Pontificia Universidad Javeriana

Taller 2: Rendimiento

Docente:
John Corredor Franco

Integrantes:
Karol Dayan Torres Vides
Andres Eduardo Meneses Rincon

FACULTAD DE INGENIERÍA
Bogota D.C
14 de noviembre 2025

Introducción

En este taller se busca realizar un análisis de las diferentes implementaciones de hilos que podemos tener en el lenguaje de programación c, además de comparar 2 diferentes tipos de algoritmos de multiplicación de matrices que utilizan la misma librería (OPENMP).
Se busca en primer lugar comparar las diferentes librerías de implementación de hilos, las cuales son:OpenMP, Fork y POSIX. Seguido de estas pruebas se analiza que tanto beneficia al rendimiento el cambio de algoritmo y por último comparamos cómo se comportan los hilos dependiendo de la arquitectura del computador.

Enlace repositorio personal: https://github.com/TorresVides/TallerEvaluacionRendimiento.git

Pruebas de las librerías

Estas pruebas se realizaron utilizando un dispositivo con la siguiente arquitectura:

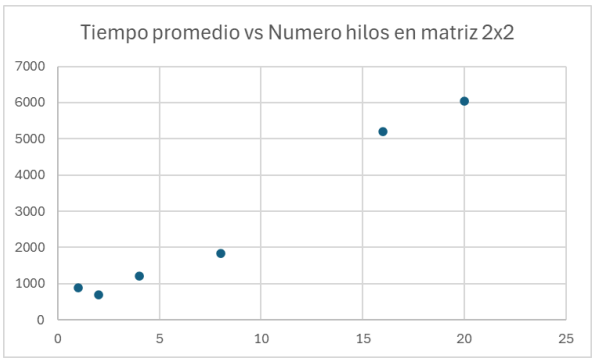
```
andres@DESKTOP-1FVM1KL:~$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  4
  On-line CPU(s) list:   0-3
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz
    CPU family:          6
    Model:               78
    Thread(s) per core:  2
    Core(s) per socket:  2
    Socket(s):           1
    Stepping:            3
    BogoMIPS:            4991.99
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse s
                         se2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon rep_good nopl xtopology cpuid
                          tsc_known_freq pni pclmulqdq ssse3 fma cx16 pdcm pcid sse4_1 sse4_2 movbe popcnt aes xsave av
                         x f16c rdrand hypervisor lahf_lm abm 3dnowprefetch pti ssbd ibrs ibpb stibp fsgsbase bmi1 hle
                         avx2 smep bmi2 erms invpcid rtm rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 xsaves md_c
                         lear flush_l1d arch_capabilities
```

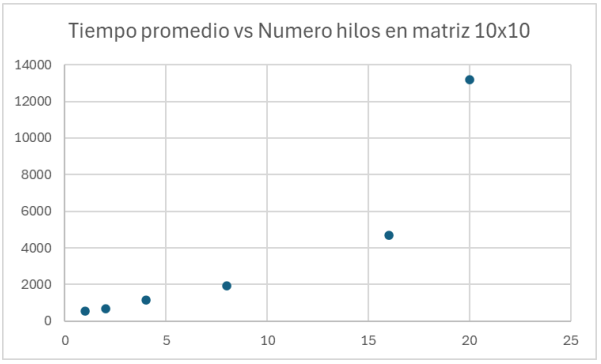
Figura 1: Arquitectura dispositivo 1

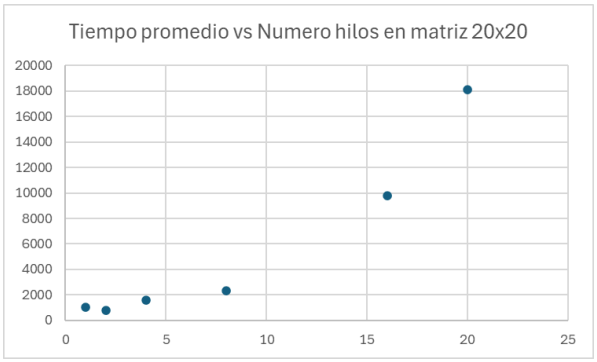
Este computador posee 2 núcleos físicos, cada uno con la capacidad de manejar 2 hilos, osea un total de 4 núcleos (Hilos manejables) [Figura 1], las pruebas realizadas en este dispositivo consiste en ejecutar cada uno de los programas cambiando las siguientes variables cada 30 repeticiones: Número de hilos y tamaño de la matriz.

