Chapter 18 - Multicore Computers

Luis Tarrataca
luis.tarrataca@gmail.com

CEFET-RJ

Table of Contents I

Table of Contents I

Motivation

Up until now we have seen several methods for increasing performance.

Can you name a few?

- **Frequency**
  - Easy way out =)
  - Inherent vantages and disadvantages. Can you name them?
- **Cache**
  - High speed memory;
  - Idea: diminish the number of reads from low latency memory (RAM, HD, etc);
  - Very expensive!
- **Pipeline**;
  - Idea: assembly line for instructions;
  - Leads to performance increases;
  - But also introduces a host of new considerations (hazards);

- **Parallel Computation**
  - SMP;
  - Clusters;
  - Vector Computation:
    - GPGPU;
    - Intel XEON Phi.

Today we will look at another form of parallel computation:

- multicore systems
- Combines two or more processors (cores) on a single piece of silicon.
- Each core is an independent processor:
  - Registers;
  - ALU;
  - Pipeline;

The obvious question to pose is why multicore systems?

The obvious question to pose is why multicore systems?

- Pipelines have hazards and resource dependencies;
- Using multiple threads over a set of pipelines also introduces complexities;
- Larger coordinating complexity and signal transfer logic:
  - Increases difficulty of designing, fabricating, and debugging the chips.
  - Also increases power consumption...

Amdahl's law states that:

$$\text{Speed-up} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{1}{(1-f) + \frac{f}{N}}$$

- f fraction that involves code that is infinitely parallelizable with no scheduling overhead.
- $(1-f)$ fraction that involves code that is inherently sequential.

Lets look at what happens with different values of $(1 - f)$:

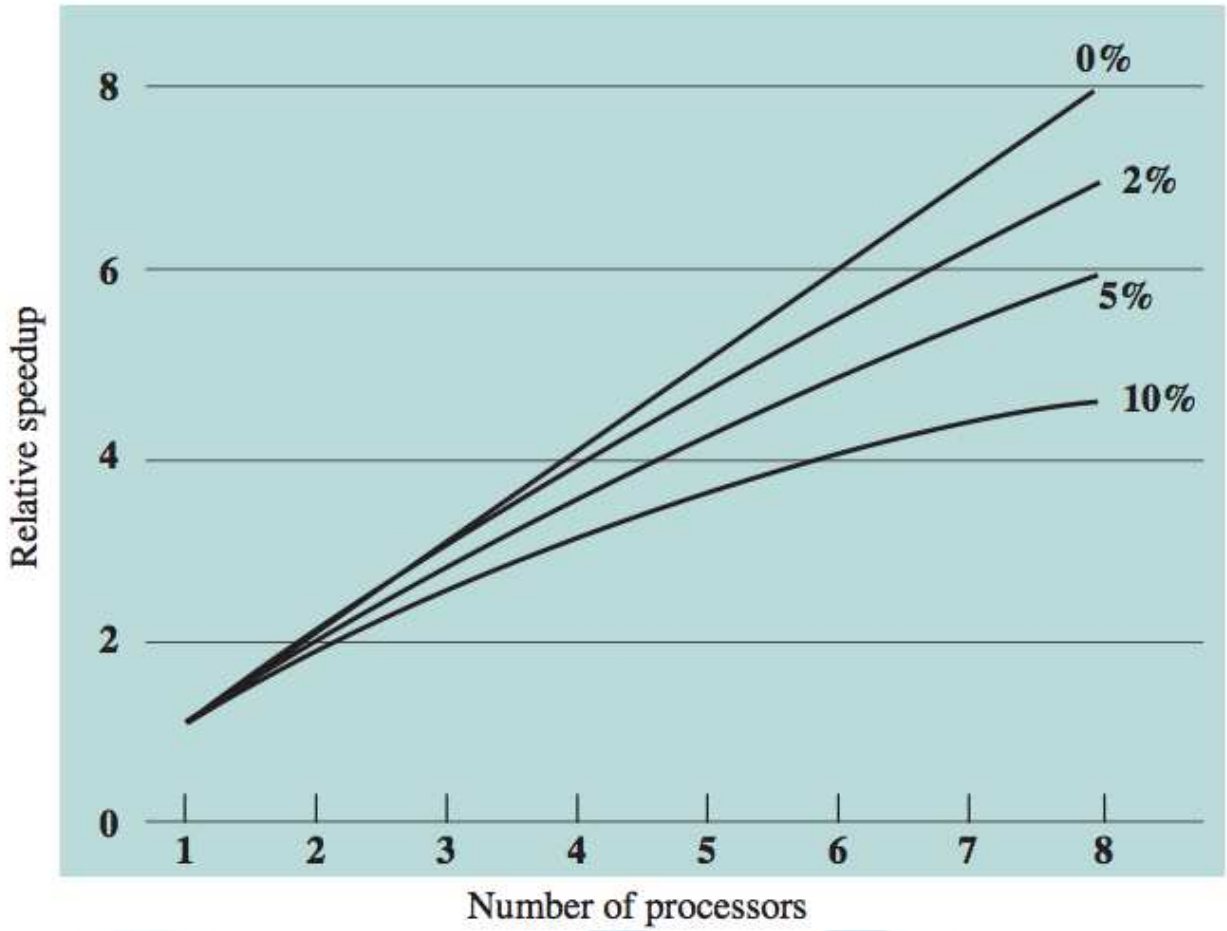

Figure: Speedup with 0%, 2%, 5%, and 10% sequential portions (Source: (Stallings, 2015))

Conclusion: Small amount of serial code has a noticeable impact!

But in **real** computers we have:

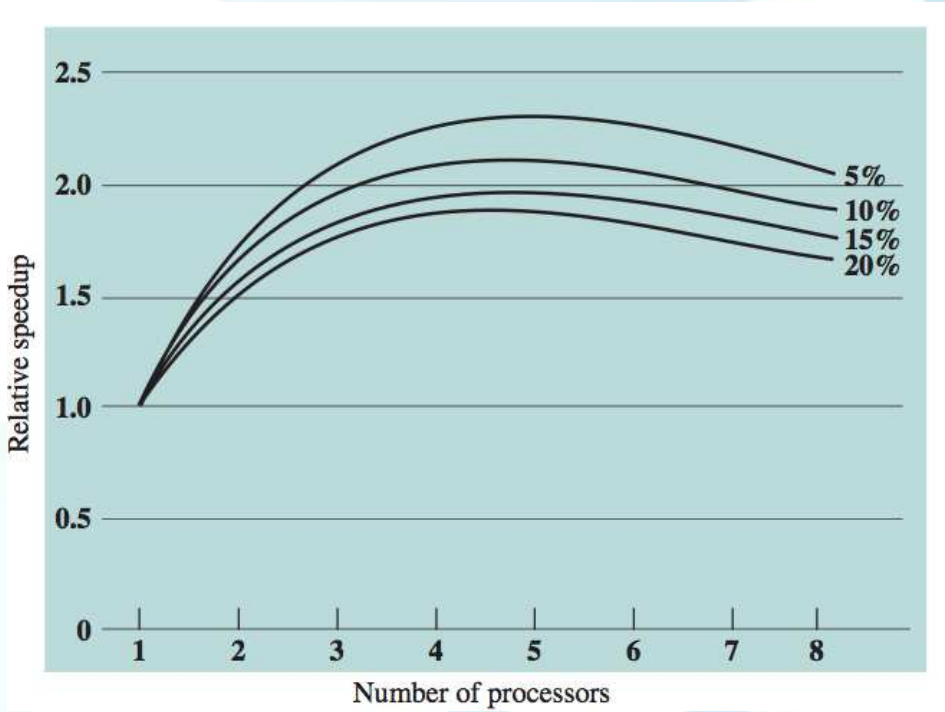

Figure: Speedup with 0%, 2%, 5%, and 10% sequential portions (Source: (Stallings, 2015))

- Performance peaks then starts losing performance due to these variables.

Why do you think this performance peak happens? Any ideas?

Why do you think this performance peak happens? Any ideas?

Multiple processors result in **overheads** required to;

- Communicate;
- Distribution work;
- Cache coherence overhead.

So what type of applications work well with multicore systems? Any ideas?

So what type of applications work well with multicore systems? Any ideas?

Applications:

- where $f \approx 1$;
- where overheads are smaller;
- DBMS are particularly well suited.
- Still not perfect...

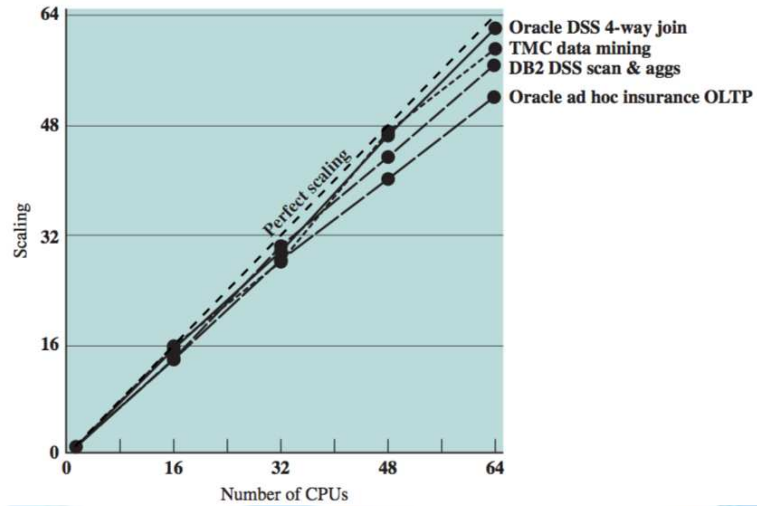

Figure: Scaling of Database Workloads on Multiple-Processor Hardware (Source: (Stallings, 2015))

Some of the main variables in a multicore organization are as follows:

- Number of core processors on the chip;
- Number of levels of cache memory;
- Amount of cache memory that is shared.

Lets look at some architectures (1/2):

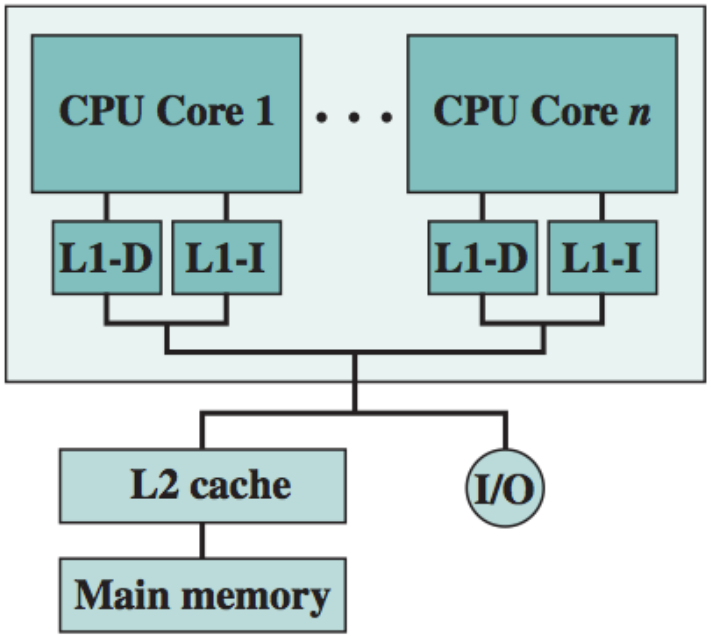

Figure: Dedicated L1 cache - Ex: ARM11 MPCore (Source: (Stallings, 2015))

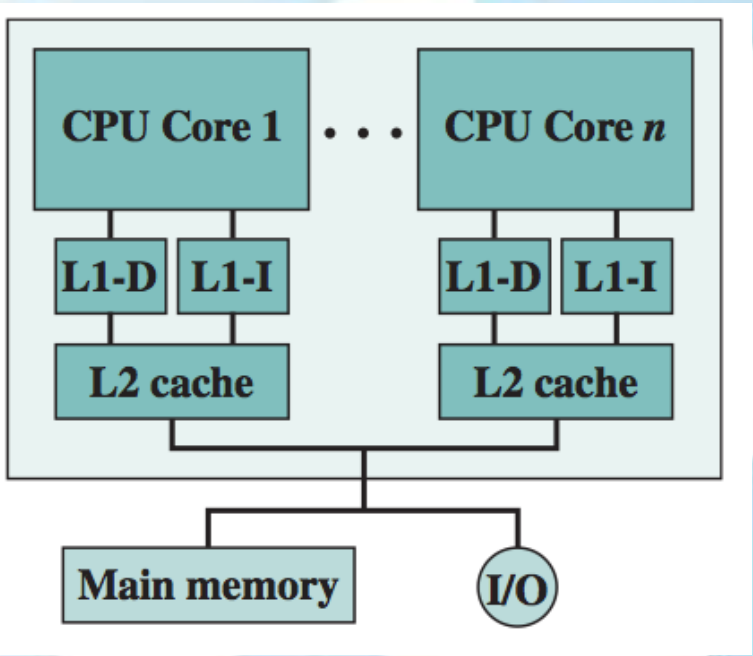

Figure: Dedicated L2 cache - Ex: AMD Opteron (Source: (Stallings, 2015))

- L1-D data cache;
- L1-I instruction cache;

Lets look at some architectures (2/2):

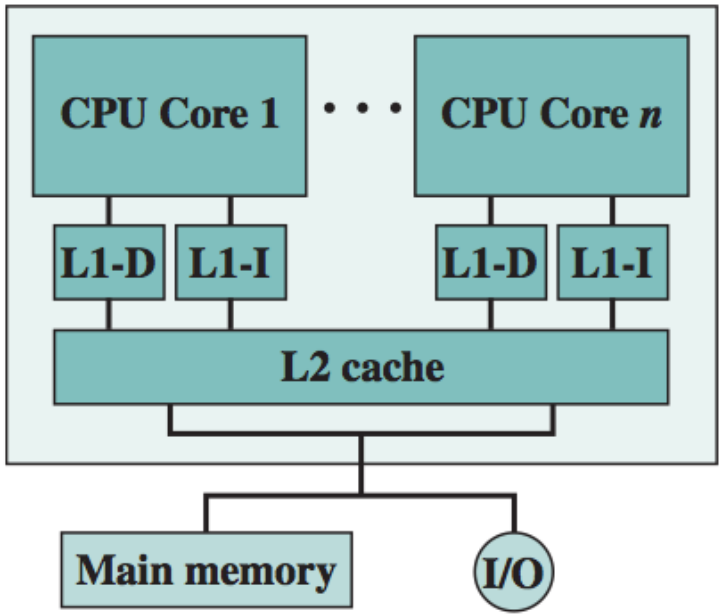

Figure: Shared L2 cache - Ex: Intel Core Duo
(Source: (Stallings, 2015))

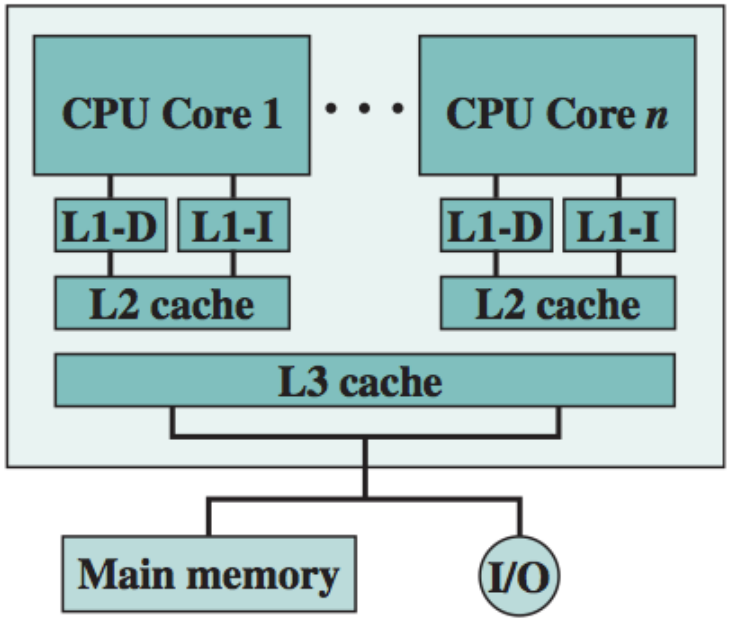

Figure: Shared L3 cache - Ex: Intel Core i7
(Source: (Stallings, 2015))

- L1-D data cache;
- L1-I instruction cache;

Why is the use of a shared cache good? Any ideas?

Why is the use of a shared cache good? Any ideas?

1. **Constructive Interference:**

   One core may cache a line that will later be used by another core;

2. **Cache Coherence:**

   No need to guarantee cache coherence since there is a single cache;

3. **Dynamic Cache Allocation:**

   Amount of shared cache allocated to each core is dynamic;

4. **Intercore communication:**

   Easy to implement with a shared memory;

Why is the use of an hierachical cache good?

Why is the use of an hierachical cache good?

- Cache latency vs. hit rate Tradeoff:
  - Larger caches have better hit rates but longer latency;
- Cost vs Space Tradeoff:
  - Faster caches are more expensive and have less space;
  - Slower caches are cheaper and have more space;

To address these **tradeoffs** computers use multiple cache levels:

- Small fast caches backed up by larger slower caches;
- Check if data is on the first levels:
  - **Cache hit:**

    Core proceeds at high speed;
  - **Cache miss:**

    Try next cache levels before accessing external memory.

As a curiosity lets take a look at the AMD Bulldozer memory hierarchy:

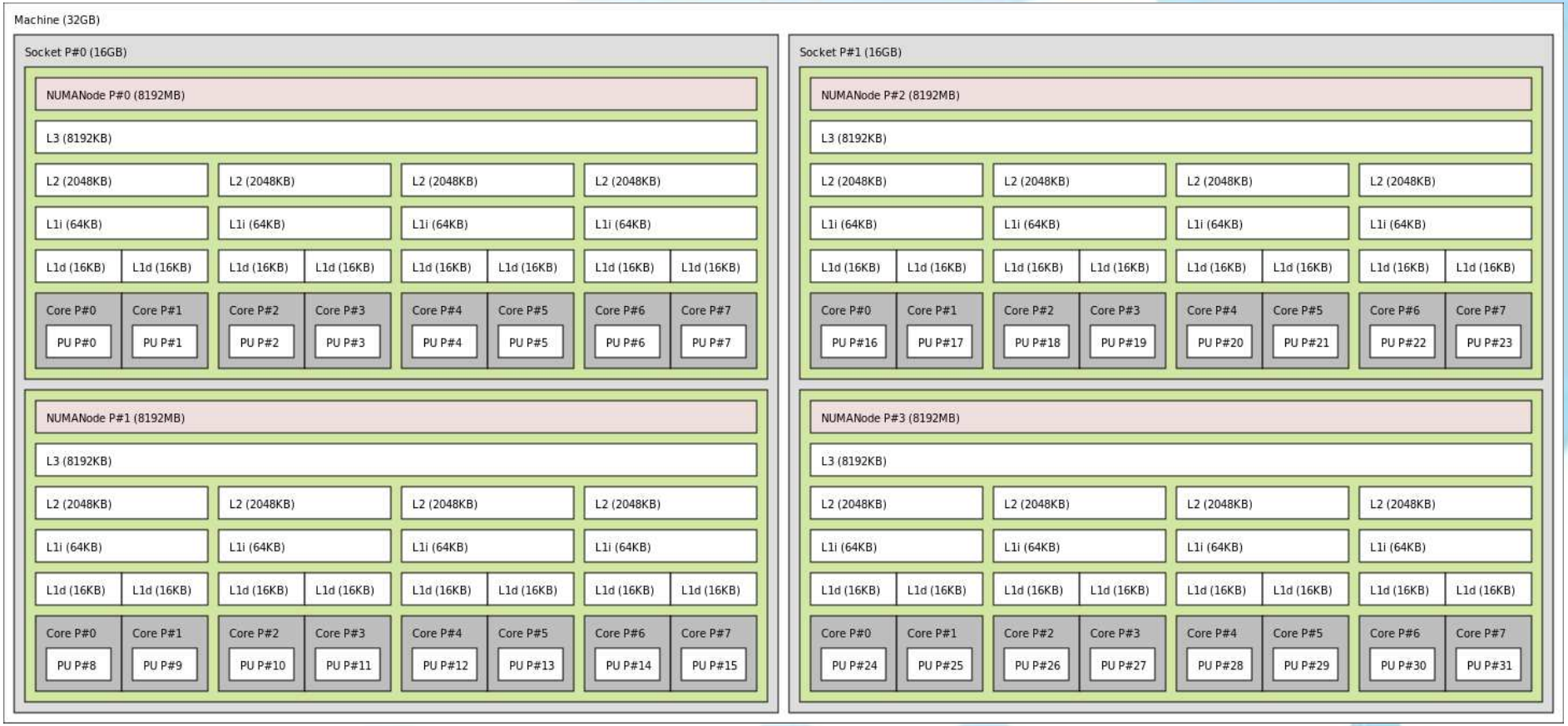

Figure: Memory hierarchy of an AMD Bulldozer server. (Source: Wikipedia)

What will the future be like?

What will the future be like?

Figure: Intel's 12^{th} generation processors (Source: Wikipedia)

Where to focus your study

After this class you should be able to:

- Understand the hardware performance issues that have driven the move to multicore computers;
- Understand the software performance issues posed by the use of multithreaded multicore computers.

Less important to know how these solutions were implemented:

- details of specific hardware solutions.

Your focus should always be on the building blocks for developing a solution =)

References I

Stallings, W. (2015).

Computer Organization and Architecture: Designing for Performance.

Pearson Education, 10th edition edition.