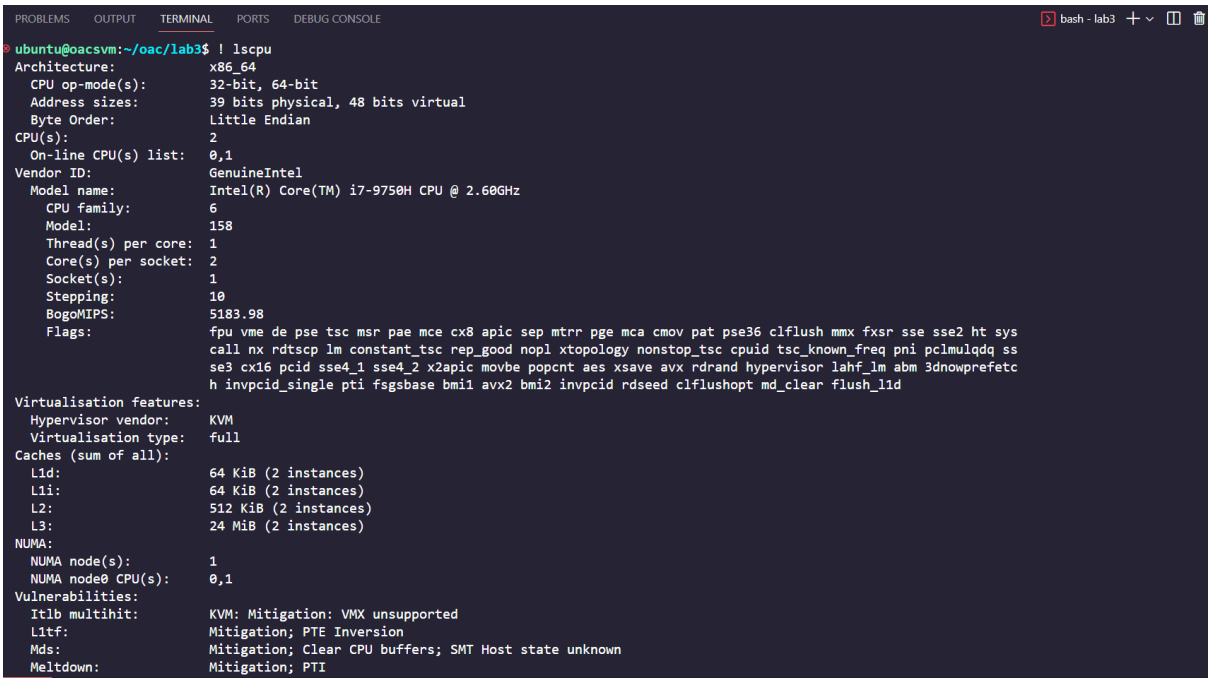
Nombre: Christian Edgardo Carrillo Aburto
Código: 20202692

Características de mi computadora:

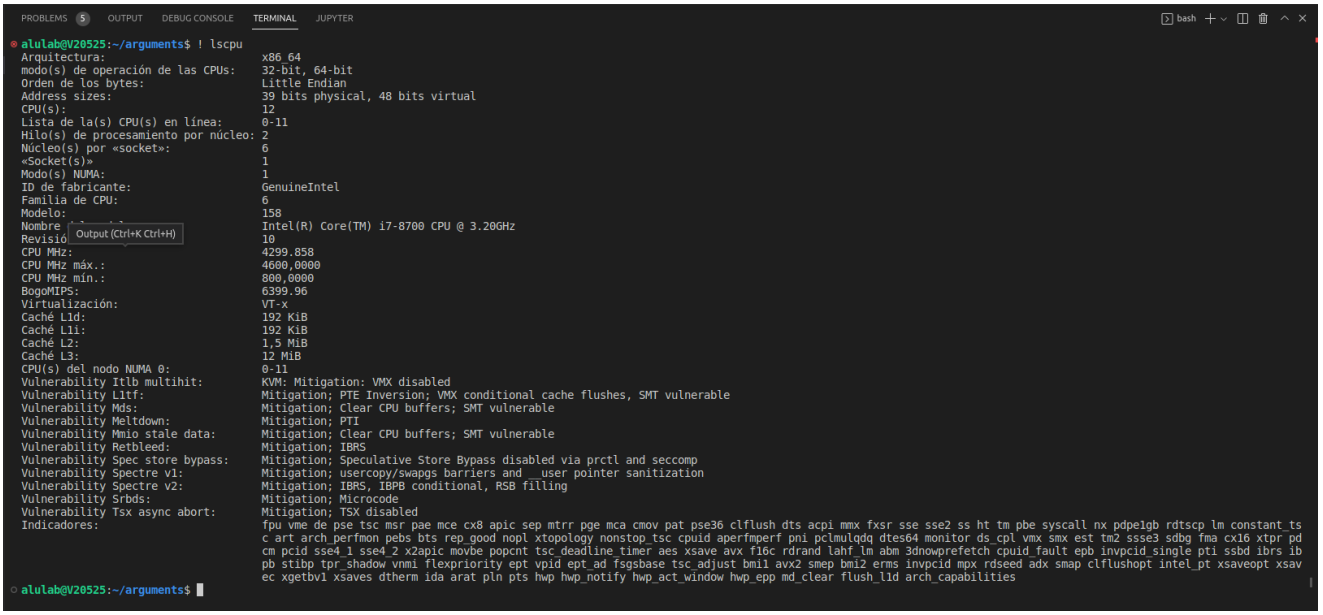


```
PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE                                bash - lab3

ubuntu@oacsvm:~/oac/lab3$ ! 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):                  2
  On-line CPU(s) list:   0,1
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
    CPU family:          6
    Model:               158
    Thread(s) per core:  1
    Core(s) per socket:  2
    Socket(s):           1
    Stepping:            10
    BogoMIPS:            5183.98
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht sys
                         call nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ss
                         se3 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor lahf_lm abm 3dnowprefetc
                         h invpcid_single pti fsgsbase bmi1 avx2 bmi2 invpcid rdseed clflushopt md_clear flush_l1d
Virtualisation features:
  Hypervisor vendor:     KVM
  Virtualisation type:   full
Caches (sum of all):
  L1d:                   64 KiB (2 instances)
  L1i:                   64 KiB (2 instances)
  L2:                    512 KiB (2 instances)
  L3:                    24 MiB (2 instances)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0,1
Vulnerabilities:
  Itlb multihit:         KVM: Mitigation: VMX unsupported
  L1tf:                  Mitigation; PTE Inversion
  Mds:                   Mitigation; Clear CPU buffers; SMT Host state unknown
  Meltdown:              Mitigation; PTI
```

```
                         h invpcid_single pti fsgsbase bmi1 avx2 bmi2 invpcid rdseed clflushopt md_clear flush_l1d
Virtualisation features:
  Hypervisor vendor:     KVM
  Virtualisation type:   full
Caches (sum of all):
  L1d:                   64 KiB (2 instances)
  L1i:                   64 KiB (2 instances)
  L2:                    512 KiB (2 instances)
  L3:                    24 MiB (2 instances)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0,1
Vulnerabilities:
  Itlb multihit:         KVM: Mitigation: VMX unsupported
  L1tf:                  Mitigation; PTE Inversion
  Mds:                   Mitigation; Clear CPU buffers; SMT Host state unknown
  Meltdown:              Mitigation; PTI
  Mmio stale data:       Mitigation; Clear CPU buffers; SMT Host state unknown
  Retbleed:              Vulnerable
  Spec store bypass:     Vulnerable
  Spectre v1:            Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:            Mitigation; Retpolines, STIBP disabled, RSB filling, PBRSB-eIBRS Not affected
  Srbds:                 Unknown: Dependent on hypervisor status
  Tsx async abort:       Not affected
ubuntu@oacsvm:~/oac/lab3$
```

Características de la computadora del V: Máquina V205-25



A)

```
void mat_vec(int *A, int *B, int *C, int N)
{
    int tmp = 0;
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            C[i*N+j]=0;
            for(int k = 0;k<N;k++){
            C[i*N+j] += A[i*N + k]*B[k*N+j];
            }
        }
    }
}
```

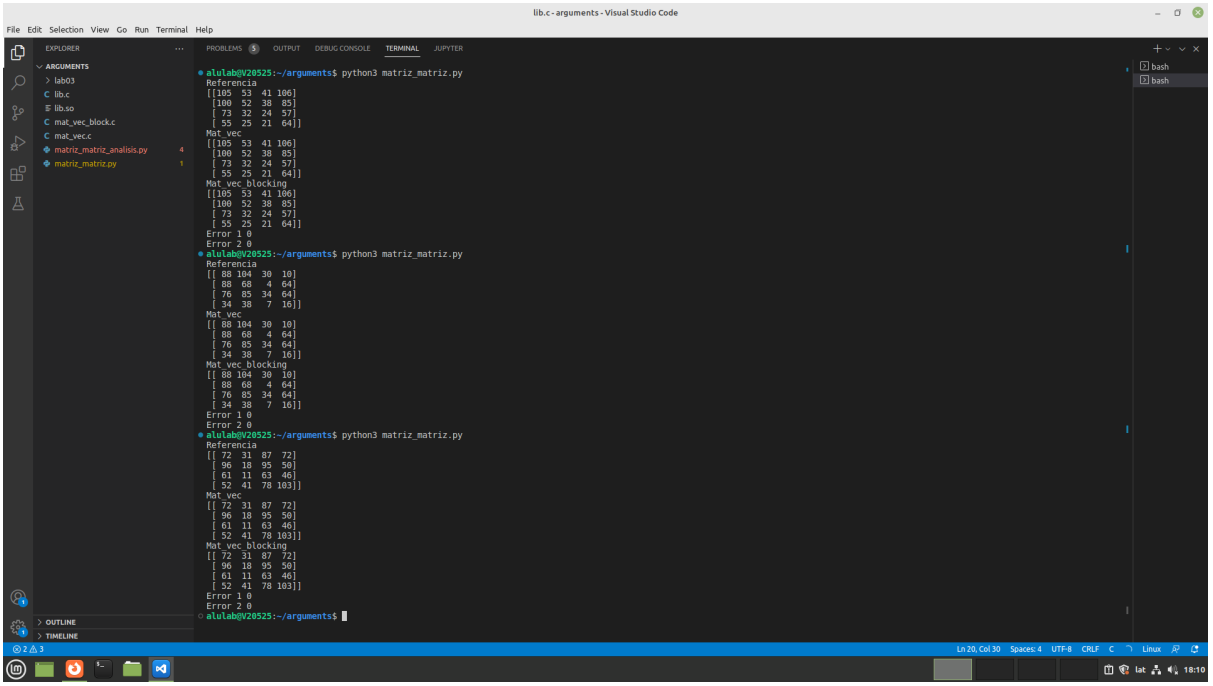
B)

```
void mat_vec_block(int *A, int *B, int *C, int N, int
block)
```

```
{
    int i,j,k,kk,jj;
    int sum;
    int en = block*(N/block);

    for (kk = 0; kk < en; kk += block) {
        for (jj = 0; jj < en; jj += block) {
            for(i = 0; i < N; i++){
                for (j = jj; j < jj + block; j++){
                sum = C[i*N+j];
                for (k = kk; k < kk + block; k++)
                    {
                        sum+= A[i*N+k]*B[k*N+j];
                    }
                    C[i*N+j]= sum;
                }
            }
        }
    }
}
```

C)

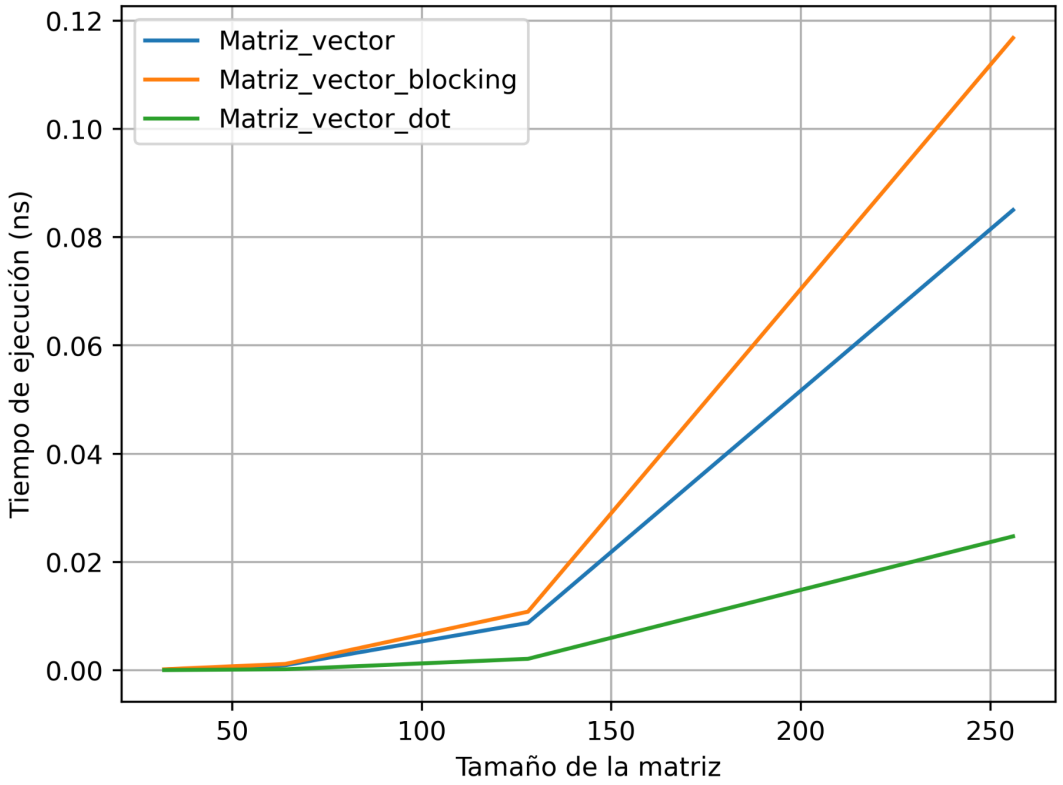


En la imagen se observa el lib.so y la ejecución del programa, en el cual se muestra que en las 3 pruebas se obtiene el mismo resultado.

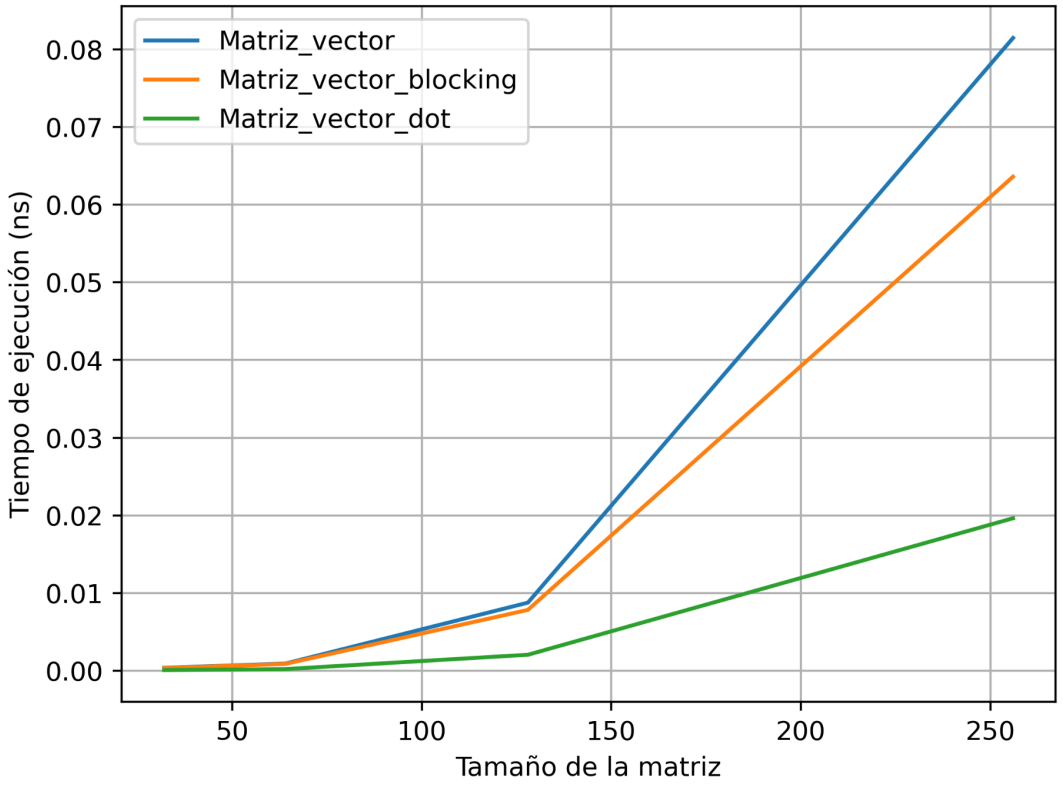
D)
Nota: Cada gráfica fue ejecutada desde mi computadora personal.

Los gráficos se realizaron con 5 iteraciones porque con 15 se demoró más de una hora en mi computadora.
Entonces
ITERACIONES = 5

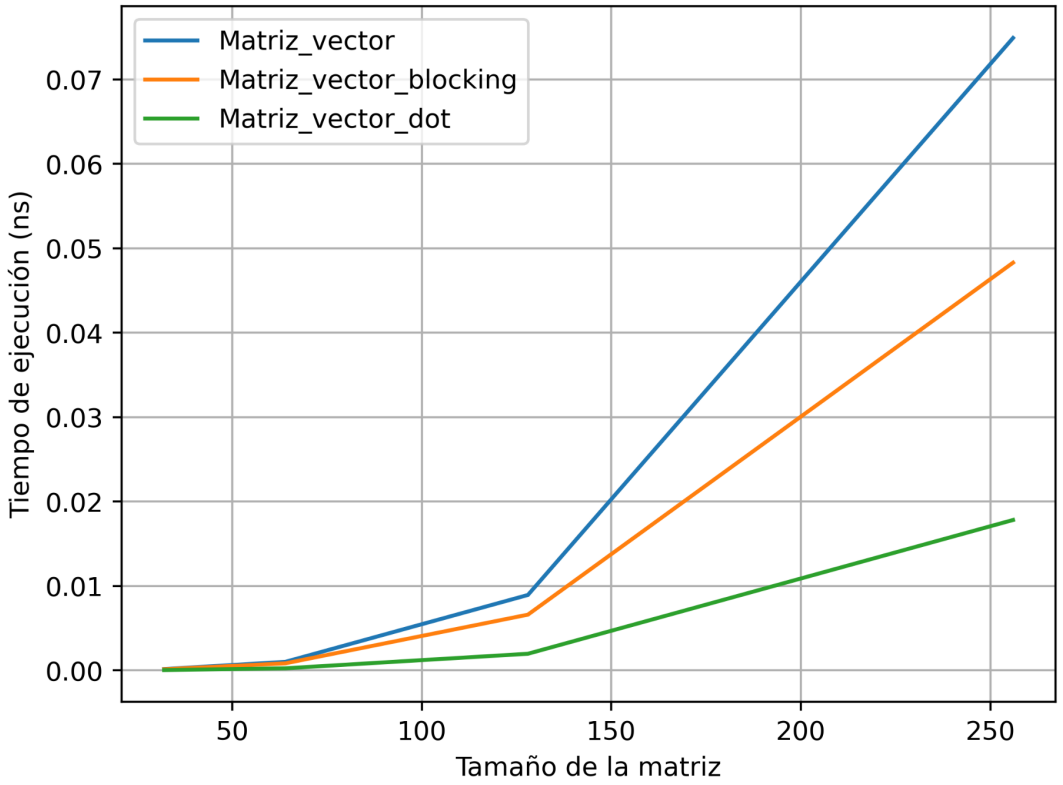
Bloque 2


Matriz_vector
Matriz_vector_blocking
Matriz_vector_dot
Tiempo de ejecución (ns)
Tamaño de la matriz
0.12
0.10
0.08
0.06
0.04
0.02
0.00
50
100
150
200
250

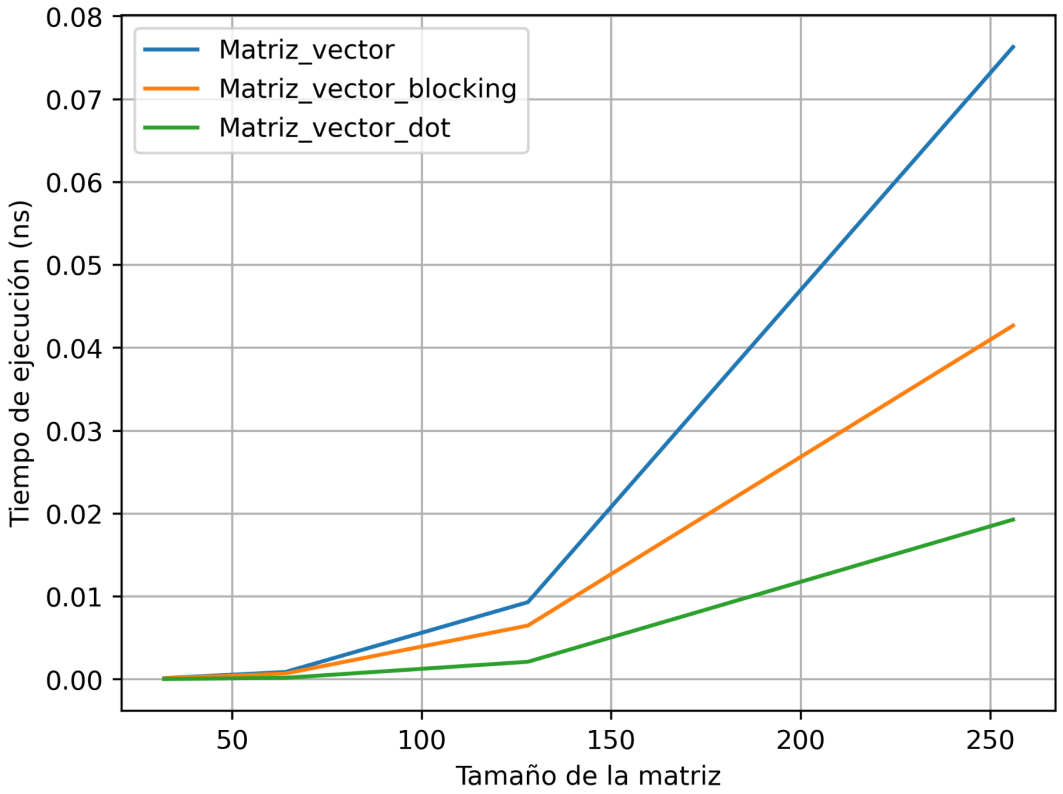
Bloque 4:


Matriz_vector
Matriz_vector_blocking
Matriz_vector_dot
Tiempo de ejecución (ns)
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
50
100
150
200
250
Tamaño de la matriz

Bloque 8:


Matriz_vector
Matriz_vector_blocking
Matriz_vector_dot
Tiempo de ejecución (ns)
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
50
100
150
200
250
Tamaño de la matriz

Bloque 16:

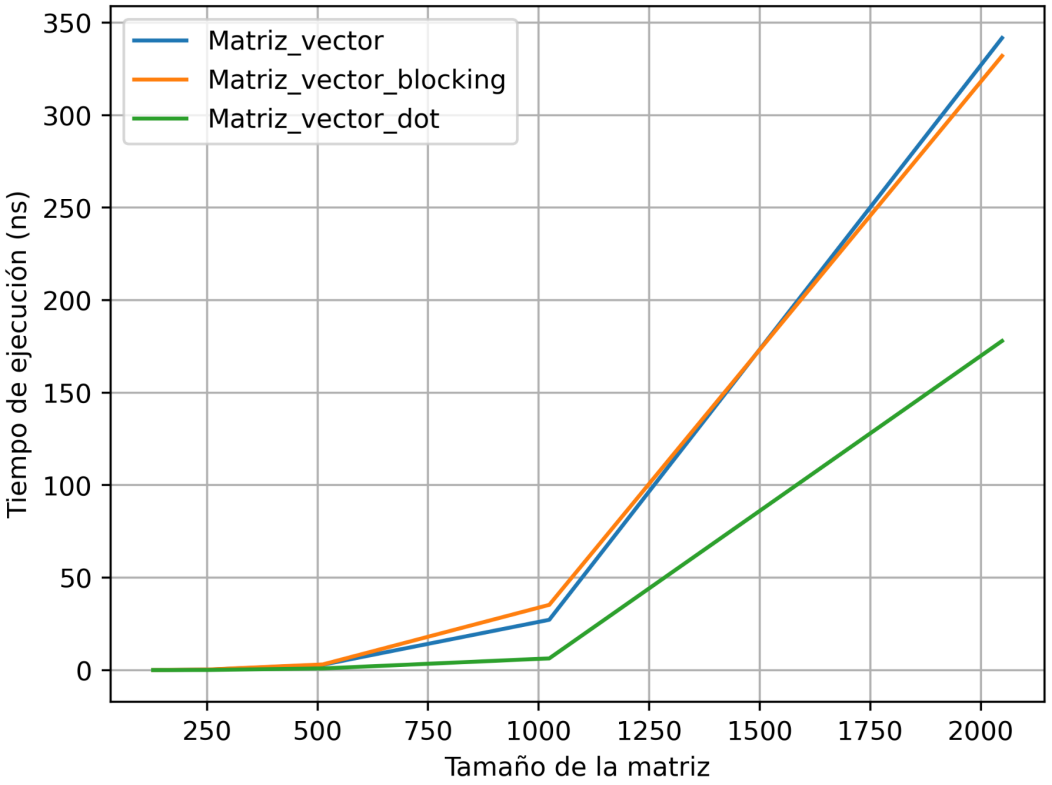


-¿A partir de qué valor de N hay una mejora?
Cuando el bloque es 2, se observa que la mejora se observa aproximadamente cuando N es mayor a 50 aproximadamente, ya que se puede ver que la función np.dot mantiene su tiempo de ejecución menor a las otras dos funciones. Para el otro caso, cuando el bloque es 4, se podría decir que la mejora también se encuentra cuando el tamaño de la matriz es mayor a 50, ya que se observa que la función np.dot posee un mejor tiempo de ejecución ahora también se observa una mejora con respecto al bloque anterior ya que la función de blocking pasa a la función sin blocking aproximadamente cerca de los 60-70 de tamaño de matriz. Y lo mismo para los demás bloques 8 y 16.

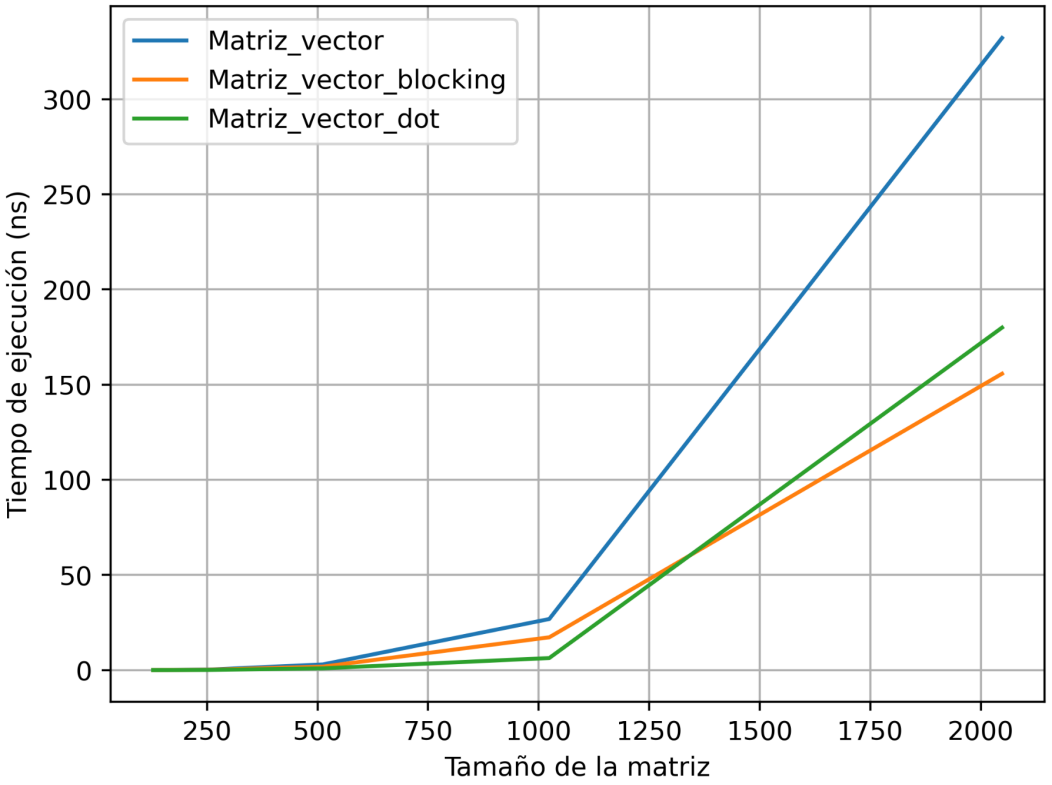
-¿Todos los tamaños de bloque representan una mejora?
No, debido a que cuando se utiliza el bloque 2 se observa que las funciones “mat_vec” y np.dot muestran un rendimiento mejor que la función “mat_vec_block”. Por otro lado, en las demás gráficas si se encuentra dicha mejora pero igual la función np.dot sigue siendo la de mejor rendimiento para la cantidad de pruebas realizadas.

E)
Iteraciones: 5

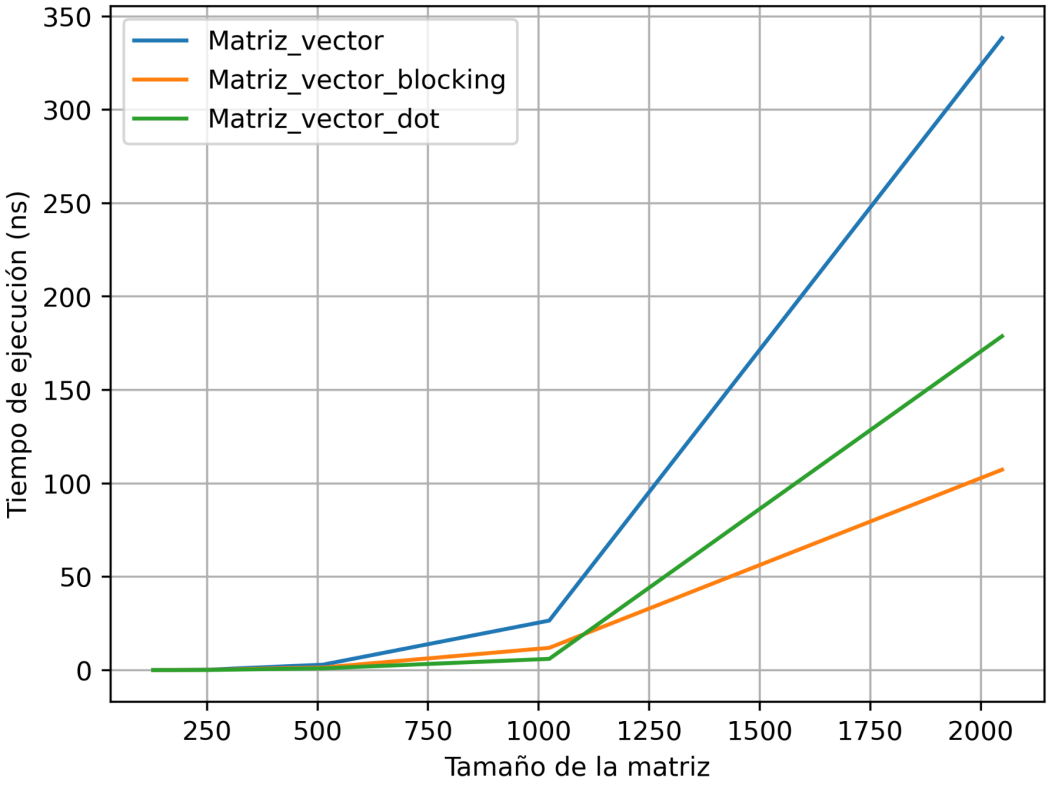
Bloque 2:


Matriz_vector
Matriz_vector_blocking
Matriz_vector_dot
Tiempo de ejecución (ns)
Tamaño de la matriz
0
50
100
150
200
250
300
350
250
500
750
1000
1250
1500
1750
2000

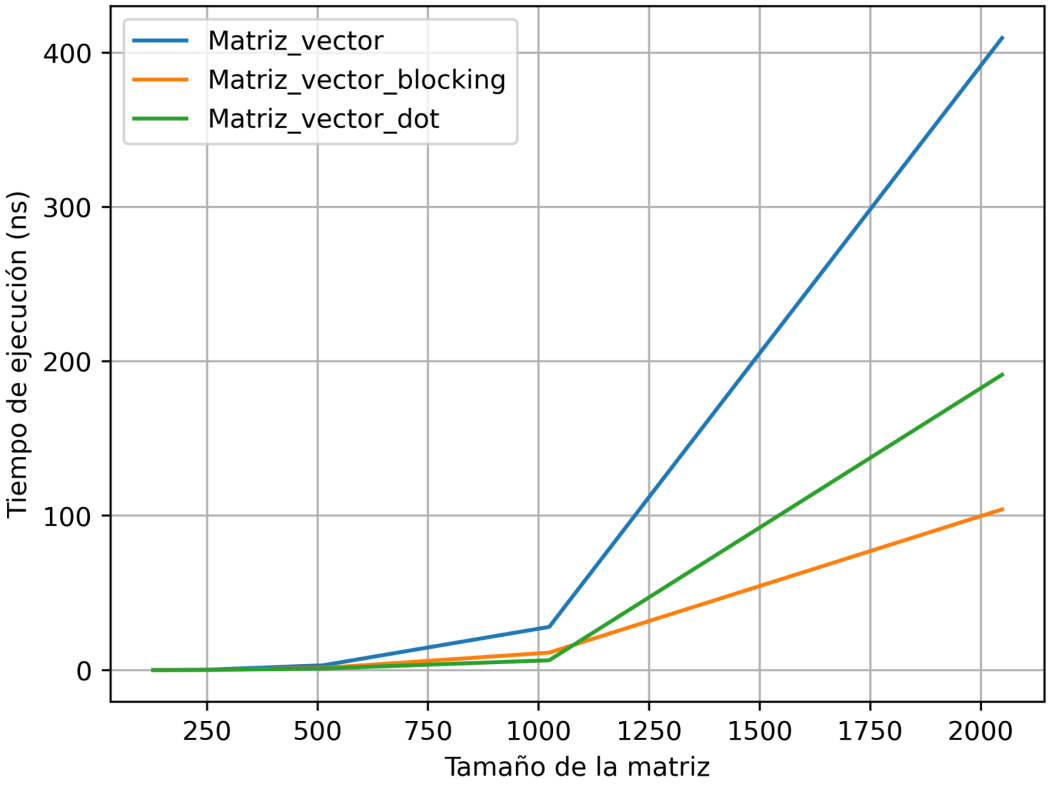
Bloque 4:


Matriz_vector
Matriz_vector_blocking
Matriz_vector_dot
Tiempo de ejecución (ns)
300
250
200
150
100
50
0
250
500
750
1000
1250
1500
1750
2000
Tamaño de la matriz

Bloque 8:


Matriz_vector
Matriz_vector_blocking
Matriz_vector_dot
Tiempo de ejecución (ns)
Tamaño de la matriz
350
300
250
200
150
100
50
0
250
500
750
1000
1250
1500
1750
2000

Bloque 16:



-¿A partir de qué valor de N hay una mejora?
Para todos los bloques, se puede observar una mejora cuando el tamaño de la matriz sobrepasa los 1000 valores. Siendo más “específico”, cuando el bloque es 2, entonces la mejora se observa pasados el tamaño de 1500, cuando el bloque es 4, la mejora se observa cuando el tamaño sobrepasa los 1300 aproximadamente, cuando el bloque es 8 y 16, se observa que la mejora se obtiene pasando los 1000 o 1150 de tamaño de matriz aproximadamente.

-¿Todos los tamaños de bloque representan una mejora?
Si, dicha mejora total es observada cuando el bloque deja de ser 2, ya que en los demás bloques se puede observar que mientras más grande sea el tamaño de la matriz entonces la función “mat_vec_block” va a ser mejor que la función np.dot. Por otro lado, cuando el bloque es 2, se puede observar una mejora cuando el tamaño del bloque es más grande. Ya que en la gráfica se muestra que la función de blocking pasa a la función sin blocking.

F)
La principal diferencia entre los algoritmos es que en el inciso D los valores de N van desde 32 hasta 256, mientras que en el inciso E los valores van desde 128 hasta 2048. Esa es la única diferencia entre los dos algoritmos. La técnica de blocking para este caso se utiliza para aumentar la localidad espacial así como la temporal, obteniendo así una posible mejora. Ahora cuando el tamaño de la matriz es similar a la del inciso D entonces la función

que es mejor es la de Python, ya que la de blocking queda como la mejor “segunda” cuando los bloques son mayores que 2, pero queda como la “peor” cuando el bloque es igual a 2, todo esto es justificable al ver los tiempos de ejecución de las gráficas en D, así como la posición de la gráficas. Por otro lado, cuando el tamaño de las gráficas es mayor o similar a la del inciso E, entonces se observa que cuando el bloque es igual a 2 la función de blocking es mejor a la de sin blocking cuando el tamaño ya es mayor a 1500, pero sigue siendo peor que la de Python. Sin embargo, cuando el bloque es mayor que 2 entonces se observa que cuando el tamaño ya es mayor a 1000 en algunos casos la función de blocking supera a la función de Python, y mientras que en otras la supera cuando es mayor a 1250 aproximadamente. Así que, la función de Python es la que mejor rendimiento posee cuando son pequeños tamaños pero cuando el tamaño es mayor a 1000 y los bloques son mayores empieza a ser superada por la función de blocking, caso contrario al ser tamaños pequeños, ya que la función de Python no deja de ser la que mejor rendimiento posee en todos los casos. Por lo que, dependiendo de los tamaños de la matriz y su uso debería elegir entre la de Python o la de blocking según sea lo que necesite.