

# Arquitectura de Computadores (AC)

## Cuaderno de prácticas.

### Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Elena Cantero Molina

Grupo de prácticas: E2

Fecha de entrega:

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:** Lo que ocurre es que nos sale un error de no haber definido la variable “n”, y la directiva `default(none)` nos obliga a a definir todas las variable.

Para arreglar este error hacemos que n sea una variable compartida para todos los threads

#### CAPTURAS DE PANTALLA:

```
usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o shared shared-clauseModificado.c
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:14:11: error: 'n' not specified in enclosing parallel
    #pragma omp parallel for default(none) shared(a)
    ^
shared-clauseModificado.c:14:11: error: enclosing parallel
usuario@asus:~/Escritorio/2.Practica 2$ |
```

#### CÓDIGO FUENTE: `shared-clauseModificado.c`

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

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

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

    #pragma parallel for default(none) shared(a,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]);
}
```

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:** Si inicializamos fuera la variable suma lo que ocurre es que cuando se llega a la región paralela la cláusula `private` va a hacer que las variables tengan un valor indefinido y por lo tanto tendrán basura y cada hebra por consecuente también tendrán valores basura. Por lo tanto el resultado será incorrecto.

### CAPTURAS DE PANTALLA:

#### Inicialización fuera del `parallel`

```
usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o private private-clauseModificado.c
usuario@asus:~/Escritorio/2.Practica 2$ ./private
thread 2 suma a[4] / thread 2 suma a[5] / thread 3 suma a[6] / thread 1 suma a[2] / thread 1 suma a[3] / thread 0 suma a[0] / thread 0 suma a[1] /
* thread 1 suma= 4196565
* thread 3 suma= 4196566
* thread 0 suma= 5
* thread 2 suma= 4196569
usuario@asus:~/Escritorio/2.Practica 2$ |
```

#### Inicialización dentro del `parallel`

```
usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o private private-clauseModificado.c
usuario@asus:~/Escritorio/2.Practica 2$ ./private
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 1 suma= 10
* thread 2 suma= 14
* thread 0 suma= 6
* thread 3 suma= 11
usuario@asus:~/Escritorio/2.Practica 2$ |
```

### CÓDIGO FUENTE: `private-clauseModificado.c`

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char ** argv)
{
    int i, n = 7;
    int a[n], suma = 5;

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

    #pragma omp parallel private(suma)
    {
        //suma=3;
        #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");
}
```

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

**RESPUESTA:** Lo que ocurre es que cuando se tiene la cláusula `private`, cada hebra tiene su suma y por lo tanto el resultado será la última ejecución de cada hebra. Sin embargo, si quitamos el `private(suma)` el resultado que nos va a aparecer es la suma de la última hebra ejecutada en el bucle `for`, independientemente de la hebra.

**CÓDIGO FUENTE: private-clauseModificado2.c**

```

/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

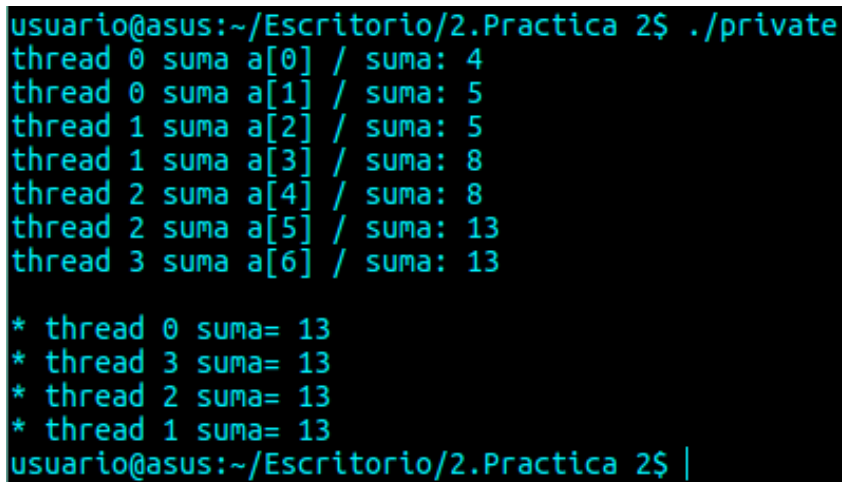
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

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


