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): Salvador Romero Cortés
Grupo de prácticas: A1
Fecha de entrega: 17/05/2021
Fecha evaluación en clase:

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

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

    n = atoi(argv[1]);
    if (n>20)
        n=20;

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

    //parametro de entrada para el numero de threads
    int x = atoi(argv[2]);

    #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:

```
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer1] 2021-05-13 jueves
$g++ -o if-clauseModificado if-clause.c -O2 -fopenmp
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer1] 2021-05-13 jueves
$./if-clauseModificado 6 4
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 3 suma de a[5]=5 sumalocal=5
 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 master=0 imprime suma=15
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer1] 2021-05-13 jueves
$./if-clauseModificado 6 2
 thread 1 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[4]=4 sumalocal=7
 thread 1 suma de a[5]=5 sumalocal=12
 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 master=0 imprime suma=15
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer1] 2021-05-13 jueves
$./if-clauseModificado 6 6
 thread 2 suma de a[2]=2 sumalocal=2
 thread 0 suma de a[0]=0 sumalocal=0
 thread 5 suma de a[5]=5 sumalocal=5
 thread 1 suma de a[1]=1 sumalocal=1
 thread 3 suma de a[3]=3 sumalocal=3
 thread 4 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=15
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer1] 2021-05-13 jueves
$
```

RESPUESTA:

Podemos ver que el codigo lo ejecutan tantas hebras como le pasemos al argumento x. Vemos en las capturas que primero pasamos 4 luego 2 y finalmente 6 y ejecutan 4,2 y 6 hebras respectivamente.

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

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:

Las diferencias está en que static planifica los threads en tiempo de compilación, mientras que dynamic y guided lo hacen en tiempo de ejecución. La diferencia entre estas dos últimas resulta en que usando guided, el tamaño de bloque va menguando. Además dynamic tiene más "overhead".
No hay diferencia práctica entre monotonic y nonmonotonic. Teóricamente, con monotonic las threads i-ésimas solo pueden ejecutar las iteraciones i-ésimas o superior a i. Con nonmonotonic no se tiene esta restricción.

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.

Static
Chunk: numero de iteraciones / número de threads.
Modifier: monotonic.
Dynamic
Chunk: 1
Modifier: nonmonotonic.
Guided:
Chunk: numero de iteraciones / número de threads.
Modifier: nonmonotonic.

Para contestar esta pregunta he consultado la documentación:
https://www.openmp.org/spec-html/5.0/openmpse49.html

https://software.intel.com/content/www/us/en/develop/articles/openmp-loop-scheduling.html

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, 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;

    #pragma omp parallel for firstprivate(suma) \
                lastprivate(suma) schedule(dynamic,chunk)
    for (i=0; i<n; i++)
    {
        if (i == 0)
        {
            omp_get_schedule(&kind,&chunk);
            printf("Dentro de PARALLEL:\ndyn-var: %d, nthreads-var: %d, thread-limit-var: %d run-sched-var : kind: %d, chunk: %d\n",
                omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk );
        }
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
                omp_get_thread_num(),i,a[i],suma);
    }
    omp_get_schedule(&kind,&chunk);
    printf("Fuera de PARALLEL:\ndyn-var: %d, nthreads-var: %d, thread-limit-var: %d run-sched-var : kind: %d, chunk: %d\n",
        omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk );
    printf("Fuera de 'parallel for' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

```
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer4] 2021-05-13 jueves
$g++ -o scheduled-clauseModificado scheduled-clauseModificado.c -O2 -fopenmp
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer4] 2021-05-13 jueves
$export OMP_NUM_THREADS=2; export OMP_SCHEDULE="static,2"
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer4] 2021-05-13 jueves
$./scheduled-clauseModificado 6 1
Dentro de PARALLEL:
dyn-var: 1, nthreads-var: 2, thread-limit-var: 2147483647 run-sched-var : kind: -2147483647, chunk: 2
 thread 0 suma a[0]=0 suma=0
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[3]=3 suma=4
 thread 1 suma a[4]=4 suma=8
 thread 1 suma a[5]=5 suma=13
 thread 0 suma a[2]=2 suma=2
Fuera de PARALLEL:
dyn-var: 1, nthreads-var: 2, thread-limit-var: 2147483647 run-sched-var : kind: -2147483647, chunk: 2
Fuera de 'parallel for' suma=13
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer4] 2021-05-13 jueves
$export OMP_NUM_THREADS=4; export OMP_SCHEDULE="static,4"; export OMP_DYNAMIC=TRUE
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer4] 2021-05-13 jueves
$./scheduled-clauseModificado 6 1
 thread 0 suma a[2]=2 suma=2
 thread 0 suma a[4]=4 suma=6
 thread 0 suma a[5]=5 suma=11
Dentro de PARALLEL:
dyn-var: 1, nthreads-var: 4, thread-limit-var: 2147483647 run-sched-var : kind: -2147483647, chunk: 4
 thread 2 suma a[0]=0 suma=0
 thread 3 suma a[1]=1 suma=1
 thread 1 suma a[3]=3 suma=3
Fuera de PARALLEL:
dyn-var: 1, nthreads-var: 4, thread-limit-var: 2147483647 run-sched-var : kind: -2147483647, chunk: 4
Fuera de 'parallel for' suma=11
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer4] 2021-05-13 jueves
$export OMP_NUM_THREADS=4; export OMP_SCHEDULE="static,4"; export OMP_DYNAMIC=TRUE
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer4] 2021-05-13 jueves
$./scheduled-clauseModificado 6 1
Dentro de PARALLEL:
dyn-var: 1, nthreads-var: 4, thread-limit-var: 2147483647 run-sched-var : kind: -2147483647, chunk: 4
 thread 1 suma a[0]=0 suma=0
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 2 suma a[3]=3 suma=3
 thread 3 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=2
Fuera de PARALLEL:
dyn-var: 1, nthreads-var: 4, thread-limit-var: 2147483647 run-sched-var : kind: -2147483647, chunk: 4
Fuera de 'parallel for' suma=9
```

RESPUESTA:

Muestran el mismo resultado tanto dentro como fuera de la región paralela.

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, 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;

    #pragma omp parallel for firstprivate(suma) \
                lastprivate(suma) schedule(dynamic,chunk)
    for (i=0; i<n; i++)
    {
        if (i == 0)
        {
            printf("Dentro de PARALLEL:\n omp_get_num_threads(): %d, omp_get_num_procs(): %d, omp_in_parallel(): %d \n",
                omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
        }
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
                omp_get_thread_num(),i,a[i],suma);
    }
    printf("Fuera de PARALLEL:\n omp_get_num_threads(): %d, omp_get_num_procs(): %d, omp_in_parallel(): %d \n",
        omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
    printf("Fuera de 'parallel for' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

```
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer5] 2021-05-13 jueves
$g++ -o scheduled-clauseModificado4 scheduled-clauseModificado4.c -O2 -fopenmp
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer5] 2021-05-13 jueves
$./scheduled-clauseModificado4 6 1
 thread 6 suma a[5]=5 suma=5
Dentro de PARALLEL:
 omp_get_num_threads(): 12, omp_get_num_procs(): 12, omp_in_parallel(): 1
 thread 11 suma a[0]=0 suma=0
 thread 9 suma a[4]=4 suma=4
 thread 7 suma a[2]=2 suma=2
 thread 4 suma a[1]=1 suma=1
 thread 2 suma a[3]=3 suma=3
Fuera de PARALLEL:
 omp_get_num_threads(): 1, omp_get_num_procs(): 12, omp_in_parallel(): 0
Fuera de 'parallel for' suma=5
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer5] 2021-05-13 jueves
$./scheduled-clauseModificado4 6 2
Dentro de PARALLEL:
 omp_get_num_threads(): 12, omp_get_num_procs(): 12, omp_in_parallel(): 1
 thread 7 suma a[0]=0 suma=0
 thread 7 suma a[1]=1 suma=1
 thread 11 suma a[2]=2 suma=2
 thread 11 suma a[3]=3 suma=5
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
Fuera de PARALLEL:
 omp_get_num_threads(): 1, omp_get_num_procs(): 12, omp_in_parallel(): 0
Fuera de 'parallel for' suma=9
```

RESPUESTA:

Se obtienen valores distintos en el número de hebras y en "`omp_in_parallel()`". Esto ocurre porque en la zona paralela se ejecutan el número de hebras indicado por `OMP_NUM_THREADS` mientras que fuera de la región paralela solo se ejecuta por una única hebra. La segunda variable indica si se encuentra en una región paralela o no, por eso muestra 1 al ejecutarse dentro del `for` concurrente y 0 tras salir de él.

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;

    #pragma omp parallel for firstprivate(suma) \
                lastprivate(suma) schedule(dynamic,chunk)
    for (i=0; i<n; i++)
    {
        if (i==0)
        {
            printf("ANTES DE MODIFICAR\n");
            omp_get_schedule(&kind,&chunk);
            printf("Dentro de PARALLEL:\ndyn-var: %d, nthreads-var: %d run-sched-var : kind: %d, chunk: %d\n",
                omp_get_dynamic(), omp_get_max_threads(), kind, chunk );
        }
```

```
        if (i==2){
            omp_set_dynamic(1);
            omp_set_num_threads(3);
            omp_set_schedule(omp_sched_guided,4);

            printf("DESPUÉS DE MODIFICAR\n");
            omp_get_schedule(&kind,&chunk);
            printf("Dentro de PARALLEL:\ndyn-var: %d, nthreads-var: %d run-sched-var : kind: %d, chunk: %d\n",
                omp_get_dynamic(), omp_get_max_threads(), kind, chunk );
        }
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
               omp_get_thread_num(),i,a[i],suma);
    }
    omp_get_schedule(&kind,&chunk);
    printf("Fuera de PARALLEL:\ndyn-var: %d, nthreads-var: %d run-sched-var : kind: %d, chunk: %d\n",
        omp_get_dynamic(), omp_get_max_threads(), kind, chunk );

    printf("MODIFICADO DESPUÉS FUERA de Parallel\n");
    omp_set_dynamic(0);
    omp_set_num_threads(5);
    omp_set_schedule(omp_sched_dynamic,1);

    omp_get_schedule(&kind,&chunk);
    printf("dyn-var: %d, nthreads-var: %d run-sched-var : kind: %d, chunk: %d\n",
        omp_get_dynamic(), omp_get_max_threads(), kind, chunk );


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

CAPTURAS DE PANTALLA:

```
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer6] 2021-05-14 viernes
$g++ -o scheduled-clauseModificado5 scheduled-clauseModificado5.c -O2 -fopenmp
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp3/ejer6] 2021-05-14 viernes
$./scheduled-clauseModificado5 6 1
ANTES DE MODIFICAR
Dentro de PARALLEL:
dyn-var: 1, nthreads-var: 12 run-sched-var : kind: -2147483647, chunk: 4
 thread 10 suma a[0]=0 suma=0
 thread 11 suma a[5]=5 suma=5
 thread 9 suma a[4]=4 suma=4
 thread 6 suma a[3]=3 suma=3
DESPUÉS DE MODIFICAR
Dentro de PARALLEL:
dyn-var: 1, nthreads-var: 3 run-sched-var : kind: 3, chunk: 4
 thread 4 suma a[2]=2 suma=2
 thread 8 suma a[1]=1 suma=1
Fuera de PARALLEL:
dyn-var: 1, nthreads-var: 12 run-sched-var : kind: -2147483647, chunk: 4
MODIFICADO DESPUÉS FUERA de Parallel
dyn-var: 0, nthreads-var: 5 run-sched-var : kind: 2, chunk: 1
Fuera de 'parallel for' suma=5
```

RESPUESTA:
Vemos que las modificaciones dentro de la zona paralela no afecta a la zona no paralela. También comprobamos como, efectivamente, estamos cambiando el valor de esas variables.

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 <stdio.h>
#include <omp.h>
#include <stdlib.h>
#include <time.h>


int main(int argc, char** argv) {
    int i,j;

    double tiempo;
    struct timespec cgt1, cgt2;

    if (argc != 2) {
        printf("Argumentos incorrectos. Uso: binario <tamaño>");
        exit(-1);
    }

    int n = atoi(argv[1]);

//reserva de memoria dinamica
    double** m = (double**)malloc(n * sizeof(double *));
    double* v1 = (double*)malloc(n * sizeof(double));
    double* v3 = (double*)malloc(n * sizeof(double));

    for (i = 0; i < n; i++) {
        m[i] = (double*)malloc(n * sizeof(double));
    }

    //inicializacion de datos de la matriz y los vectores
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (i < j)
                m[i][j] = 0;    // solo inicializamos los valores del triangulo inferior
            else
                m[i][j] = 0.1*i*n - 0.1*j*n;
        }
        v1[i] = 0.1*i*n;
        v3[i] = 0;
    }

    // calculo
    clock_gettime(CLOCK_REALTIME, &cgt1);
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (i > j)
                v3[i] += m[i][j] * v1[j];
        }
    }
    clock_gettime(CLOCK_REALTIME, &cgt2);
    tiempo = (double)(cgt2.tv_sec - cgt1.tv_sec) +
        (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
```

```
    // muestra de resultados
    if (n <= 10) {
        printf("Matriz: \n");
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                printf("%f\t", m[i][j]);
            }
            printf("\n");
        }
        printf("\n");
        printf("Vector producto: \n");
        for (i = 0; i < n; i++)
            printf("%f", v1[i]);
        printf("\n");

        printf("Vector resultado: \n");
        for (i = 0; i < n; i++) {
            printf("%f", v3[i]);
        }
        printf("\n");

    }
    else {
        printf("Matriz m[0][0] = %f <--> m[%d][%d] = %f\n",
            m[0][0], n - 1, n - 1, m[n - 1][n - 1]);
        printf("V1[0] = %f <--> V1[%d] = %f\n",
            v1[0], n - 1, v1[n - 1]);
        printf("Resultado V3[0] = %f <--> V3[%d] = %f\n",
            v3[0], n - 1, v3[n - 1]);
    }
    printf("Tiempo: %f11.9\n", tiempo);
//liberacion de memoria dinamica
    for (i = 0; i < n; i++)
        free(m[i]);
    free(m);
    free(v1);
    free(v3);
}
```

CAPTURAS DE PANTALLA:

```
[a1estudiante23@atcgrid ejer7]$ g++ -o pmtv-secuencial pmtv-secuencial.c -O2 -fopenmp
[a1estudiante23@atcgrid ejer7]$ srun -p ac -A ac ./pmtv-secuencial 8
Matriz:
0.000000        0.000000        0.000000        0.000000        0.000000        0.000000        0.000000        0.000000
0.800000        0.000000        0.000000        0.000000        0.000000        0.000000        0.000000        0.000000
1.600000        0.800000        0.000000        0.000000        0.000000        0.000000        0.000000        0.000000
2.400000        1.600000        0.800000        0.000000        0.000000        0.000000        0.000000        0.000000
3.200000        2.400000        1.600000        0.800000        0.000000        0.000000        0.000000        0.000000
4.000000        3.200000        2.400000        1.600000        0.800000        0.000000        0.000000        0.000000
4.800000        4.000000        3.200000        2.400000        1.600000        0.800000        0.000000        0.000000
5.600000        4.800000        4.000000        3.200000        2.400000        1.600000        0.800000        0.000000

Vector producto:
0.0000000.8000001.6000002.4000003.2000004.0000004.8000005.600000
Vector resultado:
0.0000000.0000000.6400002.5600006.40000012.80000022.40000035.840000
Tiempo: 0.00000011.9
[a1estudiante23@atcgrid ejer7]$ srun -p ac -A ac ./pmtv-secuencial 11
Matriz m[0][0] = 0.000000 <--> m[10][10] = 0.000000
V1[0] = 0.000000 <--> V1[10] = 11.000000
Resultado V3[0] = 0.000000 <--> V3[10] = 199.650000
Tiempo: 0.00000011.9
```

RESPUESTA:

El número total de operaciones se reduce a la mitad en comparación con el producto de la matriz por un vector ya que solo trabajamos con la mitad de la matriz. Realmente esto no afecta de manera sustancial al orden de complejidad puesto que en ambos el orden de complejidad es de O(n^2), sólo que en el caso de la matriz triangular inferior existiría una constante de 1/2. Algo que, en términos asintóticos, no afecta mucho.

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 <stdio.h>
#include <omp.h>
#include <stdlib.h>
#include <time.h>


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

    double tiempo;
    double inicio, final;

    if (argc != 2) {
        printf("Argumentos incorrectos. Uso: binario <tamaño>");
        exit(-1);
    }

    int n = atoi(argv[1]);

//reserva de memoria dinamica
    double** m = (double**)malloc(n * sizeof(double *));
    double* v1 = (double*)malloc(n * sizeof(double));
    double* v3 = (double*)malloc(n * sizeof(double));

    for (int i = 0; i < n; i++) {
        m[i] = (double*)malloc(n * sizeof(double));
    }

#pragma omp parallel
{
    //inicializacion de datos de la matriz y los vectores
    #pragma omp master
    {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i < j)
                    m[i][j] = 0;    // solo inicializamos los valores del triangulo inferior
                else
                    m[i][j] = drand48();
            }
            v1[i] = drand48();
            v3[i] = 0;
        }
    }

    #pragma omp barrier
```

```
    // calculo
    #pragma omp single
    inicio = omp_get_wtime();

    #pragma omp for schedule(runtime)
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i > j)
                v3[i] += m[i][j] * v1[j];
        }
    }

    #pragma omp single
    {
        final = omp_get_wtime();
        tiempo = final - inicio;
    }

}
    // muestra de resultados
    if (n <= 10) {
        printf("Matriz: \n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                printf("%f\t", m[i][j]);
            }
            printf("\n");
        }
        printf("\n");
        printf("Vector producto: \n");
        for (int i = 0; i < n; i++)
            printf("%f", v1[i]);
        printf("\n");

        printf("Vector resultado: \n");
        for (int i = 0; i < n; i++) {
            printf("%f", v3[i]);
        }
        printf("\n");

    }
    else {
        printf("Matriz m[0][0] = %f <--> m[%d][%d] = %f\n",
            m[0][0], n - 1, n - 1, m[n - 1][n - 1]);
        printf("V1[0] = %f <--> V1[%d] = %f\n",
            v1[0], n - 1, v1[n - 1]);
        printf("Resultado V3[0] = %f <--> V3[%d] = %f\n",
            v3[0], n - 1, v3[n - 1]);
    }
    printf("Tiempo: %f11.9\n", tiempo);
//liberacion de memoria dinamica
    for (int i = 0; i < n; i++)
        free(m[i]);
    free(m);
    free(v1);
    free(v3);
}
```

DESCOMPOSICIÓN DE DOMINIO:

Si establecemos una planificación estática con tamaño 2 de chunk obtendríamos una descomposición de dominio similar a la que nos encontramos en la lección 5.

$$c = A \bullet b; \quad c_i = \sum_{k=0}^{M-1} a_{ik} \bullet b_k = a_i^T \bullet b, \quad c(i) = \sum_{k=0}^{M-1} A(i,k) \bullet b(k), \; i = 0,\ldots N-1$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} & a_{86} \end{pmatrix}_{8x6} \bullet \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} \quad \begin{matrix} N=8 \\ M=6 \end{matrix} \quad = \quad \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{pmatrix} \quad \begin{matrix} c(1) = \sum_{k=0}^{M-1} A(1,k) \bullet b(k) \\ c(2) = \sum_{k=0}^{M-1} A(2,k) \bullet b(k) \\ c(3) = \sum_{k=0}^{M-1} A(3,k) \bullet b(k) \\ c(4) = \sum_{k=0}^{M-1} A(4,k) \bullet b(k) \\ c(5) = \sum_{k=0}^{M-1} A(5,k) \bullet b(k) \\ c(6) = \sum_{k=0}^{M-1} A(6,k) \bullet b(k) \\ c(7) = \sum_{k=0}^{M-1} A(7,k) \bullet b(k) \\ c(8) = \sum_{k=0}^{M-1} A(8,k) \bullet b(k) \end{matrix}$$

Sin embargo, en nuestro caso no sería exactamente igual puesto que solo tratamos con el triángulo inferior de la matriz por lo que todas las operaciones del triángulo superior no se realizan.

CAPTURAS DE PANTALLA:

```
[alestudiante23@atcgrid ejer8]$ g++ -o pmtv-openMP pmtv-openMP.c -O2 -fopenmp
[alestudiante23@atcgrid ejer8]$ srun -pac -Aac ./pmtv-openMP 8
Matriz:
0.000000        0.000000        0.000000        0.000000        0.000000        0.000000        0.000000        0.000000
0.041631        0.176643        0.000000        0.000000        0.000000        0.000000        0.000000        0.000000
0.091331        0.092298        0.487217        0.000000        0.000000        0.000000        0.000000        0.000000
0.454433        0.233178        0.831292        0.931731        0.000000        0.000000        0.000000        0.000000
0.556094        0.050832        0.767051        0.018915        0.252360        0.000000        0.000000        0.000000
0.875981        0.531557        0.920261        0.515431        0.810429        0.188420        0.000000        0.000000
0.570614        0.076775        0.815274        0.984891        0.118352        0.893906        0.784484        0.000000
0.253311        0.019842        0.378377        0.678877        0.680923        0.752707        0.006229        0.624407

Vector producto:
0.0009850.3646020.5267500.5680600.2981970.8863140.1009160.126462
Vector resultado:
0.0000000.0000410.0337420.5233480.4338711.2138811.8450511.463250
Tiempo: 0.00000811.9
[alestudiante23@atcgrid ejer8]$ srun -pac -Aac ./pmtv-openMP 11
Matriz m[0][0] = 0.000000 <--> m[10][10] = 0.871689
V1[0] = 0.000985 <--> V1[10] = 0.577967
Resultado V3[0] = 0.000000 <--> V3[10] = 1.761678
Tiempo: 0.00000811.9
```

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:
Pues sería una progresión en la que el primer thread ejecuta 1 operación, el segundo 2 … así sucesivamente de manera que el thread n ejecuta n operaciones. Esto es así puesto que se reparte por filas y las operaciones de cada fila va a aumentando en 1 por cada fila al utilizar solo el triángulo inferior.

(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:
En el caso de dynamic ocurrirá lo mismo que static sólo que se realiza en tiempo de ejecución. Para el caso de guided se producirá una mejor distribución de la carga de trabajo puesto que a medida que se van iterando sobre filas se va dividiendo en más trozos (chunks más pequeños) de manera que se compensa el crecimiento de operaciones por fila.

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

RESPUESTA:
Guided ofrece las mejores prestaciones porque compensa el crecimiento de operaciones al dividirlo en cada iteración en chunks más pequeños. Además lo podemos comprobar empíricamente y obtenemos el siguiente resultado, confirmando lo que hemos explicado en estos apartados:

```
[a1estudiante23@atcgrid ejer8]$ export OMP_SCHEDULE="static,1"
[a1estudiante23@atcgrid ejer8]$ srun -pac -Aac ./pmtv-openMP 10000
Matriz m[0][0] = 0.000000 <--> m[9999][9999] = 0.190485
V1[0] = 0.000985 <--> V1[9999] = 0.735385
Resultado V3[0] = 0.000000 <--> V3[9999] = 2502.491235
Tiempo: 0.19339611.9
[a1estudiante23@atcgrid ejer8]$ export OMP_SCHEDULE="dynamic,1"
[a1estudiante23@atcgrid ejer8]$ srun -pac -Aac ./pmtv-openMP 10000
Matriz m[0][0] = 0.000000 <--> m[9999][9999] = 0.190485
V1[0] = 0.000985 <--> V1[9999] = 0.735385
Resultado V3[0] = 0.000000 <--> V3[9999] = 2502.491235
Tiempo: 0.19195511.9
[a1estudiante23@atcgrid ejer8]$ export OMP_SCHEDULE="guided,1"
[a1estudiante23@atcgrid ejer8]$ srun -pac -Aac ./pmtv-openMP 10000
Matriz m[0][0] = 0.000000 <--> m[9999][9999] = 0.190485
V1[0] = 0.000985 <--> V1[9999] = 0.735385
Resultado V3[0] = 0.000000 <--> V3[9999] = 2502.491235
Tiempo: 0.13736011.9
```

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 <stdio.h>
#include <omp.h>
#include <stdlib.h>
#include <time.h>


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

    double tiempo;
    double inicio, final;

    if (argc != 2) {
        printf("Argumentos incorrectos. Uso: binario <tamaño>");
        exit(-1);
    }

    int n = atoi(argv[1]);

//reserva de memoria dinamica
    double** m = (double**)malloc(n * sizeof(double *));
    double* v1 = (double*)malloc(n * sizeof(double));
    double* v3 = (double*)malloc(n * sizeof(double));

    for (int i = 0; i < n; i++) {
        m[i] = (double*)malloc(n * sizeof(double));
    }

#pragma omp parallel
{
    //inicializacion de datos de la matriz y los vectores
    #pragma omp master
    {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i < j)
                    m[i][j] = 0;    // solo inicializamos los valores del triangulo inferior
                else
                    m[i][j] = drand48();
            }
            v1[i] = drand48();
            v3[i] = 0;
        }
    }

    #pragma omp barrier
```

```
    // calculo
    #pragma omp single
    inicio = omp_get_wtime();

    #pragma omp for schedule(runtime)
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i > j)
                v3[i] += m[i][j] * v1[j];
        }
    }

    #pragma omp single
    {
        final = omp_get_wtime();
        tiempo = final - inicio;
    }

}
    // muestra de resultados
    if (n <= 10) {
        printf("Matriz: \n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                printf("%f\t", m[i][j]);
            }
            printf("\n");
        }
        printf("\n");
        printf("Vector producto: \n");
        for (int i = 0; i < n; i++)
            printf("%f", v1[i]);
        printf("\n");

        printf("Vector resultado: \n");
        for (int i = 0; i < n; i++) {
            printf("%f", v3[i]);
        }
        printf("\n");

    }
    else {
        printf("Matriz m[0][0] = %f <--> m[%d][%d] = %f\n",
            m[0][0], n - 1, n - 1, m[n - 1][n - 1]);
        printf("V1[0] = %f <--> V1[%d] = %f\n",
            v1[0], n - 1, v1[n - 1]);
        printf("Resultado V3[0] = %f <--> V3[%d] = %f\n",
            v3[0], n - 1, v3[n - 1]);
    }
    printf("Tiempo: %f11.9\n", tiempo);
//liberacion de memoria dinamica
    for (int i = 0; i < n; i++)
        free(m[i]);
    free(m);
    free(v1);
    free(v3);
}
```

Usamos el mismo código pero utilizando un script que lance el programa con las distintas opciones de planificación.

DESCOMPOSICIÓN DE DOMINIO:

Es la misma que la del ejercicio 8 puesto que no hemos cambiado nada. Sólo cambia en el caso de guided,

que en cada iteración se va repartiendo mejor las operaciones entre las distintas hebras.

CAPTURAS DE PANTALLA:

Con tamaño de chunk por defecto:

```
[a1estudiante23@atcgrid ejer10]$ sbatch -pac -Aac -n1 -c12 --hint=nomultithread ./pmtv-OpenMP_atcgrid.sh 1
Submitted batch job 105933
[a1estudiante23@atcgrid ejer10]$ cat slurm-105933.out
Id. usuario del trabajo: a1estudiante23
Id. del trabajo: 105933
Nombre del trabajo especificado por usuario: SumaLocal
Directorio de trbajo (en el que se ejecuta el script): /home/a1estudiante23/bp3/ejer10
Cola: ac
Nodo que ejecuta este trabajo: atcgrid.ugr.es
N° de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid2
CPUs por nodo: 24
PLANIFICACIÓN CON CHUNKS POR DEFECTO TAMAÑO 11520
STATIC
Matriz m[0][0] = 0.000000 <--> m[11519][11519] = 0.606002
V1[0] = 0.000985 <--> V1[11519] = 0.387567
Resultado V3[0] = 0.000000 <--> V3[11519] = 2865.917184
Tiempo: 0.08937611.9
DYNAMIC
Matriz m[0][0] = 0.000000 <--> m[11519][11519] = 0.606002
V1[0] = 0.000985 <--> V1[11519] = 0.387567
Resultado V3[0] = 0.000000 <--> V3[11519] = 2865.917184
Tiempo: 0.06160411.9
GUIDED
Matriz m[0][0] = 0.000000 <--> m[11519][11519] = 0.606002
V1[0] = 0.000985 <--> V1[11519] = 0.387567
Resultado V3[0] = 0.000000 <--> V3[11519] = 2865.917184
Tiempo: 0.03489311.9
PLANIFICACIÓN CON CHUNKS POR DEFECTO TAMAÑO 23040
STATIC
Matriz m[0][0] = 0.000000 <--> m[23039][23039] = 0.376615
V1[0] = 0.000985 <--> V1[23039] = 0.155590
Resultado V3[0] = 0.000000 <--> V3[23039] = 5782.177710
Tiempo: 0.61514811.9
DYNAMIC
Matriz m[0][0] = 0.000000 <--> m[23039][23039] = 0.376615
V1[0] = 0.000985 <--> V1[23039] = 0.155590
Resultado V3[0] = 0.000000 <--> V3[23039] = 5782.177710
Tiempo: 0.44727211.9
GUIDED
Matriz m[0][0] = 0.000000 <--> m[23039][23039] = 0.376615
V1[0] = 0.000985 <--> V1[23039] = 0.155590
Resultado V3[0] = 0.000000 <--> V3[23039] = 5782.177710
Tiempo: 0.42140311.9
```

Con tamaño de chunk 1:

```
[a1estudiante23@atcgrid ejer10]$ sbatch -pac -Aac -n1 -c12 --hint=nomultithread ./pmtv-OpenMP_atcgrid.sh 2
Submitted batch job 105934
[a1estudiante23@atcgrid ejer10]$ cat slurm-105934.out
Id. usuario del trabajo: a1estudiante23
Id. del trabajo: 105934
Nombre del trabajo especificado por usuario: SumaLocal
Directorio de trbajo (en el que se ejecuta el script): /home/a1estudiante23/bp3/ejer10
Cola: ac
Nodo que ejecuta este trabajo: atcgrid.ugr.es
N° de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
CPUs por nodo: 24
PLANIFICACIÓN CON CHUNKS 1 TAMAÑO 11520
STATIC
Matriz m[0][0] = 0.000000 <--> m[11519][11519] = 0.606002
V1[0] = 0.000985 <--> V1[11519] = 0.387567
Resultado V3[0] = 0.000000 <--> V3[11519] = 2865.917184
Tiempo: 0.06663811.9
DYNAMIC
Matriz m[0][0] = 0.000000 <--> m[11519][11519] = 0.606002
V1[0] = 0.000985 <--> V1[11519] = 0.387567
Resultado V3[0] = 0.000000 <--> V3[11519] = 2865.917184
Tiempo: 0.06324511.9
GUIDED
Matriz m[0][0] = 0.000000 <--> m[11519][11519] = 0.606002
V1[0] = 0.000985 <--> V1[11519] = 0.387567
Resultado V3[0] = 0.000000 <--> V3[11519] = 2865.917184
Tiempo: 0.03780911.9
PLANIFICACIÓN CON CHUNKS 1 TAMAÑO 23040
STATIC
Matriz m[0][0] = 0.000000 <--> m[23039][23039] = 0.376615
V1[0] = 0.000985 <--> V1[23039] = 0.155590
Resultado V3[0] = 0.000000 <--> V3[23039] = 5782.177710
Tiempo: 0.41843311.9
DYNAMIC
Matriz m[0][0] = 0.000000 <--> m[23039][23039] = 0.376615
V1[0] = 0.000985 <--> V1[23039] = 0.155590
Resultado V3[0] = 0.000000 <--> V3[23039] = 5782.177710
Tiempo: 0.43265111.9
GUIDED
Matriz m[0][0] = 0.000000 <--> m[23039][23039] = 0.376615
V1[0] = 0.000985 <--> V1[23039] = 0.155590
Resultado V3[0] = 0.000000 <--> V3[23039] = 5782.177710
Tiempo: 0.41873911.9
```

Con tamaño de chunk 64:

```
[a1estudiante23@atcgrid ejer10]$ sbatch -pac -Aac -n1 -c12 --hint=nomultithread ./pmtv-OpenMP_atcgrid.sh 3
Submitted batch job 105935
[a1estudiante23@atcgrid ejer10]$ cat slurm-105935.out
Id. usuario del trabajo: a1estudiante23
Id. del trabajo: 105935
Nombre del trabajo especificado por usuario: SumaLocal
Directorio de trbajo (en el que se ejecuta el script): /home/a1estudiante23/bp3/ejer10
Cola: ac
Nodo que ejecuta este trabajo: atcgrid.ugr.es
N° de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
CPUs por nodo: 24
PLANIFICACIÓN CON CHUNKS 64 TAMAÑO 11520
STATIC
Matriz m[0][0] = 0.000000 <--> m[11519][11519] = 0.606002
V1[0] = 0.000985 <--> V1[11519] = 0.387567
Resultado V3[0] = 0.000000 <--> V3[11519] = 2865.917184
Tiempo: 0.05080611.9
DYNAMIC
Matriz m[0][0] = 0.000000 <--> m[11519][11519] = 0.606002
V1[0] = 0.000985 <--> V1[11519] = 0.387567
Resultado V3[0] = 0.000000 <--> V3[11519] = 2865.917184
Tiempo: 0.04611911.9
GUIDED
Matriz m[0][0] = 0.000000 <--> m[11519][11519] = 0.606002
V1[0] = 0.000985 <--> V1[11519] = 0.387567
Resultado V3[0] = 0.000000 <--> V3[11519] = 2865.917184
Tiempo: 0.05410811.9
PLANIFICACIÓN CON CHUNKS 64 TAMAÑO 23040
STATIC
Matriz m[0][0] = 0.000000 <--> m[23039][23039] = 0.376615
V1[0] = 0.000985 <--> V1[23039] = 0.155590
Resultado V3[0] = 0.000000 <--> V3[23039] = 5782.177710
Tiempo: 0.35041611.9
DYNAMIC
Matriz m[0][0] = 0.000000 <--> m[23039][23039] = 0.376615
V1[0] = 0.000985 <--> V1[23039] = 0.155590
Resultado V3[0] = 0.000000 <--> V3[23039] = 5782.177710
Tiempo: 0.33852911.9
GUIDED
Matriz m[0][0] = 0.000000 <--> m[23039][23039] = 0.376615
V1[0] = 0.000985 <--> V1[23039] = 0.155590
Resultado V3[0] = 0.000000 <--> V3[23039] = 5782.177710
Tiempo: 0.39781111.9
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

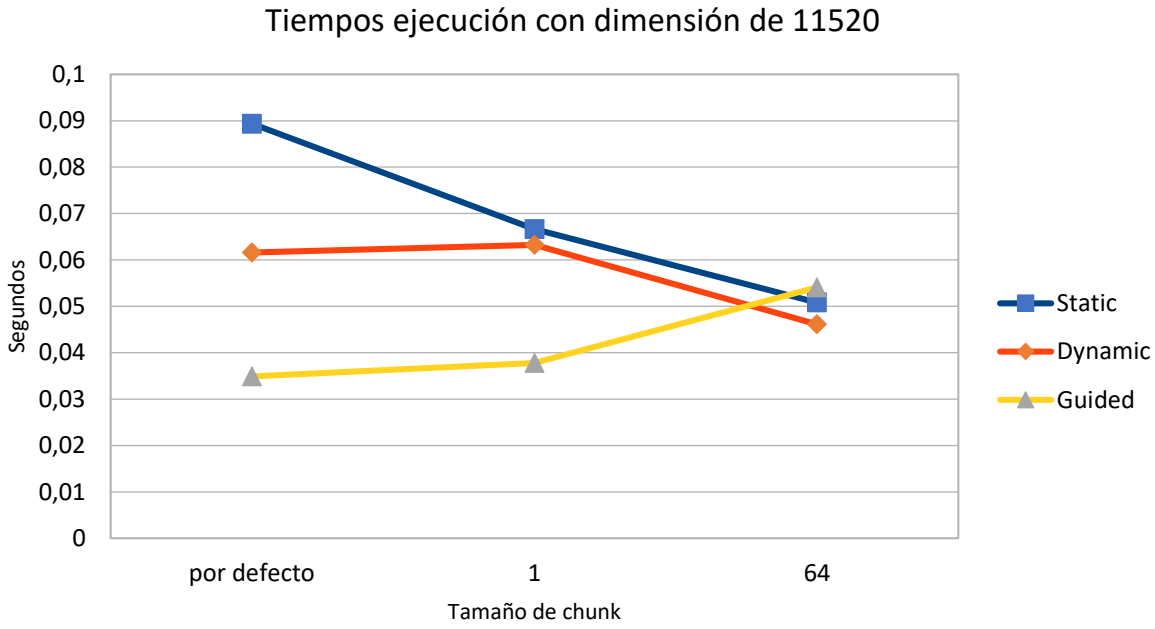
Me di cuenta de una errata en el código que hacía que al final de cada tiempo agregaba un "11.9" simplemente para rellenar los datos de la tabla he cogido los datos obtenidos en esta ejecución pero eliminando ese 11.9 del final.

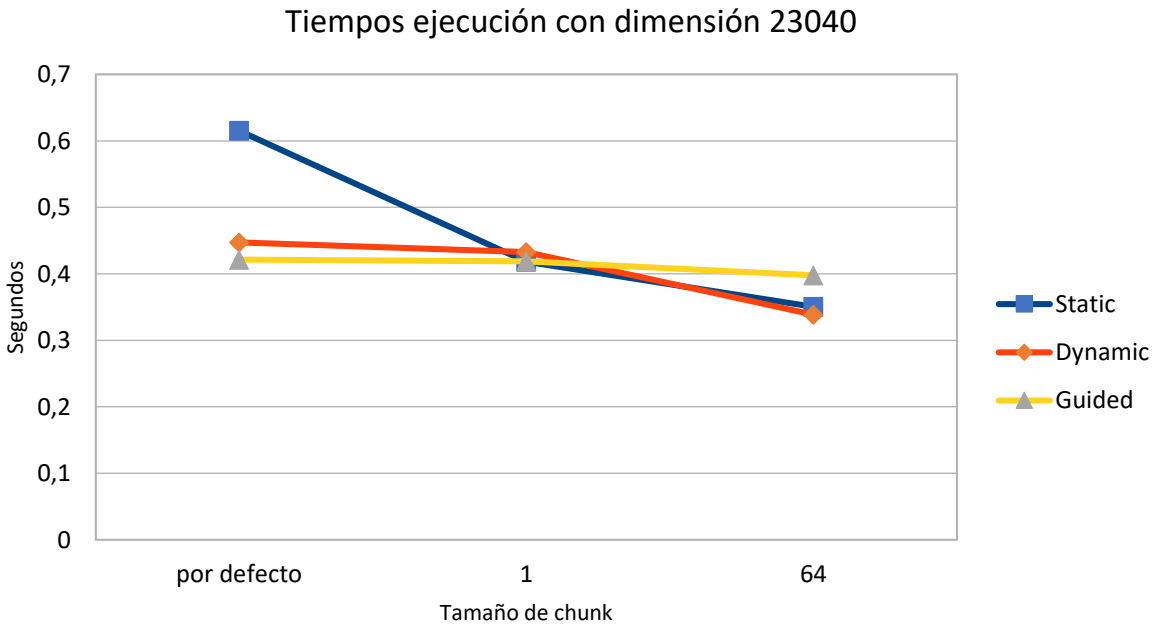
Tabla 2 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector para vectores de tamaño N= 11520 (solo se ha paralelizado el producto, no la inicialización de los datos).

Chunk	Static	Dynamic	Guided
por defecto	0,089376	0,061604	0,034893
1	0,066638	0,063245	0,037809
64	0,050806	0,046119	0,054108

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector para vectores de tamaño N= 23040 (solo se ha paralelizado el producto, no la inicialización de los datos).

Chunk	Static	Dynamic	Guided
por defecto	0,615148	0,447272	0,421403
1	0,418433	0,432651	0,418739
64	0,350416	0,338529	0,397811





SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash
#Órdenes para el Gestor de carga de un trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=SumaLocal
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac

#Obtener información de las variables del entorno del Gestor de carga de trabajo:
echo "Id. usuario del trabajo: $SLURM_JOB_USER"
echo "Id. del trabajo: $SLURM_JOBID"
echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
echo "Directorio de trbajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR"
echo "Cola: $SLURM_JOB_PARTITION"
echo "Nodo que ejecuta este trabajo: $SLURM_SUBMIT_HOST"
echo "N° de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
#Instrucciones del script para ejecutar código:

# PLANIFICACIÓN POR DEFECTO
if [[ $1 -eq 1 ]]; then
    echo PLANIFICACIÓN CON CHUNKS POR DEFECTO TAMAÑO 11520
    echo STATIC
    export OMP_SCHEDULE="monotonic:static"
    srun ./pmtv-OpenMP 11520
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic"
    srun ./pmtv-OpenMP 11520
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided"
    srun ./pmtv-OpenMP 11520

    echo PLANIFICACIÓN CON CHUNKS POR DEFECTO TAMAÑO 23040
    echo STATIC
    export OMP_SCHEDULE="monotonic:static"
    srun ./pmtv-OpenMP 23040
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic"
    srun ./pmtv-OpenMP 23040
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided"
    srun ./pmtv-OpenMP 23040
fi

# PLANIFICACIÓN CON CHUNK DE TAMAÑO 1
if [[ $1 -eq 2 ]]; then
    echo PLANIFICACIÓN CON CHUNKS 1 TAMAÑO 11520
    echo STATIC
    export OMP_SCHEDULE="monotonic:static,1"
    srun ./pmtv-OpenMP 11520
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic,1"
    srun ./pmtv-OpenMP 11520
```

```
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided,1"
    srun ./pmtv-OpenMP 11520

    echo PLANIFICACIÓN CON CHUNKS 1 TAMAÑO 23040
    echo STATIC
    export OMP_SCHEDULE="monotonic:static,1"
    srun ./pmtv-OpenMP 23040
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic,1"
    srun ./pmtv-OpenMP 23040
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided,1"
    srun ./pmtv-OpenMP 23040
fi

# PLANIFICACIÓN DE CHUNK DE TAMAÑO 64
if [[ $1 -eq 3 ]]; then
    echo PLANIFICACIÓN CON CHUNKS 64 TAMAÑO 11520
    echo STATIC
    export OMP_SCHEDULE="monotonic:static,64"
    srun ./pmtv-OpenMP 11520
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic,64"
    srun ./pmtv-OpenMP 11520
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided,64"
    srun ./pmtv-OpenMP 11520

    echo PLANIFICACIÓN CON CHUNKS 64 TAMAÑO 23040
    echo STATIC
    export OMP_SCHEDULE="monotonic:static,64"
    srun ./pmtv-OpenMP 23040
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic,64"
    srun ./pmtv-OpenMP 23040
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided,64"
    srun ./pmtv-OpenMP 23040
fi
```

Captura del script:

```
#!/bin/bash
#Órdenes para el Gestor de carga de un trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=SumaLocal
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac

#Obtener información de las variables del entorno del Gestor de carga de trabajo:
echo "Id. usuario del trabajo: $SLURM_JOB_USER"
echo "Id. del trabajo: $SLURM_JOBID"
echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
echo "Directorio de trbajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR"
echo "Cola: $SLURM_JOB_PARTITION"
echo "Nodo que ejecuta este trabajo: $SLURM_SUBMIT_HOST"
echo "N° de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
#Instrucciones del script para ejecutar código:

# PLANIFICACIÓN POR DEFECTO
if [[ $1 -eq 1 ]]; then
    echo PLANIFICACIÓN CON CHUNKS POR DEFECTO TAMAÑO 11520
    echo STATIC
    export OMP_SCHEDULE="monotonic:static"
    srun ./pmtv-OpenMP 11520
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic"
    srun ./pmtv-OpenMP 11520
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided"
    srun ./pmtv-OpenMP 11520

    echo PLANIFICACIÓN CON CHUNKS POR DEFECTO TAMAÑO 23040
    echo STATIC
    export OMP_SCHEDULE="monotonic:static"
    srun ./pmtv-OpenMP 23040
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic"
    srun ./pmtv-OpenMP 23040
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided"
    srun ./pmtv-OpenMP 23040
fi
```

```
# PLANIFICACIÓN CON CHUNK DE TAMAÑO 1
if [[ $1 -eq 2 ]]; then
    echo PLANIFICACIÓN CON CHUNKS 1 TAMAÑO 11520
    echo STATIC
    export OMP_SCHEDULE="monotonic:static,1"
    srun ./pmtv-OpenMP 11520
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic,1"
    srun ./pmtv-OpenMP 11520
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided,1"
    srun ./pmtv-OpenMP 11520

    echo PLANIFICACIÓN CON CHUNKS 1 TAMAÑO 23040
    echo STATIC
    export OMP_SCHEDULE="monotonic:static,1"
    srun ./pmtv-OpenMP 23040
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic,1"
    srun ./pmtv-OpenMP 23040
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided,1"
    srun ./pmtv-OpenMP 23040
fi

# PLANIFICACIÓN DE CHUNK DE TAMAÑO 64
if [[ $1 -eq 3 ]]; then
    echo PLANIFICACIÓN CON CHUNKS 64 TAMAÑO 11520
    echo STATIC
    export OMP_SCHEDULE="monotonic:static,64"
    srun ./pmtv-OpenMP 11520
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic,64"
    srun ./pmtv-OpenMP 11520
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided,64"
    srun ./pmtv-OpenMP 11520

    echo PLANIFICACIÓN CON CHUNKS 64 TAMAÑO 23040
    echo STATIC
    export OMP_SCHEDULE="monotonic:static,64"
    srun ./pmtv-OpenMP 23040
    echo DYNAMIC
    export OMP_SCHEDULE="monotonic:dynamic,64"
    srun ./pmtv-OpenMP 23040
    echo GUIDED
    export OMP_SCHEDULE="monotonic:guided,64"
    srun ./pmtv-OpenMP 23040
fi
```