

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Daniel Díaz Pareja

Grupo de prácticas: A2

Fecha de entrega: 21/04/2016

Fecha evaluación en clase: 22/04/2016

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: La compilación da error ya que la cláusula `default(none)` hace que el alcance de las variables usadas en la construcción de la región `parallel` tenga que ser especificado por el programador, a excepción de los índices si están dentro de una cláusula `for`.

Por ello, da un solo error en la variable “n”, ya que no se ha especificado su alcance. Para resolverlo la declaramos como `shared`. Ya que no será necesario realizar ninguna modificación de la misma, las hebras la pueden compartir.

CÓDIGO FUENTE: `shared-clauseModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif
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]);
}
```

CAPTURAS DE PANTALLA:

```
dani@dani-Aspire-5750G:$ gcc -fopenmp -O2 -o shared-clauseModificado shared-clauseModificado.c
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:13:10: error: 'n' not specified in enclosing parallel
    #pragma omp parallel for shared(a) default(none)
    ^
shared-clauseModificado.c:13:10: error: enclosing parallel
```

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: Ocurre que la variable “suma” dentro de la región `parallel` for NO esta inicializada aunque se inicialice fuera, ya que una variable que acompaña a la cláusula `private` tiene su valor indeterminado cuando entra y sale de la región `parallel`.

CÓDIGO FUENTE: `private-clauseModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
main()
{
    int i, n = 7;
    int a[n], suma = 0;
    for (i=0; i<n; i++)
        a[i] = i;
    #pragma omp parallel private(suma)
    {
        //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:

```
dani@dani-Aspire-5750G:~$ gcc -fopenmp -O2 -o private-clauseModificado private-cl
auseModificado.c
dani@dani-Aspire-5750G:~$ ./private-clauseModificado
thread 2 suma a[4] / thread 2 suma a[5] / thread 0 suma a[0] / thread 0 suma a[1]
/ thread 1 suma a[2] / thread 1 suma a[3] / thread 3 suma a[6] /
* thread 1 suma= 4196485
* thread 0 suma= 1
* thread 2 suma= 4196489
* thread 3 suma= 4196486
```

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

RESPUESTA: Ocurre que todas las hebras están acumulando los valores de sus sumas en la variable COMPARTIDA “suma”, es decir, al final del bucle `for` dicha variable tendrá un valor común para todas las hebras compuesto por las sumas parciales que hace cada hebra

CÓDIGO FUENTE: `private-clauseModificado3.c`

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

```

#endif
main()
{
    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:

```

dani@dani-Aspire-5750G:~$ gcc -fopenmp -O2 -o private-clauseModificado3 private-clauseModificado3.c
dani@dani-Aspire-5750G:~$ ./private-clauseModificado3
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 1 suma a[2] / thread 1 suma a[3] /
* thread 3 suma= 11
* thread 0 suma= 11
* thread 2 suma= 11
* thread 1 suma= 11

```

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: después de la región `parallel`, la variable “suma” tiene el valor de la variable privada de la hebra que ejecuta la última iteración. Si la última hebra siempre hace la suma de `a[6]` (como en el seminario), entonces siempre se imprimirá dicho valor fuera de la región `parallel`. Si la última hebra hace otra suma, se imprimirá ese resultado. Por ejemplo, en mi equipo la última hebra ha realizado la suma de `v[5]` y `v[6]`.

CAPTURAS DE PANTALLA:

```

dani@dani-Aspire-5750G:~$ ./firstlastprivate
thread 2 suma a[5] suma=5
thread 2 suma a[6] suma=11
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3

Fuera de la construcción parallel suma=11

```

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: Ocurre que los valores del vector `b` que deberían estar inicializados al valor de la variable “a” toman valores basura, ya que la cláusula `copyprivate(a)` que hace una copia privada de “a” a cada hebra ha desaparecido. Por lo tanto, ahora la variable “a” está sin inicializar.

CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
#include <stdio.h>
#include <omp.h>
main() {
    int n = 9, i, b[n];
    for (i=0; i<n; i++) b[i] = -1;
    #pragma omp parallel
    { int a;

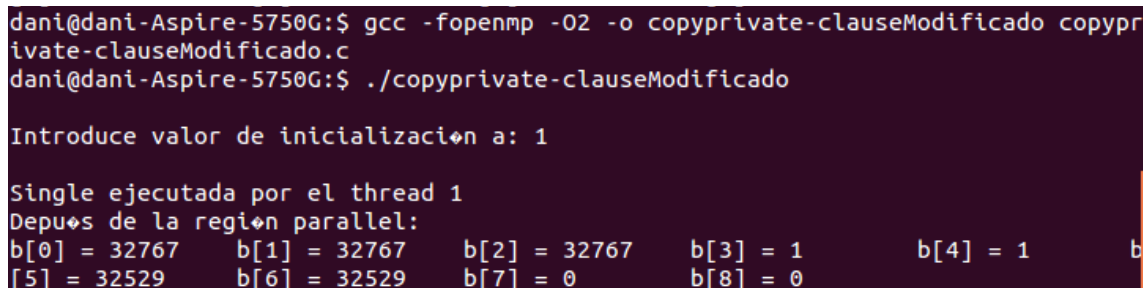
        #pragma omp single
        {
            printf("\nIntroduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("\nSingle ejecutada por el thread %d\n",
                omp_get_thread_num());
        }

        #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:



```
dani@dani-Aspire-5750G:~$ gcc -fopenmp -O2 -o copyprivate-clauseModificado copyprivate-clauseModificado.c
dani@dani-Aspire-5750G:~$ ./copyprivate-clauseModificado

Introduce valor de inicialización a: 1

Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 32767    b[1] = 32767    b[2] = 32767    b[3] = 1        b[4] = 1        b[5] = 32529    b[6] = 32529    b[7] = 0        b[8] = 0
```

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

RESPUESTA: Ahora imprime 55, cuando antes imprimía 45 (con `suma = 0`). Esto es porque la cláusula `reduction` hace la reducción de la variable indicada mediante la operación indicada en dicha cláusula, **incluyendo el valor que tenía la variable inicialmente**. Como ahora el valor inicial es 10, el resultado sale 10 unidades mayor.

CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

```

#endif
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:

```

dani@dani-Aspire-5750G:~$ gcc -fopenmp -O2 -o reduction-clauseModificado reduction-clauseModificado.c
dani@dani-Aspire-5750G:~$ ./reduction-clauseModificado 10
Tras 'parallel' suma=55

```

7. En el ejemplo reduction-clause.c, elimine for 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: No se me ha ocurrido otra manera que crear una hebra por cada componente del vector y que cada una acumule, en exclusión mutua, el valor del vector para su id de hebra en la variable compartida “suma”.

CÓDIGO FUENTE: reduction-clauseModificado7.c

```

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

    omp_set_num_threads(n);
    #pragma omp parallel
    {
        #pragma omp critical
        suma+=a[omp_get_thread_num()];
    }

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

```

}

CAPTURAS DE PANTALLA:

```
dani@dani-Aspire-5750G:~$ gcc -fopenmp -O2 -o reduction-clauseModificado7 reduccion-clauseModificado7.c
dani@dani-Aspire-5750G:~$ ./reduction-clauseModificado7 10
Tras 'parallel' suma=45
```

Resto de ejercicios

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1 (implemente una versión para variables globales y otra para variables dinámicas, use una de estas versiones en los siguientes ejercicios):

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

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, v2, 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

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <omp.h>

int main(int argc, char** argv){
    int i, j, f, c;
    double t1, t2, total;
    srand(time(NULL));

    //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
    //printf("Vector 1: \n\n[");

    for (i=0; i<N; i++)
    {
        v1[i]=i;
        //printf("%.0lf ",v1[i]);
    }

    //printf("]\n\n");

    //printf("Matriz: \n\n");

    for (f=0; f<N; f++)
    {
        //printf("\n");
        for (c=0; c<N; c++)
        {
            M[f][c] = rand()%(1-10 + 1) + 1;
            //printf("%.0lf ", M[f][c]);
        }
    }
    //Medida de tiempo
    t1 = omp_get_wtime();

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

    //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]);

    if (N<15)
    {
        printf("\nv2=[");
        for (i=0; i<N; i++)
            printf("%.0lf ",v2[i]);
        printf("]\n");
    }

    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:

