Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

# 1:Introduction High Performance Computing

**Kusum Lata Dhiman**
**Assistant Professor**
Computer Science & Engineering

# Syllabus

| Sr. | Topics | W | T |
|---|---|---|---|
| 1 | **Unit 1:**<br>**Introduction to Parallel Computing:**<br>Introduction to Parallel Computing: Motivating Parallelism, Scope of Parallel Computing, Organization and Contents of the Text, Parallel Programming Platforms: Implicit Parallelism: Trends in Microprocessor & Architectures, Limitations of Memory System Performance, Dichotomy of Parallel Computing Platforms, Physical Organization of Parallel Platforms, Communication Costs in Parallel Machines Levels of parallelism (instruction, transaction, task, thread, memory, function) Models (SIMD, MIMD, SIMT, SPMD, Dataow Models, Demand-driven Computation) Architectures: N-wide superscalar architectures, multi-core, multi-threaded. | 20 | 9 |
| 2 | **Unit 2:**<br>**Parallel Algorithm Design:**<br>Principles of Parallel Algorithm Design: Preliminaries, Decomposition Techniques, Characteristics of Tasks and Interactions, Mapping Techniques for Load Balancing, Methods for Containing Interaction Overheads, Parallel Algorithm Models. | 15 | 7 |
| 3 | **Unit 3:**<br>**Programming Using the Message-Passing Paradigm:**<br>Principles of Message- Passing Programming, The Building Blocks: Send and Receive Operations, MPI: the Message Passing Interface, Topology and Embedding, Overlapping Communication with Computation, Collective communication, and Computation Operations | 15 | 7 |

# Syllabus

| | | | |
|---|---|---|---|
| 4 | **UNIT-4:**<br>**Synchronization:**<br>Scheduling, Job Allocation, Job Partitioning, Dependency Analysis Mapping Parallel Algorithms onto Parallel Architectures, Thread Basics, The POSIX Thread API, Thread Basics: Creation and Termination, Synchronization Primitives in Pthreads, Controlling Thread and Synchronization Attributes, Thread Cancellation, Composite Synchronization Constructs, Tips for Designing Asynchronous Programs, OpenMP: a Standard for Directive Based Parallel Programming | 20 | 9 |
| 5 | **UNIT-5:**<br>**Parallel Programming Using GPGPU:**<br>An Overview of GPGPU, DGX architecture, An Overview of GPGPU Programming, An Overview of GPGPU Memory Hierarchy Features, CUDA Programming | 15 | 5 |
| 6 | **UNIT-6:**<br>**Performance measures:**<br>Speedup, efficiency, and scalability. Abstract performance metrics (work, critical paths), Amdahl`s Law, abstract vs. Real performance (granularity, scalability) | 15 | 5 |

# Books

| Reference Books | |
|---|---|
| 1. | **Introduction to Parallel Computing**<br>By Ananth ( Publication |
| 2. | **Advanced Computer Architecture: Parallelism, Scalability, Programmability**<br>By Kai Hwang, Naresh Jotwani, McGraw Hill,Second Edition, 2010 |
| 3. | **CUDA by Example ▯▯An Introduction to General Purpose GPU Programming**<br>By Edward Kandrot and Jason Sanders, Addison-Wesley Professional, 2010. |

Parul® University
Vadodara, Gujarat
NAAC GRADE A++
Information and Communication Technology

| | |
|---|---|
| 1. | Study the facilities provided by Google Colab. |
| 2. | Demonstrate basic Linux Commands. |
| 3. | Using Divide and Conquer Strategies design a class for Concurrent Quick Sort using C++. |
| 4. | Write a program on an unloaded cluster for several different numbers of nodes and record the time taken in each case. Draw a graph of execution time against the number of nodes. |
| 5. | Write a program to check task distribution using Gprof. |
| 6. | Use Intel V-Tune Performance Analyzer for Profiling. |
| 7. | Analyze the code using Nvidia-Profilers. |
| 8. | Write a program to perform load distribution on GPU using CUDA. |
| 9. | Write a simple CUDA program to print “Hello World!” |
| 10. | Write a CUDA program to add two arrays. |

# Motivating Parallelism

•The role of parallelism in accelerating computing speeds has been recognized for several decades. Developing parallel hardware and software has traditionally been time and effort intensive.

•If one is to view this in the context of rapidly improving uniprocessor speeds, one is tempted to question the need for parallel computing.

