MCOC2021-P0

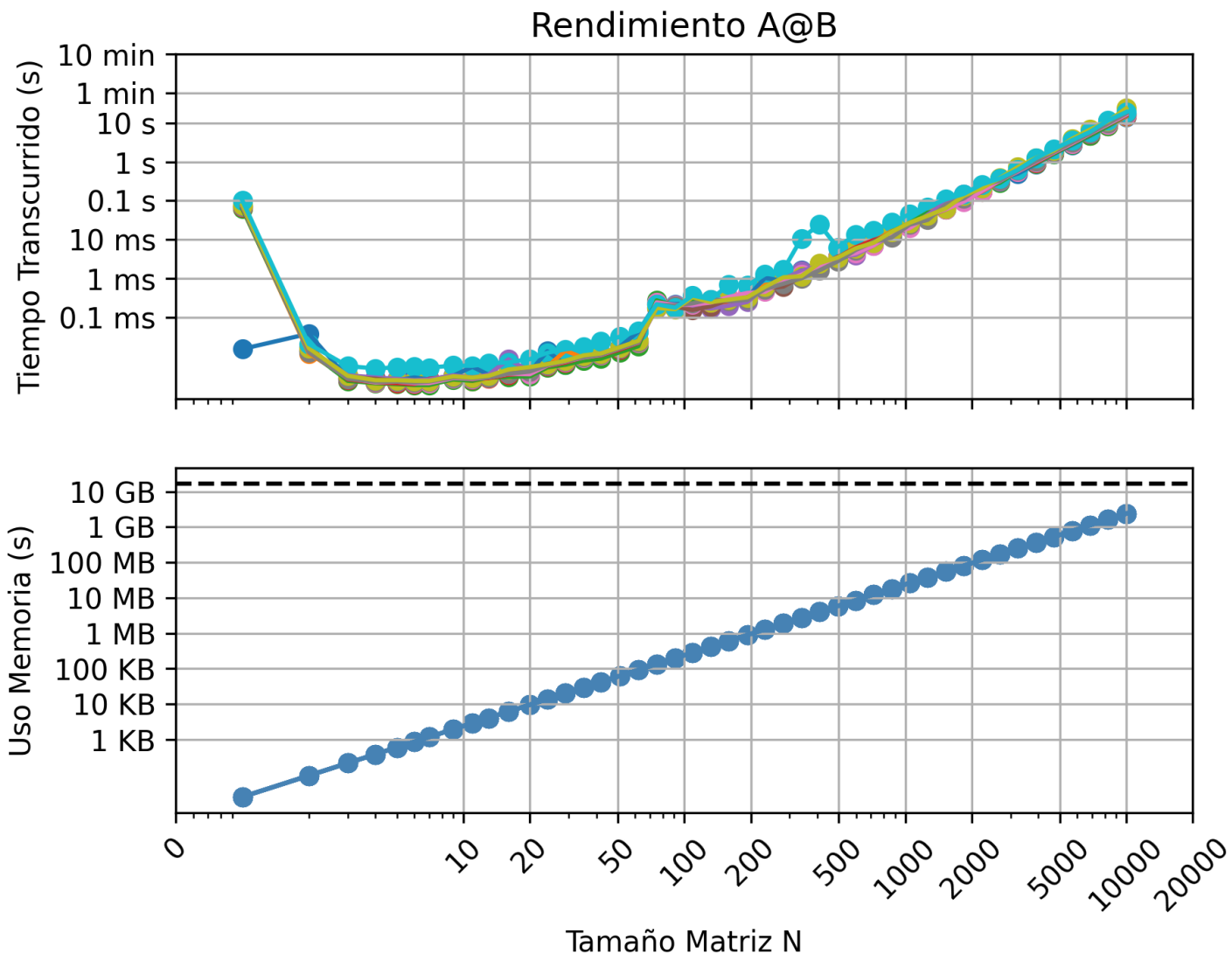
Mi computador principal

- 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 en general, los gráficos son muy similares.

