

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

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 3. Programación paralela III: Interacción con el entorno en 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. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad de esta cláusula y explicar por qué lo ilustran.

CÓDIGO FUENTE: `if-clauseModificado.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv)
{
    int i, n=20, tid,num_threads;
    int a[n],suma=0,sumalocal;
    if(argc < 3) {
        fprintf(stderr,"[ERROR]-Falta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;
    num_threads = atoi(argv[2]);
    for (i=0; i<n; i++) {
        a[i] = i;
    }

    #pragma omp parallel if(n>4) default(none) \
        private(sumalocal,tid) shared(a,suma,n)
    num_threads(num_threads)
    {
        sumalocal=0;
        tid=omp_get_thread_num();
        #pragma omp for private(i) schedule(static)
        nowait
        for (i=0; i<n; i++)
        {
            sumalocal += a[i];
            printf(" thread %d suma de a[%d]=
            %d sumalocal=%d \n",
            tid,i,a[i],sumalocal);
```

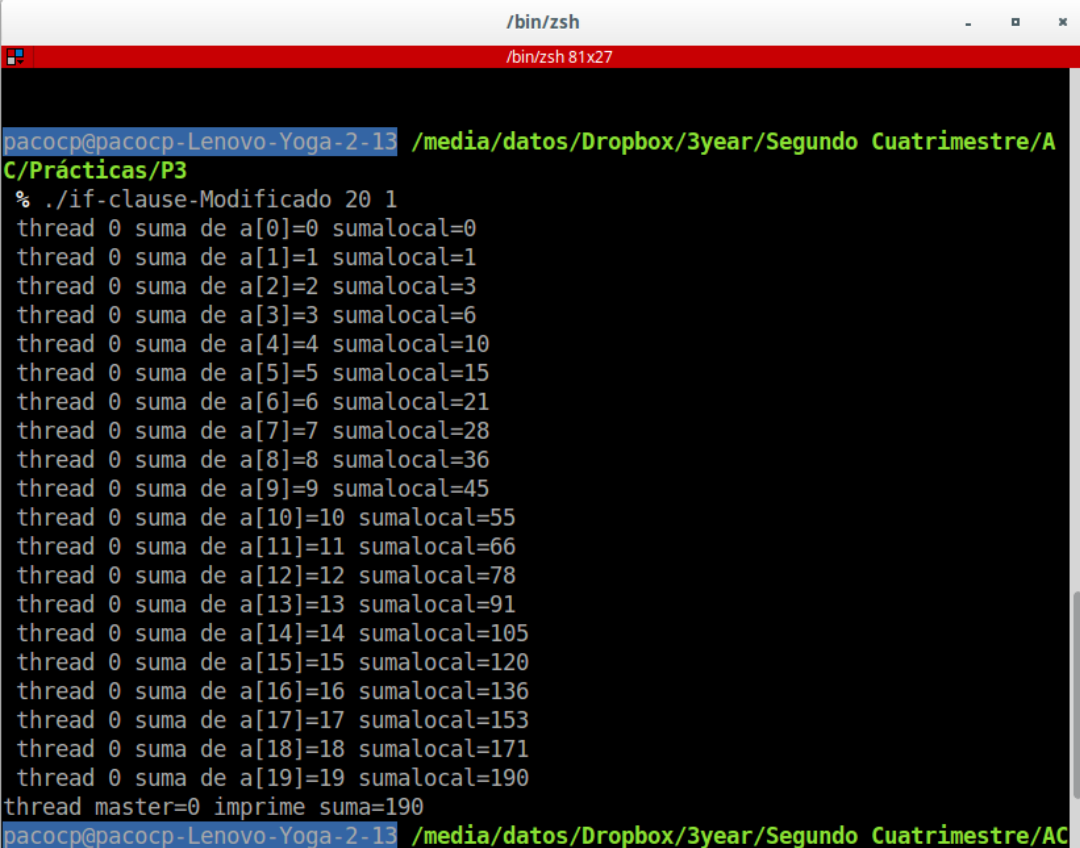
```
    }

    #pragma omp atomic
    suma += sumalocal;

    #pragma omp barrier

    #pragma omp master
    printf("thread master=%d imprime suma=
%d\n", tid, suma);
}
}
```

CAPTURAS DE PANTALLA:



```
/bin/zsh
/bin/zsh 81x27

pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3
% ./if-clause-Modificado 20 1
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 0 suma de a[2]=2 sumalocal=3
thread 0 suma de a[3]=3 sumalocal=6
thread 0 suma de a[4]=4 sumalocal=10
thread 0 suma de a[5]=5 sumalocal=15
thread 0 suma de a[6]=6 sumalocal=21
thread 0 suma de a[7]=7 sumalocal=28
thread 0 suma de a[8]=8 sumalocal=36
thread 0 suma de a[9]=9 sumalocal=45
thread 0 suma de a[10]=10 sumalocal=55
thread 0 suma de a[11]=11 sumalocal=66
thread 0 suma de a[12]=12 sumalocal=78
thread 0 suma de a[13]=13 sumalocal=91
thread 0 suma de a[14]=14 sumalocal=105
thread 0 suma de a[15]=15 sumalocal=120
thread 0 suma de a[16]=16 sumalocal=136
thread 0 suma de a[17]=17 sumalocal=153
thread 0 suma de a[18]=18 sumalocal=171
thread 0 suma de a[19]=19 sumalocal=190
thread master=0 imprime suma=190
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC
```

```

/bin/zsh
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC
/Prácticas/P3
% ./if-clause-Modificado 20 2
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 0 suma de a[2]=2 sumalocal=3
thread 0 suma de a[3]=3 sumalocal=6
thread 0 suma de a[4]=4 sumalocal=10
thread 0 suma de a[5]=5 sumalocal=15
thread 0 suma de a[6]=6 sumalocal=21
thread 0 suma de a[7]=7 sumalocal=28
thread 0 suma de a[8]=8 sumalocal=36
thread 0 suma de a[9]=9 sumalocal=45
thread 1 suma de a[10]=10 sumalocal=10
thread 1 suma de a[11]=11 sumalocal=21
thread 1 suma de a[12]=12 sumalocal=33
thread 1 suma de a[13]=13 sumalocal=46
thread 1 suma de a[14]=14 sumalocal=60
thread 1 suma de a[15]=15 sumalocal=75
thread 1 suma de a[16]=16 sumalocal=91
thread 1 suma de a[17]=17 sumalocal=108
thread 1 suma de a[18]=18 sumalocal=126
thread 1 suma de a[19]=19 sumalocal=145
thread master=0 imprime suma=190
pacocp@pacocp-Lenovo-Yoga-2-13 /media/datos/Dropbox/3year/Segundo Cuatrimestre/AC
/Prácticas/P3
%

```

RESPUESTA:

En el primero podemos observar como al ponerle un solo thread las iteraciones solo las ejecutan el thread master, es decir, el thread 0.

En el segundo, podemos observar como al ponerle 2 threads, se reparten las iteraciones entre el thread 0 y el thread 1.

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario `schedule-clause.c`, `scheduled-clause.c` y `scheduleg-clause.c` con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,...15)
- chunk= 1, 2 y 4

Tabla 1 . Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	1	0	0	0	0
1	1	0	0	1	1	0	0	0	0
2	0	1	0	0	0	0	0	0	0

3	1	1	0	0	0	0	0	0	0
4	0	0	1	0	1	1	0	0	0
5	1	0	1	0	1	1	0	0	0
6	0	1	1	0	1	1	0	0	0
7	1	1	1	0	1	1	0	0	0
8	0	0	0	0	1	0	1	1	1
9	1	0	0	0	1	0	1	1	1
10	0	1	0	0	1	0	1	1	1
11	1	1	0	0	1	0	1	1	1
12	0	0	1	0	1	0	0	0	1
13	1	0	1	0	1	0	0	0	1
14	0	1	1	0	1	0	0	0	1
15	1	1	1	0	1	0	0	0	1

(b) Rellenar otra tabla como la de la figura pero esta vez usando cuatro *threads* (0,1,2,3).

Tabla 2 . Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	1	1	2	2	2	2
1	1	0	0	2	1	2	2	2	2
2	2	1	0	0	3	2	2	2	2
3	3	1	0	3	3	2	2	2	2
4	0	2	1	1	2	3	0	0	1
5	1	2	1	1	2	3	0	0	1
6	2	3	1	1	0	3	0	0	1
7	3	3	1	1	0	3	3	1	1
8	0	0	2	1	0	0	3	1	3
9	1	0	2	1	0	0	3	1	3
10	2	1	2	1	1	0	1	3	3
11	3	1	2	1	1	0	1	3	3
12	0	2	3	0	1	1	1	0	0
13	1	2	3	0	1	1	1	0	0

14	2	3	3	0	1	1	1	0	0
15	3	3	3	0	1	1	1	0	0

Escriba en el cuaderno de prácticas las diferencias en el comportamiento de `schedule()` con `static`, `dynamic` y `guided`.

RESPUESTA:

La principal diferencia se ve en el reparto de las iteraciones: con `static` todas las hebras hacen prácticamente las mismas iteraciones en round-robin. Con `dynamic` las el orden y el reparto no se puede saber, lo que si se sabe es que todas las hebras harán como mínimo un número de iteraciones igual al chunk. Lo mismo pasa con `guided` con la excepción de que de que las iteraciones están más “equilibradas” entre las hebras y el número de iteraciones que hace cada hebra no es múltiplo del chunk.

3. Añadir al programa `scheduled-clause.c` lo necesario para que imprima el valor de las variables de control `dyn-var`, `nthreads-var`, `thread-limit-var` y `run-sched-var` dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorporar en su cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

CÓDIGO FUENTE: `scheduled-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 = 16, chunk, a[n], suma=0;
    int modifier;
    omp_sched_t kind;
    if(argc < 2) {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);
    for (i=0; i<n; i++) a[i] = i;
    #pragma omp parallel
    {
        #pragma omp for firstprivate(suma) \
```

```

                                lastprivate(suma) schedule(dynamic,chunk)
                                for (i=0; i<n; i++)
                                {

                                        suma = suma + a[i];
                                        printf(" thread %d suma a[%d] suma=
%d \n",

                                        omp_get_thread_num(),i,suma);

                                }
                                #pragma omp single
                                {

                                        printf("dyn-var: %d
\n",omp_get_dynamic());

                                        printf("nthreads_var: %d
\n",omp_get_max_threads());

                                        printf("thread_limit_var: %d
\n",omp_get_thread_limit());

                                        printf("thread_limit_var: %d
\n",omp_get_thread_limit());

                                        omp_get_schedule(&kind, &modifier);
                                        printf("get_schedulre: kind
%d,modifier %d \n",kind,modifier);

                                }

                                }
                                printf("Fuera de 'parallel for' suma=%d\n",suma);
                                printf("Fuera de la región paralela: \n");
                                printf("dyn-var: %d \n",omp_get_dynamic());
                                printf("nthreads_var: %d \n",omp_get_max_threads());
                                printf("thread_limit_var: %d \n",omp_get_thread_limit());
                                printf("thread_limit_var: %d \n",omp_get_thread_limit());
                                omp_get_schedule(&kind, &modifier);
                                printf("get_schedulre: kind %d,modifier %d \n",kind,modifier);
}

```

CAPTURAS DE PANTALLA:**RESPUESTA:**

4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

CÓDIGO FUENTE: `scheduled-clauseModificado4.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 = 16, chunk, a[n], suma=0;
    int modifier;
    omp_sched_t kind;
    if(argc < 2) {

```

```

        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);
    for (i=0; i<n; i++) a[i] = i;
    #pragma omp parallel
    {

        #pragma omp for firstprivate(suma) \
        lastprivate(suma) schedule(dynamic, chunk)
        for (i=0; i<n; i++)
        {

            suma = suma + a[i];
            printf(" thread %d suma a[%d] suma=%
%d \n",

                omp_get_thread_num(), i, suma);

        }
        #pragma omp single
        {

            printf("omp_get_num_threads: %d
\n", omp_get_num_threads());

            printf("omp_get_num_procs: %d
\n", omp_get_num_procs());

            printf("omp_in_parallel: %d
\n", omp_in_parallel());

        }

    }

    printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("Fuera de la región paralela: \n");
    printf("omp_get_num_threads: %d \n", omp_get_num_threads());
    printf("omp_get_num_procs: %d \n", omp_get_num_procs());
    printf("omp_in_parallel: %d", omp_in_parallel());
}

```

CAPTURAS DE PANTALLA:

```

/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:ma
ster*) » ./scheduled-clauseModificado4 5 pacocp@pacocp-Lenovo-Yoga-2-13
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 0 suma a[3] suma=6
thread 0 suma a[4] suma=10
thread 3 suma a[15] suma=15
thread 2 suma a[10] suma=10
thread 2 suma a[11] suma=21
thread 2 suma a[12] suma=33
thread 2 suma a[13] suma=46
thread 2 suma a[14] suma=60
thread 1 suma a[5] suma=5
thread 1 suma a[6] suma=11
thread 1 suma a[7] suma=18
thread 1 suma a[8] suma=26
thread 1 suma a[9] suma=35
omp_get_num_threads: 4
omp_get_num_procs: 4
omp_in_parallel: 1
Fuera de 'parallel for' suma=15
Fuera de la región paralela:
omp_get_num_threads: 1
omp_get_num_procs: 4
omp_in_parallel: 0
/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) »

```

P3: zsh

RESPUESTA:

Las variables que cambian son `omp_get_num_threads`, ya que en la región paralela tenemos 4 threads creados mientras que en la región secuencial hay solo 1. Y también cambia `omp_in_parallel`, ya que en la región paralela nos devuelve un 1 indicando que si se encuentra en una región paralela, mientras que fuera nos devuelve un 0.

5. Añadir al programa `scheduled-clause.c` lo necesario para modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

CÓDIGO FUENTE: `scheduled-clauseModificado5.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 = 16, chunk, a[n], suma=0;
    int modifier;
    omp_sched_t kind;
    if(argc < 2) {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);
    for (i=0; i<n; i++) a[i] = i;

    printf("omp_get_dynamic: %d \n", omp_get_dynamic());
    printf("omp_get_max_threads: %d \n", omp_get_max_threads());
    omp_get_schedule(&kind, &modifier);
    printf("omp_get_schedule: kind %d, modifier %d\n", kind, modifier);
    #pragma omp parallel
    {
        #pragma omp single
        {
            omp_set_dynamic(1);
            omp_set_num_threads(4);
            omp_set_schedule(2, chunk);
            printf("omp_get_dynamic: %d\n", omp_get_dynamic());
            printf("omp_get_max_threads: %d\n", omp_get_max_threads());
            omp_get_schedule(&kind, &modifier);
            printf("omp_get_schedule: kind %d, modifier %d \n", kind, modifier);
        }
        #pragma omp for firstprivate(suma) \
lastprivate(suma) /*schedule(dynamic, chunk)*/
    }
}
```



```

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

                                        suma = suma + a[i];
                                        printf(" thread %d suma a[%d] suma=
%d \n",

                                        omp_get_thread_num(),i,suma);

                                }

                                }
                                printf("Fuera de 'parallel for' suma=%d\n",suma);
                                }

```

CAPTURAS DE PANTALLA:

```

Archivo  Editar  Ver  Bookmarks  Preferencias  Ayuda
/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » ./scheduled-clauseModi
ficado5 5
omp_get_dynamic: 0
omp_get_max_threads: 4
omp_get_schedule: kind 2, modifier 1
omp_get_dynamic: 1
omp_get_max_threads: 4
omp_get_schedule: kind 2, modifier 5
thread 3 suma a[12] suma=12
thread 2 suma a[8] suma=8
thread 2 suma a[9] suma=17
thread 2 suma a[10] suma=27
thread 2 suma a[11] suma=38
thread 1 suma a[4] suma=4
thread 1 suma a[5] suma=9
thread 1 suma a[6] suma=15
thread 1 suma a[7] suma=22
thread 3 suma a[13] suma=25
thread 3 suma a[14] suma=39
thread 3 suma a[15] suma=54
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
Fuera de 'parallel for' suma=54
/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » █
P3 : zsh

```

RESPUESTA:

Resto de ejercicios

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por vector.

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

CÓDIGO FUENTE: pmtv-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 = i; 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=i; 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: (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```

Archivo  Editar  Ver  Bookmarks  Preferencias  Ayuda
/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » ./pmtv-secuencial 8
Tiempo(seg.):0.000000361          / Tamaño:8          / V2[0]=4.000000 V2[7]=4.000000
-----
/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » ./pmtv-secuencial 11
Tiempo(seg.):0.000000818          / Tamaño:11         / V2[0]=4.000000 V2[10]=4.000000
-----
/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » █

I

P3 : zsh

```

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Obtener en `atcgrid` los tiempos de ejecución del código paralelo (usando, como siempre, `-O2` al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 1, 64 y el `chunk` por defecto para la alternativa. Use un tamaño de vector `N` múltiplo del número de cores y de 64 que no sea inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en `atcgrid` código que imprima todos los componentes del resultado.

Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para `chunk` con `static`, `dynamic` y `guided`? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación `static` para cada uno de los chunks? (c) Con la asignación `dynamic` y `guided`, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

RESPUESTA:

CÓDIGO FUENTE: `pmtv-OpenMP.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)
    int num_threads = atoi(argv[2]); //Número de threads
    int modifier;
    omp_sched_t kind;
    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] = 0;
            for(j = i; j < N; j++ )
            {
                M[i][j] = 2;
            }
        }
        #pragma omp parallel
        {
            #pragma omp single
            {
                omp_get_schedule(&kind,
&modifier);

                printf("omp_get_schedule: kind %d, modifier %d
\n",kind,modifier);

                omp_set_num_threads(num_threads);
            }

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

            #pragma omp for private(j) schedule(runtime)
            //Calcular producto de matriz por vector v2 = M .
v1
            for(i = 0; i<N;i++)
            {
                for(j=i; j<N;j++)
                {
                    v2[i] += M[i][j] *
v1[j];

                }
            }
        }
    }
}

```

```

        #pragma omp single
        {
            //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;
}

```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA: (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```

/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » ./Script-ejercicio7-local.sh pacocp@pacocp-Lenovo-Yoga-2-13
----Static
defecto
omp_get_schedule: kind 1, modifier 0
Tiempo(seg.):0.073145474 / Tamaño:15360 / V2[0]=61440.000000 V2[15359]=4.000000
chunk 1
omp_get_schedule: kind 1, modifier 1
Tiempo(seg.):0.064164348 / Tamaño:15360 / V2[0]=61440.000000 V2[15359]=4.000000
chunk 64
omp_get_schedule: kind 1, modifier 64
Tiempo(seg.):0.065373095 / Tamaño:15360 / V2[0]=61440.000000 V2[15359]=4.000000
----Dynamic
defecto
omp_get_schedule: kind 2, modifier 1
Tiempo(seg.):0.070534368 / Tamaño:15360 / V2[0]=61440.000000 V2[15359]=4.000000
chunk 1
omp_get_schedule: kind 2, modifier 1
Tiempo(seg.):0.070549957 / Tamaño:15360 / V2[0]=61440.000000 V2[15359]=4.000000
chunk 64
omp_get_schedule: kind 2, modifier 64
Tiempo(seg.):0.062756379 / Tamaño:15360 / V2[0]=61440.000000 V2[15359]=4.000000
----Guided
defecto
omp_get_schedule: kind 3, modifier 1
Tiempo(seg.):0.087792763 / Tamaño:15360 / V2[0]=61440.000000 V2[15359]=4.000000
chunk 1
omp_get_schedule: kind 3, modifier 1
Tiempo(seg.):0.093035052 / Tamaño:15360 / V2[0]=61440.000000 V2[15359]=4.000000
chunk 64
omp_get_schedule: kind 3, modifier 64
Tiempo(seg.):0.084687160 / Tamaño:15360 / V2[0]=61440.000000 V2[15359]=4.000000
/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) »
pacocp@pacocp-Lenovo-Yoga-2-13
P3: zsh

```

TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmvt-OpenMP_atcgrid.sh

```

#!/bin/bash
10. #PBS -q ac
11.
12.
13. echo "----Static"
14.
15. echo "defecto"
16. export OMP_SCHEDULE="static"

```

```

17. . $PBS_O_WORKDIR/pmtv-OPENMP 15360 12
18.
19. echo "chunk 1"
20. export OMP_SCHEDULE="static,1"
21. . $PBS_O_WORKDIR/pmtv-OPENMP 15360 12
22.
23. echo "chunk 64"
24. export OMP_SCHEDULE="static,64"
25. $PBS_O_WORKDIR/pmtv-OPENMP 15360 12
26.
27. echo "----Dynamic"
28.
29. echo "defecto"
30. export OMP_SCHEDULE="dynamic"
31. $PBS_O_WORKDIR/pmtv-OPENMP 15360 12
32.
33. echo "chunk 1"
34. export OMP_SCHEDULE="dynamic,1"
35. $PBS_O_WORKDIR/pmtv-OPENMP 15360 12
36.
37. echo "chunk 64"
38. export OMP_SCHEDULE="dynamic,64"
39. $PBS_O_WORKDIR/pmtv-OPENMP 15360 12
40.
41. echo "----Guided"
42.
43. echo "defecto"
44. export OMP_SCHEDULE="guided"
45. $PBS_O_WORKDIR/pmtv-OPENMP 15360 12
46.
47. echo "chunk 1"
48. export OMP_SCHEDULE="guided,1"
49. $PBS_O_WORKDIR/pmtv-OPENMP 15360 12
50.
51. echo "chunk 64"
52. export OMP_SCHEDULE="guided,64"
53. $PBS_O_WORKDIR/pmtv-OPENMP 15360 12
54.

```

Tabla 3. Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño $N= 15360$, 12 threads

Chunk	Static	Dynamic	Guided
por defecto		0.073123300	0.064146855
1		0.073465009	0.065221889
64	0.064271288	0.064011190	0.064171741

Tabla 4. Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño $N= 15360$, 4 threads

Chunk	Static	Dynamic	Guided
por defecto	0.086122381	0.070031261	0.089900205
1	0.085625874	0.070111520	0.085441779
64	0.083815075	0.072889373	0.087763950

8. Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

$$A = B \bullet C; A(i, j) = \sum_{k=0}^{N-1} B(i, k) \bullet C(k, j), i, j = 0, \dots, N-1$$

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CÓDIGO FUENTE: pmm-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, k;
    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 **M1, **M2, **M3;
    M1 = (double**) malloc(N*sizeof(double *));
    M2 = (double**) malloc(N*sizeof(double *));
    M3 = (double**) malloc(N*sizeof(double *));

    for (i=0; i<N; i++){
        M1[i] = (double*) malloc(N*sizeof(double));
        M2[i] = (double*) malloc(N*sizeof(double));
        M3[i] = (double*) malloc(N*sizeof(double));
        if ( M1[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++)
    {
        for(j = 0; j < N; j++ )
        {
            M1[i][j] = 2;
            M2[i][j] = 2;
```



```

        M3[i][j] = 0;
    }

}

//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++) {
    for (k=0; k<N; k++) {
        M3[i][j] += M1[i][k] * M2[k][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,M3[0][0],N-1,M3[N-1][N-1]);

    for (i=0; i<N; i++){
        free(M1[i]);
        free(M2[i]);
        free(M3[i]);
    }
    free(M1);
    free(M2);
    free(M3);

    return 0;
}

```

CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```

Archivo  Editar  Ver  Bookmarks  Preferencias  Ayuda

/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » ./pmm-secuencial 8
Tiempo(seg.):0.000001156 / Tamaño:8 / V2[0]=32.000000 V2[7]=32.000000

/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » ./pmm-secuencial 11
Tiempo(seg.):0.000002075 / Tamaño:11 / V2[0]=44.000000 V2[10]=44.000000

/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » 
P3 : zsh

```

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2, Lección 5/Tema 2).

DESCOMPOSICIÓN DE DOMINIO:

CÓDIGO FUENTE: pmm-OpenMP.c

```

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

int main(int argc, char** argv){
    int i, j, k;
    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 **M1, **M2, **M3;
    M1 = (double**) malloc(N*sizeof(double *));
    M2 = (double**) malloc(N*sizeof(double *));
    M3 = (double**) malloc(N*sizeof(double *));

```

```

        for (i=0; i<N; i++){
            M1[i] = (double*) malloc(N*sizeof(double));
            M2[i] = (double*) malloc(N*sizeof(double));
            M3[i] = (double*) malloc(N*sizeof(double));
            if ( M1[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

#pragma omp parallel
{
    #pragma omp private(j)
    for(i = 0; i< N; i++)
    {
        for(j = 0; j < N; j++ )
        {
            M1[i][j] = 2;
            M2[i][j] = 2;
            M3[i][j] = 0;
        }
    }

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

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

        #pragma omp for private(j,k)
        for (i=0; i<N; i++){
            for (j=0; j<N; j++) {
                for (k=0; k<N; k++) {
                    M3[i][j] += M1[i][k] * M2[k][j];
                }
            }
        }

        #pragma omp single
        {
            //Medida de tiempo
            t2 = omp_get_wtime();
            total = t2 - t1;
        }
    }

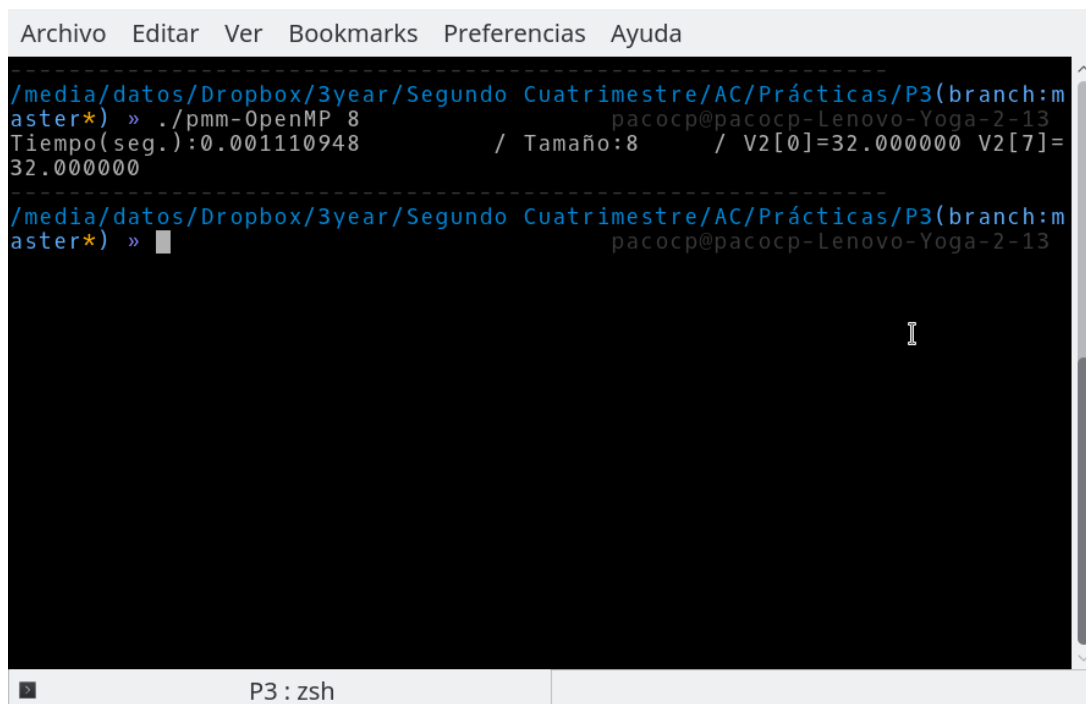
    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=
%8.6f\n", total,N,M3[0][0],N-1,M3[N-1][N-1]);

```

```
        for (i=0; i<N; i++){
            free(M1[i]);
            free(M2[i]);
            free(M3[i]);
        }
        free(M1);
        free(M2);
        free(M3);

        return 0;
}
```

**CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**



```
Archivo  Editar  Ver  Bookmarks  Preferencias  Ayuda
-----
/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) » ./pmm-OpenMP 8
Tamaño:8 / V2[0]=32.000000 V2[7]=32.000000
-----
/media/datos/Dropbox/3year/Segundo Cuatrimestre/AC/Prácticas/P3(branch:master*) »
-----
pacocp@pacocp-Lenovo-Yoga-2-13
-----
P3 : zsh
```

55. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar -O2 al compilar. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas (pruebe con valores de N entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

ESTUDIO DE ESCALABILIDAD EN ATCGRID:

SCRIPT: pmm-OpenMP_atcgrid.sh

```
#!/bin/bash
56.
57. #Se asigna al trabajo la cola ac
58. #PBS -q ac
59.
60. for ((N=100;N<1000;N=N+100))
61. do
62.     echo "S"
63.     $PBS_O_WORKDIR/pmm-secuencial $N
64.     echo "----- \n"
65.     echo "P"
66.     $PBS_O_WORKDIR/pmm-OpenMP $N
67. done
```

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pcllocal.sh

```
#!/bin/bash
68.
69. for ((N=100;N<1000;N=N+100))
70. do
71.     echo "S"
72.     ./pmm-secuencial $N
73.     echo "----- \n"
74.     echo "P"
75.     ./pmm-OpenMP $N
76. done
77.
```

