Ex: No:	SIMULATION OF INSTRUCTION LEVEL PARALLELISM

AIM:

To study the simulation of instruction level parallelism using c.

PROCEDURE:

- Pipelining can overlap the execution of construction when they are independent of one another. This potential overlap among instruction is called instruction level parallelism (IPL) since the instruction can be evaluated in parallel.
- The amount of parallelism available within a basic block is quite small. The average dynamic branch frequency in integer program was measured to be about 15%, meaning that about 7 instruction execute between a pair of branches.
- Since the instructions are likely to depend upon one another, the amount of overlap we can exploit within a basic block is likely to much less than 7.
- To obtain substantial performance enhancement, we must exploit IPL across multiple basic block.
- The simplest and most common way to increase the amount of parallelism available among instruction exploit parallelism among iteration of a loop. This type of parallelism often called loop-level parallelism.

Ex: No:	SIMULATION OF MULTI PROCESSOR

AIM:

To simulation of multiprocessor in c.

PROCEDURE:

- Multiprocessing is how a computer is able to executed multiple program concurrently.
- One of the main workhouse function that makes multiprocessing possible is a function in c \ c++ that a process & pairing an identical copy. A process is an instance of a program.

DEFINE SYSTEM ARCHITECTURE:

- Specify the number of processors, memory hierarchy, cache configurations, interconnection network, and shared resources. Determine the simulation time scale and granularity.

SELECT SIMULATION TOOLS OR FRAMEWORK:

- Choose simulation tools or frameworks suitable for your modeling needs. Options include discrete event simulators, cycle-accurate simulators, or high-level architecture simulators.

PROCESSOR MODELING:

- Model's each processor's architecture, instruction set, and pipeline structure. Include components like instruction fetch, decode, execution units, and caches. Choose an appropriate level of abstraction based on simulation goals.

MEMORY HIERARCHY MODELING:

- Model the memory hierarchy, including caches, main memory, and any other levels such as shared caches. Specify cache coherence protocols if applicable.

INTERCONNECTION NETWORK:

- Model the interconnection network that facilities communication between processors and memory. Choose the appropriate network topology and communication protocols.

SHARED RESOURCES MODELING:

- If the multi-processors system includes shared resources (e.g., buses, I\O devices), model their behavior and contention. Implement any necessary arbitration or scheduling mechanisms.

TASK OR THREAD CREATION:

- Define tasks or threads that will be executed on each processor. Specify their characteristics, such as arrival times, execution times, and dependencies.

SIMULATION INITIALIZATION:

- Set up the initial state of the system, including processor states, memory contents, and any initial tasks or threads.

SIMULATION EXECUTION:

- Execute the simulation over time, advancing in discrete steps or cycles. Model the execution of tasks on processors, memory accesses, cache hits\misses, and communication through the interconnection network.

Ex: No:	SIMULATION OF VECTOR PROCESSOR

AIM:

To study of vector processor.

PROCEDURE:

- A vector processor consists of scalar processor and a vector unit, which could be thought of as an independent functional unit capable of efficient vector operation.

VECTOR HARDWARE:

- Vector computer have hardware to perform the vector operation effectively. Operands cannot be used directly from memory rather are loaded into register and are loaded into register and are put back in register after the operation.

PIPELINING:

- The pipeline is divided up into individual segment, each of which is completely independent and involves no hardware sharing.
- This ability enable it to produce in result per clock as soon as the pipeline is full.
- The processing of a number of operands may carried out simultaneously.
- The loading of a vector register is itself a pipelined operation, with the ability to load one element each clock period after some initial startup overhead.

DEFINE ARCHITECTURE:

- Understand the architecture of the vector processor you want to simulate. This includes the structure of vector registers, instruction set, pipeline stages, and memory access patterns.

SELECT SIMULATION:

- Choose a simulation environment or platform. This could be general-purpose programming language (e.g., C++, Python) or specialized tools designed for processor simulation (e.g., Gem5, Simics).

DATA STRUCTURES:

- Implement data structures to represent vector registers, scalar registers, memory, and other components of the vector processor. Use appropriate data structures to efficiently model the vector operations.

INSTRUCTION SET ARCHITECTURE:

- Implement the instruction set of the vector processor. Define data types, vector instructions, and their corresponding opcodes. Map these instructions to corresponding functions or methods in your simulation.

PIPELINE MODEL:

- If the vector processor has a pipeline, model each stage of the pipeline. Implement functions for instruction fetch decode, execute, and write-back. Handle hazards and dependence between instructions.

MEMORY HIERARCHY:

- Simulate the memory hierarchy, including caches and main memory. Important cache management, including cache hits and misses.

VECTOR OPERATIONS:

- Implement the core vector operations based on the processor's capabilities. This includes vector addition, multiplication, and other vectorized computations.

Ex: No:	SIMULATION OF DATA PARALLELISM

AIM:

To study simulation of data parallelism.

PROCEDURE:

- Data parallelism is parallelization across multiple process in parallel.
- It focuses on distributing the data across different nodes, which operate on the data in parallel.
- It contrasts task parallelism as another from of parallelism.
- A data parallel job on an array of n element can be divided equally among all the process.
- Let us assume we want to sum all the element of the given array and the for a single addition operation is time unit.
- One important that they to node is that the locality of the data reference play an important part in evaluating the performance of a data parallel programming model.
- In a multiprocessor register executing a single set of instruction (SIMD), data parallelism is achieve when each processor perform the same task on different distributed data.

DEFINE THE PROBLEM:

- Clearly define the problem you want to simulate with data-level parallelism. Identify the data-level parallelism opportunities within the algorithm or application.

SELECT A SIMULATION TOOL OR ENVIRONMENT:

- Choose a simulation tool or environment that supports the modeling of parallel processing. This could be a general-purpose simulation tool, a parallel programming framework, or a domain-specific simulation platform.

MODEL THE ALGORITHM:

- Develop a model or representation of the algorithm you want to simulate. Identify the sections of the algorithm where data-level parallelism can be exploited. This may involve breaking down the algorithm into tasks or operations that can be performed concurrently.

PARALLELIZE DATA OPERATIONS:

- Identify data-level parallelism within the algorithm and parallelize the relevant data operations. This might include tasks like vectorization, SIMD (Single Instruction, Multiple Data) parallelism, or using parallel processing constructs in your chosen simulation environment.

DESIGN PARALLEL DATA STRUCTURES:

- Modify or design data structures that support parallel processing. Consider how data is distributed among processing units and how communication and synchronization between these units will occur.

IMPLEMENT PARALLELIZATION TECHNIQUES:

- Implement the identified parallelization techniques within you simulation environment. This might involve utilizing parallel programming constructs such as threads or processors, GPU acceleration, or specialized parallel processing libraries.

INSTRUMENTATION AND MEASUREMENT:

- Introduce instrumentation into your simulation to measure and analyze the performance of the parallelized algorithm. Track metrics such as execution time, speedup, and efficiency.