Tiempo(seg): 0.002298911         N=11    (v2[0]=55 v2[10]=165)
```

```
antoniogamizdelgado 2018-04-12 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/pmv-OpenMP-b 8
Matrix m:
0       1       2       3       4       5       6       7
1       2       3       4       5       6       7       8
2       3       4       5       6       7       8       9
3       4       5       6       7       8       9       10
4       5       6       7       8       9       10      11
5       6       7       8       9       10      11      12
6       7       8       9       10      11      12      13
7       8       9       10      11      12      13      14
Vector v:
1 1 1 1 1 1 1 1
Vector v2:
28 36 44 52 60 68 76 84
Tiempo(seg): 0.000001962         N=8     (v2[0]=28 v2[7]=84)
antoniogamizdelgado 2018-04-12 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ ./bin/pmv-OpenMP-b 11
Matrix m:
0       1       2       3       4       5       6       7       8       9       10
1       2       3       4       5       6       7       8       9       10      11
2       3       4       5       6       7       8       9       10      11      12
3       4       5       6       7       8       9       10      11      12      13
4       5       6       7       8       9       10      11      12      13      14
5       6       7       8       9       10      11      12      13      14      15
6       7       8       9       10      11      12      13      14      15      16
7       8       9       10      11      12      13      14      15      16      17
8       9       10      11      12      13      14      15      16      17      18
9       10      11      12      13      14      15      16      17      18      19
10      11      12      13      14      15      16      17      18      19      20
Vector v:
1 1 1 1 1 1 1 1 1 1 1
Vector v2:
55 66 77 88 99 110 121 132 143 154 165
Tiempo(seg): 0.000001593         N=11    (v2[0]=55 v2[10]=165)
```

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`

```
//Inicializamos la matriz m.
#pragma omp parallel for
for(int i=0; i<N; i++)
{
    for(int j=0; j<N; j++) m[i][j]=i+j;
}
//Inicializamos el vector v.
#pragma omp parallel for
for(int i=0; i<N; i++) v[i]=1;

//Inicializamos el vector v2.
#pragma omp parallel for
for(int i=0; i<N; i++) v2[i]=0;

//Calculamos la multiplicacion m*v en v2.
double start = omp_get_wtime();

int sum=0;
#pragma omp parallel
{
    #pragma omp parallel for
    for(int i=0; i<N; i++)
    {
        sum=0;
        #pragma omp parallel for reduction(+:sum)
        for(int j=0; j<N; j++) sum+=m[i][j]*v[j];
        v2[i]=sum;
    }
}
double end = omp_get_wtime();
```

RESPUESTA:

```
antoniogamizdelgado 2018-04-12 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/pmv-OpenMP-reduction.c -o ./bin/pmv-OpenMP-reduction
./source/pmv-OpenMP-reduction.c: In function 'main':
./source/pmv-OpenMP-reduction.c:69:48: error: expected ')' before '[' token
         #pragma omp parallel for reduction(+:v2[i])
                                               ^

./source/pmv-OpenMP-reduction.c:69:43: error: user defined reduction not found for 'v2'
         #pragma omp parallel for reduction(+:v2[i])
                                           ^
```

Aquí vemos que el primer problema que tuve fue que intenté aplicar reduction a la variable v2[i], pero openMP parece que no reconoce la variable (definida más arriba). Pensé que sería porque no había compartido el vector, así que añadí la directiva shared(v2), pero tampoco funcionó. Por lo que me al final hice lo siguiente:

```
int sum=0;
#pragma omp parallel
{
    #pragma omp parallel for
    for(int i=0; i<N; i++)
    {
        sum=0;
        #pragma omp parallel for reduction(+:sum)
        for(int j=0; j<N; j++) sum+=m[i][j]*v[j];
        v2[i]=sum;
    }
}
```

De esta forma ya me funciona correctamente.

(Para resolver el problema he usado este seminario y los anteriores, así como los ejercicios hechos en el mismo).

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 LA EJECUCION EN LOCAL Y ATCGRID

