# Desempenho de CPUs

## Noções básicas

João Canas Ferreira

Novembro de 2017

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

# Tópicos

1. Conceitos básicos
2. Equação fundamental do desempenho
3. Lei de Amdahl
4. Desempenho com memórias cache

Contém figuras de "Computer Organization and Design", D. Patterson & J. Hennessey, 3ª. ed., MKP

1 Conceitos básicos

2 Equação fundamental do desempenho

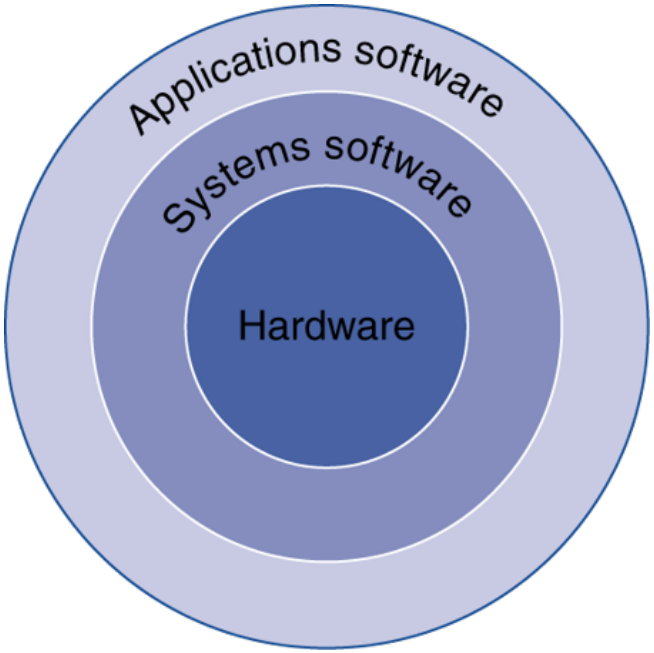
3 Lei de Amdahl

4 Desempenho com memórias cache

# O que influencia o desempenho de um CPU?

- Algoritmo usado
  Determina o número de operações executadas
- Linguagem de programação, compilador, arquitetura
  Determina o número de instruções executadas por operação
- Processador e sistema de memória
  Determina a rapidez de execução das instruções
- Sistema de E/S (incluindo o sistema operativo)
  Determina a rapidez das operações de E/S

