

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Miguel Ángel Fernández Gutiérrez
Grupo de prácticas y profesor de prácticas: Francisco Barranco, GIM2
Fecha de entrega: 24 de abril, 2019
Fecha evaluación en clase: 25 de abril, 2019

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:

- a) Como no especificamos el alcance de `n`, nos da un error.
- b) Basta añadir `n` a la cláusula `shared`.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```
GNU nano 2.9.3 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;

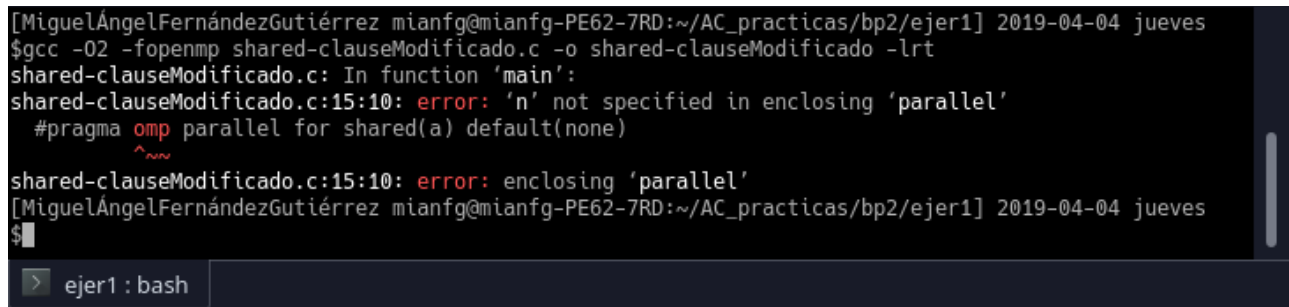
    #pragma omp parallel for shared(a, n) default(none)
    for (i=0; i<n; i++) a[i] += i;

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

^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Text ^J Justificar ^C Posición M-U Deshacer
^X Salir ^R Leer fich. ^\ Reemplazar ^U Pegar txt ^T Ortografía _ Ir a línea M-E Rehacer

> ejer1 : nano
```

CAPTURAS DE PANTALLA:



```
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer1] 2019-04-04 jueves
$gcc -O2 -fopenmp shared-clauseModificado.c -o shared-clauseModificado -lrt
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:15:10: error: 'n' not specified in enclosing 'parallel'
#pragma omp parallel for shared(a) default(none)
          ^~~
shared-clauseModificado.c:15:10: error: enclosing 'parallel'
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer1] 2019-04-04 jueves
$
```

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:

Aunque `private` haga una copia local de la variable, el valor de salida y el valor de entrada de la variable no está definido si no lo definimos. Si lo inicializamos, fuera de la región `parallel` será el definido antes de entrar en `parallel`. Por eso, si inicializamos suma a 122, fuera recuperará ese valor.

Al inicializar a otro valor dentro del `parallel`, el comportamiento será el mismo pero con dicho valor sumado.

CAPTURA CÓDIGO FUENTE: private-clauseModificado.c

```

GNU nano 2.9.3 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 = 122;

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

    #pragma omp parallel private(suma)
    {
        suma=7;
        #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("\nValor de suma fuera de parallel = %d", suma);

    printf("\n");
}

^G Ver ayuda  ^O Guardar  ^W Buscar  ^K Cortar Text  ^J Justificar  ^C Posición  M-U Deshacer
^X Salir      ^R Leer fich. ^\ Reemplazar ^U Pegar txt  ^T Ortografía  _ Ir a línea  M-E Rehacer

> ejer2 : nano

```

CAPTURAS DE PANTALLA:

```

[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer2] 2019-04-04 jueves
$ ./private-clauseModificado
thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 1 suma a[3] / thread 1 suma a[
4] / thread 2 suma a[5] / thread 2 suma a[6] /
* thread 1 suma= 14
* thread 0 suma= 10
* thread 2 suma= 18
Valor de suma fuera de parallel = 122
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer2] 2019-04-04 jueves
$

> ejer2 : bash

```

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

RESPUESTA:

Se trata de condiciones de carrera. El `printf` que se hará será del último valor asignado a `suma` (aunque puede que no, en caso de que alguna de las sumas tarde algo más).

CAPTURA CÓDIGO FUENTE: private-clauseModificado3.c

