

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): César Muñoz Reinoso

Grupo de prácticas: Grupo 2

Fecha de entrega: 08/04

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:

Si añado la cláusula `default(none)`, las variables anteriores que no se incluyan en la directiva `shared`, no serán accesibles. Si añadimos la variable `n` a la cláusula `shared`, podremos acceder desde dentro de `parallel` a dicha variable.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

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

int main(){

    int i,n=7;
    int a[n];

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

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

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

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

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas
rchivos/BP2] 2018-04-06 viernes
$gcc -O2 -fopenmp -o shared-clauseMod shared-clauseModificado.c
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:14:10: error: 'n' not specified in enclosing 'parallel'
    #pragma omp parallel for shared(a) default(none)
    ^~~~~
shared-clauseModificado.c:14:10: error: enclosing 'parallel'
```

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:

Cuando se usa la clausula `private`, los valores de entrada y de salida son desconocidos, por lo que si se inicializa fuera de la directiva `parallel`, en cada ejecución da un resultado diferente, en cambio cuando lo inicializamos dentro de dicha directiva, no ocurre ningún problema, cada thread tiene su suma y la realiza la suma adecuadamente.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

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

int main(){

    int i,n=7;
    int a[n],suma;

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

    #pragma omp parallel private(suma)
    {
        suma=5;
        #pragma omp for
        for(i=0;i<n;i++){
            suma=suma + a[i];
            printf("thread %d suma a[%d]",omp_get_thread_num
            (),i);

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

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-06
viernes
$./private-clauseMod
thread 0 suma a[0]thread 0 suma a[1]thread 2 suma a[4]thread 2 suma a[5]thread 1 suma a[2]thread 1 suma a[3]thread 3 suma a[6]
* thread 0 suma=6
* thread 1 suma=10
* thread 2 suma=14
* thread 3 suma=11
```

CAPTURA CÓDIGO FUENTE: private-clauseModificado2.c

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

int main(){

    int i,n=7;
    int a[n],suma;

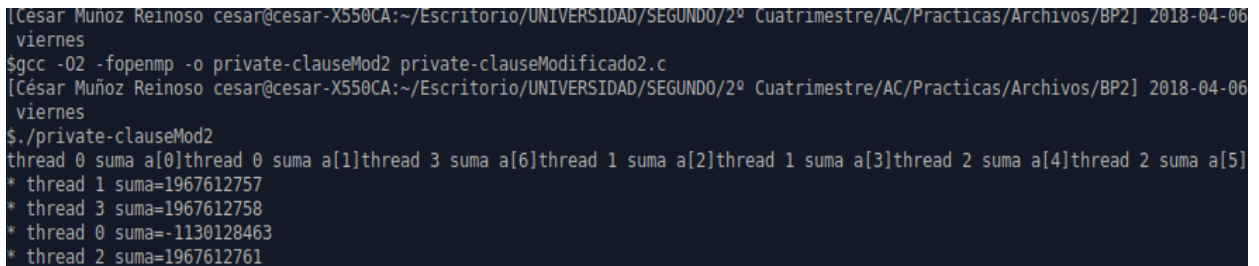
    for(i=0;i<n;i++){
        a[i]=i;

    suma=8;
    #pragma omp parallel private(suma)
    {
        #pragma omp for
        for(i=0;i<n;i++){
            suma=suma + a[i];
            printf("thread %d suma a[%d]",omp_get_thread_num
            (),i);

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

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

CAPTURAS DE PANTALLA:



```
[Cesar Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-06
viernes
$gcc -O2 -fopenmp -o private-clauseMod2 private-clauseModificado2.c
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-06
viernes
$./private-clauseMod2
thread 0 suma a[0]thread 0 suma a[1]thread 3 suma a[6]thread 1 suma a[2]thread 1 suma a[3]thread 2 suma a[4]thread 2 suma a[5]
* thread 1 suma=1967612757
* thread 3 suma=1967612758
* thread 0 suma=-1130128463
* thread 2 suma=1967612761
```

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

RESPUESTA:

Si eliminamos la cláusula `private` en la directiva `parallel`, los threads que se utilizan comparten la misma variable `suma`, por lo que al final del bucle, todos los threads tendrán el mismo resultado.

