

2° curso / 2° cuatr.

Grados 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):

Grupo de prácticas:

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

```

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

int main(int argc, char **argv)
{
    int i, n=20, tid, x = 8;
    int a[n], suma=0, sumalocal;

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

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

    x = atoi(argv[2]);
    if(x > 8)
        x = 8;

    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:

```

joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ gcc -O2 -fopenmp if-clause.c -o if-clause
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ ./if-clause 5 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 1 suma de a[3]=3 sumalocal=3
thread 1 suma de a[4]=4 sumalocal=7
thread master=0 imprime suma=10
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ ./if-clause 7 3
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 1 suma de a[3]=3 sumalocal=3
thread 1 suma de a[4]=4 sumalocal=7
thread 2 suma de a[5]=5 sumalocal=5
thread 2 suma de a[6]=6 sumalocal=11
thread master=0 imprime suma=21
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ █

```

RESPUESTA: Este programa inicializa un vector de tamaño n (pasado como parámetro, como máximo 20) y calcula la suma de todos los valores de sus componentes. En caso de ser n mayor que 4, esta suma se realiza en paralelo. Al usar `num_threads()`, estamos imponiendo el número de hebras a usar en caso de que se ejecute en paralelo ($n > 4$), y en la captura de arriba podemos ver cómo el resultado obtenido es correcto para distintos valores de n mayores que 4, además de que se usan el número de hebras impuesto como segundo parámetro.

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 ($x = 1$ y 2).

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="non-monotonic:static,2"`). En la segunda fila, 1 y 2 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 | 1 | 2 | 1 | 2 |
| 1 | 1 | 0 | 1 | 0 | 2 | 2 | 1 | 2 |
| 2 | 2 | 1 | 2 | 1 | 0 | 0 | 1 | 2 |
| 3 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 2 |
| 4 | 1 | 2 | 1 | 2 | 0 | 1 | 1 | 2 |
| 5 | 2 | 2 | 2 | 2 | 0 | 1 | 1 | 2 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 7 | 1 | 0 | 1 | 0 | 0 | 0 | 2 | 0 |
| 8 | 2 | 1 | 2 | 1 | 0 | 0 | 2 | 0 |
| 9 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | 0 |
| 10 | 1 | 2 | 1 | 2 | 0 | 1 | 0 | 1 |
| 11 | 2 | 2 | 2 | 2 | 0 | 1 | 0 | 1 |

| | | | | | | | | |
|----|---|---|---|---|---|---|---|---|
| 12 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 2 |
| 13 | 1 | 0 | 1 | 0 | 0 | 2 | 0 | 2 |
| 14 | 2 | 1 | 2 | 1 | 0 | 2 | 0 | 2 |
| 15 | 0 | 1 | 0 | 1 | 0 | 2 | 0 | 2 |

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:

Static: Las iteraciones se distribuyen en tiempo de compilación, dividiéndose en chunks de manera que las hebras se reparten las iteraciones de chunks en chunks.

Dynamic: Las iteraciones se distribuyen en tiempo de ejecución, de ahí que si ejecutamos varias veces el programa, la asignación vaya cambiando (las hebras más rápidas ejecutarán más trabajo).

Guided: Las iteraciones se distribuyen en tiempo de ejecución. La primera hebra en ejecutar se encarga de n° iteraciones/ n° hebras. Cuando se cambia de hebra, la escogida se encarga de n° iter. que quedan/ n° hebras, y así consecutivamente, hasta que se llegue al chunk impuesto, a partir del cual siempre se ejecutarán chunk iteraciones por hebra.

Monotonic: Si a una hebra le toca ejecutar la iteración i , ya no puede ejecutar iteraciones de menor índice que i , sino solo superiores.

Nonmonotonic: El efecto de `monotonic` se anula.

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: Usa un chunk de 1 unidad por defecto, mientras que `modifier` es `monotonic`.

Dynamic: Usa un chunk de 1 una unidad por defecto, mientras que `modifier` es `nonmonotonic`.

Guided: En caso de que el chunk no haya sido indicado, sabemos que cada hebra va ejecutando un n° de iteraciones determinado según lo explicado en el ejercicio anterior, hasta que ese cálculo llegue a una cifra menor que uno, entonces se usaría como chunk por defecto 1. En cuanto al `modifier`, sería `nonmonotonic`.

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`


```

#pragma omp parallel for firstprivate(suma) \
    lastprivate(suma) schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
    if(i == 0){
        int dyn_var = omp_get_dynamic();
        int nthreads_var = omp_get_max_threads();
        int thread_limit_var = omp_get_thread_limit();
        int modifier;
        omp_sched_t kind;
        omp_get_schedule(&kind,&modifier);

        printf("Dinámica (true 0, false 1): %d\n",dyn_var);
        printf("Nºthreads: %d\n",nthreads_var);
        printf("Límite de threads: %d\n",thread_limit_var);
        printf("Kind (Static 1, Dynamic 2, Guided 3): %d\n", kind);
        printf("Chunk: %d\n\n", modifier);
    }
    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 for': \n");

int dyn_var = omp_get_dynamic();
int nthreads_var = omp_get_max_threads();
int thread_limit_var = omp_get_thread_limit();
int modifier;
omp_sched_t kind;
omp_get_schedule(&kind,&modifier);

printf("Dinámica (true 0, false 1): %d\n",dyn_var);
printf("Nºthreads: %d\n",nthreads_var);
printf("Límite de threads: %d\n",thread_limit_var);
printf("Kind (Static 1, Dynamic 2, Guided 3): %d\n", kind);
printf("Chunk: %d\n", modifier);

```

CAPTURAS DE PANTALLA:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ export OMP_NUM_THREADS=4
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ export OMP_SCHEDULE="nonmonotonic:static,1"
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ ./scheduled-clause 12 3
Dinámica (true 0, false 1): 0
Nºthreads: 4
Límite de threads: 2147483647
Kind (Static 1, Dynamic 2, Guided 3): 1
Chunk: 1

Fuera de 'parallel for':
Dinámica (true 0, false 1): 0
Nºthreads: 4
Límite de threads: 2147483647
Kind (Static 1, Dynamic 2, Guided 3): 1
Chunk: 1
```

RESPUESTA:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ export OMP_SCHEDULE="nonmonotonic:dynamic,2"
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ ./scheduled-clause 12 3
Dinámica (true 0, false 1): 0
Nºthreads: 4
Límite de threads: 2147483647
Kind (Static 1, Dynamic 2, Guided 3): 2
Chunk: 2

Fuera de 'parallel for':
Dinámica (true 0, false 1): 0
Nºthreads: 4
Límite de threads: 2147483647
Kind (Static 1, Dynamic 2, Guided 3): 2
Chunk: 2
```

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ export OMP_SCHEDULE="nonmonotonic:guided,4"
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ ./scheduled-clause 12 3
Dinámica (true 0, false 1): 0
Nºthreads: 4
Límite de threads: 2147483647
Kind (Static 1, Dynamic 2, Guided 3): 3
Chunk: 4

Fuera de 'parallel for':
Dinámica (true 0, false 1): 0
Nºthreads: 4
Límite de threads: 2147483647
Kind (Static 1, Dynamic 2, Guided 3): 3
Chunk: 4
```


RESPUESTA: No se imprimen valores diferentes porque son variables de entorno, y por lo tanto no son locales a la región donde se imprimen.

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-clauseModificado2.c`

```
#pragma omp parallel for firstprivate(suma) \
                        lastprivate(suma) schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
    if(i == 0){
        printf("Nºprocesadores disponibles: %d\n",omp_get_num_procs());
        printf("Nºthreads: %d\n",omp_get_num_threads());
        printf("En región paralela (True 1, False 0): %d\n\n",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 for': \n");

printf("Nºprocesadores disponibles: %d\n",omp_get_num_procs());
printf("Nºthreads: %d\n",omp_get_num_threads());
printf("En región paralela (True 1, False 0): %d\n",omp_in_parallel());
```

CAPTURAS DE PANTALLA:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ ./scheduled-clause2 12 3
Nºprocesadores disponibles: 8
Nºthreads: 8
En región paralela (True 1, False 0): 1

Fuera de 'parallel for':
Nºprocesadores disponibles: 8
Nºthreads: 1
En región paralela (True 1, False 0): 0
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ export OMP_NUM_THREADS=3
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/A
rquitectura de Computadores/bp3$ ./scheduled-clause2 12 3
Nºprocesadores disponibles: 8
Nºthreads: 3
En región paralela (True 1, False 0): 1

Fuera de 'parallel for':
Nºprocesadores disponibles: 8
Nºthreads: 1
En región paralela (True 1, False 0): 0
```

RESPUESTA: Se obtienen valores distintos en la función `omp_in_parallel()`, que da 0 (false) fuera de la región paralela y da 1 (true) dentro de la región paralela. También en la función `omp_get_num_threads()`, ya que fuera de la región paralela solo se usa una hebra (tambien he cambiado la variable de entorno `nthreads-var` a 3 para ejecutar una segunda vez. El número de procesadores disponibles no cambia.

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-clauseModificado3.c`

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

        int modifier;
        omp_sched_t kind;
        omp_get_schedule(&kind,&modifier);

        printf("REGIÓN PARALELA ANTES DE MODIFICAR\n");
        printf("Dinámica (true 0, false 1): %d\n",omp_get_dynamic());
        printf("Nºthreads: %d\n",omp_get_max_threads());
        printf("Kind (Static 1, Dynamic 2, Guided 3): %d\n", kind);
        printf("Chunk: %d\n\n", modifier);

        omp_set_dynamic(5);
        omp_set_num_threads(6);
        omp_set_schedule(3,2);
        omp_get_schedule(&kind,&modifier);

        printf("REGIÓN PARALELA DESPUÉS DE MODIFICAR\n");
        printf("Dinámica (true 0, false 1): %d\n",omp_get_dynamic());
        printf("Nºthreads: %d\n",omp_get_max_threads());
        printf("Kind (Static 1, Dynamic 2, Guided 3): %d\n", kind);
        printf("Chunk: %d\n\n", modifier);
    }
    suma = suma + a[i];
    // printf(" thread %d suma a[%d]=%d suma=%d \n",
    // omp_get_thread_num(),i,a[i],suma);
}
```

CAPTURAS DE PANTALLA:

```

printf("Fuera de 'parallel for': \n");

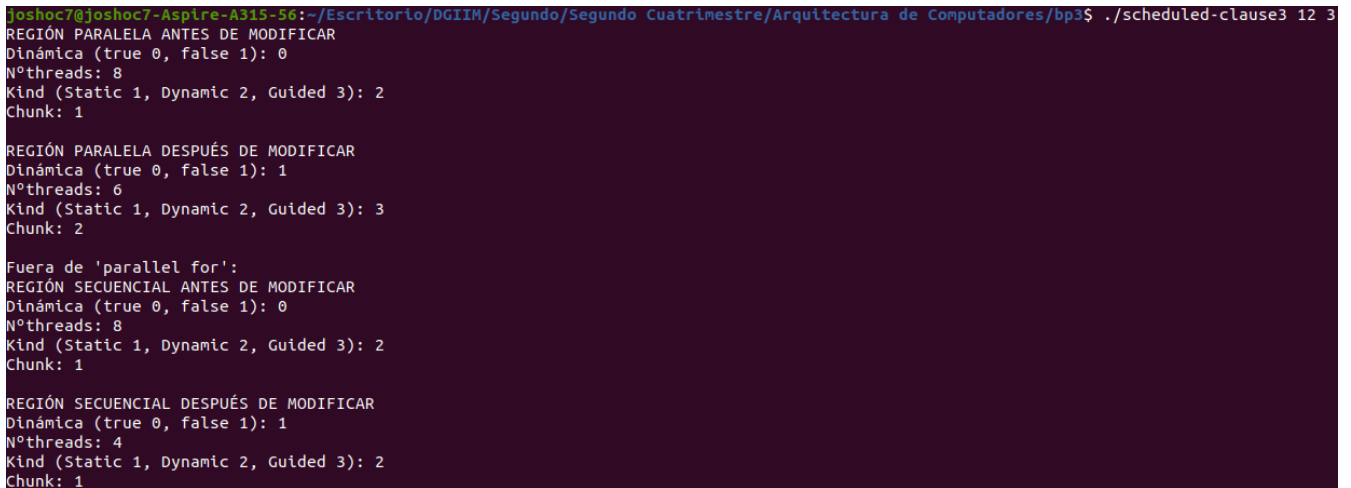
int modifier;
omp_sched_t kind;
omp_get_schedule(&kind,&modifier);

printf("REGIÓN SECUENCIAL ANTES DE MODIFICAR\n");
printf("Dinámica (true 0, false 1): %d\n",omp_get_dynamic());
printf("Nºthreads: %d\n",omp_get_max_threads());
printf("Kind (Static 1, Dynamic 2, Guided 3): %d\n", kind);
printf("Chunk: %d\n\n", modifier);

omp_set_dynamic(4);
omp_set_num_threads(4);
omp_set_schedule(2,1);
omp_get_schedule(&kind,&modifier);

printf("REGIÓN SECUENCIAL DESPUÉS DE MODIFICAR\n");
printf("Dinámica (true 0, false 1): %d\n",omp_get_dynamic());
printf("Nºthreads: %d\n",omp_get_max_threads());
printf("Kind (Static 1, Dynamic 2, Guided 3): %d\n", kind);
printf("Chunk: %d\n\n", modifier);

```



```

joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp3$ ./scheduled-clause3 12 3
REGIÓN PARALELA ANTES DE MODIFICAR
Dinámica (true 0, false 1): 0
Nºthreads: 8
Kind (Static 1, Dynamic 2, Guided 3): 2
Chunk: 1

REGIÓN PARALELA DESPUÉS DE MODIFICAR
Dinámica (true 0, false 1): 1
Nºthreads: 6
Kind (Static 1, Dynamic 2, Guided 3): 3
Chunk: 2

Fuera de 'parallel for':
REGIÓN SECUENCIAL ANTES DE MODIFICAR
Dinámica (true 0, false 1): 0
Nºthreads: 8
Kind (Static 1, Dynamic 2, Guided 3): 2
Chunk: 1

REGIÓN SECUENCIAL DESPUÉS DE MODIFICAR
Dinámica (true 0, false 1): 1
Nºthreads: 4
Kind (Static 1, Dynamic 2, Guided 3): 2
Chunk: 1

```

RESPUESTA: Como podemos ver en la última captura, los valores de dyn_var, nthreads-var, kind y chunk se modifican de forma correcta con respecto a lo que se ha puesto en el código. También hemos visto que fuera de la región paralela, los valores previos al cambio se restauran.

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

7. 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. 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()`.

Anotación: He decidido dejar la inicialización de la matriz y el vector como estaba en la versión secuencial para así asegurarme de que la versión paralela funciona correctamente. Para que el usuario pueda decidir el tipo de schedule con `OMP_SCHEDULE`, añadimos la cláusula `schedule(runtime)` al `parallel for` que paraleliza el bucle de las filas.

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

```
// Inicializar vector y matriz
for (i = 0; i < N; i++){
    v1[i] = 0.1*i;
    v2[i] = 0;
    for (j = 0; j < i; j++)
        m[i][j] = i*N+j;
    for (j = i; j < N; j++)
        m[i][j] = 0;
}

// Calcular v2 = m * v1
clock_gettime(CLOCK_REALTIME,&cgt1);

#pragma omp parallel for schedule(runtime)
for(i = 0; i < N; i++){
    for (j = 0; j < i; j++)
        v2[i] += m[i][j] * v1[j];
}
clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
      (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
```

CAPTURAS DE PANTALLA:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp3$ ./pmtv-secuencial 5
Tiempo: 0.00000451    Tamaño: 5
Matriz:
  0.000000    0.000000    0.000000    0.000000    0.000000
  5.000000    0.000000    0.000000    0.000000    0.000000
 10.000000    11.000000    0.000000    0.000000    0.000000
 15.000000    16.000000    17.000000    0.000000    0.000000
 20.000000    21.000000    22.000000    23.000000    0.000000

Vector:
  0.000000  0.100000  0.200000  0.300000  0.400000

Vector resultado:
  0.000000  0.000000  1.100000  5.000000 13.400000
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp3$ ./pmtv-paralelo 5
Tiempo: 0.008582534    Tamaño: 5
Matriz:
  0.000000    0.000000    0.000000    0.000000    0.000000
  5.000000    0.000000    0.000000    0.000000    0.000000
 10.000000    11.000000    0.000000    0.000000    0.000000
 15.000000    16.000000    17.000000    0.000000    0.000000
 20.000000    21.000000    22.000000    23.000000    0.000000

Vector:
  0.000000  0.100000  0.200000  0.300000  0.400000

Vector resultado:
  0.000000  0.000000  1.100000  5.000000 13.400000
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp3$ ./pmtv-secuencial 3000
Tiempo: 0.008061212    Tamaño: 3000    v2[0]: 0.000000    v2[N-1]: 4045500900349.899902
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/DGIIM/Segundo/Segundo Cuatrimestre/Arquitectura de Computadores/bp3$ ./pmtv-paralelo 3000
Tiempo: 0.004191057    Tamaño: 3000    v2[0]: 0.000000    v2[N-1]: 4045500900349.899902
```

8. 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:

Con esta organización, iteraciones consecutivas les corresponderían a distintas hebras. Debemos tener en cuenta que la primera fila no supondría ninguna multiplicación ni suma, la segunda una multiplicación y ninguna suma, la tercera dos multiplicaciones y una suma... etc. De esta forma, deducimos que si la matriz es de dimensión N , la fila i requerirá i multiplicaciones y $i-1$ sumas para $i \geq 1$. El problema de esta organización es que las hebras que ejecuten el cálculo de los primeros elementos del vector resultado acabarán antes que las que se encargan de los últimos elementos, manteniéndose ociosas.

(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:

Con la asignación `dynamic`, lo explicado en el apartado anterior no ocurrirá, ya que las hebras que acaben con las filas que requieren menos operaciones se les serán asignadas nuevas filas mientras que las hebras que se encargan de las filas más tediosas siguen trabajando.

Con `guided`, partiremos de un número de iteraciones y hebras, por lo que el chunk se irá reduciendo según vayan acabando las hebras, de forma que a unas hebras les tocará más iteraciones que a otras.

(c) ¿Qué alternativa cree que debería ofrecer mejores prestaciones? Razonar la respuesta.

RESPUESTA:

`Dynamic` es la alternativa que debería ofrecernos mejores prestaciones ya que evita que haya hebras ociosas como pasaba en `static`, y además realiza un mejor reparto del trabajo que si fuese `guided`, garantizando así un menor tiempo de ejecución.

9. 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`

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

TABLA RESULTADOS, SCRIPT Y GRÁFICA `atcgrid`

SCRIPT: `pmvt-OpenMP_atcgrid.sh`

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