```
GNU nano 2.9.3 private-clauseModificado3.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 = 122;

    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("\nValor de suma fuera de parallel = %d", suma);
    printf("\n");
}

^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Text ^J Justificar ^C Posición M-U Deshacer
^X Salir ^R Leer fich. ^N Reemplazar ^U Pegar txt ^T Ortografía ^_ Ir a línea M-E Rehacer

> ejer3 : nano
```

CAPTURAS DE PANTALLA:

```
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer3] 2019-04-04 jueves
$./private-clauseModificado3
thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 2 suma a[5] / thread 2 suma a[
6] / thread 1 suma a[3] / thread 1 suma a[4] /
* thread 2 suma= 18
* thread 0 suma= 18
* thread 1 suma= 18
Valor de suma fuera de parallel = 18
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer3] 2019-04-04 jueves
$

> ejer3 : bash
```

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:

No, depende del número de *threads* que utilicemos. Basta ver en la captura de pantalla qué ocurre cuando tenemos menos *threads*.

CAPTURAS DE PANTALLA:

```
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer4] 2019-04-04 jueves
$export OMP_NUM_THREADS=8
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer4] 2019-04-04 jueves
$./firstlastprivate
thread 0 suma a[0] suma=0
thread 1 suma a[1] suma=1
thread 4 suma a[4] suma=4
thread 5 suma a[5] suma=5
thread 6 suma a[6] suma=6
thread 3 suma a[3] suma=3
thread 2 suma a[2] suma=2

Fuera de la construcción parallel suma=6
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer4] 2019-04-04 jueves
$export OMP_NUM_THREADS=2
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer4] 2019-04-04 jueves
$./firstlastprivate
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 1 suma a[4] suma=4
thread 1 suma a[5] suma=9
thread 1 suma a[6] suma=15

Fuera de la construcción parallel suma=15
> ejer4 : bash
```

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:

Que sólo una de las hebras tendrá el valor de `a` actualizado, ya que hemos declarado `a` en el `parallel` y se trata de una variable local a cada una de las hebras. De este modo, el vector se inicializará algunas componentes a 0, y otras al valor de `a` insertado, en función de la distribución que haga OpenMP en el `for`.

CAPTURA CÓDIGO FUENTE: copyprivate-clauseModificado.c

```
GNU nano 2.9.3 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("Después de la región parallel:\n");
    for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
    printf("\n");
}

^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Text ^J Justificar ^C Posición M-U Deshacer
^X Salir ^R Leer fich. ^\ Reemplazar ^U Pegar txt ^T Ortografía ^_ Ir a línea M-E Rehacer
> ejer5 : nano
```

CAPTURAS DE PANTALLA:

```
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer5] 2019-04-04 jueves
$ ./copyprivate-clauseModificado

Introduce valor de inicialización a: 7

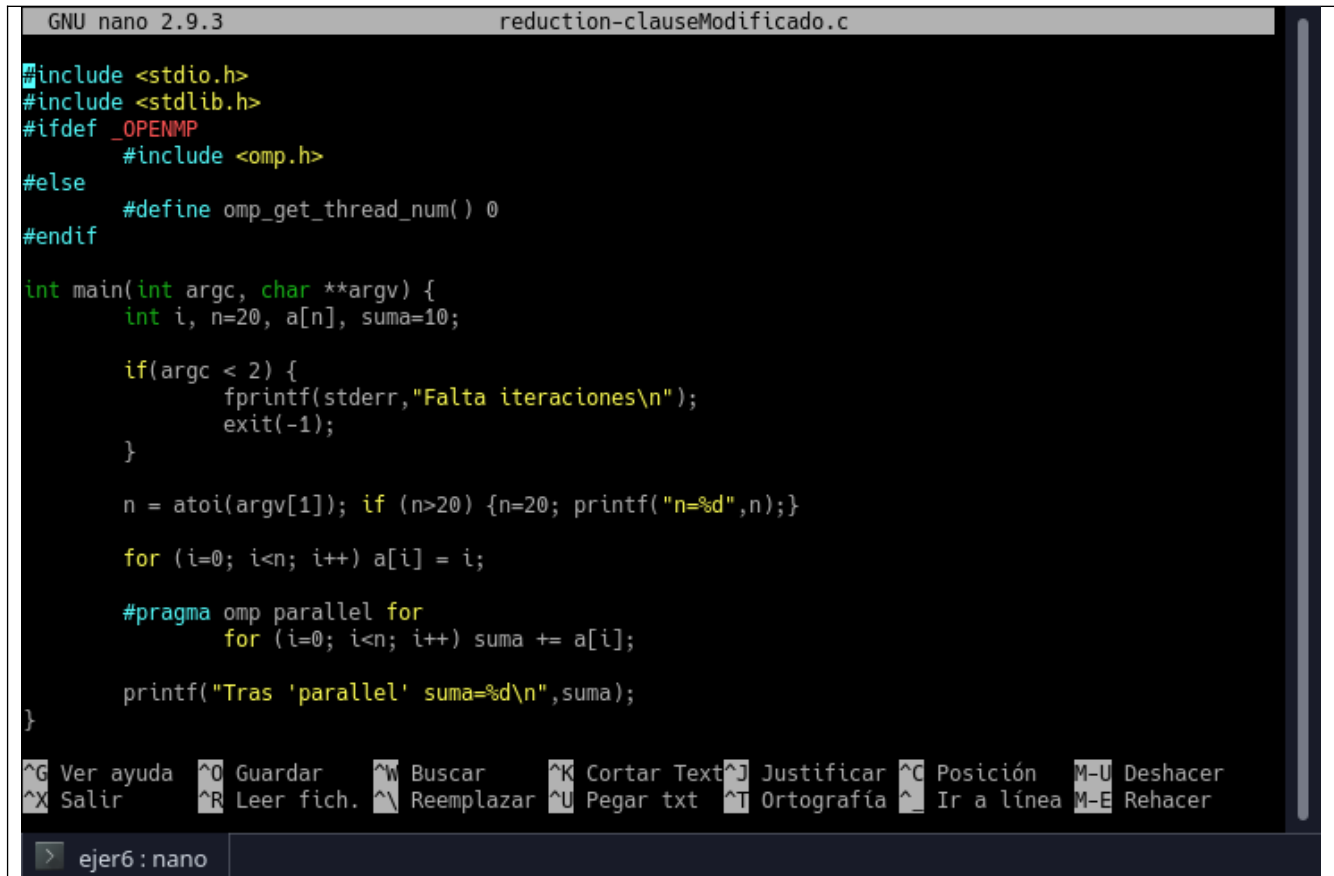
Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 7      b[1] = 7      b[2] = 7      b[3] = 7      b[4] = 7      b[5] = 0      b[6]
= 0      b[7] = 0      b[8] = 0

> ejer5 : bash
```

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:

Queda el valor de `suma + 10`, porque `master` se queda con el valor `10`. Inicializa las variables locales, no el valor de `suma` fuera.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`


```
GNU nano 2.9.3 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);}

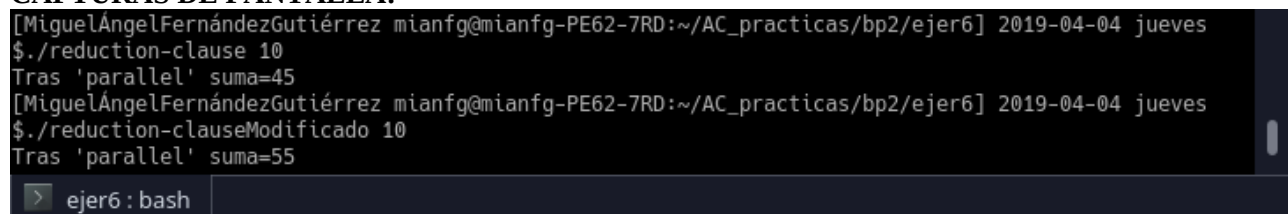
    for (i=0; i<n; i++) a[i] = i;

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

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

Ver ayuda Guardar Buscar Cortar Text Justificar Posición Deshacer
 Salir Leer fich. Reemplazar Pegar txt Ortografía Ir a línea Rehacer

> ejer6 : nano

CAPTURAS DE PANTALLA:


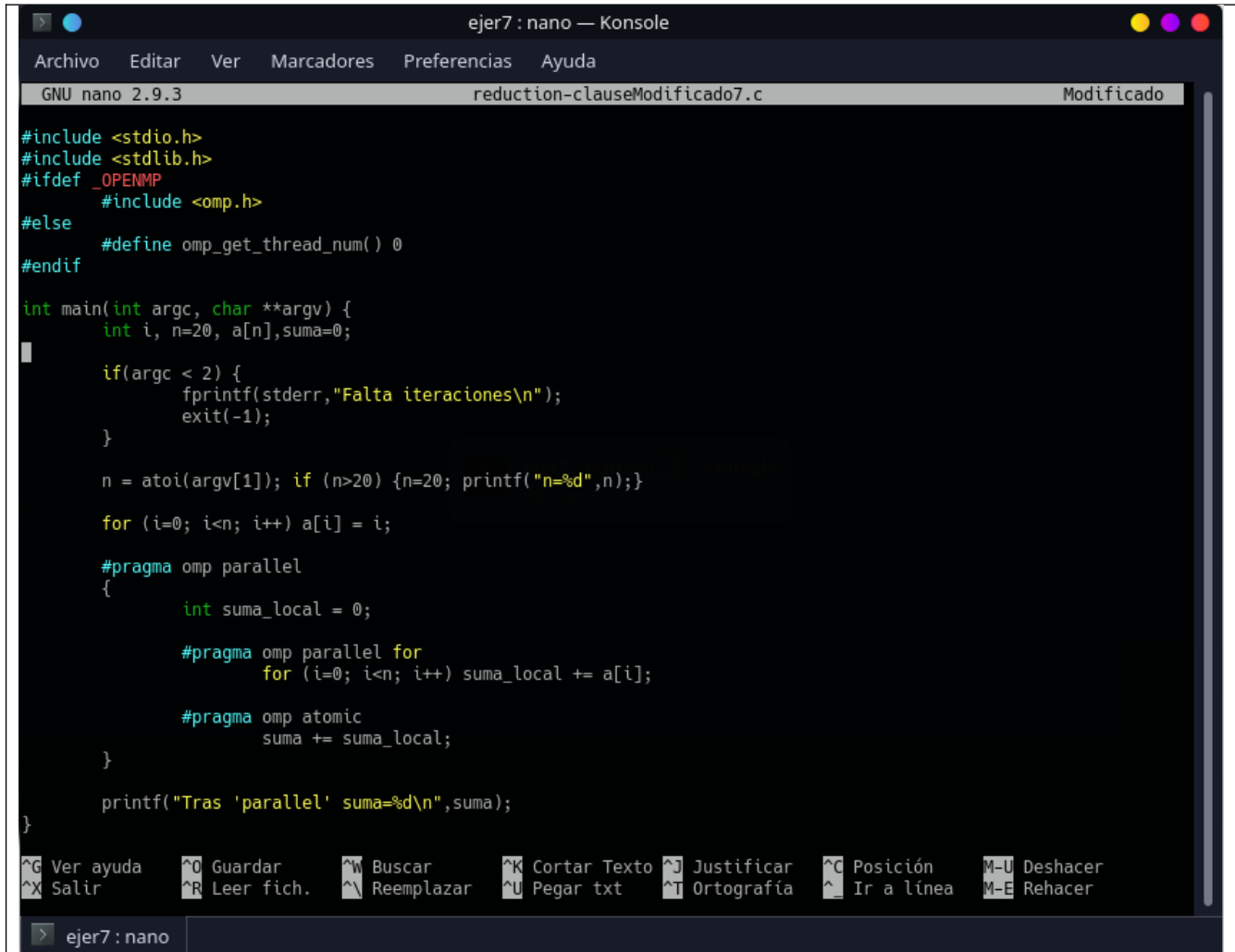
```
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer6] 2019-04-04 jueves
$./reduction-clause 10
Tras 'parallel' suma=45
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer6] 2019-04-04 jueves
$./reduction-clauseModificado 10
Tras 'parallel' suma=55
```

> ejer6 : bash

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`



```

ejer7: nano — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
GNU nano 2.9.3      reduction-clauseModificado7.c      Modificado

#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_local = 0;

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

        #pragma omp atomic
        suma += suma_local;
    }

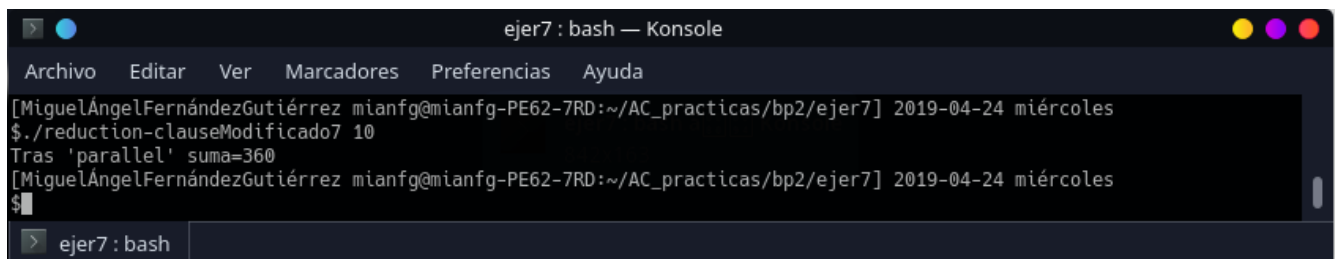
    printf("Tras 'parallel' suma=%d\n", suma);
}

^G Ver ayuda  ^O Guardar  ^W Buscar  ^K Cortar Texto  ^J Justificar  ^C Posición  M-U Deshacer
^X Salir      ^R Leer fich.  ^\ Reemplazar  ^U Pegar txt  ^T Ortografía  ^_ Ir a línea  M-E Rehacer

> ejer7: nano

```

CAPTURAS DE PANTALLA:



```

ejer7: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer7] 2019-04-24 miércoles
$ ./reduction-clauseModificado7 10
Tras 'parallel' suma=360
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer7] 2019-04-24 miércoles
$

> ejer7: bash

```


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):

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

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

```
GNU nano 2.9.3 pmv-secuencial.c
/**
 * @brief Código para calcular producto de una matriz cuadrada por un vector
 * @author Miguel Ángel Fernández Gutiérrez <mianfg@correo.ugr.es>
 * @date 11 abril, 2019
 */

#include <stdlib.h> // atoi(), malloc(), free()
#include <stdio.h> // printf()
#include <time.h> // clock_gettime()

// #define VECTOR_GLOBAL // descomentar si se desea usar vectores globales
#define VECTOR_DYNAMIC // descomentar si se desea usar vectores dinámicos

#ifdef VECTOR_GLOBAL
#define MAX 33554432
double v1[MAX], v2[MAX], M[MAX][MAX];
#elseif
int main(int argc, char** argv) {
    struct timespec cgt1, cgt2; double ncgt; // tiempos de ejecución

    if ( argc < 2 ) {
        printf("ERROR: falta el tamaño N de la matriz NxN y del vector\n");
        exit(EXIT_FAILURE);
    }

    unsigned int N = atoi(argv[1]); // Tamaño del vector

    #ifdef VECTOR_DYNAMIC
    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double));
    v2 = (double*) malloc(N*sizeof(double));
    M = (double**) malloc(N*sizeof(double*));
    for ( int i = 0; i < N; i++ )
        M[i] = (double*) malloc(N*sizeof(double));

    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ) {
        printf("ERROR: en la reserva de memoria dinámica para matriz y vectores\n");
        exit(EXIT_FAILURE);
    }
    #endif

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

        // haremos que la matriz M sea 2*Id (Id = "matriz identidad"), entonces podremos
        // comprobar el resultado simplemente viendo que debe ser v2 = 2*v1
        for ( int j = 0; j < N; j++ )
            if ( i == j )
                M[i][j] = 2;
            else
                M[i][j] = 0;
    }
}
```

```

clock_gettime(CLOCK_REALTIME, &cgt1);
// calculamos el producto v2 = M*v1
for ( int i = 0; i < N; i++ )
    for ( int k = 0; k < N; k++ )
        v2[i] += M[i][k]*v1[k];
clock_gettime(CLOCK_REALTIME, &cgt2);

ncgt = (double) (cgt2.tv_sec-cgt1.tv_sec) +
       (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

// imprimimos v2 por pantalla, y el tiempo de ejecución
printf("Resultado: v2 = ");
for ( int i = 0; i < N; i++ )
    printf("%f ", v2[i]);
printf("\n\nTamaño: %d -> Tiempo de ejecución: %f\n\n", N, ncgt);

#ifdef VECTOR_DYNAMIC
free(v1);
free(v2);
for ( int i = 0; i < N; i++ )
    free(M[i]);
free(M);
#endif
}

```

CAPTURAS DE PANTALLA:

Nota: He decidido usar el código para dinámicas. Por otra parte, podemos ver que el código es correcto teniendo en cuenta que hemos inicializado la matriz a $M=2*Id$. Por tanto, $v2 = 2Id*v1 = 2v1$. En efecto,

```

ejer8: bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer8] 2019-04-11 jueves
$ ./pmv-secuencial 16
Resultado: v2 = 0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 14.000000 16.000000
18.000000 20.000000 22.000000 24.000000 26.000000 28.000000 30.000000

Tamaño: 16 -> Tiempo de ejecución: 0.000004
ejer8: bash

```

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):

- a. una primera que paralelice el bucle que recorre las filas de la matriz y
- b. 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).

Nota: en las capturas incluyo únicamente las secciones donde aparecen las modificaciones al código (el resto de éste es exactamente el mismo que el del ejercicio 8). Análogamente para ejercicios posteriores.

CAPTURA CÓDIGO FUENTE : pmv-OpenMP-a.c

```
#ifndef _OPENMP
double start = omp_get_wtime();
#else
clock_gettime(CLOCK_REALTIME, &cgt1);
#endif
// calculamos el producto v2 = M*v1
#pragma omp parallel for
for ( int i = 0; i < N; i++ )
    for ( int k = 0; k < N; k++ )
        v2[i] += M[i][k]*v1[k];

#ifdef _OPENMP
ncgt = omp_get_wtime() - start;
#else
clock_gettime(CLOCK_REALTIME, &cgt2);
ncgt = (double) (cgt2.tv_sec-cgt1.tv_sec) +
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
#endif

// imprimimos v2 por pantalla, y el tiempo de ejecución
printf("Resultado: v2 = ");
for ( int i = 0; i < N; i++ )
    printf("%f ", v2[i]);
printf("\n\nTamaño: %d -> Tiempo de ejecución: %f\n\n", N, ncgt);
```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```
#ifndef _OPENMP
double start = omp_get_wtime();
#else
clock_gettime(CLOCK_REALTIME, &cgt1);
#endif
double suma_parcial;
// calculamos el producto v2 = M*v1
#pragma omp parallel private(suma_parcial)
for ( int i = 0; i < N; i++ ) {
    suma_parcial = 0;
    #pragma omp for
    for ( int k = 0; k < N; k++ )
        suma_parcial += M[i][k]*v1[k];
    #pragma omp critical
        v2[i] += suma_parcial;
}
#ifdef _OPENMP
ncgt = omp_get_wtime() - start;
#else
```

RESPUESTA:

CAPTURAS DE PANTALLA:

```
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer9] 2019-04-24 miércoles
$./pmv-OpenMP-a 7
Resultado: v2 = 0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 v2[0]: 0.000000, v2[6]: 12.000000

Tamaño: 7 -> Tiempo de ejecución: 0.002325

[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer9] 2019-04-24 miércoles
$./pmv-OpenMP-b 7
Resultado: v2 = 0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 v2[0]: 0.000000, v2[6]: 12.000000

Tamaño: 7 -> Tiempo de ejecución: 0.001298
```

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

```
#ifndef _OPENMP
double start = omp_get_wtime();
#else
clock_gettime(CLOCK_REALTIME, &cgt1);
#endif
double suma_parcial = 0;
// calculamos el producto v2 = M*v1
#pragma omp parallel
for ( int i = 0; i < N; i++ ) {
    #pragma omp for reduction(+:suma_parcial)
    for ( int k = 0; k < N; k++ )
        suma_parcial += M[i][k]*v1[k];

    #pragma omp single
    {
        v2[i] = suma_parcial;
        suma_parcial = 0;
    }
}
#endif _OPENMP
ncgt = omp_get_wtime() - start;
#else
```

RESPUESTA:

CAPTURAS DE PANTALLA:

```
ejer10 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer10] 2019-04-24 miércoles
$ ./pmv-OpenMP-reduction 7
Resultado: v2 = 0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 v2[0]: 0.000000, v2[6]: 12.000000

Tamaño: 7 -> Tiempo de ejecución: 0.003422

[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp2/ejer10] 2019-04-24 miércoles
$
```

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.

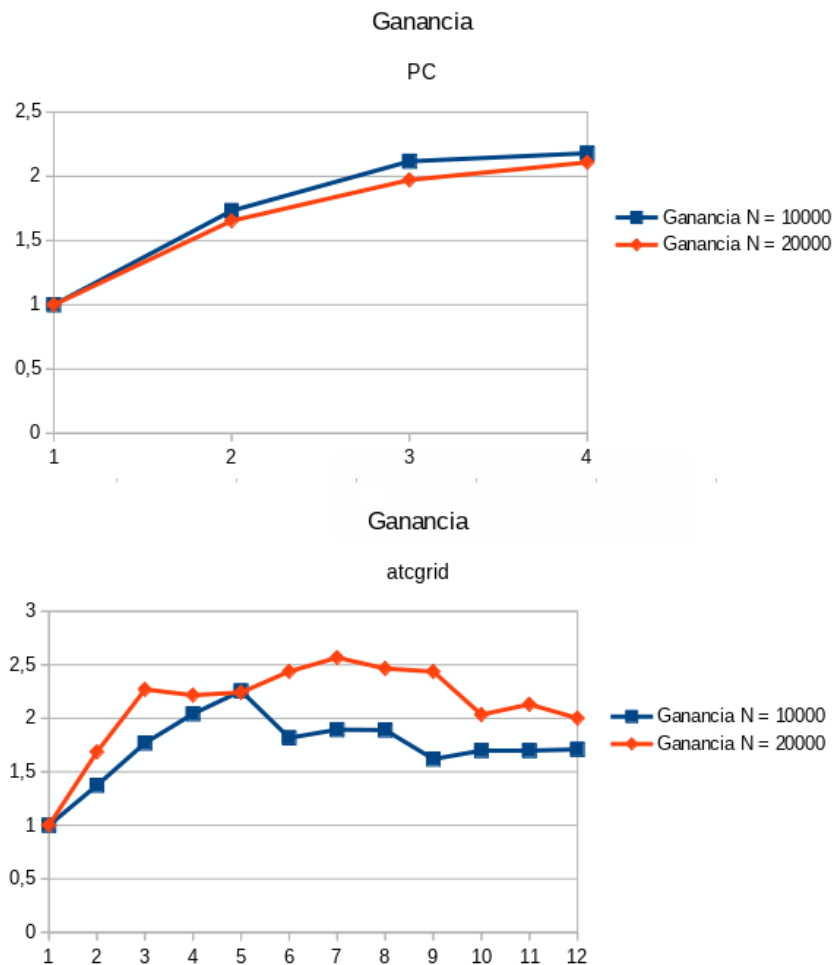
CAPTURAS DE PANTALLA (que justifique el código elegido):

El de OpenMP-b.c:

```
#ifdef _OPENMP
double start = omp_get_wtime();
#else
clock_gettime(CLOCK_REALTIME, &cgt1);
#endif
double suma_parcial;
// calculamos el producto v2 = M*v1
#pragma omp parallel private(suma_parcial)
for ( int i = 0; i < N; i++ ) {
    suma_parcial = 0;
    #pragma omp for
    for ( int k = 0; k < N; k++ )
        suma_parcial += M[i][k]*v1[k];
    #pragma omp critical
        v2[i] += suma_parcial;
}
#ifdef _OPENMP
ncgt = omp_get_wtime() - start;
#else
```

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):

Este ejercicio está hecho en el LibreOffice Calc adjunto (en la carpeta “ ejer11” se encuentra todo lo relativo a este ejercicio), ya que es imposible insertar la tabla con todos los cálculos aquí.



COMENTARIOS SOBRE LOS RESULTADOS:

Podemos ver, claramente, que la paralelización es un procedimiento bastante idóneo para este tipo de tareas, pero que la excesiva creación de *threads* conlleva un gasto a la hora de crear, comunicar y terminarlos. Por eso, cuando sobrepasamos el número de *threads* creados al de *cores* lógicos de nuestro computador, comenzaremos a ver penalizaciones significativas en velocidad.