PERFORMANCE MODELOS –PARADIGMAS-

Universidade Estadual de Mato Grosso do Sul – UEMS

Curso de Ciência da Computação

Algoritmos Paralelos e Distribuídos

Performance

- Como medir desempenho?

Como se mede o impacto de uma melhoria no desempenho de um computador?

Clock?

Flops?

Desempenho (performance) deve ser medida através de um conjunto de tarefas executadas por unidade de tempo?

Quanto maior melhor?

Desempenho deve ser medido pelo tempo gasto para realizar uma determinada tarefa?

Quanto menor melhor?

Medidas de desempenho

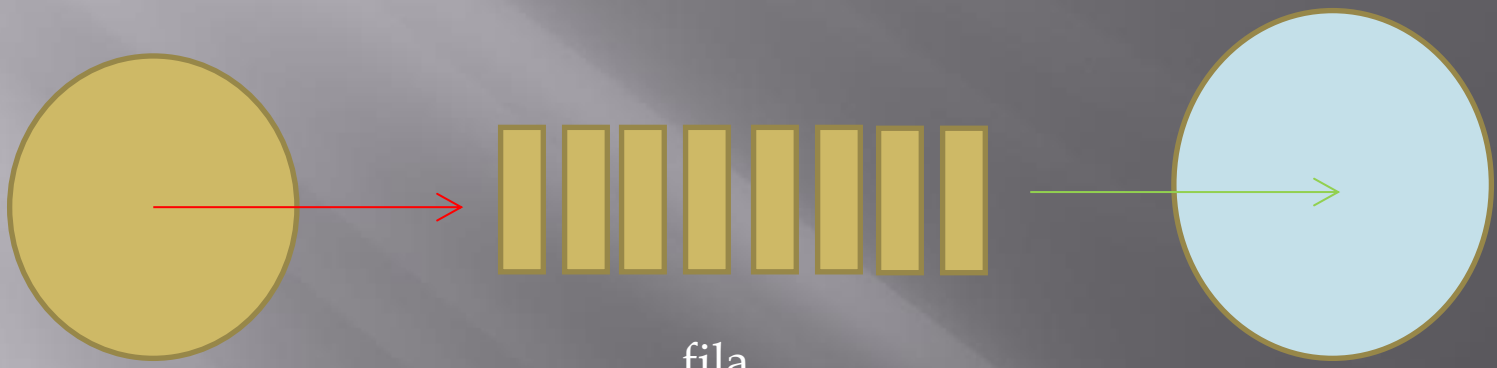
- *Throughput* (Vazão): Número de tarefas concluídas por unidade de tempo.
  - Por exemplo: instruções por segundo, MFLOPs, Mbps.
- Tempo de Resposta (latência): Tempo consumido para executar uma determinada tarefa ou conjunto de tarefas.

Dúvidas?

- Aumentar o *throughput* sempre melhora o tempo de resposta?

- Diminuir o tempo de resposta sempre melhora o *throughput?*

Modelo Simples – Produtor Consumidor

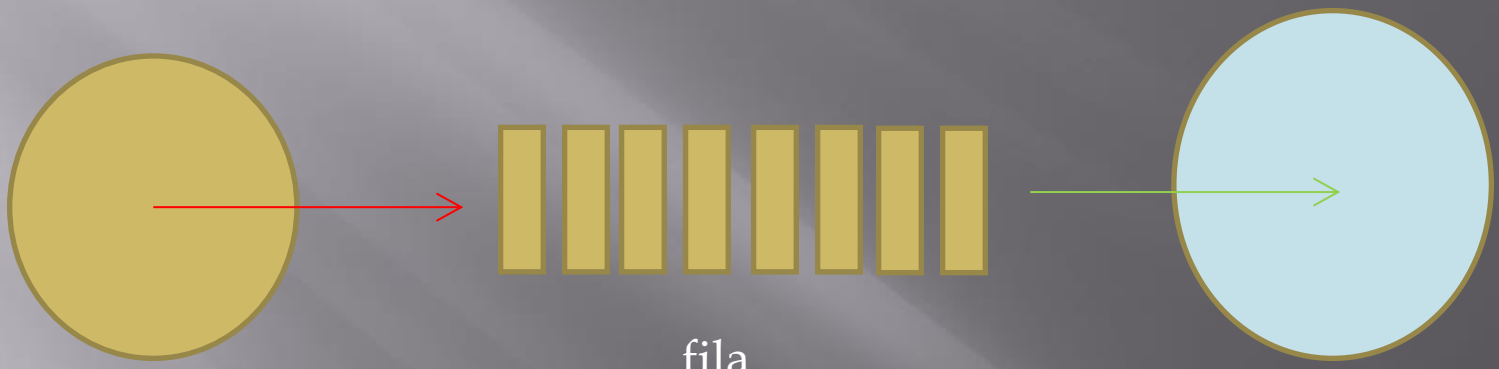


Latência (Tempo de Resposta): Tempo médio para a conclusão de uma tarefa.

Para minimizar:

- A fila deveria estar vazia
- O servidor deveria estar ocioso (idle).

