

Grai2º curso / 2º
cuatr.
Grado Ing. Inform.
Doble Grado Ing.
Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Francisco Carrillo Pérez

Grupo de prácticas: A2

Fecha de entrega:

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:

CÓDIGO FUENTE: shared-clauseModificado.c

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/  
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/  
/* INTERLINEADO SENCILLO */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <omp.h>  
  
int main(int argc, char ** argv)  
{  
...  
}
```

CAPTURAS DE PANTALLA:

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: Es indiferente el valor al que inicializemos suma ya sea dentro o fuera ya que el valor de entrada y salida está indefinido aunque se inicialice el valor fuera como hemos hecho aquí.

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;  
    for (i=0; i<n; i++)
```

```

        a[i] = i;
    suma=10;
    #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");
}

```

CAPTURAS DE PANTALLA:

```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/A
C/Prácticas/P2
% ./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 2 suma= 9
thread 0 suma= 1
thread 3 suma= 6
thread 1 suma= 5
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/A
C/Prácticas/P2
%

```

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

RESPUESTA: En este caso al no hacer la variable privada todas las hebras tienen la misma variable y por ello todas al final tienen el mismo valor, como podemos observar en la captura.

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

    suma=0;
    #pragma omp parallel
    {

        #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:

```

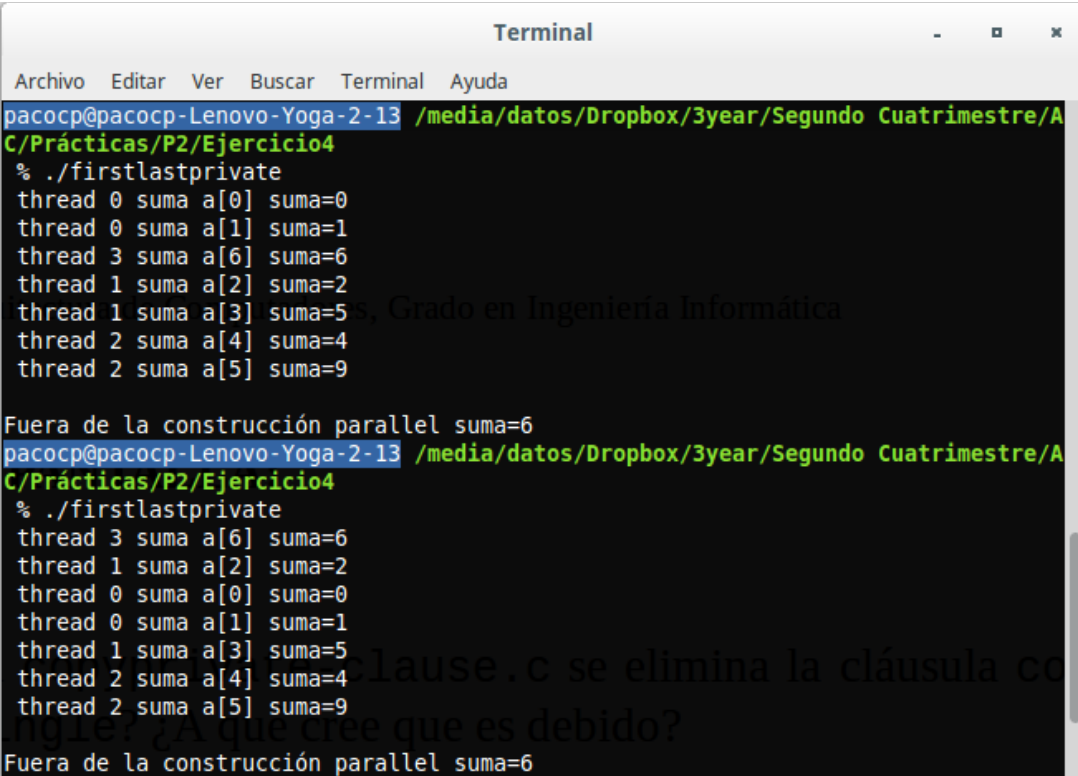
Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/A
C/Prácticas/P2
% ./private-clauseModificado3
thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma a[4] / thread 2 suma a[5]
] / thread 1 suma a[2] / thread 1 suma a[3] / thread 3 suma a[6] /
thread 1 suma= 21
thread 2 suma= 21
thread 0 suma= 21
thread 3 suma= 21
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/A
C/Prácticas/P2
%

```

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: Si, siempre imprime 6 debido a la directiva `lastprivate`, ya que siempre muestra el valor que se ha obtenido en la última iteración, el cuál es 6.