```
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -O2 -fopenmp ./source/pmv-OpenMP-a.c -o ./bin/pmv-OpenMP-a
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -O2 -fopenmp ./source/pmv-secuencial.c -o ./bin/pmv-secuencial
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -O2 -fopenmp ./source/pmv-OpenMP-a.c -o ./bin/pmv-OpenMP-a
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -O2 -fopenmp ./source/pmv-OpenMP-b.c -o ./bin/pmv-OpenMP-b
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶ gcc -O2 -fopenmp ./source/pmv-OpenMP-reduction.c -o ./bin/pmv-OpenMP-reduction
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶
```

SECUANCIAL:

```
pmv-secuencial
Tiempo(seg): 0.238937572        N=15000         (v2[0]=22471994 v2[14999]=84343748)
Tiempo(seg): 0.955681980        N=30000         (v2[0]=89612303 v2[29999]=279478790)
Tiempo(seg): 0.239965528        N=15000         (v2[0]=26385136 v2[14999]=68605121)
Tiempo(seg): 0.938681901        N=30000         (v2[0]=90955585 v2[29999]=255340830)
Tiempo(seg): 0.242295817        N=15000         (v2[0]=26187430 v2[14999]=74073783)
Tiempo(seg): 0.936178207        N=30000         (v2[0]=105006824 v2[29999]=297854591)
Tiempo(seg): 0.244279772        N=15000         (v2[0]=26159708 v2[14999]=77705522)
Tiempo(seg): 0.928151667        N=30000         (v2[0]=90491735 v2[29999]=283481194)
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶
```

```
sftp> put atcgrid_secuencial.sh
Uploading atcgrid_secuencial.sh to /home/E2estudiante14/atcgrid_secuencial.sh
atcgrid_secuencial.sh                                         100%  504     0.5KB/s   00:00
sftp> lcd ../bin/
sftp> put pmv-secuencial
Uploading pmv-secuencial to /home/E2estudiante14/pmv-secuencial
pmv-secuencial                                                100% 9032     8.8KB/s   00:00
sftp>
```

```
[E2estudiante14@atcgrid ~]$ ls
atcgrid_secuencial.sh  pmv-secuencial
[E2estudiante14@atcgrid ~]$ qsub atcgrid_secuencial.sh -q ac
73313.atcgrid
[E2estudiante14@atcgrid ~]$ ls
atcgrid_secuencial.e73313  atcgrid_secuencial.o73313  atcgrid_secuencial.sh  pmv-secuencial
[E2estudiante14@atcgrid ~]$
```

```
sftp> lcd ../outputs/
sftp> get atcgrid_secuencial.o73313
Fetching /home/E2estudiante14/atcgrid_secuencial.o73313 to atcgrid_secuencial.o73313
/home/E2estudiante14/atcgrid_secuencial.o73313                100% 2292     2.2KB/s   00:00
sftp>
```

PARALELOS:

```
pmv-OpenMP-a
Numero de threads 1
Tiempo(seg): 0.274652898          N=15000          (v2[0]=112492500 v2[14999]=337477500)
Tiempo(seg): 1.090885878          N=30000          (v2[0]=449985000 v2[29999]=1349955000)
Numero de threads 2
Tiempo(seg): 0.208082855          N=15000          (v2[0]=61913726 v2[14999]=196868211)
Tiempo(seg): 0.801692188          N=30000          (v2[0]=241467955 v2[29999]=686105990)
Numero de threads 3
Tiempo(seg): 0.148853645          N=15000          (v2[0]=64663000 v2[14999]=121786054)
Tiempo(seg): 0.556426644          N=30000          (v2[0]=138123463 v2[29999]=497811977)
Numero de threads 4
Tiempo(seg): 0.135548025          N=15000          (v2[0]=10541119 v2[14999]=76728578)
Tiempo(seg): 0.453461349          N=30000          (v2[0]=90352327 v2[29999]=375070006)
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶
```

```
pmv-OpenMP-b
Numero de threads 1
Tiempo(seg): 0.131490260          N=15000          (v2[0]=112492500 v2[14999]=337477500)
Tiempo(seg): 0.523970544          N=30000          (v2[0]=449985000 v2[29999]=1349955000)
Numero de threads 2
Tiempo(seg): 0.071179464          N=15000          (v2[0]=112492500 v2[14999]=337477500)
Tiempo(seg): 0.286635846          N=30000          (v2[0]=449985000 v2[29999]=1349955000)
Numero de threads 3
Tiempo(seg): 0.057382304          N=15000          (v2[0]=112492500 v2[14999]=337477500)
Tiempo(seg): 0.230802625          N=30000          (v2[0]=449985000 v2[29999]=1349955000)
Numero de threads 4
Tiempo(seg): 0.054979146          N=15000          (v2[0]=112492500 v2[14999]=337477500)
Tiempo(seg): 0.219101608          N=30000          (v2[0]=449985000 v2[29999]=1349955000)
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶
```

