

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Jose Miguel Hernandez Garcia

Grupo de prácticas: C3

Fecha de entrega: 19/04/17

Fecha evaluación en clase: 20/04/17

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 poner `default(none)` se debe especificar el ámbito de todas las variables que están involucradas en la región `parallel`, por lo que el compilador nos indica que hay que definir el ámbito de la variable `n`

CÓDIGO FUENTE: `shared-clauseModificado.c`

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

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

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

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

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

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

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

CAPTURAS DE PANTALLA:

```
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ gcc -O2 -fopenmp -o shared-clauseModificado shared-clauseModificado.c
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:19:13: error: 'n' not specified in enclosing parallel
    #pragma omp parallel for shared(a) default(none)
    ^
shared-clauseModificado.c:19:13: error: enclosing parallel
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$
```

Añadiendo al `shared` la variable `n`:

```

c
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./shared-clauseModificado
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
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ █

```

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 la variable `suma` se inicializa fuera de `parallel`, contendrá un valor indeterminado, por lo que se debería inicializar dentro de la sección `parallel` o recurrir a `firstprivate`.

CÓDIGO FUENTE: `private-clauseModificado.c`

```

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

#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#define omp_get_num_threads() 1
#endif

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 private(suma)
    {
        suma=1;

        #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:

Funcionamiento incorrecto:

```

ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ gcc -O2 -fopenmp -o private-clause private-clause.c
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./private-clause
thread 3 suma a[3] / thread 4 suma a[4] / thread 5 suma a[5] / thread 2 suma a[2] / thread 6 suma a[6] / thread 1 suma a[1] / thread 0
suma a[0] /
* thread 1 suma = 4196561
* thread 3 suma = 4196563
* thread 0 suma = 8
* thread 6 suma = 4196566
* thread 2 suma = 4196562
* thread 7 suma = 4196560
* thread 5 suma = 4196565
* thread 4 suma = 4196564
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ █

```

Funcionamiento correcto:

```
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./private-clauseModificado
thread 5 suma a[5]thread 6 suma a[6]thread 3 suma a[3]thread 0 suma a[0]thread 1 suma a[1]thread 2 suma a[2]thread 4 suma a[4]
* thread 3 suma= 4
* thread 2 suma= 3
* thread 5 suma= 6
* thread 1 suma= 2
* thread 0 suma= 1
* thread 4 suma= 5
* thread 7 suma= 1
* thread 6 suma= 7
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$
```

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

RESPUESTA: Si eliminamos la cláusula `private(suma)`, la variable `suma` pasará a ser compartida, por lo que los distintos hilos podrán modificarla y sobrescribir el estado previo, lo cual es incorrecto.

CÓDIGO FUENTE: `private-clauseModificado3.c`

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

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:

```
do3.c
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./private-clauseModificado3
thread 1 suma a[1]thread 3 suma a[3]thread 6 suma a[6]thread 2 suma a[2]thread 4 suma a[4]thread 5 suma a[5]thread 0 suma a[0]
* thread 5 suma= 0
* thread 6 suma= 0
* thread 1 suma= 0
* thread 2 suma= 0
* thread 7 suma= 0
* thread 0 suma= 0
* thread 3 suma= 0
* thread 4 suma= 0
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_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: Sí, dado que `lastprivate` asigna a la variable el último valor que se asignaría en una ejecución secuencial del programa, que en este caso será siempre $0 + 6$

CAPTURAS DE PANTALLA:

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: Lo que ocurre es que se produce un incorrecto funcionamiento dado que estamos usando la variable “a” que en algunos casos contiene “basura” porque la hemos leído en un thread pero no la hemos copiado en los demás, es por ello que se debe utilizar el `copyprivate` pues una vez termina el `single`, copia la variable “a” a los demás threads mediante difusión

CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

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());
        }

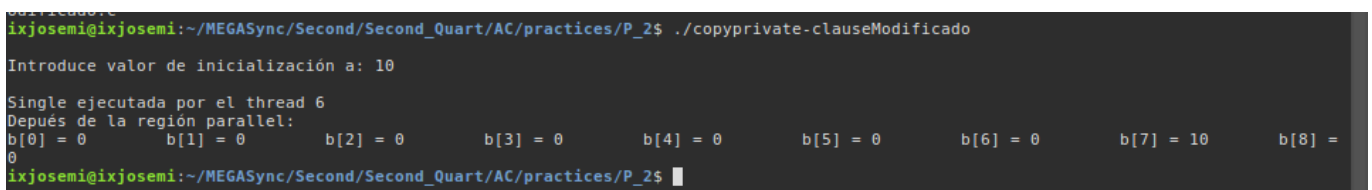
        #pragma omp for
        for (i=0; i<n; i++)
            b[i] = a;
    }

    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:



```
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./copyprivate-clauseModificado
Introduce valor de inicialización a: 10
Single ejecutada por el thread 6
Después de la región parallel:
b[0] = 0      b[1] = 0      b[2] = 0      b[3] = 0      b[4] = 0      b[5] = 0      b[6] = 0      b[7] = 10      b[8] = 0
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$
```

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

RESPUESTA: El resultado final será la suma + 10, dado que el programa suma todos los números desde 0 hasta n y los almacena en “suma”, por lo que si cambiamos el valor inicial de la variable “suma” obtendremos dicho resultado

CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#define omp_get_num_threads() 1
#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:

```
ixjosemi@ixjosemi: ~/MEGASync/Second/Second_Quart/AC/practices/P_2
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./reduction-clause 10
Tras parallel suma = 45
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./reduction-clauseModificado 10
Tras 'parallel' suma=55
```

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`

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

#ifdef _OPENMP
#include <omp.h>
#else
```

```

#define omp_get_thread_num() 0
#define omp_get_num_threads() 1
#endif

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 private(i) reduction(+:suma)
    {
        for (i = omp_get_thread_num(); i < n; i += omp_get_num_threads())
            suma += a[i];
    }

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

```

CAPTURAS DE PANTALLA:

```

ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./reduction-clauseModificado7 10
Tras 'parallel' suma=45
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$

```

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

```

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

```

```

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

    int i, j;
    double t1, t2, total;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc < 2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned
int) = 4 B)

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
devuelve NULL
    M = (double**) malloc(N*sizeof(double *));

    if ((v1 == NULL) || (v2 == NULL) || (M == NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }

    for (i = 0; i < N; i++){
        M[i] = (double*) malloc(N*sizeof(double));

        if ( M[i] == NULL ){
            printf("Error en la reserva de espacio para los vectores\n");
            exit(-2);
        }
    }

    //Inicializar matriz y vectores
    for (i = 0; i < N; i++){
        v1[i] = i;
        v2[i] = 0;

        for(j = 0; j < N; j++)
            M[i][j] = i+j;
    }

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calcular producto de matriz por vector v2 = M · v1
    for (i = 0; i < N; i++)
        for(j = 0; j < N; j++)
            v2[i] += M[i][j] * v1[j];

    //Medida de tiempo
    t2 = omp_get_wtime();
    total = t2 - t1;

    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
total,N,v2[0],N-1,v2[N-1]);

    // Imprimir todos los componentes de v2 (solo si es razonable el tamaño)
    if (N < 20)
        for (i = 0; i < N; i++)
            printf(" V2[%d]=%5.2f\n", i, v2[i]);

    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2

    for (i = 0; i < N; i++)
        free(M[i]);

    free(M);
}

```

```
    return 0;
}
```

CAPTURAS DE PANTALLA:

```
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./pmv-secuencial 100
Tiempo(seg.):0.000007421 / Tamaño:100 / V2[0]=328350.000000 V2[99]=818400.000000
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./pmv-secuencial 1000
Tiempo(seg.):0.001439909 / Tamaño:1000 / V2[0]=332833500.000000 V2[999]=831834000.000000
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./pmv-secuencial 10000
Tiempo(seg.):0.099299827 / Tamaño:10000 / V2[0]=333283335000.000000 V2[9999]=833183340000.000000
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$
```

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

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

    int i, j;
    double t1, t2, total;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc < 2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned
    int) = 4 B)
```



```

double *v1, *v2, **M;
v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
devuelve NULL
M = (double**) malloc(N*sizeof(double *));

if ((v1 == NULL) || (v2 == NULL) || (M == NULL)){
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
}

for (i = 0; i < N; i++){
    M[i] = (double*) malloc(N*sizeof(double));

    if (M[i] == NULL){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
}

//Inicializar matriz y vectores
#pragma omp parallel
{
    #pragma omp for private(j)
    for (i = 0; i < N; i++){
        v1[i] = i;
        v2[i] = 0;

        for(j = 0; j < N; j++)
            M[i][j] = i+j;
    }

    //Medida de tiempo
    #pragma omp single
    t1 = omp_get_wtime();

    //Calcular producto de matriz por vector v2 = M · v1

    #pragma omp for private(j)
    for (i = 0; i < N; i++)
        for(j = 0; j < N; j++)
            v2[i] += M[i][j] * v1[j];

    //Medida de tiempo
    #pragma omp single
    t2 = omp_get_wtime();
}

total = t2 - t1;

//Imprimir el resultado y el tiempo de ejecución
printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
total,N,v2[0],N-1,v2[N-1]);

// Imprimir todos los componentes de v2 (solo si es razonable el tamaño)
if (N < 20)
    for (i = 0; i < N; i++)
        printf(" V2[%d]=%5.2f\n", i, v2[i]);

free(v1); // libera el espacio reservado para v1
free(v2); // libera el espacio reservado para v2

for (i = 0; i < N; i++)
    free(M[i]);

free(M);

return 0;
}

```

```

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

#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#define omp_get_num_threads() 1
#endif

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

    int i, j;
    double t1, t2, total;

    // Leer argumento de entrada (no de componentes del vector)
    if (argc < 2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned
int) = 4 B)

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
devuelve NULL
    M = (double**) malloc(N*sizeof(double *));

    if ((v1 == NULL) || (v2 == NULL) || (M == NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }

    for (i = 0; i < N; i++){
        M[i] = (double*) malloc(N*sizeof(double));

        if (M[i]==NULL){
            printf("Error en la reserva de espacio para los vectores\n");
            exit(-2);
        }
    }

    //Inicializar matriz y vectores
    for (i = 0; i < N; i++){
        v1[i] = i;
        v2[i] = 0;
        #pragma omp parallel for

        for(j = 0; j < N; j++)
            M[i][j] = i + j;
    }

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calcular producto de matriz por vector v2 = M · v1
    for (i = 0; i < N; i++){
        #pragma omp parallel
        {
            double tmp = 0; // aqui nos guardamos el valor temporalmente
            #pragma omp for

            for(j = 0; j < N; j++)
                tmp += M[i][j] * v1[j];

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

    //Medida de tiempo
    t2 = omp_get_wtime();

```

```

    total = t2 - t1;

    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
    total,N,v2[0],N-1,v2[N-1]);

    // Imprimir todos los componentes de v2 (solo si es razonable el tamaño)
    if (N<20)

        for (i = 0; i < N; i++)
            printf(" V2[%d]=%5.2f\n", i, v2[i]);

    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2

    for (i = 0; i < N; i++)
        free(M[i]);

    free(M);

    return 0;
}

```

RESPUESTA:**CAPTURAS DE PANTALLA:**

```

ixjosemi@ixjosemi: ~/MEGASync/Second/Second_Quart/AC/practices/P_2
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./pmv-OpenMP-a 10
Tiempo(seg.):0.003059177      / Tamaño:10      / V2[0]=285.000000 V2[9]=690.000000
V2[0]=285.00
V2[1]=330.00
V2[2]=375.00
V2[3]=420.00
V2[4]=465.00
V2[5]=510.00
V2[6]=555.00
V2[7]=600.00
V2[8]=645.00
V2[9]=690.00
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./pmv-OpenMP-b 10
Tiempo(seg.):0.000106683      / Tamaño:10      / V2[0]=285.000000 V2[9]=690.000000
V2[0]=285.00
V2[1]=330.00
V2[2]=375.00
V2[3]=420.00
V2[4]=465.00
V2[5]=510.00
V2[6]=555.00
V2[7]=600.00
V2[8]=645.00
V2[9]=690.00
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$

```

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

```

```

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

    int i, j;
    double t1, t2, total;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc < 2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned
    int) = 4 B)

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
    devuelve NULL
    M = (double**) malloc(N*sizeof(double *));
    if ((v1==NULL) || (v2==NULL) || (M==NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }

    for (i = 0; i < N; i++){
        M[i] = (double*) malloc(N*sizeof(double));

        if (M[i] == NULL){
            printf("Error en la reserva de espacio para los vectores\n");
            exit(-2);
        }
    }

    //A partir de aqui se pueden acceder las componentes de la matriz como M[i][j]
    //Inicializar matriz y vectores
    for (i = 0; i < N; i++){
        v1[i] = i;
        v2[i] = 0;

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

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calcular producto de matriz por vector v2 = M · v1
    for (i = 0; i < N; i++){
        int tmp = 0;

        #pragma omp parallel for reduction(+:tmp)
        for(j = 0; j < N; j++){
            tmp += M[i][j] * v1[j];
        }
        v2[i] = tmp;
    }

    //Medida de tiempo
    t2 = omp_get_wtime();
    total = t2 - t1;

    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
    total,N,v2[0],N-1,v2[N-1]);

    // Imprimir todos los componentes de v2 (solo si es razonable el tamaño)
    if (N < 20)
        for (i = 0; i < N; i++)
            printf(" V2[%d]=%5.2f\n", i, v2[i]);

    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2

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

```

```

        free(M[i]);

    free(M);

    return 0;
}

```

RESPUESTA: La version con reduction simplifica el código ya que no hay que sumar en una variable, pero hemos tenido que almacenar la suma en una variable dado que no se puede hacer reduccion en un elemento del vector, por lo que primero hacemos reduction sobre la variable y luego la guardamos en el vector

CAPTURAS DE PANTALLA:

```

ixjosemi@ixjosemi: ~/MEGASync/Second/Second_Quart/AC/practices/P_2
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./pmv-OpenMP-reduction
Falta tamaño de matriz y vector
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./pmv-OpenMP-reduction 10
Tiempo(seg.):0.000014725 / Tamaño:10 / V2[0]=285.000000 V2[9]=690.000000
V2[0]=285.00
V2[1]=330.00
V2[2]=375.00
V2[3]=420.00
V2[4]=465.00
V2[5]=510.00
V2[6]=555.00
V2[7]=600.00
V2[8]=645.00
V2[9]=690.00
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$ ./pmv-OpenMP-reduction 100
Tiempo(seg.):0.000198237 / Tamaño:100 / V2[0]=328350.000000 V2[99]=818400.000000
ixjosemi@ixjosemi:~/MEGASync/Second/Second_Quart/AC/practices/P_2$

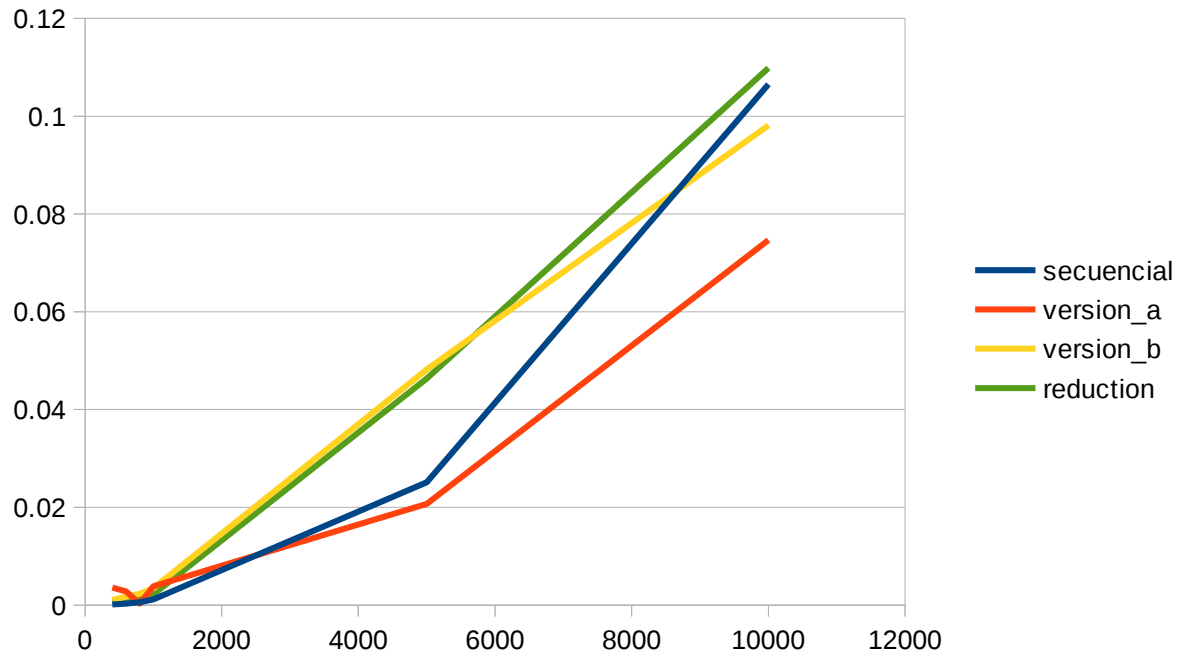
```

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):

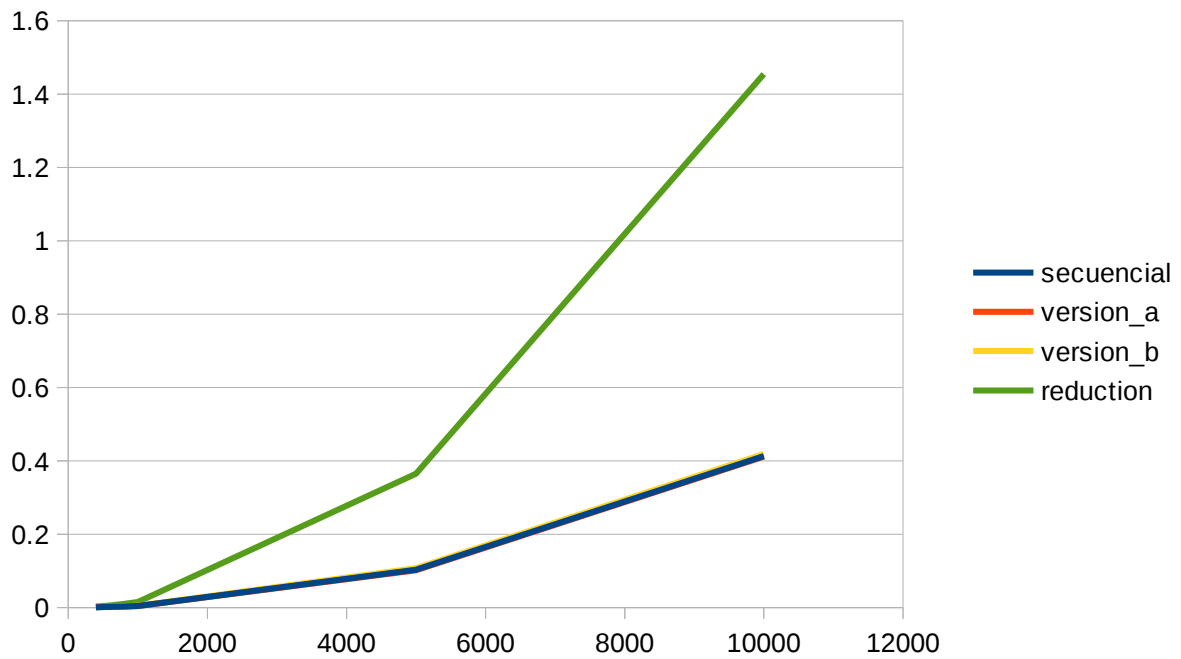
MI PC:

	SECUENCIAL	VERSION A	VERSION B	REDUCTION
400	0.000133416	0.003563911	0.001042303	0.000590671
600	0.000318106	0.002828558	0.001703247	0.000972076
800	0.000602365	0.000284969	0.002326172	0.001346835
1000	0.001183849	0.003877577	0.003454183	0.002197318
5000	0.025092611	0.02072865	0.048203461	0.046352181
10000	0.106509818	0.074678011	0.098107981	0.109853748



ATCGRID:

	SECUENCIAL	VERSION A	VERSION B	REDUCTION
400	0.000804581	0.000873584	0.001082553	0.002467379
600	0.001613351	0.001603015	0.002184937	0.005387336
800	0.002781667	0.002696462	0.003348785	0.009499913
1000	0.004264554	0.004141353	0.005033934	0.014818523
5000	0.102761658	0.101489693	0.106997715	0.364548875
10000	0.412693502	0.411684703	0.418505197	1.454766249



COMENTARIOS SOBRE LOS RESULTADOS:

COMENTARIOS SOBRE LOS RESULTADOS:

Podemos apreciar que la versión por filas es más eficiente en mi pc local frente al resto de versiones, sin embargo en el caso de atcgrid, las versiones por filas, columnas y secuencial están a la par en cuanto a tiempos, quedando bastante por debajo de la versión reduction.