

Performance

Lucas

26 de setembro de 2015

pg 26 O transistor é um interruptor simples de ligar/desligar, controlado por eletricidade. O circuito integrado (CI) combina dezenas de centenas de transistores em um único chip. VLSI (milhares de transistores), chamamos de circuito integrado de larguíssima escala (very large-scale integrated circuit).

pg 28 Tempo de Resposta (Responde Time) também chamado Tempo de Execução. O tempo total requerido para o computador completar uma tarefa qualquer, incluindo acessos ao disco, memória, I/O, overhead do sistema operacional, tempo de execução da CPU, etc.

Vazão (Throughput) também chamado de largura de banda. É outra medida de desempenho, é o número de tarefas completadas por unidade de processamento.

pg 29 Na página 29 temos uma equação simples, comparativa: $Performance_x = \frac{1}{Execution\ Time_x}$. Se o desempenho de x é melhor que o de y então $Performance_x > Performance_y$ ou ainda $\frac{1}{Execution\ Time_x} > \frac{1}{Execution\ Time_y}$ ou também $Execution\ Time_x < Execution\ Time_y$. Isso significa que o tempo de execução em y é mais longo que o tempo de execução em x , bastante simples.

$$\frac{Performance_x}{Performance_y} = \frac{Execution\ Time_y}{Execution\ Time_x} = n$$

pg 30 O conceito de medir desempenho de computador se baseia em: entender qual montante de (trabalho) ele estará processando, em um dado tempo. O processador que realizar o mesmo montante no menor tempo, teoricamente é mais rápido. Tempo de execução do programa é medido em segundos por programa. A definição mais direta é chamada de 'wall clock time', 'response time', ou 'elapsed time'. Esses termos significam o total para uma tarefa ser completada.

pg 31 Por consistência, vamos manter a distinção entre desempenho, baseado em 'elapsed time' e por sua vez baseada em CPU 'execution time'. Clock cycle (Ciclos de Relógio) também chamado de 'clock tick', 'period', 'cycle'. É o tempo para um período de relógio (geralmente em picosegundos, microsegundos, nanosegundos, ou segundos, depende do problema), usualmente o relógio do processador, o qual está rodando à uma certa taxa constante (por exemplo: 4 GHz)

pg 32 Uma fórmula simples que relaciona fatores simples é:

$$CPU \text{ execution time for a program} = CPU \text{ clock cycles for a program} \times Clock \text{ cycle time}$$

Um exemplo genérico de melhoramento de performance usando essa fórmula acima é:

$$CPU \text{ time}_A = \frac{CPU \text{ clock cycles}_A}{Clock \text{ rate}_A}$$

$$\frac{Seconds}{Program} = \frac{CPU \text{ clock cycles}_A}{f \times 10^n \text{ Hz}}$$

$$CPU \text{ clock cycles}_A = (t \text{ [s]}) \times (f \times 10^n \text{ [Hz]})$$

Onde t é por exemplo: 12 segundos, ou 3 nanosegundos. E $f \times 10^n$ é por exemplo: 9×10^9 hertz, ou 9 Gigahertz. E assim por diante.

$$\frac{Seconds}{Program} = \frac{(CPU \text{ clock cycles}_B = CPU \text{ clock cycles}_A \times scalar \text{ cost})}{Clock \text{ Rate}_B}$$

assim, podemos simplificar para:

$$\frac{Seconds}{Program} = \frac{(CPU \text{ clock cycles}_A \times scalar \text{ cost})}{Clock \text{ Rate}_B}$$

Ou seja se eu tenho um computador A que processa um programa qualquer em 13 segundos, sendo que esse tem 3 GHz. Para que eu gere um computador B que processe o mesmo 30% mais rápido que o computador A. Eu precisaria que ele tivesse 4,68 GHz se quisemos que o CPU B processasse a mesma informação em 10 segundos (3 segundos a menos, melhoria de 30%), pagando um custo de 1,4x ciclos a mais, aí vai do seu critério, quanto você quer pagar a mais?

Changes	$\frac{Seconds}{Program} = \frac{(CPU \text{ clock cycles}_A \times scalar \text{ cost})}{Clock \text{ Rate}_B}$
$\uparrow \frac{Seconds}{Program}_B$ (e.g: 3 s \rightarrow 4 s)	Decresce $Clock \text{ rate}_B$
$\uparrow (CPU \text{ clock cycles}_A \times improvement)$	Aumenta $Clock \text{ rate}_B$

pg 33 O termo clock cycles per intruccion CPI, o qual é a média numérica de ciclos de clock para cada instruções executada, varia bastante de classes de instruções para classes de instruções, exemplos o CPI de instruções do tipo R não vai ser o mesmo CPI para instruções do tipo I, de fato instruções do tipo I contém em seu corpo branch, loads, stores, essas instruções contém veneno para o datapath, da forma que ele leva mais ciclos para processar.

$$CPU \text{ clock cycles} = Instructions \text{ for a program} \times Average \text{ clock cycles per instruction}$$

Suponha você agora que temos um processador X com tempo para cada ciclo de relógio de N pico segundos e em média faz s ciclos por instrução (CPI), e um processador Y com tempo de clock cycle de K pico segundos e em média faz p ciclos por instrução (CPI). Quem é mais rápido?

$CPU\ clock\ cycles_A = (I\ \mathbf{instr.} \times s\ \mathbf{CPI})\ \mathbf{ciclos}$

$CPU\ clock\ cycles_B = (I\ \mathbf{instr.} \times p\ \mathbf{CPI})\ \mathbf{ciclos}$

$CPU\ time_A = CPU\ clock\ cycles_A \times Clock\ cycle\ time^a$

$CPU\ time_A = I \times s \times N\ \mathbf{pico\ segundos}$

$CPU\ time_B = CPU\ clock\ cycles_B \times Clock\ cycle\ time$

$CPU\ time_B = I \times p \times K\ \mathbf{pico\ segundos}$

Basta comparar ambos os resultados de tempo de CPU e ver qual se desempenhou melhor.

$$\frac{CPU\ time_B}{CPU\ time_A} = \frac{I \times p \times K}{I \times s \times N} = \frac{p\ \mathbf{CPI} \times K\ \mathbf{pico\ segundos}}{s\ \mathbf{CPI} \times N\ \mathbf{pico\ segundos}}$$

Se por acaso, $p \times K > s \times N$ então CPU_B se desempenha melhor no conjunto de instruções que você passou como parâmetro. Caso contrário CPU_A se desempenha melhor.

$1,7\ \mathbf{CPI} \times 190\ \mathbf{pico\ segundos} < 2\ \mathbf{CPI} \times 180\ \mathbf{pico\ segundos}$ pois $323 < 360$ então CPU_A tem 11% menos desempenho que CPU_B .

^aO fator tempo de ciclo de relógio, é o tempo de 1 ciclo de relógio. O montante total de ciclos para executar um programa completamente, executar uma tarefa qualquer, vai levar (pensando logicamente) $n\ \mathbf{cycles} \times clock\ cycle\ time$.

1 Questões de performance

1.3.1

1.3.2

1.3.4

1.4.1

1.4.2

1.5.1a

1.6.1a

1.7.2

1.7.3

1.7.4

1.8.1

1.8.2

1.9.3a

1.10.2a

1.10.4a

1.10.5a

1.10.6a