2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP

Estudiante (nombre y apellidos): David Martínez Díaz
Grupo de prácticas: Grupo 2
Fecha de entrega:13/05/2021
Fecha evaluación en clase:14/05/2021

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

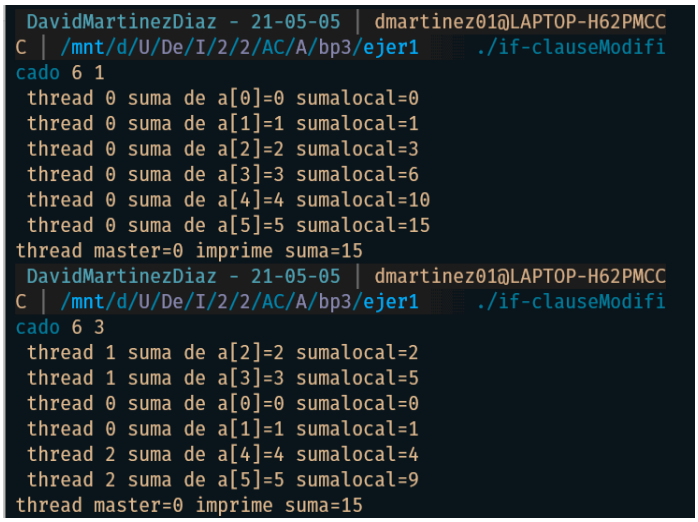
Ejercicios basados en los ejemplos del seminario práctico

1. 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.

CAPTURA CÓDIGO FUENTE: if-clauseModificado.c

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

int main(int argc, char **argv){

    int i, n=20, tid, x;
    int a[n],suma=0,sumalocal;

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

    n = atoi(argv[1]);
    x = atoi(argv[2]);

    if (n>20)
        n=20;

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

    #pragma omp parallel num_threads(x) 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:



RESPUESTA:

Lo que se consigue con esta cláusula es que a la hora de ejecutar el programa podamos decidir según el número que le demos por parámetro el número de hebras que se utilizaran, así podemos ir variando las sumas indicándole si queremos utilizar más o menos hebras.

2. Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) usando scheduler-clause.c con tres *threads* (0,1,2) y un número de iteraciones de 16 (0 a 15 en la tabla). Con este ejercicio se pretende comparar distintas alternativas de planificación de bucles. Se van a usar distintos tipos (static, dynamic, guided), modificadores (monotonic y nonmonotonic) y tamaños de chunk (x = 1, 2 y 4).

Tabla 1 . Tabla schedule. Rellenar esta tabla ejecutando scheduler-clause.c asignando previamente a la variable de entorno OMP_SCHEDULE los valores que se indican en la tabla (por ej.: export OMP_SCHEDULE="nonmonotonic:static,2). En la segunda fila, 1, 2 4 representan el tamaño del chunk

Iteración	"monotonic:static,x"		"nonmonotonic:static,x"		"monotonic:dynamic,x"		"monotonic:guided,x"	
	x=1	x=2	x=1	x=2	x=1	x=2	x=1	x=2
0	0	0	0	0	0	0	1	1
1	1	0	1	0	2	0	1	1
2	2	1	2	1	1	2	1	1
3	0	1	0	1	0	2	1	1
4	1	2	1	2	0	1	1	1
5	2	2	2	2	0	1	1	1
6	0	0	0	0	0	0	0	2
7	1	0	1	0	0	0	0	2
8	2	1	2	1	0	0	0	2
9	0	1	0	1	0	0	0	2
10	1	2	1	2	0	0	2	0
11	2	2	2	2	0	0	2	0
12	0	0	0	0	0	0	1	1
13	1	0	1	0	0	0	1	1
14	2	1	2	1	0	0	1	1
15	0	1	0	1	0	0	1	1

Destacar las diferencias entre las 4 alternativas de planificación de la tabla, en particular, las que hay entre static, dynamic y guided y las diferencias entre usar monotonic y nonmonotonic.

RESPUESTA:

En cuanto al static, sus hebras se distribuyen en tiempo de compilación, las iteraciones se dividen en unidades de chunk iteraciones y las unidades se asignan en round-robin.

