

**UNIVERSIDAD NACIONAL DE CÓRDOBA**  
**FACULTAD DE CIENCIAS EXACTAS, FÍSICAS y NATURALES**



**Sistemas de Computación**  
**Trabajo Práctico 1**

**Autores:**

Matrícula	Apellidos y Nombres	Carrera
43523855	Marcos Raimondi	Ing. en Computación
43189293	Giuliano Palombarini	Ing. en Computación
41504806	Andrea Taborda	Ing. Electrónica

# Índice

<b>UNIVERSIDAD NACIONAL DE CÓRDOBA.....</b>	<b>1</b>
<b>Índice.....</b>	<b>2</b>
<b>Consigna.....</b>	<b>3</b>
<b>Desarrollo.....</b>	<b>4</b>
Tutorial Profiling.....	4
Con GPROF - inyección de código.....	4
Con PERF - sampling.....	6
Comparación.....	8
Conclusiones sobre el uso del tiempo de las funciones.....	9
Benchmarks.....	9
Armar una lista de benchmarks.....	9
Rendimiento para compilar el kernel de linux.....	10
Pruebas en ESP32.....	12

# Consigna

El objetivo de esta tarea es poner en práctica los conocimientos sobre performance y rendimiento de los computadores. El trabajo consta de dos partes, la primera es utilizar benchmarks de terceros para tomar decisiones de hardware y la segunda consiste en utilizar herramientas para medir la performance de nuestro código.

Responder a las siguientes preguntas y mostrar con capturas de pantalla la realización del tutorial descrito en time profiling adjuntando las conclusiones sobre el uso del tiempo de las funciones.

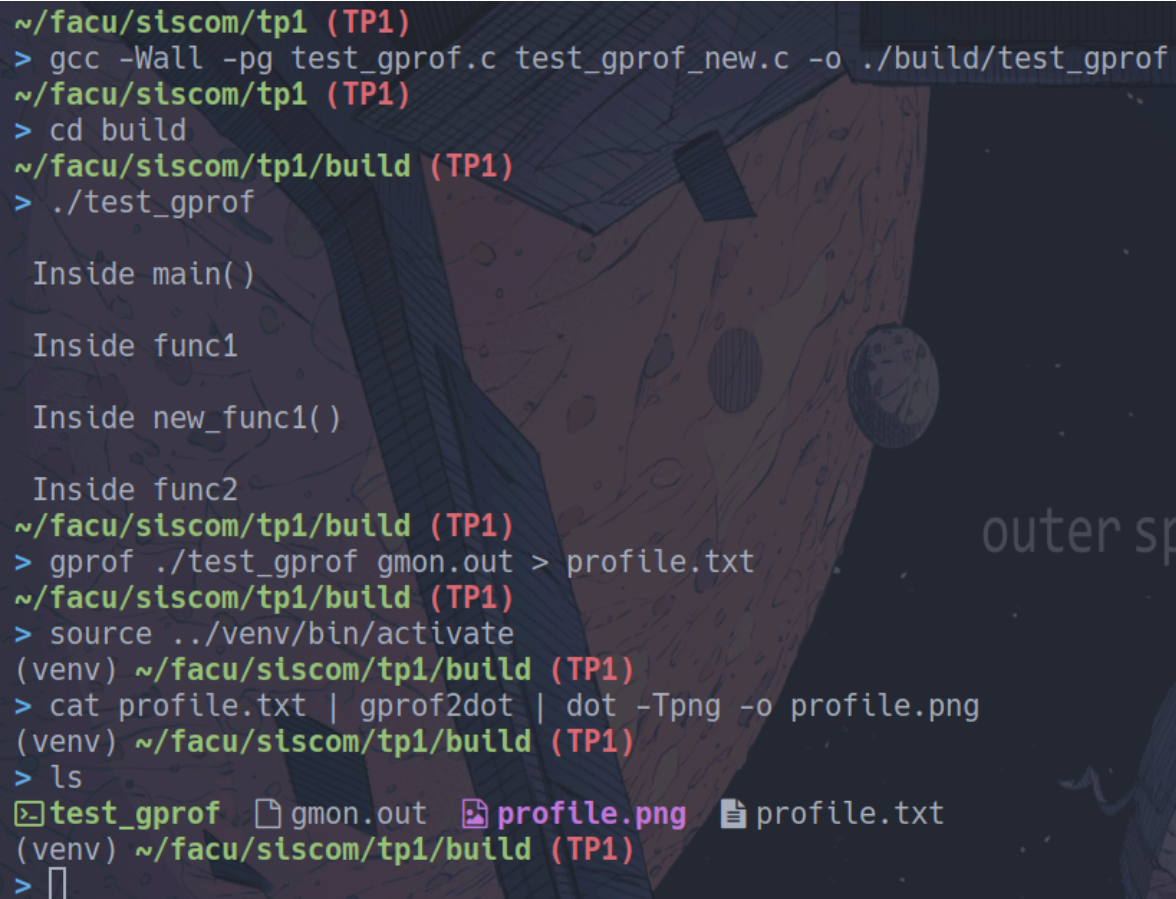
1. Armar una lista de benchmarks
  - a) ¿cuales les serían más útiles a cada uno? ¿Cuáles podrían llegar a medir mejor las tareas que ustedes realizan a diario?
  - b) Pensar en las tareas que cada uno realiza a diario y escribir en una tabla de dos entradas las tareas y que benchmark la representa mejor.
2. ¿Cuál es el rendimiento de estos procesadores para compilar el kernel de linux?
  - a) Intel Core i5-13600K
  - b) AMD Ryzen 9 5900X 12-Core
  - c) ¿Cuál es la aceleración cuando usamos un AMD Ryzen 9 7950X 16-Core?
3. Conseguir un esp32 o cualquier procesador al que se le pueda cambiar la frecuencia. Ejecutar un código que demore alrededor de 10 segundos. Puede ser un bucle for con sumas de enteros por un lado y otro con suma de floats por otro lado. ¿Qué sucede con el tiempo del programa al duplicar (variar) la frecuencia ?

# Desarrollo

## Tutorial Profiling

Con GPROF - inyección de código

Se utiliza gprof para generar el profiling y posteriormente se usa gprof2dot y dot para hacer la representación gráfica del mismo:



```
~/facu/siscom/tp1 (TP1)
> gcc -Wall -pg test_gprof.c test_gprof_new.c -o ./build/test_gprof
~/facu/siscom/tp1 (TP1)
> cd build
~/facu/siscom/tp1/build (TP1)
> ./test_gprof

Inside main()

Inside func1

Inside new_func1()

Inside func2
~/facu/siscom/tp1/build (TP1)
> gprof ./test_gprof gmon.out > profile.txt
~/facu/siscom/tp1/build (TP1)
> source ../venv/bin/activate
(venv) ~/facu/siscom/tp1/build (TP1)
> cat profile.txt | gprof2dot | dot -Tpng -o profile.png
(venv) ~/facu/siscom/tp1/build (TP1)
> ls
test_gprof  gmon.out  profile.png  profile.txt
(venv) ~/facu/siscom/tp1/build (TP1)
> 
```

- Figura 1.0, comandos

```
(venv) ~/facu/siscom/tp1/build (TP1)
> cat profile.txt
```

File: profile.txt

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
33.74	8.82	8.82	1	8.82	17.50	func1
33.20	17.50	8.68	1	8.68	8.68	new_func1
33.20	26.19	8.68	1	8.68	8.68	func2
0.12	26.22	0.03				main

Call graph

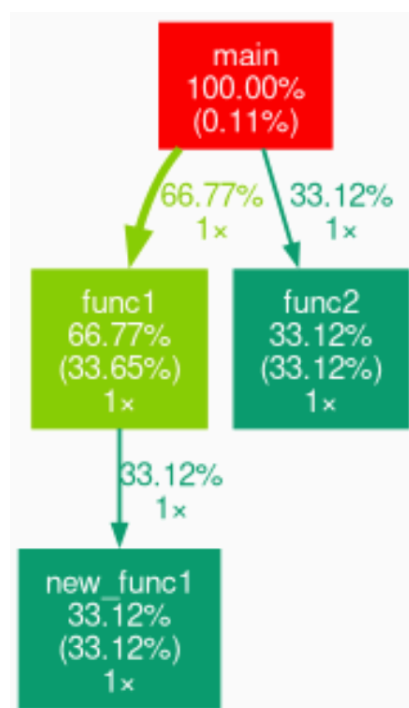
granularity: each sample hit covers 2 byte(s) for 0.04% of 26.22 seconds

index	% time	self	children	called	name
[1]	100.0	0.03	26.19		<spontaneous> main [1]
		8.82	8.68	1/1	func1 [2]
		8.68	0.00	1/1	func2 [4]
[2]	66.8	8.82	8.68	1	main [1]
		8.82	8.68	1/1	func1 [2]
		8.68	0.00	1/1	new_func1 [3]
[3]	33.1	8.68	0.00	1	func1 [2]
		8.68	0.00	1	new_func1 [3]
[4]	33.1	8.68	0.00	1	main [1]
		8.68	0.00	1	func2 [4]

Index by function name

[2] func1	[1] main
[4] func2	[3] new_func1

• Figura 1.1, profiling output



• Figura 1.2, profiling image output

## Con PERF - sampling

Compilar con información de debug (-g) ayuda a mostrar más información en los reportes acerca de las instrucciones ejecutadas, caso contrario se muestran únicamente instrucciones de assembly.

```
~/facu/siscom/tp1 (TP1)
> gcc -g test_gprof.c test_gprof_new.c -o ./build/test_gprof
~/facu/siscom/tp1 (TP1)
> cd build
~/facu/siscom/tp1/build (TP1)
> perf record ./test_gprof

Inside main()

Inside func1

Inside new_func1()

Inside func2
[ perf record: Woken up 15 times to write data ]
[ perf record: Captured and wrote 4.063 MB perf.data (106476 samples) ]
~/facu/siscom/tp1/build (TP1)
> []
```

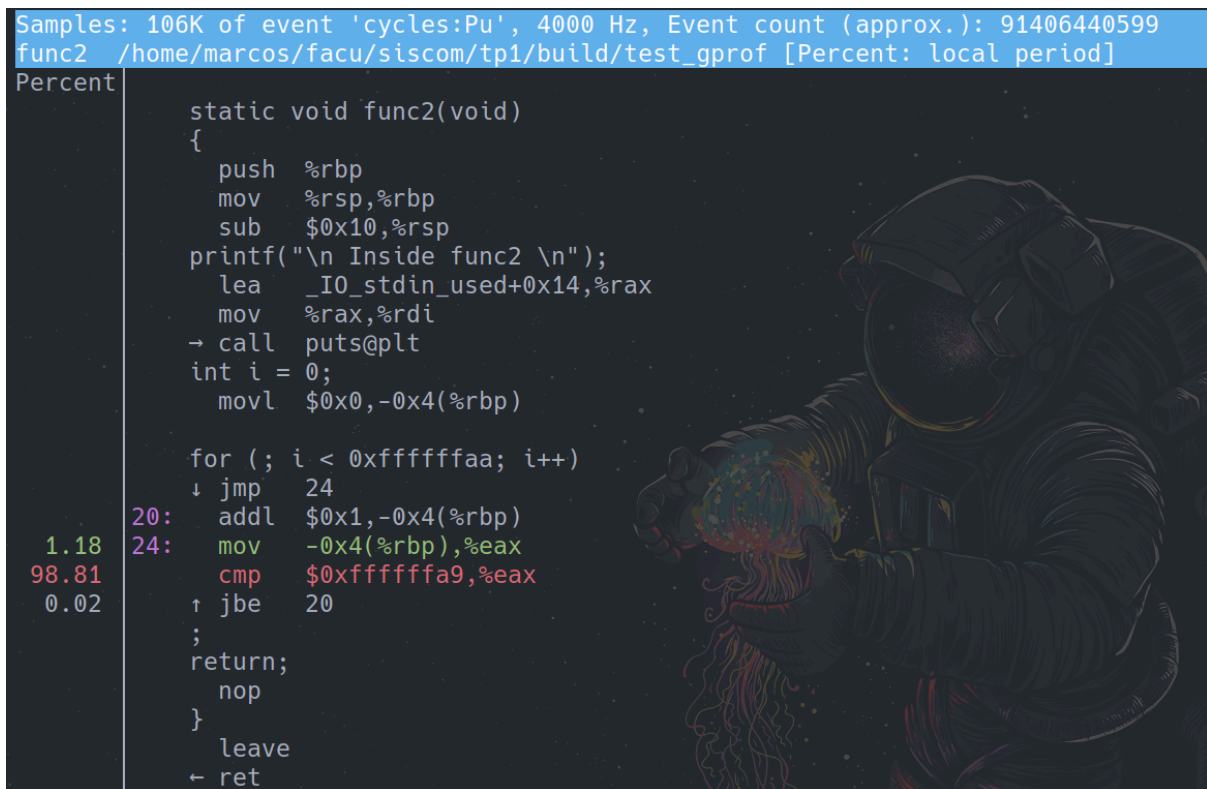
• Figura 2.0, perf record

Reporte: perf report

```
Samples: 106K of event 'cycles:Pu', Event count (approx.): 91112707293
Children    Self    Command    Shared Object    Symbol
+ 100.00%   0.00%   test_gprof  libc.so.6        [.] 0x00007d3020e1acd0
- 100.00%   0.13%   test_gprof  test_gprof       [.] main
- 99.87% main
  - 66.81% func1
    33.15% new_func1
    33.06% func2
+ 66.81%   33.65%   test_gprof  test_gprof       [.] func1
+ 33.15%   33.14%   test_gprof  test_gprof       [.] new_func1
+ 33.06%   33.06%   test_gprof  test_gprof       [.] func2
```

• Figura 2.1, perf report 1

También permite observar el perfilado en las líneas de código:

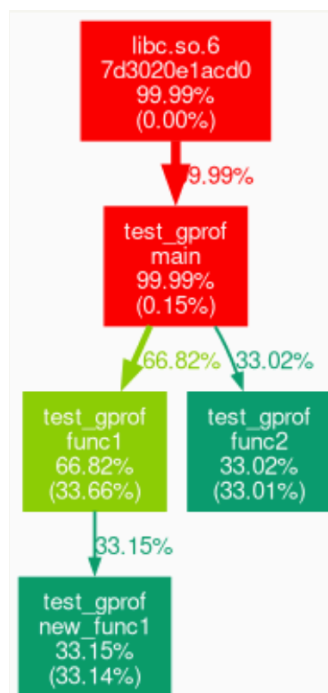


• Figura 2.2, perf report 1

Si mostramos el gráfico con gprof2dot:

***perf record -g -- ./test\_gprof***

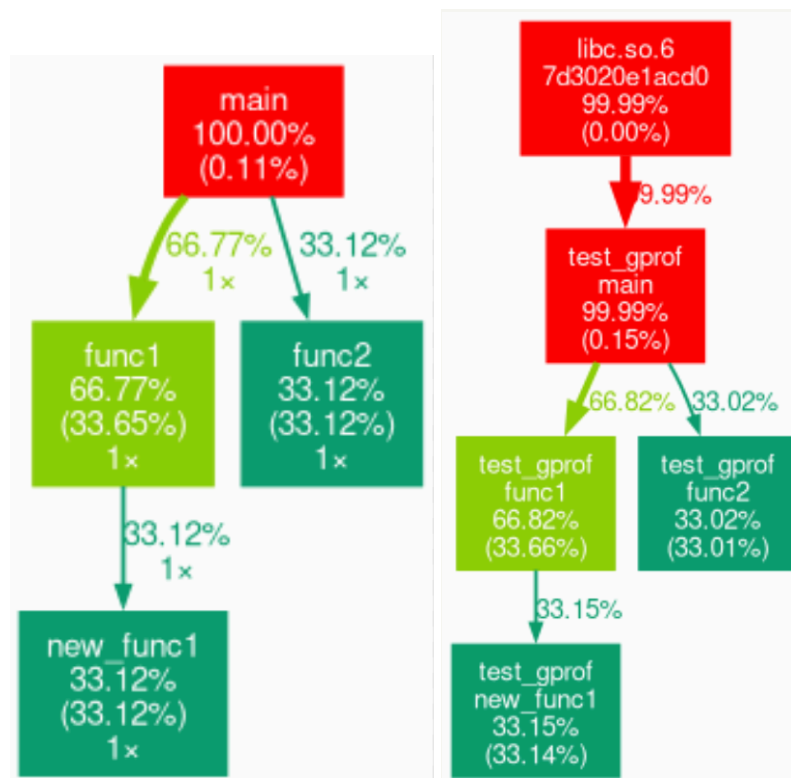
***perf script | c++filt | gprof2dot -f perf | dot -Tpng -o perf.png***



• Figura 2.3, profile image output

## Comparación

Si comparamos ambos resultados con gprof y perf, vemos que se traducen fundamentalmente al mismo resultado:



• Figura 2.4, comparativa gprof vs perf

Las diferencias principales se basan en que perf realiza muestreos cada determinado tiempo de la ejecución del programa, tomando nota de que funciones se están ejecutando y de esa forma calcula un estimado de los tiempo, mientras que con gprof se inyecta código extra que permite al perfilador obtener información más precisa acerca de los tiempos de cada bloque de código.

Dependiendo con la precisión que se desea medir se puede optar por uno o por otro, además usar gprof requiere compilar el programa con flags especiales mientras que perf se puede ejecutar incluso con el programa optimizado (dificulta su análisis pero permite estudiar el programa en su estado final).

## Conclusiones sobre el uso del tiempo de las funciones

Los resultados del profiling tienen sentido y se corresponden con el código de prueba. Cada función contiene un for idéntico, por lo que el tiempo se distribuye equitativamente entre las tres funciones. Los gráficos además muestran que al haber sido la función 1 quien llama la nueva función 1, el tiempo total de la primera abarca su propio tiempo más el de la nueva función.



## Benchmarks

Armar una lista de benchmarks

- a) ¿cuales les serían más útiles a cada uno? ¿Cuáles podrían llegar a medir mejor las tareas que ustedes realizan a diario?

Alumno	Benchmarks	Tareas diarias
Marcos	Counter-Strike 2 iPerf	<ul style="list-style-type: none"><li>• Gaming</li><li>• Transferencias de archivos por internet</li></ul>
Giuliano	Grand Theft Auto V Timed MPlayer Compilation 7-Zip Compression	<ul style="list-style-type: none"><li>• Gaming</li><li>• Reproducción de video</li><li>• Compresión/descompresión de archivos</li></ul>
Andrea	Ngspice Quantum ESPRESSO	<ul style="list-style-type: none"><li>• Simular circuitos en Spice</li><li>• Cálculos de estructuras electrónicas y modelado de materiales a nanoescala.</li></ul>

- b) Pensar en las tareas que cada uno realiza a diario y escribir en una tabla de dos entradas las tareas y que benchmark la representa mejor.

Tarea diaria	Benchmark representativo
Juegos	Counter-Strike 2 F1 22 Grand Theft Auto V
Edición de video	FFmpeg
Operaciones matemáticas Simulaciones numéricas	SciMark Algebraic Multi-Grid Benchmark
Compresión/descompresión de archivos	RAR Compression 7-Zip Compression
Transferencia de archivos por internet	iPerf
Inicio automático de varios programas que se ejecutan en Microsoft Windows	AppTimer
Reproducción de multimedia	Timed MPlayer Compilation
Operaciones comunes en Git	Git

## Rendimiento para compilar el kernel de linux

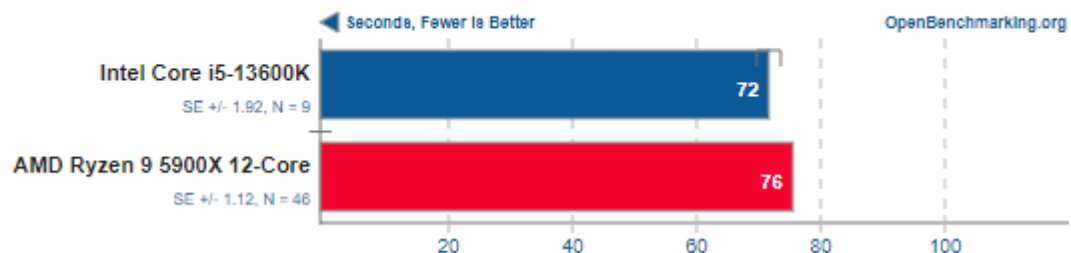
¿Cuál es el rendimiento de estos procesadores para compilar el kernel de linux?

- Intel Core i5-13600K
- AMD Ryzen 9 5900HX 12-Core
- ¿Cuál es la aceleración cuando usamos un AMD Ryzen 9 7950X 16-Core?

Utilizando los graficos proporcionados por openbenchmarking.org se observa el tiempo que demora cada procesador en la compilación del Kernel de Linux (6.1):

### Timed Linux Kernel Compilation 6.1

Build: defconfig



• Figura 3.0, kernel compilation 1

También se saca como referencia las especificaciones de los procesadores comparados:

	INTEL CORE I5-13600K	AMD RYZEN 9 5900X 12-CORE
Core Count	14	12
Thread Count	20	24
CPU Clock *	5.1 GHz	3.7 GHz
Core Family	Raptor Lake	Zen 3
First Appeared	2022	2020
Overall Percentile	68th	67th
Total OpenBenchmarking.org Results	7,357	52,686
Qualified Test Combinations	354	585
Performance Per Dollar	320	0
Affiliate Shopping Links *	Amazon	
More Details On OpenBenchmarking.org	<a href="#">Q</a>	<a href="#">Q</a>
Remove From Comparison	<a href="#">X</a>	<a href="#">X</a>

• Figura 3.1, comparativa procesadores 1

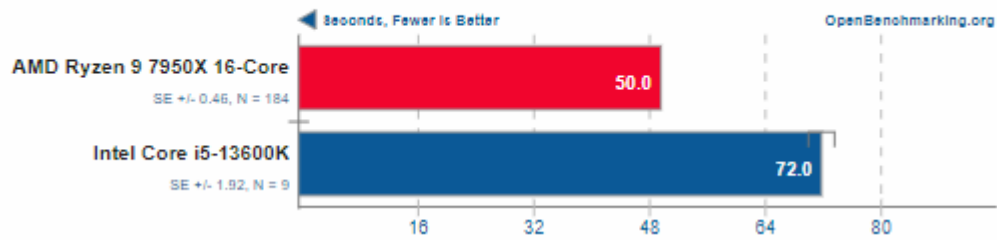
En el análisis de las características de cada procesador y las puntuaciones en las pruebas de benchmark se observa una predominancia por parte del procesador Intel Core i5-13600 K. En las mismas se ve que el procesador AMD Ryzen 9 5900X 12-Core demora aproximadamente un 5.5% más de tiempo en compilar el kernel que el rival.

En la tabla de especificaciones técnicas el Intel Core i5-13600K cuenta con más cores físicos y trabaja a una mayor frecuencia que su competencia. Pero el AMD Ryzen 9 5900X 12-Core que tiene más hilos a disposición, trabaja a una frecuencia menor y es 2 años más longevo, mostrándose muy bien en lo que respecta a tiempos de espera en la compilación del Kernel de Linux.

Si se compara ahora con el AMD Ryzen 9 7950X 16-Core:

### Timed Linux Kernel Compilation 6.1

Build: defconfig



• Figura 3.2, kernel compilation 2

	INTEL CORE I5-13600K	AMD RYZEN 9 7950X 16-CORE
Core Count	14	16
Thread Count	20	32
CPU Clock *	5.1 GHz	4.5 GHz
Core Family	Raptor Lake	Zen 4
First Appeared	2022	2022
Overall Percentile	68th	77th
Total OpenBenchmarking.org Results	7,357	45,082
Qualified Test Combinations	354	1,269
Performance Per Dollar	320	569
Affiliate Shopping Links *	<a href="#">Amazon</a>	<a href="#">Amazon</a>
More Details On OpenBenchmarking.org	<a href="#">Q</a>	<a href="#">Q</a>
Remove From Comparison	<a href="#">x</a>	<a href="#">x</a>

• Figura 3.3, comparativa procesadores 1

En este caso el procesador AMD Ryzen 9 7950X 16-Core reduce el tiempo en el que realiza la tarea de compilación, logrando reducir en un 30% el tiempo insumido en la tarea. Por otra parte el AMD Ryzen 9 7950X se diferencia por tener 2 cores más que el Intel Core i5-13600 K, 12 hilos más y trabajando a una menor frecuencia., logrando reducción del 34% en el tiempo de compilación respecto a su predecesor el AMD Ryzen 9 5900X.

$$\eta = \frac{1}{T_{\text{ejecución}}}$$

$$Speedup = \frac{\eta_{\text{mejorado}}}{\eta_{\text{original}}} = \frac{EX_{CPU \text{ Original}}}{EX_{CPU \text{ Mejorado}}} = \frac{76}{50} = 1.52$$

El speedup indica que hay una mejora considerable en la velocidad de procesamiento en el AMD Ryzen 9 7950X respecto al AMD Ryzen 9 5900X.

$$Eficiencia_{7950x} = \frac{Speedup_n}{n} = \frac{1.52}{16} = 0.095$$

$$Eficiencia_{5900x} = \frac{Speedup_n}{n} = \frac{1.52}{12} = 0.1266$$

La eficiencia baja ya que se agregaron 4 cores adicionales en la nueva revisión del AMD Ryzen 9 7950X.

## Pruebas en ESP32

Se configura una esp32 con el mismo programa a diferentes frecuencias. El programa consiste en un ciclo for en donde se ejecutan operaciones con enteros y con punto flotante.

Para una frecuencia de CPU de 80 [MHz]:

```
12:37:02.715 -> Frecuencia de la CPU: 80 MHz
12:37:02.755 -> start
12:37:13.507 -> end
```

- **Figura 4.0, prueba 80 MHz**

Se observa un tiempo de ejecución del ciclo de alrededor de 10,752 [s].

Para una frecuencia de CPU de 160 [MHz]:

```
12:40:10.000 -> Frecuencia de la CPU: 160 MHz
12:40:10.037 -> start
12:40:15.322 -> end
```

- **Figura 4.1, prueba 160 MHz**

Se observa un tiempo de ejecución del ciclo de alrededor de 5,285 [s].

Para una frecuencia de CPU de 240[MHz]:

```
12:45:18.277 -> Frecuencia de la CPU: 240 MHz
12:45:18.317 -> start
12:45:21.787 -> end
```

- **Figura 4.2, prueba 240 MHz**

Se observa un tiempo de ejecución del ciclo de alrededor de 3,470 [s].

Se puede observar que a medida que se aumenta la frecuencia de la CPU, disminuye proporcionalmente el tiempo de ejecución. Ante un mismo programa, con las mismas instrucciones, aumentar la frecuencia implica más ciclos de clock por segundo. La cantidad de ciclos por instrucción permanece constante por lo que el tiempo total disminuye.

Al aumentar dos veces la frecuencia de la CPU (80 -> 160 MHz) el tiempo disminuye a la mitad (10 -> 5 s). Cuando se aumenta tres veces (80 -> 240 MHz), el tiempo disminuye a un tercio del tiempo original (10 -> 3 s).