```
pmv-OpenMP-reduction
Numero de threads 1
Tiempo(seg): 0.109174781          N=15000          (v2[0]=112492500 v2[14999]=337477500)
Tiempo(seg): 0.442600965          N=30000          (v2[0]=449985000 v2[29999]=1349955000)
Numero de threads 2
Tiempo(seg): 0.117127366          N=15000          (v2[0]=112492500 v2[14999]=337477500)
Tiempo(seg): 0.460477680          N=30000          (v2[0]=449985000 v2[29999]=1349955000)
Numero de threads 3
Tiempo(seg): 0.119092211          N=15000          (v2[0]=112492500 v2[14999]=337477500)
Tiempo(seg): 0.458398640          N=30000          (v2[0]=449985000 v2[29999]=1349955000)
Numero de threads 4
Tiempo(seg): 0.120661132          N=15000          (v2[0]=112492500 v2[14999]=337477500)
Tiempo(seg): 0.601508737          N=30000          (v2[0]=449985000 v2[29999]=1349955000)
antoniogamizdelgado 2018-04-13 viernes  @ antonio ~/ArquitecturaDeComputadores/Práctica 3 (master)
└─ $ ▶
```

```
[E2estudiante14@atcgrid ~]$ ls
atcgrid_OMP_a.sh  atcgrid_OMP_b.sh  atcgrid_OMP_reduction.sh  pmv-OpenMP-a  pmv-OpenMP-b  pmv-OpenMP-reduction
[E2estudiante14@atcgrid ~]$
Uploading pmv-OpenMP-b to /home/E2estudiante14/pmv-OpenMP-b
pmv-OpenMP-b                                                  100%   13KB  13.2KB/s   00:00
sftp> put pmv-OpenMP-reduction
Uploading pmv-OpenMP-reduction to /home/E2estudiante14/pmv-OpenMP-reduction
pmv-OpenMP-reduction                                          100%   13KB  13.3KB/s   00:00
