Práctico #1 Rendimientos

Nombres: Berra Facundo, Espejo Alejandro, Quaglia Mateo

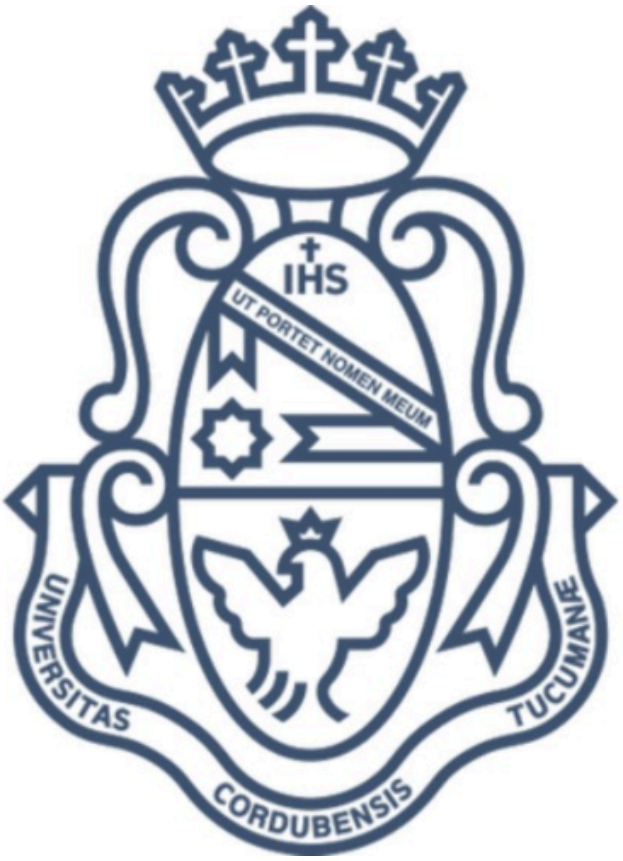
Nombre del grupo: Debian Enjoyers

Facultad de Ciencias Exactas Físicas y Naturales

Sistemas de Computación

Profesores: Jorge Javier Alejandro, Solinas Miguel Angel

31/03/2025



Introducción

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.

Paso 1: creación de perfiles habilitada durante la compilación

En este primer paso habilitamos la generación de perfiles al compilar nuestro código y esto lo logramos utilizando la opción -pg de GCC como dice el tutorial, esta opción añade código extra al binario que generamos que permitirá escribir la información de perfiles para la herramienta gprof y que esta analice el rendimiento de cada equipo que lo corra

para eso ejecutamos el comando ya teniendo creados los archivos .c propuestos por el tutorial:

```
$ gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
```

Paso 2: Ejecución del código

En este paso ejecutamos el archivo binario test_gprof que generamos en el paso 1, con esto hacemos que durante la ejecución se recopilen los datos de los perfiles de rendimiento que se usaran en los pasos posteriores y así obtener los resultados propuestos, entonces primero listamos los archivos en el directorio de trabajo para verificar que el binario está presente:

```
$ ls  test_gprof  test_gprof.c  test_gprof_new.c
$ ./test_gprof
```

La salida muestra la ejecución del programa, indicando los diferentes puntos del flujo del código donde se invocan las funciones. En este caso, observamos las siguientes líneas en la salida:

```
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ gcc -Wall -pg t
est_gprof.c test_gprof_new.c -o test_gprof
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls
test_gprof  test_gprof.c  test_gprof_new.c
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls test_gprof
test_gprof
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls test_gprof.c
test_gprof.c
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls test_gprof_n
ew.c
test_gprof_new.c
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ./test_gprof

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$
```

Después de la ejecución volvemos a listar los archivos en el directorio de trabajo y notamos la aparición del nuevo archivos de salida .out

```
$ ls  gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
```

El archivo generado gmon.out lo que hace es que obtiene toda la información de los perfiles generada durante la ejecución del programa y lo usamos para el analisis de rendimiento con gprof.

```
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls gmon.out
gmon.out
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls test_gprof
test_gprof
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls test_gprof.c
test_gprof.c
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls test_gprof_n
ew.c
test_gprof_new.c
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$
```

Paso 3: Ejecución de la herramienta gprof

En este paso usamos la herramienta gprof para analizar la información del perfil que generamos anteriormente con la ejecución del programa en el paso 2 y con esto podremos hacer un análisis de los datos y métricas obtenidos como el tiempo de ejecución de las funciones 1 y 2 además de identificar los posibles puntos críticos de rendimiento,
para esto usamos

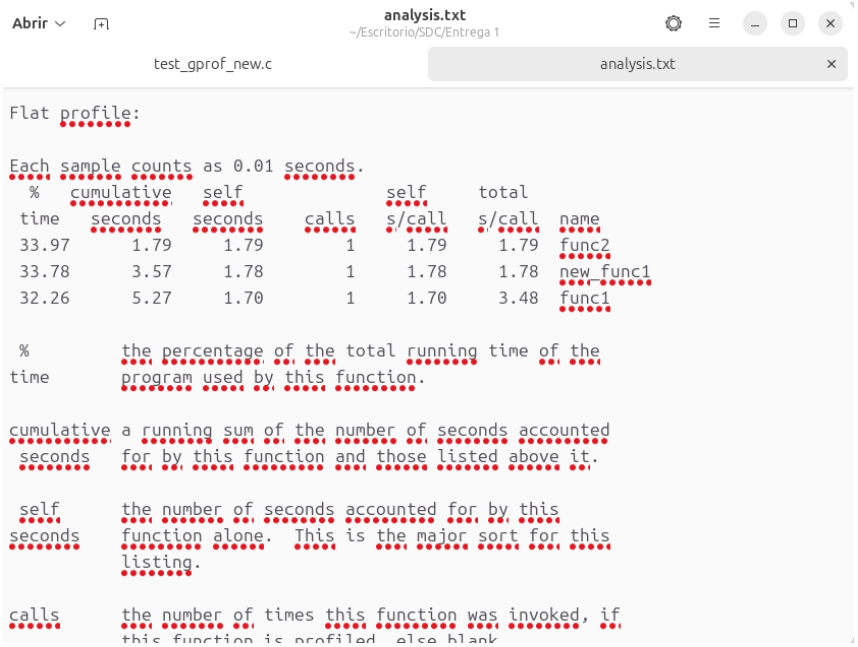
```
$ gprof test_gprof gmon.out > analysis.txt
```

el cual lo que hace es utilizar gprof del paso 1, ejecuta el test_gprof que generamos en el paso 2 y redirige el análisis que obtiene a una salida “analysis.txt”
luego con comando siguiente generamos el archivo .txt que contiene el resultado del análisis de performance del perfil, para luego proceder a examinar los resultados y entender mejor el desempeño de las funciones del programa en nuestra computadora

```
$ analysis.txt  gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
```

```
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ gprof test_gpro
f gmon.out > analysis.txt
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls analysis.txt
analysis.txt
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls gmon.out
gmon.out
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls test_gprof
test_gprof
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls test_gprof.c
test_gprof.c
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ ls test_gprof_n
ew.c
test_gprof_new.c
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$
```

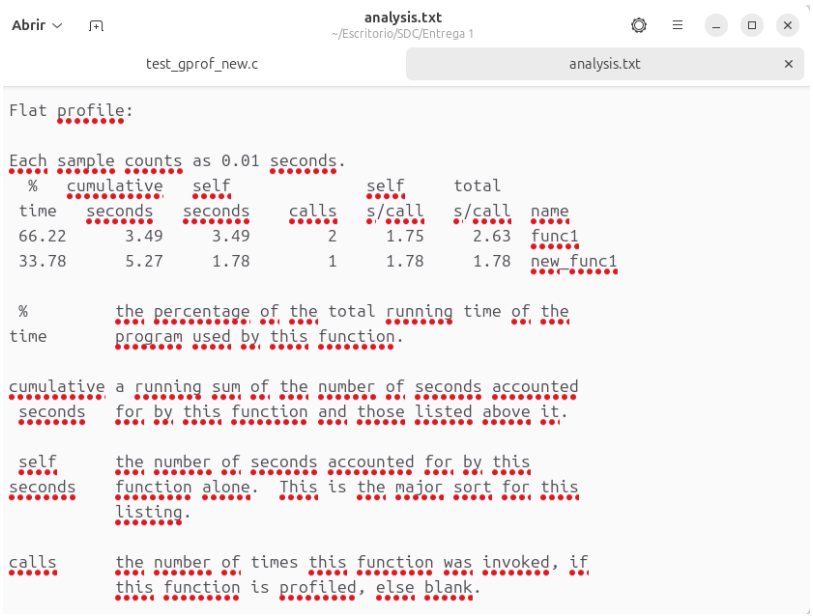
Verificamos dentro del directorio que se creó el archivo “analysis.txt” el cual contiene la cantidad de tiempos de ejecución de las funciones en una PC de un integrante, siendo que va variando la ejecución en las distintas computadoras que cada integrante del grupo tiene



Ya habiendo obtenido el txt generado, procedemos a modificar las formas de ejecución del mismo.
Ejecutando el siguiente código:

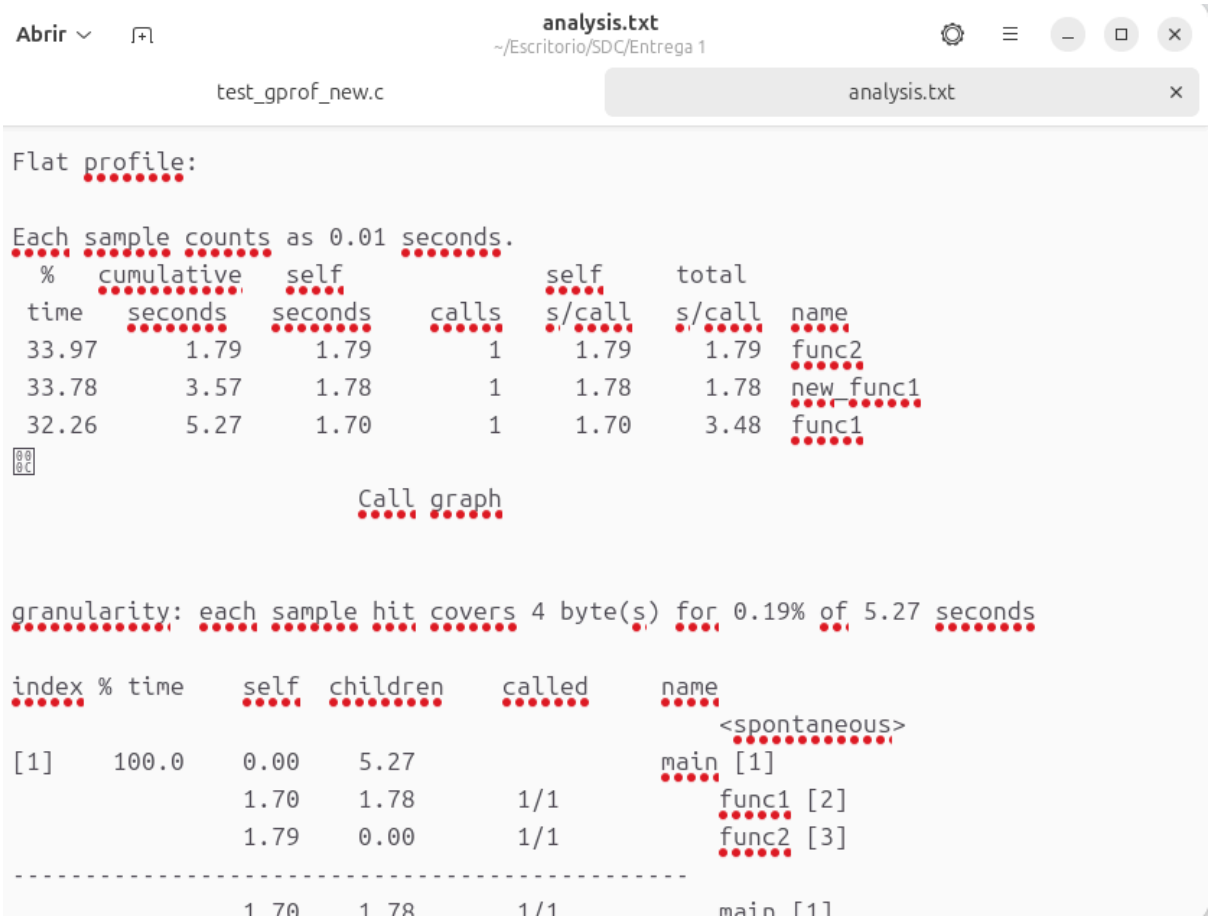
```
$ gprof -a test_gprof gmon.out > analysis.txt
```

Suprimimos la impresión de funciones declaradas estáticamente, obteniendo el siguiente analysis.txt:



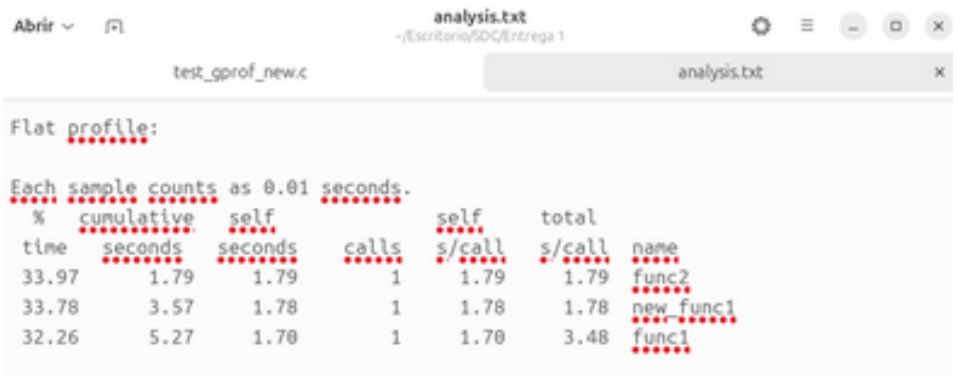
Posteriormente ejecutamos el siguiente código para eliminar todo el texto detallado que se encontraba dentro del “analysis.txt”:

```
$ gprof -b test_gprof gmon.out > analysis.txt
```



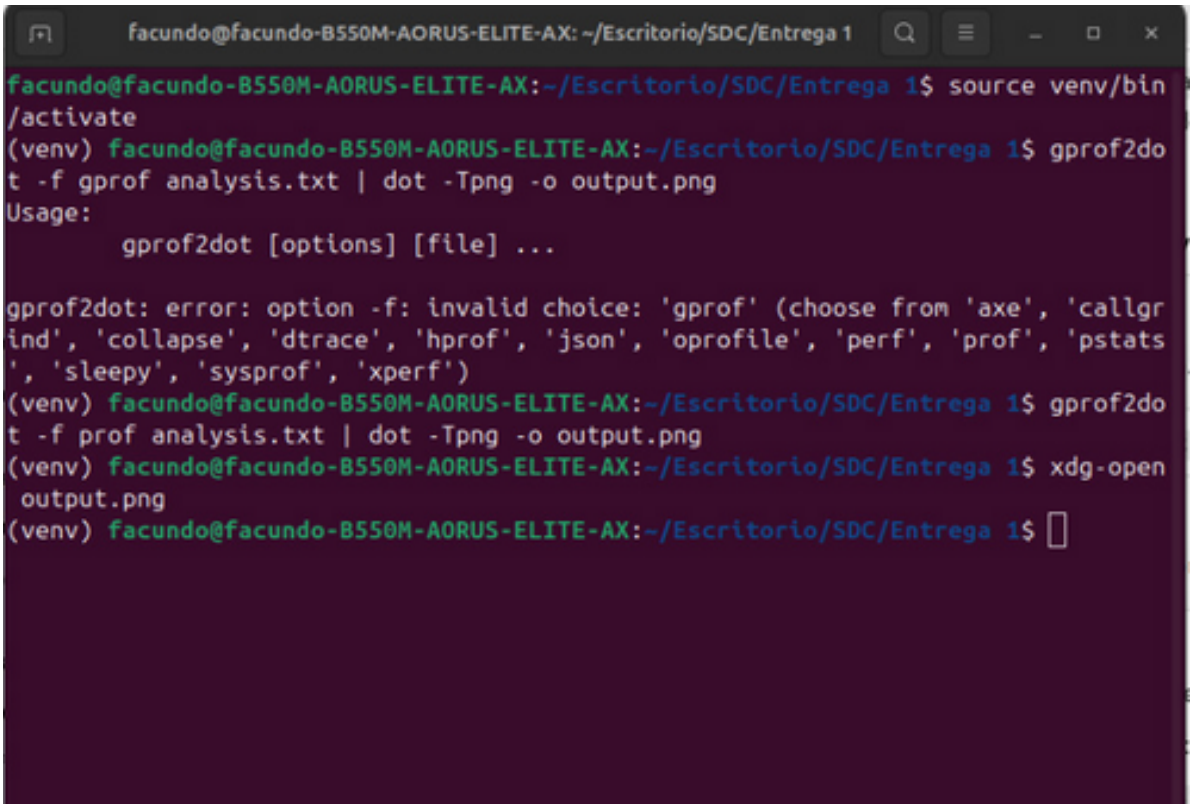
Y para mostrar solo el perfil del “analysis.txt” ejecutamos:

```
$ gprof -p -b test_gprof gmon.out > analysis.txt
```

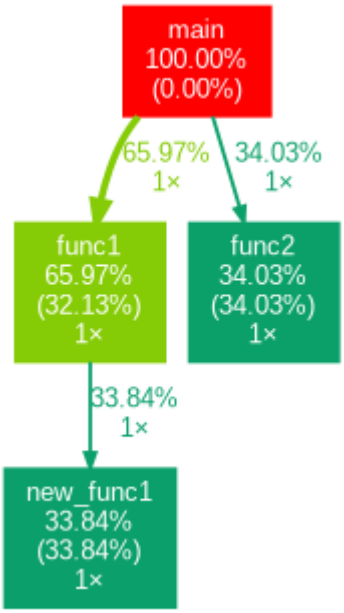


Paso 4: Generación de un Gráfico

Utilizamos “gprof2dot” que es la herramienta que nos permite realizar un grafico por medio del análisis generado mediante el gprof, por esto debimos implementarlo por medio de venv, un entorno virtual de python



Y mediante el “xdg-open output.png” abrimos el gráfico generado por “gprof2dot” mencionado anteriormente:



Paso 5: Profiling con linux

Para estudiar el rendimiento de nuestro sistema utilizamos la técnica de profiling en linux con perf, ejecutando el siguiente comando:

```
$ sudo perf record ./test_gprof
```

```
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ sudo perf recor
d ./test_gprof

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
[ perf record: Woken up 3 times to write data ]
[ perf record: Captured and wrote 0,816 MB perf.data (21069 samples) ]
facundo@facundo-B550M-AORUS-ELITE-AX:~/Escritorio/SDC/Entrega 1$ sudo perf repor
t
```

El cual posteriormente si queríamos observar el rendimiento global del sistema debimos ejecutar:

```
$ sudo perf record
```

Mostrándonos finalmente por consola:

facundo@facundo-B550M-AORUS-ELITE-AX: ~/Escritorio/SDC/Entrega 1

```
Samples: 21K of event 'cycles:P', Event count (approx.): 24184224908
Overhead  Command      Shared Object          Symbol
  33,79%  test_gprof   test_gprof             [.] func2
  33,72%  test_gprof   test_gprof             [.] new_func1
  32,02%  test_gprof   test_gprof             [.] func1
   0,13%  test_gprof   test_gprof             [.] main
   0,05%  test_gprof   [kernel.kallsyms]      [k] nohz_balancer_kick
   0,04%  test_gprof   [kernel.kallsyms]      [k] sched_use_asym_prio
   0,02%  test_gprof   [kernel.kallsyms]      [k] srso_alias_safe_ret
   0,02%  test_gprof   [kernel.kallsyms]      [k] srso_alias_return_thunk
   0,01%  test_gprof   [kernel.kallsyms]      [k] clockevents_program_event
   0,01%  test_gprof   [kernel.kallsyms]      [k] perf_event_update_userpage
   0,01%  test_gprof   [kernel.kallsyms]      [k] native_irq_return_iret
   0,01%  test_gprof   [kernel.kallsyms]      [k] __hrtimer_run_queues
   0,01%  test_gprof   [kernel.kallsyms]      [k] hrtimer_interrupt
   0,01%  test_gprof   [kernel.kallsyms]      [k] __irqentry_text_end
   0,01%  test_gprof   [kernel.kallsyms]      [k] tick_nohz_handler
   0,00%  test_gprof   [kernel.kallsyms]      [k] native_sched_clock
   0,00%  test_gprof   [kernel.kallsyms]      [k] __get_user_nocheck_8
   0,00%  test_gprof   [kernel.kallsyms]      [k] lapic_next_event
   0,00%  test_gprof   [kernel.kallsyms]      [k] __memcg_slab_post_alloc_hook
   0,00%  test_gprof   [kernel.kallsyms]      [k] __run_timers
   0,00%  test_gprof   [kernel.kallsyms]      [k] arch_asym_cpu_priority
Cannot load tips.txt file, please install perf!
```

Conclusiones y Respuestas adicionales

A partir de este trabajo de laboratorio, con cada integrante del grupo debatimos y nos pusimos de acuerdo sobre qué hasta donde se nos haría necesario la utilización de un microprocesador, puesto que en nuestro caso no necesitamos lo más moderno del mercado puesto que nuestros requerimientos no son tan altos, pero tampoco tener una PC obsoleta. Debido a esto consideramos que para realizar las tareas que llevamos a cabo diariamente, necesitamos aproximadamente a partir de un "AMD Ryzen 5 5600 6-Core" pasando la gama baja, que con esto debería ser suficiente, obviamente destacando que si tenemos un procesador de mayor gama obtendremos mejores resultados.
Teniendo en cuenta que la aceleración se calcula como :
Aceleración = Tiempo del procesador base / Tiempo del procesador comparado
entonces podemos calcular por ejemplo

Aceleración = Tiempo Intel Core i5-13600K / Tiempo AMD Ryzen 9 7950 X

Obtenemos según la tabla y Analyze Test Configuration en forma de
"pts/build-linux-kernel-1.16.x - Build: defconfig"
obtenemos una aceleración de:
A = 83 +/- 3 s / 53 +/- 3 s
A = 1,566 ±0.11 VECES
Esto significa que el AMD Ryzen 9 7950X es aproximadamente 1.57 veces más rápido que el Intel Core i5-13600K en esta tarea, con una incertidumbre de ±0.11

Pero ahora veamos : *cual de ellos hace un uso más eficiente de la cantidad de núcleos que tiene? y cuál es más eficiente en términos de costo ?*

Uso eficiente de los núcleos

Intel Core i5-13600K:
Este procesador tiene una arquitectura híbrida con 14 núcleos (6 de rendimiento y 8 de eficiencia) y 20 hilos. Su diseño está optimizado para tareas que requieren un balance entre rendimiento y consumo energético. Sin embargo, debido a que tiene menos núcleos de alto rendimiento, su capacidad para manejar tareas altamente paralelizadas es más limitada en comparación con el Ryzen 9 7950X

AMD Ryzen 9 7950X:

Con 16 núcleos y 32 hilos, este procesador está diseñado para maximizar el rendimiento en tareas altamente paralelizadas, como la compilación de software o el renderizado. Su arquitectura Zen 4 y su frecuencia base de 4.5 GHz (con boost hasta 5.7 GHz) lo hacen más eficiente en el uso de todos sus núcleos para tareas intensivas

Por eso concluimos que el Ryzen 9 7950X hace un uso más eficiente de su cantidad de núcleos en tareas que aprovechan el paralelismo, mientras que el i5-13600K es más eficiente para tareas menos intensivas o de uso general.

Eficiencia en términos de costo

Intel Core i5-13600K:
Tiene un precio de lanzamiento de aproximadamente $319 USD, lo que lo posiciona como una opción de gama media con una excelente relación precio-rendimiento para tareas generales y gaming.

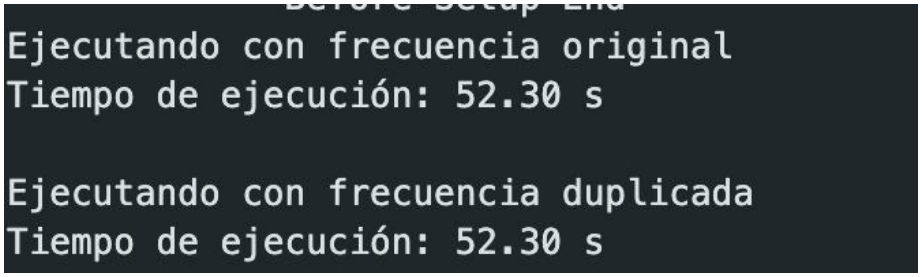
AMD Ryzen 9 7950X:
Su precio inicial fue de $699 USD. A pesar de su alto costo, ofrece un rendimiento excepcional en tareas profesionales y de alto rendimiento.

El procesador con un buen balance entre costo y rendimiento para tareas generales es el i5-13600K es más eficiente en términos de costo puesto que este cuesta menos de la mitad y tiene una performance aceptable.

Variación
de Frecuencia en una ESP32

¿Qué sucede con el tiempo del programa al duplicar (variar) la frecuencia ?

```
#include "Arduino.h"


void medirTiempo() {
    uint32_t start = micros();

    // Simulación de un proceso que toma aproximadamente 10 segundos
    volatile int sum = 0;
    for (long i = 0; i < 500000000; i++) {
        sum += i;
        if (i % 10000000 == 0) {
            delay(1);  // Pequeña pausa para evitar watchdog reset
        }
    }

    uint32_t end = micros();
    Serial.print("Tiempo de ejecución: ");
    Serial.print((end - start) / 1000000.0);
    Serial.println(" s");
}

void setup() {
    Serial.begin(115200);
    delay(2000);

    Serial.println("Ejecutando con frecuencia original");
    medirTiempo();

    // Duplicar la frecuencia de la CPU
    setCpuFrequencyMhz(240); // Máxima frecuencia para la ESP32
    Serial.println("\nEjecutando con frecuencia duplicada");
    medirTiempo();
}
```



Ejecutando el Código visto en la imagen, se puede observar que aumentar la frecuencia del CPU no siempre acelera la ejecución de un proceso, especialmente cuando el rendimiento está limitado por otros factores, como la velocidad de la memoria.

En este caso, el tiempo de ejecución se mantiene constante a pesar de aumentar la frecuencia, lo que indica que el cuello de botella no es el procesador, sino la velocidad de acceso a la memoria. Esto es común en sistemas embebidos como la ESP32, donde la memoria externa (PSRAM) o los periféricos pueden ser más lentos que la CPU y, por lo tanto, limitar el rendimiento general del sistema.