NETWORK TOPOLOGIES

BUS-BASED NETWORKS

Bus-based interconnects are a common method of connecting multiple components within a computer system, such as CPUs, memory modules, and peripherals. The way these interconnects are designed can vary based on whether or not they incorporate local caches (memory) for the connected components. Let's explore both scenarios:

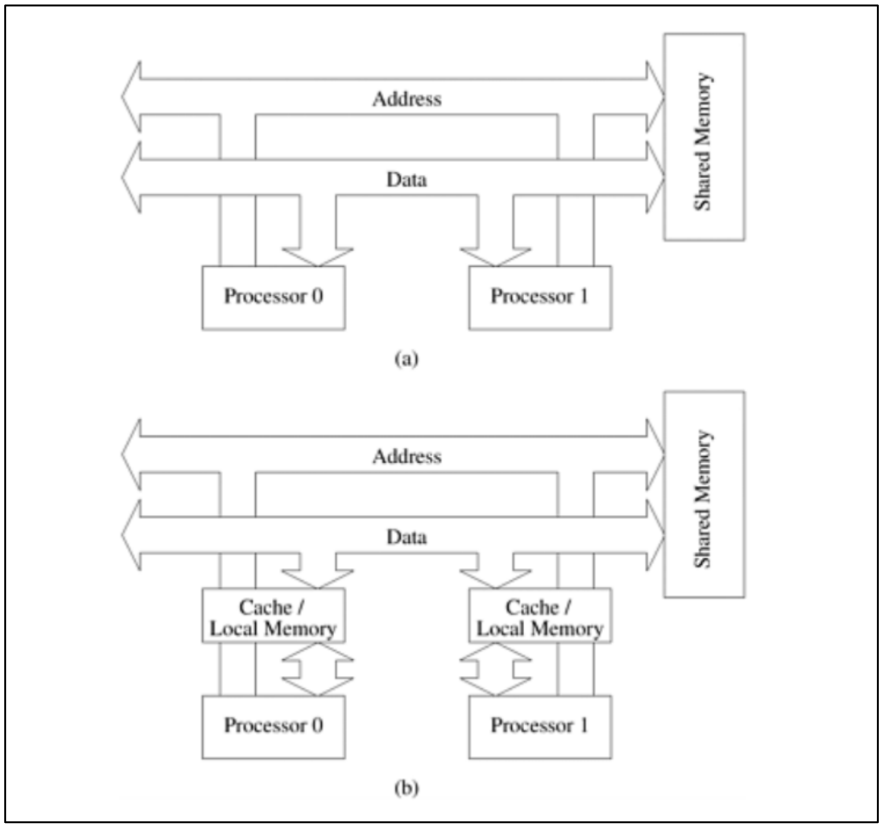


Figure 1 Bus-based interconnects (a) with no local caches;

a) Bus-Based Interconnects with No Local Caches:

In a bus-based interconnect without local caches, data is transferred directly between components without any intermediate storage. This means that components like CPUs, memory modules, and peripherals communicate with each other solely through the shared bus. Here's an example:

Consider a simple computer system with a CPU, main memory (RAM), and a hard drive connected via a bus-based interconnect. In this scenario, when the CPU needs to read data from the hard drive, it sends a request to the hard drive over the bus. The hard drive retrieves the data and sends it back to the CPU directly over the bus. There are no local caches involved in this process.

Advantages:

- Simple design: This architecture is straightforward and easy to implement.
- Low latency: Data transfer between components happens without any intermediate storage, resulting in lower latency.

Disadvantages:

- Limited performance: Without caches, the system may not efficiently exploit temporal and spatial data locality, leading to slower overall performance.
- Higher bus contention: Components must contend for access to the shared bus, potentially causing bottlenecks.

b) Bus-Based Interconnects with Local Memory/Caches:

In this scenario, each component (e.g., CPU, memory) has its own local cache memory. These caches store frequently used data, reducing the need to access main memory or other components over the bus. Here's an example:

Imagine a multi-core CPU with individual core caches, main memory (RAM), and a graphics card in a computer system. Each CPU core has its own cache, and the graphics card also has its memory. When a CPU core needs data, it first checks its local cache. If the data is present, it can be accessed much faster than if it had to retrieve it from the main memory or graphics card memory. If the data is not in the local cache, the CPU retrieves it from the main memory or graphics card memory via the bus.

Advantages:

- Improved performance: Caches help reduce the latency of memory accesses by storing frequently used data closer to the processing unit.
- Reduced bus traffic: With caches, components don't need to access the shared bus as frequently, reducing congestion and improving overall system performance.

Disadvantages:

- Complexity: Implementing caches adds complexity to the system design, including cache coherence mechanisms to ensure data consistency.
- Increased hardware cost: Local caches require additional hardware, which can increase the cost of the system.

Numerical Problem:

Let's use numerical values to illustrate the difference between bus-based interconnects with and without local caches. We'll assume a simple scenario with a CPU, main memory, and a hard drive connected via a bus-based interconnect.

(a) Without Local Caches:

In this scenario, there are no local caches, and the CPU directly accesses data from the hard drive. Let's assume:

- Time to read data from the hard drive (T_HD) = 100 milliseconds (0.1 seconds).

Now, if the CPU needs to read data from the hard drive, the total time it takes without local caches is simply T_HD:

- Total Time (No Caches) = T_HD = 0.1 seconds

(b) With Local Caches:

In this scenario, we introduce a local cache that can store frequently accessed data. We'll assume:

- Cache hit rate (HR) = 0.8 (80% of the time, data is found in the cache).
- Time to access data from the cache when there's a cache hit (T_Cache_Hit) = 10 milliseconds (0.01 seconds).
- Time to read data from the hard drive (T_HD) remains the same: 100 milliseconds (0.1 seconds).

Now, we can calculate the total time it takes to read data from the hard drive when caches are used:

Total Time (With Caches) = (HR * T_Cache_Hit) + ((1 - HR) * T_HD)

= (0.8 * 0.01 seconds) + (0.2 * 0.1 seconds)

= 0.008 seconds + 0.02 seconds

= 0.028 seconds

CROSSBAR NETWORK

Crossbar networks are a type of interconnection network topology used in computer architecture and high-performance computing systems. They provide a scalable and efficient way to connect multiple components, such as processors and memory modules. In a crossbar network, each input is directly connected to each output through a switching matrix, allowing for simultaneous communication between any input and output pairs. This architecture provides high bandwidth and low latency, making it suitable for systems with heavy communication demands. Let's explain crossbar networks with an example:

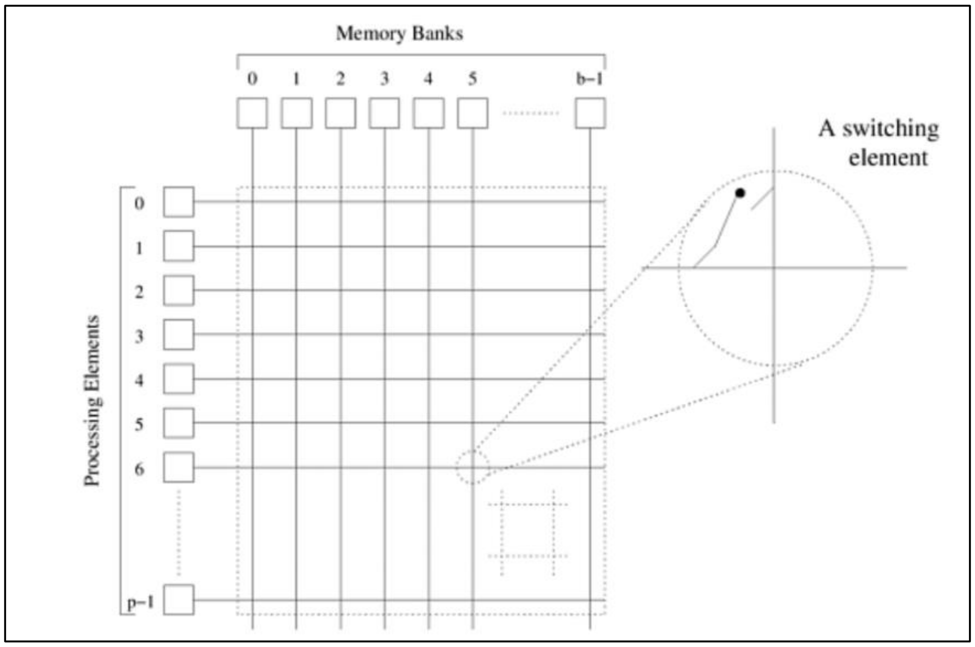
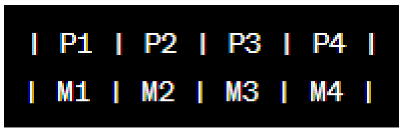


Figure 2 A completely non-blocking crossbar network connecting p

Example: Crossbar Network in a Multiprocessor System

Imagine a multiprocessor system with four processors (P1, P2, P3, and P4) and four memory modules (M1, M2, M3, and M4). In this system, we want to design a crossbar network to connect these processors to memory modules efficiently.

1. **Crossbar Network Configuration**:
   - In a crossbar network, every processor can communicate directly with every memory module simultaneously. This means that there are direct connections between each processor and each memory module, forming a grid-like structure.

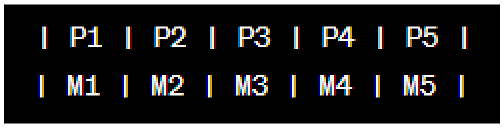


```
| P1 | P2 | P3 | P4 |
| M1 | M2 | M3 | M4 |
```

   - In this configuration, there are 4 processors (P1 to P4) and 4 memory modules (M1 to M4). Each processor has a direct connection to each memory module, and vice versa.

2. **Data Transfer:**
    - Let's say processor P1 needs to read data from memory M3. In a crossbar network, this communication can happen directly without any interference or contention because there's a dedicated connection between P1 and M3.
    - Similarly, if processor P2 needs to write data to memory M2, it can do so without affecting other communication channels.
    - This simultaneous and direct communication between processors and memory modules is a significant advantage of crossbar networks. It minimizes contention for shared resources and leads to low-latency, high-bandwidth data transfers.

3. **Scalability:**
    - Crossbar networks are highly scalable. If we add more processors or memory modules to the system, we can simply extend the grid to accommodate them. For example, if we add another processor (P5) and memory module (M5), we can easily modify the crossbar network as follows:



```
| P1 | P2 | P3 | P4 | P5 |
| M1 | M2 | M3 | M4 | M5 |
```

    - Now, P5 can communicate directly with all memory modules, and M5 can communicate directly with all processors.

Advantages:

- Low-latency communication: Direct connections ensure low latency, making crossbar networks suitable for applications that require quick data transfers.
- High bandwidth: Multiple simultaneous data transfers are possible, leading to high overall bandwidth.
- Scalability: Crossbar networks can easily accommodate additional components as the system grows.

Disadvantages:

- Complexity: As the number of components increases, the complexity of the switching matrix and routing algorithms can become challenging.

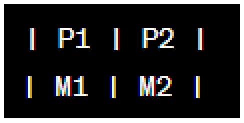
- Cost: Implementing a large crossbar network can be expensive due to the numerous connection points required.

Example: Crossbar Network with Two Processors and Two Memory Modules

Consider a simplified scenario with a small crossbar network connecting two processors (P1 and P2) to two memory modules (M1 and M2). In this example, we will calculate the bandwidth and demonstrate how simultaneous communication between processors and memory modules is achieved.

- Number of Processors (P) = 2
- Number of Memory Modules (M) = 2

Crossbar Network Configuration:
In a crossbar network, every processor can communicate directly with every memory module simultaneously. Here's the configuration:



```
| P1 | P2 |
| M1 | M2 |
```

Each processor (P1 and P2) has a direct connection to each memory module (M1 and M2).

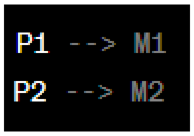
Bandwidth Calculation:
In a crossbar network, the bandwidth represents the number of simultaneous communication channels available. It is equal to the number of inputs (processors) multiplied by the number of outputs (memory modules).
Bandwidth = Number of Inputs (P) x Number of Outputs (M)
Bandwidth = 2 x 2
Bandwidth = 4 channels
So, in this example, the crossbar network provides a total of 4 communication channels. This means that at any given moment, all four combinations of processor-memory module communication can occur simultaneously.

Simultaneous Communication:
Now, let's demonstrate simultaneous communication between processors and memory modules.

Processor P1 wants to read data from Memory M1.

Processor P2 wants to write data to Memory M2.

Both of these operations can happen simultaneously because the crossbar network allows for it. There are dedicated connections for each pair of processor-memory module, so there is no contention or interference between these operations.



```
P1 --> M1
P2 --> M2
```

This demonstrates the key advantage of crossbar networks, which is the ability to achieve simultaneous communication without any conflicts or bottlenecks.