

## Capítulo 5

### Paralelismo em nível de threads (TLP)

Adaptado por  
**Marcos Barreto**  
DCC/UFBA  
(2015.2)

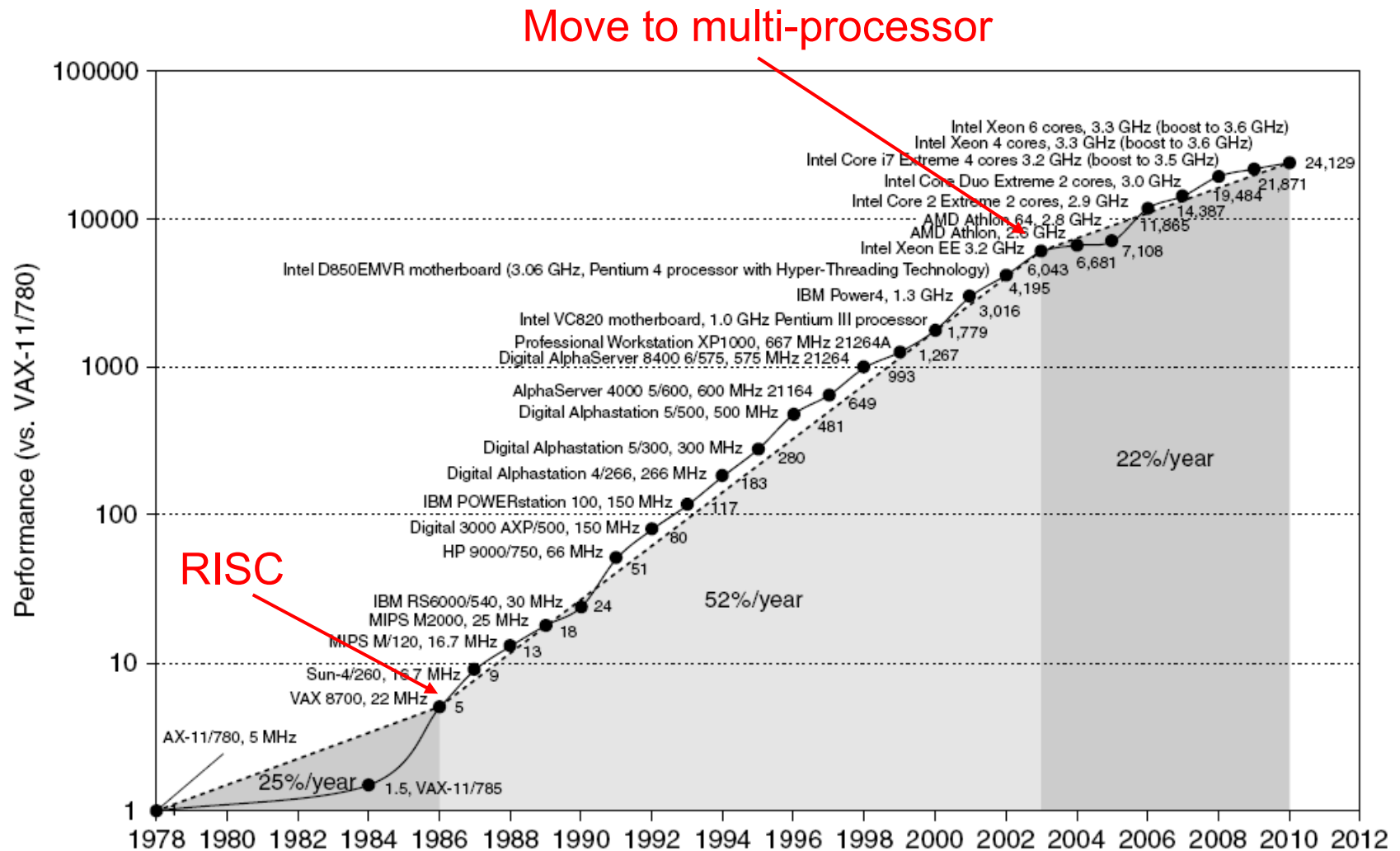
# Roteiro

---

- Contextualização
- Categorias de multiprocessadores
- Questões relativas ao desempenho

# Contextualização – evolução do desempenho

- 1986 – 2003: aumento no desempenho de uniprocessadores.