CAPTURA CÓDIGO FUENTE: private-clauseModificado3.c

```
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread() 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=5;
        #pragma omp for
        for(i=0;i<n;i++){
            suma=suma + a[i];
            printf("thread %d suma a[%d]",omp_get_thread_num(),i);
        }
        printf("\n* thread %d suma=%d",omp_get_thread_num(),suma);
    }
    printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$gcc -O2 -fopenmp private-clauseModificado3.c -o private-clauseMod3
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$./private-clauseMod3
thread 1 suma a[2]thread 1 suma a[3]thread 2 suma a[4]thread 2 suma a[5]thread 0 suma a[0]thread 0 suma a[1]thread 3 suma a[6]
* thread 1 suma=20
* thread 2 suma=20
* thread 3 suma=20
* thread 0 suma=20
```

- 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:

La clausula lastprivate devuelve el ultimo valor del bucle for, en este caso como suma se inicializa a 0, da la causalidad de que en esta ultima iteración el thread solo realiza suma = 0 + 6.

En otros casos cuando el número de threads es diferente o se inicializa la variable suma a otro valor, no siempre devuelve 6 fuera de la region parallel.

CAPTURA CÓDIGO FUENTE: firstlastprivate.c

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

int main(){

    int i,n=7;
    int a[n],suma=0;

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

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

    printf("\nFuera de la construcción parallel suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
\$gcc -O2 -fopenmp firstlastprivate.c -o firstlastprivate
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
\$export OMP_NUM_THREADS=4
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
\$./firstlastprivate
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 3 suma a[6] suma=6

Fuera de la construcción parallel suma=6
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
\$export OMP_NUM_THREADS=2
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
\$./firstlastprivate
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 0 suma a[3] suma=6
thread 1 suma a[4] suma=4
thread 1 suma a[5] suma=9
thread 1 suma a[6] suma=15

Fuera de la construcción parallel suma=15
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
\$export OMP_NUM_THREADS=3
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
\$./firstlastprivate
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 2 suma a[5] suma=5
thread 2 suma a[6] suma=11

Fuera de la construcción parallel 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:

La clausula copyprivate copia las variables privadas de un thread al resto de threads, por lo que si la elimino, vemos en la ejecución que el thread 1 es el que pide el valor, y este no lo copia al resto de threads, por lo que en el for la variable a se encuentra a 0 en el resto de threads.

CAPTURA CÓDIGO FUENTE: copyprivate-clauseModificado.c

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

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

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

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

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

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$gcc -O2 -fopenmp copyprivate-clauseModificado.c -o copyprivate-clauseMod
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$./copyprivate-clauseMod

Introduce valor de inicialización a: 3

Single ejecutada por el thread 1Después de la región parallel:
b[0]=0 b[1]=0 b[2]=0 b[3]=3 b[4]=3 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

RESPUESTA:

Si cambio suma a valor 10, lo único que cambia en la ejecución del programa es que suma empieza en valor 10 y el resultado será 10 más.

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado.c

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

int main(int argc, char**argv){

    int i, n=20, a[n], suma=10;
    if(argc<2){
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }
    n=atoi(argv[1]);
    if(n>20){
        n=20;
        printf("n=%d", n);
    }

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

    #pragma omp parallel for reduction(+:suma)
    for(i=0; i<n; i++) suma += a[i];

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

}
```

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$gcc -O2 -fopenmp reduction-clauseModificado.c -o reduction-clauseMod
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$./reduction-clauseMod 6
Tras 'parallel' suma=25
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$./reduction-clauseMod 10
Tras 'parallel' suma=55
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$./reduction-clauseMod 20
Tras 'parallel' suma=200
```

- 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 .