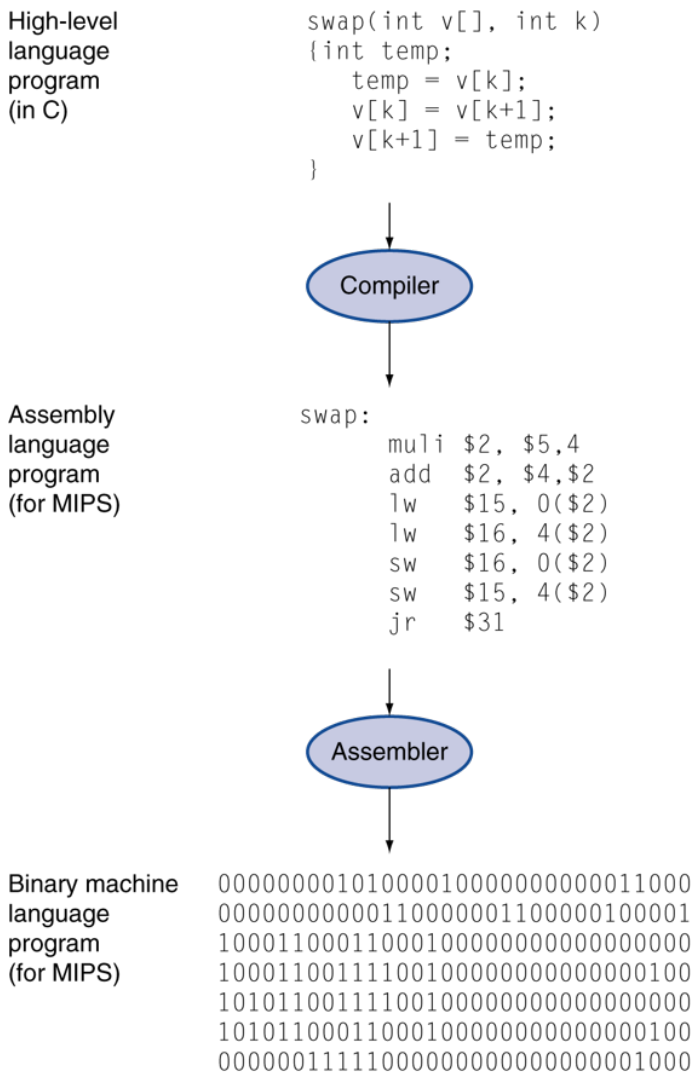
# Da aplicação ao suporte físico



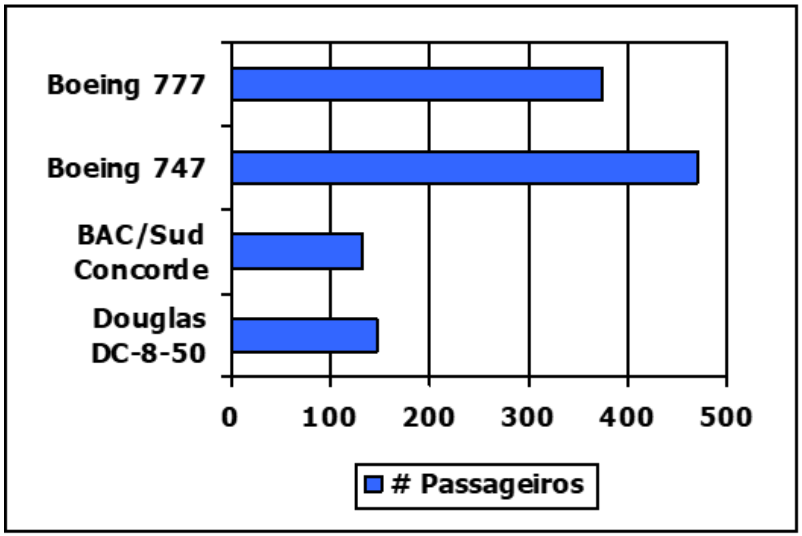

- Aplicação
  - Escrita numa linguagem de alto nível
- Software de sistema
  - Compilador
    – Traduz código de linguagem de alto nível para código-máquina
  - Sistema operativo: serviços
    – Gestão de entradas/saídas de dados
    – Gestão de memória e armazenamento
    – Escalonamento de tarefas e partilha de recursos
- Suporte físico *(hardware)*
  - Processador, memória, controladores de E/S

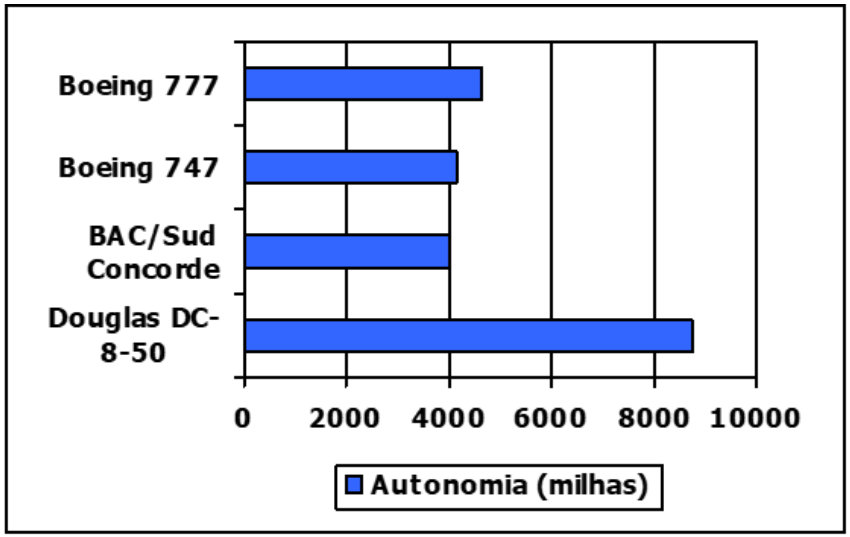
# Níveis de código

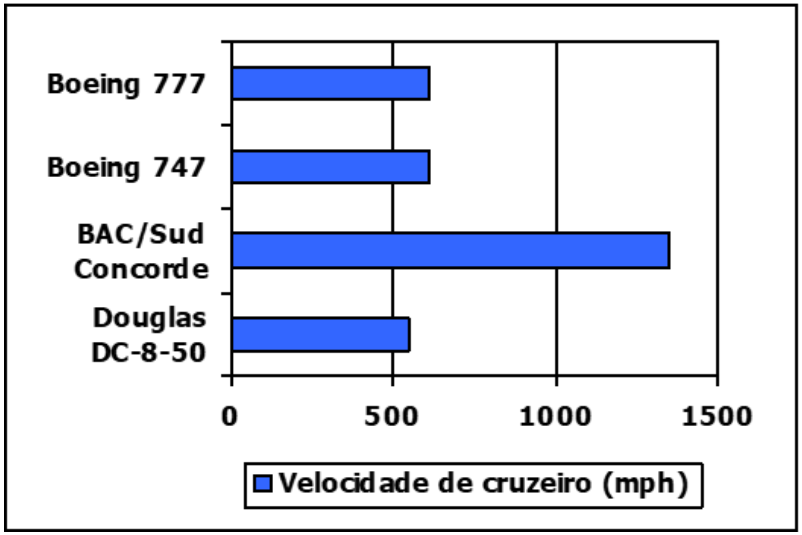
- Linguagem de alto nível
  – Nível de abstração mais perto do domínio do problema
  – Maior produtividade e portabilidade
- Linguagem *assembly*
  – Representação textual das instruções do CPU
- Código-máquina
  – Dígitos binários (bits)
  – Codificação de dados e instruções

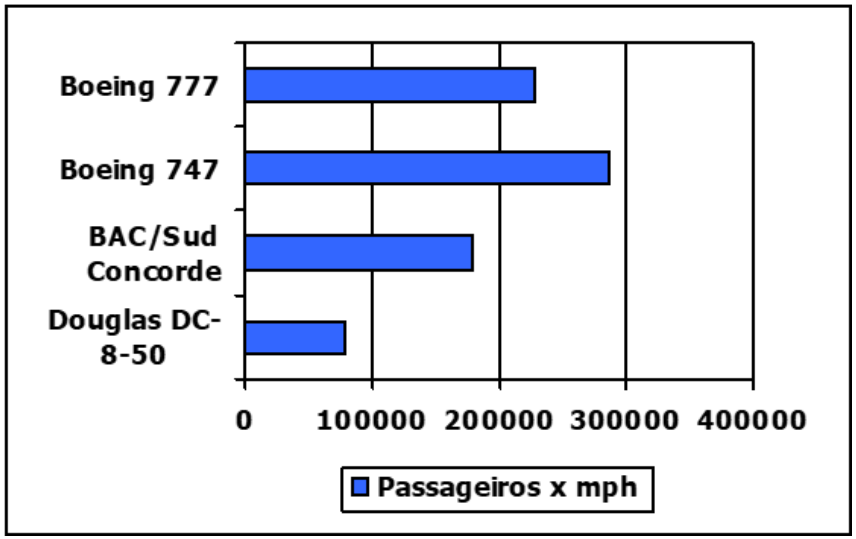


# Critérios de desempenho




Fonte: [COD4]

☞ Qual é o melhor avião?

# Tempo de resposta e débito

- *Tempo de resposta* ou *latência*
  Def: O tempo que o CPU demora a executar uma tarefa
  – Tempo necessário para começar a serem produzidos resultados contado a partir do início da tarefa

- *Débito*
  Def: Trabalho efetuado por unidade de tempo
  – Exemplos: número de tarefas por hora; número de instruções executadas por segundo.

- Questão: Como é que débito e tempo de resposta são afetados ao:
  – substituir-se um processador por outro mais rápido?
  – acrescentar-se mais processadores (cada um pode executar uma tarefa)?

- Foco desta introdução: **tempo de resposta** como medida de desempenho

1 Conceitos básicos

2 Equação fundamental do desempenho

3 Lei de Amdahl

4 Desempenho com memórias cache

# Desempenho e desempenho relativo

- Definição: **Desempenho = 1 / (Tempo de execução)** (unidades: $s^{-1}$)

- **Desempenho relativo**

**O sistema X é *n* vezes mais rápido que o sistema Y**

$$n = \frac{\text{Tempo de execução}_Y}{\text{Tempo de execuçao}_X} = \frac{\text{Desempenho}_X}{\text{Desempenho}_Y}$$

- Exemplo:
  Tempo de execução de um dado programa: 10 s no computador A, 15 s em B

$$n = \frac{\text{Tempo de execução}_B}{\text{Tempo de execuçao}_A} = \frac{15\text{ s}}{10\text{ s}} = 1{,}5$$

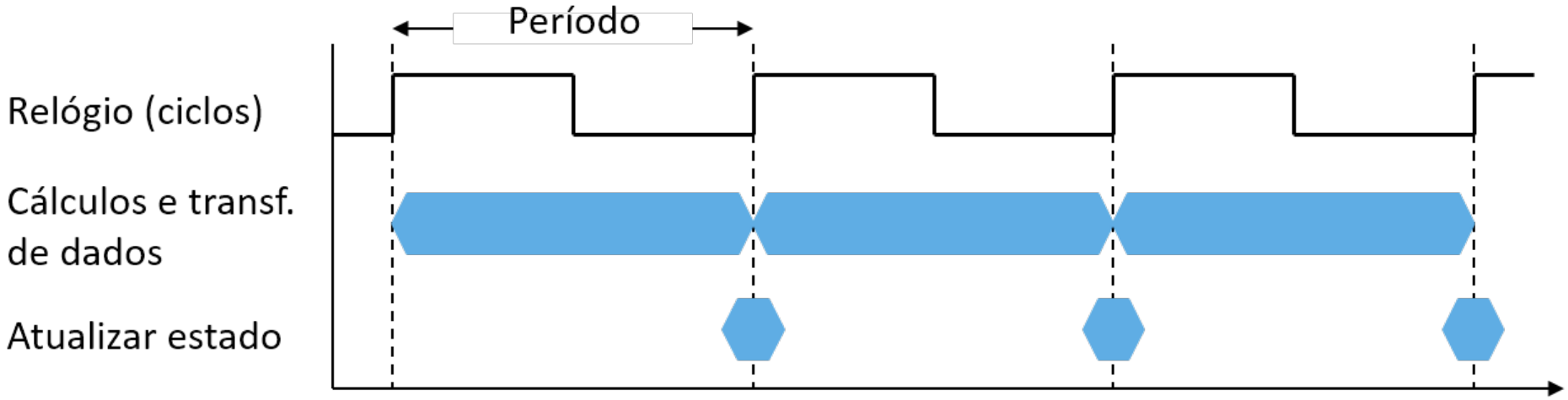
- O computador é 1,5 vezes mais rápido que B (para este programa!)

# Tempo de execução

- Tempo transcorrido *(wall-clock time)*: tempo de resposta **total**
  – Inclui processamento, E/S, *overhead* do sistema operativo, inatividade, tempo gasto pelo sistema com outras tarefas...
  – Determina o desempenho do sistema (mas depende de muitos fatores e não apenas do CPU)
- Tempo de CPU *(CPU time)*:
  Tempo gasto pelo CPU no processamento de uma dada tarefa
  – Não inclui tempo E/S, nem tempo de outras tarefas
  – Tem duas componentes:
    - tempo de utilizador *(user time)*: tempo gasto pelo CPU a executar código do "utilizador"
    - tempo de sistema *(system time)*; tempo gasto pelo sistema operativo a executar trabalho para a tarefa (aceder a ficheiros, etc.)
  – Cada programa pode ser afetado de maneira diferente por cada componente
- O tempo despendido numa tarefa depende do CPU, do código da tarefa (algoritmo, compilador) e dos **dados de entrada**

# CPU como circuito sequencial síncrono

➠ Operação de circuitos síncronos é governada por um sinal de relógio (de período constante).




- Período: duração de um ciclo de relógio
  p. ex., 250 ps = 0,25 ns = $250 \times 10^{-12}$ s
- Frequência de relógio: ciclos por segundo
  p. ex., 4,0 GHz = 4000 MHz = $4,0 \times 10^{9}$ Hz

# Tempo de CPU

Para um dado programa com um conjunto específico de dados de entrada:

$$\text{Tempo de CPU} = \text{Número de ciclos de execução} \times \text{Período de um ciclo}$$

$$= \frac{\text{Número de ciclos de execução}}{\text{Frequência do relógio}}$$

- Desempenho numa tarefa pode ser melhorado:
  1. Reduzindo o número de ciclos de relógio necessários
     - Como? Aumentando a quantidade de trabalho efetuado em cada ciclo.
  2. Aumentando a frequência do relógio
- Em geral, os dois objetivos são antagónicos!
  - Projetista do CPU deve obter um compromisso

# Tempo de CPU: Exemplo

- Computador A: relógio de 2 GHz, tarefa demora 10 s
- Com um novo computador B: tarefa deve demorar 6 s
  Pode-se usar relógio mais rápido, mas tarefa passa a requerer 1,2× mais ciclos.

Qual deve ser a frequência de relógio do computador B?

$$\text{Frequência}_B = \frac{\text{Ciclos de execução}_B}{\text{Tempo de CPU}_B} = \frac{1{,}2 \times \text{Ciclos de execução}_A}{6\,\text{s}}$$

$$\text{Ciclos de execução}_A = \text{Tempo de CPU}_A \times \text{Frequência}_A$$

$$= 10\,\text{s} \times 2\,\text{GHz} = 20 \times 10^9$$

$$\text{Frequência}_B = \frac{1{,}2 \times 20 \times 10^{19}}{6\,\text{s}} = \frac{24 \times 10^9}{6\,\text{s}} = 4\,\text{GHz}$$

# Número de instruções e CPI

- CPI: número **médio** de ciclos por instrução
- A média deve ser válida para o programa cujo desempenho está a ser avaliado.

$$
\begin{aligned}
\#\text{ de ciclos excutados} &= \#\text{ instruções executadas} \times \text{CPI} \\
\text{Tempo de CPU} &= \#\text{ instruções executadas} \times \text{CPI} \times \text{período de relógio} \\
&= \frac{\#\text{ instruções executadas} \times \text{CPI}}{\text{Frequência}} \\
T_{\text{exec}} &= N \times \text{CPI} \times T = \frac{N \times \text{CPI}}{F} \quad \text{(equação fundamental)}
\end{aligned}
$$

- # instruções executadas de um programa:
  – Determinado pelo programa, conjunto de instruções e compilador
- Número de ciclos por instrução (CPI)
  – Para uma instrução: Determinado pela implementação
  – Para programa: o CPI médio depende da distribuição de instruções

# Influência de CPI (exemplo)

- Computador A: Período = 250 ps, CPI = 2,0
  Computador B: Período = 500 ps, CPI = 1,2
  Mesmo conjunto de instruções e mesmo programa.

- Qual computador é mais rápido e por quanto?

$$
\begin{aligned}
\text{Tempo de CPU}_A &= \#\,\text{instruções} \times \text{CPI} \times \text{Período de relógio}_A \\
&= N \times 2{,}0 \times 250\,\text{ps} = 500\,\text{ps} \\
\text{Tempo de CPU}_B &= \#\,\text{instruções} \times \text{CPI} \times \text{Período de relógio}_B \\
&= N \times 1{,}2 \times 500\,\text{ps} = N \times 600\,\text{ps}
\end{aligned}
$$

$$
\frac{\text{Tempo de CPU}_B}{\text{Tempo de CPU}_A} = \frac{N \times 600\,\text{ps}}{N \times 500\,\text{ps}} = 1{,}2
$$

- O computador A é mais **1,2** vezes mais rápido que o computador B.

# Diferentes classes de instruções

- Quando diferentes classes de instruções requerem diferente número de ciclos:

$$\#\,\text{ciclos} = \sum_{i=1}^{n} (\text{CPI}_i \times \#\,\text{instruções de classe } i)$$

- **CPI médio** (média pesada pela taxa de ocorrência de cada classe):

$$\text{CPI} = \frac{\#\,\text{ciclos}}{\#\,\text{instruções}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \frac{\#\,\text{instruções de classe } i}{\#\,\text{instruções}} \right)$$

- Exemplo: Duas pequenas sequências de código alternativas usando instruções das classes A, B e C.

| Classe | A | B | C |
|---|---|---|---|
| CPI | 1 | 2 | 3 |
| Sequência 1 (N=5) | 2 | 1 | 2 |
| Sequência 2 (N=6) | 4 | 1 | 1 |

- Sequência 1:
  # ciclos = 10 $\rightarrow$ CPI médio = 10/5 = 2
- Sequência 2:
  # ciclos = 9 $\rightarrow$ CPI médio = 9/6 = 1,5

# Desempenho: Equação fundamental

- Equação fundamental de desempenho:

$$\text{Tempo de CPU} = \#\,\text{instruções} \times \frac{\#\,\text{ciclos}}{\#\,\text{instruções}} \times \frac{\#\,\text{segundos}}{\text{ciclo}}$$

$$T_{\text{exec}} = N \times \text{CPI} \times T$$

- Desempenho de CPU depende de:
  - Algoritmo: afeta # instruções, possivelmente também o CPI
  - Linguagem de programação: # instruções, CPI
  - Compilador: afeta # instruções, CPI
  - Arquitetura do conjunto de instruções: afeta # instruções, período, CPI

# Avaliação global de desempenhos por "benchmarks"

A equação fundamental determina o desempenho de um CPU executando **um** programa específico para dados de entrada definidos.

- Benchmarks: Programas usados para medir o desempenho
  - Programas devem ser típicos do domínio de utilização
- Organismo Standard Performance Evaluation Corp. (SPEC)
  - Desenvolve "benchmarks" para CPU, E/S, Web,...
- "Benchmarks" SPEC CPU2006, CPU2017
  - Tempo transcorrido na execução do programa
  - Débito para processadores multi-core (versão CPU2017)
  - E/S negligenciável: foco no desempenho de CPU
  - **Valores normalizados relativo a um computador de referência**
  - Programas com e sem vírgula flutuante
  - Sumário: **média geométrica das razões de desempenho**

$$\sqrt[n]{\prod_{i=1}^{n}(\text{razão dos tempos de execução de benchmark } i)}$$

# CINT2006 para Intel Core i7 920

| Description | Name | Instruction Count x $10^9$ | CPI | Clock cycle time (seconds x $10^{-9}$) | Execution Time (seconds) | Reference Time (seconds) | SPECratio |
|---|---|---|---|---|---|---|---|
| Interpreted string processing | perl | 2252 | 0.60 | 0.376 | 508 | 9770 | 19.2 |
| Block-sorting compression | bzip2 | 2390 | 0.70 | 0.376 | 629 | 9650 | 15.4 |
| GNU C compiler | gcc | 794 | 1.20 | 0.376 | 358 | 8050 | 22.5 |
| Combinatorial optimization | mcf | 221 | 2.66 | 0.376 | 221 | 9120 | 41.2 |
| Go game (AI) | go | 1274 | 1.10 | 0.376 | 527 | 10490 | 19.9 |
| Search gene sequence | hmmer | 2616 | 0.60 | 0.376 | 590 | 9330 | 15.8 |
| Chess game (AI) | sjeng | 1948 | 0.80 | 0.376 | 586 | 12100 | 20.7 |
| Quantum computer simulation | libquantum | 659 | 0.44 | 0.376 | 109 | 20720 | 190.0 |
| Video compression | h264avc | 3793 | 0.50 | 0.376 | 713 | 22130 | 31.0 |
| Discrete event simulation library | omnetpp | 367 | 2.10 | 0.376 | 290 | 6250 | 21.5 |
| Games/path finding | astar | 1250 | 1.00 | 0.376 | 470 | 7020 | 14.9 |
| XML parsing | xalancbmk | 1045 | 0.70 | 0.376 | 275 | 6900 | 25.1 |
| Geometric mean | – | – | – | – | – | – | 25.7 |

Fonte: [COD4]

➠ `https://www.spec.org/benchmarks.html`

1 Conceitos básicos

2 Equação fundamental do desempenho

3 Lei de Amdahl

4 Desempenho com memórias cache

# "Lei" de Amdahl

➠ Melhorar o desempenho de um aspeto de um computador **não leva** a um melhoramento proporcional do desempenho global

Porquê? O melhoramento não pode ser utilizado durante todo o tempo de execução original do programa.

➠ O tempo de execução do programa com o melhoramento é:

$$T_{\text{melhorado}} = \frac{T_{\text{afetado}}}{\text{fator de melhoramento}} + T_{\text{não afetado}}$$

➠ Exemplo: programa gasta 80 s de 100 s a fazer multiplicações

De quanto é preciso melhorar o desempenho do multiplicador para obter uma desempenho total 5× maior?

$$20 = \frac{80}{f} + 20 \qquad \text{Impossível para } f > 0$$

# Lei de Amdahl: aumento de rapidez

➠ Aumento de rapidez *(speedup)*: fator de melhoramento do desempenho

$$S = \text{Speedup} = \frac{\text{Desempenho com melhoramento}}{\text{Desempenho original}}$$

➠ **Cenário de utilização parcial:** Melhoramento com *speedup* $S_{\text{parcial}}$ é usado durante uma percentagem $p$ do tempo de execução ($0 \leq p \leq 1$).

Qual é o aumento de rapidez **global**?

$$S_{\text{global}} = \frac{\text{Tempo de execução original}}{\text{Tempo de execução melhorado}} = \frac{1}{\frac{p}{S_{\text{parcial}}} + (1 - p)}$$

(Demonstrar!)

# Aumento de rapidez: exemplo

➠ Um programa gasta 80 s de 100 s em multiplicações.

☞ Um novo multiplicador é 2× mais rápido que o original. Qual é o fator de melhoramento *(speedup)*?

$$S_{\text{global}} = \frac{1}{\frac{0{,}8}{2} + (1 - 0{,}8)} = \frac{1}{0{,}6} = \frac{10}{6} \approx 1{,}67$$

☞ De quanto é preciso melhorar o desempenho do multiplicador para obter um desempenho total 2× maior?

$$2 = \frac{1}{\frac{0{,}8}{S_{\text{parcial}}} + (1 - 0{,}8)} \Rightarrow S_{\text{parcial}} = \frac{16}{6} = 2\frac{2}{3} \approx 2{,}67$$

# Erro metodológico: Usar medidas parciais de desempenho

➠ MIPS: *millions of instructions per second*

➠ Não tem em conta:

- Diferenças entre conjuntos de instruções
- Diferenças de complexidade entre instruções ($\rightarrow$ CPI)

$$\text{MIPS} = \frac{\#\text{ instruções}}{\text{Tempo de execução} \times 10^6}$$

$$= \frac{\#\text{ instruções}}{\frac{\#\text{ instruções} \times \text{CPI}}{\text{Frequência de relógio}}} \times 10^6 = \frac{\text{Frequência de relógio}}{\text{CPI} \times 10^6}$$

☞ Mas para o mesmo CPU, o número de instruções executadas varia entre programas (ou para o mesmo programa com dados diferentes)!

1 Conceitos básicos

2 Equação fundamental do desempenho

3 Lei de Amdahl

4 Desempenho com memórias cache

# Equações básicas de desempenho de caches

- O tempo (em ciclos) para acerto *(hit)* faz parte do tempo "normal" de execução de uma instrução.
- $T_{\text{CPU}} = (C_{\text{exec}} + C_{\text{prot}}) \times T$
  - $T_{\text{CPU}}$: tempo de CPU
  - $C_{\text{exec}}$: nº de ciclos em que CPU executou instruções
  - $C_{\text{prot}}$: nº de ciclos em que o CPU protelou devido a faltas no acesso a memória cache
- Assumir penalidades de falta iguais na escrita e na leitura.
- $C_{\text{prot}} = N_{rw} \times t_f \times p_f$
  - $N_{rw}$: número de acessos do programa (leitura e escrita)
  - $t_f$: taxa de faltas
  - $p_f$: penalidade de faltas (em ciclos de relógio do CPU)
- Se taxa de faltas ou a penalidade forem diferentes para leituras e escritas, então é preciso separar a equação anterior em duas parcelas.

# Exemplos de cálculo de desempenho

- Especificação: $\text{CPI}_{\text{ideal}}$=2, $t_{\text{fi}} = 0{,}02$ para instruções e $t_{\text{fd}} = 0{,}04$ para dados, $p_f = 100$, acessos a memória (dados) para 36 % das instruções.
- Comparar o desempenho "real" com o desempenho obtido se a memória *cache* fosse perfeita.
- $C_i$: nº de ciclos de protelamento na obtenção de instruções:
  $C_i = N \times 0{,}02 \times 100 = 2 \times N$
- $C_d$: nº de ciclos de protelamento no acesso a dados:
  $C_d = N \times 0{,}36 \times 0{,}04 \times 100 = 1{,}44 \times N$
- $\text{CPI}_{\text{real}} = \text{CPI}_{\text{ideal}} + C_i/\text{N} + C_d/\text{N} = 2 + 1{,}44 + 2 = 5{,}44$
- Comparando os desempenhos:

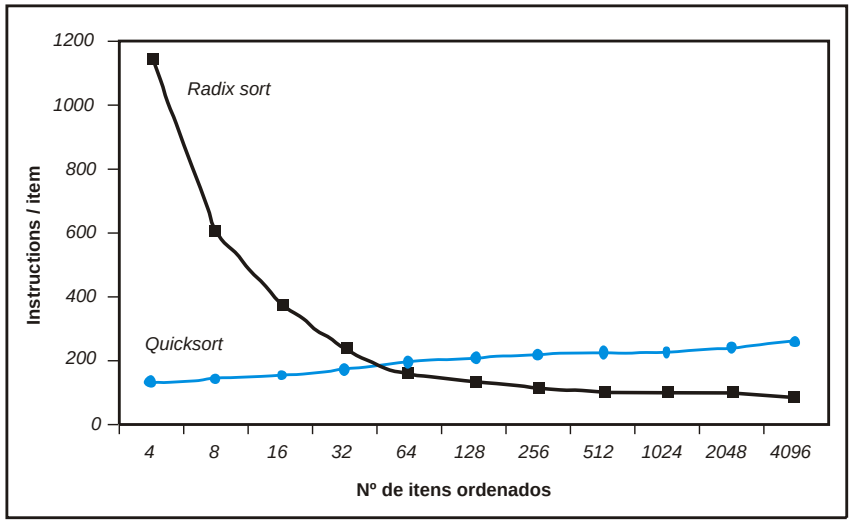
$$r = \frac{N \times \text{CPI}_{\text{real}} \times T}{N \times \text{CPI}_{\text{ideal}} \times T} = \frac{5{,}44}{2} = 2{,}72$$

- E se $\text{CPI}_{\text{ideal}} = 1$ ?
- $\text{CPI}_{\text{real}} = 4{,}44 \rightarrow r = 4{,}44$
  Percentagem de ciclos de protelamento piora:
  $3{,}44/5{,}44 = 63\% \rightarrow 3{,}44/4{,}44 = 77\%$.

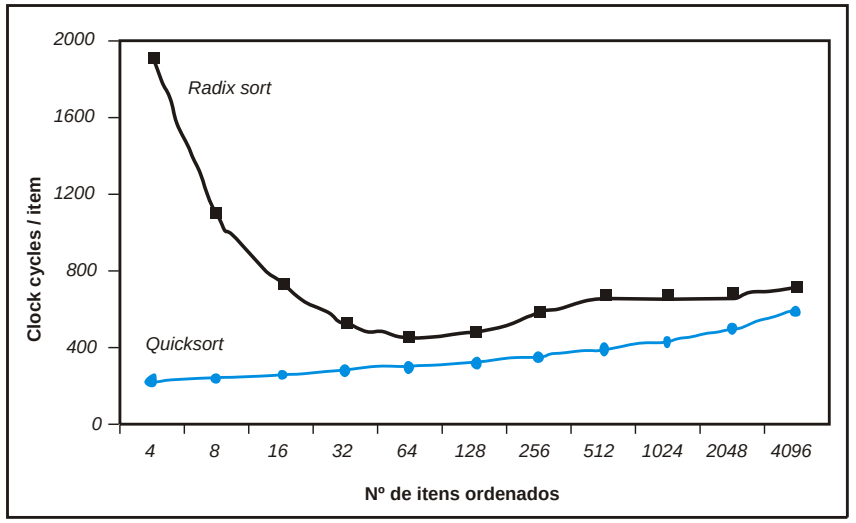
# Efeito do aumento de frequência

- Melhorar o CPI sem melhorar o subsistema de memória leva a um aumento da percentagem de tempo perdida em protelamentos (cf. exemplo anterior).
- O que acontece se a frequência de relógio aumentar, mas o subsistema de memória permanecer igual?
- Neste caso a penalidade de falta aumenta: $p_f = 200$ (p. ex.)
- $C_i + C_d = 0{,}02 \times 200 + 0{,}36 \times 0{,}04 \times 200 = 6{,}88$
- $\text{CPI}_{\text{rápido}} = 2 + 6{,}88 = 8{,}88$
- $$\text{speedup} = \frac{N \times \text{CPI}_{\text{lento}} \times T}{N \times \text{CPI}_{\text{rápido}} \times \frac{T}{2}} = \frac{5{,}44}{8{,}88 \times 0{,}5} = 1{,}23$$
- Se a influência da memória *cache* tivesse sido ignorada, seria speedup=2.
- Processador que melhore simultaneamente CPI e T sofre um impacto (relativo) negativo duplo.

