

UNIT 1

FUNDAMENTAL CONCEPT OF PARALLEL PROCESSING

1.1 Introduction

1.1.1 History of computer

1.1.2 Parallel Computer structure

1.1.3 Motivation of parallelism

1.1.4 Moore's Law

1.1.5 Grand Challenge problems

1.2 Types of parallelism

1.3 Granularity

1.3.1 Fined-grained parallelism

1.3.2 Coarse-grained parallelism

1.3.3 Medium-grained parallelism

1.4 Performance of parallel computer

1.5 Speed up Performance Law

1.5.1 Amdahl's Law

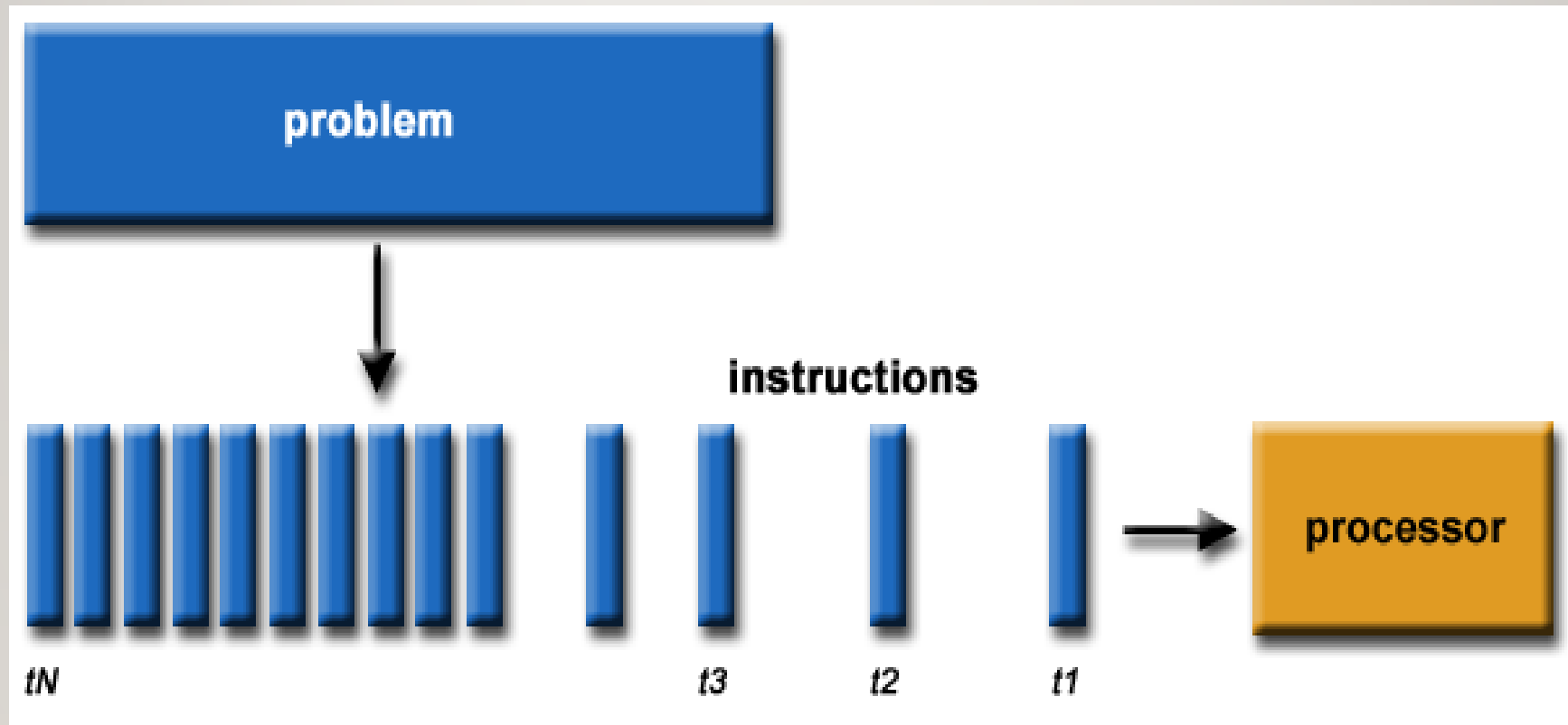
1.5.2 Gustafson's Law

DISTRIBUTED COMPUTING

- Computation is usually performed on one machine i.e you have a computer, you give it input, it processes this input and it throws out some output. In our day to day life this is performed and it is possible for small scale project.
- But for a large scale project a single computer may be too slow to solve a large problem and that is where **distributed computing** comes on.
- In **distributed computing** a large complex task is broken into small tasks and distribute this workload over a large number of computer so that each computer only needs to perform a small tasks which helps in time consuming.