```

usuario@asus:~/Escritorio/2.Practica 2$ ./private
thread 0 suma a[0] / suma: 4
thread 0 suma a[1] / suma: 5
thread 1 suma a[2] / suma: 5
thread 1 suma a[3] / suma: 8
thread 2 suma a[4] / suma: 8
thread 2 suma a[5] / suma: 13
thread 3 suma a[6] / suma: 13

* thread 0 suma= 13
* thread 3 suma= 13
* thread 2 suma= 13
* thread 1 suma= 13
usuario@asus:~/Escritorio/2.Practica 2$ |

```

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:** El código imprime 6 fuera de la región paralela siempre debido a que con la clausula firstprivate inicializamos la variable suma a 0 y cada hebra hace una copia de ese valor. Al salir de la región paralela el valor de suma en este caso va a ser 6 ya que la cláusula lastprivate coge la copia del valor suma en la última iteración del bucle que en nuestro caso es a[6]=6.

**CAPTURAS DE PANTALLA:**

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o private firstlastprivate-clause.c
usuario@asus:~/Escritorio/2.Practica 2$ ./private
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 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5

Fuera de la construcción parallel suma=6
usuario@asus:~/Escritorio/2.Practica 2$ ./private
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 3 suma a[6] suma=6
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9

Fuera de la construcción parallel suma=6
usuario@asus:~/Escritorio/2.Practica 2$ ./private
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9

Fuera de la construcción parallel suma=6

```

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

**RESPUESTA:** Al eliminar la cláusula copyprivate(a) todas las hebras tienen el valor correcto, este lo tendrán sólo las hebras que pasen por la cláusula single, el resto de hebras tendrán un valor basura. Esto es debido a que la cláusula copyprivate difunde el valor de la variable leída al resto de hebras.

#### CÓDIGO FUENTE: copyprivate-clauseModificado.c

```

/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

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

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

    #pragma omp parallel
    {
        int a;

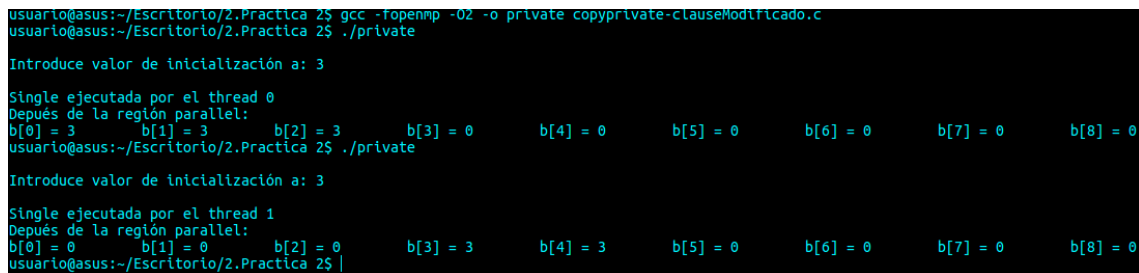
        #pragma omp single
        {
            printf("\nIntroduce valor de inicialización a: ");
            scanf("%d", &a);
            printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}

```

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

printf("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:



```
usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o private copyprivate-clauseModificado.c
usuario@asus:~/Escritorio/2.Practica 2$ ./private

Introduce valor de inicialización a: 3

Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 3      b[1] = 3      b[2] = 3      b[3] = 0      b[4] = 0      b[5] = 0      b[6] = 0      b[7] = 0      b[8] = 0
usuario@asus:~/Escritorio/2.Practica 2$ ./private

Introduce valor de inicialización a: 3

Single ejecutada por el thread 1
Después 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
usuario@asus:~/Escritorio/2.Practica 2$ |
```

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

**RESPUESTA:** Por ejemplo para 20 iteraciones, con `suma=0` se imprime 190 y con `suma=10` se imprime 200. Esto se debe a que cuando la suma es 0 la variable compartida no puede reducirse, sin embargo, con `suma=10` esta variable se va reduciendo en cada iteración.

### CÓDIGO FUENTE: `reduction-clauseModificado.c`

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

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:**

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o reduction reduction-clauseModificado.c
usuario@asus:~/Escritorio/2.Practica 2$ export OMP_NUM_THREADS=3
usuario@asus:~/Escritorio/2.Practica 2$ ./reduction 10
Tras 'parallel' suma=55
usuario@asus:~/Escritorio/2.Practica 2$ ./reduction 20
Tras 'parallel' suma=200
usuario@asus:~/Escritorio/2.Practica 2$ ./reduction 6
Tras 'parallel' suma=25

```

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 .

**RESPUESTA:** Para que se siga realizando la suma de los componentes del vector en paralelo hay que usar la cláusula default(shared) para que todas las variables dentro de la región parallel sean compartidas.

**CÓDIGO FUENTE:** reduction-clauseModificado7.c

```

/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char ** argv)
{
    int i, n=20, a[n], suma=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 default(shared)
    for (i=0; i<n; i++) suma += a[i];

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

```

**CAPTURAS DE PANTALLA:**

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o reduction1 reduction-clauseModificado7.c
usuario@asus:~/Escritorio/2.Practica 2$ ./reduction1 10
Tras 'parallel' suma=45

```

## 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):

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

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).

**CÓDIGO FUENTE:** pmv-secuencial.c

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

#define VECTOR_GLOBAL
// #define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
#define MAX 100
    int v[MAX], matriz[MAX][MAX];
#endif

int main(int argc, char const *argv[])
{
    // comprobamos que los argumentos son correctos
    if (argc < 2) {
        printf("Falta el tamaño de la matriz\n");
        exit(-1);
    }

    int N = atoi(argv[1]);
    int i, j;

    #ifdef VECTOR_GLOBAL
    if (N > MAX) N = MAX;
    #endif

    #ifdef VECTOR_DYNAMIC
    int *v, **matriz;
    v = (int *) malloc(N * sizeof(double)); // malloc necesita el tamaño en bytes
    matriz = (int **) malloc(N * sizeof(int *));
    for (i = 0; i < N; i++)
        matriz[i] = (int *) malloc(N * sizeof(int));

    if ( (v == NULL) || (matriz == NULL) ){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
    #endif

    // inicializamos la matriz y el vector
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++)
            matriz[i][j] = 5;
        v[i] = 3;
    }

    struct timespec cgt1, cgt2; double ncgt; // para tiempo de ejecución
    int vector_resultado[N], k = 0, suma = 0;
```

```

clock_gettime(CLOCK_REALTIME, &cgt1);

// Calculamos el producto
for (i=0; i<N;i++) {
    // calculamos la suma de la columna de la matriz:
    for (k=0; k<N; k++)
        suma+=matriz[k][i];

    // y ahora multiplicamos la suma por el vector:
    vector_resultado[i] = suma * v[i];
    suma = 0;
}

clock_gettime(CLOCK_REALTIME, &cgt2);

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

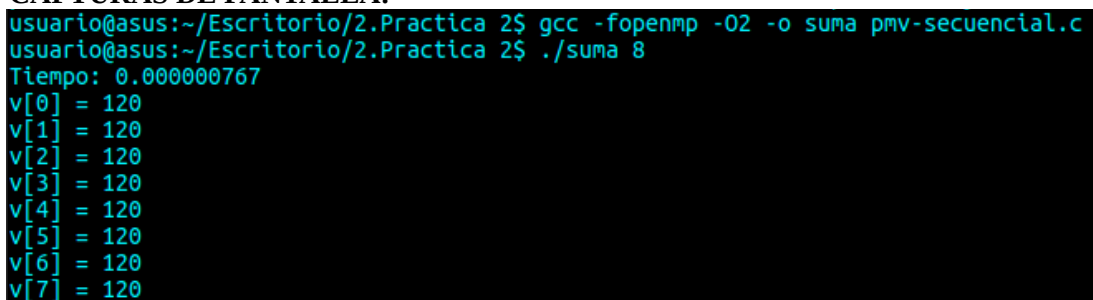
if (N < 11) {
    printf("Tiempo: %11.9f\n", ncgt);
    for (i=0; i<N; i++)
        printf("v[%d] = %d\n", i, vector_resultado[i]);
}

else
    printf("Tiempo: %11.9f \t v[0] = %d \t v[%d] = %d \n", ncgt,
vector_resultado[0],
        N-1, vector_resultado[N-1]);

