# Multiplicación de matrices de forma secuencial y paralela utilizando API OpenMP

Presentado por:
Santiago Sosa Herrera

Presentado a:
Ramiro Andrés Barrios Valencia

HPC: High Performance Computing
Ingeniería de Sistemas y Computación
Universidad Tecnológica de Pereira
2023

# Tabla de contenido

## Resumen

En este trabajo se aborda el problema de la multiplicación de matrices en C, utilizando el api de OpenMP para mejorar su rendimiento. Primero, se utilizó la solución secuencial elaborada en un trabajo anterior de la materia y se midió su tiempo de ejecución para matrices de diferentes tamaños. Luego, se comparó su desempeño con los métodos usados también en el otro trabajo, los cuales se volverán a mostrar.

# Introducción

La multiplicación de matrices es una operación fundamental en la computación científica y el procesamiento de datos. Esta operación se utiliza en una amplia gama de aplicaciones, incluyendo el análisis de datos, el aprendizaje automático, la simulación de sistemas físicos, entre otros. Sin embargo, la multiplicación de matrices puede ser computacionalmente costosa y presenta varios desafíos en cuanto a la complejidad del algoritmo y la optimización de código.

Uno de los principales problemas asociados con la multiplicación de matrices es su complejidad computacional, que crece de manera exponencial con el tamaño de las matrices. Esto puede limitar el uso de la multiplicación de matrices en aplicaciones en las que se requiere procesamiento rápido de grandes conjuntos de datos. Para hacer frente a este problema, se han desarrollado diversos enfoques y técnicas de optimización de código, como la utilización de algoritmos paralelos y la explotación de la memoria caché.

La programación paralela es un enfoque comúnmente utilizado para optimizar la multiplicación de matrices. Esto implica dividir la tarea de multiplicación de matrices en tareas más pequeñas que se ejecutan simultáneamente en diferentes hilos o procesos. Esta técnica puede mejorar significativamente el rendimiento y la eficiencia del código.

En este trabajo se presentan los resultados de un estudio en el que se compararon diferentes enfoques para la multiplicación de matrices en el lenguaje de programación C. Se evaluó el rendimiento del api de OpenMP comparándola con métodos como el secuencial, por hilos, por procesos, optimización por consola, y matriz transpuesta. Los resultados muestran que el enfoque de programación paralela utilizando hilos y la matriz transpuesta son altamente eficientes en términos de tiempo de ejecución y rendimiento en comparación con otros enfoques.

## Marco conceptual

**High Performance Computing:** High Performance Computing es un área de la informática que se dedica a crear sistemas informáticos y software que puedan manejar tareas de alta complejidad y procesamiento de grandes volúmenes de datos en un período de tiempo muy corto.

**Multiplicación matricial:** La multiplicación matricial es una operación que se realiza entre dos matrices y que arroja como resultado una tercera matriz. Esta operación es muy importante en distintos campos de la ciencia y la ingeniería, y se utiliza en problemas lineales y de optimización.

**Complejidad Computacional:** La complejidad computacional es una forma de medir los recursos que se necesitan para resolver un problema computacional. Sirve para determinar la dificultad de un problema y para comparar la eficiencia de distintos algoritmos.

**Programación paralela:** La programación paralela es una técnica de programación que se basa en dividir un problema en tareas más pequeñas y ejecutarlas en múltiples procesadores al mismo tiempo. Esta técnica se utiliza para mejorar la eficiencia de los sistemas informáticos y acelerar la resolución de problemas.

**Hilos:** Los hilos son una forma de dividir la ejecución de un programa en unidades más pequeñas y manejables, lo que permite que varias tareas se realicen simultáneamente. Cada hilo funciona de manera independiente, con su propia pila de memoria y compartiendo recursos con otros hilos dentro del mismo proceso. El uso de hilos puede mejorar significativamente la eficiencia y velocidad de ejecución de una aplicación.

**Speedup:** El speedup se utiliza para medir la mejora en la velocidad de ejecución de un programa cuando se ejecuta en un sistema de cómputo más rápido o en paralelo. Se mide en términos de una relación entre el tiempo que tarda un programa en ejecutarse en un sistema de cómputo determinado y el tiempo que tarda en ejecutarse en un sistema más rápido. Cuanto mayor sea el speedup, mayor será la mejora en el rendimiento.

**OpenMP**: OpenMP es una interfaz de programación de aplicaciones para la programación multiproceso de memoria compartida en múltiples plataformas. Permite

añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join.

## Marco Contextual

### Características de la maquina

Las características del pc donde se realizaron las pruebas son los siguientes:

- Procesador AMD A8-7410 apu with amd Radeon r5 graphics x 4
- Memoria 6,7 GiB
- SO 64 bits
- Disco de 295 GB
- Sistema operativo Linux Mint 19
    - Gnome 3.28.2

### Desarrollo

Para el inicio del trabajo se empezó utilizando la multiplicación de matrices de forma normal, es decir secuencial, este código tenía que recibir por parámetro en consola el tamaño de las matrices, que eran cuadradas, para luego modificarlo con el openMP, se le agrego una forma de medir el tiempo en el que se demoraba el programa en ejecutar todo, con ayuda de la librería time.h.

Posteriormente con ayuda de un script .sh se ejecutaban estos programas una cantidad de veces determinada para cada tamaño de las matrices calculadas, las cuales fueron 10, 100, 200, 400, 800, 1600 y 3.200.

Entendiendo como funciona la multiplicación con hilos básicamente se puede entender como funcionan con openMP, pero ejecutar esto en c en realidad es de manera sencilla; se utiliza la directiva #pragma omp parallel for para paralelizar el bucle externo de la multiplicación de matrices. La variable sum se declara como privada para que cada hilo tenga su propia copia y evitar conflictos de memoria.

**Pruebas**

Para las pruebas de los otros métodos se tuvieron en cuenta las conclusiones a las que se habían llegado en el trabajo anterior y se rescataron las mejores versiones de cada método para poder comparar.

1. Resultados de ejecutar el algoritmo en forma secuencial:

| tiempo\matriz | 10 | 100 | 200 | 400 | 600 | 800 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|---|
| Tiempo transcurrido: | 0,000112 | 0,014843 | 0,098059 | 0,888076 | 5,552384 | 15,000685 | 31,377433 | 266,019738 |
| Tiempo transcurrido: | 0,000204 | 0,016888 | 0,093718 | 0,877490 | 5,719945 | 15,002777 | 31,093422 | 264,814952 |
| Tiempo transcurrido: | 0,000205 | 0,015730 | 0,094430 | 0,871610 | 5,388444 | 14,974775 | 31,121343 | 265,222653 |
| Tiempo transcurrido: | 0,000239 | 0,013825 | 0,095561 | 0,871919 | 5,442154 | 14,970025 | 31,072066 | 264,939481 |
| Tiempo transcurrido: | 0,000095 | 0,020637 | 0,094184 | 0,865209 | 5,373655 | 14,995896 | 31,147720 | 264,807805 |
| Tiempo transcurrido: | 0,000090 | 0,011772 | 0,094802 | 0,886910 | 5,498816 | 14,992217 | 31,262192 | 265,177446 |
| Tiempo transcurrido: | 0,000210 | 0,013281 | 0,094455 | 0,869835 | 5,413974 | 14,988915 | 31,231712 | 265,524536 |
| Tiempo transcurrido: | 0,000088 | 0,011850 | 0,095684 | 0,869758 | 5,405491 | 14,939653 | 31,977184 | 265,122534 |
| Tiempo transcurrido: | 0,000216 | 0,016110 | 0,094074 | 0,866782 | 5,402242 | 14,905383 | 31,795027 | 264,827617 |
| Tiempo transcurrido: | 0,000210 | 0,018044 | 0,094225 | 0,861536 | 5,395378 | 15,521809 | 31,814963 | 275,715047 |
| Promedio | 0,0001669 | 0,015298 | 0,0949192 | 0,8729125 | 5,4592483 | 15,0292135 | 31,3893062 | 266,2171809 |

2. Mejor Resultado de ejecutar el algoritmo con hilos (2):

| tiempo\matriz | 10 | 100 | 200 | 400 | 600 | 800 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|---|
| Tiempo transcurrido: | 0,000709 | 0,017464 | 0,125778 | 1,033544 | 6,423726 | 16,628052 | 36,349787 | 296,779504 |
| Tiempo transcurrido: | 0,000950 | 0,016912 | 0,132485 | 1,033160 | 6,197521 | 15,973700 | 32,726146 | 285,298079 |
| Tiempo transcurrido: | 0,000853 | 0,017015 | 0,128233 | 1,036969 | 6,186428 | 15,875732 | 32,515568 | 277,336486 |
| Tiempo transcurrido: | 0,000802 | 0,025096 | 0,124889 | 1,028659 | 6,782820 | 15,957374 | 32,440586 | 278,268634 |
| Tiempo transcurrido: | 0,001075 | 0,023157 | 0,129226 | 1,033115 | 6,172454 | 15,915244 | 32,519389 | 277,444121 |
| Tiempo transcurrido: | 0,001148 | 0,017002 | 0,136628 | 1,052246 | 6,187337 | 15,866347 | 32,533681 | 293,540716 |
| Tiempo transcurrido: | 0,001121 | 0,025109 | 0,127673 | 1,099258 | 6,234362 | 16,008939 | 32,432589 | 276,682155 |
| Tiempo transcurrido: | 0,000730 | 0,024534 | 0,125077 | 1,031679 | 6,172880 | 15,843013 | 32,445940 | 277,891267 |
| Tiempo transcurrido: | 0,000818 | 0,019286 | 0,124015 | 1,031721 | 6,192647 | 15,900849 | 32,456881 | 277,402064 |
| Tiempo transcurrido: | 0,001101 | 0,023799 | 0,124134 | 1,028148 | 6,154579 | 15,906563 | 32,433061 | 276,884794 |
| Promedio | 0,0009307 | 0,0209374 | 0,1278138 | 1,0408499 | 6,2704754 | 15,9875813 | 32,8853628 | 281,752782 |
| speedup | 0,17932739 | 0,73065424 | 0,74263655 | 0,83865358 | 0,8706275 | 0,94005549 | 0,95450692 | 0,944860878 |

3. Mejor resultado de ejecutar el algoritmo con procesos (8):

| tiempo\matriz | 10 | 100 | 200 | 400 | 600 | 800 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|---|
| Tiempo transcurrido: | 0,000747 | 0,001527 | 0,004235 | 0,012737 | 0,027108 | 0,047338 | 0,073156 | 0,292343 |
| Tiempo transcurrido: | 0,000668 | 0,001375 | 0,003952 | 0,011025 | 0,027536 | 0,042291 | 0,064258 | 0,252196 |
| Tiempo transcurrido: | 0,000935 | 0,001923 | 0,004475 | 0,013226 | 0,029795 | 0,045497 | 0,066879 | 0,254428 |
| Tiempo transcurrido: | 0,000664 | 0,002031 | 0,004514 | 0,014159 | 0,027683 | 0,044390 | 0,069274 | 0,247959 |
| Tiempo transcurrido: | 0,000651 | 0,001374 | 0,004637 | 0,013177 | 0,027724 | 0,044811 | 0,067023 | 0,254650 |
| Tiempo transcurrido: | 0,000900 | 0,001966 | 0,004685 | 0,013515 | 0,027125 | 0,045295 | 0,065203 | 0,250138 |
| Tiempo transcurrido: | 0,000945 | 0,002524 | 0,003956 | 0,013472 | 0,029877 | 0,047368 | 0,067222 | 0,255497 |
| Tiempo transcurrido: | 0,000888 | 0,001477 | 0,003869 | 0,012534 | 0,025341 | 0,046815 | 0,065078 | 0,257173 |
| Tiempo transcurrido: | 0,000685 | 0,001461 | 0,004155 | 0,011629 | 0,028065 | 0,042704 | 0,067172 | 0,252121 |
| Tiempo transcurrido: | 0,000926 | 0,001753 | 0,004434 | 0,011664 | 0,023802 | 0,046038 | 0,068439 | 0,260033 |
| Promedio | 0,0008009 | 0,0017411 | 0,0042912 | 0,0127138 | 0,0274056 | 0,0452547 | 0,0673704 | 0,2576538 |
| speedup | 0,20839056 | 8,7863994 | 22,1195004 | 68,6586622 | 199,201926 | 332,10282 | 465,921327 | 1033,236 |

