

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Juan Manuel Salcedo Serrano

Grupo de prácticas: D1

Fecha de entrega: 20/4/2017

Ejercicios basados en los ejemplos del seminario práctico

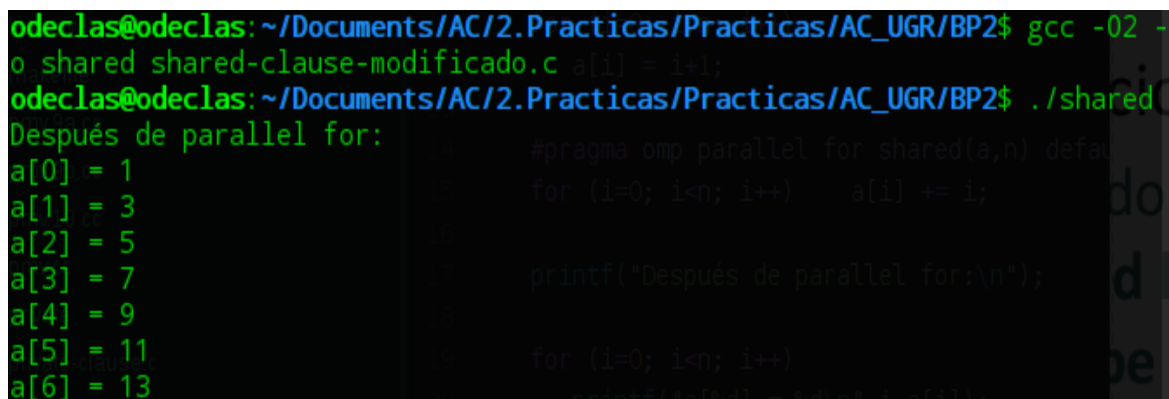
1. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? (añada una captura de pantalla que muestre lo que ocurre) **(b)** Resuelva el problema generado sin eliminar `default(none)`. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA: Al utilizar la directiva `default(none)` se debe especificar el ámbito de todas las variables, si no daría error al compilar. Se soluciona añadiendo la variable `n` a la directiva `shared`.

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]);
    return 0;
}
```

CAPTURAS DE PANTALLA:



```
odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp shared-clause-modificado.c
odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./shared-clause-modificado
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
```

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

RESPUESTA: El valor de `suma`, al estar inicializado fuera de `parallel`, se va

sobreescribiendo.

CÓDIGO FUENTE: private-clauseModificado.c

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

int main(){
    int i, n = 7;
    int a[n], suma;
    for (i=0; i<n; i++)
        a[i] = i;

#pragma omp parallel 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");
    return 0;
}
```

CAPTURAS DE PANTALLA:



```
odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o privatec private-clause.c
odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./privatec
thread 1 suma a[2] / thread 1 suma a[3] / thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma a[4] / thread 2 suma a[5] / thread 3 suma a[6]
) /
* thread 0 suma= 5
* thread 1 suma= -1363175163
* thread 3 suma= -1363175162
* thread 2 suma= -1363175159
```

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

RESPUESTA: Muestra siempre el mismo resultado puesto que cada hebra no usa localmente la variable, sobreescribiéndola, haciendo que se imprima solo el último valor.

CÓDIGO FUENTE: private-clauseModificado3.c

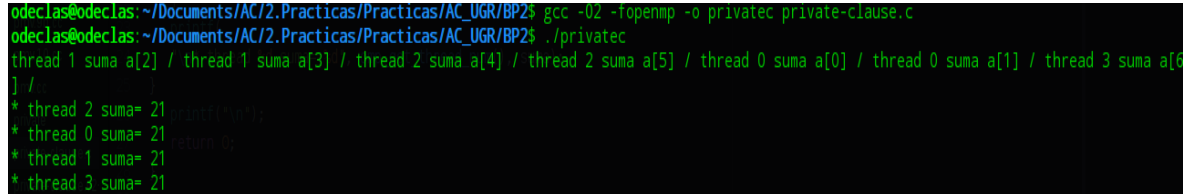
```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

int main(){
    int i, n = 7;
    int a[n], suma;
    for (i=0; i<n; i++)
        a[i] = i;
#pragma omp parallel
{
    #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");
return 0;
}

