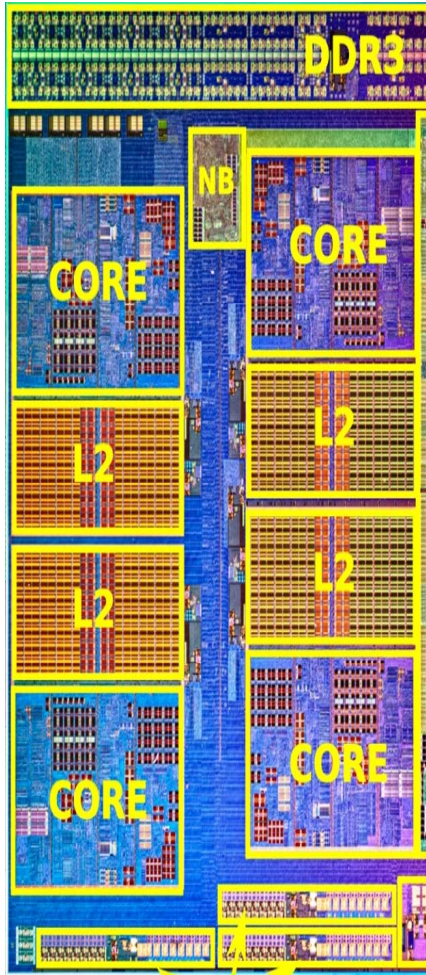


# 0273359 - Arquitetura e Organização de Computadores 1



Fonte: <http://www.techspot.com/article/904-history-of-the-personal-computer-part-5/>

## Desempenho

Luciano de Oliveira Neris

[luciano@dc.ufscar.br](mailto:luciano@dc.ufscar.br)

Adaptado de slides do prof. Marcio Merino Fernandes  
Figuras: David Patterson, John Hennessy  
Arquitetura e Organização de Computadores – 4Ed, Elsevier, 2014

Departamento de Computação  
Universidade Federal de São Carlos



# Desempenho

- Vamos discutir como medir, relatar, e sumarizar o desempenho de um sistema de computador.
- Os sistemas de computador são tão intrincados, que é impossível prever analiticamente como o desempenho deve ser na prática.
  - Adicionalmente, depende pesadamente do programa
- Além disso, os vendedores de computadores fazem a “tentativa do computador parecer bom” uma forma de arte.
- O que nós podemos fazer é avaliar o impacto no desempenho causado por uma mudança na arquitetura /organização/configuração, para um dado programa
  - Muito menos difícil do que tomando dois computadores completamente diferentes e tentando dizer “A é melhor do que B”, o que é quase sempre impossível
- Até para fazer isto, nós temos que fazer muitas aproximações, suposições e simplificações.

# Produção e Tempo de Resposta

Qual aeronave possui o melhor desempenho ?

Air plane	Pa ss enge r capacity	Cr uisi ng ra nge (miles)	Cr uisi ng spe ed (m.p.h.)
Boeing 777	375	4630	610
Boeing 747	470	4150	610
BAC/Sud Co ncorde	132	4000	1350
Douglas DC-8-50	146	8720	544

FIGURE 1.13 The capacity, range, and speed for a number of commercial airplanes. Copyright © 2009 Elsevier, Inc. All rights reserved.

# Produção e Tempo de Resposta

Qual aeronave possui o melhor desempenho ?

Air plane	Pa ss enge r capacity	Cr uisi ng ra nge (miles)	Cr uisi ng spe ed (m.p.h.)	Pa ss enge r thr ough put (passen ger s × m.p. h.)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Co ncorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

**FIGURE 1.13** The capacity, range, and speed for a number of commercial airplanes.

The last column shows the rate at which the airplane transports passengers, which is the capacity times the cruising speed (ignoring range and takeoff and landing times).

Copyright © 2009 Elsevier, Inc. All rights reserved.

# Definindo Desempenho

- Há muitas maneiras de definir o desempenho de um computador
- Dois dos parâmetros mais comuns são:
  - **TEMPO DE RESPOSTA (RESPONSE TIME):** *Quão rapidamente o computador executa um programa.*
  - **PRODUÇÃO (THROUGHPUT):** *Quanto "trabalho" o computador pode fazer por unidade de tempo, supondo ter uma quantidade de trabalho infinita a fazer.*  
$$\text{Produção} = N / \text{tempo de resposta}$$

onde

$$N = \text{nro de tarefas executadas}$$
- Tempo de Resposta e Produção estão relacionados

# Produção e Tempo de Resposta

---

- Tipicamente, diminuir o tempo de resposta resulta em aumento da produção:
  - Se eu aumento minha taxa clock do processador, um único programa funciona mais rapidamente, e assim eu posso executar mais programas por a unidade de tempo
- O efeito da **concorrência** pode acelerar a produção:
  - Em computadores modernos algumas operações ocorrem simultaneamente
  - Consequentemente, executar dois programas ao mesmo tempo pode tomar menos tempo do que executar dois programas um após o outro!

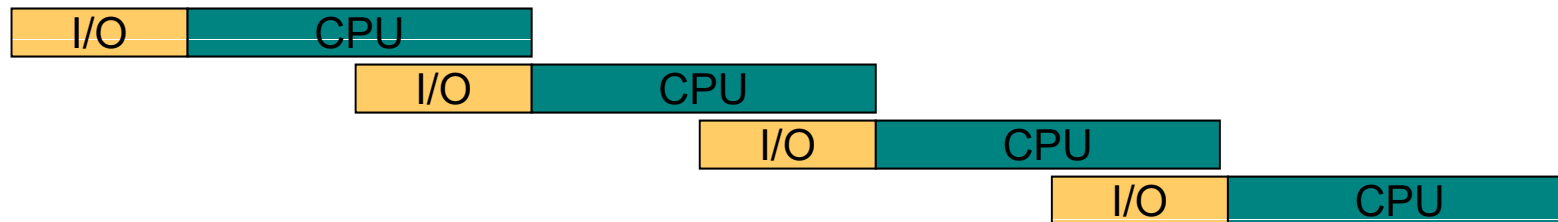
# Produção e Tempo de Resposta

- Exemplo:

- Considere um programa A que gaste 5 minutos c/ acesso a disco, e 10 minutos em cálculos aritméticos.



