

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos):

Grupo de prácticas:

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:

cuando ponemos `default(none)`, el compilador no sabe el ambito de dicha variable por lo que nos pide que la definamos

CAPTURA CÓDIGO FUENTE: `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] );
}
```

CAPTURAS DE PANTALLA:

```
$gcc -O2 -fopenmp -o shared-clausemodificado shared-clauseModificado.c
shared-clauseModificado.c:5:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:13:11: error: 'n' not specified in enclosing parallel
    #pragma omp parallel for shared(a) default(none)
    ^
shared-clauseModificado.c:13:11: error: enclosing parallel
```

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:

cuando inicializamos `suma` fuera de la region paralela, al entrar en cada hebra, esta tendrá una copia privada sin inicializar, por lo que no nos sirve, y al no inicialiozarla dentro de la region el valor `suma` es indeterminado al inicio.

SI la inicializamos dentro de la region paralela, cada copia de cada hebra tendra un valor concreto al iniciar la ejecucion, pr lo tanto podremos obtener un resultado correcto.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```
#include <stdio.h>

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

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

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

    // suma = 50;

    #pragma omp parallel private(suma)
    {
        suma = 50;

        printf("Valor inicial de suma: %d\n", suma);

        #pragma omp barrier

        #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 fuera de region paralela de suma: %d\n", suma);
}
```

CAPTURAS DE PANTALLA:

fuera
dentro

```

$./private-clause
Valor inicial de suma: 50
Valor inicial de suma: 50
Valor inicial de suma: 50
Valor inicial de suma: 50
thread 3 suma a[6] / thread 2 suma a[4] / thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma a[5] /
thread 1 suma a[2] / thread 1 suma a[3] /
* thread 3 suma= 56
* thread 2 suma= 59
* thread 0 suma= 51
* thread 1 suma= 55
Valor fuera de region paralela de suma: 0
Valor fuera de region paralela de suma: 50

```

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

RESPUESTA:

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```

#include <stdio.h>

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

int main(int argc, char const *argv[]) {
    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:

```

$./private-clause
thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma a[4] / thread 2 suma a[5] / thread 3 suma a[6] /
thread 1 suma a[2] / thread 1 suma a[3] /
* thread 3 suma= 15
* thread 0 suma= 15
* thread 2 suma= 15
* thread 1 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.

RESPUESTA:

Como se combinan el uso de `firstprivate` y `lastprivate` sobre `suma`, esto da lugar a que

las hebras tengan cada una una copia privada inicializada a 0, y también, hace que el último valor de suma de la región paralela sea el último que se guarda en un hipotético caso que se ejecutara secuencialmente.

CAPTURAS DE PANTALLA:

```
[IgnacioMorillasPadi@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$./firstlastprivate
thread 3 suma a[6] suma=6
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

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?

RESPUESTA:

Cada hebra tiene su propia variable propia `a` y solo se modificara su valor la primera hebra que entre a la directiva `single`, siendo el valor de todas las `a` indeterminado, excepto dicha hebra. Al usar `copyprivate(a)` se realiza una copia en el resto de `a` de las otras hebras.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

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

int main() {
    int n = 9, i, b[n];

    for (i=0; i<n; i++)    b[i] = -1;

    #pragma omp parallel
    {
        int a;
        #pragma omp single
        {
            printf("\nIntroduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("\nSingle ejecutada por el thread %d\n",
                omp_get_thread_num());
        }
        #pragma omp for
        for (i=0; i<n; i++) b[i] = a;
    }

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

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
$./copyprivate-clause
Introduce valor de inicialización a: 10

Single ejecutada por el thread 2
Después de la región parallel:
b[0] = 0      b[1] = 0      b[2] = 0      b[3] = 0      b[4] = 0      b[5] = 10      b[6] = 10
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

RESPUESTA:

el resultado con 10 iteraciones es 55, esto es por que cada hebra tiene una copia propia de suma inicializada a 0 y realiza su propia suma local cada hebra. Una vez finalizado, suman todas estas sumas locales.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