sftp> lcd ../scripts
sftp> put atcgrid_OMP_a.sh
Uploading atcgrid_OMP_a.sh to /home/E2estudiante14/atcgrid_OMP_a.sh
atcgrid_OMP_a.sh                                              100%  261     0.3KB/s   00:00
sftp> put atcgrid_OMP_b.sh
Uploading atcgrid_OMP_b.sh to /home/E2estudiante14/atcgrid_OMP_b.sh
atcgrid_OMP_b.sh                                              100%  261     0.3KB/s   00:00
sftp> put atcgrid_OMP_reduction.sh
Uploading atcgrid_OMP_reduction.sh to /home/E2estudiante14/atcgrid_OMP_reduction.sh
atcgrid_OMP_reduction.sh                                      100%  277     0.3KB/s   00:00
sftp>
```

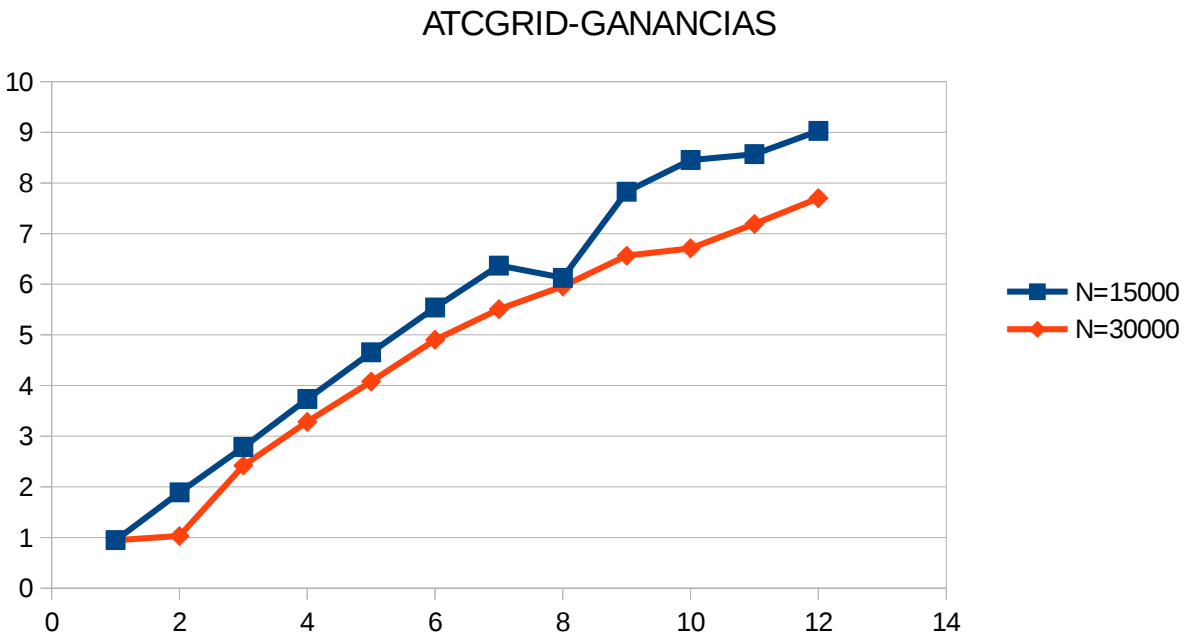
```
[E2estudiante14@atcgrid ~]$ ls
atcgrid_OMP_a.sh  atcgrid_OMP_b.sh  atcgrid_OMP_reduction.sh  pmv-OpenMP-a  pmv-OpenMP-b  pmv-OpenMP-reduction
[E2estudiante14@atcgrid ~]$ qstat
[E2estudiante14@atcgrid ~]$ qsub atcgrid_OMP_a.sh -q ac
73297.atcgrid
[E2estudiante14@atcgrid ~]$ qsub atcgrid_OMP_b.sh -q ac
73298.atcgrid
[E2estudiante14@atcgrid ~]$ qsub atcgrid_OMP_reduction.sh -q ac
73299.atcgrid
[E2estudiante14@atcgrid ~]$ ls
atcgrid_OMP_a.e73297  atcgrid_OMP_b.e73298  atcgrid_OMP_reduction.e73299  pmv-OpenMP-a
atcgrid_OMP_a.o73297  atcgrid_OMP_b.o73298  atcgrid_OMP_reduction.o73299  pmv-OpenMP-b
atcgrid_OMP_a.sh      atcgrid_OMP_b.sh      atcgrid_OMP_reduction.sh      pmv-OpenMP-reduction
[E2estudiante14@atcgrid ~]$
```