•There are some unmistakable trends in hardware design, which indicate that uniprocessor (or implicitly parallel) architectures may not be able to sustain the rate of *realizable* performance increments in the future.

•This is the result of a number of fundamental physical and computational limitations.

•The emergence of standardized parallel programming environments, libraries, and hardware have significantly reduced time to (parallel) solution.

- The speed of an application is determined by more than just processor speed.
  - Memory speed
  - Disk speed
  - Network speed
- Multiprocessors typically improve the aggregate speeds:
  - Memory bandwidth is improved by separate memories.
  - Multiprocessors usually have more aggregate cache memory.
  - Each processor in a cluster can have its own disk and network adapter, improving aggregate speeds.
- Communication enables parallel applications:
  - Harnessing the computing power of distributed systems over the Internet is a popular example of parallel processing.
- Constraints on the location of data
  - Huge data sets could be difficult, expensive, or otherwise infeasible to store in a central location.
  - Distributed data and parallel processing is a practical solution.

## Moore's law states [1965]:

*``The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000.''*

# The Computational Power Argument

Moore attributed this doubling rate to exponential behavior of die sizes, finer minimum dimensions, and ``circuit and device cleverness''.

In 1975, he revised this law as follows:

``*There is no room left to squeeze anything out by being clever. Going forward from here we have to depend on the two size factors - bigger dies and finer dimensions.''*

He revised his rate of *circuit complexity* doubling to 18 months and projected from 1975 onwards at this reduced rate.

- A die in the context of integrated circuits is a small block of semiconducting material, on which a given functional circuit is fabricated.

- By 2004, clock frequencies had gotten fast enough-around 3 GHz that any further increases would have caused the chips to melt from the heat they generated. So while the manufacturers continued to increase the number of transistors per chip, they no longer increased the clock frequencies. Instead, they started putting multiple processor cores on the chip.

# The Computational Power Argument

- If one is to buy into Moore's law, the question still remains - how does one translate transistors into useful OPS (operations per second)?
- The logical recourse is to rely on parallelism, both implicit and explicit.
- Most serial (or seemingly serial) processors rely extensively on implicit parallelism.

- Implicit parallelism is a characteristic of a programming language that allows a compiler or interpreter to automatically exploit the parallelism inherent to the computations expressed by some of the language's constructs. A pure implicitly parallel language does not need special directives, operators or functions to enable parallel execution

# The Memory/Disk Speed Argument

- While clock rates of high-end processors have increased at roughly 40% per year over the past decade, DRAM access times have only improved at the rate of roughly 10% per year over this interval.
- This mismatch in speeds causes significant performance bottlenecks.
- Principles of locality of data reference and bulk access, which guide parallel algorithm design also apply to memory optimization.
- Some of the fastest growing applications of parallel computing utilize not their raw computational speed, rather their ability to pump data to memory and disk faster.

# The Data Communication Argument

- As the network evolves, the vision of the Internet as one large computing platform has emerged.
- This view is exploited by applications such as SETI@home and Folding@home.
- In many other applications (typically databases and data mining) the volume of data is such that they cannot be moved.
- Any analyses on this data must be performed over the network using parallel techniques.

- The search for extraterrestrial intelligence (SETI) is the collective name for a number of activities undertaken to search for intelligent extraterrestrial life

- Folding@home is a distributed computing project for disease research that simulates protein folding, computational drug design, and other types of molecular dynamics. The project uses the idle processing resources of thousands of personal computers owned by volunteers who have installed the software on their systems.

•Datapath is the hardware that performs all the required data processing operations, for example, ALU, registers, and internal buses.

• Control is the hardware that tells the data path what to do, in terms of switching, operation selection, data movement between ALU components, etc.

# Scope of Parallel Computing Applications

- Parallelism finds applications in very diverse application domains for different motivating reasons.

- These range from improved application performance to cost considerations.

# Scientific Applications

- Functional and structural characterization of genes and proteins.
- Advances in computational physics and chemistry have explored new materials, understanding of chemical pathways, and more efficient processes.
- Applications in astrophysics have explored the evolution of galaxies, thermonuclear processes, and the analysis of extremely large datasets from telescopes.
- Weather modeling, mineral prospecting, flood prediction, etc., are other important applications.
- Bioinformatics and astrophysics also present some of the most challenging problems with respect to analyzing extremely large datasets.

# Commercial Applications

- Data mining and analysis for optimizing business and marketing decisions.
- Large scale servers (mail and web servers) are often implemented using parallel platforms.
- Applications such as information retrieval and search are typically powered by large clusters.

# Applications in Computer Systems

- Network intrusion detection, cryptography, multiparty computations are some of the core users of parallel computing techniques.
- Embedded systems increasingly rely on distributed control algorithms.
- A modern automobile consists of tens of processors communicating to perform complex tasks for optimizing handling and performance.
- Conventional structured peer-to-peer networks impose overlay networks and utilize algorithms directly from parallel computing.

# Scope of Parallelism

- Conventional architectures coarsely comprise of a processor, memory system, and the data path.
- Each of these components present significant performance bottlenecks.
- Parallelism addresses each of these components in significant ways.
- Different applications utilize different aspects of parallelism - e.g., data intensive applications utilize high aggregate throughput, server applications utilize high aggregate network bandwidth, and scientific applications typically utilize high processing and memory system performance.

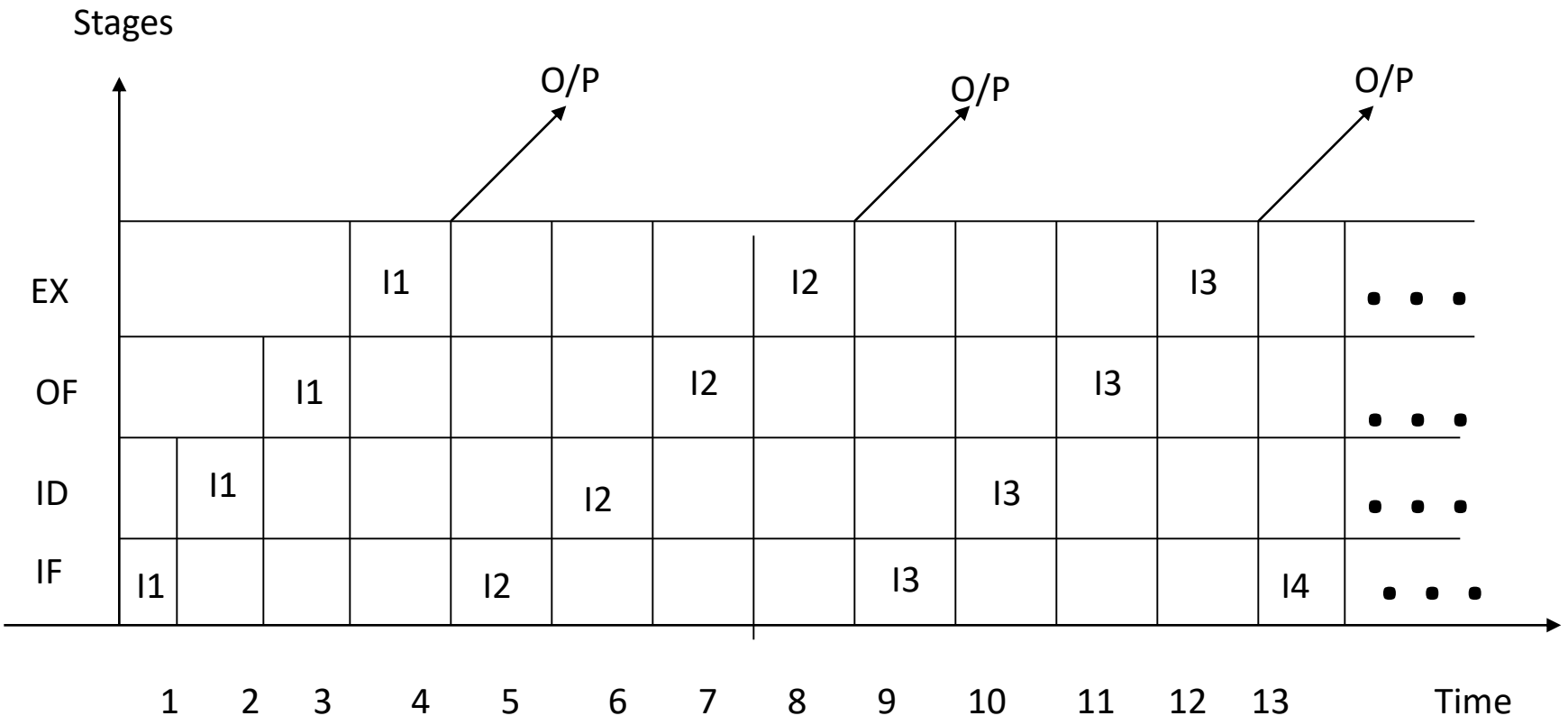
## Implicit Parallelism: Trends in Microprocessor Architectures

- Microprocessor clock speeds have posted impressive gains over the past two decades (two to three orders of magnitude).
- Higher levels of device integration have made available a large number of transistors.
- The question of how best to utilize these resources is an important one.
- Current processors use these resources in multiple functional units and execute multiple instructions in the same cycle.
- The precise manner in which these instructions are selected and executed provides impressive diversity in architectures.

# Pipelining

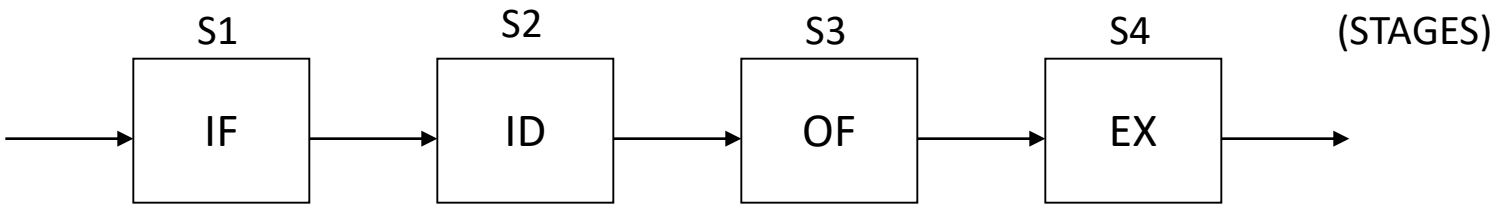
- A technique used in advanced microprocessors where the microprocessor begins executing a second instruction before the first has been completed. That is, several instructions are in the *pipeline* simultaneously, each at a different processing stage.

- The pipeline is divided into segments and each segment can execute its operation concurrently with the other segments. When a segment completes an operation, it passes the result to the next segment in the pipeline and fetches the next operation from the preceding segment. The final results of each instruction emerge at the end of the pipeline in rapid succession.




C) Space-time dig. For a non-pipelined processor

Fig. :- Basic Concepts of pipelined processor and overlapped instruction execution




A) A Pipeline Processor

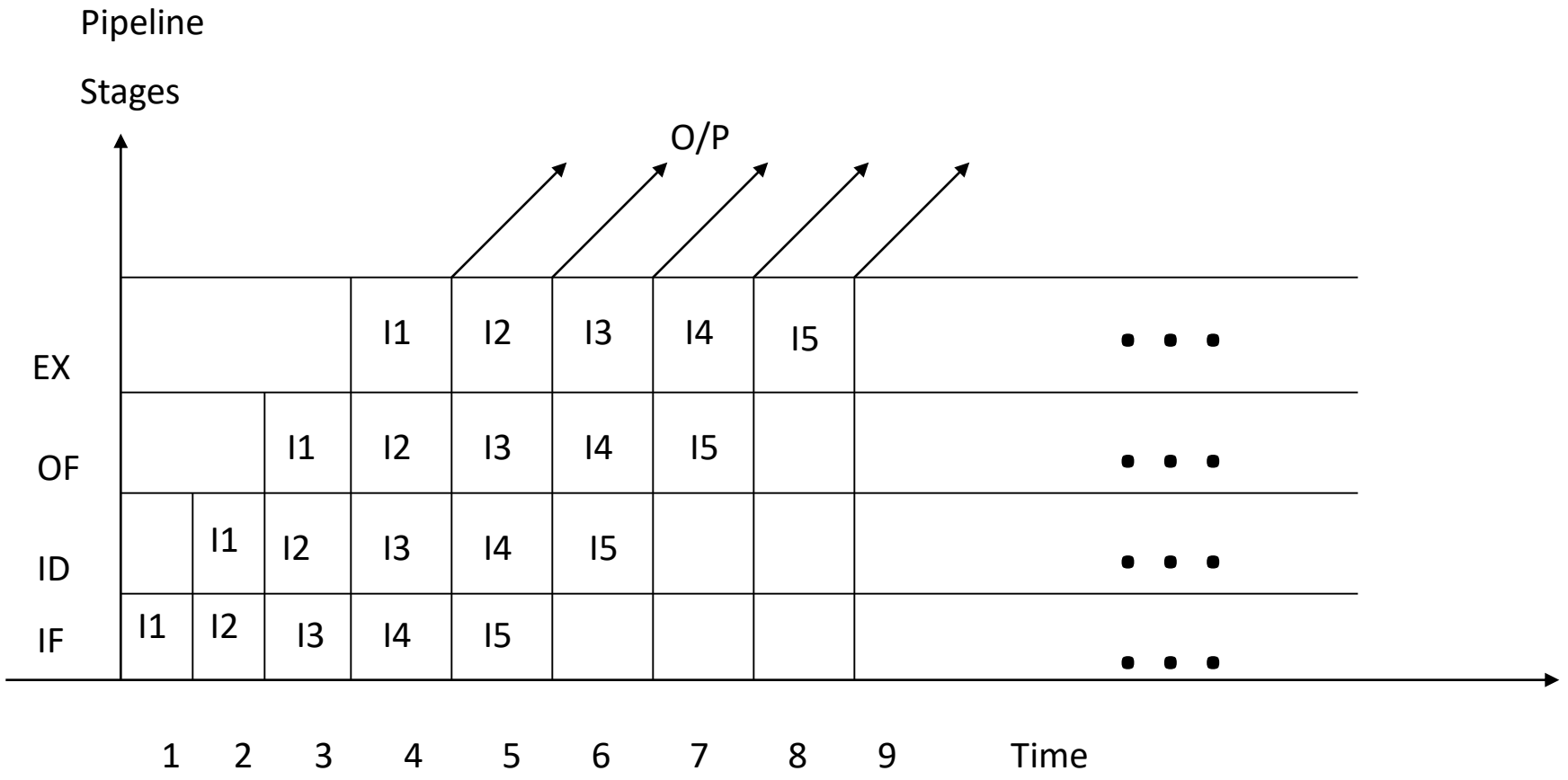



Fig. :-Space-time Diagram for Pipelined Processor

# Superscalar Processor

A superscalar processor is a CPU that implements a form of parallelism called instruction-level parallelism within a single processor. It therefore allows faster CPU throughput (the number of instructions that can be executed in a unit of time) than would otherwise be possible at a given clock rate. A superscalar processor executes more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to different execution units on the processor. Each execution unit is not a separate processing unit (called "core") as in multi-core processors, but an execution resource within a single CPU such as an arithmetic logic unit, a bit shifter, or a multiplier.

# Pipelining and Superscalar Execution

- Pipelining overlaps various stages of instruction execution to achieve performance.
- At a high level of abstraction, an instruction can be executed while the next one is being decoded and the next one is being fetched.
- This is akin to an assembly line for manufacture of cars.

# Pipelining and Superscalar Execution

- Pipelining, however, has several limitations.
- The speed of a pipeline is eventually limited by the slowest stage.
- For this reason, conventional processors rely on very deep pipelines (20 stage pipelines in state-of-the-art Pentium processors).
- However, in typical program traces, every 5-6th instruction is a conditional jump! This requires very accurate branch prediction.
- The penalty of a misprediction grows with the depth of the pipeline, since a larger number of instructions will have to be flushed.

# Pipelining and Superscalar Execution

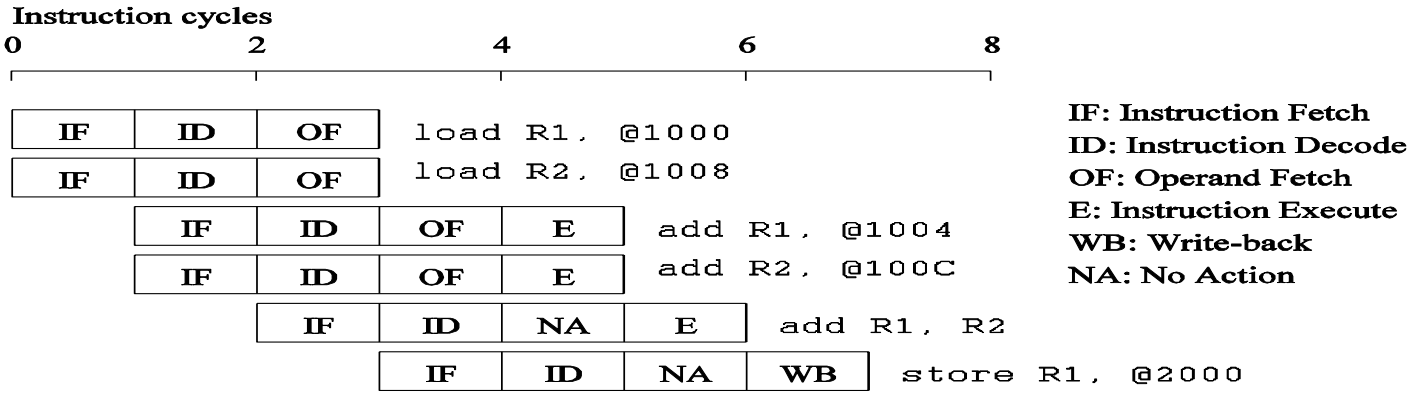
- One simple way of alleviating these bottlenecks is to use multiple pipelines.
- The question then becomes one of selecting these instructions.

# Superscalar Execution: An Example

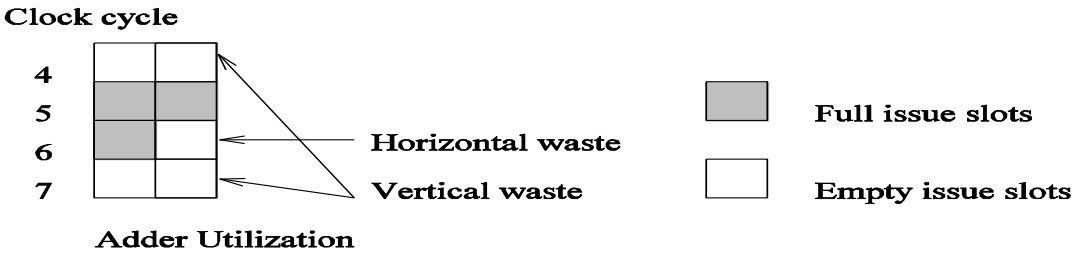
```
1. load R1, @1000        1. load R1, @1000        1. load R1, @1000
2. load R2, @1008        2. add R1, @1004         2. add R1, @1004
3. add R1, @1004         3. add R1, @1008         3. load R2, @1008
4. add R2, @100C         4. add R1, @100C         4. add R2, @100C
5. add R1, R2            5. store R1, @2000       5. add R1, R2
6. store R1, @2000                                6. store R1, @2000
```

(i) (ii) (iii)

(a) Three different code fragments for adding a list of four numbers.




(b) Execution schedule for code fragment (i) above.




(c) Hardware utilization trace for schedule in (b).

Example of a two-way superscalar execution of instructions.

# Superscalar Execution: An Example

- In the above example, there is some wastage of resources due to data dependencies.
- The example also illustrates that different instruction mixes with identical semantics can take significantly different execution time.

# Superscalar Execution

- Scheduling of instructions is determined by a number of factors:
  - True Data Dependency: The result of one operation is an input to the next.
  - Resource Dependency: Two operations require the same resource.
  - Branch Dependency: Scheduling instructions across conditional branch statements cannot be done deterministically a-priori.
  - The scheduler, a piece of hardware looks at a large number of instructions in an instruction queue and selects appropriate number of instructions to execute concurrently based on these factors.
  - The complexity of this hardware is an important constraint on superscalar processors.

# Superscalar Execution: Issue Mechanisms

- In the simpler model, instructions can be issued only in the order in which they are encountered. That is, if the second instruction cannot be issued because it has a data dependency with the first, only one instruction is issued in the cycle. This is called *in-order* issue.
- In a more aggressive model, instructions can be issued out of order. In this case, if the second instruction has data dependencies with the first, but the third instruction does not, the first and third instructions can be co-scheduled. This is also called dynamic issue.
- Performance of in-order issue is generally limited.

# Superscalar Execution: Efficiency Considerations

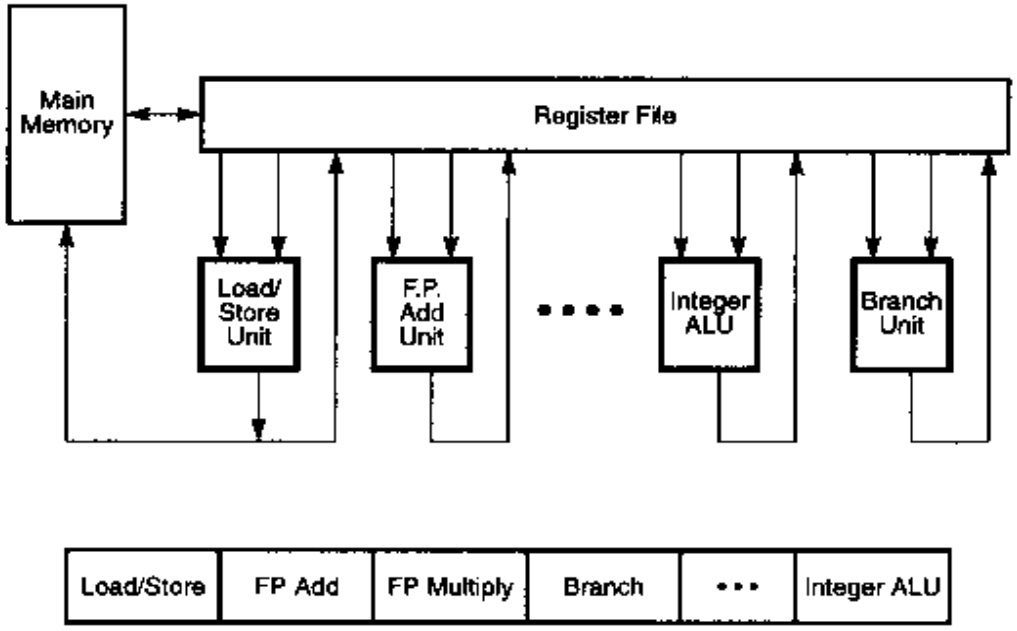
- Not all functional units can be kept busy at all times.
- If during a cycle, no functional units are utilized, this is referred to as vertical waste.
- If during a cycle, only some of the functional units are utilized, this is referred to as horizontal waste.
- Due to limited parallelism in typical instruction traces, dependencies, or the inability of the scheduler to extract parallelism, the performance of superscalar processors is eventually limited.
- Conventional microprocessors typically support four-way superscalar execution.

## Superscalar Execution: Efficiency Considerations

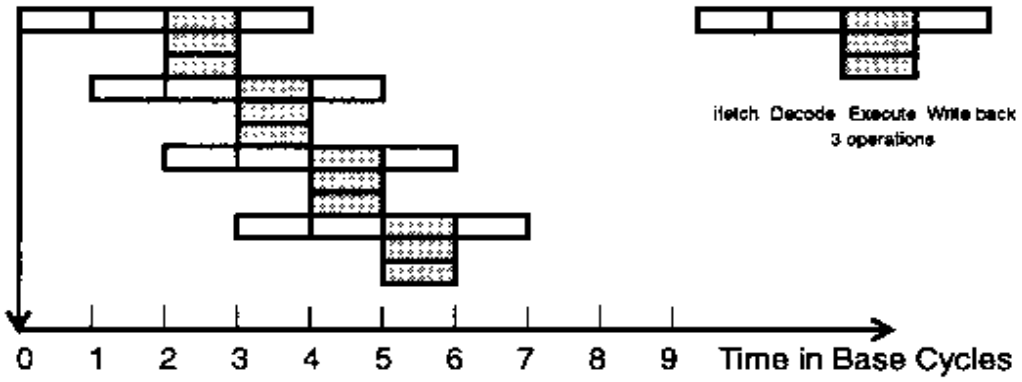
- Not all functional units can be kept busy at all times.
- If during a cycle, no functional units are utilized, this is referred to as vertical waste.
- If during a cycle, only some of the functional units are utilized, this is referred to as horizontal waste.
- Due to limited parallelism in typical instruction traces, dependencies, or the inability of the scheduler to extract parallelism, the performance of superscalar processors is eventually limited.
- Conventional microprocessors typically support four-way superscalar execution.

## Very Long Instruction Word (VLIW) Processors: Considerations

- Issue hardware is simpler.
- Compiler has a bigger context from which to select co-scheduled instructions.
- Compilers, however, do not have runtime information such as cache misses. Scheduling is, therefore, inherently conservative.
- Branch and memory prediction is more difficult.
- VLIW performance is highly dependent on the compiler. A number of techniques such as loop unrolling, speculative execution, branch prediction are critical.
- Typical VLIW processors are limited to 4-way to 8-way parallelism.




(a) A typical VLIW processor and instruction format




(b) VLIW execution with degree $m = 3$

**Figure 4.14 The architecture of a very long instruction word (VLIW) processor and its pipeline operations.** (Courtesy of Multiflow Computer, Inc., 1987)

- Memory system, and not processor speed, is often the bottleneck for many applications.
- Memory system performance is largely captured by two parameters, latency and bandwidth.
- Latency is the time from the issue of a memory request to the time the data is available at the processor.
- Bandwidth is the rate at which data can be pumped to the processor by the memory system.

# Memory System Performance: Bandwidth and Latency

- It is very important to understand the difference between latency and bandwidth.
- Consider the example of a fire-hose. If the water comes out of the hose two seconds after the hydrant is turned on, the latency of the system is two seconds.
- Once the water starts flowing, if the hydrant delivers water at the rate of 5 gallons/second, the bandwidth of the system is 5 gallons/second.
- If you want immediate response from the hydrant, it is important to reduce latency.
- If you want to fight big fires, you want high bandwidth.

# Memory Latency: An Example

- Consider a processor operating at 1 GHz (1 ns clock) connected to a DRAM with a latency of 100 ns (no caches). Assume that the processor has two multiply-add units and is capable of executing four instructions in each cycle of 1 ns. The following observations follow:

  - The peak processor rating is 4 GFLOPS.
  - Since the memory latency is equal to 100 cycles and block size is one word, every time a memory request is made, the processor must wait 100 cycles before it can process the data.

# Memory Latency: An Example

- On the above architecture, consider the problem of computing a dot-product of two vectors.
  - A dot-product computation performs one multiply-add on a single pair of vector elements, i.e., each floating point operation requires one data fetch.
  - It follows that the peak speed of this computation is limited to one floating point operation every 100 ns, or a speed of 10 MFLOPS, a very small fraction of the peak processor rating!

## Improving Effective Memory Latency Using Caches

- Caches are small and fast memory elements between the processor and DRAM.
- This memory acts as a low-latency high-bandwidth storage.
- If a piece of data is repeatedly used, the effective latency of this memory system can be reduced by the cache.
- The fraction of data references satisfied by the cache is called the cache *hit ratio* of the computation on the system.
- Cache hit ratio achieved by a code on a memory system often determines its performance.

# Impact of Caches: Example

Consider the architecture from the previous example. In this case, we introduce a cache of size 32 KB with a latency of 1 ns or one cycle. We use this setup to multiply two matrices A and B of dimensions 32 × 32. We have carefully chosen these numbers so that the cache is large enough to store matrices A and B, as well as the result matrix C.

# Impact of Caches: Example (continued)

- The following observations can be made about the problem:
  - Fetching the two matrices into the cache corresponds to fetching 2K words, which takes approximately 200 µs.
  - Multiplying two $n \times n$ matrices takes $2n^3$ operations. For our problem, this corresponds to 64K operations, which can be performed in 16K cycles (or 16 µs) at four instructions per cycle.
  - The total time for the computation is therefore approximately the sum of time for load/store operations and the time for the computation itself, i.e., 200 + 16 µs.
  - This corresponds to a peak computation rate of 64K/216 or 303 MFLOPS.

# Impact of Caches

- Repeated references to the same data item correspond to temporal locality.
- In our example, we had $O(n^{2})$ data accesses and O(n3) computation. This asymptotic difference makes the above example particularly desirable for caches.
- Data reuse is critical for cache performance.

# Impact of Memory Bandwidth

- Memory bandwidth is determined by the bandwidth of the memory bus as well as the memory units.
- Memory bandwidth can be improved by increasing the size of memory blocks.
- The underlying system takes *l* time units (where *l* is the latency of the system) to deliver *b* units of data (where *b* is the block size).

# Impact of Memory Bandwidth: Example

- Consider the same setup as before, except in this case, the block size is 4 words instead of 1 word. We repeat the dot-product computation in this scenario:
  - Assuming that the vectors are laid out linearly in memory, eight FLOPs (four multiply-adds) can be performed in 200 cycles.
  - This is because a single memory access fetches four consecutive words in the vector.
  - Therefore, two accesses can fetch four elements of each of the vectors. This corresponds to a FLOP every 25 ns, for a peak speed of 40 MFLOPS.

# Impact of Memory Bandwidth

- It is important to note that increasing block size does not change latency of the system.
- Physically, the scenario illustrated here can be viewed as a wide data bus (4 words or 128 bits) connected to multiple memory banks.
- In practice, such wide buses are expensive to construct.
- In a more practical system, consecutive words are sent on the memory bus on subsequent bus cycles after the first word is retrieved.

Parul® University

NAAC GRADE A++

https://paruluniversity.ac.in/