Assignment

Group no.”4”
SriHarsha Majeti - 15114044
Miryala Harshavardhan-15114045
Sanju Prabhath Reddy-15114042
Utsav Mangal-15114075
Dinesh Reddy-15114026

What is parallel computing?
A computation where multiple calculations or the execution of the processes are done simultaneously.
Generally when we try to solve a problem we do the following :
- Construction of algorithm
- Implementation of the algo with instructions in a serial manner **i.e** only one instruction at a time

But in Parallel computing we solve the problem using multiprocessing units [2 or more processors solving the problem].This is accomplished by breaking the problem into **independent** subparts and allocating and executing with each processor to each part simultaneously.
Types of parallel computing :
- Bit level
- Instruction level
- Data
- Task parallelism

What is distributed computing?
- Distributed computing is a model in which components of a software system are shared among multiple computers.
- Distributed computing is limited to a geographic area.It means that something is shared among multiple systems which may also be in different locations.
- Typically,this kind of distributed computing uses the client/server communications model.
- The Distributed Computing Environment(DCE) is a wide-used industry standard that provides this kind of distributed computing.
- The ultimate goal of Distributed computing is to maximise performance by connecting users and IT resources in a cost-effective,transparent and reliable manner.

- It also ensures fault tolerance and enables resource accessibility in the event that one of the component fails.

What is data parallelism?
The name itself suggests that it operates and works with data in parallel.Let's understand it with more clarity from the examples.

- Find the sum of elements of an array of size **n** assuming it takes **'t'** for every addition.
    - Addition in sequential manner takes **n*t** total time of execution.
    - But if we use data parallelism **i.e** let's divide the data of n-elements into **m** parts where each part contains **n/m** elements.Now allocate these m parts to m processors.These processors execute these m parts simultaneously in a total execution time of **(n/m)*t + merge-time.**Out of the two the latter is much better in performance of the given task.
- Similarly let us consider a image which contains large number of pixels.Using data parallelism we divide data of pixels into independent parts and procure the final result of desired task.

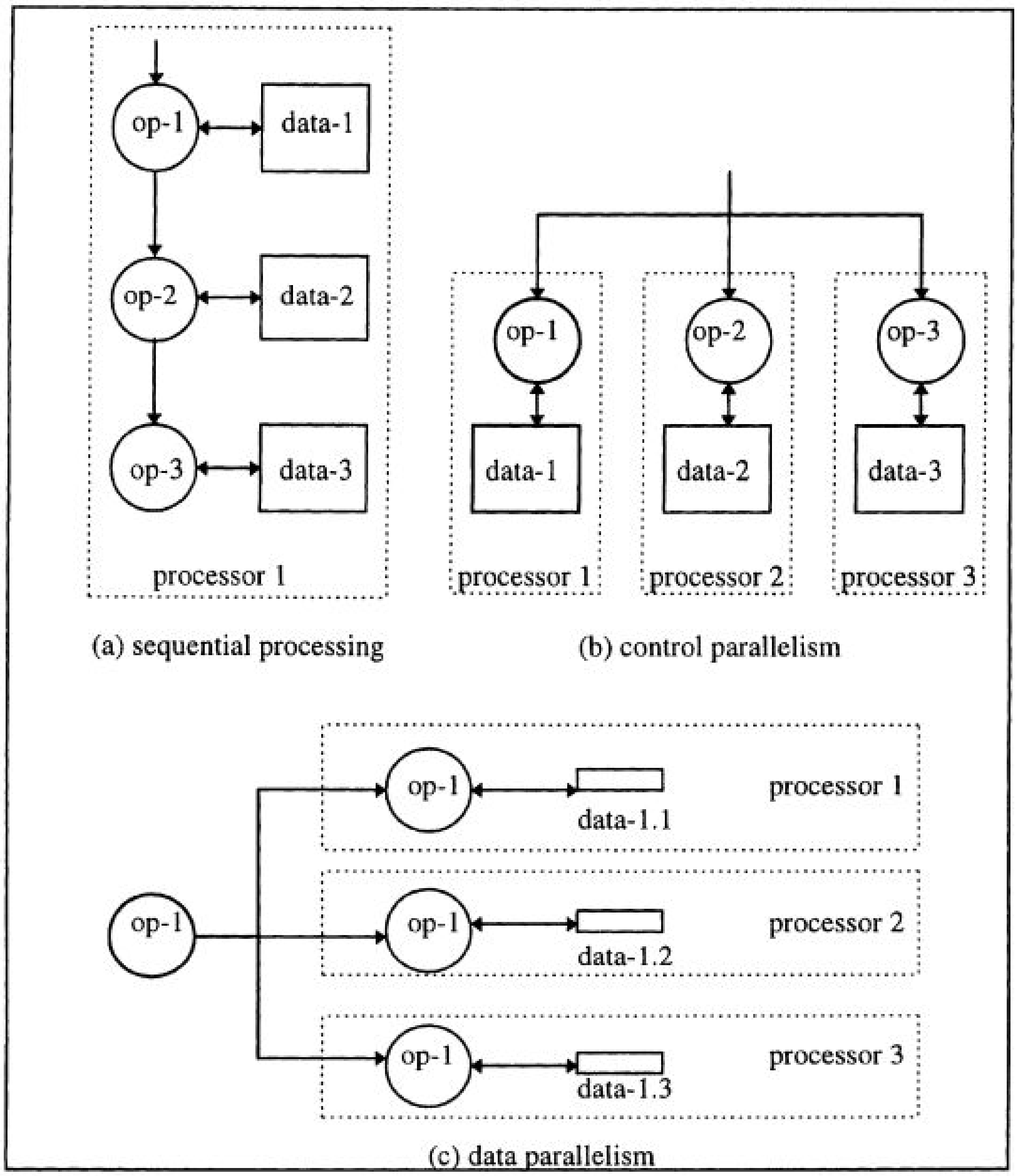
<u>Characteristics of data parallelism</u>