Por otro lado, el dynamic, utiliza hebras que se distribuyen en tiempo de ejecución, por lo que no sabemos qué iteraciones realizará cada hebra y además, las hebras que sean las más rápidas serán las que ejecutan más unidades, pero como mínimo ejecutarán las chunk iteraciones que le corresponden.

Por último, las hebras de guided se distribuyen también en tiempo de ejecución, estas comienzan con un bloque largo, el cual se va disminuyendo hacia el número de iteraciones restantes entre el número de threads, y nunca es más pequeño que chunk.

En cuanto a las diferencias entre monotonic y nonmonotonic, es que el primero va repartiendo los distintos chunks a los threads de forma creciente en funcion de la carga lógica, mientras que el nonmonotonic es aleatorio.

3. ¿Qué valor por defecto usa OpenMP para chunk y modifier con static, dynamic y guided? Explicar qué ha hecho para contestar a esta pregunta.

En cuanto al modifier, podemos decir que su valor por defecto es el monotonic. Para el caso del elemento chunk, tiene dos opciones diferentes, para dynamic y guided es 1, sin embargo para static no tiene uno especificado, ya que a la hora de calcularlo, se realiza al principio de este ya que al dividirlo en funcion del numero de iteraciones que tenga en relación a las hebras, siendo dicha operación de forma equitativa sin darle importancion a la carga lógica que estas tengan.

Ademas, podemos llamar a la función omp_get_schedule(omp_sched_t * kind, int * chunk). Donde gracias a este podemos ver información del chunck y de kind.

Una vez analizado, podemos ver que el valor por defecto es de 0 para static mientras que para dynamic y guided el valor de chunk es 1.

4. 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?

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado.c

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

int main(int argc, char **argv) {

    int i, n = 200, chunk, modif, a[n],suma=0;
    omp_sched_t tipo;

    if(argc < 3) {

        fprintf(stderr,"\nFalta chunk \n");
        exit(-1);
    }

    chunk = atoi(argv[2]);
    n = atoi(argv[1]);

    if(n>200)
        n=200;

    int dyn_var = omp_get_dynamic();
    int nthreads_var = omp_get_max_threads();
    int thread_limit_var = omp_get_thread_limit();
    omp_get_schedule(&tipo, &modif);

    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] suma=%d \n", omp_get_thread_num(),i,suma);
    }

    #pragma omp single
        printf("\n thread %d dyn-var %d \n", omp_get_thread_num(), dyn_var);
        printf("\n thread %d nthreads-var %d \n", omp_get_thread_num(), nthreads_var);
        printf("\n thread %d thread-limit-var %d \n", omp_get_thread_num(), thread_limit_var);
        printf("\n thread %d run-sched-var %d \n", omp_get_thread_num(), modif);

    printf("Fuera de 'parallel for' suma=%d\n dyn-var %d\n nthreads-var %d\n thread-limit-var %d\n run-sc
}
```

CAPTURAS DE PANTALLA:

```
DavidMartinezDiaz - 21-05-05 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp3/ejer4
gcc -O2 -fopenmp schedule-clauseModificado.c -o schedule-clauseModificado
DavidMartinezDiaz - 21-05-05 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp3/ejer4
./schedule-clauseModificado 8 4
 thread 0 suma a[4] suma=4
 thread 0 suma a[5] suma=9
 thread 0 suma a[6] suma=15
 thread 0 suma a[7] suma=22
 thread 1 suma a[0] suma=0
 thread 1 suma a[1] suma=1
 thread 1 suma a[2] suma=3
 thread 1 suma a[3] suma=6

 thread 0 dyn-var 0

 thread 0 nthreads-var 12

 thread 0 thread-limit-var 2147483647

 thread 0 run-sched-var 1
Fuera de 'parallel for' suma=22
 dyn-var 0
 nthreads-var 12
 thread-limit-var 2147483647
 run-sched-var mod 1
DavidMartinezDiaz - 21-05-05 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp3/ejer4
./schedule-clauseModificado 10 6
 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 11 suma a[6] suma=6
 thread 11 suma a[7] suma=13
 thread 11 suma a[8] suma=21
 thread 11 suma a[9] suma=30