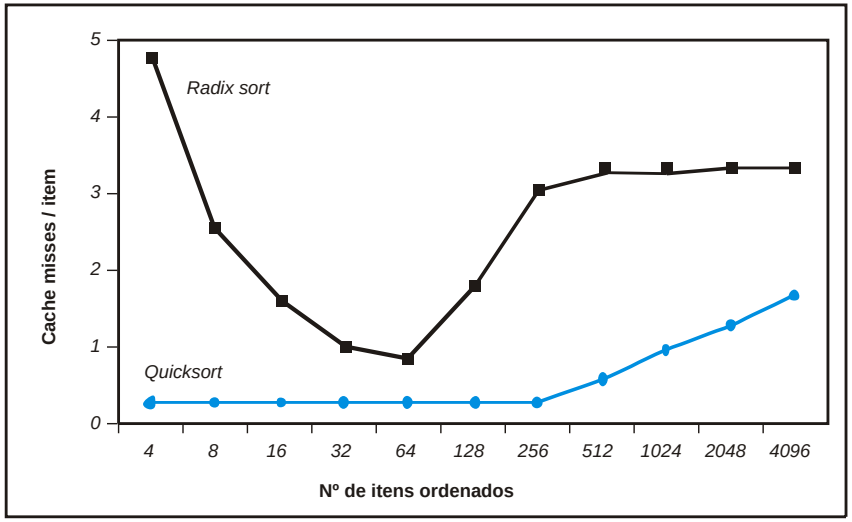
# Influência do desempenho de cache




Nº de instruções executadas por item




Ciclos de relógio por item



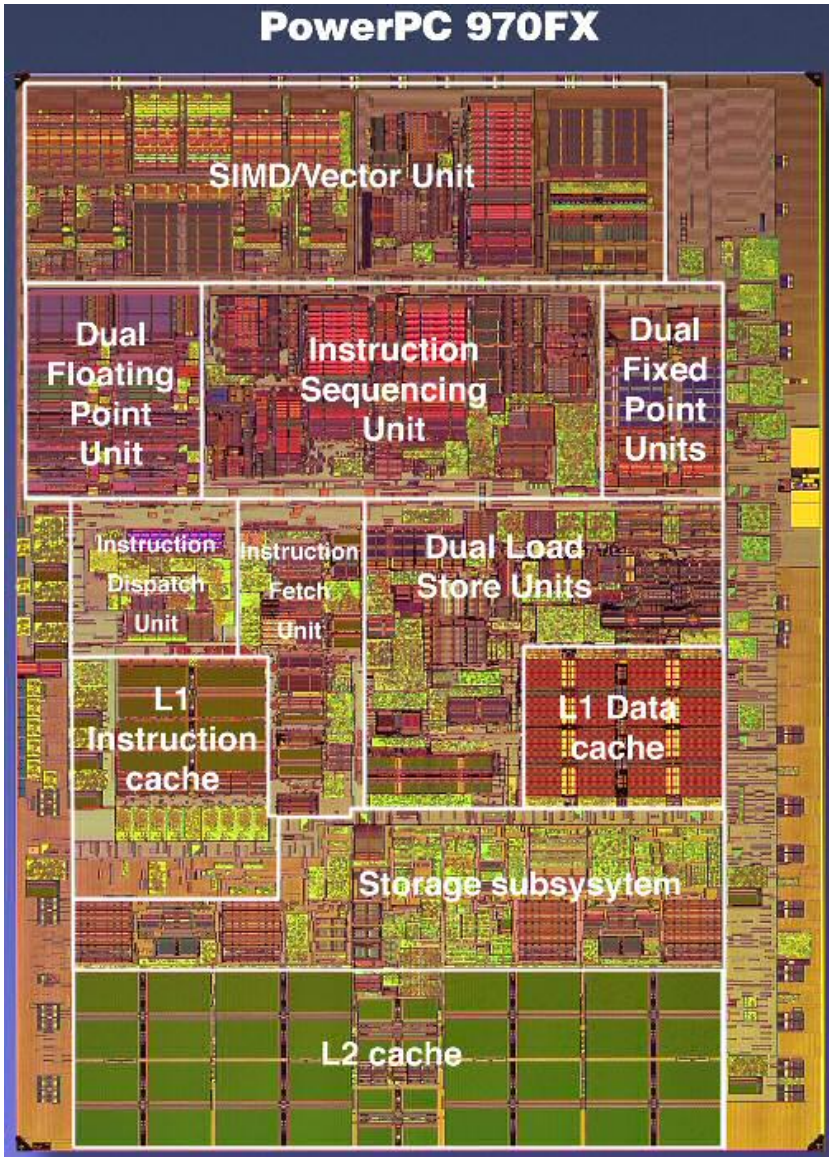

Nº de falhas de cache por item

Fonte: [COD3]

# Múltiplos níveis de memória cache

- Para reduzir a penalidade de faltas de uma memória *cache* pode aplicar-se *a mesma abordagem que se usa para reduzir o tempo de acesso à memória principal*: usar uma memória *cache* (de 2º nível) para a *cache*
- A cache de nível 2 permite reduzir a penalidade de faltas da cache de nível 1.
- A cache de nível 2 contém geralmente os dados residente na cache de nível 1, é de maior capacidade e mais lenta.
- Arranjo comum: memórias cache separadas de nível 1 + cache unificada de nível 2.
- Em sistemas de elevado desempenho: 3 níveis de cache *on-chip*.

# Processador com dois níveis de memória cache interna




- Usado no Mac G5.
- 58 milhões transístores, F=2,2 GHz.
- L1-ICache: 64 KB;128 bytes/bloco.
- L1-DCache: 32 KB; 128 bytes/bloco;write-through,no-allocate.
- L2-Cache: 512 KB; 128 bytes/bloco;write-back, allocate-on-miss.

# Desempenho de memórias cache multinível

Cálculo do desempenho de um sistema com dois níveis de cache.

➠ Especificação: $CPI_{ideal} = 1$, F=5 GHz (T=0,2 ns); acesso a memória principal: 100 ns, taxa de faltas em L1 de 2%.

Qual a melhoria de desempenho obtida por uma memória cache L2 (tempo de acesso 5 ns) e com capacidade suficiente para reduzir a taxa de faltas para memória principal para 0,5%?

- Com um nível: penalidade de falta (mem. principal): 100/0,2 = 500 ciclos.
- $CPI_{real\_1} = 1 + 0{,}02 \times 500 = 11$
- Com 2 níveis:
  Penalidade de falta por acesso a L2: 5/0,2 = 25 ciclos
- $CPI_{real\_2} = 1 + Prot_{L1} + Prot_{L2}$
  $CPI_{real\_2} = 1 + 0{,}02 \times 25 + 0{,}005 \times 500 = 1 + 0{,}5 + 2{,}5 = 4$
- speedup = 11/4 = 2,8
- (Extra) taxa de faltas *local* de L2: 0,005/0,02 = 0,25

# Referências


COD4 D. A. Patterson & J. L. Hennessey, Computer Organization and Design, 4 ed.

COD3 D. A. Patterson & J. L. Hennessey, Computer Organization and Design, 3 ed.


Os tópicos tratados nesta apresentação são descritos nas seguintes secções de [COD4]:

- 1.4, 1.7–1.8, 5.3

Também são tratados nas seguintes secções de [COD3]:

- 4.1–4.5, 7.3