|  |  |  |
| --- | --- | --- |
| **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): Brian Sena Simons  Grupo de prácticas: 2ºB – B2  Fecha de entrega: ¿¿?¿?  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

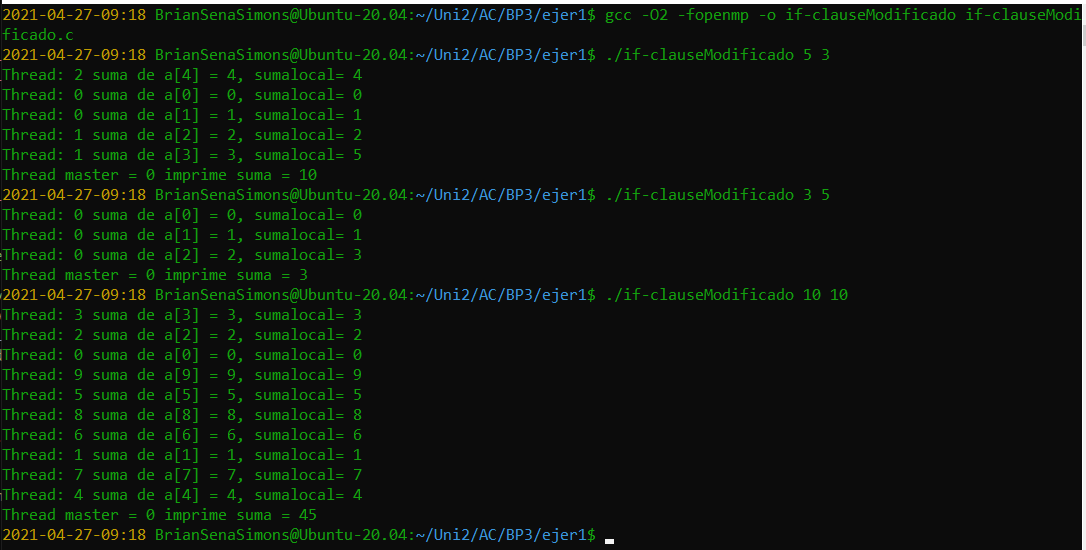
# Ejercicios basados en los ejemplos del seminario práctico

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

|  |
| --- |
|  |

**CAPTURAS DE PANTALLA:**

****

**RESPUESTA**:

Básicamente con la cláusula “if” puedo establecer un umbral para el cual no se ejecutará el código en paralelo si no que más bien utilizará apenas una hebra, es decir, de forma secuencial. Sin embargo, si n>4 entonces podré establecer el número de hebras que quiero utilizar para repartir el trabajo. Dando así mayor flexibilidad a los posibles problemas a resolver.

**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).

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

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**:

La cláusula “static” divide uniformemente los “chunks”, normalmente de tamaño 1, a todas las hebras. Mientas que la cláusula “Dynamic”, que normalmente es para cuando se desconoce el tiempo de ejecución de las iteraciones, permite que las hebras más rápidas ejecuten más iteraciones del bucle. Esta última es igual que “Guided” solo que ahora el tamaño del bloque empieza siendo largo y luego va disminuyendo de tamaño, no más pequeño que chunk.

Según: **“Advanced OpenMP**”: La diferencia entre “monotoic” y “nonmonotonic” es que “nonmonotonic” le da más flexibilidad a la distribución de las iteraciones. Permitiendo que una hebra, por ejemplo, no siempre tenga que ejecutar secuencialmente “1,3,6” sino que pueda ir hacia atrás “3,6,1”;

• **simd**: the chunk\_size must be a multiple of the simd width. 1 • **monotonic**: If a thread executed iteration i, then the thread must execute iterations larger than i subsequently. 1 • **non**-**monotonic**: Execution order not subject to the monotonic restriction

**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.

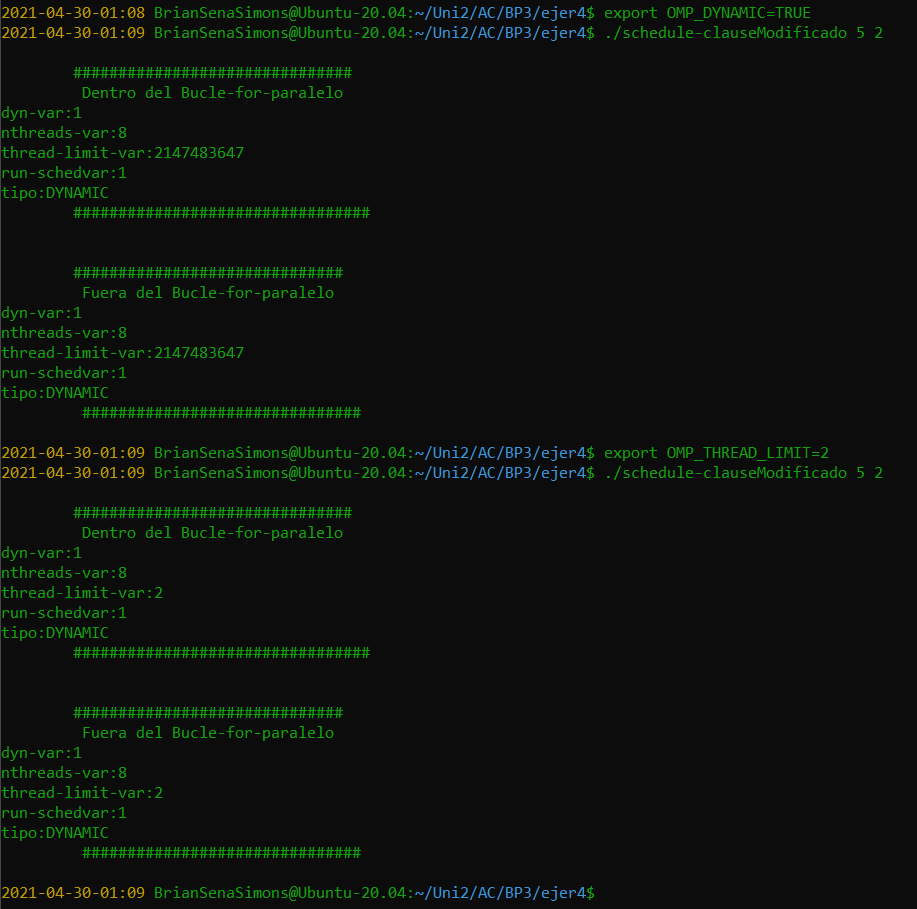
Tras realizar un par de ejecuciones en mi terminal he determinado que el chunk por defecto de las reglas chunk/modifier es de un tamaño adaptativo al número de hebras y tamaño del bucle (trabajo/nºhebra), y solo cambiaría las hebras que participan y/o trabajan debido al modifier (diferencias entre Static, Dynamic and Guided).

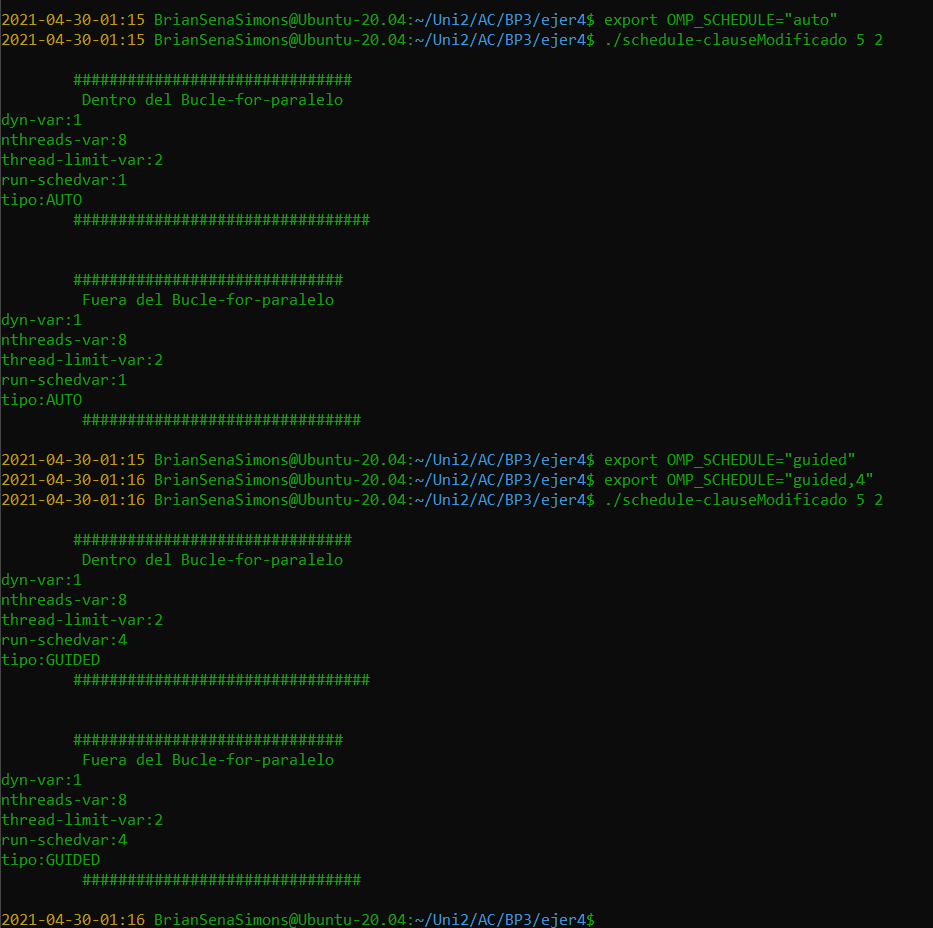
**4.** Añadir al programa scheduled-clause.c lo necesario para que imprima el valor de las variables de control dyn-var, nthreads-var, thread-limit-var y run-sched-var dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar 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:**

****

****

**RESPUESTA**:

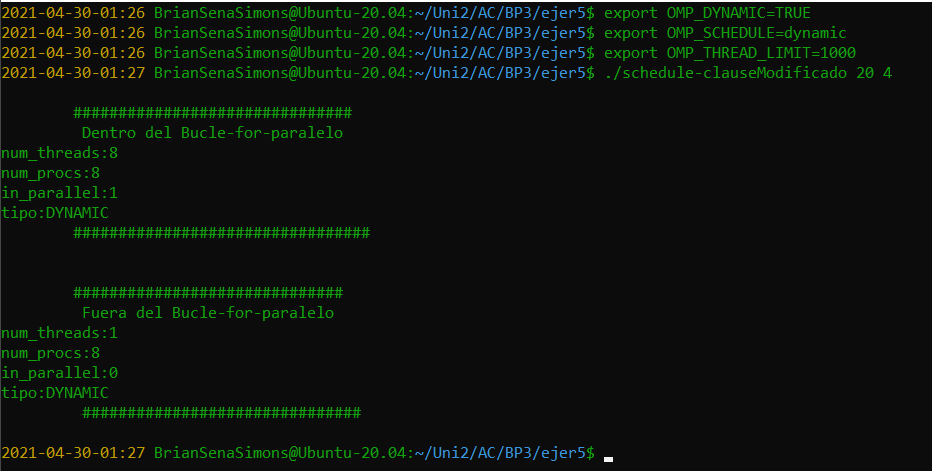
Siempre se imprime los mismos valores, ya sea dentro o fuera del bucle-for-paralelo.