 thread 0 dyn-var 0

 thread 0 nthreads-var 12

 thread 0 thread-limit-var 2147483647

 thread 0 run-sched-var 1
Fuera de 'parallel for' suma=30
 dyn-var 0
 nthreads-var 12
 thread-limit-var 2147483647
 run-sched-var mod 1
```

RESPUESTA:
Independientemente de los valores de los parámetros sigue imprimiendo dentro y fuera del pragma parallel los mismos valores, que habíamos definidos en las anteriores variables de entorno.

5. 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.

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado4.c

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

int main(int argc, char **argv) {

    int i, n = 200, chunk, modif, a[n],suma=0;
    omp_sched_t tipo;

    if(argc < 3) {

        fprintf(stderr,"\nFalta chunk \n");
        exit(-1);
    }

    chunk = atoi(argv[2]);
    n = atoi(argv[1]);

    if(n>200)
        n=200;

    omp_get_schedule(&tipo, &modif);

    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++){

            if(i == 0)
                printf("Dentro del Parallel: num_threads=%d, num_procs=%d, omp_in_parallel=%d \n", omp_ge

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


    printf("\n Fuera del 'Parallel for' Suma=%d \n",suma);
    printf("\n omp_get_num_threads()=%d \n",omp_get_num_threads());
    printf("\n omp_get_num_procs()=%d \n",omp_get_num_procs());
    printf("\n omp_in_parallel()=%d \n",omp_in_parallel());

}
```

CAPTURAS DE PANTALLA:

```
 DavidMartinezDiaz - 21-05-05 | dmartinez01@LAPTOP-H62PMCCC | /
mnt/d/U/De/I/2/2/AC/A/bp3/ejer5      ./schedule-clauseModificado
4 10 4
thread 0 suma a[4] suma=0
thread 0 suma a[5] suma=4
thread 0 suma a[6] suma=9
thread 0 suma a[7] suma=15
Dentro del Parallel: num_threads=12, num_procs=12, omp_in_paral
lel=1
thread 5 suma a[0] suma=0
thread 5 suma a[1] suma=0
thread 5 suma a[2] suma=1
thread 5 suma a[3] suma=3
thread 3 suma a[8] suma=0
thread 3 suma a[9] suma=8

 Fuera del 'Parallel for' Suma=17

 omp_get_num_threads()=1

 omp_get_num_procs()=12

 omp_in_parallel()=0
```

RESPUESTA: Como podemos observar para la variable de las hebras cambia su valor, además de si está o no en una región paralela, diciendo que cuando está dentro de una utiliza 12 hebras y el valor del omp_parallel es de 1, mientras que fuera de este solo se utiliza una hebra y la región en parallel está a 0.

6. Añadir al programa scheduled-clause.c lo necesario para, usando funciones, modificar las variables de control dyn-var, nthreads-var y run-sched-var dentro de la región paralela y fuera de la región paralela. En la modificación de run-sched-var se debe usar un valor de kind distinto al utilizado en la cláusula schedule(). Añadir lo necesario para imprimir el contenido de estas variables antes y después de cada una de las dos modificaciones. Comentar los resultados.

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado5.c

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

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

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

    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]);

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

    omp_sched_t kind; int chunk_value;

    #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 \n",
                omp_get_thread_num(),i,a[i],suma);

        if(i == 3){

          omp_set_dynamic(0);
          omp_set_num_threads(4);
          omp_set_schedule(1,2);
        }
        if(i == 0 || i == 3){
          if(i == 0){
            printf("Antes de modificarse...\n");
          }
          else{
            printf("Despues de modificarse...\n");
          }

          omp_get_schedule(&kind, &chunk_value);
          printf("Dentro del parallel: \n dyn-var: %d | nthreads-var: %d \
          thread-limit-var: %d, run-sched-var: %d, chunk: %d \n", \
          omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_value);
        }
    }

    printf("Fuera de 'parallel for' suma=%d\n",suma);
    printf("Fuera del parallel: \n dyn-var: %d | nthreads-var: %d \
    thread-limit-var: %d, run-sched-var: %d, chunk: %d \n", \
    omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk);

    return(EXIT_SUCCESS);
}
```

CAPTURAS DE PANTALLA:

```
 DavidMartinezDiaz - 21-05-11 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp3/ejer6     ./scheduled-clauseMod
ificado5 7 3
 thread 0 suma a[0]=0 suma=0
Antes de modificarse...
Dentro del parallel:
 dyn-var: 1 | nthreads-var: 1          thread-limit-var: 2147483647, run-sched-var: 2, chunk: 3
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 0 suma a[3]=3 suma=6
Despues de modificarse...
Dentro del parallel:
 dyn-var: 0 | nthreads-var: 4          thread-limit-var: 2147483647, run-sched-var: 1, chunk: 2
 thread 0 suma a[4]=4 suma=10
 thread 0 suma a[5]=5 suma=15
 thread 0 suma a[6]=6 suma=21
Fuera de 'parallel for' suma=21
Fuera del parallel:
 dyn-var: 1 | nthreads-var: 1     thread-limit-var: 2147483647, run-sched-var: 1, chunk: 3
```

RESPUESTA:

En primer lugar, mostramos desde la iteración 0 hasta la iteración 3, es donde se enseña los valores antes de ser modificados, sin embargo, una vez pasado, se consiguen modificar “dyn_var” que pasa a tener valor 0. Para el nthreads_var que pasa a tener valor 4; run-sched-var con valor 1, y chunk con valor 2.

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

7. Implementar un programa secuencial en C que multiplique una matriz triangular inferior por un vector (use variables dinámicas y tipo de datos double). Comparar el orden de complejidad y el número total de operaciones (sumas y productos) de este código respecto al 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.

CAPTURA CÓDIGO FUENTE: pmtv-secuencial.c

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

int main(int argc, char** argv){

    struct timespec cgt1, cgt2;
    double ncgt;

    if(argc<2){
        printf("Faltan no componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    int i, j;
    int *v, *mv, **m;

    v = (int*) malloc(N*sizeof(int));
    mv = (int*) malloc(N*sizeof(int));
    m = (int*) malloc(N*sizeof(int *));

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

    if ( v==NULL || mv==NULL || m==NULL ){
        printf("Error en la reserva de espacio para la matriz y el vector");
        exit(-2);

    }

    for(i=0; i<N; i++)
        if(m[i] == NULL){
            printf("Error en la reserva de espacio para la matriz y el vector");
            exit(-3);
        }


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

        for(j=i; j<N; j++){
            m[i][j] = 9;
        }

        v[i] = 2;
        mv[i] = 0;
    }
    clock_gettime(CLOCK_REALTIME,&cgt1);

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

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


    clock_gettime(CLOCK_REALTIME,&cgt2);

    printf("Matriz: \n");
    for(i=0; i<N; i++){

        for(j=0; j<N; j++)
            if(j >= i)
                printf("%d ", m[i][j]);
            else
                printf("0 ");
        printf("\n");
    }

    printf("Vector: \n");
    for(i=0; i<N; i++)
        printf("%d ", v[i]);
    printf("\n");
```

```
74
75      printf("Resultado: \n");
76      for(i=0; i<N; i++)
77          printf("%d ", mv[i]);
78      printf("\n");
79
80      ncgt=(double)(cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
81
82      printf("Tamaño de la matriz y vector: %d \t / Tiempo(seg.) %11.9f \n", N, ncgt);
83
84      for(i=0; i<N; i++)
85          free(m[i]);
86
87      free(m);
88      free(v);
89      free(mv);
90
91      return 0;
92
93  }
```

CAPTURAS DE PANTALLA:

```
[e2estudiante14@atcgrid ejer7]$ ./pmtv-secuencial 5
DIMENSION DE VECTOR Y MATRIZ: 5 (4 B)
MATRIZ M:
0.500000000     0.000000000     0.000000000     0.000000000     0.000000000
0.600000000     0.500000000     0.000000000     0.000000000     0.000000000
0.700000000     0.600000000     0.500000000     0.000000000     0.000000000
0.800000000     0.700000000     0.600000000     0.500000000     0.000000000
0.900000000     0.800000000     0.700000000     0.600000000     0.500000000

VECTOR V: [0.500000000  0.600000000     0.700000000     0.800000000     0.90
0000000         ]
RESULTADO DE MULTIPLICACION: [0.000000000       0.300000000     0.7100000001
.240000000      1.900000000     ]
TIEMPO DE EJECUCION: 0.000000256 || DIMENSION: 5
[e2estudiante14@atcgrid ejer7]$ s
```

8. Implementar en paralelo la multiplicación de una matriz triangular inferior 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. La inicialización de los datos la debe hacer el thread 0. 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. Mostrar en una captura de pantalla que el código resultante funciona correctamente. NOTA: usar para generar los valores aleatorios, por ejemplo, drand48_r().

