2º curso / 2º cuatr.

Grado Ing. Inform.

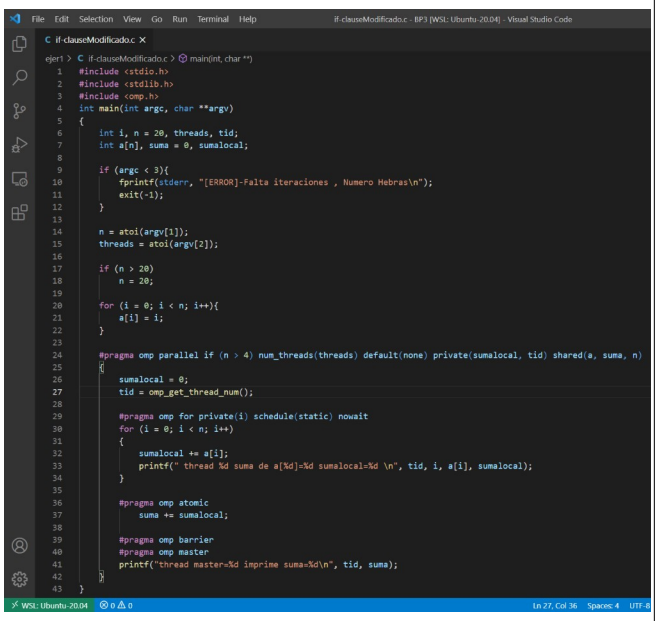
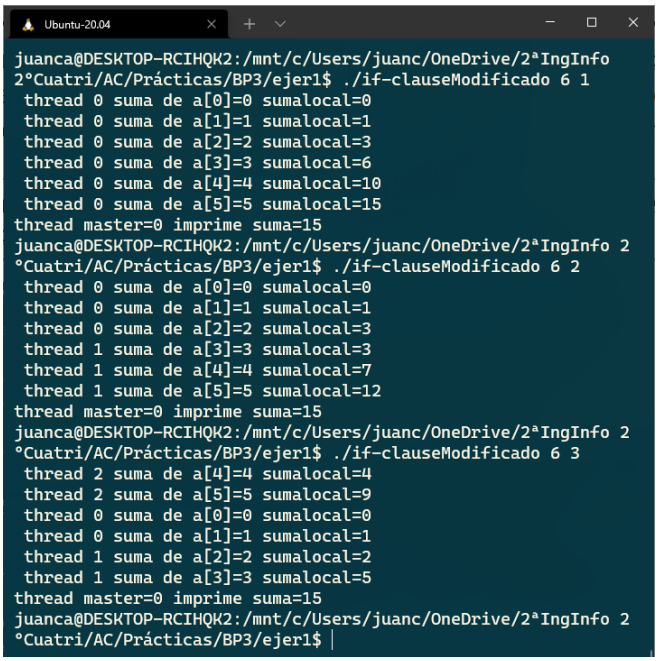
Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos): Juan Carlos Ruiz Fernández
Grupo de prácticas: B3
Fecha de entrega: 25 de mayo del 2021
Fecha evaluación en clase:

---

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`	**CAPTURAS DE PANTALLA:**
	

RESPUESTA:

Al ser un argumento del programa, facilita ver las diferencias entre las ejecuciones para 6 iteraciones con 1, 2 o 3 hebras. La salida muestra como se reparte el trabajo las hebras y que parte de éste: Para una hebra, obviamente, se lleva todo el trabajo; para dos hebras se reparte equitativamente entre ellas y de igual forma con tres hebras. Si se diera un numero de hebra no múltiplo de las iteraciones se repartiría por igual entre las n-1 hebras dejando a solo una hebra con distinta carga restante.

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="non-monotonic: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	2	2	2	2
1	1	0	1	0	1	2	2	2
2	2	1	2	1	0	1	2	2
3	0	1	0	1	0	1	2	2
4	1	2	1	2	1	0	2	2
5	2	2	2	2	1	0	2	2
6	0	0	0	0	1	2	1	1
7	1	0	1	0	1	2	1	1
8	2	1	2	1	1	0	1	1
9	0	1	0	1	1	0	1	1
10	1	2	1	2	1	0	0	0
11	2	2	2	2	1	0	0	0
12	0	0	0	0	1	0	1	2
13	1	0	1	0	1	0	1	2
14	2	1	2	1	1	0	2	0
15	0	1	0	1	1	0	2	0

RESPUESTA:

En esta tabla no se representa la diferencia entre *monotonic* y *nonmonotonic* porque está ordenado en funcion de la iteracion. Entre ellos dos se diferencian en que en que el primero lo realiza en orden creciente y el segundo lo pueden hacer de forma creciente o decreciente durante la ejecucion.

Entre *static, dynamic* y *guided* la diferencia reside en el como y cuando asigna *trabajo a las hebras. Static*, como su nombre indica, ya esta establecido y siempre será igual la distribucion del trabajo ejecucion tras ejecucion. *Dynamic* en cambio lo hace en funcion de velocidad, si una hebra es mas rapida le asigna mas iteraciones. *Guided* cambia el chunk, al principio la granularidad es mas grande y asigna mas tareas a cada hebra y la va reduciendo con forme se acaban, poniendo el limite en el chunk especificado (se aprecia en la ultima columna que hasta la iteracion 9 las hebras 2 y 1 ejecutan muchas mas iteraciones que las especificadas, hasta el resto que se reparte por el chunk = 2).

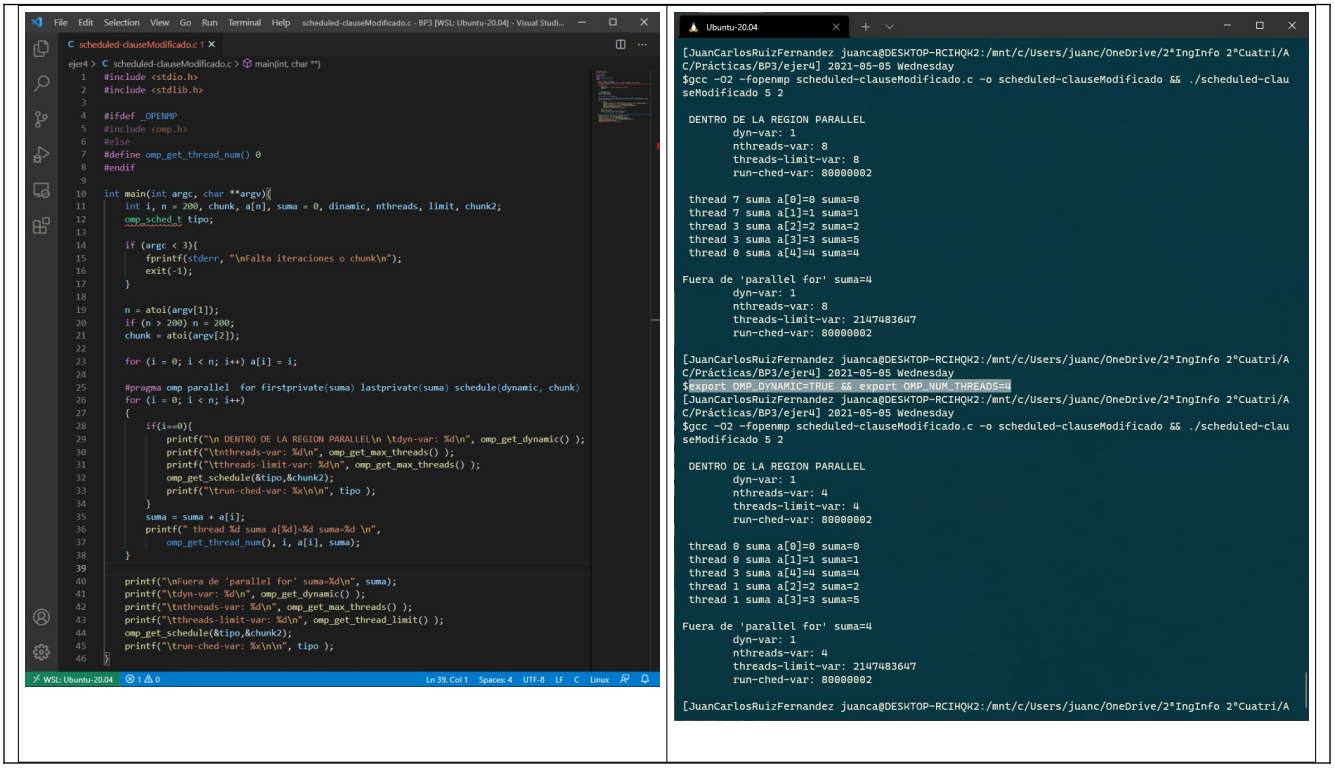
3. ¿Qué valor por defecto usa OpenMP para `chunk` y `modifier` con `static`, `dynamic` y `guided`? Explicar qué ha hecho para contestar a esta pregunta.

Al principio pensé que se seteaba por defecto a 1. Al comprobarlo me di cuenta que para dynamic y guided lo cumplian pero no static. Buscando en el manual de *OpenMP3.1CompleteSpecifications* de la pagina oficial (https://www.openmp.org/wp-content/uploads/OpenMP3.1.pdf pagina 52 del pdf, 44 del documento) encontré que efectivamente que el valor por defecto en dynamic y guided es 1 pero para static se reparte por igual entre el numero de iteraciones, mas o menos, ya que seguro que alguna de las hebras se llevará una iteración mas dependiendo de el numero de éstas y de hebras.

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`

CAPTURAS DE PANTALLA:



```
#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, dinamic, nthreads, limit, chunk2;
    omp_sched_t tipo;

    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("\n DENTRO DE LA REGION PARALLEL\n \tdyn-var: %d\n", omp_get_dynamic() );
            printf("\tnthreads-var: %d\n", omp_get_max_threads() );
            printf("\tthreads-limit-var: %d\n", omp_get_max_threads() );
            omp_get_schedule(&tipo,&chunk2);
            printf("\trun-ched-var: %x\n\n", tipo );
        }
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
            omp_get_thread_num(), i, a[i], suma);
    }

    printf("\nFuera de 'parallel for' suma=%d\n", suma);
    printf("\tdyn-var: %d\n", omp_get_dynamic() );
    printf("\tnthreads-var: %d\n", omp_get_max_threads() );
    printf("\tthreads-limit-var: %d\n", omp_get_thread_limit() );
    omp_get_schedule(&tipo,&chunk2);
    printf("\trun-ched-var: %x\n\n", tipo );
}
```

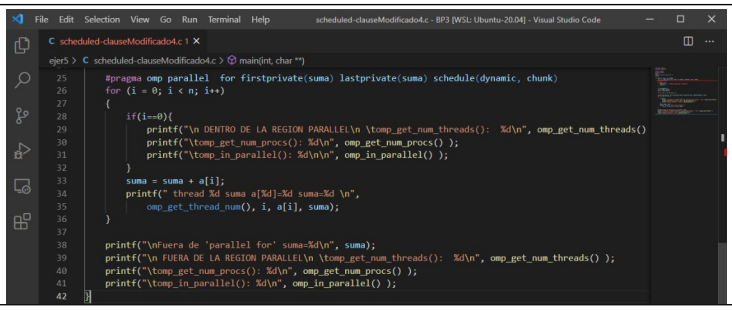
```
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP3/ejer4] 2021-05-05 Wednesday
$gcc -O2 -fopenmp scheduled-clauseModificado.c -o scheduled-clauseModificado && ./scheduled-clauseModificado 5 2

 DENTRO DE LA REGION PARALLEL
        dyn-var: 1
        nthreads-var: 8
        threads-limit-var: 8
        run-ched-var: 80000002

 thread 7 suma a[0]=0 suma=0
 thread 7 suma a[1]=1 suma=1
 thread 3 suma a[2]=2 suma=2
 thread 3 suma a[3]=3 suma=5
 thread 0 suma a[4]=4 suma=4

Fuera de 'parallel for' suma=4
        dyn-var: 1
        nthreads-var: 8
        threads-limit-var: 2147483647
        run-ched-var: 80000002

[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP3/ejer4] 2021-05-05 Wednesday
$export OMP_DYNAMIC=TRUE && export OMP_NUM_THREADS=4
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP3/ejer4] 2021-05-05 Wednesday
$gcc -O2 -fopenmp scheduled-clauseModificado.c -o scheduled-clauseModificado && ./scheduled-clauseModificado 5 2

 DENTRO DE LA REGION PARALLEL
        dyn-var: 1
        nthreads-var: 4
        threads-limit-var: 4
        run-ched-var: 80000002

 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 3 suma a[4]=4 suma=4
 thread 1 suma a[2]=2 suma=2
 thread 1 suma a[3]=3 suma=5

Fuera de 'parallel for' suma=4
        dyn-var: 1
        nthreads-var: 4
        threads-limit-var: 2147483647
        run-ched-var: 80000002

[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/A
```

RESPUESTA:

La unica diferencia entre fuera y dentro de la region parallel es *`thread-limit-var`*, la que fuera de la region queda indefinida aparentemente (ha cogido el maximo valor de int en 32bits). Haciendo export de dos variables entre ejecuciones si las hace cambiar, pero pasa lo mismo dentro y fuera de la region.

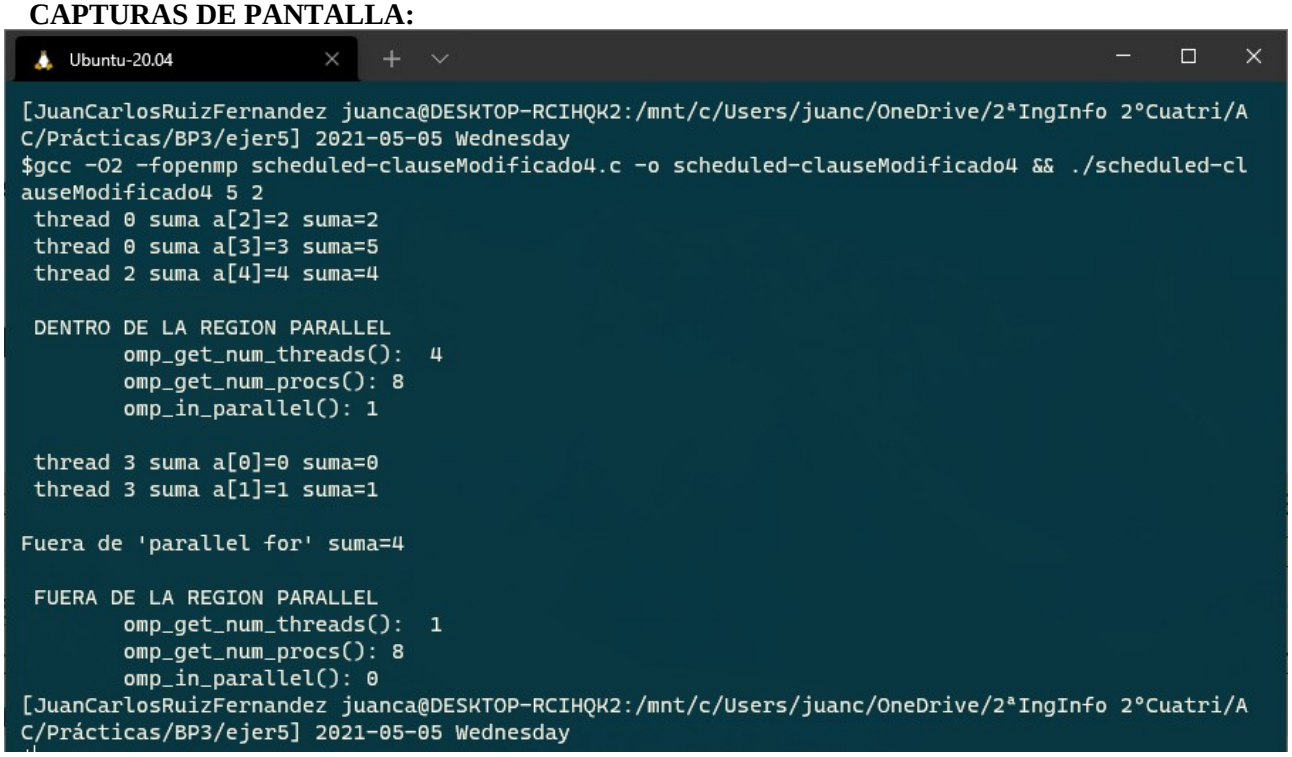
5. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno 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`



```
#pragma omp parallel  for firstprivate(suma) lastprivate(suma) schedule(dynamic, chunk)
for (i = 0; i < n; i++)
{
    if(i==0){
        printf("\n DENTRO DE LA REGION PARALLEL\n \tomp_get_num_threads():  %d\n", omp_get_num_threads()
        printf("\tomp_get_num_procs(): %d\n", omp_get_num_procs() );
        printf("\tomp_in_parallel(): %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("\nFuera de 'parallel for' suma=%d\n", suma);
printf("\n FUERA DE LA REGION PARALLEL\n \tomp_get_num_threads():  %d\n", omp_get_num_threads() );
printf("\tomp_get_num_procs(): %d\n", omp_get_num_procs() );
printf("\tomp_in_parallel(): %d\n", omp_in_parallel() );
}
```

