

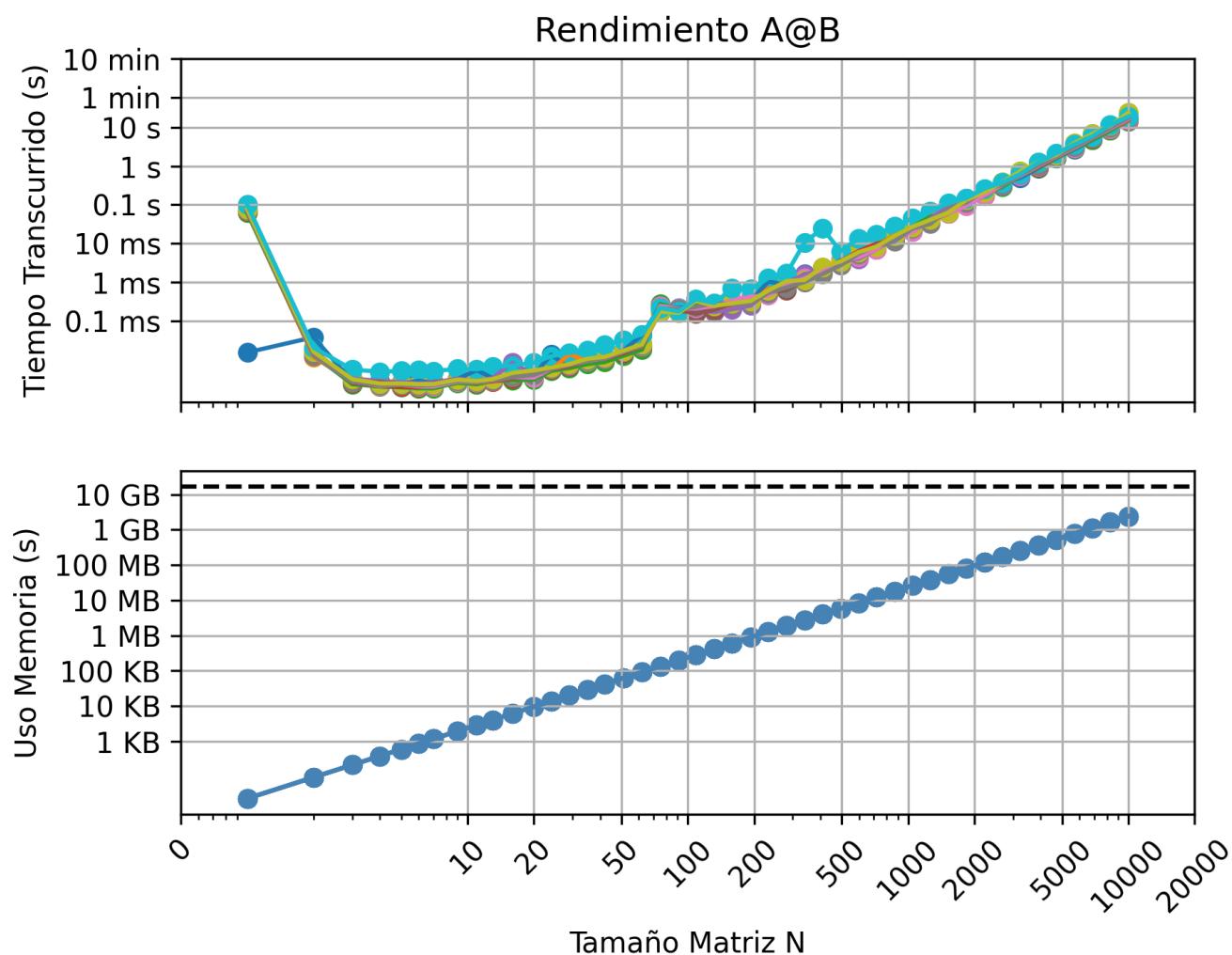
Mi computador principal

- Marca/modelo: Asus VivoBook S15 X530UF
- Tipo: Notebook
- Año adquisición: 2019
- Procesador:
 - Marca/Modelo: Intel Core i7-8550U
 - Velocidad Base: 1.80 GHz
 - Velocidad Máxima: 4.00 GHz
 - Numero de núcleos: 4
 - Numero de hilos (Procesadores lógicos): 8
 - Arquitectura: x86_64
 - Set de instrucciones: Intel® SSE4.1, Intel® SSE4.2, Intel® AVX2
- Tamaño de las cachés del procesador
 - L1: 256KB
 - L1: 1.0MB
 - L2: 8.0MB
- Memoria
 - Total: 16 GB
 - Tipo memoria: DDR4
 - Velocidad 2400 MHz
 - Numero de (SO)DIMM: 2
- Tarjeta Gráfica
 - Marca / Modelo: Nvidia GeForce MX130
 - Memoria dedicada: 2048 MB GDDR5
 - Resolución: 1920 x 1080
- Disco 1:
 - Marca: SanDisk
 - Tipo: SSD
 - Tamaño: 238GB (en teoría tiene 256GB)
 - Particiones: 3 (OS, SYSTEM, RECOVERY)
 - Sistema de archivos: NTFS (además tiene FAT32 en la partición SYSTEM)
- Disco 2:

- Marca: Toshiba
- Tipo: HDD
- Tamaño: 931GB (en teoría tiene 1TB)
- Particiones: 1
- Sistema de archivos: NTFS
- Dirección MAC de la tarjeta wifi: 20:16:B9:44:BD:A3
- Dirección IP (Interna, del router): 192.168.100.2
- Dirección IP (Externa, del ISP): 186.67.234.139
- Proveedor internet: Entel Chile S.A. (fibra óptica)

Desempeño MATMUL

En primer lugar cabe señalar que rendimiento.txt corresponde a un archivo que por cada línea es una lista de listas, en donde cada una de estas es considerada una nueva corrida y está ordenada de la siguiente manera : [[Ns],[dts],[mems]]



Preguntas

- **¿Cómo difiere del gráfico del profesor/ayudante?**

En primer lugar se puede notar como tengo una partida similar en tiempo a las del profesor/ayudante, luego las 9 siguientes tienen un tiempo de partida mayor. Luego es interesante señalar como en matrices con tamaños entre 50 y 60 de evidencia un salto en el tiempo de memoria en el cual en mi notebook es menor que el del profesor/ayudante. Por último el comportamiento final es prácticamente igual, terminando por cada partida un poco inferior a un minuto al igual que el profesor/ayudante. Otra diferencia significativa podría ser debido a la cantidad de N en el eje x que el profesor/ayudante seleccionó vs los que yo establecí (45).

En estricto rigor se puede evidenciar como el pc del profesor/ayudante tiene un mejor rendimiento en un inicio y en el final, mientras que mi PC tiene mejor rendimiento en los valores de N intermedios.

Lo anteriormente señalado ocurre debido a las distintas características de caché, RAM, y disco entre el pc del profesor/ayudante y el mio. Pero en general, los gráficos son muy similares.

- **¿A qué se pueden deber las diferencias en cada corrida?**

Esto ocurre, ya que, python para el uso de memoria siempre intenta utilizar la menor cantidad de memoria posible. Además, la memoria funciona por bloques, entonces al necesitar más memoria irá en búsqueda del siguiente módulo de memoria y es por esto que se producen los "saltos" en el tiempo que se observan en el gráfico, en estos cambios de memoria manda la jerarquía de memoria es decir, primero caché, luego RAM y por último el disco. Este proceso de "cambio de módulo de memoria" es muy variable y es por eso que se puede evidenciar diferencias en los tiempos de ejecución en cada corrida.

Otro factor que puede influir es el sistema operativo, el cual prioriza distintos procesos (hace una cosa a la vez con distintas prioridades, por lo que hacer diferentes acciones el el pc (como navegar en internet) puede influir directamente), los cuales pueden influir directamente en el tiempo de ejecución, estas pueden ser acciones que uno hace directamente, como también acciones que hace el sistema operativo reliza por detrás sin que uno se de cuenta.

Por último, otros factores que pueden influir son que el computador este utilizando una gran cantidad de recursos que relentice el proceso (por ejemplo que la temperatura del PC aumente).

- **El gráfico de uso de memoria es lineal con el tamaño de matriz, pero el de tiempo transcurrido no lo es ¿porqué puede ser?**

Esto ocurre ya que, en una multiplicación de matrices existe un número predeterminado de "acciones" por lo que la memoria utilizada en la operación siempre será la misma (es por eso que al graficar las 10 corridas sigue viendose una única gráfica). Es decir, la memoria utilizada esta establecida por la operación y el porte de las matrices y no influira si esta se desarrolla antes o después a diferencia de lo que ocurre con el tiempo.

Con respecto a la linealidad del gráfico, la memoria utilizada es lineal ya que si una matriz es el doble de grande simplemente utilizará el doble de memoria sin importar otros factores (lo mismo ocurre con la operación MATMUL), pero en el caso del tiempo, a medida que se multiplica una matriz con valores de N más grande, los recursos necesarios van creciendo exponencialmente (es decir una matriz el doble de grande no demorara necesariamente el doble del tiempo sino que más), esto debido a la complejidad de la operación MATMUL que va aumentando exponencialmente a medida que aumenta los valores de N.