```

CAPTURAS DE PANTALLA:


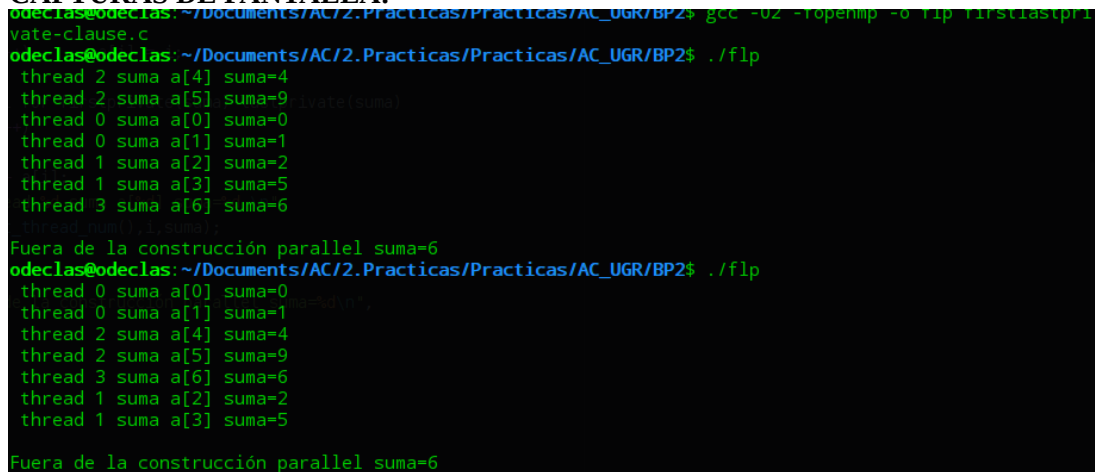
```

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o privatec private-clause.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./privatec
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 2 suma= 21
* thread 0 suma= 21
* thread 1 suma= 21
* thread 3 suma= 21

```

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: Al usar lastprivate en suma solo se guarda el último valor(6).

CAPTURAS DE PANTALLA:


```

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o flp firstlastprivate-clause.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./flp
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
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
Fuera de la construcción parallel suma=6
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./flp
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

```

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: El valor de a solo se guardaría en la hebra que ejecute single, por tanto en pragma omp for solo esa tendrá el valor “correcto”.

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 //copyprivate(a)
    {
        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("Depués 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:

```

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o copyp copyprivat
e-clone.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./copyp
Introduce valor de inicialización a: 3
Single ejecutada por el thread 0
Depués 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

```

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

RESPUESTA: Imprime el mismo resultado añadiéndole los 10 de la inicialización de suma.

CÓDIGO FUENTE: reduction-cloneModificado.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:

```

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o red reduction-clone.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./red
Falta iteraciones
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./red 10
Tras 'parallel' suma=45
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./red 11
Tras 'parallel' suma=55
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./red 13
Tras 'parallel' suma=78
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o red reduction-clone.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./red 10
Tras 'parallel' suma=55
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./red 11
Tras 'parallel' suma=65
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./red 13
Tras 'parallel' suma=88

```

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:

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)
{
...
}
```

CAPTURAS DE PANTALLA:

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):

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

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`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define BILLION 1000000000L
#define MAX 16000
double M[MAX][MAX], v1[MAX], v2[MAX];
int main(int argc, char **argv){
    struct timespec inicio, final;
    double total;
    int N;
    int i, k;
    if(argc < 2) {
        fprintf(stderr, "Parametro: Tama matriz\n");
        exit(-1);
    }
    N = atoi(argv[1]);
    if (N>MAX)
```

```

N=MAX;

clock_gettime(CLOCK_REALTIME, &inicio);
for(i = 0; i < N; i++){
    v1[i] = i;
    v2[i] = 0;
    for(k = 0; k < N; k++)
        M[i][k] = k + N*i;
}
for(i = 0; i < N; i++)
    for(k = 0; k < N; k++)
        v2[i] += M[i][k] * v1[k];

clock_gettime(CLOCK_REALTIME, &final);
total = ( final.tv_sec - inicio.tv_sec ) + (double)
( final.tv_nsec - inicio.tv_nsec ) / (double)BILLION;
printf("Tiempo (seg): %11.9f\t", total);
printf("v2[0] = %8.6f / ", v2[0]);
printf("v2[%d] = %8.6f \n\n", N-1, v2[N-1]);
}