#ifdef VECTOR_DYNAMIC
    free(v); // libera el espacio reservado para v
    free(matriz); // libera el espacio reservado para matriz
#endif

return 0;
}

```

**CAPTURAS DE PANTALLA:**


```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o suma pmv-secuencial.c
usuario@asus:~/Escritorio/2.Practica 2$ ./suma 8
Tiempo: 0.000000767
v[0] = 120
v[1] = 120
v[2] = 120
v[3] = 120
v[4] = 120
v[5] = 120
v[6] = 120
v[7] = 120

```

9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior usando la directiva `for`. Debe implementar dos versiones del código (consulte la lección 5/Tema 2):
  - a. una primera que paralelice el bucle que recorre las filas de la matriz y
  - b. una segunda que paralelice el bucle que recorre las columnas.

Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula `reduction`**. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:



- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

#### **CÓDIGO FUENTE : pmv-OpenMP-a.c**

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

#define VECTOR_GLOBAL
// #define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
#define MAX 100
    int v[MAX], matriz[MAX][MAX];
#endif

int main(int argc, char const *argv[])
{
    // comprobamos que los argumentos son correctos
    if (argc < 2) {
        printf("Falta el tamaño de la matriz\n");
        exit(-1);
    }

    int N = atoi(argv[1]);
    int i, j;

    #ifdef VECTOR_GLOBAL
    if (N > MAX) N = MAX;
    #endif

    #ifdef VECTOR_DYNAMIC
    int *v, **matriz;
    v = (int *) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
    matriz = (int **) malloc(N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz[i] = (int *) malloc(N*sizeof(int));

    if ( (v==NULL) || (matriz==NULL) ){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
    #endif
}
```

```

// inicializamos la matriz y el vector
#pragma omp parallel for private(i,j)
for (i = 0; i<N; i++) {
    for (j = 0; j<N; j++)
        matriz[i][j] = 5;
    v[i] = 3;
}

int vector_resultado[N], k = 0, suma=0;
double inicio, fin, tiempo;
inicio = omp_get_wtime();

#pragma omp parallel for private(i,k,suma)
for (i=0; i<N;i++) {
    for (k=0; k<N; k++)
        suma+=matriz[k][i];

    vector_resultado[i] = suma * v[i];
    suma = 0;
}

fin = omp_get_wtime();
tiempo = fin - inicio;

if (N < 11) {
    printf("Tiempo: %11.9f\n", tiempo);
    for (i=0; i<N; i++)
        printf("v[%d] = %d\n", i, vector_resultado[i]);
}

else
    printf("Tiempo: %11.9f \t v[0] = %d \t v[%d] = %d \n", tiempo,
vector_resultado[0],
        N-1, vector_resultado[N-1]);

#ifdef VECTOR_DYNAMIC
    free(v); // libera el espacio reservado para v
    free(matriz); // libera el espacio reservado para matriz
#endif

return 0;
}

```

**CÓDIGO FUENTE: pmv-OpenMP-b.c**

```

/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

#define VECTOR_GLOBAL
// #define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
#define MAX 100
    int v[MAX], matriz[MAX][MAX];
#endif

```

```

int main(int argc, char const *argv[])
{
    // comprobamos que los argumentos son correctos
    if (argc < 2) {
        printf("Falta el tamaño de la matriz\n");
        exit(-1);
    }

    int N = atoi(argv[1]);
    int i, j;

    #ifdef VECTOR_GLOBAL
    if (N>MAX) N=MAX;
    #endif

    #ifdef VECTOR_DYNAMIC
    int *v, **matriz;
    v = (int *) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
    matriz = (int **) malloc(N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz[i] = (int *) malloc(N*sizeof(int));

    if ( (v==NULL) || (matriz==NULL) ){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
    #endif

    // inicializamos la matriz y el vector
    #pragma omp parallel for private(i,j)
    for (i = 0; i<N; i++) {
        for (j = 0; j<N; j++)
            matriz[i][j] = 5;
        v[i] = 3;
    }

    int vector_resultado[N], k = 0, suma=0, suma1=0;
    double inicio, fin, tiempo;
    inicio = omp_get_wtime();

    for (i=0; i<N;i++) {
        suma = 0;
        #pragma omp parallel firstprivate(suma1)
        {
            #pragma omp for private(k)
            for (k=0; k<N; k++)
                suma1+=matriz[k][i];

            #pragma omp atomic
            suma += suma1;
        }
        vector_resultado[i] = suma * v[i];
    }

    fin = omp_get_wtime();
    tiempo = fin - inicio;

    if (N < 11) {
        printf("Tiempo: %11.9f\n", tiempo);
        for (i=0; i<N; i++)
            printf("v[%d] = %d\n", i, vector_resultado[i]);
    }
}