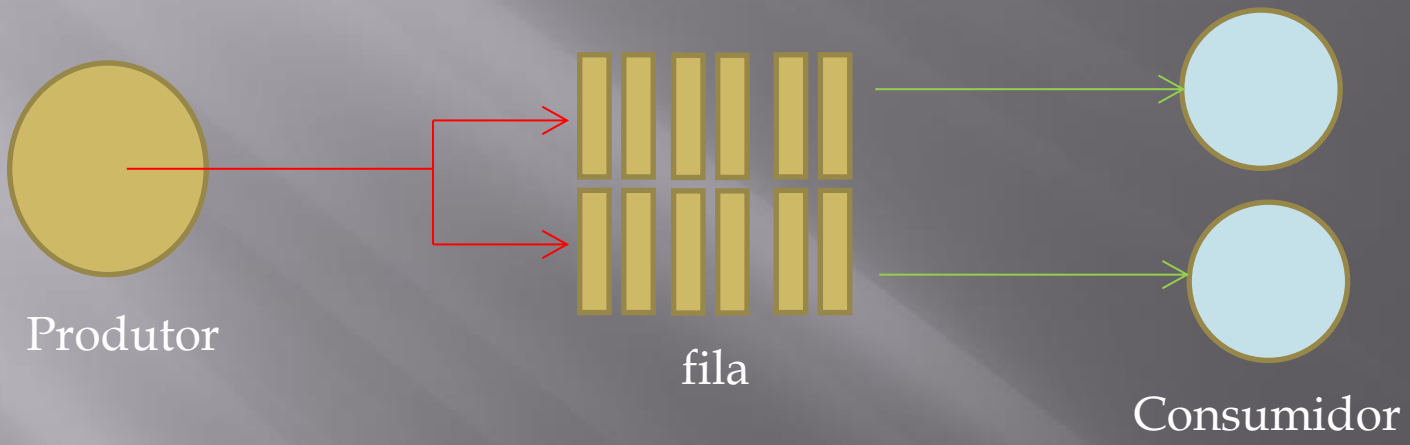
Modelo Simples – Produtor Consumidor



Throughput ("Taxa de Finalização"): Número de tarefas concluídas por unidade de tempo.
Para maximizar:

- A fila **nunca** deveria estar vazia
- O servidor **nunca** deveria estar ocioso (idle).

Aumentando o Throughput



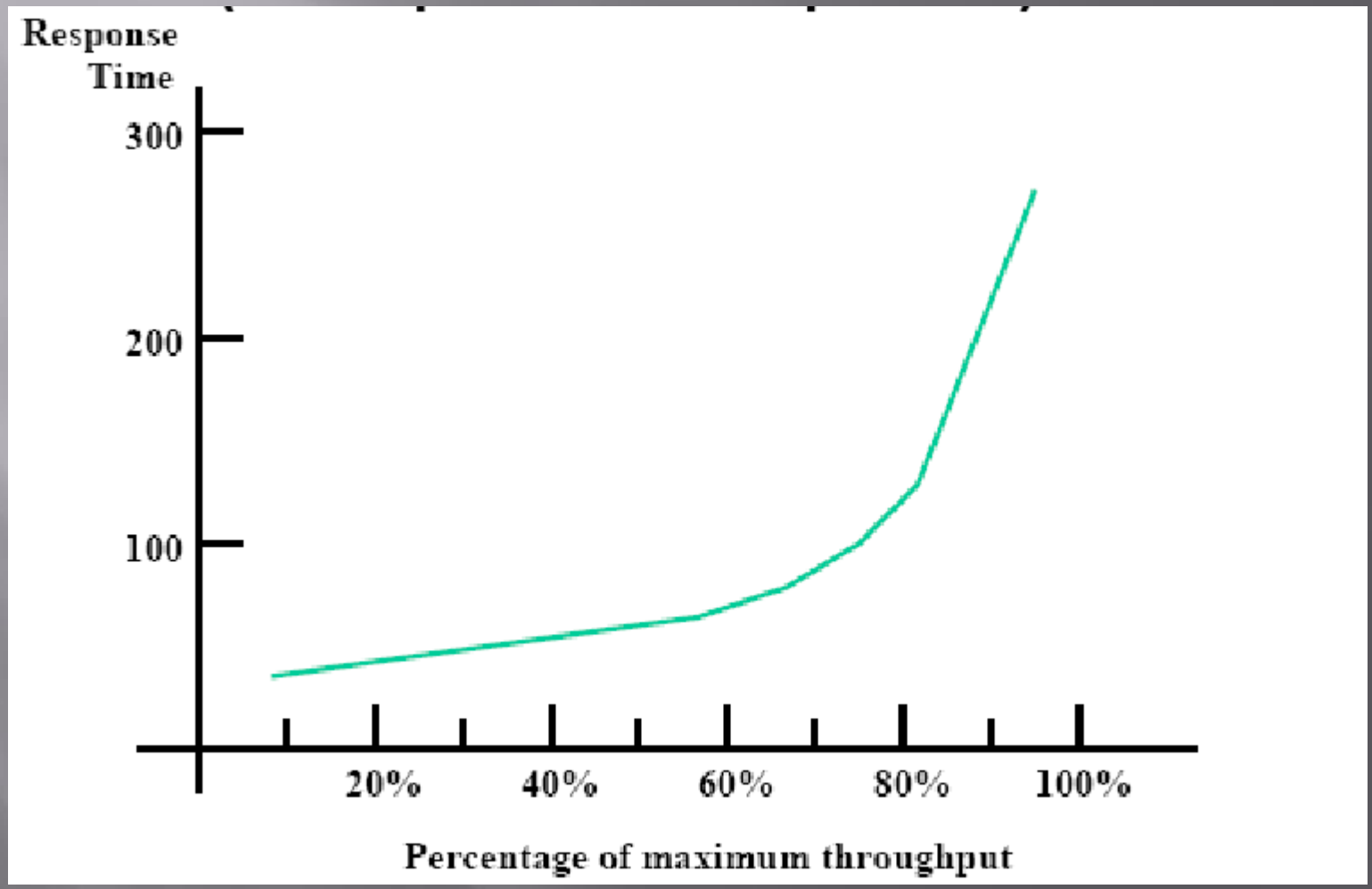
Em geral o *Throughput* pode ser incrementado:

- Colocando mais *hardware* (reduzindo latência relacionada a carga).