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

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

int main(int argc, char** argv){

    struct timespec cgt1, cgt2;
    double ncgt, t_ini, t_final, t_total;

    if(argc<2){
        printf("Faltan no componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    int i, j, suma;
    int *v, *mv, **m;

    v = (int*) malloc(N*sizeof(int));
    mv = (int*) malloc(N*sizeof(int));
    m = (int*) malloc(N*sizeof(int *));

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

    if ( v==NULL || mv==NULL || m==NULL ){
        printf("Error en la reserva de espacio para la matriz y el vector");
        exit(-2);

    }

    for(i=0; i<N; i++)
        if(m[i] == NULL){
            printf("Error en la reserva de espacio para la matriz y el vector");
            exit(-3);
        }

    #pragma omp parallel for firstprivate(j) schedule(runtime)
    for(i=0; i<N; i++){

        for(j=i; j<N; j++){
            m[i][j] = 9;
        }

        v[i] = 2;
        mv[i] = 0;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    #pragma omp parallel
    {
      #pragma omp single
        t_ini = omp_get_wtime();

      #pragma omp for schedule(runtime)
      for(int i = 0; i<N ; i++)
        for(int j = 0; j<N; j++)
            mv[i] += m[i][j] * v[i];

      #pragma omp single
        t_final = omp_get_wtime();
    }


    t_total= t_final - t_ini;

    clock_gettime(CLOCK_REALTIME,&cgt2);

    printf("Matriz: \n");
    for(i=0; i<N; i++){
        for(j=0; j<N; j++)
            if(j >= i)
                printf("%d ", m[i][j]);
            else
                printf("0 ");
        printf("\n");
    }

    printf("Vector: \n");
    for(i=0; i<N; i++)
        printf("%d ", v[i]);
    printf("\n");

    printf("Resultado: \n");
    for(i=0; i<N; i++)
        printf("%d ", mv[i]);
    printf("\n");

    ncgt=(double)(cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    printf("Tamaño de la matriz y vector: %d \t / Tiempo(seg.) %11.9f \n", N, ncgt);

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

    free(m);
    free(v);
    free(mv);

    return 0;

}
```

DESCOMPOSICIÓN DE DOMINIO:

	0	1	2	3	n-1	n	v0		r0
0	m	M01	M02	M03	M0(n-1)	M0n	V1		R1
1	0	M11	M12	M13	M1(n-1)	M1n	V2		R2
2	0	0	M22	M23	M2n	M2n	V3		R3
3	0	0	0	M33	M3(n-1)	M3n	V4	=	R4
	0	0	0	0					
n-1	0	0	0	0	M(n-1)(n-1)	Mn-1n	Vn-1		Rn-1
n	0	0	0	0	0	Mmn	Vn		rn

CAPTURAS DE PANTALLA:

```
[e2estudiante14@atcgrid ejer8]$ srun -p ac ./pmtv-OpenMP 5
Matriz:
9 9 9 9 9
0 9 9 9 9
0 0 9 9 9
0 0 0 9 9
0 0 0 0 9
Vector:
2 2 2 2 2
Resultado:
90 51820126 -965809338 51833230 -913594696
Tamaño de la matriz y vector: 5          / Tiempo(seg.) 0.
000009127
```

9. Contestar a las siguientes preguntas sobre el código del ejercicio anterior:

(a) ¿Qué número de operaciones de multiplicación y qué número de operaciones de suma realizan cada uno de los threads en la asignación static con monotonic y un chunk de 1?

RESPUESTA:
Al ser una matriz triangular, esta depende del número de iteraciones que tenga su fila. Así, la primera fila tendrá que realizar n iteraciones, la siguiente hará n-1, la siguiente n-2 y así sucesivamente hasta llegar a la última fila que realizara sola 1 iteración.

(b) 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: Bueno el número de operaciones que deberá realizar cada una de las hebras dependerá claramente de la disponibilidad que estas tengan. Donde cada hebra hará un cierto de número de operaciones diferente, dependiendo si están ocupadas o libres.

(c) ¿Qué alternativa ofrece mejores prestaciones? Razonar la respuesta.

RESPUESTA: En nuestro caso sería la planificación guided, puesto que reduce bastante el tiempo de ejecución, pero de forma teorica es mejor la planificación dynamic, por que aprovechar las iteraciones dividiéndolas por el número de chunk iteraciones y se le asignan en tiempo de ejecución.

10. 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 (con monotonic en todos los casos). Usar un tamaño de vector N múltiplo del número de cores y de 64 que esté entre 11520 y 23040. El número de threads en las ejecuciones debe coincidir con el número de núcleos del computador. 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

(representar los valores de las dos tablas). Incluir los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

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

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

#define VAR_DYNAMIC

int main(int argc, char** argv){

    struct timespec cgt1, cgt2;
    double ncgt, t_ini, t_final, t_total;

    if(argc<2){
        printf("Faltan no componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    int i, j, suma;
    int *v, *mv, **m;

    v = (int*) malloc(N*sizeof(int));
    mv = (int*) malloc(N*sizeof(int));
    m = (int*) malloc(N*sizeof(int *));

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

    if ( v==NULL || mv==NULL || m==NULL ){
        printf("Error en la reserva de espacio para la matriz y el vector");
        exit(-2);
    }

    for(i=0; i<N; i++)
        if(m[i] == NULL){
            printf("Error en la reserva de espacio para la matriz y el vector");
            exit(-3);
        }


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

        for(j=i; j<N; j++){
            m[i][j] = 9;
        }

        v[i] = 2;
        mv[i] = 0;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    omp_sched_t kind;
    int chunk_value;


    #pragma omp parallel
    {
        #pragma omp single
        t_ini = omp_get_wtime();

        #pragma omp for firstprivate(suma) schedule(runtime)
        for(int i = 0; i<N ; i++){

        #ifdef SEE_CHUNK_DEFAULT
            if(i==0){
                omp_get_schedule(&kind,&chunk_value);
                printf("KIND: %d | CHUNK: %d \n",kind,chunk_value);
            }
        #endif

            suma = 0;
            for(int j = 0; j<N; j++)
                suma += m[i][j] * v[i];

            mv[i] = suma;
        }

        #pragma omp single
        t_final = omp_get_wtime();

    }


    t_total= t_final - t_ini;

    clock_gettime(CLOCK_REALTIME,&cgt2);

    printf("Matriz: \n");
    for(i=0; i<N; i++){

        for(j=0; j<N; j++)
            if(j >= i)
                printf("%d ", m[i][j]);
            else
                printf("0 ");
        printf("\n");
    }

    printf("Vector: \n");
    for(i=0; i<N; i++)
        printf("%d ", v[i]);
    printf("\n");

    printf("Resultado: \n");
    for(i=0; i<N; i++)
        printf("%d ", mv[i]);


    ncgt=(double)(cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    printf("Tamaño de la matriz y vector: %d \t / Tiempo(seg.) %11.9f \n", N, ncgt);

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

    free(m);
    free(v);
    free(mv);

    return 0;

}
```

DESCOMPOSICIÓN DE DOMINIO:

	0	1	2	3	n-1	n	v0		r0
0	m	M01	M02	M03	M0(n-1)	M0n	V1		R1
1	0	M11	M12	M13	M1(n-1)	M1n	V2		R2
2	0	0	M22	M23	M2n	M2n	V3		R3
3	0	0	0	M33	M3(n-1)	M3n	V4	=	R4
n-1	0	0	0	0	M(n-1)(n-1)	Mn-1n	Vn-1		Rn-1
n	0	0	0	0	0	Mmn	Vn		rn

CAPTURAS DE PANTALLA:

```
[e2estudiante14@atcgrid ejer10]$ gcc -O2 -fopenmp pmtv-OpenMP.c -o pmtv-OpenMP
[e2estudiante14@atcgrid ejer10]$ chmod u+x pmtv-OpenMP
[e2estudiante14@atcgrid ejer10]$ ./pmtv-OpenMP_atcgrid.sh
PLANIFICACION: static | CHUNK: DEFAULT
DIMENSION DE VECTOR Y MATRIZ: 15400 (4 B)
MATRIZ CREADA: m[0][0]=1540.000000000 || m[15399][15399]=1540.000000000
VECTOR CREADO: v[0]=1540.000000000 || v[15399]=3079.900000000
RESULTADO DE MULTIPLICACION: mv[0]=24349612441.000000000 || mv[15399]=4743046.000000000
TIEMPO DE EJECUCION: 0.158837538 || DIMENSION: 15400


PLANIFICACION: static | CHUNK: 1
DIMENSION DE VECTOR Y MATRIZ: 15400 (4 B)
MATRIZ CREADA: m[0][0]=1540.000000000 || m[15399][15399]=1540.000000000
VECTOR CREADO: v[0]=1540.000000000 || v[15399]=3079.900000000
RESULTADO DE MULTIPLICACION: mv[0]=24349612441.000000000 || mv[15399]=4743046.000000000
TIEMPO DE EJECUCION: 0.160108473 || DIMENSION: 15400


PLANIFICACION: static | CHUNK: 64
DIMENSION DE VECTOR Y MATRIZ: 15400 (4 B)
MATRIZ CREADA: m[0][0]=1540.000000000 || m[15399][15399]=1540.000000000
VECTOR CREADO: v[0]=1540.000000000 || v[15399]=3079.900000000
RESULTADO DE MULTIPLICACION: mv[0]=24349612441.000000000 || mv[15399]=4743046.000000000
TIEMPO DE EJECUCION: 0.159618005 || DIMENSION: 15400


PLANIFICACION: dynamic | CHUNK: DEFAULT
DIMENSION DE VECTOR Y MATRIZ: 15400 (4 B)
MATRIZ CREADA: m[0][0]=1540.000000000 || m[15399][15399]=1540.000000000
VECTOR CREADO: v[0]=1540.000000000 || v[15399]=3079.900000000
RESULTADO DE MULTIPLICACION: mv[0]=24349612441.000000000 || mv[15399]=4743046.000000000
TIEMPO DE EJECUCION: 0.159440901 || DIMENSION: 15400


PLANIFICACION: dynamic | CHUNK: 1
DIMENSION DE VECTOR Y MATRIZ: 15400 (4 B)
MATRIZ CREADA: m[0][0]=1540.000000000 || m[15399][15399]=1540.000000000
VECTOR CREADO: v[0]=1540.000000000 || v[15399]=3079.900000000
RESULTADO DE MULTIPLICACION: mv[0]=24349612441.000000000 || mv[15399]=4743046.000000000
TIEMPO DE EJECUCION: 0.159155034 || DIMENSION: 15400
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash

export OMP_NUM_THREADS=12

declare -a planificacion=("static" "dynamic" "guided")

for i in "${planificacion[@]}"
do
        for (( j = 0; j < 3; ++j )); do
                if [ $j -eq 0 ]; then
                        echo "PLANIFICACION: $i | CHUNK: DEFAULT"
                        export OMP_SCHEDULE=$i
                elif [ $j -eq 1 ]; then
                        echo "PLANIFICACION: $i | CHUNK: 1"
                        export OMP_SCHEDULE="$i,1"
                elif [ $j -eq 2 ]; then
                        echo "PLANIFICACION: $i | CHUNK: 64"
                        export OMP_SCHEDULE="$i,64"
                fi
                srun ./pmtv-OpenMP 15400
                echo -e "\n"
        done
done
```

Tabla 2 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector para vectores de tamaño N= (solo se ha paralelizado el producto, no la inicialización de los datos).

Chunk	Static	Dynamic	Guided
por defecto	0.158837538	0.159440901	0.158992730
1	0.160108473	0.159155034	0.158731978
64	0.159618005	0.158802900	0.159027945
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.157796141	0.158730395	0.158527248
1	0.159374367	0.158904031	0.158617981
64	0.158881940	0.158495646	0.159619723