San Pablo Catholic University (UCSP)
Undergraduate Program in Computer Science
SILABO

CS3P1. Parallel and Distributed Computing (Mandatory)

2020-I

1. General information

1.1 School	:	Ciencia de la Computación
1.2 Course	:	CS3P1. Parallel and Distributed Computing
1.3 Semester	:	8^{vo} Semestre.
1.4 Prerrequisites	:	• CS212. Algorithm Analysis and Design. (5^{th} Sem) • CS231. Networking and Communication. (7^{th} Sem)
1.5 Type of course	:	Mandatory
1.6 Learning modality	:	Virtual
1.7 Horas	:	2 HT; 2 HP; 2 HL;
1.8 Credits	:	4

2. Professors

Lecturer

- Alvaro Henry Mamani-Aliaga <ahmamani@ucsp.edu.pe>
  - PhD in Ciencia de la Computación, UNSA, Perú, 2019.
  - MSc in Ciencia de la Computación, IME-USP, Brasil, 2011.

3. Course foundation

The last decade has brought explosive growth in computing with multiprocessors, including Multi-core processors and distributed data centers. As a result, computing parallel and distributed has become a widely elective subject to be one of the main components in the mesh studies in computer science undergraduate. Both parallel and distributed computing the simultaneous execution of multiple processes, whose operations have the potential to intercalar in a complex way. Parallel and distributed computing builds on foundations in many areas, including understanding the fundamental concepts of systems, such as: concurrency and parallel execution, consistency in state / memory manipulation, and latency. The communication and coordination between processes has its foundations in the passage of messages and models of shared memory of computing and algorithmic concepts like atomicity, consensus and conditional waiting.Achieving acceleration in practice requires an understanding of parallel algorithms, strategies for decomposition problem, systems architecture, implementation strategies and analysis of performance. Distributed systems highlight the problems of security and tolerance to Failures, emphasize the maintenance of the replicated state and introduce additional problems in the field of computer networks.

4. Summary

1. Parallelism Fundamentals 2. Parallel Architecture 3. Parallel Decomposition 4. Communication and Coordination 5. Parallel Algorithms, Analysis, and Programming 6. Parallel Performance

5. Generales Goals
• That the student is able to create parallel applications of medium complexity by efficiently leveraging machines with multiple cores. • That the student is able to compare sequential and parallel applications. • That the student is able to convert, when the situation warrants, sequential applications to parallel efficiently

6. Contribution to Outcomes
This discipline contributes to the achievement of the following outcomes: **a)** An ability to apply knowledge of mathematics, science. (**Usage**) **b)** An ability to design and conduct experiments, as well as to analyze and interpret data. (**Usage**) **i)** An ability to use the techniques, skills, and modern computing tools necessary for computing practice. (**Usage**) **j)** Apply the mathematical basis, principles of algorithms and the theory of Computer Science in the modeling and design of computational systems in such a way as to demonstrate understanding of the equilibrium points involved in the chosen option. (**Usage**)

7. Content

UNIT 1: Parallelism Fundamentals (18)	
Competences: a	
Content	**Generales Goals**

• Multiple simultaneous computations
• Goals of parallelism (e.g., throughput) versus concurrency (e.g., controlling access to shared resources)
• Parallelism, communication, and coordination
  – Parallelism, communication, and coordination
  – Need for synchronization
• Programming errors not found in sequential programming
  – Data races (simultaneous read/write or write/write of shared state)
  – Higher-level races (interleavings violating program intention, undesired non-determinism)
  – Lack of liveness/progress (deadlock, starvation)
	• Distinguish using computational resources for a faster answer from managing efficient access to a shared resource [Familiarity]
	• Distinguish multiple sufficient programming constructs for synchronization that may be inter-implementable but have complementary advantages [Familiarity]
	• Distinguish data races from higher level races [Familiarity]

Readings: Pacheco (2011), Matloff (2014), **quinnz**, Georg Hager (2010)	

<table>
<tr><td colspan="2">UNIT 2: Parallel Architecture (12)</td></tr>
<tr><td colspan="2">Competences: a,b</td></tr>
<tr><td>Content</td><td>Generales Goals</td></tr>
<tr><td>

Multicore processors
Shared vs distributed memory
Symmetric multiprocessing (SMP)
SIMD, vector processing
GPU, co-processing
Flynn's taxonomy
Instruction level support for parallel programming
Atomic instructions such as Compare and Set
Memory issues
Multiprocessor caches and cache coherence
Non-uniform memory access (NUMA)
Topologies
Interconnects
Clusters
Resource sharing (e.g., buses and interconnects)

</td><td>

Explain the differences between shared and distributed memory [Assessment]
Describe the SMP architecture and note its key features [Assessment]
Characterize the kinds of tasks that are a natural match for SIMD machines [Usage]
Describe the advantages and limitations of GPUs vs CPUs [Usage]
Explain the features of each classification in Flynn's taxonomy [Usage]
Describe the challenges in maintaining cache coherence [Familiarity]
Describe the key performance challenges in different memory and distributed system topologies [Familiarity]

</td></tr>
<tr><td colspan="2">Readings: Pacheco (2011), Kirk and Hwu (2013), Sanders and Kandrot (2010), Georg Hager (2010)</td></tr>
</table>

<table>
<tr><td colspan="2">UNIT 3: Parallel Decomposition (18)</td></tr>
<tr><td colspan="2">Competences: a,b</td></tr>
<tr><td>Content</td><td>Generales Goals</td></tr>
<tr><td>

Need for communication and coordination/synchronization
Independence and partitioning
Basic knowledge of parallel decomposition concept
Task-based decomposition
Implementation strategies such as threads
Data-parallel decomposition
Strategies such as SIMD and MapReduce
Actors and reactive processes (e.g., request handlers)

</td><td>

Explain why synchronization is necessary in a specific parallel program [Usage]
Identify opportunities to partition a serial program into independent parallel modules [Familiarity]
Write a correct and scalable parallel algorithm [Usage]
Parallelize an algorithm by applying task-based decomposition [Usage]
Parallelize an algorithm by applying data-parallel decomposition [Usage]
Write a program using actors and/or reactive processes [Usage]

</td></tr>
<tr><td colspan="2">Readings: Pacheco (2011), Matloff (2014), Quinn (2003), Georg Hager (2010)</td></tr>
</table>

<table>
<tr><td colspan="2">UNIT 4: Communication and Coordination (18)</td></tr>
<tr><td colspan="2">Competences: a,b</td></tr>
<tr><td>Content</td><td>Generales Goals</td></tr>
<tr><td>

Shared Memory
Consistency, and its role in programming language guarantees for data-race-free programs
Message passing

Point-to-point versus multicast (or event-based) messages
Blocking versus non-blocking styles for sending and receiving messages
Message buffering (cross-reference PF/Fundamental Data Structures/Queues)

Atomicity

Specifying and testing atomicity and safety requirements
Granularity of atomic accesses and updates, and the use of constructs such as critical sections or transactions to describe them
Mutual Exclusion using locks, semaphores, monitors, or related constructs
Potential for liveness failures and deadlock (causes, conditions, prevention)
Composition

Composing larger granularity atomic actions using synchronization
Transactions, including optimistic and conservative approaches

Consensus
(Cyclic) barriers, counters, or related constructs
Conditional actions
Conditional waiting (e.g., using condition variables)

</td><td>

Use mutual exclusion to avoid a given race condition [Usage]
Give an example of an ordering of accesses among concurrent activities (eg, program with a data race) that is not sequentially consistent [Familiarity]
Give an example of a scenario in which blocking message sends can deadlock [Usage]
Explain when and why multicast or event-based messaging can be preferable to alternatives [Familiarity]
Write a program that correctly terminates when all of a set of concurrent tasks have completed [Usage]
Give an example of a scenario in which an attempted optimistic update may never complete [Familiarity]
Use semaphores or condition variables to block threads until a necessary precondition holds [Usage]

</td></tr>
<tr><td colspan="2">Readings: Pacheco (2011), Matloff (2014), Quinn (2003), Georg Hager (2010)</td></tr>
</table>

UNIT 5: Parallel Algorithms, Analysis, and Programming (18)	
Competences: a,b	
Content	**Generales Goals**
• Critical paths, work and span, and the relation to Amdahl's law • Speed-up and scalability • Naturally (embarrassingly) parallel algorithms • Parallel algorithmic patterns (divide-and-conquer, map and reduce, master-workers, others) – Specific algorithms (e.g., parallel MergeSort) • Parallel graph algorithms (e.g., parallel shortest path, parallel spanning tree) (cross-reference AL/Algorithmic Strategies/Divide-and-conquer) • Parallel matrix computations • Producer-consumer and pipelined algorithms • Examples of non-scalable parallel algorithms	• Define "critical path", "work", and "span" [Familiarity] • Compute the work and span, and determine the critical path with respect to a parallel execution diagram [Usage] • Define "speed-up" and explain the notion of an algorithm's scalability in this regard [Familiarity] • Identify independent tasks in a program that may be parallelized [Usage] • Characterize features of a workload that allow or prevent it from being naturally parallelized [Familiarity] • Implement a parallel divide-and-conquer (and/or graph algorithm) and empirically measure its performance relative to its sequential analog [Usage] • Decompose a problem (eg, counting the number of occurrences of some word in a document) via map and reduce operations [Usage] • Provide an example of a problem that fits the producer-consumer paradigm [Usage] • Give examples of problems where pipelining would be an effective means of parallelization [Usage] • Implement a parallel matrix algorithm [Usage] • Identify issues that arise in producer-consumer algorithms and mechanisms that may be used for addressing them [Usage]
Readings: Matloff (2014), Quinn (2003), Georg Hager (2010)	

UNIT 6: Parallel Performance (18)	
Competences: a,b	
Content	**Generales Goals**

• Load balancing
• Performance measurement
• Scheduling and contention (cross-reference OS/Scheduling and Dispatch)
• Evaluating communication overhead
• Data management
  – Non-uniform communication costs due to proximity (cross-reference SF/Proximity)
  – Cache effects (e.g., false sharing)
  – Maintaining spatial locality
• Power usage and management
	• Detect and correct a load imbalance [Usage]
	• Calculate the implications of Amdahl's law for a particular parallel algorithm (cross-reference SF/Evaluation for Amdahl's Law) [Usage]
	• Describe how data distribution/layout can affect an algorithm's communication costs [Familiarity]
	• Detect and correct an instance of false sharing [Usage]
	• Explain the impact of scheduling on parallel performance [Familiarity]
	• Explain performance impacts of data locality [Familiarity]
	• Explain the impact and trade-off related to power usage on parallel performance [Familiarity]

Readings: Pacheco (2011), Matloff (2014), Kirk and Hwu (2013), Sanders and Kandrot (2010), Georg Hager (2010)	

8. Methodology
El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos. El profesor del curso presentará demostraciones para fundamentar clases teóricas. El profesor y los alumnos realizarán prácticas Los alumnos deberán asistir a clase habiendo leído lo que el profesor va a presentar. De esta manera se facilitará la comprensión y los estudiantes estarán en mejores condiciones de hacer consultas en clase.

9. Assessment
Continuous Assessment 1 : 20 % **Partial Exam** : 30 % **Continuous Assessment 2** : 20 % **Final exam** : 30 %

References

Georg Hager, Gerhard Wellein (2010). *Introduction to High Performance Computing for Scientists and Engineers (Chapman & HallCRC Computational Science)*. Ed. by CRC Press. 1st. ISBN: 978-1439811924.

Kirk, David B. and Wen-mei W. Hwu (2013). *Programming Massively Parallel Processors: A Hands-on Approach*. 2nd. Morgan Kaufmann. ISBN: 978-0-12-415992-1.

Matloff, Norm (2014). *Programming on Parallel Machines*. University of California, Davis.

Pacheco, Peter S. (2011). *An Introduction to Parallel Programming*. 1st. Morgan Kaufmann. ISBN: 978-0-12-374260-5.

Quinn, Michael J. (2003). *Parallel Programming in C with MPI and OpenMP*. 1st. McGraw-Hill Education Group. ISBN: 0071232656.

Sanders, Jason and Edward Kandrot (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming*. 1st. Addison-Wesley Professional. ISBN: 0131387685, 9780131387683.