CAPTURAS DE PANTALLA:

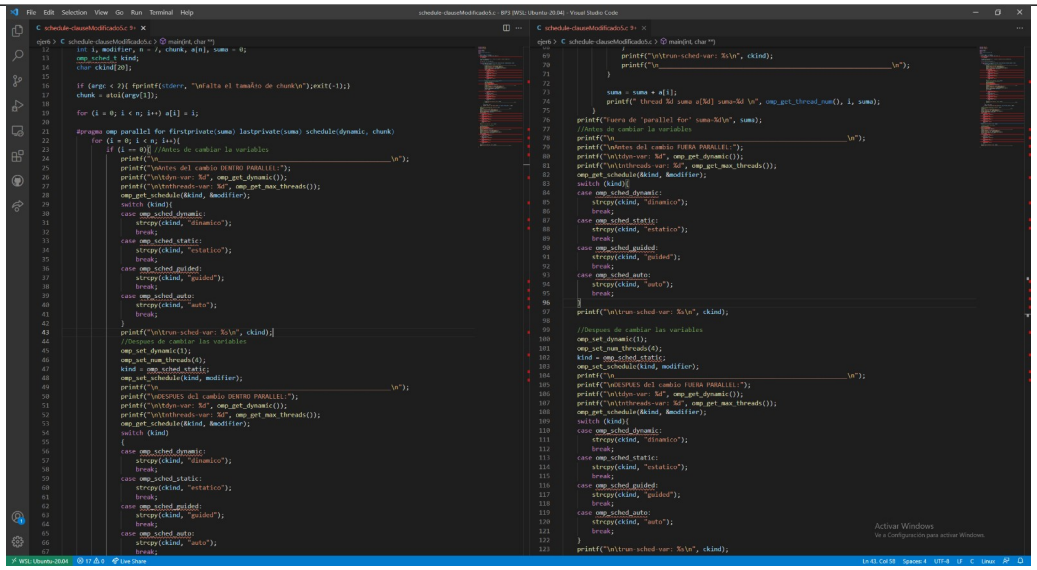


RESPUESTA:

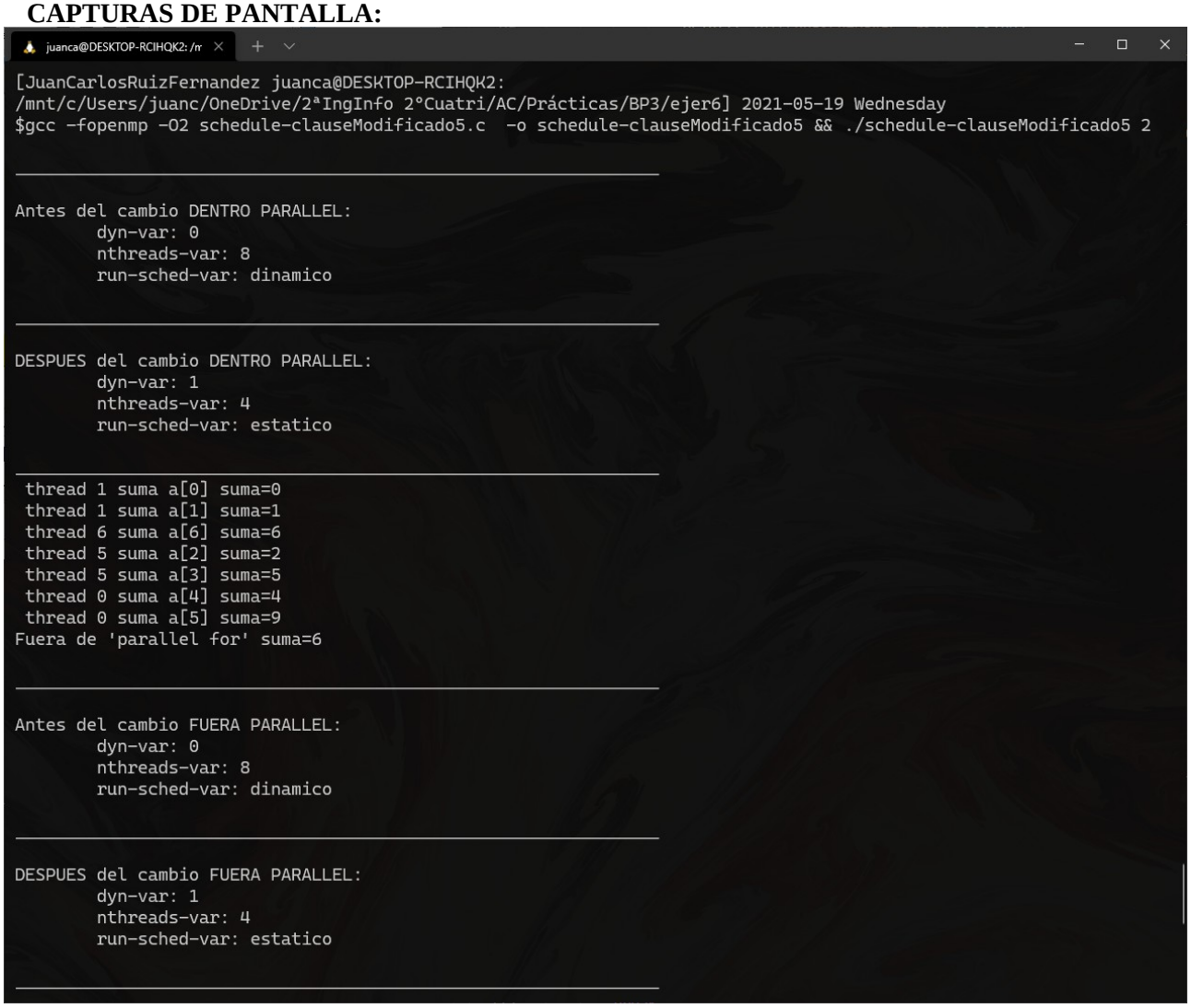
Se obtienen diferencias para *num_threads* y para *in_parallel.* Dentro del parallel tiene el valor que yo exporte en la captura del ejercicio anterior (4) y es la cantidad de threads ejecutandose; pues fuera de la región se convierte en 1. Para *in_parallel* como no tenemos condicionales en la directiva parallel sabemos que siempre se va a ejecutar en modo paralelo, por eso dentro de la región muestra un 1 y fuera un 0, esta ultima demostrando que no hay región paralela activa.

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`



CAPTURAS DE PANTALLA:



RESPUESTA:

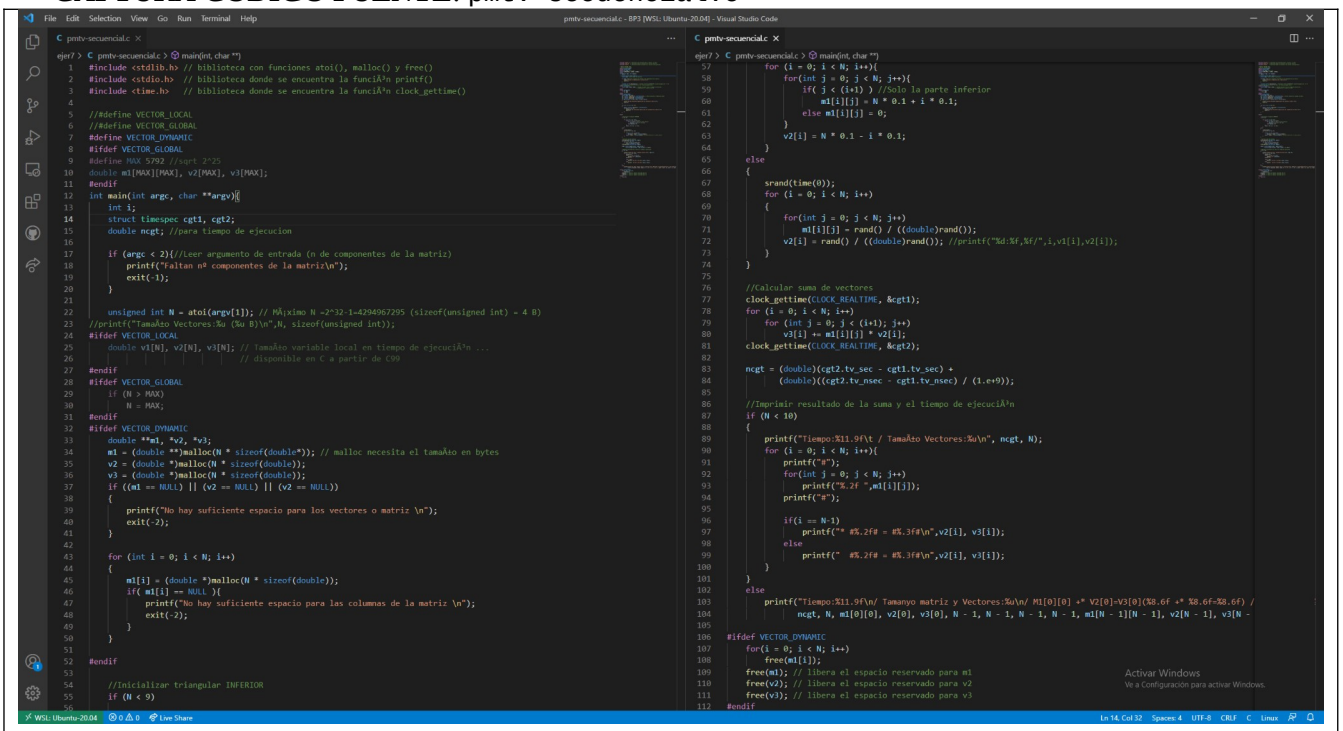
Los cambios se realizan corectamente pero después de salir de la region *parallel* recuperan los valores anteriores al cambio.

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

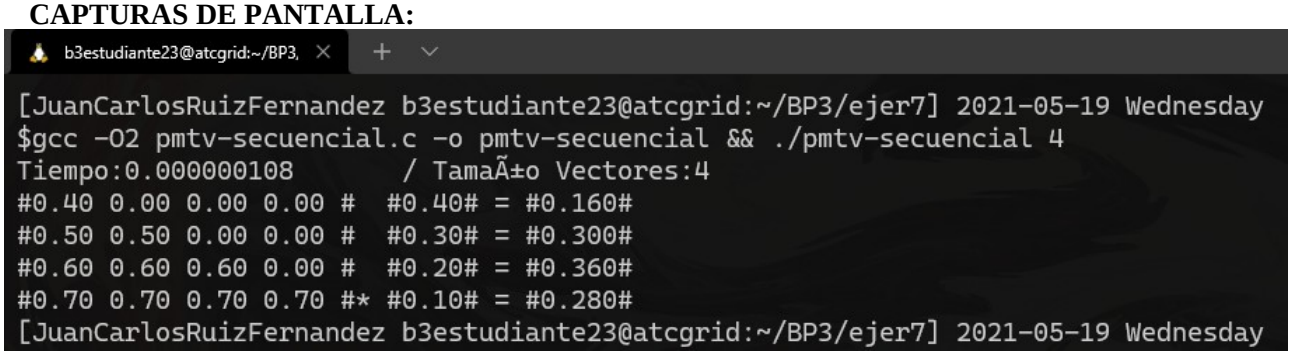
7. Implementar un programa secuencial en C que multiplique una matriz triangular inferior por un vector (use variables dinámicas y tipo de datos double). Comparar el orden de complejidad y el número total de operaciones (sumas y productos) de este código respecto al que implementó para el producto matriz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



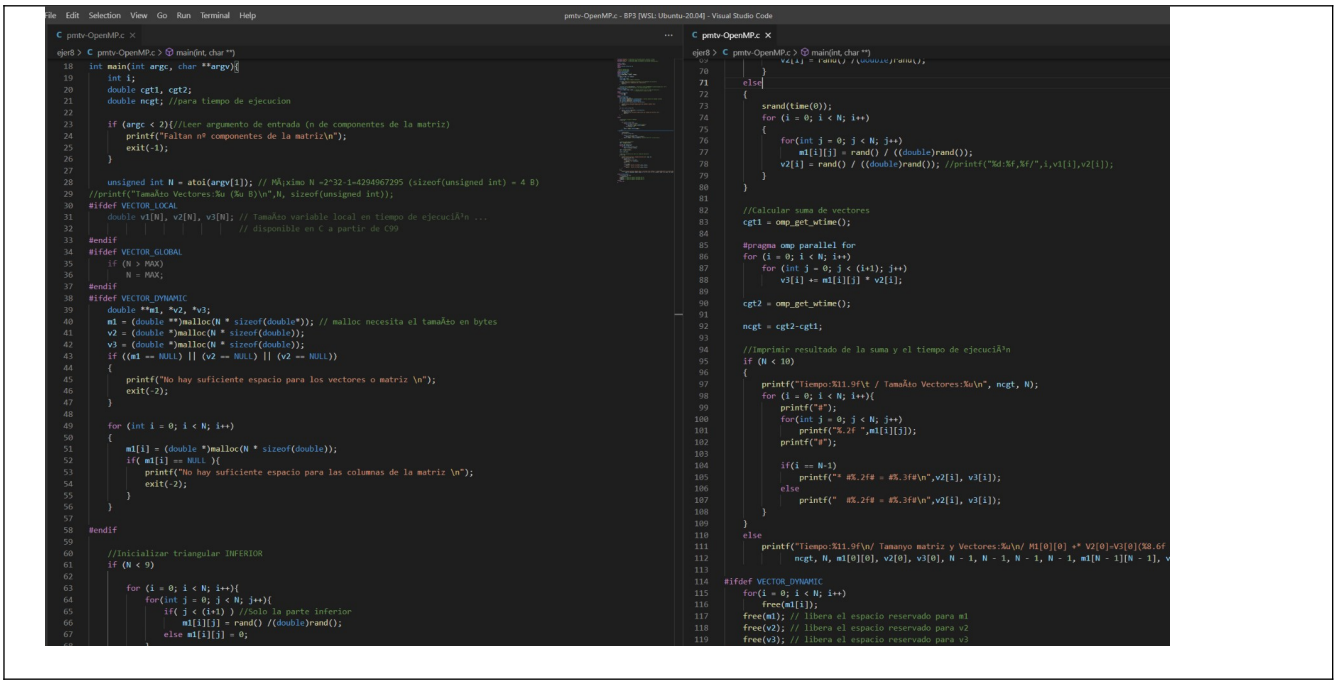
CAPTURAS DE PANTALLA:



Para el ejemplo de la matriz completa seguía un orden $O(n^2)$, N*N sumas y multiplicaciones. Claramente en este caso tenemos que numero de cálculos es menor siguiendo un orden aproximado de O(n*logn). Haciendo el calculo para el primer ejemplo con matriz de 4x4 tendríamos 32 cálculos, con la matriz triangular obtenemos 20. Para 8x8, 128 para la completa y 72 para la triangular. Podemos deducir que siempre tendremos un numero menor de cálculos para este caso.

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`



DESCOMPOSICIÓN DE DOMINIO:

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

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

CAPTURAS DE PANTALLA:

```
juanca@DESKTOP-RCIHQK2: /m   b3estudiante23@atcgrid:~/BP3,
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP3/ejer8] 2021-05-24 Monday
$gcc -O2 -fopenmp pmtv-OpenMP.c -o pmtv-OpenMP && sbatch --wrap "./pmtv-OpenMP 6"
Submitted batch job 107746
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP3/ejer8] 2021-05-24 Monday
$cat slurm-107746.out
Tiempo:0.000080731       / TamaÃ±o Vectores:6
#2.13 0.00 0.00 0.00 0.00 0.00 #  #0.98# = #2.089#
#4.61 0.44 0.00 0.00 0.00 0.00 #  #0.50# = #2.533#
#0.76 0.71 1.04 0.00 0.00 0.00 #  #0.89# = #2.224#
#0.23 0.07 0.17 0.39 0.00 0.00 #  #1.19# = #1.028#
#4.58 0.61 2.07 1.22 0.51 0.00 #  #0.38# = #3.406#
#0.68 0.45 0.80 0.88 0.07 6.11 #* #0.29# = #2.609#
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP3/ejer8] 2021-05-24 Monday
$
```

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: Cada thread ejecutará una cantidad diferente ya que es una matriz triangular inferior. Por lo que la primera hebra ejecutará 1, la siguiente ejecutará 2 y asi sucesivamente hasta la ultima hebra que ejecutará la cantidad igual al numero de filas o columnas.

(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: Dependerá de la velocidad de como se ejecute cada thread, pero será mas equitativo ya que por ejemplo la primera no solo ejecutará una iteración, si puede ejecutará mas.

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

RESPUESTA: Según la teoría, guided dara mejores respuestas ya que provoca menos sobrecarga que dynamic. Ademas como empieza con mayores asignaciones al principio y las va reduciendo, se equilibra el reparto a los threads al tener las primeras iteraciones menos cálculos y las ultimas más. Como se aprecia en las ultimas columnas de la tabla del ejercicio 2.

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`

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	