Master Informatics Eng.

2021/22

A.J.Proença

Background to this course
(*slides from several sources*)

The course website:
http://gec.di.uminho.pt/mei/cp/

Computação Paralela

Mestrado Eng.ª Informática
2021/2022
Docente responsável: A. Proença

Ultima Modificação: 30 Set 2021

departamento de informática

Equipa docente

- **Docentes responsáveis pela lecionação da UC**

Teóricas
Alberto José Proença
email: aproenca<at>di.uminho.pt

Práticas Laboratoriais
André Martins Pereira
email: ampereira<at>di.uminho.pt
Rui Silva
email: ruisilva<at>di.uminho.pt

Notas complementares:
- UC para MIEI, MEI & LEF
- UC de JLS, excecionalmente com AJP este ano
- *background* esperado: Lic Engª Informática
- 1ªs semanas para revisão de conceitos básicos
- língua de comunicação escrita: inglês
- data última modificação do *website*: confiável
- convém ler sempre os avisos
- sumários e slides: atualizados semanalmente
- feriados em dezembro…
- turnos práticos: possível usar horário de PLs em LEF

Courses timetable

	2-Segunda	3-Terça	4-Quarta	5-Quinta		6-Sexta	
08:30							08:30
09:00			Computação Paralela MEI	Computação Paralela MEI	Computação Paralela MEI		09:00
09:30			PL4	PL1	PL3		09:30
10:00			Rui Silva	André Pereira	Rui Silva		10:00
10:30			DI 1.10	DI 0.09	DI 0.11		10:30
11:00	Programação Paralela LEF		Computação Paralela MEI			Programação Paralela LEF	11:00
11:30	T1		PL5			PL1	11:30
12:00	Alberto Proença		Rui Silva			André Pereira	12:00
12:30	CP2 1.11		DI 1.10			DI 0.04	12:30
13:00	ALMOÇO			Computação Paralela MEI			13:00
13:30				T AJP			13:30
14:00		Programação Paralela LEF		CP1 0.08			14:00
14:30		PL2		Computação Paralela MEI	Computação Paralela MEI		14:30
15:00		André Pereira		PL2	PL6		15:00
15:30		DI 0.04		André Pereira	Rui Silva		15:30
16:00				DI 0.09	DI 0.11		16:00

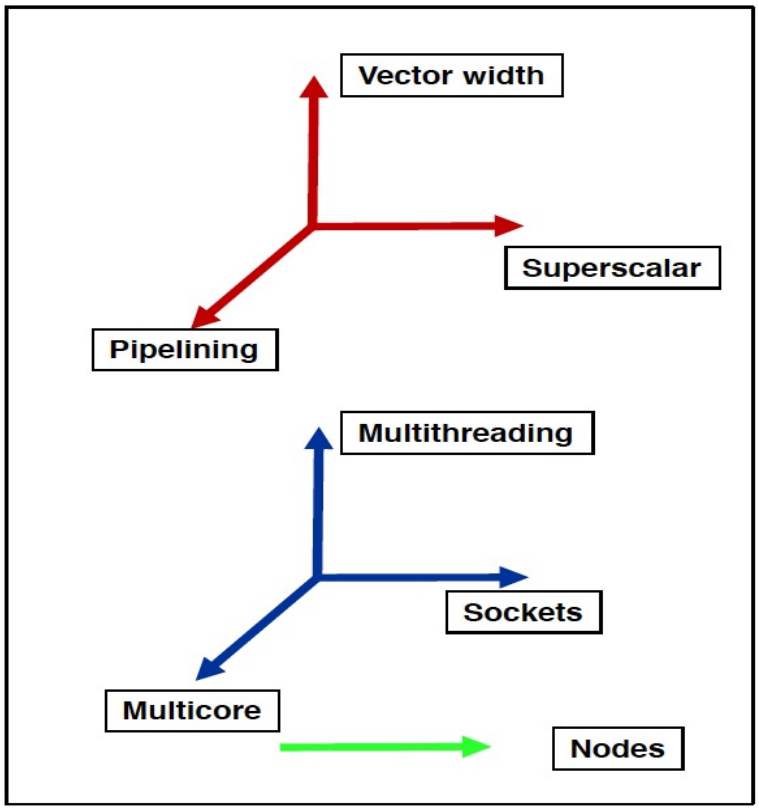
Focus on this course: performance engineering

How:

- understanding the organization of computer system (its architecture) to develop efficient algorithms and program & data structures
- profiling & measuring the execution efficiency

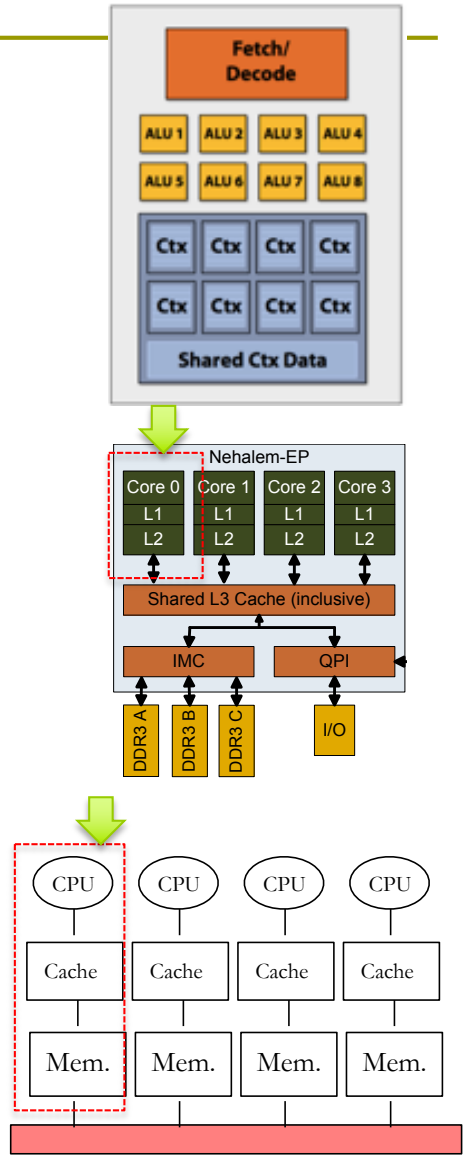
Where in the hardware:

- in a memory hierarchy
  - multi-level caches
- in sequential code with ILP
  - pipelining
  - superscalar w/ out-of-order exec
  - vector processing
- in code with thread parallelism
  - multithreading in-core (SMT)
  - multithreading in multicore
  - multithreading in multiple devices
- in code with process parallelism
  - multiprocessing in a computer cluster



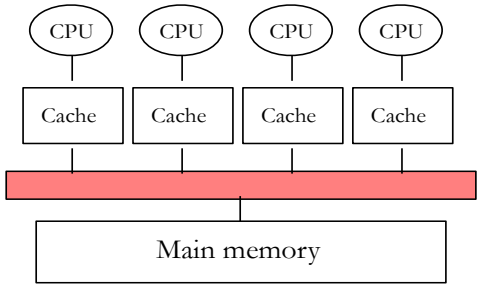
Níveis de paralelismo

- **Instrução (ILP)**
  - Execução de múltiplas instruções de um programa em paralelo
  - Processamento vetorial
  - Limitado pelas dependências de dados/controlo do programa
- **Tarefas / fios de execução**
  - múltiplos fluxos instruções de um mesmo programa executam em paralelo
  - Limitado pelas dependências e características do algoritmo
- **Processos**
  - Múltiplos processos de um mesmo programa / ou vários programas

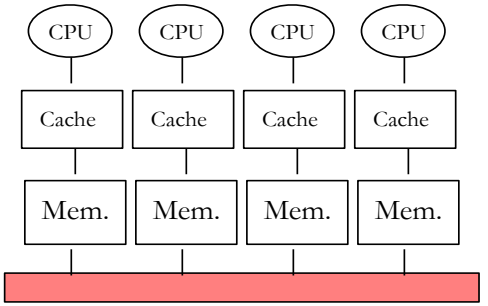


Memória partilhada *vs.* memória distribuída

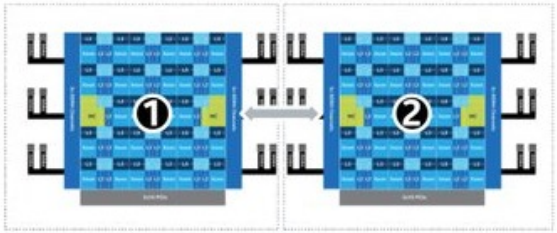
- **Memória partilhada**
  - Processadores partilham um canal de acesso à memória
  - *Caches* reduzem o tráfego e a latência dos acessos à memória
  - Largura de banda de acesso à memória partilhada por várias UPs => limitação à escalabilidade deste tipo de arquitetura

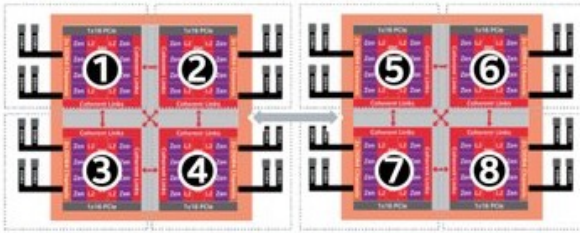


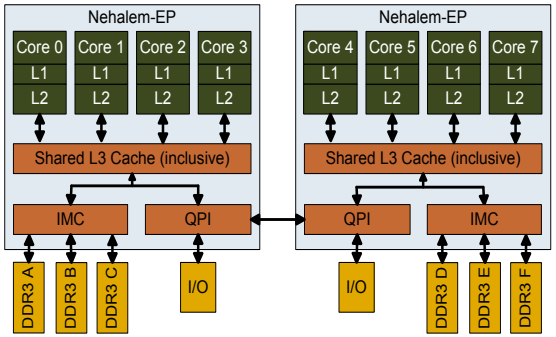
- **Memória distribuída**
  - Cada processador contém a sua própria memória, existindo uma rede de interligação entre os processadores



- **Sistemas híbridos**
  - Acesso não uniforme à memória (NUMA)







Intel® Xeon® Scalable Processor Family
Architected for the Data Center
- **2 socket servers** have 2 CPUs – 2 NUMA domains
- Consistent performance, consistent latency

AMD Naples
Repurposed Desktop Die for Server
- **2 socket server** has **8 different desktop dies**
- 8 different NUMA domains
- Difficult to get optimal application performance

Computação Paralela

Os alunos deverão ser capazes de projetar e **otimizar** aplicações que executem de forma **eficiente** nas arquiteturas (paralelas) atuais:

- Escrever códigos sequenciais eficientes
- Implementar e otimizar códigos paralelos em sistemas de memória partilhada
- Desenhar e implementar códigos para sistemas de memória distribuída

Programa (versão resumida)

- **Desempenho "single core"**
  - Medição de desempenho e métricas
  - Hierarquia de memória
  - Otimizações para paralelismo ao nível da instrução
- **Programação de sistemas de memória partilhada**
  - Modelos baseados em fios de execução
  - Medição e otimização de desempenho
- **Sistemas de memória distribuída**
  - Modelo de processos comunicantes
  - Desenho de aplicações
- **Algoritmos paralelos**
  - Análise da complexidade e escalabilidade

Background for Advanced Architectures

Key concepts to revise:

- *numerical data representation (integers & FP)*
- *ISA (Instruction Set Architecture)*
- *how C compilers generate code (a look into assembly code)*
  - *how scalar and structured data are allocated*
  - *how control structures are implemented*
  - *how to call/return from function/procedures*
  - *what architecture features impact performance*
- *improvements to enhance performance in a single PU*
  - *ILP: pipeline, multiple issue, …*
  - *data parallelism: SIMD/vector processing, ...*
  - *thread-level parallelism*
  - *memory hierarchy: cache levels, ...*

Keyword: **paralellism**

Key textbook (1)

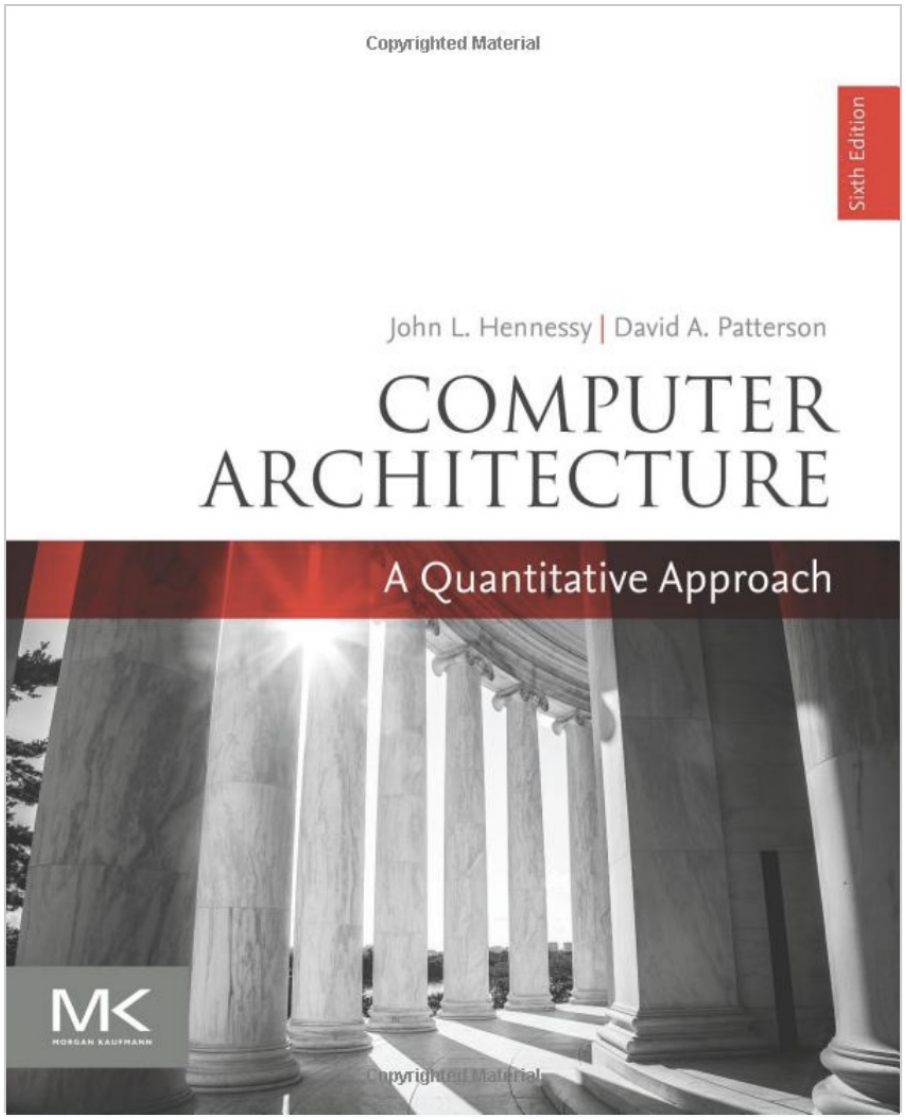


Table of Contents

Printed Text

1. Fundamentals of Quantitative Design and Analysis
2. Memory Hierarchy Design
3. Instruction-Level Parallelism and Its Exploitation
4. Data-Level Parallelism in Vector, SIMD, and GPU Architectures
5. Multiprocessors and Thread-Level Parallelism
6. The Warehouse-Scale Computer
7. Domain Specific Architectures

A. Instruction Set Principles
B. Review of Memory Hierarchy
C. Pipelining: Basic and Intermediate Concepts

Online

D. Storage Systems
E. Embedded Systems
F. Interconnection Networks
G. Vector Processors
H. Hardware and Software for VLIW and EPIC
I. Large-Scale Multiprocessors and Scientific Applications
J. Computer Arithmetic
K. Survey of Instruction Set Architectures
L. Advanced Concepts on Address Translation
M. Historical Perspectives and References

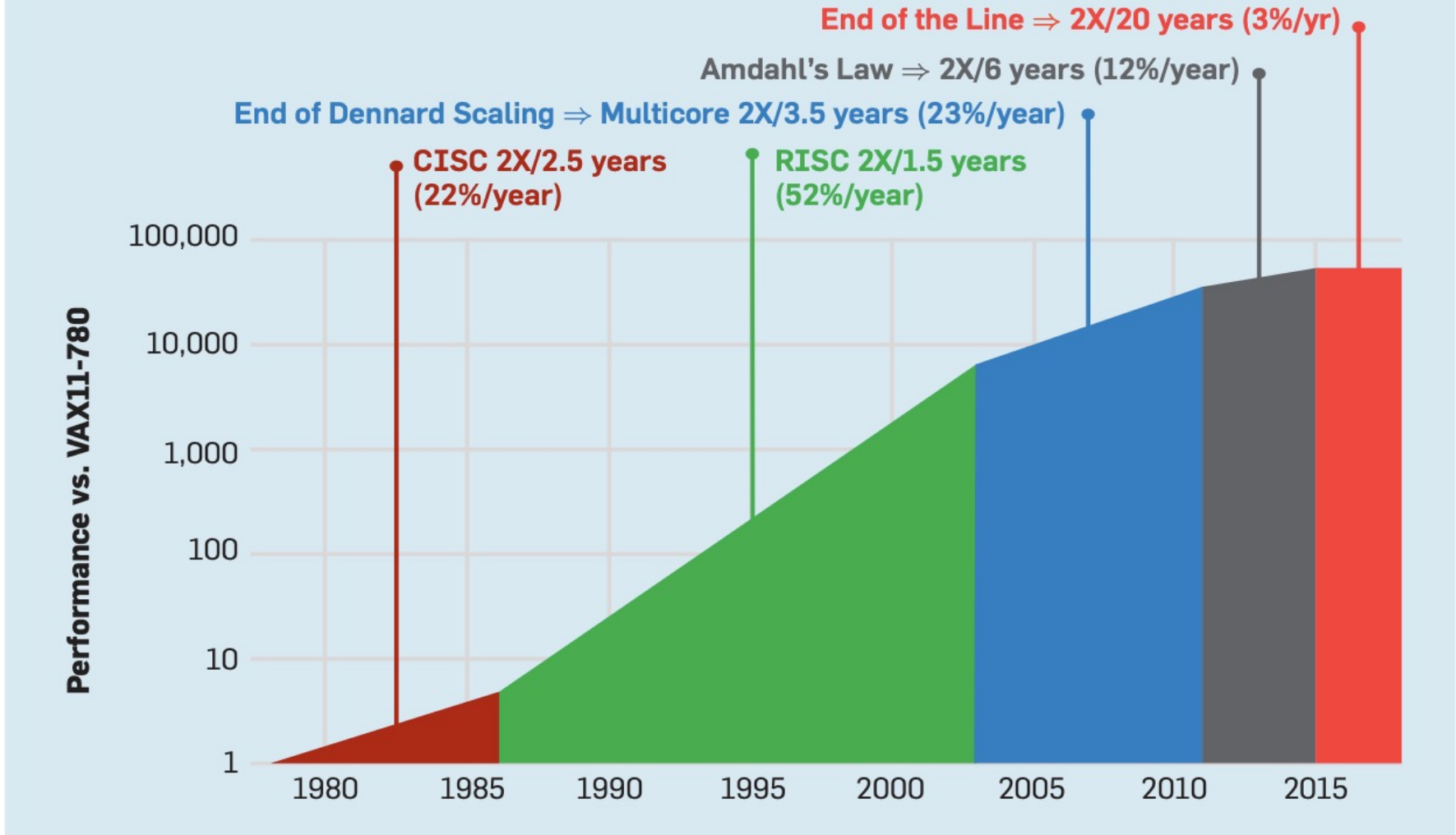
A New Golden Age for Computer Architecture

Growth of computer performance

BY JOHN L. HENNESSY AND DAVID A. PATTERSON

COMMUNICATIONS OF THE ACM | FEBRUARY 2019 | VOL. 62 | NO. 2

Figure 6. Growth of computer performance using integer programs (SPECintCPU).



Key textbook (2)

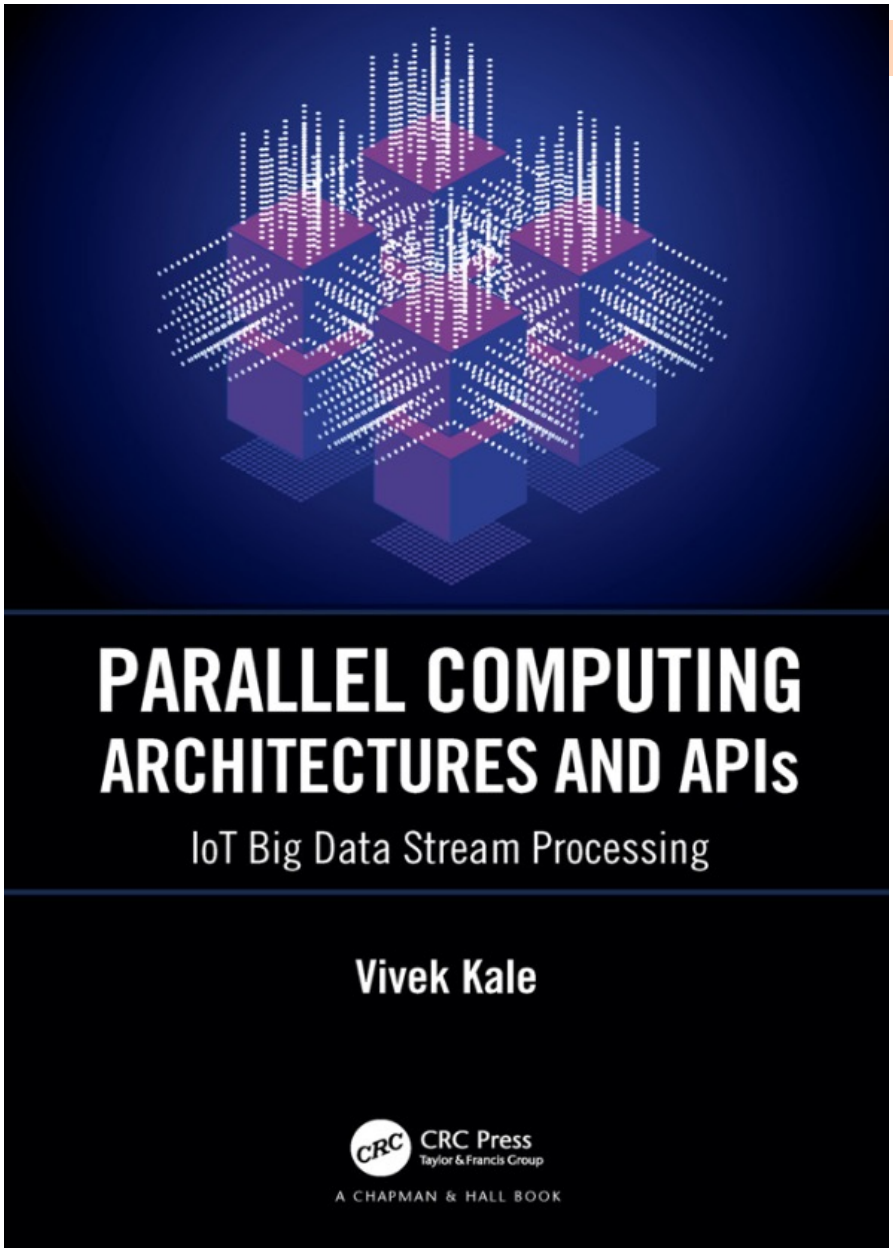


TABLE OF CONTENTS

Key textbook (3)

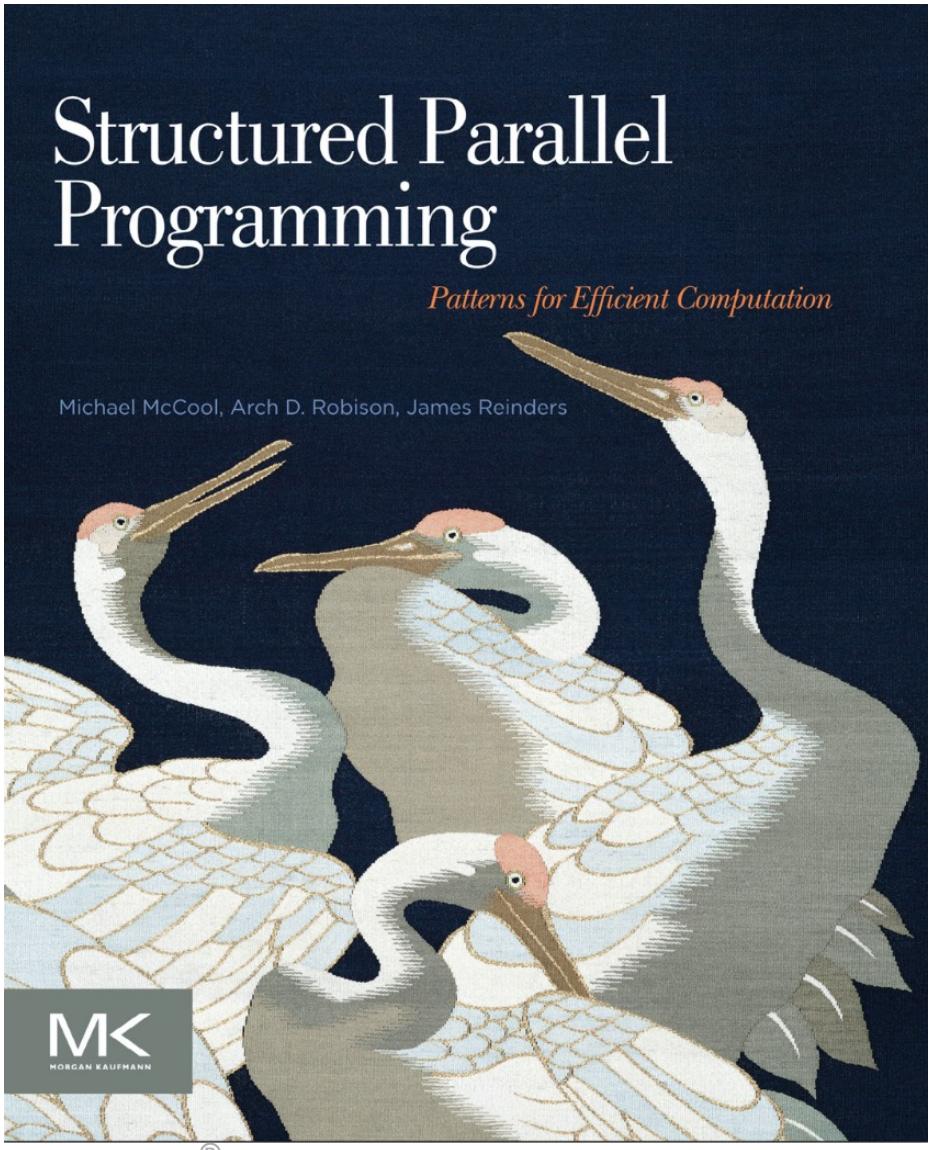
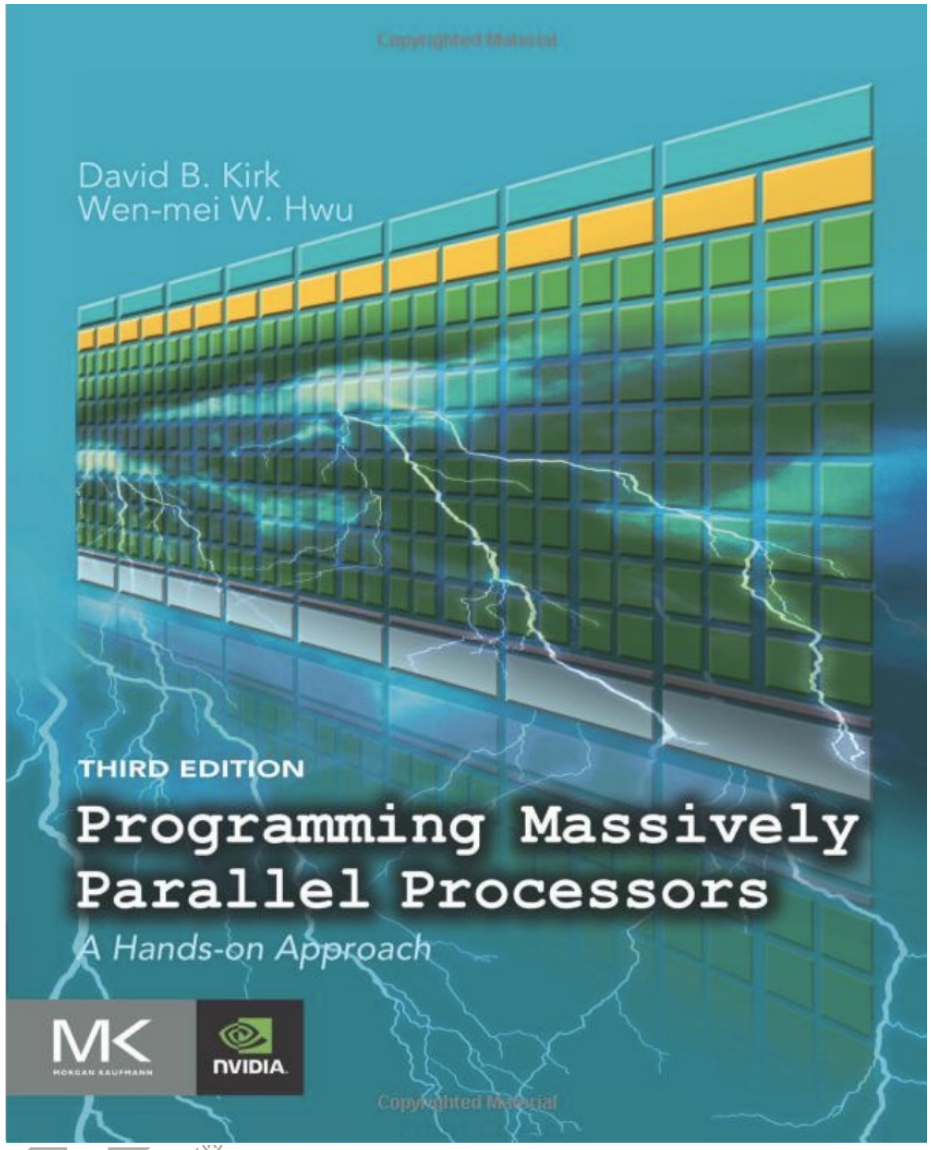


Table of contents

MK®

Recommended textbook (1)



Contents

Recommended textbook (2)

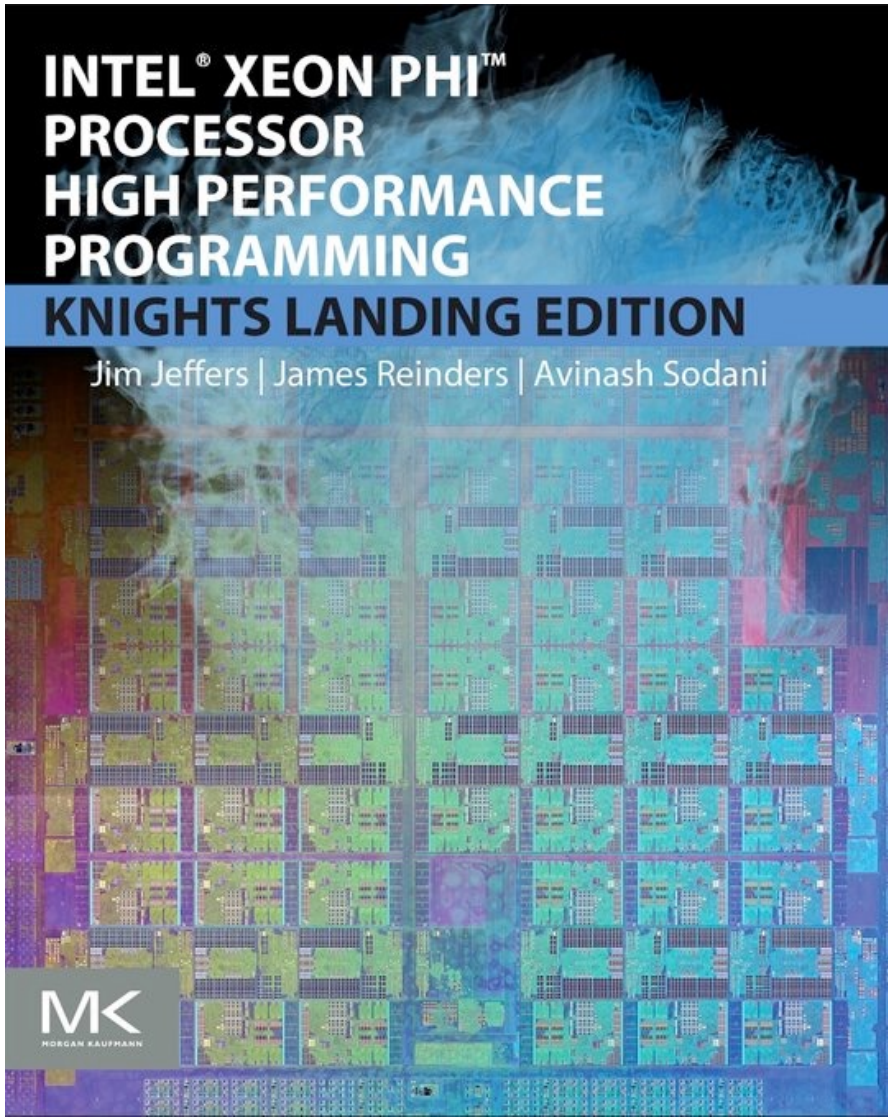


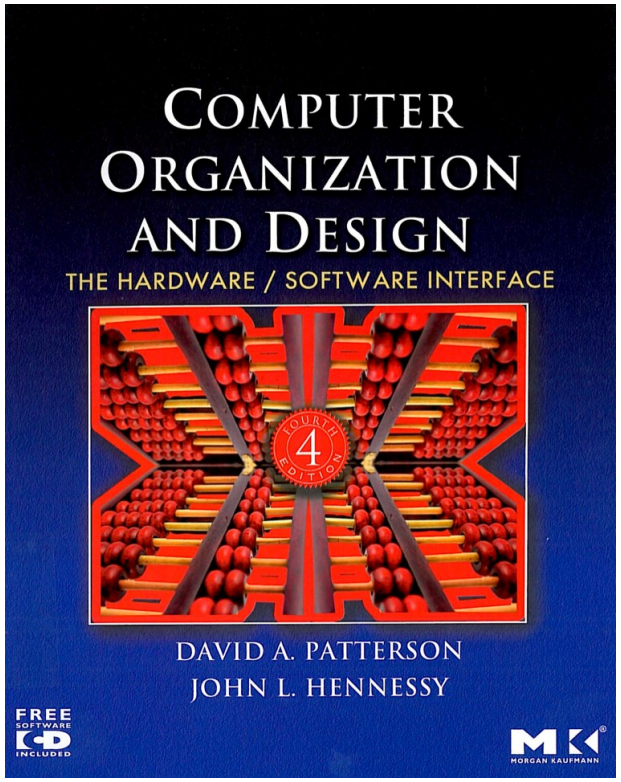
Table of Contents

MK®

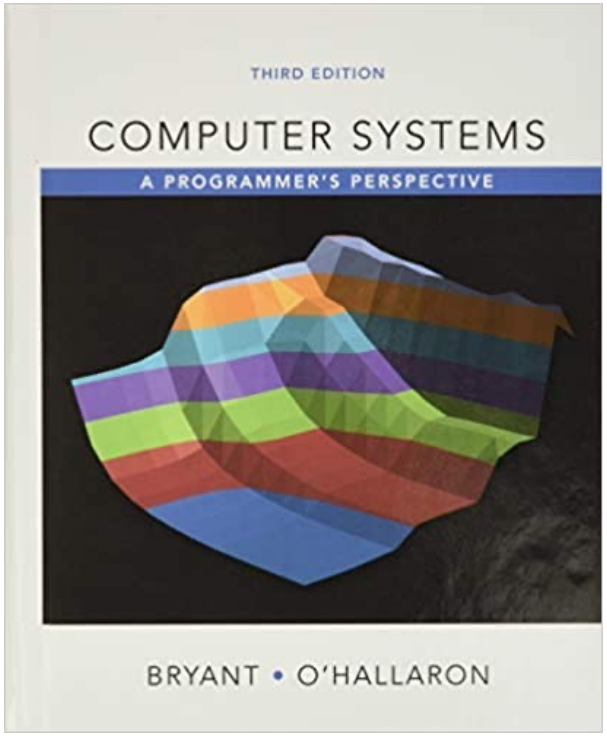
Background concepts from a basic Computer Systems course

Some notes/comments for this course:

– *some slides are borrowed from*



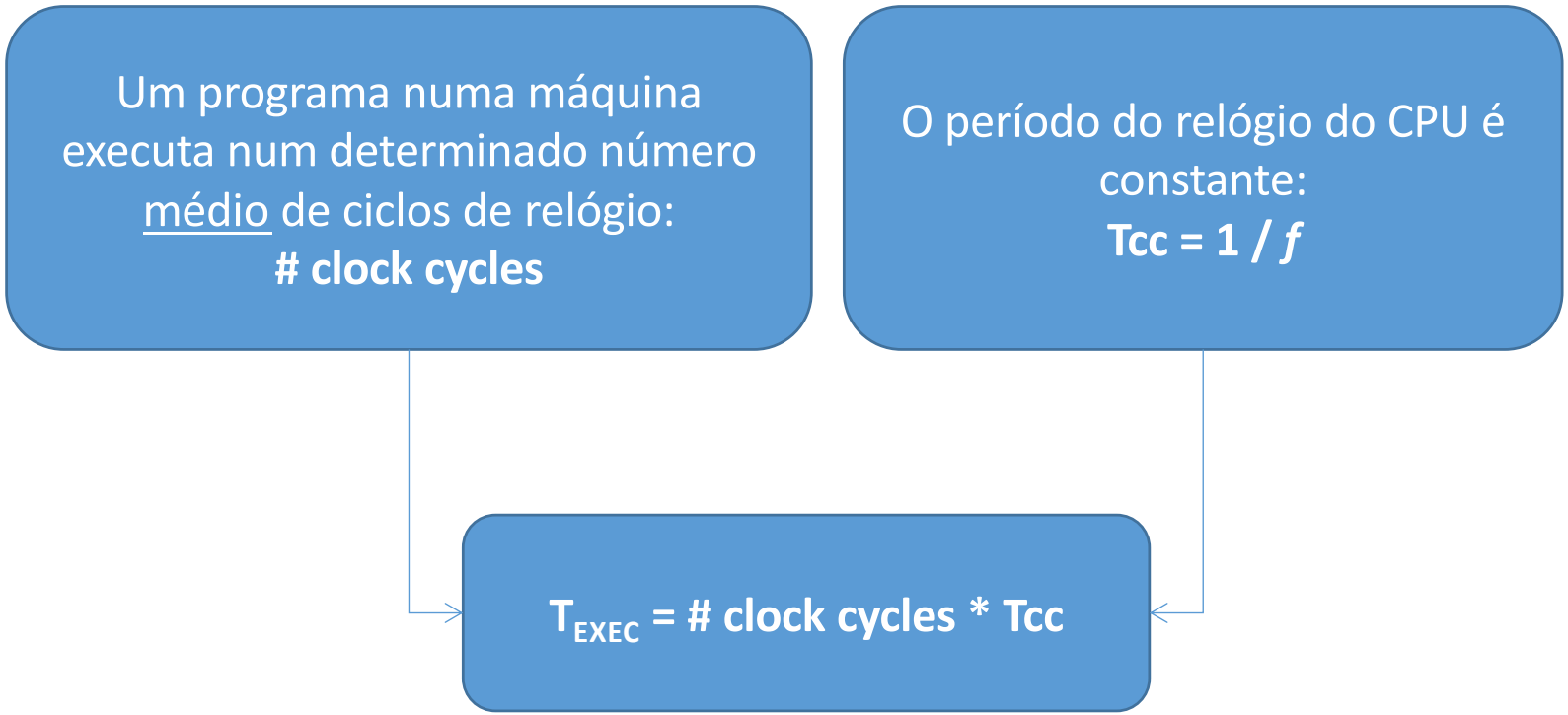
and some from



more details at
http://gec.di.uminho.pt/miei/sc2021/

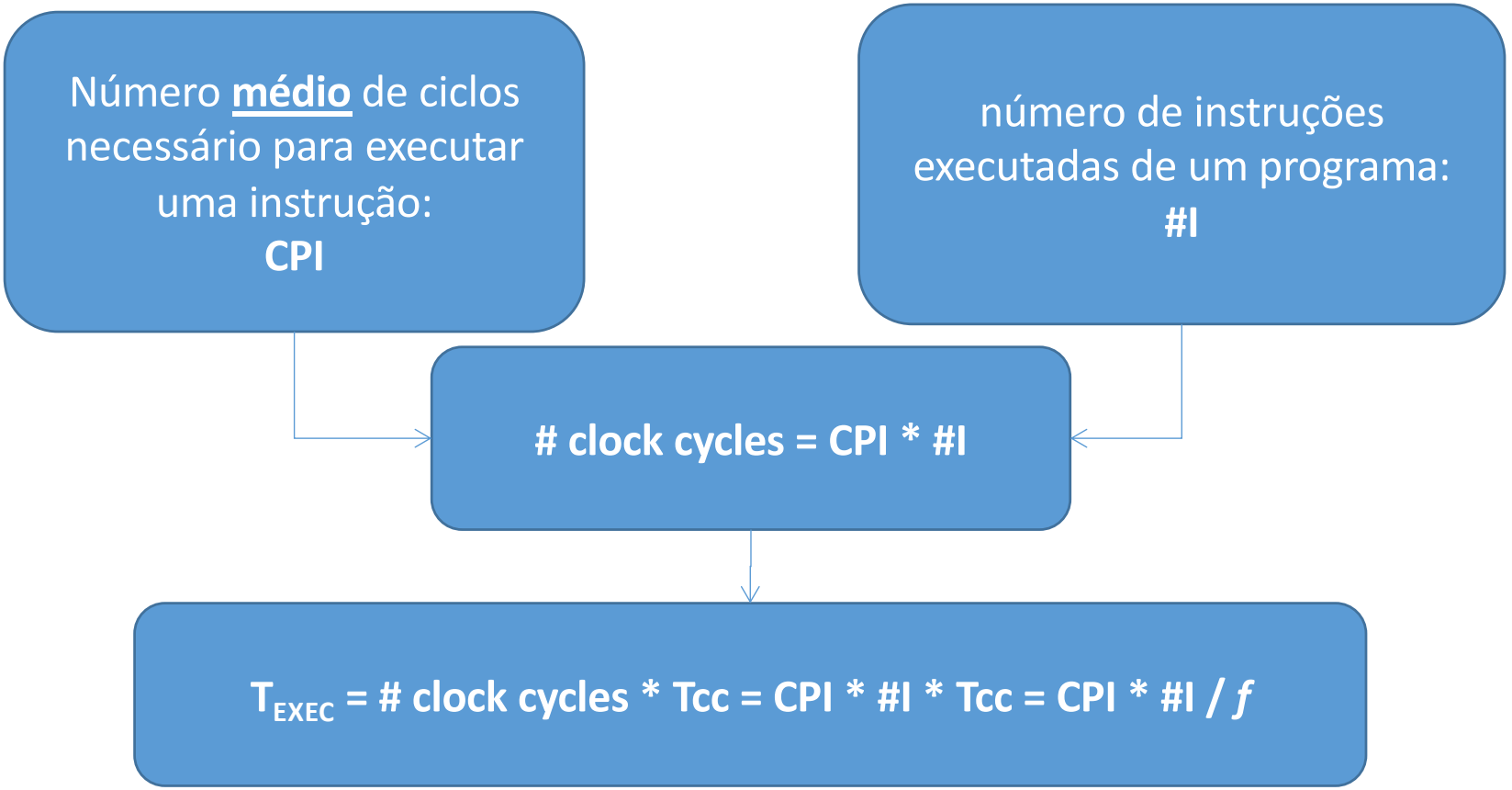
Desempenho do CPU

Previsão do tempo de execução (T_{EXEC}) de um programa numa máquina - requer um **modelo** que relacione o desempenho com as características do sistema de computação (*hw+sw*)



Desempenho do CPU

- De que depende o número médio de ciclos necessários para executar um programa?



The BIG Picture

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Paralellism helps to significantly reduce CPI

Desempenho do CPU

- Calcule o tempo de execução do programa abaixo numa máquina com um relógio de 2 GHz e CPI=1.5

```
  movl 10, %eax
  movl 0, %ecx
ciclo:
  addl %eax, %ecx
  decl %eax
  jnz ciclo
```

#I = 32

NOTA: número de instruções **executadas.**

T_{exec} = 32 * 1.5 / 2 E9 = 24 E-9 s = 24 ns

Relação entre as métricas

$$T_{EXEC} = CPI * \#I / f$$

- #I – depende do algoritmo, do compilador e da arquitectura (ISA)
- CPI – depende da arquitectura (ISA), da mistura de instruções efectivamente utilizadas, da organização do processador e da organização dos restantes componentes do sistema (ex., memória)
- *f* – depende da organização do processador e da tecnologia utilizada

"A única métrica completa e fiável para avaliar o desempenho de um computador é o tempo de execução"

As métricas CPI, *f* e #I não podem ser avaliadas isoladamente, devendo ser sempre consideradas em conjunto, pois dependem umas das outras.

Desempenho do CPU - MIPS

MIPS (milhões de instruções por segundo) – uma métrica enganadora

$$\text{MIPS} = \frac{\#I}{T_{exec} * 10^6}$$

1. MIPS especifica a taxa de execução das instruções, mas não considera o trabalho feito por cada instrução. CPUs com diferentes *instruction sets* não podem ser comparados.
2. MIPS varia entre diferentes programas no mesmo CPU
3. MIPS pode variar inversamente com o desempenho

Esta métrica pode ser usada para comparar o desempenho do mesmo programa em CPUs com o mesmo conjunto de instruções, mas micro-arquitecturas e/ou frequências do relógio diferentes.

Desempenho do CPU- MIPS

MIPS de pico– máxima taxa **possível** de execução de instruções

É a métrica mais enganadora, pois corresponde a sequências de código que apenas tenham instruções com o CPI mais baixo possível.
Este tipo de sequências de instruções não realizam, regra geral, trabalho útil; consistem apenas em operações elementares com operandos em registos.

Pode ser visto como "a velocidade da luz" do CPU, e portanto, inatingível.

O principal problema é que é muitas vezes publicitada pelos fabricantes/vendedores como uma medida de desempenho das suas máquinas!

Background for Advanced Architectures

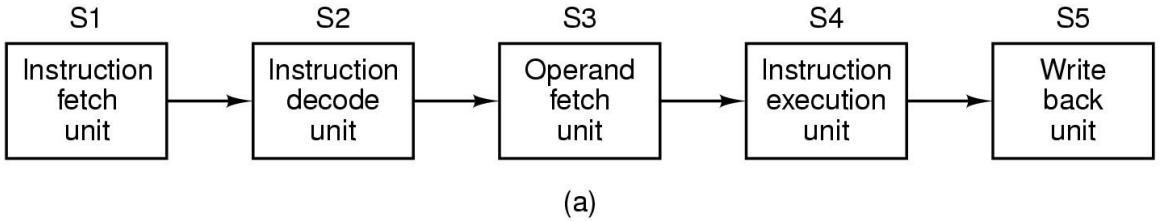
Key concepts to revise:

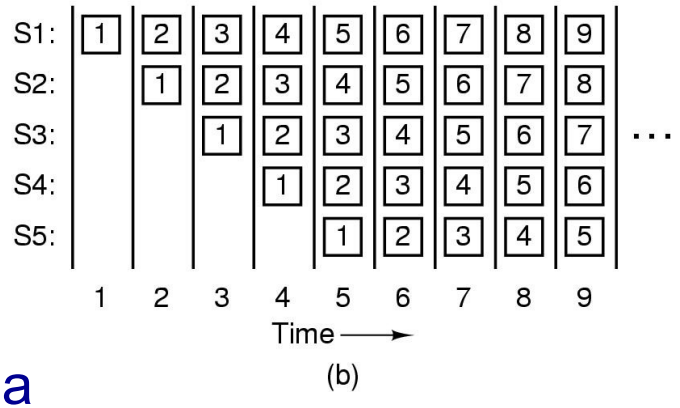
- *numerical data representation (integers & FP)*
- *ISA (Instruction Set Architecture)*
- *how C compilers generate code (a look into assembly code)*
  - *how scalar and structured data are allocated*
  - *how control structures are implemented*
  - *how to call/return from function/procedures*
  - *what architecture features impact performance*
- ***improvements to enhance performance in a single PU***
  - ***ILP: pipeline, multiple issue, …***
  - ***data parallelism: SIMD/vector processing, ...***
  - ***thread-level parallelism***
  - ***memory hierarchy: cache levels, ...***

} Keyword: **paralellism**

Paralelismo no processador
Exemplo 1

Exemplo de *pipeline*





Objetivo

- **CPI = 1**

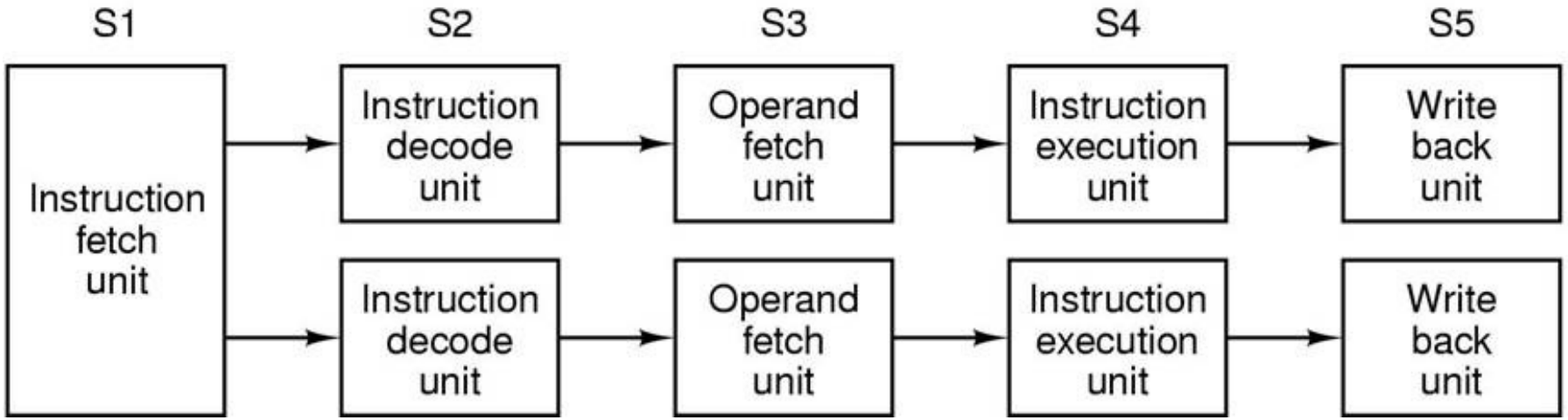
Problemas:

- dependências de dados
- latências nos acessos à memória
- saltos condicionais; propostas de solução para minimizar perdas:
  - executar sempre a instrução "que se segue"
  - usar o historial dos saltos anteriores (1 ou mais bits)
  - executar os 2 percursos alternativos até à tomada de decisão

The BIG Picture

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

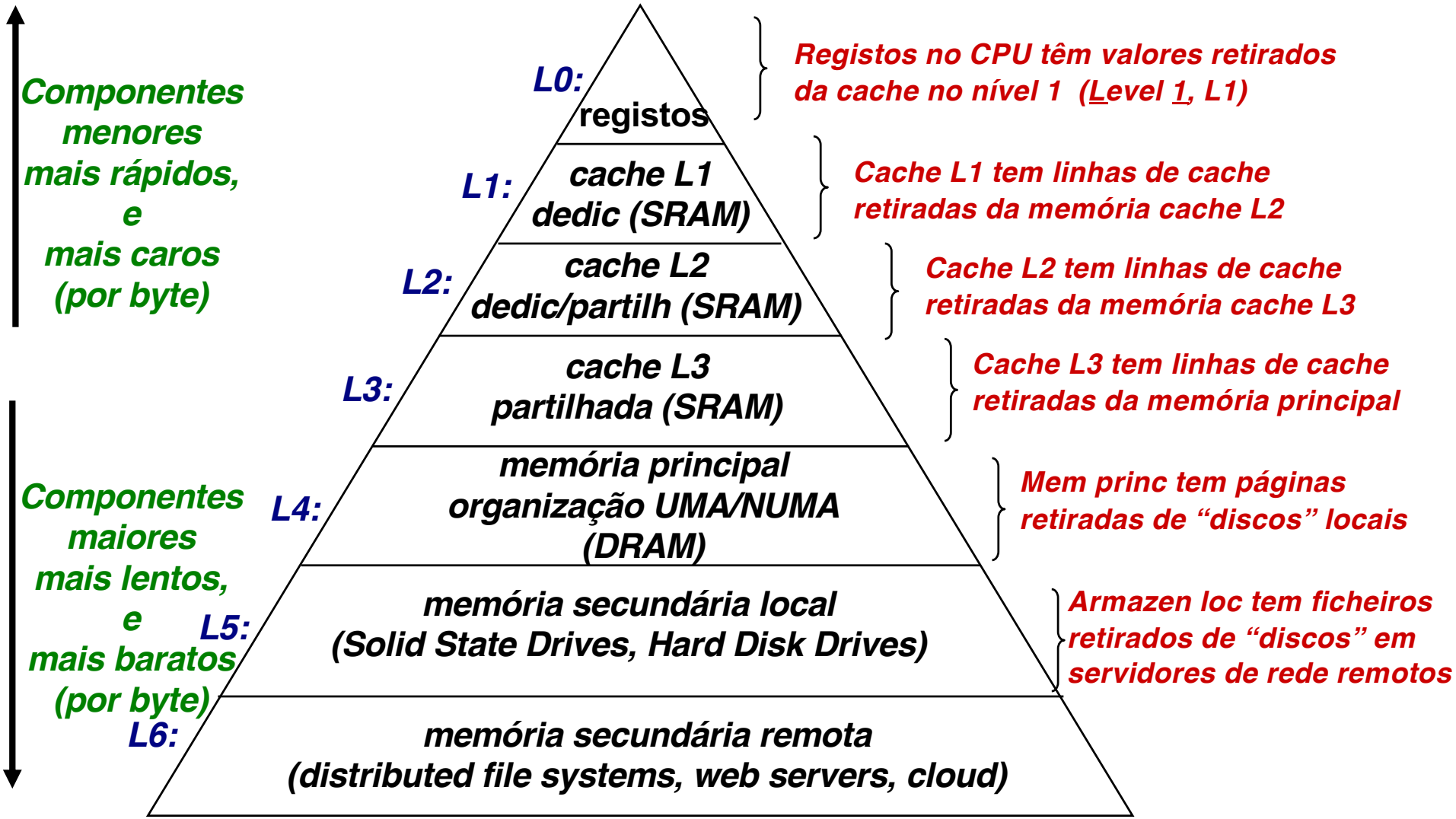
Exemplo de superescalaridade (2 vias)



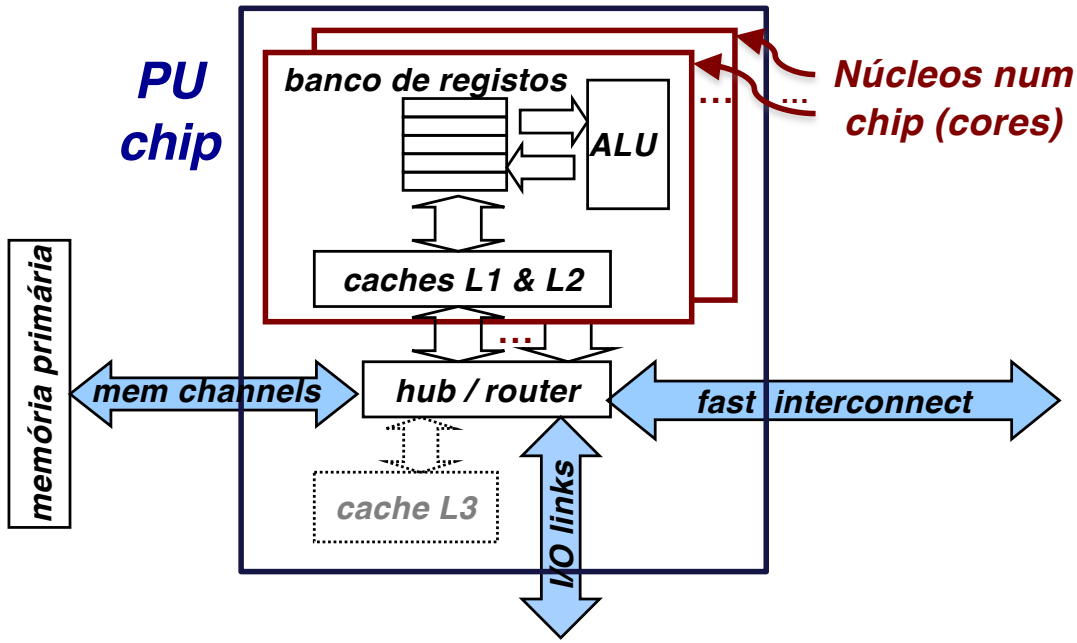
The BIG Picture

- Yes, but not as much as we'd like
- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
  - e.g., pointer aliasing
- Some parallelism is hard to expose
  - Limited window size during instruction issue
- Memory delays and limited bandwidth
  - Hard to keep pipelines full
- Speculation can help if done well

Organização hierárquica da memória



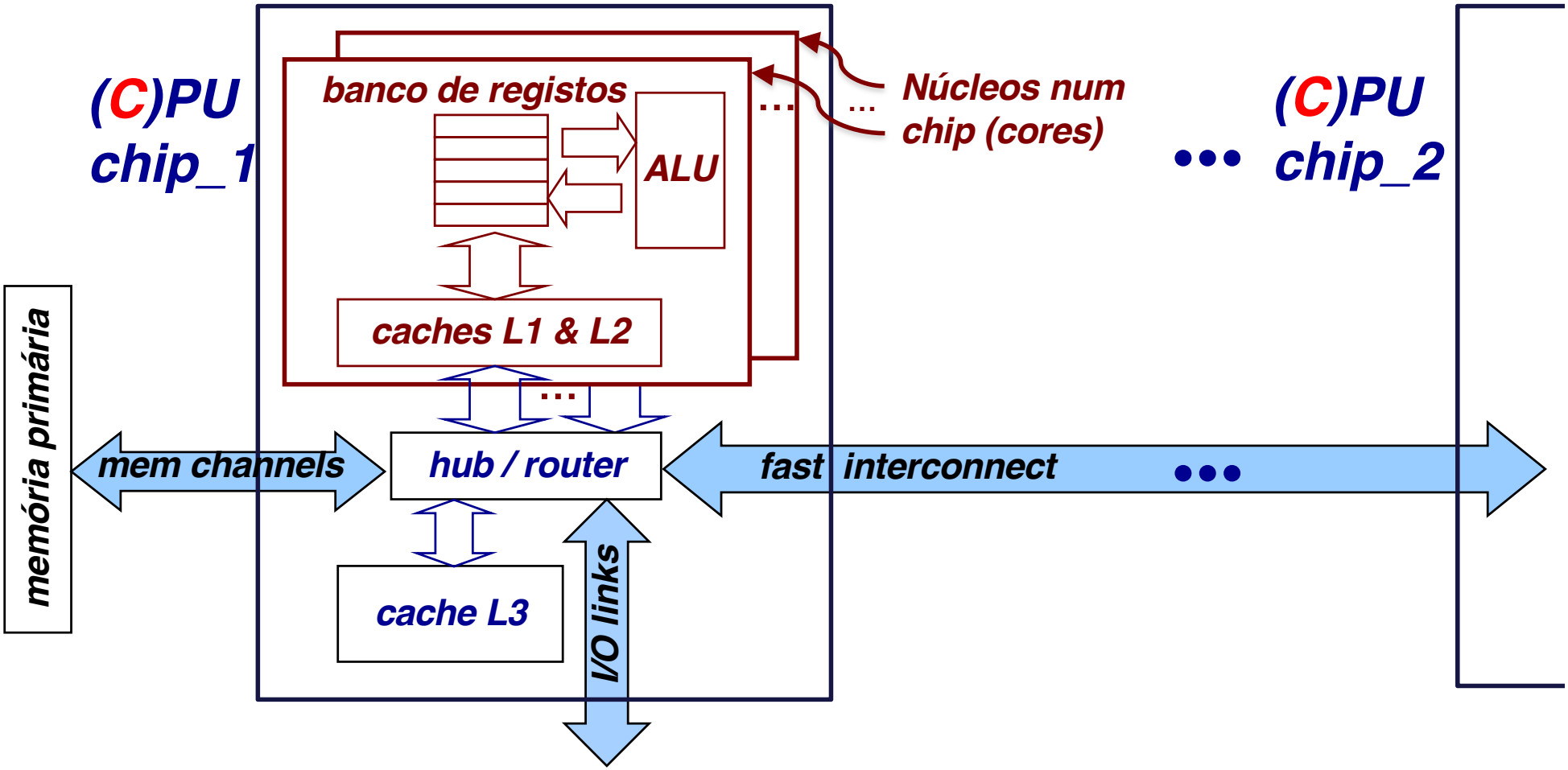
A cache em arquiteturas multicore



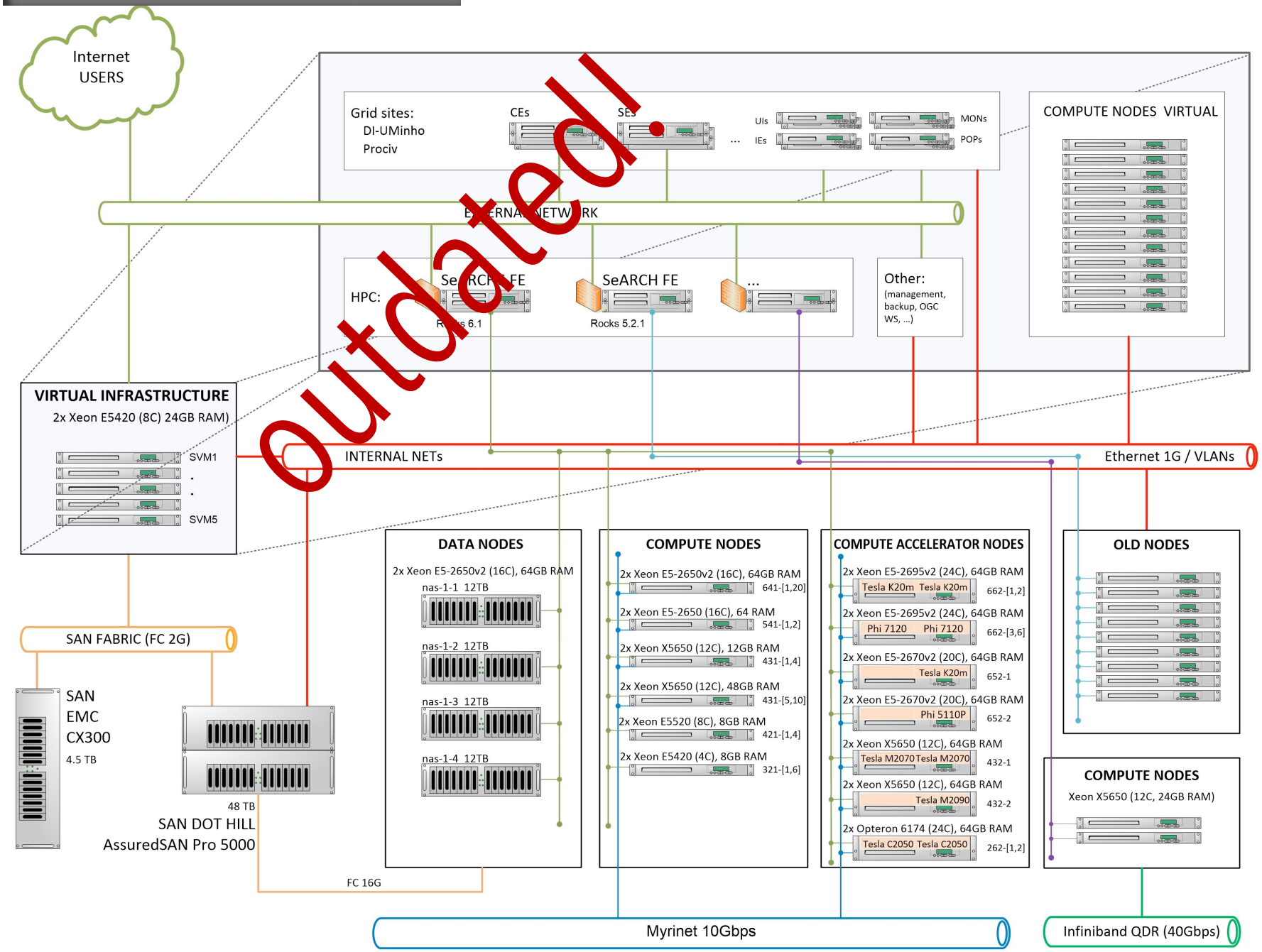
Notas:

- as *caches* L1 de dados e de instruções são normalmente distintas
- as *caches* L2 em *multicores* podem ser partilhadas por outras *cores*
- muitos *cores* partilhando uma única memória traz complexidades:
  - manutenção da coerência da informação nas *caches*
  - encaminhamento e partilha dos circuitos de acesso à memória

Multicore architecture in a server system



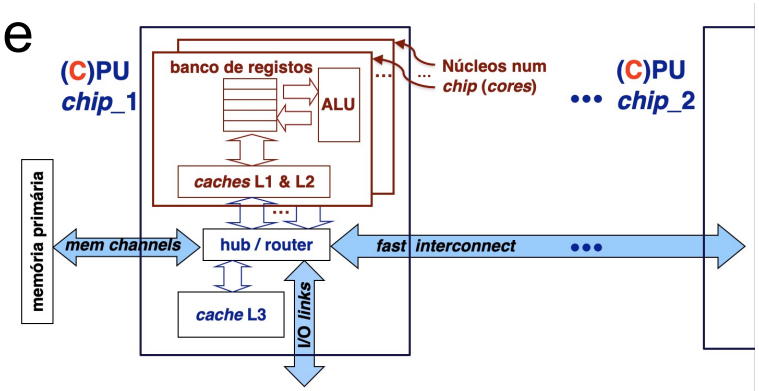
The cluster at DI



http://search6.di.uminho.pt/wordpress/?page_id=274

Questions/homework:

1. Identify the current available devices with the largest #cores; state how many in the device/package & show an image
   a) Designed by Intel
   b) Designed by AMD
   c) Designed by ARM
   d) Designed by a japanese company
   e) Designed by chinese company
   f) Worldwide



2. What are the key chalenges to design a chip with a very large number of cores?