|  |  |  |
| --- | --- | --- |
| **2º curso / 2º cuatr.**  **Grado Ing. Inform.**  **Doble Grado Ing. Inform. y Mat.** |  | **Arquitectura de Computadores (AC)**  **Cuaderno de prácticas.**  **Bloque Práctico 1. Programación paralela I: Directivas OpenMP**  Estudiante (nombre y apellidos): Alberto Llamas González  Grupo de prácticas y profesor de prácticas: D3, Juan Carlos Gómez López  Fecha de entrega: 18 abril 2021  Fecha evaluación en clase: 19 abril 2021 |

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 directiva parallel combinada con directivas de trabajo compartido en los ejemplos bucle-for.c y sections.c del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

**RESPUESTA**: Captura que muestre el código fuente bucle-forModificado.c

|  |
| --- |
| ../../../../../../Downloads/caps/Captura%20de%20pantalla%20de%202021-03-22%2019-14-15.png |

**RESPUESTA**: Captura que muestre el código fuente sectionsModificado.c

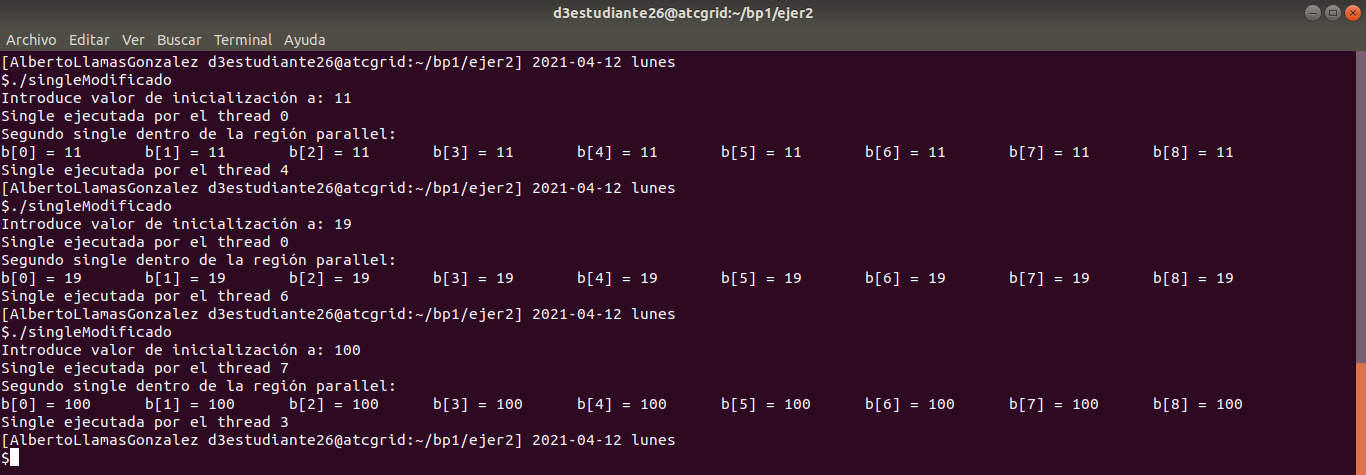
|  |
| --- |
| ../../../../../../Downloads/caps/Captura%20de%20pantalla%20de%202021-04-01%2011-57-39.png |

1. Imprimir los resultados del programa single.c usando una directiva single dentro de la construcción parallel en lugar de imprimirlos fuera de la región parallel. Añadir lo necesario, dentro de la nueva directiva single incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva single. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

**RESPUESTA**: Captura que muestre el código fuente singleModificado.c

|  |
| --- |
| ../../../../../../Downloads/caps/Captura%20de%20pantalla%20de%202021-04-01%2012-32-42.png |

CAPTURAS DE PANTALLA:

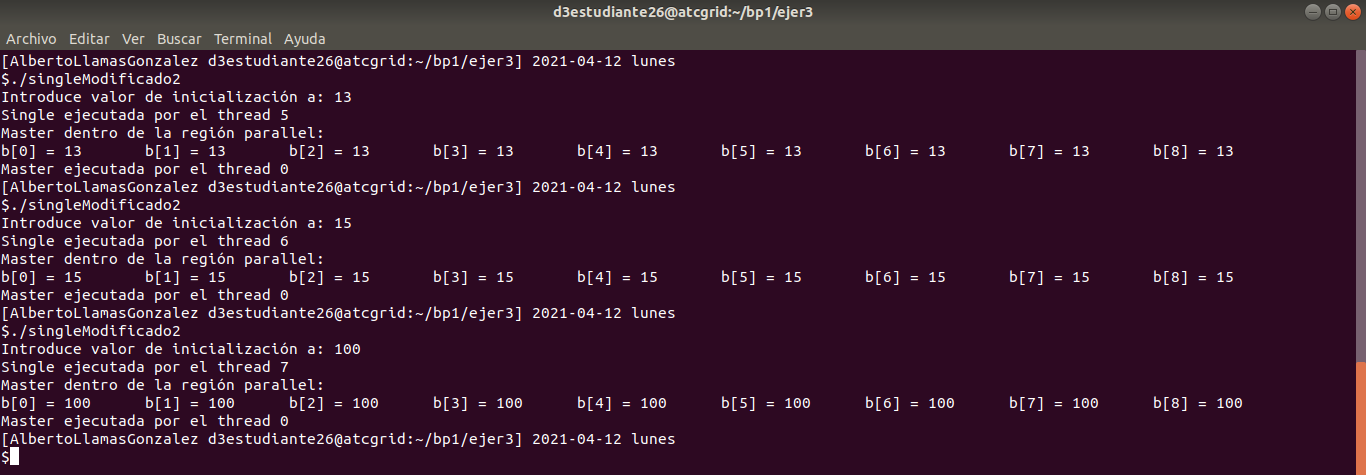


1. Imprimir los resultados del programa single.c usando una directiva master dentro de la construcción parallel en lugar de imprimirlos fuera de la región parallel. Añadir lo necesario, dentro de la nueva directiva master incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva master. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

**RESPUESTA**: Captura que muestre el código fuente singleModificado2.c

|  |
| --- |
| ../../../../../../Downloads/caps/Captura%20de%20pantalla%20de%202021-04-01%2012-41-21.png |

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA: La principal diferencia es que todas las instrucciones contenidas en la directiva "master" se ejecutan siempre por el thread maestro, es decir, el thread que corresponde al identificador 0. Para comprobar esto, podemos ver que tras varias ejecuciones el identificador que se imprime tras el resultado es siempre 0.

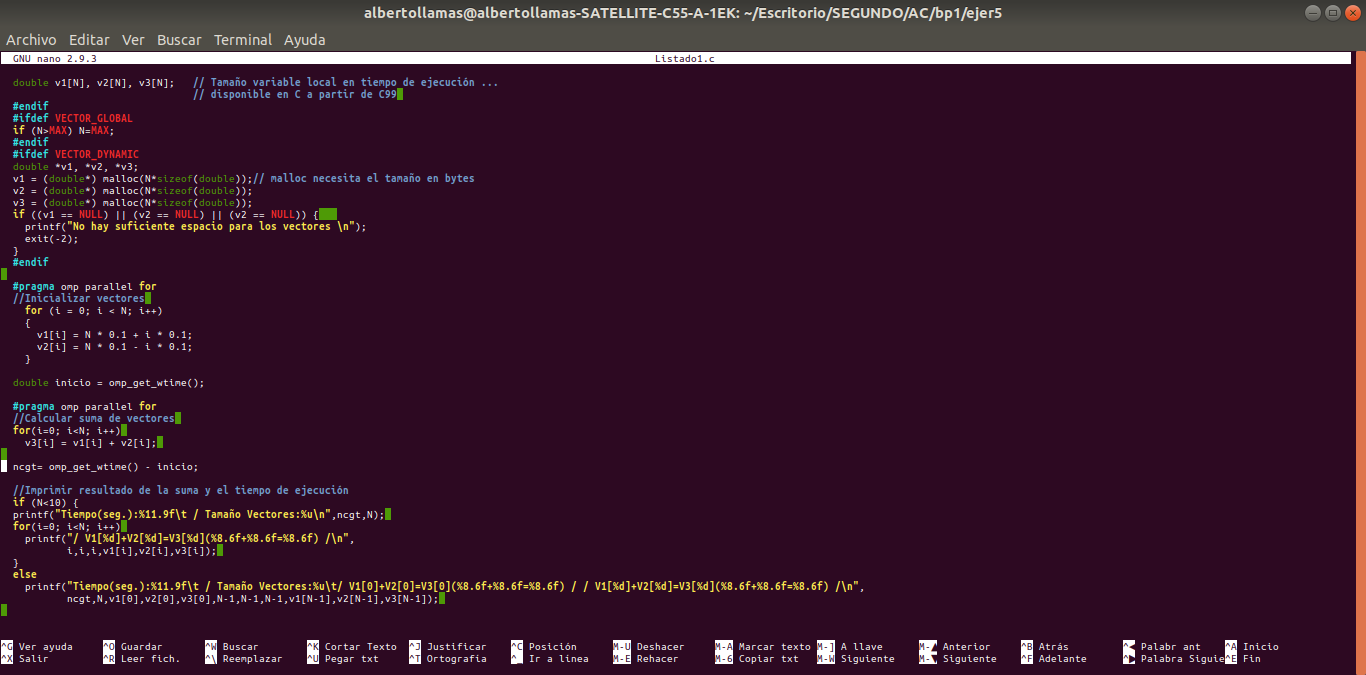
1. ¿Por qué si se elimina directiva barrier en el ejemplo master.c la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

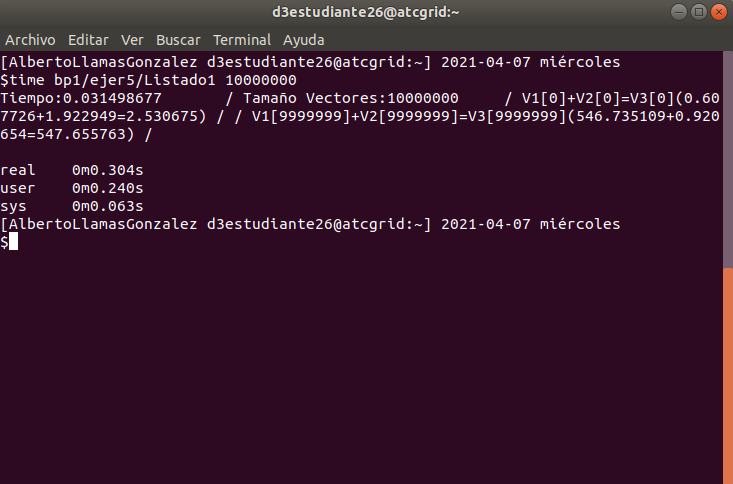
RESPUESTA: Esto ocurre porque tras “atomic” no hay barrera implícita. Esto, en consecuencia, implica que es posible ejecutar el “printf” antes de que se hayan acumulado todas las sumas, por lo que el resultado sería incorrecto (uno de los hilos, tan pronto como no esté ocupado, ejecutará dicho “printf”). La barrera impide esto: espera a que todos los hilos hayan realizado la instrucción de adición acumulativa sobre “suma”, y a continuación se ejecuta el “printf”, teniendo en consecuencia un resultado correcto.

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

1. El programa secuencial C del Listado 1 calcula la suma de dos vectores (v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,…N-1). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar time (Lección 3/ Tema 1) en la línea de comandos para obtener, en atcgrid, el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema generado. Obtenga los tiempos para vectores con 10000000 componentes. ¿La suma de los tiempos de CPU del usuario y del sistema es menor, mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

CAPTURAS DE PANTALLA:

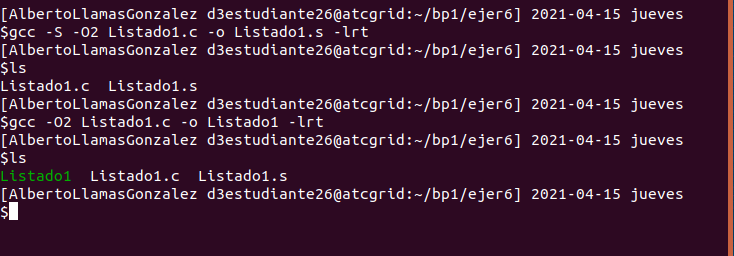




**RESPUESTA:** La suma del tiempo de CPU de usuario y del tiempo del sistema va a ser menor o igual al tiempo al tiempo real. En mi caso, 0,063+0,240 = 0,303 < 0,304. Esto se debe a que el tiempo real es una aproximación del tiempo que puede tardar el programa en ejecutarse. El tiempo de usuario es el tiempo que tarda el usuario en recuperar el control de la terminal tras ejecutar el programa y el tiempo del sistema es el tiempo de espera debido a operaciones de E/S, por ejemplo, es decir, operaciones que utilizan recursos del sistema.

1. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores globales** (para generar el código ensamblador tiene que compilar usando -S en lugar de -o). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para atcgrid los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of FLOating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones clock\_gettime()); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC). Razonar cómo se han obtenido los valores que se necesitan para calcular los MIPS y MFLOPS. Incorporar **el código ensamblador de la parte de la suma de vectores** (no de todo el programa) en el cuaderno.

**CAPTURAS DE PANTALLA** (que muestren la generación del código ensamblador y del código ejecutable, y la obtención de los tiempos de ejecución):

****

Cálculo para 10 y 10000000 componentes

****

RESPUESTA: cálculo de los MIPS y los MFLOPS

Para 10 componentes:

MIPS = NI / (T\_cpu \* 10^6 ) = (6\*10)/(0,000391119\*10^6) = 0,1534059966 MIPS

MFLOPS = NI\_float / (T\_cpu \* 10^6 ) =(1\*10)/(0,000391119\*10^6) = 0,02556766611 MFLOPS

Para 10000000 componentes:

MIPS = NI / (T\_cpu \* 10^6 ) = (6\*10000000)/(0,039321731\*10^6) = 1525,8738227979 MIPS

MFLOPS = NI\_float / (T\_cpu \* 10^6 ) = (1\*10000000)/(0,039321731\*10^6) = 254,3123037996 MFLOPS

RESPUESTA: Captura que muestre el código ensamblador generado de la parte de la suma de vectores

|  |
| --- |
| ../../../../../../Downloads/nuevascaps/Captura%20de%20pantalla%20de%202021-04-15%2023-52-03.png |

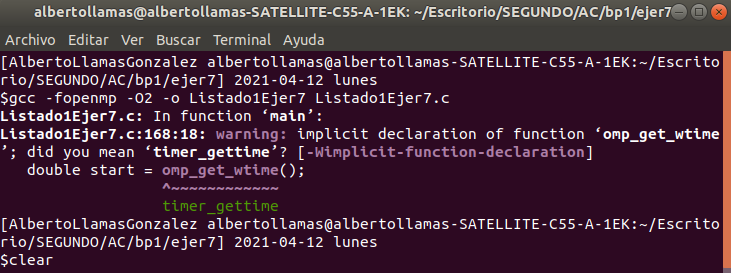
1. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores (v3 = v1 + v2; v3(i)=v1(i)+v2(i), i=0,…N-1) usando las directivas parallel y for. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Listado 1 se debe obtener el tiempo (*elapsed time*) que supone el cálculo de la suma. Para obtener este tiempo usar la función omp\_get\_wtime()**,** que proporciona el estándar OpenMP, en lugar de clock\_gettime(). NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para varios tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

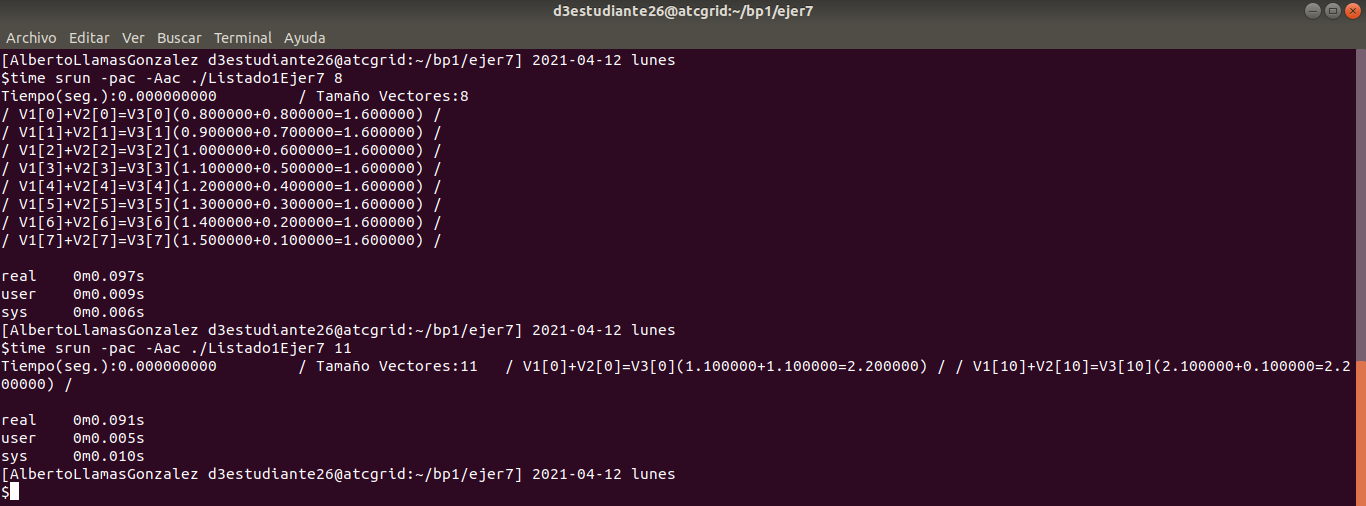
**RESPUESTA:** Captura que muestre el código fuente implementado sp-OpenMP-for.c

|  |
| --- |
| ../../../../../../Downloads/drive-download-20210416T221435Z-001/Captura%20de%20pantalla%20de%202021-04-17%2000-09-55.png |

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

****

****

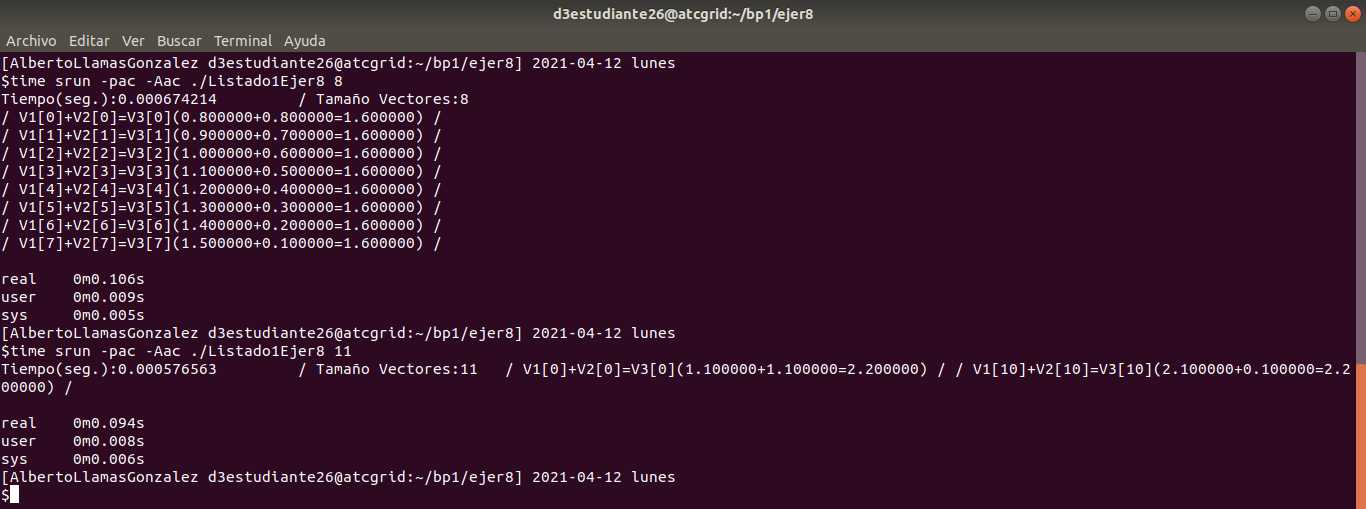
1. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las parallel y sections/section (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva for); es decir, hay que repartir el trabajo (tareas) entre varios threads usando sections/section. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Para obtener este tiempo usar la función omp\_get\_wtime() en lugar de clock\_gettime(). NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

**RESPUESTA:** Captura que muestre el código fuente implementado sp-OpenMP-sections.c

|  |
| --- |
| ../../../../../../Downloads/drive-download-20210416T221435Z-001/Captura%20de%20pantalla%20de%202021-04-17%2000-09-11.png  ../../../../../../Downloads/drive-download-20210416T221435Z-001/Captura%20de%20pantalla%20de%202021-04-17%2000-09-26.png |

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

****

1. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta. NOTA: Al contestar piense sólo en el código, no piense en el computador en el que lo va a ejecutar.

RESPUESTA: En el ejercicio 7, podríamos utilizar un máximo de N threads, sin embargo, este número se limita por el número de cores lógicos del ordenador en el que estemos trabajando, o de la constante definida por OpenMP. Por otra parte, el ejercicio 8 siempre usará un máximo de 4 threads, porque hemos dividido el “for” en cuatro secciones paralelas.

1. Rellenar una tabla como la Tabla 2Error: no se encontró el origen de la referencia para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. Escribir un script para realizar las ejecuciones necesarias utilizando como base el script del seminario de BP0 (se deben imprimir en el script al menos las variables de entorno que ya se imprimen en el script de BP0). En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos (use el máximo número de cores físicos del computador que como máximo puede aprovechar el código, no use un número de threads superior al número de cores físicos). Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado. Observar que el número de componentes en la tabla llega hasta **67108864.**

**RESPUESTA:** Captura del script implementado sp-OpenMP-script10.sh

|  |
| --- |
| ../../../../../../Downloads/drive-download-20210416T221717Z-001/Captura%20de%20pantalla%20de%202021-04-17%2000-07-09.png |

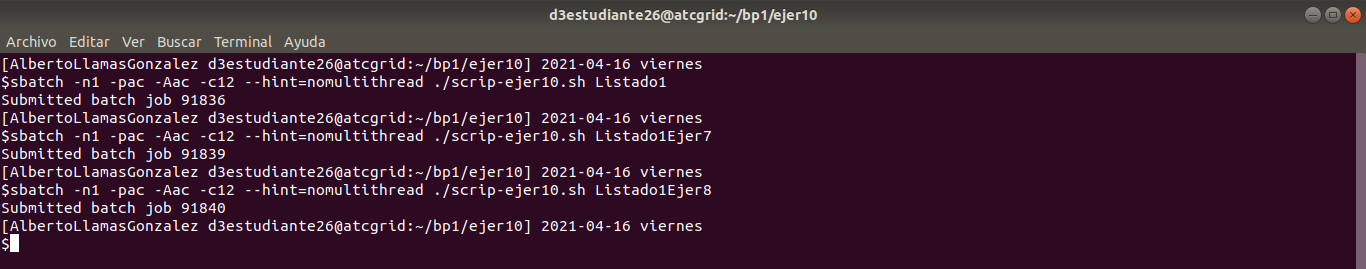
(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):

Compilación en atcgrid:

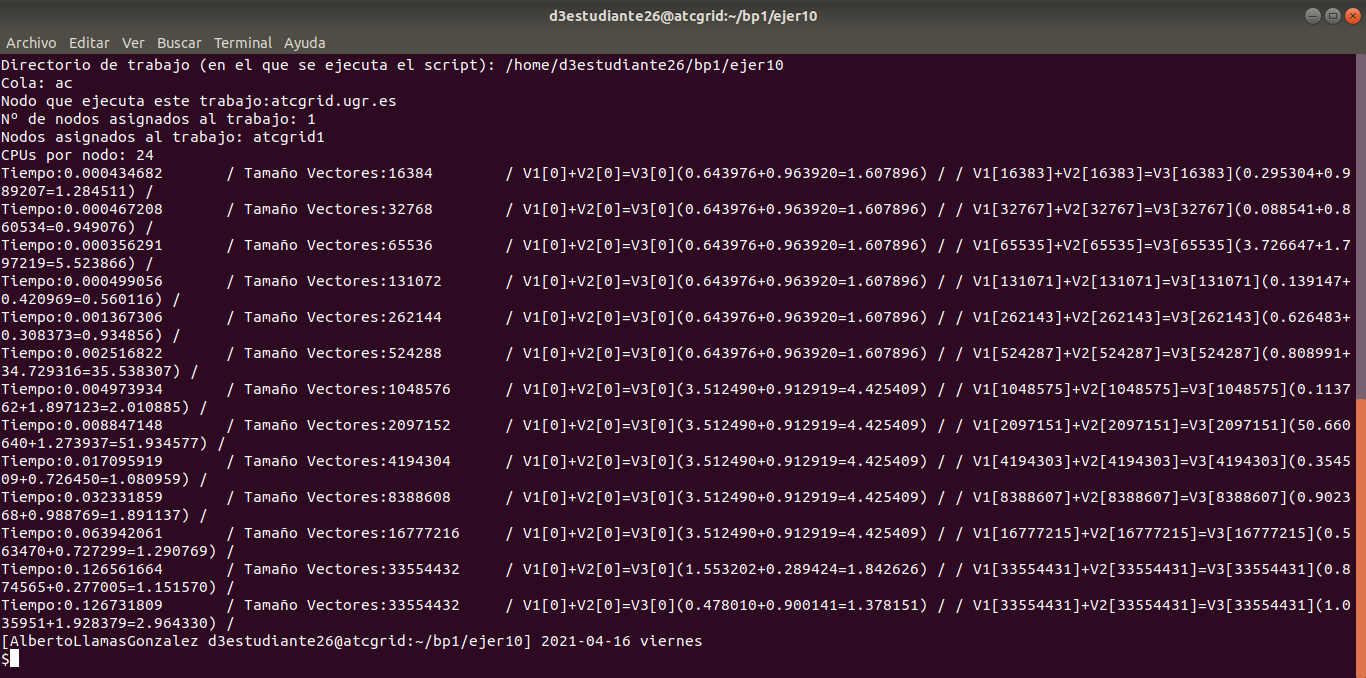
****

Envío a la cola de ejecución:

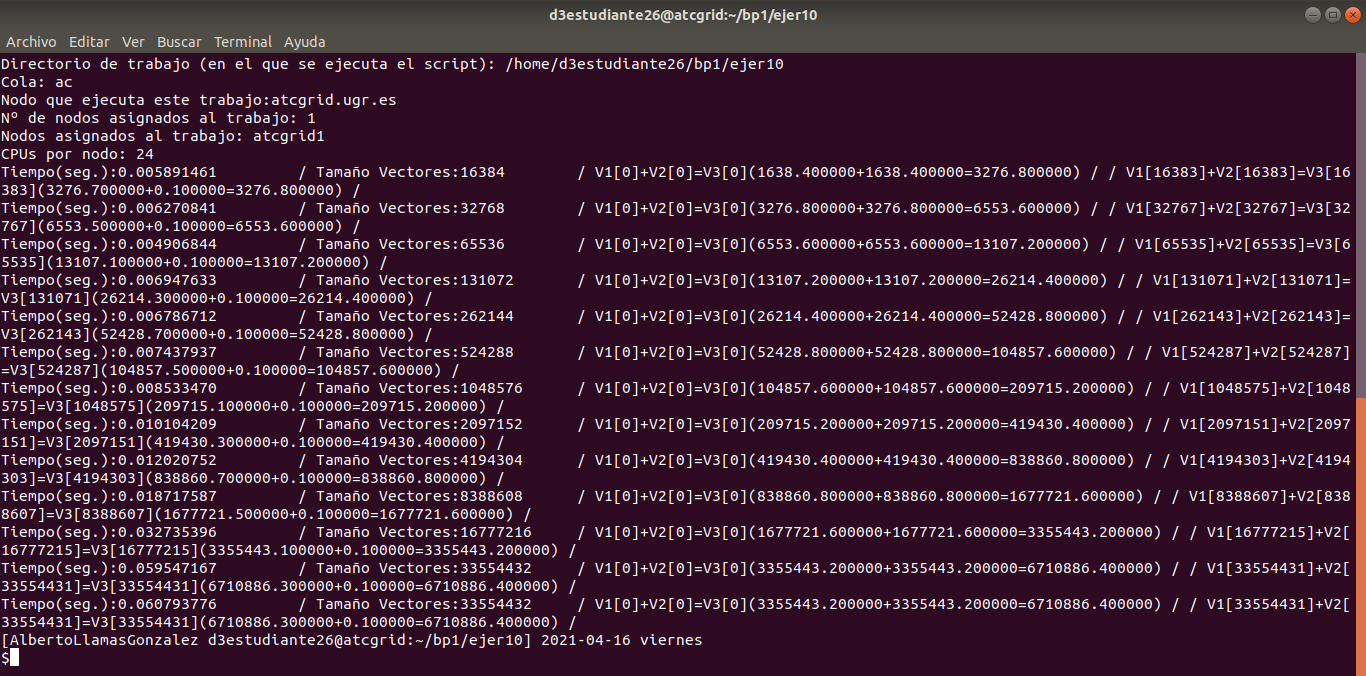


**Ejecución en atcgrid:**

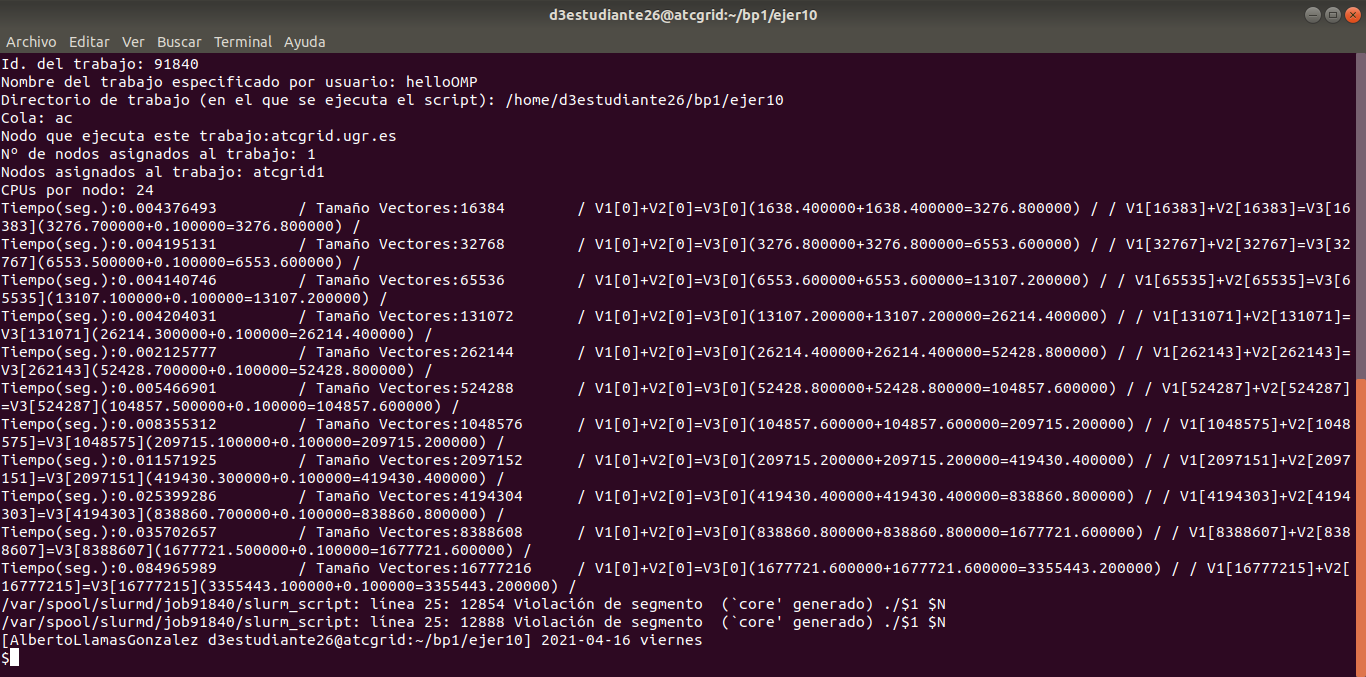
Listado1 (Secuencial):



Listado1Ejer7 (Parallel-for):

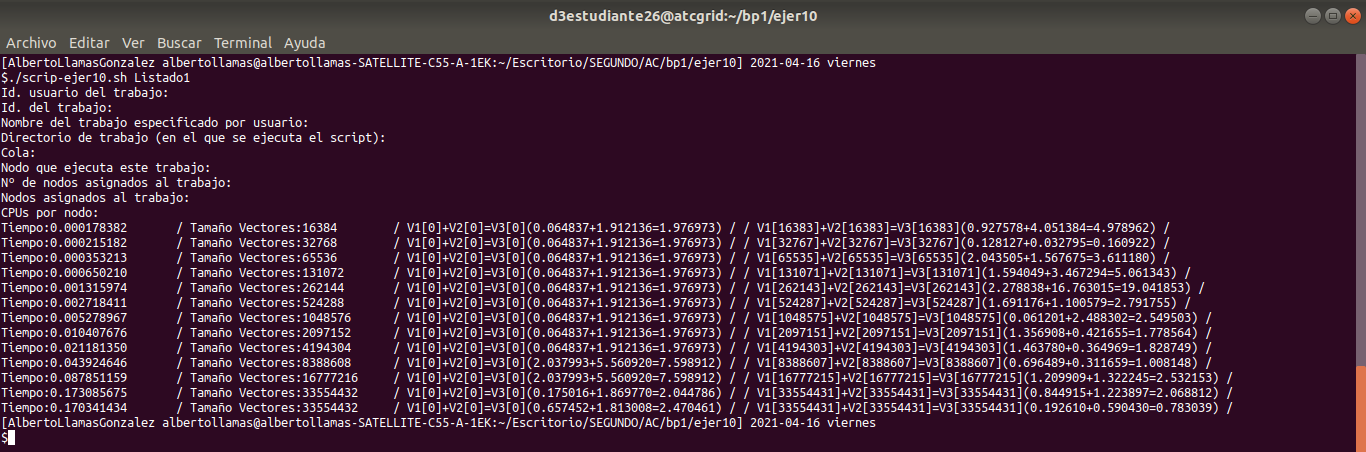


Listado1Ejer8 (Parallel-sections):

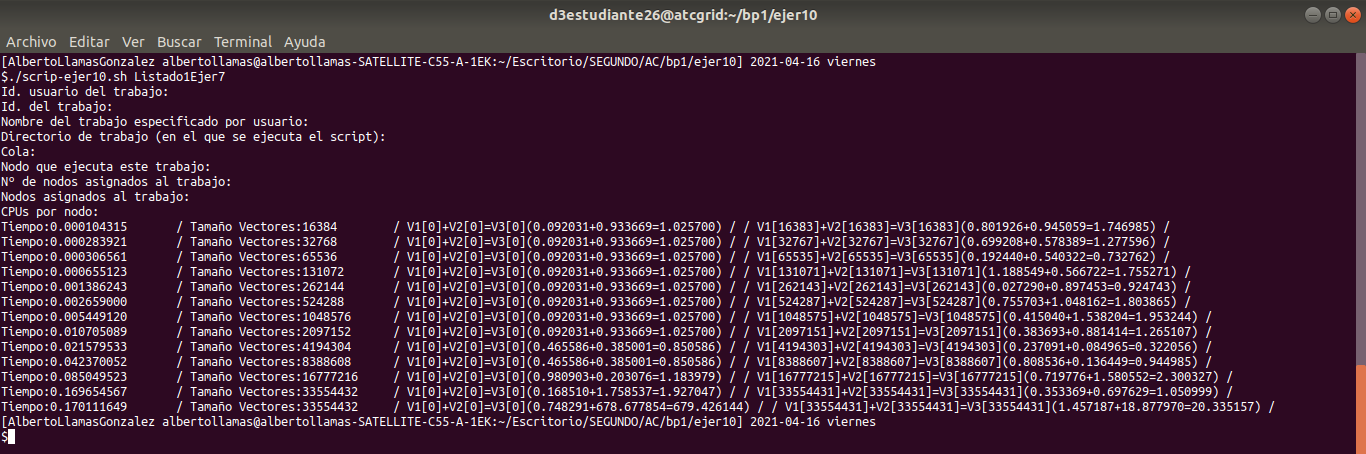


**Ejecución en PC:**

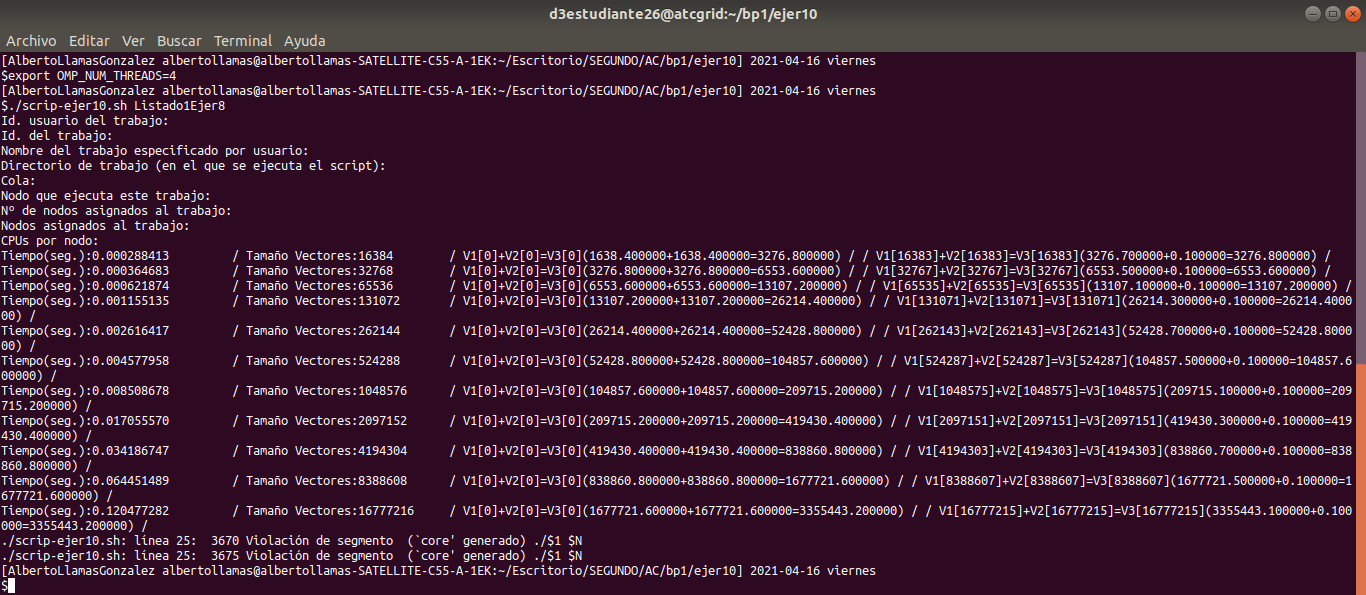
Listado1 (secuencial):

****

Listado1Ejer7 (Parallel-for):

****

Listado1Ejer8 (Parallel-sections):

****

|  |
| --- |
| **Tabla 2.** Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados, que debe coincidir con el número de cores físicos y cores lógicos utilizados.  PARA ATCGRID: |
| |  |  |  |  | | --- | --- | --- | --- | | **Nº de Componentes** | **T. secuencial vect. Globales**  **1 thread=core** | **T. paralelo (versión** for**)**  **12 threads = cores lógicos = cores físicos** | **T. paralelo (versión** sections)  **12 threads = cores lógicos = cores físicos** | | **16384** | 0.000434682 | 0.005891461 | 0.004376493 | | **32768** | 0.000467208 | 0.006270841 | 0.004195131 | | **65536** | 0.000356291 | 0.004906844 | 0.004140746 | | **131072** | 0.000499056 | 0.006947633 | 0.004204031 | | **262144** | 0.001367306 | 0.006786712 | 0.002125777 | | **524288** | 0.002516822 | 0.007437937 | 0.005466901 | | **1048576** | 0.004973934 | 0.008533470 | 0.008355312 | | **2097152** | 0.008847148 | 0.010104209 | 0.011571925 | | **4194304** | 0.017095919 | 0.012020752 | 0.025399286 | | **8388608** | 0.032331859 | 0.018717587 | 0.035702657 | | **16777216** | 0.063942061 | 0.032735396 | 0.084965989 | | **33554432** | 0.126561664 | 0.059547167 | - | | **67108864** | 0.126731809 | 0.060793776 | - | |

|  |
| --- |
| PARA PC: |
| |  |  |  |  | | --- | --- | --- | --- | | **Nº de Componentes** | **T. secuencial vect. Globales**  **1 thread=core** | **T. paralelo (versión** for**)**  **4 threads = cores lógicos = cores físicos** | **T. paralelo (versión** sections)  **4 threads = cores lógicos = cores físicos** | | **16384** | 0.000178382 | 0.000104315 | 0.000288413 | | **32768** | 0.000215182 | 0.000283921 | 0.000364683 | | **65536** | 0.000353213 | 0.000306561 | 0.000621874 | | **131072** | 0.000650210 | 0.000655123 | 0.001155135 | | **262144** | 0.001315974 | 0.001386243 | 0.002616417 | | **524288** | 0.002718411 | 0.002659000 | 0.004577958 | | **1048576** | 0.005278967 | 0.005449120 | 0.008508678 | | **2097152** | 0.010407676 | 0.010705089 | 0.017055570 | | **4194304** | 0.021181350 | 0.021579533 | 0.034186747 | | **8388608** | 0.043924646 | 0.042370052 | 0.064451489 | | **16777216** | 0.087851159 | 0.085049523 | 0.120477282 | | **33554432** | 0.173085675 | 0.169654567 | - | | **67108864** | 0.170341434 | 0.170111649 | - | |

1. Rellenar una tabla como la Tabla 3 para atcgrid con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con time para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads (que debe coincidir con el número cores físicos y lógicos) que usan los códigos. Escribir un script para realizar las ejecuciones necesarias utilizando como base el script del seminario de BP0 (se deben imprimir en el script al menos las variables de entorno que ya se imprimen en el script de BP0) ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

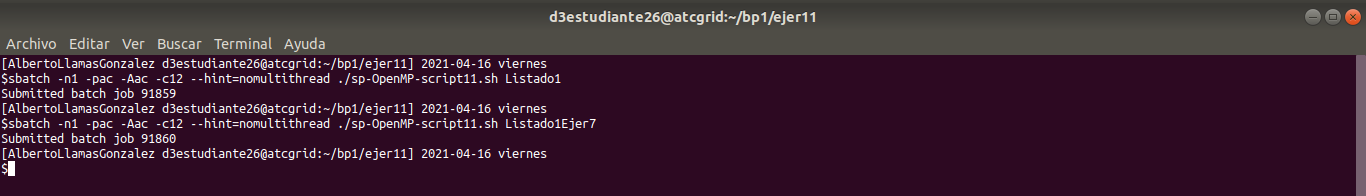
**RESPUESTA:** Captura del script implementado sp-OpenMP-script11.sh

|  |
| --- |
| ../../../../../../Downloads/drive-download-20210416T221717Z-001/Captura%20de%20pantalla%20de%202021-04-17%2000-06-46.png |

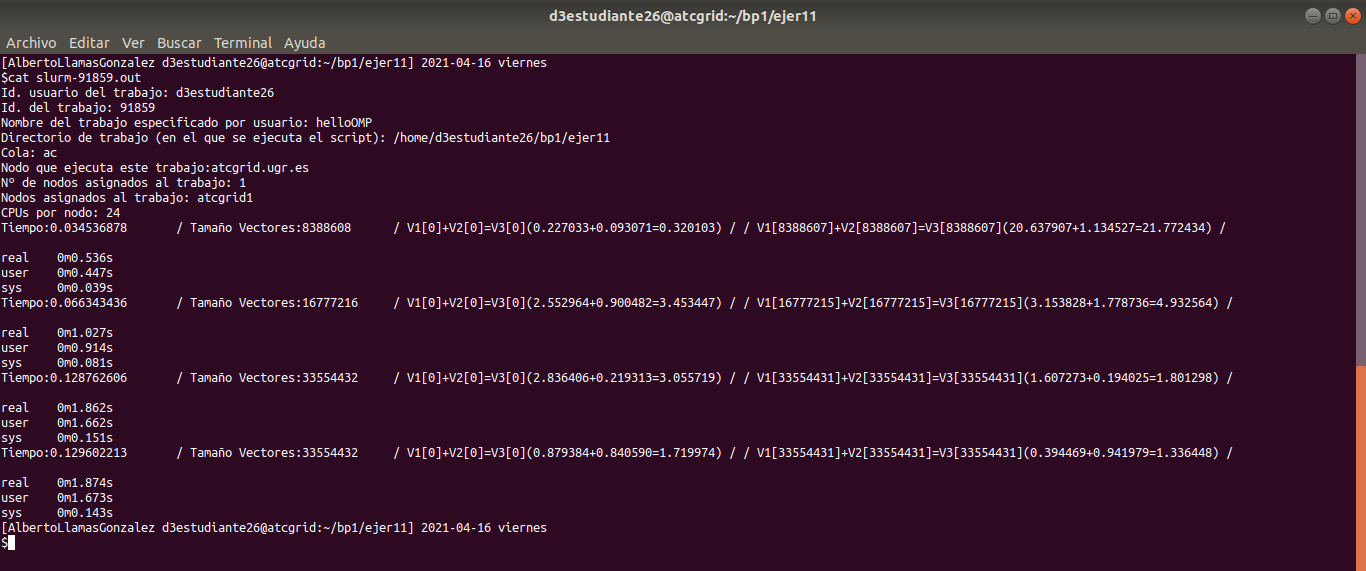
(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

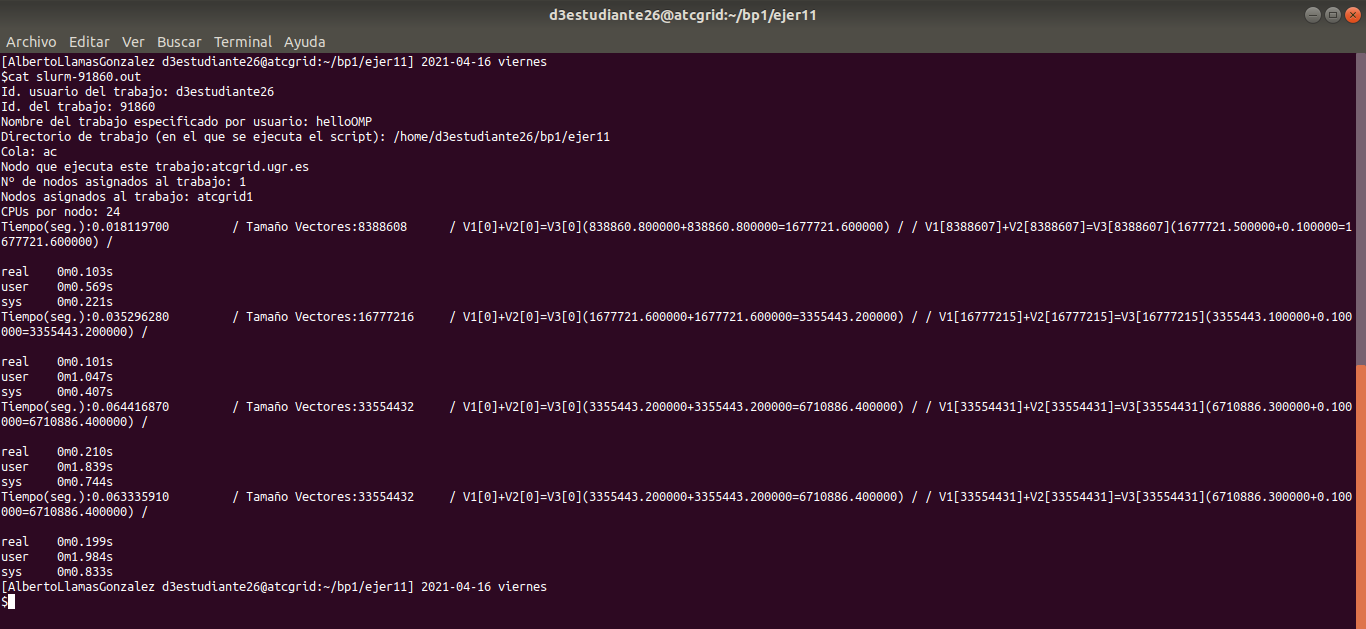
Envío a la cola de ejecución atcgrid:



Listado1 (secuencial):



Listado1Ejer7 (Parallel-for):



|  |
| --- |
| **Tabla 3.** Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados. |
| |  |  |  | | --- | --- | --- | | **Nº de Componentes** | **Tiempo secuencial vect. Globales**  **1 thread = 1 core lógico = 1 core físico**  ***Elapsed CPU-user CPU- sys*** | **Tiempo paralelo/versión** for  **12 Threads = cores lógicos=cores físicos**  ***Elapsed CPU-user CPU- sys*** | | **8388608** | 0.536s 0.447s 0.039s | 0.103s 0.569s 0.221s | | **16777216** | 1.027s 0.914s 0.081s | 0.101s 1.047s 0.407s | | **33554432** | 1.862s 1.662s 0.151s | 0.210s 1.839s 0.744s | | **67108864** | 1.874s 1.673s 0.143s | 0.199s 1.984s 0.833s | |

En la secuencial la suma del tiempo cpu-user y del cpu-sys es igual al elapsed time, sin embargo, en la que utiliza paralelización la suma de ambos es mayor que el elapsed time y esto se debe a que se cuenta el tiempo que consume cada core físico y al sumarlo no suma el tiempo transcurrido realmente ya que son simultáneos y no secuenciales.