COURSE CONTENT

PARALLEL COMPUTATION: RUST

III Semester: CSM

IV Semester: CSD / CSE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
24X0598	Core	0	0	2	1	40	60	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 30				Total Classes: 30		

Prerequisites: Problem Solving with C and C++

Course Overview:

This course provides a comprehensive foundation in **parallel computation using Rust**, focusing on safe and efficient concurrent programming. Students learn to design and implement parallel algorithms using Rust's ownership and type system. The course emphasizes thread safety, data sharing, and synchronization mechanisms. Learners gain hands-on experience with Rust concurrency primitives and parallel libraries. Performance optimization and scalability on multi-core systems are covered. By the end, students can build reliable and high-performance parallel Rust applications.

Course Objectives:

1. To understand the fundamentals of parallel and concurrent computation .
2. To learn the steps involved in developing parallel programs in Rust.
3. To understand the syntax and semantics of the Rust programming language.
4. To apply Rust's ownership model for safe concurrency.
5. To use structured parallel programming techniques to solve computational problems.

Course Outcomes: After Completion of the Course, Students should be able to

1. Apply parallel control structures and concurrency constructs to solve computational problems efficiently.
2. Design modular and scalable Rust programs using threads, tasks, and safe data-sharing mechanisms.
3. Develop parallel programs using collections, iterators, and data-parallel patterns.
4. Analyze and implement recursive and concurrent solutions using Rust's ownership, traits, and synchronization primitives.
5. Build high-performance applications involving parallel algorithms, file processing, and efficient data handling on multi-core systems.

Module I – Basics of Rust

1. Hello, World!
   - Basic Rust program structure using fn main().
2. Variable Declaration and Mutability
   - Using let, mut, and type inference.
3. Data Types Demo
   - Use i32, f64, char, bool, etc., in a single program.
4. Arithmetic and Logical Operators

- Calculator-style program showing arithmetic and logical operations.

5. Using cargo for a Simple Project
   - Create a simple project using cargo new, build, and run it.

Module II – Ownership and Control Flow

6. Ownership Transfer
   - Pass variables to functions and explore ownership and move.
6. Borrowing and References
   - Demonstrate &T and &mut T references and restrictions.
7. Lifetimes Example
   - Use lifetime annotations in a function returning references.
8. If-Else Statement
   - Even/odd checker or grade calculator.
9. Looping Examples
   - Use for, while, and loop to iterate over arrays or counters.
10. Pattern Matching with match
    - Match numbers to print weekdays or match enums.
11. Nested Control Statements
    - Combine loops and conditionals in one program (e.g., a simple number guessing game).

Module III – Functions and Structs

12. Functions with Arguments and Return Values
    - Create a function to calculate factorial or square of a number.
13. Ownership in Functions
    - Pass by value vs. reference in function parameters.
14. Using Structs
    - Define a Rectangle struct and calculate area.
15. Tuple Structs and Field Init Shorthand
    - Define a Color(u8, u8, u8) and use field shorthand for init.
16. Enums and Pattern Matching
    - Define an enum for TrafficLight and match its values.
17. Option Enum Usage
    - Safe division function returning Option<f64>.

Module IV – Project Management and Smart Pointers

18. Error Handling with Result
    - File reading or division with error handling using Result.
19. Using Box, Rc, and RefCell
    - Demonstrate smart pointers with a simple linked list or counter program.

Module- V – Object oriented Programming

20. "Polymorphic Behavior with Trait Objects: A Speakable Animal Zoo"
    Covers: Traits, dynamic dispatch, trait objects (&dyn Trait)
22. "Implementing Strategy Pattern with Traits for Payment Processing" Covers: Object-oriented design using traits and Box<dyn Trait> for dynamic strategy switching
23. "Trait-Based Drawing Application with Heterogeneous UI Components" Covers: Object-oriented characteristics, trait objects, and allowing different drawable types in a single collection

TEXT BOOKS:

1. Nicholas Matsakis and Aaron Turon, *The Rust Programming Language*, 2nd Edition, No Starch Press.
2. Michael McNamara, *Rust in Action*, Manning Publications.

REFERENCE BOOKS:

1. Jim Blandy, Jason Orendorff, and Leonora F. S. Tindall, *Programming Rust*, O'Reilly Media.
2. Mara Bos, *Rust Atomics and Locks*, O'Reilly Media.
3. Anthony Williams, *C++ Concurrency in Action* (for concurrency concepts), Manning Publications.
4. Tim McNamara, *Modern Systems Programming with Rust*, Manning Publications.
5. Brian L. Troutwine, *Rust Web Programming*, Packt Publishing.
6. Alex Crichton and Steve Klabnik, *The Rust Programming Language Cookbook*, O'Reilly Media.

ELECTRONIC RESOURCES:

1. **https://www.geeksforgeeks.org/rust/introduction-to-rust-programming-language/**
2. **https://www.w3schools.com/rust/**
3. **https://www.programiz.com/rust**
4. **https://thenewstack.io/rust-programming-language-guide/**
5. **https://en.wikipedia.org/wiki/Rust_(programming_language)**

MATERIALS ONLINE:

1. Course template
2. Open-ended experiments
3. Definitions and terminology
4. Lab Manual