

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): QUINTÍN MESA ROMERO

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:

El error que se produce es el siguiente:

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp shared-clause.c -o shared-clause
shared-clause.c: In function 'main':
shared-clause.c:83:12: error: 'n' not specified in enclosing 'parallel'
   83 |     #pragma omp parallel for shared(a), default(none)
      |            ^~
shared-clause.c:83:12: error: enclosing 'parallel'
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$
```

Se debe a que no se ha incluido la variable `n` a la cláusula `shared`. Para solucionarlo, simplemente se añade a los parámetros de `shared`: `(a,n)`

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

```
75 int main()
76 {
77     int i, n = 7;
78     int a[n];
79
80     for (i=0; i<n; i++)
81         a[i] = i+1;
82
83     #pragma omp parallel for shared(a,n), default(none)
84     for (i=0; i<n; i++)
85     {
86         a[i] += i;
87     }
88
89     printf("Después de parallel for:\n");
90     for (i=0; i<n; i++)
91         printf("a[%d] = %d\n", i, a[i]);
92
93     return(0);
94 }
```

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp shared-clause.c -o shared-clause
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/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
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 generando sin eliminar default(none). Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla que muestren lo que ocurre)
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$
```

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:**

Si inicializamos la suma dentro de la región de la directiva `parallel`, obtenemos una suma que no se corresponde con la correcta; es suma de valores basura.:

```
39 int main()
40 {
41     int i, n = 7;
42     int a[n], suma;
43
44     for (i=0; i<n; i++)
45         a[i] = i;
46
47 #pragma omp parallel private(suma)
48 {
49     suma=0;
50     #pragma omp for
51     for (i=0; i<n; i++)
52     {
53         suma = suma + a[i];
54         printf("Hebra %d suma a[%d] / ",
55             omp_get_thread_num(), i);
56     }
57
58     printf("\n* Hebra %d suma = %d", omp_get_thread_num(), suma);
59
60 }
61
62 printf("\n* Hebra %d suma fuera del parallel= %d",
63     omp_get_thread_num(), suma);
64 printf("\n"); return(0);
65 }
```

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp private-clause_modified.c -o private-clause_modified
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./private-clause_modified
Hebra 0 suma a[0] / Hebra 2 suma a[2] / Hebra 3 suma a[3] / Hebra 1 suma a[1] / Hebra 6 suma a[6] / Hebra 4 suma a[4] / Hebra 5 suma a[5] /
* Hebra 1 suma = 1
* Hebra 3 suma = 3
* Hebra 0 suma = 0
* Hebra 2 suma = 2
* Hebra 4 suma = 4
* Hebra 6 suma = 6
* Hebra 7 suma = 0
* Hebra 5 suma = 5
* Hebra 0 suma fuera del parallel= 32766
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$
```

Esto se debe a que la directiva `private`, se limita a crear la variable y no a inicializarla también, por eso se obtienen valores basura. Para solucionar el problema, se inicializa la suma fuera del `parallel`.

**(b) RESPUESTA:**

**CAPTURA CÓDIGO FUENTE:** `private-clauseModificado_b.c`

```

39 int main()
40 {
41     int i, n = 7;
42     int a[n], suma;
43
44     for (i=0; i<n; i++)
45         a[i] = i;
46
47     suma=7;
48
49 #pragma omp parallel private(suma)
50 {
51     suma;
52 #pragma omp for
53     for (i=0; i<n; i++)
54     {
55         suma = suma + a[i];
56         printf("Hebra %d suma a[%d] / ",
57             omp_get_thread_num(), i);
58     }
59     printf("\n* Hebra %d suma = %d", omp_get_thread_num(), suma);
60 }
61
62
63
64 printf("\n* Hebra %d suma fuera del parallel= %d",
65     omp_get_thread_num(), suma);
66 printf("\n"); return(0);
67 }

```

```

quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp private-clause_modified.c -o private-clause_modified
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./private-clause_modified
Hebra 4 suma a[4] / Hebra 2 suma a[2] / Hebra 0 suma a[0] / Hebra 3 suma a[3] / Hebra 6 suma a[6] / Hebra 5 suma a[5] / Hebra 1 suma a[1] /
* Hebra 3 suma = 3
* Hebra 0 suma = -1583705209
* Hebra 4 suma = 4
* Hebra 7 suma = 0
* Hebra 6 suma = 6
* Hebra 1 suma = 1
* Hebra 5 suma = 5
* Hebra 2 suma = 2
* Hebra 0 suma fuera del parallel= 7
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$

```

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`

```

38 */
39 int main()
40 {
41     int i, n = 7;
42     int a[n], suma;
43
44     for (i=0; i<n; i++)
45         a[i] = i;
46
47 #pragma omp parallel
48 {
49     suma=0;
50     #pragma omp for
51     for (i=0; i<n; i++)
52     {
53         suma = suma + a[i];
54         printf("Hebra %d suma a[%d] / ",
55             omp_get_thread_num(), i);
56     }
57     printf("\n* Hebra %d suma= %d",
58         omp_get_thread_num(), suma);
59 }
60 printf("\n* Hebra %d suma fuera de parallel= %d",
61     omp_get_thread_num(), suma);
62 printf("\n"); return(0);
63 }

```

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:**

Si cambiamos el tamaño del vector a 10, observamos que el valor de suma que se imprime fuera de la región `parallel` es 9 (cuando el tamaño era 7, se imprimía 6, como valor de la suma). Esto se debe a que, por un lado, `firstprivate` combina el `private` de la suma, con la inicialización de las variables del vector y `last` combina el `private` de suma con la copia, al salir de la región paralela, del valor de la última iteración. Por eso, lo que se ve por pantalla es el valor de la última iteración.

**CAPTURAS DE PANTALLA:**



```

quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp firstlastprivate-clause_modified.c -o firstlastprivate-clause_modified
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./firstlastprivate-clause_modified
Hebra 0 suma a[0] suma=0
Hebra 2 suma a[4] suma=4
Hebra 0 suma a[1] suma=1
Hebra 4 suma a[6] suma=6
Hebra 3 suma a[5] suma=5
Hebra 6 suma a[8] suma=8
Hebra 5 suma a[7] suma=7
Hebra 1 suma a[2] suma=2
Hebra 7 suma a[9] suma=9
Hebra 1 suma a[3] suma=5

Fuera de la construcción parallel suma=9
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$

```

b) Si no se cambia el tamaño del vector, no se imprime otro valor de suma ya que, lo que se imprime es el valor de la última iteración. Es independiente este valor del tamaño del vector, porque, de una u otra forma lo que se imprime no es más que la última de las iteraciones, que, dado que por defecto en la directiva for se reparten las iteraciones de forma estática (a la 0 las primeras hebras, según el tamaño que sea, y así sucesivamente).

```

quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp firstlastprivate-clause_modified.c -o firstlastprivate-clause_modified X
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./firstlastprivate-clause_modified X
Hebra 0 suma a[0] suma=0
Hebra 5 suma a[7] suma=7
Hebra 3 suma a[5] suma=5
Hebra 2 suma a[4] suma=4
Hebra 4 suma a[6] suma=6
Hebra 1 suma a[2] suma=2
Hebra 1 suma a[3] suma=5
Hebra 7 suma a[9] suma=9
Hebra 0 suma a[1] suma=1
Hebra 6 suma a[8] suma=8

Fuera de la construcción parallel suma=9
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./firstlastprivate-clause_modified n
Hebra 0 suma a[0] suma=0
Hebra 7 suma a[9] suma=9
Hebra 2 suma a[4] suma=4
Hebra 1 suma a[2] suma=2
Hebra 1 suma a[3] suma=5
Hebra 6 suma a[8] suma=8
Hebra 3 suma a[5] suma=5
Hebra 5 suma a[7] suma=7
Hebra 0 suma a[1] suma=1
Hebra 4 suma a[6] suma=6

Fuera de la construcción parallel suma=9
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$

```

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:**

Lo que se obtiene al quitar la cláusula **`copyprivate(a)`** es lo siguiente:

```

77 int main()
78 {
79     int n = 9;
80     int i, b[n];
81
82     for (i=0; i<n; i++)
83         b[i] = -1;
84 #pragma omp parallel
85 {
86     int a;
87     #pragma omp single // copyprivate(a)
88     {
89         printf("Introduce valor de inicialización a: "); scanf("%d",&a);
90         printf("Single ejecutada por la hebra %d\n",
91             omp_get_thread_num());
92     }
93     #pragma omp for
94     for (i=0; i<n; i++)
95         b[i] = a;
96 }
97
98 printf("Después de la región parallel:\n");
99 for (i=0; i<n; i++)
100     printf(" b[%d] = %d\t", i, b[i]);
101 printf("\n");
102 return(0);
103 }

```

```

quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp copy_private-clause_modified.c -o copy_private-clause_modified
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./copy_private-clause_modified
Introduce valor de inicialización a: 3
Single ejecutada por la hebra 3
Después de la región parallel:
 b[0] = 21920 b[1] = 21920 b[2] = 0 b[3] = 0 b[4] = 3 b[5] = 0
 b[6] = 0 b[7] = 0 b[8] = 0
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$

```

Como podemos observar, al quitar el **`copyprivate(a)`**, estamos evitando que el valor de la variable privada del thread que ejecuta la región del `single`, se copie a las variables `a` del resto de threads, con lo cual, después de la región `parallel`, solo la hebra que ha ejecutado el `single` imprimirá el valor de `b[i] = a`, correctamente, las demás, han cogido valores basura.

5. 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:**

Si observamos, al sustituir `suma=0` por `suma=10`, lo que se observa es que, el valor total de la suma, al final se ha incrementado en 10 unidades. Esto se debe a que la cláusula `reduction` mantiene el contenido de la variable antes de realizar la operación, y por lo tanto, la suma se hace correctamente, sólo que con el incremento de 10 unidades que hemos realizado.

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

```

41 int main(int argc, char **argv)
42 {
43     int i, n=20;
44     int a[n], suma=10;    // modificación
45     if(argc < 2) {
46         fprintf(stderr, "[ERROR]-Falta iteraciones\n");
47         exit(-1);
48     }
49     n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d\n",n);}
50
51     for (i=0; i<n; i++) {
52         a[i] = i;
53     }
54
55     #pragma omp parallel for reduction(+:suma)//default(none) private(i) shared(a,n)
56                                     reduction(+:suma)
57     for (i=0; i<n; i++)
58     { suma += a[i];
59     }
60
61     printf("Tras 'parallel' suma=%d\n", suma);
62     return(0);
63 }

```

```

quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./reduction-clause 4
Tras 'parallel' suma=6
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp reduction-clause_modified.c -o reduction-clause_modified
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./reduction-clause_modified 10
Tras 'parallel' suma=55
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$

```

6. 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`

```

41 int main(int argc, char **argv)
42 {
43     int i, n=20;
44     int a[n], suma=0;
45
46     if(argc < 2) {
47         fprintf(stderr, "[ERROR]-Falta iteraciones\n");
48         exit(-1);
49     }
50     n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d\n",n);}
51
52     for (i=0; i<n; i++) {
53         a[i] = i;
54     }
55
56     #pragma omp parallel
57     {
58         int suma2 = 0;
59
60         #pragma omp for
61         for (i=0; i<n; i++)
62             suma2 += a[i];
63
64         #pragma omp atomic
65         suma += suma2;
66     }
67     printf("Tras 'parallel' suma=%d\n", suma);
68     return(0);
69 }

```



```

quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp reduction-clause_modified2.c -o reduction-clause_modified2
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./reduction-clause_modified2 10
Tras 'parallel' suma=45
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./reduction-clause_modified2 8
Tras 'parallel' suma=28
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ 

```

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

7. 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).

A)

```

82
83 #pragma omp parallel
84 {
85     // Inicializar vector y matriz
86     #pragma omp for private(i,j)
87     for (i = 0; i < N; i++){
88         v1[i] = 0.1*i;
89         v2[i] = 0;
90         for (j = 0; j < N; j++){
91             m[i][j] = i*N+j;
92         }
93     }
94
95     // Calcular v2 = m * v1
96     #pragma omp single
97     time1 = omp_get_wtime();
98
99     #pragma omp for private(i,j)
100    for(i = 0; i < N; i++){
101        suma = 0;
102        for (j = 0; j < N; j++){
103            suma += m[i][j] * v1[j];
104        }
105        v2[i] = suma;
106    }
107 }
108
109 }
110
111 time2 = omp_get_wtime();
112
113 ncgt=time1+time2;
114

```

```

quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp pmv-OpenMP-a.c -o pmv-OpenMP-a
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./pmv-OpenMP-a 4
// Inicializar vector y matriz
Tiempo: 23420.374053589 Tamaño: 4
Matriz:
88      0.000000    v1[i]  1.000000        2.000000        3.000000
89      4.000000    v2[i]  5.000000        6.000000        7.000000
90      8.000000    for (j=0; j<N; j++){
91      12.000000    m[i][j]=i*N+j; 10.000000        11.000000
92                                     13.000000        14.000000        15.000000
93      }
Vector:
94      0.000000 0.100000 0.200000 0.300000
95      // Calcular v2 = m * v1
96      #pragma omp single
Vector resultado:
97      2.300000 4.200000 5.600000 8.600000
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$

```

B)

```

83
84      // Inicializar vector y matriz
85      for (i = 0; i < N; i++){
86          v1[i] = 0.1*i;
87          v2[i] = 0;
88          #pragma omp parallel for
89          for (j = 0; j < N; j++){
90              m[i][j] = i*N+j;
91          }
92      }
93
94      // Calcular v2 = m * v1
95      time1 = omp_get_wtime();
96
97      for(i = 0; i < N; i++){
98          suma = 0;
99          #pragma omp parallel firstprivate(suma)
100          {
101              #pragma omp for
102              for (j = 0; j < N; j++){
103                  suma += m[i][j] * v1[j];
104              }
105              #pragma omp atomic
106              v2[i] += suma;
107          }
108      }
109
110
111      time2 = omp_get_wtime();
112
113      ncgt=time1+time2;
114
115

```

```

quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ gcc -fopenmp pmv-OpenMP-b.c -o pmv-OpenMP-b
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./pmv-OpenMP-b 4
Tiempo: 24338.311297709 Tamaño: 4
Matriz:
94      0.000000      1.000000      2.000000      3.000000
95      4.000000      5.000000      6.000000      7.000000
96      8.000000      9.000000      10.000000     11.000000
97     12.000000     13.000000     14.000000     15.000000
98
99      suma = 0;
100
101      #pragma omp parallel firstprivate(suma)
102      for (j = 0; j < N; j++)
103      {
104          suma += m[i][j] * v1[j];
105      }
106      v2[i] += suma;
107
108      #pragma omp parallel for
109      for (j = 0; j < N; j++)
110      {
111          v2[j] += suma;
112      }
113
114      // Vector resultado:
115      1.400000 3.800000 6.200000 8.600000
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$

```

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).

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

```

// Inicializar vector y matriz
for (i = 0; i < N; i++){
    v1[i] = 0.1*i;
    v2[i] = 0;
    #pragma omp parallel for
    for (j = 0; j < N; j++)
        m[i][j] = i*N+j;
}

// Calcular v2 = m * v1
time1 = omp_get_wtime();

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

        v2[i] += suma;
    }
}

```

**RESPUESTA:** No hay errores.

**CAPTURAS DE PANTALLA:**

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Inf
ormática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica2/bp2$ ./reduction 8
Tiempo: 149057.578223670      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
```

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.



**Tabla 1.** Tiempos de ejecución del código secuencial y de la versión paralela para atcgrid y para el PC personal

	atcgrid1, atcgrid2 o atcgrid3				atcgrid4				PC			
	Tamaño= entre 5000 y 10000		Tamaño= entre 10000 y 100000		Tamaño= entre 5000 y 10000		Tamaño= entre 10000 y 100000		Tamaño= entre 5000 y 10000		Tamaño= entre 10000 y 100000	
Nº de núcleos (p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)
<b>Código Secuencial</b>		----		----		----		----		----		----
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
32												

**COMENTARIOS SOBRE LOS RESULTADOS:**