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): Noelia Escalera Mejías
Grupo de prácticas y profesor de prácticas: A2 (Christian Morillas)
Fecha de entrega: 09/04/19
Fecha evaluación en clase: 10/04/19

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**:

   Lo que ocurre es que da un error porque tenemos que especificar el alcance de las variables de la construcción. Hemos especificado el de a con shared, pero no el de n. El de i no haría falta ya que las variables iteradoras son privadas.

   **CAPTURA CÓDIGO FUENTE**: `shared-clauseModificado.c`

```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]);
}
```

   **CAPTURAS DE PANTALLA:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer1] 2019-04-03 miércoles
$gcc -O2 -fopenmp shared-clauseModificado.c -o shared-clauseModificado
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:14:10: error: 'n' not specified in enclosing 'parallel'
  #pragma omp parallel for shared(a) default(none)
          ^~~
shared-clauseModificado.c:14:10: error: enclosing 'parallel'
```

Compilación del codigo erróneo

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer1] 2019-04-03 miércoles
$gcc -O2 -fopenmp shared-clauseModificado.c -o shared-clauseModificado
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer1] 2019-04-03 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
```

Compilación y ejecución del código arreglado

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**:
El código fuera de parallel siempre imprime 0, aunque modifiquemos la variable dentro de parallel. Esto se debe a que la variable suma está en la lista de private y por lo tanto su valor antes y después de parallel queda indefinido. Podría haber sido otro valor ya que fuera de parallel no definimos el valor en ningún momento. Si por el contrario, modificamos la variable suma fuera de parallel, si que nos imprime este valor y sin afectar a la región paralela.

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

```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 private(suma)
{
    suma=1;
    #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("\nSuma fuera de parallel: %d",suma);

    printf("\n");
}
```

Código con la primera modificación

```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 = 3;

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

#pragma omp parallel private(suma)
{
    suma=1;
    #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("\nSuma fuera de parallel: %d",suma);

    printf("\n");
}
```

Código con la segunda modificación

**CAPTURAS DE PANTALLA:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer2] 2019-04-03 miércoles
$gcc -O2 -fopenmp private-clauseModificado.c -o private-clauseModificado
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer2] 2019-04-03 miércoles
$./private-clauseModificado
thread0 suma a[0]/thread0 suma a[1]/thread2 suma a[4]/thread2 suma a[5]/thread1 suma a[2]/thread1 suma a[3]/thread3 suma a[6]/
* thread 2 suma= 10
* thread 1 suma= 6
* thread 0 suma= 2
* thread 3 suma= 7
Suma fuera de parallel: 0
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer2] 2019-04-03 miércoles
$./private-clauseModificado
thread0 suma a[0]/thread0 suma a[1]/thread2 suma a[4]/thread2 suma a[5]/thread3 suma a[6]/thread1 suma a[2]/thread1 suma a[3]/
* thread 2 suma= 10
* thread 3 suma= 7
* thread 0 suma= 2
* thread 1 suma= 6
Suma fuera de parallel: 0
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer2] 2019-04-03 miércoles
$./private-clauseModificado
thread2 suma a[4]/thread2 suma a[5]/thread1 suma a[2]/thread1 suma a[3]/thread0 suma a[0]/thread0 suma a[1]/thread3 suma a[6]/
* thread 0 suma= 2
* thread 1 suma= 6
* thread 2 suma= 10
* thread 3 suma= 7
Suma fuera de parallel: 0
```

Compilación y varias ejecuciones de la modificación 1

Compilación y ejecución de la segunda modificación

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer2] 2019-04-03 miércoles
$gcc -O2 -fopenmp private-clauseModificado2.c -o private-clauseModificado2
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer2] 2019-04-03 miércoles
$./private-clauseModificado2
thread1 suma a[2]/thread1 suma a[3]/thread0 suma a[0]/thread0 suma a[1]/thread2 suma a[4]/thread2 suma a[5]/thread3 suma a[6]/
* thread 2 suma= 10
* thread 0 suma= 2
* thread 1 suma= 6
* thread 3 suma= 7
Suma fuera de parallel: 3
```

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

   **RESPUESTA**:

   Ocurre que todas las hebras comparten la variable, luego el resultado de la suma es el mismo.

   **CAPTURA CÓDIGO FUENTE**: `private-clauseModificado3.c`

```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:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer3] 2019-04-03 miércoles
$gcc -O2 -fopenmp private-clauseModificado3.c -o private-clauseModificado3
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer3] 2019-04-03 miércoles
$./private-clauseModificado3
thread1 suma a[2]/thread1 suma a[3]/thread2 suma a[4]/thread2 suma a[5]/thread0 suma a[0]/thread0 suma a[1]/thread3 suma a[6]/
* thread 2 suma= 15
* thread 0 suma= 15
* thread 1 suma= 15
* thread 3 suma= 15
```

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**:

   Sí imprime siempre 6 ya que al usar firstprivate y lastprivate se mantiene el valor inicial (antes de parallel) y el final (al final de parallel) de la variable suma.

   **CAPTURAS DE PANTALLA:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer4] 2019-04-03 miércoles
$gcc -O2 -fopenmp firstlastprivate.c -o firstlastprivate
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer4] 2019-04-03 miércoles
$./firstlastprivate
 thread 2 suma a[4] suma=4
 thread 2 suma a[5] suma=9
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 3 suma a[6] suma=6

Fuera de la construcción parallel suma=6
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer4] 2019-04-03 miércoles
$./firstlastprivate
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 2 suma a[4] suma=4
 thread 2 suma a[5] suma=9
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 3 suma a[6] suma=6

Fuera de la construcción parallel suma=6
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer4] 2019-04-03 miércoles
$./firstlastprivate
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 2 suma a[4] suma=4
 thread 2 suma a[5] suma=9
 thread 3 suma a[6] suma=6
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5

Fuera de la construcción parallel suma=6
```

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**:

Ocurre que la variable privada que lee el thread que entra en single no se comparte a los otros threads, luego no todas las posiciones del vector tienen el mismo valor.

**CAPTURA CÓDIGO FUENTE**: `copyprivate-clauseModificado.c`

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

**CAPTURAS DE PANTALLA:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer5] 2019-04-19 viernes
$gcc -O2 -fopenmp copyprivate-clauseModificado.c -o copyprivate-clauseModificado
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer5] 2019-04-19 viernes
$./copyprivate-clauseModificado

Introduce valor de inicialización a: 9

Single ejecutada por el thread 1
Depuésde la región parallel:
b[0] = 0        b[1] = 0        b[2] = 0        b[3] = 9        b[4] = 9        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 (añada capturas de pantalla que muestren lo que ocurre)

**RESPUESTA**:

Ocurre que a la suma se le añaden 10 unidades más, ya que ahora está incializada a 10 y no a 0.

**CAPTURA CÓDIGO FUENTE**: `reduction-clauseModificado.c`

```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:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer6] 2019-04-19 viernes
$gcc -O2 -fopenmp reduction-clauseModificado.c -o reduction-clauseModificado
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer6] 2019-04-19 viernes
$./reduction-clauseModificado 10
Tras 'parallel' suma=55
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer6] 2019-04-19 viernes
$./reduction-clauseModificado 20
Tras 'parallel' suma=200
```

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`

```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_i, 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 private(suma_i)
    {
        suma_i=0;
        #pragma omp for
        for (i=0; i<n; i++) suma_i += a[i];

        #pragma omp critical
        suma+=suma_i;

        #pragma omp barrier
    }


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

**CAPTURAS DE PANTALLA:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer7] 2019-04-20 sábado
$gcc -O2 -fopenmp reduction-clauseModificado7.c -o reduction-clauseModificado7
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer7] 2019-04-20 sábado
$./reduction-clauseModificado7 10
Tras 'parallel' suma=45
```

## Resto de ejercicios

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

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

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`

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

int main(int argc, char ** argv){
    if (argc != 2){
        printf("Faltan las dimensiones de la matriz y vector\n");
        exit(1);
    }

    int i, j, contador, N = atoi(argv[1]);
    double **m, *v, *resultado, t_inicio, t_fin;

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

    for(i=0; i<N; i++){
        m[i] = (double*)malloc(N*sizeof(double));
    }

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

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

    contador = 0;

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            m[i][j] = contador;
            contador++;
        }
    }

    contador = 0;

    for(i=0; i<N; i++){
        v[i] = contador;
        contador++;
    }

    for(i=0; i<N; i++){
        resultado[i] = 0;
    }

    t_inicio = omp_get_wtime();

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            resultado[i] += m[i][j]*v[j];
        }
    }

    t_fin = omp_get_wtime();

    double tiempo = t_fin - t_inicio;

    if(N<11){
        printf ("Resultados:\n");
        for(i=0; i<N; i++){
            printf("%f ",resultado[i]);
        }
        printf("\n");
    }
    else{
        printf("resultado[0]=%f",resultado[0]);
        printf("\nresultado[N-1]=%f\n",resultado[N-1]);
    }

    printf("Tiempo: %f\n", tiempo);

    free(v);

    for(i=0; i<N; i++){
        free(m[i]);
    }

    free(m);

    free(resultado);

    return (0);
}
```

**CAPTURAS DE PANTALLA:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer8] 2019-04-20 sábado
$gcc -O2 -fopenmp pmv-secuencial.c -o pmv-secuencial
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer8] 2019-04-20 sábado
$./pmv-secuencial 8
Resultados:
140.000000 364.000000 588.000000 812.000000 1036.000000 1260.000000 1484.000000 1708.000000
Tiempo: 0.000001
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer8] 2019-04-20 sábado
$./pmv-secuencial 11
resultado[0]=385.000000
resultado[N-1]=6435.000000
Tiempo: 0.000001
```

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).

**CAPTURA CÓDIGO FUENTE** : `pmv-OpenMP-a.c`

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

int main(int argc, char ** argv){
    if (argc != 2){
        printf("Faltan las dimensiones de la matriz y vector\n");
        exit(1);
    }

    int i, j, contador, N = atoi(argv[1]);
    double **m, *v, *resultado, t_inicio, t_fin;

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

    for(i=0; i<N; i++){
        m[i] = (double*)malloc(N*sizeof(double));
    }

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

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

    contador = 0;

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            m[i][j] = contador;
            contador++;
        }
    }

    contador = 0;

    for(i=0; i<N; i++){
        v[i] = contador;
        contador++;
    }

    for(i=0; i<N; i++){
        resultado[i] = 0;
    }

    t_inicio = omp_get_wtime();

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            resultado[i] += m[i][j]*v[j];
        }
    }

    t_fin = omp_get_wtime();

    double tiempo = t_fin - t_inicio;

    if(N<11){
        printf ("Resultados:\n");
        for(i=0; i<N; i++){
            printf("%f ",resultado[i]);
        }
        printf("\n");
    }
    else{
        printf("resultado[0]=%f",resultado[0]);
        printf("\nresultado[N-1]=%f\n",resultado[N-1]);
    }

    printf("Tiempo: %f\n", tiempo);

    free(v);

    for(i=0; i<N; i++){
        free(m[i]);
    }

    free(m);

    free(resultado);

    return (0);
}
```

**CAPTURA CÓDIGO FUENTE**: `pmv-OpenMP-b.c`

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

int main(int argc, char ** argv){
    if (argc != 2){
        printf("Faltan las dimensiones de la matriz y vector\n");
        exit(1);
    }

    int i, j, contador, N = atoi(argv[1]);
    double **m, *v, *resultado, t_inicio, t_fin, parcial=0;

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

    for(i=0; i<N; i++){
        m[i] = (double*)malloc(N*sizeof(double));
    }

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

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

    contador = 0;

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            m[i][j] = contador;
            contador++;
        }
    }

    contador = 0;

    for(i=0; i<N; i++){
        v[i] = contador;
        contador++;
    }

    for(i=0; i<N; i++){
        resultado[i] = 0;
    }

    t_inicio = omp_get_wtime();

    #pragma omp parallel private(i) firstprivate(parcial)
    {
        for(i=0; i<N; i++){
            #pragma omp for
            for(j=0; j<N; j++){
                parcial += m[i][j]*v[j];
            }

            #pragma omp atomic
            resultado[i] += parcial;
            #pragma omp barrier

            parcial = 0;
        }
    }

    t_fin = omp_get_wtime();

    double tiempo = t_fin - t_inicio;

    if(N<11){
        printf ("Resultados:\n");
        for(i=0; i<N; i++){
            printf("%f ",resultado[i]);
        }
        printf("\n");
    }
    else{
        printf("resultado[0]=%f",resultado[0]);
        printf("\nresultado[N-1]=%f\n",resultado[N-1]);
    }

    printf("Tiempo: %f\n", tiempo);

    free(v);

    for(i=0; i<N; i++){
        free(m[i]);
    }

    free(m);

    free(resultado);

    return (0);
}
```

**RESPUESTA**:

En la resolución por columnas he tenido un error de ejecución en la segunda parte del ejercicio, no daba siempre el mismo resultado. Lo he solucionado añadiendo una variable local que calculara sumas parciales y después las añadiera al vector solución.

**CAPTURAS DE PANTALLA:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer9] 2019-04-22 lunes
$gcc -O2 -fopenmp pmv-OpenMP-a.c -o pmv-OpenMP-a
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer9] 2019-04-22 lunes
$./pmv-OpenMP-a 8
Resultados:
140.000000 364.000000 588.000000 812.000000 1036.000000 1260.000000 1484.000000 1708.000000
Tiempo: 0.000226
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer9] 2019-04-22 lunes
$./pmv-OpenMP-a 11
resultado[0]=385.000000
resultado[N-1]=6435.000000
Tiempo: 0.000205
```

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer9] 2019-04-22 lunes
$gcc -O2 -fopenmp pmv-OpenMP-b.c -o pmv-OpenMP-b
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer9] 2019-04-22 lunes
$./pmv-OpenMP-b 8
Resultados:
140.000000 364.000000 588.000000 812.000000 1036.000000 1260.000000 1484.000000 1708.000000
Tiempo: 0.000350
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer9] 2019-04-22 lunes
$./pmv-OpenMP-b 11
resultado[0]=385.000000
resultado[N-1]=6435.000000
Tiempo: 0.000332
```

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`

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

int main(int argc, char ** argv){
    if (argc != 2){
        printf("Faltan las dimensiones de la matriz y vector\n");
        exit(1);
    }

    int i, j, contador, N = atoi(argv[1]);
    double **m, *v, *resultado, t_inicio, t_fin;

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

    for(i=0; i<N; i++){
        m[i] = (double*)malloc(N*sizeof(double));
    }

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

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

    contador = 0;

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            m[i][j] = contador;
            contador++;
        }
    }

    contador = 0;

    for(i=0; i<N; i++){
        v[i] = contador;
        contador++;
    }

    for(i=0; i<N; i++){
        resultado[i] = 0;
    }

    t_inicio = omp_get_wtime();

    #pragma omp parallel private(i)
    {
        for(i=0; i<N; i++){
            #pragma omp for reduction (+:resultado[i])
            for(j=0; j<N; j++){
                resultado[i] += m[i][j]*v[j];
            }
        }
    }

    t_fin = omp_get_wtime();

    double tiempo = t_fin - t_inicio;

    if(N<11){
        printf ("Resultados:\n");
        for(i=0; i<N; i++){
            printf("%f ",resultado[i]);
        }
        printf("\n");
    }
    else{
        printf("resultado[0]=%f",resultado[0]);
        printf("\nresultado[N-1]=%f\n",resultado[N-1]);
    }

    printf("Tiempo: %f\n", tiempo);

    free(v);

    for(i=0; i<N; i++){
        free(m[i]);
    }

    free(m);

    free(resultado);

    return (0);
}
```

**RESPUESTA**:

**CAPTURAS DE PANTALLA:**

```
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer10] 2019-04-23 martes
$gcc -O2 -fopenmp pmv-OpenMP-reduction.c -o pmv-OpenMP-reduction
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer10] 2019-04-23 martes
$./pmv-OpenMP-reduction 8
Resultados:
140.000000 364.000000 588.000000 812.000000 1036.000000 1260.000000 1484.000000 1708.000000
Tiempo: 0.000239
[NoeliaEscaleraMejias noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/AC/Prácticas/bp2/ejer10] 2019-04-23 martes
$./pmv-OpenMP-reduction 11
resultado[0]=385.000000
resultado[N-1]=6435.000000
Tiempo: 0.000304
```

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):**

Para comprobar cuál es el mejor código, he usado mi PC y he ejecutado cada código variando los cores de 1 a 4 con tamaño 5000 y 20000 y he comprobado cuál es el más rápido. Nos ha quedado la siguiente comparativa:

| N.º CPU'S | Tamaño 5000 filas | Tamaño 20000 filas | Tamaño 5000 columnas | Tamaño 20000 columnas | Tamaño 5000 reduction | Tamaño 20000 reduction |
|---|---|---|---|---|---|---|
| 1 | 0,027712 | 2,279499 | 0,02972 | 0,505569 | 0,02626 | 0,506307 |
| 2 | 0,025878 | 0,337044 | 0,023181 | 0,350372 | 0,022911 | 0,341595 |
| 3 | 0,02456 | 0,313002 | 0,029255 | 0,359903 | 0,031312 | 0,331426 |
| 4 | 0,026905 | 0,313372 | 0,040292 | 0,340295 | 0,034341 | 0,365506 |

He escogido por tanto el de filas, ya que es el que mejor funciona en paralelo.

**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)**:

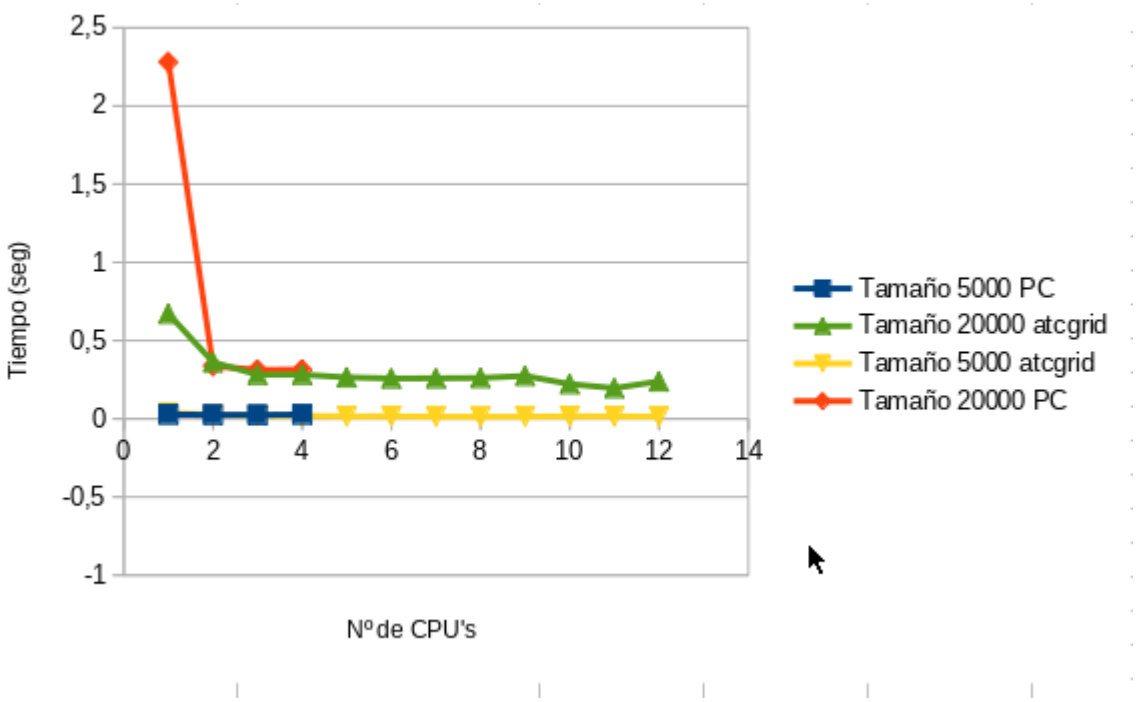
| N.º CPU'S | Tamaño 5000 | Tamaño 20000 | Ganancia para tamaño 5000 | Ganancia para tamaño 20000 |
|---|---|---|---|---|
| 1 | 0,027712 | 2,279499 | 0,970879041570439 | 0,137474067766645 |
| 2 | 0,025878 | 0,337044 | | |
| 3 | 0,02456 | 0,313002 | | |
| 4 | 0,026905 | 0,313372 | | |

Tabla para el PC

| N.º CPU'S | Tamaño 5000 atcgrid | Tamaño 20000 atcgrid | Ganancia para tamaño 5000 | Ganancia para tamaño 20000 |
|---|---|---|---|---|
| 1 | 0,036968 | 0,671122 | 0,363476520233716 | 0,356158790801076 |
| 2 | 0,027782 | 0,361846 | | |
| 3 | 0,02018 | 0,281731 | | |
| 4 | 0,018376 | 0,281998 | | |
| 5 | 0,015083 | 0,264672 | | |
| 6 | 0,015715 | 0,258033 | | |
| 7 | 0,013828 | 0,25909 | | |
| 8 | 0,012667 | 0,260937 | | |
| 9 | 0,013741 | 0,274119 | | |
| 10 | 0,015318 | 0,221757 | | |
| 11 | 0,014877 | 0,195908 | | |
| 12 | 0,013437 | 0,239026 | | |

Tabla de atcgrid

**COMENTARIOS SOBRE LOS RESULTADOS:**


2,5
2
1,5
1
0,5
0
-0,5
-1
0
2
4
6
8
10
12
14
Tiempo (seg)
Nº de CPU's
Tamaño 5000 PC
Tamaño 20000 atcgrid
Tamaño 5000 atcgrid
Tamaño 20000 PC