RESPUESTA:

Si incluimos la directiva `atomic`, conseguimos que solo un thread pueda acceder a la variable suma cada vez, y así ir sumando los valores del vector adecuadamente.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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

int main(int argc, char**argv){

    int i,n=20,a[n],suma=0,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 for
    for(i=0;i<n;i++){
        #pragma omp atomic
        suma += a[i];
    }
    printf("Tras 'parallel' suma=%d\n",suma);

}
```

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$gcc -O2 -fopenmp reduction-clauseModificado7.c -o reduction-clauseMod7
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$./reduction-clauseMod7 20
Tras 'parallel' suma=190
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$./reduction-clauseMod7 10
Tras 'parallel' suma=45
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2] 2018-04-08 domingo
$./reduction-clauseMod7 6
Tras 'parallel' suma=15
```


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):

8

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

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-dynamic.c

```
int main(int argc, char** argv){
    int suma=0;
    struct timespec cgt1, cgt2; double ncgt;
    int **matrix=NULL, *vector = NULL, *salida = NULL;

    if (argc<2){
        printf("Faltan nª componentes del vector\n");
        exit(-1);
    }

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

    vector = (int*) malloc(N*sizeof(int));
    matrix = (int**) malloc(N*sizeof(int*));
    salida = (int*) malloc(N*sizeof(int));

    if((vector == NULL) || (matrix == NULL) || (salida == NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }

    // RESERVAMOS MEMORIA PARA LA MATRIZ
    for (int i = 0; i < N; i++){
        matrix[i] = (int*)malloc(N*sizeof(int));
        if(matrix[i] == NULL) perror("Error: ");
    }

    //INICIALIZAMOS MATRIZ Y VECTOR
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            matrix[i][j]= i+j;
        }
    }

    for(int i = 0; i < N; i++){
        vector[i] = i;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    //MULTIPLICACIÓN
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            suma += matrix[i][j]*vector[j];
        }
        salida[i] = suma;
        suma = 0;
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
}
```

CAPTURA CÓDIGO FUENTE: pmv-secuencial-global.c

```

int main(int argc, char** argv){
    int suma=0;
    struct timespec cgt1, cgt2; double ncgt;

    if (argc<2){
        printf("Faltan nª componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    if(N>MAX) N=MAX;

    int matrix[N][N], vector[N], salida[N];

    //INICIALIZAMOS MATRIZ Y VECTOR
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            matrix[i][j]= i+j;
        }
    }

    for(int i = 0; i < N; i++){
        vector[i] = i;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    //MULTIPLICACIÓN
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            suma += matrix[i][j]*vector[j];
        }
        salida[i] = suma;
        suma = 0;
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);

    //SALIDA
    printf("Vector resultante: ");
    for(int i = 0; i < N; i++){
        printf("%d ",salida[i]);
    }

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

    printf("Tiempo(seg.):%11.9f\n Tamaño:%u\n salida[0]=%d / / salida[%d]=%d \n", ncgt,N,salida
[0],N-1,salida[N-1]);

    return 0;
}

```

CAPTURAS DE PANTALLA:

Ejecución de matriz y vectores dinámicos N=8 y N= 11

```

[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$gcc -fopenmp -O2 pmv-secuencial-dynamic.c -o pmv-secuencial-dynamic
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$ ./pmv-secuencial-dynamic 8
Vector resultante: 140 168 196 224 252 280 308 336 Tiempo(seg.):0.000000932
Tamaño:8
salida[0]=140 / / salida[7]=336
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$ ./pmv-secuencial-dynamic 11
Vector resultante: 385 440 495 550 605 660 715 770 825 880 935 Tiempo(seg.):0.000001092
Tamaño:11
salida[0]=385 / / salida[10]=935

```

Ejecución de matriz y vectores globales N=8 y N=11

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$gcc -fopenmp -O2 pmv-secuencial-global.c -o pmv-secuencial-global
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$./pmv-secuencial-global 8
Vector resultante: 140 168 196 224 252 280 308 336 Tiempo(seg.):0.000001002
Tamaño:8
salida[0]=140 / / salida[7]=336
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$./pmv-secuencial-global 11
Vector resultante: 385 440 495 550 605 660 715 770 825 880 935 Tiempo(seg.):0.000001123
Tamaño:11
salida[0]=385 / / salida[10]=935
```

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`

