Computer Organization and Architecture (EET 2211)

Chapter-2

Lecture 01

Chapter 2

Performance Issues

Learning Objectives

After studying this chapter, you should be able to:

- Understand the key performance issues that relate to computer design.
- Explain the reasons for the move to multicore organization, and understand the trade-off between cache and processor resources on a single chip.
- Distinguish among multicore, MIC, and GPGPU organizations.
- Summarize some of the issues in computer performance assessment.
- Discuss the SPEC benchmarks.
- Explain the differences among arithmetic, harmonic, and geometric means.

Designing for Performance

- Year by year, the cost of computer systems continues to drop dramatically, while the performance and capacity of those systems continue to rise equally dramatically.

- What is fascinating about all this from the perspective of computer organization and architecture is that, on the one hand, the basic building blocks for today's computer miracles are virtually the same as those of the IAS computer from over 50 years ago, while on the other hand, the techniques for squeezing the maximum performance out of the materials at hand have become increasingly sophisticated.

- Here in this section, we highlight some of the driving factors behind the need to design for performance.

- **Microprocessor Speed :**The evolution of these machines continues to bear out Moore's law, as described in Chapter 1.

  - **Pipelining:**
  - **Branch prediction:**
  - **Superscalar execution:**
  - **Data flow analysis:**
  - **Speculative execution:**

- **Performance Balance:**

  While processor power has raced ahead at breakneck speed, other critical components of the Computer have not kept up. The result is a need to look for performance balance: an adjustment/tuning of the organization and architecture to compensate for the mismatch among the capabilities of the various components.

- The problem created by such mismatches is particularly critical at the interface between processor and main memory.

- If memory or the pathway fails to keep pace with the processor's insistent demands, the processor stalls in a wait state, and valuable processing time is lost.

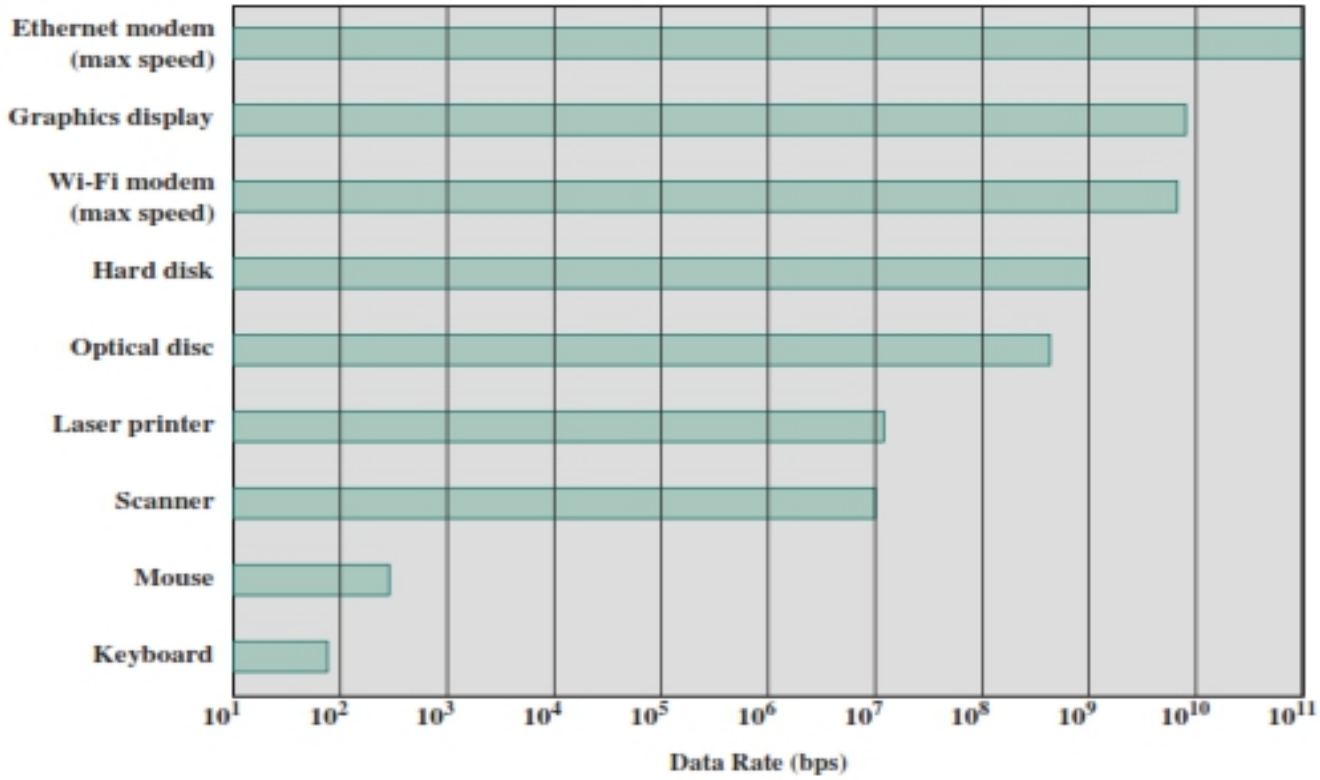
A system architect can attack this problem in a number of ways, all of which are reflected in contemporary computer designs. Consider the following examples:

- Increase the number of bits that are retrieved at one time by making DRAMs “wider” rather than “deeper” and by using wide bus data paths.
- Change the DRAM interface to make it more efficient by including a cache or other buffering scheme on the DRAM chip.

- Reduce the frequency of memory access by incorporating increasingly complex and efficient cache structures between the processor and main memory.

- Increase the interconnect bandwidth between processors and memory by using higher-speed buses and a hierarchy of buses to buffer and structure data flow.

Another area of design focus is the handling of I/O devices. As computers become faster and more capable, more sophisticated applications are developed that support the use of peripherals with intensive I/O demands.



Typical I/O Device Data Rates

- The key in all this is balance. This design must constantly be rethought to cope with two constantly evolving factors:

(i) The rate at which performance is changing in the various technology areas (processor, buses, memory, peripherals) differs greatly from one type of element to another.

(ii) New applications and new peripheral devices constantly change the nature of the demand on the system in terms of typical instruction profile and the data access patterns.

- **Improvements in Chip Organization and Architecture**:

As designers wrestle with the challenge of balancing processor performance with that of main memory and other computer components, the need to increase processor speed remains. There are three approaches to achieving increased processor speed:

(i) Increase the hardware speed of the processor

(ii) Increase the size and speed of caches

(iii) Increase the effective speed of instruction execution

- Traditionally, the dominant factor in performance gains has been in increases in clock speed due and logic density. However, as clock speed and logic density increase, a number of obstacles become more significant [INTE04]:

Power: As the density of logic and the clock speed on a chip increase, so does the power density (Watts/cm^2).

RC delay: The speed at which electrons can flow on a chip between transistors is limited by the resistance and capacitance of the metal wires connecting them; specifically, delay increases as the RC product increases.

Memory latency and throughput: Memory access speed (latency) and transfer speed (throughput) lag processor speeds, as previously discussed.

MULTICORE,MICS,GPGPUS

- With all of the difficulties cited in the preceding section in mind, designers have turned to a fundamentally new approach to improving performance: placing multiple processors on the same chip, with a large shared cache. The use of multiple processors on the same chip, also referred to as multiple cores or **multicore,** provides the potential to increase performance without increasing the clock rate.

- Chip manufacturers are now in the process of making a huge leap forward in the number of cores per chip, with more than 50 cores per chip. The leap in performance as well as the challenges in developing software to exploit such a large number of cores has led to the introduction of a new term: **many integrated core (MIC).**

- The multicore and MIC strategy involves a homogeneous collection of general purpose processors on a single chip. At the same time, chip manufacturers are pursuing another design option: a chip with multiple general-purpose processors plus **graphics processing units (GPUs)** and specialized cores for video processing and other tasks.

- The line between the GPU and the CPU [AROR12, FATA08, PROP11]. When a broad range of applications are supported by such a processor, the term **general-purpose computing on GPUs (GPGPU)** is used**.**

Amdahl's Law & Little's Law

- **Amdahl's Law**

- Amdahl's law was first proposed by Gene Amdahl in 1967 ([AMDA67], [AMDA13]) and deals with the potential speedup of a program using multiple processors compared to a single processor.

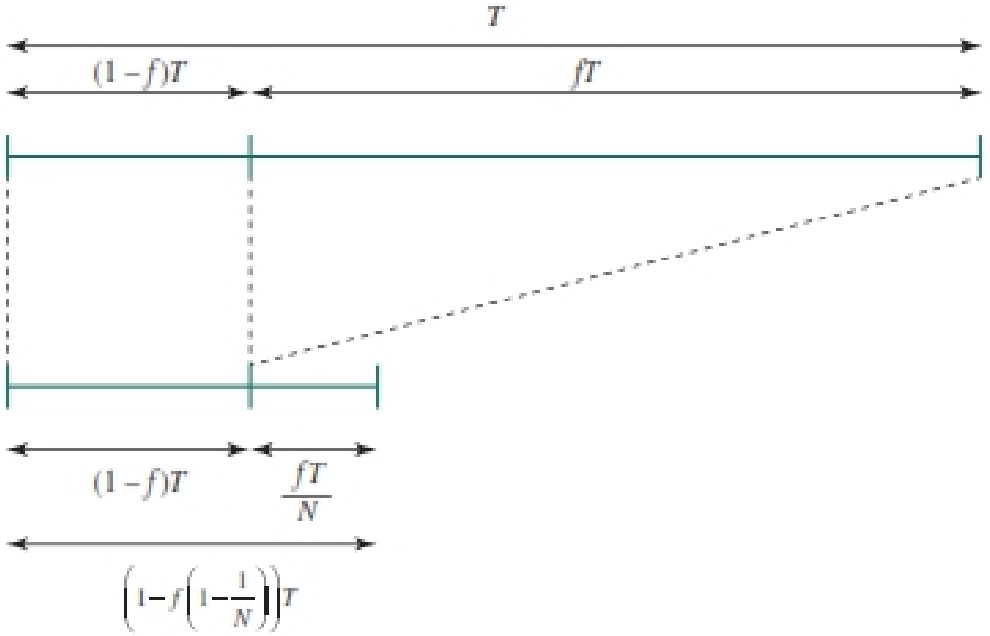


Illustration of Amdahl's Law

$$\text{Speedup} = \frac{\text{Time to execute program on a single processor}}{\text{Time to execute program on } N \text{ parallel processors}}$$

$$= \frac{T(1-f) + Tf}{T(1-f) + \frac{Tf}{N}} = \frac{1}{(1-f) + \frac{f}{N}}$$

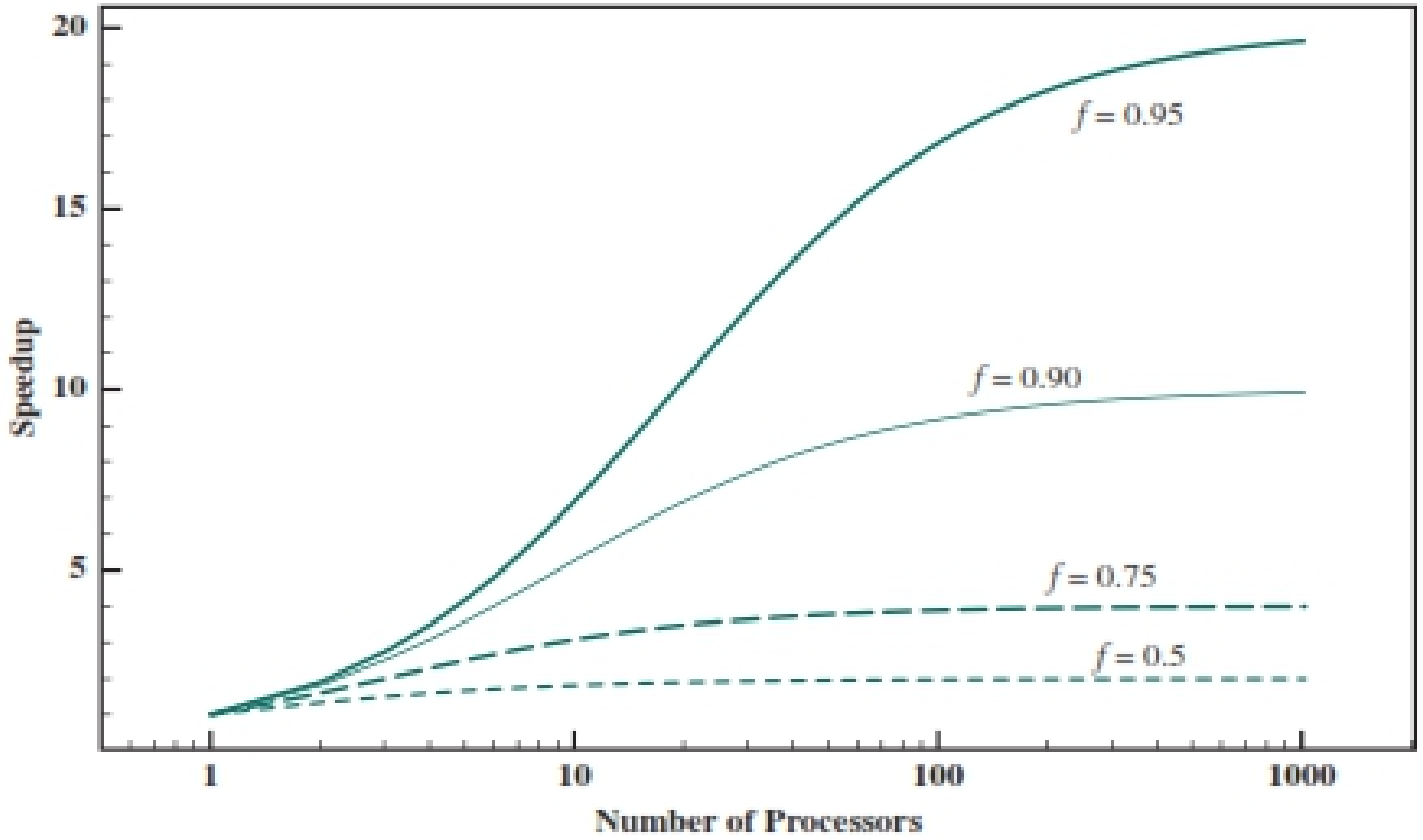
From this equation two important conclusions can be drawn:

1. When f is small, the use of parallel processors has little effect.

2. As N approaches infinity, speedup is bound by 1/ (1 - f), so that there are diminishing returns for using more processors.

- Amdahl's law can be generalized to evaluate any design or technical improvement in a computer system. Consider any enhancement to a feature of a system that results in a speedup. The speedup can be expressed as

$$\text{Speedup} = \frac{\text{Performance after enhancement}}{\text{Performance before enhancement}} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}}$$



Amdahl's Law for Multiprocessors

Suppose that a feature of the system is used during execution a fraction of the time *f, before enhancement, and that the speedup of that feature after enhancement* is SU_f. Then the overall speedup of the system is

$$\text{Speedup} = \frac{1}{(1-f) + \frac{f}{SU_f}}$$

EXAMPLE 2.1 Suppose that a task makes extensive use of floating-point operations, with 40% of the time consumed by floating-point operations. With a new hardware design, the floating-point module is sped up by a factor of K. Then the overall speedup is as follows:

$$\text{Speedup} = \frac{1}{0.6 + \frac{0.4}{K}}$$

Thus, independent of K, the maximum speedup is 1.67.

- **Little's Law**
- A fundamental and simple relation with broad applications is Little's Law [LITT61,LITT11]. We can apply it to almost any system that is statistically in steady state, and in which there is no leakage.
- we have a steady state system to which items arrive at an average rate of λ items per unit time. The items stay in the system an average of W units of time. Finally, there is an average of L units in the system at any one time.

Little's Law relates these three variables as $L = \lambda W$

- To summarize, under steady state conditions, the average number of items in a queuing system equals the average rate at which items arrive multiplied by the average time that an item spends in the system.

- Consider a multicore system, with each core supporting multiple threads of execution. At some level, the cores share a common memory. The cores share a common main memory and typically share a common cache memory as well.

- For this purpose, each user request is broken down into subtasks that are implemented as threads. We then have λ = the average rate of total thread processing required after all members' requests have been broken down into whatever detailed subtasks are required. Define *L* as the average number of stopped threads waiting during some relevant time. Then *W*= average response time.