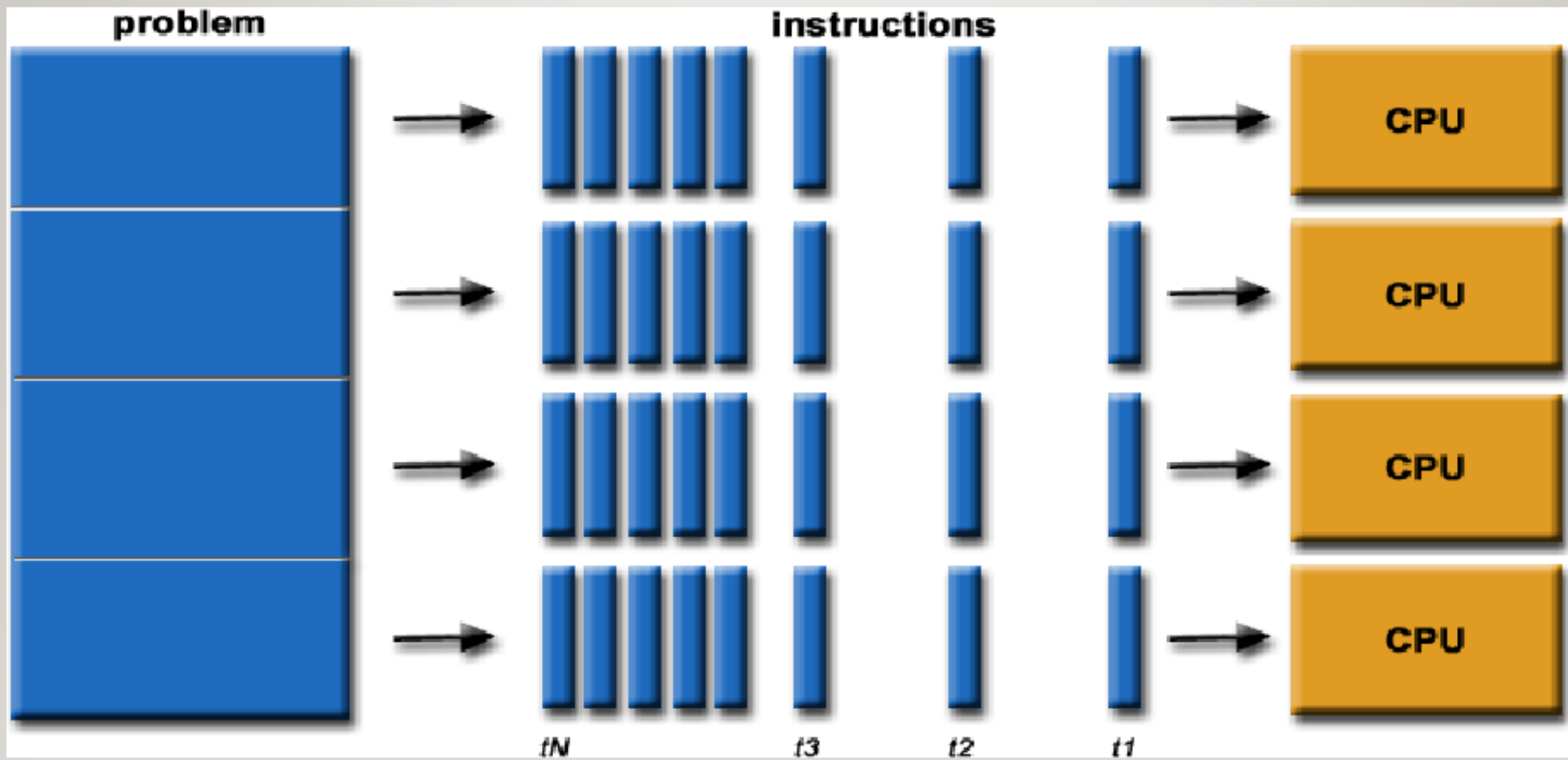
SERIAL COMPUTATION

- Software has been written for serial computation:
- To be run on a single computer having a single Central Processing Unit(CPU).
- A problem is broken into discrete series of instructions.
- Instruction are executed one after another.
- Only one instruction may execute at any moment in time.
- Type of processing in which one task is completed at a time and all the tasks are executed by the processor sequentially.



