Universidad Católica San Pablo (UCSP)
Escuela Profesional de
Ciencia de la Computación
SILABO

CS3P1. Computación Paralela y Distribuída
(Obligatorio)

2020-I

1. Información general		
1.1 Escuela	:	Ciencia de la Computación
1.2 Curso	:	CS3P1. Computación Paralela y Distribuída
1.3 Semestre	:	8^{vo} Semestre.
1.4 Prerrequisitos	:	• CS212. Análisis y Diseño de Algoritmos. (5^{to} Sem) • CS231. Redes y Comunicación. (7^{mo} Sem)
1.5 Condición	:	Obligatorio
1.6 Modalidad de aprendizaje	:	Virtual
1.7 horas	:	2 HT; 2 HP; 2 HL;
1.8 Créditos	:	4

2. Profesores

3. Fundamentación del curso

La última década ha traído un crecimiento explosivo en computación con multiprocesadores, incluyendo los procesadores de varios núcleos y centros de datos distribuidos. Como resultado, la computación paralela y distribuida se ha convertido de ser un tema ampliamente electivo para ser uno de los principales componentes en la malla estudios en ciencia de la computación de pregrado. Tanto la computación paralela como la distribuida implica la ejecución simultánea de múltiples procesos, cuyas operaciones tienen el potencial para intercalar de manera compleja. La computación paralela y distribuida construye sobre cimientos en muchas áreas, incluyendo la comprensión de los conceptos fundamentales de los sistemas, tales como: concurrencia y ejecución en paralelo, consistencia en el estado/manipulación de la memoria, y latencia. La comunicación y la coordinación entre los procesos tiene sus cimientos en el paso de mensajes y modelos de memoria compartida de la computación y conceptos algorítmicos como atomicidad, el consenso y espera condicional. El logro de aceleración en la práctica requiere una comprensión de algoritmos paralelos, estrategias para la descomposición problema, arquitectura de sistemas, estrategias de implementación y análisis de rendimiento. Los sistemas distribuidos destacan los problemas de la seguridad y tolerancia a fallos, hacen hincapié en el mantenimiento del estado replicado e introducen problemas adicionales en el campo de las redes de computadoras.

4. Resumen

1. Fundamentos de paralelismo 2. Arquitecturas paralelas 3. Descomposición en paralelo 4. Comunicación y coordinación 5. Análisis y programación de algoritmos paralelos 6. Desempeño en paralelo

5. Objetivos Generales

- Que el alumno sea capaz de crear aplicaciones paralelas de mediana complejidad aprovechando eficientemente máquinas con múltiples núcleos.
- Que el alumno sea capaz de comparar aplicaciones secuenciales y paralelas.
- Que el alumno sea capaz de convertir, cuando la situación lo amerite, aplicaciones secuenciales a paralelas de forma eficiente.

6. Contribución a los resultados (*Outcomes*)

Esta disciplina contribuye al logro de los siguientes resultados de la carrera:

1) Analizar un problema computacional complejo y aplicar los principios computacionales y otras disciplinas relevantes para identificar soluciones. (**Usar**)

6) Aplicar fundamentos de teoria de ciencias de la computación y desarrollo de software para producir soluciones basados en computación. (**Usar**)

7. Contenido

UNIDAD 1: Fundamentos de paralelismo (18)

Competencias:

Contenido	**Objetivos Generales**
• Procesamiento Simultáneo Múltiple. • Metas del Paralelismo (ej. rendimineto) frente a Concurrencia (ej. control de acceso a recursos compartidos) • Paralelismo, comunicación, y coordinación: – Paralelismo, comunicación, y coordinación – Necedidad de Sincronización • Errores de Programación ausentes en programación secuencial: – Tipos de Datos (lectura/escritura simultánea o escritura/escritura compartida) – Tipos de Nivél más alto (interleavings violating program intention, no determinismo no deseado) – Falta de vida/progreso (deadlock, starvation)	• Distinguir el uso de recursos computacionales para una respuesta mas rápida para administrar el acceso eficiente a un recurso compartido [Familiarizarse] • Distinguir múltiples estructuras de programación suficientes para la sincronización que pueden ser interimplementables pero tienen ventajas complementarias [Familiarizarse] • Distinguir datos de carrera (*data races*) a partir de carreras de mas alto nivel [Familiarizarse]

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

UNIDAD 2: Arquitecturas paralelas (12)

Competencias:

Contenido	Objetivos Generales

• Procesadores mutlinúcleo.
• Memoria compartida vs memoria distribuida.
• Multiprocesamiento simétrico.
• SIMD, procesamiento de vectores.
• GPU, coprocesamiento.
• Taxonomia de Flynn.
• Soporte a nivel de instrucciones para programación paralela.
  – Instrucciones atómicas como Compare/Set (Comparar / Establecer)
• Problemas de Memoria:
  – Caches multiprocesador y coherencia de cache
  – Acceso a Memoria no uniforme (NUMA)
• Topologías.
  – Interconecciones
  – Clusters
  – Compartir recursos (p.e., buses e interconexiones)
	• Explicar las diferencias entre memoria distribuida y memoria compartida [Evaluar]
	• Describir la arquitectura SMP y observar sus principales caracteristicas [Evaluar]
	• Distinguir los tipos de tareas que son adecuadas para máquinas SIMD [Usar]
	• Describir las ventajas y limitaciones de GPUs vs CPUs [Usar]
	• Explicar las caracteristicas de cada clasificación en la taxonomía de Flynn [Usar]
	• Describir los desafíos para mantener la coherencia de la caché [Familiarizarse]
	• Describir los desafíos clave del desempeño en diferentes memorias y topologías de sistemas distribuidos [Familiarizarse]

Lecturas: Pacheco (2011), Kirk and Hwu (2013), Sanders and Kandrot (2010), Georg Hager (2010)

UNIDAD 3: Descomposición en paralelo (18)

Competencias:

Contenido	Objetivos Generales

• Necesidad de Comunicación y coordinación/sincronización.
• Independencia y Particionamiento.
• Conocimiento Básico del Concepto de Descomposición Paralela.
• Decomposición basada en tareas:
  – Implementación de estrategias como hebras
• Descomposición de Información Paralela
  – Estrategias como SIMD y MapReduce
• Actores y Procesos Reactivos (solicitud de gestores)
	• Explicar por qué la sincronización es necesaria en un programa paralelo especifico [Usar]
	• Identificar oportunidades para particionar un programa serial en módulos paralelos independientes [Familiarizarse]
	• Escribir un algoritmo paralelo correcto y escalable [Usar]
	• Paralelizar un algoritmo mediante la aplicación de descomposición basada en tareas [Usar]
	• Paralelizar un algoritmo mediante la aplicación de descomposición datos en paralelo [Usar]
	• Escribir un programa usando actores y/o procesos reactivos [Usar]

Lecturas: Pacheco (2011), Matloff (2014), Quinn (2003), Georg Hager (2010)

UNIDAD 4: Comunicación y coordinación (18)

Competencias:

Contenido	Objetivos Generales
• Memoria Compartida. • La consistencia, y su papel en los lenguaje de programación garantias para los programas de carrera libre. • Pasos de Mensaje: – Mensajes Punto a Punto versus multicast (o basados en eventos) – Estilos para enviar y recibir mensajes Blocking vs non-blocking – Buffering de mensajes • Atomicidad: – Especificar y probar atomicidad y requerimientos de seguridad – Granularidad de accesos atómicos y actualizaciones, y uso de estructuras como secciones críticas o transacciones para describirlas – Exclusión mutua usando bloques, semáforos, monitores o estructuras relacionadas ∗ Potencial para fallas y bloqueos (*deadlock*) (causas, condiciones, prevención) – Composición ∗ Componiendo acciones atómicas granulares más grandes usando sincronización ∗ Transacciones, incluyendo enfoques optimistas y conservadores • Consensos: – (Ciclicos) barerras, contadores y estructuras relacionadas • Acciones condicionales: – Espera condicional (p.e., empleando variables de condición)	• Usar exclusión mútua para evitar una condición de carrera [Usar] • Dar un ejemplo de una ordenación de accesos entre actividades concurrentes (por ejemplo, un programa con condición de carrera) que no son secuencialmente consistentes [Familiarizarse] • Dar un ejemplo de un escenario en el que el bloqueo de mensajes enviados pueden dar *deadlock* [Usar] • Explicar cuándo y por qué mensajes de multidifusión (*multicast*) o basado en eventos puede ser preferible a otras alternativas [Familiarizarse] • Escribir un programa que termine correctamente cuando todo el conjunto de procesos concurrentes hayan sido completados [Usar] • Dar un ejemplo de un escenario en el que un intento optimista de actualización puede nunca completarse [Familiarizarse] • Usar semaforos o variables de condición para bloquear hebras hasta una necesaria precondición de mantenga [Usar]

