El rendimiento de las computadoras

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**, es decir, cuanto mayor sea el tiempo que necesite, menor será el rendimiento.

Los computadores ejecutan las instrucciones que componen los programas, por lo tanto el rendimiento de un computador está relacionado con el tiempo que tarda en ejecutar los programas. De esto se deduce que el **tiempo** es la medida del rendimiento de un computador.

Comparación de desempeño

- Comparación en términos relativos (no absolutos)
- Un sistema A tiene mejor rendimiento que un sistema B si el sistema A tiene menor tiempo de ejecución (para un conjunto de programas) que el sistema B.

$$\frac{Rendimiento_A}{Rendimiento_B} = \frac{1/EX_{CPU\,A}}{1/EX_{CPU\,B}} = \frac{EX_{CPU\,B}}{EX_{CPU\,A}}$$

Medidas de desempeño

- Ciclos por instrucción (CPI)
- Instrucciones por ciclo (IPC)
- Rendimiento (Throughput)
- Latencia
- Speedup
- Eficiencia

Rendimiento del procesador

El rendimiento del procesador depende de los siguientes parámetros:

1. **Frecuencia** de la CPU (fCPU) : es el número de ciclos por segundo al que trabaja el procesador o CPU. No confundir la frecuencia de la CPU con la frecuencia del sistema, el bus del sistema trabaja a menor frecuencia que la CPU.

$$f_{CPU} = \frac{n^{\circ}\ ciclos}{segundo}$$

2. **Periodo** de la CPU (TCPU) : es el tiempo que dura un ciclo y es la inversa de la frecuencia de la CPU.

$$T_{CPU} = \frac{1}{f_{CPU}}$$

3. **Ciclos por instrucción** (CPI) : las instrucciones se descomponen en microinstrucciones, que son operaciones básicas que se hacen en un ciclo de reloj. En un programa se llama CPI al promedio de microinstrucciones que tienen las instrucciones del programa, es decir, los ciclos de reloj que se tarda de media en ejecutar una instrucción.

$$CPI = \frac{\sum_{i=1}^{n} N^{o}\ Instruc_i * CPI_i}{N^{o}\ InstrucTot}$$

4. **Número de instrucciones del programa** : cuantas más instrucciones haya en el programa más tiempo se tarda en ejecutarlo luego baja el rendimiento. El que tengamos un número reducido de instrucciones dependerá del programador y de que dispongamos de un buen compilador.

5. **Multitarea** : hace referencia a la capacidad que tiene un computador de atender simultáneamente varias tareas.

Práctico

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 ?

Rendimiento

$$T_{instrucción} = CPI * T_{CPU}$$

$$T_{prog} = N° \text{ instrucciones} * CPI * T_{CPU}$$

$$\eta_{prog} = \frac{1}{T_{prog}} = \frac{1}{N° instruc * CPI * T_{CPU}} = \frac{f_{CPU}}{N° instruc * CPI} \; s^{-1}$$

Comparación de desempeño

El speedup es la razón entre el rendimiento de un sistema mejorado y el rendimiento de su implementación original

$$Speedup = \frac{Rendimiento\ Mejorado}{Rendimiento\ Original} = \frac{EX_{CPU\ Original}}{EX_{CPU\ Mejorado}}$$

Eficiencia

- Mientras el speedup es la ganancia por mejorar un sistema.
- La eficiencia mide la utilización de un recurso.
- Si Speedup n es la ganancia por mejorar el sistema con n recursos, la eficiencia mide la utilización de esos recursos.

$$Eficiencia = \frac{Speedup_n}{n}$$

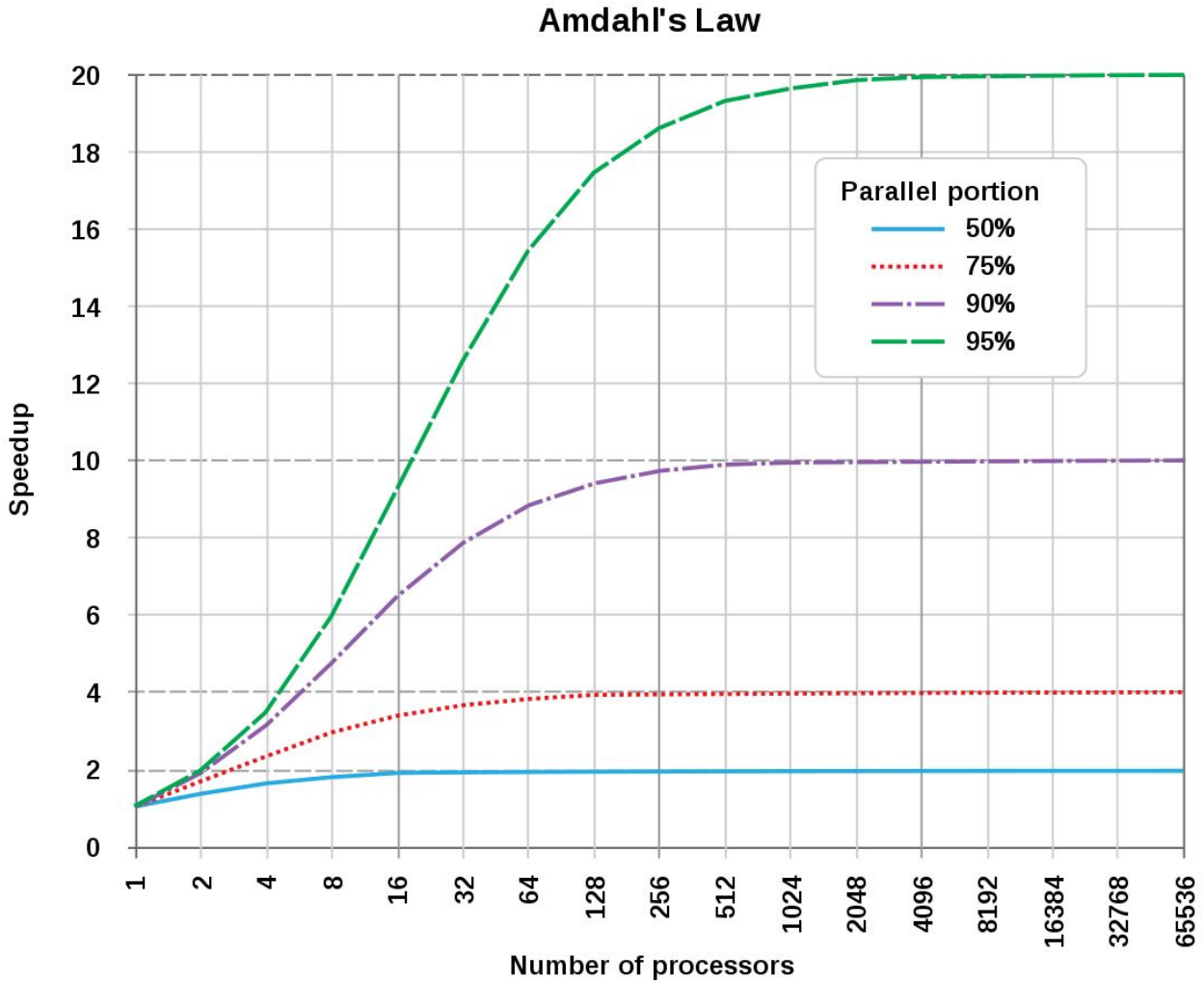
MIPS/FLOPS

Los MIPS son los millones de instrucciones por segundo que ejecuta un procesador para un programa determinado.

Floating point operations per second

Los FLOPS por sí solos no son un estándar muy útil para computadoras modernas. Existen muchos otros factores de rendimiento tales como E/S (entrada-salida), comunicación interprocesador, coherencia del caché y jerarquía de memoria. Esto significa que las computadoras en general son solamente capaces de una fracción del pico teórico en FLOPS, obtenido adicionando el pico teórico en FLOPS de cada uno de los componentes del sistema. Aun cuando se trabaje en problemas grandes y altamente paralelos, su rendimiento será irregular, debido en gran medida a efectos residuales de la ley de Amdahl.

ley de amdahl



Programas de prueba o benchmark

La mejor y más fiable forma de calcular el rendimiento es medir el tiempo que los diversos computadores tardan en ejecutar los programas que realmente el usuario va a utilizar posteriormente. Ese será el mejor rendimiento para ese usuario, pero no para todos los usuarios, ya que el rendimiento es un valor relativo de acuerdo con la aplicación que se va a hacer.

El rendimiento de una estación de trabajo se mide analizando una serie de componentes físicos que determinan el rendimiento completo del sistema. A la hora de determinar el rendimiento global de un sistema, también hay que evaluar el sistema operativo, los equipos lógicos de red, los compiladores y las librerías gráficas, etc.

BENCHMARK

BENCHMARK SINTÉTICOS

generalmente pequeños, escritos originalmente utilizados para comprobar el comportamiento ante un tipo determinado de carga.

BENCHMARKS REDUCIDOS

BENCHMARK KERNEL O DE NÚCLEO

PROGRAMAS REALES

Ejercicio

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 que benchmark la representa mejor.

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

https://www.tomshardware.com/reviews/cpu-hierarchy,4312.html

cual es el rendimiento de estos procesadores para compilar el kernel

Intel Core i5-13600K

AMD Ryzen 9 5900X 12-Core

Cual es la aceleración cuando usamos un AMD Ryzen 9 7950X 16-Core, cual de ellos hace un uso más eficiente de la cantidad de núcleos que tiene? y cuál es más eficiente en términos de costo ?

Profiling

Cuando escribimos código, generalmente tenemos cierta "intuición" de cuán eficiente es (tiempo de ejecución y/o uso de memoria) Eso podría estar respaldado por un análisis de complejidad de O grande de los algoritmos que usamos (por ejemplo, O(logN), O(N), etc).

Desafortunadamente, eso no significa que estemos en lo correcto, por lo que a menudo queremos una validación experimental de qué tan rápido se ejecuta (o cuánta memoria usa) en la práctica. Las herramientas para analizar el tiempo de ejecución del programa/uso de memoria se llaman generadores de perfiles (“profilers”).

Cómo funcionan los perfiladores de código (tiempo). Los generadores de perfiles de código a menudo se usan para analizar no solo cuánto tiempo tarda en ejecutarse un programa (podemos obtenerlo de herramientas a nivel de shell como /usr/bin/time), sino también cuánto tiempo tarda en ejecutarse cada función o método (tiempo de CPU). Dos técnicas principales utilizadas por los perfiladores: **inyección de código, muestreo.**

Análisis de memoria

Además del tiempo de ejecución, a menudo queremos analizar cuánta memoria utiliza un programa. Ideas de implementación similares, pero implica la inserción de código para verificar cuánta memoria está en uso por el programa en los puntos de control. También podría incluir código para rastrear dinámicamente cada elemento de memoria asignado, para monitorear su tamaño, ver si está liberado o no, verificar si hay fugas de memoria, etc.

Profiling

Tiempo: inyección de código (gprof)

Estos perfiladores requieren volver a compilar el programa con banderas especiales. En cada punto donde se realiza una llamada/retorno de función, se inserta este código para verificar la hora actual del sistema (hasta microsegundos) y registrar esa información. El programa de creación de perfiles lee y resume el archivo de registro, generando un informe sobre cuánto tiempo se dedicó a cada función/método, cuántas veces se llamó, etc.

Tiempo: sampling (perf)

Estos requieren privilegios de nivel de sistema operativo. El programa se ejecuta en un modo especial, con interrupciones del sistema operativo generadas a intervalos fijos. En cada una de estas interrupciones, verifica y registra qué función/método está ejecutando el programa actualmente. Esto acumula datos estadísticos sobre cuánto tiempo se dedicó a cada función. El perfilador analiza e informa sobre los datos estadísticos.

Memoria (Heaptrack)

Heaptrack rastrea todas las asignaciones de memoria y anota estos eventos con seguimientos de pila. Luego, las herramientas de análisis dedicadas le permiten interpretar el perfil de la memoria del montón para:

- encontrar puntos de acceso que necesitan ser optimizados para reducir la huella de memoria de su aplicación
- encontrar fugas de memoria, es decir, ubicaciones que asignan memoria que nunca se desasigna
- encontrar puntos de acceso de asignación, es decir, ubicaciones de código que desencadenan muchas llamadas de asignación de memoria
- encontrar asignaciones temporales, que son asignaciones seguidas directamente por su desasignación

heaptrack <your application and its parameters>

heaptrack output will be written to "/tmp/heaptrack.APP.PID.gz"