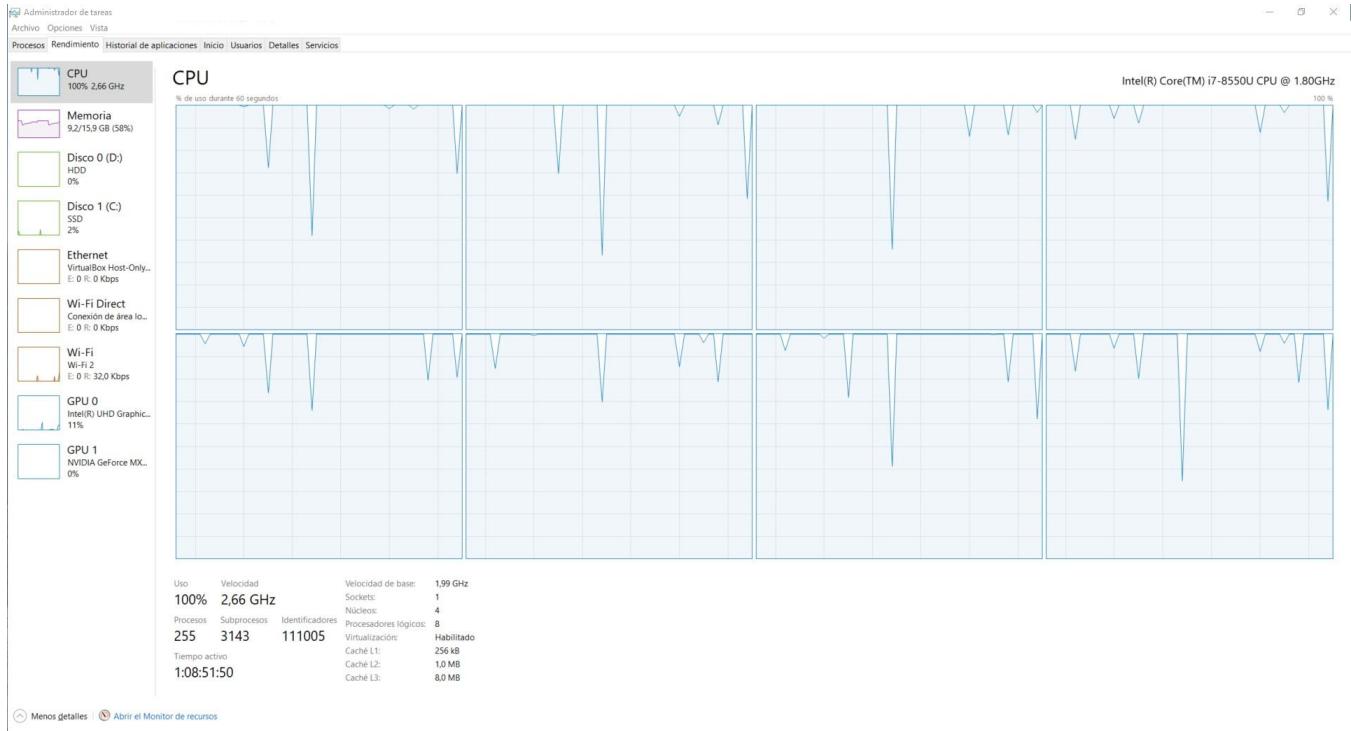
- ¿Qué versión de python está usando?

Python Version: 3.9.6

- ¿Qué versión de numpy está usando?

NumPy Version: 1.21.1

- Durante la ejecución de su código ¿se utiliza más de un procesador? Muestre una imagen (screenshot) de su uso de procesador durante alguna corrida para confirmar.



Tal como se evidencia en la imagen, durante la ejecución del código se utilizan 8 procesadores, los cuales corresponden a todos los de mi CPU.

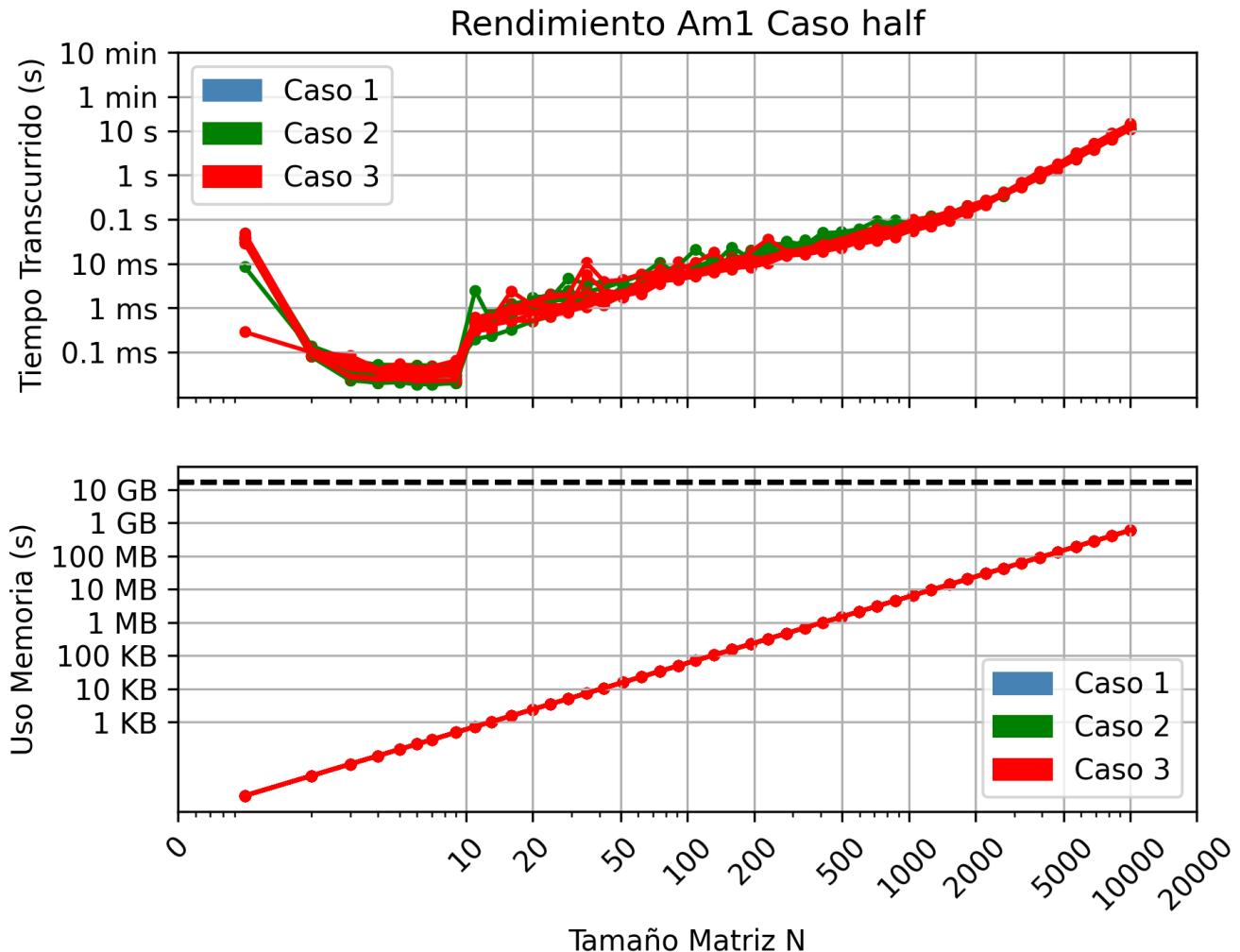
El hecho de que utilice los 8 procesadores (y con tanto %uso), es ya que el código necesita demasiados recursos y de esta manera utiliza demasiada CPU a tal punto de necesitar el 100% de su capacidad.

Desempeño de INV

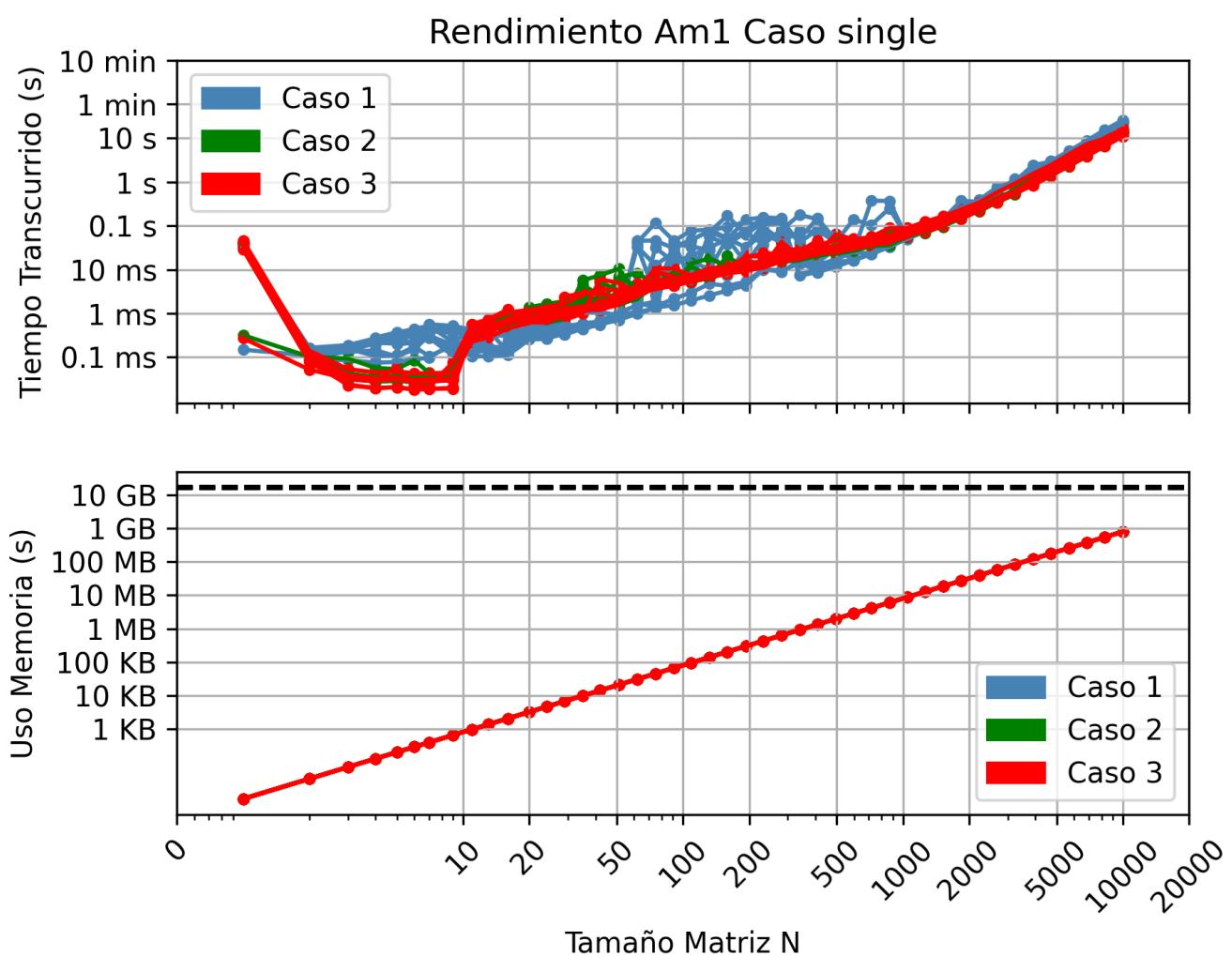
Se realizan 12 archivos .txt, uno por cada tipo en cada uno de los tres casos. Es importante señalar que tanto el tipo half y longdouble en el caso 1 (usando librería numpy) no se logran realizar, ya que son tipos que linalg de numpy no soportan.

A continuación se muestran las comparaciones de los casos por cada tipo de dato:

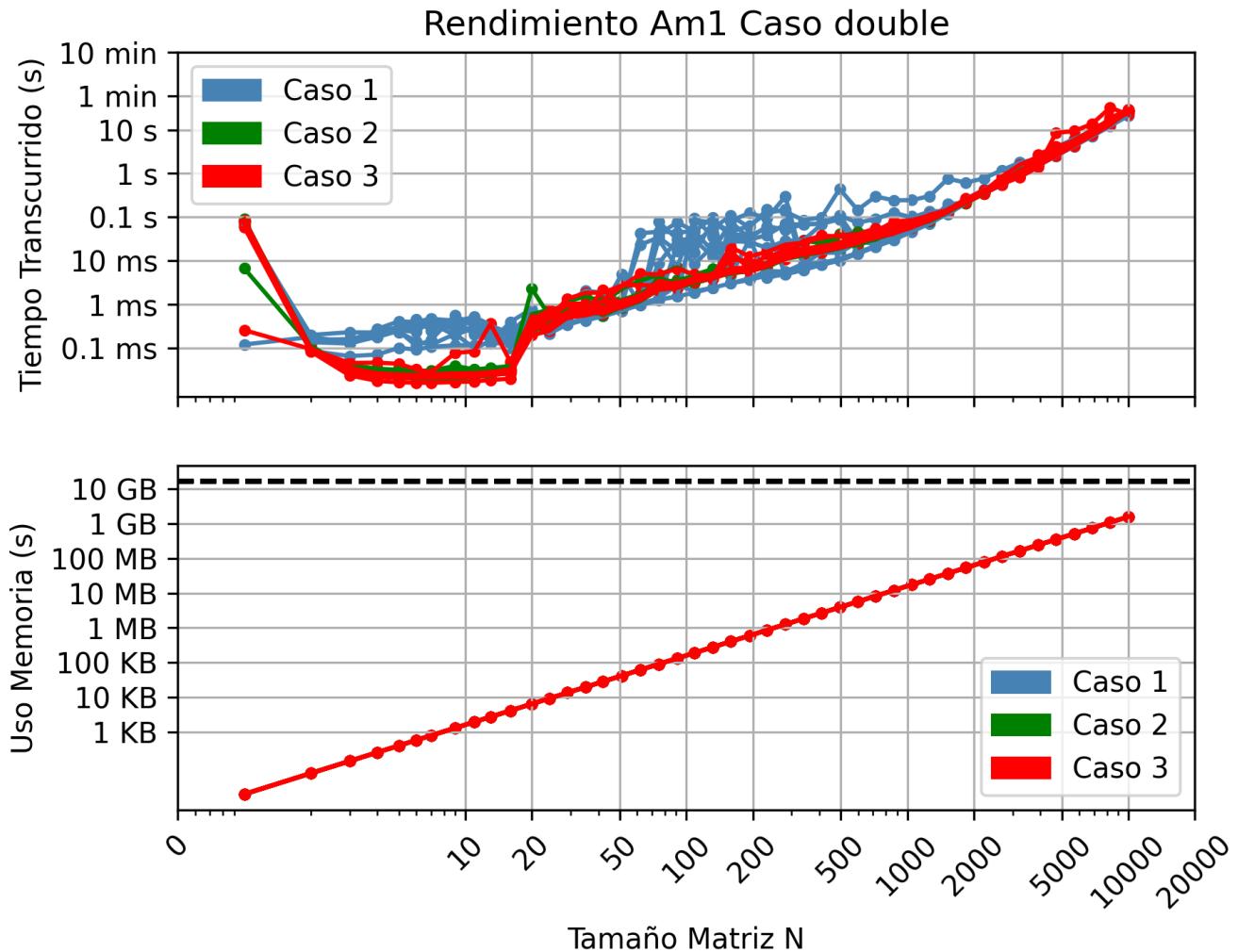
DTYPE: HALF



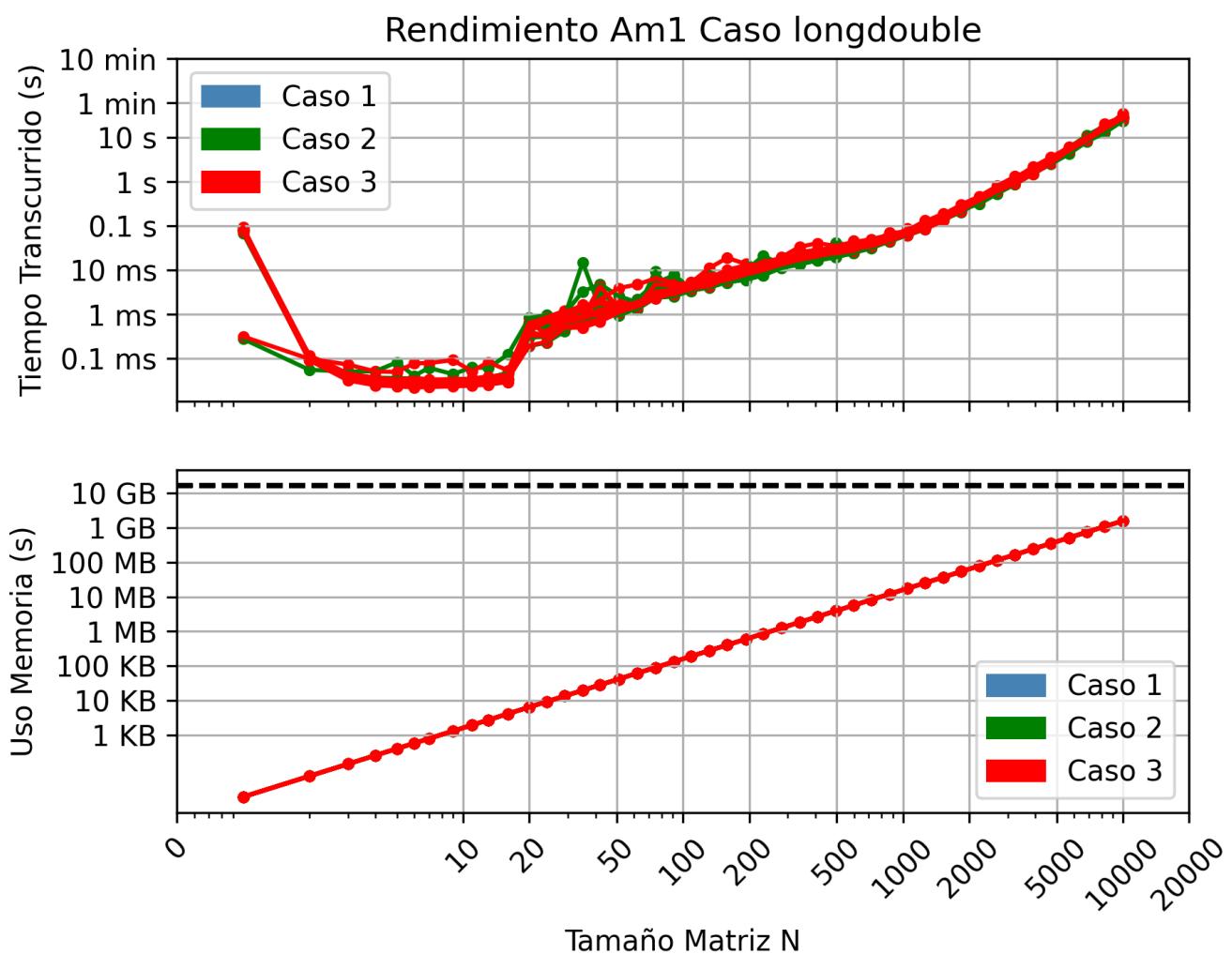
DTYPE: SINGLE



DTYPE: DOUBLE

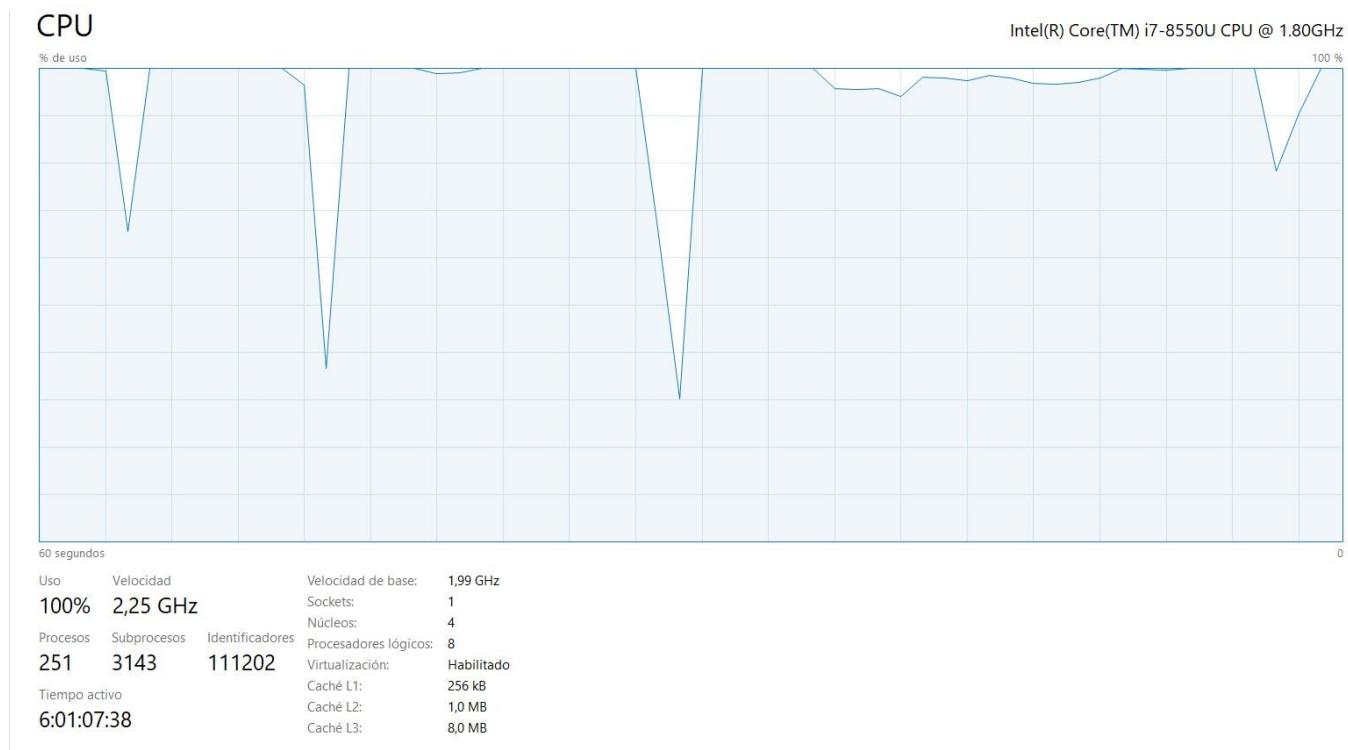


DTYPE: LONGDOUBLE



Además tanto el desempeño del procesador como de la memoria en todos los casos que los códigos funcionaron los resultado fueron prácticamente idénticos, es decir el procesador usando casi todos sus recursos (todos incluso), mientras que la memoria prácticamente no cambiaba y si lo hacía era muy leve. Si hubo alguna diferencia fue prácticamente imperceptible.

Procesador durante corridas:



Memoria durante corridas:



Preguntas

- ¿Qué algoritmo de inversión cree que utiliza cada método (ver wiki)? Justifique claramente su respuesta.

NumPy: numpy.linalg.inv(A) lo que hace es llamar a la función numpy.linalg.solve(A,I) (esta función realiza $Ax = I$, en donde x correspondería en este caso a la matriz inversa de A), en donde I corresponde a la matriz identidad, y luego mediante lapack's LU factorization (paquete de Descomposición LU) resuelve el sistema de solve. Esto significa que finalmente ocupa eliminación Gaussiana en donde la ortogonalidad no se detecta por default. Cabe destacar que este método aumenta el error si el array dado no es cuadrado o la inversión falla.

SciPy: Muy parecido a NumPy, lo que hace Scipy (scipy.linalg.inv(A)) es llamar directamente a los paquetes LAPACK (get_lapack_funcs("función que se quiere")) y desde ahí mediante descomposición LU realiza la inversión de la matriz. Es importante señalar que la opción overwrite_a lo que realiza es reemplazar los valores de la matriz (los va descartando), es por eso que esta opción en True podría incrementar el rendimiento. Importante decir que se puede notar que scipy realiza en general el proceso más rápido ya que llama directamente a los paquetes para realizar la descomposición Lu, en cambio, numpy.linalg.inv() llama a numpy.linalg.solve(), y es esta función la que llama al mismo paquete, es decir "utiliza un paso más"

- ¿Cómo incide el paralelismo y la estructura de caché de su procesador en el desempeño en cada caso? Justifique su comentario en base al uso de procesadores y memoria observado durante las corridas.

La lógica del paralelismo es realizar varias tareas al mismo tiempo, para esto el computador divide el procesador en "mini procesadores" que cada uno realiza diferentes acciones. Sabiendo esto, al correr el programa con datos como half vs longdouble el procesador con half ("menos información"), trabaja de mejor manera con estos "mini procesadores", ya que esa tarea puede distribuirla mejor entre ellos sin tener que colapsar los caché, en cambio con longdouble el procesador necesita usar mayores recursos destinando más cantidad de "mini procesadores" y así dificulta y alarga más el tiempo de ejecución, ya que estos estarán entregando mayor rendimiento con un tamaño de caché al máximo.

Es importante señalar que en mi caso, para todos los tipos de datos el tiempo de corrida fue similar, levemente los datos más pequeños fueron un poco más rápido, pero es de manera prácticamente imperceptible. Esto quiere decir que mi pc, para poder correr datos de half utiliza varios de estos "mini procesadores" y con la capacidad de los caché si no es al máximo, muy cercano, en el caso de un dato más grande como longdouble, probablemente este usando todo los recursos.

Por último, para el caso de la memoria, al tener una gran cantidad y con mucho desuso no significó un gran problema para todos los tipos de datos, sin embargo con datos pequeños esta trabajo leve e imperceptiblemente menos que con datos más grandes.

Desempeño de SOLVE y EIGH

Se realizaron 4 archivos .py (uno para todos los puntos del caso A, otro para todos los puntos del caso B y esto por cada tipo de dato), además de estos se obtuvieron 4 archivos .txt los cuales por línea contienen los subcasos de A y B, pero solo los valores promedios de las 10 corridas por subcaso (son del tipo [[Ns],[dts]]).

A continuación se muestra una tabla con los desempeños del procesador y la memoria por cada caso (subcaso) realizado.

CASO	PROCESADOR (CPU)	MEMORIA (RAM)
A.I (float) ($x = \text{Am1} \times b$)	<p>CPU</p> <p>100% 2.67 GHz Utilidad de base: 1 100 MHz</p> <p>Procesos: 268 Subprocesos: 5362 Identificaciones: 128263 Nodos: 4 Procesadores lógicos: 12848 Habilidades: 12848 Cache L1: 1.0 MB Cache L2: 1.0 MB Cache L3: 8.0 MB</p> <p>8054123</p>	<p>Memoria</p> <p>Uso de memoria</p> <p>16.0 GB 113.08</p> <p>Al asignar: 0 KB Compartir memoria: 0 KB</p> <p>En uso compartido: 8.8 GB (85 MB) Disponible: 7.0 GB Velocidad: 2400 MHz Factor de forma: 500MM Recomendado para la memoria: 117 MB</p> <p>12.1/18.3 GB 5.3 GB</p> <p>Blas en ejecución: 718 MB Blas en caché: 1014 MB</p>
A.II (float) (<code>scipy.linalg.solve</code> by default)	<p>CPU</p> <p>77% 2.35 GHz Utilidad de base: 1 100 MHz</p> <p>Procesos: 363 Subprocesos: 612 Identificaciones: 125757 Nodos: 4 Procesadores lógicos: 125648 Habilidades: 125648 Cache L1: 1.0 MB Cache L2: 1.0 MB Cache L3: 8.0 MB</p> <p>8055510</p>	<p>Memoria</p> <p>Uso de memoria</p> <p>16.0 GB 113.08</p> <p>Al asignar: 0 KB Compartir memoria: 0 KB</p> <p>En uso compartido: 7.5 GB (551 MB) Disponible: 8.2 GB Velocidad: 2400 MHz Factor de forma: 500MM Recomendado para la memoria: 117 MB</p> <p>11.1/18.3 GB 4.6 GB</p> <p>Blas en ejecución: 718 MB Blas en caché: 1012 MB</p>
A.III (float) (using <code>assume_a='pos'</code>)	<p>CPU</p> <p>100% 2.25 GHz Utilidad de base: 1 100 MHz</p> <p>Procesos: 264 Subprocesos: 3107 Identificaciones: 126454 Nodos: 4 Procesadores lógicos: 126448 Habilidades: 126448 Cache L1: 1.0 MB Cache L2: 1.0 MB Cache L3: 8.0 MB</p> <p>8053934</p>	<p>Memoria</p> <p>Uso de memoria</p> <p>16.0 GB 113.08</p> <p>Al asignar: 0 KB Compartir memoria: 0 KB</p> <p>En uso compartido: 7.7 GB (587 MB) Disponible: 8.0 GB Velocidad: 2400 MHz Factor de forma: 500MM Recomendado para la memoria: 117 MB</p> <p>11.5/18.3 GB 5.3 GB</p> <p>Blas en ejecución: 718 MB Blas en caché: 1016 MB</p>
A.IV (float) (using <code>assume_a='sym'</code>)	<p>CPU</p> <p>100% 2.52 GHz Utilidad de base: 1 100 MHz</p> <p>Procesos: 267 Subprocesos: 3107 Identificaciones: 127918 Nodos: 4 Procesadores lógicos: 127912 Habilidades: 127912 Cache L1: 1.0 MB Cache L2: 1.0 MB Cache L3: 8.0 MB</p> <p>8050452</p>	<p>Memoria</p> <p>Uso de memoria</p> <p>16.0 GB 113.08</p> <p>Al asignar: 0 KB Compartir memoria: 0 KB</p> <p>En uso compartido: 7.0 GB (429 MB) Disponible: 8.7 GB Velocidad: 2400 MHz Factor de forma: 500MM Recomendado para la memoria: 117 MB</p> <p>10.1/18.3 GB 5.1 GB</p> <p>Blas en ejecución: 718 MB Blas en caché: 1016 MB</p>
A.V (float) (using <code>overwrite_a=True</code>)	<p>CPU</p> <p>74% 2.52 GHz Utilidad de base: 1 100 MHz</p> <p>Procesos: 264 Subprocesos: 3166 Identificaciones: 127098 Nodos: 4 Procesadores lógicos: 127096 Habilidades: 127096 Cache L1: 1.0 MB Cache L2: 1.0 MB Cache L3: 8.0 MB</p> <p>8060937</p>	<p>Memoria</p> <p>Uso de memoria</p> <p>16.0 GB 113.08</p> <p>Al asignar: 0 KB Compartir memoria: 0 KB</p> <p>En uso compartido: 7.3 GB (488 MB) Disponible: 8.5 GB Velocidad: 2400 MHz Factor de forma: 500MM Recomendado para la memoria: 117 MB</p> <p>10.8/18.3 GB 5.3 GB</p> <p>Blas en ejecución: 719 MB Blas en caché: 1016 MB</p>
A.VI (float) (using <code>overwrite_b=True</code>)	<p>CPU</p> <p>49% 2.63 GHz Utilidad de base: 1 100 MHz</p> <p>Procesos: 265 Subprocesos: 3199 Identificaciones: 127132 Nodos: 4 Procesadores lógicos: 127130 Habilidades: 127130 Cache L1: 1.0 MB Cache L2: 1.0 MB Cache L3: 8.0 MB</p> <p>8061420</p>	<p>Memoria</p> <p>Uso de memoria</p> <p>16.0 GB 113.08</p> <p>Al asignar: 0 KB Compartir memoria: 0 KB</p> <p>En uso compartido: 7.5 GB (521 MB) Disponible: 8.2 GB Velocidad: 2400 MHz Factor de forma: 500MM Recomendado para la memoria: 117 MB</p> <p>10.8/18.3 GB 5.4 GB</p> <p>Blas en ejecución: 719 MB Blas en caché: 1017 MB</p>
A.VII (float) (using <code>overwrite_a=True</code> and <code>overwrite_b=True</code>)	<p>CPU</p> <p>82% 2.22 GHz Utilidad de base: 1 100 MHz</p> <p>Procesos: 266 Subprocesos: 3264 Identificaciones: 127908 Nodos: 4 Procesadores lógicos: 127904 Habilidades: 127904 Cache L1: 1.0 MB Cache L2: 1.0 MB Cache L3: 8.0 MB</p> <p>8061937</p>	<p>Memoria</p> <p>Uso de memoria</p> <p>16.0 GB 113.08</p> <p>Al asignar: 0 KB Compartir memoria: 0 KB</p> <p>En uso compartido: 8.4 GB (502 MB) Disponible: 7.3 GB Velocidad: 2400 MHz Factor de forma: 500MM Recomendado para la memoria: 117 MB</p> <p>11.8/18.3 GB 5.3 GB</p> <p>Blas en ejecución: 720 MB Blas en caché: 1014 MB</p>

CASO	PROCESADOR (CPU)	MEMORIA (RAM)
A.I (double) ($x = \text{Am1} \times b$)	<p>CPU</p> <p>100% 2.63 GHz Procesador: 264 3144 127270 Memoria: 8070053</p> <p>Velocidad de base: 1 GHz Velocidad dinámica: 1 GHz Procesador: 264 3144 127270 Memoria: 8070053</p> <p>807.0053</p>	<p>Memoria</p> <p>16.0 GB 10.0 GB 8.0 GB 6.0 GB 4.0 GB 2.0 GB 0.0 GB</p> <p>Velocidad: 2400 MHz Capacidad: 2 x 8 GB Fabricante: SODIMM Frecuencia para hardware: 177 MHz</p> <p>8.0 GB (548 MB) 6.7 GB 9.0 GB (607 MB) 6.7 GB 13.2/18.3 GB 5.6 GB 725 MB 1020 MB</p>
A.II (double) (<code>scipy.linalg.solve</code> by default)	<p>CPU</p> <p>100% 2.71 GHz Procesador: 266 3236 128277 Memoria: 807.1403</p> <p>Velocidad de base: 1 GHz Velocidad dinámica: 1 GHz Procesador: 266 3236 128277 Memoria: 807.1403</p> <p>807.1403</p>	<p>Memoria</p> <p>16.0 GB 10.0 GB 8.0 GB 6.0 GB 4.0 GB 2.0 GB 0.0 GB</p> <p>Velocidad: 2400 MHz Capacidad: 2 x 8 GB Fabricante: SODIMM Frecuencia para hardware: 177 MHz</p> <p>9.0 GB (607 MB) 6.7 GB 13.2/18.3 GB 5.6 GB 725 MB 1020 MB</p>
A.III (double) (using <code>assume_a='pos'</code>)	<p>CPU</p> <p>82% 2.87 GHz Procesador: 265 3249 128355 Memoria: 807.1828</p> <p>Velocidad de base: 1 GHz Velocidad dinámica: 1 GHz Procesador: 265 3249 128355 Memoria: 807.1828</p> <p>807.1828</p>	<p>Memoria</p> <p>16.0 GB 10.0 GB 8.0 GB 6.0 GB 4.0 GB 2.0 GB 0.0 GB</p> <p>Velocidad: 2400 MHz Capacidad: 2 x 8 GB Fabricante: SODIMM Frecuencia para hardware: 177 MHz</p> <p>7.2 GB (492 MB) 8.5 GB 11.1/18.3 GB 5.4 GB 724 MB 1017 MB</p>
A.IV (double) (using <code>assume_a='sym'</code>)	<p>CPU</p> <p>80% 2.77 GHz Procesador: 264 3239 128393 Memoria: 807.2227</p> <p>Velocidad de base: 1 GHz Velocidad dinámica: 1 GHz Procesador: 264 3239 128393 Memoria: 807.2227</p> <p>807.2227</p>	<p>Memoria</p> <p>16.0 GB 10.0 GB 8.0 GB 6.0 GB 4.0 GB 2.0 GB 0.0 GB</p> <p>Velocidad: 2400 MHz Capacidad: 2 x 8 GB Fabricante: SODIMM Frecuencia para hardware: 177 MHz</p> <p>8.4 GB (510 MB) 7.3 GB 12.4/18.3 GB 5.4 GB 724 MB 1020 MB</p>
A.V (double) (using <code>overwrite_a=True</code>)	<p>CPU</p> <p>55% 3.17 GHz Procesador: 266 3340 127601 Memoria: 807.3000</p> <p>Velocidad de base: 1 GHz Velocidad dinámica: 1 GHz Procesador: 266 3340 127601 Memoria: 807.3000</p> <p>807.3000</p>	<p>Memoria</p> <p>16.0 GB 10.0 GB 8.0 GB 6.0 GB 4.0 GB 2.0 GB 0.0 GB</p> <p>Velocidad: 2400 MHz Capacidad: 2 x 8 GB Fabricante: SODIMM Frecuencia para hardware: 177 MHz</p> <p>7.3 GB (510 MB) 8.5 GB 11.2/18.3 GB 5.4 GB 724 MB 1019 MB</p>
A.VI (double) (using <code>overwrite_b=True</code>)	<p>CPU</p> <p>31% 3.34 GHz Procesador: 264 3261 128947 Memoria: 807.3226</p> <p>Velocidad de base: 1 GHz Velocidad dinámica: 1 GHz Procesador: 264 3261 128947 Memoria: 807.3226</p> <p>807.3226</p>	<p>Memoria</p> <p>16.0 GB 10.0 GB 8.0 GB 6.0 GB 4.0 GB 2.0 GB 0.0 GB</p> <p>Velocidad: 2400 MHz Capacidad: 2 x 8 GB Fabricante: SODIMM Frecuencia para hardware: 177 MHz</p> <p>7.7 GB (505 MB) 8.1 GB 11.3/18.3 GB 5.4 GB 723 MB 1019 MB</p>
A.VII (double) (using <code>overwrite_a=True</code> and <code>overwrite_b=True</code>)	<p>CPU</p> <p>94% 2.53 GHz Procesador: 267 3360 128517 Memoria: 807.3659</p> <p>Velocidad de base: 1 GHz Velocidad dinámica: 1 GHz Procesador: 267 3360 128517 Memoria: 807.3659</p> <p>807.3659</p>	<p>Memoria</p> <p>16.0 GB 10.0 GB 8.0 GB 6.0 GB 4.0 GB 2.0 GB 0.0 GB</p> <p>Velocidad: 2400 MHz Capacidad: 2 x 8 GB Fabricante: SODIMM Frecuencia para hardware: 177 MHz</p> <p>7.7 GB (490 MB) 8.0 GB 11.9/18.3 GB 5.4 GB 725 MB 1019 MB</p>

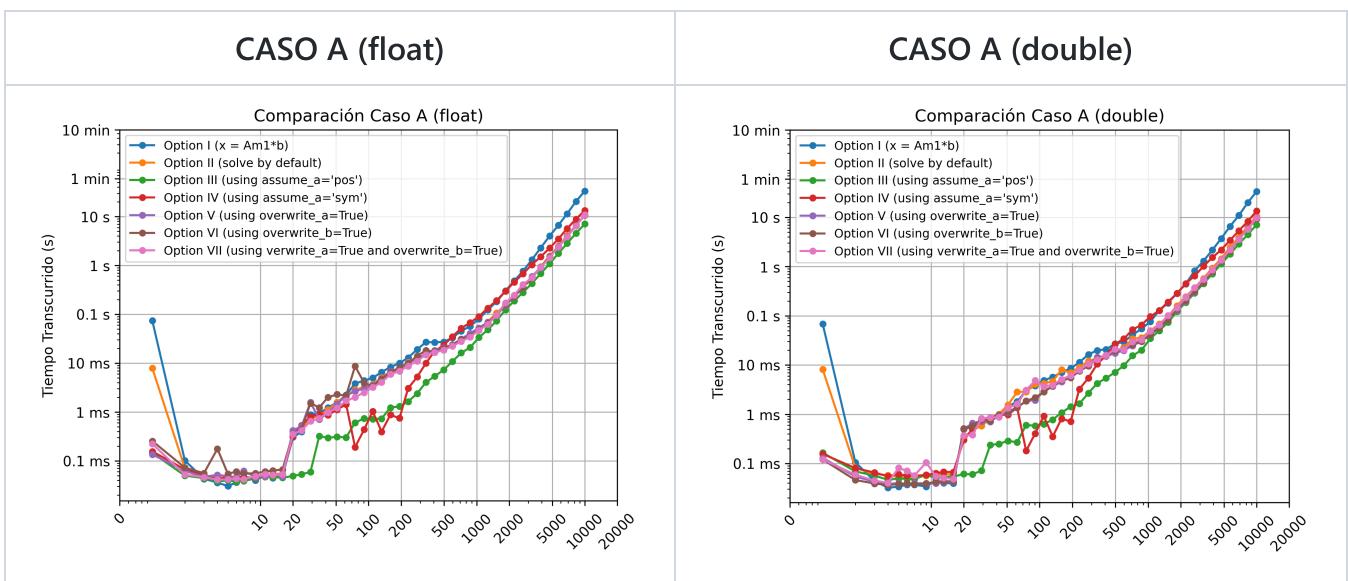
CASO	PROCESADOR (CPU)	MEMORIA (RAM)
B.I (float) (<code>scipy.linalg.eigh</code> by default)	<p>CPU</p> <p>Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>87% Utilizado Procesador: 2.67 GHz Procesador: 3.00 GHz Identificación: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz Número de núcleo: 4 Virtuación: 128x Habilitado: Sí Cache L1: 32 KB Cache L2: 1.0 MB Cache L3: 12.0 MB Tiempo activo: 8:13:26.35</p>	<p>Memoria</p> <p>16.0 GB 11:24</p> <p>8.0 GB (472 MB) 7.7 GB 8.0 GB (472 MB) 7.7 GB 12.0/18.3 GB 6.0 GB 8.0 GB (472 MB) 7.7 GB 913 MB 1.1 GB</p>
B.II.1 (float) (<code>driver='ev'</code> and <code>overwrite_a=False</code>)	<p>CPU</p> <p>Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>34% Utilizado Procesador: 3.77 GHz Procesador: 4.00 GHz Identificación: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz Número de núcleo: 1 Virtuación: 128x Habilitado: Sí Cache L1: 32 KB Cache L2: 1.0 MB Cache L3: 12.0 MB Tiempo activo: 8:13:37.49</p>	<p>Memoria</p> <p>16.0 GB 11:24</p> <p>7.8 GB (477 MB) 7.9 GB 7.8 GB (477 MB) 7.9 GB 12.2/18.3 GB 6.4 GB 7.8 GB (477 MB) 7.9 GB 919 MB 1.1 GB</p>
B.II.2 (float) (<code>driver='ev'</code> and <code>overwrite_a=True</code>)	<p>CPU</p> <p>Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>42% Utilizado Procesador: 3.56 GHz Procesador: 4.00 GHz Identificación: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz Número de núcleo: 1 Virtuación: 128x Habilitado: Sí Cache L1: 32 KB Cache L2: 1.0 MB Cache L3: 12.0 MB Tiempo activo: 8:13:58.05</p>	<p>Memoria</p> <p>16.0 GB 11:24</p> <p>7.2 GB (449 MB) 8.5 GB 7.2 GB (449 MB) 8.5 GB 11.3/18.3 GB 6.5 GB 7.2 GB (449 MB) 8.5 GB 918 MB 1.1 GB</p>
B.III.1 (float) (<code>driver='evd'</code> and <code>overwrite_a=False</code>)	<p>CPU</p> <p>Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>100% Utilizado Procesador: 2.73 GHz Procesador: 3.00 GHz Identificación: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz Número de núcleo: 1 Virtuación: 128x Habilitado: Sí Cache L1: 32 KB Cache L2: 1.0 MB Cache L3: 12.0 MB Tiempo activo: 8:14:22.05</p>	<p>Memoria</p> <p>16.0 GB 11:24</p> <p>6.8 GB (535 MB) 9.1 GB 6.8 GB (535 MB) 9.1 GB 11.2/18.3 GB 6.8 GB 6.8 GB (535 MB) 9.1 GB 918 MB 1.1 GB</p>
B.III.2 (float) (<code>driver='evd'</code> and <code>overwrite_a=True</code>)	<p>CPU</p> <p>Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>92% Utilizado Procesador: 2.68 GHz Procesador: 3.00 GHz Identificación: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz Número de núcleo: 1 Virtuación: 128x Habilitado: Sí Cache L1: 32 KB Cache L2: 1.0 MB Cache L3: 12.0 MB Tiempo activo: 8:14:25.02</p>	<p>Memoria</p> <p>16.0 GB 11:24</p> <p>7.0 GB (535 MB) 8.9 GB 7.0 GB (535 MB) 8.9 GB 11.5/18.3 GB 6.8 GB 7.0 GB (535 MB) 8.9 GB 919 MB 1.1 GB</p>
B.VI.1 (float) (<code>driver='evr'</code> and <code>overwrite_a=False</code>)	<p>CPU</p> <p>Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>99% Utilizado Procesador: 2.69 GHz Procesador: 3.00 GHz Identificación: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz Número de núcleo: 1 Virtuación: 128x Habilitado: Sí Cache L1: 32 KB Cache L2: 1.0 MB Cache L3: 12.0 MB Tiempo activo: 8:14:27.22</p>	<p>Memoria</p> <p>16.0 GB 11:24</p> <p>6.9 GB (535 MB) 8.9 GB 6.9 GB (535 MB) 8.9 GB 11.2/18.3 GB 6.8 GB 6.9 GB (535 MB) 8.9 GB 920 MB 1.1 GB</p>
B.VI.2 (float) (<code>driver='evr'</code> and <code>overwrite_a=True</code>)	<p>CPU</p> <p>Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>100% Utilizado Procesador: 2.65 GHz Procesador: 3.00 GHz Identificación: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz Número de núcleo: 1 Virtuación: 128x Habilitado: Sí Cache L1: 32 KB Cache L2: 1.0 MB Cache L3: 12.0 MB Tiempo activo: 8:14:36.02</p>	<p>Memoria</p> <p>16.0 GB 11:24</p> <p>7.1 GB (532 MB) 8.7 GB 7.1 GB (532 MB) 8.7 GB 11.4/18.3 GB 6.8 GB 7.1 GB (532 MB) 8.7 GB 920 MB 1.1 GB</p>

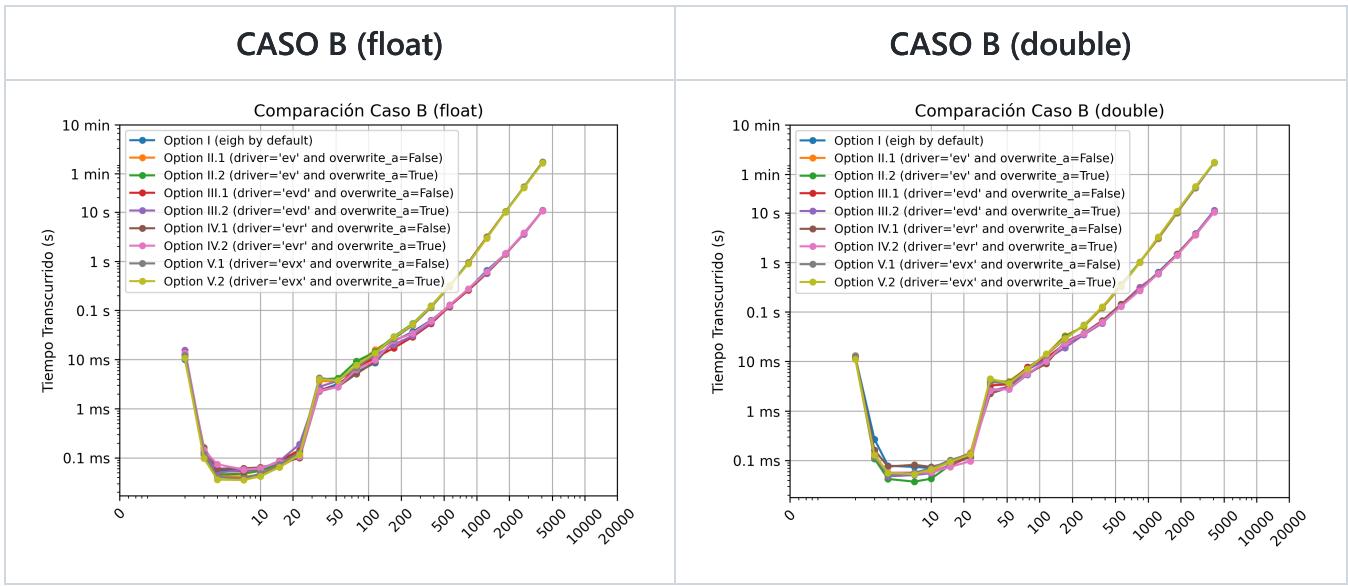
CASO	PROCESADOR (CPU)	MEMORIA (RAM)
B.V.1 (float) (driver='evx' and overwrite_a=False)	<p>CPU</p> <p>Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>39% 3.57 GHz 37% 3.95 GHz 270 3.57 136452 Tiempo actual: 8:14:36:00</p> <p>Marcas de tiempo: 100 ms</p> <p>Velocidad: 1000 mV</p> <p>Monitores de base: 1</p> <p>Sockets: 1</p> <p>Procesador: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>Cache L1: 32 KB Cache L2: 256 KB Cache L3: 1.5 MB</p> <p>Memoria: 16.0 GB</p>	<p>Memoria</p> <p>16.0 GB</p> <p>11.5/18.3 GB 1.1 GB</p> <p>2400 MHz 2.4-4 SODIMM Memoria para la memoria</p> <p>800 MHz 1.0 GB</p>
B.V.2 (float) (driver='evx' and overwrite_a=True)	<p>CPU</p> <p>42% 3.39 GHz 39% 3.95 GHz 261 3.39 132311 Tiempo actual: 8:14:59:51</p> <p>Marcas de tiempo: 100 ms</p> <p>Velocidad: 1000 mV</p> <p>Monitores de base: 1</p> <p>Sockets: 1</p> <p>Procesador: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>Cache L1: 32 KB Cache L2: 256 KB Cache L3: 1.5 MB</p> <p>Memoria: 16.0 GB</p>	<p>Memoria</p> <p>16.0 GB</p> <p>11.5/18.3 GB 1.1 GB</p> <p>2400 MHz 2.4-4 SODIMM Memoria para la memoria</p> <p>800 MHz 921 MB 1.1 GB</p>
B.I (double) (scipy.linalg.eigh by default)	<p>CPU</p> <p>96% 2.85 GHz 95% 3.95 GHz 269 4351 131044 Tiempo actual: 8:21:19:58</p> <p>Marcas de tiempo: 100 ms</p> <p>Velocidad: 1000 mV</p> <p>Monitores de base: 1</p> <p>Sockets: 1</p> <p>Procesador: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>Cache L1: 32 KB Cache L2: 256 KB Cache L3: 1.5 MB</p> <p>Memoria: 16.0 GB</p>	<p>Memoria</p> <p>16.0 GB</p> <p>11.5/18.3 GB 1.1 GB</p> <p>2400 MHz 2.4-4 SODIMM Memoria para la memoria</p> <p>800 MHz 677 MB 1.1 GB</p>
B.II.1 (double) (driver='ev' and overwrite_a=False)	<p>CPU</p> <p>40% 3.54 GHz 39% 3.95 GHz 264 4305 129741 Tiempo actual: 8:21:22:31</p> <p>Marcas de tiempo: 100 ms</p> <p>Velocidad: 1000 mV</p> <p>Monitores de base: 1</p> <p>Sockets: 1</p> <p>Procesador: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>Cache L1: 32 KB Cache L2: 256 KB Cache L3: 1.5 MB</p> <p>Memoria: 16.0 GB</p>	<p>Memoria</p> <p>16.0 GB</p> <p>11.5/18.3 GB 1.1 GB</p> <p>2400 MHz 2.4-4 SODIMM Memoria para la memoria</p> <p>800 MHz 678 MB 1.1 GB</p>
B.II.2 (double) (driver='ev' and overwrite_a=True)	<p>CPU</p> <p>42% 3.12 GHz 42% 3.95 GHz 258 2967 124777 Tiempo actual: 8:21:49:23</p> <p>Marcas de tiempo: 100 ms</p> <p>Velocidad: 1000 mV</p> <p>Monitores de base: 1</p> <p>Sockets: 1</p> <p>Procesador: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>Cache L1: 32 KB Cache L2: 256 KB Cache L3: 1.5 MB</p> <p>Memoria: 16.0 GB</p>	<p>Memoria</p> <p>16.0 GB</p> <p>9.4/18.3 GB 1.0 GB</p> <p>2400 MHz 2.4-4 SODIMM Memoria para la memoria</p> <p>800 MHz 679 MB 1.0 GB</p>
B.III.1 (double) (driver='evd' and overwrite_a=False)	<p>CPU</p> <p>100% 2.87 GHz 98% 3.95 GHz 256 3055 123774 Tiempo actual: 8:22:12:52</p> <p>Marcas de tiempo: 100 ms</p> <p>Velocidad: 1000 mV</p> <p>Monitores de base: 1</p> <p>Sockets: 1</p> <p>Procesador: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>Cache L1: 32 KB Cache L2: 256 KB Cache L3: 1.5 MB</p> <p>Memoria: 16.0 GB</p>	<p>Memoria</p> <p>16.0 GB</p> <p>9.9/18.3 GB 1.0 GB</p> <p>2400 MHz 2.4-4 SODIMM Memoria para la memoria</p> <p>800 MHz 685 MB 1.0 GB</p>
B.III.2 (double) (driver='evd' and overwrite_a=True)	<p>CPU</p> <p>100% 2.71 GHz 99% 3.95 GHz 257 3098 122913 Tiempo actual: 8:22:15:20</p> <p>Marcas de tiempo: 100 ms</p> <p>Velocidad: 1000 mV</p> <p>Monitores de base: 1</p> <p>Sockets: 1</p> <p>Procesador: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz</p> <p>Cache L1: 32 KB Cache L2: 256 KB Cache L3: 1.5 MB</p> <p>Memoria: 16.0 GB</p>	<p>Memoria</p> <p>16.0 GB</p> <p>9.8/18.3 GB 1.0 GB</p> <p>2400 MHz 2.4-4 SODIMM Memoria para la memoria</p> <p>800 MHz 690 MB 1.0 GB</p>

CASO	PROCESADOR (CPU)	MEMORIA (RAM)
B.IV.1 (double) (driver='evr' and overwrite_a=False)	<p>100% Utilizado 2.84 GHz 257 2913 122567 8.22:18:21</p>	<p>16.0 GB 7.4 GB (158 MB) 8.2 GB 9.5/8.3 GB 4.0 GB 690 MB 1.0 GB</p>
B.IV.2 (double) (driver='evr' and overwrite_a=True)	<p>100% Utilizado 2.84 GHz 253 2851 120907 8.22:21:12</p>	<p>16.0 GB 7.4 GB (157 MB) 8.2 GB 9.6/8.3 GB 4.1 GB 688 MB 1.0 GB</p>
B.V.1 (double) (driver='evx' and overwrite_a=False)	<p>42% Utilizado 3.43 GHz 248 2850 118731 8.22:25:16</p>	<p>16.0 GB 6.9 GB (154 MB) 8.7 GB 9.1/8.3 GB 4.1 GB 689 MB 1.0 GB</p>
B.V.2 (double) (driver='evx' and overwrite_a=True)	<p>44% Utilizado 3.53 GHz 238 3029 124466 8.22:32:27</p>	<p>16.0 GB 6.9 GB (171 MB) 8.7 GB 9.6/8.3 GB 4.3 GB 697 MB 1.0 GB</p>

Se puede evidenciar que en general en las corridas el procesador alcanza su máximo en los puntos con mayores valores de N, y es en esos casos que la memoria ram tiende a subir un poco, para ayudar a procesar la información. Otro aspecto interesante a señalar, que para el caso B opción II y V el procesador no usaba en promedio ni el 50% y la memoria nunca se vio afectada, lo curioso es que estos casos fueron los que más tiempo demoraron en ejecutarse (esto explica mejor su tiempo de duración ya que al utilizar menos recursos su tiempo de realización es más lento, quizas esto puede deberse a que la operación no tenía una dificultad tan grande pero si un proceso extenso).

A continuación se muestran los gráficos encontrados:





De los gráficos para el caso A, se puede apreciar que son muy similares entre ellos, casi no hay diferencias, y si las hay no tienen un claro patrón hasta el final (es decir hasta los valores más grandes de N), queda claro que la opción de calcular la inversa y luego multiplicar esta por la matriz b es la manera más lenta de realizar la operación, mientras que la más rápida es utilizando `scipy.linalg.solve` asumiendo una matriz A positiva (`assume_a='pos'`). Es importante señalar que los resultados entre el caso A usando datos de tipo float vs usando datos de tipo double son prácticamente idénticos.

Por otro lado, observando lo ocurrido en el caso B, se puede apreciar que si existen mayores diferencias entre las diferentes opciones, en donde las opciones II.1, II.2, V.1 y V.2 (`driver='ev` y `driver='evx'`) son evidentemente más lentas, mientras que todas las demás tienen tiempos prácticamente iguales obteniendo un mejor rendimiento. Al igual que el caso anterior, la diferencia entre el tipo de dato double y float es prácticamente imperceptible.

Preguntas

- Haga un comentario completo respecto de todo lo que ve en términos de desempeño en cada problema.

Tal como se menciona antes se puede apreciar como en el caso A son todos los casos muy parecidos, siendo el método de utilizar la inversa el más lento, mientras que el más rápido es asumiendo que la matriz A es positiva.

Para el caso B, usar `driver='ev'` y `driver='evx'` relentizan considerablemente el proceso, mientras que los demás trabajan de manera más eficiente.

(Más comentarios debajo de los gráficos...).

- ¿Como es la variabilidad del tiempo de ejecucion para cada algoritmo?

En el caso A se puede evidenciar que no hay tanta variabilidad de tiempo, pero que para valores de N entre aproximadamente 60 y 220 el caso más optimo será asumir una matriz A simétrica, mientras que para los todos los demás casos de N (menores y mayores), el tiempo de ejecución de usar `scipy.linalg.solve` asumiendo una matriz positiva es mejor.

Para el caso B utilizando `scipy.linalg.eigh`, claramente hay una diferencia notable de tiempo entre los subcasos con `driver='ev'` y `driver='evx'`, los cuales serán siempre más lentos, mientras que todos los demás casos tienen un comportamiento casi identico con un tiempo de ejecución considerablemente mejor.

Cabe agregar que el método `scipy.linalg.solve` es evidentemente más rápido que `scipy.linalg.eigh`.

- ¿Qué algoritmo gana (en promedio) en cada caso?

En el caso A en promedio el algoritmo más rápido es `scipy.linalg.solve(assume_a='pos')`.

En el caso B en promedio el algoritmo más rápido es `scipy.linalg.eigh(driver="evd", overwrite_a=True)` --> Es levemente más rápido, prácticamente imperceptible.

- ¿Depende del tamaño de la matriz?

Para el caso A, si es influyente el tamaño de la matriz, porque como ya mencioné antes el subcaso IV tiene mejor rendimiento que cualquiera con matrices de tamaño entre $60 < N < 220$.

Para el caso B, el porte de la matriz no es un factor influyente, siempre existe el mismo comportamiento (si hay diferencias es prácticamente imperceptible).

- ¿A que se puede deber la superioridad de cada opción?

La superioridad de cada opción tanto en el caso A como en el caso B, se debe plenamente a los paquetes y el proceso que reliza "por detrás" los diferentes métodos con sus diferentes parámetros. Por ejemplo, si se utiliza un argumento `overwrite_a=True`, se utilizarán menos recursos, ya que no se esta creando una nueva matriz, sino que se está reescribiendo la misma, otros ejemplos podrían ser que al asumirla de un tipo (positiva o simétrica), en donde el cálculo es más fácil, y el paquete al estar informado, realiza "trucos" de resolución más óptimos.

- ¿Su computador usa más de un proceso por cada corrida?

Tal como se evidencia en las imágenes del procesador, este utiliza los 8 procesadores lógicos para la mayoría de los casos, hay un par de excepciones en donde utiliza 2 al máximo y los demás en mitad del rendimiento, pero en estricto rigor siempre esta utilizando cerca del 100% de la memoria de la CPU en los casos con N altos. Que el computador utilice más de un proceso significa que esta realizando muchas acciones diferentes simultaneamente y esto viene definido en parte por el computador (por que decide usar), pero también por las librerías y paquetes que se utilicen, los cuales pueden requerir de hacer varias acciones en simultáneo para hacerlas de manera más eficiente.

- ¿Que hay del uso de memoria (como crece)?

En mi caso, la memoria se vio muy poco afectada, ya que mi procesador es bastante potente, solo cambiaba levemente cuando el procesador sobrepasaba el 100% de su rendimiento, y cuando lo hacía de todas maneras lo que cambiaba la memoria era muy poco. Probablemente esto haya ocurrido para los valores de N que eran más grandes.

Todo lo anteriormente señalado, ocurre por la jerarquización de la memoria en los computadores.

Matrices dispersas y complejidad computacional

Se realizó un archivo .py, el cual desarrollaba la operación MATMUL tanto con matrices llenas como con dispersas, se puede notar que tanto el tiempo de ensamblaje de matrices laplaciañas, tanto como resolver la operación, era infinitamente más rápido usando matrices dispersas, es más, para poder notar el gráfico, para las matrices dispersas se tuvo que utilizas un $N_{max} = 1.000.000$.

Ensamblaje

Para realizar el ensamblaje tanto de las matrices dispersas como de las llenas, se definieron las siguientes funciones para poder desarrollarlas:

Función Laplaciana Matrices Llenas

```
def matriz_laplaciana_llena(N, dtype):
    A = zeros((N,N), dtype = dtype)
    for i in range(N):
        A[i,i] = 2
        for j in range(max(0,i-2),i):
            if abs(i - j) == 1:
                A[i,j] = -1
                A[j,i] = -1
    return A
```

Función Laplaciana Matrices Dispersas

```
def matriz_laplaciana_dispersa(N, dtype):
    return 2*sparse.eye(N, dtype = dtype) - sparse.eye(N, N, 1, dtype = dtype) -
sparse.eye(N, N, -1, dtype = dtype)
```

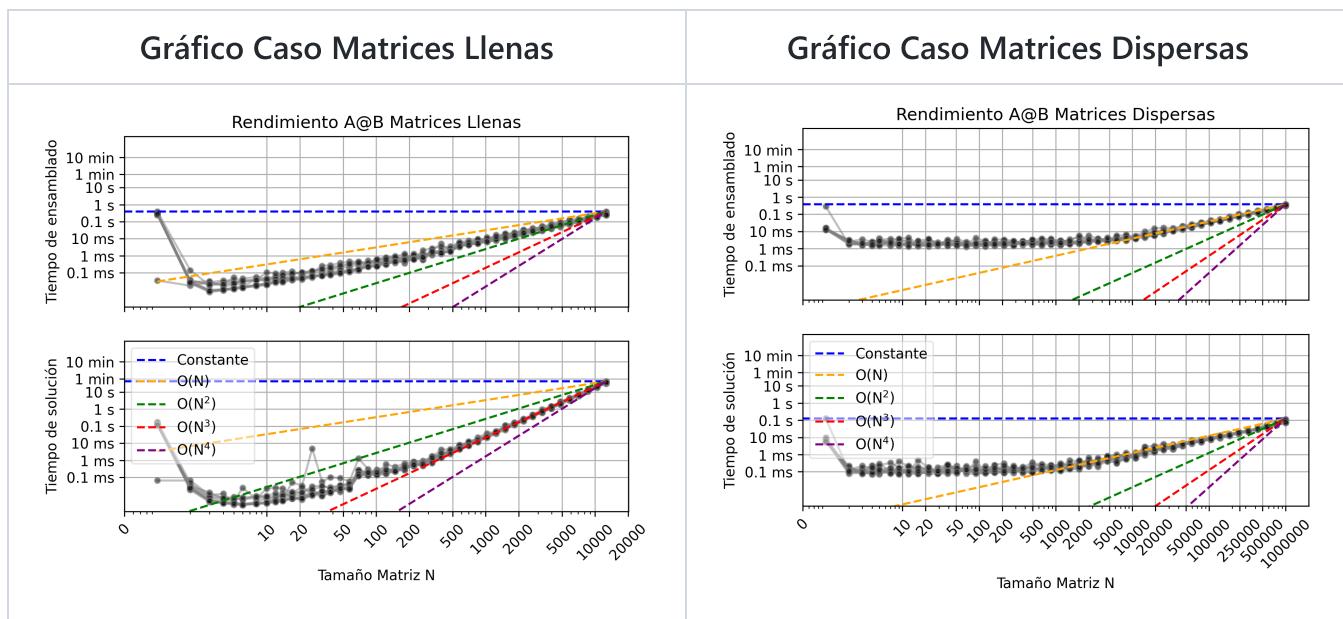
Para las matrices llenas no se pudo utilizar el método eye, ya que para el tipo de dato double no lo soportaba, mientras que para el caso de matrices dispersas si.

Solución

Para la solución simplemente se uso el método matmul (@), pero es importante para el caso de las matrices dispersas antes de realizar la operación, tanto las matrices A como B se transformaron a una matriz de tipo CSR.

Gráficos

A continuación se muestran los gráficos encontrados:



Es importante destacar que para poder desarrollar los gráficos representativos de complejidad computacional, se tuvo que trabajar pensando que se tenía un gráfico doblemente logarítmico, es decir del tipo $\log(y) = k \cdot \log(x) + \log(a)$ que otra manera de representarlo sería $y = a \cdot x^k$. De esta manera de despejan los distintos niveles de complejidad, ya que se conoce un punto (el máximo) y su pendiente, quedando de la siguiente manera:

$$\frac{y_{max}}{N_{max}^k} \cdot N_{values}^k.$$

Luego se puede interpretar como para el caso de las matrices llenas para el caso del ensamblaje tiene un nivel de complejidad máxima de $O(N^2)$ y para la solución de MATMUL en la parte final casi de $O(N^4)$.

Para el caso de las matrices dispersas tuvieron que agregarse más valores ya que osino no se podía ver que hubiera una complejidad dada. Finalmente se puede notar que para el ensamblaje existe una complejidad máxima de $O(N^2)$ y para el caso de la solución de $O(N^2)$.