- Better speedup as there is one execution thread operating on all sets of data.
- Amount of parallelization is directly proportional to the input data size
- Optimum load balance on a multiprocessor system
- Used in matrix computation, convolutional neural network

What is Control Parallelism?

- Task parallelism is also known as <u>task parallelism</u> or <u>function parallelism</u> or <u>operational parallelism</u>.
- It is a form of parallelisation of code across multiple processors in parallel computing environments.
- It refers to the concurrent execution of multiple operations or instructions.
- In a multiprocessor system, task parallelism is achieved when each processor executes a different thread (or process) on the same or different data. The threads may execute the same or different code.
- In the general case, different execution threads communicate with one another as they work.
- Communication usually takes place by passing data from one thread to the next as part of a workflow.

- Task parallelism emphasizes the distributed (parallelized) nature of the processing (i.e. threads), as opposed to the data(data parallelism).



- **Thread-level parallelism** (**TLP**) is the parallelism inherent in an application that runs multiple threads at once.
- This type of parallelism is found largely in applications written for commercial servers such as databases.

- The exploitation of thread-level parallelism has also begun to make inroads into the desktop market with the advent of multi-core microprocessors.
- This has occurred because, for various reasons, it has become increasingly impractical to increase either the clock speed or instructions per clock of a single core.
- This contrasts with previous microprocessor innovations in which existing code was automatically sped up by running it on a newer/faster computer.

Data Parallelism	Task Parallelism
Same operations are performed on different subsets of same data.	Different operations are performed on the same or different data.
Synchronous computation	Asynchronous computation
Speedup is more as there is only one execution thread operating on all sets of data.	Speedup is less as each processor will execute a different thread or process on the same or different set of data.
Amount of parallelization is proportional to the input data size.	Amount of parallelization is proportional to the number of independent tasks to be performed.
Designed for optimum load balance on multi processor system.	Load balancing depends on the availability of the hardware and scheduling algorithms like static and dynamic scheduling.

Role of Stirred up Efficiency:

- A software system can be structured as a collection of largely independent subtasks, significant reductions in elapsed time can be realized by executing these subtasks in parallel on multiple processors.