CAPTURAS DE PANTALLA:



```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/A
C/Prácticas/P2/Ejercicio4
% ./firstlastprivate
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9

Fuera de la construcción parallel suma=6
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/A
C/Prácticas/P2/Ejercicio4
% ./firstlastprivate
thread 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 1 suma a[3] suma=5
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9

Fuera de la construcción parallel suma=6
  
```

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: Solo ciertas posiciones se inicializan la valor de `a` y esto es debido a que solo se le da ese valor `a` en la región `single`, por lo que solo una hebra tendrá el valor que le hemos dado. Como hemos quitado la cláusula `copyprivate`, no se comparte al resto de hebras.

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("\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");
}

```

CAPTURAS DE PANTALLA:

```

Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P2/Ejercic
io5
% ./copyprivate-clauseModificado
Introduce valor de inicialización a: 5
Single ejecutada por el thread 0
Después de la región 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
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P2/Ejercic
io5
%

```

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

RESPUESTA:

La suma aumenta en 10 ya que hemos inicializado el valor en 10 en lugar de en 0.

CÓDIGO FUENTE: reduction-clauseModificado.c

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

main(int argc, char **argv){
    int i, n=20, a[n], suma=10;

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

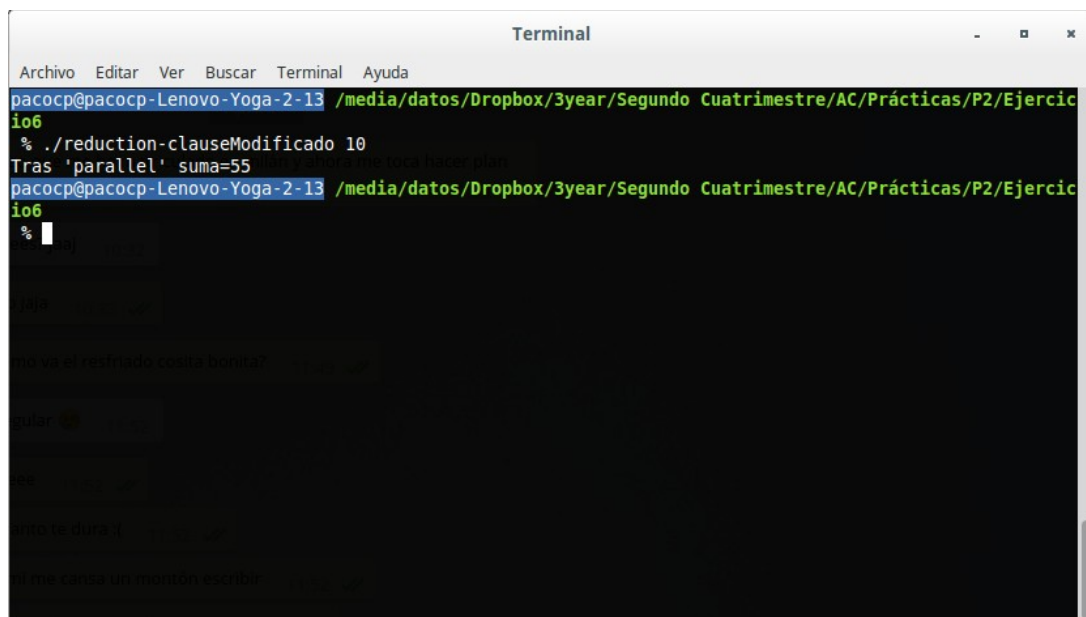
    n = atoi(argv[1]);
    if(n>20)
    {
        n=20; printf("n=%d", n);
    }

    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:



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:

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) private(i)
    {
        for (i=omp_get_thread_num(); i<n;
i+=omp_get_num_threads()) suma += a[i];

    }

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

CAPTURAS DE PANTALLA:

```

Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P2
% ./reduction-clauseModificado 10
Tras 'parallel' suma=45
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P2
%

```

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

```

// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

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

    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }
}

```



```

        unsigned int N = atoi(argv[1]); // Máximo N =2^32-
        1=4294967295 (sizeof(unsigned int) = 4 B)

        double *v1, *v2, **M;
        v1 = (double*) malloc(N*sizeof(double)); // malloc necesita
        el tamaño en bytes
        v2 = (double*) malloc(N*sizeof(double)); //si no hay
        espacio suficiente malloc devuelve NULL
        M = (double**) malloc(N*sizeof(double *));
        if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
            printf("Error en la reserva de espacio para
los vectores\n");
            exit(-2);
        }

        for (i=0; i<N; i++){
            M[i] = (double*) malloc(N*sizeof(double));
            if ( M[i]==NULL ){
                printf("Error en la reserva de
espacio para los vectores\n");
                exit(-2);
            }
        }
        //A partir de aqui se pueden acceder las componentes de la
        matriz como M[i][j]

        //Inicializar matriz y vectores
        for(i = 0; i< N; i++)
        {
            v1[i] = 2;
            v2[i] = 2;
            for(j = 0; j < N; j++ )
            {
                M[i][j] = 2;
            }
        }

        //Medida de tiempo
        t1 = omp_get_wtime();

        //Calcular producto de matriz por vector v2 = M · v1
        for(i = 0; i<N;i++)
        {
            for(j=0; j<N;j++)
            {
                v2[i] = M[i][j] * v1[j];
            }
        }

        //Medida de tiempo
        t2 = omp_get_wtime();
        total = t2 - t1;

```

```

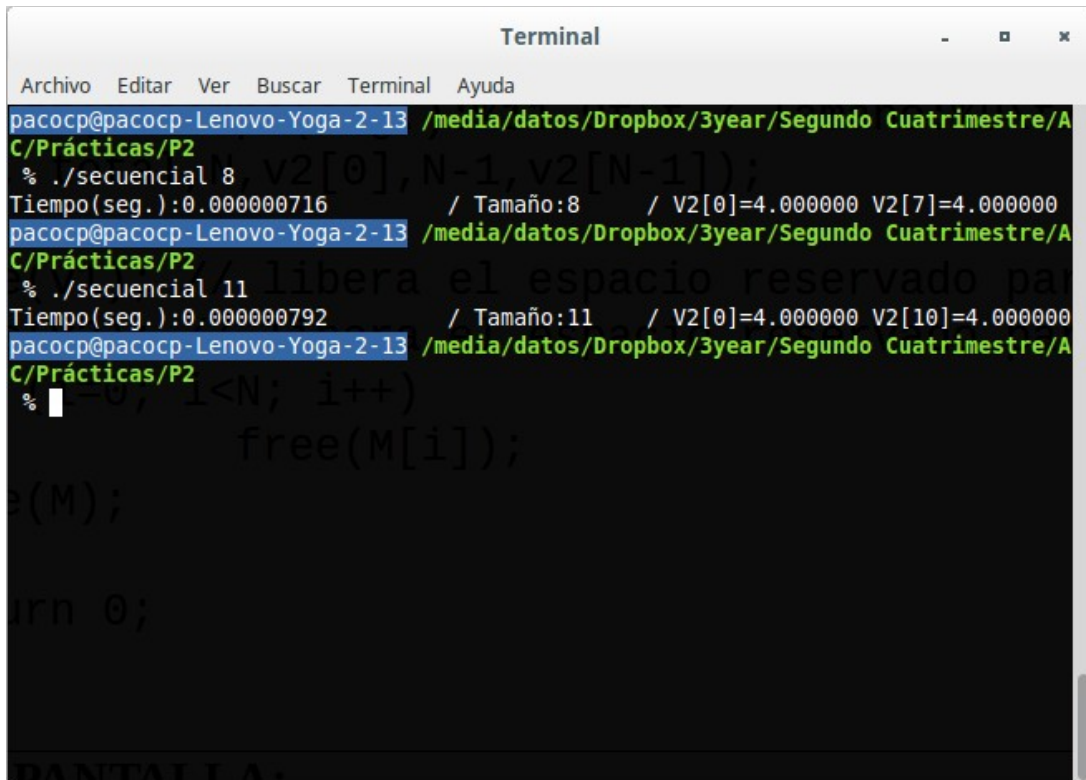
        //Imprimir el resultado y el tiempo de ejecución
        printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f
V2[%d]=%8.6f\n", total,N,v2[0],N-1,v2[N-1]);

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

        free(M);

        return 0;
}

```

CAPTURAS DE PANTALLA:

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

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

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

    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-
    1=4294967295 (sizeof(unsigned int) = 4 B)

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double)); // malloc necesita
    el tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay
    espacio suficiente malloc devuelve NULL
    M = (double**) malloc(N*sizeof(double *));
    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
        printf("Error en la reserva de espacio para
    los vectores\n");
        exit(-2);
    }

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
        if ( M[i]==NULL ){
            printf("Error en la reserva de
    espacio para los vectores\n");
            exit(-2);
        }
    }
    //A partir de aqui se pueden acceder las componentes de la
    matriz como M[i][j]
    #pragma omp parallel
    {
        //Inicializar matriz y vectores
        #pragma omp for private(j)
        for(i = 0; i< N; i++)
        {
            v1[i] = 2;
```

```

        v2[i] = 2;
        for(j = 0; j < N; j++ )
        {
            M[i][j] = 2;
        }
    }

    //Medida de tiempo
    #pragma omp single
    {
        t1 = omp_get_wtime();
    }

    //Calcular producto de matriz por vector v2 =
M · v1

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

        }
    }

    //Medida de tiempo
    #pragma omp single
    {
        t2 = omp_get_wtime();
    }

    }

    total = t2 - t1;

    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f
V2[%d]=%8.6f\n", total,N,v2[0],N-1,v2[N-1]);

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

    free(M);

    return 0;
}

```

CÓDIGO FUENTE: pmv-OpenMP-b.c

```

// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>

```

```

#include <omp.h>

int main(int argc, char** argv){
    int i, j;
    double t1, t2, total;
    int suma_parcial=0,suma=0;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-
    1=4294967295 (sizeof(unsigned int) = 4 B)

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double)); // malloc necesita
    el tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay
    espacio suficiente malloc devuelve NULL
    M = (double**) malloc(N*sizeof(double *));
    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
        printf("Error en la reserva de espacio para
    los vectores\n");
        exit(-2);
    }

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
        if ( M[i]==NULL ){
            printf("Error en la reserva de
    espacio para los vectores\n");
            exit(-2);
        }
    }
    //A partir de aqui se pueden acceder las componentes de la
    matriz como M[i][j]
    #pragma omp parallel private(j) private(i)
    private(suma_parcial)
    {
        //Inicializar matriz y vectores

        #pragma omp for
        for(i = 0; i< N; i++)
        {
            v1[i] = 2;
            v2[i] = 0;

            for(j = 0; j < N; j++ )
            {
                M[i][j] = 2;
            }
        }
    }

```

```

//Medida de tiempo
#pragma omp single
{
    t1 = omp_get_wtime();
}

//Calcular producto de matriz por vector v2 =
M . v1

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

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

}

//Medida de tiempo
#pragma omp single
{
    t2 = omp_get_wtime();
}

}

total = t2 - t1;

//Imprimir el resultado y el tiempo de ejecución
printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f
V2[%d]=%8.6f\n", total,N,v2[0],N-1,v2[N-1]);

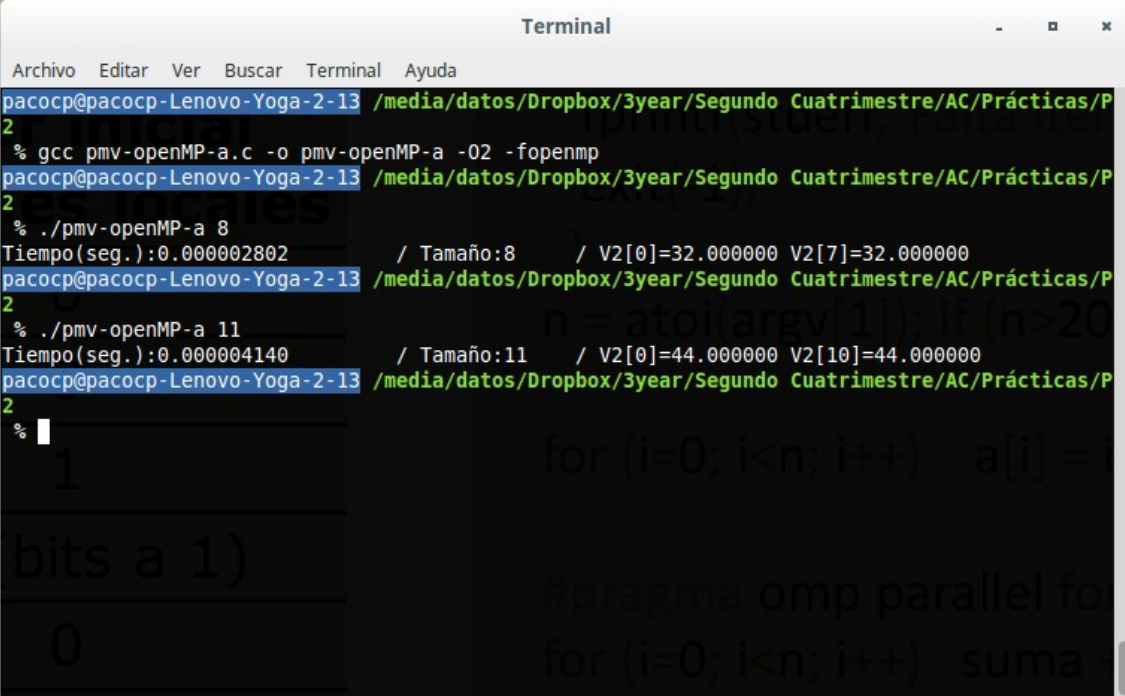
free(v1); // libera el espacio reservado para v1
free(v2); // libera el espacio reservado para v2
for (i=0; i<N; i++)
    free(M[i]);
free(M);

return 0;
}

```

RESPUESTA:

CAPTURAS DE PANTALLA:



```
Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P
2
% gcc pmv-openMP-a.c -o pmv-openMP-a -O2 -fopenmp
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P
2
% ./pmv-openMP-a 8
Tiempo(seg.):0.000002802 / Tamaño:8 / V2[0]=32.000000 V2[7]=32.000000
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P
2
% ./pmv-openMP-a 11
Tiempo(seg.):0.000004140 / Tamaño:11 / V2[0]=44.000000 V2[10]=44.000000
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P
2
%
1
bits a 1)
0
```

```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P2
% ./pmv-openMP-b 8
Tiempo(seg.):0.000011079 / Tamaño:8 / V2[0]=32.000000 V2[7]=32.000000
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P2
% ./pmv-openMP-b 11
Tiempo(seg.):0.000015115 / Tamaño:11 / V2[0]=44.000000 V2[10]=44.000000
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P2
%

```

10. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula `reduction`. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

CÓDIGO FUENTE: pmv-OpenMP-reduction.c

```

// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
    int i, j;
    double t1, t2, total;
    int suma_parcial=0,suma=0;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-
    1=4294967295 (sizeof(unsigned int) = 4 B)

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

```



```

el tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay
espacio suficiente malloc devuelve NULL
    M = (double**) malloc(N*sizeof(double *));
    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
        printf("Error en la reserva de espacio para
los vectores\n");
        exit(-2);
    }

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
        if ( M[i]==NULL ){
            printf("Error en la reserva de
espacio para los vectores\n");
            exit(-2);
        }
    }
    //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]
    #pragma omp parallel private(j) private(i,suma)
    {
        //Inicializar matriz y vectores

        #pragma omp for
        for(i = 0; i< N; i++)
        {
            v1[i] = 2;
            v2[i] = 0;

            for(j = 0; j < N; j++ )
            {
                M[i][j] = 2;
            }
        }

        //Medida de tiempo
        #pragma omp single
        {
            t1 = omp_get_wtime();
        }

        //Calcular producto de matriz por vector v2 =
M · v1

        for(i = 0; i< N; i++)
        {
            #pragma omp single
            {
                suma_parcial=0;
            }
            #pragma omp for
            reduction(+:suma_parcial)
            for(j=0; j<N;j++)

```

```

        {
            suma_parcial += M[i]
[j] * v1[j];

        }

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

    }

    //Medida de tiempo
#pragma omp single
{
    t2 = omp_get_wtime();
}

}

total = t2 - t1;

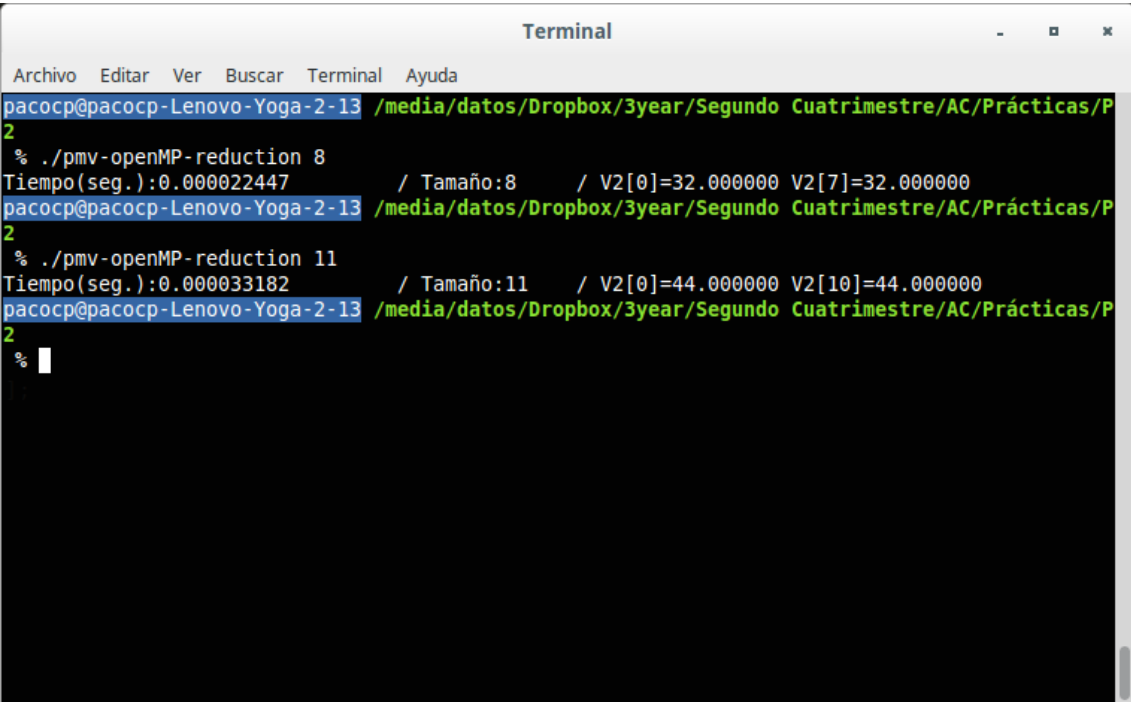
//Imprimir el resultado y el tiempo de ejecución
printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f
V2[%d]=%8.6f\n", total,N,v2[0],N-1,v2[N-1]);

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

return 0;
}

```

RESPUESTA:**CAPTURAS DE PANTALLA:**



```
Terminal
Archivo Editar Ver Buscar Terminal Ayuda
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P
2
% ./pmv-openMP-reduction 8
Tiempo(seg.):0.000022447 / Tamaño:8 / V2[0]=32.000000 V2[7]=32.000000
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P
2
% ./pmv-openMP-reduction 11
Tiempo(seg.):0.000033182 / Tamaño:11 / V2[0]=44.000000 V2[10]=44.000000
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P
2
% 
```

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: