

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Miguel Angel Robles Urquiza

Grupo de prácticas:A1

Fecha de entrega:20/04/2017

Fecha evaluación en clase:21/04/2017

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: Puesto que usamos `default(none)`, es necesario indicar el ámbito de la variable `n`, puesto que se debe especificar el ámbito de todas las variables del código que se ejecuta en paralelo

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 del parallel for:\n");

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

CAPTURAS DE PANTALLA:

```
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ gcc -O2 -fopenmp shared-clauseModificado.c
-o ../Bin/shared-clauseModificado
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:14:9: error: 'n' not specified in enclosing parallel
#pragma omp parallel for shared(a) default(none)
      ^~~
shared-clauseModificado.c:14:9: 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: Al inicializar la variable fuera del parallel, cuando estemos en el parallel tendrá basura, no el valor establecido

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;

    //suma=10;
#pragma omp parallel private(suma)
    {
        suma=10;
#pragma omp for
        for(i=0; i<n; i++)
        {
            suma = suma + a[i];
            printf("thread %d suma a[%d] /\n", omp_get_thread_num(), i);
        }
        printf("\n* thread %d suma = %d", omp_get_thread_num(), suma);
    }
    printf("\n");
}
```

CAPTURAS DE PANTALLA:



```
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ../Bin/private-clauseModificado
thread 0 suma a[0] /thread 0 suma a[1] /thread 3 suma a[6] /thread 2 suma a[4] /thread 2 s
uma a[5] /thread 1 suma a[2] /thread 1 suma a[3] /
* thread 2 suma = 19
* thread 3 suma = 16
* thread 1 suma = 15
* thread 0 suma = 11
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ gcc -O2 -fopenmp private-clauseModificado.
c -o ../Bin/private-clauseModificado
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ../Bin/private-clauseModificado
thread 0 suma a[0] /thread 0 suma a[1] /thread 1 suma a[2] /thread 1 suma a[3] /thread 3 s
uma a[6] /thread 2 suma a[4] /thread 2 suma a[5] /
* thread 1 suma = 1923766661
* thread 3 suma = 1923766662
* thread 0 suma = 5
* thread 2 suma = 1923766665
```

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

RESPUESTA: Que la variable suma deja de ser privada, por lo que todas las hebras pasan a editar la misma variable y por lo tanto machacan su contenido

CÓDIGO FUENTE: private-clauseModificado2.c

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

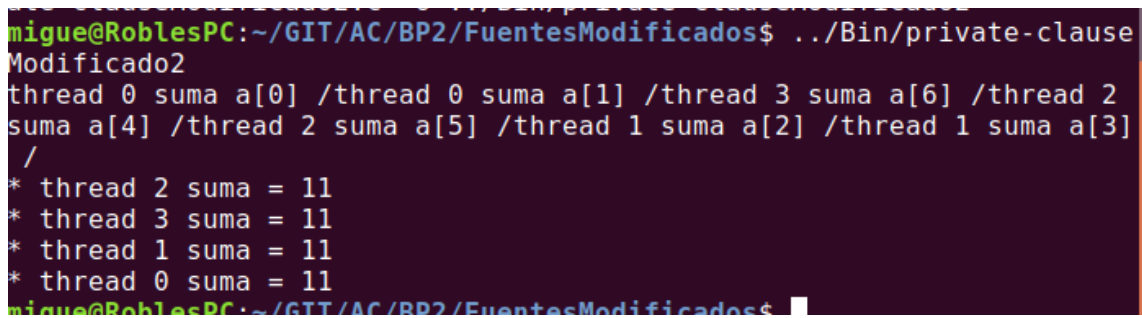
```
#endif

int main()
{
    int i,n = 7;
    int a[n], suma;

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

    #pragma omp parallel
    {
        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:



```
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ./Bin/private-clause
Modificado2
thread 0 suma a[0] /thread 0 suma a[1] /thread 3 suma a[6] /thread 2
suma a[4] /thread 2 suma a[5] /thread 1 suma a[2] /thread 1 suma a[3]
/
* thread 2 suma = 11
* thread 3 suma = 11
* thread 1 suma = 11
* thread 0 suma = 11
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$
```

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:

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: No funciona correctamente ya que estamos usando la variable “a” que la hemos leído en un thread pero no la hemos copiado a los demás thread.

CÓDIGO FUENTE: copyprivate-clauseModificado.c

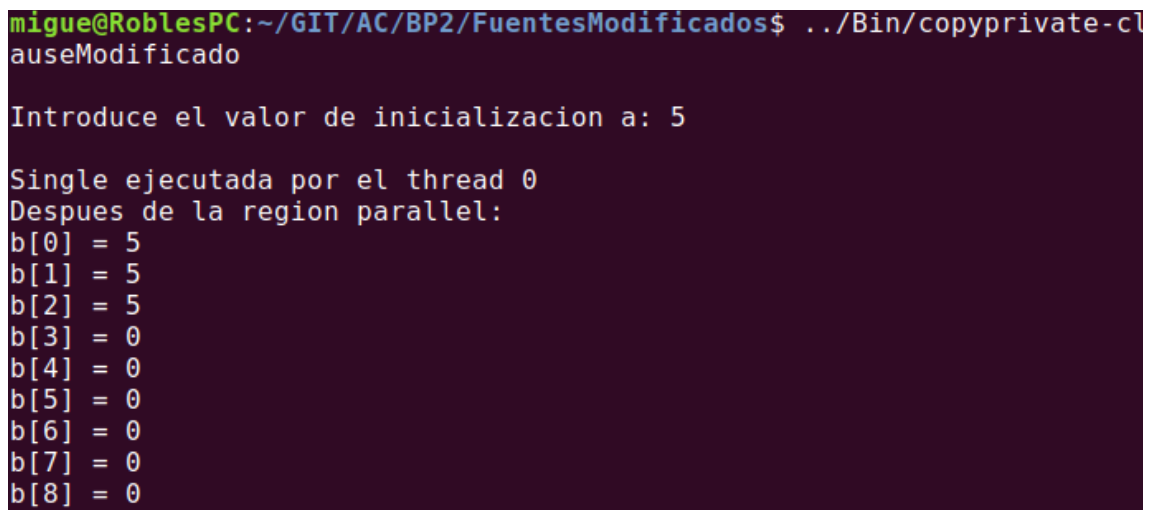
```
#include <stdio.h>
#include <omp.h>

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
  {
      printf("\nIntroduce el valor de inicializacion 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("Despues de la region parallel:\n");
for(i=0; i<n; i++) printf("b[%d] = %d\n",i,b[i]);
printf("\n");
}
```

CAPTURAS DE PANTALLA:


```
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ../Bin/copyprivate-clauseModificado

Introduce el valor de inicializacion a: 5

Single ejecutada por el thread 0
Despues de la region parallel:
b[0] = 5
b[1] = 5
b[2] = 5
b[3] = 0
b[4] = 0
b[5] = 0
b[6] = 0
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: Se imprime el resultado anterior + 10 puesto que suma comienza en 10 en vez de en 0.

CÓDIGO FUENTE: reduction-clauseModificado.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 const **argv) {
    int i, n = 20, a[n], suma = 10;

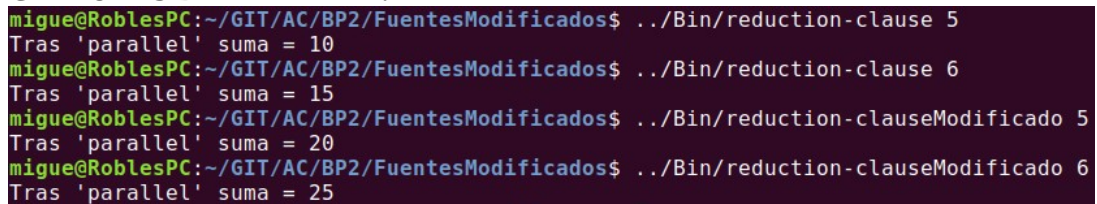
    if(argc < 2){
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }

    n = atoi(argv[1]);
    if(n > 20){
        n = 20;
        printf("n = %d\n",n);
    }

    for (int i = 0; i < n; i++) a[i] = i;
    #pragma omp parallel for reduction(+:suma)
    for (int i = 0; i < n; i++) suma += a[i];

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

```

CAPTURAS DE PANTALLA:


```

migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ../Bin/reduction-clause 5
Tras 'parallel' suma = 10
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ../Bin/reduction-clause 6
Tras 'parallel' suma = 15
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ../Bin/reduction-clauseModificado 5
Tras 'parallel' suma = 20
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ../Bin/reduction-clauseModificado 6
Tras 'parallel' suma = 25

```

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: Usamos la directiva critical para que la suma de los componentes del vector a en paralelo se ejecute correctamente sin usar directivas de trabajo compartido

CÓDIGO FUENTE: reduction-clauseModificado2.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 const **argv) {
    int i, n = 20, a[n], suma = 0, sumalocal;

    if(argc < 2){
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }

    n = atoi(argv[1]);
    if(n > 20){
        n = 20;
    }

```

```

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

#pragma omp parallel private(sumalocal)
{
    sumalocal = 0;
    #pragma omp for schedule(static)
    for (int i = 0; i < n; i++) suma += a[i];
    #pragma omp critical
    suma += sumalocal;
}
printf("Tras 'parallel' suma = %d\n",suma );
}

```

CAPTURAS DE PANTALLA:

```

migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ./reduction-clauseModificado2 12
Tras 'parallel' suma = 66
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ./reduction-clauseModificado2 11
Tras 'parallel' suma = 55
migue@RoblesPC:~/GIT/AC/BP2/FuentesModificados$ ./reduction-clauseModificado2 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):

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

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, t3;

    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);
        }
    }
    for (i=0; i<N;i++){
        v1[i] = i;
        v2[i] = 0;
        for(j=0;j<N;j++)
            M[i][j] = i+j;
    }

    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];

    t2 = omp_get_wtime();
    t3 = t2 - t1;

    printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=
%8.6f\n", t3,N,v2[0],N-1,v2[N-1]);

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

    free(v1);
    free(v2);
    for (i=0; i<N; i++)
        free(M[i]);
    free(M);

    return 0;
}

```

CÓDIGO FUENTE: pmv-secuencial-Globales.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

#define MAX 1000

int main(int argc, char** argv){
    int i, j;
    double t1, t2, t3;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    if (N>MAX) N=MAX;

    double v1[MAX], v2[MAX], M[MAX][MAX];

    for (i=0; i<N;i++){
        v1[i] = i;
        v2[i] = 0;
        for(j=0;j<N;j++)
            M[i][j] = i+j;
    }

    t1 = omp_get_wtime();

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

    t2 = omp_get_wtime();
    t3 = 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", t3, N, v2[0], N-1, v2[N-1]);

    return 0;
}

```


CAPTURAS DE PANTALLA:

```

migue@RoblesPC:~/GIT/AC/BP2/FuentesCreadas$ ./pmv-secuencial 10
Tiempo(seg.):0.000000922 / 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
migue@RoblesPC:~/GIT/AC/BP2/FuentesCreadas$ ./pmv-secuencial-Globales 10
Tiempo(seg.):0.000001413 / Tamaño:10 / V2[0]=285.000000 V2[9]=690.000000

```

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 <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, t3;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

```

```

        unsigned int N = atoi(argv[1]);

        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);
            }
        }

        #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;
            }

            #pragma omp single
            t1 = omp_get_wtime();

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

            #pragma omp single
            t2 = omp_get_wtime();
        }

        t3 = t2 - t1;

        printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=
%8.6f\n", t3,N,v2[0],N-1,v2[N-1]);

        if (N<20)
            for (i=0; i<N;i++)
                printf(" V2[%d]=%5.2f\n", i, v2[i]);

        free(v1);
        free(v2);
        for (i=0; i<N; i++)
            free(M[i]);
        free(M);

```

```

        return 0;
    }

```

CÓDIGO FUENTE: pmv-OpenMP-b.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, t3;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double));
    v2 = (double*) malloc(N*sizeof(double));
    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);
        }
    }

    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;
        }

        t1 = omp_get_wtime();

        for (i=0; i<N;i++){
            #pragma omp parallel
            {
                double tmp = 0;
                #pragma omp for
                for(j=0;j<N;j++){

```

```

                                tmp += M[i][j] * v1[j];
                                }

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

                                }

                                t2 = omp_get_wtime();

