Unit 4 : Analytical modelling of parallel program

Understanding the parallel system

1. A parallel system → Combination of an algorithm + Parallel Architecture
2. Time of parallel algorithm → Depends on Number of processing elements, their relative computation, interprocess communication speeds.

Sources of overhead in parallel programs

1. In addition to performing computation, a parallel program many also spend time in interprocess communication.
2. Profile indicates times spent on : Performing computation, communication and idling.
3. Interprocess interaction → Processes interact and communicate data. Time spent on processing elements is more contributing source of overhead.
4. Idling → Elements becoming idle due to load imbalnce, synchronisation and serial componenents.
5. Excess computation → different between computation of parallel and best serial program.

1. **Interprocess Communication**:
    - Parallel programs often consist of multiple processes or threads that need to communicate with each other to exchange data, synchronize their activities, or coordinate their execution.
    - Interprocess communication (IPC) overhead occurs when processes need to send messages, share data structures, or coordinate their activities,

which can introduce delays and impact overall performance.

- Examples of IPC mechanisms include message passing (e.g., MPI), shared memory, remote procedure calls (RPC), and synchronization primitives (e.g., mutexes, semaphores).

2. **Idling**:
    - Idling occurs when processing elements (e.g., CPU cores, threads) are not actively performing useful work due to various reasons.
    - Load imbalance: If the workload is not evenly distributed among processing elements, some elements may finish their tasks earlier and become idle while others are still working.
    - Synchronization: Processing elements may need to wait for synchronization points, such as barrier synchronization, which can cause idle time if some elements complete their work before others.
    - Serial components: Parallel programs often contain sequential sections of code that cannot be parallelized, such as initialization or cleanup tasks, which may lead to idle time for processing elements while waiting for these serial components to complete.
3. **Excess Computation**:
    - Excess computation refers to the additional computational overhead incurred by parallel programs compared to the best possible serial implementation of the same task.
    - Parallelization introduces overhead in terms of parallelization constructs, synchronization, communication, and load balancing, which may result in slower execution compared to an ideal, perfectly parallelized program.
    - The difference between the actual computation time of the parallel program and the computation time of the best serial program represents the excess computation overhead.

Performance measures and analysis

Metrics

1. Execution time

2. Parallel overhead
3. Speedup
4. Efficiency
5. Cost

Execution Time → Serial runtime is the time elapsed between the beginning and the end of its execution on a sequential computer. Parallel runtime → time elapsing from the moment a parallel computer starts to the moment the last processing element finishes execution.

Total parallel overhead → Overhead incurred by a parallel program is encapsulated into a single expression. Overhead function is the total overhead of a parallel system, as the total time collectively spent by all processing elements.

Speedup → Captures the relative benefit of solving a prb in parallel.

Ratio of time take to solve a prb on single elemnt to solving same prb on parallel computer with p processing elements.

Efficiency → Measure of the fraction of time for which a processing element is usefully employed, defined as ratio of speedup to number of processing elements.

Cost → Product of parallel runtime and the number of processing elements used. Cost → sum of the time that each processing element spends on the solving the problem.

> Parallel system is said to be cost-optimal if the cost of solving a problem on a parallel computer as the same asymptotic growth as a function of the input size as the fastest known sequential algorithm on a single processing element.

Effect of Granularity

1. **Granularity Levels**: Granularity refers to the amount of workload or the level of detail at which a problem is divided into tasks for parallel processing. It can vary from fine granularity (where tasks are small and numerous) to coarse granularity (where tasks are larger and fewer).
2. **Concurrency and Granularity**: As granularity becomes finer, the degree of concurrency increases, meaning more tasks can be executed simultaneously. Conversely, coarser granularity reduces concurrency.
3. **Performance Trade-offs**: The choice of granularity level is crucial as it affects performance. Fine-grained tasks increase parallelism and potential speedup, but they also introduce overheads such as synchronization and scheduling, which can degrade performance. Coarse-grained tasks may reduce overhead but limit parallelism and speedup potential.
4. **Scaling Down Parallel Systems**: Scaling down a parallel system involves using fewer processing elements than the maximum possible. One approach is to design a parallel algorithm for one input element per processing element and then use fewer processing elements to simulate a larger number. However, this may not always be cost-optimal due to increased computation per processing element and associated overheads.
5. **Cost-Optimality**: Cost-optimality depends on how computation is mapped onto processing elements. Scaling down may not always make a non-cost-optimal system cost-optimal. The choice of granularity and algorithm design plays a significant role in determining cost-optimality.

Minimum execution time and minimum cost

1. **Objective of Determining Minimum Execution Time**: In parallel computing, it's often important to know the fastest possible time a problem can be solved, assuming an unlimited number of processing elements. This helps in understanding the inherent efficiency of a parallel algorithm.
2. **Behavior with Increasing Processing Elements**: As the number of processing elements increases for a fixed problem size, the parallel runtime may continue

to decrease until it approaches a minimum value, or it may start increasing after reaching a certain point.

3. **Determining Minimum Parallel Runtime**: The minimum parallel runtime Tmin for a given problem size W can be found by differentiating the expression for parallel runtime $Tp(W,p)$ with respect to the number of processing elements pp and equating the derivative to zero. This helps in finding the optimal number of processing elements for minimum runtime.
4. **Isoefficiency Function**: This function indicates the relationship between the problem size, the number of processing elements, and the efficiency of parallel execution. Parallel systems where the runtime can be significantly reduced by using more processing elements than suggested by the isoefficiency function are rare.
5. **Degree of Concurrency**: The maximum number of processing elements that can be effectively utilized is bounded by the degree of concurrency $C(W)C(W)$ of the parallel algorithm. This ensures that adding more processing elements beyond this limit doesn't provide further speedup.
6. **Limitation of Processing Elements**: In some cases, the optimal number of processing elements $popo$ determined by the differentiation process may exceed the degree of concurrency $C(W)C(W)$ of the algorithm. In such scenarios, $popo$ becomes meaningless.