PARALLEL COMPUTATION

- Parallel computation is the simultaneous use of multiple compute resources to solve a large complex problems.
- To be run using multiple CPUs.
- A problem is broken into discrete parts that can be solved concurrently.
- Each part is further broken down to a series of instructions.
- Instructions from each parts execute simultaneously on different CPUs.



WHY WE USE PARALLEL COMPUTING

- Save time
- Higher speed
- Solve larger problem
- Provide Concurrency
- Higher performance
- Work load of processor is lower

APPLICATION OF PARALLEL PROCESSING

1. Engineering and Design
 - Mechanical Engineering (intennal combustion engine)
 - Civil Engineering(lift)
 - Electrical and Electronics(High speed circuit)
2. Scientific Application
 - Astrophysics
 - Weather Prediction
 - Medical Science
3. Commercial Application
 - E-commerce Websites
4. Computer System
 - Cryptography
5. Military application

MOTIVATING PARALLELISM

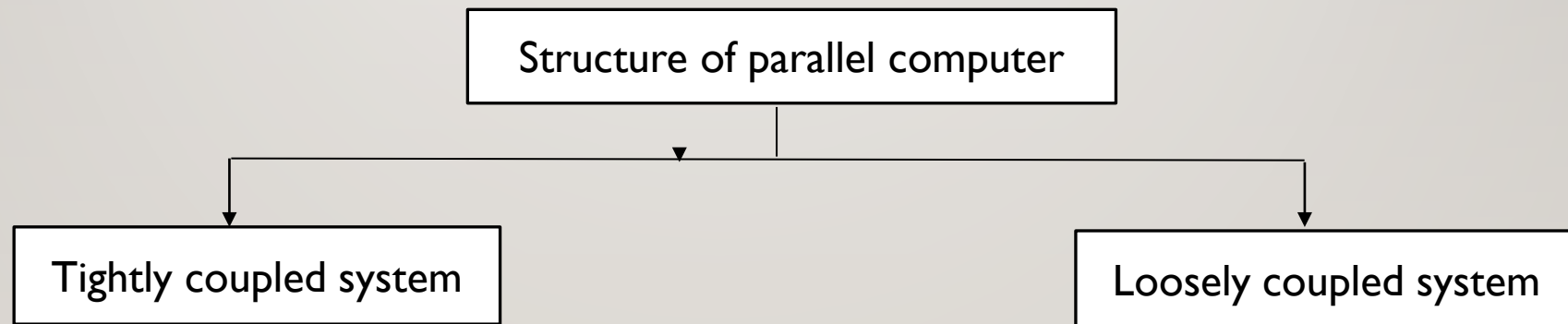
- The role of parallelism in accelerating computing speeds has been recognized for several decades.
- Its role in providing multiplicity of datapaths and increased access to storage elements has been significant in commercial application.
- The scalable performance and lower cost of parallel platforms is reflected in the wide variety of applications.
- Developing parallel hardware and software has traditionally been time and effort intensive.

MOTIVATING PARALLELISM CONTD.

- 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 architectures may not be able to sustain the rate of realizable performance increments in the future.
- The emergence of standardized parallel programming environments, libraries, and hardware have significantly reduced time to (parallel) solution.

PARALLEL COMPUTER STRUCTURE

- A **parallel computer** is a computer with multiple processors that is capable of operating on a shared set of data simultaneously, improving the overall performance.
- Parallel computer can be classified on the basis of structure.
- A parallel computer can be characterized as a set of multiple processor and shared memory and memory modules communicating through interconnection network.



TIGHTLY COUPLED SYSTEM AND LOOSELY COUPLED SYSTEM

Tightly coupled system: When multiprocessor communicates through the global shared memory modules then this organization is shared memory computer or Tightly coupled system. They are used for high speed real time processing.

Loosely coupled system: when every processor in multiprocessor system has its own Local memory and the processor communicate through message transmitted between their local memory then this organization is distributed memory computer or loosely coupled system. The system does not share global memory which in returns slow down execution time.



