

2º curso / 2º cuatr.

Grados Ing.  
Inform.

# Arquitectura de Computadores (AC)

## Cuaderno de prácticas.

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

Estudiante (nombre y apellidos):

Grupo de prácticas y profesor de prácticas:

### Ejercicios basados en los ejemplos del seminario práctico

1. (a) Añadir la cláusula `default(none)` a la directiva `parallel` del ejemplo del seminario `shared-clause.c`? ¿Qué ocurre? ¿A qué se debe? (b) Resolver el problema generado sin eliminar `default(none)`. Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla que muestren lo que ocurre)

**RESPUESTA:** Si añadimos al código la cláusula `default(none)`:

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

Nos da el siguiente error de compilación:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ gcc -O2 -fopenmp -o shared-clause shared-clause.c
shared-clause.c: In function 'main':
shared-clause.c:83:12: error: 'n' not specified in enclosing 'parallel'
  83 |     #pragma omp parallel for default(none) shared(a)
      |           ^~~
shared-clause.c:83:12: error: enclosing 'parallel'
```

Consultando el manual online de OpenMP, esto se debe a que al usar `default(none)`, es el propio programador el que se encarga de especificar el ámbito de cada variable usada en la región que se ejecuta en paralelo. En este caso, es la variable `n` la que no se ha especificado su ámbito de alcance, por ello da un error de compilación. Esto se soluciona añadiendo `n` en la cláusula `shared` donde teníamos `a`.

**CAPTURA CÓDIGO FUENTE:** `shared-clauseModificado.c`

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

**CAPTURAS DE PANTALLA:** En la siguiente captura vemos que ya sí compila y el comportamiento es el esperado:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ gcc -O2 -fopenmp -o shared-clause shared-clause.c
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./shared-clause
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
```

2. (a) Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel`. Inicializar suma dentro del `parallel` a un valor distinto de 0. Ejecutar varias veces el código ¿Qué imprime el código fuera del `parallel`? (mostrar lo que ocurre con una captura de pantalla) Razonar respuesta. (b) Modificar el código del apartado (a) para que se inicialice suma fuera del `parallel` en lugar de dentro ¿Qué ocurre? Comparar todo lo que imprime el código ahora con la salida en (a) (mostrar la salida con una captura de pantalla) Razonar respuesta.

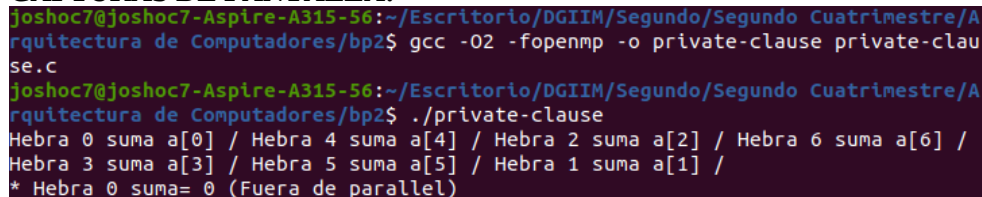
(a) RESPUESTA:

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.c`

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

    printf("\n* Hebra %d suma= %d",
        omp_get_thread_num(),suma);
    printf("\n"); return(0);
}
```

CAPTURAS DE PANTALLA:



```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ gcc -O2 -fopenmp -o private-clause private-clause.c
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./private-clause
Hebra 0 suma a[0] / Hebra 4 suma a[4] / Hebra 2 suma a[2] / Hebra 6 suma a[6] /
Hebra 3 suma a[3] / Hebra 5 suma a[5] / Hebra 1 suma a[1] /
* Hebra 0 suma= 0 (Fuera de parallel)
```

(b) RESPUESTA:

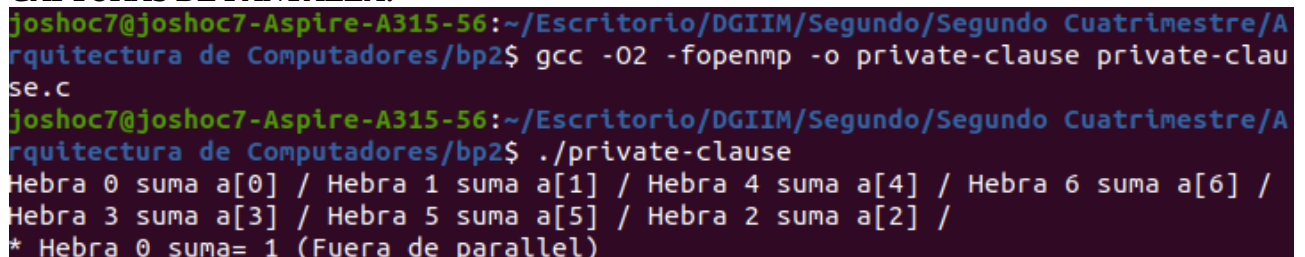
CAPTURA CÓDIGO FUENTE: `private-clauseModificado_b.c`

