2.7 Lendas e Falhas

"Esperar que a melhora em um dos aspectos que influem na performance da máquina resulte em uma melhora na performance total, proporcional ao tamanho do ganho inicial."
Isto é verdade? Por quê?

Exemplo:
Considere que um programa rode em 100 s em uma determinada máquina, e que as operações de multiplicação sejam responsáveis por 80 dos 100 s totais. Quanto deve melhorar a velocidade da multiplicação para que o programa rode 5 vezes mais rápido?

O tempo de execução de um programa após a implementação de uma melhoria é dado por:

Tempo de exec. após = a melhoria	Tempo de execução afetado pela melhora / Montante da melhora +	Tempo de execução não afetado pela melhora

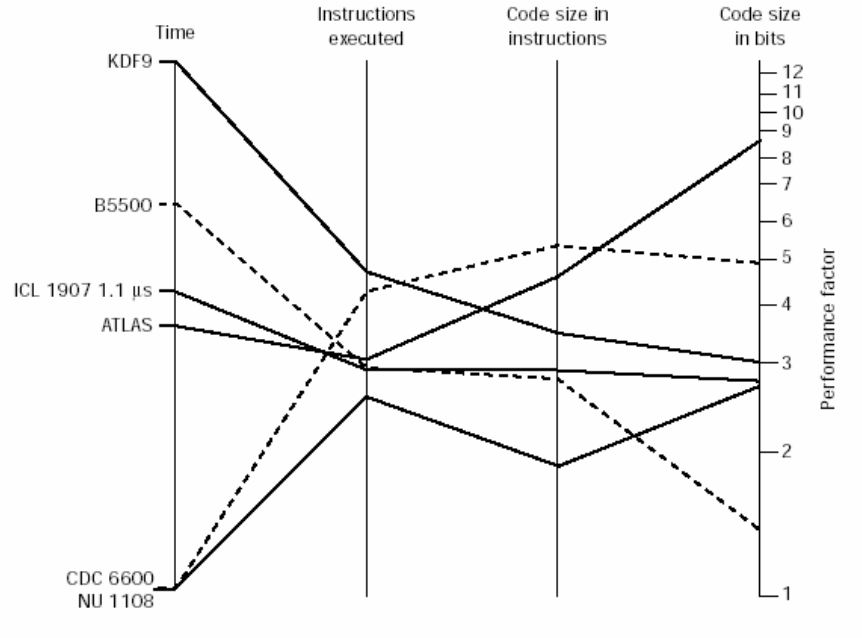
Conclusão: não há como conseguir uma melhora de 5 vezes na multiplicação. A melhoria da performance é limitada pela freqüência de uso da operação que sofreu a melhora.

Portanto essa afirmação é falha!

"As métricas que não dependem do hardware são boas para calcular a performance."

Muitos projetistas têm desenvolvido métodos que não necessitem do tempo de execução.
Este método foi usado com freqüência para **comparar diferentes conjuntos de instruções** e concluir sobre a performance.
Um desses métodos de comparação, usada no passado, foi o **tamanho de código** como uma medida de velocidade.
Arquitetura do conjunto de instruções com **menor programa** na execução de uma tarefa pertence à **máquina mais rápida**.
Este método parecia ser correto já que a memória das máquinas era limitada, mas, usar tamanho de código como medida de performance em arquiteturas diferentes é um erro!

Exemplo:
Compare as relações do tamanho do código executado pelo CDC 6600 e pela B5500.



Se considerarmos a mesma arquitetura para compararmos tamanho de código, ainda termos uma imprecisão na determinação da medida de performance.
Portanto, não devemos considerar o tamanho de código como medida de performance.

Métricas de marketing

"O uso do parâmetro de MIPS como métrica de performance"

MIPS – Milhões de instruções por segundo.

$$MIPS = \frac{\text{Número de instruções}}{\text{Tempo de exec.} \times 10^6}$$

Máquinas mais rápidas > MIPS.

Existem três problemas associados ao uso do MIPS:

1. Considera somente a **taxa de instruções executadas**, mas não considera que uma determinada instrução executa **mais ou menos trabalho** que outra. Portanto, não podemos comparar máquinas com ISA diferentes.
2. MIPS **varia** com programas diferentes no mesmo computador, então não é obtido um valor característico para determinada máquina.
3. MIPS pode variar **inversamente** a performance.

Exemplo:
Considere uma máquina que possua três diferentes classes de instruções, e medidas de CPI:

Classe de instrução	CPI para esta classe de instrução
A	1
B	2
C	3

Suponha que a medida de tamanho do código gerado para o mesmo programa por dois compiladores diferentes tenha apresentado os resultados:

Seqüência de código	Número de instrução para a classe (**em bilhões**)		
	A	B	C
Compilador 1	5	1	1
Compilador 2	10	1	1

A máquina roda com um clock de 500 MHz. Qual a seqüência de código que executa mais rápido de acordo a definição de MIPS? E de acordo com o tempo de execução?
Ao compararmos os valores do MIPS com a relação de performance observamos que o MIPS falha!

MFLOPS (milhões de operações de ponto flutuante por segundo):

$$\text{MFLOPS} = \frac{\text{Número de operações em ponto flutuante de um programa}}{\text{Tempo de exec.} \times 10^6}$$

Uma FLOP pode ser: adição, subtração, multiplicação ou divisão aplicada sobre números com precisão dupla ou simples (float, real, double, double precision).
O problema do MFLOPS é a variação das operações de inteiros e ponto flutuante, ou seja, **varia com o mix** de instruções.

Benchmark sintéticos: programas gerados artificialmente para tentar retratar as características de um grande conjunto de programas.
São exemplos de benchmarks sintéticos:

Whetstone - *Benchmark* sintético para computação numérica. Originalmente desenvolvido em ALGOL-60 para exercitar 42 declarações básicas. Composto de 11 módulos para testar operações matemáticas elementares, processamento de *arrays*, aritmética de inteiros, funções trigonométricas, chamadas de procedimentos, cálculo em ponto flutuante e. desvios condicionais.

```
j = 1;
for (i = 1; i <= N4; i + = 1)
 {
    if (j = 1)
       j = 2;
    else j = 3;
       if (j > 2)
         j = 0;
       else j = 1;
    if (j < 1)
       j = 1;
    else j = 0;
 }
```

Desvantagens dos benchmarks sintéticos:
Benchmarks sintéticos não são reais e não se comportam como programas. Otimizações do hardware e do compilador podem inflar a performance desses benchmarks além das obtidas em programas reais.

Dhrystone. *Benchmark* sintético (Weicker, 1984) para testar o desempenho de declarações Ada, operações e acesso a dados. Foi convertido para outras linguagens como o C, com o seguinte *mix*:

- **53% declarações de atribuição**
- **32% declarações de controle**
- **15% chamadas de função/procedimentos**

Falha: "O uso da média aritmética dos tempos de execução normalizada para prever a performance".

Um método que poderia ser usado para calcular a performance da máquina é normalizar seus tempos de execução tendo como referência outra máquina. O uso da média aritmética dos tempos de execução vai depender da máquina tomada como referência.

	Tempo em A	Tempo em B	Normalizado para A		Normalizado para B	
			A	B	A	B
Programa 1	1	10	1	10	0,1	1
Programa 2	1000	100	1	0,1	10	1
Média aritmética do tempo ou tempo normalizado	550,5	55	1	**5,05**	**5,05**	1
Média aritmética do tempo ou tempo normalizado	31,6	31,6	1	1	1	1

Ao normalizarmos para A, a média aritmética indica que A é mais rápida que B na razão de 5,05/1. Ao normalizarmos para B, a média aritmética indica que B é mais rápida que a na razão de 5,05/1.
Então não podemos concluir qual a máquina mais rápida, mas verificamos que este procedimento está errado! **O erro advém do uso da média aritmética das razões.**

Os resultados deveriam usar uma **média geométrica**:

$$\sqrt[n]{\prod_{i=1}^{n} Razão\,do\,tempo\,de\,execucao_i}$$

onde: Razão do tempo de execução$_i$ é o tempo de execução normalizado para uma máquina de referência, para o i-ésimo programa do total de n programas.

A média geométrica independe da série de dados utilizada para a normalização, ou seja, a média das razões ou a razão das médias produz o mesmo resultado.
A **desvantagem** do uso da média geométrica é que ela não prevê a relação dos tempos de processamento dos programas individuais.
Solução: medir o workload real e atribuir pesos aos programas de acordo com as respectivas freqüências de execução.

Lei de Amdahl:

Esta lei evidencia como acelerar ou aumentar o ganho de performance de uma máquina após a introdução de uma melhoria.

Ela relaciona o comportamento antes e depois da implementação da melhoria.

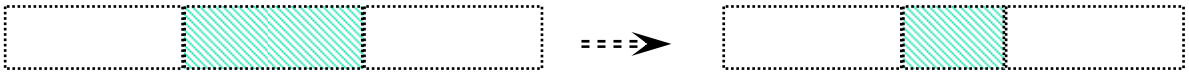
Uma máquina que após a melhoria obteve uma aceleração de 2 em relação ao seu comportamento sem melhoria, possui uma performance duas vezes superior do que seu comportamento anterior.

$$\textit{Speed-up} = \textit{Aceleração} = \frac{\textit{Performance após melhoria}}{\textit{Performance antes melhoria}}$$

$$\textit{Aceleração} = \frac{\textit{Tempo antes melhoria}}{\textit{Tempo após melhoria}}$$

$$\textit{Tempo de execução após a melhora} = \textit{Tempo de execução afetado pela melhoria} \div \textit{Qtde. de melhoria} + \textit{Tempo de execução não afetado}$$

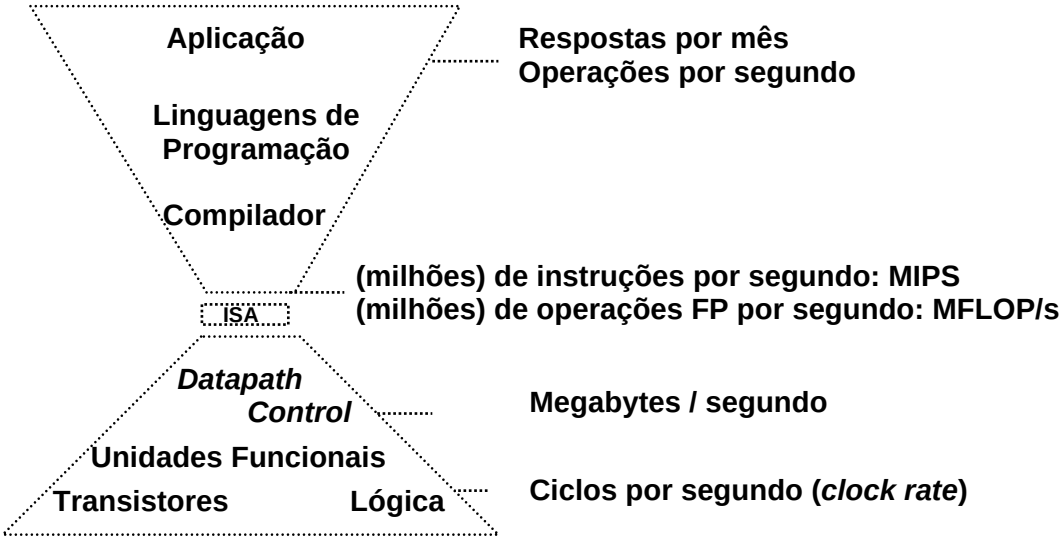
Corolário: *Torne mais rápidos os casos mais comuns*



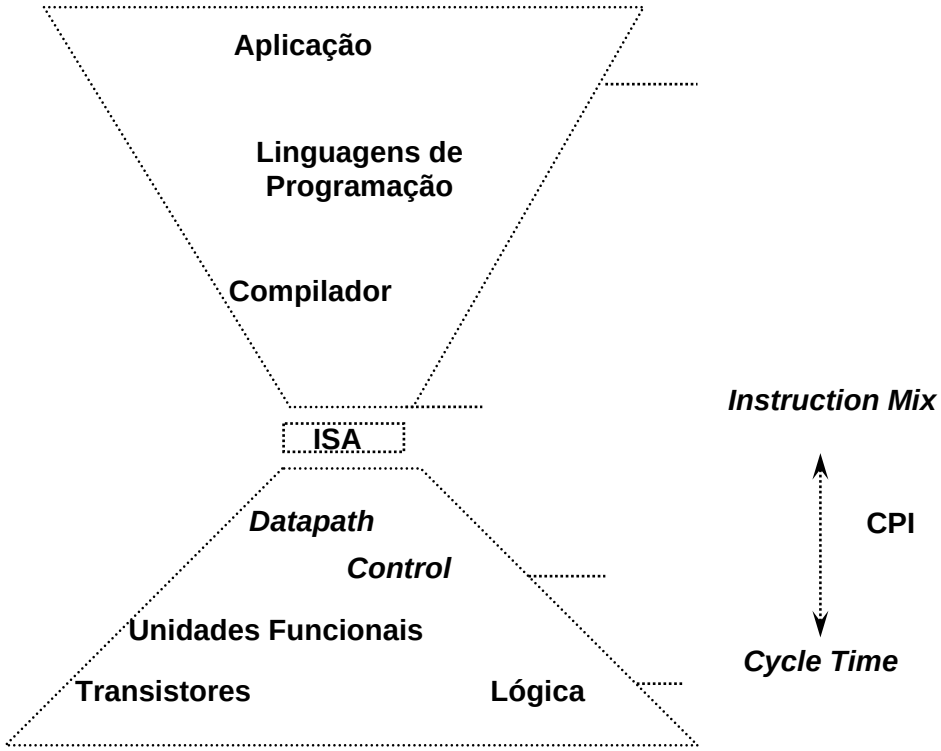
$$\mathbf{ExTime_{new} = ExTime_{old} \times \left[(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]}$$

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

Resumo: Métricas de performance



Compromissos da Organização da CPU



Lista de Exercícios: 2.1, 2.2, 2.3, 2.10, 2.13, 2.18, 2.26,2.31, 2.44 e 2.46. (tempo estimado pelo livro 1:15h).