```

```

    else
        printf("Tiempo: %11.9f \t v[0] = %d \t v[%d] = %d \n", tiempo,
vector_resultado[0],
            N-1, vector_resultado[N-1]);

#ifdef VECTOR_DYNAMIC
    free(v); // libera el espacio reservado para v
    free(matriz); // libera el espacio reservado para matriz
#endif

    return 0;
}

```

**RESPUESTA:**

pvm-OpenMP-a.c

Al compilar la primera vez, me han aparecido un par de errores con respecto a la sintaxis de la cláusula private y al uso de clock\_gettime. En este caso sólo he tenido que quitar los corchetes que sobraban y cambiar de manera apropiada el clock\_gettime por el omp\_get\_wtime().

Arreglados los fallos de la primera compilación, al volver a compilar no he tenido ningún error y salen los resultados que deben de salir.

Pvm-OpenMP-b.c

Primero he probado a cambiar de sitio el #pragma pero lo que ocurre es que no realiza bien las operaciones. Y entonces he cambiado la cláusula private por firstprivate, pero he obtenido el mismo resultado que en la anterior compilación.

Con la ayuda de un amigo que había cursado la asignatura he podido llegar a arreglar mis errores usando primero la cláusula firstprivate para poder tomar el valor de una variable fuera de la región paralela, segundo la cláusula private como habíamos usado en el anterior apartado y por último la directiva atomic para conseguir que no todas las hebras modifiquen a la vez la variable dentro de esta directiva.

**CAPTURAS DE PANTALLA:**

A- captura de pantalla 1: Compilación con errores

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o suma pvm-OpenMP-a.c
pvm-OpenMP-a.c: In function 'main':
pvm-OpenMP-a.c:61:42: error: expected '#pragma omp' clause before '{' token
    #pragma omp parallel for private(i,j){
                                   ^
In file included from /usr/include/time.h:41:0,
                 from pvm-OpenMP-a.c:4:
pvm-OpenMP-a.c: At top level:
pvm-OpenMP-a.c:69:19: error: expected declaration specifiers or '...' before numeric constant
    clock_gettime(CLOCK_REALTIME, &cgt1);
                  ^
pvm-OpenMP-a.c:69:35: error: expected declaration specifiers or '...' before '&' token
    clock_gettime(CLOCK_REALTIME, &cgt1);
                              ^
pvm-OpenMP-a.c:71:13: error: expected declaration specifiers before '#pragma'
    #pragma omp parallel for private(i,k,suma,vector_resultado){
                ^
pvm-OpenMP-a.c:101:1: error: expected identifier or '(' before ')' token
}
^
usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o suma pvm-OpenMP-a.c
pvm-OpenMP-a.c: In function 'main':
pvm-OpenMP-a.c:83:36: error: 'ncgt' undeclared (first use in this function)
    printf("Tiempo: %11.9f\n", ncgt);
                               ^
pvm-OpenMP-a.c:83:36: note: each undeclared identifier is reported only once for each function it appears in

```

A-captura de pantalla 2: Compilación sin errores

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o suma pmv-OpenMP-a.c
usuario@asus:~/Escritorio/2.Practica 2$ ./suma 8
Tiempo: 0.000004136
v[0] = 120
v[1] = 120
v[2] = 120
v[3] = 120
v[4] = 120
v[5] = 120
v[6] = 120
v[7] = 120
usuario@asus:~/Escritorio/2.Practica 2$ |

```

B-captura de pantalla 1: Compilación con errores 1

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o suma pmv-OpenMP-b.c
usuario@asus:~/Escritorio/2.Practica 2$ ./suma 8
Tiempo: 0.000036894
v[0] = 0
v[1] = 0
v[2] = 0
v[3] = 0
v[4] = 0
v[5] = 0
v[6] = 0
v[7] = 0

```

B-captura de pantalla 2: Compilación sin errores

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o suma pmv-OpenMP-b.c
usuario@asus:~/Escritorio/2.Practica 2$ ./suma 8
Tiempo: 0.000022958
v[0] = 120
v[1] = 120
v[2] = 120
v[3] = 120
v[4] = 120
v[5] = 120
v[6] = 120
v[7] = 120