4. Resultado de ejecutar el algoritmo con “la matriz transpuesta” el cual consiste en multiplicar línea por línea

| tiempo\matriz | 10 | 100 | 200 | 400 | 600 | 800 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|---|
| Tiempo transcurrido: | 0,001251 | 0,018245 | 0,139194 | 1,335608 | 7,625626 | 19,357562 | 39,900332 | 357,074447 |
| Tiempo transcurrido: | 0,001251 | 0,017623 | 0,136773 | 1,296270 | 7,431005 | 19,349726 | 39,813752 | 353,028457 |
| Tiempo transcurrido: | 0,000796 | 0,017762 | 0,135493 | 1,261941 | 7,249602 | 18,921681 | 39,296889 | 348,654123 |
| Tiempo transcurrido: | 0,001192 | 0,017662 | 0,142030 | 1,252363 | 7,271393 | 18,813237 | 39,198193 | 347,840245 |
| Tiempo transcurrido: | 0,001205 | 0,017858 | 0,135765 | 1,255282 | 7,315442 | 18,921431 | 39,434000 | 346,221380 |
| Tiempo transcurrido: | 0,000842 | 0,017961 | 0,135723 | 1,259195 | 7,278198 | 19,116688 | 39,268730 | 352,655130 |
| Tiempo transcurrido: | 0,001179 | 0,017892 | 0,135946 | 1,248427 | 7,295930 | 19,024492 | 39,138111 | 345,640279 |
| Tiempo transcurrido: | 0,000897 | 0,017879 | 0,135861 | 1,263945 | 7,300090 | 18,863366 | 39,278499 | 349,104166 |
| Tiempo transcurrido: | 0,000798 | 0,017710 | 0,138235 | 1,254051 | 7,326372 | 19,167805 | 39,526523 | 348,163095 |
| Tiempo transcurrido: | 0,001478 | 0,017722 | 0,135695 | 1,264202 | 7,349434 | 19,110710 | 39,434593 | 353,080798 |
| Promedio | 0,0010889 | 0,0178314 | 0,1370715 | 1,2691284 | 7,3443092 | 19,0646698 | 39,4289622 | 350,146212 |
| speedup | 0,15327395 | 0,85792478 | 0,69247947 | 0,68780472 | 0,7433304 | 0,78832803 | 0,7960977 | 0,7603029 |

5. Resultado de ejecutar el algoritmo con optimización por consola con O3 (para este se utilizó el programa basado en hilos)

| tiempo\matriz | 10 | 100 | 200 | 400 | 600 | 800 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|---|
| Tiempo transcurrido: | 0,000753 | 0,004525 | 0,029347 | 0,254745 | 2,622681 | 7,580328 | 17,078606 | 159,972599 |
| Tiempo transcurrido: | 0,001299 | 0,004452 | 0,028180 | 0,251122 | 2,602054 | 7,608626 | 17,213217 | 152,308526 |
| Tiempo transcurrido: | 0,001033 | 0,004405 | 0,028344 | 0,250584 | 2,638467 | 7,225962 | 16,204637 | 155,446091 |
| Tiempo transcurrido: | 0,001131 | 0,004920 | 0,029431 | 0,299703 | 2,876759 | 7,858464 | 17,515124 | 156,016079 |
| Tiempo transcurrido: | 0,000799 | 0,004411 | 0,027902 | 0,254227 | 2,598293 | 7,434792 | 17,096038 | 154,578270 |
| Tiempo transcurrido: | 0,001020 | 0,004485 | 0,027989 | 0,257431 | 2,578475 | 7,310787 | 16,529685 | 157,773493 |
| Tiempo transcurrido: | 0,001158 | 0,004414 | 0,028212 | 0,254085 | 2,633246 | 7,618838 | 16,607813 | 150,037200 |
| Tiempo transcurrido: | 0,001099 | 0,004504 | 0,028093 | 0,272297 | 2,696481 | 7,372919 | 16,211457 | 150,168715 |
| Tiempo transcurrido: | 0,000632 | 0,004333 | 0,027735 | 0,284278 | 2,733056 | 7,405200 | 16,231725 | 148,321383 |
| Tiempo transcurrido: | 0,001008 | 0,004641 | 0,028271 | 0,261401 | 2,662098 | 7,335154 | 16,315410 | 149,599843 |
| Promedio | 0,0009932 | 0,004509 | 0,0283504 | 0,2639873 | 2,664161 | 7,475107 | 16,7003712 | 153,42222 |
| speedup | 0,16804269 | 3,39277002 | 3,34807269 | 3,30664581 | 2,04914354 | 2,01056834 | 1,8795574 | 1,73519312 |