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

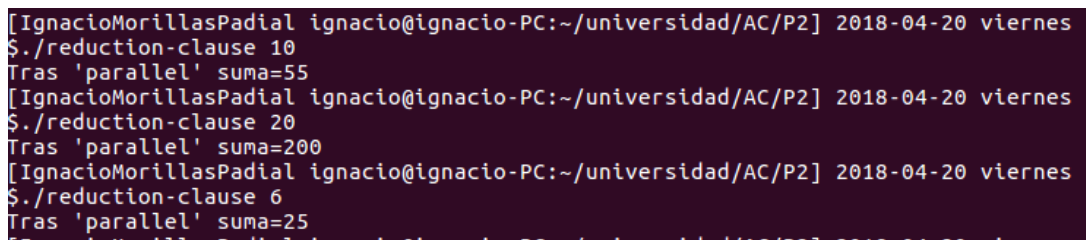
    if(argc < 2) {
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d",n);}

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

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

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

CAPTURAS DE PANTALLA:



```
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$./reduction-clause 10
Tras 'parallel' suma=55
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$./reduction-clause 20
Tras 'parallel' suma=200
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$./reduction-clause 6
Tras 'parallel' suma=25
```

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 usar directivas de trabajo compartido.

RESPUESTA:

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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

```

int main(int argc, char **argv) {
    int i, n=20, a[n], suma=0, suma_aux;

    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_aux)
    {
        suma_aux = 0;
        #pragma omp for
        for (int i = 0; i < n; i++)
            suma_aux += a[i];
        #pragma omp atomic
        suma += suma_aux;
    }

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

```

CAPTURAS DE PANTALLA:

```

[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$export OMP_NUM_THREADS=3
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$./reduction-clause 10
Tras 'parallel' suma=45
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$./reduction-clause 20
Tras 'parallel' suma=190
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$./reduction-clause 6
Tras 'parallel' suma=15

```

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).

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

```

/* pmv-secuencial.c
Producto de una matriz cuadrada por un vector
Para compilar usar (-lrt: real time library):
gcc -O2 pmv-secuencial.c -o pmv-secuencial -lrt
gcc -O2 -S pmv-secuencial.c -lrt //para generar el código ensamblador

```

```

Para ejecutar use: pmv-secuencial.c n_filas n_columnas
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y
free()
#include <stdio.h> // biblioteca donde se encuentra la función
printf()
#include <time.h> // biblioteca donde se encuentra la función
clock_gettime()
#define PRINTF_ALL// comentar para quitar el printf ...

// que imprime todos
los componentes
//Sólo puede estar definida una de las tres constantes VARIABLES_
(sólo uno de los ...
//tres defines siguientes puede estar descomentado):
// #define VARIABLES_LOCALES // descomentar para
que las variables sean locales ...

// locales (si se supera el tamaño de la pila
se ...

// generará el error "Violación de Segmento")
// #define VARIABLES_GLOBALES // descomentar para que las variables
sean globales ...

// globales (su longitud no estará limitada por el ...

// tamaño de la pila del programa)
#define VARIABLES_DINAMICAS // descomentar para que las variables sean
dinamicas ...
// dinámicas (memoria reutilizable durante la ejecución)
#ifdef VARIABLES_GLOBALES
#define MAX 8192 //2^13
double M[MAX][MAX], v1[MAX], v2[MAX];
#endif

int main(int argc, char** argv)
{
    int i;
    struct timespec cgt1,cgt2;
    double ncgt; //para tiempo de ejecución
    //Leer argumento de entrada (nº de componentes del vector)
    if (argc<2)
    {
        printf("Faltan filas y columnas\n");
        exit(-1);
    }

    unsigned int filas_columnas = atoi(argv[1]);
#ifdef VARIABLES_LOCALES
    double M[filas_columnas][filas_columnas],
    v1[filas_columnas], v2[filas_columnas]; // Tamaño variable local en