                                t3 = t2 - t1;

                                printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n", t3,N,v2[0],N-1,v2[N-1]);

                                if (N<20)
                                    for (i=0; i<N;i++)
                                        printf(" V2[%d]=%5.2f\n", i, v2[i]);

                                free(v1);
                                free(v2);
                                for (i=0; i<N; i++)
                                    free(M[i]);
                                free(M);

                                return 0;
}

```

RESPUESTA: No he tenido errores de compilación ya que he compilado una vez que ya tenía todo el programa comprobado línea por línea. Para la realización de este ejercicio he consultado en Internet para tener una idea clara de la diferencia entre atomic y critical, por lo que al final he usado critical. También he consultado con un compañero de un curso superior que me ha orientado a la hora de realizarlo.

CAPTURAS DE PANTALLA:

```

migue@RoblesPC:~/GIT/AC/BP2/FuentesCreadas$ gcc -O2 -fopenmp pmv-OpenMP-a.c -o
../Bin/pmv-OpenMP-a
migue@RoblesPC:~/GIT/AC/BP2/FuentesCreadas$ gcc -O2 -fopenmp pmv-OpenMP-b.c -o
../Bin/pmv-OpenMP-b

```

```

migue@RoblesPC:~/GIT/AC/BP2/FuentesCreadas$ ../Bin/pmv-OpenMP-a 15
Tiempo(seg.):0.000003704          / Tamaño:15          / V2[0]=1015.000000 V2[14]=2485
.000000
V2[0]=1015.00
V2[1]=1120.00
V2[2]=1225.00
V2[3]=1330.00
V2[4]=1435.00
V2[5]=1540.00
V2[6]=1645.00
V2[7]=1750.00
V2[8]=1855.00
V2[9]=1960.00
V2[10]=2065.00
V2[11]=2170.00
V2[12]=2275.00
V2[13]=2380.00
V2[14]=2485.00

```

```

migue@RoblesPC:~/GIT/AC/BP2/FuentesCreadas$ ../Bin/pmv-OpenMP-b 15
Tiempo(seg.):0.000072462      / Tamaño:15      / V2[0]=1015.000000 V2[14]=2485
.000000
V2[0]=1015.00
V2[1]=1120.00
V2[2]=1225.00
V2[3]=1330.00
V2[4]=1435.00
V2[5]=1540.00
V2[6]=1645.00
V2[7]=1750.00
V2[8]=1855.00
V2[9]=1960.00
V2[10]=2065.00
V2[11]=2170.00
V2[12]=2275.00
V2[13]=2380.00
V2[14]=2485.00

```

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-OpenMP-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, t3;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double));
    v2 = (double*) malloc(N*sizeof(double));
    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);
        }
    }

    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;
    }

    t1 = omp_get_wtime();

    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;
    }

    t2 = omp_get_wtime();
    t3 = t2 - t1;

    printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n", t3,N,v2[0],N-1,v2[N-1]);

    if (N<20)
        for (i=0; i<N;i++)
            printf(" V2[%d]=%5.2f\n", i, v2[i]);

    free(v1);
    free(v2);
    for (i=0; i<N; i++)
        free(M[i]);
    free(M);

    return 0;
}

```

RESPUESTA: No he tenido errores de compilación ya que he compilado una vez que ya tenía todo el programa comprobado línea por línea. He consultado con un compañero de un curso superior que me ha orientado a la hora de realizarlo.

CAPTURAS DE PANTALLA:

```
migue@RoblesPC:~/GIT/AC/BP2/FuentesCreadas$ ../Bin/pmv-OpenMP-reduction 15
Tiempo(seg.):0.000030963 / Tamaño:15 / V2[0]=1015.000000 V2[14]=2485
.000000
V2[0]=1015.00
V2[1]=1120.00
V2[2]=1225.00
V2[3]=1330.00
V2[4]=1435.00
V2[5]=1540.00
V2[6]=1645.00
V2[7]=1750.00
V2[8]=1855.00
V2[9]=1960.00
V2[10]=2065.00
V2[11]=2170.00
V2[12]=2275.00
V2[13]=2380.00
V2[14]=2485.00
```

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: