

Grai2º curso / 2º
cuatr.
Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos):

Grupo de prácticas y profesor de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

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)`? **(b)** Resuelva el problema generado sin eliminar `default(none)`. Añada el código con la modificación al cuaderno de prácticas. (Añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA:

CAPTURA 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:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer1] 2019-04-24 miércoles
$gcc -O2 -fopenmp shared-clauseModificado.c -o shared-clauseModificado
shared-clauseModificado.c:5:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^~~~~~
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:13:13: error: 'n' not specified in enclosing 'parallel'
    #pragma omp parallel for shared(a) default(none)
                    ^~~~~~
shared-clauseModificado.c:13:13: error: enclosing 'parallel'
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer1] 2019-04-24 miércoles
$
```

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer1] 2019-04-24 miércoles
$gcc -O2 -fopenmp shared-clauseModificado.c -o shared-clauseModificado
shared-clauseModificado.c:5:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^~~~~~
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer1] 2019-04-24 miércoles
$export OMP_DYNAMIC=FALSE
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer1] 2019-04-24 miércoles
$export OPM_NUM_THREADS=2
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer1] 2019-04-24 miércoles
$./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
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer1] 2019-04-24 miércoles
$
```

2. Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel` e inicializar suma a un valor distinto de 0. Ejecute varias veces el código ¿Qué imprime el código fuera del `parallel`? (muéstrelo con una captura de pantalla) ¿Qué ocurre si en esta versión de `private-clause.c` se inicia la variable suma fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta (añada capturas de pantalla que muestren lo que ocurre). Añadir el código con las modificaciones al cuaderno de prácticas.

RESPUESTA:**CAPTURA CÓDIGO FUENTE:** `private-clauseModificado.c`

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

int main(int argc, char **argv){
    int i, n=7;
    int a[n], suma;
```

```

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

suma=0;

#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 Fuera de la region Parallel thread %d suma= %d",
omp_get_thread_num(), suma);
}

```

```

$gcc -O2 -fopenmp -o private-clauseModificado private-clauseModificado.c
private-clauseModificado.c:8:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer2] 2019-04-25 jueves
$export OMP_DYNAMIC=FALSE
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer2] 2019-04-25 jueves
$export OMP_NUM_THREADS=4
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer2] 2019-04-25 jueves
$./private-clauseModificado
thread0 suma a[0] suma=0
thread0 suma a[1] suma=1
thread1 suma a[2] suma=1290652002
thread1 suma a[3] suma=1290652005
thread2 suma a[4] suma=1290652212
thread2 suma a[5] suma=1290652217
thread3 suma a[6] suma=1290652422

Thread suma=1

Fuera de la construccion parallel --> Thread suma=1
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer2] 2019-04-25 jueves
$

```

```

[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer2] 2019-04-25 jueves
$gcc -O2 -fopenmp private-clauseModificado.c -o private-clauseModificado
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer2] 2019-04-25 jueves
$./private-clauseModificado
thread 0 suma a[0] thread 0 suma a[1] thread 2 suma a[4] thread 2 suma a[5] thread 1
suma a[2] thread 1 suma a[3] thread 3 suma a[6]
Thread 0 suma= 1280736849
Thread 1 suma= 702010197
Thread 2 suma= 702010201
Thread 3 suma= 702010198
Fuera de la region Parallel thread 0 suma= 0[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer2] 2019-04-25 jueves
$

```

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

RESPUESTA:

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

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

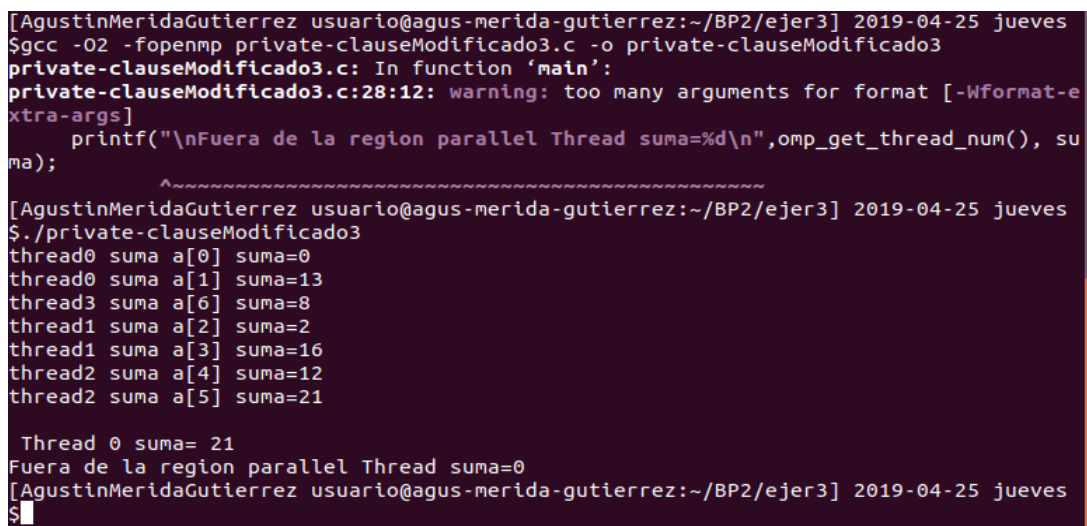
int main(int argc, char **argv)
{
    int i, n=7;
    int a[n], suma;

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

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

    printf("\nFuera de la region parallel Thread suma=%d\n",omp_get_thread_num(),
        suma);
}
```

