

Trabajo Práctico #1: Rendimiento

Introducción

En el siguiente informe se desarrolla el trabajo práctico 1, correspondiente a la asignatura Sistemas de Computación. El objetivo principal es analizar el rendimiento de los computadores, de forma de poder cuantificar el mismo. Para ello, se utilizan benchmarks. Para medir las performances de los computadores de los integrantes del grupo, se correrán algunos de ellos, en este caso serán:

- Benchmarks de terceros para tomar decisiones de hardware
- Herramientas (GPROF y perf) para medir performance de código

Los resultados de time profiling realizados serán analizados y explicados.

Finalmente, para cerrar el concepto de rendimiento y su cuantificación, se analizará qué pasa al overclockear un microcontrolador Raspberry Pi Pico que ejecuta un simple programa de sumas de números.

Marco teórico

Benchmarking



Figura 1. PassMark benchmark UI.

Al evaluar la eficacia de un ordenador, surge la cuestión de cómo medir su rendimiento. Esto implica la evaluación de cada uno de sus componentes para identificar áreas de mejora, pero especialmente se enfoca en optimizar el rendimiento global del sistema. En consecuencia, el indicador más confiable suele ser el tiempo de ejecución.

Sin embargo, surge la interrogante sobre qué estándar utilizar para medir este tiempo de ejecución. Para garantizar un rendimiento óptimo del sistema, las pruebas deben realizarse con programas reales o similares a los que se ejecutarán en condiciones de operación normales. Posteriormente, se comparan los tiempos resultantes con los de computadoras similares para determinar si se está alcanzando un rendimiento competitivo en el mercado actual. Para comparar el rendimiento entre distintas máquinas, se emplea un benchmark como referencia.

Un benchmark es un conjunto de programas de prueba o programas reales que sirven para medir el rendimiento de un componente concreto o de una computadora en su conjunto, mediante la comparación de los tiempos de ejecución obtenidos de esos programas de prueba con respecto a otras máquinas similares.

Tipos de benchmarks

Existen diferentes tipos de benchmarks, dependiendo de las características de la computadora a evaluar y de la sofisticación del mecanismo utilizado.

- Programas de juguete o microbenchmark: consisten en pequeños fragmentos de código con entre 10 y 100 líneas de código que se utilizan para medir una determinada característica de la computadora. Ejemplos de esto son: Java Micro Benchmark, Puzzle, Quicksort, criba de Eratóstenes.
- Benchmarks sintéticos: se trata de programas de prueba que simulan programas reales en carga de trabajo y reparto de instrucciones, sirven más que nada para medir el rendimiento de componentes concretos o del PC en general. Ejemplos son: Whetstone, Dhrystone
- Benchmarks de kernel: consisten en un fragmento de código extraído de un programa real. La parte escogida es la más representativa, y por tanto, la que más influye en el rendimiento del sistema para ese determinado software (como puede ser por ejemplo el kernel de Linux).
- Programas reales: son los más utilizados hoy en día. Son programas reales que son ejecutados con un conjunto de datos reducido para no alargar su ejecución. Se tienen como ejemplos los de la familia SPEC (clasificados en SPECint y SPECfp, según operen con números enteros o reales).
- Otros: existen numerosos tipos de benchmarks para probar características y componentes específicos de un ordenador, como pueden ser benchmarks de entrada/salida, de bases de datos, paralelos. Ejemplos son: Linpack, Livermoore, NAS y PARSEC.

Familia SPEC

SPEC (Standard Performance Evaluation Corporation, por sus siglas en inglés) es una asociación que fue creada por los mayores fabricantes de computadoras del mundo (IBM, HP, INTEL, SUN, etc.) Se creó con el objetivo de definir unos benchmarks que consistieran en programas reales y que pudieran ser utilizados como referentes o estándares para todas las marcas. Los benchmarks SPEC sirven para medir el rendimiento de componentes concretos o de microprocesadores en general.



Figura 2. Logo de SPEC.

Se muestra a continuación la ejecución de algunos benchmarks SPECfp2006 en procesadores.

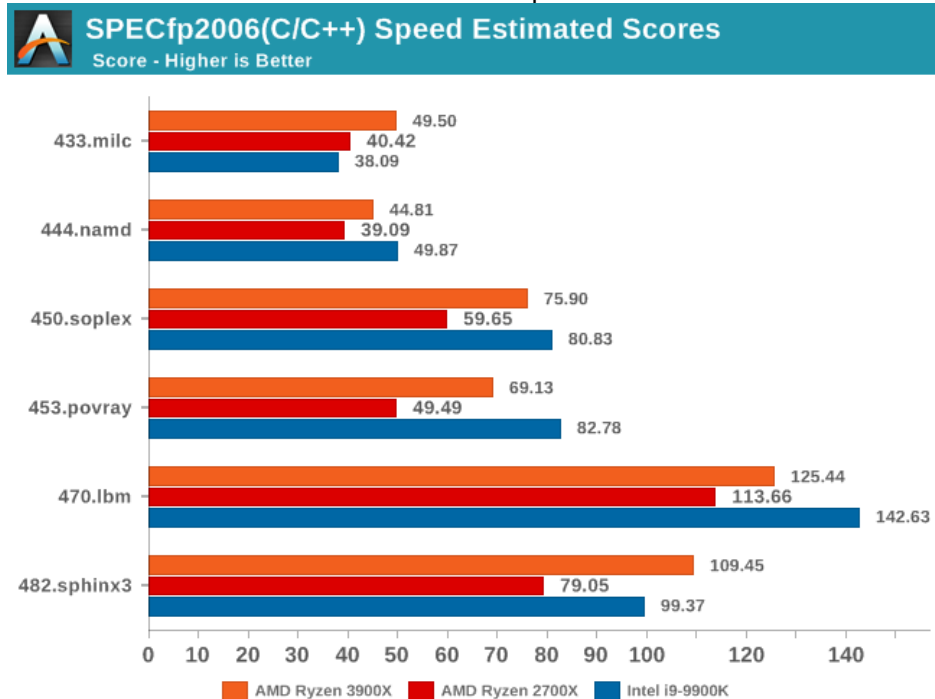


Figura 3. Ejemplo de resultados del benchmark SPECfp2006 (C/C++).

Rendimiento

Definición

El rendimiento de una computadora se refiere a la cantidad de trabajo que puede realizar un sistema informático en un período de tiempo dado. Este rendimiento puede variar según el contexto y puede incluir uno o más de los siguientes aspectos:

- Tiempo de respuesta reducido para una tarea específica.
- Alto throughput, es decir, una alta tasa de procesamiento de trabajo.
- Utilización baja de recursos computacionales.
- Alta disponibilidad del sistema informático o de la aplicación.
- Capacidad rápida (o altamente eficiente) de compresión y descompresión de datos.
- Amplio ancho de banda para la transferencia de datos.
- Tiempos cortos de transmisión de datos.

Ingeniería de rendimiento

La ingeniería de rendimiento, dentro del ámbito de la ingeniería de sistemas, engloba una serie de funciones, habilidades, actividades, prácticas, herramientas y productos aplicados en todas las fases del ciclo de vida del desarrollo de sistemas. Su objetivo es garantizar que una solución sea diseñada, implementada y operativa de manera que cumpla con los requisitos de rendimiento definidos para dicha solución.

Esta disciplina implica constantemente la consideración de compensaciones entre diferentes tipos de rendimiento. En ocasiones, un diseñador de CPU puede lograr mejorar el rendimiento general de una CPU mediante la optimización de un aspecto específico del rendimiento, sin comprometer el rendimiento en otras áreas. Por ejemplo, mediante la utilización de transistores más avanzados y rápidos.

No obstante, en algunas situaciones, el intento de maximizar un aspecto del rendimiento puede resultar en un rendimiento general inferior de la CPU. Esto puede ocurrir cuando se sacrifican otros aspectos

importantes para obtener resultados impresionantes en uno específico, como en el caso de la frecuencia del reloj del chip.

Mito del megahercio

Un error común, conocido coloquialmente como el "mito del megahercio", sugiere que un microprocesador será más rápido que otro si su frecuencia es mayor. Sin embargo, esta afirmación no es necesariamente precisa, como señala la fuente Wikipedia. Para conocer el rendimiento real, se deben considerar varios parámetros que van más allá de la simple frecuencia de reloj.

Por ejemplo, dentro del ámbito de la arquitectura informática, existen dos clases principales: CISC (Complex Instruction Set Computing) y RISC (Reduced Instruction Set Computing). Las instrucciones RISC tienden a ejecutarse más rápido en promedio, mientras que las instrucciones CISC son más complejas. Por lo tanto, un procesador CISC funcionando a una frecuencia más baja podría ofrecer una eficiencia superior.

Incluso dentro de un tipo específico de arquitectura, como al comparar dos procesadores x86 a la misma frecuencia, factores como el diseño de transistores, la litografía, la estructura interna incluyendo unidades y elementos, así como los conjuntos de instrucciones, ejercen una influencia más significativa en el rendimiento que la frecuencia de reloj por sí sola. Por ejemplo, un procesador de un solo núcleo de 2001 funcionando a 2 GHz no puede ser equiparado en eficiencia con un procesador multinúcleo de 2011 que opera a la misma velocidad de reloj. Estas dinámicas similares se aplican también a los procesadores gráficos. Para medir con precisión el rendimiento, se recomienda realizar pruebas de referencia exhaustivas.

Aspectos del rendimiento

- Disponibilidad: el tiempo durante el cual el sistema o el equipo están operativos y listos para su uso.
- Tiempo de respuesta: el tiempo transcurrido entre una solicitud y la recepción de la respuesta correspondiente.
- Capacidad del canal: la cantidad máxima de datos que puede transmitirse a través de un canal de comunicación en un período de tiempo determinado.
- Latencia: el tiempo que tarda un dato en viajar de un punto a otro en un sistema.
- Completion time: el tiempo total necesario para completar una tarea o proceso.
- Service time: el tiempo que un sistema o un componente tarda en realizar un servicio específico.
- Ancho de banda: la capacidad máxima de transferencia de datos de un sistema.
- Throughput: la cantidad de trabajo realizada por un sistema en un período de tiempo determinado.
- Eficiencia: la relación entre la entrada y la salida de un sistema, considerando los recursos utilizados.
- Escalabilidad: la capacidad de un sistema para manejar un aumento en la carga de trabajo o el tamaño sin perder rendimiento.
- Rendimiento por vatio: la cantidad de trabajo realizado por unidad de energía consumida.
- Ratio de compresión: la relación entre el tamaño original y el tamaño comprimido de un conjunto de datos.
- Instrucción de longitud de la trayectoria: la longitud promedio de las instrucciones en una ruta de ejecución.
- Aumento de velocidad: la mejora en la velocidad o rendimiento de un sistema.

Optimización del rendimiento

La optimización del rendimiento se refiere a mejorar el rendimiento de un sistema, que puede ser una aplicación informática, pero también puede aplicarse a mercados económicos, burocracias u otros sistemas complejos. La motivación detrás de esta actividad se conoce como un problema de rendimiento, que puede ser actual o anticipado. La mayoría de los sistemas experimentarán una disminución en su rendimiento

cuando se enfrenten a un aumento en la carga. La capacidad de un sistema para manejar una carga más alta se conoce como escalabilidad, y realizar modificaciones en un sistema para que pueda manejar una carga mayor se denomina optimización del rendimiento.

1. El proceso de optimización del rendimiento sigue una serie de pasos sistemáticos:
2. Evaluar el problema y establecer los criterios que definen un comportamiento aceptable en términos numéricos.
3. Medir el rendimiento del sistema antes de realizar modificaciones.
4. Identificar la parte del sistema que limita el rendimiento, conocida como el cuello de botella.
5. Modificar la parte del sistema que constituye el cuello de botella para eliminar esa limitación.
6. Medir el rendimiento del sistema después de realizar las modificaciones.
7. Si las modificaciones mejoran el rendimiento, implementarlas permanentemente. En caso contrario, revertir los cambios y mantener el sistema en su estado original.

Desarrollo

Listado de benchmarks

A continuación, se adjunta una tabla que muestra diferentes benchmarks posibles de realizar para verificar la performance de computadores en la vida cotidiana (para usos promedio en usuarios como los alumnos que están desarrollando este práctico).

Benchmark	Descripción
Cinebench R23	Es un benchmark que evalúa las capacidades de hardware de la computadora.
UL Benchmarks	Es un software diseñado específicamente para evaluar y comparar el rendimiento de tarjetas de video y GPU.
3DMark CPU-Z	Es un programa que detalla las características de la computadora y también permite hacer ciertos benchmark con una interfaz muy sencilla.
Google Benchmark	Es una biblioteca de código abierto que permite realizar microbenchmarks en programas de lenguaje C++, es decir, pruebas de fragmentos de códigos.
Hyperfine	Es una herramienta que se utiliza en la terminal, la cual compara la performance de los comandos que se ejecuten en la misma. Esto permite ver de una manera sencilla cómo estos afectan la performance de la computadora.
Corona	Este programa puede chequear si la performance de la computadora es tan buena como debería ser comparado con computadoras similares, como también que mejoras de hardware o configuraciones se pueden hacer para mejorar dicha performance.
Y-cruncher	Es un programa que puede computar pi y otras constantes, que sean irracionales is a program that can compute Pi and other constants to trillions of digits. Mientras mayor sea la capacidad de procesamiento de la computadora mayor cantidad de dígitos puede calcular.
7-zip	Es un programa que permite hacer un benchmark de la capacidad que tiene la computadora de comprimir y descomprimir archivos.

glmark2

Es una herramienta de código abierto que se utiliza para evaluar el rendimiento gráfico en sistemas Linux utilizando OpenGL y OpenGL ES.

Tabla 1. Distintos benchmarks para usuarios comunes de un computador.

Para un uso cotidiano, se considera que los siguientes benchmarks pueden medir mejor las tareas realizadas. Si bien es posible utilizar cualquiera de las opciones anteriores, quizás algunas resultan un tanto específicas, o pueden llegar a ofrecer una gran variedad de pruebas, como ocurre con la familia SPEC. Se considera importante mostrar alternativas sencillas y rápidas de ejecutar.

Benchmark	Actividad(es)
CPU-X	Comprobar características del computador con Linux.
glmark2	Comprobar rendimiento de la tarjeta gráfica.
Google Benchmark	Bloques de código en C/C++.
Phoronix Test Suite	Instalación de kernels customs.
Hyperfine, Time	Tiempo de apertura de aplicaciones.

Tabla 2. Benchmarks para tareas cotidianas.

Ejemplos de ejecución de benchmarks

Hyperfine

Se probó el benchmark Hyperfine junto con la utilidad de speedtest-cli. Se muestra una captura del terminal.

```
Benchmark 1: speedtest-cli
Time (mean ± σ):    24.757 s ±  1.537 s    [User: 2.148 s, System: 1.567 s]
Range (min ... max): 23.225 s ... 27.769 s    10 runs
→ ~
```

Figura 4. Benchmark Hyperfine junto con SpeedTest-CLI.

glmark2

Se probó el rendimiento de la GPU con glmark2, en su versión 2021.02. A continuación una captura de pantalla del proceso, y, finalmente los resultados arrojados.

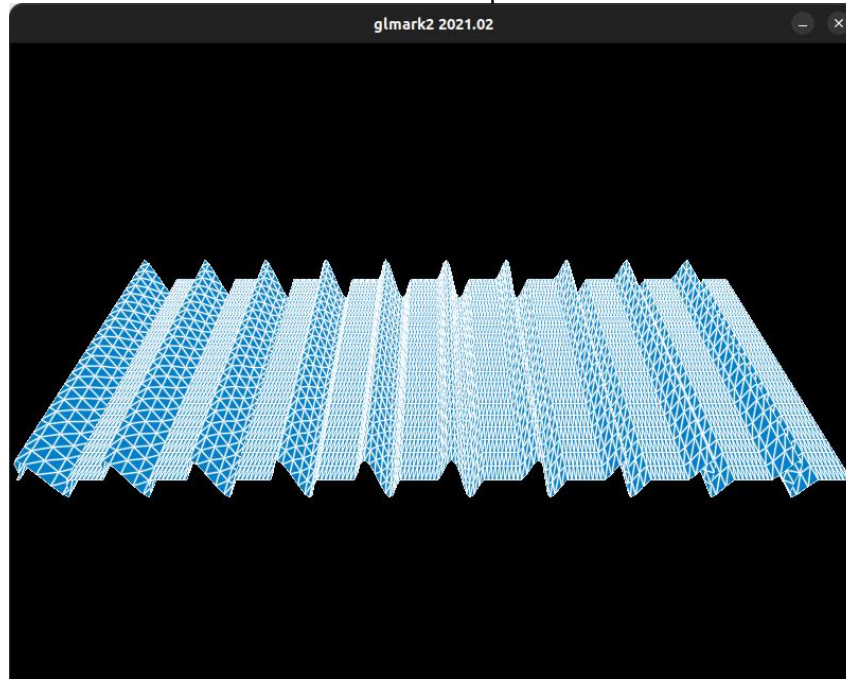


Figura 5. Ejecución de glmark2.

```

~ glmark2
=====
glmark2 2021.02
=====
OpenGL Information
GL_VENDOR: Intel
GL_RENDERER: Mesa Intel(R) HD Graphics 620 (KBL GT2)
GL_VERSION: 4.6 (Compatibility Profile) Mesa 23.2.1-1ubuntu3.1-22.04.2
=====
[build] use-vbo=false: FPS: 2314 FrameTime: 0.432 ms
[build] use-vbo=true: FPS: 2693 FrameTime: 0.371 ms
[texture] texture-filter=nearest: FPS: 2325 FrameTime: 0.430 ms
[texture] texture-filter=linear: FPS: 2355 FrameTime: 0.425 ms
[texture] texture-filter=mipmap: FPS: 2095 FrameTime: 0.477 ms
[shading] shading=gouraud: FPS: 2464 FrameTime: 0.406 ms
[shading] shading=blinn-phong-inf: FPS: 2280 FrameTime: 0.439 ms
[shading] shading=phong: FPS: 2320 FrameTime: 0.431 ms
[shading] shading=cel: FPS: 2277 FrameTime: 0.439 ms
[bump] bump-render=high-poly: FPS: 1579 FrameTime: 0.633 ms
[bump] bump-render=normals: FPS: 2761 FrameTime: 0.362 ms
[bump] bump-render=height: FPS: 2761 FrameTime: 0.362 ms
[effect2d] kernel=0,1,0;1,-4,1;0,1,0: FPS: 2075 FrameTime: 0.482 ms
[effect2d] kernel=1,1,1,1;1,1,1,1;1,1,1,1;1,1,1,1: FPS: 1102 FrameTime: 0.907 ms
[pulsar] light=false:quads=5:texture=false: FPS: 2893 FrameTime: 0.346 ms
[desktop] blur-radius=5:effect=blur:passes=1:separable=true:windows=4: FPS: 1007 FrameTime: 0.993 ms
[desktop] effect=shadow:windows=4: FPS: 1677 FrameTime: 0.596 ms
[buffer] columns=200:interleave=false:update-dispersion=0.9:update-fraction=0.5:update-method=map: FPS: 475 FrameTime: 2.105 ms
[buffer] columns=200:interleave=false:update-dispersion=0.9:update-fraction=0.5:update-method=subdata: FPS: 627 FrameTime: 1.595 ms
[buffer] columns=200:interleave=true:update-dispersion=0.9:update-fraction=0.5:update-method=map: FPS: 617 FrameTime: 1.621 ms
[ideas] speed=duration: FPS: 2139 FrameTime: 0.468 ms
[jellyfish] <default>: FPS: 1848 FrameTime: 0.541 ms
[terrain] <default>: FPS: 202 FrameTime: 4.950 ms
[shadow] <default>: FPS: 1488 FrameTime: 0.672 ms
[refract] <default>: FPS: 388 FrameTime: 2.577 ms
[conditionals] fragment-steps=0:vertex-steps=0: FPS: 1785 FrameTime: 0.560 ms
[conditionals] fragment-steps=5:vertex-steps=0: FPS: 2103 FrameTime: 0.476 ms
[conditionals] fragment-steps=0:vertex-steps=5: FPS: 2092 FrameTime: 0.478 ms
[function] fragment-complexity=low:fragment-steps=5: FPS: 1978 FrameTime: 0.506 ms
[function] fragment-complexity=medium:fragment-steps=5: FPS: 1839 FrameTime: 0.544 ms
[loop] fragment-loop=false:fragment-steps=5:vertex-steps=5: FPS: 1893 FrameTime: 0.528 ms
[loop] fragment-steps=5:fragment-uniform=false:vertex-steps=5: FPS: 1847 FrameTime: 0.541 ms
[loop] fragment-steps=5:fragment-uniform=true:vertex-steps=5: FPS: 1870 FrameTime: 0.535 ms
=====
glmark2 Score: 1823
=====

```

Figura 6. Resultados de ejecución de glmark2.

PassMark

Este benchmark fue ejecutado en una computadora desktop, con Windows 10, Intel Core i5 7400 y 24GB de RAM, y se obtuvieron los siguientes resultados.

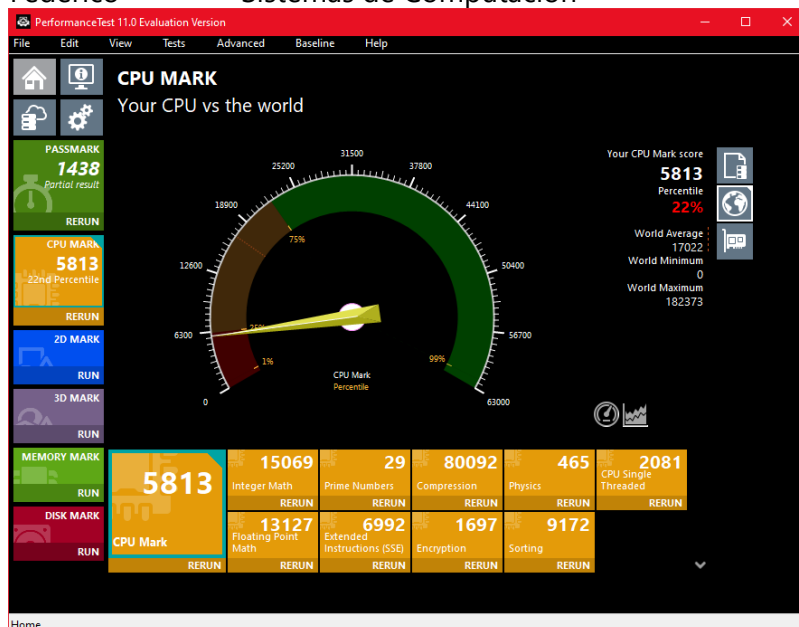


Figura 7. Ejecución de PassMark en Windows 10.

Luego, el mismo benchmark fue ejecutado en una computadora MacBook Air M1, con 8GB de RAM y sistema operativo MacOS Sonoma 14.0, obteniéndose los siguientes resultados.

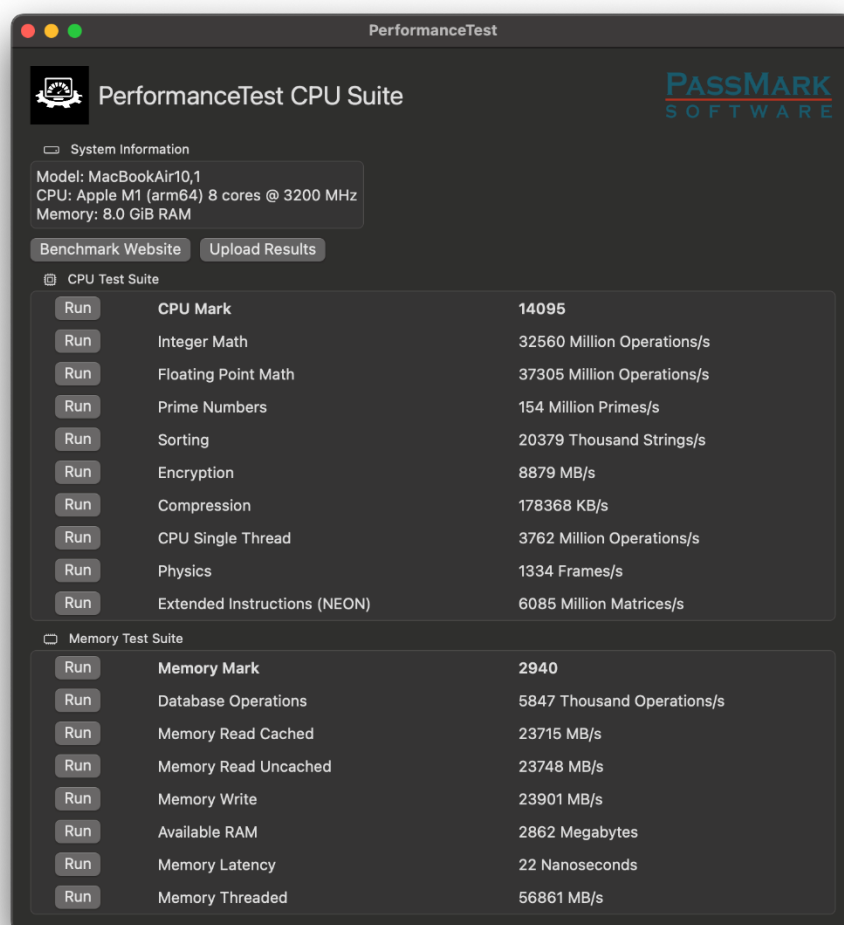


Figura 8. Ejecución de PassMark en MacOS.

Rendimiento de algunos procesadores

En la consigna del trabajo práctico se solicita analizar y comparar el rendimiento de dos procesadores en particular. Se los presenta en un principio, junto con sus especificaciones, para luego realizar el análisis.

Intel Core i5 13600k

- Número total de núcleos y subtipos: El procesador cuenta con un total de 14 núcleos, divididos en 6 núcleos de alto rendimiento y 8 núcleos eficientes. Esta configuración puede ofrecer un buen equilibrio entre rendimiento y eficiencia energética.
- Frecuencia máxima de Turbo Boost: El procesador tiene una frecuencia máxima de Turbo Boost de 5.1 GHz, lo que indica un excelente potencial de rendimiento en tareas intensivas.
- Caché Intel Smart: Con 24 MB de caché Intel Smart, el procesador puede acceder rápidamente a datos frecuentemente utilizados, lo que mejora el rendimiento general del sistema.
- Consumo de energía: Con un TDP (Thermal Design Power) de 125 W y una potencia máxima de Turbo de 181 W, es importante tener en cuenta los requisitos de energía del procesador para la refrigeración y la fuente de alimentación del sistema.
- Compatibilidad de memoria: Es compatible con una amplia variedad de tipos y velocidades de memoria, incluidas DDR5 hasta 5600 MT/s y DDR4 hasta 3200 MT/s, con un máximo de 192 GB de tamaño de memoria. Esto permite una flexibilidad considerable en la configuración del sistema.
- Gráficos integrados Intel UHD Graphics 770: Aunque no es un procesador centrado en gráficos, los gráficos integrados pueden ser útiles para tareas ligeras de visualización y permiten hasta cuatro pantallas simultáneas.
- Tecnologías avanzadas de Intel: Incluye una variedad de tecnologías avanzadas de Intel, como Intel Thread Director, Intel DL Boost, Intel Speed Shift Technology, Intel Hyper-Threading Technology, y más, que pueden mejorar el rendimiento, la eficiencia y la seguridad del sistema.
- Seguridad y confiabilidad: Ofrece una gama de características de seguridad y confiabilidad, desde Intel Hardware Shield hasta Intel Threat Detection Technology, que ayudan a proteger el sistema contra amenazas de seguridad.

AMD Ryzen 9 5900X

- Número de núcleos y hilos: Este procesador cuenta con 12 núcleos y 24 hilos, lo que proporciona un gran potencial de multitarea y rendimiento en aplicaciones que pueden aprovechar múltiples núcleos.
- Frecuencia máxima de impulso: El procesador tiene una frecuencia máxima de impulso de hasta 4.8GHz, lo que indica un excelente rendimiento en tareas de una sola rosca y cargas de trabajo intensivas.
- Caché L3 grande: Con 64MB de caché L3, el procesador puede acceder rápidamente a datos utilizados con frecuencia, lo que mejora el rendimiento general del sistema.
- TDP (Thermal Design Power): Tiene un TDP de 105W, lo que proporciona una indicación del consumo de energía y los requisitos de refrigeración del procesador.
- Tecnología de proceso: Fabricado en tecnología TSMC 7nm FinFET, lo que sugiere una eficiencia energética mejorada y un potencial de overclocking.
- Desbloqueo para overclocking: Permite el overclocking para aquellos usuarios que deseen ajustar el rendimiento del procesador más allá de las especificaciones de fábrica.
- Zócalo del procesador y compatibilidad de memoria: Compatible con el zócalo AM4 y soporta memoria DDR4 con una velocidad de hasta 3200MT/s, lo que ofrece flexibilidad en la elección de la placa base y la memoria RAM.

- PCI Express 4.0: Ofrece compatibilidad con PCIe 4.0, lo que permite un ancho de banda más alto para dispositivos periféricos compatibles, como tarjetas gráficas y unidades de almacenamiento.
- No incluye gráficos integrados: Requiere una tarjeta gráfica discreta para la salida de video, lo que es común en los procesadores de escritorio de gama alta.
- Tecnologías compatibles de AMD: Incluye características como AMD StoreMI Technology, AMD "Zen 3" Core Architecture, AMD Ryzen™ Master Utility, y AMD Ryzen™ VR-Ready Premium, que pueden mejorar la experiencia general del usuario en términos de rendimiento, administración y compatibilidad.

AMD Ryzen 9 7950X

- Arquitectura: Basado en la arquitectura "Zen 4".
- Núcleos y hilos: Ofrece 16 núcleos y 32 hilos, lo que permite un alto rendimiento en tareas multitarea.
- Frecuencia máxima de impulso: Alcanza una frecuencia de impulso de hasta 5.7GHz para un rendimiento rápido en tareas de una sola rosca.
- Caché: Cuenta con 1MB de caché L1, 16MB de caché L2 y 64MB de caché L3, lo que garantiza un acceso rápido a los datos.
- TDP: Tiene un TDP de 170W, indicando un consumo de energía relativamente alto.
- Tecnología de proceso: Fabricado en tecnología TSMC 5nm FinFET para eficiencia energética.
- Socket del procesador: Utiliza el socket AM5.
- Chipsets compatibles: Soporta chipsets A620, X670E, X670, B650E y B650.
- Conectividad: Ofrece conectividad USB Type-C®, USB 3.2 Gen 2 y PCIe® 5.0.
- Memoria del sistema: Admite DDR5 con una capacidad máxima de 128GB y velocidades de hasta DDR5-5200.
- Gráficos integrados: Incluye gráficos AMD Radeon™ con 2 núcleos y una frecuencia de 2200 MHz.
- Tecnologías admitidas: Soporta tecnologías como AMD EXPO™ y otras características de la familia AMD Ryzen™.

Rendimiento en la compilación del Kernel de Linux

En el sitio web de [OpenBenchmarking](https://openbenchmarking.org/) se ofrecen resultados de benchmarks para distintos procesadores, entre los que se encuentran los anteriores. Se pide primero una comparación del rendimiento entre los primeros 2 procesadores mencionados, esto puede verse en la tabla, como lo siguiente.

Procesador	Ranking percentil	Resultados públicos compatibles	Segundos (promedio)
Intel Core i5 13600k	57	9	72 +/- 6
AMD Ryzen 9 5900X	55	47	76 +/- 8

Tabla 3. Datos de los procesadores solicitados en OpenBenchmarking.

De la definición de rendimiento dada en la clase práctica, se calcula para los procesadores lo siguiente:

$$\eta_{i5} = \frac{1}{T_{EX}} = \frac{1}{72} = 1.388\%$$

$$\eta_{R9} = \frac{1}{T_{EX}} = \frac{1}{76} = 1.315\%$$

Puede verse un rendimiento similar en ambos procesadores. También hay que entender que no son del todo comparables ambos procesadores, ya que el Ryzen tiene menos núcleos, pero son todos de alto rendimiento, contra la arquitectura big.LITTLE del procesador de Intel.

Se encontró también en la página de OpenBenchmarking un promedio sobre los benchmarks luego realizados a cada uno de los procesadores por separado, independientemente de la prueba de compilación del kernel de Linux, y puede encontrarse los siguiente.

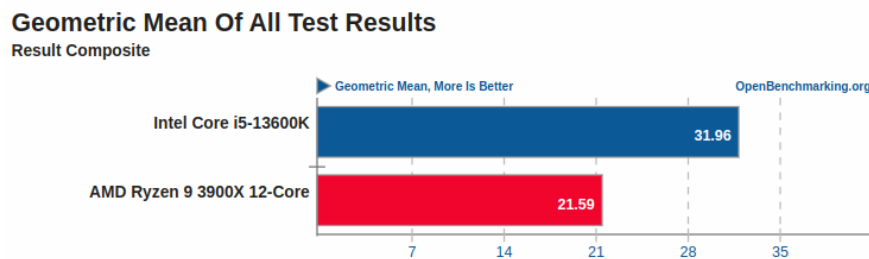


Figura 9. Comparación de benchmarks en Intel Core i5 13600k y AMD Ryzen 9 5900X.

Aceleración para un AMD Ryzen 7950X

El speed up es la razón entre el rendimiento de un sistema mejorado y el rendimiento de su implementación original, esta es también la definición de aceleración. En este caso, el rendimiento de la implementación original correspondería al modelo anterior de Ryzen 9, es decir, el del Ryzen 9 5900X, y el mejorado el del procesador mencionado en el título superior, se extraen los resultados de este último.

Procesador	Ranking percentil	Resultados públicos compatibles	Segundos (promedio)
AMD Ryzen 9 7950X	73	185	50 +/- 6

Tabla 4. AMD Ryzen 9 7950X en OpenBenchmarking.

$$\eta_{\text{mejorado}} = \frac{1}{T_{EX\text{mejorado}}} = \frac{1}{50} = 2\%$$

$$\text{speed up} = \frac{\eta_{\text{mejorado}}}{\eta_{\text{original}}} = \frac{2\%}{1.315\%} = 1.521$$

Puede verse que, al pasar de 12 a 16 núcleos, en este procesador se tuvo un aumento del rendimiento en 52% aproximadamente.

Relación entre eficiencia y costos

La eficiencia está definida como:

$$\text{eficiencia} = \frac{\text{speed up}_n}{n}$$

Entonces, para cada procesador, se tiene una determinada eficiencia, que va a estar dada por el speed up de cada procesador en base a su cantidad de núcleos, se tiene así:

$$\text{eficiencia}_{i5} = 0.0753$$

$$\text{eficiencia}_{R95} = 0.0833$$

$$\text{eficiencia}_{R97} = 0.095$$

Los costos son:

- 290 dólares para el Intel Core i5 13600k
- 260 dólares para el AMD Ryzen 9 5900X
- 550 dólares para el AMD Ryzen 9 7950X

Entonces, la relación entre costo y eficiencia para cada procesador resulta en:

- Intel Core i5 13600k: 0.00363
- AMD Ryzen 9 5900X: 0.00384
- AMD Ryzen 9 7950X: 0.00276

Concluyendo así en que el procesador AMD Ryzen 9 5900X es el que mejor relación eficiencia costo tiene.

Time profiling

Se realizan pruebas en el sistema operativo Ubuntu 22.04 LTS, en dos computadoras diferentes, sin embargo, se detalla el procedimiento para una única para evitar repetición de información.

Características del sistema

Se comienza solicitando al sistema las características del hardware sobre el que corre, para eso se utilizó un determinado comando en la terminal, obteniéndose el resultado adjuntado a continuación.

```
1. fedev@fedev:~$ sudo lshw -short
2. H/W path          Device          Class          Description
3. =====
4.
5. /0
6. /0/0              bus            H110M-H (Default string)
7. /0/3d             memory         64KiB BIOS
8. /0/3d/0           memory         24GiB System Memory
9. /0/3d/1           memory         8GiB DIMM DDR4 Synchronous 2133
10. /0/3d/2           memory         16GiB DIMM DDR4 Synchronous 213
11. /0/3d/3           memory         [empty]
12. /0/43             memory         256KiB L1 cache
13. /0/44             memory         1MiB L2 cache
14. /0/45             memory         6MiB L3 cache
15. /0/46             processor      Intel(R) Core(TM) i5-7400 CPU @
16. /0/100            bridge         Xeon E3-1200 v6/7th Gen Core Pr
17. /0/100/1          bridge         6th-10th Gen Core Processor PCI
18. /0/100/1/0        display        GP106 [GeForce GTX 1060 6GB]
19. /0/100/1/0.1      card1          multimedia     GP106 High Definition Audio Con
20. /0/100/1/0.1/0    input14        input          HDA NVidia HDMI/DP,pcm=3
21. /0/100/1/0.1/1    input15        input          HDA NVidia HDMI/DP,pcm=7
22. /0/100/1/0.1/2    input16        input          HDA NVidia HDMI/DP,pcm=8
23. /0/100/1/0.1/3    input17        input          HDA NVidia HDMI/DP,pcm=9
24. /0/100/2          display        HD Graphics 630
25. /0/100/14         bus            100 Series/C230 Series Chipset
26. /0/100/14/0       usb1           bus            xHCI Host Controller
27. /0/100/14/0/1     generic        802.11ac NIC
28. /0/100/14/0/5     input4         input          Logitech G502 HERO Gaming Mouse
29. /0/100/14/0/6     input10        input          HP, Inc HyperX Alloy Elite 2 Mo
30. /0/100/14/0/8     communication   Board in FS mode
31. /0/100/14/1       usb2           bus            xHCI Host Controller
32. /0/100/16         communication   100 Series/C230 Series Chipset
33. /0/100/17         scsi0          storage        Q170/Q150/B150/H170/H110/Z170/C
34. /0/100/17/0       /dev/sda       disk           1TB WDC WD10EURX-63C
35. /0/100/17/0/1     /dev/sda1      volume         500MiB Windows NTFS volume
36. /0/100/17/0/2     /dev/sda2      volume         871GiB Windows NTFS volume
37. /0/100/17/0/3     /dev/sda3      volume         528MiB Windows NTFS volume
38. /0/100/17/0/4     /dev/sda4      volume         58GiB Extended partition
39. /0/100/17/0/4/5   /dev/sda5      volume         11GiB Linux swap volume
40. /0/100/17/0/4/6   /dev/sda6      volume         243MiB Windows FAT volume
41. /0/100/17/0/4/7   /dev/sda7      volume         24GiB EXT4 volume
```

42. /0/100/17/0/4/8	/dev/sda8	volume	23GiB EXT4 volume
43. /0/100/17/1	/dev/sdb	disk	480GB KINGSTON SA400S3
44. /0/100/17/1/1	/dev/sdb1	volume	446GiB Windows NTFS volume
45. /0/100/17/1/2	/dev/sdb2	volume	530MiB Windows NTFS volume
46. /0/100/1c		bridge	100 Series/C230 Series Chipset
47. /0/100/1c/0	enp2s0	network	RTL8111/8168/8411 PCI Express G
48. /0/100/1c.5		bridge	100 Series/C230 Series Chipset
49. /0/100/1c.6		bridge	100 Series/C230 Series Chipset
50. /0/100/1c.7		bridge	100 Series/C230 Series Chipset
51. /0/100/1d		bridge	100 Series/C230 Series Chipset
52. /0/100/1d.1		bridge	100 Series/C230 Series Chipset
53. /0/100/1f		bridge	H110 Chipset LPC/eSPI Controlle
54. /0/100/1f/0		system	PnP device PNP0c02
55. /0/100/1f/1		system	PnP device PNP0c02
56. /0/100/1f/2		system	PnP device PNP0c02
57. /0/100/1f/3		system	PnP device PNP0b00
58. /0/100/1f/4		generic	PnP device INT3f0d
59. /0/100/1f/5		system	PnP device PNP0c02
60. /0/100/1f/6		system	PnP device PNP0c02
61. /0/100/1f/7		system	PnP device PNP0c02
62. /0/100/1f/8		system	PnP device PNP0c02
63. /0/100/1f.2		memory	Memory controller
64. /0/100/1f.3	card0	multimedia	100 Series/C230 Series Chipset
65. /0/100/1f.3/0	input18	input	HDA Intel PCH Front Mic
66. /0/100/1f.3/1	input19	input	HDA Intel PCH Rear Mic
67. /0/100/1f.3/2	input20	input	HDA Intel PCH Line
68. /0/100/1f.3/3	input21	input	HDA Intel PCH Line Out
69. /0/100/1f.3/4	input22	input	HDA Intel PCH Front Headphone
70. /0/100/1f.3/5	input23	input	HDA Intel PCH HDMI/DP,pcm=3
71. /0/100/1f.3/6	input24	input	HDA Intel PCH HDMI/DP,pcm=7
72. /0/100/1f.3/7	input25	input	HDA Intel PCH HDMI/DP,pcm=8
73. /0/100/1f.4		bus	100 Series/C230 Series Chipset
74. /1		power	To Be Filled By O.E.M.
75. /2	/dev/fb0	display	VESA VGA
76. /3	input0	input	Sleep Button
77. /4	input1	input	Power Button
78. /5	input2	input	Power Button
79. /6	input3	input	Video Bus
80. /7	wlxc006c39f8358	network	Wireless interface
81.			

GNU GCC Profiling (gprof)

Se utilizó la herramienta gprof para poder realizar el profiling de los siguientes códigos escritos en C. El procedimiento seguido es el mencionado en la bibliografía de este trabajo práctico, presente en la página principal del classroom. Cabe destacar que para poder ejecutar este código no fue necesaria la instalación de ninguna librería. Sin embargo, se trabajó en el editor de texto de Visual Studio Code, por la simplicidad que conlleva poder visualizar de forma gráfica con pestañas los códigos (en este caso escritos en lenguaje C) y la terminal por debajo.

Esta herramienta de generación de perfiles se encuentra dentro de GNU GCC, lo que permite que se pueda utilizar fácilmente en sistemas operativos Linux. Se investigó si existe también para MacOS por tratarse de otro sistema operativo basado en UNIX, sin embargo no existe, pero se suele utilizar la alternativa de [Google Benchmarks](#) a través de [gperftools](#) (originalmente Google Performance Tools), instalado mediante el gestor de paquetes HomeBrew.

```
//test_gprof.c
#include<stdio.h>

void new_func1(void);

void func1(void)
{
    printf("\n Inside func1 \n");
    int i = 0;

    for(;i<0xffffffff;i++);
    new_func1();

    return;
}

static void func2(void)
{
    printf("\n Inside func2 \n");
    int i = 0;

    for(;i<0xfffffaa;i++);
    return;
}

int main(void)
{
    printf("\n Inside main() \n");
    int i = 0;

    for(;i<0xfffff;i++);
    func1();
    func2();

    return 0;
}
```

Figura 10. Código test_gprof.c

```
//test_gprof_new.c
#include<stdio.h>

void new_func1(void)
{
    printf("\n Inside new_func1() \n");
    int i = 0;

    for(;i<0xfffffee;i++);

    return;
}
```

Figura 11. Código test_gprof_new.c

Es necesario asegurarse de que la generación de perfiles esté habilitada, es por eso que se compila el código con el siguiente comando de consola:

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

Finalmente, se realizan varias llamadas a gprof, además de listar los elementos en el directorio un par de veces para corroborar la correcta ejecución del comando. Se muestra a continuación la secuencia de comandos.



```
fedev@fedev:~/Documents/practico# gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
fedev@fedev:~/Documents/practico# ls
test_gprof  test_gprof.c  test_gprof_new.c
fedev@fedev:~/Documents/practico# ./test_gprof
Inside main()
Inside func1
Inside new_func1()
Inside func2
fedev@fedev:~/Documents/practico# ls
gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
fedev@fedev:~/Documents/practico# gprof test_gprof gmon.out > analysis1.txt
fedev@fedev:~/Documents/practico# ls
analysis1.txt  gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
fedev@fedev:~/Documents/practico# gprof -a test_gprof gmon.out > analysis2.txt
fedev@fedev:~/Documents/practico# gprof -b test_gprof gmon.out > analysis3.txt
fedev@fedev:~/Documents/practico# gprof -p -b test_gprof gmon.out > analysis4.txt
fedev@fedev:~/Documents/practico# gprof -pfunc1 -b test_gprof gmon.out > analysis5.txt
fedev@fedev:~/Documents/practico# gprof test_gprof | gprof2dot | dot -Tpng -o output1.png
fedev@fedev:~/Documents/practico# gprof -a test_gprof | gprof2dot | dot -Tpng -o output2.png
fedev@fedev:~/Documents/practico# gprof -b test_gprof | gprof2dot | dot -Tpng -o output3.png
fedev@fedev:~/Documents/practico# gprof -f pstats -b test_gprof | gprof2dot | dot -Tpng -o output4.png
fedev@fedev:~/Documents/practico# gprof -f pstats -pfunc1 -b test_gprof | gprof2dot | dot -Tpng -o output5.png
fedev@fedev:~/Documents/practico# gprof -f pfunc1 -b test_gprof | gprof2dot | dot -Tpng -o output5.png
```

Figura 12. Consola durante la ejecución de la herramienta gprof.

En la figura 10 puede verse que se realizó lo siguiente:

1. Compilar el código bajo los flags requeridos en la consigna
2. Ejecutar el código
3. Ejecutar la herramienta gprof
4. Suprimir la impresión de funciones declaradas estáticamente
5. Eliminar textos detallados
6. Imprimir solo perfil plano
7. Imprimir información relacionada con funciones específicas en perfil plano
8. Se generan gráficos para las diferentes corridas

Resultados

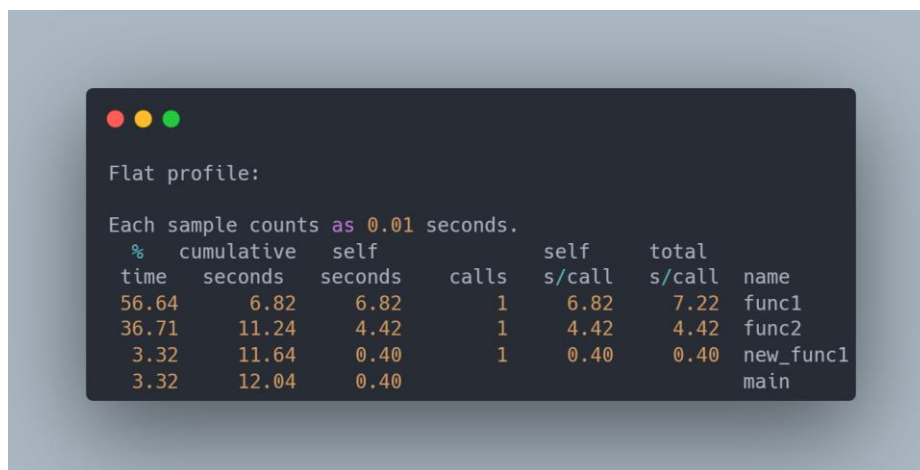


Figura 13. Resultados flat profile gprof.

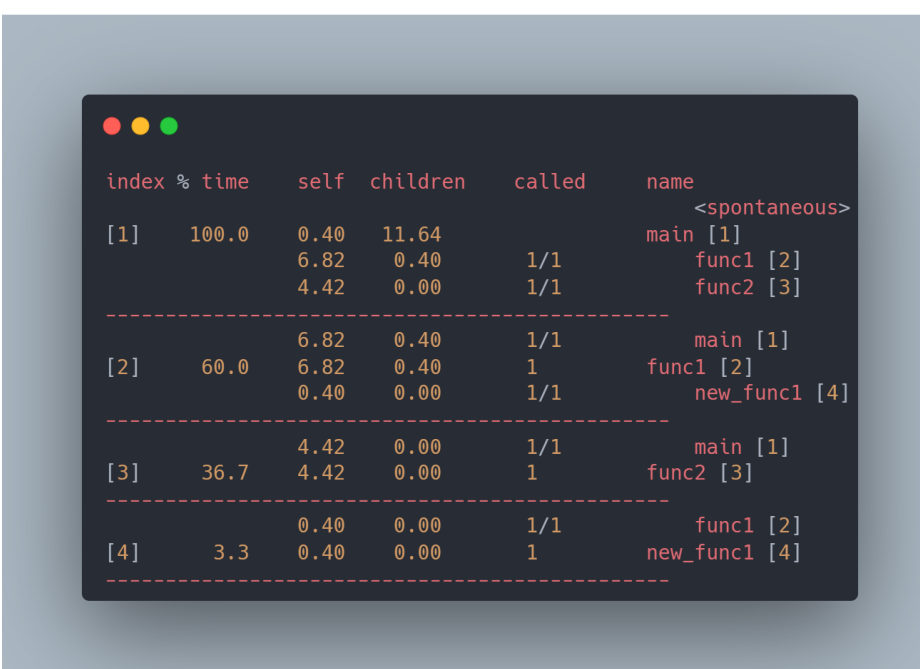


Figura 14. Call graph de la ejecución de gprof.

Luego, las imágenes que se obtuvieron de graficar son 2. Pueden verse a continuación. Esto se realizó con la herramienta gprof2dot, con un comando como el siguiente:

```
1. gprof path_to_exec | gprof2dot | dot -Tpng -o outputFile.png
```

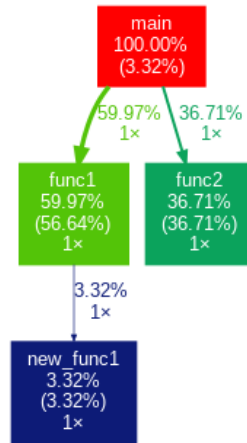


Figura 15. Ejecución de la primera prueba con gprof.

Luego, en la ejecución en donde se suprime la impresión de funciones declaradas estáticamente, se obtiene el siguiente dot file.

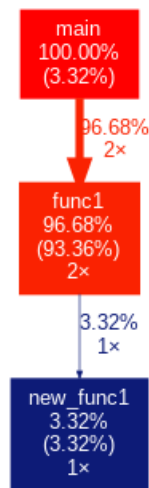


Figura 16. Ejecución de gprof suprimiendo funciones estáticas.

Se puede decir lo siguiente de los resultados de la prueba completa:

- El perfil plano (**flat profile**) muestra el porcentaje de tiempo de CPU consumido por cada función y su llamado desde otras funciones.
- La función **func1()** consume el 56.64% del tiempo total de ejecución, seguida de cerca por **func2()** con el 36.71%.
- **new_func1()** solo consume el 3.32% del tiempo total.
- En el gráfico de llamadas, se puede observar que **main()** es la función principal, que llama a **func1()** y **func2()**.
- **func1()** a su vez llama a **new_func1()**.

Concluyendo lo siguiente:

- La función **func1()** es el cuello de botella del programa, consumiendo la mayor parte del tiempo de ejecución. Esto sugiere que las mejoras de rendimiento en **func1()** tendrán un impacto significativo en el tiempo de ejecución total del programa.
- La función **func2()** también consume una parte considerable del tiempo de ejecución, pero en menor medida que **func1()**.

- **new_func1()** contribuye con una parte mínima del tiempo total de ejecución y, por lo tanto, no es un área prioritaria para la optimización.
- Para mejorar el rendimiento del programa, se deben considerar estrategias de optimización, como la reducción de la complejidad algorítmica en **func1()** y **func2()**, la paralelización de tareas si es posible, o la optimización de bucles mediante técnicas como el desenrollado de bucles o el uso de instrucciones SIMD.
- Es fundamental realizar más pruebas y análisis detallados para identificar las áreas específicas que requieren optimización y evaluar el impacto de las mejoras realizadas en el rendimiento general del programa.

Ahora, en cuanto a la ejecución ignorando las funciones estáticas, se tiene lo siguiente:

- **func1()** es la función que consume la mayor parte del tiempo de ejecución, con un 93.36% del tiempo total.
- **new_func1()** contribuye con un 3.32% del tiempo total de ejecución.
- **main()** también tiene un tiempo asociado, aunque no tiene ninguna función llamada directamente desde ella.

Se tienen las siguientes conclusiones:

- La función **func1()** es claramente el foco principal del tiempo de ejecución del programa, consumiendo la gran mayoría del tiempo de CPU.
- **new_func1()**, aunque llamada desde **func1()**, no contribuye significativamente al tiempo de ejecución en comparación con **func1()**.
- La función **main()** solo muestra el tiempo que tarda en ejecutarse directamente, sin ninguna función llamada directamente desde ella.
- Dado que la mayor parte del tiempo de ejecución se gasta en **func1()**, cualquier esfuerzo de optimización debe enfocarse en esta función. Estrategias como la reducción de la complejidad algorítmica o la optimización de bucles podrían ayudar a mejorar el rendimiento.
- La función **new_func1()** podría ser relevante para la optimización si se encuentra que su llamado se realiza en un contexto crítico para el rendimiento del programa.

Profiling con Linux perf

Utilizando la herramienta perf, instalada mediante los siguientes paquetes:

- linux-tools-6.5.0-26-generic
- linux-tools-generic

Se pueden lograr el siguiente tipo de resultados:

- Perfil de rendimiento del sistema: perf permite recopilar datos detallados sobre la utilización de la CPU, el uso de memoria, las llamadas al sistema, el tiempo de ejecución de las funciones, entre otros aspectos. Esto ayuda a identificar cuellos de botella y áreas de mejora en el rendimiento del sistema.
- Análisis de llamadas al sistema y eventos de hardware: perf puede rastrear llamadas al sistema, interrupciones de hardware y otros eventos importantes del sistema. Esto es útil para comprender cómo se utiliza el sistema y cómo responde a diferentes cargas de trabajo.
- Muestreo de contador de hardware: perf puede utilizar los contadores de hardware disponibles en los procesadores modernos para realizar muestreos periódicos del estado del sistema. Esto proporciona información detallada sobre el uso de la CPU, la latencia de la memoria, la actividad de E/S, entre otros aspectos.

- Análisis de cuellos de botella de CPU: perf puede identificar las áreas del código que consumen más tiempo de CPU y proporcionar información detallada sobre qué funciones o instrucciones están causando el cuello de botella.
- Rastreo de eventos de software: perf permite rastrear eventos específicos en el código, como la ejecución de funciones o la ocurrencia de ciertas instrucciones. Esto es útil para comprender el flujo de ejecución del programa y detectar posibles problemas de rendimiento.

Para el mismo computador en donde se realizó la prueba del gprof, se ejecuta ahora el siguiente comando:

```
1. fedev@fedev:~/Documents/practico#1$ sudo perf record test_gprof
```

Resultados

De la ejecución del comando mencionado anteriormente, se obtuvo la siguiente salida.

Samples: 51K of event 'cycles:P', Event count (approx.): 44559406079			
Overhead	Command	Shared Object	Symbol
0,01%	test_gprof	[kernel.kallsyms]	[k] xhci_update_erst_dequeue
0,01%	test_gprof	[kernel.kallsyms]	[k] __irqentry_text_end
0,01%	test_gprof	[kernel.kallsyms]	[k] mt_find
0,00%	test_gprof	[kernel.kallsyms]	[k] read_tsc
0,00%	test_gprof	[kernel.kallsyms]	[k] irqentry_exit_to_user_mode
0,00%	test_gprof	[kernel.kallsyms]	[k] error_entry
0,00%	test_gprof	[kernel.kallsyms]	[k] syscall_exit_to_user_mode
0,00%	test_gprof	[kernel.kallsyms]	[k] clear_page_erms
0,00%	test_gprof	[kernel.kallsyms]	[k] native_write_msr
0,00%	test_gprof	[kernel.kallsyms]	[k] xhci_ring_ep_doorbell
0,00%	test_gprof	[kernel.kallsyms]	[k] run_posix_cpu_timers
0,00%	test_gprof	[kernel.kallsyms]	[k] xhci_irq
0,00%	test_gprof	[kernel.kallsyms]	[k] __mod_node_page_state
0,00%	test_gprof	[kernel.kallsyms]	[k] atomic_dec_and_lock_irqsave
0,00%	test_gprof	[kernel.kallsyms]	[k] entry_SYSCALL_64_after_hwframe
0,00%	test_gprof	[kernel.kallsyms]	[k] error_return
0,00%	test_gprof	[kernel.kallsyms]	[k] native_irq_return_iret
0,00%	test_gprof	[kernel.kallsyms]	[k] task_tick_fair
0,00%	test_gprof	[kernel.kallsyms]	[k] account_user_time
0,00%	test_gprof	[kernel.kallsyms]	[k] rep_movs_alternative
0,00%	test_gprof	[kernel.kallsyms]	[k] collect_posix_cputimers
0,00%	test_gprof	[kernel.kallsyms]	[k] rcu_sched_clock_irq
0,00%	test_gprof	[kernel.kallsyms]	[k] fpu_alloc_mathframe
0,00%	test_gprof	[kernel.kallsyms]	[k] sync_regs
0,00%	test_gprof	[kernel.kallsyms]	[k] __find_next_bit
0,00%	test_gprof	[kernel.kallsyms]	[k] kmem_cache_alloc
0,00%	test_gprof	[kernel.kallsyms]	[k] __get_user_noccheck_8
0,00%	test_gprof	[kernel.kallsyms]	[k] tick_sched_do_timer
0,00%	test_gprof	[kernel.kallsyms]	[k] inc_rlimit_get_ucounts
0,00%	test_gprof	[kernel.kallsyms]	[k] __hrtimer_run_queues
0,00%	test_gprof	[kernel.kallsyms]	[k] fpregs_assert_state_consistent
0,00%	test_gprof	[kernel.kallsyms]	[k] __copy_from_user
0,00%	test_gprof	[kernel.kallsyms]	[k] llist_add_batch
0,00%	test_gprof	[kernel.kallsyms]	[k] __raw_spin_lock
0,00%	test_gprof	[kernel.kallsyms]	[k] hrtimer_run_queues
0,00%	test_gprof	[kernel.kallsyms]	[k] native_flush_tlb_local
0,00%	perf-exec	[kernel.kallsyms]	[k] native_write_msr

Cannot load tips.txt file, please install perf!

Figura 17. Resultado de ejecución de la herramienta perf.

La salida proporcionada por perf muestra información sobre los eventos de muestreo de ciclo (cycles) en el programa **test_gprof**. Aquí está la interpretación de la salida:

- Samples: Indica el número total de muestras recopiladas durante la ejecución del programa para el evento cycles:P (ciclos de CPU). En este caso, se recopilaron 51000 muestras.
- Event count (approx.): Proporciona una estimación del recuento total de eventos durante la ejecución del programa. En este caso, el recuento aproximado de ciclos de CPU es de 44559406079.

- Overhead: Es el porcentaje de sobrecarga asociado con el evento de muestreo. Indica cuánto tiempo adicional consume la recopilación de datos de rendimiento. En este caso, la sobrecarga es muy baja, del 0.01%.
- Command: Es el nombre del programa que se está perfilando. En este caso, el programa se llama `test_gprof`.
- Shared Object: Indica el objeto compartido (biblioteca) en el que se encuentra el símbolo (función) que se está perfilando.
- Symbol: Es el nombre de la función o símbolo del código que se está perfilando.
- %: Representa el porcentaje de tiempo de CPU dedicado a cada símbolo durante la ejecución del programa.
- [k]: Indica que el símbolo está en el espacio del kernel.

Por ejemplo, la primera línea de la salida indica que el 0.01% del tiempo de CPU se gastó en la función **`xhci_update_erst_dequeue`** del kernel, mientras que la segunda línea indica que el mismo porcentaje se gastó en **`__irqentry_text_end`** del kernel.

Prueba de performance con un microprocesador

Raspberry Pi Pico RP2040




Figura 18. Raspberry Pi Pico.

Las siguientes son las principales especificaciones técnicas del microcontrolador en cuestión:

- Dual ARM Cortex-M0+ @ 125MHz (standard)
- 264kB on-chip SRAM in six independent banks
- Support for up to 16MB of off-chip Flash memory via dedicated QSPI bus
- DMA controller
- Fully-connected AHB crossbar
- Interpolator and integer divider peripherals
- On-chip programmable LDO to generate core voltage
- 2 on-chip PLLs to generate USB and core clocks
- 30 GPIO pins, 4 of which can be used as analogue inputs
- Peripherals
 - 2 UARTs
 - 2 SPI controllers
 - 2 I2C controllers
 - 16 PWM channels
 - USB 1.1 controller and PHY, with host and device support
 - 8 PIO state machines

Prueba realizada

En la consigna del práctico se pide utilizar un procesador ESP32 o similar que permita modificar la frecuencia de trabajo. A partir de eso, se solicita ejecutar un código que demore alrededor de 10 segundos. Para este caso, el código elegido es el siguiente:



```
import time
import machine

machine.freq(125000000)

def suma_enteros():
    total = 0
    for i in range(333333):
        total += i
    return total

def suma_floats():
    total = 0.0
    for i in range(240000):
        total += float(i)
    return total

start_time = time.ticks_ms()
resultado_enteros = suma_enteros()
tiempo_enteros = time.ticks_ms() - start_time

start_time = time.ticks_ms()
resultado_floats = suma_floats()
tiempo_floats = time.ticks_ms() - start_time

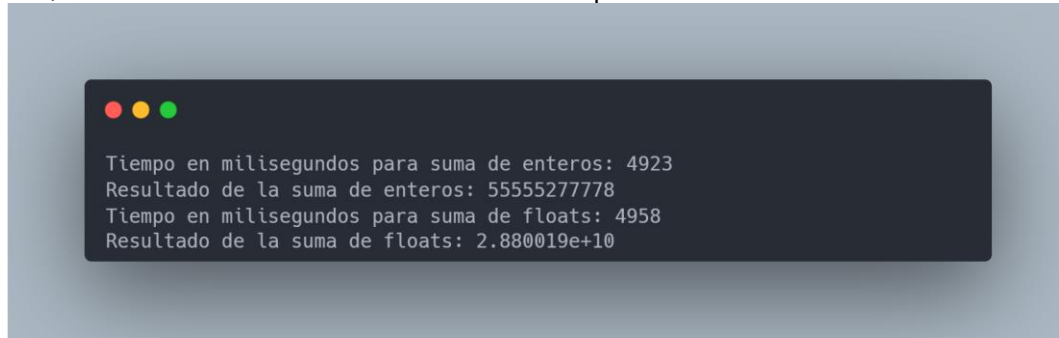
print("Tiempo en milisegundos para suma de enteros:", tiempo_enteros)
print("Resultado de la suma de enteros:", resultado_enteros)
print("Tiempo en milisegundos para suma de floats:", tiempo_floats)
print("Resultado de la suma de floats:", resultado_floats)
```

Figura 19. Código a ejecutar en Raspberry Pi Pico.

En el código puede verse cómo luego de importar las librerías se setea la frecuencia por defecto de funcionamiento del clock del procesador, para luego poder modificarla a exactamente el doble.

Resultados

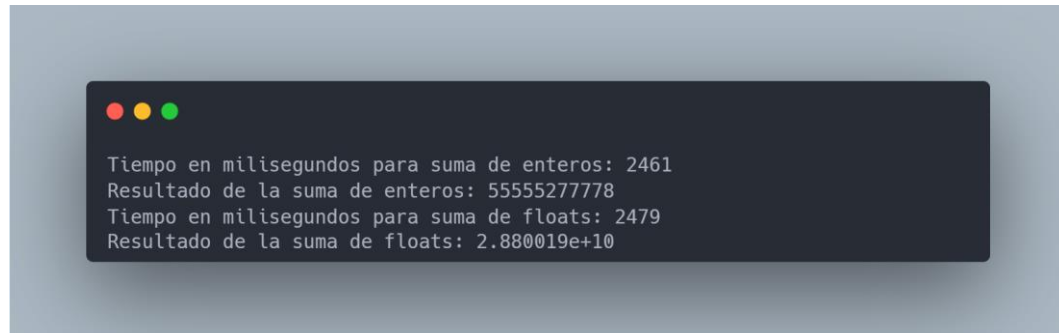
Utilizando la placa con la frecuencia de oscilador stock, se obtiene la siguiente salida del código ejecutado:



```
Tiempo en milisegundos para suma de enteros: 4923
Resultado de la suma de enteros: 5555527778
Tiempo en milisegundos para suma de floats: 4958
Resultado de la suma de floats: 2.880019e+10
```

Figura 20. Resultado de ejecución del código en Raspberry Pi Pico stock.

Ahora, cuando se modifica la frecuencia de clock al doble, se tiene la siguiente salida:



```
Tiempo en milisegundos para suma de enteros: 2461
Resultado de la suma de enteros: 5555527778
Tiempo en milisegundos para suma de floats: 2479
Resultado de la suma de floats: 2.880019e+10
```

Figura 21. Resultado de ejecución del código en Raspberry Pi Pico overclockeada.

Puede verse cómo claramente el tiempo de ejecución del programa se reduce a la mitad para una frecuencia de procesador del doble de la original. Esto guarda una relación directa, lineal, ya que el procesador ejecuta instrucciones que demoran una determinada cantidad de ciclos de reloj, por lo que, al disminuir la duración de los ciclos de reloj, también lo hará en la misma proporción el tiempo total de ejecución del programa. A continuación, se muestra un gráfico de cómo evoluciona el tiempo de ejecución del programa según diferentes frecuencias de clock.

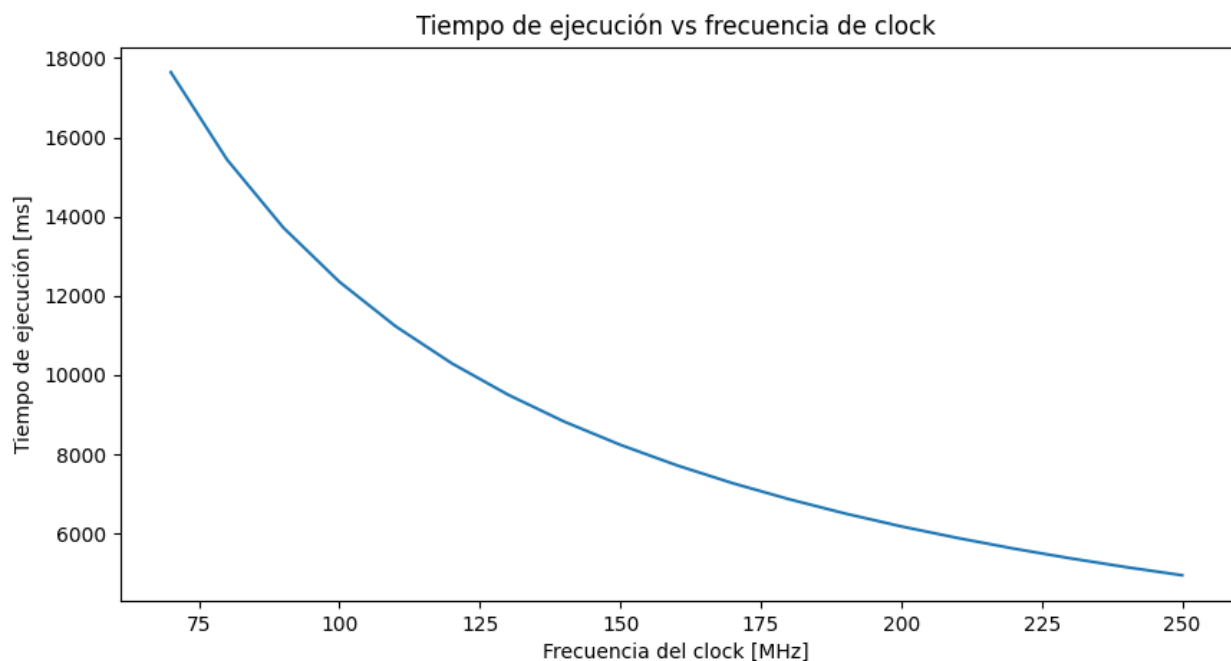


Figura 22. Evolución del tiempo de ejecución del programa en función de la frecuencia de clock.

Si bien una función cuadrática no es la mejor forma de aproximar este tipo de comportamiento, si se tiene en cuenta que existe un límite físico para el overclock de un microprocesador, en este caso un RP2040, podría tomarse la parte izquierda de una función cuadrática con coeficiente principal positivo en la forma $ax^2 + bx + c$. Se busca quedarse con la parte izquierda de forma de no visualizar más allá del cambio de pendiente de la curva, en donde se vuelve creciente. Con un programa de Python fue posible ajustar la curva que se muestra en la siguiente figura, en donde también se muestran los coeficientes de la función cuadrática.

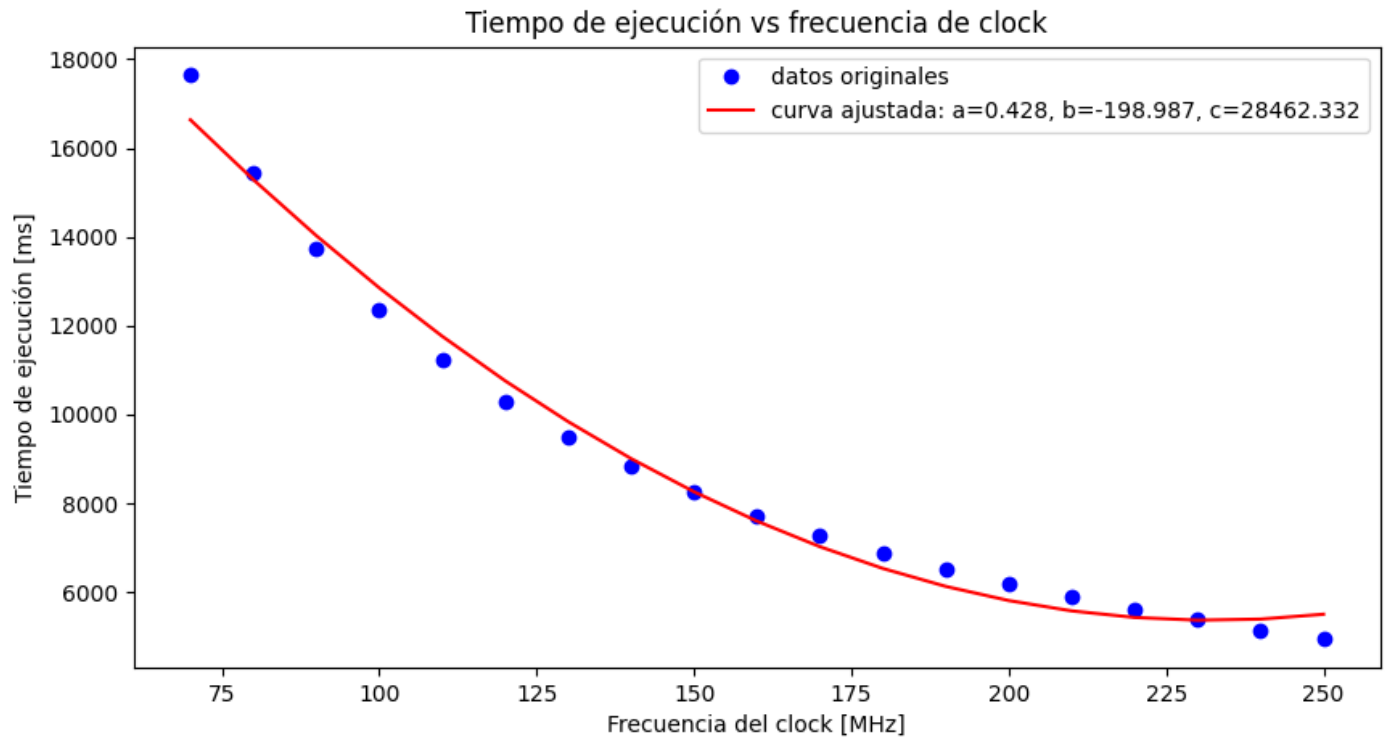


Figura 23. Interpolación cuadrática de la función que relaciona tiempo de ejecución con frecuencia de clock.