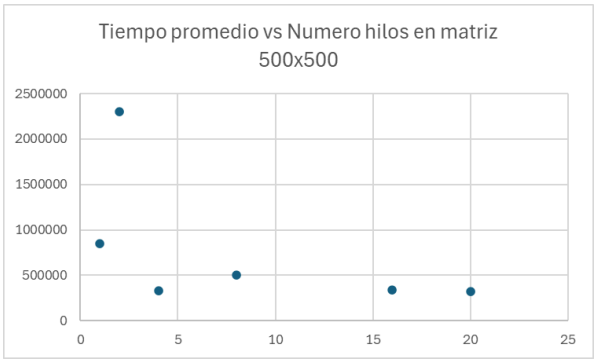
- **Gráficas del programa con Fork**

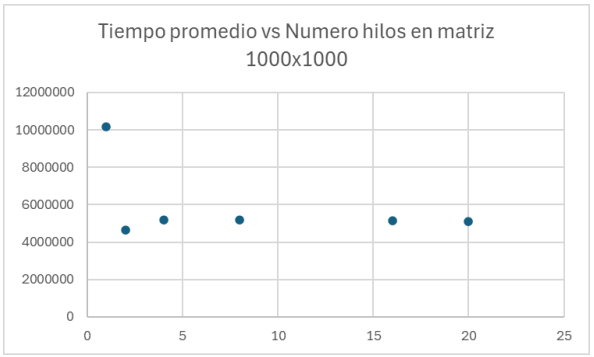
Las primeras pruebas realizadas fueron en el programa que utilizaba fork para las creación de hilos, en este obtuvimos los siguientes resultado:











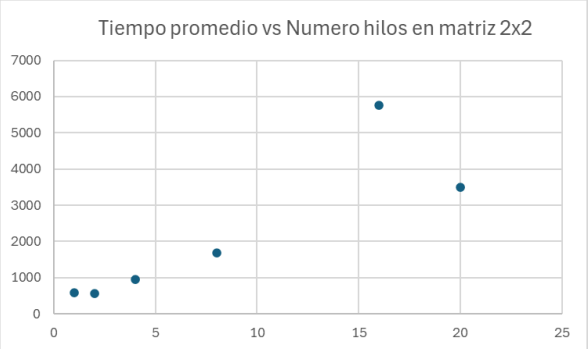
Aquí podemos realizar varias afirmaciones teniendo en cuenta los resultados:

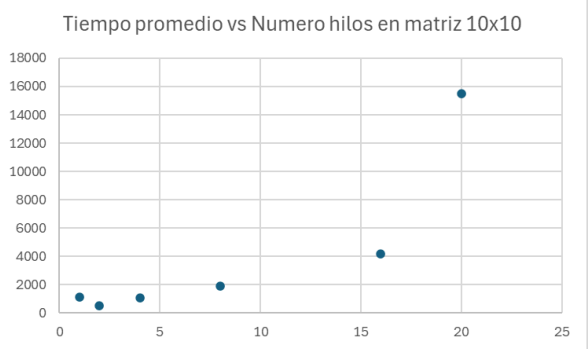
- En las matrices pequeñas se observa cómo a mayor cantidad de hilos mas tiempo promedio de ejecución, obteniendo el menor tiempo de ejecución con 1 o 2 hilos
- En las matrices de tamaños grandes siempre obtenemos un rendimiento parecido con todas las cantidades de hilos a excepción de con 1 o 2 hilos, y tendremos mejor resultados con 4 hilos
- Podemos evidenciar que las gráficas de las matrices pequeñas tienen forma de exponencial creciente y las de tamaños superiores tienen forma de exponencial decreciente, esto demostrando que en tamaños pequeños tendremos un overhead que supera el speedup lo que se demuestra en que a maño cantidad de recursos a manejar el rendimiento no mejora, en cambio en matrices de mayor tamaño el speedup supera con creces el overhead ya que aunque manejar más recursos (hilos) sea costoso, el beneficio de separar estas grandes tareas mejora el rendimiento

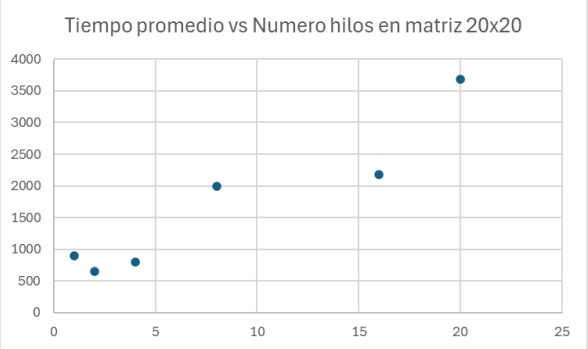
- **Gráficas del programa con POSIX**

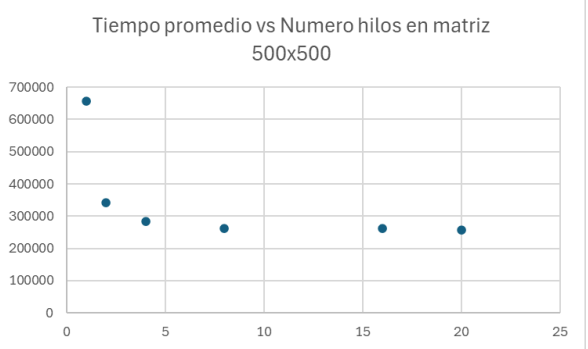
Seguido de las pruebas en el programa de Fork, continuamos con las de POSIX, las pruebas fueron las mismas y los resultados son muy parecidos, en cuanto a algunas diferencias a destacar podemos ver que:

