COMPUTER ORGANIZATION AND ARCHITECTURE (COA)

EET 2211
4TH SEMESTER – CSE & CSIT
CHAPTER 18, LECTURE 36
By Ms. Arya Tripathy

MULTICORE COMPUTERS

(Already covered, just go thro')

TOPICS TO BE COVERED

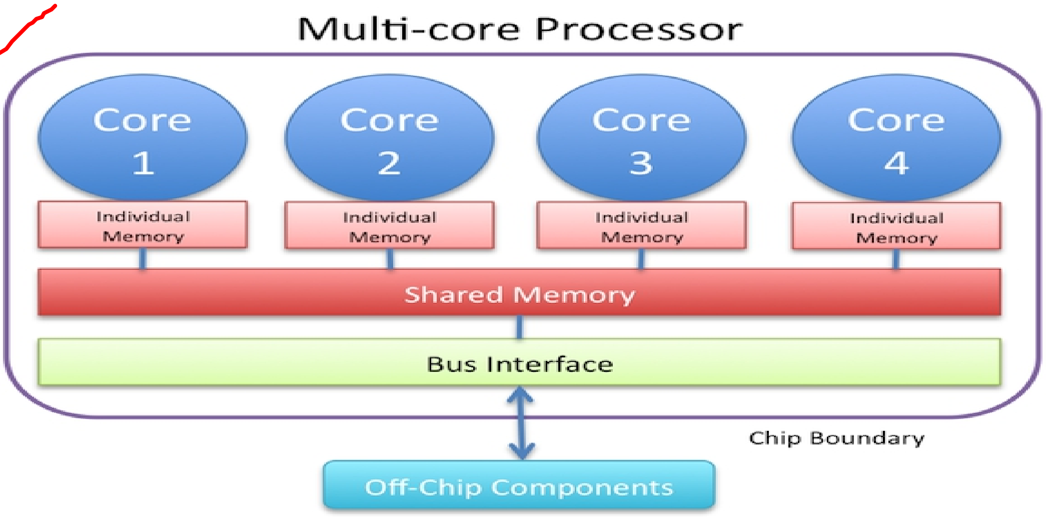
- HARDWARE PERFORMANCE ISSUES
1. Increase in Parallelism and Complexity
2. Power Consumption
- SOFTWARE PERFORMANCE ISSUES
1. Software on Multicore
- MULTICORE ORGANIZATION
1. Levels of Cache
2. Simultaneous Multithreading
- HETEROGENEOUS MULTICORE ORGANIZATION
1. Different Instruction Set Architectures
2. Equivalent Instruction Set Architecture

LEARNING OBJECTIVES

- 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.
- Present an overview of the two principal approaches to **heterogeneous multicore organization.**

MULTICORE PROCESSOR

- A **multicore processor**, also known as a **chip multiprocessor**, combines two or more processor units (called cores) on a single piece of silicon (called a die).
- Typically, each core consists of all of the components of an independent processor, such as registers, ALU, pipeline hardware, and control unit, plus L1 instruction and data caches.
- In addition to multiple cores , contemporary multicore chips also includes L2 cache and L3 cache also.
- The most highly integrated multicore processors, known as systems on chip (SoCs), also include memory and peripheral controllers.



HARDWARE PERFORMANCE ISSUES

- Microprocessor systems have experienced a steady increase in execution performance for decades. This increase is due to a number of factors, including increase in clock frequency, increase in transistor density, and refinements in the organization of the processor on the chip. All this leads to increase in complexity of the chip.

- 1st hardware performance issue is **INCREASE IN PARALLELISM AND COMPLEXITY**

- The organizational changes in processor design have primarily been focused on exploiting ILP, so that more work is done in each clock cycle. These changes include, in chronological order:

1. **Pipelining:** Individual instructions are executed through a pipeline of stages so that while one instruction is executing in one stage of the pipeline, another instruction is executing in another stage of the pipeline.

2. **Superscalar:** Multiple pipelines are constructed by replicating execution resources. This enables parallel execution of instructions in parallel pipelines, so long as hazards are avoided.

3. Simultaneous multithreading (SMT): Register banks are expanded so that multiple threads (thread: is the smallest sequence of programmed instructions that can be managed independently by a scheduler, where scheduling is the method by which work is assigned to resources that complete the work) can share the use of pipeline resources.

❖With each of these innovations, designers have over the years attempted to increase the performance of the system by adding complexity.

❖In the case of pipelining, for example, simple three-stage pipelines were replaced by pipelines with five stages.

❖There is a practical limit to how far this trend can be taken, because with more stages, there is the need for more logic, more interconnections, and more control signals.

❖Similarly, with superscalar organization, increased performance can be achieved by increasing the number of parallel pipelines.

❖Again, there are diminishing returns as the number of pipelines increases.

❖More logic is required to manage hazards and to stage instruction resources.

❖This same point of diminishing returns is reached with SMT, as the complexity of managing multiple threads over a set of pipelines limits the number of threads and number of pipelines that can be effectively utilized.

❖The increase in complexity to deal with all of the logical issues related to very long pipelines, multiple superscalar pipelines, and multiple SMT register banks means that increasing amounts of the chip area are occupied with coordinating and signal transfer logic.

❖This increases the difficulty of designing, fabricating, and debugging the chips.

❖In general terms, the experience of recent decades has been encapsulated in a rule of thumb known as **Pollack's rule,** which states that performance increase is roughly proportional to square root of increase in complexity.

❖In other words, if you double the logic in a processor core, then it delivers only 40% more performance.

❖In principle, the use of multiple cores has the potential to provide near-linear performance improvement with the increase in the number of cores—but only for software that can take advantage.

❖2nd hardware performance issue is **POWER CONSUMPTION**

✓To maintain the trend of higher performance as the number of transistors per chip rises, designers have resorted to more elaborate processor designs (pipelining, superscalar, SMT) and to high clock frequencies.

✓Unfortunately, power requirements have grown exponentially as chip density and clock frequency have risen.

✓One way to control power density is to use more of the chip area for cache memory.

✓Memory transistors are smaller and have a power density an order of magnitude lower than that of logic.

✓Power considerations provide another motive for moving toward a multicore organization. Because the chip has such a huge amount of cache memory, it becomes unlikely that any one thread of execution can effectively use all that memory.

✓Even with SMT, multithreading is done in a relatively limited fashion and cannot therefore fully exploit a gigantic cache, whereas a number of relatively independent threads or processes has a greater opportunity to take full advantage of the cache memory.

SOFTWARE PERFORMANCE ISSUES

❖The potential performance benefits of a multicore organization depend on the ability to effectively exploit the parallel resources available to the application.

❖Let us focus first on a single application running on a multicore system.

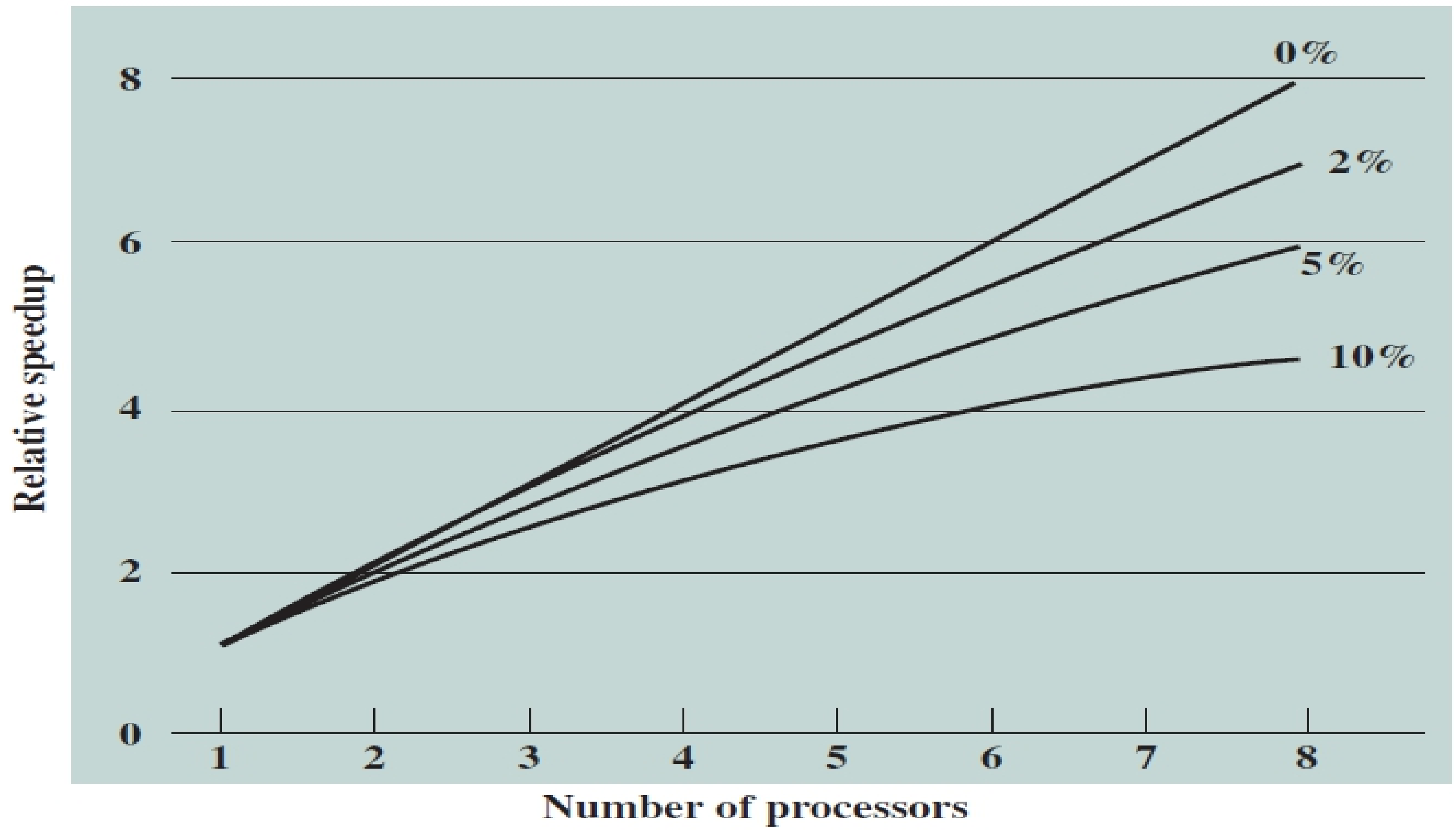
❖Amdahl's law states that:

❖Speed up $= \dfrac{\text{time to execute program on a single processor}}{\text{time to execute program on N parallel processors}}$

❖This law appears to make the prospect of a multicore organization attractive.

❖But as Figure (a) on the next slide shows, even a small amount of serial code has a noticeable impact.

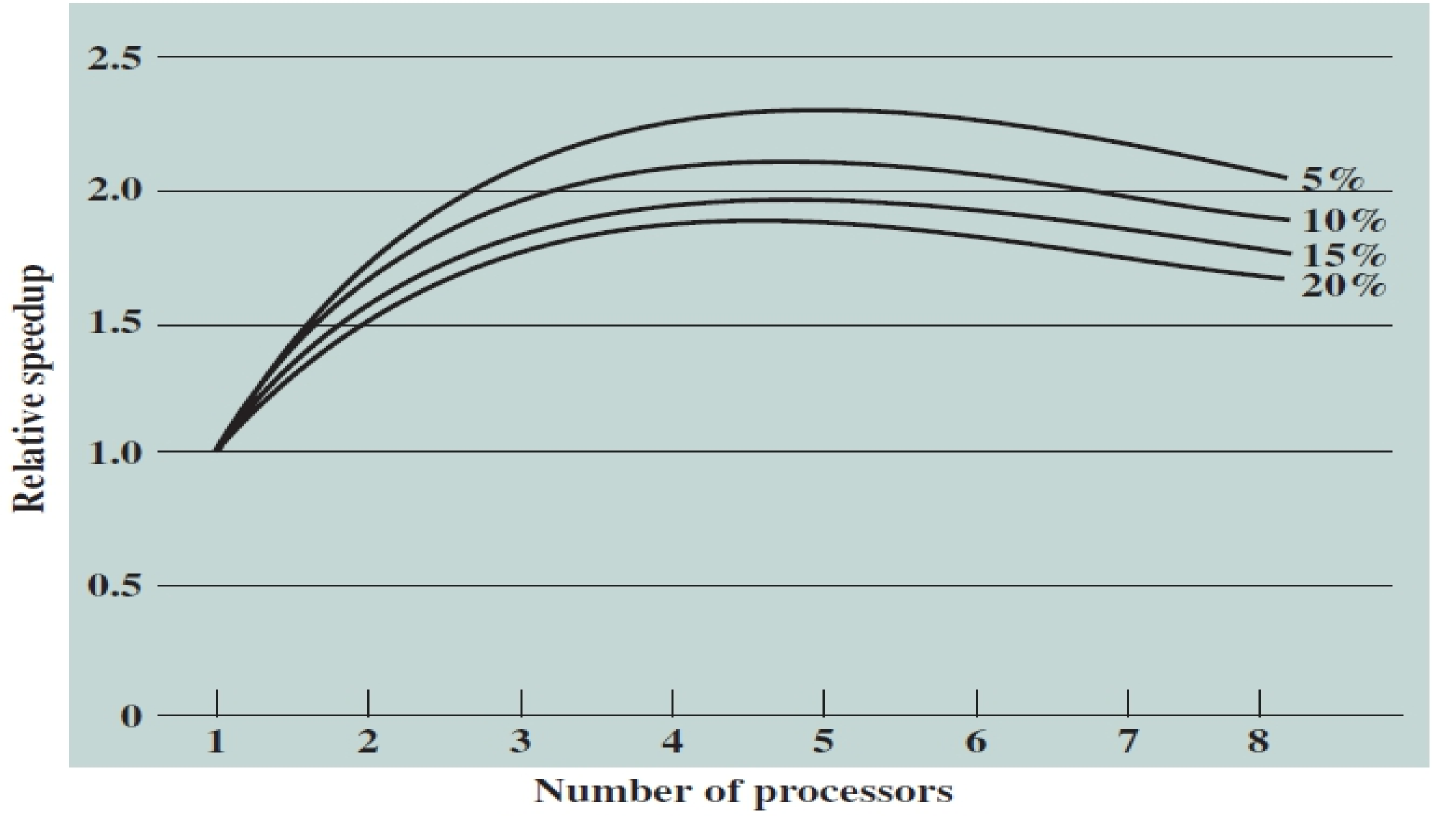
❖If only 10% of the code is inherently serial, running the program on a multicore system with eight processors yields a performance gain of only a factor of 4.7.



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions

It shows even a small amount of serial code has a noticeable impact. If only 10% of the code is inherently serial (f=0.9), running the program on a multicore system with eight processors yields a performance gain of only a factor of 4.7.

In addition, software typically incurs overhead as a result of communication and distribution of work among multiple processors and as a result of cache coherence overhead. This overhead results in a curve where performance peaks and then begins to degrade because of the increased burden of the overhead of using multiple processors (e.g., coordination and OS management) as shown in Figure (b) below.



(b) Speedup with overheads

Contd.

❖However, software engineers have been addressing this problem and there are numerous applications in which it is possible to effectively exploit a multicore system.

❖Database management systems and database applications are one area in which multicore systems can be used effectively.

❖Many kinds of servers can also effectively use the parallel multicore organization, because servers typically handle numerous relatively independent transactions in parallel.

❖In addition to general-purpose server software, a number of classes of applications benefit directly from the ability to scale throughput with the number of cores.

Contd.

- Some of these include the following:

1. Multithreaded native applications (thread-level parallelism) : Multithreaded applications are characterized by having a small number of highly threaded processes.
2. Multiprocess applications (process-level parallelism) : Multiprocess applications are characterized by the presence of many single-threaded processes.
3. Java applications : Java applications embrace threading in a fundamental way.
4. Multi-instance applications (application-level parallelism) : even if an individual application does not scale to take advantage of a large number of threads, it is still possible to gain from multicore architecture by running multiple instances of applications in parallel.

Thank you!