CAPTURAS DE PANTALLA:



```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer3] 2019-04-25 jueves
$gcc -O2 -fopenmp private-clauseModificado3.c -o private-clauseModificado3
private-clauseModificado3.c: In function 'main':
private-clauseModificado3.c:28:12: warning: too many arguments for format [-Wformat-extra-args]
    printf("\nFuera de la region parallel Thread suma=%d\n",omp_get_thread_num(), su
ma);

[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer3] 2019-04-25 jueves
$./private-clauseModificado3
thread0 suma a[0] suma=0
thread0 suma a[1] suma=13
thread3 suma a[6] suma=8
thread1 suma a[2] suma=2
thread1 suma a[3] suma=16
thread2 suma a[4] suma=12
thread2 suma a[5] suma=21

Thread 0 suma= 21
Fuera de la region parallel Thread suma=0
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer3] 2019-04-25 jueves
$
```

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 (añada capturas de pantalla que muestren lo que ocurre).

RESPUESTA: El código ha de imprimir siempre 6, ya que este es el resultado de la última iteración del bucle y `lastprivate(suma)` lo que hace la última hebra copie su valor de la variable privada “suma” a la variable (dándose la casualidad) “suma”.

CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer4] 2019-04-25 jueves
$gcc -O2 -fopenmp firstlastprivate_modificado.c -o firstlastprivate_modificado
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer4] 2019-04-25 jueves
$./firstlastprivate_modificado
thread 0 suma a[0] suma=0
thread 6 suma a[6] suma=6
thread 2 suma a[2] suma=2
thread 1 suma a[1] suma=1
thread 4 suma a[4] suma=4
thread 3 suma a[3] suma=3
thread 5 suma a[5] suma=5

Fuera de la construcción parallel suma=6
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer4] 2019-04-25 jueves
$./firstlastprivate_modificado
thread 4 suma a[4] suma=4
thread 0 suma a[0] suma=0
thread 6 suma a[6] suma=6
thread 3 suma a[3] suma=3
thread 5 suma a[5] suma=5
thread 1 suma a[1] suma=1
thread 2 suma a[2] suma=2

Fuera de la construcción parallel suma=6
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer4] 2019-04-25 jueves
$
```

5. ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido? (añada una captura de pantalla que muestre lo que ocurre)

RESPUESTA: Como `copyprivate(a)` en la directiva `single` lo que hace es darle el valor de “a” a las otras hebras, cuando lo eliminamos no se produce esta difusión, y solo se inicializan las zonas que haya iniciado la hebra que ejecute el `single`.

CAPTURA 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 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");
}

```

CAPTURAS DE PANTALLA:

```

$gcc -O2 -fopenmp copyprivate-clauseModificado.c -o copyprivate-clauseModificado
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer5] 2019-04-25 ju
eves
$./copyprivate-clauseModificado

Introduce valor de inicialización a: 4

Single ejecutada por el thread 2
Depués de la región parallel:
b[0] = 0      b[1] = 0      b[2] = 0      b[3] = 4      b[4] = 0      b
[5] = 0 b[6] = 0      b[7] = 0      b[8] = 0
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer5] 2019-04-25 ju
eves
$

```

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

Justifique el resultado (añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA: El resultado que imprime es el mismo que con suma=0, solo que en este caso es por que se le ha sumado 10. Esto es porque reduction lo que hace es mantener el valor inicial de la variable que se va a ir acumulando (en este caso, la variable “suma”).

CAPTURA 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 **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);}
}

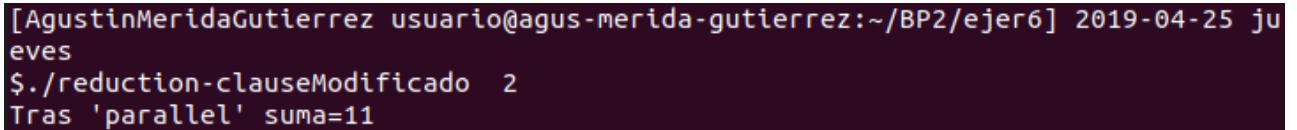
```

```

    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:


```

[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer6] 2019-04-25 juves
$./reduction-clauseModificado 2
Tras 'parallel' suma=11

```

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 añadir más directivas de trabajo compartido (añada capturas de pantalla que muestren lo que ocurre).

RESPUESTA:**CAPTURA CÓDIGO FUENTE: reduction-clauseModificado7.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=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
    {
        int suma_parallel=0;

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

        #pragma omp atomic
            suma += suma_parallel;
    }
    printf("Tras 'parallel' suma=%d\n", suma);
}

```


CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer7] 2019-04-25 ju
eves
$gcc -O2 -fopenmp reduction-clauseModificado7.c -o reduction-clauseModificado7
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer7] 2019-04-25 ju
eves
$./reduction-clauseModificado7 2
Tras 'parallel' suma=1
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer7] 2019-04-25 ju
eves
$
```

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).

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

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

#define VECTOR_DYNAMIC //Como se nos ha indicado en clase, hacemos uso de este tipo de
vectores

int main(int argc, char** argv)
{
    int i, j;
    double t_ini, t_fin, t_total, res;

    //Leemos si el numero de datos introducidos es correcto
    if (argc<2){
        printf("Falta el tamaño de la matriz y e el tamaño del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); //MAXIMO N = 2^32

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

    //Comprobamos que la reserva de memoria se ha realizado correctamente
    if ((v1==NULL) || (v2==NULL) || (M==NULL)){
        printf("Error al reservar el espacio de la matriz\n");
        exit(-2);
    }
}
```



```

}

for(i=0; i<N; i++){
    M[i] = (double*) malloc(N*sizeof(double));
    if(M[i]==NULL){
        printf("Error al reservar el espacio de la matriz\n");
        exit(-2);
    }
}

//Inicializamos la matriz
for(i=0; i<N; i++){
    v1[i]=1;
}

for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        M[i][j]=1;
    }
}

//Calculamos v2 = M * v1 y lo que tarda en realizar esta operación
t_ini = omp_get_wtime();

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

t_fin = omp_get_wtime();
t_total = t_fin-t_ini;

//Mostramos el resultado y su tiempo de ejecución;
printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f v2[%d]=%8.6f\n", t_total, N,
v2[0], N-1, v2[N-1]);
if(N<30){
    printf("%f ", v2[i]);
    printf("\n");
}

//Liberamos el espacio reservado
free(v1);
free(v2);
for(i=0; i<N; i++)
    free(M[i]);

free(M);

return 0;
}

```

CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer8] 2019-04-25 ju
eves
$gcc -O2 -fopenmp pmv-secuencial.c -o pmv-secuencial
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer8] 2019-04-25 ju
eves
$./pmv-secuencial 8
Tiempo(seg.): 0.000000502          / Tamaño:8          / V2[0]=8.000000 v2[7]=8.000000
22157981528010711248435635008630357880874654428328956066413604136069017372146026
14637632036471178139518119242741162114263813868206923479346786706659412764817954
1722916816438470087611212763409735427716764779369463808.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer8] 2019-04-25 ju
eves
$./pmv-secuencial 11
Tiempo(seg.): 0.000000453          / Tamaño:11         / V2[0]=11.000000 v2[10]=11.0000
00
0.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer8] 2019-04-25 ju
eves
$
```

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).

CAPTURA CÓDIGO FUENTE : pmv-OpenMP-a.c

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

#define VECTOR_DYNAMIC //Como se nos ha indicado en clase, hacemos uso de este
tipo de vectores

int main(int argc, char** argv)
{
    int i, j;
```

```

double t_ini, t_fin, t_total, res;

//Leemos si el numero de datos introducidos es correcto
if (argc<2){
    printf("Falta el tamaño de la matriz y e el tamaño del vector\n");
    exit(-1);
}

unsigned int N = atoi(argv[1]); //MAXIMO N = 2^32

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

//Comprobamos que la reserva de memoria se ha realizado correctamente
if ((v1==NULL) || (v2==NULL) || (M==NULL)){
    printf("Error al reservar el espacio de la matriz\n");
    exit(-2);
}

for(i=0; i<N; i++){
    M[i] = (double*) malloc(N*sizeof(double));
    if(M[i]==NULL){
        printf("Error al reservar el espacio de la matriz\n");
        exit(-2);
    }
}

#pragma omp parallel
{
    //Inicializamos la matriz
    #pragma omp for
        for(i=0; i<N; i++){
            v1[i]=1;
        }
    #pragma omp for private(j)
        for(i=0; i<N; i++){
            for(j=0; j<N; j++){
                M[i][j]=1;
            }
        }
}

//Calculamos v2 = M * v1 y lo que tarda en realizar esta operación
#pragma omp single
{
    t_ini = omp_get_wtime();
}

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

```

```

    }

#pragma omp single
{
    t_fin = omp_get_wtime();
}

}
t_total = t_fin-t_ini;

//Mostramos el resultado y su tiempo de ejecución;
printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f v2[%d]=%8.6f\n",
t_total, N, v2[0], N-1, v2[N-1]);
if(N<30){
    printf("%f ", v2[i]);
    printf("\n");
}

//Liberamos el espacio reservado
free(v1);
free(v2);
for(i=0; i<N; i++)
    free(M[i]);

free(M);

return 0;
}

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

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

#define VECTOR_DYNAMIC //Como se nos ha indicado en clase, hacemos uso de este
tipo de vectores

int main(int argc, char** argv)
{
    int i, j;
    double t_ini, t_fin, t_total, res;

    //Leemos si el numero de datos introducidos es correcto
    if (argc<2){
        printf("Falta el tamaño de la matriz y e el tamaño del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); //MAXIMO N = 2^32

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

```

```

//Comprobamos que la reserva de memoria se ha realizado correctamente
if ((v1==NULL) || (v2==NULL) || (M==NULL)){
    printf("Error al reservar el espacio de la matriz\n");
    exit(-2);
}

for(i=0; i<N; i++){
    M[i] = (double*) malloc(N*sizeof(double));
    if(M[i]==NULL){
        printf("Error al reservar el espacio de la matriz\n");
        exit(-2);
    }
}

#pragma omp parallel private(i)
{
    #pragma omp for
    //Inicializamos la matriz
    for(i=0; i<N; i++){
        v1[i]=1;
    }

    #pragma omp for
    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            M[i][j]=1;
        }
    }

    //Calculamos v2 = M * v1 y lo que tarda en realizar esta operación
    #pragma omp single
    {
        t_ini = omp_get_wtime();
    }

    for(i=0; i<N; i++){
        res=0;
        #pragma omp for
        for(j=0; j<N; j++){
            res=res+(M[i][j]*v1[j]);
        }

        #pragma omp atomic
        v2[i] += res;
    }

    #pragma omp single
    {
        t_fin = omp_get_wtime();
    }
}
t_total = t_fin-t_ini;

//Mostramos el resultado y su tiempo de ejecución;

```

```

    printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f v2[%d]=%8.6f\n",
t_total, N, v2[0], N-1, v2[N-1]);
    if(N<30){
        printf("%f ", v2[i]);
        printf("\n");
    }

    //Liberamos el espacio reservado
    free(v1);
    free(v2);
    for(i=0; i<N; i++)
        free(M[i]);

    free(M);

    return 0;
}

```

RESPUESTA:

La página que se ha utilizado para tener en cuenta la definición de las cláusulas de compartición de datos es la siguiente: https://lsi.ugr.es/jmantas/ppr/ayuda/omp_ayuda.php?ayuda=omp_clausulas

En **compilación**, la mayoría de los errores se deben a que no se ha realizado de manera correcta la paralelización o a que no se sabía que es lo que se tenía que poner como compartido o privado. Se han solucionado gracias al repaso de las transparencias vistas en clase. La página

En **ejecución** en principio no ha habido ningún fallo que haya detectado.

CAPTURAS DE PANTALLA:

```

[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer9] 2019-04-25 ju
eves
$gcc -O2 -fopenmp pmv-OpenMP-a.c -o pmv-OpenMP-a
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer9] 2019-04-25 ju
eves
$./pmv-OpenMP-a 8
Tiempo(seg.): 0.000005799          / Tamaño:8          / V2[0]=8.000000 v2[7]=8.000000
22157981528010711248435635008630357880874654428328956066413604136069017372146026
14637632036471178139518119242741162114263813868206923479346786706659412764817954
1722916816438470087611212763409735427716764779369463808.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer9] 2019-04-25 ju
eves
$./pmv-OpenMP-a 11
Tiempo(seg.): 0.000005445          / Tamaño:11          / V2[0]=11.000000 v2[10]=11.0000
00
0.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer9] 2019-04-25 ju
eves
$

```

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer9] 2019-04-25 ju
eves
$gcc -O2 -fopenmp pmv-OpenMP-b.c -o pmv-OpenMP-b
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer9] 2019-04-25 ju
eves
$./pmv-OpenMP-b 8
Tiempo(seg.): 0.000009635 / Tamaño:8 / V2[0]=503737752228788171933890
03874839910566648924119518163990997184785902086485071825604408804870705289098862
738898067877294201473623064331178738612293549598767787916357212194209792.000000
v2[7]=8.000000
22157981528010711248435635008630357880874654428328956066413604136069017372146026
14637632036471178139518119242741162114263813868206923479346786706659412764817954
1722916816438470087611212763409735427716764779369463808.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer9] 2019-04-25 ju
eves
$./pmv-OpenMP-b 11
Tiempo(seg.): 0.000013894 / Tamaño:11 / V2[0]=6.000000 v2[10]=12.00000
0
0.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer9] 2019-04-25 ju
eves
$
```

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).

CAPTURA CÓDIGO FUENTE: `pmv-OpenmMP-reduction.c`

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

#define VECTOR_DYNAMIC //Como se nos ha indicado en clase, hacemos uso de este
tipo de vectores

int main(int argc, char** argv)
{
    int i, j;
    double t_ini, t_fin, t_total, res;

    //Leemos si el numero de datos introducidos es correcto
    if (argc<2){
        printf("Falta el tamaño de la matriz y e el tamaño del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); //MAXIMO N = 2^32

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



```

M = (double**) malloc(N*sizeof(double *));

//Comprobamos que la reserva de memoria se ha realizado correctamente
if ((v1==NULL) || (v2==NULL) || (M==NULL)){
    printf("Error al reservar el espacio de la matriz\n");
    exit(-2);
}

for(i=0; i<N; i++){
    M[i] = (double*) malloc(N*sizeof(double));
    if(M[i]==NULL){
        printf("Error al reservar el espacio de la matriz\n");
        exit(-2);
    }
}

#pragma omp parallel private(i)
{
    #pragma omp for
    //Inicializamos la matriz
    for(i=0; i<N; i++){
        v1[i]=1;
    }

    #pragma omp for private(j)
    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            M[i][j]=1;
        }
    }

    //Calculamos v2 = M * v1 y lo que tarda en realizar esta operación
    #pragma omp single
    {
        t_ini = omp_get_wtime();
    }

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

        #pragma omp single
        {
            v2[i] = res;
            res = 0;
        }
    }

    #pragma omp single
    {
        t_fin = omp_get_wtime();
    }
}
t_total = t_fin-t_ini;

```

```

//Mostramos el resultado y su tiempo de ejecución;
printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f v2[%d]=%8.6f\n",
t_total, N, v2[0], N-1, v2[N-1]);
if(N<30){
    printf("%f ", v2[i]);
    printf("\n");
}

//Liberamos el espacio reservado
free(v1);
free(v2);
for(i=0; i<N; i++)
    free(M[i]);

free(M);

return 0;
}

```

RESPUESTA:

CAPTURAS DE PANTALLA:

```

[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer10] 2019-04-25 j
ueves
$gcc -O2 -fopenmp pmv-OpenMP-reduction.c -o pmv-OpenMP-reduction
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer10] 2019-04-25 j
ueves
$./pmv-OpenMP-reduction 8
Tiempo(seg.): 0.000015629          / Tamaño:8          / V2[0]=8.000000 v2[7]=8.000000
22157981528010711248435635008630357880874654428328956066413604136069017372146026
14637632036471178139518119242741162114263813868206923479346786706659412764817954
1722916816438470087611212763409735427716764779369463808.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer10] 2019-04-25 j
ueves
$./pmv-OpenMP-reduction 9
Tiempo(seg.): 0.000017425          / Tamaño:9          / V2[0]=9.000000 v2[8]=9.000000
0.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer10] 2019-04-25 j
ueves
$

```

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 su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar -O2 al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

CAPTURAS DE PANTALLA (que justifique el código elegido):

TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia) (para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 20000 y 100000, y otro entre 5000 y 20000):

COMENTARIOS SOBRE LOS RESULTADOS:

Tamaño	Ejer9a	Ejer9b	Ejer10
1000	0.002612887	0.001031582	0.001652606
2000	0.005072630	0.003507379	0.003819713
3000	0.011367861	0.005780311	0.006500639
4000	0.019872193	0.010263212	0.011030453
5000	0.031216862	0.015492008	0.015683038
6000	0.044249670	0.022078229	0.022459575
7000	0.060707000	0.026834650	0.028496640
8000	0.078878717	0.036847217	0.036589983
9000	0.099350881	0.043591856	0.045735279
10000	0.122733355	0.052531158	0.055268174
11000	0.148572886	0.063036836	0.066731086
12000	0.176764203	0.074366751	0.078049461
13000	0.207939308	0.088253390	0.090558367
14000	0.240233137	0.100333585	0.104050137
15000	0.276180092	0.114442992	0.118184196
16000	0.325817641	0.131020783	0.134348945
17000	0.393967110	0.147171566	0.151953427
18000	0.399714158	0.164728786	0.169955536
19000	0.444990104	0.184383204	0.187805387
20000	0.57016095	0.206028228	0.219853678

Como podemos ver, el resultado más óptimo es el del programa 9b, por lo que utilizaremos ese para este ejercicio. Para la ejecución de los datos, haremos uso de dos scripts, uno para el PC-LOCAL y otro para el ATCGRID

PC-LOCAL

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer11] 2019-04-25 jueves
$ ./pc_local.sh
Tiempo(seg.): 0.001474990 / Tamaño:1000 / V2[0]=1000.000000 v2[999]=1000.000000
Tiempo(seg.): 0.001187362 / Tamaño:1000 / V2[0]=500.000000 v2[999]=1000.000000
Tiempo(seg.): 0.000816631 / Tamaño:1000 / V2[0]=333.000000 v2[999]=999.000000
Tiempo(seg.): 0.000768100 / Tamaño:1000 / V2[0]=250.000000 v2[999]=1000.000000
./pc_local.sh: línea 13: 3151 Terminado (killed) ./pmv-OpenMP-b 100000
./pc_local.sh: línea 13: 3152 Terminado (killed) ./pmv-OpenMP-b 100000
./pc_local.sh: línea 13: 3154 Terminado (killed) ./pmv-OpenMP-b 100000
./pc_local.sh: línea 13: 3158 Terminado (killed) ./pmv-OpenMP-b 100000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP2/ejer11] 2019-04-25 jueves
$
```

ATCGRID

```
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP2/ejer11] 2019-04-25 jueves
$ls
atcgrid.sh  pmv-OpenMP-b  pmv-OpenMP-b.e19859  pmv-OpenMP-b.o19859
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP2/ejer11] 2019-04-25 jueves
$cat pmv-OpenMP-b.o19859
Tiempo(seg.): 0.002770246      / Tamaño:1000 / V2[0]=1000.000000 v2[999]=1000.000000
Tiempo(seg.): 0.002329862      / Tamaño:1000 / V2[0]=500.000000 v2[999]=1000.000000
Tiempo(seg.): 0.002052431      / Tamaño:1000 / V2[0]=334.000000 v2[999]=1002.000000
Tiempo(seg.): 0.001869535      / Tamaño:1000 / V2[0]=250.000000 v2[999]=1000.000000
Tiempo(seg.): 0.002214666      / Tamaño:1000 / V2[0]=400.000000 v2[999]=1000.000000
Tiempo(seg.): 0.002501816      / Tamaño:1000 / V2[0]=166.000000 v2[999]=996.000000
Tiempo(seg.): 0.002608907      / Tamaño:1000 / V2[0]=286.000000 v2[999]=1001.000000
Tiempo(seg.): 0.003535306      / Tamaño:1000 / V2[0]=375.000000 v2[999]=1000.000000
Tiempo(seg.): 0.003701170      / Tamaño:1000 / V2[0]=222.000000 v2[999]=999.000000
Tiempo(seg.): 0.003584847      / Tamaño:1000 / V2[0]=300.000000 v2[999]=1000.000000
Tiempo(seg.): 0.004276373      / Tamaño:1000 / V2[0]=273.000000 v2[999]=1001.000000
Tiempo(seg.): 0.003112925      / Tamaño:1000 / V2[0]=333.000000 v2[999]=996.000000
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP2/ejer11] 2019-04-25 jueves
$
```