- This effect, known as speedup,typically increases (up to some limit) with the number of processors dedicated to the problem.
- Along with an increase in speedup comes a decrease inefficiency: as more processors are devoted to the execution of a software system, the total amount of processor idle time can be expected to increase, due to factors such as contention, communication,and software structure.
- We bound the extent to which both speedup and efficiency can simultaneously be poor: we show that for any software system and any number of processors, the sum of the average processor utilization (i.e., efficiency) and the attained fraction of the maximum possible speedup must exceed One.
- We give bounds on speedup and efficiency, and on the incremental benefit and cost of allocating additional processors.We give an explicit formulation, as well as bounds, for the location of the “knee” of the execution time-efficiency profile,where the benefit per unit cost is maximized

<u>Amdahl’s law</u>

- Amdahl's Law is a law governing the speedup of using parallel processors on a problem, versus using only one serial processor.It implies that the overall performance improvement is limited by the range of impact when an optimization is applied.

- In simpler terms, it means that it is the algorithm that decides the speedup not the number of processors.In technical terms, the law is concerned with the speedup achievable from an improvement to a computation that affects a proportion P of that computation where the improvement has a speedup of S.

- Amdahl’s law states that the overall speedup of applying the improvement will be 1/[(1-P)+(P/S)]. The final speedup is computed by dividing the old running time by the new running time, which is what the above formula does.

- In the special case of parallelization, Amdahl’s law states that if F is the fraction of a calculation that is sequential (that cannot benefit from parallelization), and (1-F) is the fraction that can be parallelized, then the maximum speedup that can be achieved using N processors is 1/[F+{(1-F)/N}].
- Overall speedup = 1/[(1-f)+f/s]

  where, f= fraction of the program that is enhanced,

  s= speedup of the enhanced portion.
- Amdahl’s law only applies to cases where the problem size is fixed.

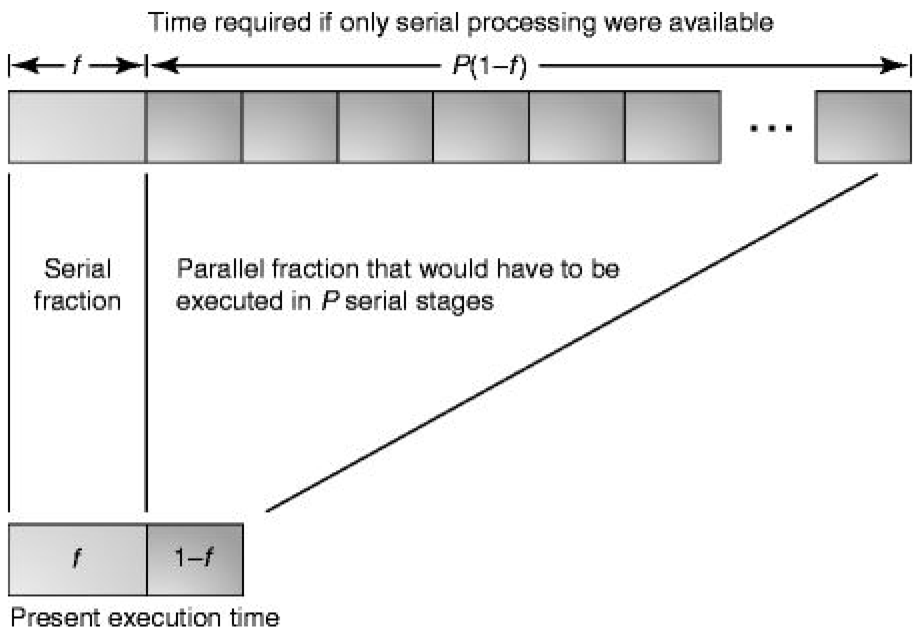
Gustafson's Law

Gustafson's Law says that if you apply *P* processors to a task that has serial fraction *f*, scaling the task to take the same amount of time as before, the speedup is

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

It shows more generally that the serial fraction does not theoretically limit parallel speed enhancement, if the problem or workload scales in its parallel component. It models a different situation from that of Amdahl's Law, which predicts time reduction for a fixed problem size.

Graphical Explanation


Time required if only serial processing were available
f
P(1−f)
Serial fraction
Parallel fraction that would have to be executed in P serial stages
f
1−f
Present execution time

Graphical derivation of Gustafson's Law

The time the user is willing to wait to solve the workload is unity (lower bar). The part of the work that is observably serial, f, is unaffected by parallelization. The remaining fraction of the work, $1 - f$, parallelizes perfectly so that a serial processor would take P times longer to execute it. The ratio of the top bar to the bottom bar is thus $f + P(1 - f)$. Some prefer to rearrange this algebraically as $P - f(P - 1)$.