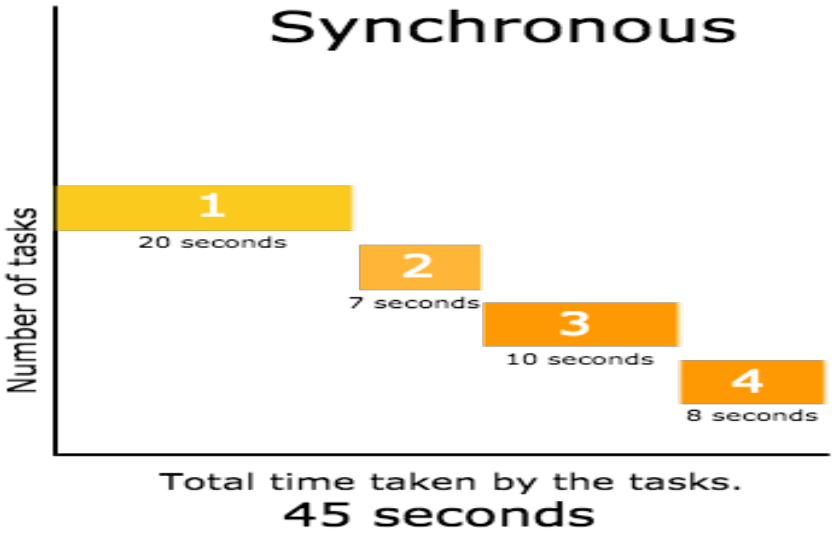
Big Data and Cloud Computing, 23/24

Inês Dutra

DCC-FCUP
room 1.31
ines@dcc.fc.up.pt

Apr 17th, 2024



Recalling Topics on Parallel Programming

1. Introduction
2. Parallel Programming Models
3. Parallel Architectures
4. Synchronization
5. Message Passing
6. Parallel Constructs and Techniques
7. Languages and runtime systems for parallel programming
8. Performance Issues

Introduction

- Why Parallelism?
- Dimensions of parallel programming
- Design and Verification of Parallel programs

Why parallelism?

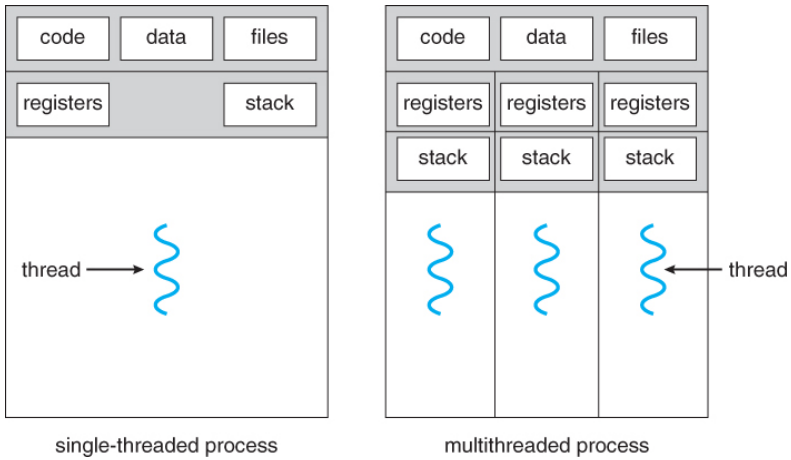
- Physical limits of sequential processor speed
- Natural parallelism in some applications
- System software
- Intrinsic interest

Dimensions of Parallelism

- Processes and threads
- Programming Models
- Concurrent x parallel x distributed
- Parallel and distributed systems
- Parallel architectures
- Languages and runtime
- Performance metrics

Processes and Threads

- Process → workspace
- Thread → same workspace as parent process
- Process (logical) != processor (physical hardware)
- Process is an abstraction of a processor
- Von Neumann model → one control flow
- concurrent program → 1+ flow



https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html

Processes and Threads

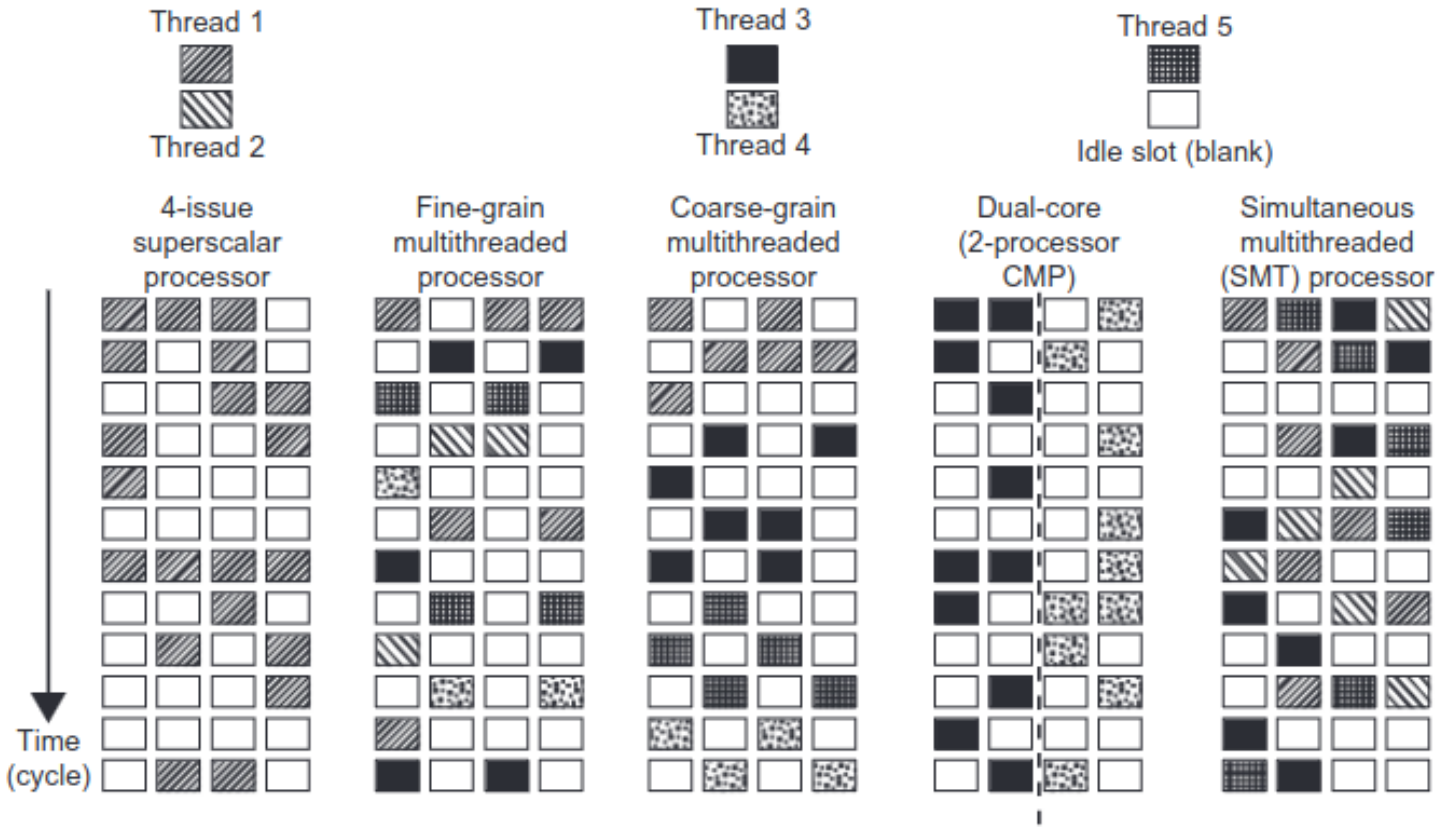


FIGURE 1.6

Five micro-architectures in modern CPU processors, that exploit ILP and TLP supported by multicore and multithreading technologies.

https://cutepooji.files.wordpress.com/2017/01/

distributed-and-cloud-computing-from-parallel-processing-to-the-internet-of-things.pdf

Programming Models

- Define interface used by the programmer
- Parallelism, communication, synchronization etc
- Examples: sequential, shared/centralized memory, message passing

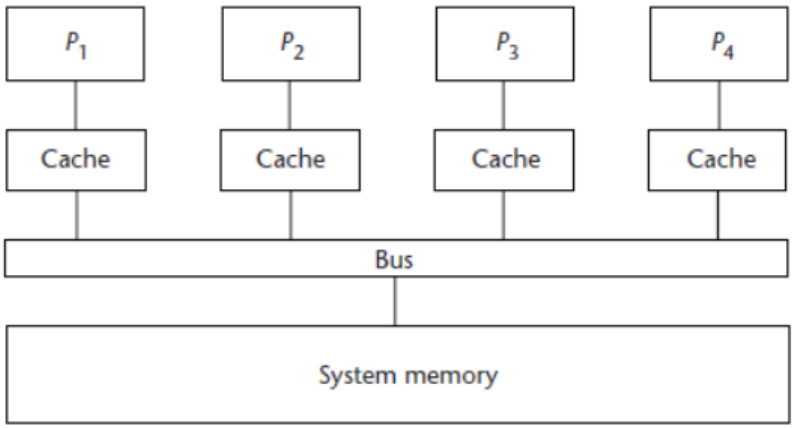
Concurrent x Parallel x Distributed

- Concurrent: 1+ control flow
- Parallel: concurrent with shared memory
- Distributed: concurrent with message passing

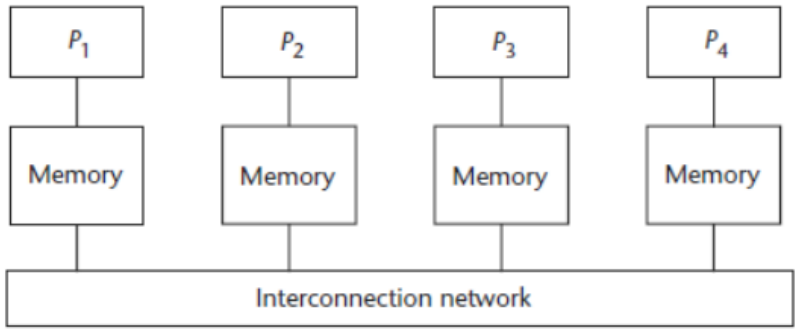
Parallel and Distributed Systems

- Parallel: hardware with just one memory space
- Distributed: multiple memories
- It is possible to run distributed programs in parallel systems and vice-versa

Shared memory

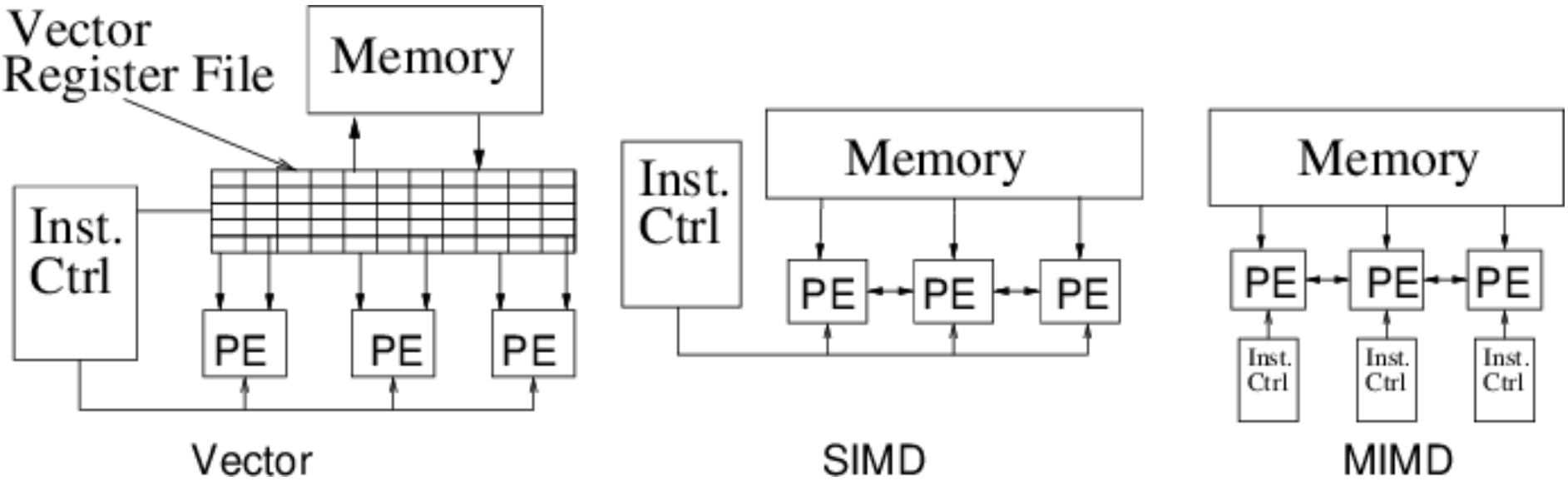


Distributed memory



Parallel Architectures

Most common: **SIMD** and **MIMD**



https://www.researchgate.net/publication/4049051_Universal_mechanisms_for_data-parallel_architectures/figures?lo=1

Languages, Compilers and Libraries

- Languages: special syntax, side effects and implicit context, type verification, threads, exception handling etc
- Compilers: makes the programming model simpler
- Library: easy to modify, use with existing languages, use with several languages

Performance Metrics

Amdahl's Law:

- Speedup $s = \frac{T(1)}{T(p)}$
- Total work $c = T_s + T_p = T(1)$
- $T(p) = T_s + \frac{T_p}{p}$
- $s = \frac{(T_s+T_p)}{(T_s+\frac{T_p}{p})} =$
  $= \frac{c}{(T_s+\frac{T_p}{p})} \rightarrow \frac{c}{T_s}$ when $p \rightarrow$ inf

Design and Verification of parallel programs

- Important: guarantee liveness and safety
  → Liveness: good things eventually happen
  → Safety: bad things never happen!
- Examples of liveness: no process waits forever, the program terminates
- Examples of safety: mutual exclusion, no buffer overflow

Most common parallel programming models

- Sequential
- Shared memory
- Message passing
- SPMD vs. MPMD or data parallelism vs. task parallelism

Other programming models

- Linda
- Actors
- Dataflow
- Logic
- Functional
- Constraints

Sequential programming model

- The simplest of all...
- Parallelism implemented by the compiler or by the system

```
for i = 1 to N
    a[i] = 1
```

- e.g.: HPF and other Fortran versions (compiler); some declarative languages (runtime)

Shared memory model

- More complex, but close to sequential
- Parallelism implemented by the programmer with constructs and calls to functions provided by a language or by the system software
- Synchronization is needed
- Transparent communication (implemented in sw or hw)
- e.g: C#, Java (language), OpenMP (runtime, library), PFS (file system, OS)

Shared memory model

```
doall i = 1 to N
    a[i] = 1

for j = 1 to NPROCS-1
    fork(compute,j)
compute(0)

lock(mutex)
    x = x + 1
unlock(mutex)
```