**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

|  |
| --- |
|  |

**CAPTURAS DE PANTALLA:**

****

**RESPUESTA**:

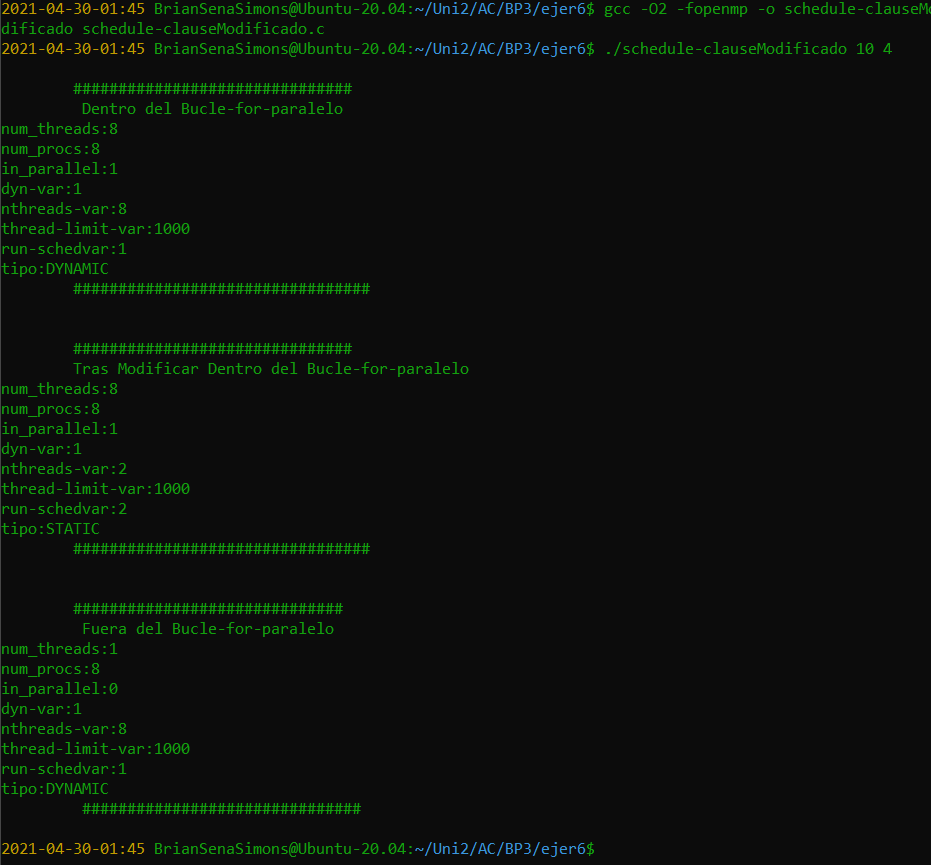
Como podemos ver con “**in\_parallel”** podemos verificar si estamos o no dentro de una zona de trabajo compartido. (1->Sí, 0->No). Y también otra diferencia es el número de “**threads**” en ejecución, como es de esperar, en la zona compartida contra la zona secuencial, 8 vs. 1 “**num\_threads**”. Sin embargo, sigue siendo el mismo **“num\_procs”.**

**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**:

# Como podemos observar si se han modificado correctamente los valores dentro de la ejecución de la zona de trabajo compartido, y al salir, los valores volvieron a ser los mismos de antes. **(mirar nthreads-var, run-schedvar y tipo para percibir las diferencias antes/después de modificar).**

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

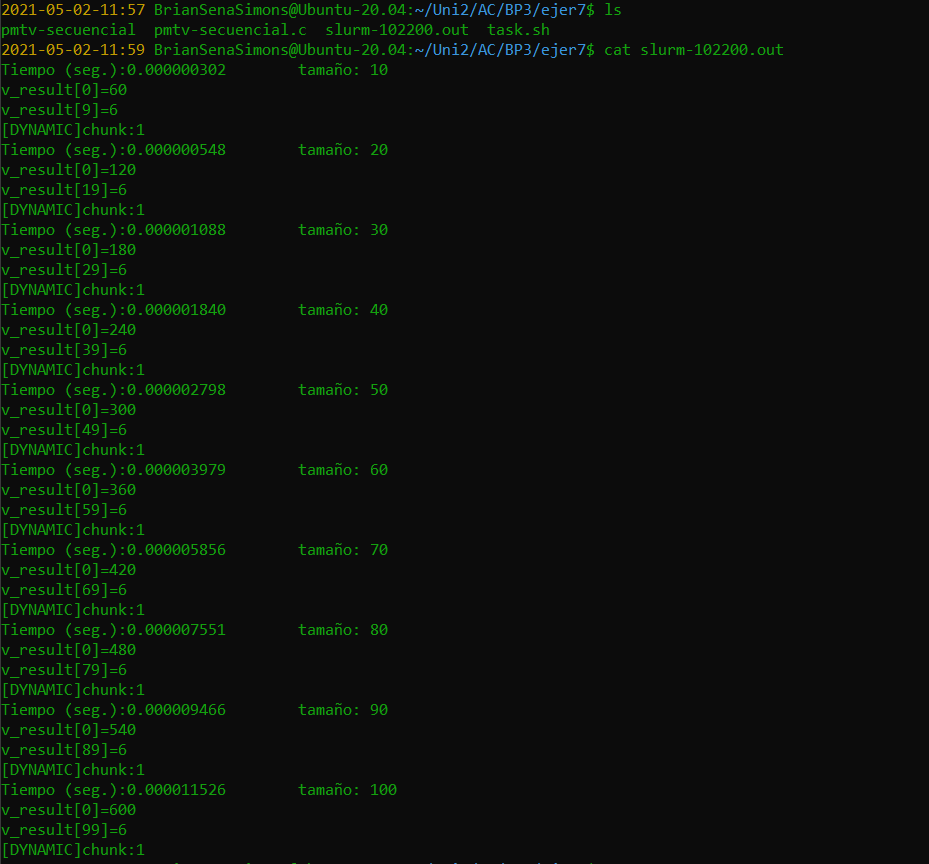
**7.** Implementar un programa secuencial en C que multiplique una matriz triangular inferior por un vector (use variables dinámicas y tipo de datos double). Comparar el orden de complejidad y el número total de operaciones (sumas y productos) de este código respecto al que implementó para el producto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

|  |
| --- |
|  |

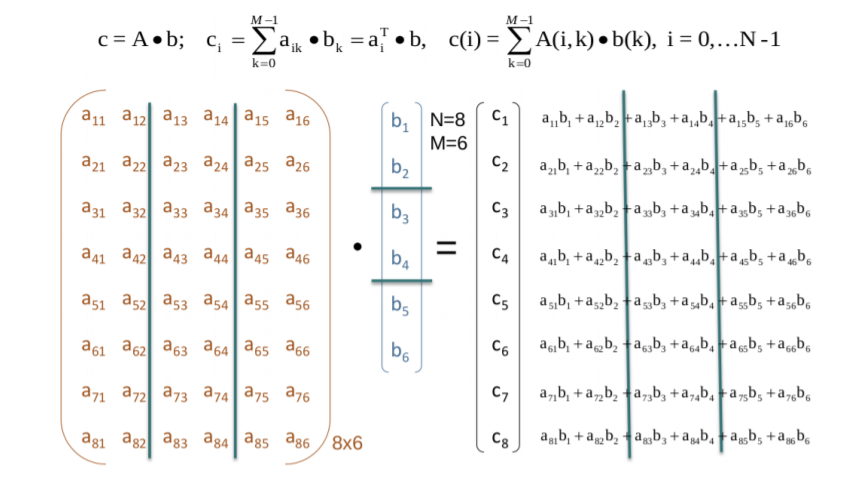
**CAPTURAS DE PANTALLA:**

****

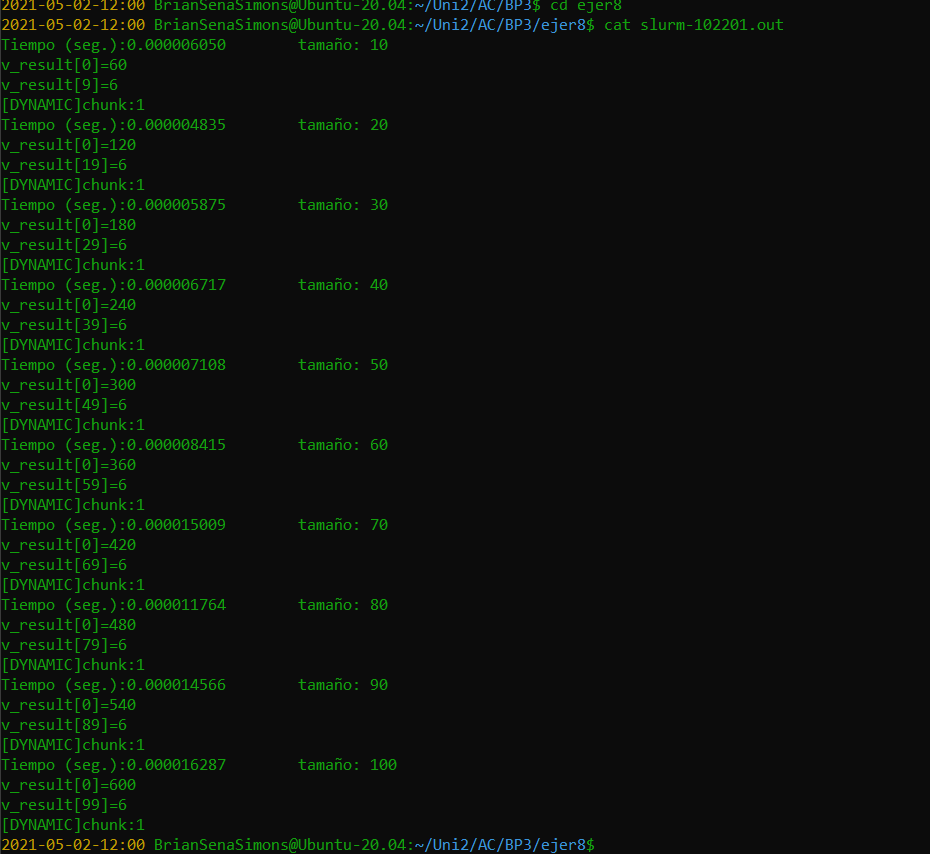
**8.** Implementar en paralelo la multiplicación de una matriz triangular inferior por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva for de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. La inicialización 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:**

**CAPTURAS DE PANTALLA:**



**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**:

Observando la estructura del cálculo, vemos que en cada iteración se realiza una cantidad total de N multiplicaciones y N sumas (Inner-loop); Por lo que podemos suponer que con una versión static monotonic, cada hebra realizará un total de N/nºhebras ejecuciones del loop-exterior.

Dando un total estimado de (N/nºhebras)\*(2N) operaciones de suma y multiplicaciones.

(Sin contar aquella hebra que llegue a ejecutar la última suma en el #pragma omp single).

**(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?

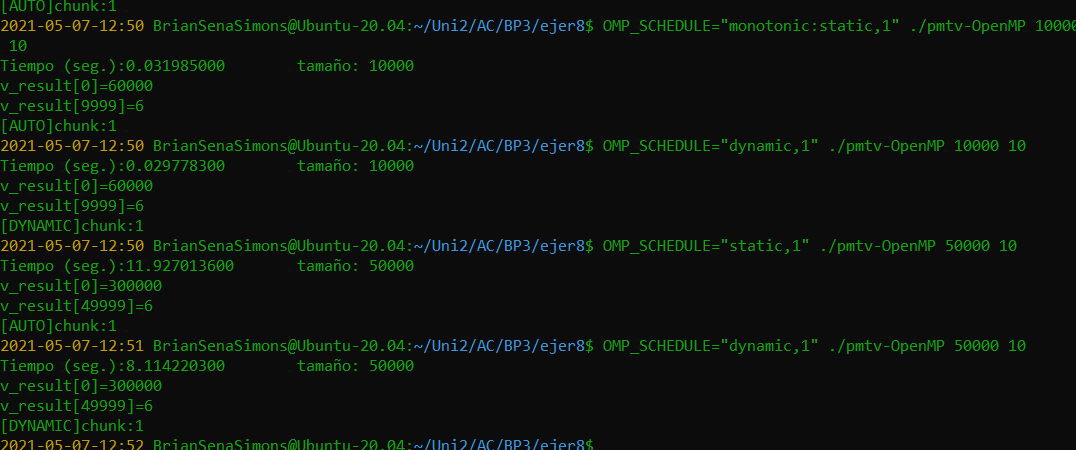
**RESPUESTA**:

No podemos suponer la carga laboral para cada hebra ya que depende de la rapidez de ejecución de cada una de ellas.

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

**RESPUESTA**:

He hecho varios tests (incluyo alguno a continuación) y al parecer la versión dinámica le gana a la versión estática por algunos segundos (cada vez más notable para un n < no). En mi opinión debe de ser porque la versión dinámica permite obviar aquellas hebras que estén tardando en finalizar su tarea por cualquier razón y así favorecer el rendimiento total de la ejecución. (Repartición no equitativa => sobrecarga en algunas hebras es mayor => pero tiempo de ejecución es menor).

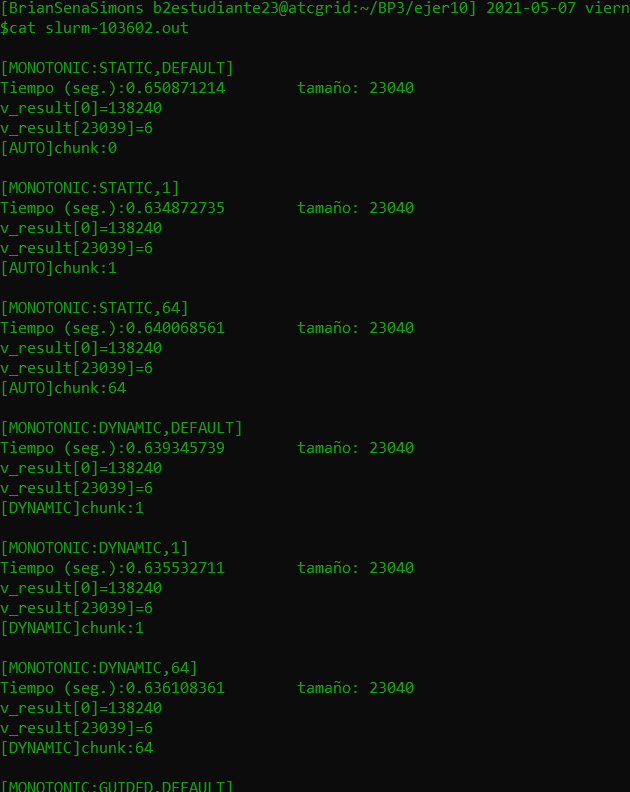


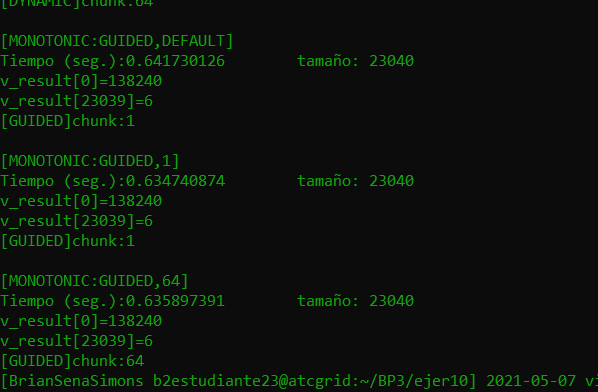
**10.** Obtener en atcgrid los tiempos de ejecución del código paralelo (usando, como siempre, -O2 al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación static, dynamic y guided para chunk de 1, 64 y el chunk por defecto para la alternativa (con monotonic 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.

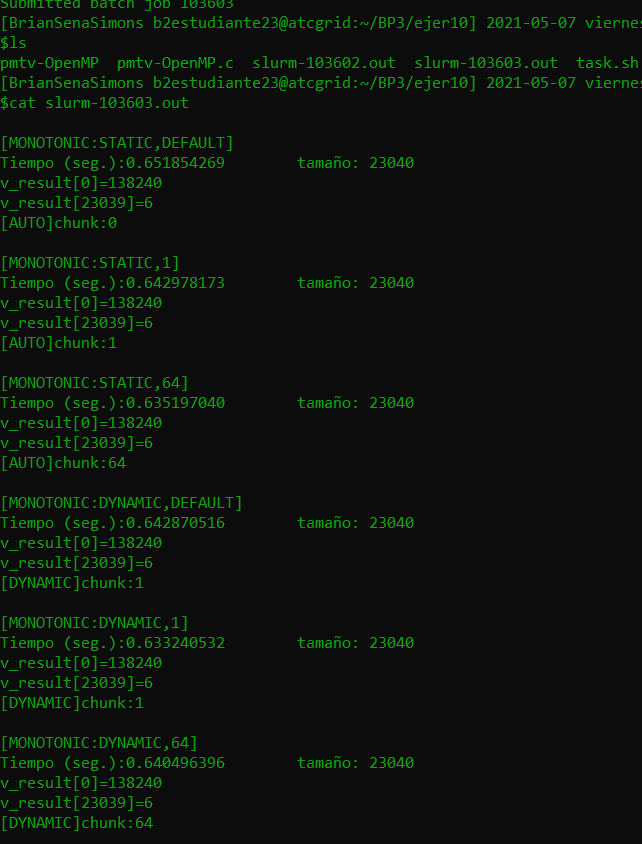
**CAPTURA CÓDIGO FUENTE**: pmtv-OpenMP.c

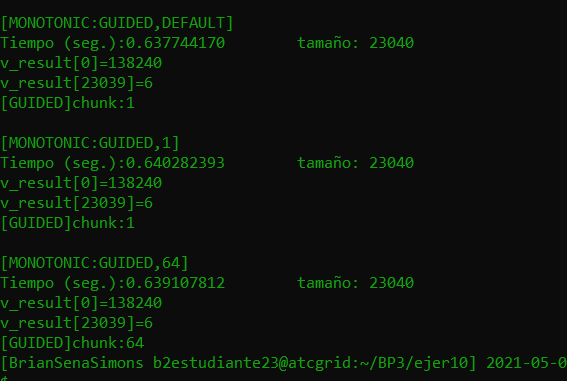
|  |
| --- |
| He Utilizado el mismo código de paralelización por filas que en el **ejericicio8** |

**CAPTURAS DE PANTALLA:**

****

****

****

****

**TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid**

**SCRIPT:** pmvt-OpenMP\_atcgrid.sh

|  |
| --- |
|  |

|  |
| --- |
| 1. 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** | 0.650871214 | 0.639345739 | 0.641730126 | | **1** | 0.634872735 | 0.635532711 | 0.634740874 | | **64** | 0.640068561 | 0.636108361 | 0.635897391 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **Chunk** | **Static** | **Dynamic** | **Guided** | | **por defecto** | 0.651854269 | 0.642870516 | 0.637744170 | | **1** | 0.642978173 | 0.633240532 | 0.640282393 | | **64** | 0.635197040 | 0.640496396 | 0.639107812 | |