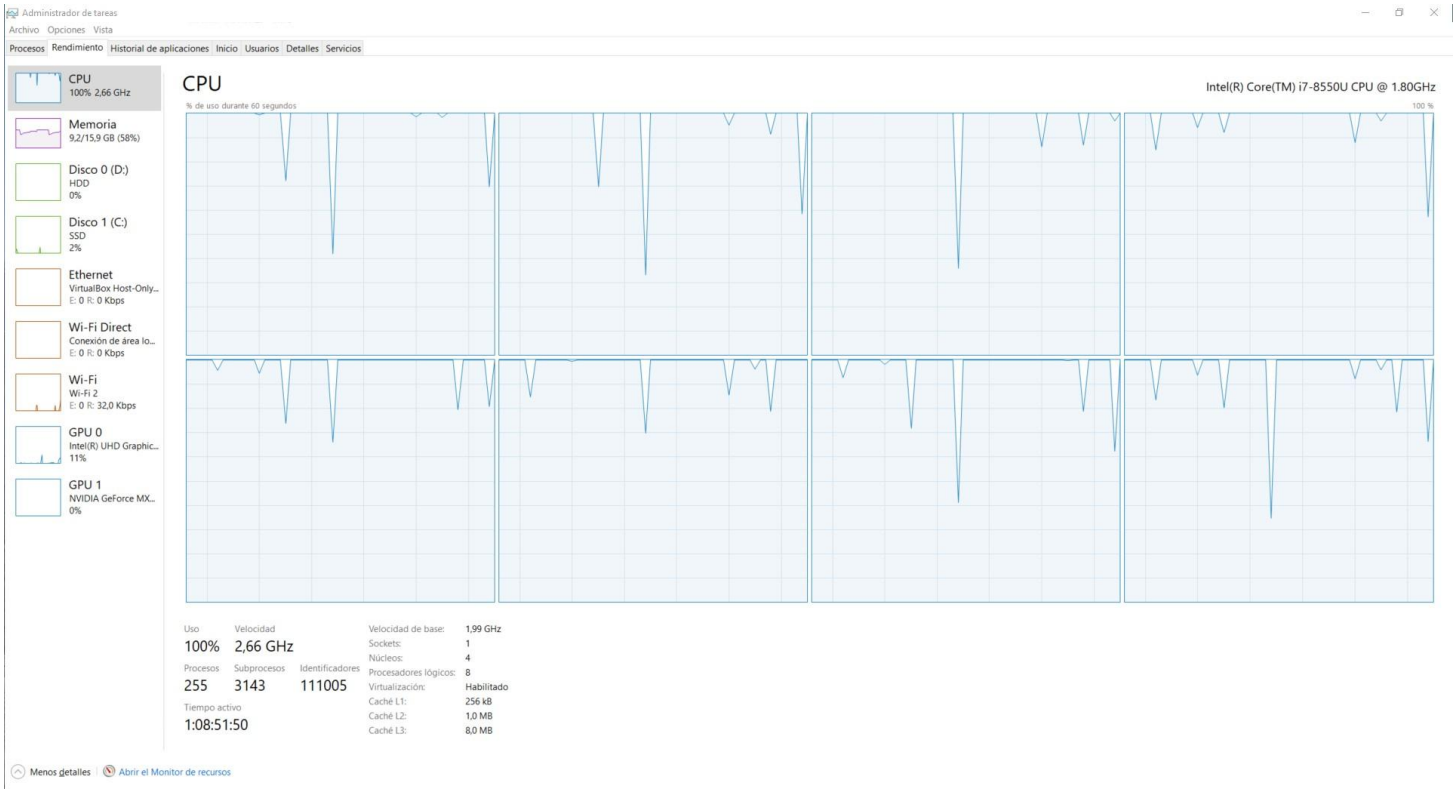
- **¿A qué se pueden deber las diferencias en cada corrida?**
  Esto ocurre, ya que, python para el uso de memoria siempre intenta utilizar la menor cantidad de memoria posible. Además, la memoria funciona por bloques, entonces al necesitar más memoria ira en busqueda del siguiente módulo de memoria 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 jerarquia 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, estás 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 exponencialemente (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.**



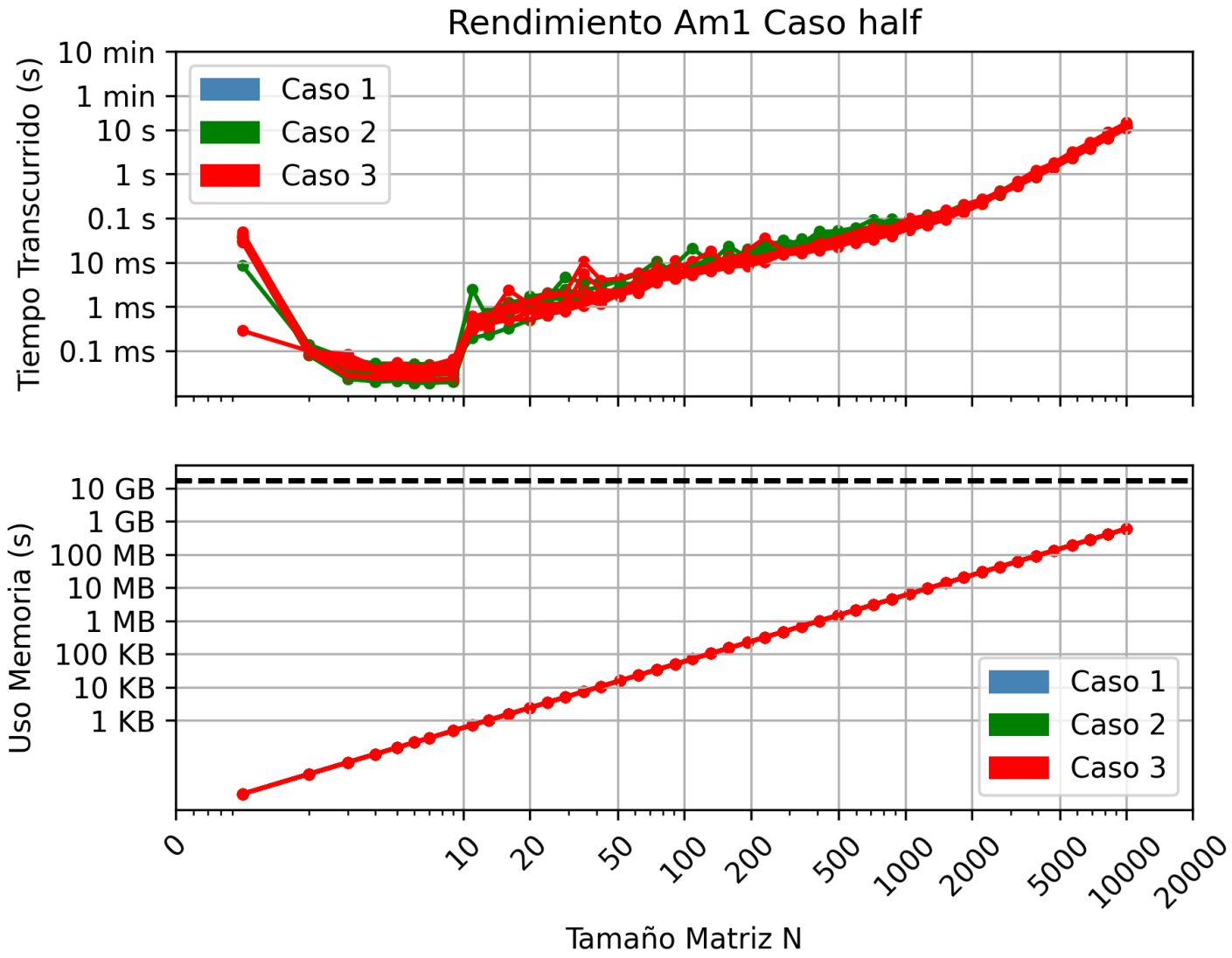
Tal como se evidencia en la imágen, durante la ejecución del código se utilizan 8 procesadores, los cuales corresponden a todos los de mi CPU.
El hecho de que utilice los 8 procesadores (y con tanto %uso), es ya que el código necesita demasiados recursos y de esta manera utiliza demasiada CPU a tal punto de necesitar el 100% de su capacidad.

Desempeño de INV

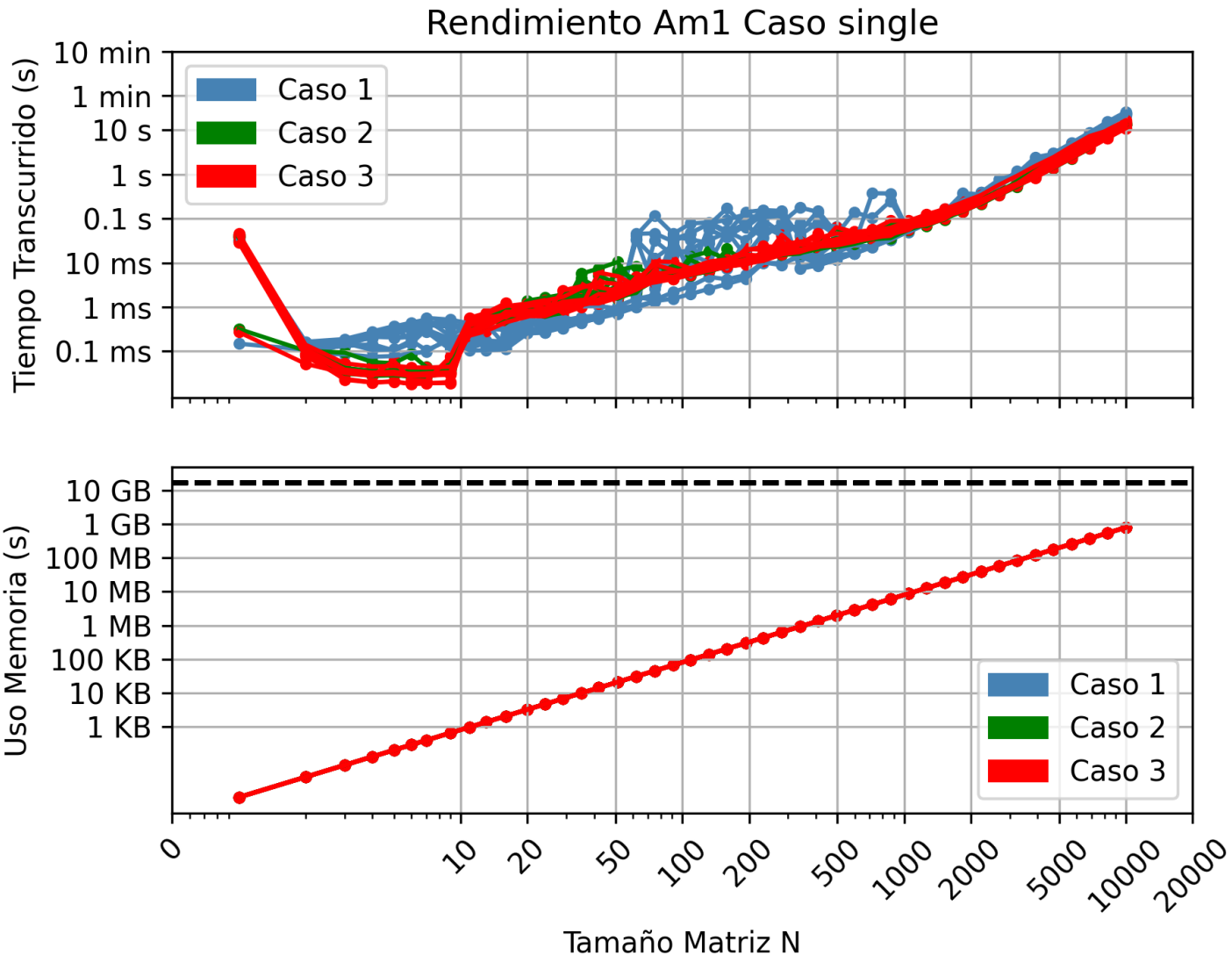
Se realizan 12 archivos .txt, uno por cada tipo en cada uno de los tres casos. Es importante señalar que tanto el tipo half y longdouble en el caso 1 (usando librería numpy) no se logran realizar, ya que son tipos que linalg de numpy no soportan.

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

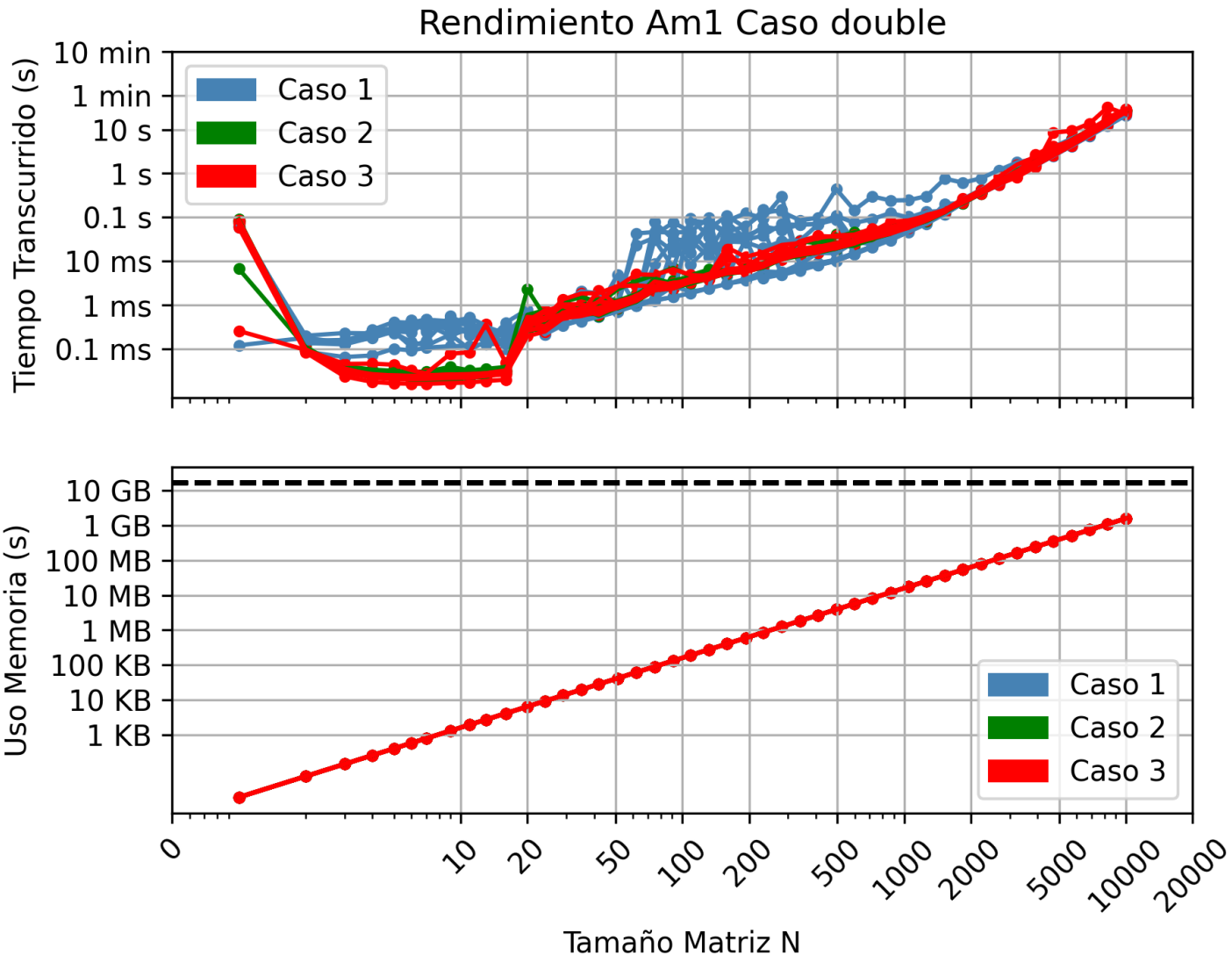
DTYPE: HALF


Rendimiento Am1 Caso half
Tiempo Transcurrido (s)
10 min
1 min
10 s
1 s
0.1 s
10 ms
1 ms
0.1 ms
Caso 1
Caso 2
Caso 3
Uso Memoria (s)
10 GB
1 GB
100 MB
10 MB
1 MB
100 KB
10 KB
1 KB
Caso 1
Caso 2
Caso 3
0
10
20
50
100
200
500
1000
2000
5000
10000
20000
Tamaño Matriz N

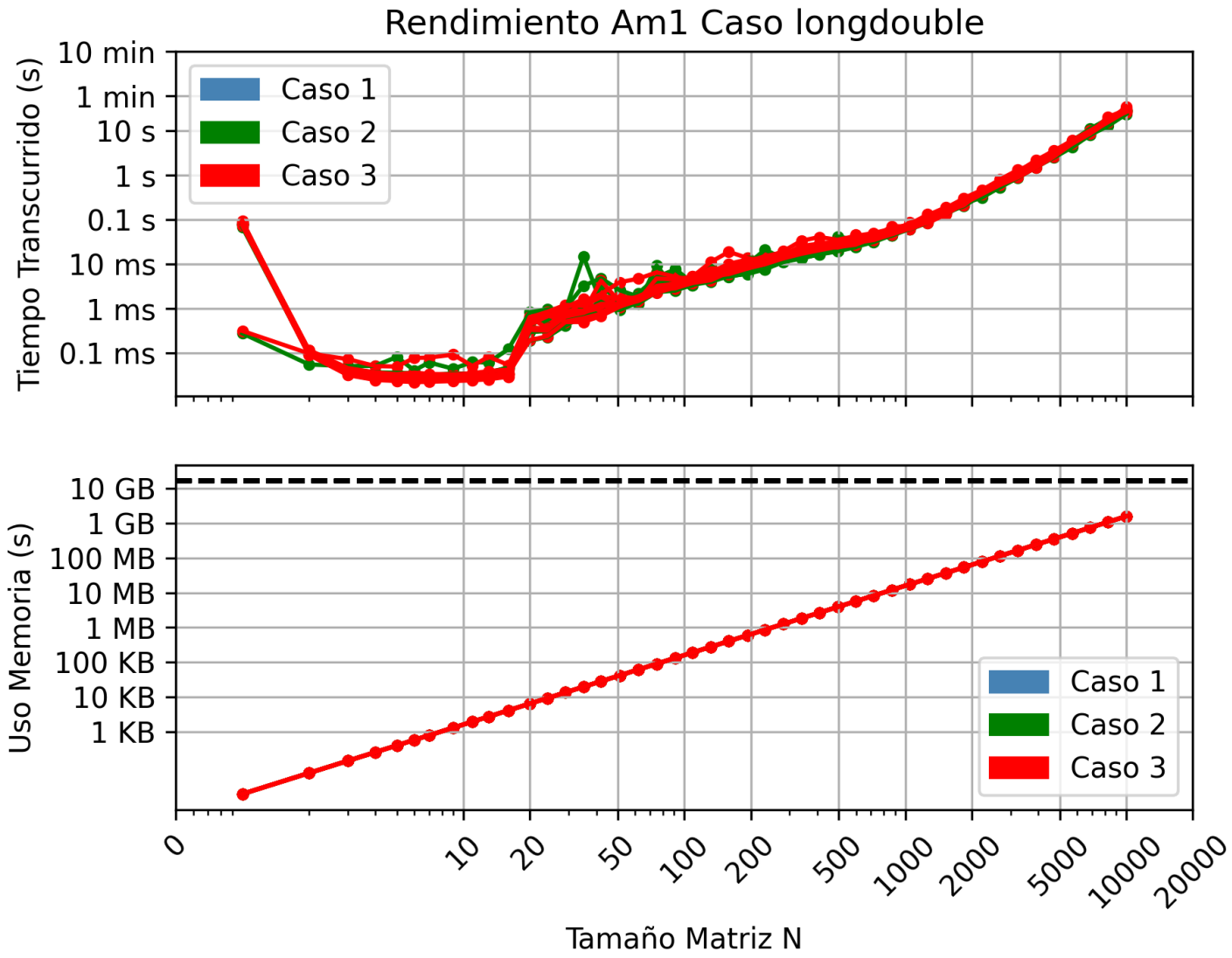
DTYPE: SINGLE



DTYPE: DOUBLE

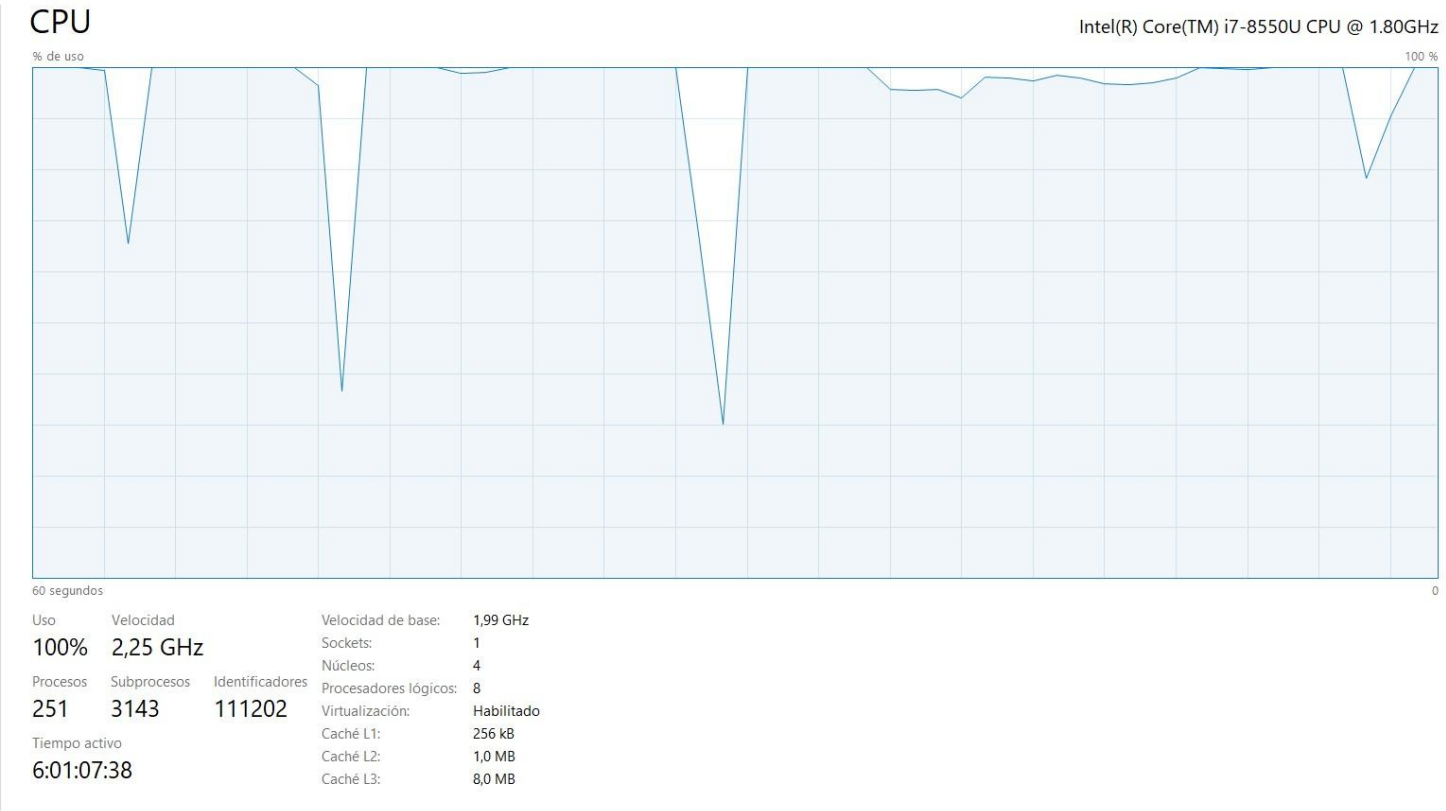


DTYPE: LONGDOUBLE

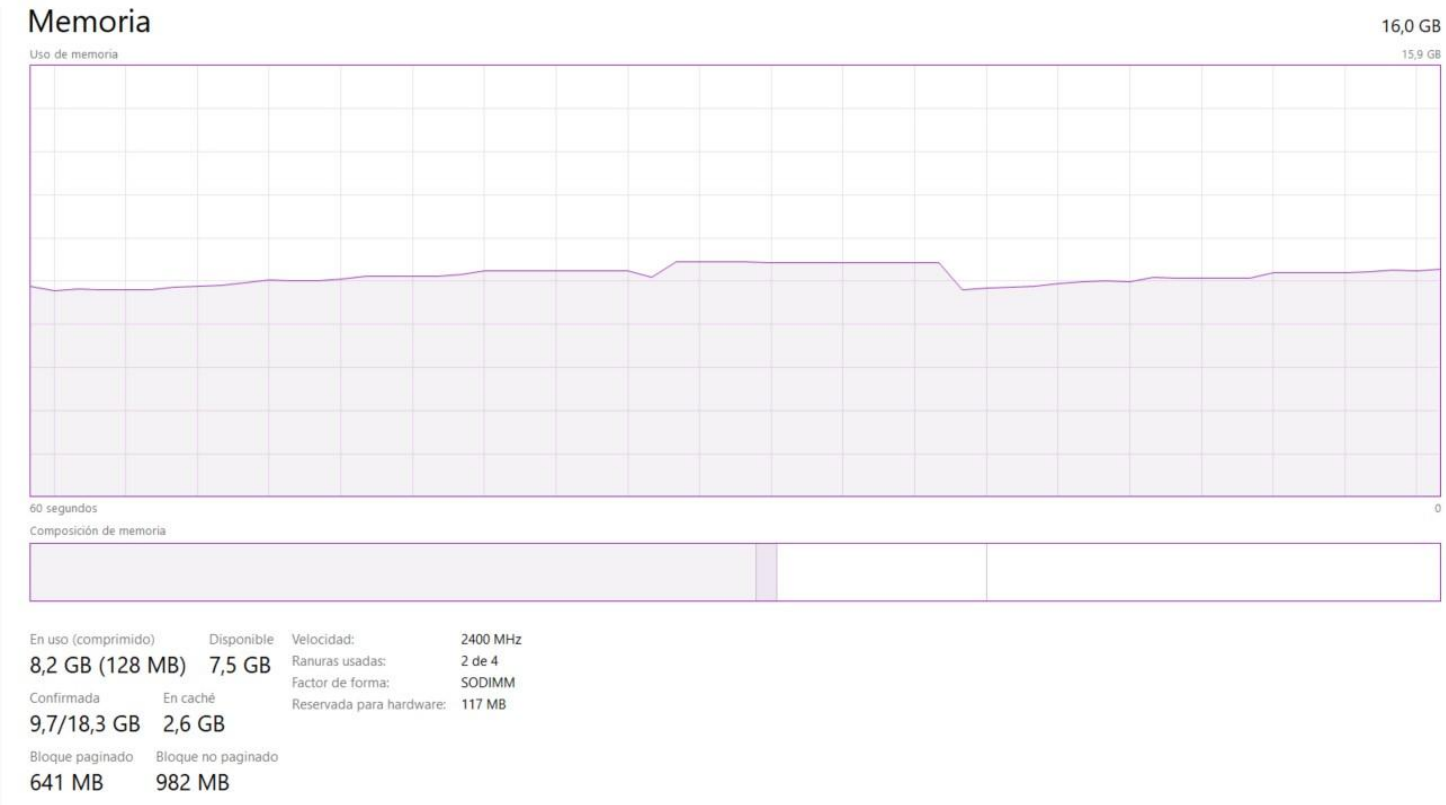


Además tanto el desempeño del procesador como de la memoria en todos los casos que los códigos funcionaron los resultado fueron prácticamente idénticos, es decir el procesador usando casi todos sus recursos (todos incluso), mientras que la memoria prácticamente no cambiaba y si lo hacía era muy leve. 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 correpsonde a la matriz identidad, y luego mediante lapack's LU factorization (paquete de Descomposición LU) resuleve 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 rendimeinto 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 imperciptiblemente menos que con datos más grandes.