Master Informatics Eng.

2022/23

Alberto José Proença & João Luís Sobral

Background to this course
(slides from several sources)

Course foundations

- ***Lecturing/teaching team***
  - *Lectures (only 1 T/week, 9h00 in 0.04): Alberto Proença & João Sobral*
  - *Lab classes (less than 8 PL): João Sobral, Rui Silva & Eduardo Conceição*
- ***Documentation & oral lectures/classes***
  - *Documentation (slides, lab guides, books, reports…): English, on BB*
  - *Oral presentations and work defence: optional, but usually Portuguese*
- ***Public holidays*** *(5th Oct & 1st Nov)*
  - *Move PL classes to 2 slots on Friday?...*
- ***Expected background***
  - *Imperative Programming and Computer Architecture, similar to what is in http://gec.di.uminho.pt/miei/sc2021/*
  - *Lab skills in Unix, C programming and debugging*
- ***Assessment***
  - *1x written test/exam (3rd Jan, weight: 30%)*
  - *2x homeworks w/ reports (delivery dates 28th Oct & 11th Nov, weight: ~30%)*
  - *1x final work w/ report & oral defence (3rd Jan, weight: ~40%)*

Course contents in a Lic. degree

http://gec.di.uminho.pt/miei/sc2021/

B. Programa Detalhado

1. Organização e estrutura de um computador

a. Conceito de computador. Representação da informação no computador: texto, números, informação multimédia e comandos codificados para o processador. Sistemas de numeração e conversão de bases; representação binária de valores positivos e negativos. Representação binária de valores reais em vírgula flutuante; a norma IEEE 754.

b. Análise da estrutura interna dum computador, com destaque para o processador e para a organização da memória. Análise da execução de instruções num computador. Níveis de abstracção num computador. Mecanismos para execução de programas; introdução a um ambiente laboratorial (Intel IA-32 em Unix/Linux).

2. Análise da arquitetura do *instruction set*

a. Análise do funcionamento dum processador e respectivo *instruction set*: operações/operandos, acesso a dados, tipos e formatos de instruções, modelo de programação dum processador.

b. Análise teórica e laboratorial do nível ISA do Intel IA-32 em Unix/Linux: operações aritméticas/lógicas e acesso a operandos, estruturas de controlo presentes em C, funcionamento das funções/procedimentos.

c. Análise comparativa do modelo de implementação numa arquitetura IA-32 estendida para 64 bits e em outras segundo o modelo RISC (ARM e MIPS), *versus* IA-32.

3. Avaliação do desempenho de computadores

a. Metodologia de avaliação de desempenho de computadores. Medição de tempos de execução de programas e respectiva análise prática e crítica.

b. Fatores da arquitectura dum computador que influenciam o seu desempenho. Breve introdução ao funcionamento dum processador em *pipeline* e respectivas limitações. Organização hierárquica da memória; noção e caracterização da *cache*. A organização de vários núcleos de processamento (*core*) num mesmo circuito integrado.

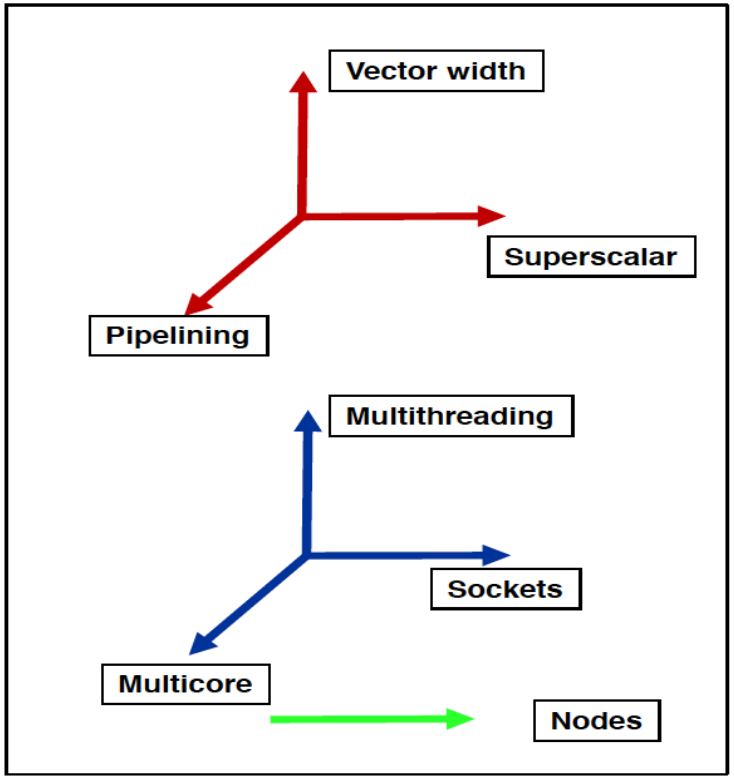
Focus on this course: performance engineering

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



Reading recommendations

http://gec.di.uminho.pt/mei/cp2122/ (from previous year)

Bibliografia

- *Computer Organization and Design*, 5th Ed., David Patterson & John Hennessy, Morgan Kaufmann, 2013
  ***** nos sumários com a sigla COD *****
- *Computer Architecture. A Quantitative Approach*, 6th Ed., David Patterson & John Hennessy, Morgan Kaufmann, 2017
  ***** nos sumários com a sigla CAQA *****
- *Parallel Computing Architectures and APIs*, Vivek Kale, Chapman and Hall/CRC, 2019
  ***** nos sumários com a sigla PCA *****

- *Designing and Programming Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Ian Foster, Addison-Wesley, 1995
  (free online at http://www.mcs.anl.gov/~itf/dbpp/text/book.html, or here)
  ***** nos sumários com a sigla DPPP *****
- *Parallel Programming in C with MPI and OpenMP*, Michael J. Quinn, McGraw-Hill Education, 2003
  ***** nos sumários com a sigla PPC *****
- *Structured Parallel Programming*, Michael McCool, Arch Robison & James Reinders, 2018
  ***** nos sumários com a sigla SPP *****
- *Programming Massively Parallel Processors, A Hands-on Approach*, 3rd Ed., David Kirk & Wen-mei Hwu, Morgan Kaufmann, 2016
  ***** nos sumários com a sigla PMPP *****

Key textbook (1)

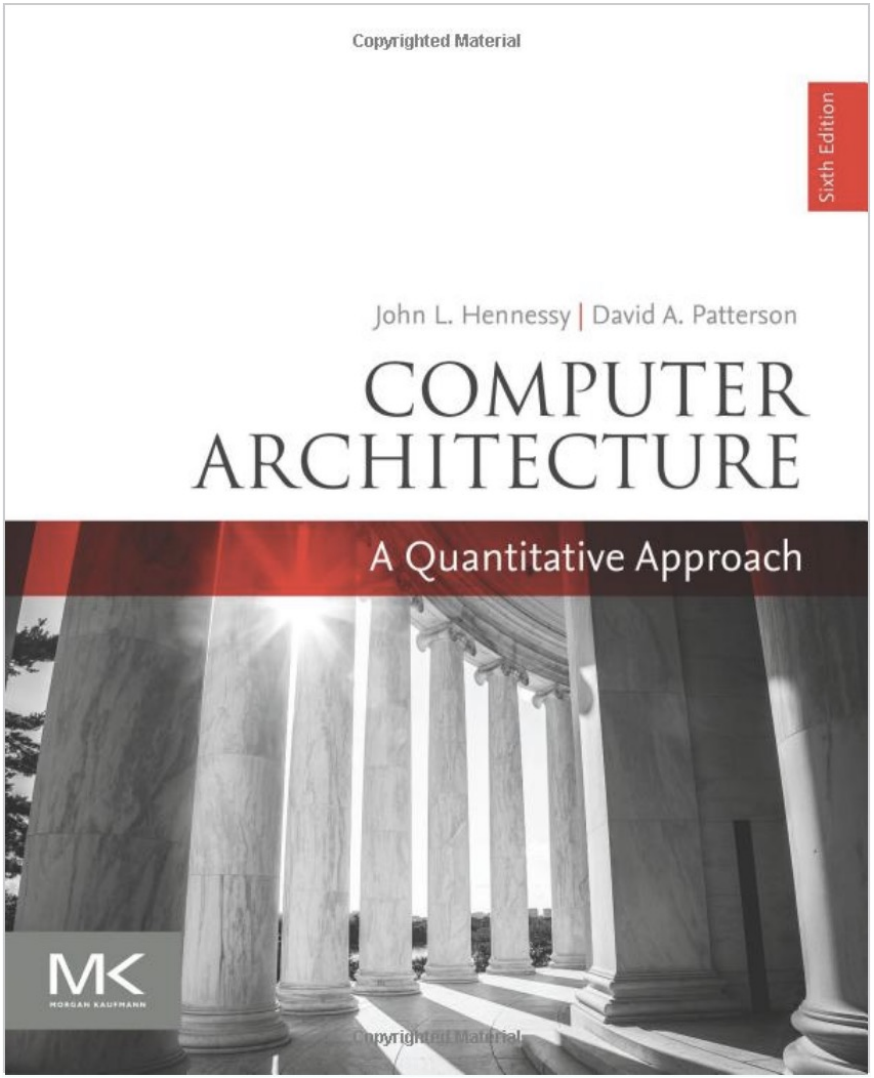


Table of Contents

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

MK®

Key textbook (2)

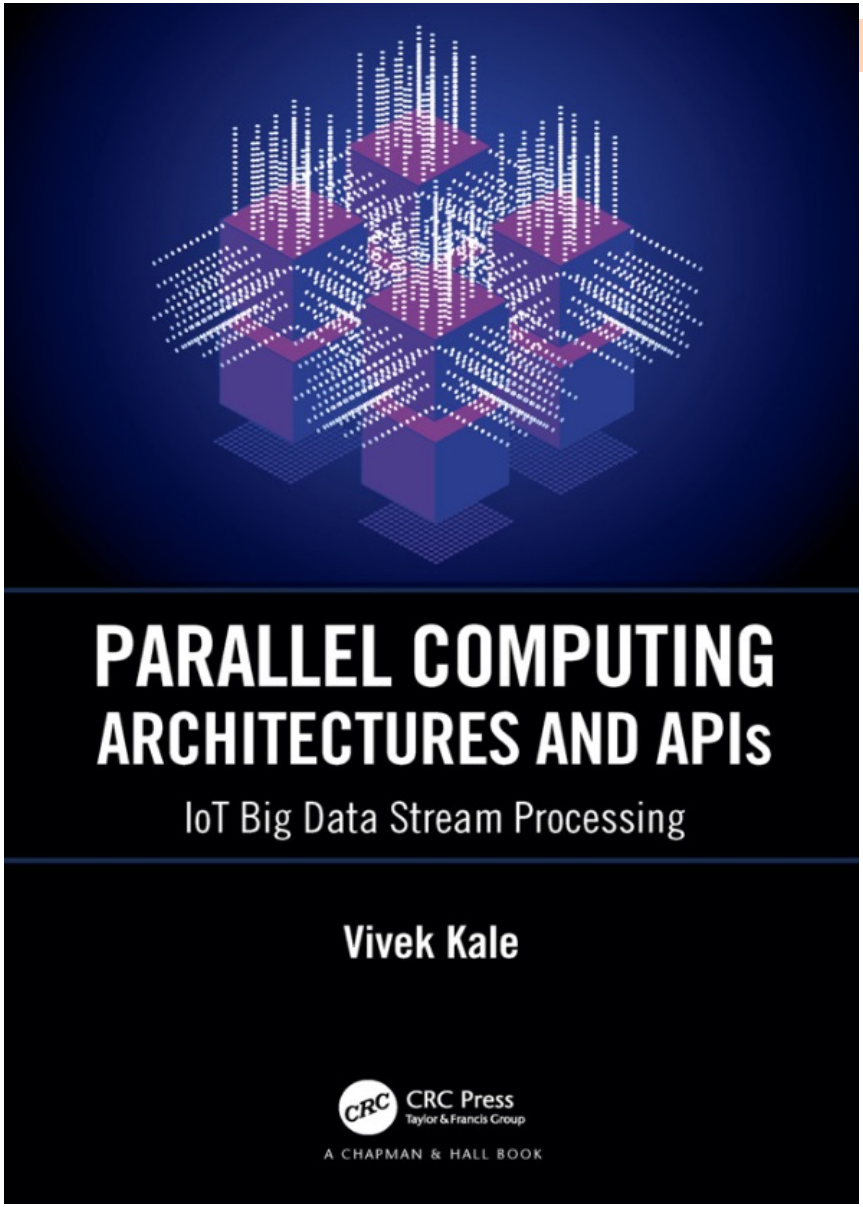


TABLE OF CONTENTS

Key textbook (3)

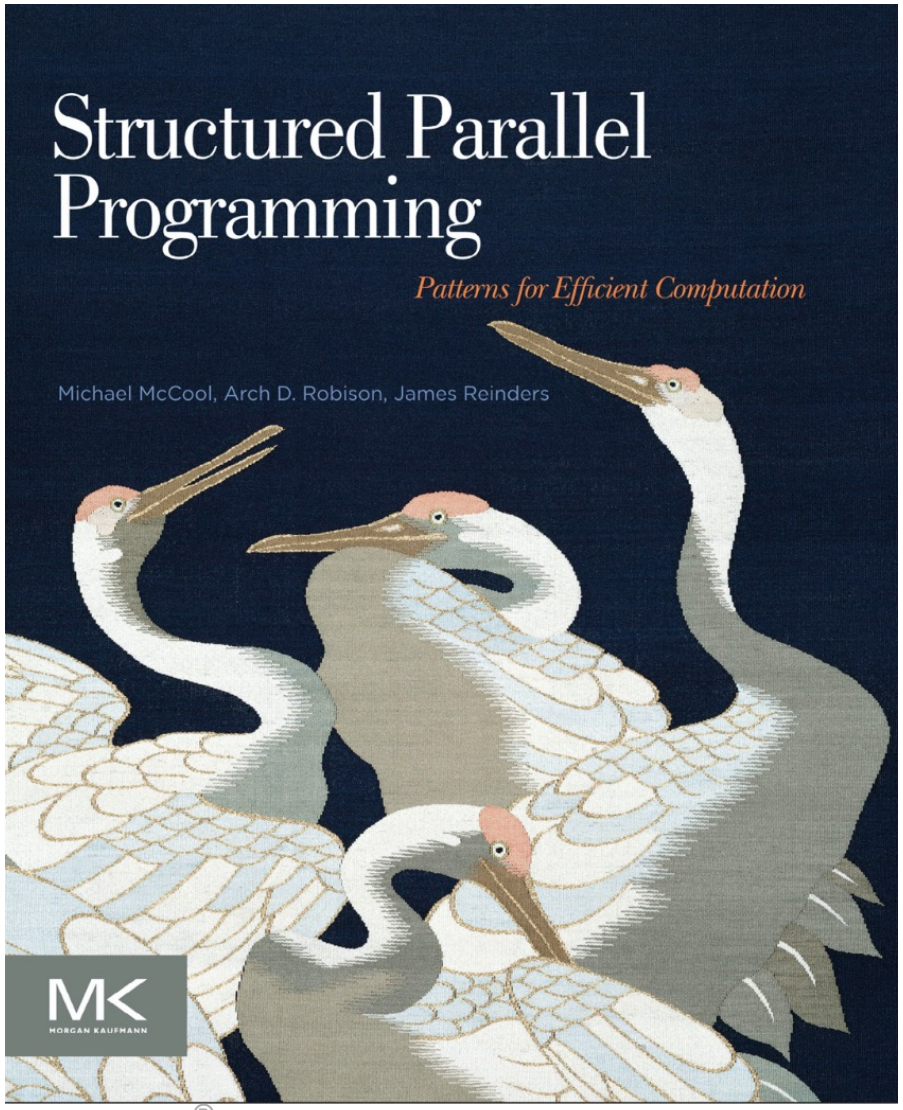
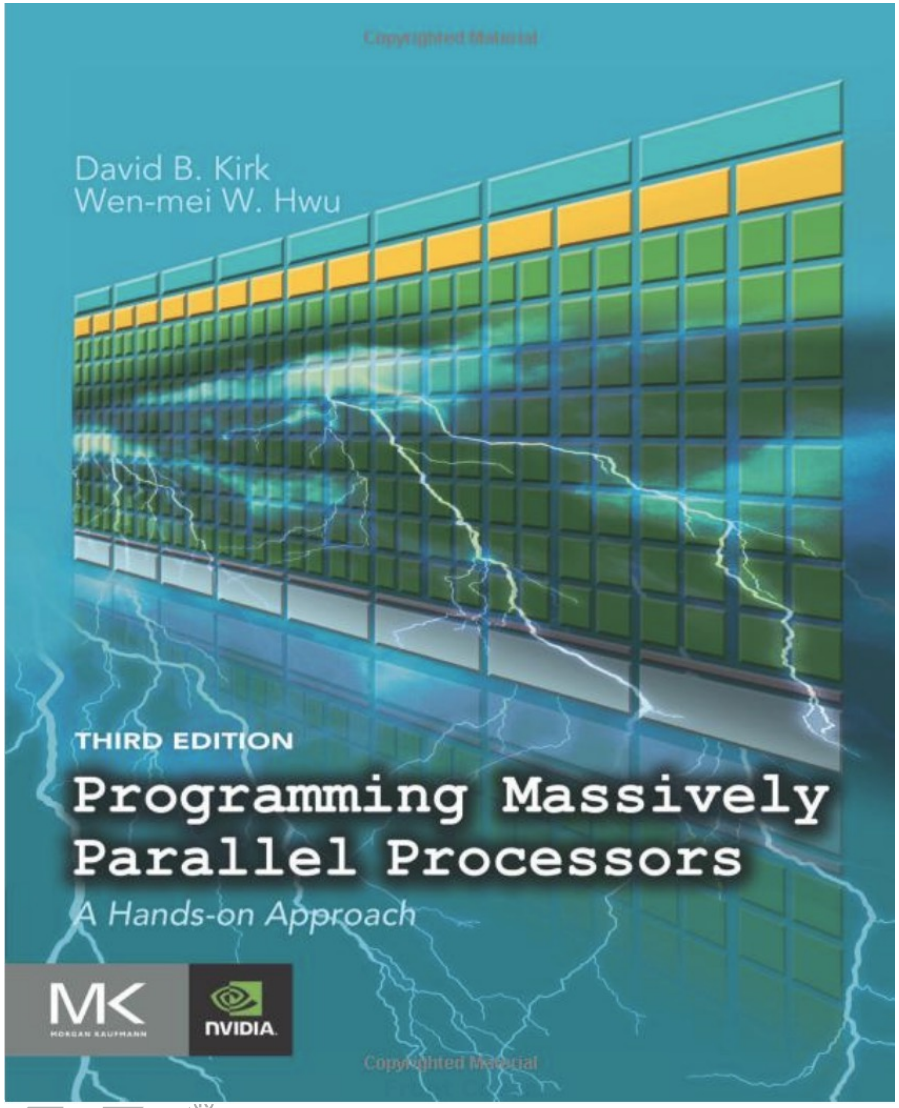


Table of contents

Recommended textbook (1)

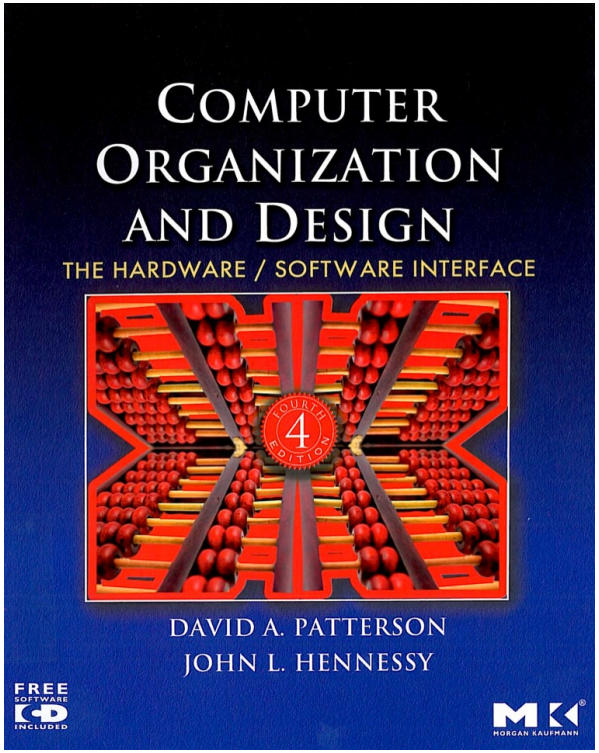


Contents

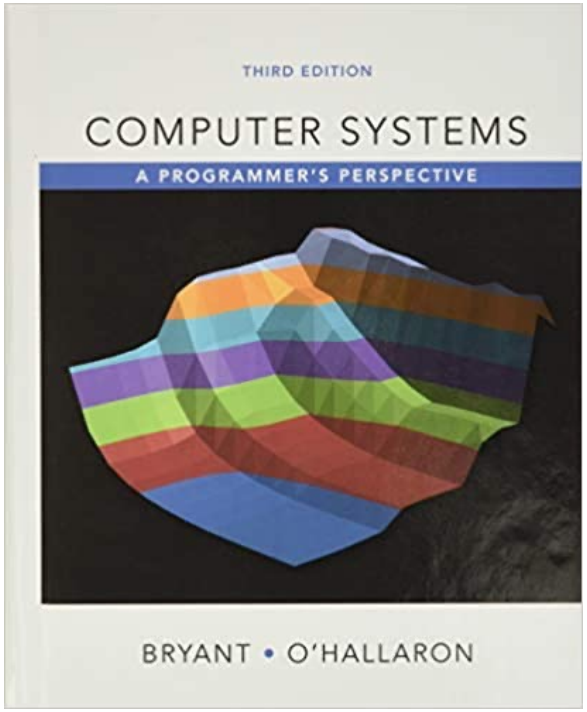
Background concepts from a basic Computer Systems course

Some notes/comments for this course:

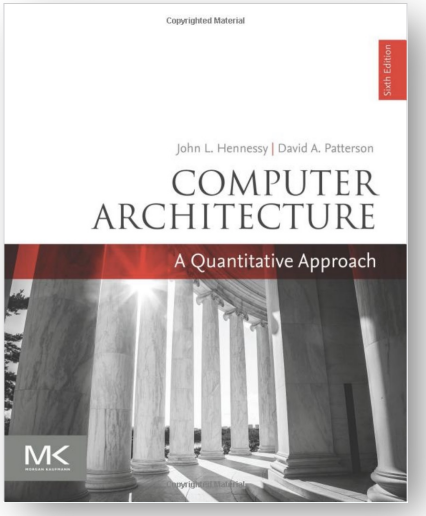
– *some slides are borrowed from*



and some from



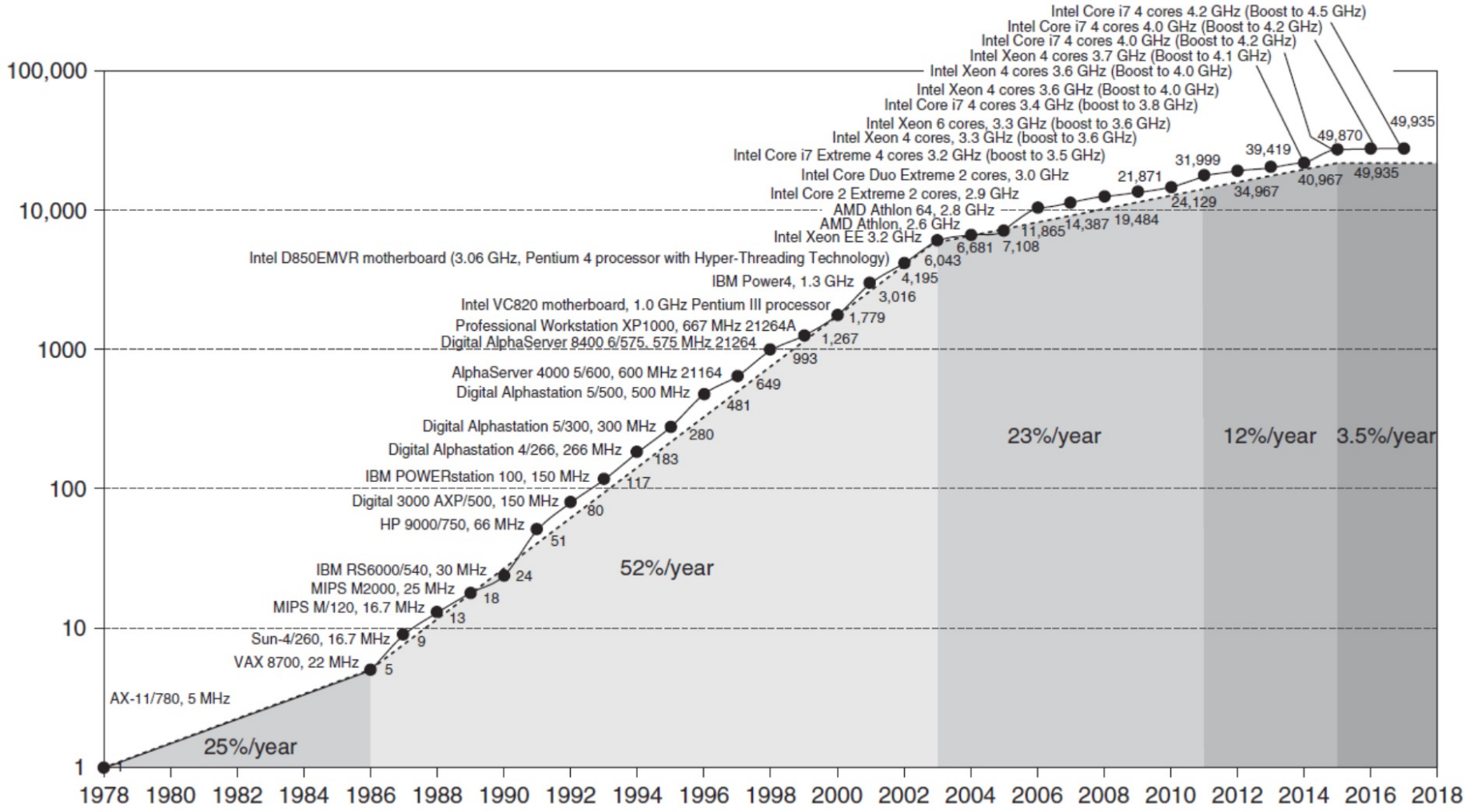
more details at
http://gec.di.uminho.pt/miei/sc2021/



Chapter 1

Fundamentals of Quantitative Design and Analysis

Single Processor Performance



Current Trends in Architecture

- Cannot continue to leverage Instruction-Level parallelism (ILP)
  - Single processor performance improvement ended in 2003
- New models for performance:
  - Data-level parallelism (DLP)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)
- These require explicit restructuring of the application

Classes of Computers

- Personal Mobile Device (PMD)
  - e.g. start phones, tablet computers
  - Emphasis on energy efficiency and real-time
- Desktop Computing
  - Emphasis on price-performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers
  - Used for “Software as a Service (SaaS)”
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Internet of Things/Embedded Computers
  - Emphasis: price

Parallelism

- Classes of parallelism in applications:
  - Data-Level Parallelism (DLP)
  - Task-Level Parallelism (TLP)

- Classes of architectural parallelism:
  - Instruction-Level Parallelism (ILP)
  - Vector architectures/Graphic Processor Units (GPUs)
  - Thread-Level Parallelism
  - Request-Level Parallelism

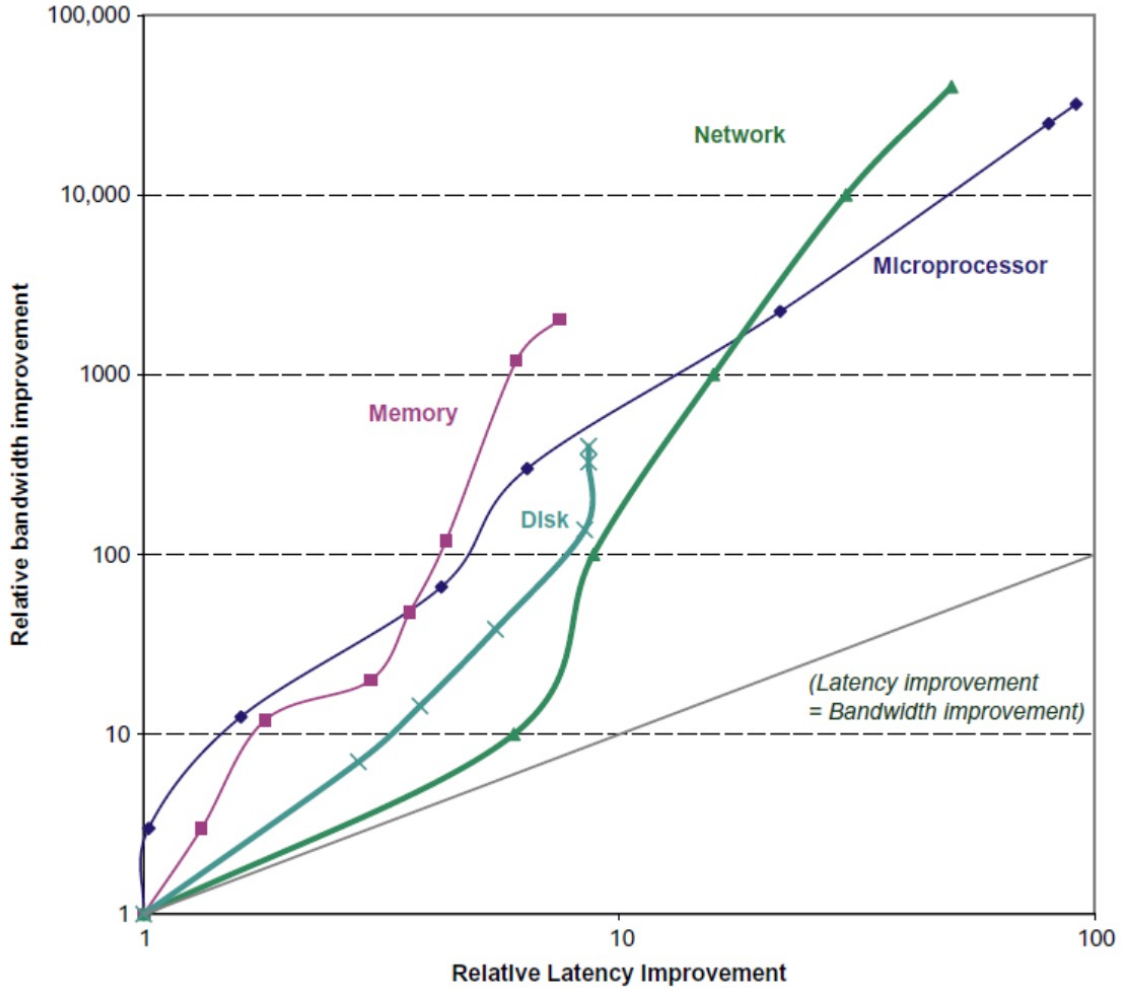
Flynn's Taxonomy

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data streams (SIMD)
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units
- Multiple instruction streams, single data stream (MISD)
  - No commercial implementation
- Multiple instruction streams, multiple data streams (MIMD)
  - Tightly-coupled MIMD
  - Loosely-coupled MIMD

Bandwidth and Latency

- Bandwidth or throughput
  - Total work done in a given time
  - 32,000-40,000x improvement for processors
  - 300-120x improvement for memory and disks
- Latency or response time
  - Time between start and completion of an event
  - 50-90x improvement for processors
  - 6-8x improvement for memory and disks

Bandwidth and Latency

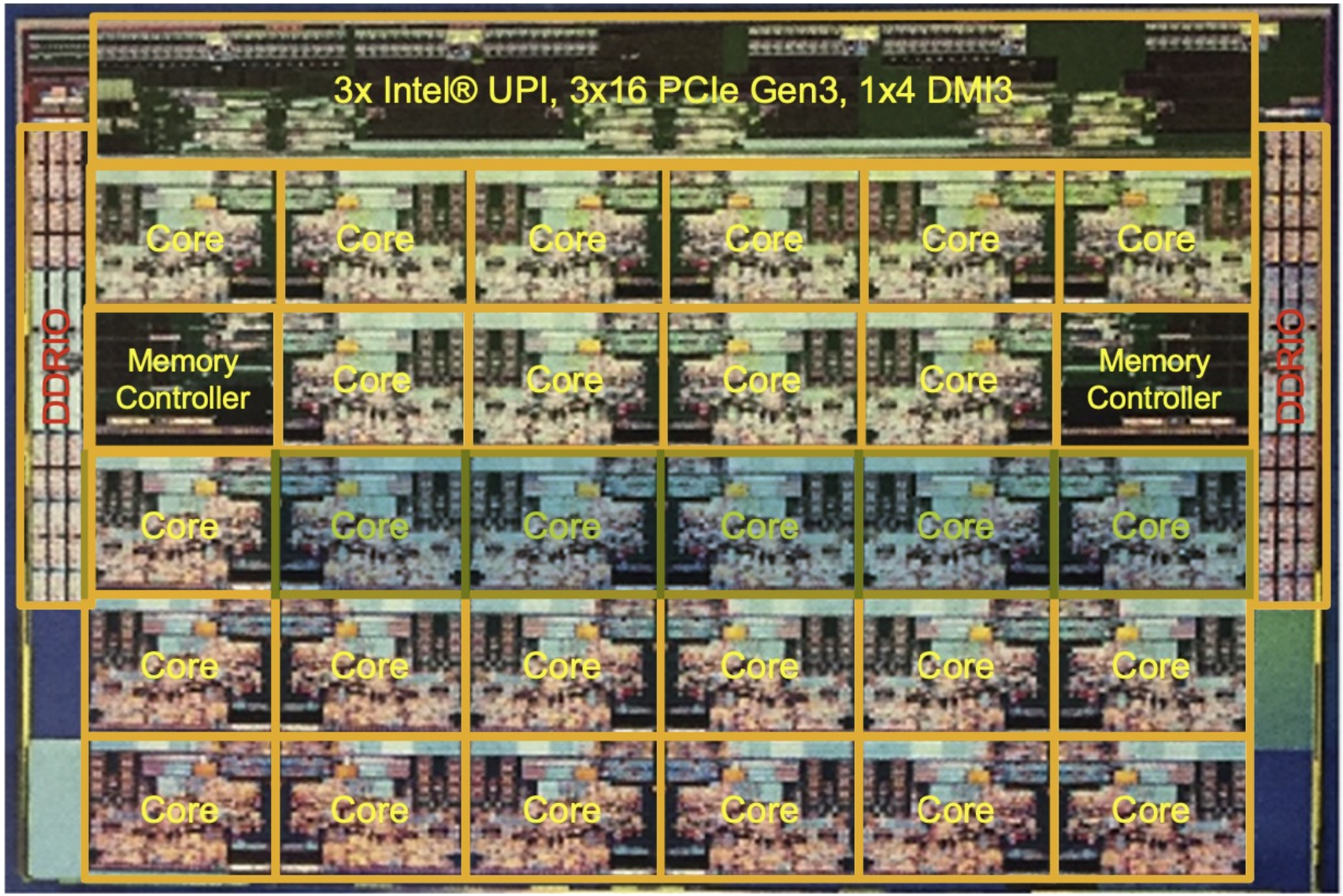


Log-log plot of bandwidth and latency milestones

Performance milestones

Microprocessor	16-Bit address/ bus, microcoded	32-Bit address/ bus, microcoded	5-Stage pipeline, on-chip I & D caches, FPU	2-Way superscalar, 64-bit bus	Out-of-order 3-way superscalar	Out-of-order superpipelined, on-chip L2 cache	Multicore OOO 4-way on chip L3 cache, Turbo
Product	Intel 80286	Intel 80386	Intel 80486	Intel Pentium	Intel Pentium Pro	Intel Pentium 4	Intel Core i7
Year	1982	1985	1989	1993	1997	2001	2015
Die size (mm^2)	47	43	81	90	308	217	122
Transistors	134,000	275,000	1,200,000	3,100,000	5,500,000	42,000,000	1,750,000,000
Processors/chip	1	1	1	1	1	1	4
Pins	68	132	168	273	387	423	1400
Latency (clocks)	6	5	5	5	10	22	14
Bus width (bits)	16	32	32	64	64	64	196
Clock rate (MHz)	12.5	16	25	66	200	1500	4000
Bandwidth (MIPS)	2	6	25	132	600	4500	64,000
Latency (ns)	320	313	200	76	50	15	4
Memory module	DRAM	Page mode DRAM	Fast page mode DRAM	Fast page mode DRAM	Synchronous DRAM	Double data rate SDRAM	DDR4 SDRAM
Module width (bits)	16	16	32	64	64	64	64
Year	1980	1983	1986	1993	1997	2000	2016
Mbits/DRAM chip	0.06	0.25	1	16	64	256	4096
Die size (mm^2)	35	45	70	130	170	204	50
Pins/DRAM chip	16	16	18	20	54	66	134
Bandwidth (MBytes/s)	13	40	160	267	640	1600	27,000
Latency (ns)	225	170	125	75	62	52	30

Intel Skylake microprocessor die



Measuring Performance

- Typical performance metrics:
  - Response time
  - Throughput
- Speedup of X relative to Y
  - Execution time$_Y$ / Execution time$_X$
- Execution time
  - Wall clock time: includes all system overheads
  - CPU time: only computation time
- Benchmarks
  - Kernels (e.g. matrix multiply)
  - Toy programs (e.g. sorting)
  - Synthetic benchmarks (e.g. Dhrystone)
  - Benchmark suites (e.g. SPEC06fp, TPC-C)

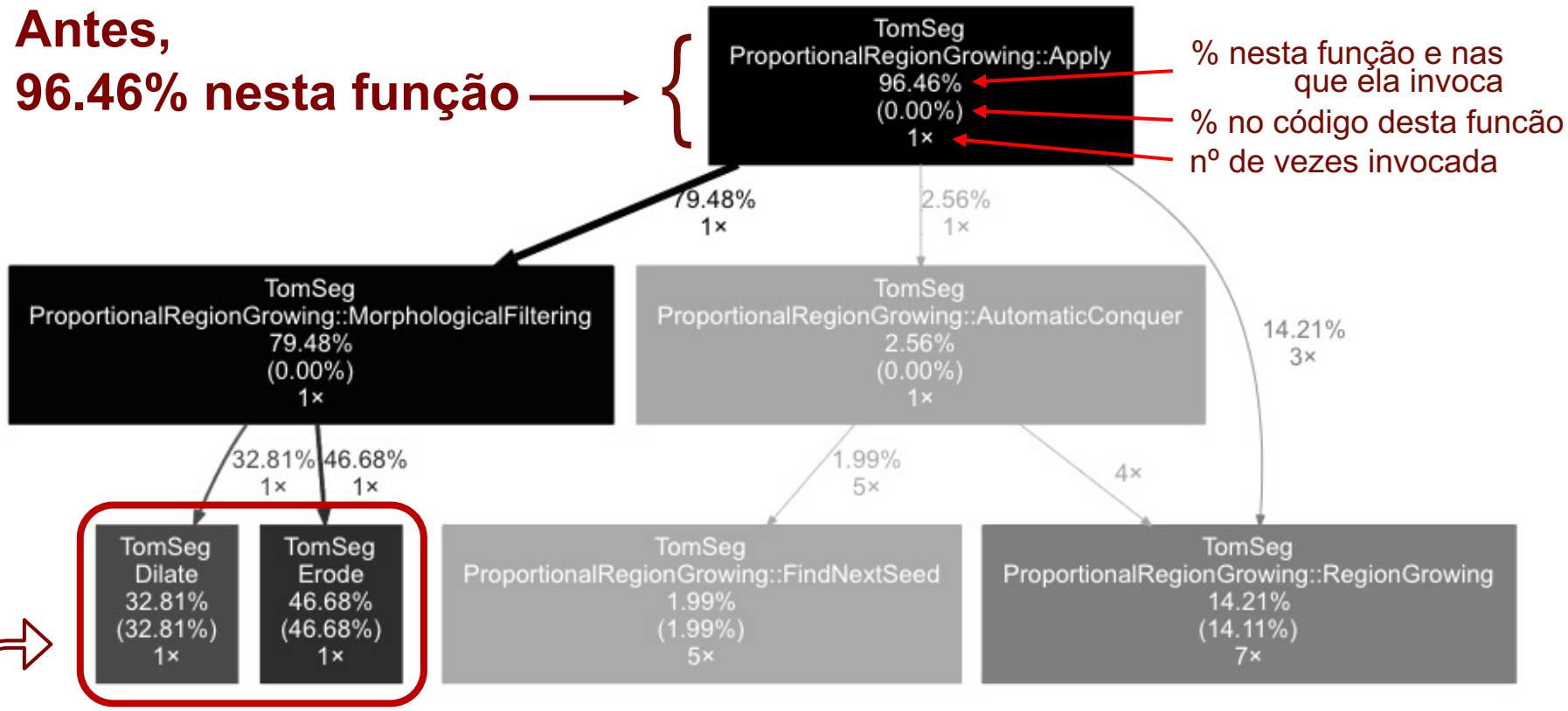
Principles of Computer Design

- Take Advantage of Parallelism
  - e.g. multiple processors, disks, memory banks, pipelining, multiple functional units
- Principle of Locality
  - Reuse of data and instructions
- Focus on the Common Case
  - Amdahl's Law

$$Execution\ time_{new} = Execution\ time_{old} \times \left((1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)$$

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

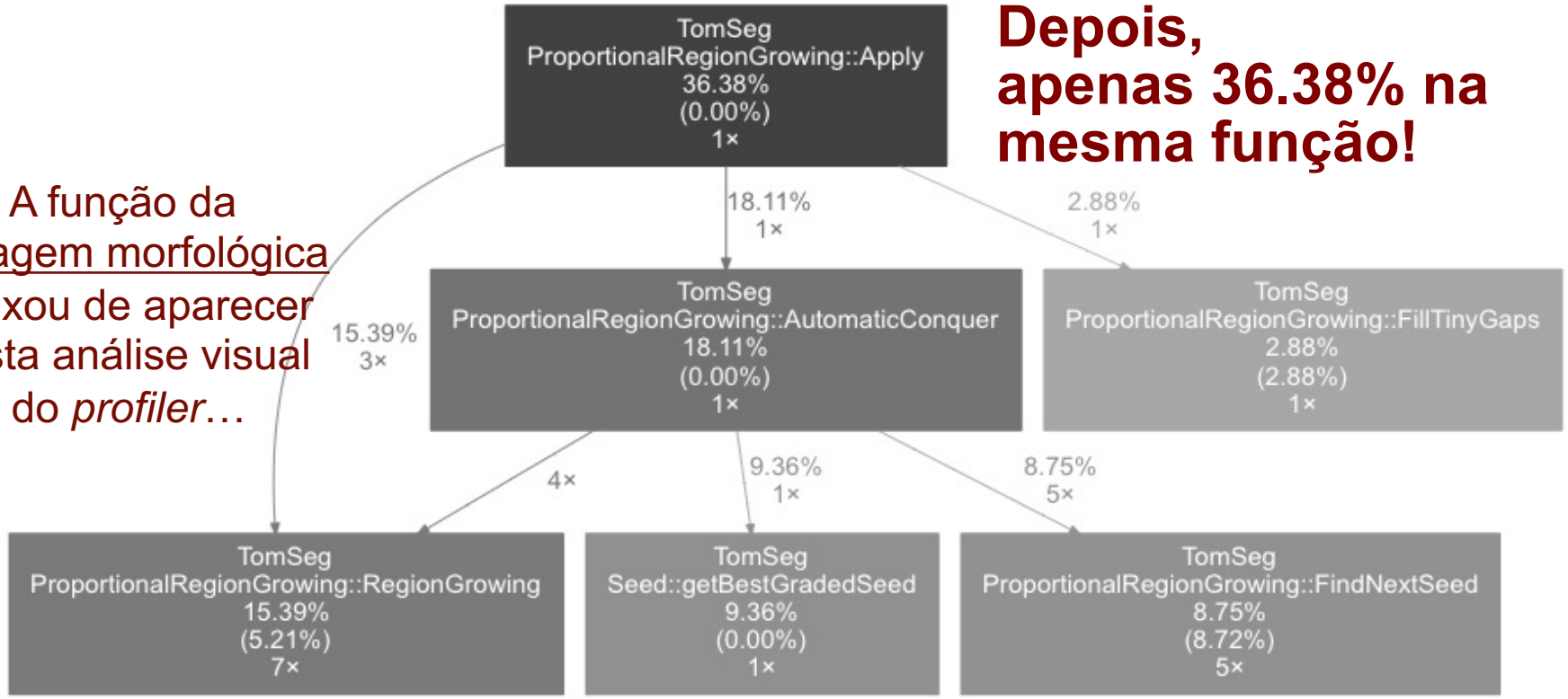
Code profiling: análise visual da melhoria de código duma função (antes)



Quase 80% do tempo total é gasto nestas 2 funções da filtragem morfológica!
Conclusão: é aqui que se deve investir para melhorar a *performance* global

Figure 5.7.: Call-graph of the first version of *Propor. Region Growing* (DS3)

Code profiling: análise visual da melhoria de código duma função (depois)



Depois, apenas 36.38% na mesma função!

A função da filtragem morfológica deixou de aparecer nesta análise visual do *profiler*…

Figure 5.9.: Call-graph from the last version of *Propor. Region Growing* (DS3)

Code profiling e visualização: dicas para a sua realização

Mais comum para aplicações em ambiente GNU/Linux:

- `gprof` : disponível em `gcc/binutils`
- `gprof` requer compilação com `-pg` para gerar o ficheiro `gmon.out` contendo os dados para o *profiler*
- `gprof` lê a informação em `gmon.out` e produz um relatório em `main.gprof`
- `gprof2dot` lê `main.gprof` e gera uma representação visual dos dados tal como apresentado nos slides anteriores
- `gprof2dot` disponível em https://github.com/jrfonseca/gprof2dot
- para visualizar a imagem do *call graph* usar o comando `xdot`

https://developer.ridgerun.com/wiki/index.php/Profiling_GNU/Linux_applications

Lei de Amdahl

O ganho no desempenho – *speedup* –
obtido com a melhoria do tempo de execução de uma parte do sistema,
está limitado pela fração de tempo que essa parte do sistema pode ser usada.

$$Speedup_{overall} = \frac{Tempo\_exec_{antigo}}{Tempo\_exec_{novo}} = \frac{1}{\Sigma\ (f_i\ /\ s_i)}$$

f_i - fracções com melhoria s_i
s_i - *speedup* de cada fracção

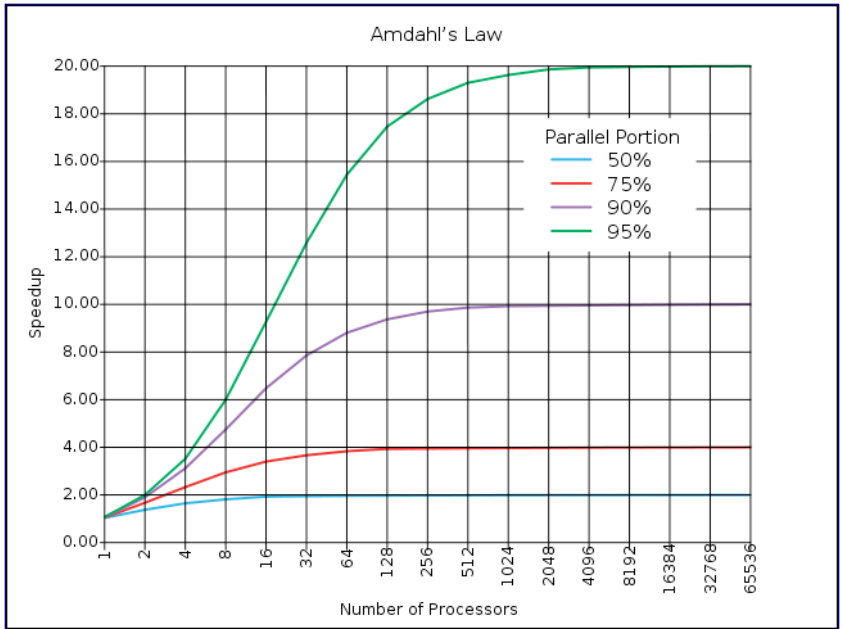
Ex.1: Se **10%** de um programa executa **90x** mais rápido

***Overall speedup* = 1.11**

Ex.2: Se **90%** de um prog executa **90x** mais rápido

***Overall speedup* = 9.09**

Paralelismo:
se N_{proc} Ξ ***speedup***, trocar s_i por N_{proc}



Amdahl's Law

$$\text{Overall Speedup} = \frac{\text{Old execution time}}{\text{New execution time}}$$

$$= \frac{1}{\left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)}$$

$$Speedup = \frac{1}{(1-p) + p/N}$$

$$\text{Speedup}(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

Serial part of job =
1 (100%) - Parallel part

Parallel part is divided up by N workers

Principles of Computer Design

- The Processor Performance Equation

$$CPU\ time = CPU\ clock\ cycles\ for\ a\ program \times Clock\ cycle\ time$$

$$CPU\ time = \frac{CPU\ clock\ cycles\ for\ a\ program}{Clock\ rate}$$

$$CPI = \frac{CPU\ clock\ cycles\ for\ a\ program}{Instruction\ count}$$

$$CPU\ time = Instruction\ count \times Cycles\ per\ instruction \times Clock\ cycle\ time$$

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

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

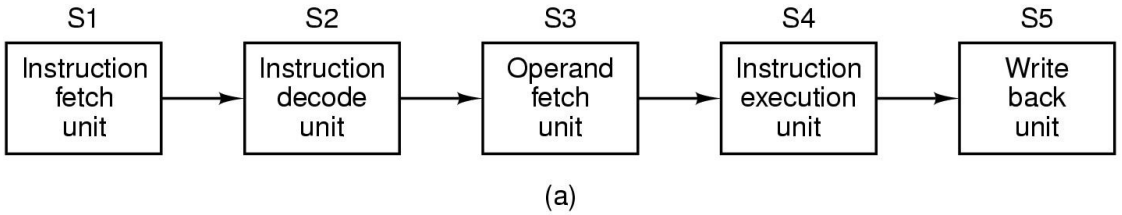
Background for Advanced Architectures in Parallel Computing

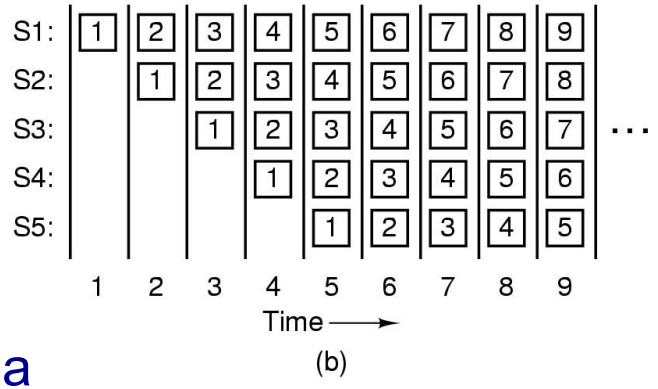
Key concepts to revise:

- *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**

Exemplo de *pipeline*





Objetivo

- **CPI = 1**

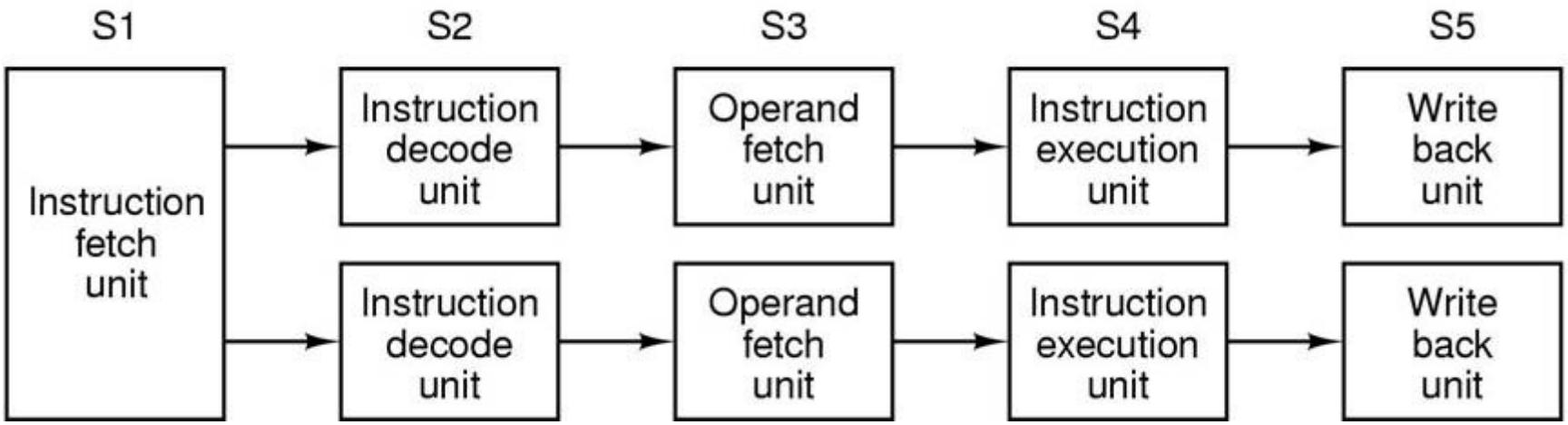
Problemas:

- 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

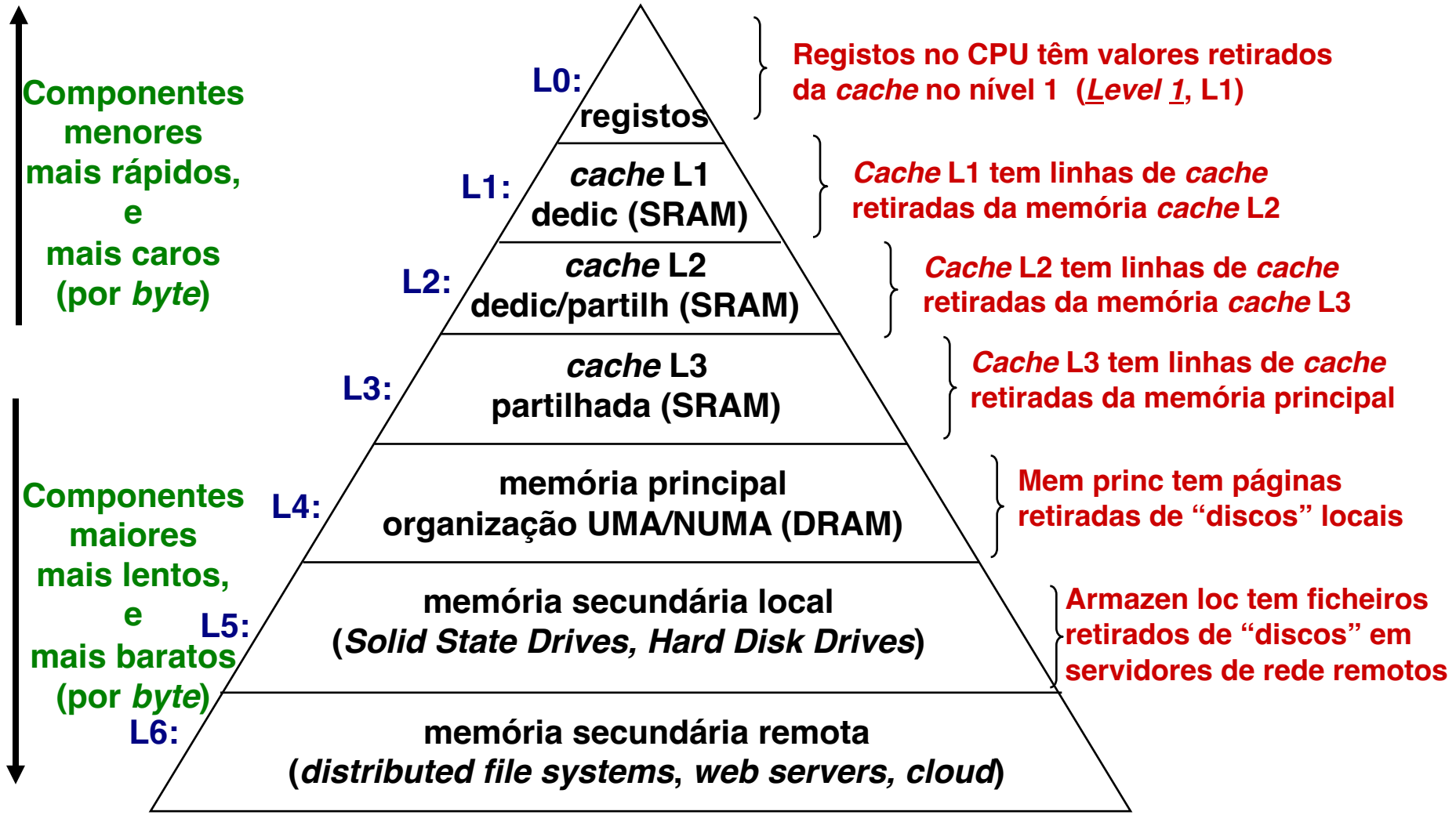
Exemplo de superescalaridade (2 vias)



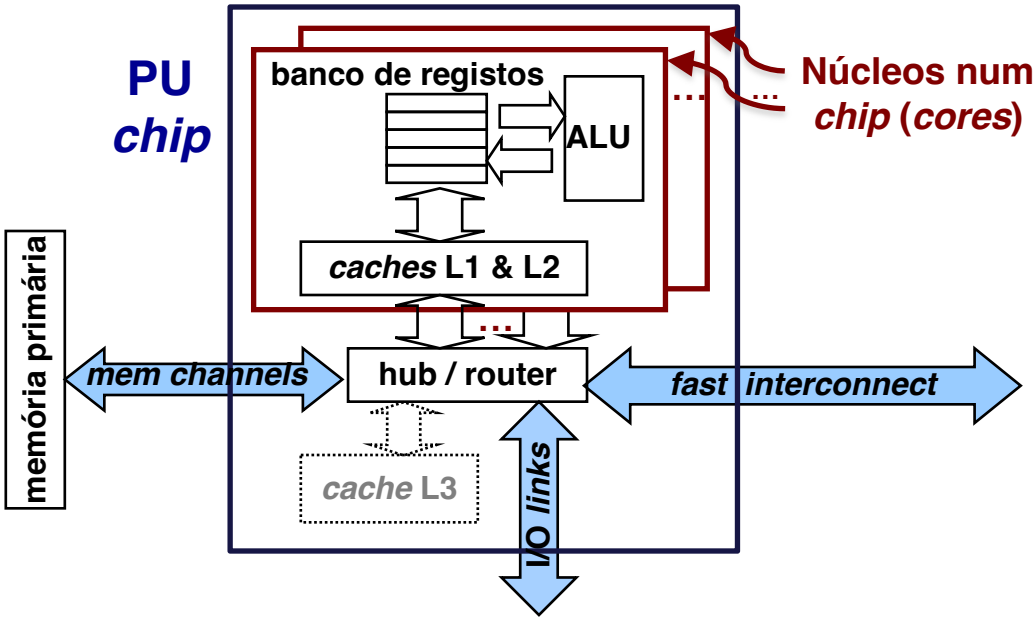
The BIG Picture

- 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



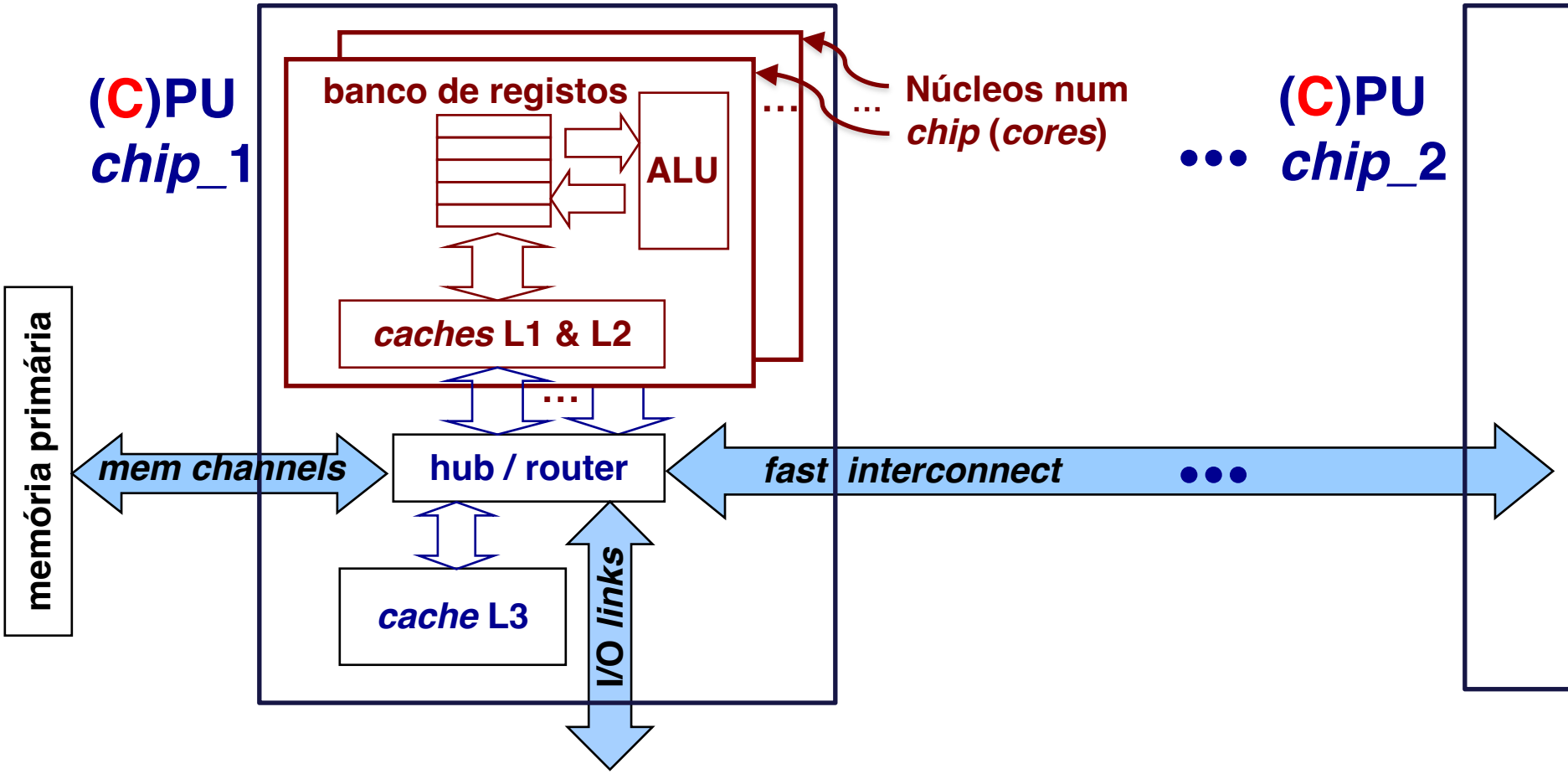
A cache em arquiteturas multicore



Notas:

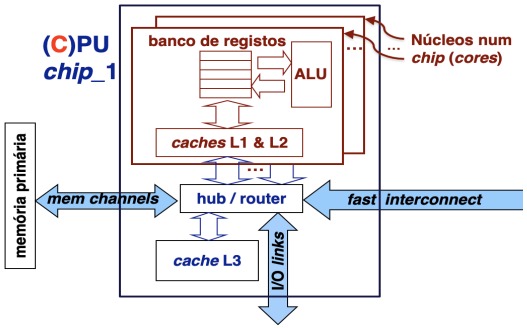
- as *caches* L1 de dados e de instruções são normalmente distintas
- as *caches* L2 em *multicores* podem ser partilhadas por outras *cores*
- muitos *cores* partilhando uma única memória traz complexidades:
  - manutenção da coerência da informação nas *caches*
  - encaminhamento e partilha dos circuitos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

Multicore architecture in a server system

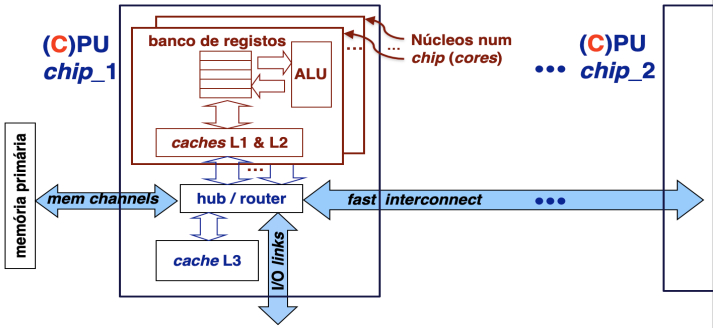


Memory systems in current computer architectures

Shared memory systems

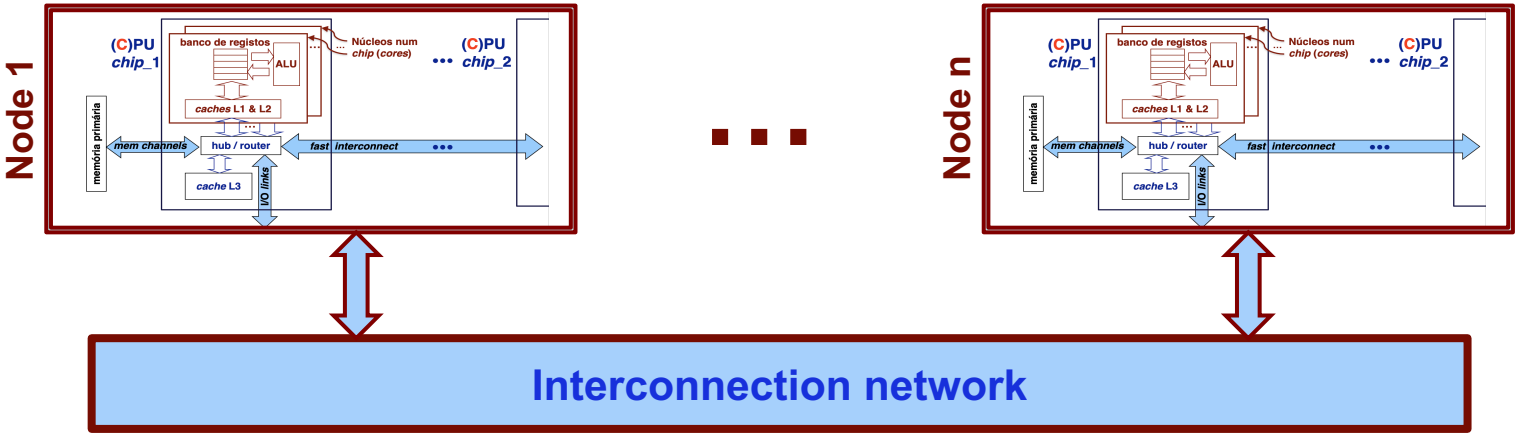


UMA: **U**niform **M**emory **A**ccess



NUMA: **N**on-**U**niform **M**emory **A**ccess

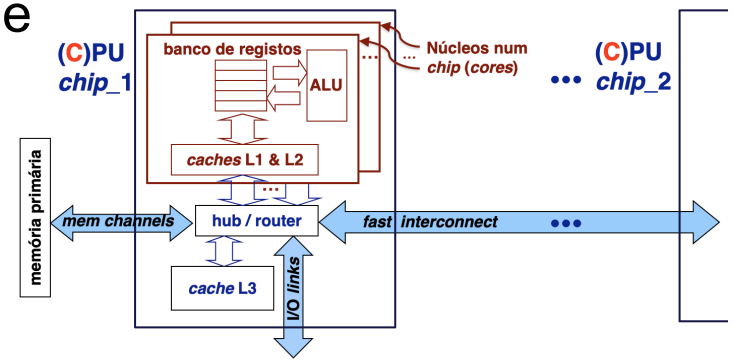
Distributed shared-memory systems



Multicore architectures: homework...

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?

Improving code performance to explore ILP: an example from the Computer Systems course

The following slides are a selection from CS.

The originals are in:

- http://gec.di.uminho.pt/mei/cp2122/slides_sc.zip

Two years ago lectures were recorded and the videos were placed on the e-platform; they are available here:

- http://gec.di.uminho.pt/mei/cp2122/videos_sc.zip