```

```

tiempo de ejecución ...
        // disponible en C a partir de actualización C99
#endif
#ifdef VARIABLES_GLOBALES
        if (filas_columnas>MAX) filas_columnas=MAX;
#endif
#ifdef VARIABLES_DINAMICAS
        double **M, *v1, *v2;
        M = (double **) malloc(filas_columnas*sizeof(double*));
        v1 = (double*) malloc(filas_columnas*sizeof(double)); //
        malloc necesita el tamaño en bytes
        v2 = (double*) malloc(filas_columnas*sizeof(double)); //si
        no hay espacio suficiente malloc devuelve NULL
        if ( (v1==NULL) || (v2==NULL) || (M==NULL)){
            printf("Error en la reserva de espacio para los
vectores\n");
            exit(-2);
        }
#endif

        //Inicialiar v1 y M
        for(i=0; i<filas_columnas; i++){
            v1[i] = filas_columnas*0.1-i*0.1; //los valores
dependen de filas_columnas
            v2[i] = 0.0;
            M[i] = (double*)
        malloc(filas_columnas*sizeof(double));
            for (int j = 0; j < filas_columnas; j++) {
                M[i][j] =
filas_columnas*0.1+i*0.1;
            }
        }

        clock_gettime(CLOCK_REALTIME,&cgt1);
        //Calcular producto de v1 por M, en v2
        for (int i = 0; i < filas_columnas; i++) {
            for (int j = 0; j < filas_columnas; j++) {
                v2[i] += M[i][j] * v1[j];
            }
        }
        clock_gettime(CLOCK_REALTIME,&cgt2);
        ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double)
        ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
        //Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
        printf("Tiempo(seg.):%11.9f\t Tamaño vectores y matriz:
%u\n",ncgt,filas_columnas);
        for(i=0; i<filas_columnas; i++)
            printf("V2[%d]=(%8.6f) \n", i,v2[i]);
#else
        printf("Tiempo(seg.):%11.9f\t Tamaño vectores y matriz:
%u\t V2[0]=(%8.6f) V2[%d]=(%8.6f) \n",
ncgt,filas_columnas,v2[0],filas_columnas-1,v2[filas_columnas-1]);
#endif
#ifdef VECTOR_DYNAMIC
        free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2

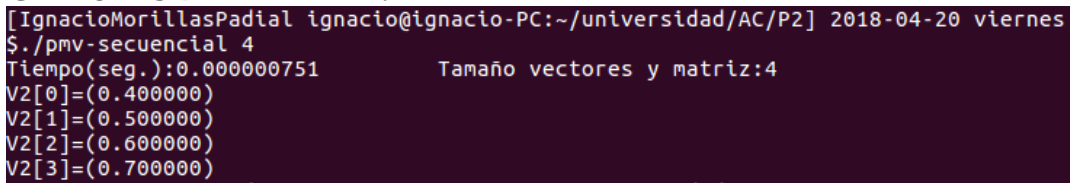
```



```

        free(*M); // libera el espacio reservado para v3
    #endif
        return 0;
}

```

CAPTURAS DE PANTALLA:


```

[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$./pmv-secuencial 4
Tiempo(seg.):0.000000751          Tamaño vectores y matriz:4
v2[0]=(0.400000)
v2[1]=(0.500000)
v2[2]=(0.600000)
v2[3]=(0.700000)

```

9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior usando la directiva `for`. Debe implementar dos versiones del código (consulte la lección 5/Tema 2):

- una primera que paralelice el bucle que recorre las filas de la matriz y
- una segunda que paralelice el bucle que recorre las columnas.

Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula `reduction`**. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:

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

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, `v3`, para tamaños pequeños de los vectores (por ejemplo, $N = 8$ y $N=11$); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CAPTURA CÓDIGO FUENTE : pmv-OpenMP-a.c**CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c****RESPUESTA:****CAPTURAS DE PANTALLA:**

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

```

/* pmv-OpenmMP-reduction.c
Producto de una matriz cuadrada por un vector
Para compilar usar (-lrt: real time library):
gcc -O2 pmv-OpenmMP-reduction.c -o pmv-OpenmMP-reduction -lrt -fopenmp
gcc -O2 -S pmv-OpenmMP-reduction.c -lrt -fopenmp //para generar el código
ensamblador
Para ejecutar use: pmv-OpenmMP-reduction.c n_filas n_columnas
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif
#define PRINTF_ALL // comentar para quitar el printf ...

// que imprime todos los
componentes
//Sólo puede estar definida una de las tres constantes VARIABLES_ (sólo uno de
los ...
//tres defines siguientes puede estar descomentado):
// #define VARIABLES_LOCALES // descomentar para que
las variables sean locales ...

// locales (si se supera el tamaño de la pila
se ...

// generará el error "Violación de Segmento")
// #define VARIABLES_GLOBALES // descomentar para que las variables sean
globales ...

// globales (su longitud no estará limitada por el ...

// tamaño de la pila del programa)
#define VARIABLES_DINAMICAS // descomentar para que las variables sean
dinamicas ...
// dinámicas (memoria reutilizable durante la ejecución)
#ifdef VARIABLES_GLOBALES
#define MAX 8192 //2^13
double M[MAX][MAX], v1[MAX], v2[MAX];
#endif

int main(int argc, char** argv)
{
    int i;
    struct timespec cgt1,cgt2;
    double ncgt; //para tiempo de ejecución
    //Leer argumento de entrada (nº de componentes del vector)
    if (argc<2)
    {
        printf("Faltan filas y columnas\n");
    }
}

```

```

        exit(-1);
    }

    unsigned int filas_columnas = atoi(argv[1]);
#ifdef VARIABLES_LOCALES
    double M[filas_columnas][filas_columnas], v1[filas_columnas],
v2[filas_columnas]; // Tamaño variable local en tiempo de ejecución ...
    // disponible en C a partir de actualización C99
#endif
#ifdef VARIABLES_GLOBALES
    if (filas_columnas>MAX) filas_columnas=MAX;
#endif
#ifdef VARIABLES_DINAMICAS
    double **M, *v1, *v2;
    M = (double **) malloc(filas_columnas*sizeof(double*));
    v1 = (double*) malloc(filas_columnas*sizeof(double)); // malloc
necesita el tamaño en bytes
    v2 = (double*) malloc(filas_columnas*sizeof(double)); //si no
hay espacio suficiente malloc devuelve NULL
    if ( (v1==NULL) || (v2==NULL) || (M==NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }
#endif
#ifdef _OPENMP
    //Inicialiar v1 y M
    for(i=0; i<filas_columnas; i++){
        v1[i] = filas_columnas*0.1-i*0.1;//los valores
dependen de filas_columnas
        v2[i] = 0.0;
        M[i] = (double*)
malloc(filas_columnas*sizeof(double));
        for (int j = 0; j < filas_columnas; j++) {
            M[i][j] = filas_columnas*0.1+i*0.1;
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt1);
    //Calcular producto de v1 por M, en v2
    for (int i = 0; i < filas_columnas; i++) {
        for (int j = 0; j < filas_columnas; j++) {
            v2[i] += M[i][j] * v1[j];
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
#else
    // Inicializar v1 y M
    #pragma omp parallel for
    for(i=0; i<filas_columnas; i++){
        v1[i] = filas_columnas*0.1-i*0.1;//los
valores dependen de filas_columnas
        v2[i] = 0.0;
        M[i] = (double*)
malloc(filas_columnas*sizeof(double));
        for (int j = 0; j < filas_columnas; j+
+) {
            M[i][j] =
filas_columnas*0.1+i*0.1;
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (int i = 0; i < filas_columnas; i++) {

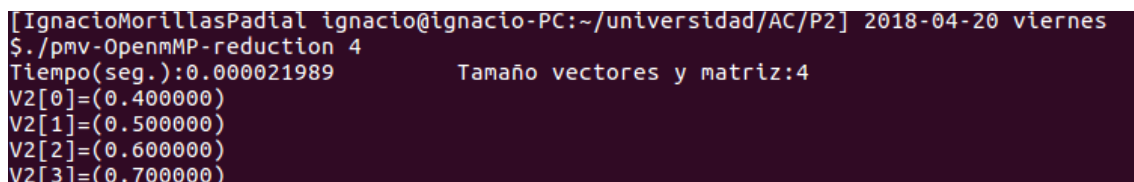
```

```

        double suma = 0.0;
        #pragma omp parallel for reduction(+:suma)
        for (int j = 0; j < filas_columnas; j++) {
            suma += M[i][j] * v1[j];
        }
        v2[i] = suma;
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
#endif

    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double)
    ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
    //Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f\t Tamaño vectores y matriz:
    %u\n",ncgt,filas_columnas);
    for(i=0; i<filas_columnas; i++)
        printf("V2[%d]=(%8.6f) \n", i,v2[i]);
#else
    printf("Tiempo(seg.):%11.9f\t Tamaño vectores y matriz:%u\t
    V2[0]=(%8.6f) V2[%d]=(%8.6f) \n", ncgt,filas_columnas,v2[0],filas_columnas-
    1,v2[filas_columnas-1]);
#endif
#ifdef VECTOR_DYNAMIC
    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
    free(*M); // libera el espacio reservado para v3
#endif
    return 0;
}

```

RESPUESTA:**CAPTURAS DE PANTALLA:**


```

[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P2] 2018-04-20 viernes
$ ./pmv-OpenMP-reduction 4
Tiempo(seg.):0.000021989          Tamaño vectores y matriz:4
V2[0]=(0.400000)
V2[1]=(0.500000)
V2[2]=(0.600000)
V2[3]=(0.700000)

```

11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar -O2 al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

CAPTURAS DE PANTALLA (que justifique el código elegido):

TABLA Y GRÁFICA (por ejemplo para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N:- un N entre 30000 y 100000, y otro entre 5000 y 30000):

COMENTARIOS SOBRE LOS RESULTADOS: