

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Antonio Miguel Morillo Chica

Grupo de prácticas: D1

Fecha de entrega: 20/04/2016

Fecha evaluación en clase:

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: Todas las variables se tomarán como privadas de cada hebra y habrá que indicar manualmente con `shared` que variables queremos que no lo sean.

CÓDIGO FUENTE: `shared-clauseModificado.c`

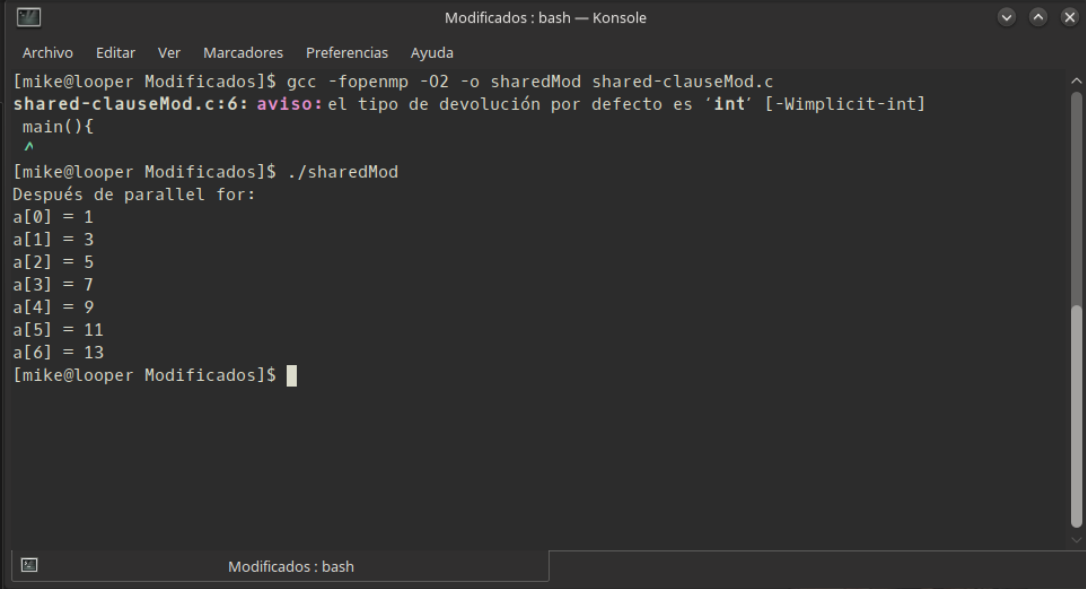
```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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;
        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:



```
Modificados: bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Modificados]$ gcc -fopenmp -O2 -o sharedMod shared-clauseMod.c
shared-clauseMod.c:6: aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(){
^
[mike@looper Modificados]$ ./sharedMod
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
[mike@looper Modificados]$
```

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: Que a partir de la segunda hebra el valor de la variable `suma` comienza con un valor indeterminado.

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;    // Inicio suma fuera de parallel
    for (i=0; i<n; i++)
        a[i] = i;

    #pragma omp parallel private(suma)
    {
        printf("\nAntes de omp for\n");

        #pragma omp for
        for (i=0; i<n; i++){
            /*
             * Solo funcionará correctamente en la primera hebra el resto funciona con
             * basura ya que está inicializada fuera del parallel.
             */
            suma = suma + a[i];
            printf("thread %d suma a[%d] \n", omp_get_thread_num(), i);
        }

        printf("\n* Fuera del bucle\n* thread %d suma = %d\n", omp_get_thread_num(), suma);
    }

    printf("\n* Fuera de parallel\n* thread %d suma = %d\n", omp_get_thread_num(), suma);
}
```

CAPTURAS DE PANTALLA:

```
Modificados: bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Modificados]$ gcc -fopenmp -O2 -o privateClauseMod private-clauseMod.c
private-clauseMod.c:8:1: aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(){
^
[mike@looper Modificados]$ ./privateClauseMod

Antes de omp for

Antes de omp for
thread 2 suma a[4]
thread 2 suma a[5]

Antes de omp for
thread 3 suma a[6]
thread 0 suma a[0]
thread 0 suma a[1]

Antes de omp for
thread 1 suma a[2]
thread 1 suma a[3]

* Fuera del bucle
* thread 3 suma= 4196342

* Fuera del bucle
* thread 0 suma= 5

* Fuera del bucle
* thread 2 suma= 4196345

* Fuera del bucle
* thread 1 suma= 4196341

* Fuera de parallel
* thread 0 suma= 0
[mike@looper Modificados]$
```

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

RESPUESTA: Si no indicamos que la variable `suma` es privada, cada hebra no podrá disponer de una copia de dicha variable, sino que todas trabajarán con la misma, `suma` no se inicializará y no se borrará pasada la frontera del código `parallel`.

CÓDIGO FUENTE: `private-clauseModificado3.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;    // Inicizo suma fuera de parallel
    for (i=0; i<n; i++)
        a[i] = i;

    #pragma omp parallel
    {
        suma = 0;
        printf("\nAntes de omp for\n");

        #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* Fuera del bucle\n* thread %d suma = %d\n", omp_get_thread_num(), suma);
    }

    printf("\n* Fuera de parallel\n* thread %d suma = %d\n", omp_get_thread_num(), suma);
}
```

CAPTURAS DE PANTALLA:

```
Modificados: bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Modificados]$ gcc -fopenmp -O2 -o privateClauseMod2 private-clauseMod2.c
private-clauseMod2.c:8:1: aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(){
^
[mike@looper Modificados]$ ./privateClauseMod2

Antes de omp for

Antes de omp for
thread 1 suma a[2]
thread 1 suma a[3]

Antes de omp for
thread 2 suma a[4]
thread 2 suma a[5]
thread 0 suma a[0]
thread 0 suma a[1]

Antes de omp for
thread 3 suma a[6]

* Fuera del bucle
* thread 2 suma = 21

* Fuera del bucle
* thread 0 suma = 21

* Fuera del bucle
* thread 1 suma = 21

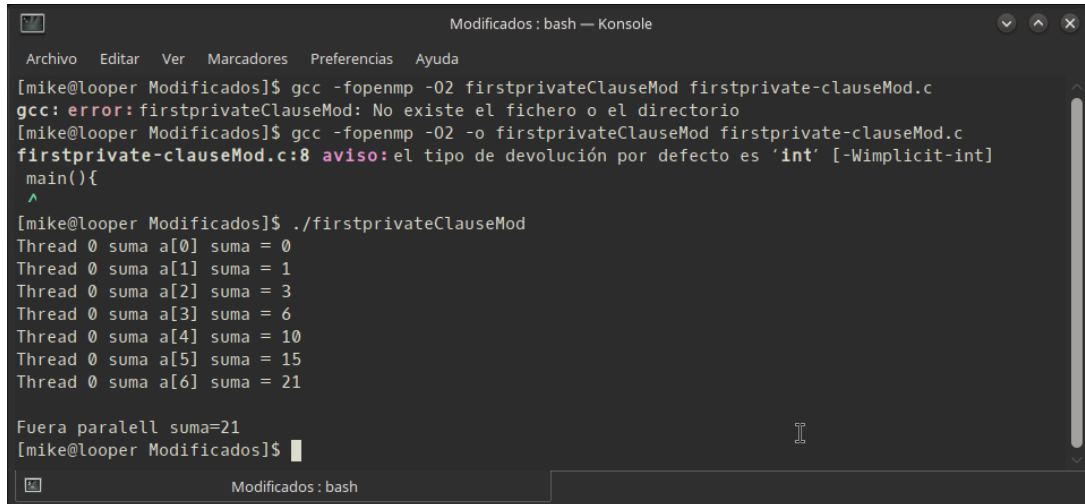
* Fuera del bucle
* thread 3 suma = 21

* Fuera de parallel
* thread 0 suma = 21
[mike@looper Modificados]$
```

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, siempre y cuando el valor en la hebra que realiza la última iteración sea ése, en caso contrario, no. En mi pc, por ejemplo da un valor de 21.

CAPTURAS DE PANTALLA:



```

Modificados : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Modificados]$ gcc -fopenmp -O2 firstprivateClauseMod firstprivate-clauseMod.c
gcc: error: firstprivateClauseMod: No existe el fichero o el directorio
[mike@looper Modificados]$ gcc -fopenmp -O2 -o firstprivateClauseMod firstprivate-clauseMod.c
firstprivate-clauseMod.c:8 aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(){
^
[mike@looper Modificados]$ ./firstprivateClauseMod
Thread 0 suma a[0] suma = 0
Thread 0 suma a[1] suma = 1
Thread 0 suma a[2] suma = 3
Thread 0 suma a[3] suma = 6
Thread 0 suma a[4] suma = 10
Thread 0 suma a[5] suma = 15
Thread 0 suma a[6] suma = 21

Fuera paralell suma=21
[mike@looper Modificados]$

```

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: La función de `copyprivate` copia variable seleccionada a todas las demás hebras y, puesto que eliminamos esa opción, excepto la primera hebra, todas trabajarán con una variable con contenido indeterminado, basura.

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("\nValor de la inicialización a: ");
            scanf("%d", &a);
            printf("\nSingle ejecutado por el thread %d\n", omp_get_thread_num());
        }

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

    printf("Después de parallel:\n");
    for (i=0; i<n;i++){
        printf("b[%d]=%d\t", i, b[i]);
        printf("\n");
    }
}

```

CAPTURAS DE PANTALLA:

```

Modificados: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

[mike@looper Modificados]$ ./copyprivateClauseMod

Valor de la inicialización a: 5

Single ejecutado por el thread 0
Después de 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
[mike@looper Modificados]$

```

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

RESPUESTA: Que el resultado de la suma será el mismo que el anterior, solo que aumentando dicho resultado el valor 10, siempre y cuando introduzcamos un valor mayor o igual que 2, en caso contrario siempre devolverá el valor 10.

CÓDIGO FUENTE: `reduction-clauseModificado.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=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("Después de 'parallel' suma =%d\n", suma);
}

```

CAPTURAS DE PANTALLA:

```

Modificados: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[mike@looper Modificados]$ gcc -fopenmp -O2 -o reductionClauseMod reduction-clauseModificado.c
reduction-clauseModificado.c: $ aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(int argc, char **argv){
^
[mike@looper Modificados]$ ./reductionClauseMod 8
Después de 'parallel' suma =38
[mike@looper Modificados]$ ./reductionClauseMod 9
Después de 'parallel' suma =46
[mike@looper Modificados]$

```

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: Únicamente introduzco justo después de la nueva línea de código #pragma omp parallel reduction(+:suma) una nueva línea justo antes del bucle for que contenga #pragma omp for y el código volverá a funcionar correctamente.

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;

    #pragma omp parallel reduction(+:suma)
    #pragma omp for
        for(i=0; i<n; i++)
            suma+=a[i];
    printf("Tras 'parallel' suma =%d\n", suma);
}

```

CAPTURAS DE PANTALLA:

```

Modificados : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Modificados]$ gcc -fopenmp -O2 -o reductionClauseMod2 reduction-clauseMod2.c
reduction-clauseMod2.c:9 aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(int argc, char **argv){
^
[mike@looper Modificados]$ ./reductionClauseMod2
Falta iteraciones
[mike@looper Modificados]$ ./reductionClauseMod2 8
Tras 'parallel' suma =28
[mike@looper Modificados]$ ./reductionClauseMod2 9
Tras 'parallel' suma =36
[mike@looper Modificados]$ ./reductionClauseMod2 7
Tras 'parallel' suma =21
[mike@looper Modificados]$

```

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> // biblioteca con atoi()
#include <stdio.h>
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()

#define VECTOR_DYNAMIC // descomentar para que los vectores sean variables ...
#define PRINTF_ALL // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 33554432 // 2^25
double matriz[MAX*MAX], vector[MAX], resultado[MAX];
#endif

int main(int argc, char **argv){
    int i,j;
    double suma;
    struct timespec cgt1, cgt2;
    double ncgt;

    if(argc<2){
        printf("No has indicado la capacidad de la matriz cuadrada\n");
        exit(-1);
    }

```

```

unsigned int N = atoi(argv[1]);
#ifdef VECTOR_GLOBAL
double matriz[N*N], vector[N], resultado[N];
#endif

#ifdef VECTOR_GLOBAL
if(N>MAX)
    N=MAX;
#endif

#ifdef VECTOR_DYNAMIC
double *matriz, *vector, *resultado;
matriz = (double*) malloc(N*N*sizeof(double));
vector = (double*) malloc(N*sizeof(double));
resultado = (double*) malloc(N*sizeof(double));

if((resultado==NULL) || (matriz==NULL) || (vector==NULL)){
    printf("\nError a la hora de reservar memoria para la matriz y/o los vectores\n");
    exit(-2);
}

#endif

//inicialicemos
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        matriz[(i*N)+j]=N*0.1 +j*0.1;
        vector[i]=N*0.1-i*0.1;
        resultado[i]=0.0;
    }
}

clock_gettime(CLOCK_REALTIME, &cgt1);

for(i=0; i<N; i++){
    suma=0.0;
    for(j=0; j<N; j++){
        suma+=matriz[(i*N)+j]*vector[j]; //Aquí hacemos la multiplicación "matriz" * vector

        resultado[i]=suma;
    }
    clock_gettime(CLOCK_REALTIME, &cgt2);
    ncgt=((double)(cgt2.tv_sec-cgt1.tv_sec)+(double)((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9)))*10.0;

#ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f / Tamaño del vector: %u / resultado[0]=%f / resultado[%d]=%f \n",ncgt, N, resultado[0], N-1, resultado[N-1]);
#endif

    if (N<18){
        for(i=0; i<N; i++)
            printf("\nComponente %d de resultado[%d] = %f", i,i,resultado[i]);
    }

    printf("\n");
#endif

#ifdef VECTOR_DYNAMIC
    free(matriz);
    free(vector);
    free(resultado);
#endif

return 0;
}

```


CAPTURAS DE PANTALLA:

```

Modificados : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[mike@looper Modificados]$ ./productomatricesSecu 10
Tiempo(seg.):0.000006240 / Tamaño del vector: 10 / resultado[0]=7.150000 / resultado[9]=7.150000

Componente 0 de resultado[0] = 7.150000
Componente 1 de resultado[1] = 7.150000
Componente 2 de resultado[2] = 7.150000
Componente 3 de resultado[3] = 7.150000
Componente 4 de resultado[4] = 7.150000
Componente 5 de resultado[5] = 7.150000
Componente 6 de resultado[6] = 7.150000
Componente 7 de resultado[7] = 7.150000
Componente 8 de resultado[8] = 7.150000
Componente 9 de resultado[9] = 7.150000
[mike@looper Modificados]$ ./productomatricesSecu 20
Tiempo(seg.):0.000009550 / Tamaño del vector: 20 / resultado[0]=55.300000 / resultado[19]=55.300000

[mike@looper Modificados]$

```

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

```

/*
Suma de dos vectores: v3 =v1 +v2
*/
#include <stdlib.h> // biblioteca con atoi()
#include <stdio.h>
#include <omp.h>

#define VECTOR_DYNAMIC /
#define PRINTF_ALL

```

```

#ifdef VECTOR_GLOBAL
#define MAX 33554432 //2^25
double matriz[MAX*MAX], vector[MAX], resultado[MAX];
#endif

int main(int argc, char **argv){
    int i,j;
    double suma;
    //struct timespec cgt1, cgt2;
    double ncgt, tInicio, tFin;
    if(argc<2){
        printf("No has indicado la capacidad de la matriz cuadrada\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]);

    #ifdef VECTOR_GLOBAL
        double matriz[N*N], vector[N], resultado[N];
    #endif

    #ifdef VECTOR_GLOBAL
        if(N>MAX)
            N=MAX;
    #endif

    #ifdef VECTOR_DYNAMIC
        double *matriz, *vector, *resultado;
        matriz = (double*) malloc(N*N*sizeof(double));
        vector = (double*) malloc(N*sizeof(double));
        resultado = (double*) malloc(N*sizeof(double));

        if((resultado==NULL) || (matriz==NULL) || (vector==NULL)){
            printf("\nError a la hora de reservar memoria para la matriz y/o los vectores\n");
            exit(-2);
        }
    #endif
    //inicialicemos
    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            matriz[(i*N)+j]=N*0.1 +j*0.1;
            vector[i]=N*0.1-i*0.1;
            resultado[i]=0.0;
        }
    }

    //clock_gettime(CLOCK_REALTIME, &cgt1);

    tInicio=omp_get_wtime();

    #pragma omp parallel for default(none)
    for(i=0; i<N; i++){ // así conseguimos que cada hebra ejecute el programa con las
        // variables privadas, para evitar solapar resultados
        suma=0.0;
        for(j=0; j<N; j++){
            suma+=matriz[(i*N)+j]*vector[j]; //Aquí hacemos la multiplicación "matriz" * vector
        }
        resultado[i]=suma;
    }

    tFin=omp_get_wtime();

    //clock_gettime(CLOCK_REALTIME, &cgt2);
    ncgt=(tFin-tInicio)*10.0;

    #ifdef PRINTF_ALL
        printf("Tiempo(seg.):%11.9f / Tamaño del vector: %u / resultado[0]=%f / resultado[%d]=%f \n",ncgt, N, resultado[0], N-1, resultado[N-1]);
        if (N<18){
            for(i=0; i<N; i++){
                printf("\nComponente %d de resultado[%d] = %f", i,i,resultado[i]);
            }
        }
        printf("\n");
    #endif
}

```

```

#endif

#ifdef VECTOR_DYNAMIC
    free(matriz);
    free(vector);
    free(resultado);
#endif

    return 0;
}

```

CÓDIGO FUENTE: pmv-OpenMP-b.c

```

/*
Suma de dos vectores: v3 =v1 +v2
*/
#include <stdlib.h> // biblioteca con atoi()
#include <stdio.h>
#include <omp.h>
#define VECTOR_DYNAMIC // descomentar para que los vectores sean variables ...
#define PRINTF_ALL // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 33554432 //2^25
double matriz[MAX*MAX], vector[MAX], resultado[MAX];
#endif

int main(int argc, char **argv){
    int i,j;
    double suma, aux;
    //struct timespec cgt1, cgt2;
    double ncgt, tInicio, tFin;
    if(argc<2){
        printf("No has indicado la capacidad de la matriz
cuadrada\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); //como vamos a comparar en varias
ocasiones N con el tamaño MAX, la ponemos en mayúscula para
#ifdef VECTOR_GLOBAL //indicar que se trata de
un entero que puede ser increíblemente grande, pero arbitrario.
double matriz[N*N], vector[N], resultado[N];
#endif

#ifdef VECTOR_GLOBAL
if(N>MAX)
    N=MAX;
#endif

#ifdef VECTOR_DYNAMIC
double *matriz, *vector, *resultado;
matriz = (double*) malloc(N*N*sizeof(double));
vector = (double*) malloc(N*sizeof(double));
resultado = (double*) malloc(N*sizeof(double));

if((resultado==NULL) || (matriz==NULL) || (vector==NULL)){ //si N es
demasiado grande, igual nuestro pc no puede reservar tanta mem.
printf("\nError a la hora de reservar memoria para la
matriz y/o los vectores\n");
exit(-2);
}
#endif

//inicialicemos
for(i=0; i<N; i++){
    for(j=0; j<N; j++)
        matriz[(i*N)+j]=N*0.1 +j*0.1; //así
tratamos a la matriz en cuestión como un vector(dimension 1), rellenando todos
vector[i]=N*0.1-i*0.1; //sus
campos
        resultado[i]=0.0;

```

```

    }
    //clock_gettime(CLOCK_REALTIME, &cgt1);

    tInicio=omp_get_wtime();

    //Ahora indicamos que queremos realizar la multiplicación de forma
    paralela, paralelizando el recorrido por columnas (for interno)
    for(i=0; i<N; i++){ //así conseguimos que cada hebra ejecute el
    programa con las variables privadas, para evitar solapar resultados
        suma=0.0;
        #pragma omp parallel private(aux)
        {
            aux=0.0;
            #pragma omp for schedule(static) //cada
            hebra realizará un tramo del bucle decididos mediante método Round-Robin
            for(j=0; j<N; j++) //dicho
            de otra forma, tramos equitativos y en un orden racional para las hebras
                aux+=matriz[(i*N)
+ j]*vector[j]; //Aquí hacemos la multiplicación "matriz" * vector
            #pragma omp critical //Sección crítica,
            sólo puede entrar una hebra al mismo tiempo (no hay paralelización)
            {
                suma+=aux; //así evitamos
            }
            machacar resultados unos encima de otros
        }
        resultado[i]=suma;
    }

    tFin=omp_get_wtime();

    //clock_gettime(CLOCK_REALTIME, &cgt2);
    ncgt=(tFin-tInicio)*10.0;

    #ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f / Tamaño del vector: %u / resultado[0]=%f /
    resultado[%d]=%f \n",ncgt, N, resultado[0], N-1, resultado[N-1]);
    if (N<18){
        for(i=0; i<N; i++)
            printf("\nComponente %d de resultado[%d] =
    %f", i,i,resultado[i]);
    }
    printf("\n");
    #endif

    #ifdef VECTOR_DYNAMIC
    free(matriz);
    free(vector);
    free(resultado);
    #endif

    return 0;
}

```

RESPUESTA:

CAPTURAS DE PANTALLA:

a)

```

Modificados : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[mike@looper Modificados]$ ./producto-matricesOMPa
No has indicado la capacidad de la matriz cuadrada
[mike@looper Modificados]$ ./producto-matricesOMPa 12
Tiempo(seg.):0.000005660 / Tamaño del vector: 12 / resultado[0]=12.220000 / resultado[11]=12.220000

Componente 0 de resultado[0] = 12.220000
Componente 1 de resultado[1] = 12.220000
Componente 2 de resultado[2] = 12.220000
Componente 3 de resultado[3] = 12.220000
Componente 4 de resultado[4] = 12.220000
Componente 5 de resultado[5] = 12.220000
Componente 6 de resultado[6] = 12.220000
Componente 7 de resultado[7] = 12.220000
Componente 8 de resultado[8] = 12.220000
Componente 9 de resultado[9] = 12.220000
Componente 10 de resultado[10] = 12.220000
Componente 11 de resultado[11] = 12.220000
[mike@looper Modificados]$

```

b)

```

Modificados : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[mike@looper Modificados]$ ./producto-matricesOMPb
No has indicado la capacidad de la matriz cuadrada
[mike@looper Modificados]$ ./producto-matricesOMPb 12
Tiempo(seg.):0.051145030 / Tamaño del vector: 12 / resultado[0]=12.220000 / resultado[11]=12.220000

Componente 0 de resultado[0] = 12.220000
Componente 1 de resultado[1] = 12.220000
Componente 2 de resultado[2] = 12.220000
Componente 3 de resultado[3] = 12.220000
Componente 4 de resultado[4] = 12.220000
Componente 5 de resultado[5] = 12.220000
Componente 6 de resultado[6] = 12.220000
Componente 7 de resultado[7] = 12.220000
Componente 8 de resultado[8] = 12.220000
Componente 9 de resultado[9] = 12.220000
Componente 10 de resultado[10] = 12.220000
Componente 11 de resultado[11] = 12.220000
[mike@looper Modificados]$

```

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

```

/*
Suma de dos vectores: v3 =v1 +v2
*/

```

```

#include <stdlib.h> // biblioteca con atoi()
#include <stdio.h>
#include <omp.h>
#define VECTOR_DYNAMIC // descomentar para que los vectores sean variables ...
#define PRINTF_ALL // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 33554432 //2^25
double matriz[MAX*MAX], vector[MAX], resultado[MAX];
#endif

int main(int argc, char **argv){
    int i,j;
    double suma;
    //struct timespec cgt1, cgt2;
    double ncgt, tInicio, tFin;
    if(argc<2){
        printf("No has indicado la capacidad de la matriz
cuadrada\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); //como vamos a comparar en varias
ocasiones N con el tamaño MAX, la ponemos en mayúscula para
#ifdef VECTOR_GLOBAL //indicar que se trata de
un entero que puede ser increíblemente grande, pero arbitrario.
double matriz[N*N], vector[N], resultado[N];
#endif

#ifdef VECTOR_GLOBAL
if(N>MAX)
    N=MAX;
#endif

#ifdef VECTOR_DYNAMIC
double *matriz, *vector, *resultado;
matriz = (double*) malloc(N*N*sizeof(double));
vector = (double*) malloc(N*sizeof(double));
resultado = (double*) malloc(N*sizeof(double));

if((resultado==NULL) || (matriz==NULL) || (vector==NULL)){ //si N es
demasiado grande, igual nuestro pc no puede reservar tanta mem.
    printf("\nError a la hora de reservar memoria para la
matriz y/o los vectores\n");
    exit(-2);
}
#endif

//inicialicemos
for(i=0; i<N; i++){
    for(j=0; j<N; j++)
        matriz[(i*N)+j]=N*0.1 +j*0.1; //así
tratamos a la matriz en cuestión como un vector(dimension 1), rellenando todos
vector[i]=N*0.1-i*0.1; //sus
campos
        resultado[i]=0.0;
}
//clock_gettime(CLOCK_REALTIME, &cgt1);

tInicio=omp_get_wtime();

//Ahora indicamos que queremos realizar la multiplicación de forma
paralela, paralelizando el recorrido por columnas (for interno)
for(i=0; i<N; i++){ //así conseguimos que cada hebra ejecute el
programa con las variables privadas, para evitar solapar resultados
    suma=0.0;
#pragma omp parallel for reduction(+:suma)
    for(j=0; j<N; j++)
        suma+=matriz[(i*N)+j]*vector[j];
    resultado[i]=suma;
}
}

```

```

        tFin=omp_get_wtime();

        //clock_gettime(CLOCK_REALTIME, &cgt2);
        ncgt=(tFin-tInicio)*10.0;

        #ifdef PRINTF_ALL
        printf("Tiempo(seg.):%11.9f / Tamaño del vector: %u / resultado[0]=%f /
resultado[%d]=%f \n",ncgt, N, resultado[0], N-1, resultado[N-1]);
        if (N<18){
            for(i=0; i<N; i++)
                printf("\nComponente %d de resultado[%d] =
%f", i,i,resultado[i]);
        }
        printf("\n");
        #endif

        #ifdef VECTOR_DYNAMIC
        free(matriz);
        free(vector);
        free(resultado);
        #endif

        return 0;
}

```

RESPUESTA: Cambiamos el primer `#pragma omp` del apartado b por el que he puesto anteriormente y, `reduction` se encarga de ejecutar correctamente todo lo que he tenido que forzar manualmente en el apartado b del ejercicio anterior.

CAPTURAS DE PANTALLA:

```

Modificados : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Modificados]$ gcc -fopenmp -O2 -o pmOMPreduction producto-matricesOMP-reduction.c
[mike@looper Modificados]$ ./pmOMPreduction 11
Tiempo(seg.):0.000005190 / Tamaño del vector: 11 / resultado[0]=9.460000 / resultado[10]=9.460000

Componente 0 de resultado[0] = 9.460000
Componente 1 de resultado[1] = 9.460000
Componente 2 de resultado[2] = 9.460000
Componente 3 de resultado[3] = 9.460000
Componente 4 de resultado[4] = 9.460000
Componente 5 de resultado[5] = 9.460000
Componente 6 de resultado[6] = 9.460000
Componente 7 de resultado[7] = 9.460000
Componente 8 de resultado[8] = 9.460000
Componente 9 de resultado[9] = 9.460000
Componente 10 de resultado[10] = 9.460000
[mike@looper Modificados]$

```

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: