|  |  |  |
| --- | --- | --- |
| **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): Brian Sena Simons  Grupo de prácticas y profesor 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 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

|  |
| --- |
|  |

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

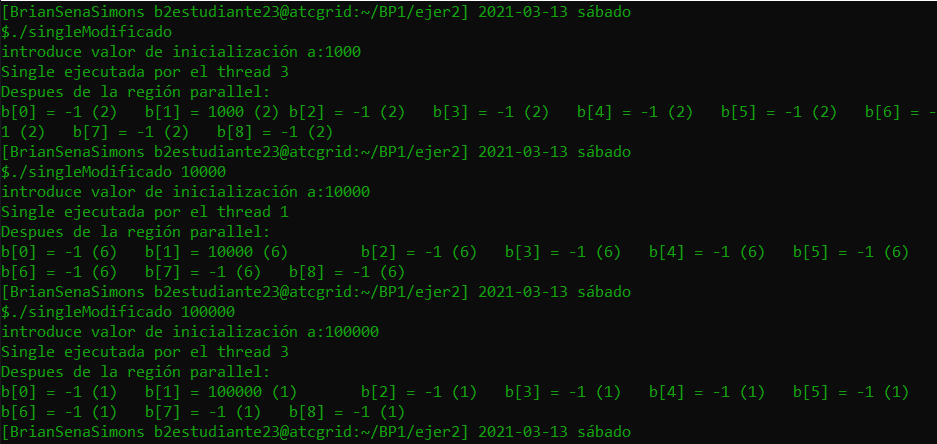
|  |
| --- |
|  |

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

|  |
| --- |
|  |

CAPTURAS DE PANTALLA:

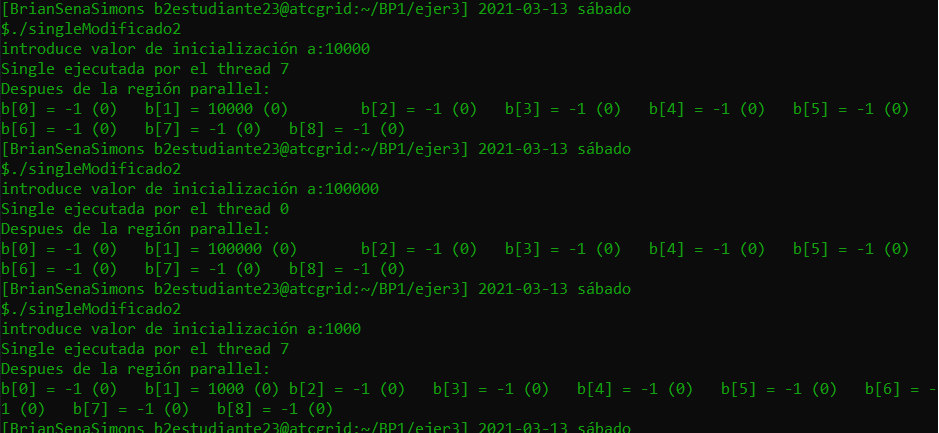


1. Imprimir los resultados del programa single.c usando una directiva master dentro de la construcción parallel en lugar de imprimirlos fuera de la región parallel. Añadir lo necesario, dentro de la nueva directiva master incorporada, para que se imprima el identificador del thread que ejecuta el bloque 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

|  |
| --- |
|  |

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA:

Hemos podido observar que en ambas ejecuciones, apenas una hebra es la que finaliza el código después de la región paralela. Sin embargo, utilizando la directiva single esa hebra es escogida aleatoriamente según esté ocupada o ociosa. Mientras que en la directiva master ocurre que siempre es la hebra 0 (Maestra). Lo único es que no tiene barreras por lo que podría ocasionar valores erróneos debido a la falta de sincronización entre hebras.

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.

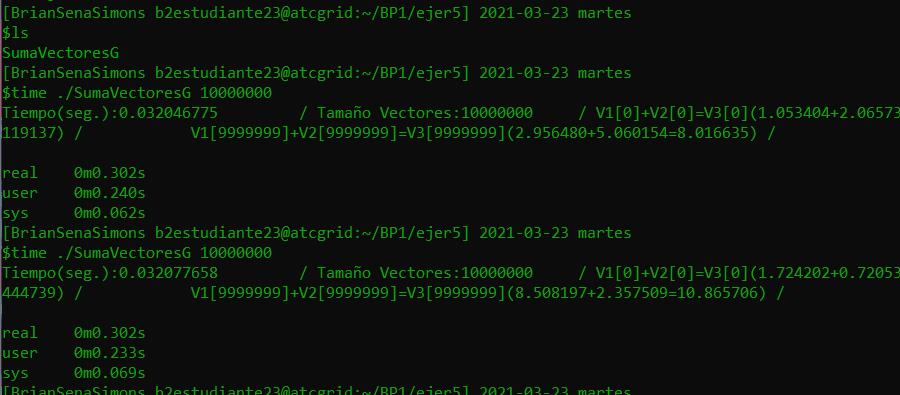
RESPUESTA:

Al quitar la directiva *barrier*, la hebra maestra (0), y cualquiera de las demás, si ha finalizado sus cálculos seguirá la ejecución secuencial del código y terminará imprimiendo por pantalla antes de que las que aún estén pendientes por terminar hayan registrado la suma.

# 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:

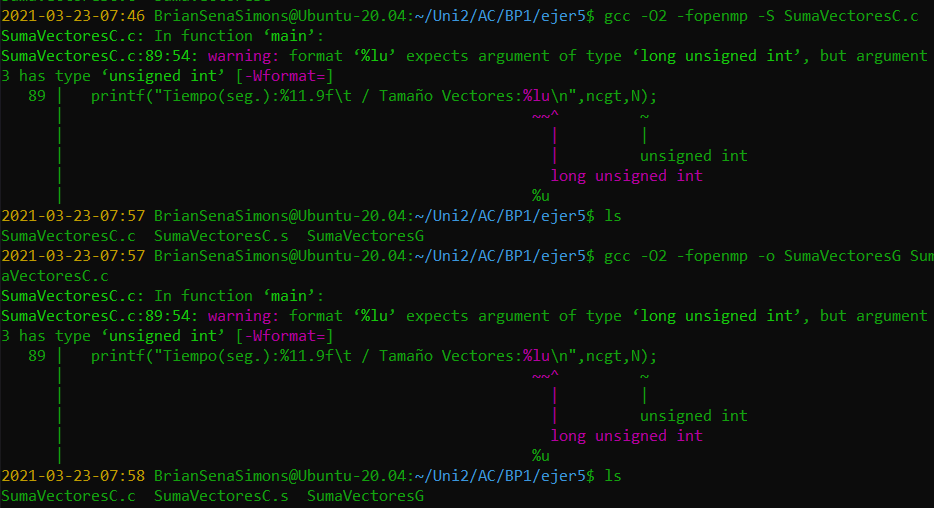


**RESPUESTA:**

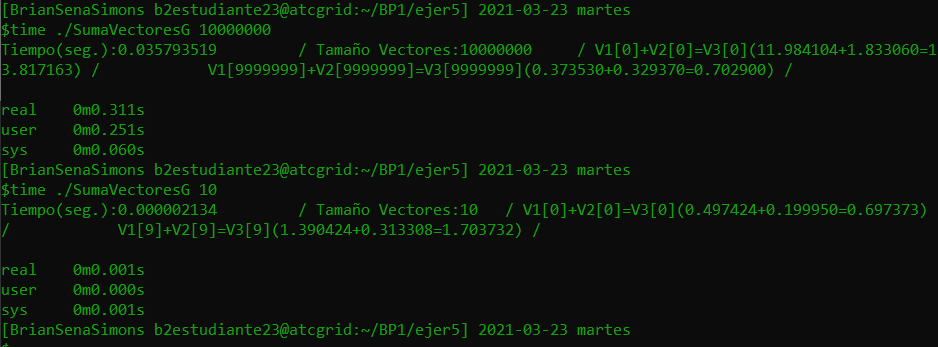
La suma de los **tiempos de CPU del usuario y del sistema** son siempre igual o menor que al tiempo real en programas secuenciales, en mi caso son iguales que el tiempo real. Porque el tiempo real es una aproximación de lo que puede tardar el programa; El tiempo de usuario es lo que tarda que el usuario le da enter hasta que recupera el control de la terminal y el tiempo de sistema es lo que hace por debajo del programa usando 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):

****

RESPUESTA: cálculo de los MIPS y los MFLOPS



Si recordamos el código, tiene 6 instrucciones por bucle y 3 para inicializar las variables. Con los tiempos obtenidos del programa puedo calcular los MIPS que tiene. Para los MFLOPS se aplica la misma lógica pero teniendo en cuenta que ahora hablamos de operaciones en coma flotante.

Para 1000:

MIPS = NI/Tcpu = F / CPI = 3 + (6\*10000000)/(0,035793519\*10^6) =

MFLOPS = Oper\_coma\_flotante / Tcpu = 7\*10000000/(0,035793519\*10^6) =

Para 10:

MIPS = NI/Tcpu = F / CPI = 3 + (6\*10)/(0,000002134\*10^6) =

MFLOPS = Oper\_coma\_flotante / Tcpu = 7\*10/(0,000002134\*10^6) =

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

|  |
| --- |
|  |

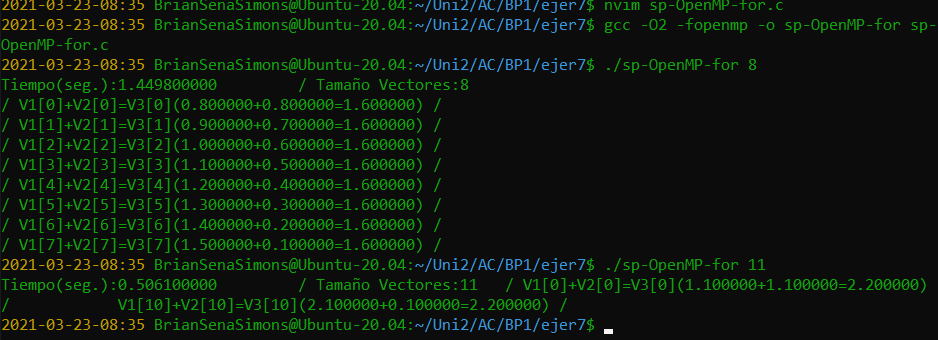
1. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores (v3 = v1 + v2; v3(i)=v1(i)+v2(i), i=0,…N-1) usando las directivas parallel y for. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en 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

|  |
| --- |
|  |

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

****

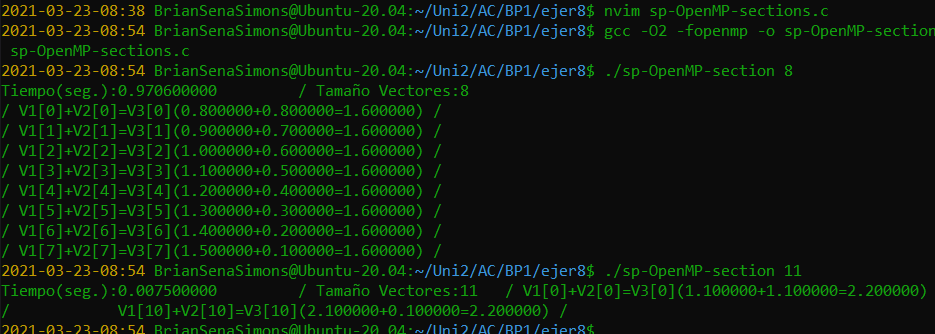
1. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las parallel y sections/section (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva for); es decir, hay que repartir el trabajo (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

|  |
| --- |
|  |

(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 como usábamos “omp parallel for”, automáticamente el código se ejecuta con el máximo número de hebras del sistema, siempre y cuando no hayamos establecido un máximo manualmente con “omp\_set\_num\_threads(x)”.

En el caso del ejercicio 8, al utilizar sections y apenas haber desarrollado 4 de ellas, la computadora ejecutará apenas 4 hebras concurrentemente.

1. Rellenar una tabla como la Tabla 2 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

|  |
| --- |
| Pd: He cambiado los prints de los archivos para qué solo me impriman el tiempo de ejecución y así sea más fácil copiar y pegar en la tabla. |

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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

Tiempo ejecución en mi PC:

|  |
| --- |
|  |
| |  |  |  |  | | --- | --- | --- | --- | | **Nº de Componentes** | **T. secuencial vect. Globales**  **1 thread=core** | **T. paralelo (versión** for**)**  **¿?threads = cores lógicos = cores físicos** | **T. paralelo (versión** sections)  **¿?threads = cores lógicos = cores físicos** | | **16384** | 0.000041200 | 0.001006600 | 0.001321500 | | **32768** | 0.000059100 | 0.001043800 | 0.000992500 | | **65536** | 0.000131500 | 0.000109100 | 0.000485200 | | **131072** | 0.000318200 | 0.001294200 | 0.000740100 | | **262144** | 0.001326300 | 0.001727800 | 0.000722500 | | **524288** | 0.002134000 | 0.001788000 | 0.001733000 | | **1048576** | 0.004699100 | 0.003950200 | 0.003379900 | | **2097152** | 0.009201100 | 0.005618900 | 0.006826000 | | **4194304** | 0.018782900 | 0.012008700 | 0.013549800 | | **8388608** | 0.033418900 | 0.022461100 | 0.026907500 | | **16777216** | 0.066456700 | 0.053086700 | 0.046812400 | | **33554432** | 0.128733100 | 0.112055300 | 0.102757800 | | **67108864** | 0.146928700 | 0.105811300 | 0.100362300 | |

Tiempo de Ejecución en ATC-GRID:

|  |
| --- |
|  |
| |  |  |  |  | | --- | --- | --- | --- | | **Nº de Componentes** | **T. secuencial vect. Globales**  **1 thread=core** | **T. paralelo (versión** for**)**  **¿?threads = cores lógicos = cores físicos** | **T. paralelo (versión** sections)  **¿?threads = cores lógicos = cores físicos** | | **16384** | 0.000430140 | 0.000458203 | 0.000451583 | | **32768** | 0.000884755 | 0.000985712 | 0.000976808 | | **65536** | 0.000385149 | 0.000497684 | 0.000568420 | | **131072** | 0.000526832 | 0.001067813 | 0.000624850 | | **262144** | 0.001340818 | 0.001326110 | 0.001573656 | | **524288** | 0.002575860 | 0.002114229 | 0.002370473 | | **1048576** | 0.005019039 | 0.004555743 | 0.004605968 | | **2097152** | 0.008938022 | 0.008390326 | 0.008610044 | | **4194304** | 0.016864381 | 0.015554648 | 0.016047191 | | **8388608** | 0.034699956 | 0.031688750 | 0.031323101 | | **16777216** | 0.066598025 | 0.060837433 | 0.059916690 | | **33554432** | 0.131836073 | 0.120414563 | 0.117755380 | | **67108864** | 0.133162577 | 0.118927237 | 0.119066168 | |

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

|  |
| --- |
|  |

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

**Tiempo Ejecución en mi PC:**

|  |
| --- |
|  |
| |  |  |  | | --- | --- | --- | | **Nº de Componentes** | **Tiempo secuencial vect. Globales**  **1 thread = 1 core lógico = 1 core físico**  ***Elapsed CPU-user CPU- sys*** | **Tiempo paralelo/versión** for  **¿? Threads = cores lógicos=cores físicos**  ***Elapsed CPU-user CPU- sys*** | | **8388608** | real 0m0,289s  user 0m0,266s  sys 0m0,016s | real 0m0,067s  user 0m0,141s  sys 0m0,094s | | **16777216** | real 0m0,558s  user 0m0,516s  sys 0m0,031s | real 0m0,122s  user 0m0,438s  sys 0m0,266s | | **33554432** | real 0m1,124s  user 0m1,016s  sys 0m0,109s | real 0m0,268s  user 0m0,750s  sys 0m0,703s | | **67108864** | real 0m1,111s  user 0m1,031s  sys 0m0,078s | real 0m0,268s  user 0m0,969s  sys 0m0,516s | |
| 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  **¿? Threads = cores lógicos=cores físicos**  ***Elapsed CPU-user CPU- sys*** | | **8388608** | real 0m0.544s  user 0m0.459s  sys 0m0.044s | real 0m0.083s  user 0m0.081s  sys 0m0.054s | | **16777216** | real 0m0.996s  user 0m0.900s  sys 0m0.070s | real 0m0.149s  user 0m0.163s  sys 0m0.111s | | **33554432** | real 0m1.927s  user 0m1.738s  sys 0m0.139s | real 0m0.281s  user 0m0.326s  sys 0m0.213s | | **67108864** | real 0m1.899s  user 0m1.740s  sys 0m0.148s | real 0m0.276s  user 0m0.322s  sys 0m0.205s | |

En el caso de la ejecución secuencial el tiempo total, elapsed, es >= que la suma de “user” + “sys” debido a que como no tiene una ejecución en paralela el tiempo total coincide con lo que ha tardado el programa. Pero en el caso del “Omp-for” el tiempo elapsed es siempre < que la suma “user” + “sys” debido a que tiene una ejecución en paralelo. Time lo que hace es sumar el tiempo que ha tardado cada hebra dando un tiempo que no coincide con lo que ha tardado realmente.