|  |  |  |
| --- | --- | --- |
| **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

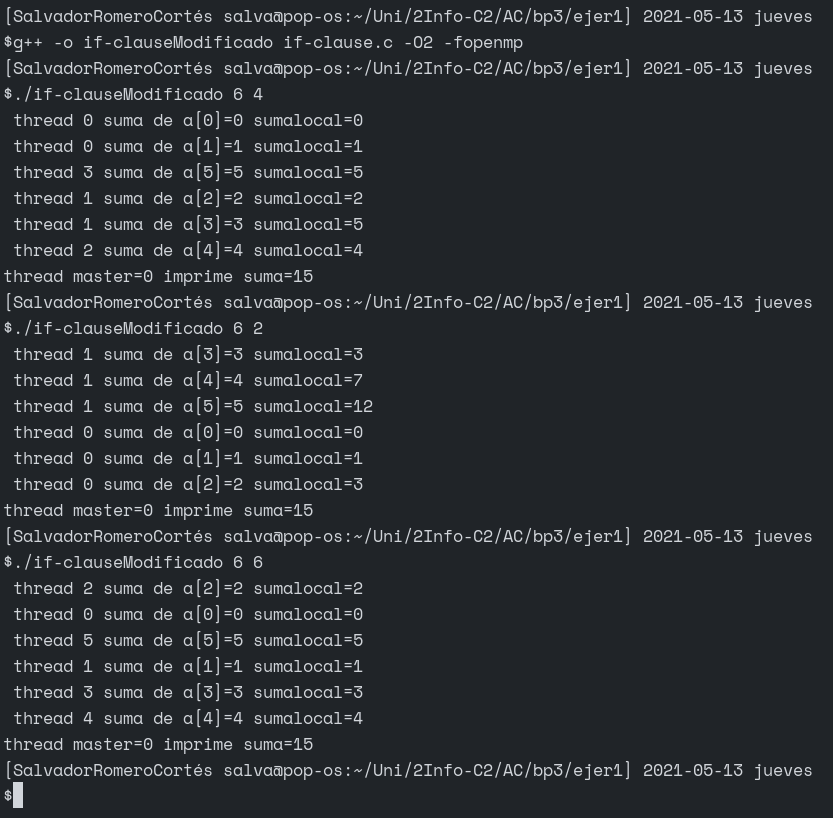
# Ejercicios basados en los ejemplos del seminario práctico