Tempo de resposta é muito mais difícil de reduzir:

- É preciso otimizar a arquitetura e/ou tecnologia de implementação.

Gráfico Qualitativo – Aumento de Throuput X Latência (Tempo de resposta).



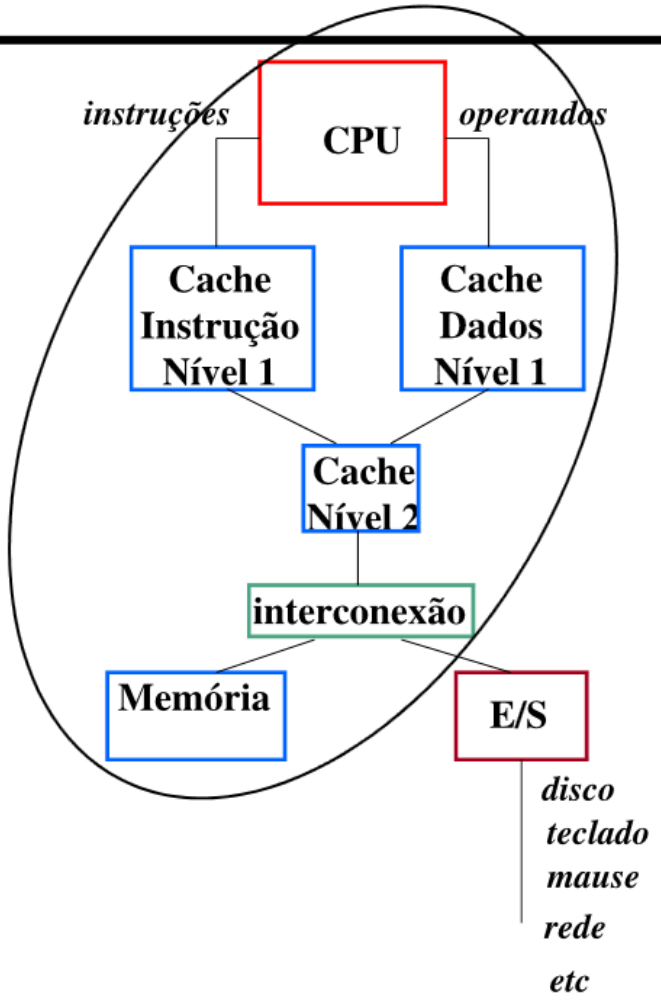
Desempenho

- Embora se preocupe com o *throughput* ou a latência, o mais importante é o tempo total de conclusão das tarefas de interesse do usuário.
  - O computador mais rápido é aquele que atende a seu cliente em menor tempo.
- Quando o número de tarefas é muito elevado e resultados parciais relevantes para o cliente podem ser produzidos, é possível "ocultar a latência".
  - O que você acha que isso significa?

Avaliar só o necessário

Avaliando só o necessário

- Decompondo a execução em componentes permite focar em pontos importantes da arquitetura
  - Operações de E/S são freqüentemente sobrepostas por outras tarefas na CPU
  - O subsistema de E/S pode ser desenvolvido de forma quase independente do resto
- Tempo de *CPU* ignora o tempo de execução do componente de E/S



Tempo de CPU - Métrica

- É o tempo gasto pela CPU, cache (às vezes mais de uma) e a memória para a execução de uma tarefa;
- Ignora o tempo requerido por operações de E/S;
- Composto de :
  - Tempo de CPU do usuário: Tempo gasto pelo programa do usuário;
  - Tempo de CPU do sistema: Tempo gasto pelo sistema operacional;

Tempo de CPU - Métrica

- Tempo de CPU do usuário é normalmente avaliado usando:
  - Vários *benchmarks* padrão, como SPEC com baixa atividade do SO;
  - Vários simuladores de arquitetura não suportam SO;
    - Muitas vezes a programação requer a inclusão de bibliotecas;
  - O código do SO frequentemente não está disponível para avaliação;

Tempo de CPU

- Tempo de CPU = CICLOS X TC = INST X CPI X TC
- CICLOS
  - Número total de ciclos para executar um programa;
- TC
  - Tempo do ciclo do relógio (período do relógio);
  - 1/ frequência do relógio;
- INST
  - Número total de instruções assembly executadas;
- CPI
  - Número médio de ciclos do relógio por instrução executada;
  - Total de ciclos do relógio / Total de instruções executadas
  - (Ciclos / Inst)
  - Diferentes tipos de instruções (add, divide, sub, etc) podem ter diferentes números de ciclos de relógio para executar;

Desempenho

CICLOS TC

$$T_{CPU} = \frac{segundos}{programa} = \frac{ciclos}{programa} \times \frac{segundos}{ciclos}$$

$$T_{CPU} = \frac{segundos}{programa} = \frac{instruções}{programa} \times \frac{ciclos}{instrução} \times \frac{segundos}{ciclos}$$

Desempenho - Terminologia

$$T_{CPU} = \frac{segundos}{programa} = \frac{instruções}{programa} \times \frac{ciclos}{instrução} \times \frac{segundos}{ciclos}$$

Nº de instruções por programa

Ciclos de clock por instrução (CPI)

Período de clock

$$T_{exec} = INST \times CPI \times TC$$

Exemplo de Tempo de CPU

lw $4, 0($2)	**2 cycles**
lw $6, 4($2)	**2 cycles**
add $4, $4, $6	**1 cycle**
sw $4, 0($2)	**1 cycle**

- CICLOS = 6
- INST = 4
- CPI = 6/4 = 1.5
- TC = 1ns
- CPU Time = CICLOS x TC = 6 x 1ns = 6ns

Exemplo de Tempo de CPU

- Considere duas implementações do mesmo conjunto de instruções (*Instruction Set Architecture* – ISA).

- Para um mesmo programa:
  - Máquina A: TC = 0,25 ns e CPI = 2,0
  - Máquina B: TC = 0,5 ns e CPI = 1,2

- Qual a máquina mais rápida para estes programas? Quanto mais rápida?

Exemplo de Tempo de CPU

- Ambas as máquinas executam o mesmo número de instruções (INST), pois executam o mesmo programa;
- Primeiro Passo: encontrar o tempo de execução do programa para cada máquina;

$$T_{CPU\_A} = INST \times CPI_A \times TC \Rightarrow T_{CPU\_A} = INST \times 2{,}0 \times 250ps$$

$$T_{CPU\_B} = INST \times CPI_B \times TC \Rightarrow T_{CPU\_B} = INST \times 1{,}2 \times 500ps$$

Exemplo de Tempo de CPU

- Segundo passo: Encontrar a razão entre os desempenhos das máquinas;

$$\frac{Desempenho_A}{Desempenho_B} = \frac{T_{CPU\_B}}{T_{CPU\_A}} = \frac{INST \times 1{,}2 \times 500}{INST \times 2{,}0 \times 250} = 1{,}2$$

- Portanto, a máquina A é 1,2 vez mais rápida que a máquina B.

Definindo desempenho de um sistema em uma "Tarefa"

- Considerando tempo de execução, temos:
  - Desempenho(x)= 1/TempoExecução(X)

- Logo, dizer que X é n vezes mais rápido que Y, significa:

$$\frac{Desempenho(x)}{Desempenho(y)} = n$$

ou,

$$\frac{Tempo\_Execução(Y)}{Tempo\_Execução(X)} = n$$

Medindo o Aumento de desempenho

Aumento de desempenho ocorrido, devido a uma melhoria E.

$$speedup(E) = \frac{ex\_time(\ )}{ex\_time(E)} = \frac{performance(E)}{performance(\ )}$$

Existem várias técnicas, componentes e meios de comunicação que podem ser alvo de aperfeiçoamentos. Por outro lado, existem também vários “tipos” de instruções quais devem ser melhoradas preferencialmente.

O que deve ser priorizado para possível melhoria?

Caso comum e a Lei de Amdahl

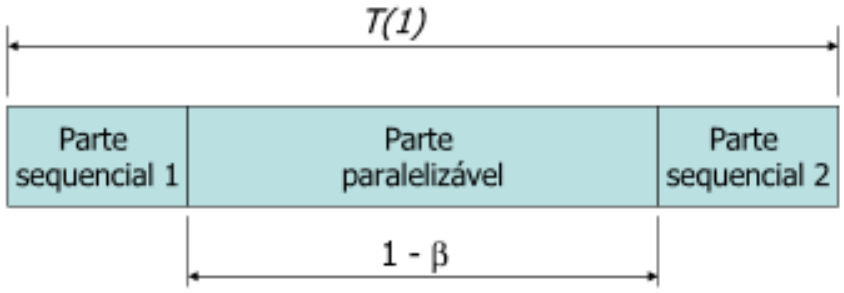
Lei de Amdahl

- Alguns exemplos de melhoria de desempenho de computadores passam pelo uso de processadores paralelos, hierarquia de cache e *speedup* no tempo de acesso da memória e na taxa de transferência de E/S;
- A Lei de Amdahl, proposta por Gene Amdahl lida com o potencial de *speedup* de um programa usando múltiplos processares em comparação com um único processador.

Lei de Amdahl

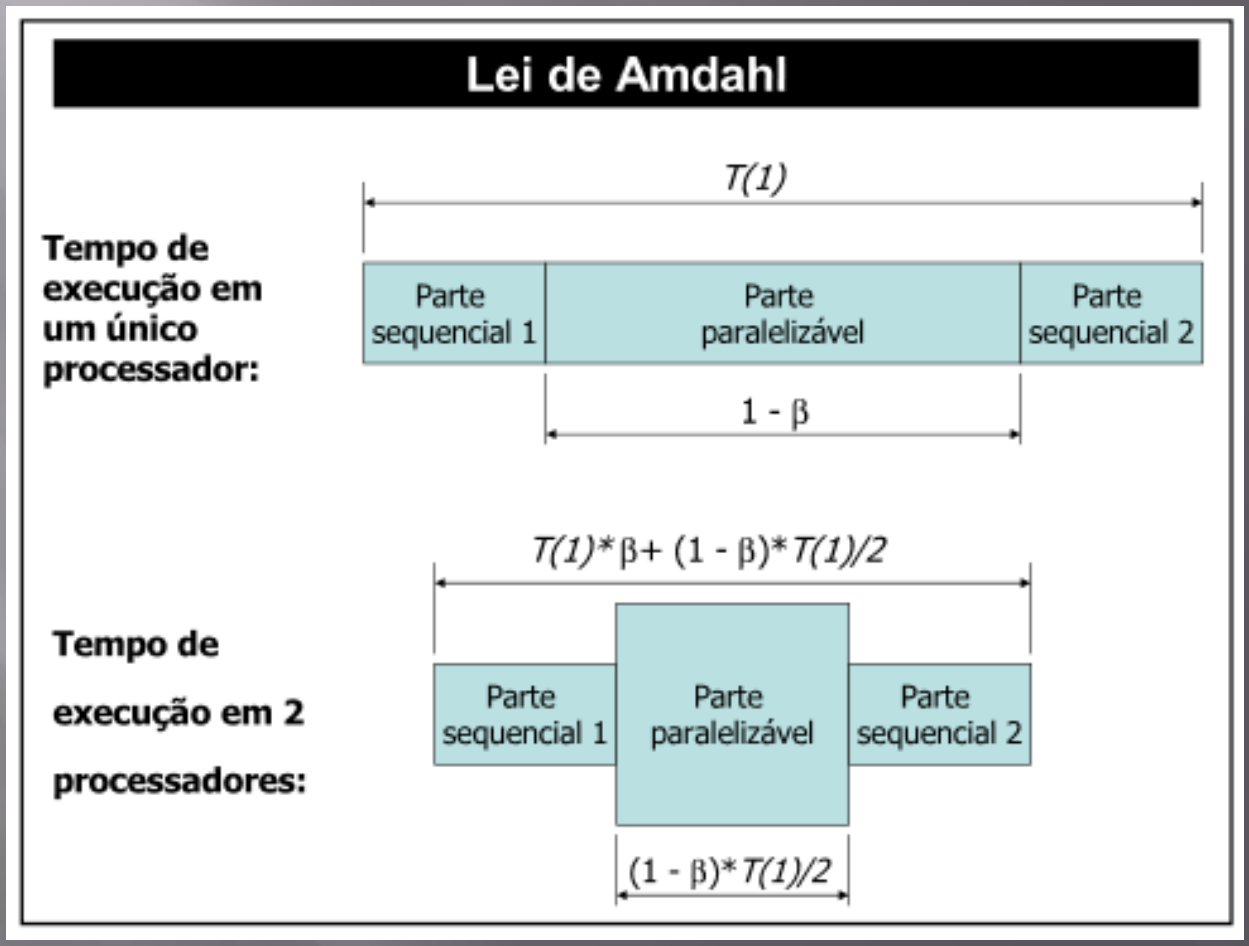
Lei de Amdahl (1967)

Tempo de execução em um único processador:



β = fraçao do programa que é sequencial

Lei de Amdahl



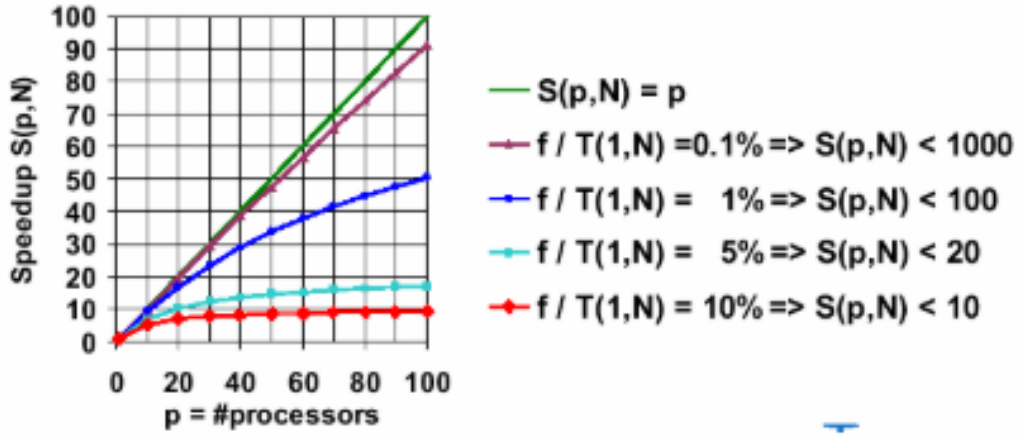
Lei de Amdahl

Lei de Amdahl

$T(1,N) = f + (T(1,N) - f)$ f ... sequential part of code that can not be done in parallel

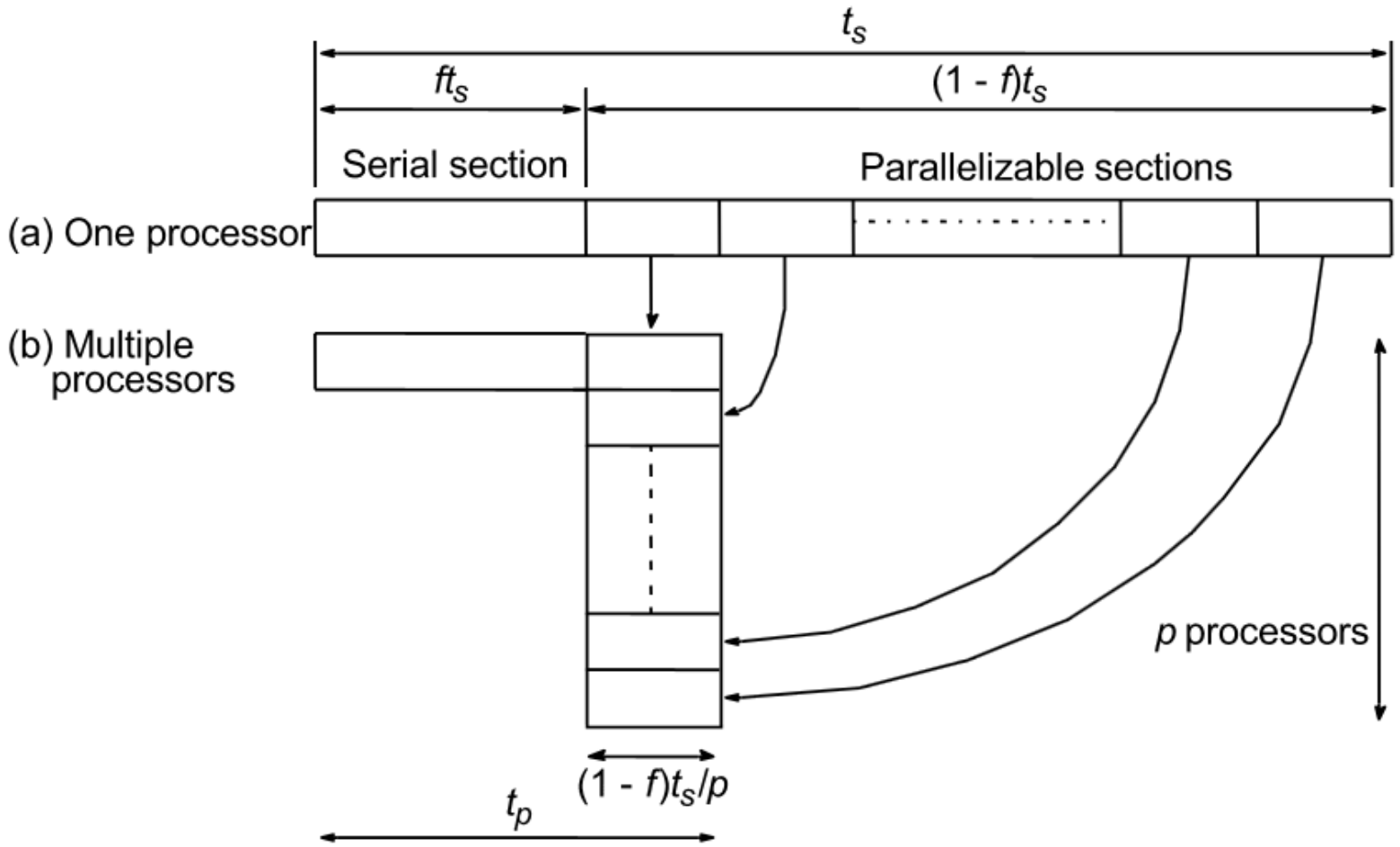
$S(p,N) = T(1,N) / T(p,N) = T(1,N) / (f + (T(1,N) - f) / p)$

For p —> infinity, speedup is limited by $S(p,N) < T(1,N) / f$



Lei de Amdahl

The speedup using multiple processors is limited by the time needed for the sequential fraction of the program



Lei de Amdahl

- Seja
  - f = proporção do programa que pode ser totalmente paralelizada.
    - (1 – f) é a proporção que permanece serial;
  - Então, o máximo *speedup* que se pode encontrar usando p processadores é:

$$Sp(P) = \frac{1}{(1-f) + f/P}$$

  - Amdahl usou esse argumento para dar suporte ao desemvolvimento de processadores únicos de ultra-velocidade na década de 1960.

Lei de Gustafson-Barsis

- A lei de Amdahl tornou-se um incômodo para os fabricantes de máquinas de grande porte;
- No final da década de 80, Gordon Bell ofereceu um prêmio anual de U$1.000,00 para quem pudesse utilizar processamento paralelo de maneira eficiente para resolver problemas reais;
- Em 1987, um grupo de pesquisadores do laboratório de Sandia (computação científica nos EUA) obtiveram speedups de aproximadamente 1000 para problemas com β (fração sequencial) entre 0.004 e 0.008, enquanto a lei de Amdahl previa speedup de apenas 125 a 250 para estes casos.

Lei de Gustafson-Barsis

- John Gustafson e Ed Barsis formalizaram o conceito por básico da aparente contradição.
  - A chave é que Amdahl assume que o valor de β (fração sequencial) seja constante para qualquer valor de *p* (número de processadores); de fato (1 – β) quase nunca é independente de N;
- Lei de Gustafson-Barsis

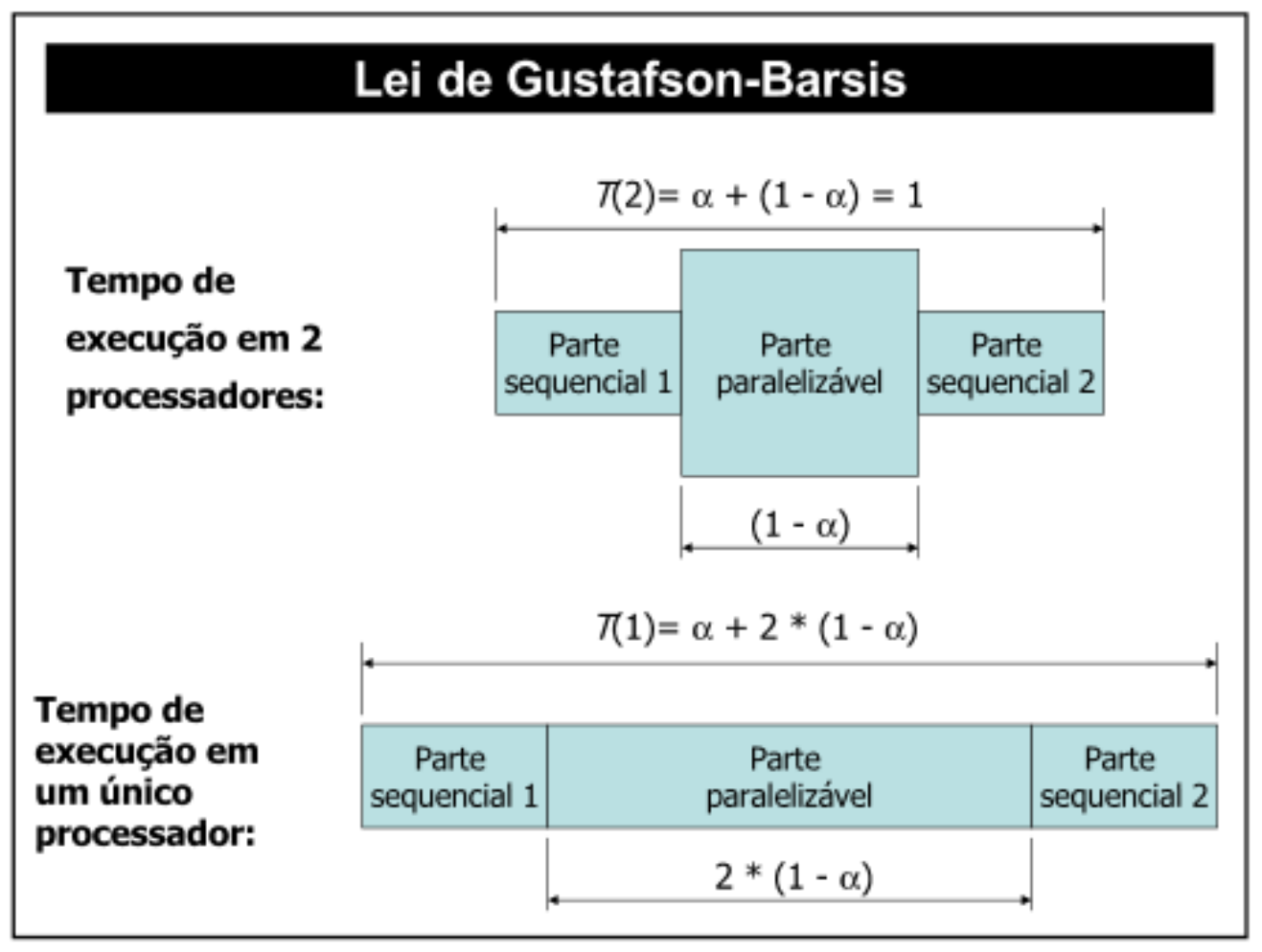
$$S = T(1)/T(p)$$

$$T(p) = 1$$

$$T(p) = \alpha + (1-\alpha) = 1$$

fraçao sequencial do programa

Lei de Gustafson-Barsis



Comparação Amdahl - Gustafson

Lei de Amdahl

$$S = \frac{1}{\left(\beta + \frac{(1-\beta)}{p}\right)}$$

S =Speed-up

p = n. de porcessadores

β = fraçao sequencial do programa

Lei de Gustafson-Barsis

$$S = p - \alpha(p-1)$$

S =Speed-up

p = n. de porcessadores

α = fraçao sequencial do programa

Recordando alguns pontos e comparando com processamento distribuído

- Ambas áreas têm os mesmos complicadores
  - Custo de comunicação
  - Distribuição dos dados
  - Dependências
- Motivações diferentes
  - Motivação do processamento paralelo é o ganho de desempenho
    - Unidades ativas estão normalmente na mesma máquina → custos de comunicação menores (baixa latência)
  - Motivação do processamento distribuído é a modelagem e o aproveitamento de recursos
    - Unidades ativas estão normalmente afastadas → custos de comunicação maiores (alta latência)

Processamento Paralelo x Distribuído

- Aumento de unidades ativas não aumenta *necessariamente* o desempenho
  - Quantidade de trabalho e arquitetura alvo limitam o número de unidades ativas usadas eficientemente
  - Muitas unidades ativas para uma quantidade limitada de trabalho prejudica o desempenho do sistema
    - O tempo pode aumentar!!
  - Complicadores
    - Dependências de dados
    - Distribuição dos dados
    - Sincronização
    - Áreas críticas

Exemplo

- Construção de um muro
  - 1 pedreiro faz este muro em 3 horas
  - 2 pedreiros fazem em 2 horas
  - 3 pedreiros fazem em 1 hora e meia
  - 4 pedreiros fazem em 2 horas (aumentou o tempo !!!)
  - 8 pedreiros fazem em 3 horas e meia (tempo pior que a versão sem paralelismo !!!)
  - Avaliação dos dados obtidos e porque ocorreu o aumento de tempo?
  - Incidência de muitos complicadores → ganho de desempenho *não é proporcional* ao acréscimo de unidades ativas
    - Duplicação do número de pedreiros de 1 para dois não reduz o tempo de execução pela metade

Exemplo

- Complicadores encontrados e a analogia com a computação paralela
  - O muro só pode ser feito de baixo para cima
    - Dependência de dados
  - Os tijolos têm que ser distribuídos entre os pedreiros
    - Distribuição dos dados
  - Um pedreiro não pode levantar o muro do seu lado muito na frente dos outros pedreiros. Ritmo de subida do muro é dado pelo pedreiro mais lento
    - Sincronização
  - Se só existir um carrinho com cimento este será disputado por todos os pedreiros
    - Áreas críticas

Desempenho da Aplicação Speedup (Fator de Aceleração)

- Indica quantas vezes o programa paralelo é mais rápido que a versão seqüencial para executar uma dada tarefa
- É calculado pela razão entre o melhor tempo seqüencial e o melhor tempo da versão paralela

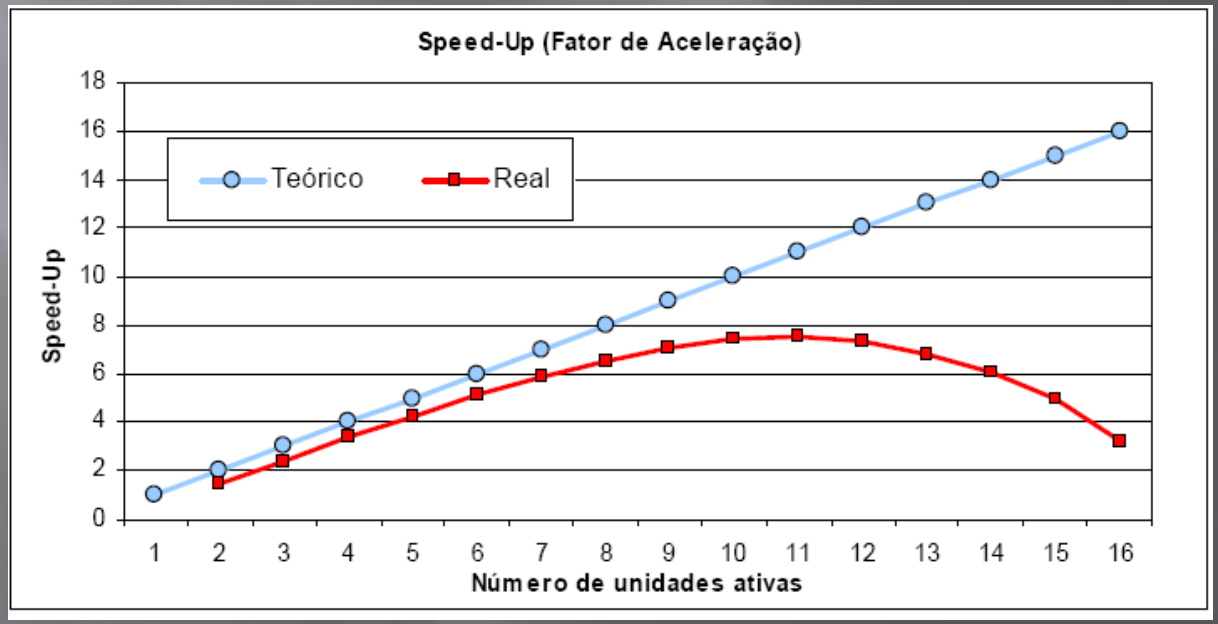
SpeedUp(p) ou $SU(p) = TS / TP(p)$

  - Onde:
    - *TS é o tempo de execução da aplicação na versão seqüencial*
    - *TP é o tempo de execução na versão paralela*
    - *p* é o número de unidades ativas (processadores) utilizadas

- Se $SU > 1$ a versão paralela reduziu o tempo de execução (ficou mais rápido que a seqüencial)
- Se $SU < 1$ a versão paralela aumentou o tempo de execução (ficou mais lenta que a seqüencial)

Desempenho da Aplicação Speedup

- Cada aplicação tem sua curva que depende do trabalho e da incidência de complicadores
- Todo o algoritmo de uma aplicação tem um número de unidades ativas ideal para a obtenção do melhor desempenho em uma dada arquitetura alvo
  - Não sendo verdade que quanto mais unidades ativas melhor



Desempenho da Aplicação Speedup

- Arquiteturas, onde tarefas podem ser distribuídas de forma eqüitativa entre processadores, TP(p) pode ser aproximado por:
- $TP(p) \cong TS / p + T_{CONTROLE} + T_{COMUNICAÇÃO}$
  - Onde:
    - TS / p é o tempo de execução seqüencial dividido nos p processadores
    - $T_{CONTROLE}$ é o tempo gasto para controlar a operação com as máquinas paralelas
    - $T_{COMUNICAÇÃO}$ é o tempo gasto para troca de dados e controle entre as máquinas paralelas
- As parcelas $T_{CONTROLE}$ e $T_{COMUNICAÇÃO}$ é que geram a redução do speed-up a partir de um certo número de processadores
  - Estas parcelas são fortemente dependentes da arquitetura alvo
- Avaliar o Speed up considerando que o tempo de execução é composto pelos tempos de processamento, comunicação e armazenamento, muitas vezes requer modelos muito complexos, que dependem fortemente da arquitetura alvo

Desempenho da Aplicação Eficiência

- Indica a taxa de utilização média das unidades ativas
- Mostra se os recursos foram bem aproveitados
- É calculado pela razão entre o *Speed-Up* e o número de *unidades ativas* utilizadas

- Eficiência(p) ou $E(p) = SU(p) / p$

- Normalmente, as unidades ativas ficam parte de seu tempo esperando por resultados de vizinhos
  - Reduz sua taxa de utilização e conseqüentemente a eficiência

Desempenho da Aplicação Eficiência

- Eficiência ideal
  - Cada unidade ativa com 100% do tempo ativa (linha azul)
- A melhor taxa de utilização média não significa o menor tempo de execução
  - Exemplo: o menor tempo de execução ocorreu com 11 unidades ativas e a melhor taxa de utilização média com 5 unidades ativas