
Universidad Nacional de Córdoba


Facultad de Ciencias Exactas Físicas y Naturales

# Universidad Nacional de Córdoba

## Facultad de Ciencias Exactas, Físicas y Naturales

Sistemas de computación

Trabajo práctico N°1: El rendimiento de las computadoras


| Nombre | DNI |
|---|---|
| Clemenz Jeremías | 43449566 |
| Mansilla, Josías Leonel | 42978026 |
| Schroder Florencia | 42853785 |

Docentes

Jorge, Javier
Solinas, Miguel


23 de marzo de 2024. Córdoba, Argentina.

# Objetivos

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.

**En un informe deberán 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.**

- Armar una lista de benchmarks:
  - ¿cuales les serían más útiles a cada uno ?
  - ¿Cuáles podrían llegar a medir mejor las tareas que ustedes realizan a diario?
- Pensar en las tareas que cada uno realiza a diario y escribir en una tabla de dos entradas las tareas y qué benchmark la representa mejor.
- ¿Cuál es el rendimiento de estos procesadores para compilar el kernel de linux ?
  - Intel Core i5-13600K
  - AMD Ryzen 9 5900X 12-Core
  - Cual es la aceleración cuando usamos un AMD Ryzen 9 7950X 16-Core

  https://openbenchmarking.org/test/pts/build-linux-kernel-1.15.0

# Desarrollo

Siguiendo con los objetivos propuestos, se realizó una lista de benchmarks y, respectivamente, se indicó para qué actividades cotidianas serían útiles.

Previamente se presentan algunas definiciones útiles:

## Benchmark

Un benchmark es una medida estándar que se utiliza para evaluar el rendimiento de un sistema o componente, ya sea de hardware o de software. Estos benchmarks pueden consistir en programas o pruebas diseñadas para representar situaciones de uso real o no, siempre con el fin de medir el rendimiento relativo de diferentes sistemas o componentes.

- **Benchmarks sintéticos**: Estos son programas pequeños diseñados para representar una carga de trabajo específica de forma simplificada. Se suelen utilizar para realizar pruebas rápidas y repetibles en el rendimiento de un componente o sistema en condiciones controladas. Los benchmarks sintéticos pueden ofrecer una visión general del rendimiento, pero a veces no reflejan de manera precisa el rendimiento en situaciones del mundo real.
- **Benchmarks reducidos**: Son benchmarks que se enfocan en una parte específica o reducida del sistema o componente a evaluar.
- **Benchmark kernel o de núcleo**: Este término se refiere a un benchmark que se centra en la evaluación del rendimiento de las funciones básicas o fundamentales de un sistema o componente.
- **Programas reales**: Estos son benchmarks que utilizan aplicaciones y escenarios de uso real para evaluar el rendimiento de un sistema o componente. Simulan condiciones de uso más realistas y pueden proporcionar una evaluación más precisa del rendimiento en situaciones del mundo real.

## Lista de Benchmarks

| Benchmark |
|---|
| Phoronix Test Suite |
| VMmark |
| Gzip-Benchmark |
| Hyperfine |
| Google Benchmark |
| Iperf |
| Bonnie++ |
| Linpack |
| Heaven Benchmark |

De los benchmarks listados, los que podrían ser más útiles para las tareas que realizamos a diario serían:

Phoronix Test Suite, Gzip-Benchmark, Hyperfine, Google Benchmark, Bonnie++, Linpack.

A continuación una lista con los benchmarks anteriores y tareas típicas o ejemplos sobre para qué se podrían utilizar.

**Benchmarks asociados a tareas cotidianas:**

| Tarea | Benchmark más útil |
|---|---|
| Compilación de código | Phoronix Test Suite (incluye una variedad de benchmarks de compilación de software) |
| Virtualización con KVM (kernel-based virtual machine) | VMmark (para pruebas de rendimiento de virtualización) |
| Compresión y descompresión de archivos | Gzip-Benchmark |
| Uso general de línea de comandos | Hyperfine |
| Rendimiento de programas en C++ | Google Benchmark |
| Rendimiento de la red (por ejemplo, transferencia de archivos a través de SSH) | Iperf (para pruebas de rendimiento de red) |
| Rendimiento de operaciones de E/S de disco | Bonnie++ (para pruebas de rendimiento de disco) |
| Cálculos matemáticos | Linpack |
| Pruebas de rendimiento y estabilidad para GPU | Heaven Benchmark (para pruebas de rendimiento gráfico) |

# Rendimiento de procesadores para compilar el kernel de Linux

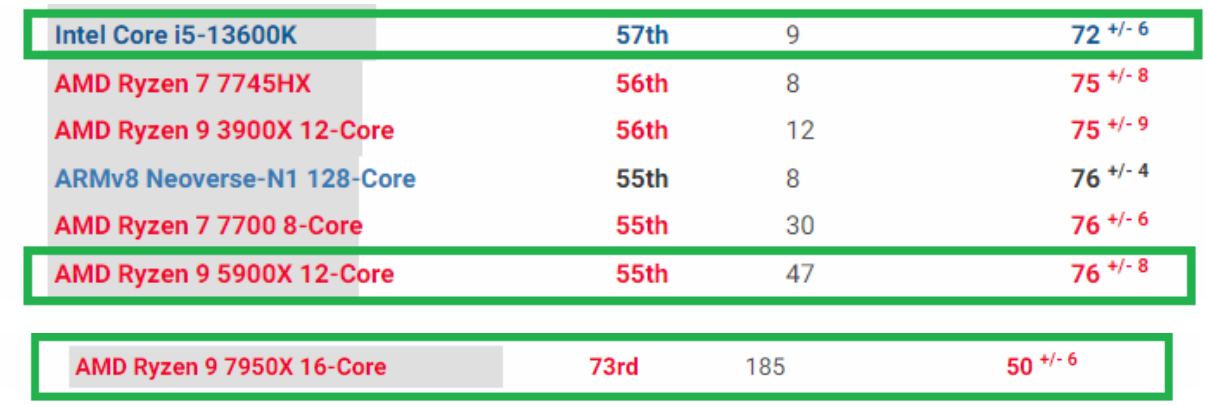
En este punto se solicita calcular el rendimiento para compilar el kernel de Linux de los siguientes procesadores:

- Intel Core i5-13600K
- AMD Ryzen 9 5900X 12-Core

El rendimiento se define rendimiento de un sistema como la capacidad que tiene dicho sistema para realizar un trabajo en un determinado tiempo. Es inversamente proporcional al

tiempo, osea que a mayor tiempo de ejecución de dicho trabajo menor será el rendimiento. En nuestro caso se evalúa el tiempo en compilar el kernel de linux.

Con el sitio https://openbenchmarking.org/test/pts/build-linux-kernel&eval=2e091b8032ec35a1fdb1b678beec9531ea9c956b#metrics podremos encontrar el resultado de test de rendimiento de múltiples usuarios para múltiples procesadores para compilar el kernel de Linux.



| | | | |
|---|---|---|---|
| Intel Core i5-13600K | 57th | 9 | 72 +/- 6 |
| AMD Ryzen 7 7745HX | 56th | 8 | 75 +/- 8 |
| AMD Ryzen 9 3900X 12-Core | 56th | 12 | 75 +/- 9 |
| ARMv8 Neoverse-N1 128-Core | 55th | 8 | 76 +/- 4 |
| AMD Ryzen 7 7700 8-Core | 55th | 30 | 76 +/- 6 |
| AMD Ryzen 9 5900X 12-Core | 55th | 47 | 76 +/- 8 |
| AMD Ryzen 9 7950X 16-Core | 73rd | 185 | 50 +/- 6 |

## Calculo de rendimiento para Intel Core i5-13600K

El tiempo promedio de ejecución del kernel de Linux para el procesador **Intel Core i5-13600K** es de $72 \pm 6\ [s]$, y calculando el rendimiento:

$$\eta_{INTEL\ CORE\ i5-13600K} = \frac{1}{T_{EX\ I5}} = \frac{1}{72}$$

$$\eta_{INTEL\ CORE\ i5-13600K} = 0.0139 = 1.39\ \%$$

## Cálculo de rendimiento para AMD Ryzen 9 5900X 12-Core

Como se pudo observar el tiempo promedio que tarda en compilar el kernel de linux el procesador AMD Ryzen 9 5900X 12-Core es de $76 \pm 8\ [s]$. Considerando este dato se puede calcular el rendimiento del procesador de la forma:

$$\eta_{AMD\ Ryzen\ 9\ |\ 12-core} = \frac{1}{T_{EX-AMD}} = \frac{1}{76\ [s]} = 0,0131$$

$$\eta_{AMD\ Ryzen\ 9\ |\ 12-core} = 1,31\%$$

# Aceleración cuando usamos AMD Ryzen 9 7950X 16-Core

La aceleración o **speedup** es la razón entre el rendimiento de un sistema mejorado y el rendimiento de su implementación original. Para este caso se tomó como rendimiento mejorado el del procesador AMD Ryzen 9 7950X 16-Core y como original, se utilizaron tanto el del procesador AMD Ryzen 9 5900X 12-Core como el del Intel Core i5-13600K.

## Cálculo de rendimiento AMD Ryzen 9 7950X 16-Core

el tiempo promedio de ejecución del kernel de linux para el procesador AMD Ryzen 9 7950X 16-Core es de $50 \pm 6\ [s]$. Con este dato podemos calcular el rendimiento:

$$\eta_{AMD\ Ryzen\ 9\ |\ 16-core} = \frac{1}{T_{EX-AMD}} = \frac{1}{50[s]} = 2\%$$

## Cálculo de speedup

Con los datos previamente calculados se procede a calcular el speedup cuando usamos un procesador **AMD Ryzen 9 7950X 16-Core.**

$$Speedup = \frac{\eta_{R9-16\ CORE}}{\eta_{I5\ 13600}} = \frac{T_{I5\ 13600}}{T_{R9-16\ CORE}} = \frac{72}{50} = 1.44$$

$$Speedup = \frac{\eta_{R9-16\ CORE}}{\eta_{R9-\ 12\ CORE}} = \frac{T_{I5\ 13600}}{T_{R9-16\ CORE}} = \frac{76}{50} = 1.52$$

Con el speedup podemos comparar los rendimientos del AMD Ryzen 9 5900X y el Intel Core i5-13600K respecto al AMD Ryzen 9 7950X 16-Core. Como observamos en el cálculo el AMD Ryzen 9 7950X 16-Core tendrá una mejora respecto a los dos anteriores procesadores, es decir, tendremos un tiempo de ejecución menor.

### Comparación de precios

(precios obtenidos de https://pcpartpicker.com/ )

**Intel Core i5-13600K = 284 usd**

**AMD Ryzen 9 5900X 12-Core= 278 usd**

**AMD Ryzen 9 7950X 16-Core= 549 usd**

Se puede observar que la diferencia de precios entre el procesador de 16 núcleos es notoria respecto de los otros dos.

Podemos calcular la eficiencia respecto al precio con la fórmula:

$$Eficiencia = \frac{\eta}{precio}$$

$$eficiencia = 1.31/284 = 0.489\%$$

$$eficiencia = 1.31/284 = 0.471\%$$

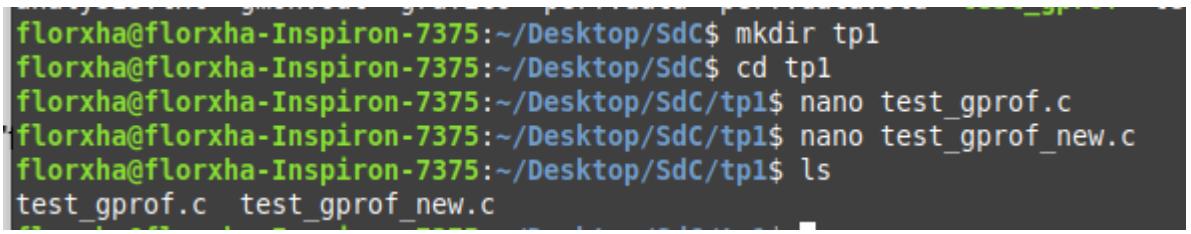
$$eficiencia = 1.31/284 = 0.364\%$$

Con los cálculos previos se puede concluir que el procesador que tiene mejor relación calidad-precio es el AMD Ryzen 9 5900X 12-Core.

# Profiling

## Pasos

1. Primero se crea la carpeta y los dos archivos *test_gprof.c* y *test_gprof_new.c* con sus funciones corresponndientes:



2. Cuando compilamos los archivos agregamos la opción *-pg* que habilita la generación de perfiles:

```
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ ls
test_gprof  test_gprof.c  test_gprof_new.c
```

3. Ejecutamos el binario *test_gprof* para que genere la información de perfiles

```
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ ./test_gprof

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ ls
gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
```

4. Ejecutamos gprof con el nombre del ejecutable y el 'gmon.out' generado anteriormente como argumento. Esto produce un archivo de análisis que contiene toda la información de perfil deseada en el archivo *analysis.txt*.

```
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ gprof test_gprof gmon.out > analysis.txt
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ ls
analysis.txt  gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
```

5. Abrimos el archivo de texto con el comando *cat analysis.txt*

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 33.68      9.18     9.18        1     9.18    18.22  func1
 33.16     18.22     9.04        1     9.04     9.04  new_func1
 33.02     27.22     9.00        1     9.00     9.00  func2
  0.15     27.26     0.04                             main
```

- **%:** el porcentaje del tiempo total de funcionamiento del programa de tiempo utilizado para esta función.
- **cumulative:** una suma corriente del número de segundos contabilizados, segundos por esta función y las enumeradas anteriormente.
- **self:** el número de segundos contabilizados por este los segundos funcionan solos. Este es el tipo principal para este listado.
- **calls:** al número de veces que se invocó esta función, si esta función está perfilada, de lo contrario en blanco.
- **self:** el número promedio de milisegundos gastados en este ms/función de llamada por llamada, si esta función está perfilada, de lo contrario en blanco.
- **total:** el número promedio de milisegundos gastados en este función ms/call y sus descendientes por llamada si la función está perfilada, de lo contrario, en blanco.
- **name:** el nombre de la función. Este es el tipo menor para este listado. El índice muestra la ubicación de la función en la lista de gprof. Si el índice es entre paréntesis muestra dónde aparecería en el listado de gprof si fuera a ser impreso.

```
index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.04   27.22                 main [1]
                9.18    9.04       1/1           func1 [2]
                9.00    0.00       1/1           func2 [4]
-----------------------------------------------
                9.18    9.04       1/1           main [1]
[2]     66.8    9.18    9.04       1         func1 [2]
                9.04    0.00       1/1           new_func1 [3]
-----------------------------------------------
                9.04    0.00       1/1           func1 [2]
[3]     33.2    9.04    0.00       1         new_func1 [3]
-----------------------------------------------
                9.00    0.00       1/1           main [1]
[4]     33.0    9.00    0.00       1         func2 [4]
-----------------------------------------------
```

Esta tabla describe el árbol de llamadas del programa y fue ordenada por la cantidad total de tiempo empleado en cada función y sus hijos.

Cada entrada en esta tabla consta de varias líneas. La línea con el número de índice en el margen izquierdo enumera la función actual. Las líneas arriba enumeran las funciones que llamaron a esta función, y las líneas debajo enumeran las funciones a las que llama.

Esta línea enumera:

- **index:** Un número único dado a cada elemento de la tabla. Los números de índice se ordenan numéricamente. El número de índice está impreso al lado de cada nombre de función para que sea más fácil buscar dónde está la función en la tabla.
- **% time:** Este es el porcentaje del tiempo 'total' que se dedicó en esta función y sus hijos. Tenga en cuenta que debido a diferentes puntos de vista, funciones excluidas por opciones, etc. estos números NO sumarán 100%.
- **self:** Esta es la cantidad total de tiempo empleado en esta función.
- **children:** Esta es la cantidad total de tiempo propagado en esta función por sus hijos.
- **calls:** Este es el número de veces que se llamó a la función. Si la función se llama a sí misma recursivamente, el número solo incluye llamadas no recursivas, y es seguido por un `+' y el número de llamadas recursivas.
- **name:** El nombre de la función actual. El número de índice es impreso después de él. Si la función es miembro de un ciclo, el número de ciclo se imprime entre el el nombre de la función y el número de índice.

(…)

```
Index by function name

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

# Customize gprof output using flags

- -a para no imprimir funciones privadas

```
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ gprof -a test_gprof gmon.out > analysis.txt
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 66.69     18.18    18.18        2     9.09    13.61  func1
 33.16     27.22     9.04        1     9.04     9.04  new_func1
  0.15     27.26     0.04                             main
```

- -b para eliminar los textos de detalle

```
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ gprof -b test_gprof gmon.out > analysis.txt
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 33.68      9.18     9.18        1     9.18    18.22  func1
 33.16     18.22     9.04        1     9.04     9.04  new_func1
 33.02     27.22     9.00        1     9.00     9.00  func2
  0.15     27.26     0.04                             main


                     Call graph


granularity: each sample hit covers 4 byte(s) for 0.04% of 27.26 seconds

index % time    self  children    called     name
                                               <spontaneous>
[1]    100.0    0.04   27.22                 main [1]
                9.18    9.04       1/1           func1 [2]
                9.00    0.00       1/1           func2 [4]
-----------------------------------------------
                9.18    9.04       1/1           main [1]
[2]     66.8    9.18    9.04       1         func1 [2]
                9.04    0.00       1/1           new_func1 [3]
-----------------------------------------------
                9.04    0.00       1/1           func1 [2]
[3]     33.2    9.04    0.00       1         new_func1 [3]
-----------------------------------------------
                9.00    0.00       1/1           main [1]
[4]     33.0    9.00    0.00       1         func2 [4]
-----------------------------------------------


Index by function name

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

- -p para imprimir solo el perfil plano

```
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ gprof -p -b test_gprof gmon.out > analysis.txt
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 33.68      9.18     9.18        1     9.18    18.22  func1
 33.16     18.22     9.04        1     9.04     9.04  new_func1
 33.02     27.22     9.00        1     9.00     9.00  func2
  0.15     27.26     0.04                             main
```

- Para funciones específicas:

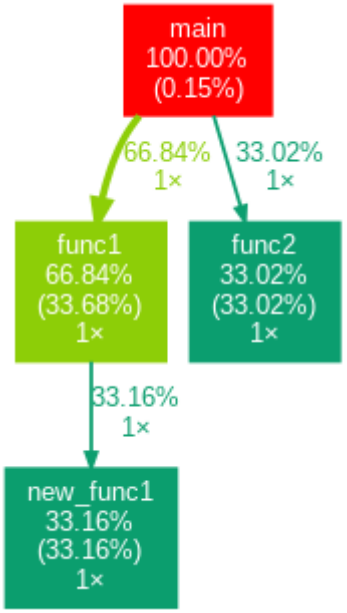
```
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ gprof -pfunc1 -b test_gprof gmon.out > analysis.txt
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.00      9.18     9.18        1     9.18     9.18  func1
```

## Para graficar

Convertimos el archivo de perfil a un formato entendible para gprof2dot

1. convertimos el archivo .dot en una imagen para poder visualizarla

```
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ gprof2dot -f prof analysis.txt -o salida.dot
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ dot -Tpng salida.dot -o salida.png
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ ls
analysis.txt  gmon.out  salida.dot  salida.png  test_gprof  test_gprof.c  test_gprof_new.c
```




## Profiling con linux perf

Con el siguiente comando se graban eventos de rendimiento, como el tiempo de CPU, los ciclos de CPU, las llamadas al sistema, etc. Esto lo podemos hacer con la herramienta perf

```
florxha@florxha-Inspiron-7375:~/Desktop/SdC/tp1$ sudo perf record ./test_gprof
[sudo] password for florxha:

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
[ perf record: Woken up 15 times to write data ]
[ perf record: Captured and wrote 4,185 MB perf.data (109155 samples) ]
```

Para visualizar el análisis de rendimiento ejecutamos el comando *sudo perf report*

```
  33,51%  test_gprof  test_gprof           [.] func1
  32,94%  test_gprof  test_gprof           [.] func2
  32,84%  test_gprof  test_gprof           [.] new_func1
   0,13%  test_gprof  test_gprof           [.] main
```