- O tempo de resposta é 15 minutos. Qual é a produção?
- Em computadores modernos, quando um processo usar o processador central um outro processo pode usar o disco quase independentemente;
- Consequentemente, a execução seria da forma:



# Produção e Tempo de Resposta

- Produção para 4 execuções =  $4 / (15 + 10 + 10 + 10) = 4 / 45 > 1 / 15$
- Produção para n execuções =  $n / (15 + (n-1)*10)$
- Quando n aumenta, a produção tende a  $1/10$ , que é  $> 1/15$
- Consequentemente, devido à concorrência, a produção é  $>$  que  $1 / (\text{tempo de resposta})$



# Tempo de Resposta

- Vamos usar o tempo de resposta para nossa análise de desempenho:
  - Tempo de resposta alto é ruim
- Neste caso, pode-se definir o desempenho como o inverso do tempo de resposta:

$$\text{Performance (X)} = 1 / \text{Tempo Execução (X)}$$

(onde X é um sistema de computação)

- Quando se diz que o sistema X é n vezes mais rápido que o sistema Y significa:

$$\text{Performance(X)} / \text{Performance(Y)} = n$$

ou

$$\text{Tempo Execução(Y)} / \text{Tempo Execução(X)} = n$$

# Tempo de Execução

- Como se define o tempo de execução?
  - parece fácil, mas...
- A noção mais intuitiva é algo chamado tempo do “relógio-de-parede” (Wall Clock Time).
  - Dispare um temporizador e inicie a execução de um programa no mesmo instante
  - Quando o programa terminar, pare o temporizador
  - O tempo decorrido é o tempo do “relógio-de-parede”.
    - $\text{Tempo Fim} - \text{Tempo Início}$

# Tempo de Execução

---

- Outra medida é o tempo de CPU
  - CPU TIME = tempo de relógio, porém NÃO incluindo o tempo gasto com I/O, ou esperando a CPU ser liberada por outro processo.
- CPU TIME tem dois componentes:
  - User time: tempo gasto no programa
  - System time: tempo gasto com rotinas do S.O.

# Como medir o tempo de Execução ?

---

- Manual: totalmente impreciso!
- Usando Ferramentas do sistema:
  - UNIX: comando `time`

# UNIX: comando time

```
% time ls /home/john/ -la -R
```

```
0.520u 1.570s 0:20.58 10.1% 0+0k 570+105io 0pf+0w
```

- 0.520u 0.52 seconds of user time
- 1.570s 1.57 seconds of system time
- 0:20.58 20.58 seconds of wall-clock time
- 10.1% 10.1% of CPU was used
- 0+0k memory used (text + data)
- 570+105io 570 input, 105 output (file system I/O)
- 0pf+0w 0 page faults and 0 swaps

\*Importante: o sistema analisado dever ser dedicado ao programa em execução, apenas rodando os processos padrão do S.O.

\*Existem várias funções similares para serem usadas apenas em trechos do programa, com melhor precisão.

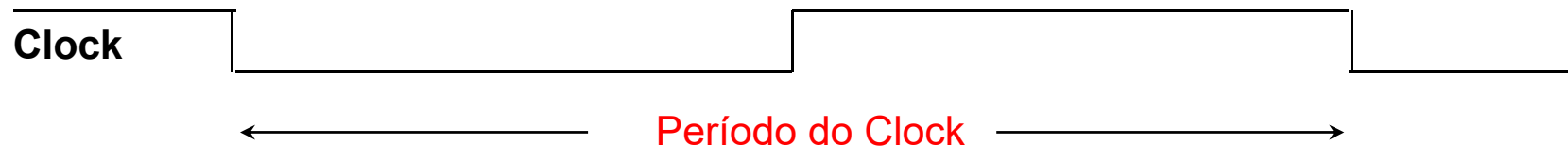
# Medindo o Tempo de CPU

---

???

# Taxa de Clock

- A execução de tarefas em um computador é controlado por um dispositivo (cristal) chamado **clock**, o qual oscila em intervalos de tempo constantes, determinando de forma **sincronizada** a execução de eventos de hardware.
- O **tempo de um ciclo de clock** é a quantidade de tempo decorrido em um período de clock completo (ex. 5 ns).



- A **taxa de clock** é o **inverso** do tempo do ciclo de clock.
  - ▣ Exemplo: Se um computador tem um tempo de ciclo de clock = 5 ns, então a sua taxa de clock é:  $1 / 5 \times 10^{-9} = 200 \text{ MHz}$

# Observação

## □ Sistema Internacional de Unidades - Prefixos p/ quantidades:

- $10^{18}$  Exa
- $10^{15}$  Peta
- $10^{12}$  Tera
- $10^9$  Giga
- $10^6$  Mega
- $10^3$  Kilo
- $10^2$  Hecto
- $10^1$  Deca
- $10^{-1}$  Deci
- $10^{-2}$  Centi
- $10^{-3}$  Mili
- $10^{-6}$  Micro
- $10^{-9}$  Nano
- $10^{-12}$  Pico
- $10^{-15}$  Femto
- $10^{-18}$  Ato



# Tempo de CPU

---


- O Tempo de CPU para executar um dado programa pode ser calculado através da seguinte expressão:
- ***Tempo de CPU:***  
*Nro. de ciclos de clock da CPU x tempo do ciclo de um clock*

# Tempo de CPU

- *Tempo de CPU:*
- *Nro. de ciclos de clock da CPU x tempo do ciclo de um clock*
- Dado que o tempo de ciclo de um clock e a taxa de clock são inversos:

- *Tempo de CPU:*

*Nro. de ciclos de clock da CPU / taxa de clock*


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

# Tempo de CPU

- *Tempo de CPU = Nro. de ciclos de clock da CPU x tempo do ciclo de um clock*
- *Tempo de CPU = Nro. de ciclos de clock da CPU / taxa de clock*
- *O número de ciclos de clock da CPU pode ser determinado pela seguinte expressão:*

$$\text{Número de ciclos de clock da CPU} = IC \times CPI$$

onde:

***IC = Número de instruções do programa***

***CPI = Número de ciclos de clock por instrução***

*Obs.: CPI é o número médio de ciclos por instrução*

# Tempo de CPU

- *Tempo de CPU = Nro. de ciclos de clock da CPU x tempo do ciclo de um clock*
- *Tempo de CPU = Nro. de ciclos de clock da CPU / taxa de clock*
- *Número de ciclos de clock da CPU = IC x CPI*

□ Assim,

▣ *Tempo de CPU = IC x CPI x tempo do ciclo do clock*

ou

▣ *Tempo de CPU = IC x CPI / taxa de clock*

□ Sendo medido em segundos →

$$\text{CPU time} = \frac{\text{IC} \times \text{CPI}}{\text{Clockrate}}$$

# Tempo de CPU

---

## □ Exemplo:

- ▣ Se um computador tem uma taxa de clock de 600 MHz, quanto tempo ele levaria para executar um programa com 2000 instruções, assumindo-se que o número médio de ciclos por instrução é 3 ?

