

MSc Informatics Eng.

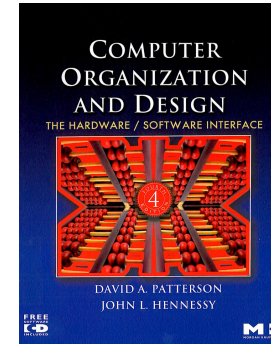
2014/15
A.J.Proença

Concepts from undergrad Computer Systems (1)

(most slides are borrowed, mod's in green)

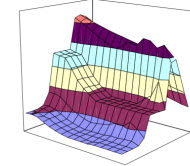
Concepts from undergrad Computer Systems

– most slides are borrowed from



and some from

Computer Systems
A Programmer's Perspective¹
(Beta Draft)



Randal E. Bryant
David R. O'Hallaron
August 1, 2001

more details at
<http://gec.di.uminho.pt/lei/sc/>

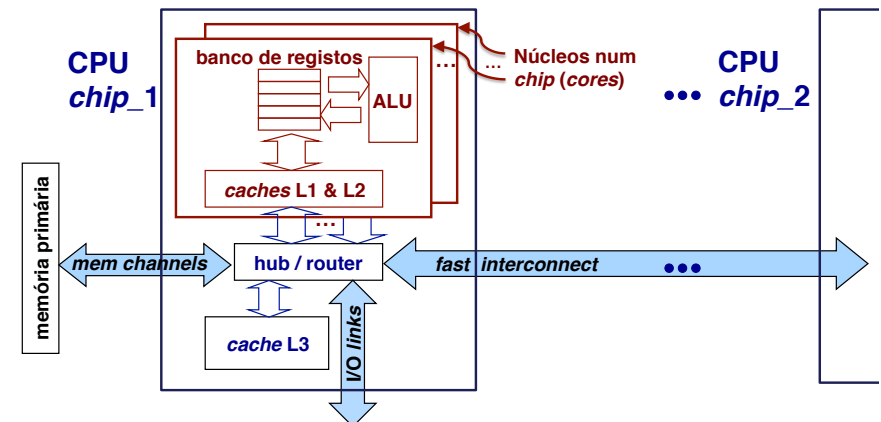
Background for Advanced Architectures

Key concepts to revise:

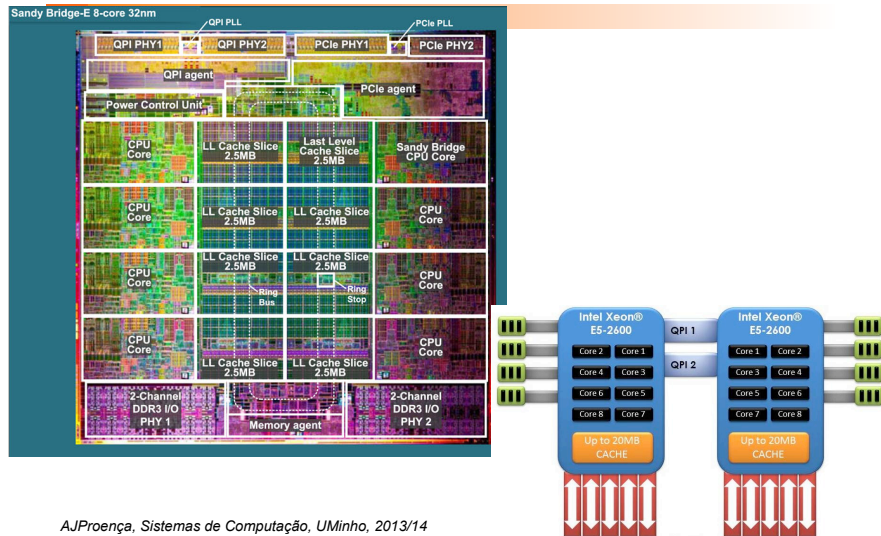
- numerical data representation (for error analysis)
- ISA (Instruction Set Architecture)
- how C compilers generate code (a look into assembly code)
 - how scalar and structured data are allocated
 - how control structures are implemented
 - how to call/return from function/procedures
 - what architecture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

A hierarquia de cache em arquiteturas multicore

As arquiteturas multicore mais recentes:

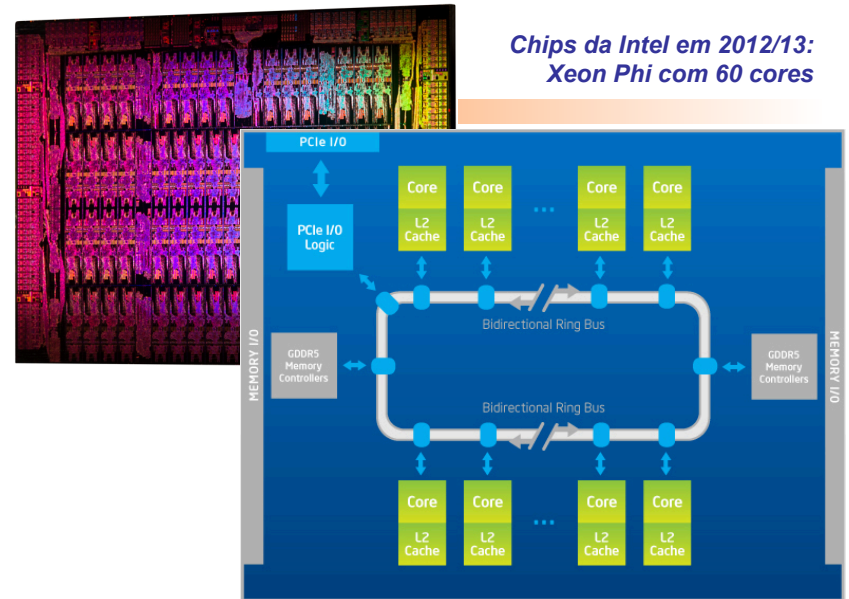


**Lançamento da Intel em 2012:
Sandy/Ivy Bridge (8-core)**



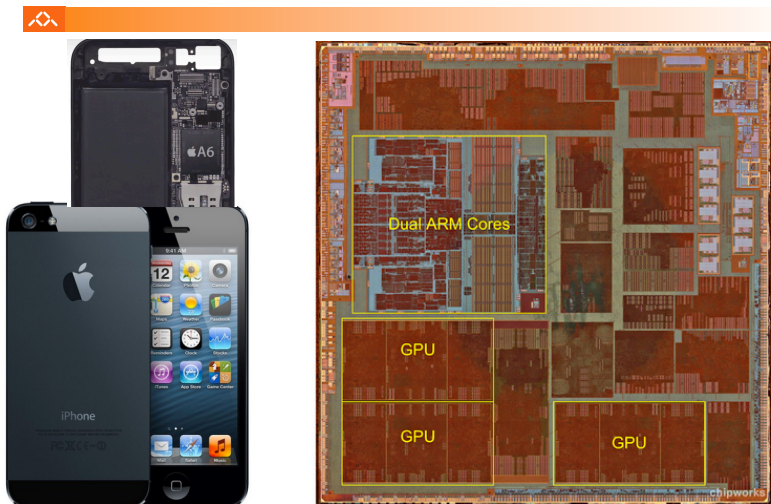
AJProença, Sistemas de Computação, UMinho, 2013/14

**Chips da Intel em 2012/13:
Xeon Phi com 60 cores**



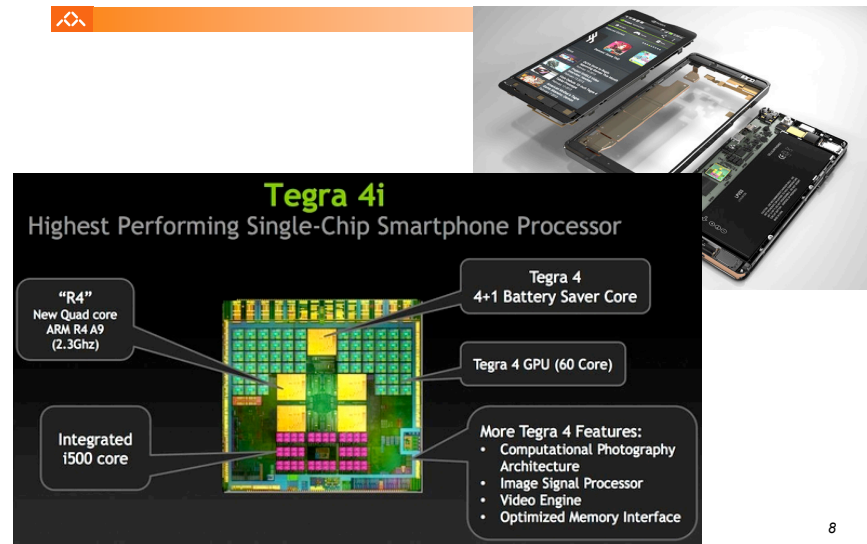
AJProença, Sistemas de Computação, UMinho, 2013/14

**Exemplo de chip com processadores RISC:
2x ARM's no A6 do iPhone 5**

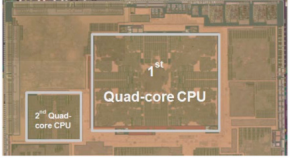
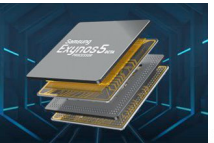
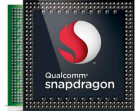


AJProença, Sistemas de Computação, UMinho, 2013/14

**Exemplo de chip com processadores RISC:
4+1 ARM's no Tegra 4i da Nvidia**



Exemplo de chip com processadores RISC: 4+4 ARM's no Exynos 5 Octa, Galaxy S 4


Performance and Energy-Efficiency

LITTLE Most energy-efficient processor from ARM Cortex-A7

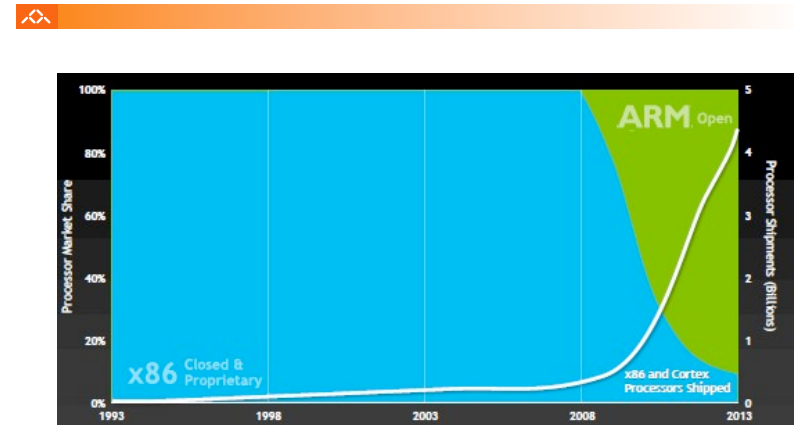
- Simple, in-order, 8 stage pipeline
- Performance better than today's mainstream, high-volume smartphones

big Highest performance in mobile power envelope Cortex-A15

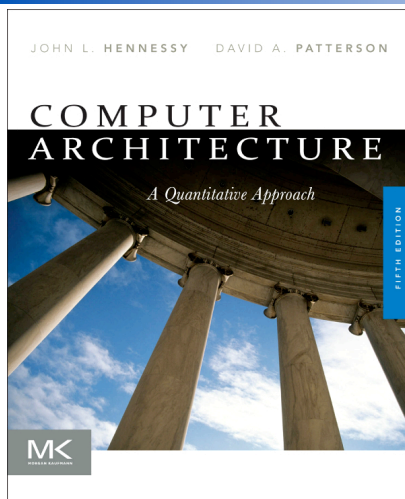
- Complex, out-of-order, multi-issue pipeline
- Up to 5x the performance of today's mainstream, high-volume smartphones



Processadores Intel x86 versus ARM



Key textbook for AA



Computer Architecture, 5th Edition

Hennessy & Patterson

Table of Contents

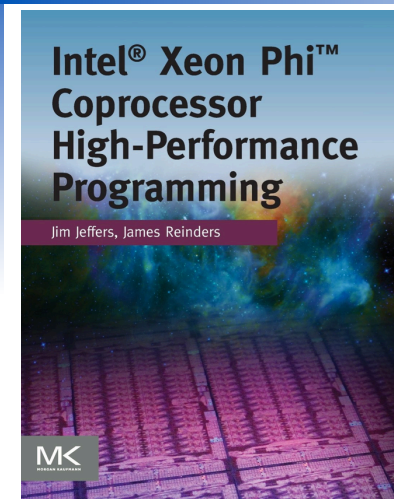
Printed Text

- Chap 1: Fundamentals of Quantitative Design and Analysis
- Chap 2: Memory Hierarchy Design
- Chap 3: Instruction-Level Parallelism and Its Exploitation
- Chap 4: Data-Level Parallelism in Vector, SIMD, and GPU Architectures
- Chap 5: Multiprocessors and Thread-Level Parallelism
- Chap 6: The Warehouse-Scale Computer
- App A: Instruction Set Principles
- App B: Review of Memory Hierarchy
- App C: Pipelining: Basic and Intermediate Concepts

Online

- App D: Storage Systems
- App E: Embedded Systems
- App F: Interconnection Networks
- App G: Vector Processors
- App H: Hardware and Software for VLIW and EPIC
- App I: Large-Scale Multiprocessors and Scientific Applications
- App J: Computer Arithmetic
- App K: Survey of Instruction Set Architectures
- App L: Historical Perspectives

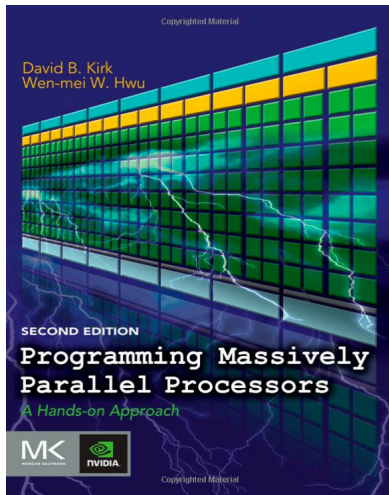
Recommended textbook (1)



Contents

1. Introduction
2. High Performance examples
3. Benchmarking Apps
4. Real-world Situations
5. Lots of Data (Vectors)
6. Lots of Tasks (not Threads)
7. Processing Parallelism
8. Coprocessor Architecture
9. Coprocessor System Software
10. Linux on the Coprocessor
11. Math Library
12. MPI
13. Profiling
14. Summary

Recommended textbook (2)



Contents

- 1 Introduction
- 2 History of GPU Computing
- 3 Introduction to Data Parallelism and CUDA C
- 4 Data-Parallel Execution Model
- 5 CUDA Memories
- 6 Performance Considerations
- 7 Floating-Point Considerations
- 8 Parallel Patterns: Convolution
- 9 Parallel Patterns: Prefix Sum
- 10 Parallel Patterns: Sparse Matrix-Vector Multiplication
- 11 Application Case Study: Advanced MRI Reconstruction
- 12 Application Case Study: Molecular Visualization and Analysis
- 13 Parallel Programming and Computational Thinking
- 14 An Introduction to OpenCL
- 15 Parallel Programming with OpenACC
- 16 Thrust: A Productivity-Oriented Library for CUDA
- 17 CUDA FORTRAN
- 18 An Introduction to C11 AMP
- 19 Programming a Heterogeneous Computing Cluster
- 20 CUDA Dynamic Parallelism
- 21 Conclusion and Future Outlook



Understanding Performance

- Algorithm + Data Structures
 - Determines number of operations executed
 - Determines how efficient data is assessed
- Programming language, compiler, architecture
 - Determine number of machine instructions executed per operation
- Processor and memory system
 - Determine how fast instructions are executed
- I/O system (including OS)
 - Determines how fast I/O operations are executed



COD: Chapter 1 — Computer Abstractions and Technology — 14

Response Time and Throughput

- Response time
 - How long it takes to do a task
- Throughput
 - Total work done per unit time
 - e.g., tasks/transactions/... per hour
- How are response time and throughput affected by
 - Replacing the processor with a faster version?
 - Adding more processors?
- We'll focus on response time for now...



Chapter 1 — Computer Abstractions and Technology — 15

CPU Time

(single-core)

$$\begin{aligned} \text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}} \end{aligned}$$

- Performance improved by
 - Reducing number of clock cycles
 - Increasing clock rate
 - Hardware designer must often trade off clock rate against cycle count



Chapter 1 — Computer Abstractions and Technology — 16

Instruction Count and CPI

Clock Cycles = Instruction Count × Cycles per Instruction

CPU Time = Instruction Count × CPI × Clock Cycle Time

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

- Instruction Count, **IC**, for a program
 - Determined by program, ISA and compiler
- Average cycles per instruction (**CPI**)
 - Determined by CPU hardware
 - If different instructions have different CPI
 - Average CPI affected by instruction mix



Performance Summary *(single-core)*

The BIG Picture

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

- Performance depends on
 - Algorithm: affects IC, possibly CPI
 - Programming language: affects IC, CPI
 - Compiler: affects IC, CPI
 - Instruction set architecture: affects IC, CPI, T_c
 - Processor design: **ILP**, **memory hierarchy**, ...



Pipeline Summary

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



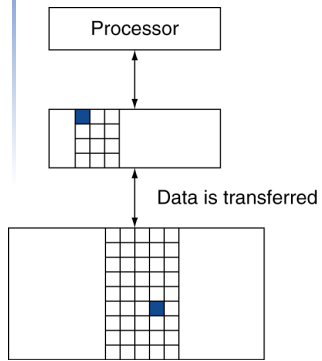
Does Multiple Issue Work?

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



Memory Hierarchy Levels



- Block (aka line): unit of copying
 - May be multiple words
- If accessed data is present in upper level
 - Hit: access satisfied by upper level
 - Hit ratio: hits/accesses
- If accessed data is absent
 - Miss: block copied from lower level
 - Time taken: miss penalty
 - Miss ratio: misses/accesses = 1 – hit ratio
 - Then accessed data supplied from lower level



The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
 - Based on notions of caching
- Decisions at each level in the hierarchy
 - Block placement
 - Finding a block
 - Replacement on a miss
 - Write policy

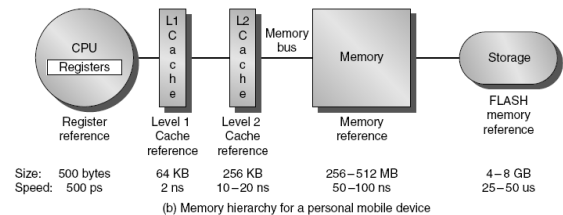
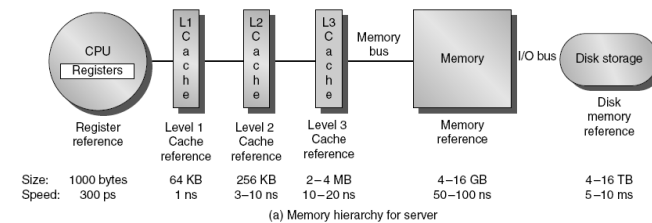


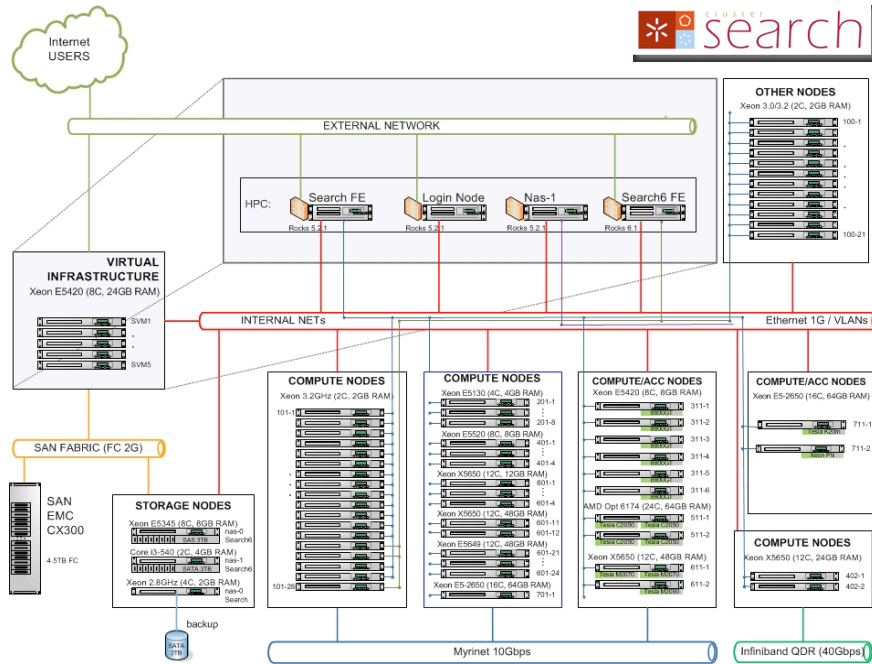
Multilevel Caches

- Primary cache private to CPU/core
 - Small, but fast
- Level-2 cache services misses from primary cache
 - Larger, slower, but still faster than main memory
- High-end systems include L3 cache
- Main memory services L2/3 cache misses



Memory Hierarchy





Homework



- Identify all Intel Xeon processors' microarchitecture from Core till the latest releases, and build a table with:
 - year, max clock frequency, # pipeline stages, degree of superscalarity, # simultaneous threads, vector support, # cores, type/bandwidth of external interfaces, ...
 - UMA/NUMA; for each cache level: size, latency, line size, direct/associative, bandwidth to access lower memory hierarchy levels, ... *(homework for following week)*
- Identify the CPU generations at the SeARCH cluster
- Suggestion:** create a *GoogleDocs* table, shared by all students, and all *critically* contribute to build the table