```
suma=1;

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

    printf("\n* Hebra %d suma= %d (Fuera de parallel)",
        omp_get_thread_num(),suma);
    printf("\n"); return(0);
}
```

CAPTURAS DE PANTALLA:



```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ gcc -O2 -fopenmp -o private-clause private-clause.c
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./private-clause
Hebra 0 suma a[0] / Hebra 1 suma a[1] / Hebra 4 suma a[4] / Hebra 6 suma a[6] /
Hebra 3 suma a[3] / Hebra 5 suma a[5] / Hebra 2 suma a[2] /
* Hebra 0 suma= 1 (Fuera de parallel)
```

Todo este comportamiento se debe a que los cálculos realizados sobre la variable `suma` se quedan en la región paralela, es decir, `suma` es una variable privada a cada thread, y lo que se imprime fuera de la región paralela es el valor de la variable `suma` que se ha declarado fuera de la región paralela.

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

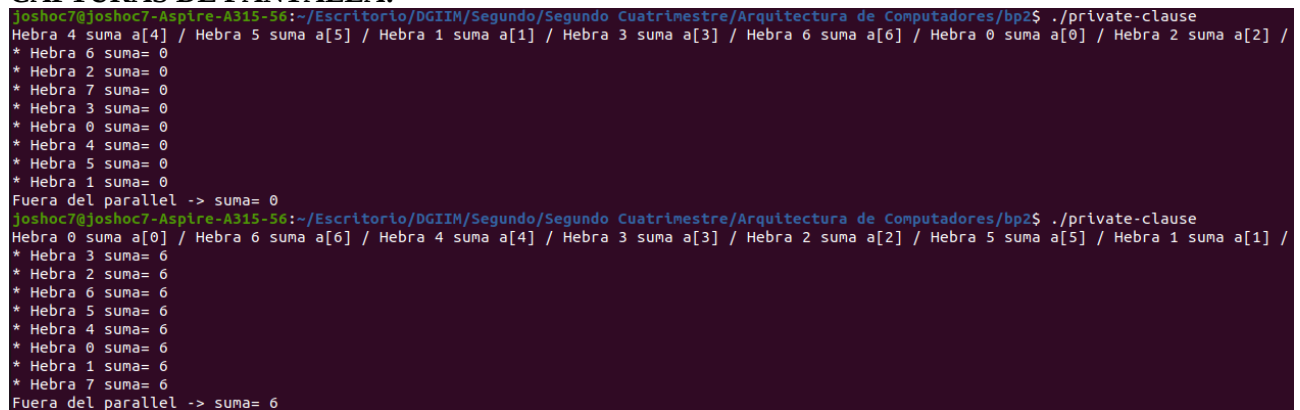
RESPUESTA:

**CAPTURA CÓDIGO FUENTE: private-clauseModificado3.c**

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

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

printf("\nFuera del parallel -> suma= %d", suma);
printf("\n"); return(0);
}
```

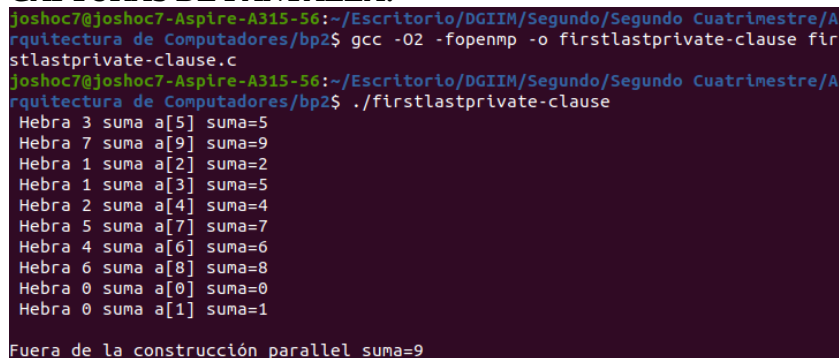
**CAPTURAS DE PANTALLA:**


```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./private-clause
Hebra 4 suma a[4] / Hebra 5 suma a[5] / Hebra 1 suma a[1] / Hebra 3 suma a[3] / Hebra 6 suma a[6] / Hebra 0 suma a[0] / Hebra 2 suma a[2] /
* Hebra 6 suma= 0
* Hebra 2 suma= 0
* Hebra 7 suma= 0
* Hebra 3 suma= 0
* Hebra 0 suma= 0
* Hebra 4 suma= 0
* Hebra 5 suma= 0
* Hebra 1 suma= 0
Fuera del parallel -> suma= 0
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./private-clause
Hebra 0 suma a[0] / Hebra 6 suma a[6] / Hebra 4 suma a[4] / Hebra 3 suma a[3] / Hebra 2 suma a[2] / Hebra 5 suma a[5] / Hebra 1 suma a[1] /
* Hebra 3 suma= 6
* Hebra 2 suma= 6
* Hebra 6 suma= 6
* Hebra 5 suma= 6
* Hebra 4 suma= 6
* Hebra 0 suma= 6
* Hebra 1 suma= 6
* Hebra 7 suma= 6
Fuera del parallel -> suma= 6
```

