Guía docente
250COD006 - 250COD006 - Programación Científica y Computación de Alto Rendimiento

Última modificación: 16/06/2026

Unidad responsable: Escuela Técnica Superior de Ingeniería de Caminos, Canales y Puertos de Barcelona
Unidad que imparte: 751 - DECA - Departamento de Ingeniería Civil y Ambiental.

Titulación: MÁSTER UNIVERSITARIO EN INGENIERÍA COMPUTACIONAL Y ASISTIDA POR DATOS (Plan 2026). (Asignatura obligatoria).

Curso: 2026 **Créditos ECTS:** 5.0 **Idiomas:** Inglés

PROFESORADO

Profesorado responsable: Zorrilla Martínez, Rubén

Otros: Zorrilla Martínez, Rubén
Rossi Bernecoli, Riccardo

METODOLOGÍAS DOCENTES

El curso adopta un enfoque interactivo y práctico en el que los conceptos teóricos se integran estrechamente con la implementación. Por lo tanto, cada sesión combina:
- Explicaciones conceptuales sobre programación científica y principios de la computación de alto rendimiento.
- Demostraciones de programación en directo, en las que se desarrollan las ideas clave paso a paso, mostrando cómo los conceptos teóricos se traducen en implementaciones computacionales eficientes.
- Ejercicios prácticos, que los alumnos completan durante la sesión, lo que permite la aplicación inmediata de los conceptos utilizando Python.

Una parte importante del curso se imparte utilizando cuadernos interactivos (por ejemplo, Google Colab), que permiten la experimentación, la reproducibilidad y la iteración rápida sin una configuración compleja. Este enfoque facilita el aprendizaje activo y permite a los alumnos explorar variaciones de los métodos en tiempo real.

A lo largo del curso, los alumnos trabajan en tareas prácticas y en un proyecto progresivo. Estas actividades hacen hincapié en el análisis del rendimiento, la escalabilidad y la integración de componentes numéricos y basados en datos.

El trabajo independiente complementa las actividades en el aula, centrándose en completar las tareas, mejorar las implementaciones y analizar el rendimiento computacional, lo que fomenta la autonomía y una comprensión más profunda.

OBJETIVOS DE APRENDIZAJE DE LA ASIGNATURA

Este curso presenta los principios y prácticas necesarios para desarrollar software científico eficiente, escalable y fácil de mantener para aplicaciones de ingeniería computacional y asistida por datos. Se centra en la implementación de métodos numéricos mediante paradigmas de programación modernos y técnicas de computación de alto rendimiento dentro de un ecosistema basado en Python.

El curso abarca todo el flujo de trabajo de la ingeniería computacional, desde el diseño de software y la robustez numérica hasta la optimización del rendimiento y la ejecución paralela. Los temas principales incluyen álgebra lineal densa y dispersa de alto rendimiento y conceptos de ingeniería del rendimiento, como la jerarquía de memoria, la vectorización y la creación de perfiles. Estos se amplían a paradigmas de computación paralela, incluyendo el paralelismo de memoria compartida, la programación de memoria distribuida con MPI y la aceleración por GPU.

Además, el curso aborda los retos actuales de la ingeniería asistida por datos, como el manejo de conjuntos de datos a gran escala mediante enfoques fuera de núcleo y distribuidos, y la integración de flujos de trabajo de simulación con canalizaciones de procesamiento de datos.

A través de una combinación de sesiones teóricas y tareas prácticas, los estudiantes desarrollan la capacidad de diseñar, implementar, analizar y optimizar soluciones computacionales para problemas de ingeniería a gran escala.

HORAS TOTALES DE DEDICACIÓN DEL ESTUDIANTADO

Tipo	Horas	Porcentaje
Horas aprendizaje autónomo	80,0	64.00
Horas grupo grande	45,0	36.00

Dedicación total: 125 h

CONTENIDOS

Paradigmas de programación e introducción a Python

Descripción:
- Programación procedural vs orientada a objetos (OOP)
- Conceptos de programación funcional
- OOP en códigos de computación a gran escala (encapsulación, abstracción)
- Patrones funcionales en cálculo numérico (mapa, reducción, inmutabilidad)
- Lenguajes compilados vs interpretados
- Tipos estáticos vs dinámmicos
- Introducción a la sintaxis y entornos virtuales de Python

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

Complejidad algorítmica & estructuras de datos

Descripción:
- Complejidad algorítmica & estructuras de datos
- Diseño modular con Python (funciones, clases, tipos)
- Visualización con Python (matplotlib y PyVista)

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

Desarrollo de código profesional

Descripción:
- Control de versiones
- Desarrollo profesional: calidad del código y revisión del código
- Estrategias de test (integración contínua) y desarrollo basado en tests
- Entornos de programación asistidos por IA (IDEs)

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

Precisión numérica & estabilidad en simulación a gran escala

Descripción:
- Aritmética en coma flotante
- Condicionamiento de los problemas vs estabilidad algorítmica
- Complejidad algorítmica en flujos de trabajo de simulación
- Jerarquía de memoria y comportamiento de la memoria caché
- Modelo Roofline
- Vectorización y disposición de los datos en memoria
- Identificación de cuellos de botella en bucles tipo elementos finitos

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

Álgebra lineal de alto rendimiento

Descripción:
- Conceptos de BLAS/LAPACK subyacentes a NumPy
- Operaciones densas eficientes y evaluación del rendimiento (benchmarking)
- Formatos de matrices dispersas (CSR, COO)
- Ensamblaje eficiente de matrices dispersas
- Análisis de la huella de memoria
- Breve introducción a los métodos de Krylov y a su implementación
- Estrategias de precondicionamiento
- Solucionadores iterativos: convergencia frente a escalabilidad
- Solucionadores iterativos: robustez numérica

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

Introducción al paralelismo

Descripción:
- Ley de Amdahl
- Escalabilidad fuerte frente a escalabilidad débil
- Paralelismo en CPU y disposición de los datos en memoria: paralelismo en memoria compartida vs paralelismo en memoria distribuida
- Paralelismo en GPU

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

Paralelismo en memoria compartida

Descripción:
- Paralelismo en memoria compartida en Python (GIL y multiprocessing, fundamentales de asyncio, task orchestration)
- Trabajo de SMP

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

Paralelismo en memoria distribuida con MPI

Descripción:
- Estructuras de datos distribuidas
- Patrones de comunicación
- Producto matrix-vector distribuido

Dedicación: 15h
Grupo grande/Teoría: 6h
Aprendizaje autónomo: 9h

Paralelismo en GPU

Descripción:
- Introducción a la aceleración en GPU con PyTorch
- Trabajo de GPU

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

Resumen de paralelismo

Descripción:
- Sesión práctica para trabajar en los proyectos de DMP, MPI y GPU.

Dedicación: 7h 30m
Actividades dirigidas: 3h
Aprendizaje autónomo: 4h 30m

Depuración eficiente y análisis de performancia

Descripción:
- Experiencia práctica con un ciclo de vida de desarrollo "corrupto"
- Herramientas de depuración (debugging)
- Cuellos de botella en operaciones de reducción
- Experiencia práctica con herramientas de perfilado (profiling)

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

Gestión de conjuntos de datos extremadamente grandes

Descripción:
- Arreglos fuera de memoria (out-of-core arrays)
- Estrategias de particionamiento en bloques (chunking)
- Grafos de tareas y evaluación diferida (lazy evaluation)
- Planificador distribuido de Dask
- Paradigmas MPI vs Dask
- Entrada/salida científica eficiente (HDF5)

Dedicación: 7h 30m
Grupo grande/Teoría: 3h
Aprendizaje autónomo: 4h 30m

SISTEMA DE CALIFICACIÓN

La evaluación del curso se basa en la evaluación continua y en un proyecto final:
- Evaluación continua (40 %): evaluación mediante trabajos prácticos y tareas de aplicación. Los criterios de evaluación incluyen la corrección, la calidad del código y aspectos básicos relacionados con el rendimiento.
- Proyecto final (45 %): desarrollo de un proyecto informático que integre los conceptos del curso, tales como métodos numéricos, computación paralela y optimización del rendimiento. Los entregables del proyecto incluyen el código fuente (con control de versiones), así como un informe técnico que incluya un análisis del rendimiento y una demostración de los resultados. Los criterios de evaluación incluyen la corrección técnica, la escalabilidad, el análisis del rendimiento y la calidad del diseño del software.
- Defensa oral (15 %): evaluación oral individual basada en el proyecto final, en la que se evalúa la comprensión de los métodos implementados, las decisiones de diseño y paralelización, y la capacidad para analizar el rendimiento.

BIBLIOGRAFÍA

Básica:
- Dongarra, Jack ... [et al.]. Sourcebook of Parallel Computing. Amsterdam [etc.]: Morgan Kaufmann, 2003. ISBN 1558608710.
- Eijkhout, Victor. Introduction to High-Performance Scientific Computing [en línea]. 3rd ed. 2022 [Consulta: 25/06/2026]. Disponible a: https://github.com/VictorEijkhout/TheArtofHPC_pdfs/blob/main/vol1/EijkhoutIntroToHPC.pdf.
- Scott, L.R.; Clark, T.; Bagheri, B. Scientific Parallel Computing. Princeton, New Jersey ; Oxford: Princeton University Press, 2005. ISBN 069111935X.
- Dongarra, Jack ... [et al.]. Numerical Linear Algebra on High-Performance Computers. Philadelphia: SIAM, 1998. ISBN 0898714281.