# Introduction

In this lesson, we will look at different type of servers and see different architectures and how they take advantage of parallelism.

# Servers

As the shift to desktop computing occurred in the 1980s, the role of servers grew to provide larger-scale and more reliable file and computing services. Such servers have become the backbone of large-scale enterprise computing, replacing the traditional mainframe.

For servers, different characteristics are important. First, **availability is critical**. Consider the servers running ATM machines for banks or airline reservation systems. Failure of such server systems is far more catastrophic than failure of a single desktop, since these servers must **operate seven days a week, 24 hours a day**. Figure 1.3 estimates revenue costs of downtime for server applications.



**Figure 1.3** estimates revenue costs of downtime for server applications. Figure 1.3 Costs rounded to nearest $ 100,000 of an unavailable system are shown by analysing the cost of downtime (in terms of immediately lost revenue), assuming three different levels of availability and that downtime is distributed uniformly. These data are from Kembel [2000] and were collected and analysed by Contingency Planning Research.

A second key feature of server systems is **scalability**. Server systems often grow in response to an increasing demand for the services they support or an increase in functional requirements. Thus, the ability to scale up the computing capacity, the memory, the storage, and the I/ O bandwidth of a server is crucial.

Finally, servers are designed for **efficient throughput**. That is, the overall performance of the server— in terms **of transactions per minute or Web pages served per second**— is what is crucial. Responsiveness to an individual request remains important, **but overall efficiency and cost-effectiveness**, as determined by **how many requests can be handled in a unit time**, are the key metrics for most servers. We return to the issue of assessing performance for different types of computing environments in Section 1.8 .

**Home Work Question:** Could you please investigate energy efficiency and cooling requirements of Servers and Cluster systems and report on the steps taken to reduce their energy consumption? (Please Discuss in a small article).

# Clusters/ Warehouse-Scale Computers

The growth of Software as a Service (SaaS) for applications like search, social networking, video sharing, multiplayer games, online shopping, and so on has led to the growth of **a class of computers called clusters**. Clusters are collections of **desktop computers or servers connected by local area networks to act as a single larger computer**. Each node runs **its own operating system**, and nodes communicate using **a networking protocol**. The largest of the clusters are called **warehouse-scale** computers (WSCs), in that they are designed so that tens of thousands of servers can act as one.

**Price-performance and power** are critical to WSCs since they are so large. **80% of the cost of a $ 90M** warehouse is associated **with power and cooling of the computers inside**. The computers themselves and **networking gear cost another $ 70M and they must be replaced every few years**. When you are buying that much computing, you need to buy wisely, as a **10% improvement in price-performance means a savings of $ 7M** (10% of $ 70M).

WSCs **are related to servers**, in that availability is critical. For example, Amazon.com had $ 13 billion in sales in the fourth quarter of 2010. As there are about 2200 hours in a quarter, the average revenue per hour was almost $ 6M. During a peak hour for Christmas shopping, the potential loss would be many times higher. As Chapter 6 explains, the difference from servers is that WSCs use redundant inexpensive components as the building blocks, relying on a software layer to catch and isolate the many failures that will happen with computing at this scale.

**Supercomputers are related to WSCs** in that they are equally expensive, costing hundreds of millions of dollars**, but supercomputers differ by emphasizing floating-point performance and by running large, communication-intensive batch programs that can run for weeks at a time**. This tight coupling leads to use of much faster internal networks. In contrast, **WSCs emphasize interactive applications, large-scale storage, dependability, and high Internet bandwidth**.

# Embedded Computers

**Embedded computers are found in everyday machines; microwaves, washing machines, most printers, most networking switches, and all cars contain simple embedded microprocessors.**

The processors in a PMD (Personal Mobile Device) are often considered embedded computers, but we are keeping them as a separate category because PMDs are platforms that can run externally developed software and they share many of the characteristics of desktop computers. Other embedded devices are more limited in hardware and software sophistication. We use the ability to run **third-party software as the dividing line between non-embedded and embedded** computers.

Embedded computers have the widest spread of processing power and cost. They **include 8 -bit and 16-bit processors that may cost less than a dime**, **32-bit microprocessors that execute 100 million instructions per second and cost under $ 5**, and **high-end processors for network switches that cost $ 100 and can execute billions of instructions per second**. Although the range of computing power in the embedded computing market is very large**, price is a key factor in the design of computers for this space**. Performance requirements do exist, of course, but the primary goal is often meeting the performance need at a minimum price, rather than achieving higher performance at a higher price.

# Classes of Parallelism and Parallel Architectures

**Parallelism at multiple levels is now the driving force of computer design across all four classes of computers,** with energy and cost being the primary constraints. There are basically two kinds of parallelism in applications:

**1. Data-Level Parallelism (DLP)** arises because there are many data items that can be operated on at the same time.

**2. Task-Level Parallelism (TLP)** arises because tasks of work are created that can operate independently and largely in parallel.

Computer hardware in turn can exploit these two kinds of application parallelism in four major ways:

1. Instruction-Level Parallelism exploits data-level parallelism at modest levels with compiler help using ideas like pipelining and at medium levels using ideas like speculative execution. (**Example**?)

2. Vector Architectures and Graphic Processor Units (GPUs) exploit data-level parallelism by applying a single instruction to a collection of data in parallel (**Example**?).

3. Thread-Level Parallelism exploits either data-level parallelism or task-level parallelism in a tightly coupled hardware model that allows for interaction among parallel threads.(**Example**?)

4. Request-Level Parallelism exploits parallelism among largely decoupled tasks specified by the programmer or the operating system (**Example**?).

These four ways for hardware to support the data-level parallelism and task-level parallelism go back 50 years. When Michael Flynn [1966] studied the parallel computing efforts in the 1960s, he found a simple classification whose abbreviations we still use today. He looked at the parallelism in the instruction and data streams called for by the instructions at the most constrained component of the multiprocessor, and placed all computers into one of four categories:

**1. Single instruction stream, single data stream (SISD )—** This category is the uniprocessor . The programmer thinks of it as the standard sequential computer, but it can exploit instruction-level parallelism. Chapter 3 covers SISD architectures that use ILP techniques such as superscalar and speculative execution.

**2. Single instruction stream, multiple data streams (SIMD)—** The same instruction is executed by multiple processors using different data streams. SIMD computers exploit data-level parallelism by applying the same operations to multiple items of data in parallel. Each processor has its own data memory (hence the MD of SIMD), but there is a single instruction memory and control processor, which fetches and dispatches instructions.

**3. Multiple instruction streams, single data stream (MISD)—** No commercial multiprocessor of this type has been built to date, but it rounds out this simple classification.

**4. Multiple instruction streams, multiple data streams ( MIMD )—** Each processor fetches its own instructions and operates on its own data, and it targets task-level parallelism. In general, MIMD is more flexible than SIMD and thus more generally applicable, but it is inherently more expensive than SIMD. For example, MIMD computers can also exploit data-level parallelism , although the overhead is likely to be higher than would be seen in an SIMD computer. This overhead means that grain size must be sufficiently large to exploit the parallelism efficiently.

**Let’s talk about pipelined computing now as a way of increasing the speed at which we perform a well-defined task. An example could be a PicoCell base station for next generation wireless communication (5G).**

Hennessy, John L.; Patterson, David A. (2011-10-07). Computer Architecture: A Quantitative Approach (The Morgan Kaufmann Series in Computer Architecture and Design) (Kindle Locations 718-721). Elsevier Science. Kindle Edition.