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

|  |
| --- |
|  |

**CAPTURAS DE PANTALLA:**

****

**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).

|  |
| --- |
| 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>

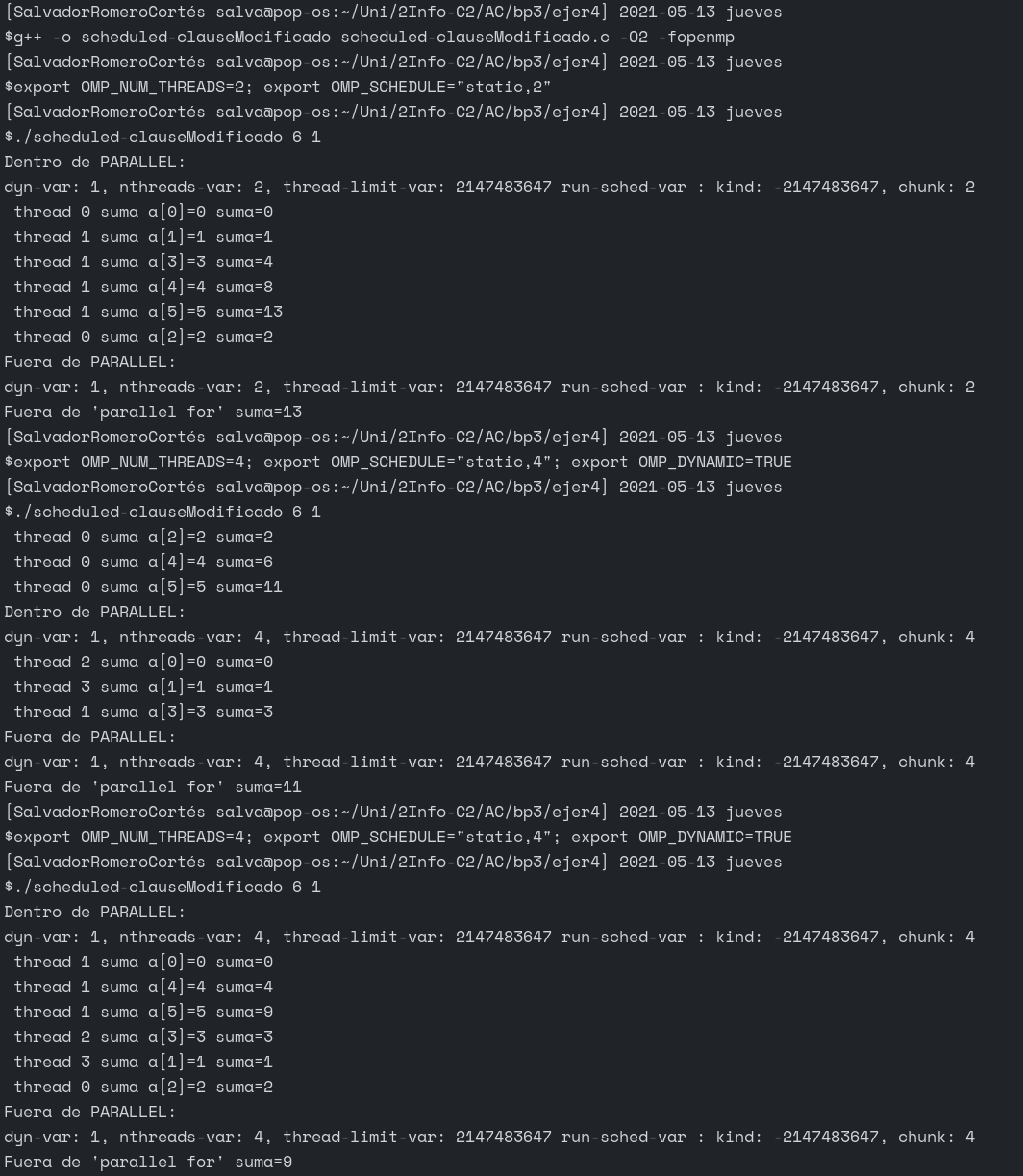
<https://software.intel.com/content/www/us/en/develop/articles/openmp-loop-scheduling.html>

**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

|  |
| --- |
|  |

**CAPTURAS DE PANTALLA:**

****

**RESPUESTA**:

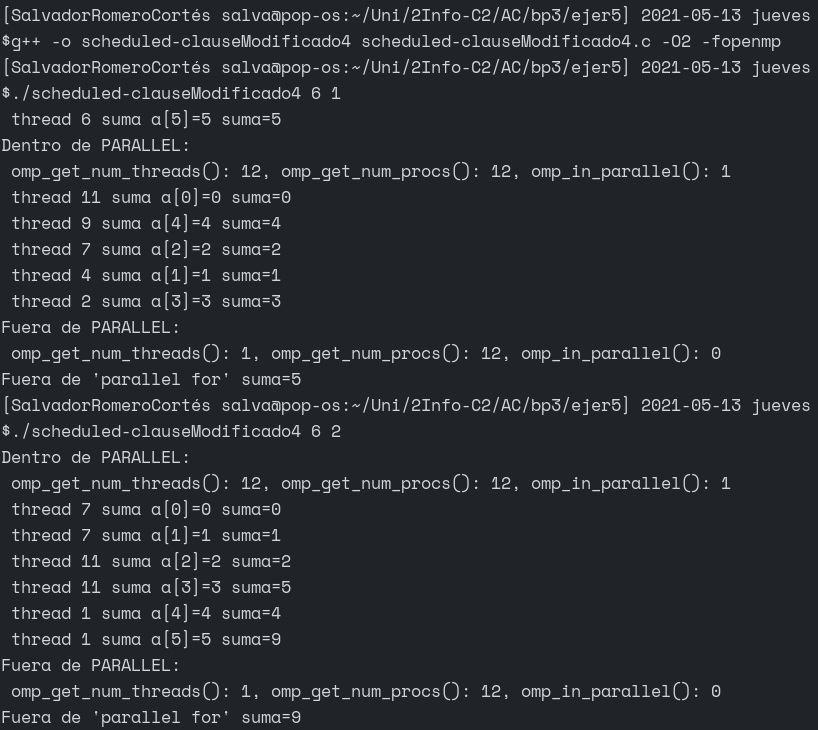
Muestran el mismo resultado tanto dentro como fuera de la región paralela.

**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

|  |
| --- |
|  |

**CAPTURAS DE PANTALLA:**

****

**RESPUESTA**:

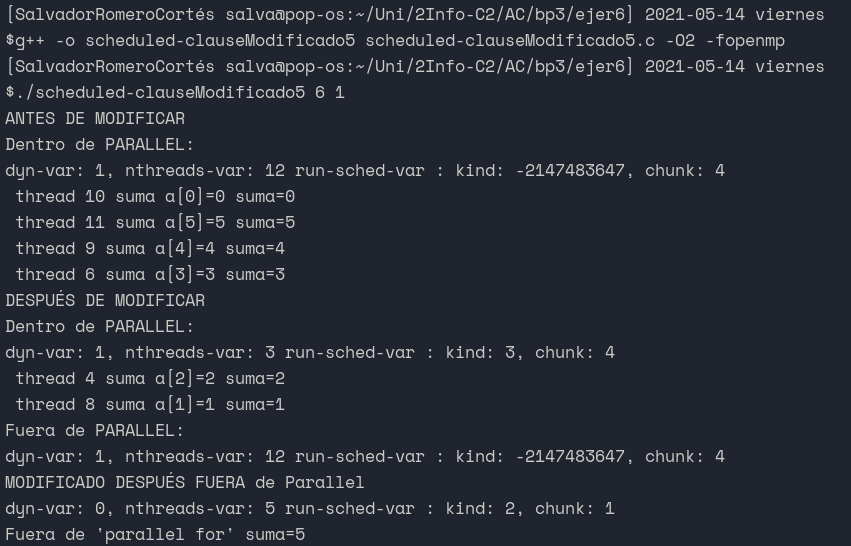
Se obtienen valores distintos en el número de hebras y en “omp\_in\_parallel()”. Esto ocurre porque en la zona paralela se ejecutan el número de hebras indicado por OMP\_NUM\_THREADS mientras que fuera de la región paralela solo se ejecuta por una única hebra. La segunda variable indica si se encuentra 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

|  |
| --- |
|  |

**CAPTURAS DE PANTALLA:**

****

**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)

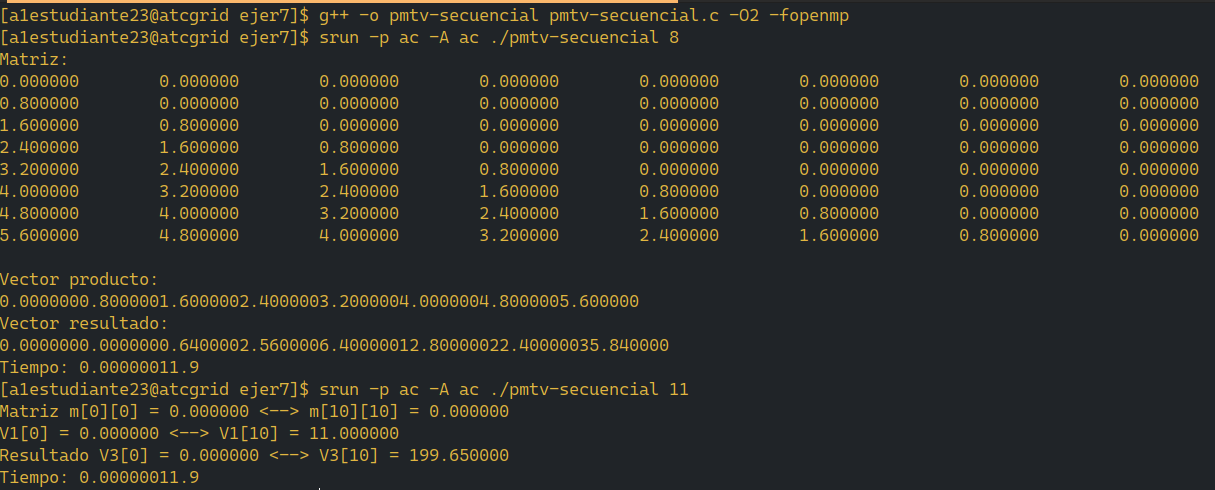
**7.** Implementar un programa secuencial en C que multiplique una matriz triangular inferior por un vector (use variables dinámicas y tipo de datos double). Comparar el orden de complejidad y el número total de operaciones (sumas y productos) de este código respecto al que implementó para el producto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

|  |
| --- |
|  |

**CAPTURAS DE PANTALLA:**

**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.

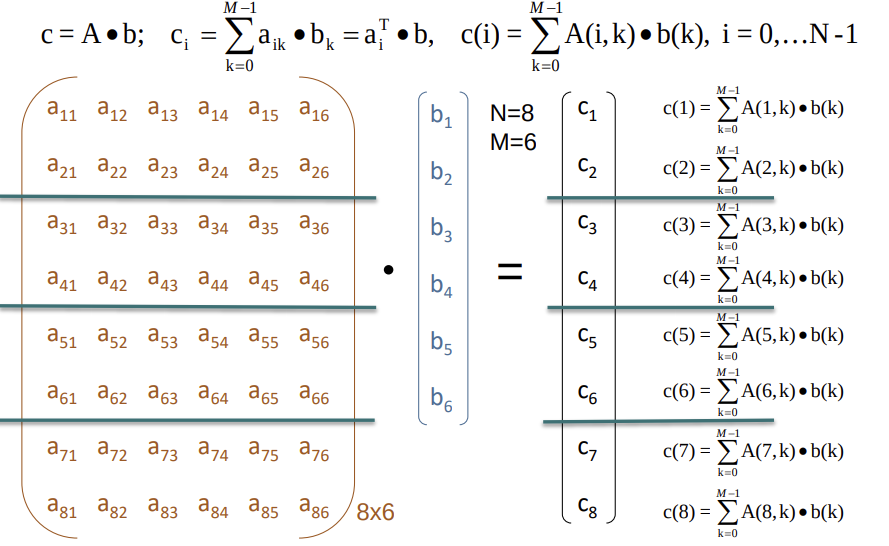
**8.** Implementar en paralelo la multiplicación de una matriz triangular inferior por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva for de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. La inicialización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

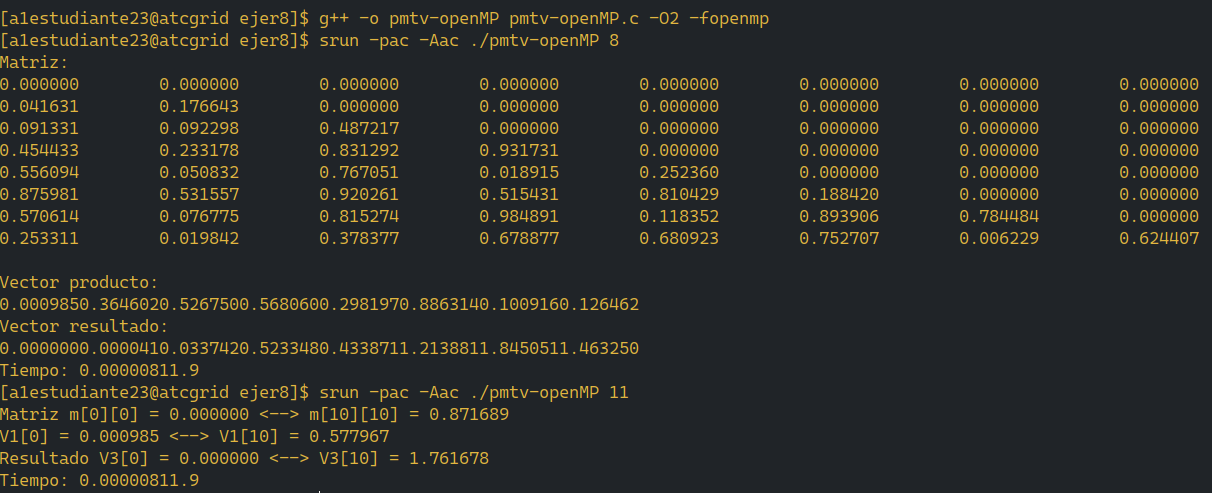
|  |
| --- |
|  |

**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.

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:**



**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.

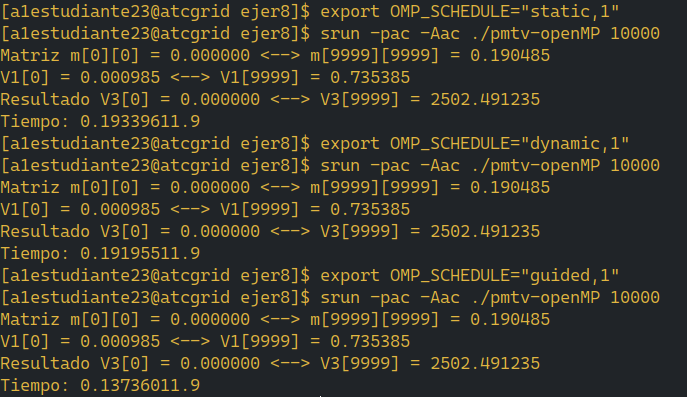
**(b)** Con la asignación dynamic y guided, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

**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:

**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

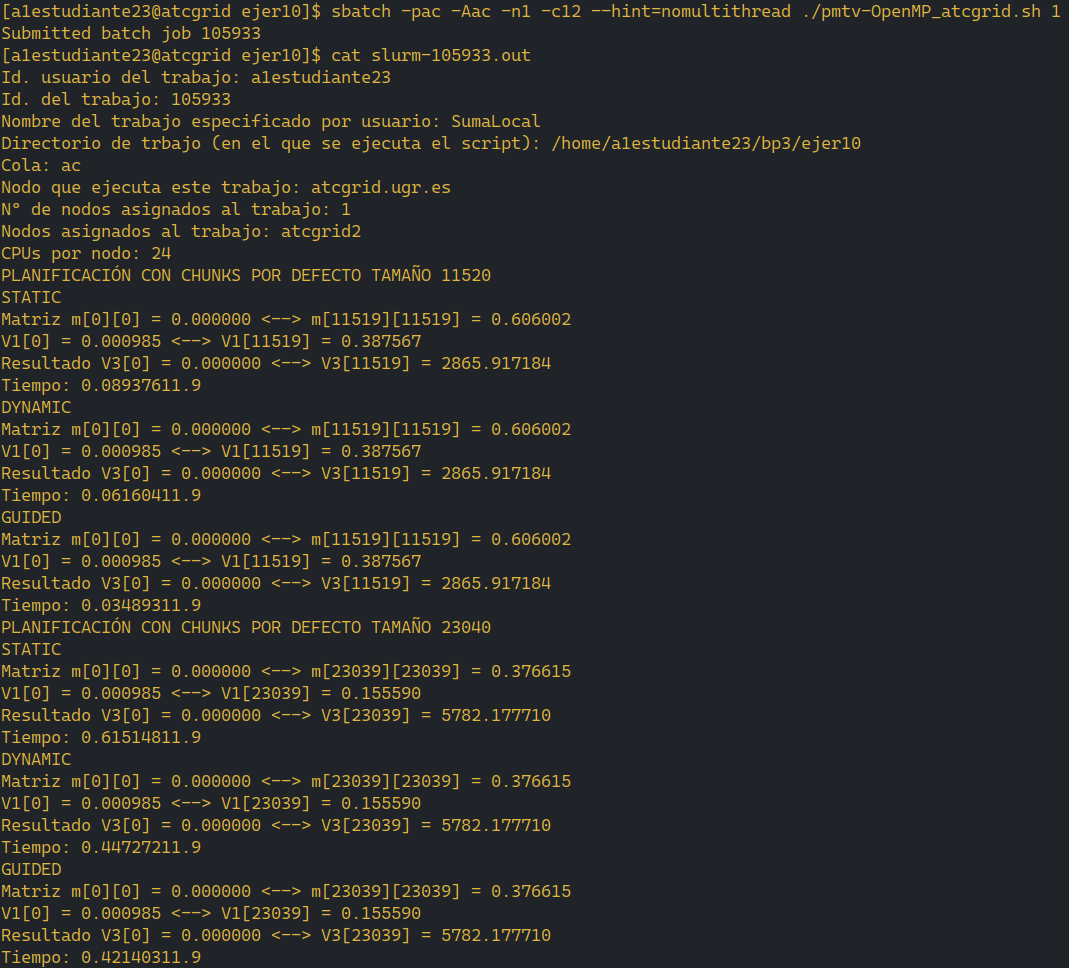
|  |
| --- |
| Usamos el mismo código pero utilizando un script que lance el programa con las distintas opciones de planificación. |

**DESCOMPOSICIÓN DE DOMINIO:**

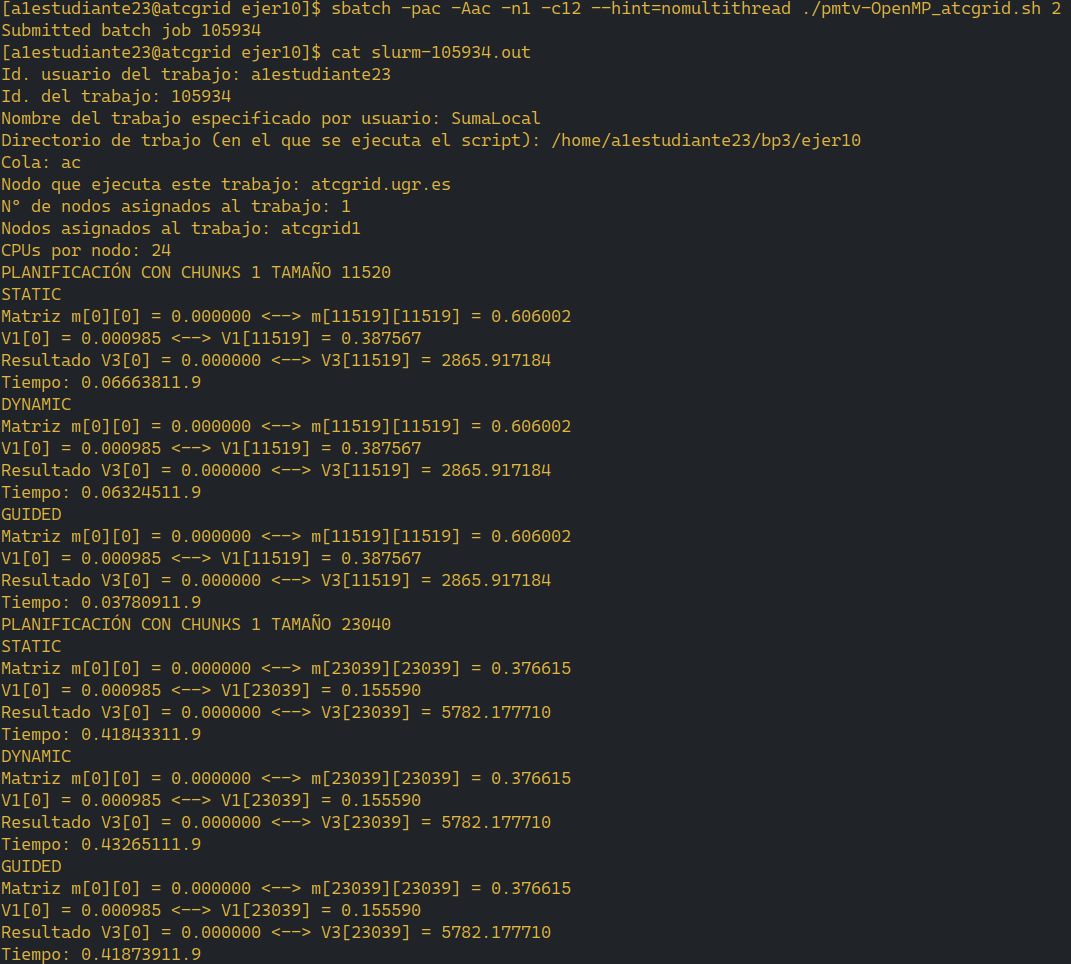
Es la misma que la del ejercicio 8 puesto que no hemos cambiado nada. Sólo cambia en el caso de guided, que en cada iteración se va repartiendo mejor las operaciones entre las distintas hebras.

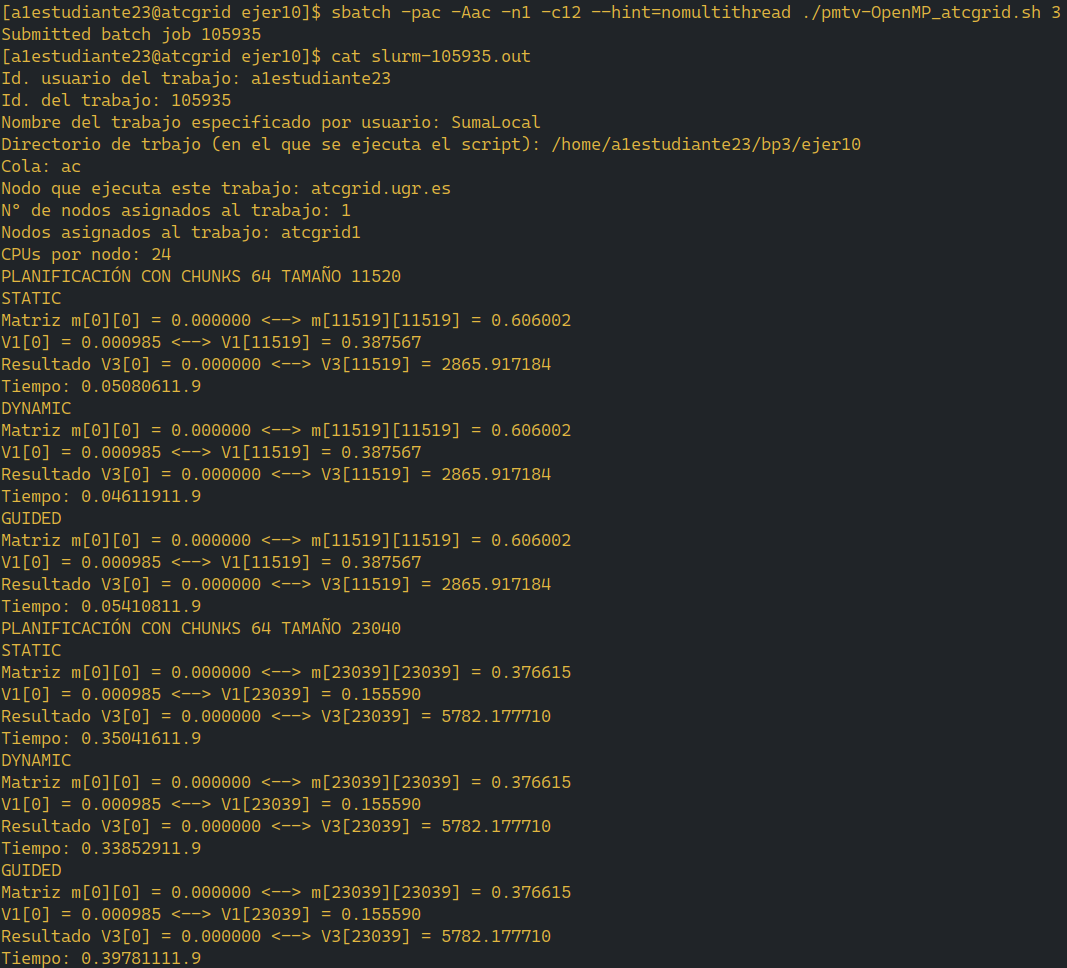
**CAPTURAS DE PANTALLA:**

Con tamaño de chunk por defecto:



Con tamaño de chunk 1:

Con tamaño de chunk 64:



**TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid**

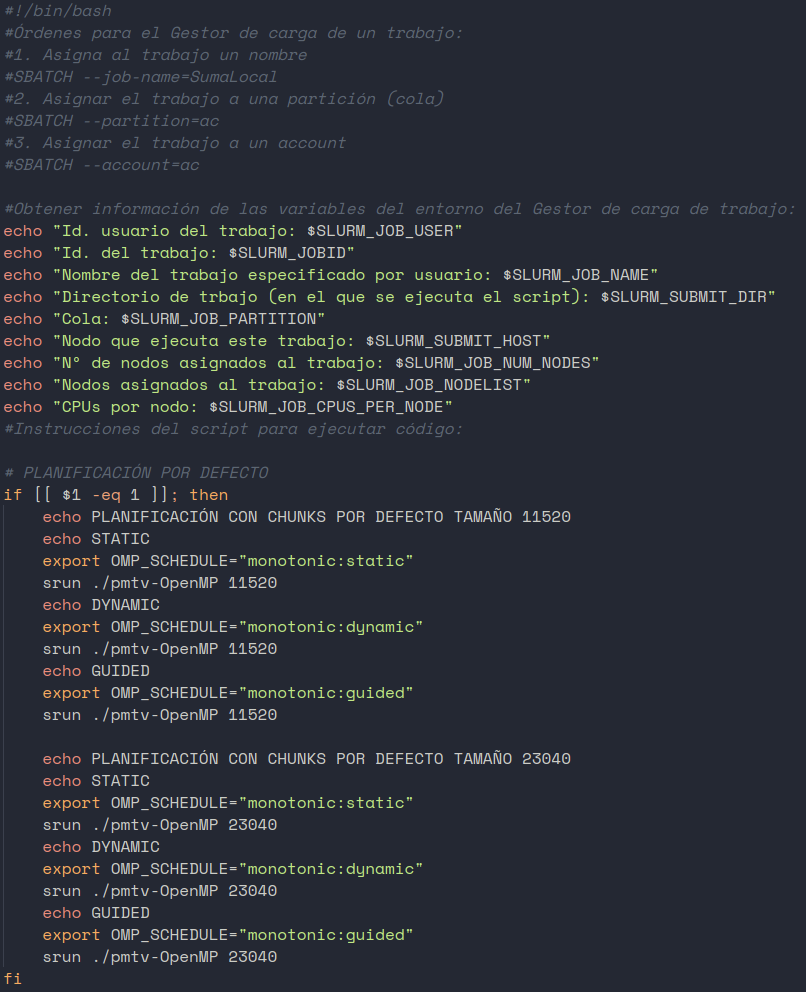
Me di cuenta de una errata en el código que hacía que al final de cada tiempo agregaba un “11.9” simplemente para rellenar los datos de la tabla he cogido los datos obtenidos en esta ejecución pero eliminando ese 11.9 del final.

|  |
| --- |
| 1. 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 |  |  | | --- | | 1. 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 "Nº 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:**

****