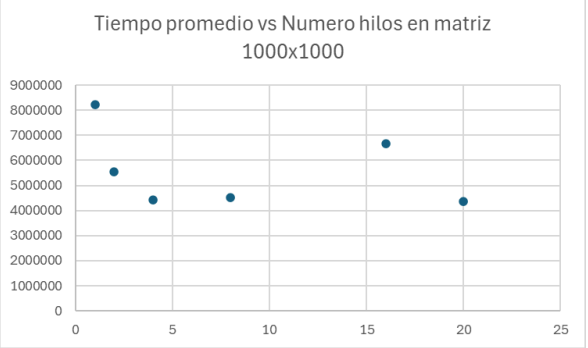
- Hay una mayor estabilidad en el tiempo promedio de ejecución con pocos hilos en las matrices 2x2 y 10x10, esto se ve al comparar la mejor ejecución con las demas en donde vemos que sus diferencias son muy pequeñas
- El rendimiento comparado con Fork mejora un poco en los peores casos, esto demostrando que la gestión de recursos de esta librería es mejor disminuyendo asi el overhead
- El comportamiento de las gráficas es igual al de Fork, obteniendo una exponencial creciente en las matrices pequeñas y una exponencial decreciente en las matrices grandes







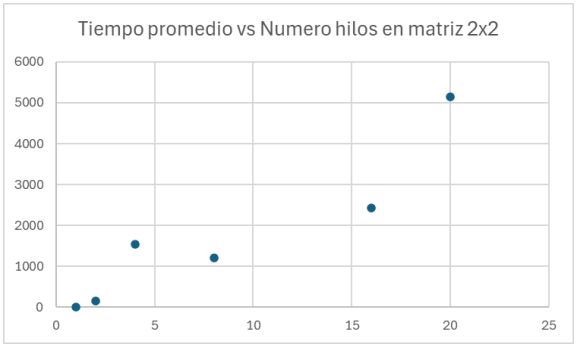


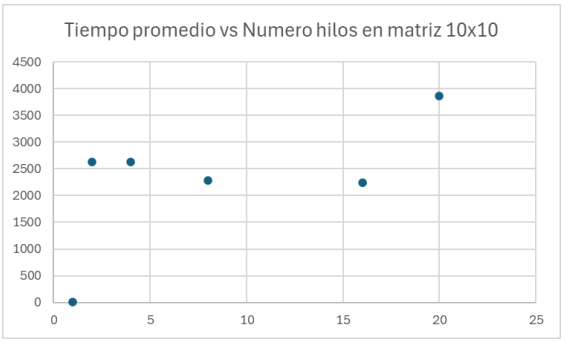


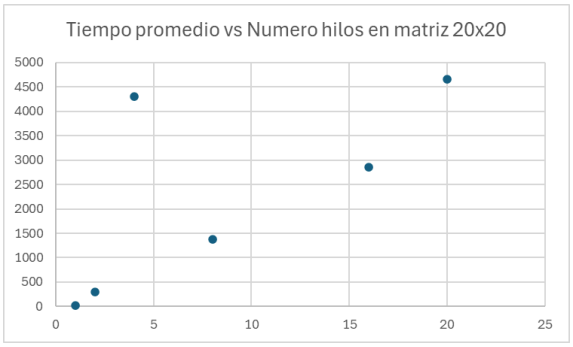
- **Gráficas del programa con OPENMP**

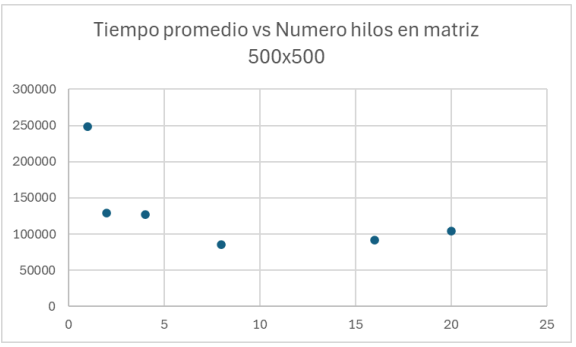
Por último se realizaron pruebas utilizando la librería OPENMP, estos resultados siguieron el mismo comportamiento que los otros 2 programas, pero tuvieron los siguientes cambios:

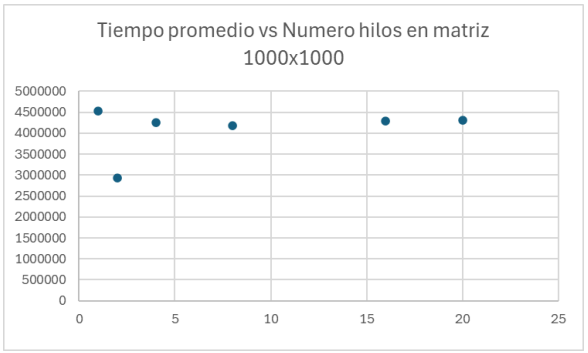
- El rendimiento en todos los casos es mejor que el de las pruebas utilizando las otras 2 librerías, esto se evidencia en cómo los rangos de eje y (Eje que mide el tiempo promedio) son muchísimos menores a las demás graficas
- En matrices pequeñas se puede evidenciar la eficiencia de la librería mostrándonos tiempos de ejecución demasiado pequeños, y en matrices grandes podemos evidenciar fácilmente la cantidad óptima de hilos a utilizar











- **Conclusiones**

Al terminar estas pruebas podemos concluir que la librería que mejor rendimiento nos proporciona es OPENMP teniendo un perfecto manejo de los recursos, haciendo que el overhead disminuya muchísimo en los casos de las matrices pequeñas, además mejorando significativamente el rendimiento en en las tareas más difíciles, como lo fueron las matrices de tamaños grandes.

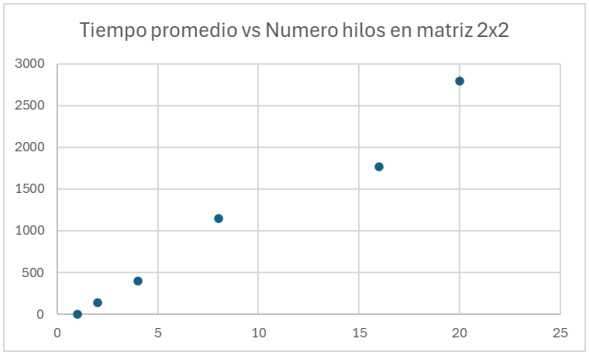
Según estas pruebas clasificamos las librerías de la siguiente manera: OPENMP la mejor opción a utilizar, POSIX la segunda mejor librería en cuanto a rendimiento y buen manejo del overhead al asignar un poco mas de hilos a tareas simples, Fork la peor de las 3, tiene el peor rendimiento y no tiene nada que la destaque de las otras 2.

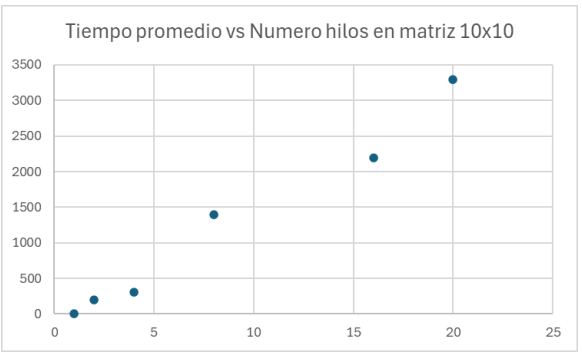
Habiendo terminado la comparación de las diferentes librerías, el siguiente paso de estudio fue comparar la mejor librería al utilizar diferentes algoritmos, las pruebas anteriores se realizaron con el algoritmo clásico de multiplicación de matrices y la siguiente se realizó con el algoritmo de multiplicación de matrices por filas, la cual teóricamente tiene una menor complejidad comparado con el algoritmo anterior

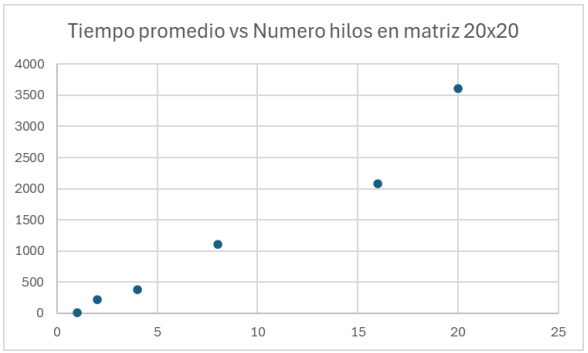
- **Gráfica del programa con OPENMP y función de multiplicación por líneas**

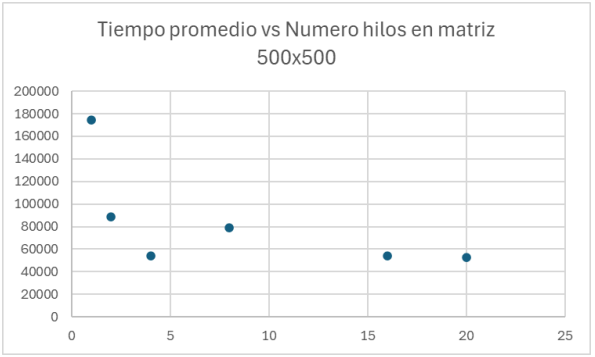
Al realizar esta pruebas identificamos las siguientes notables diferencias con su contraparte:

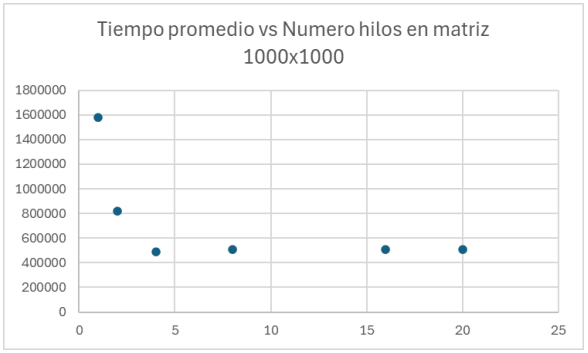
- Aumento exponencial del rendimiento, esto lo evidenciamos realizando el mismo análisis que en OPENMP, al comparar los rangos del eje y evidenciamos que disminuyen en gran medida, siendo la mejor mejora de un 64% (Cambio de 5M a 1.8M)
- Un comportamiento lineal negativo en matrices pequeñas, demostrando así que el overhead generado al utilizar una cantidad innecesario de hilos es menor
- Mejor estabilidad en los resultados, esto lo podemos evidenciar al comparar todas las gráficas, ya que vemos que esta es la que más concuerda con la teoría, esto debido a que en las matrices pequeñas la mejor cantidad de hilos es 1 (este comportamiento es compartido con las otras pruebas de la librería OPENMP) y en matrices grandes la cantidad ideal siendo 4 hilos (en algunas de las otras gráficas esto también sucede pero nunca sucede en ambas, ósea en 500 y 1000 al mismo tiempo)





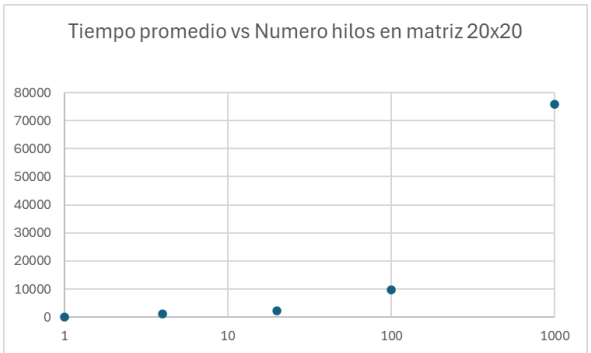


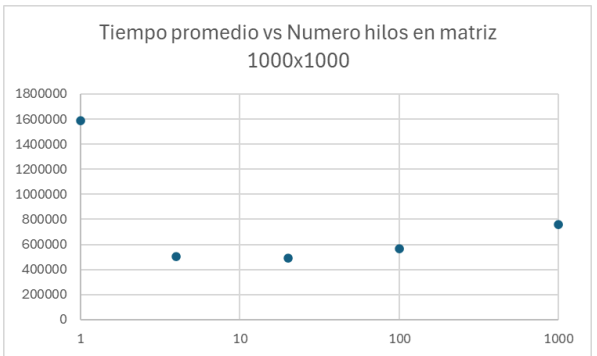




- **Gráfica del programa con OPENMP y función de multiplicación por líneas (Con alta cantidad de hilos)**

Al terminar de realizar estas pruebas nos dimos cuenta que en ninguna gráfica se puede observar claramente la curva de rendimiento, por esto mismo se realizaron dos pruebas extra utilizando la mejor combinación (OPENMP y líneas) y las siguientes configuraciones: rango de hilos [1,4,20,100,1000], rango de tamaños ['20','1000'].





En estas dos gráficas se puede evidenciar perfectamente el comportamiento que aprendimos en clase, en donde al tener una tarea sencilla y administrarle demasiados recursos, el rendimiento no va a mejorar, y al tener una tarea complicada si obtendremos una mejora en el rendimiento al darle más recursos pero, como dice la teoría no siempre a mayor cantidad de recursos mayor rendimiento, podemos observar como después del uso de 4 hilos, la mejora comparada con 20 hilos es casi imperceptible y luego al ver la de 100 hilos podemos ver que el rendimiento empeoro.

- **Pruebas de comparación entre dispositivos:**

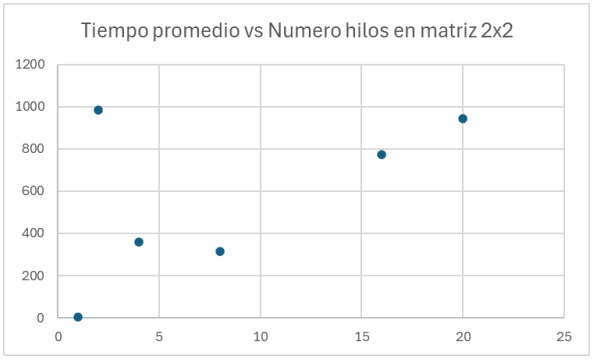
Una vez terminamos las pruebas de los diferentes algoritmos, realizaremos una evaluación del comportamiento del mejor de los algoritmos en diferentes dispositivos, teniendo en cuenta cambios como cantidad de núcleos, cantidad de hilos manejados por núcleo y velocidad del procesador, el caso base con el cual realizaremos algunas comparaciones es el que utilizamos para las anteriores pruebas[Figura 1], a continuación se presentan los resultados de los demás dispositivos

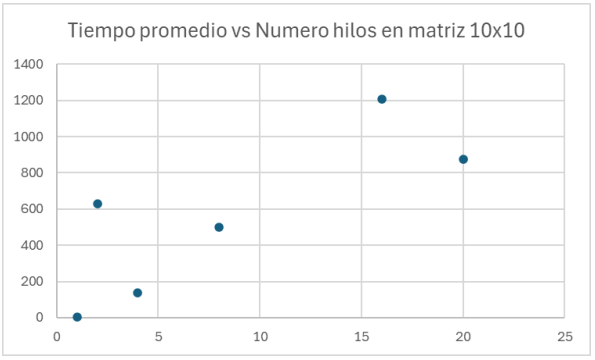
- **PC 1:**

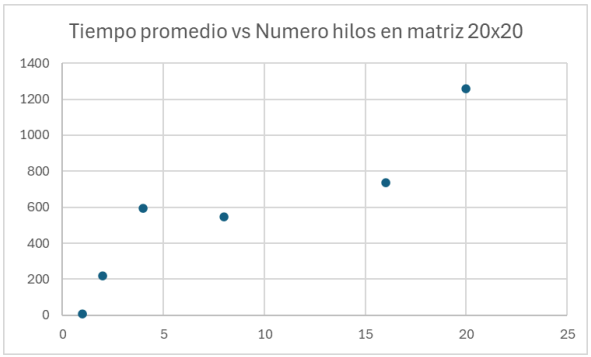
Información del dispositivo:
Núcleos: 1
Hilos por núcleo: 1
Cantidad ideal de hilos: 2
velocidad (Segun informacion en linea): 2.45 GHz - 3.5 GHz

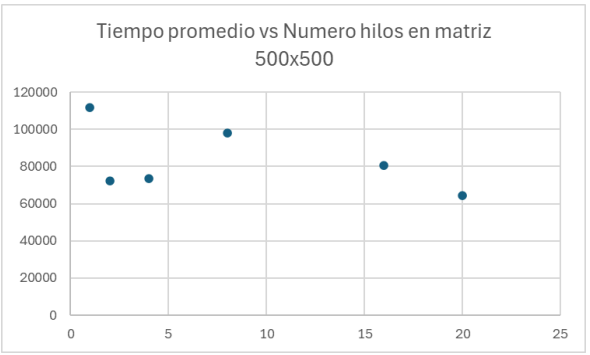
```
@ShalkYT →/workspaces/Sistemas-Operativos/Tallerrendimiento (main) $ lscpu
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          48 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   2
  On-line CPU(s) list:    0,1
Vendor ID:                AuthenticAMD
  Model name:             AMD EPYC 7763 64-Core Processor
    CPU family:           25
    Model:                1
    Thread(s) per core:   2
    Core(s) per socket:   1
    Socket(s):            1
    Stepping:             1
    BogoMIPS:             4890.86
    Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht s
                          yscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl tsc_reliable nonstop_tsc cpuid
                          extd_apicid aperfmperf tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 movbe popcnt aes
                           xsave avx f16c rdrand hypervisor lahf_lm cmp_legacy svm cr8_legacy abm sse4a misalignsse 3dnowprefet
                          h osvw topoext vmmcall fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb sha_
                          i xsaveopt xsavec xgetbv1 xsaves user_shstk clzero xsaveerptr rdpru arat npt nrip_save tsc_scale vmcb_
                          clean flushbyasid decodeassists pausefilter pfthreshold v_vmsave_vmload umip vaes vpclmulqdq rdpid fsr
                          m
```

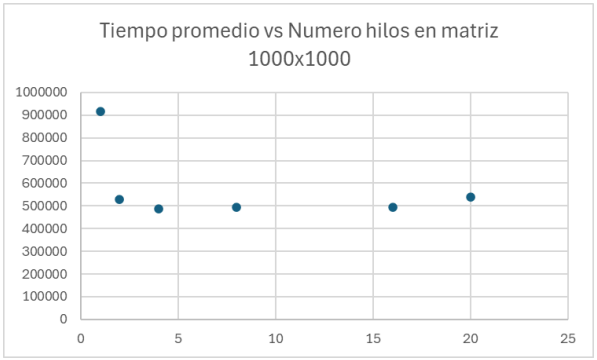
Resultados:











En este caso podemos notar una diferencia en las gráficas con menor tamaño, ya que no obtenemos el mismo comportamiento de una recta lineal con pendiente positiva, en algunas casos conserva este comportamiento, pero algunos de los ejemplos que difieren son: 2 hilos en 2x2 y 10x10, 8 hilos en 20x20. Además en la de 500x500 también observamos algunos comportamientos diferentes, como por ejemplo el empeorar rendimiento al usar 8 hilos, todos estos cambios se deben a algunas interferencias con otros procesos del sistema o a causa de la diferente arquitectura.
Un aspecto importante a resaltar es que esta prueba se realizó con un workspace de github, los cuales funcionan con virtualización de un server, con esto nos referimos a que todos los workspaces de github comparten un único dispositivo y esto en algunos casos puede afectar el rendimiento, aún son casos muy aislados, se considera por ende que las diferencias con las pruebas anteriores se deben únicamente a la diferencia de arquitectura del procesador.

- **PC2:**

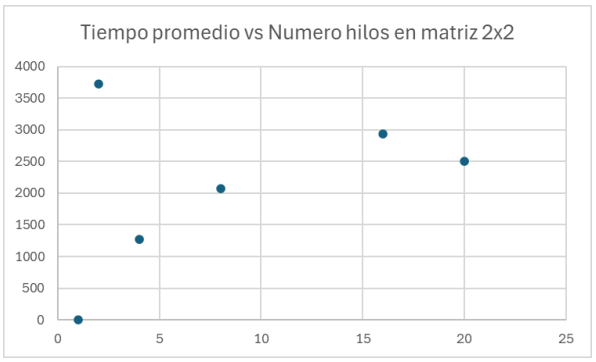
Información del dispositivo:
Núcleos: 2
Hilos por núcleo: 1

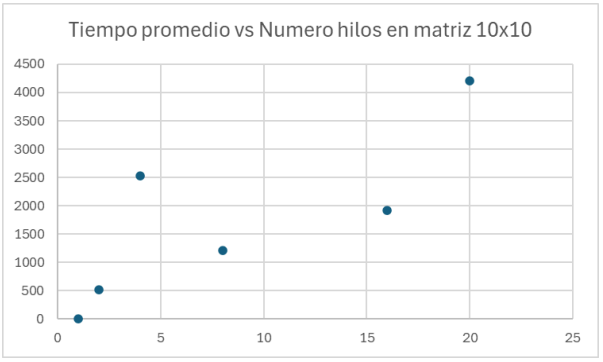
Cantidad ideal de hilos: 2

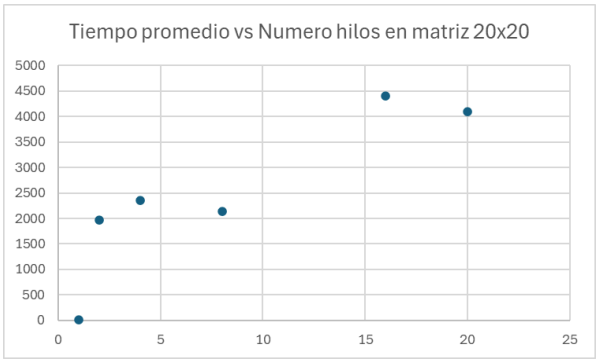
velocidad : 2.40 GHz

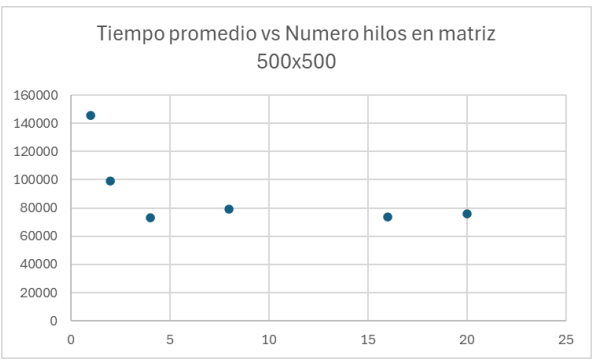
```
karol@mint-vm:/media/sf_linux_compartida/TallerRendimiento$ lscpu
Arquitectura:                         x86_64
  modo(s) de operación de las CPUs:   32-bit, 64-bit
  Address sizes:                      39 bits physical, 48 bits virtual
  Orden de los bytes:                 Little Endian
CPU(s):                               2
  Lista de la(s) CPU(s) en línea:     0,1
ID de fabricante:                     GenuineIntel
  Nombre del modelo:                  Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz
    Familia de CPU:                   6
    Modelo:                           61
    Hilo(s) de procesamiento por núcleo: 1
    Núcleo(s) por «socket»:           2
    «Socket(s)»                       1
    Revisión:                         4
    BogoMIPS:                         4788,90
    Indicadores:                      fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscal
                                      l nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fm
                                      a cx16 pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch pti f
                                      sgsbase bmi1 avx2 bmi2 invpcid rdseed adx arat md_clear flush_l1d arch_capabilities
Virtualization features:
```

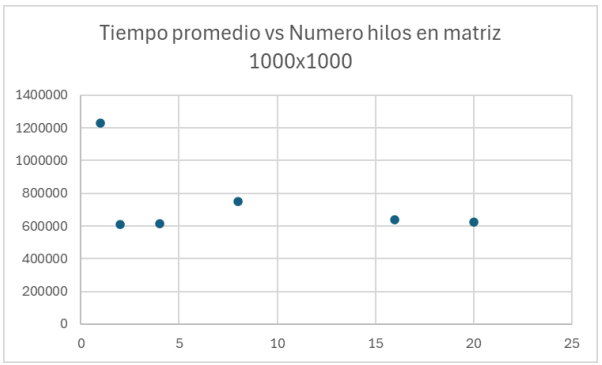
Resultados:











En este caso las pruebas se realizaron en un entorno virtual al igual que los casos originales, esto refleja la similitud en los resultados de alta exigencia, pero en los casos sencillos hay una gran diferencia, esto se debe a que a pesar de tener el mismo tipo de entorno y condiciones, las arquitecturas son diferentes, lo que refleja el cambio en las

matrices de tamaños pequeños, el cambio fundamental a tener en cuenta son las velocidades de los procesadores lo que puede afectar un poco la coordinación de los hilos y el manejo de recursos, aumentando en algunos casos el overhead, como por ejemplo en los casos de 2 hilos de las 3 matrices más pequeñas, ya que comparadas con la prueba original estas presentan grandes cambios en cuanto a tiempo de ejecución.

- **PC3:**

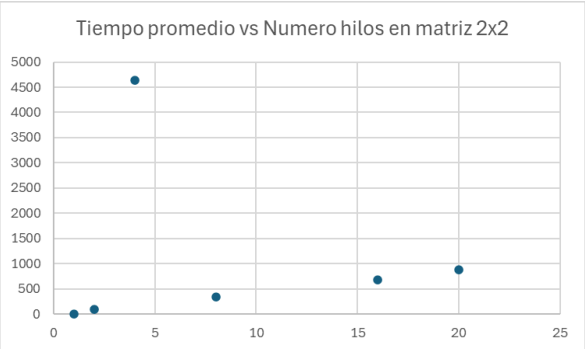
Información del dispositivo:
Núcleos: 4

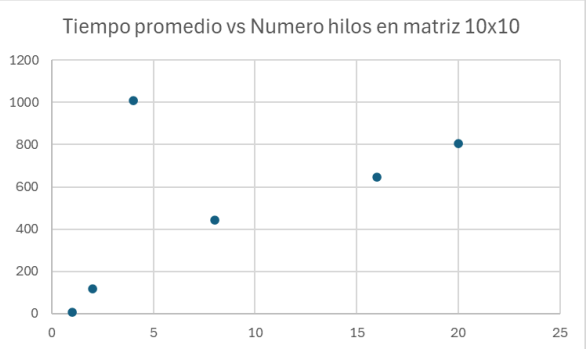
Hilos por núcleo: 1

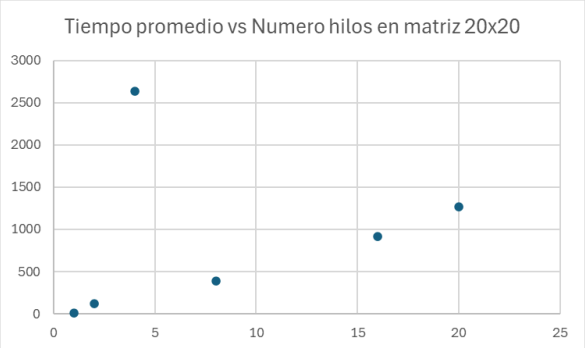
Cantidad ideal de hilos: 4

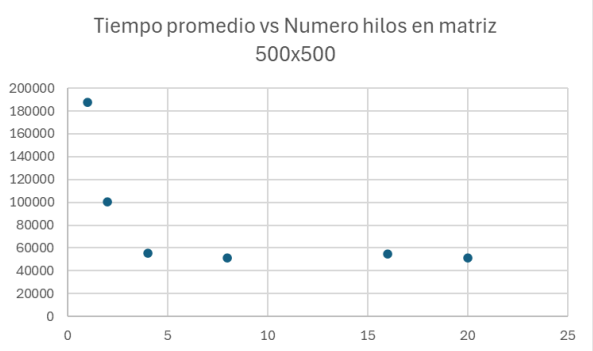
velocidad (Segun informacion en linea): 2.60 GHz

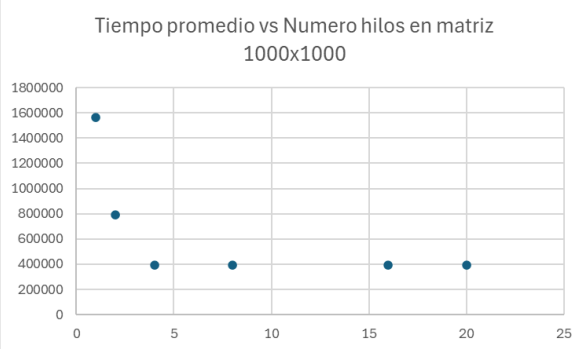
Resultados:











Estos resultados concuerdan en mayor medida con el caso de estudio original, pero al igual que con las pruebas del PC2 hay un aumento de tiempo de ejecución en los que se suponen deben ser sus casos ideales en las matrices de menor tamaño, esto puede ser causado por un uso innecesario de recursos en estos casos, el overhead supera por mucho al speedup causando que no sea útil utilizar la mejor configuración en tareas que no lo requieren, en este caso es facil tambien evidenciar la curva de rendimiento en las matrices grandes notando cuando el speedup menos el overhead generan el caso de mayor rendimiento en la configuración ideal del procesador (4 hilos)

- **Conclusiones**

A través de todas estas pruebas podemos entender que hay muchísimas cosas a tener en cuanto al momento de utilizar paralelismo para un programa, las más importantes que pudimos identificar son:

- ¿El programa necesita paralelismo? Podemos observar que en los casos en los cuales no se necesitaba muchísimo trabajo (matrices pequeñas) el caso con mejor rendimiento siempre era utilizar 1 solo hilo, esto demostrando que debido al overhead no siempre utilizar paralelismo es la mejor opción.

- ¿Qué librería debería utilizar? A pesar de haber varias librerías para implementar paralelismo y concurrencia, observamos como hay algunas que manejan de mejor manera la gestión de recursos (overhead), en nuestras pruebas, como ya habíamos concluido calificamos las librerías usadas de la siguiente manera: OpenMP la mejor opción para un overhead mínimo, POSIX la segunda mejor opción con un buen manejo de overhead que no supera a OPENMP pero destaca de manera espectacular al momento de utilizar una mayor cantidad de hilos y por último Fork el cual por su enfoque basado en procesos, nos dio los peores resultados en cuanto a manejo de recursos y disminución del overhead.
- ¿Cómo puedo mejorar el rendimiento de mi programa? El paralelismo es una herramienta muy útil al momento de programar orientando el desarrollo a un mejor rendimiento, pero algo a tener en cuenta es que esta no es la solución a todos los problemas, en nuestro proceso encontramos la mejor librería para nuestro programa, pero los tiempos de ejecución siguen siendo algo elevados, al momento de evaluar el algoritmo utilizado y reemplazarlo el rendimiento aumentó de forma increíble, demostrando asi que al momento de programar orientando el desarrollo al rendimiento es muy necesario tener en cuenta los métodos con los cuales desarrollamos las tareas dentro del programa.