```

10. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula `reduction`. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

**CÓDIGO FUENTE:** pmv-OpenmMP-reduction.c

```

/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

#define VECTOR_GLOBAL
// #define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL

```

```

#define MAX 100
int v[MAX], matriz[MAX][MAX];
#endif

int main(int argc, char const *argv[])
{
    // comprobamos que los argumentos son correctos
    if (argc < 2) {
        printf("Falta el tamaño de la matriz\n");
        exit(-1);
    }

    int N = atoi(argv[1]);
    int i, j;

    #ifndef VECTOR_GLOBAL
    if (N>MAX) N=MAX;
    #endif

    #ifndef VECTOR_DYNAMIC
    int *v, **matriz;
    v = (int *) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
    matriz = (int **) malloc(N*sizeof(int *));
    for(i=0; i<N; i++)
        matriz[i] = (int *) malloc(N*sizeof(int));

    if ( (v==NULL) || (matriz==NULL) ){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
    #endif

    // inicializamos la matriz y el vector
    #pragma omp parallel for private(i,j)
    for (i = 0; i<N; i++) {
        for (j = 0; j<N; j++)
            matriz[i][j] = 5;
        v[i] = 3;
    }

    int vector_resultado[N], k = 0, suma=0;
    double inicio, fin, tiempo;
    inicio = omp_get_wtime();

    for (i=0; i<N;i++) {
        suma = 0;
        #pragma omp parallel for reduction(+:suma)

        for (k=0; k<N; k++)
            suma+=matriz[k][i];

        vector_resultado[i] = suma * v[i];
    }

    fin = omp_get_wtime();
    tiempo = fin - inicio;

    if (N < 11) {
        printf("Tiempo: %11.9f\n", tiempo);
        for (i=0; i<N; i++)
            printf("v[%d] = %d\n", i, vector_resultado[i]);
    }
}

```

```

    else
        printf("Tiempo: %11.9f \t v[0] = %d \t v[%d] = %d \n", tiempo,
vector_resultado[0],
            N-1, vector_resultado[N-1]);

#ifdef VECTOR_DYNAMIC
    free(v); // libera el espacio reservado para v
    free(matriz); // libera el espacio reservado para matriz
#endif

    return 0;
}

```

**RESPUESTA:** Lo primero que he hecho ha sido sustituir la cláusula firstprivate por la cláusula reduction y quitar la cláusula private del segundo bucle for y como se observa en la captura de pantalla 1, los resultados son erróneos.

Después para intentar solucionar lo anterior e intentado quitar la directiva atomic y lo que llevaba después. Pero como se observa en la captura de pantalla 2, he vuelto a tener errores a la hora de mostrar los resultados.

Para solucionar lo anteriormente dicho, he cambiado la lista que se le pasa a reduction y como se puede ver en la última captura de pantalla, los resultados son correctos.

#### CAPTURAS DE PANTALLA:

Captura de pantalla 1: Compilación con errores 1

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o suma pmv-OpenMP-reduction.c
usuario@asus:~/Escritorio/2.Practica 2$ ./suma 8
Tiempo: 0.000015868
v[0] = 480
v[1] = 480
v[2] = 480
v[3] = 480
v[4] = 480
v[5] = 480
v[6] = 480
v[7] = 480

```

Captura de pantalla 2: Compilación con errores 2

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o suma pmv-OpenMP-reduction.c
usuario@asus:~/Escritorio/2.Practica 2$ ./suma 8
Tiempo: 0.000035877
v[0] = 360
v[1] = 360
v[2] = 360
v[3] = 360
v[4] = 360
v[5] = 240
v[6] = 360
v[7] = 360

```

Captura de pantalla 3: Compilación sin errores

```

usuario@asus:~/Escritorio/2.Practica 2$ gcc -fopenmp -O2 -o suma pmv-OpenMP-reduction.c
usuario@asus:~/Escritorio/2.Practica 2$ ./suma 8
Tiempo: 0.000080147
v[0] = 120
v[1] = 120
v[2] = 120
v[3] = 120
v[4] = 120
v[5] = 120
v[6] = 120
v[7] = 120

```

11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local 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.

**TABLA Y GRÁFICA (por ejemplo para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: alguno del orden de cientos de miles):**

**COMENTARIOS SOBRE LOS RESULTADOS:**