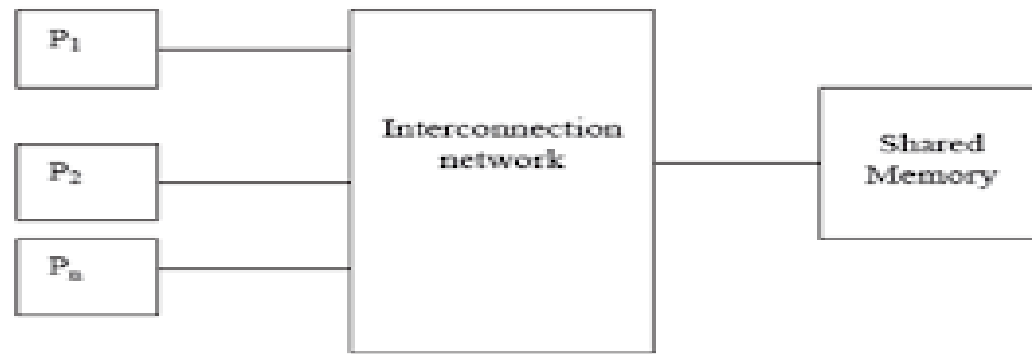


Figure 9: Tightly coupled system



Figure 10 Loosely coupled system

TIGHTLY COUPLED SYSTEM VS. LOOSELY COUPLED SYSTEM

Tightly coupled system

1. There is shared memory, in tightly coupled multiprocessor system.
2. Tightly coupled multiprocessor system has high data rate.
3. Tightly coupled multiprocessor system is more costly.
4. They perform better and are physically small.
5. They consume low power.
6. System is dependent with each other.

Loosely coupled system

1. There is distributed memory in loosely coupled multiprocessor system.
2. Loosely Coupled Multiprocessor System has low data rate.
3. The cost of loosely coupled multiprocessor system is less.
4. They are slower and are physically large.
5. They consume high power.
6. System is dependent with each other.

MOORE'S LAW

- Moore's Law is a prediction formulated by Gordon E. Moore (co-founder of Intel) in 1965 about transistor density on integrated chips (ICs).
- According to Moore's law, "the number of transistors used per square inch in the Integrated Circuits (IC's) will double about every 2 years."
- As a result the scale gets smaller and smaller and the performance of microprocessor has enjoyed an exponential growth.
- Speed and capability increase every couple of years.
- In 1970 there were around 1000 transistors placed in an integrated chip whereas in 2010 there were nearly 1 billion transistors placed in a chip.

IS MOORE'S LAW STILL HOLDING?

- Moore's Law is not natural law, it is an observation did by Gordon E. Moore.
- The law has been remarkably accurate from 1965 to 2013 with transistors density doubling roughly every two years.
- Since 2013, the trend has changes slightly, with transistor density now doubling every three year, instead of every two years.

IS MOORE'S LAW STILL HOLDING?

Facts


Misconception

- Reduction of transistor gate length less than 22nm is not possible.

New Inventions

- Recently INTEL has reduced transistor gate length less than 22nm

Result

- The invention indicates that Moore's law road does not end.
 - As long as the thirst for innovation is present Moore's law trend will continue.
 - Performance and speed is increasing so it is valid up to now.
- 

GRAND CHALLENGE PROBLEMS

The generic “Grand Challenge” problem proposed is that of parallel computing. The challenge is:

- For general-purpose computing, develop a cost effective architecture for improving single task completion time by exploiting parallelism.
- Finding and expressing concurrency.
- Managing data distributions.
- Managing inter- processor communication.
- Balancing the computational load, and simply implementing the parallel algorithm correctly.

TYPES OF PARALLELISM

The term **Parallelism** refers to techniques to make programs faster by performing several computations at the same time. This requires hardware with multiple processing units.

1. Bit-level parallelism:

- It is the form of parallel computing which is based on increasing processor word size.
- Increasing the word size reduces the number of instructions.
- The processor must execute to perform an operation on variables whose sizes are greater than the length of word.

-
- For example: Consider a scenario where an 8-bit processor must compute 16-bit addition operation. The processor must first sum up the 8 lower-order bits, then add the 8 higher-order bits, thus requiring two instructions to perform the operation. A 16-bit processor can perform the operation with just one instruction.
 - Advancement in computer architecture were done increasing bit level parallelism as 4 bit microprocessors were replaced by 8 bit then 16 bit then 32 bit then 64 bit.

2. Instruction-level parallelism (ILP) :

- ILP is a measure of how many of the instructions in a computer program can be executed simultaneously.
- A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. This is called instruction-level parallelism.
- With ILP method we found that the performance of the computer processor can be increased and more than two program can be executing simultaneously at a time.

Consider the following program:

1. $e = a + b$
2. $f = c + d$
3. $m = e * f$

Operation 3 depends on the results of operations 1 and 2, so it cannot be calculated until both of them are completed. However, operations 1 and 2 do not depend on any other operation, so they can be calculated simultaneously. If we assume that each operation can be completed in one unit of time then these three instructions can be completed in a total of two units of time, giving an ILP of 3/2

Architectural techniques that are used to exploit ILP include:

1. Superscalar execution

2. VLIW

1.SUPERSCALAR EXECUTION

- multiple execution units are used to execute multiple instructions in parallel.
- Multiple independent instruction executes in parallel. Common instruction like arithmetic. Load/store etc can be initiated simultaneously and executed independently.
- More commonly used in RISC.
- The main principle of superscalar approach is that it execute instruction independently in different instruction pipelining which leads to parallel processing thereby speeding the processing of instruction.

Instruction pipelining where the execution of multiple instructions can be partially overlapped.

Basic five-stage pipeline

<div><div></div><div>Instr. No.</div></div> <div><div>Clock cycle</div><div></div></div>	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX

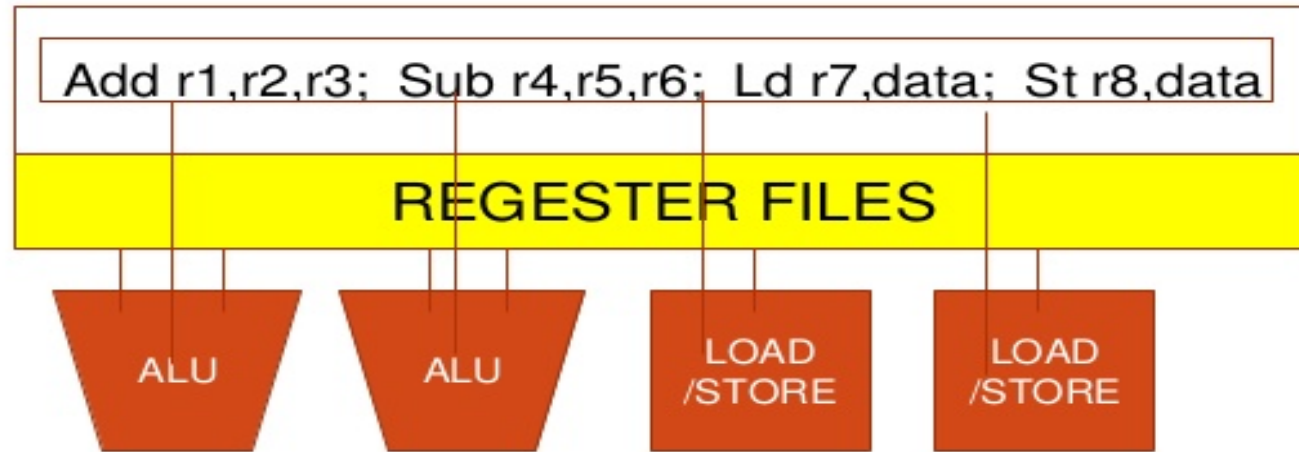
(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).

In the fourth clock cycle (the green column), the earliest instruction is in MEM stage, and the latest instruction has not yet entered the pipeline.

2. VLIW

- Very long instruction word (VLIW) describes a computer processing architecture in which a language compiler or pre-processor breaks program instruction down into basic operations that can be performed by the processor in parallel (that is, at the same time).
- These operations are put into a very long instruction word which the processor can then take apart without further analysis, handing each operation to an appropriate functional unit.

VLIW Instruction



3. Task level Parallelism or Thread level parallelism:

- Task parallelism involves the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform execution of sub tasks concurrently.
- Task Parallelism means concurrent execution of the different task or threads on multiple computing cores.

4. Data level Parallelism:

- Data Parallelism means concurrent execution of the same task on each multiple computing core.
- Let's take an example, summing the contents of an array of size N . For a single-core system, one thread would simply sum the elements $[0] \dots [N - 1]$. For a dual-core system, however, thread A, running on core 0, could sum the elements $[0] \dots [N/2 - 1]$ and while thread B, running on core 1, could sum the elements $[N/2] \dots [N - 1]$. So the Two threads would be running in parallel on separate computing cores.
- In data level parallelism large amount of data element needs to be processed to produce a result in parallel .

5. Memory level parallelism

- The number of concurrent memory requests that a given architecture can handle at a time.
- A memory able to provide multiple data at different addresses at the same time.

GRANULARITY

- In parallel computing, **granularity** (or grain size) of a task is a measure of the amount of work (or computation) which is performed by that task. In another word granularity is a measure which determines how much computation is involved in a process.
- The communication overhead between multiple processors or processing elements.
- So it defines granularity as the ratio of computation time to communication time.
- computation time is the time required to perform the computation of a task and communication time is the time required to exchange data between processors.

GRANULARITY CONT.

- Let say T_{comp} denotes the computation time and T_{comm} denotes the communication time, then the Granularity G of a task can be calculated as:

$$G = \frac{T_{comp}}{T_{comm}}$$

- Granularity is determined by counting the number of instruction executed in a particular task.
- Alternately, granularity can also be specified in terms of the execution time of a program, combining the computation time and communication time.

TYPES OF GRANULARITY

Depending on the amount of work which is performed by parallel task, parallelism can be classified into three categories:

1. Fine-grained parallelism: Program are broken into large number of small tasks.
2. Coarse-grained parallelism : Program are broken into small number of large tasks.
3. Medium grained parallelism : It is used relatively to fine-grained and coarse-grained parallelism.

FINE-GRAINED PARALLELISM

- Program is broken down to a large number of small tasks. These tasks are assigned individually to many processors.
- The amount of work associated with a parallel task is low and the work is evenly distributed among the processors. Hence, fine-grained parallelism facilitates load balancing.
- Due to fine granularity there is a need of large number of processor but with less data so communication and synchronization overhead is large.

FINE-GRAINED PARALLELISM CONT.

- Fine-grained parallelism is best exploited in architectures which support fast communication.
- An example of a fine-grained system (from outside the parallel computing domain) is the system of neurons in our brain.

COARSE-GRAINED PARALLELISM

- Program is split into large tasks. Due to this, a large amount of computation takes place in processors.
- This might result in load imbalance, wherein certain tasks process the bulk of the data while others might be idle.
- Sometime coarse-grained parallelism do not implement parallelism because most of the computation is performed sequentially on a processor.
- The advantage is low communication and synchronization overhead.
- Message-passing architecture takes a long time to communicate data among processes which makes it suitable for coarse-grained parallelism

MEDIUM GRAINED PARALLELISM

- Medium-grained parallelism is used relatively to fine-grained and coarse-grained parallelism.
- Medium-grained parallelism is a compromise between fine-grained and coarse-grained parallelism, where we have task size and communication time greater than fine-grained parallelism and lower than coarse-grained parallelism.
- Most general-purpose parallel computers fall in this category.
- Intel ipsc is an example of medium grained parallelism with grain size of 10.

IMPACT OF GRANULARITY ON PERFORMANCE

- Granularity affects the performance of parallel computers.
- Using fine grains or small tasks results in more parallelism and hence increases the speedup. However, synchronization overhead, scheduling strategies etc. can negatively impact the performance of fine-grained tasks. Increasing parallelism alone cannot give the best performance.
- Coarse grained tasks have less communication overhead but they often cause load imbalance.
- Result is that granularity should be done according to problem.

FINE-GRAINED VS COARSE-GRAINED

FINE-GRAINED

- Program are broken into large number of small tasks.
- Fine Grain have less computation time then the coarse grain architecture.
- Fine Grain have much higher level of parallelism then Coarse grain .

COARSE-GRAINED

- Program are broken into small number of large tasks.
- Coarse Grain have more computation time then the Fine grain architecture
- Coarse grain have lower level of parallelism then Fine Grain.

FINE-GRAINED VS COARSE-GRAINED

FINE-GRAINED

- Here, two types of parallelism can be obtained –
 - a) Instruction Level Parallelism
 - b) Loop Level Parallelism
- In Fine Grain, Load Balancing is proper.
- **Examples:** Connection Machine (CM-2), J-Machine etc

COARSE-GRAINED

- Here, these two types of parallelism can be obtained –
 - a) Sub-program
 - b) Program Level Parallelism
- In Coarse Grain Load Balancing is improper.
- **Examples :** CRAY Y, etc.

PERFORMANCE OF PARALLEL PROCESSOR

There are many ways to measure performance of parallel processor. Some of them are

1. Speedup
2. Efficiency
3. Redundancy
4. Utilization

1. Speedup

Speedup is a measure of performance. It measures the ratio between the sequential execution time and the parallel execution time.

$$S(n) = \frac{T(1)}{T(n)}$$

$T(1)$ is the execution time with one processing unit

$T(n)$ is the execution time with n processing units

For example: $T(1)=1\text{sec}$ and if $n=2$ processor then $T(2)=\frac{1}{2}=0.5\text{ sec}$

Hence $S(n) = \frac{T(1)}{T(2)} = \frac{1}{0.5} = 2$ means Speedup increases by 2 times.

2. Efficiency

The efficiency is defined as the ratio of speedup to the number of processors i.e n .
Efficiency measures the fraction of time for which a processor is usefully utilized.

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{n * T(n)}$$

$S(n)$ is the speedup for n processing units.

3. Redundancy

It measures the ratio between the number of operations performed by the parallel execution and by the sequential execution.

$$R(n) = \frac{O(n)}{O(1)}$$

$O(1)$ is the total number of operations performed by one processing unit

$O(n)$ is the total number of operations performed by n processing units

4. Utilization

Utilization is a measure of the good use of the computational capacity. It measures the ratio between the computational capacity used during parallel execution and the capacity that was available.

$$U(n) = R(n) \times E(n)$$

SPEED UP PERFORMANCE LAW

There also some laws/metrics that try to explain and assert the potential performance of a parallel processor. The best known are:

- Amdahl's Law
- Gustafson's law

AMDAHL'S LAW

- **Amdahl's Law** was named after Gene Amdahl, who presented it in 1967.
- It is based on a fixed problem size (or fixed work load).
- Actually speedup increase linearly as the number of processor increases .But in real time this is not the case .In real time we have Amdahl's Law.
- Amdahl's Law tells that for a given problem size , the speedup doesn't increase linearly as the number of processor increases . In fact, speed-up tends to saturated. This is the consequences of Amdahl's Law.
- Amdahl's law states that **a small portion of the program which cannot be parallelized(serial part) , will limit the overall speed-up , available from parallelization.**
- Computation problem = Serial Part + Parallel Part

-
- Let 'f' be the fraction of operations in a computation that must be performed sequentially , where $0 \leq f \leq 1$. The maximum speedup achievable by a parallel computer with n processor is :

Maximum Speedup

$$S(n) \leq \frac{1}{f + \frac{(1-f)}{n}}$$

This is known as Amdahl's Law.

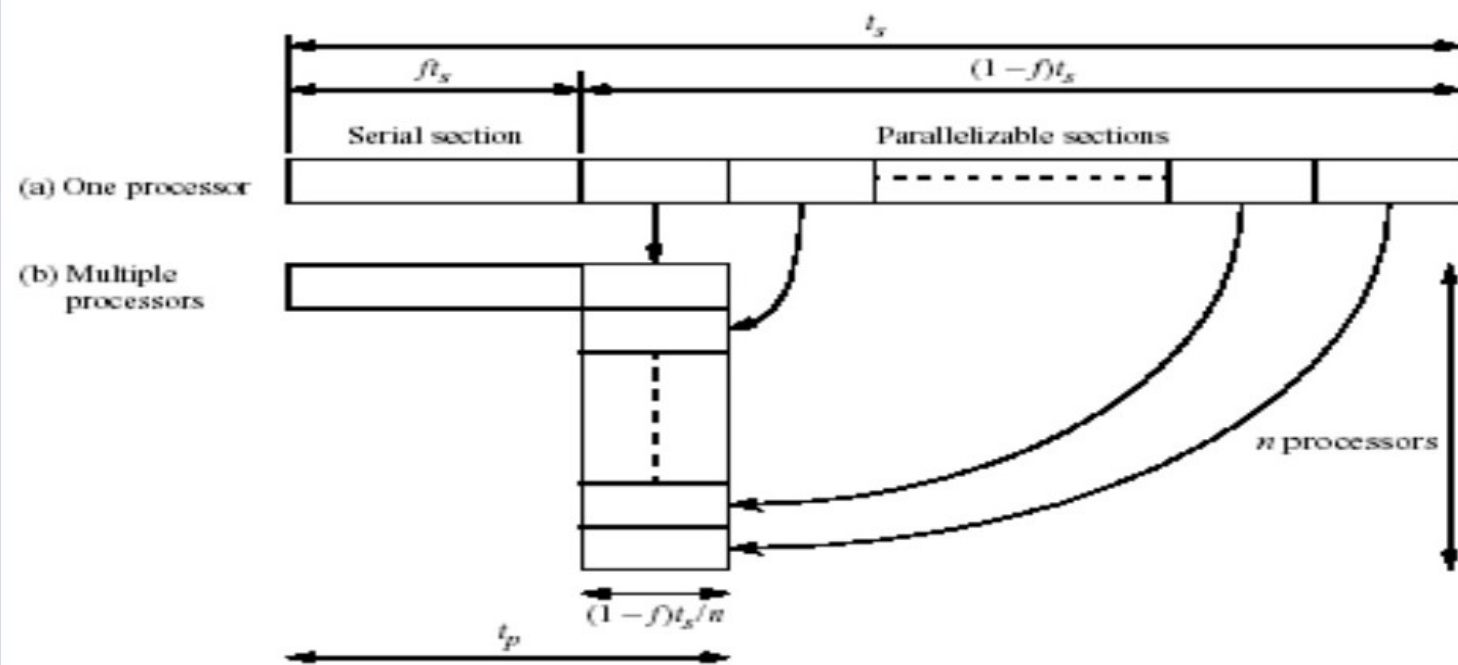


Figure 1.29 Parallelizing sequential problem — Amdahl's law.

-
- If the fraction of computation that can be executed in serial is 'f' and the remaining computation that can be divided into concurrent parts and executed in parallel is (1-f), then the time to perform the computation with single processor (t_s) and multiple 'n' processor (t_p) is given by:

- $$\begin{aligned} t_s &= f t_s + (1-f) t_s \\ &= f t_s + t_s - f t_s \\ &= t_s \end{aligned}$$

- $$t_p \geq f t_s + \frac{(1-f)t_s}{n}$$

-
- Hence, the Overall Speedup factor is given by :

$$S(n) = \frac{T(1)}{T(n)} = \frac{t_s}{t_p} = \frac{\text{Serial Execution time on single processor}}{\text{Parallel Execution time on } n \text{ multiple processor}}$$

$$S(n) = \frac{t_s}{t_p} \leq \frac{t_s}{f t_s + \frac{(1-f)t_s}{n}}$$

$$= \frac{1}{f + \frac{(1-f)}{n}}$$

$$S(n) \leq \frac{1}{f + \frac{(1-f)}{n}}$$

This is known as Amdahl's Law.

-
- Sequential operations will tend to dominate the speedup as n becomes very large.

$$\text{As } n \rightarrow \infty, S(n) \rightarrow \frac{1}{f}$$

- This means, no matter how many processor are employed, the speedup in this problem is limited to $\frac{1}{f}$.

This is known as sequential bottleneck of the problem.

- A major shortcoming in applying the Amdahl's is that the total workload or the problem size is fixed. Thus, execution time decreases with increasing number of processor.
- Thus, a successful method of overcoming this shortcoming is to increase the problem size.

EXAMPLE I

- 70% of a program's execution time occurs inside a loop that can be executed in parallel and rest 30% in serial. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPU?

⇒ Percentage of program which is parallel = 70% = 0.70

Percentage of program which is serial (f) = 30% = 0.30

Total number of processor (n) = 8

$$S(n) = \frac{1}{f + \frac{(1-f)}{n}} = \frac{1}{0.30 + \frac{(1-0.30)}{8}} = \frac{1}{0.3875} = 2.6$$

-
- 80% of a program's execution time occurs inside a loop that can be executed in parallel and rest 20% in serial. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPU?

GUSTAFSON'S LAW

- Gustafson proposed a change to Amdahl's law.
- Gustafson's Law states that instead of running the same size problem for all N , we can also consider solving largest problem size with greater resources with almost the same execution time.
- As the machine size(processor) increases, the work load(or problem size) is also increased so as to keep the fixed execution time for the problem.
- For scaled problems, where problem size increases with machine size(the number of processor)
- Let, t_s be the constant time taken to perform sequential operations; and

$t_p(n, W)$ be the time taken to perform parallel operation of problem size or workload W using n processor;

-
- Then the speedup with using n processors is:

$$s'(n) = \frac{\text{Serial Execution time on single processor}}{\text{Parallel Execution time on } n \text{ multiple processor}}$$
$$= \frac{t_s + t_p(1, W)}{t_s + t_p(n, W)}$$

- Assuming that parallel operations achieve a linear speedup (i.e these operations take $1/n$ of the time to perform on one processor)
- Then , $t_p(1, W) = n \cdot t_p(n, W)$
- Let f be the fraction of sequential work load in problem i.e

$$f = \frac{t_s}{t_s + t_p(n, W)}$$

-
- Then the speedup can be expressed as : with n processor is:

$$s'(n) = \frac{t_s + n \cdot t_p(n, W)}{t_s + t_p(n, W)}$$

$$= f + n(1-f)$$

$$[t_p(n, W) = 1-f]$$

$$= f + n - nf$$

$$= n - nf + f$$

$$= n - f(n-1)$$

$$s'(n) = n - f(n-1)$$