```
// RESERVAMOS MEMORIA PARA LA MATRIZ
#pragma omp parallel for
for (int i = 0; i < N; i++){
    matrix[i] = (int*)malloc(N*sizeof(int));
    if(matrix[i] == NULL) perror("Error: ");
}

//INICIALIZAMOS MATRIZ Y VECTOR
#pragma omp parallel for
for(int i = 0; i < N; i++){
    for(int j = 0; j < N; j++){
        matrix[i][j] = i+j;
    }
}

#pragma omp parallel for
for(int i = 0; i < N; i++){
    vector[i] = i;
}

clock_gettime(CLOCK_REALTIME, &cgt1);

//MULTIPLICACIÓN
#pragma omp parallel for
for(int i = 0; i < N; i++){
    for(int j = 0; j < N; j++){
        suma += matrix[i][j]*vector[j];
    }
    salida[i] = suma;
    suma = 0;
}
```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

// RESERVAMOS MEMORIA PARA LA MATRIZ
#pragma omp parallel for
for (int i = 0; i < N; i++){
    matrix[i] = (int*)malloc(N*sizeof(int));
    if(matrix[i] == NULL) perror("Error: ");
}

//INICIALIZAMOS MATRIZ Y VECTOR
for(int i = 0; i < N; i++){
    #pragma omp parallel for
    for(int j = 0; j < N; j++){
        matrix[i][j] = i*j;
    }
}

for(int i = 0; i < N; i++){
    vector[i] = i;
}

clock_gettime(CLOCK_REALTIME, &cgt1);

//MULTIPLICACIÓN
for(int i = 0; i < N; i++){
    #pragma omp parallel for
    for(int j = 0; j < N; j++){
        suma += matrix[i][j]*vector[j];
    }
    salida[i] = suma;
    suma = 0;
}

```

RESPUESTA:

No he obtenido ningún error de compilación ni ejecución

CAPTURAS DE PANTALLA:

Ejecución de pmv-OpenMP-a.c N=8 y N=11

```

[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$gcc -fopenmp -O2 pmv-OpenMP-a.c -o pmv-OpenMP-a
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$./pmv-OpenMP-a 8
Vector resultante: 140 168 196 224 252 280 308 336 Tiempo(seg.):0.000005843
Tamaño:8
salida[0]=140 / / salida[7]=336
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$./pmv-OpenMP-a 11
Vector resultante: 385 440 495 550 605 660 715 770 825 880 935 Tiempo(seg.):0.000006174
Tamaño:11
salida[0]=385 / / salida[10]=935

```

Ejecución de pmv-OpenMP-b.c N=8 y N=11

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$gcc -fopenmp -O2 pmv-OpenMP-b.c -o pmv-OpenMP-b
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$./pmv-OpenMP-b 8
Vector resultante: 0 127 254 297 396 495 594 889 Tiempo(seg.):0.000040628
Tamaño:8
salida[0]=0 / / salida[7]=889
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$./pmv-OpenMP-b 11
Vector resultante: 0 335 670 1005 944 1675 1416 2345 2680 2124 2360 Tiempo(seg.):0.000038595
Tamaño:11
salida[0]=0 / / salida[10]=2360
```

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`

```
//MULTIPLICACIÓN
for(int i = 0; i < N; i++){
    #pragma omp parallel for reduction(+:suma)
    for(int j = 0; j < N; j++){
        suma += matrix[i][j]*vector[j];
    }
    salida[i] = suma;
    suma = 0;
}
```

RESPUESTA:

No he obtenido ningún error de compilación ni ejecución

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$gcc -fopenmp -O2 pmv-OpenmMP-reduction.c -o pmv-OpenmMP-reduction
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$./pmv-OpenmMP-reduction 8
Vector resultante: 140 168 196 224 252 280 308 336 Tiempo(seg.):0.000013787
Tamaño:8
salida[0]=140 / / salida[7]=336
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP2/Extra] 2018-04-19 jueves
$./pmv-OpenmMP-reduction 11
Vector resultante: 385 440 495 550 605 660 715 770 825 880 935 Tiempo(seg.):0.000030730
Tamaño:11
salida[0]=385 / / salida[10]=935
```