Lecturas: Pacheco (2011), Matloff (2014), Quinn (2003), Georg Hager (2010)

UNIDAD 5: Análisis y programación de algoritmos paralelos (18)

Competencias:

Contenido	**Objetivos Generales**
• Caminos críticos, el trabajo y la duración y la relación con la ley de Amdahl. • Aceleración y escalabilidad. • Naturalmente (vergonzosamente) algoritmos paralelos. • Patrones Algoritmicos paralelos (divide-y-conquista, map/reduce, amos-trabajadores, otros) – Algortimos específicos (p.e., MergeSort paralelo) • Algoritmos de grafos paralelo (por ejemplo, la ruta más corta en paralelo, árbol de expansión paralela) • Cálculos de matriz paralelas. • Productor-consumidor y algoritmos paralelos segmentados. • Ejemplos de algoritmos paralelos no-escalables.	• Definir: camino crítico, trabajo y *span* [Familiarizarse] • Calcular el trabajo y el *span* y determinar el camino crítico con respecto a un diagrama de ejecución paralela. [Usar] • Definir *speed-up* y explicar la noción de escalabilidad de un algoritmo en este sentido [Familiarizarse] • Identificar tareas independientes en un programa que debe ser paralelizado [Usar] • Representar características de una carga de trabajo que permita o evite que sea naturalmente paralelizable [Familiarizarse] • Implementar un algoritmo dividir y conquistar paralelo (y/o algoritmo de un grafo) y medir empiricamente su desempeño relativo a su analogo secuencial [Usar] • Descomponer un problema (por ejemplo, contar el número de ocurrencias de una palabra en un documento) via operaciones *map* y *reduce* [Usar] • Proporcionar un ejemplo de un problema que se corresponda con el paradigma productor-consumidor [Usar] • Dar ejemplos de problemas donde el uso de *pipelining* sería un medio eficaz para la paralelización [Usar] • Implementar un algoritmo de matriz paralela [Usar] • Identificar los problemas que surgen en los algoritmos del tipo productor-consumidor y los mecanismos que pueden utilizarse para superar dichos problemas [Usar]

Lecturas: Matloff (2014), Quinn (2003), Georg Hager (2010)

UNIDAD 6: Desempeño en paralelo (18)	
Competencias:	
Contenido	**Objetivos Generales**
• Equilibrio de carga. • La medición del desempeño. • Programación y contención. • Evaluación de la comunicación de arriba. • Gestión de datos: – Costos de comunicación no uniforme debidos a proximidad – Efectos de Cache (p.e., false sharing) – Manteniendo localidad espacial • Consumo de energía y gestión.	• Detectar y corregir un desbalanceo de carga [Usar] • Calcular las implicaciones de la ley de Amdahl para un algoritmo paralelo particular [Usar] • Describir como la distribuición/disposición de datos puede afectar a los costos de comunicación de un algoritmo [Familiarizarse] • Detectar y corregir una instancia de uso compartido falso (*false sharing*) [Usar] • Explicar el impacto de la planificación en el desempeño paralelo [Familiarizarse] • Explicar el impacto en el desempeño de la localidad de datos [Familiarizarse] • Explicar el impacto y los puntos de equilibrio relacionados al uso de energía en el desempeño paralelo [Familiarizarse]
Lecturas: Pacheco (2011), Matloff (2014), Kirk and Hwu (2013), Sanders and Kandrot (2010), Georg Hager (2010)	

8. Metodología

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. Evaluar

Evaluación Continua 1 : 20 %

Examen parcial : 30 %

Evaluación Continua 2 : 20 %

Examen final : 30 %

References

Georg Hager, Gerhard Wellein (2010). *Introduction to High Performance Computing for Scientists and Engineers (Chapman & Hall CRC 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.