```

dani@dani-Aspire-5750G:$ gcc -fopenmp -O2 -o pmv-secuencial pmv-secuencial.c
dani@dani-Aspire-5750G:$ ./pmv-secuencial 8
Vector 1:

[0 1 2 3 4 5 6 7 ]

Matriz:

2 1 3 3 5 3 4 8
8 6 8 1 2 7 6 6
8 4 1 4 4 5 3 6
4 4 7 1 5 6 8 7
6 3 1 2 5 4 1 4
1 1 4 2 7 1 7 6
4 7 1 7 4 3 5 7
7 3 7 3 8 7 1 6

Tiempo(seg.):0.000000792          / Tamaño:8          / V2[0]=131.000000 V2[7]=141.000000
v2=[131 146 119 168 85 132 140 141 ]

dani@dani-Aspire-5750G:$ gcc -fopenmp -O2 -o pmv-secuencial pmv-secuencial.c
dani@dani-Aspire-5750G:$ ./pmv-secuencial 500
Tiempo(seg.):0.001194932          / Tamaño:500          / V2[0]=575135.000000 V2[499]=542697.000
000

```

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

```

// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

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

```



```

int i, j, f, c;
double t1, t2, total;
srand(time(NULL));

//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
//printf("Vector 1: \n\n");

#pragma omp parallel for
for (i=0; i<N; i++)
    v1[i]=rand()%(1-10 + 1) + 1;

/*for (i=0; i<N; i++)
    printf("%.0lf ",v1[i]);*/

//printf("]\n\n");

//printf("Matriz: \n\n");

#pragma omp parallel for
for (f=0; f<N; f++)
    for (c=0; c<N; c++)
        M[f][c] = rand()%(1-10 + 1) + 1;

/*for (f=0; f<N; f++)
{
    printf("\n");
    for (c=0; c<N; c++)
        printf("%.0lf ", M[f][c]);
}

```

```

    */
    double suma;

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calcular producto de matriz por vector v2 = M · v1
    #pragma omp parallel for private(suma)
    for (f=0; f<N; f++)
    {
        suma=0;
        for (c=0; c<N; c++)
            suma += M[f][c] * v1[c];

        v2[f] = suma;
    }

    //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]);

    if (N<15)
    {
        printf("\nv2=[");
        for (i=0; i<N; i++)
            printf("%.0lf ", v2[i]);
        printf("]\n");
    }

    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;
}

```

CÓDIGO FUENTE: pmv-OpenMP-b.c

```

// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
    int i, j, f, c;
    double t1, t2, total;
    srand(time(NULL));

    //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
        //printf("Vector 1: \n\n");

        #pragma omp parallel for
        for (i=0; i<N; i++)
            v1[i]=rand()%(1-10 + 1) + 1;

        /*for (i=0; i<N; i++)
            printf("%.0lf ",v1[i]);*/

        //printf("]\n\n");

        //printf("Matriz: \n\n");

        #pragma omp parallel for
        for (f=0; f<N; f++)
            for (c=0; c<N; c++)
                M[f][c] = rand()%(1-10 + 1) + 1;

/*
        for (f=0; f<N; f++)
        {
            printf("\n");
            for (c=0; c<N; c++)
                printf("%.0lf ", M[f][c]);

        }*/
        double suma=0;
        //Medida de tiempo
        t1 = omp_get_wtime();

        //Calcular producto de matriz por vector v2 = M · v1
        for (f=0; f<N; f++)
        {
            #pragma omp parallel firstprivate(suma)
            {
                #pragma omp for
                for (c=0; c<N; c++)
                    suma += M[f][c] * v1[c];
            }
        }

```

```

                                #pragma omp critical
                                v2[f] += suma;
                                }
                                }
                                //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]);

                                if (N<15)
                                {
                                        printf("\nv2=[");
                                        for (i=0; i<N; i++)
                                                printf("%.0lf ",v2[i]);
                                        printf("]\n");
                                }

                                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: En el apartado A se paralelizan las filas. Para ello, cada hebra realiza la suma de una fila de manera local y finalmente la guarda en la componente del vector resultado correspondiente.

En el apartado B se paralelizan las columnas. Para ello, cada hebra realiza la suma local de forma privada y luego la acumula en exclusión mutua en la componente del vector resultado correspondiente.

CAPTURAS DE PANTALLA:

```

dani@dani-Aspire-5750G:~$ gcc -fopenmp -O2 -o pmv-OpenMP-a pmv-OpenMP-a.c
dani@dani-Aspire-5750G:~$ ./pmv-OpenMP-a 8
Tiempo(seg.):0.000004931 / Tamaño:8 / V2[0]=64.000000 V2[7]=111.000000
v2=[64 137 196 51 72 96 54 111 ]
dani@dani-Aspire-5750G:~$ ./pmv-OpenMP-a 500
Tiempo(seg.):0.000227998 / Tamaño:500 / V2[0]=5956.000000 V2[499]=9292.000000

```

```

v2=[102 202 85 149 71 120 111 82 ]
dani@dani-Aspire-5750G:~$ ./pmv-OpenMP-b 8
Tiempo(seg.):0.000053015 / Tamaño:8 / V2[0]=127.000000 V2[7]=151.000000
v2=[127 60 42 128 49 72 56 151 ]
dani@dani-Aspire-5750G:~$ ./pmv-OpenMP-b 8
Tiempo(seg.):0.000039770 / Tamaño:8 / V2[0]=117.000000 V2[7]=133.000000

```

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

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
    int i, j, f, c;
    double t1, t2, total;
    srand(time(NULL));

    //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
    //printf("Vector 1: \n\n");

    #pragma omp parallel for
    for (i=0; i<N; i++)
        v1[i]=rand()%(1-10 + 1) + 1;

    /*for (i=0; i<N; i++)
```

```

        printf("%.0lf ", v1[i]); */

//printf("]\n\n");

//printf("Matriz: \n\n");

#pragma omp parallel for
for (f=0; f<N; f++)
    for (c=0; c<N; c++)
        M[f][c] = rand()%(1-10 + 1) + 1;

/*for (f=0; f<N; f++)
{
    printf("\n");
    for (c=0; c<N; c++)
        printf("%.0lf ", M[f][c]);

}*/
double suma;
//Medida de tiempo
t1 = omp_get_wtime();

//Calcular producto de matriz por vector v2 = M · v1
for (f=0; f<N; f++)
{
    suma=0;
    #pragma omp parallel for reduction(+:suma)
    for (c=0; c<N; c++)
        suma += M[f][c] * v1[c];

    v2[f] += suma;
}

//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]);

if (N<15)
{
    printf("\nv2=[");
    for (i=0; i<N; i++)
        printf("%.0lf ", v2[i]);
    printf("]\n");
}

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: Usamos reduction para que al final de la región parallel se acumule la suma de cada variable “suma” de cada hebra en la componente del vector v2 correspondiente.

CAPTURAS DE PANTALLA:

```
dani@dani-Aspire-5750G:$ gcc -fopenmp -O2 -o pmv-OpenMP-reduction pmv-OpenMP-reducti
dani@dani-Aspire-5750G:$ ./pmv-OpenMP-reduction 8
Tiempo(seg.):0.000028931      / Tamaño:8      / V2[0]=21.000000 V2[7]=86.000000

v2=[21 179 142 173 108 197 61 86 ]
dani@dani-Aspire-5750G:$ ./pmv-OpenMP-reduction 500
Tiempo(seg.):0.000859081      / Tamaño:500    / V2[0]=6634.000000 V2[499]=2078.000
```

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: