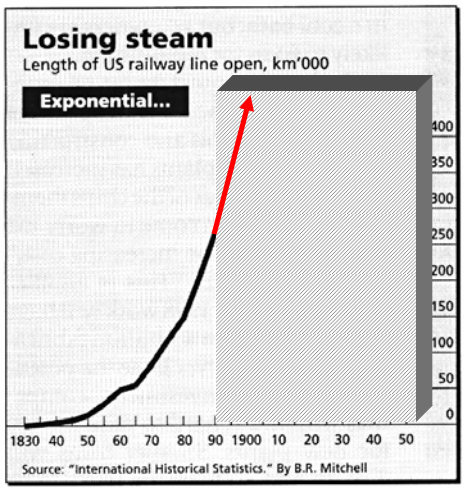
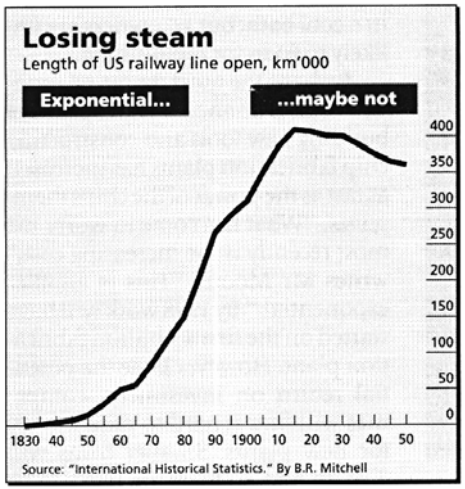
# Aula 03: Análise de Performance e Benchmarks

## O Perigo das Previsões...

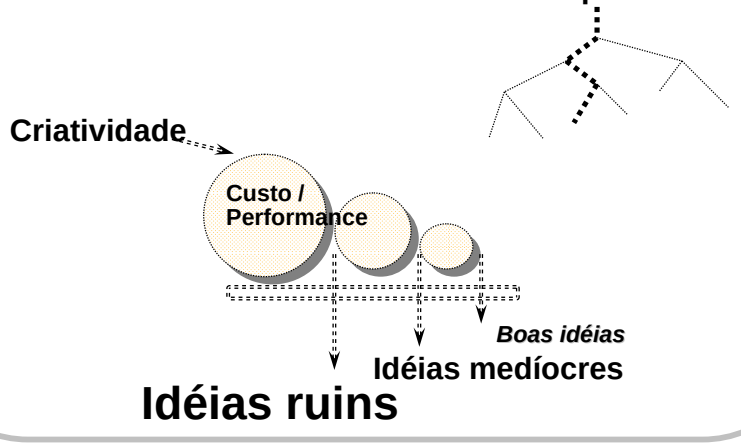



## O Perigo das Previsões...




## Medidas e Avaliações
## A Era RISC

**Arquitetura como processo iterativo:**
- **Busca da solução ótima em todos os níveis dos sistemas de computadores**



## Ferramentas para Medidas de Performance

- *Benchmarks, Traces, Mixes*
- Custo, atraso, área, estimativa de potência
- Simulação (muitos níveis)
  - ISA, RT, *Gate*, Circuito
- Teoria das filas
- Basic Rules
- Leis fundamentais

## Performance (e Custo)

| Avião | DC à Paris | Veloc. | Passageiros | *Throughput* (pmph) |
|---|---|---|---|---|
| **Boeing 747** | 6.5 horas | 610 mph | 470 | 286,700 |
| **BAD/Sud Concorde** | 3 horas | 1350 mph | 132 | 178,200 |

- **Tempo para rodar tarefa (ExTime ou ET)**
  - **tempo de execução, tempo de resposta, latência**
- **Tarefas por dia, hora, semana, sec, ns … (Performance)**
  - ***Throughput*, *bandwidth***

## Performance (e Custo)

**"X é n vezes mais veloz que Y" significa**

$$\frac{ExTime(Y)}{ExTime(X)} = \frac{Performance(X)}{Performance(Y)}$$

- ***Velocidade*** **do Concorde vs. Boeing 747**
- ***Throughput*** **do Boeing 747 vs. Concorde**

## Terminologia de Performance

"X é n% mais veloz que Y" significa:

$$\frac{ExTime(Y)}{ExTime(X)} = \frac{Performance(X)}{Performance(Y)} = 1 + \frac{n}{100}$$

$$n = \frac{100(Performance(X) - Performance(Y))}{Performance(Y)}$$

## Exemplo

**Exemplo: Y leva 15 segundos para completar tarefa, X leva 10 segundos. Quantos % X é mais rápido?**

$$\frac{ExTime(Y)}{ExTime(X)} = \frac{15}{10}$$

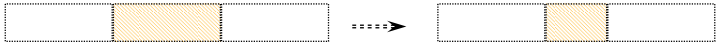
$$n = \frac{100(1.5 - 1.0)}{1.0}$$

$$n = 50\%$$

## Lei de Amdahl

*Speedup* devido à melhoria E:

$$Speedup(E) = \frac{ExTime\ sem\ E}{ExTime\ com\ E} = \frac{Performance\ com\ E}{Performance\ sem\ E}$$



Suponha que melhoria E acelere porção F da tarefa por um fator S, e que restante da tarefa permanece sem alteração, então

**ExTime(E) =**

**Speedup(E) =**

## Lei de Amdahl

$$ExTime_{new} = ExTime_{old} \times \left[ (1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

***Em última análise, performance de qualquer sistema será limitada por porção que não é melhorada...***

## Lei de Amdahl

- **Instruções de ponto flutuante são melhoradas para rodar 2X mais rápidas, mas somente 10% das instruções são de ponto flutuante (FP).**

**$ExTime_{new}$ =**

**$Speedup_{overall}$ =**

# Lei de Amdahl

- **Instruções de ponto flutuante são melhoradas para rodar 2X mais rápidas, mas somente 10% das instruções são de ponto flutuante (FP).**

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times (0.9 + 0.1/2) = \text{ExTime}_{old} \times 0.95$$

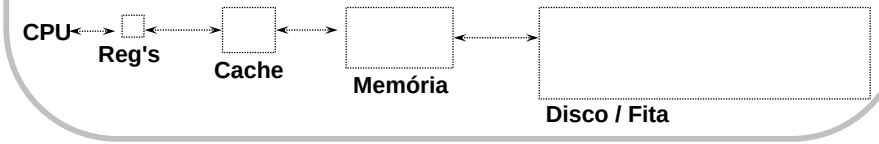
$$\text{Speedup}_{overall} = \frac{1}{0.95} = 1.053$$

# Corolário: *Make the common case fast*

- Todas as instruções requerem busca de instrução, somente uma fração requer busca ou escrita de dados.
  - Otimize acessos à instruções em vez de otimizar o acesso a dados
- Programas exibem localidade de referência

Localidade espacial Localidade temporal

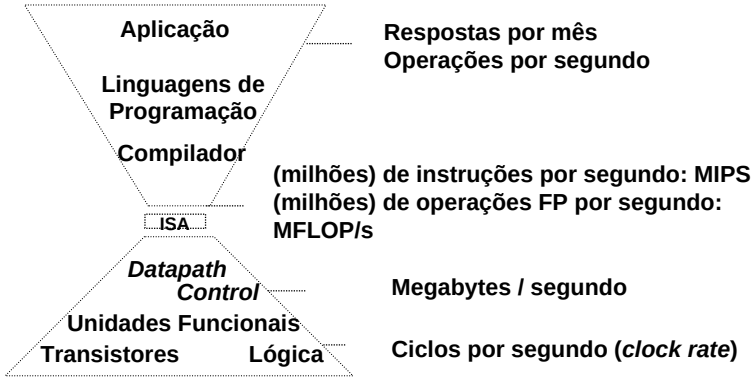
- Acesso a memórias menores é mais rápido
  - Mantenha hierarquia de memória de modo a fazer acessos mais freqüentes estar localizados nas memórias menores.




# Regra Básica: *Make the common case fast*

- Os casos mais simples são usualmente mais freqüentes e mais fáceis de serem otimizados!!!
- Faça as coisas simples e rápidas em hardware e certifique-se de que as demais sejam feitas corretamente em software.

# Métricas de Performance



# Aspectos da Performance de uma CPU

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

| | Instruction Count | CPI | Clock Rate |
|---|---|---|---|
| Programa | X | | |
| Compilador | X | (X) | |
| Conjunto de Instruções | X | X | |
| Organização | X | X | |
| Tecnologia | | | X |

# Métricas de Marketing

**MIPS** = IC / CPU time * 10^6 = Clock Rate / CPI * 10^6

- Máquinas com diferentes conjuntos de instruções ?
- Programas com instruções diferentes ?
  - Freqüência dinâmica das instruções
- Sem correlação com performance

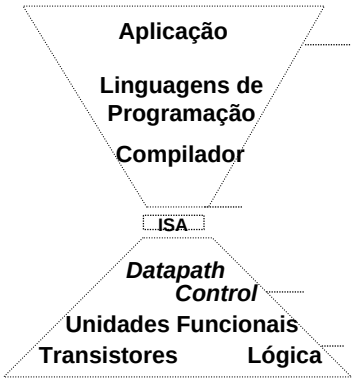
**MFLOP/S** = Operações FP / CPU time * 10^6

- Dependente de máquina
- Tempo da máquina pode estar sendo gasto em outro lugar

**Normalizado:**

| | |
|---|---|
| add,sub,compare,mult | 1 |
| div, sqrt | 4 |
| exp, sin, . . . | 8 |

# Compromissos da Organização da CPU




***Instruction Mix***

**CPI**

***Cycle Time***

# Ciclos por Instrução

*Cycles per Instruction*

**"Média do número de ciclos por instrução"**

$$CPI = \frac{(CPU\ Time \times Clock\ Freq)}{Instrucion\ Count}$$

$$= \frac{Cycles}{Instruction\ Count}$$

$$CPU\ Time = Cycle\ Time \times \sum_{i=1}^{n} CPI_i \times IC_i$$

**"Freqüência de instruções"**

$$CPI = \sum_{i=1}^{n} CPI_i \times F_i \quad \text{onde} \quad F_i = \frac{IC_i}{Instruction\ Count}$$

# Exemplo de Cálculo de CPI

**Máquina Base (Reg / Reg)**

| Operação | Freq | Ciclos | $CPI_i$ | (% Time) |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | (33%) |
| Load | 20% | 2 | .4 | (27%) |
| Store | 10% | 2 | .2 | (13%) |
| Branch | 20% | 2 | .4 | (27%) |
| | | **CPI** | **1.5** | |

Mix Típico

# Exemplo

**Analisar o impacto de acrescentarem-se operações Registrador / Memória na máquina anterior, sabendo-se que:**

- **Um operando em memória**
- **Um operando em registrador**
- **Leva 2 ciclos para completar**
- **Branch passa a levar 3 ciclos para completar.**

**Qual a fração dos Loads tem que ser eliminada para que a modificação seja vantajosa?**

# Solução

$$ExTime = Instr\ Cnt \times CPI \times Clock$$

| Op | Freq | Ciclos | $CPI_i$ | Freq | Ciclos | $CPI_i$ |
|---|---|---|---|---|---|---|
| ALU | 0.50 | 1 | 0.5 | 0.5 – x | 1 | 0.5 – x |
| Load | 0.20 | 2 | 0.4 | 0.2 – x | 2 | 0.4 – 2x |
| Store | 0.10 | 2 | 0.2 | 0.1 | 2 | 0.2 |
| Branch | 0.20 | 2 | 0.4 | 0.2 | 3 | 0.6 |
| Reg/Mem | | | | x | 2 | 2x |
| | 1.00 | | 1.5 | 1 – x | | (1.7 – x)/(1 – x) |

$$\begin{aligned} Instr\ Cnt_{Old} \times CPI_{Old} \times Clock_{Old} &= Instr\ Cnt_{New} \times CPI_{New} \times Clock_{New} \\ 1.00 \times 1.5 &= (1-x) \times \frac{(1.7-x)}{(1-x)} \\ 1.5 &= 1.7 - x \\ 0.2 &= x \end{aligned}$$

TODOS os loads devem ser eliminados para esta versão ser vantajosa!

# Programas para Avaliação da Performance de Processadores

- (Toy) Benchmarks
  - Programas de 10-100 linhas
  - e.g.: sieve, puzzle, quicksort
- Benchmarks sintéticos
  - Tentativa de se ter na média a mesma freqüência de instruções de programas reais
  - e.g., Whetstone, dhrystone
- Kernels
  - Partes críticas de programas reais
  - e.g., *Livermore loops*
- Programas reais
  - e.g., gcc, spice

## Truques utilizados para se ganhar o *Jogo dos Benchmarks*

- Configurações diferentes das máquinas utilizadas para rodar o mesmo programa;
- Compilador otimizado para o benchmark;
- Utilizar benchmark escrito para melhorar performance de uma única máquina;
- Sincronizar jobs intensivos em CPU/IO;
- Utilização de benchmarks pequenos;
- Benchmarks “compilados à mão” para otimizar performance;

## Erros Comuns em *Benchmarking*

- Variações das demandas de dispositivos ignorados
- Nível de carga do sistema controlado de maneira inadequada
- Efeitos de cache ignorados
- Tamanho de *buffers* inapropriados
- Ignorar a imprecisão devido à amostragem

## Erros Comuns em *Benchmarking*

- Ignorar *overhead* dos monitores;
- Medições sem validação;
- Não manter mesmas condições iniciais;
- Não medir performance transiente (*cold start*);
- Coleta de muitos dados com pouca análise;

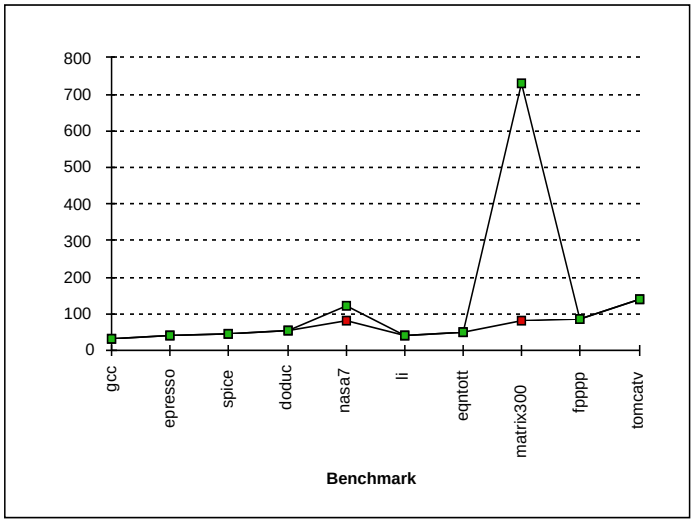
## SPEC: ***S**ystem **P**erformance **E**valuation **C**ooperative*

- Primeiro *Round* 1989
  - 10 programas
- Segundo *Round* 1992
  - SpecInt92 (6 programas inteiros) e SpecFP92 (14 programas de ponto flutuantes)
  - Sem limite para flags de compiladores
- Terceiro *Round* 1995
  - Limite de um único flag para todos os programas
  - Novo conjunto de programas

# SPEC Primeiro *Round*

- Um programa: 99% do tempo em única linha de código
- Fácil para compilador (não testa a máquina)

# SPEC Primeiro *Round*




# Como agrupar dados de medições de Performance

1. Tempo total de Execução
   a. Tempo → Média Aritmética

$$\sum_{i=1}^{n} \frac{T_i}{n}$$

   b. Taxa (rate) → Média Harmônica

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{R_i}}$$

   – *Rastreiam o tempo de execução*

# Como agrupar dados de medições de Performance

2. Tempo de Execução Ponderado
   a. Tempo → Média Aritmética Ponderada

$$\sum_{i=1}^{n} \frac{T_i}{n}$$

   b. Taxa (rate) → Média Harmônica Ponderada

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{R_i}}$$

   – *Rastreiam o tempo de execução*

## Como agrupar dados de medições de Performance

3. Tempo de Execução Normalizado
   a. Média Geométrica

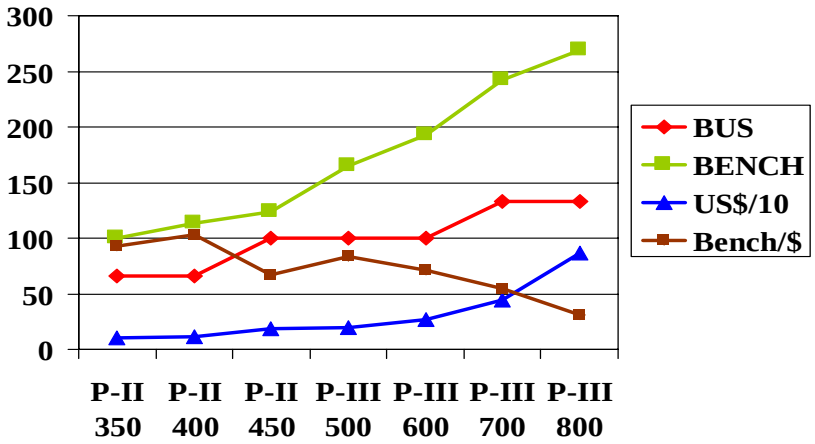
$$\sqrt[n]{\prod_{i=1}^{n} Execution\ time\ ratio_i}$$

*- Não há, necessariamente, correspondência com o tempo de execução*

## Avaliação de Performance

- Venda é uma função da performance relativo à competição. Logo, espere investimentos altos em melhoria da performance do produto dado pela melhoria da performance dos *benchmarks*
- Bons produtos aparecem quando:
  - Bons *benchmarks*
  - Boas maneiras de sumarizar a performance
- Se o *benchmark* ou sumário são inadequados, então melhor o produto para melhorar vendas sempre ganha de outras opções
  Ex. *Tempo* é a única medida real de performance!
- E quanto ao custo?

## Avaliação de Benchmarks / Preços de CPUs




http://www.cpuscorecard.com