# Tempo de CPU

## □ Exemplo:

- ▣ Se um computador tem uma taxa de clock de 600 MHz, quanto tempo ele levaria para executar um programa com 2000 instruções, assumindo-se que o número médio de ciclos por instrução é 3 ?

## □ Tempo de CPU = $IC \times CPI / \text{taxa de clock}$

- ▣ Tempo de CPU =  $2000 \times 3 / (600 \times 10^6)$
- ▣ Tempo de CPU =  $10 \times 10^{-6}$  segundos = 10 microsegundos

# Tempo de CPU

$$\text{CPU time} = \frac{\text{IC} \times \text{CPI}}{\text{Clockrate}}$$

- Como calcular o *IC* de um programa ?
- IC = Número de instruções executadas
  - ▣ Contar as instruções que são efetivamente executadas (detalhes mais à frente)

# Tempo de CPU

$$\text{CPU time} = \frac{\text{IC} \times \text{CPI}}{\text{Clockrate}}$$

- Como calcular o *CPI* de um programa ?
- CPI = Número médio de ciclos por instrução.

$$\text{CPI} = \sum_{j=1}^n (\text{CPI}_j \times F_j)$$

- onde

$\text{CPI}_j$  é o número de ciclos p/ executar a instrução  $j$ ,

$F_j$  é a frequência da instrução  $j$  no programa.



# Tempo de CPU

- Como calcular o *CPI* de um programa ?

- Exemplo:

Op	F	CPI
ALU	50%	1
Load	20%	5
Store	10%	3
Branch	20%	2
Total	100%	

# Tempo de CPU

- Como calcular o **CPI** de um programa ?

- Exemplo:

Op	F	CPI	CPI x F	% Tempo
ALU	50%	1	.5	23% (.5 / 2.2)
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%
Total	100%		<b>2.2</b>	100%

Valor calculado p/ CPI



# Determinando precisamente IC

- **IC** é difícil de determinar apenas olhando p/ o programa (código assembly), devido à imprevisibilidade durante o tempo de execução.

- **Exemplo 1: execução depende da entrada de dados do usuário:**

```
while(fgets(buffer,256,f))
{
    if (buffer[0] == '#')
        continue;
    if (buffer[0] == '>')
        process_command(&buffer[1]);
    else
        process_data(&(buffer[0]));
}
```

- **Exemplo 2 - limites do vetor:**

```
for (i=0; i<n; i++)
{
    a[i]++;
}
```

$$\text{CPU time} = \frac{\text{IC} \times \text{CPI}}{\text{Clockrate}}$$

# Determinando precisamente IC

- Conforme visto anteriormente, precisamos saber:
  - Instruções efetivamente executadas
  - Frequência relativa de cada uma delas
- O fluxo de instruções completo de um programa é chamado execution profile
- Mas como obtê-lo ?

# Execution profile

- Três formas de obtê-lo
  - Opção 1: usar contadores de hardware presentes nos processadores
  - Opção 2: usar execução instrumentada para incrementar contadores de software
  - Opção 3: Construa um interpretador para simular a execução do programa, incluindo contadores de instruções

Uma vez obtido o profile, é fácil obter o IC

# Determinando precisamente CPI

- CPI: Clock cycles per instruction
- Em um processador simples, bastaria verificar como cada instrução é implementada internamente
- Infelizmente, a complexidade dos processadores atuais não possibilita fazer isso de maneira precisa, devido a fatores como:
  - Execução paralela
  - Pipelining
  - Memória cache (vários níveis)
  - Memória virtual
  - I/O

# Determinando precisamente CPI

- O principal método usado para determinar CPI é chamado “trace-driven” simulation
  - Essencialmente, é um simulador da arquitetura, usado na execução de programas, que inclui todos os detalhes (e muitos outros) mencionados nos slides anteriores
  - Exemplos:
    - Perf
    - Valgrind
    - SimpleScalar
      - <http://www.simplescalar.com/>

# O que Analisar ?

- Agora podemos escrever código que se auto-cronometra... **mas que código ?**
  - O que são bons programas para rodar de modo a gerar conclusões sobre o desempenho relativo de diversas máquinas?
  - **O que são bons benchmarks?**
- Desafio: Se eu executo 10 benchmarks e lhe mostro seus resultados, mas o seu uso do computador não tem nada a ver c/ estes benchmarks, não vai ajudar em nada.
  - Felizmente, os computadores são usados em maneiras consideravelmente similares por muita pessoas.
  - Ex: desenvolvimento de programas, aplicações comerciais, jogos, assistir vídeos, simulações, etc.



# Tipos de Benchmarks

- **Aplicações Reais**
  - Apenas escolha um conjunto de aplicações reais, com dados de entrada razoáveis
  - Problema: portabilidade
    - As aplicações reais são sempre bagunçadas
- **Aplicações modificadas**
  - Tome alguma aplicação real e
    - Remova algumas das características não-portáteis remova algumas características menos interessantes/relevantes
      - Ex: remova todo o I/O para um benchmark de CPU
      - Adicione scripts para simular a interatividade com o usuário.

# Tipos de Benchmarks

---

- **Kernels**

- Extraia pequenos trechos críticos para o desempenho de uma aplicação

- **Benchmarks Sintéticos**

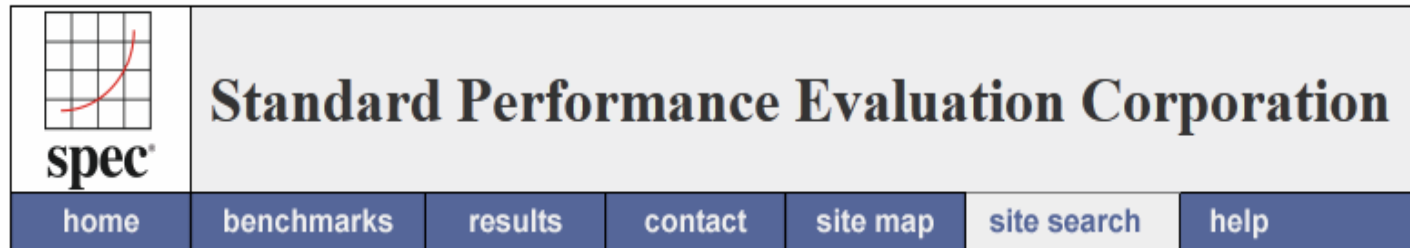
- Código artificialmente gerado
- Tenta imitar misturas representativas de operações e de operandos
- Dois exemplos famosos: Whetstone e Dhrystone

# Benchmark Suites












---

- Tipicamente, benchmarks são pacotes chamados "suites"
- Mais famoso/bem sucedido:
  - SPEC (Standard Performance Evaluation Corporation) benchmark
  - Cobre diferentes tipos de aplicações
  - [www.spec.org](http://www.spec.org)

# Benchmark Suites



## Benchmarks

-  [CPU](#)
-  [Graphics/Workstations](#)
-  [MPI/OMP](#)
-  [Java Client/Server](#)
-  [Mail Servers](#)
-  [Network File System](#)
-  [Power](#)
-  [SIP](#)
-  [SOA](#)
-  [Virtualization](#)
-  [Web Servers](#)

## Results Search

## Submitting Results

[CPU/Java/Power/SFS](#)  
[SIP/Virtualization](#)  
[MPI/OMP](#)  
[SPECapc/SPECviewperf](#)

## Order Benchmarks

## SPEC's Benchmarks and Published Results

### CPU

- **SPEC CPU2006**

[\[benchmark info\]](#) [\[published results\]](#) [\[support\]](#) [\[order benchmark\]](#)

Designed to provide performance measurements that can be used to compare compute-intensive workloads on different computer systems, SPEC CPU2006 contains two benchmark suites: CINT2006 for measuring and comparing compute-intensive integer performance, and CFP2006 for measuring and comparing compute-intensive floating point performance.

- **SPEC CPUv6**

[\[info\]](#)

The CPU Search Program seeks to encourage those outside of SPEC to assist us in locating applications that could be used in the next CPU-intensive benchmark suite, currently designated as SPEC CPUv6.

# Benchmark Suites

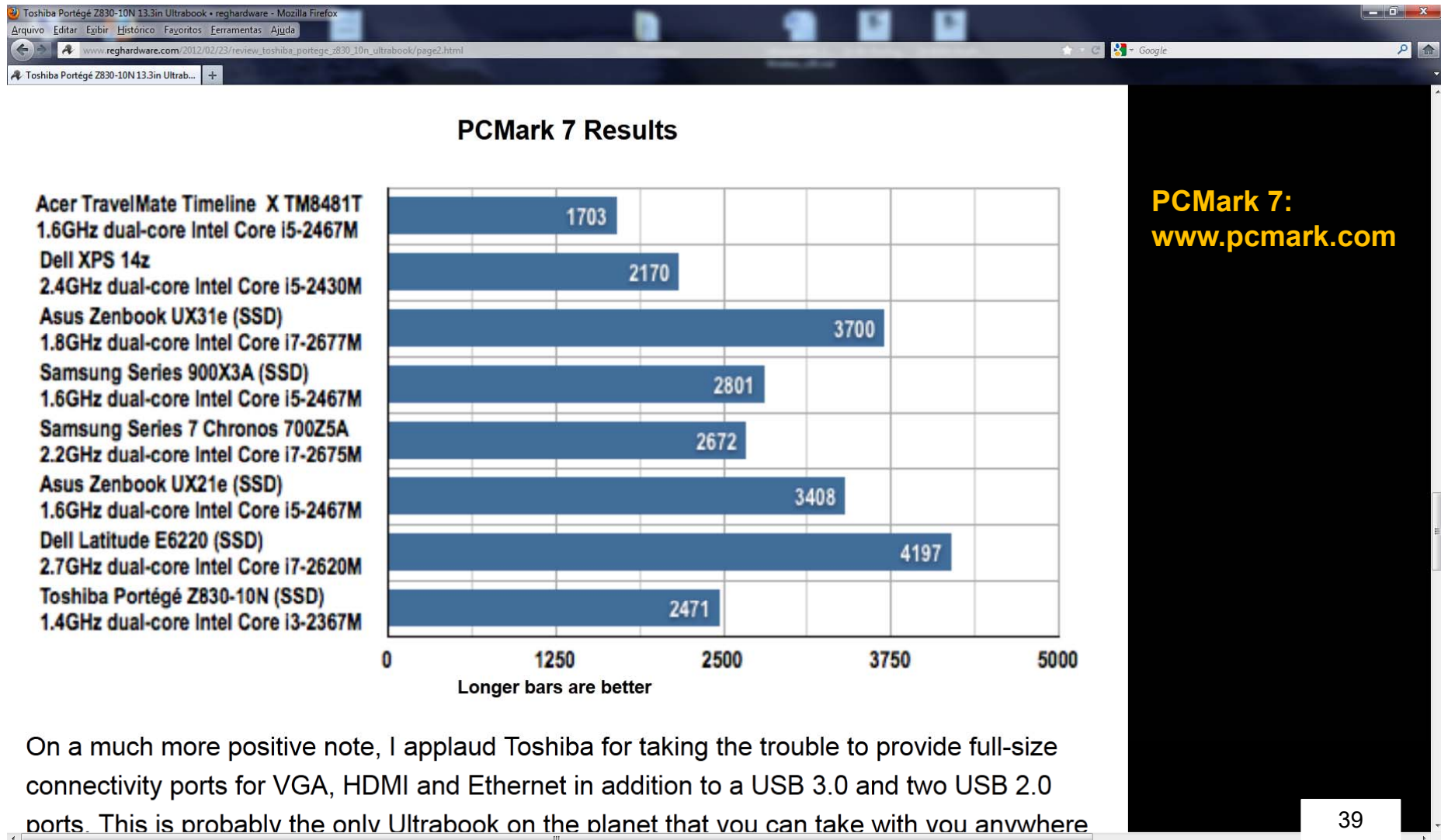
- **SPECviewperf® 11**  
[\[benchmark info\]](#) [\[published results\]](#) [\[support\]](#) [\[download benchmark\]](#) [\[order benchmark DVD\]](#)
- **SPECapc<sup>SM</sup> for 3ds Max™ 2011**  
[\[benchmark info\]](#) [\[published results\]](#) [\[faq\]](#) [\[support\]](#) [\[purchase benchmark\]](#)
- **SPECapc<sup>SM</sup> for Lightwave 3D® 9.6**  
[\[benchmark info\]](#) [\[published results\]](#) [\[support\]](#) [\[download benchmark\]](#)
- **SPECapc<sup>SM</sup> for Maya® 2009**  
[\[benchmark info\]](#) [\[published results\]](#) [\[support\]](#) [\[download benchmark\]](#)
- **SPECapc<sup>SM</sup> for Pro/ENGINEER™ Wildfire 2.0**  
[\[benchmark info\]](#) [\[published results\]](#) [\[support\]](#) [\[download benchmark\]](#)
- **SPECapc<sup>SM</sup> for SolidWorks 2007™**  
[\[benchmark info\]](#) [\[published results\]](#) [\[support\]](#) [\[download benchmark\]](#)
- **SPECapc<sup>SM</sup> for UGS NX 4**  
[\[benchmark info\]](#) [\[published results\]](#) [\[support\]](#) [\[download benchmark\]](#)
- [Previous versions of SPECapc and SPECviewperf benchmarks](#)

# SPEC CINT2006 p/ Opteron X4 2356

Name	Description	IC×10 <sup>9</sup>	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							11.7

High cache miss rates

# Benchmarks Comerciais- Ex



# Benchmarks Comerciais- Ex

## SYSMark 2007 - Overall

Score in SYSMarks - Higher is Better



SYSMark:  
<http://www.bapco.com>



# Server Benchmarks

- CPU benchmarking para servidores é similar ao de desktops, porém a ênfase é em throughput para sistemas multiprocessadores ou multi-core
- Uma questão básica para servidores é I/O
  - disco (file servers)
  - rede (Web servers)
- SPECFS: File server benchmark
- SPECWeb: Web server benchmark
- Transaction processing (TP) benchmarks
  - Acesso e atualização em bancos de dados
  - métrica: quantos por segundo
  - <http://www.tpc.org>

TPC - Benchmarks - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

www.tpc.org/information/benchmarks.asp

tpc benchmark - Pesquisa Google

TPC TPC - Benchmarks

TPC TPC-E - Top Ten Performance Results

TPC TPC-C - Top Ten Performance Results

# TPC Transaction Processing Performance Council

Advanced Search

The TPC defines transaction processing and database benchmarks and delivers trusted results to the industry.

- Home
- Results
  - TPC-C
  - TPC-E
  - TPC-H
- Benchmarks
  - TPC-C
    - Results
    - Description
    - FAQ
  - TPC-E
  - TPC-DS
  - TPC-H
  - Pricing Spec
  - TPC Energy
  - Obsolete
    - TPC-A
    - TPC-B
    - TPC-D
    - TPC-R
    - TPC-W
    - TPC-App
- Technical Articles
- Related Links
- What's New
- About the TPC
  - What is the TPC
  - Mailing List
  - Applications
  - Press
  - Documentation
- Who We Are
  - Members
  - Affiliates
- Member Login
- Contact Us
- Quick Links
  - Benchmark Status
  - Sign up for BSR
  - Univ & Research
  - How can I join?

## TPC Benchmarks

### TPC-C

TPC-C simulates a complete computing environment where a population of users executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. While the benchmark portrays the activity of a wholesale supplier, TPC-C is not limited to the activity of any particular business segment, but, rather represents any industry that must manage, sell, or distribute a product or service.

TPC-C involves a mix of five concurrent transactions of different types and complexity either executed on-line or queued for deferred execution. It does so by exercising a breadth of system components associated with such environments, which are characterized by:

- The simultaneous execution of multiple transaction types that span a breadth of complexity
- On-line and deferred transaction execution modes
- Multiple on-line terminal sessions
- Moderate system and application execution time
- Significant disk input/output
- Transaction integrity (ACID properties)
- Non-uniform distribution of data access through primary and secondary keys
- Databases consisting of many tables with a wide variety of sizes, attributes, and relationships
- Contention on data access and update

TPC-C performance is measured in new-order transactions per minute. The primary metrics are the transaction rate (tpmC), the associated price per transaction (\$/tpmC), and the availability date of the priced configuration.

[more >>](#)

### TPC-E

TPC Benchmark™ E (TPC-E) is a new On-Line Transaction Processing (OLTP) workload developed by the TPC. The TPC-E benchmark uses a database to model a brokerage firm with customers who generate transactions related to trades, account inquiries, and market research. The brokerage firm in turn interacts with financial markets to execute orders on behalf of the customers and updates relevant account information.

The benchmark is "scalable," meaning that the number of customers defined for the brokerage firm can be varied to represent the workloads of different-size businesses. The benchmark defines the required mix of transactions the benchmark must maintain. The TPC-E metric is given in transactions per second (tps). It specifically refers to the number of Trade-Result transactions the server can sustain over a period of time.

Although the underlying business model of TPC-E is a brokerage firm, the database schema, data population, transactions, and implementation rules have been designed to be broadly representative of modern OLTP systems.

TPC-E - Top Ten Performance Results - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

TPC www.tpc.org/tpce/results/tpce\_perf\_results.asp

tpc benchmark - Pesquisa Google x TPC - Benchmarks x TPC TPC-E - Top Ten Performance Results x TPC TPC-C - Top Ten Performance Results x

# TPC Transaction Processing Performance Council

Advanced Search

The TPC defines transaction processing and database benchmarks and delivers trusted results to the industry.

- Home
- Results
  - TPC-C
  - TPC-E
  - TPC-H
- Benchmarks
  - TPC-C
  - Results
  - Description
  - FAQ
  - TPC-E
  - TPC-DS
  - TPC-H
  - Pricing Spec
  - TPC Energy
  - Obsolete
  - TPC-A
  - TPC-B
  - TPC-D
  - TPC-R
  - TPC-W
  - TPC-App
- Technical Articles
- Related Links
- What's New
- About the TPC
  - What is the TPC
  - Mailing List
  - Applications
  - Press
  - Documentation
- Who We Are
  - Members
  - Affiliates
- Member Login
- Contact Us
- Quick Links
  - Benchmark Status
  - 43 Sign up for BSR
  - Univ & Research

## TPC-E - Top Ten Performance Results

### Version 1 Results

As of 16-Mar-2012 7:39 PM [GMT]

**Note 1:** The TPC believes it is not valid to compare prices or price/performance of results in different currencies.

☐ All Results
 ☐ Clustered Results
 ☐ Non-Clustered Results
 Currency

Rank	Company	System	Performance (tpsE)	Price/tpsE	Watts/tpsE	System Availability	Database	Operating System
1	IBM	IBM System x3850 X5	4,593.17	140.56 USD	NR	08/26/11	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Server 2008 R2 Enterprise Edition with SP1
2	FUJITSU	PRIMERGY RX900 S2	4,555.54	217.27 USD	1.00	07/01/11	Microsoft SQL Server 2008 R2 Datacenter Edition	Microsoft Windows Server 2008 R2 Enterprise Edition with SP1
3	FUJITSU	PRIMEQUEST 1800E2	4,414.79	226.19 USD	1.09	07/01/11	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Server 2008 R2 Enterprise Edition with SP1
	NEC	NEC	4,200.61	287.42 USD	NR	08/31/11	Microsoft SQL Server 2008 R2	Microsoft Windows Server 2008 R2

http://www.tpc.org/advancedsearch.asp

# Embedded Benchmarks

- Desenvolvidos devido à explosão no uso de dispositivos embarcados
- Um desafio é que os dispositivos embarcados são usados para uma grande variedade de aplicações
- Existe um padrão:: **EEMBC** (Embedded Microprocessor Benchmark Consortium)
  - Automotive/Industrial (scientific computing)
  - Consumer (multimedia)
  - Networking
  - Office automation (graphics/text)
  - Telecommunications (filtering, DSP)



The Embedded Microprocessor  
Benchmark Consortium  
A Non-profit Association

Founded in 1997, EEMBC (pronounced 'embassy') has its origins as the Embedded Microprocessor Benchmark Consortium. As a non-profit, industry association, EEMBC develops embedded benchmark software to help system designers select the optimal processors. EEMBC organizes this software into benchmark suites targeting telecom/networking, digital media, Java, automotive/industrial, consumer, and office equipment products. The MultiBench suite specifically targets the capabilities of multicore processors. Processor evolution into systems-on-chips (SoCs) has led EEMBC to evolve its benchmark suites to target Smartphones and browsers, networking systems, and hypervisors. Obtain all benchmarks by joining EEMBC's open membership or through corporate or university licensing. The EEMBC Technology Center manages new benchmark development and certification of benchmark test results.

### Available for Members and Licensees



### EEMBC NEWS, PRESS & MORE

[Subscribe to EEMBC News](#)

NEWS: EEMBC Brings Order to the Chaos of Android Benchmarking. [日本語](#)

NEWS: Benchmark allows Android performance to be evaluated.

NEWS: Android performance tools hit the market.

NEWS: Android devices get performance benchmarks.

MEMBER: ARM, Freescale launch Flycatcher - lowest power 32bit mcu yet.

MEMBER: ARM announces Cortex M0.

### EEMBC EVENTS



**Multicore Developer Conference**  
March 27-29, 2012— Santa Clara, CA

The 7th Annual Multicore Developers Conference will take place on March 27 - 29, 2012, co-located with Embedded Systems Conference Silicon Valley. [more »](#)



April 18 - 20, 2012

**COOL Chips XV**  
April 18-20, 2012— Yokohama Joho Bunka Center

### EEMBC BENCHMARKS

#### SYSTEM

#### Android, Smartphone Browsers and Java

**AndEBench™** provides a standardized, industry-accepted method of evaluating Android platform performance.

**BrowsingBench™** (Version 2.0 in development) - Evaluate the browsing experience on Smartphones and other systems.

**GrinderBench™** - Test a system's performance running Java applications.

#### Telecom/Networking System Benchmarks

**ETCPBench™** - TCP/IP performance and conformance for platforms from micro-controllers to high-end processors.

**DPBench™** (in development) - Evaluate throughput and latency to highlight the strengths and weaknesses of DPI systems, processors, and middleware.

#### Benchmark Testing Services

**EEMBC Technology Center (ETC)** - Services include porting and benchmark execution, performance analysis, and preparation of platforms for benchmark score certification. These services allow any company to make EEMBC benchmark testing an integral part of its product development and release process - without tying up internal engineering resources.

#### PROCESSOR

#### Microprocessor Benchmark Suites

**AutoBench™** (Version 2.0 in development) - Automotive, industrial, and general-purpose applications.

**ConsumerBench™** - Digital cameras, printers, and other embedded systems doing digital imaging tasks.

**CoreMark™** - Single number score for quick comparison of processor and microcontroller core functionality.

**DENBench™** - Digital entertainment products such as PDAs, mobile phones, MP3 players, cameras, TV set-top boxes, and in-car entertainment systems.

**EnergyBench™** - Power and energy performance with insights to power budget costs.

**FPBench™** (in development) - Floating-point performance in graphics, audio, motor control, and other high-end processing tasks.

**MultiBench™** - Multicore architectures, memory bottlenecks, OS thread scheduling, synchronization efficiency.

**Networking** - Moving and analyzing packets in networking applications.

**OABench™** - Office Automation tasks in printers, plotters, and other systems that handle text and image processing tasks.

**TeleBench™** - Telecommunications processors in modem, xDSL, and related fixed-telecom applications.





## Resumindo Resultados da análise c/ Benchmarks

- Os resultados de benchmarks são frequentemente “enganadores”
  - Informação incompletas sobre como os resultados foram obtidas
  - Táticas secretos das companhias e projetistas
- Para piorar, benchmarks contêm programas múltiplos, de modo que o relatado é o desempenho de um sistema em uma variedade de programas
- Pergunta: como resumir os resultados apresentados ?

# Resumindo Resultados da análise c/ Benchmarks

- Exemplo
  - Considere um benchmark que consista em 2 programas
  - Considere os seguintes resultados publicados a respeito de 3 sistemas diferentes

	System A	System B	System C
P1	1s	10s	20s
P2	1000s	100s	20s
<b>Total</b>	<b>1001s</b>	<b>110s</b>	<b>40s</b>

# Resumindo Resultados da análise c/ Benchmarks

- Métrica mais simples: Média Aritmética
  - supor que na vida real todos os programas do benchmark são executados um número idêntico de vezes
  - \*Na prática, nem sempre isso é verdade.
- Média Aritmética Ponderada
  - adicione pesos para ponderar uma mistura irregular de programas
  - exemplo: Eu rodo P1 5 vezes mais frequentemente do que P2, assim dou o peso  $5/6$  a P1 e  $1/6$  a P2
  - cada usuário/comprador potenciais fornecem os pesos que julgam ideal



## Resumindo Resultados da análise c/ Benchmarks

- Tempos de execução normalizados
  - Escolha de “uma máquina referência”
  - Para cada programa no benchmark, normalize seu tempo de execução àquela na máquina da referência (isto é, divida pelo tempo de execução da máquina referência)
  - Calcule a média destes tempos de execução normalizados
  - O objetivo é dizer que a “máquina B é 3x mais rápida do que a máquina referência”, conseqüentemente B é mais rápido do que A.

# Que tipo de Média ?

Arithmetic:

$$\frac{1}{n} \sum_{i=1}^n \text{normalized execution time}_i$$

Geometric:

$$\left[ \prod_{i=1}^n \text{normalized execution time}_i \right]^{1/n}$$

- Ambas apresentam vantagens e desvantagens ☹  
(ver detalhes no livro)

# O que fazer?

- Solução Ideal:
  - Avalie usando uma carga de trabalho real
  - estabeleça alguma similaridade entre os programas executados na vida real e os programas do benchmark
    - por exemplo, todos os compiladores parecem c/ o GCC, que está no benchmark de SPECint
  - compute os pesos (isto é, frequências da execução)
  - compute a média aritmética ponderada
- Mensagem principal: saiba o que você fará com a máquina
- Não confie totalmente nos resultados resumidos por outros
  - Faça seu próprio resumo sempre que possível

# Speedup

- O ganho do desempenho devido às melhorias a um sistema é medido pelo *speedup* (aceleração):

$$\text{speedup} = \frac{\text{Execution time without enhancement}}{\text{Execution time with enhancement}}$$

- Exemplo: "Dobrar o tamanho do Cache L2 conduziu a um speedup 1.34 para a benchmark SPECint."
- Quando o speedup é menor que 1, significa que houve perda de desempenho.

# Lei de Amdahl's

---

- A lei de Amdahl
  - O ganho de desempenho que pode ser obtido melhorando uma determinada parte do sistema é limitado pela fração de tempo que essa parte é utilizada
  - Speedup:
    - A fração do programa que pode ser convertida para tirar proveito da melhoria no sistema
    - Por exemplo, se a melhoria faz a CPU mais rápida, provavelmente não vai acelerar o I/O. Assim, se um programa gasta 10s com I/O, ainda gastará estes 10 segundos com I/O, mesmo com a CPU mais rápida

# Má Notícia: Lei de Amdahl's

---

- Suponha que você vá comprar pão na padaria
  - Você caminha 25 minutos até lá
  - Chegando lá, você espera 25 minutos na fila
- Ao invés de caminhar, você pode ir de carro em 5 minutos
  - O tempo total caiu de 50 para 30 minutos

# Má Notícia: Lei de Amdahl's

- Suponha que você vá comprar pão na padaria
  - Você caminha 25 minutos até lá
  - Chegando lá, você espera 25 minutos na fila
- Ao invés de caminhar, você pode ir de carro em 5 minutos
- O tempo total caiu de 50 p/ 30 minutos
- Assim, acelerando seu trajeto por um fator de  $25/5 = 5$ , você acelerou a atividade total apenas por um fator de  $5/3 = 1.66$ 
  - Valeu a pena ?
- Se eu gasto seis meses de esforço e \$100K para acelerar uma parte de um sistema em 200%, e apenas obtenho um ganho total de 60%, pode não valer a pena.

# Lei de Amdahl formalizada

- Considere um programa cuja a execução consista em duas fases
- Uma possível melhoria no sistema pode apenas acelerar a fase #2

Programa no sistema original



Tempo original:  $T = T_1 + T_2$

$T_1$  tempo que não pode ser melhorado

$T_2$  = tempo que pode ser melhorado

Programa no sistema c/ a melhoria:



Novo Tempo:  $T' = T_1' + T_2'$

$T_2'$  = tempo após o melhoramento



# Lei de Amdahl formalizada

Programa no sistema original



Programa no sistema c/ melhoria

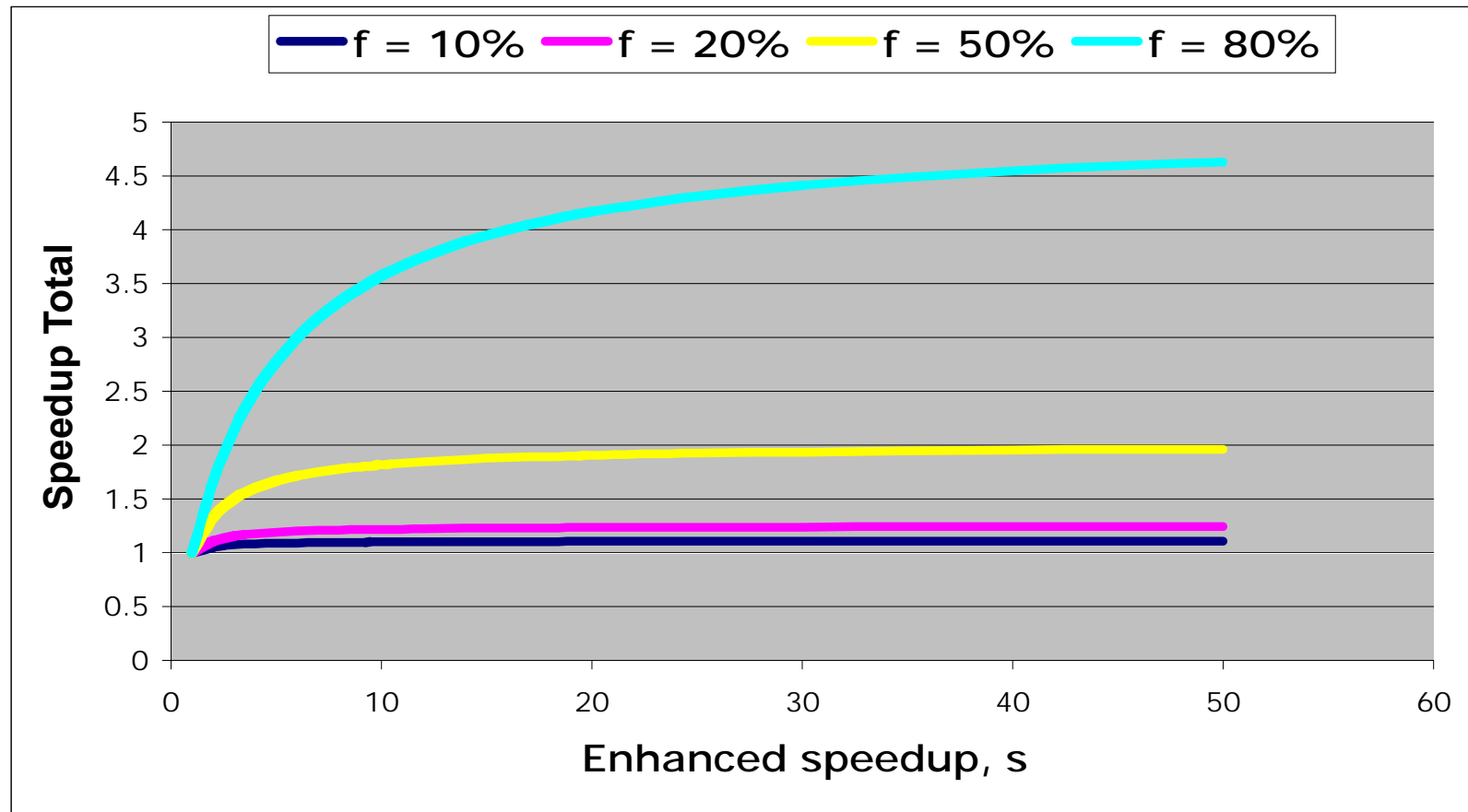


- **f**: fração do tempo de execução original que se beneficia da melhoria:  
 $T_2 / T$
- **s**: speedup da fração do código que se beneficia da melhoria:  
 $= T_2 / T_2'$

Lei de Amdahl's:  $\text{Speedup} = 1 / (1 - f + f/s)$

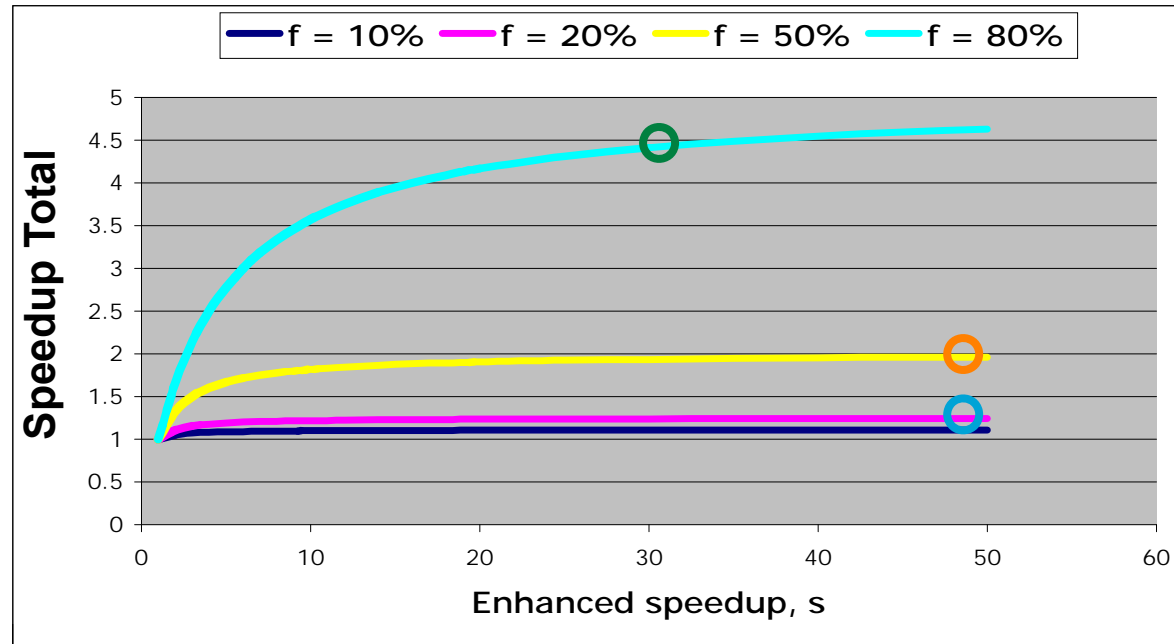
Guarde esta fórmula

# Lei de Amdahl: Exemplo



Plot de  $1/(1 - f + f/s)$  p/ 4 valores de  $f$  e valores crescentes de  $s$

# Lei de Amdahl: Exemplo



Lembre-se  
sempre deste  
gráfico !

- Uma melhoria de 50x que beneficia apenas 20% do programa resulta em um speedup geral de apenas 1.25
- Uma melhoria de 50x que beneficia apenas 50% do programa resulta em um speedup geral de apenas 2.0
- Uma melhoria de 30x que beneficia apenas 80% do programa resulta em um speedup geral de apenas 4.5
- Uma melhoria que beneficia apenas 10% é praticamente inutil

# Lei de Amdahl: Exemplo

- Frequentemente usada p/ comparar duas opções de projeto
- **Exemplo:**
  - As aplicações gráficas usam muita o cálculo da raiz quadrada em ponto flutuante (FPSQR)
  - Suponha que FPSQR corresponda a 20% do tempo de execução total do benchmark p/ aplicações gráficas, enquanto 70% do tempo total corresponde a outras operações em ponto flutuante.
  - **Opção #1:** melhorar FPSQR por um fator de 10, usando hardware customizado
  - **Opção #2:** melhorar todas as operações em ponto flutuante por um fator de 1.6, usando um novo processador

# Lei de Amdahl: Exemplo

$$\text{Speedup} = 1 / (1 - f + f/s).$$

- **Opção #1:** melhorar FPSQR por um fator de 10, usando hardware customizado
  - $\text{speedup} = 1 / (1 - 0.2 + 0.2 / 10) = 1/0.82 = 1.22$
- **Opção #2:** melhorar todas as operações em ponto flutuante por um fator de 1.6, usando um novo processador
  - $\text{speedup} = 1 / (1 - 0.7 + 0.7/1.6) = 1.35$
- **Conclusão:** Opção #2 é melhor, e provavelmente mais barata

# Conclusões

---

- Aspectos relacionados à medição do desempenho de sistemas de computação:
  - Por que medir
  - Como medir
  - Metodologia e ferramentas de medição
  - Benchmarks
- Ideia básica: torne o caso comum mais rápido
  - Lei de Amdahl : O ganho de desempenho que pode ser obtido melhorando uma determinada parte do sistema é limitado pela fração de tempo que essa parte é utilizada durante a sua operação.
    - Aspecto importantíssimo em todo trabalho de hardware e/ou software visando ganhos de desempenho.