6. Resultado de utilizar OpenMP:

| tiempo\matriz | 10 | 100 | 200 | 400 | 600 | 800 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|---|
| Tiempo transcurrido: | 0,027514 | 0,023116 | 0,122765 | 1,078847 | 6,286944 | 16,633064 | 34,572026 | 308,027555 |
| Tiempo transcurrido: | 0,005270 | 0,021238 | 0,112159 | 1,074101 | 6,342239 | 16,685422 | 35,243672 | 308,027776 |
| Tiempo transcurrido: | 0,000194 | 0,007186 | 0,110572 | 1,040642 | 6,354928 | 16,883029 | 35,139851 | 310,318011 |
| Tiempo transcurrido: | 0,000197 | 0,021669 | 0,114026 | 1,047266 | 6,384695 | 17,219871 | 35,072115 | 309,670178 |
| Tiempo transcurrido: | 0,000224 | 0,008042 | 0,100309 | 1,046844 | 6,340211 | 16,950950 | 34,856345 | 296,573501 |
| Tiempo transcurrido: | 0,000342 | 0,020065 | 0,106096 | 1,054105 | 6,377521 | 16,899067 | 35,305026 | 301,413728 |
| Tiempo transcurrido: | 0,000179 | 0,010083 | 0,101870 | 1,038397 | 6,329824 | 16,891395 | 34,893271 | 291,164347 |
| Tiempo transcurrido: | 0,000335 | 0,010755 | 0,107781 | 1,043414 | 6,306598 | 16,940138 | 34,898054 | 309,241551 |
| Tiempo transcurrido: | 0,000196 | 0,011634 | 0,105708 | 1,042147 | 6,326363 | 17,122956 | 35,256593 | 300,955258 |
| Tiempo transcurrido: | 0,000190 | 0,014207 | 0,115096 | 1,040967 | 6,348214 | 16,783638 | 34,939295 | 292,250420 |
| Promedio | 0,0034641 | 0,0147995 | 0,1096382 | 1,050673 | 6,3397537 | 16,900953 | 35,0176248 | 302,7642325 |
| SpeedUP | 0,0481799 | 1,03368357 | 0,86574935 | 0,83081273 | 0,86111363 | 0,88925243 | 0,89638593 | 0,879288741 |

**Graficas comparativas**

Para ver un poco mejor y poder ver las diferencias entre las implementaciones, las vemos comparadas en gráficos para poder diferenciarlo de otra manera, los valores del eje X representan el tamaño de la matriz siendo 8 el máximo de 2000 valores

Primero hacemos una comparativa del speedup de los hilos para ver cual tuvo mejor calificación entre los mejores hilos y escoger uno para las otras graficas comparativas:

Hilos: 2

Procesos: 8

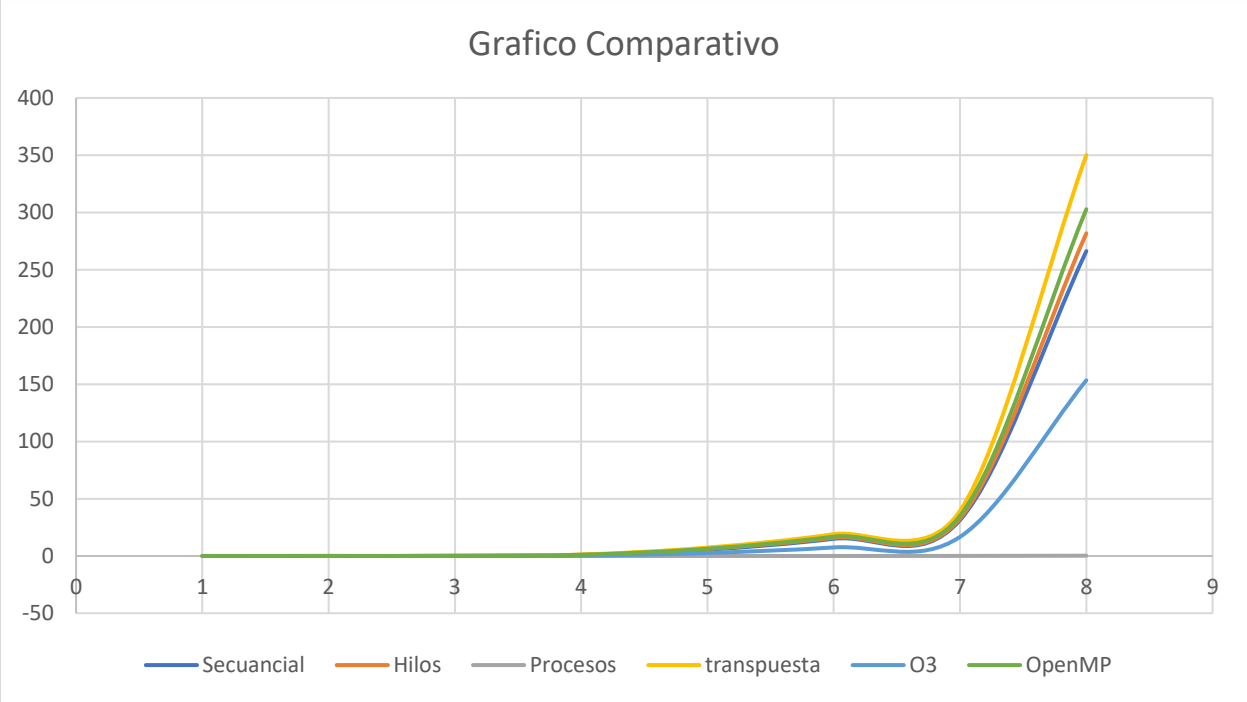
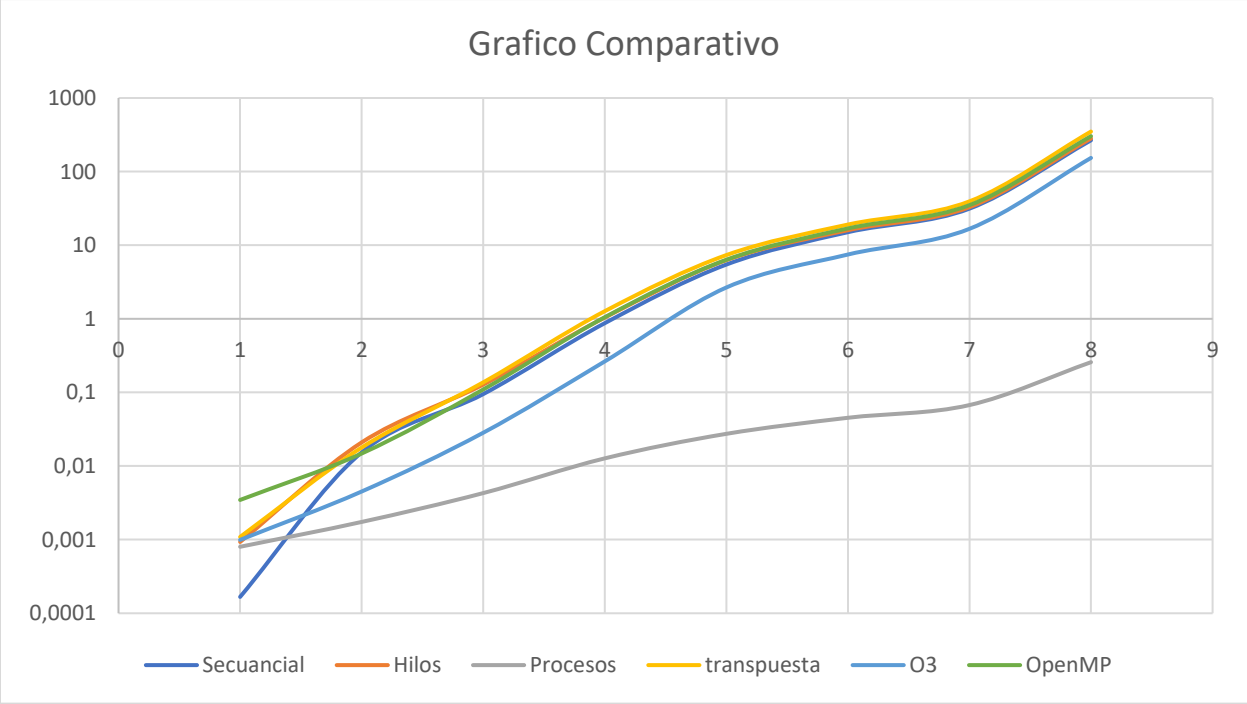


**Grafico en Logaritmo base 10**

Hilos: 2

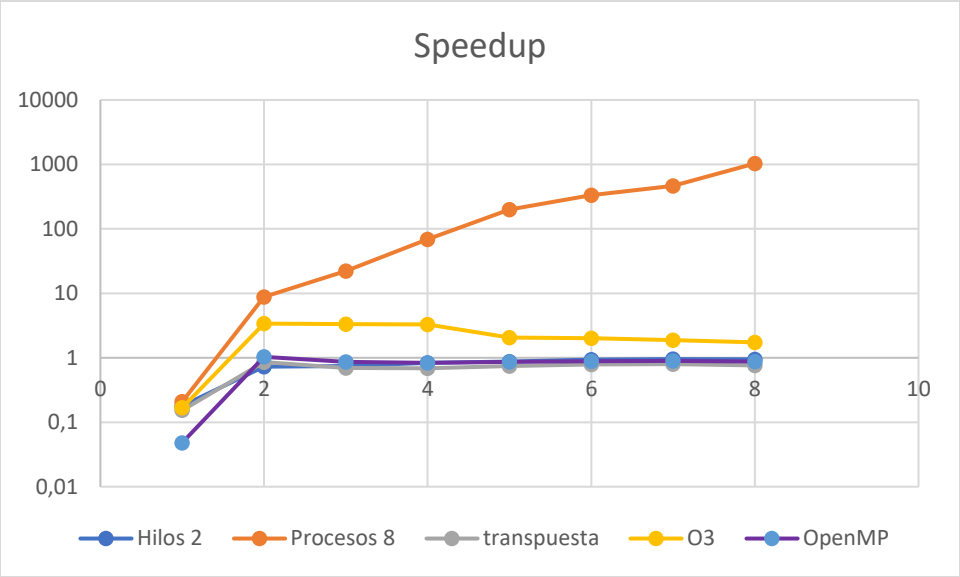
Procesos: 8



**Mejora de velocidades con respecto a la secuencial**

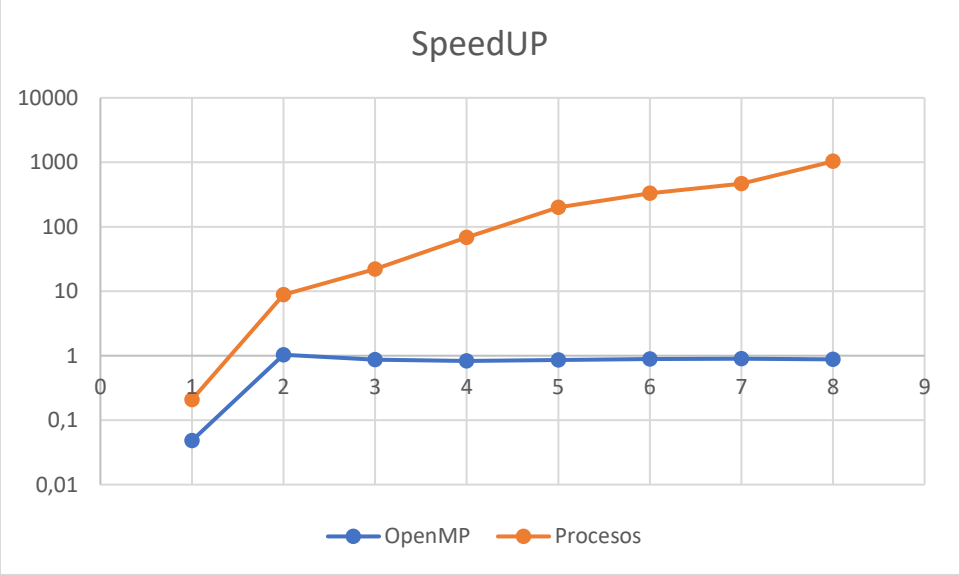
Hilos: 2

Procesos: 8




**Mejora de velocidades con respecto a el método de procesos (el cual es el mejor en este caso)**

Procesos: 8



## Conclusiones

1. Este trabajo podría decir que el uso de la api si incrementa o ayuda a la optimización de un código paralelizable, pero que a lo mejor en este ambiente de trabajo no representa una gran ventaja.

2. Podemos ver que el sistema no respondía muy bien utilizando hilos como parte de su desarrollo ya que la matriz transpuesta al estar hecha en base al código de los hilos, también mostro deficiencia comparado con los otros, como por ejemplo con procesos.

3. La diferencia con las ejecuciones paralelas podría deberse a la ejecución de varios programas a la vez en este entorno de trabajo.

## Bibliografía


- HPC for Research (Sitio web de recursos)
- Introduction to Matrix Multiplication (Artículo)
- CUDA Programming (Sitio web de recursos)
- Understanding Speedup in Parallel Computing (Artículo)
- "Introduction to Multithreading in C" (artículo)