```
sftp> lcd ../outputs/
sftp> lls
sftp> get atcgrid_OMP_a.o73297
Fetching /home/E2estudiante14/atcgrid_OMP_a.o73297 to atcgrid_OMP_a.o73297
/home/E2estudiante14/atcgrid_OMP_a.o73297                     100% 2492     2.4KB/s   00:00
sftp> get atcgrid_OMP_b.o73298
Fetching /home/E2estudiante14/atcgrid_OMP_b.o73298 to atcgrid_OMP_b.o73298
/home/E2estudiante14/atcgrid_OMP_b.o73298                     100% 2530     2.5KB/s   00:00
sftp> get atcgrid_OMP_reduction.o73299
Fetching /home/E2estudiante14/atcgrid_OMP_reduction.o73299 to atcgrid_OMP_reduction.o73299
/home/E2estudiante14/atcgrid_OMP_reduction.o73299             100% 2538     2.5KB/s   00:00
```

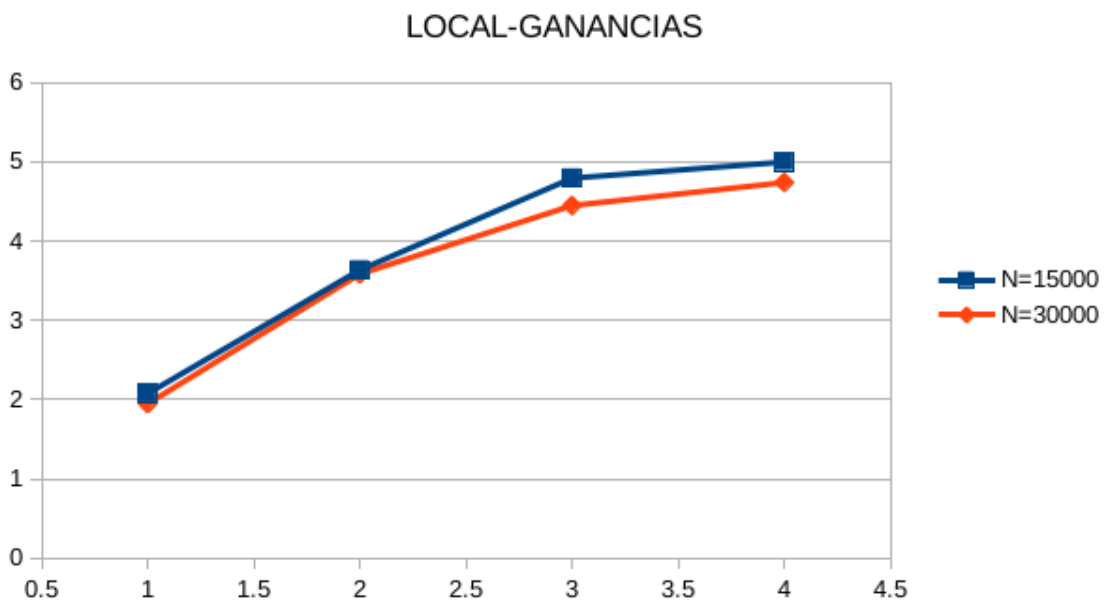
CAPTURAS DE PANTALLA (que justifique el código elegido):
Viendo las capturas del código anterior vemos que el programa pmv-OpenPM-b es el que se ejecuta más rápido, por lo que escogemos ese para la representación de la ganancia.

TABLA Y GRÁFICA (por *ejemplo* para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 30000 y 100000, y otro entre 5000 y 30000):

ATCGRID pmv-openMP-secuencial			ATCGRID pmv-openMP-b			GANANCIA	
Nº Threads	N=15000	N=30000	Nº Threads	N=15000	N=30000	N=15000	N=30000
1	0.30068332	1.056068778	1	0.316974521	1.112721682	0.9486040678	0.9490861867
2	0.300512314	1.056288123	2	0.159058064	1.026156306	1.8893246054	1.0293637693
3	0.300379872	1.056230426	3	0.107834645	0.436361223	2.785559984	2.4205414467
4	0.300417095	1.056270123	4	0.080459416	0.321655959	3.7337717564	3.2838506281
5	0.300468981	1.056210041	5	0.064521618	0.259039044	4.6568730034	4.0774163797
6	0.300348103	1.05623281	6	0.054211605	0.21542494	5.540291659	4.903020096
7	0.302591383	1.056309104	7	0.047506951	0.191761538	6.3694128255	5.5084513559
8	0.300482899	1.056106925	8	0.049048014	0.177275449	6.1263010364	5.9574347771
9	0.300462663	1.056716681	9	0.038372565	0.16095455	7.8301427856	6.565311021
10	0.300487906	1.056964278	10	0.035544191	0.1575322	8.4539244683	6.7095125822
11	0.300611079	1.056175232	11	0.035083905	0.146863729	8.5683471951	7.191532172
12	0.300386429	1.056883216	12	0.033272211	0.137252122	9.0281475133	7.7003051071



LOCAL pmv-openMP-secuencial			LOCAL pmv-openMP-b			GANANCIA	
Nº Threads	N=15000	N=30000	Nº Threads	N=15000	N=30000	N=15000	N=30000
1	0.273211837	1.021336555	1	0.13149026	0.523970544	2.0778104553	1.9492251362
2	0.258905619	1.027917385	2	0.071179464	0.286635846	3.6373639875	3.5861438803
3	0.275141865	1.026885033	3	0.057382304	0.230802625	4.7948905119	4.4491913079
4	0.274795026	1.038477659	4	0.054979146	0.219101608	4.9981683237	4.7397080673



Creo que código pmv-openMP-b es el más eficiente ya que es que el mejor reparte el trabajo entre las hebras,