

NATIONAL UNIVERSITY OF SAN MARCOS

FACULTY OF MATHEMATICAL SCIENCES

SYLLABUS

1. GENERAL INFORMATION

1.1 Course name	:	Parallel and Distributed Computing
1.2 Code	:	CS3P1
1.3 Academic Program	:	Scientific Computing
1.4 Credits	:	3
1.5 Prerrequisites	:	CS210. Algorithms and Data Structures. (4^{th} Sem)
1.6 Type of Course	:	Mandatory
1.7 Area of Studies	:	Estudios Generales
1.8 Number of Weeks	:	16
1.9 Hours per week	:	1 HT; 4 HP;
1.10 Cycle	:	I
1.11 Learning modality	:	Face to face
1.12 Academic Period	:	2026-I
1.13 Semester Start Date	:	
1.14 Semester End Date	:	

2. PROFESSORS

Meetings after coordination with the professor

3. INTRODUCTION TO THE COURSE

With the end of frequency scaling in single-core processors, parallel and distributed computing has become essential for high-performance software development. This course introduces students to the programming models, communication patterns, and coordination mechanisms required to exploit multi-core systems and distributed clusters. Students will learn to design scalable algorithms and evaluate their performance under different computational constraints.

4. GOALS

- Design and implement parallel programs using shared and distributed memory.
- Master communication and synchronization protocols in distributed systems.
- Evaluate performance and scalability through formal metrics.
- Apply parallel algorithms to solve complex computational problems.

5. COMPETENCES

ABET-3) An ability to develop and conduct experiments or test hypotheses, analyze and interpret data, and use scientific judgment to draw conclusions. (Usage)

ABET-2) An ability to formulate or design a system, process, procedure, or program for the intended purpose. (Usage)

6. TOPICS

Unit 1: Programs: Declarative Parallelism (4 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Parallelism – Declarative parallelism: Determining which actions can, or must not, be performed in parallel, at the level of instructions, functions, closures, compound actions, sessions, tasks, and services is the core idea underlying PDC algorithms; failing to do so is the primary source of errors. See also: Algorithms – Defining order: for example, using "happens-before" relationships or series-parallel directed acyclic graphs (DAGs) representing programs. – Independence: determining when order does not matter, in terms of commutativity, dependencies, preconditions. – Guaranteeing order between otherwise parallel actions when necessary, including locks, safe publication; and enforcing communication - sending a message happens before receiving it; and relaxing it when not necessary.	• Graphically represent (as a Directed Acyclic Graph - DAG) how to parallelize a compound numeric expression; for example, $a = (b + c) * (d + e)$ [Design] • Explain why consistency and fault tolerance concepts do not arise in purely sequential programs [Explain]
Readings : [**Pacheco2021**]	

Unit 2: Programs: Starting Activities (3 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Starting activities – Options allowing actions to be performed (eventually) at locations range from hardwired to configuration scripts; also establishing communication and resource management; these are expressed differently in languages and contexts, typically relying on automated provisioning and management by platforms. See also: Resource Management – Procedural: Allowing multiple actions to start at a given program point; for example, starting new threads, possibly delimiting their scope or organizing them into hierarchical groups. – Reactive: Enabling upon event occurrence by installing an event handler, with less control over when actions start or finish, and may apply even on single processors. – Dependent: Enabling upon completion of others; for example, sequencing sets of parallel actions. See also: Coordination. – Granularity: The execution cost of action bodies should outweigh the overhead of organizing them.	• Write a service that creates a thread (or other form of procedural activation) to return a requested web page to each new client [Write]
Readings : [**Pacheco2021**]	

Unit 3: Programs: Execution Properties (3 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Execution Properties – Non-deterministic execution of actions without order. – Consistency: Ensuring agreement among parties on values and predicates when necessary to avoid races, maintain safety and atomicity, or reach consensus. – Fault tolerance: Handling failures in parts or communication, including (Byzantine) misbehavior due to unreliable parties and protocols, when necessary to maintain progress or availability. See also: System Reliability. – Trade-offs are a focus of evaluation. See also: Evaluation.	• Write a function that counts events, such as network packet receptions, efficiently [Write]
Readings : [**VanSteen2023**]	

Unit 4: Programs: Distribution (2 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Distribution – Defining locations as devices that execute actions, including hardware components, remote hosts; may also include external uncontrolled devices, hosts, and users. See also: Interfacing and Communication – A device may time-share or emulate multiple parallel actions using fewer processors through scheduling and virtualization. See also: Scheduling – Naming or identifying locations (e.g., device IDs) and actions as parts (e.g., thread IDs). – Activities across locations may communicate through media. See also: Communication	• Write a filter/map/reduce program in multiple styles [Write]
Readings : [**VanSteen2023**]	

Unit 5: Programs: Implementation Mappings (2 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• One or more of the following mappings and mechanisms across layered systems: – Instruction-level and data parallelism in the CPU. See also: Functional Organization. – SIMD and heterogeneous data parallelism. See also: Heterogeneous Architectures. – Scheduled concurrency on multi-core, tasks, actors. See also: Scheduling. – Clusters, clouds; elastic provisioning. See also: Common Aspects: Platforms, APIs and Constraints, Common Aspects: Platform Languages and Web Programming Patterns. – Networked distributed systems. See also: Networked Applications. – Emerging technologies such as quantum computing and molecular computing.	• Explain the trade-offs between different mapping strategies in terms of performance, cost, and implementation complexity [Explain]
Readings : [**Pacheco2021**], [**Hwu2022**]	

Unit 6: GPU Programming (4 hours)

Competences Expected: ABET-2

Topics	**Learning Outcomes**
• General-Purpose GPU Computing (GPGPU) – GPU architecture overview: streaming multiprocessors (SMs), CUDA cores, and warp execution model. – SIMT (Single Instruction, Multiple Threads) execution model and its differences from CPU SIMD. – Host-device interaction: data transfer between CPU (host) and GPU (device) memory via PCIe. – Use cases: scientific computing, machine learning, image processing, and large-scale simulation. • CUDA Programming Model – CUDA execution hierarchy: grids, thread blocks, and individual threads; mapping to GPU hardware. – Kernel definition and launch syntax; configuration parameters (grid and block dimensions). – Thread indexing: threadIdx, blockIdx, blockDim, gridDim; computing global indices in 1D, 2D, and 3D. – Warp divergence: performance implications of branching within a warp. – Intra-block synchronization primitive: __syncthreads(). • CUDA Memory Hierarchy – Global memory: large, high-latency, accessible by all threads; coalesced access patterns for performance. – Shared memory: low-latency on-chip memory shared within a thread block; bank conflicts. – Registers and local memory: per-thread private storage. – Constant and texture memory: read-only caches optimized for specific access patterns. – Unified Memory: simplified programming model with automatic host-device data migration.	• Write a CUDA kernel that performs a data-parallel operation (e.g., vector addition, matrix multiplication) and correctly configure grid and block dimensions [Write] • Analyze the impact of global memory access patterns (coalesced vs. strided) and shared memory usage on GPU kernel performance [Analyze] • Design a parallel algorithm using the CUDA programming model, including thread hierarchy, memory allocation strategy, and host-device data transfers [Design]

Readings : [**Kirk2016**], [**CUDADocs**]

Unit 7: Communication (8 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Media – Varieties: channels (message passing or I/O), shared memory, heterogeneous, data stores – Reliance on the availability and nature of underlying hardware, connectivity, and protocols; language support, emulation. See also: Interfacing and Communication • Channels – Explicit (usually named) party-to-party communication media – APIs: Sockets, architectural, language-based, and toolkit constructs, such as Message Passing Interface (MPI), and layered constructs such as Remote Procedure Call (RPC). See also: Fundamentals of Networks and Communications – I/O channel APIs • Memory – Shared memory architectures in which parties directly communicate only with memory at given addresses, with extensions to heterogeneous memory supporting multiple memory stores with explicit data transfer across them; for example, GPU local and shared memory, Direct Memory Access (DMA) – Memory hierarchies: Multiple layers of sharing domains, scopes, and caches; locality: latency, false-sharing – Consistency properties: Bitwise atomicity limits, coherence, local ordering • Data Stores – Cooperatively maintained data structures implementing maps and related ADTs – Varieties: Owned, shared, sharded, replicated, immutable, versioned	• Explain the similarities and differences among: (1) Party A sends a message on channel X with contents 1 received by party B (2) A sets shared variable X to 1, read by B (3) A sets "X=1" in a distributed shared map accessed by B [Explain] • Write a program that distributes different segments of a data set to multiple workers, and collects results (for the simplest example, summing segments of an array) [Write] • Write a parallel program that requests data from multiple sites and summarizes them using some form of reduction [Write] • Compare the performance of buffered versus unbuffered versions of a producer-consumer program [Compare]
Readings : [**VanSteen2023**], [**Kleppmann2017**]	

Unit 8: Communication: Properties and Extensions (3 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• One or more of the following properties and extensions – Topologies: Unicast, Multicast, Mailboxes, Switches; Routing via hardware and software interconnection networks – Media concurrency properties: Ordering, consistency, idempotency, overlapping communication with computation – Media performance: Latency, bandwidth (throughput) contention (congestion), responsiveness (liveness), reliability (error and drop rates), protocol-based progress (acks, timeouts, mediation) – Media security properties: integrity, privacy, authentication, authorization. See also: Information Flow and Non-Interference, Injection and Input Validation, Memory Safety and Types, Malware Analysis and Advanced Security – Data formats: Marshaling, validation, encryption, compression – Channel policies: Endpoints, sessions, buffering, saturation response (waiting vs dropping), rate control – Multiplexing and demultiplexing many relatively slow I/O devices or parties; completion-based and scheduler-based techniques; async-await, select and polling APIs – Formalization and analysis of channel communication; for example, CSP – Applications of queuing theory to model and predict performance.	• Determine whether a given communication scheme provides sufficient security properties for a given usage [Determine] • Give an example of a scenario in which blocking message sends can deadlock [Create] • Describe at least one design technique for avoiding liveness failures in programs using multiple locks [Describe]
Readings : [**VanSteen2023**], [**Kleppmann2017**]	

Unit 9: Memory and Consistency (3 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Memory models, data consistency, and fault tolerance – Memory models: sequential and release/acquire consistency – Memory management; including reclamation of shared data; reference counts and alternatives – Bulk data placement and transfer; reducing message traffic and improving locality; overlapping data transfer and computation; impact of data layout such as array-of-structs vs struct-of-arrays – Emulating shared memory: distributed shared memory, Remote Direct Memory Access (RDMA) – Data store consistency: Atomicity, linearizability, transactionality, coherence, causal ordering, conflict resolution, eventual consistency, blockchains – Faults, partitioning, and partial failures; voting; protocols such as Paxos and Raft. – Design tradeoffs among consistency, availability, partition (fault) tolerance; impossibility of meeting all at once – Security and trust: Byzantine failures, proof of work and alternatives	• Give an example of an ordering of accesses among concurrent activities (e.g., program with a data race) that is not sequentially consistent [Create] • Write a program that illustrates memory-access or message reordering [Write] • Describe the relative merits of optimistic versus conservative concurrency control under different rates of contention among updates [Describe] • Give an example of a scenario in which an attempted optimistic update may never complete [Create] • Modify a concurrent system to use a more scalable, reliable, or available data store [Create] • Using an existing platform supporting replicated data stores, write a program that maintains a key-value mapping even when one or more hosts fail [Write]
Readings : [**VanSteen2023**], [**Kleppmann2017**]	

Unit 10: Coordination (7 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Dependencies – Initiation or progress of one activity may be dependent on other activities, so as to avoid race conditions, ensure termination, or meet other requirements – Ensuring progress by avoiding dependency cycles, using monotonic conditions, removing inessential dependencies • Control constructs and design patterns – Completion-based: Barriers, joins, including termination control – Data-enabled: Queues, producer-consumer designs – Condition-based: Polling, retrying, backoffs, helping, suspension, signaling, timeouts – Reactive: Enabling and triggering continuations	• Show how to ensure that a program correctly terminates when all of a set of concurrent tasks have completed [Design] • Write a function that efficiently counts events such as sensor inputs or networking packet receptions [Write] • Write a filter/map/reduce program in multiple styles [Write] • Write a program in which the termination of one set of parallel actions is followed by another [Write] • Write a service that creates a thread (or other procedural form of activation) to return a requested web page to each new client [Write]
Readings : [**Herlihy2020**], [**VanSteen2023**]	

Unit 11: Coordination: Synchronization and Atomicity (3 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Atomicity – Atomic instructions, enforced local access orderings – Locks and mutual exclusion; lock granularity – Using locks in a specific language; maintaining liveness without introducing races – Deadlock avoidance: Ordering, coarsening, randomized retries; backoffs, encapsulation via lock managers – Common errors: Failing to lock or unlock when necessary, holding locks while invoking unknown operations – Avoiding locks: replication, read-only, ownership, and non-blocking constructions	• Show how to avoid or repair a race error in a given program [Analyze]
Readings : [**Herlihy2020**]	

Unit 12: Coordination: Advanced Properties (4 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• One or more of the following properties and extensions – Progress properties including lock-free, wait-free, fairness, priority scheduling, interactions with consistency, reliability – Performance with respect to contention, granularity, convoying, scaling – Non-blocking data structures and algorithms – Ownership and resource control – Lock variants and alternatives: sequence locks, read-write locks; Read-Copy-Update (RCU), reentrancy; tickets; controlling spinning versus blocking – Transaction-based control: Optimistic and conservative – Distributed locking: reliability – Alternatives to barriers: Clocks; counters, virtual clocks; dataflow and continuations; futures and RPC; consensus-based, gathering results with reducers and collectors – Speculation, selection, cancellation; observability and security consequences – Resource control using semaphores and condition variables – Control flow: Scheduling computations, series-parallel loops with (possibly elected) leaders, pipelines and streams, nested parallelism – Exceptions and failures. Handlers, detection, timeouts, fault tolerance, voting	• Write a program that speculatively searches for a solution by multiple activities, terminating others when one is found [Write] • Write a program in which a numerical exception (such as divide by zero) in one activity causes termination of others [Write] • Write a program for multiple parties to agree upon the current time of day; discuss its limitations compared to protocols such as network transfer protocol (NTP) [Write]
Readings : [**Herlihy2020**]	

Unit 13: Evaluation (7 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Safety and liveness requirements in terms of temporal logic constructs to express "always" and "eventually" See also: Parallel and Distributed Computing • Identifying, testing for, and repairing violations, including common forms of errors such as failure to ensure necessary ordering (race errors), atomicity (including check-then-act errors), and termination (livelock) • Performance requirements metrics for throughput, responsiveness, latency, availability, energy consumption, scalability, resource usage, communication costs, waiting and rate control, fairness; service level agreements. See also: Latency, Cache and Memory Hierarchy, Virtualization and Isolation • Performance impact of design and implementation choices, including granularity, overhead, consensus costs, and energy consumption. See also: Sustainable Design and Pervasive Computing, Environmental Footprint of Computing Systems, Systemic Effects and Social Context • Estimating scalability limitations, for example using Amdahl's Law or Universal Scalability Law. See also: Events, Tools and Experimentation, Performance Metrics and Benchmarks, Analytical Performance Models	• Revise a specification to enable parallelism and distribution without violating other essential properties or features [Redesign] • Explain how concurrent notions of safety and liveness extend their sequential counterparts [Explain] • Specify a set of invariants that must hold at each bulk-parallel step of a computation [Analyze] • Write a test program that can reveal a data race error; for example, missing an update when two activities both try to increment a variable [Write] • In a given context, explain the extent to which introducing parallelism in an otherwise sequential program would be expected to improve throughput and/or reduce latency, and how it may impact energy efficiency [Explain] • Show how scaling and efficiency change for sample problems without and with the assumption of problem size changing with the number of processors; further explain whether and how scalability would change under relaxations of sequential dependencies [Design]
Readings : [**Sterling2024**], [**Pacheco2021**]	

Unit 14: Evaluation: Formal Methods (3 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Formal verification and analysis methods – Extensions to formal sequential requirements such as linearizability – Protocol, session, and transactional specifications – Use of tools such as Unified Modelling Language (UML), Temporal Logic of Actions (TLA), program logics – Security analysis: safety and liveness in the presence of hostile or buggy behaviors by other parties; required properties of communication mechanisms (for example lack of cross-layer leakage), input screening, rate limiting. See also: AI and Hardware Security, Access Control and Applications, Security Mindset and Principles – Static analysis applied to correctness, throughput, latency, resources, energy. See also: Sustainable Design and Pervasive Computing, Environmental Footprint of Computing Systems, Systemic Effects and Social Context – Directed Acyclic Graph (DAG) model analysis of algorithmic efficiency (work, span, critical paths)	• Specify and measure behavior when a service is requested by unexpectedly many clients [Analyze] • Identify and repair a performance problem due to sequential bottlenecks [Analyze] • Empirically compare throughput of two implementations of a common design (perhaps using an existing test harness framework) [Compare]
Readings : [**Sterling2024**], [**Pacheco2021**]	

Unit 15: Evaluation: Testing and Measurement (2 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Testing tools and measurement techniques – Testing and debugging; tools such as race detectors, fuzzers, lock dependency checkers, unit/stress/torture tests, visualizations, continuous integration, continuous deployment, and test generators – Measuring and comparing throughput, overhead, waiting, contention, communication, data movement, locality, resource usage, behavior in the presence of excessive numbers of events, clients, or threads. See also: Events, Tools and Experimentation, Performance Metrics and Benchmarks, Analytical Performance Models – Application domain specific analyses and evaluation techniques	• Identify and repair a performance problem due to communication or data latency [Analyze] • Identify and repair a performance problem due to resource management overhead [Analyze] • Identify and repair a reliability or availability problem [Analyze]
Readings : [**Sterling2024**], [**Pacheco2021**]	

Unit 16: Algorithms (4 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Expressing and implementing algorithms in given languages and frameworks, to initiate activities (for example threads), use shared memory constructs, and channel, socket, and/or remote procedure call APIs. See also: Parallel and Distributed Computing. – Data parallel examples including map/reduce. – Using channel, socket, and/or RPC APIs in a given language, with program control for sending (usually procedural) vs receiving. (usually reactive or RPC-based). – Using locks, barriers, and/or synchronizers to maintain liveness without introducing races.	• Implement a parallel/distributed component based on a known algorithm [Implement] • Write a data-parallel program that for example computes the average of an array of numbers [Write] • Write a producer-consumer program in which one component generates numbers, and another computes their average. Measure speedups when the numbers are small scalars versus large multi-precision values [Write]
Readings : [**Pacheco2021**], [**Herlihy2020**]	

Unit 17: Algorithms: Application Domains Survey (4 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• Survey of common application domains across multicore, reactive, data parallel, cluster, cloud, open distributed systems, and frameworks (with reference to the following table). – Multicore: Typical Execution agents: Threads. Typical Communication mechanisms: Shared memory, Atomics, locks. Typical Algorithmic domains: Resource management, data processing. Typical Engineering goals: Throughput, latency, energy. – Reactive: Typical Execution agents: Handlers, threads. Typical Communication mechanisms: I/O Channels. Typical Algorithmic domains: Services, real-time. Typical Engineering goals: Latency. – Data parallel: Typical Execution agents: GPU, SIMD, accelerators, hybrid. Typical Communication mechanisms: Heterogeneous memory. Typical Algorithmic domains: Linear algebra, graphics, data analysis. Typical Engineering goals: Throughput, energy. – Cluster: Typical Execution agents: Managed hosts. Typical Communication mechanisms: Sockets, channels. Typical Algorithmic domains: Simulation, data analysis. Typical Engineering goals: Throughput. – Cloud: Typical Execution agents: Provisioned hosts. Typical Communication mechanisms: Service APIs. Typical Algorithmic domains: Web applications. Typical Engineering goals: Scalability. – Open distributed: Typical Execution agents: Autonomous hosts. Typical Communication mechanisms: Sockets, Data stores. Typical Algorithmic domains: Fault tolerant data stores and services. Typical Engineering goals: Reliability.	• Extend an event-driven sequential program by establishing a new activity in an event handler (for example a new thread in a GUI action handler) [Design] • Improve the performance of a sequential component by introducing parallelism and/or distribution [Create] • Choose among different parallel/distributed designs for components of a given system [Assess]
Readings : [**Pacheco2021**], [**Herlihy2020**]	

Unit 18: Algorithms: Algorithmic Domains (4 hours)	
Competences Expected: ABET-2,ABET-3	
Topics	**Learning Outcomes**
• One of more of the following algorithmic domains. See also: Algorithmic Strategies: – Linear algebra: Vector and matrix operations, numerical precision/stability, applications in data analytics and machine learning. – Data processing: sorting, searching and retrieval, concurrent data structures. – Graphs, search, and combinatorics: Marking, edge-parallelization, bounding, speculation, network-based analytics. – Modeling and simulation: differential equations; randomization, N-body problems, genetic algorithms. – Computational logic: satisfiability (SAT), concurrent logic programming. – Graphics and computational geometry: Transforms, rendering, ray-tracing. – Resource management: Allocating, placing, recycling and scheduling processors, memory, channels, and hosts; exclusive vs shared resources; static, dynamic and elastic algorithms; Real-time constraints; Batching, prioritization, partitioning; decentralization via work-stealing and related techniques. – Services: Implementing web APIs, electronic currency, transaction systems, multiplayer games.	• Design, implement, analyze, and evaluate a component or application for X operating in a given context, where X is in one of the listed domains, for example, a genetic algorithm for factory floor design [Design] • Critique the design and implementation of an existing component or application, or one developed by classmates [Critique] • Compare the performance and energy efficiency of multiple implementations of a similar design, for example, multicore versus clustered versus GPU [Compare]
Readings : [**Pacheco2021**], [**Herlihy2020**]	

7. WORKPLAN

7.1 Methodology

Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

7.2 Theory Sessions

The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

7.3 Practical Sessions

The practical sessions are held in class where a series of exercises and/or practical concepts are developed through problem solving, problem solving, specific exercises and/or in application contexts.

8. EVALUATION SYSTEM

********* EVALUATION MISSING ********

9. BASIC BIBLIOGRAPHY