Ahora la variable suma es compartida. Con mayor probabilidad, se imprime suma = 0 tanto para cada hebra como fuera del parallel. Esto se debe a que la hebra 0 lee suma antes de ser escrita por cualquier otra hebra y es la última en escribir en suma el valor a[0] = 0, de ahí que al final todas las hebras se queden con el último valor escrito. Sin embargo, alguna vez se imprime un valor distinto, como podemos observar en la captura, que en este caso habrá sido la hebra 6 la última en escribir.

4. En la ejecución de firstlastprivate.c de la pag. 21 del seminario se imprime un 6 fuera de la región parallel. **(a)** Cambiar el tamaño del vector a 10. Razonar lo que imprime el código en su PC con esta modificación. (añadir capturas de pantalla que muestren lo que ocurre). **(b)** Sin cambiar el tamaño del vector ¿podría imprimir el código otro valor? Razonar respuesta (añadir capturas de pantalla que muestren lo que ocurre).

**(a) RESPUESTA:** Se imprime siempre el valor 9. Esto se debe a que al incluir la cláusula lastprivate(suma), el valor de suma que se obtiene fuera de la región paralela es el de la variable privada suma de la última iteración, en la que se asigna a suma el valor 9.

**CAPTURAS DE PANTALLA:**


```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ gcc -O2 -fopenmp -o firstlastprivate-clause firstlastprivate-clause.c
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./firstlastprivate-clause
Hebra 3 suma a[5] suma=5
Hebra 7 suma a[9] suma=9
Hebra 1 suma a[2] suma=2
Hebra 1 suma a[3] suma=5
Hebra 2 suma a[4] suma=4
Hebra 5 suma a[7] suma=7
Hebra 4 suma a[6] suma=6
Hebra 6 suma a[8] suma=8
Hebra 0 suma a[0] suma=0
Hebra 0 suma a[1] suma=1
Fuera de la construcción parallel suma=9
```

**(b) RESPUESTA:** Para poder obtener un valor distinto de 9 sin cambiar el tamaño del vector, podemos

cambiar el número de hebras a usar por uno menor, obligando a que el valor de suma de la última iteración resulte de la suma acumulada de varios elementos del vector (es decir, antes se imprimía 9 porque justo en la última iteración correspondía a una única hebra, la 7, que sumaba el valor de  $a[9]$ . Ahora haremos que la última iteración le corresponda a una hebra que ya hubiese realizado alguna iteración anterior).

### CAPTURAS DE PANTALLA:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ export OMP_NUM_THREADS=4
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./firstlastprivate-clause
Hebra 0 suma a[0] suma=0
Hebra 0 suma a[1] suma=1
Hebra 0 suma a[2] suma=3
Hebra 3 suma a[8] suma=8
Hebra 3 suma a[9] suma=17
Hebra 2 suma a[6] suma=6
Hebra 2 suma a[7] suma=13
Hebra 1 suma a[3] suma=3
Hebra 1 suma a[4] suma=7
Hebra 1 suma a[5] suma=12
Fuera de la construcción parallel suma=17
```

Aquí, la hebra 3 se encarga de las dos últimas iteraciones ( $8+9=17$ ), de ahí que salga 17.

5. **(a)** ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? **(b)** ¿A qué cree que es debido? (añadir una captura de pantalla que muestre lo que ocurre)

### RESPUESTA:

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

```
#pragma omp parallel
{
    int a;
    #pragma omp single
    {
        printf("Introduce valor de inicialización a: ");scanf("%d",&a);
        printf("Single ejecutada por la hebra %d\n",
            omp_get_thread_num());
    }

    #pragma omp for
    for (i=0; i<n; i++){
        printf("Hebra %d asigna a b[%d] el valor %d\n",omp_get_thread_num(),i,a);
        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");
return(0);
}
```

### CAPTURAS DE PANTALLA:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ gcc -O2 -fopenmp -o copyprivate-clause copyprivate-clause.c
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./copyprivate-clause
Introduce valor de inicialización a: 4
Single ejecutada por la hebra 3
Hebra 7 asigna a b[8] el valor 0
Hebra 3 asigna a b[4] el valor 4
Hebra 6 asigna a b[7] el valor 0
Hebra 4 asigna a b[5] el valor 0
Hebra 5 asigna a b[6] el valor 0
Hebra 2 asigna a b[3] el valor 0
Hebra 0 asigna a b[0] el valor 21959
Hebra 0 asigna a b[1] el valor 21959
Hebra 1 asigna a b[2] el valor 0
Después de la región parallel:
b[0] = 21959    b[1] = 21959    b[2] = 0        b[3] = 0        b[4] = 4
b[5] = 0        b[6] = 0        b[7] = 0        b[8] = 0
```

Este comportamiento se debe a que al quitar `copyprivate(a)`, el valor de `a` que ha sido leído de forma secuencial por la hebra 3 no es copiado a la variable `a` del resto de hebras, de ahí que las demás hebras asignen valores basura a los elementos del vector.

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado (añada capturas de pantalla que muestren lo que ocurre)

**RESPUESTA:**

**CAPTURA CÓDIGO FUENTE: reduction-clauseModificado.c**

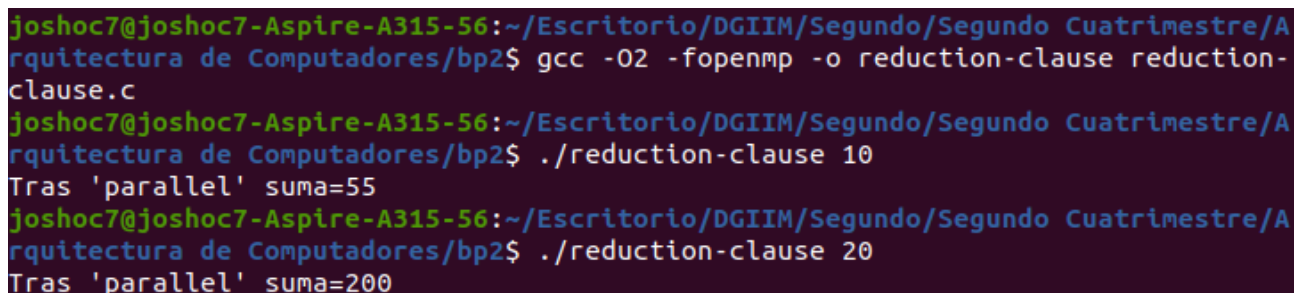
```
int main(int argc, char **argv)
{
    int i, n=20;
    int a[n], suma=10;
    if(argc < 2) {
        fprintf(stderr, "[ERROR]-Falta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d",n);}

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

    #pragma omp parallel for default(none) private(i) shared(a,n) reduction(+:suma)
    for (i=0; i<n; i++){
        suma += a[i];
    }

    printf("Tras 'parallel' suma=%d\n", suma);
    return(0);
}
```

**CAPTURAS DE PANTALLA:**



```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ gcc -O2 -fopenmp -o reduction-clause reduction-clause.c
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./reduction-clause 10
Tras 'parallel' suma=55
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./reduction-clause 20
Tras 'parallel' suma=200
```

El resultado ahora es el que debería dar pero sumándole 10 (antes para 10 el resultado era 45 y para 20 era 190).

7. En el ejemplo `reduction-clause.c`, elimine `reduction()` de `#pragma omp parallel for` `reduction(+:suma)` y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector `a` en paralelo sin añadir más directivas de trabajo compartido (añada capturas de pantalla que muestren lo que ocurre).

**RESPUESTA:**

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

```
int main(int argc, char **argv)
{
    int i, n=20;
    int a[n], suma=0;
    if(argc < 2) {
        fprintf(stderr, "[ERROR]-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(none) private(i) shared(a,n,suma)
    for (i=0; i<n; i++){
        #pragma omp atomic
        suma += a[i];
    }

    printf("Tras 'parallel' suma=%d\n", suma);
    return(0);
}
```

es

He añadido suma dentro del shared ya que de lo contrario, no compilaba debido a lo explicado en el ejercicio 1. Si se pone suma privada, pasaría lo explicado en el ejercicio 2. Además, antes suma era una variable privada para cada hebra y al final del bucle, gracias a reduction, se sumaban todas estas sumas. Ahora suma es compartida y para evitar que varias hebras escriban el valor en cada iteración sobrescribiendo el valor que tuviese antes suma, añadimos un atomic (no es de trabajo compartido) para asegurarnos de que entran una a una en el bucle sin sobrescribir el valor actualizado por las demás hebras.

#### CAPTURAS DE PANTALLA:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ gcc -O2 -fopenmp -o reduction-clause reduction-clause.c
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./reduction-clause 10
Tras 'parallel' suma=45
```

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

8. Implementar en paralelo el producto matriz por vector con OpenMP usando la directiva `for`. Partir del código secuencial disponible en SWAD. Debe implementar dos versiones del código (consulte la lección 5/Tema 2):

- una primera que paralelice el bucle que recorre las filas de la matriz y
- 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, `v2`, para tamaños pequeños de los vectores (por ejemplo, `N = 8` y `N=11`); (4) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código que calcula el producto matriz vector, el número de hilos que usa 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

```
#pragma omp parallel
{

#pragma omp single // Así sólo se imprime una vez el número de hebras
{
printf("Número de hilos:%d\n",omp_get_num_threads());
}

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

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



```

for(i = 0; i < N; i++){

    #pragma omp parallel private(suma)
    {
        #pragma omp for
        for (j = 0; j < N; j++)
            suma[i] += m[i][j] * v1[j];

        #pragma omp critical
        v2[i]+=suma[i];
    }
}

```

RESPUESTA:

CAPTURAS DE PANTALLA:

```

joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-secuencial 8
Tiempo: 0.000000540      Tamaño: 8
Matriz:
0.000000  1.000000  2.000000  3.000000  4.000000  5.000000  6.000000  7.000000
8.000000  9.000000  10.000000  11.000000  12.000000  13.000000  14.000000  15.000000
16.000000  17.000000  18.000000  19.000000  20.000000  21.000000  22.000000  23.000000
24.000000  25.000000  26.000000  27.000000  28.000000  29.000000  30.000000  31.000000
32.000000  33.000000  34.000000  35.000000  36.000000  37.000000  38.000000  39.000000
40.000000  41.000000  42.000000  43.000000  44.000000  45.000000  46.000000  47.000000
48.000000  49.000000  50.000000  51.000000  52.000000  53.000000  54.000000  55.000000
56.000000  57.000000  58.000000  59.000000  60.000000  61.000000  62.000000  63.000000

Vector:
0.000000  0.100000  0.200000  0.300000  0.400000  0.500000  0.600000  0.700000

Vector resultado:
14.000000  36.400000  58.800000  81.200000  103.600000  126.000000  148.400000  170.800000
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-paralelo-filas 8
Número de hilos:8
Tiempo: 0.000434745      Tamaño: 8
Matriz:
0.000000  1.000000  2.000000  3.000000  4.000000  5.000000  6.000000  7.000000
8.000000  9.000000  10.000000  11.000000  12.000000  13.000000  14.000000  15.000000
16.000000  17.000000  18.000000  19.000000  20.000000  21.000000  22.000000  23.000000
24.000000  25.000000  26.000000  27.000000  28.000000  29.000000  30.000000  31.000000
32.000000  33.000000  34.000000  35.000000  36.000000  37.000000  38.000000  39.000000
40.000000  41.000000  42.000000  43.000000  44.000000  45.000000  46.000000  47.000000
48.000000  49.000000  50.000000  51.000000  52.000000  53.000000  54.000000  55.000000
56.000000  57.000000  58.000000  59.000000  60.000000  61.000000  62.000000  63.000000

Vector:
0.000000  0.100000  0.200000  0.300000  0.400000  0.500000  0.600000  0.700000

Vector resultado:
14.000000  36.400000  58.800000  81.200000  103.600000  126.000000  148.400000  170.800000

```

```

joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-secuencial 8
Tiempo: 0.000000566      Tamaño: 8
Matriz:
0.000000  1.000000  2.000000  3.000000  4.000000  5.000000  6.000000  7.000000
8.000000  9.000000  10.000000  11.000000  12.000000  13.000000  14.000000  15.000000
16.000000  17.000000  18.000000  19.000000  20.000000  21.000000  22.000000  23.000000
24.000000  25.000000  26.000000  27.000000  28.000000  29.000000  30.000000  31.000000
32.000000  33.000000  34.000000  35.000000  36.000000  37.000000  38.000000  39.000000
40.000000  41.000000  42.000000  43.000000  44.000000  45.000000  46.000000  47.000000
48.000000  49.000000  50.000000  51.000000  52.000000  53.000000  54.000000  55.000000
56.000000  57.000000  58.000000  59.000000  60.000000  61.000000  62.000000  63.000000

Vector:
0.000000  0.100000  0.200000  0.300000  0.400000  0.500000  0.600000  0.700000

Vector resultado:
14.000000  36.400000  58.800000  81.200000  103.600000  126.000000  148.400000  170.800000
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-paralelo-columnas 8
Tiempo: 0.000000529      Tamaño: 8
Matriz:
0.000000  1.000000  2.000000  3.000000  4.000000  5.000000  6.000000  7.000000
8.000000  9.000000  10.000000  11.000000  12.000000  13.000000  14.000000  15.000000
16.000000  17.000000  18.000000  19.000000  20.000000  21.000000  22.000000  23.000000
24.000000  25.000000  26.000000  27.000000  28.000000  29.000000  30.000000  31.000000
32.000000  33.000000  34.000000  35.000000  36.000000  37.000000  38.000000  39.000000
40.000000  41.000000  42.000000  43.000000  44.000000  45.000000  46.000000  47.000000
48.000000  49.000000  50.000000  51.000000  52.000000  53.000000  54.000000  55.000000
56.000000  57.000000  58.000000  59.000000  60.000000  61.000000  62.000000  63.000000

Vector:
0.000000  0.100000  0.200000  0.300000  0.400000  0.500000  0.600000  0.700000

Vector resultado:
14.000000  36.400000  58.800000  81.200000  103.600000  126.000000  148.400000  170.800000

```

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-secuencial 11
Tiempo: 0.000000785      Tamaño: 11      v2[0]: 38.500000      v2[N-1]: 643.500000
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-paralelo-filas 11
Número de hilos:8
Tiempo: 0.000395385      Tamaño: 11      v2[0]: 38.500000      v2[N-1]: 643.500000
```

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-secuencial 11
Tiempo: 0.000000698      Tamaño: 11      v2[0]: 38.500000      v2[N-1]: 643.500000
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-paralelo-columnas 11
Tiempo: 0.000000602      Tamaño: 11      v2[0]: 38.500000      v2[N-1]: 643.500000
```

Cuando hemos paralelizado el bucle que recorre las columnas, hemos tenido que añadir un `#pragma omp critical` mientras que en la versión en la que se paraleliza el bucle de las filas no ha sido necesario. Esto se debe a que en la primera versión, cada elemento del vector `v2` es accedido y actualizado por una hebra en concreto, mientras que en la segunda versión no, por lo que es necesario declarar como región crítica el cálculo de cada elemento de `v2` (de no hacerlo así, las hebras sobrescribirían lo escrito por las demás y el resultado no sería correcto).

9. 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`

```
for(i = 0; i < N; i++){
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Número de hilos:%d\n",omp_get_num_threads());
        }

        #pragma omp for reduction(+:v2[i])
        for (j = 0; j < N; j++)
            v2[i] += m[i][j] * v1[j];
    }
}
```

**RESPUESTA:**

**CAPTURAS DE PANTALLA:** Comprobación de que funciona correctamente

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-secuencial 11
Tiempo: 0.000000621      Tamaño: 11      v2[0]: 38.500000      v2[N-1]: 643.500000
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-OpenMP-reduction 11
Número de hilos:8
Número de hilos:8
Número de hilos:8
Número de hilos:8
Número de hilos:8
Número de hilos:8
Número de hilos:8
Número de hilos:8
Número de hilos:8
Número de hilos:8
Número de hilos:8
Tiempo: 0.000605877      Tamaño: 11      v2[0]: 38.500000      v2[N-1]: 643.500000
```

10. Realizar una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en `atcgrid4`, en uno de los nodos de la cola `ac` y en su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (`N`) distintos (consulte la Lección 6/Tema 2). Usar `-O2` al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en `atcgrid` código que imprima todos los componentes del resultado.

**CAPTURAS DE PANTALLA (que justifique el código elegido):**



```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-paralelo-filas 8000
Número de hilos:8
Tiempo: 0.019438015      Tamaño: 8000      v2[0]: 17063466800.000000      v2[N-1]: 204765866666800.000000
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-paralelo-columnas 8000
Tiempo: 6.387781986      Tamaño: 8000      v2[0]: 17063466800.000000      v2[N-1]: 204765866666800.000000
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp2$ ./pmv-OpenMP-reduction 8000
Tiempo: 0.033732472      Tamaño: 8000      v2[0]: 17063466800.000000      v2[N-1]: 204765866666800.000000
```

Se escoge la primera versión del producto en paralelo.

**JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:** La diferencia de tiempos se debe a que en la segunda versión, declarar una región como crítica obliga a que las hebras van a tener que esperar a que cada una realice su trabajo, dando lugar al programa con ejecución más lenta. El tercer programa tarda bastante menos ya que las hebras no tienen que esperarse entre sí, aunque tarda mas que el primer programa porque hay comunicación implicada por el reduction, lo cual añade tiempo. Por lo tanto, la primera versión es la más rápida al no requerir esperas entre hebras ni comunicación.

### CAPTURA DE PANTALLA del script pmv-OpenMP-script.sh

Créditos a Daniel Pérez Ruiz, que hizo un script para ejecutar en su PC, en atcgrid(1,2 ó 3) y en atcgrid4 el programa cuyo código paraleliza el bucle que recorre las filas para distintos tamaños de N, además de las tablas y las gráficas:

```
#!/bin/bash
#Nombre: getTimesTablaPC.sh
#Author: Daniel Pérez Ruiz
#Fecha: 08/04/2020
#Descripción: Obtiene los tiempos del programa especificado y devuelve la salida a un fichero .dat
#Uso: ./getTimesTablaPC.sh

#CONSTRUCCION DE NOMBRES
DIR_BIN="bin"
PMV="pmv-"
SEC="secuencial"
OMP="OpenMP-"
A="a"
B="b"
RED="reduction"

#FICHERO OPENMP-A
FILE="$DIR_BIN/$PMV$OMP$A"

#CONSTRUCCION DE FICHERO DE SALIDA
OUT_DIR="TIEMPOS"
EXT=".dat"
ULINE=" "
FICHERO="$OUT_DIR/$PMV$OMP$A$ULINE$OUT_DIR$EXT"

#INTERVALO DE TIEMPOS
TIME1=5000
TIME2=30000
INC=1000

#=====
#OBTENCION TIEMPOS
mkdir -p $OUT_DIR

#TIEMPOS PARA 1 HEBRA
NUM_HEBRAS=1
export OMP_NUM_THREADS=$NUM_HEBRAS

echo "TIEMPOS PARA NUMERO DE HEBRAS :: ($NUM_HEBRAS)"
echo "TIEMPOS PARA NUMERO DE HEBRAS :: ($NUM_HEBRAS)" >> $FICHERO
echo "===== " >> $FICHERO
echo " " >> $FICHERO
for ((N=$TIME1; N<=$TIME2; N+=INC))
do
    echo "EJECUCION [$FILE] PARA DIMENSION [$N] :: RESTANTE [$N <= $TIME2]"
    ./FILE $N >> $FICHERO
done
echo " " >> $FICHERO

CAPTURA 23: SCRIPT PARA OBTENCIÓN DE TIEMPOS DE OPENMP-A
(PC LOCAL). NOTA: EL DE ATCGRID ES UNA VERSIÓN ALGO MÁS
REDUCIDA, VÉASE EJER11/SCRIPTS/GETITMESTABLAATCGRID.SH
```

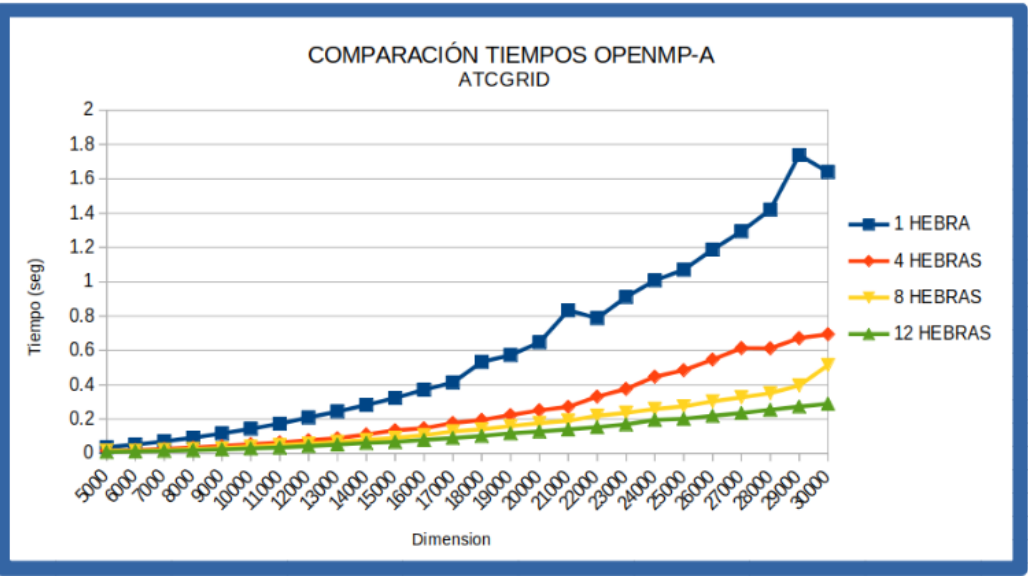
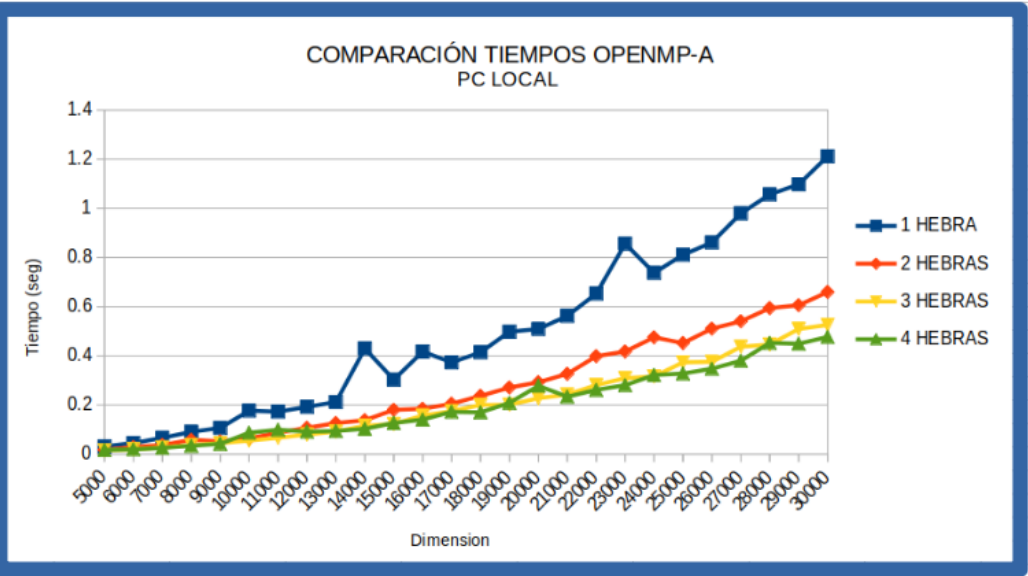
**CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):**

**TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia):**

COMPARATIVA TIEMPOS EN FUNCIÓN DE HEBRAS UTILIZADAS - PMV-OPENMP-A - PC LOCAL				
TAMANO	1 HEBRA	2 HEBRAS	3 HEBRAS	4 HEBRAS
5000	0.031290652	0.016956165	0.014664164	0.016515029
6000	0.045037597	0.028796073	0.019893824	0.019554766
7000	0.065991063	0.036962047	0.027068152	0.025150221
8000	0.090655557	0.058012814	0.036351871	0.03355689
9000	0.106416882	0.051776047	0.044524517	0.041648542
10000	0.176581112	0.063860988	0.054909447	0.087002077
11000	0.172355846	0.087204945	0.066376752	0.09931524
12000	0.192152312	0.106877787	0.079815454	0.091440594
13000	0.211886544	0.126328257	0.091715236	0.094139829
14000	0.429578452	0.13780322	0.115274366	0.1027717
15000	0.303048371	0.180048427	0.122436564	0.12674655
16000	0.417020805	0.184348053	0.156844981	0.140366097
17000	0.373200536	0.204840397	0.174752075	0.172037377
18000	0.414145577	0.237024949	0.200101711	0.169798917
19000	0.49812719	0.27062116	0.20088957	0.208492865
20000	0.509012086	0.292380956	0.227504972	0.277899579
21000	0.56237337	0.326285005	0.242199434	0.233646737
22000	0.652978332	0.398263293	0.280743994	0.261163495
23000	0.856205122	0.417585087	0.309029766	0.280432574
24000	0.737440007	0.475123454	0.315792964	0.321736078
25000	0.811035712	0.451627495	0.373617974	0.327817653
26000	0.861134306	0.510442275	0.375453591	0.347479001
27000	0.979829071	0.540364791	0.436956797	0.379844646
28000	1.05666307	0.593917446	0.445990492	0.452514403
29000	1.097664882	0.605808865	0.509245932	0.449085009
30000	1.212079031	0.659650483	0.525730741	0.476706358

CAPTURA 28: TIEMPOS PARA PC LOCAL

COMPARATIVA TIEMPOS EN FUNCIÓN DE HEBRAS UTILIZADAS - PMV-OPENMP-A - ATCGRID				
TAMANO	1 HEBRA	4 HEBRAS	8 HEBRAS	12 HEBRAS
5000	0.035933371	0.014897081	0.016885073	0.0083325
6000	0.051689448	0.019546378	0.014948349	0.011355471
7000	0.070360607	0.028228806	0.019821927	0.014458928
8000	0.09179941	0.033868644	0.025775038	0.018387109
9000	0.116038268	0.042876154	0.030052157	0.023373684
10000	0.14354812	0.0537435	0.03961662	0.028802786
11000	0.173468795	0.063551093	0.048048364	0.033804427
12000	0.209099464	0.075982582	0.057027914	0.042609466
13000	0.244093392	0.088515218	0.066769645	0.051695026
14000	0.282136869	0.110060798	0.078339906	0.060879372
15000	0.323157737	0.134788563	0.090669127	0.066795293
16000	0.371014554	0.146557258	0.106287077	0.078661203
17000	0.412782371	0.177461764	0.127967648	0.0896711
18000	0.533218265	0.194823975	0.140554477	0.101703538
19000	0.572847165	0.223556217	0.159651002	0.117022548
20000	0.647072213	0.251242939	0.176759308	0.127638619
21000	0.833492104	0.271203239	0.19051568	0.13957791
22000	0.788146473	0.330907281	0.219224466	0.153345702
23000	0.911311107	0.375514656	0.236635359	0.169314118
24000	1.007863708	0.447093418	0.258219909	0.195513235
25000	1.070221273	0.484777214	0.274283066	0.201613661
26000	1.187537402	0.546590429	0.303335086	0.218471063



**COMENTARIOS SOBRE LOS RESULTADOS:** Al incrementar el número de hebras, en los tres sitios distintos donde hemos ido ejecutando se mejora el tiempo de ejecución. Para calcular los respectivos valores de las ganancias, se usa la fórmula vista en teoría:  $\text{Ganancia con } p \text{ núcleos} = \text{Tiempo secuencial} / \text{Tiempo paralelo usando } p \text{ núcleos}$  (secuencial sería con 1 hebra).