```

CAPTURAS DE PANTALLA:

```

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o pmvsec pmv-secuencial.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmvsec
Parametro: Tama matriz
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmvsec 15
Tiempo (seg): 0.000055868      v2[0] = 1015.000000 / v2[14] = 23065.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmvsec 150
Tiempo (seg): 0.000729930      v2[0] = 1113775.000000 / v2[149] = 250875025.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmvsec 1500
Tiempo (seg): 0.010028904      v2[0] = 1123875250.000000 / v2[1499] = 2529000000250.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmvsec 15000
Tiempo (seg): 0.794467317      v2[0] = 1124887502500.000000 / v2[14999] = 2531024999999476.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ 50000
bash: 50000: no se encontró la orden
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmvsec 50000
Tiempo (seg): 0.959485472      v2[0] = 1365205336000.000000 / v2[15999] = 3276526933332196.000000

```

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):

- 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, 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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define BILLION 1000000000L
#define MAX 16000
double M[MAX][MAX], v1[MAX], v2[MAX];
int main(int argc, char **argv){
    struct timespec inicio, final;
    double total;
    int N;
    int i, k;
    if(argc < 2) {
        fprintf(stderr, "Parametro: Tama matriz\n");
        exit(-1);
    }
    N = atoi(argv[1]);
    if (N>MAX)
        N=MAX;

    clock_gettime(CLOCK_REALTIME, &inicio);
    #pragma omp parallel for
    for(i = 0; i < N; i++){
        v1[i] = i;
        v2[i] = 0;
        #pragma omp parallel for
        for(k = 0; k < N; k++)
            M[i][k] = k + N*i;
    }
    omp_set_num_threads(N);
    #pragma omp parallel private(k)
    for(i = 0; i < N; i++){
        #pragma omp for
        for(k = 0; k < N; k++)
            v2[i] += M[i][k] * v1[k];
    }

    clock_gettime(CLOCK_REALTIME, &final);
    total = ( final.tv_sec - inicio.tv_sec ) + (double)
    ( final.tv_nsec - inicio.tv_nsec ) / (double)BILLION;
    printf("Tiempo (seg): %11.9f\t", total);
    printf("v2[0] = %8.6f / ", v2[0]);
    printf("v2[%d] = %8.6f \n\n", N-1, v2[N-1]);
}
```

CÓDIGO FUENTE: pmv-OpenMP-b.c

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define BILLION 1000000000L
#define MAX 16000
double M[MAX][MAX], v1[MAX], v2[MAX];
int main(int argc, char **argv){
    struct timespec inicio, final;
    double total;
    int N;
    int i, k;
    if(argc < 2) {
        fprintf(stderr, "Parametro: Tama matriz\n");
        exit(-1);
    }
    N = atoi(argv[1]);
    if (N>MAX)
        N=MAX;

    clock_gettime(CLOCK_REALTIME, &inicio);
    #pragma omp parallel for
    for(i = 0; i < N; i++){
        v1[i] = i;
        v2[i] = 0;
        #pragma omp parallel for
        for(k = 0; k < N; k++)
            M[i][k] = k + N*i;
    }
    #pragma omp parallel private(i)
    for(i = 0; i < N; i++)
        #pragma omp for
        for(k = 0; k < N; k++)
            v2[i] += M[i][k] * v1[k];

    clock_gettime(CLOCK_REALTIME, &final);
    total = ( final.tv_sec - inicio.tv_sec ) + (double)
( final.tv_nsec - inicio.tv_nsec ) / (double)BILLION;
    printf("Tiempo (seg): %11.9f\t", total);
    printf("v2[0] = %8.6f / ", v2[0]);
    printf("v2[%d] = %8.6f \n\n", N-1, v2[N-1]);
}

```


CAPTURAS DE PANTALLA:

```

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o pmv9b pmv-OpenMP-b.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 15
Tiempo (seg): 0.006068793      v2[0] = 649.000000 / v2[14] = 17045.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 150
Tiempo (seg): 0.018059005      v2[0] = 1113775.000000 / v2[149] = 48111325.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 300
Tiempo (seg): 0.009381285      v2[0] = 975950.000000 / v2[299] = 249055325.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 600
Tiempo (seg): 0.012078291      v2[0] = 71820100.000000 / v2[599] = 28317341275.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 999
Tiempo (seg): 0.007155284      v2[0] = 5177125.000000 / v2[998] = 217165048876.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 1500
Tiempo (seg): 0.008983479      v2[0] = 122836000.000000 / v2[1499] = 790400320375.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 5000
Tiempo (seg): 0.054004865      v2[0] = 24083073125.000000 / v2[4999] = 97633462760625.000000

```

```

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o pmv9b pmv-OpenMP-b.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 15
Tiempo (seg): 0.006068793      v2[0] = 649.000000 / v2[14] = 17045.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 150
Tiempo (seg): 0.018059005      v2[0] = 1113775.000000 / v2[149] = 48111325.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 300
Tiempo (seg): 0.009381285      v2[0] = 975950.000000 / v2[299] = 249055325.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 600
Tiempo (seg): 0.012078291      v2[0] = 71820100.000000 / v2[599] = 28317341275.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 999
Tiempo (seg): 0.007155284      v2[0] = 5177125.000000 / v2[998] = 217165048876.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 1500
Tiempo (seg): 0.008983479      v2[0] = 122836000.000000 / v2[1499] = 790400320375.000000

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./pmv9b 5000
Tiempo (seg): 0.054004865      v2[0] = 24083073125.000000 / v2[4999] = 97633462760625.000000

```

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

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define BILLION 1000000000L
#define MAX 16000
double M[MAX][MAX], v1[MAX], v2[MAX];
int main(int argc, char **argv){
    struct timespec inicio, final;
    double total, aux = 0;
    int N;
    int i, k;

```

```

        if(argc < 2) {
            fprintf(stderr, "Parametro: Tama matriz\n");
            exit(-1);
        }
        N = atoi(argv[1]);
        if (N>MAX)
            N=MAX;

        clock_gettime(CLOCK_REALTIME, &inicio);
        #pragma omp parallel for
        for(i = 0; i < N; i++){
            v1[i] = i;
            v2[i] = 0;
            #pragma omp parallel for
            for(k = 0; k < N; k++)
                M[i][k] = k + N*i;
        }
        #pragma omp parallel private(i)
        for(i = 0; i < N; i++){
            aux = 0;
            #pragma omp for reduction(+:aux)
            for(k = 0; k < N; k++)
                aux = M[i][k] * v1[k];

            v2[i] = aux;
        }

        clock_gettime(CLOCK_REALTIME, &final);
        total = ( final.tv_sec - inicio.tv_sec ) + (double)
        ( final.tv_nsec - inicio.tv_nsec ) / (double)BILLION;
        printf("Tiempo (seg): %11.9f\t", total);
        printf("v2[0] = %8.6f / ", v2[0]);
        printf("v2[%d] = %8.6f \n\n", N-1, v2[N-1]);
    }

```

CAPTURAS DE PANTALLA:

```

odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ gcc -O2 -fopenmp -o ej10 pmv-OpenMMP-reduction.c
odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./ej10 5
Tiempo (seg): 0.004559842      v2[0] = 36.000000 / v2[4] = 140.000000

odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./ej10 15
Tiempo (seg): 0.005073088      v2[0] = 406.000000 / v2[14] = 3950.000000

odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./ej10 50
Tiempo (seg): 0.004763709      v2[0] = 4851.000000 / v2[49] = 183438.000000

odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./ej10 200
Tiempo (seg): 0.008602148      v2[0] = 161003.000000 / v2[199] = 19814804.000000

odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./ej10 500
Tiempo (seg): 0.002489130      v2[0] = 0.000000 / v2[499] = 311343254.000000

odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./ej10 1000
Tiempo (seg): 0.011535373      v2[0] = 0.000000 / v2[999] = 2495374004.000000

odeclas@odeclas: ~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP2$ ./ej10 1500
Tiempo (seg): 0.010753918      v2[0] = 0.000000 / v2[1499] = 8427092254.000000

```

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: