

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): Antonio Miguel Morillo Chica

Grupo de prácticas: D1

Fecha de entrega: 11/05/2016

Fecha evaluación en clase: 12/05/2016

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_thread;
    int a[n], suma=0, sumalocal;

    if(argc<3){
        fprintf(stderr, "[ERROR]-Falta iteraciones\n");
        exit(-1);
    }

    n=atoi(argv[1]);
    num_thread=atoi(argv[2]);
    omp_set_num_threads(num_thread);

    if(n>20)
        n=20;

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

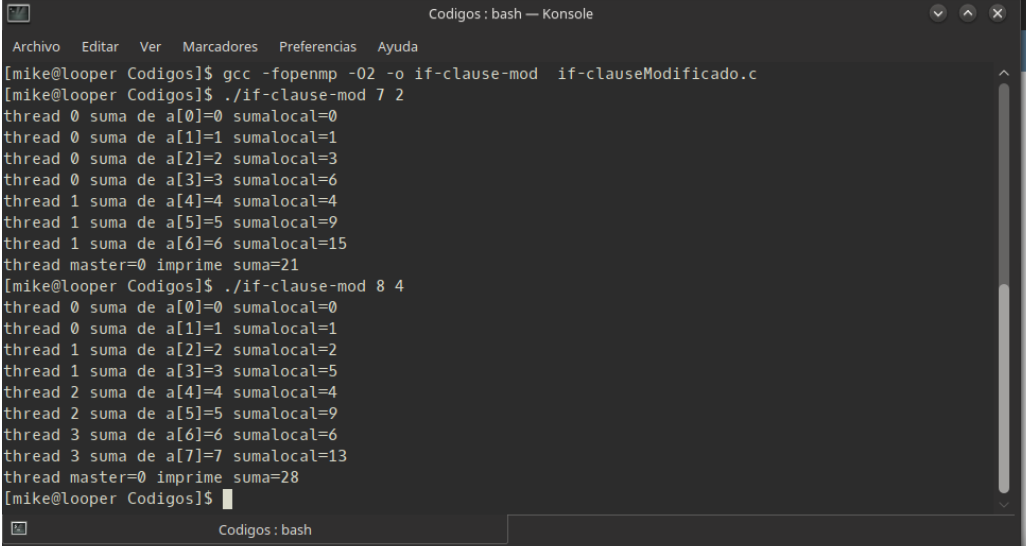
    #pragma omp parallel if(n>4) default(none) private(sumalocal, tid)
    shared(a,suma,n)
    {
        sumalocal=0;
        tid=omp_get_thread_num();

        #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:



```
Codigos : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[mike@looper Codigos]$ gcc -fopenmp -O2 -o if-clause-mod  if-clauseModificado.c
[mike@looper Codigos]$ ./if-clause-mod 7 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 1 suma de a[4]=4 sumalocal=4
thread 1 suma de a[5]=5 sumalocal=9
thread 1 suma de a[6]=6 sumalocal=15
thread master=0 imprime suma=21
[mike@looper Codigos]$ ./if-clause-mod 8 4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread 2 suma de a[4]=4 sumalocal=4
thread 2 suma de a[5]=5 sumalocal=9
thread 3 suma de a[6]=6 sumalocal=6
thread 3 suma de a[7]=7 sumalocal=13
thread master=0 imprime suma=28
[mike@looper Codigos]$
```

RESPUESTA: Depende del valor que le demos a la variable *x* obtendremos diferentes resultados, ya que *x* será el número de hebras de la región paralela.

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	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	0	1	0	1	1	0	0	0	0
3	1	1	0	1	1	0	0	0	0
4	0	0	1	1	0	1	0	0	0
5	1	0	1	1	0	1	0	0	0
6	0	1	1	1	0	1	0	0	0
7	1	1	1	1	0	1	0	0	0
8	0	0	0	1	1	0	1	1	1
9	1	0	0	1	1	0	1	1	1
10	0	1	0	1	1	0	1	1	1
11	1	1	0	1	1	0	1	1	1
12	0	0	1	1	1	0	0	0	0
13	1	0	1	1	1	0	0	0	0
14	0	1	1	1	1	0	0	0	0
15	1	1	1	1	1	0	0	0	0

(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	2	1	2	2	1	2
1	1	0	0	0	1	2	2	1	2
2	2	1	0	3	2	2	2	1	2
3	3	1	0	1	2	2	2	1	2
4	0	2	1	2	0	0	1	2	3
5	1	2	1	2	0	0	1	2	3
6	2	3	1	2	3	0	1	2	3
7	3	3	1	2	3	0	0	0	3
8	0	0	2	2	0	3	0	0	1
9	1	0	2	2	0	3	0	0	1
10	2	1	2	2	0	3	3	3	1
11	3	1	2	2	0	3	3	3	1
12	0	2	3	2	0	1	0	0	0
13	1	2	3	2	0	1	0	0	0
14	2	3	3	2	0	1	0	0	0
15	3	3	3	2	0	1	0	0	0

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

RESPUESTA: Static se asigna las threads mediante round-robin, es decir, si chunk es 1 se alterna cada threads. Dynamic asigna los threads en tiempo de ejecución, es decir, se asigna el trabajo a los siguientes threads libres. Guided asigna una gran carga al principio y poco a poco se decremента esa carga de asignación.

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

main(int argc, char **argv) {
    int i, n = 12, chunk, a[n], suma=0;
    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(static,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 master
        {
            printf("Dentro de la region paralela\n");
            printf("thread padre --> %d ==>\n", omp_get_thread_num());
            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());
            omp_get_schedule(&kind, &chunk);
            printf("run-sched-var ==> kind=%d y chunk=%d\n", kind, chunk);
        }
    }
    printf("/-----/\nFuera de 'parallel for' suma=%d\n", suma);
    printf("thread padre --> %d ==>\n", omp_get_thread_num());
    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());
    omp_get_schedule(&kind, &chunk);
    printf("run-sched-var ==> kind=%d y chunk=%d\n", kind, chunk);
}
```

CAPTURAS DE PANTALLA:

```

Codigos : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Codigos]$ gcc -fopenmp -O2 -o sheduled-clauseModificado sheduled-clauseModificado.c
sheduled-clauseModificado.c:9 aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
main(int argc, char **argv) {
^
[mike@looper Codigos]$ ./sheduled-clauseModificado 16 2
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 0 suma a[5] suma=15
thread 0 suma a[6] suma=21
thread 0 suma a[7] suma=28
thread 0 suma a[8] suma=36
thread 0 suma a[9] suma=45
thread 0 suma a[10] suma=55
thread 0 suma a[11] suma=66
Dentro de la region paralela
thread padre --> 0 ==>
dyn-var --> 0
nthreads-var --> 4
thread-limit-var --> 2147483647
run-sched-var ==> kind=2 y chunk=1
/-----/
Fuera de 'parallel for' suma=66
thread padre --> 0 ==>
dyn-var --> 0
nthreads-var --> 4
thread-limit-var --> 2147483647
run-sched-var ==> kind=2 y chunk=1
[mike@looper Codigos]$

```

RESPUESTA:

- “dyn-var”: variable de control que determina si el ajuste dinámico del número de procesos está activado o desactivado. (0 desactivado y 1 activado). Será igual para la región paralela y la región secuencial.
- “nthreads-var”: devuelve el número máximo de hebras que pueden utilizarse en una nueva región paralela. Será igual dentro y fuera de la región paralela.
- “thread-limit-var”: especifica el número máximo de hebras disponibles para el programa. Será igual dentro y fuera de la región paralela.
- “run-sched-var”: determinada por la cláusula Schedule, define el reparto de las iteraciones sobre las hebras.

Cómo vemos en la captura de pantalla todos los valores son iguales tanto dentro como fuera de la región paralela.

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

main(int argc, char **argv) {

    int i, n = 7, chunk, a[n], suma=0;

    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(static, 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 master
        {
            printf("Dentro de la region paralela\n");
            printf("thread padre --> %d ==>\n", omp_get_thread_num());
            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("/-----/\nFuera de 'parallel for' suma=
%d\n", suma);
    printf("thread padre --> %d ==>\n", omp_get_thread_num());
    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());
}

```

CAPTURAS DE PANTALLA:

```

Codigos : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Codigos]$ ./scheduled-clauseModificado4 2
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 3 suma a[6] suma=6
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
Dentro de la region paralela
thread padre --> 0 ==>
omp_get_num_threads() --> 4
omp_get_num_procs() --> 4
omp_in_parallel() -->1
/-----/
Fuera de 'parallel for' suma=6
thread padre --> 0 ==>
omp_get_num_threads() --> 1
omp_get_num_procs() --> 4
omp_in_parallel() -->0
[mike@looper Codigos]$

```

RESPUESTA:

- “omp_get_num_threads”: devuelve las hebras que se utilizan en la región paralela. En la región paralela usamos 4 hebras y en la región secuencial sólo 1 (la hebra padre).
- omp_get_num_procs(): devuelve el número de procesadores disponibles cuando llamamos a la función. Se utilizan el mismo número de procesadores en ambas regiones.
- “omp_in_parallel()”: Devuelve distinto de cero si se llama desde dentro de una región paralela. Región paralela 1 y región secuencial 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>
#endif

main(int argc, char **argv) {
    int i, n=200, chunk, a[n], suma=0;
    int nthreads;

    if(argc < 3) {
        fprintf(stderr, "\nFalta iteraciones o chunk \n");
        exit(-1);
    }

    n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
    nthreads=2;

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

    #pragma omp parallel for firstprivate(suma) lastprivate(suma)

```

```

schedule(dynamic, chunk)
for (i=0; i<n; i++)
{
    suma = suma + a[i];
    printf(" thread %d suma a[%d]=%d suma=%d omp_get_num_threads()=%d,
omp_get_num_procs()=%d y omp_in_parallel()=%d \n
",omp_get_thread_num(),i,a[i],suma,omp_get_num_threads(),omp_get_num_procs(),o
mp_in_parallel());
}

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

```

CAPTURAS DE PANTALLA:

```

Codigos : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Codigos]$ ./scheduled-clauseModificado5 16 2
thread 2 suma a[0]=0 suma=0 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 2 suma a[1]=1 suma=1 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 2 suma a[8]=8 suma=9 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 2 suma a[9]=9 suma=18 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 0 suma a[6]=6 suma=6 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 0 suma a[7]=7 suma=13 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 0 suma a[12]=12 suma=25 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 0 suma a[13]=13 suma=38 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 0 suma a[14]=14 suma=52 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 0 suma a[15]=15 suma=67 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 1 suma a[2]=2 suma=2 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 1 suma a[3]=3 suma=5 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 2 suma a[10]=10 suma=28 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 2 suma a[11]=11 suma=39 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 3 suma a[4]=4 suma=4 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
thread 3 suma a[5]=5 suma=9 omp_get_num_threads()=4, omp_get_num_procs()=4 y omp_in_parallel()=1
Fuera de 'parallel for' suma=67
omp_get_num_threads()=1, omp_get_num_procs()=4 y omp_in_parallel()=0[mike@looper Codigos]$

```

RESPUESTA:

- Con la función `omp_get_schedule(int *kind,int *chunk)` y `omp_set_schedule(int kind,int chunk)` modificamos `kind` y `chunk` o consultarlo.
- La variable `kind` determinara si el reparto es estático (`static`), dinámico(`dynamic`) o guiado(`guided`)
- La variable `chunk` definirá el valor de reparto entre las iteraciones y las hebras.

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

```

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define VECTOR_DYNAMIC
#ifdef VECTOR_GLOBAL
#define MAX 33554432 //2^25
double matriz[MAX*MAX], vector[MAX], vTotal[MAX]; //denotaremos siempre por
comodidad una matriz cuadrada de orden n como un vector de tamaño n x n
#endif

int main(int argc, char** argv){
    int i,j;
    double suma, ncgt;
    struct timespec cgt1,cgt2;

        if (argc<2){
            printf("La dimensión de la matriz no puede ser
nula, introducir un valor obviamente positivo\n");
            exit(-1);
        }

    unsigned int TAM = atoi(argv[1]);

#ifdef VECTOR_LOCAL
    double matriz[TAM*TAM], vector[TAM], vTotal[TAM];
#endif
#ifdef VECTOR_GLOBAL
    if (TAM>MAX)
        TAM=MAX;
#endif
#ifdef VECTOR_DYNAMIC
    double *matriz, *vector, *vTotal;
    matriz = (double*) malloc(TAM*TAM*sizeof(double));
    vector = (double*) malloc(TAM*sizeof(double));
    vTotal = (double*) malloc(TAM*sizeof(double));
    if ( (matriz==NULL) || (vector==NULL) || (vTotal==NULL) ){
        printf("Error a la hora de reservar memoria para
alguno de los vectores y/o 'matriz'\n");
        exit(-2);
    }
#endif

    for(i=0; i<TAM; i++){
        for(j=0; j<TAM; j++){
            if(j < i)
                matriz[(i*TAM)+j] = 0.0;
            else
                matriz[(i*TAM)+j] = TAM*0.1+j*0.1;
        }

        vector[i] = TAM*0.1-i*0.1;
        vTotal[i] = 0.0;
    }

        clock_gettime(CLOCK_REALTIME,&cgt1);
    for(i=0; i<TAM; i++){
        suma=0.0;
        for(j=(TAM-1); j>=i; j--){
            suma = suma + (matriz[(i*TAM)+j]*vector[j]);
            vTotal[i]=suma;
        }

        clock_gettime(CLOCK_REALTIME,&cgt2);
        ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
#ifdef PRINTF_ALL

```

```

                                printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:
%u\n",ncgt,N);
                                for(i=0; i<TAM; i++)
                                printf("vTotal[%d] = %f /\n",i,vTotal[i]);
                                #else
                                printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ vTotal[0]=
%f / vTotal[%d]=%f /\n", ncgt,TAM,vTotal[0],(int) TAM-1,vTotal[(int)TAM-1]);
                                #endif
                                #ifdef VECTOR_DYNAMIC
                                free(matriz);
                                free(vector);
                                free(vTotal);
                                #endif
                                return 0;
}

```

CAPTURAS DE PANTALLA:

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: Por defecto `static` usa uno, `dynamic` usa unidades de una iteración, `guided` coje el máximo posible y va menguando.

CÓDIGO FUENTE: `pmtv-OpenMP.c`

```

#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
#define VECTOR_DYNAMIC

```

```

#ifdef VECTOR_GLOBAL
#define MAX 33554432 //2^25
double matriz[MAX*MAX], vector[MAX], vTotal[MAX]; //denotaremos siempre por
comodidad una matriz cuadrada de orden n como un vector de tamaño n x n
#endif

int main(int argc, char** argv){
    int i, j, chunk, kind;
    double suma, ncgt, start, end;

    if (argc<4){
        printf("Te ha faltado por introducir algún dato\n
        Debes introducir en este dorden:\n
        -Dimension de la matriz \n
        -Tipo de scchedule(1 static, 2 dynamic, 3 guided)\n
        -Chunk");
        exit(-1);
    }
    unsigned int TAM = atoi(argv[1]);

    if((atoi(argv[2])==1)|| (atoi(argv[2])==2)|| (atoi(argv[2])==3))
        kind = atoi(argv[2]);
    else{
        printf("Error en el tipo de schedule usad 1 static, 2 dynamic, 3
        guided\n");
        exit(-1);
    }

    chunk = atoi(argv[3]);
    omp_set_schedule(kind, chunk);
#ifdef VECTOR_LOCAL
    double matriz[TAM*TAM], vector[TAM], vTotal[TAM];
#endif

#ifdef VECTOR_GLOBAL
    if (TAM>MAX)
        TAM=MAX;
#endif

#ifdef VECTOR_DYNAMIC
    double *matriz, *vector, *vTotal;
    matriz = (double*) malloc(TAM*TAM*sizeof(double));
    vector = (double*) malloc(TAM*sizeof(double));
    vTotal = (double*) malloc(TAM*sizeof(double));

    if ( (matriz==NULL) || (vector==NULL) || (vTotal==NULL) ){
        printf("Error a la hora de reservar memoria para alguno de los vectores
        y/o 'matriz'\n");
        exit(-2);
    }
#endif

#pragma omp parallel for default(none) private(j) shared (TAM, matriz,
vector, vTotal, i)
for(i=0; i<TAM; i++){
    for(j=0; j<TAM; j++){
        if(j < i)
            matriz[(i*TAM)+j] = 0.0;
        else
            matriz[(i*TAM)+j] = TAM*0.1+j*0.1;
    }
    vector[i] = TAM*0.1-i*0.1;
    vTotal[i] = 0.0;
}

start=omp_get_wtime();

```

```

#pragma omp parallel for default(none) private(j) shared(TAM, matriz,
vector, vTotal, i) firstprivate(suma) lastprivate(suma) schedule(runtime)
for(i=0; i<TAM; i++){
    suma=0.0;

    #pragma omp parallel for default(none) private(j) shared(TAM, matriz,
vector, vTotal, i) firstprivate(suma) lastprivate(suma) schedule(runtime)
    for(j=(TAM-1); j>=i; j--){
        suma = suma + (matriz[(i*TAM)+j]*vector[j]);

    vTotal[i]=suma;
    }
end=omp_get_wtime(); //obtenemos el tiempo de la ejecución
ncgt=end - start;

#ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
    for(i=0; i<TAM; i++)
        printf("vTotal[%d] = %f /\n",i,vTotal[i]);
#else
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ vTotal[0]=%f /
vTotal[%d]=%f /\n", ncgt,TAM,vTotal[0],(int) TAM-1,vTotal[(int)TAM-1]);
#endif

#ifdef VECTOR_DYNAMIC //liberamos el espacio reservado
    free(matriz);
    free(vector);
    free(vTotal);
#endif
return 0;
}

```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

```

Codigos : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[mike@looper Codigos]$ gcc -O2 -fopenmp -o pmtv-OpenMP pmtv-OpenMP.c -lrt
[mike@looper Codigos]$ ./pmtv-OpenMP
Te ha faltado por introducir algún dato
Debes introducir en este dorden:
-Dimension de la matriz
-Tipo de scchedule(1 static, 2 dynamic, 3 guided)
-Chunk
[mike@looper Codigos]$ ./pmtv-OpenMP 50 2 64
Tiempo(seg.):0.001699391          / Tamaño Vectores:50   / vTotal[0]=845.750000 / vTotal[49]=0.990000 /
[mike@looper Codigos]$

```

TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID**SCRIPT:** pmvt-OpenMP_atcgrid.sh

No he usado script.

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

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			
Chunk	Static	Dynamic	Guided
por defecto			
1			
64			

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

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

int main(int argc, char **argv){
    int num_filas=atoi(argv[1]);
    int num_columnas=atoi(argv[1]);
    int chunk=atoi(argv[2]);
    int **matriz1, **matriz2, **resultado;

    matriz1 = (int **)malloc(num_filas*sizeof(int*));
    matriz2 = (int **)malloc(num_filas*sizeof(int*));
    resultado=(int **)malloc(num_filas*sizeof(int*));
    int fila,columna,j;
    struct timespec cgt1,cgt2; double ncgt;

    for (fila=0;fila<num_filas;fila++){
        matriz1[fila] = (int*)malloc(num_filas*sizeof(int));
```

```

for (fila=0;fila<num_filas;fila++)
    matriz2[fila] = (int*)malloc(num_filas*sizeof(int));

for (fila=0;fila<num_filas;fila++)
    resultado[fila] = (int*)malloc(num_filas*sizeof(int));

    for(fila=0; fila<num_filas; fila++){
        for(columna=0; columna<num_columnas; columna++){
            matriz1[fila][columna]=2;
        }
    }

printf("Mostramos la matriz1\n");
for(fila=0; fila<num_filas; fila++){
    for(columna=0; columna<num_columnas; columna++){
        printf(" %d ", matriz1[fila][columna] );
    }
    printf("\n");
}

for(fila=0; fila<num_filas; fila++){
    for(columna=0; columna<num_columnas; columna++){
        matriz2[fila][columna]=2;
    }
}

printf("Mostramos la matriz2\n");
for(fila=0; fila<num_filas; fila++){
    for(columna=0; columna<num_columnas; columna++){
        printf(" %d ", matriz2[fila][columna] );
    }
    printf("\n");
}

printf("Multiplicamos la matrices\n");
clock_gettime(CLOCK_REALTIME,&cgt1);

    for (fila = 0 ; fila < num_filas ; fila++ ){
        for (columna = 0 ; columna < num_columnas; columna++ ){
            int producto = 0 ;
            for (j = 0 ; j < num_columnas ; j++ ){
                producto += matriz1[fila][j] * matriz2[j][columna];
                resultado[fila][columna] = producto ;
            }
        }
    }

clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9));
printf("Tiempo(seg.):%11.9f\t",ncgt);

printf("Mostramos el resultado\n");
for(fila=0; fila<num_filas; fila++){
    for(columna=0; columna<num_columnas; columna++){
        printf(" %d ", resultado[fila][columna] );
    }
    printf("\n");
}

for (fila=0;fila<num_filas;fila++)
    free(matriz1[fila]);
free(matriz1);

for (fila=0;fila<num_filas;fila++)
    free(matriz2[fila]);
free(matriz2);

for (fila=0;fila<num_filas;fila++)

```

```

    free(resultado[filas]);
    free(resultado);
}

```

CAPTURAS DE PANTALLA:

```

Codigos : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Codigos]$ ./pmm-secuencial 2 1
Mostramos la matriz1
 2  2
 2  2
Mostramos la matriz2
 2  2
 2  2
Multiplicamos la matrices
Tiempo(seg.):0.000000256      Mostramos el resultado
 8  8
 8  8
[mike@looper Codigos]$

```

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

```

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

int main(int argc, char **argv){
    int num_filas=atoi(argv[1]);
    int num_columnas=atoi(argv[1]);
    int chunk=atoi(argv[2]);
    int **matriz1, **matriz2, **resultado;

    matriz1 = (int **)malloc(num_filas*sizeof(int*));
    matriz2 = (int **)malloc(num_filas*sizeof(int*));
    resultado=(int **)malloc(num_filas*sizeof(int*));
    int fila,columna,j;
    struct timespec cgt1,cgt2; double ncg;

    for (fila=0;fila<num_filas;fila++)
        matriz1[fila] = (int*)malloc(num_filas*sizeof(int));

```

```

for (fila=0;fila<num_filas;fila++)
    matriz2[fila] = (int*)malloc(num_filas*sizeof(int));

for (fila=0;fila<num_filas;fila++)
    resultado[fila] = (int*)malloc(num_filas*sizeof(int));

#pragma omp parallel for default(none) private(fila,columna)
shared(matriz1,num_filas,num_columnas,chunk) schedule(dynamic,chunk)
for(fila=0; fila<num_filas; fila++){
    for(columna=0; columna<num_columnas; columna++){
        matriz1[fila][columna]=2;
    }
}

printf("Mostramos la matriz1\n");
for(fila=0; fila<num_filas; fila++){
    for(columna=0; columna<num_columnas; columna++){
        printf(" %d ", matriz1[fila][columna] );
    }
    printf("\n");
}

#pragma omp parallel for default(none) private(fila,columna)
shared(matriz2,num_filas,num_columnas,chunk) schedule(dynamic,chunk)
for(fila=0; fila<num_filas; fila++){
    for(columna=0; columna<num_columnas; columna++){
        matriz2[fila][columna]=2;
    }
}

printf("Mostramos la matriz2\n");
for(fila=0; fila<num_filas; fila++){
    for(columna=0; columna<num_columnas; columna++){
        printf(" %d ", matriz2[fila][columna] );
    }
    printf("\n");
}

printf("Multiplicamos la matrices\n");

clock_gettime(CLOCK_REALTIME,&cgt1);
#pragma omp parallel for default(none) private(fila,columna,j)
shared(matriz1,matriz2,num_filas,num_columnas,resultado,chunk) schedule(dynamic,chunk)
for (fila = 0 ; fila < num_filas ; fila++ ){
    for (columna = 0 ; columna < num_columnas; columna++ ){
        int producto = 0 ;
        for (j = 0 ; j < num_columnas ; j++ ){
            producto += matriz1[fila][j] * matriz2[j][columna];
            resultado[fila][columna] = producto ;
        }
    }
}

clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9));
printf("Tiempo(seg.):%11.9f\t",ncgt);

printf("Mostramos el resultado\n");
for(fila=0; fila<num_filas; fila++){
    for(columna=0; columna<num_columnas; columna++){
        printf(" %d ", resultado[fila][columna] );
    }
    printf("\n");
}

```



```

for (fila=0;fila<num_filas;fila++)
    free(matriz1[fila]);
free(matriz1);

for (fila=0;fila<num_filas;fila++)
    free(matriz2[fila]);
free(matriz2);

for (fila=0;fila<num_filas;fila++)
    free(resultado[fila]);
free(resultado);
}

```

CAPTURAS DE PANTALLA:

```

Codigos : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[mike@looper Codigos]$ ./pmm-0peMPn 2 1
Mostramos la matriz1
2 2
2 2
Mostramos la matriz2
2 2
2 2
Multiplicamos la matrices
Tiempo(seg.):0.00005122      Mostramos el resultado
8 8
8 8
[mike@looper Codigos]$

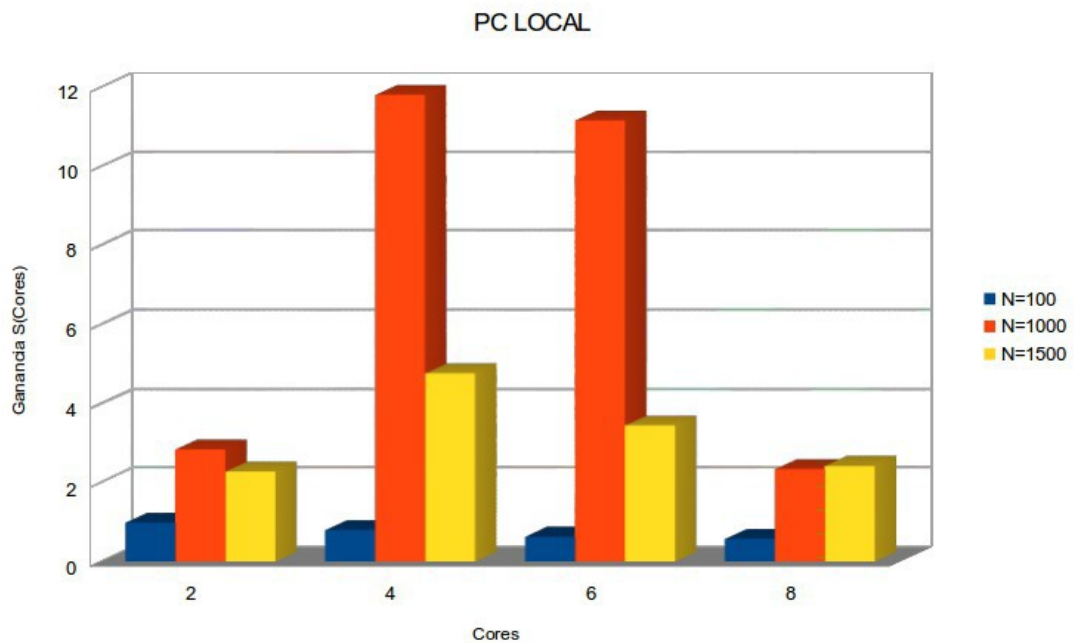
```

10. 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.

PC-LOCAL

Cores	100	1000	1500
1	0.0084306230	10.35061097	38.901914603
2	0.0085401360	3.650516600	17.068882860
4	0.0105208500	0.877368737	8.1676470000
6	0.0133853070	0.927544257	11.306501466
8	20.013385307	4.422386952	16.087501707

Cores	100	1000	1500
2	0.987176668	2.8353825237	2.2791131044
4	0.801328701	11.797332790	4.7629280015
6	0.629841587	11.159155902	3.4406677185
8	0.574118069	2.3405032355	2.4181451733



ATCGRID

Cores	100	1000	1500
1	0.004938312	10.72626270	35.206075047
2	0.008520310	4.083475198	16.147398397
4	0.010701230	1.175429536	6.0188763640
6	0.012433020	1.162651277	4.3432748460
8	0.011901230	1.209827588	3.5935852240
12	0.015848453	1.360803917	3.8124525760

Cores	100	1000	1500
2	0.579592996	2.4751619973	2.1802939509
4	0.461474380	8.5987822472	5.8492769942
6	0.397193280	8.6932881999	8.1058823803
8	0.414941313	8.3543000071	9.7969222524
12	0.311595838	7.4274202923	9.2344952088