# Contextualização – questões técnicas

- Durante a década de 90, a exploração de paralelismo em nível de instrução (ILP) tornou-se menos eficiente.
- Grande preocupação com questões energéticas.
  - => impulsionou o estabelecimento de multiprocessadores.
  - => foco em paralelismo em nível de *thread* (TLP)
  - => interesse crescente em servidores *high-end* (*cloud computing* e *software-as-a-service*) e aplicações intensivas de dados.
  - => aumento de desempenho em *desktops* não é tão importante  
→ requisitos dos usuários são mais simples.
  - => opção por projetos centrados em replicação de recursos (ex. multiprocessadores) do que projeto único/diferenciado para um novo uniprocessador.

# Contextualização – definições

- TLP pressupõe a existência de múltiplos contadores de programa e são explorados primariamente em máquinas MIMD.
- **Multiprocessador** = computadores com múltiplos **processadores fortemente acoplados**, controlados por um único sistema operacional e com acesso a uma **memória global (compartilhada)**.
  - => tipicamente, 2 a 32 processadores (tamanho moderado) ou >33 (projetos maiores).
  - => compartilhamento de memória pode ser:
    - centralizado: sistemas *single chip*, *multiple cores*.
    - distribuído: sistemas *multiple chips*

# Contextualização – definições

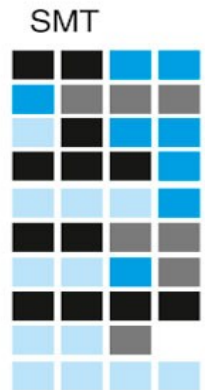
- Como o TLP é explorado?

=> processamento paralelo: conjunto de threads fortemente acopladas colaborando para uma única aplicação.

=> request-level parallelism: múltiplos processos independentes oriundos de:

- uma única aplicação executando em múltiplos processadores (ex. servidor de banco de dados atendendo requisições), ou
- múltiplas aplicações isoladas (multiprogramação).

- Multiprocessadores também exploram *multithreading* simultâneo (SMT) dentro cada processador.



# Contextualização – abordagens

- Para explorar uma máquina MIMD com  $n$  processadores são necessários  $n$  *threads* ou processos.
- Como os threads são criados?
  - => pelo usuário, através de uma biblioteca de programação.
  - => pelo sistema operacional, para atender às requisições dos usuários e programas.
  - => pelo compilador, para explorar ILP e também paralelismo de laço → iterações paralelas em um laço.
- ILP x TLP
  - => TLP é identificado em alto nível pelo software ou pelo programador, gerando dezenas de *threads* com centenas de instruções a serem executadas em paralelo.

# Categorias de multiprocessadores

- Multiprocessadores são classificados de acordo com a quantidade de processadores que empregam, o que reflete na organização de memória.
- Duas classes:
  - => multiprocessadores com memória compartilhada centralizada ou multiprocessadores simétricos (SMP – *symmetric multiprocessors*)
  - => multiprocessadores com memória compartilhada distribuída (DSM – *distributed shared memory*).

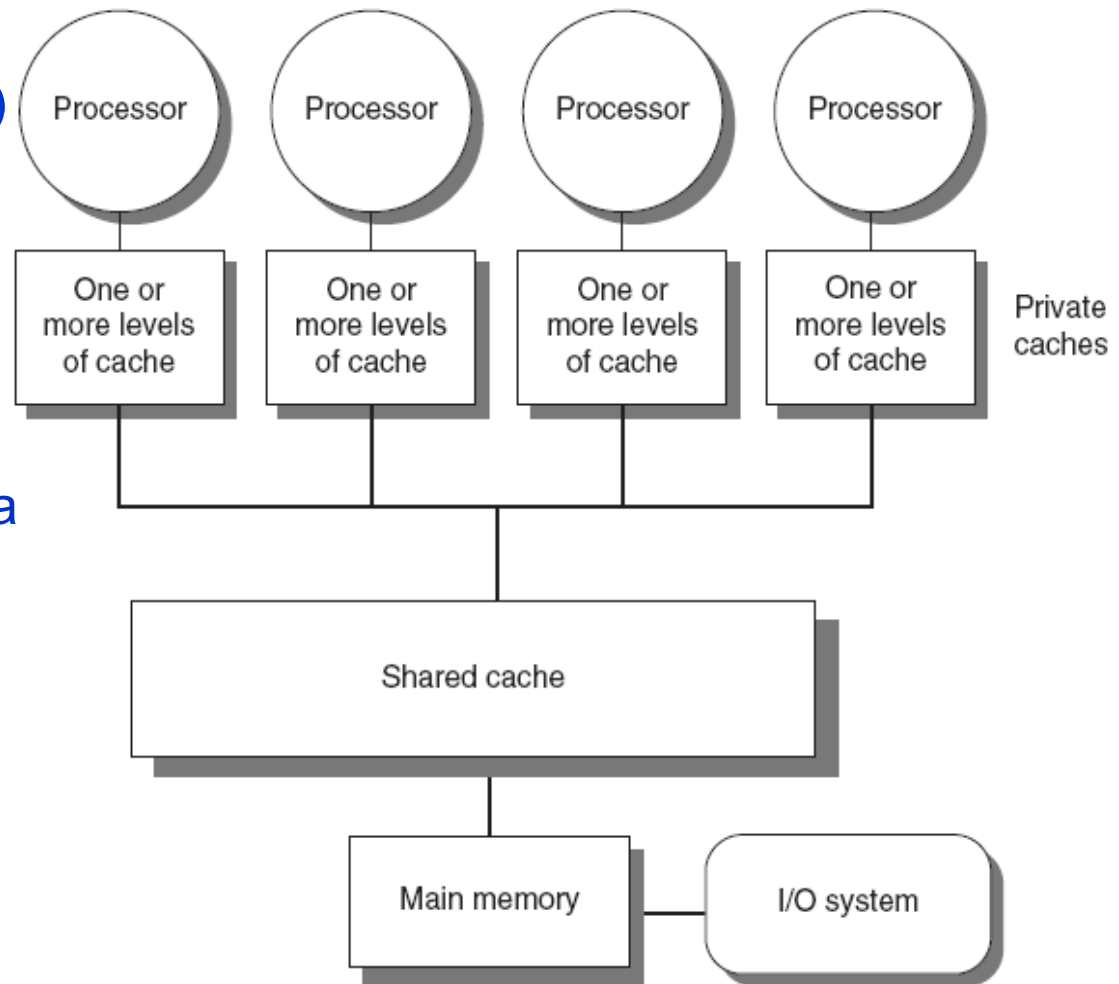
## **Memória compartilhada**

- os threads têm acesso a um único espaço de endereçamento.
- qualquer processador pode acessar qualquer endereço de memória (tanto para SMP quanto para DSM)



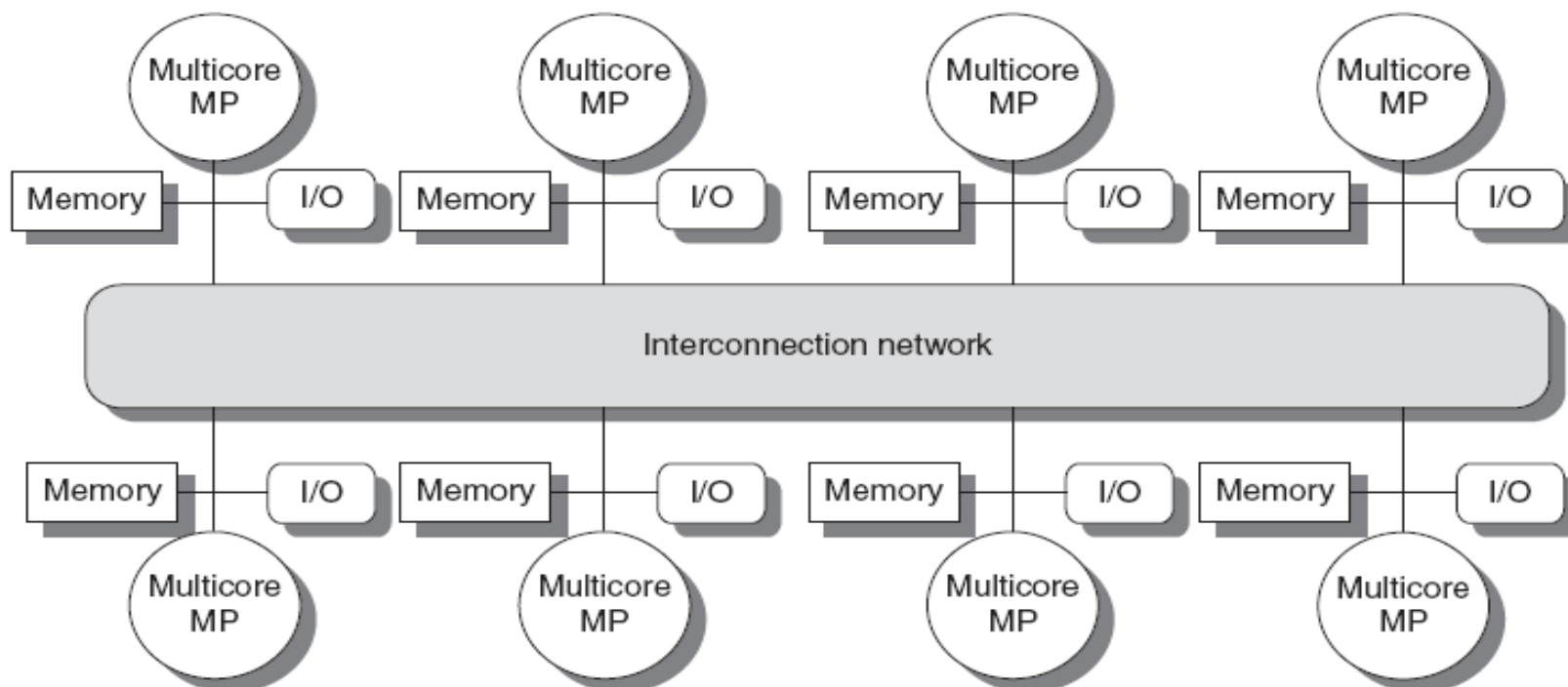
# Categorias de multiprocessadores

- Multiprocessadores simétricos (SMP)
  - Pequeno número de núcleos (geralmente 8)
  - Arquitetura UMA (*uniform memory access*)
  - Todos os processadores acessam a memória com a mesma latência
  - Principal questão: **coerência de cache** para dados compartilhados e privados



# Categorias de multiprocessadores

- Memória compartilhada distribuída (DSM)
  - Memória deve ser distribuída para suportar um maior número de processadores.
  - Necessidade de um barramento de interconexão direta (*switch*) ou indireta (malha bidimensional).
  - Arquitetura NUMA (*non-uniform memory access*).
  - *Multicore* com mais de um chip processador são todos DSM!



# Questões de desempenho

---

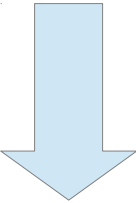
- Dois obstáculos principais:
  - Pouco paralelismo disponível nos programas
  - Alto custo de comunicação entre processadores

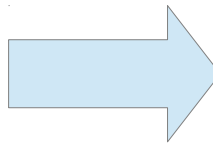
# Questões de desempenho

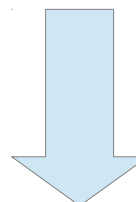
- 1) Pouco paralelismo disponível nos programas
- Ex.: deseja-se *speedup* de 80 com 100 processadores. Qual porcentagem do código pode ser sequencial?

$$\text{Speedup} = \frac{1}{\frac{\text{FraçãoP}}{\text{SpeedupP}} + (1 - \text{FraçãoP})}$$

SpeedupP = quantidade de processadores  
(assumindo que todos os processadores  
são usados)


$$80 = \frac{1}{\frac{\text{FraçãoP}}{100} + (1 - \text{FraçãoP})}$$


$$\begin{aligned} 0,8 \times \text{FraçãoP} + 80 \times (1 - \text{FraçãoP}) &= 1 \\ 80 - 79,2 \times \text{FraçãoP} &= 1 \\ \text{FraçãoP} &= (80 - 1) / 79.2 \\ \text{FraçãoP} &= 0,9975 \end{aligned}$$



Somente 25% do código pode ser sequencial para que se tenha um *speedup* de 80 com 100 processadores

# Questões de desempenho

- 2) Alto custo de comunicação
- Acesso remoto à memória entre processadores paralelos.
- 35 a 50 ciclos de relógio entre núcleos do mesmo chip.
- 100 a >500 ciclos de relógio entre núcleos de chips diferentes.

Ex.: aplicação executando em 32 processadores, com 200 ns para acesso à memória remota. Outros requisitos:

- frequência do relógio = 3.3 Ghz
- maioria das referências encontra os dados na cache local
- processadores bloqueiam durante requisições remotas
- CPI efetivo é 0,5 ns

=> quão rápido é o multiprocessador se:

- 1) não houver acesso à memória remota?
- 2) se 0,2% dos acessos forem à memória remota?

# Questões de desempenho

## ■ 2) Alto custo de comunicação

### 1º) Cálculo do CPI com 0,2% de referências remotas

CPI = CPI base + taxa de requisições remotas X custo de requisições remotas

CPI = 0,5 + 0,2% X custo de requisições remotas

$$\text{Custo de requisições remotas} = \frac{\text{Custo de acesso remoto}}{\text{Tempo de ciclo}} = \frac{200 \text{ ns}}{0,3 \text{ ns}} = 666 \text{ ciclos}$$

$$\text{Cálculo do CPI} = 0,5 + 0,2\% \times 666 \Rightarrow 0,5 + 1,2 \Rightarrow 1,7$$

### 2º) Se todas as referências forem locais:

Um microprocessador com todas as referências locais é  $1,7 / 0,5 = 3,4$  mais rápido.