Programación de Altas Prestaciones

Curso Académico: (2019 / 2020) **Fecha de revisión: 29-04-2019**

Departamento asignado a la asignatura: Departamento de Informática

Coordinador/a: GARCIA SANCHEZ, JOSE DANIEL

Tipo: Obligatoria Créditos ECTS : 6.0

Curso : 1 Cuatrimestre : 1

OBJETIVOS

Competencias:

* Que los estudiantes sepan aplicar los conocimientos adquiridos y su capacidad de resolución de problemas en entornos nuevos o poco conocidos dentro de contextos más amplios (o multidisciplinares) relacionados con su área de estudio
* Que los estudiantes posean las habilidades de aprendizaje que les permitan continuar estudiando de un modo que habrá de ser en gran medida autodirigido o autónomo.
* Capacidad para comprender y aplicar métodos y técnicas del ámbito de la Ingeniería Informática en los mercados financieros

Resultados del aprendizaje

* Conocer los principales lenguajes de programación que se utilizan para el desarrollo de software financiero.
* Capacidad para implementar software para el sector financiero.
* Conocimientos sobre la programación de altas prestaciones.

DESCRIPCIÓN DE CONTENIDOS: PROGRAMA

1. Fundamentos de la computación de altas prestaciones.
1.1. El problema del rendimiento.
1.2. Los límites del rendimiento: Leyes del rendimiento.
1.3. Altas prestaciones en aplicaciones secuenciales.
1.4. Paralelismo y computación de altas prestaciones. Tipos de paralelismo.

2. Lenguajes de programación
2.1. Perspectiva de lenguajes de programación y rendimiento.
2.2. Introducción a C++.
2.3. Herencia de interfaz y herencia de implementación.
2.4. Polimorfismo dinámico.

3. Gestión de memoria
3.1. Almacenamiento automático y estático.
3.2. El almacén libre.
3.3. Paso de parámetros y devolución de resultados por valor y por referencia.
3.4. Semántica de movimiento.
3.5. Smart pointers.

4. Programación genérica.
4.1. Plantillas y programación genérica. Polimorfismo estático.
4.2. Conceptos y programación genérica.
4.3. Objetos función y expresiones lambda.
4.4. Templates variádicos.
4.5. Arquitectura de la STL.
4.6. STL: Contenedores e iteradores.
4.7. STL: Algoritmos.

5. Bibliotecas e interoperabilidad
5.1. Tipos de bibliotecas: estáticas y dinámicas.
5.2. Bibliotecas header-only.
5.3. Includes y tiempo de compilación. Declaraciones adelantadas. Optimización de includes.
5.4. Organización de bibliotecas y espacios de nombre.
5.5. Motores de construcción.

5.6. Interoperabilidad entre lenguajes.
5.7. Visión general de bibliotecas de uso frecuente.

6. Optimización de código.
6.1. El optimizador de código: efectos y límites.
6.2. Vectorización de código: Limitaciones de la vectorización automática.
6.3. Soporte a la vectorización manual de código.
6.4. Expansión en línea (inline) de funciones: mitos y realidades.
6.5. Eliminación de objetos temporales.
6.6. Evaluación de expresiones en tiempo de compilación.
6.7. Desenrrollado de la pila: el especificador noexcept.

7. Análisis de rendimiento de aplicaciones
7.1. Mecanismos de análisis de rendimiento.
7.2. Análisis intrusivo en el código.
7.3. Perfilado de aplicaciones.
7.4. Análisis de contadores hardware y de sistema.
7.5. Impacto de la caché del procesador: análisis.
7.6. Análisis de rendimiento en código multi-hilo.

8. Hilos y frameworks de concurrencia y paralelismo
8.1. Programación concurrente en C++.
8.2. Hilos, mutex y variables condición.
8.3. Futuros y tareas empaquetadas.
8.4. Alternativas de programación paralela de memoria compartida.
8.5. Programación paralela estándar: soluciones de lenguaje y biblioteca
8.6. Frameworks de paralelismo.

9. Programación de baja latencia.
9.1. Aplicaciones de baja latencia.
9.2. Arquitectura de aplicaciones baja latencia.
9.3. Construcciones incompatibles con el software de baja latencia.
9.4. Impacto de la memoria dinámica. Pre-asignación.
9.5. Impacto de las excepciones. Alternativas de diseño.
9.6. Estructuras de datos libres de cerrojos.

10. Programación en Cluster
10.1. Introducción a clústers.
10.2. Gestión de trabajos. Sistemas de colas.
10.3. Modelo de programación de paso de mensajes.
10.4. Introducción a MPI.

ACTIVIDADES FORMATIVAS, METODOLOGÍA A UTILIZAR Y RÉGIMEN DE TUTORÍAS

Actividades:
* Clase teóricas: Presentaciones teóricas acompañadas de material electrónico, como presentaciones digitales.
* Clases teórico prácticas: Combinación de clases teóricas acompañadas de la resolución de ejercicios prácticos.
* Prácticas de laboratorio: Prácticas a desarrollar en laboratorios específicos para las distintas asignaturas.
* Tutorías: Tutorías de carácter presencial.
* Actividades de e-learning: Foros de las asignaturas, foros de discusión y otras actividades formativas de e-learning.
* Trabajo individual del estudiante: Actividades individuales del alumno que complementan al resto de actividades (tanto presenciales como no presenciales), así como la preparación de exámenes.

Metodologias docentes
* -Exposiciones en clase del profesor con soporte de medios informáticos y audiovisuales, en las que se desarrollan los conceptos principales de la materia y se proporciona la bibliografía para complementar el aprendizaje de los alumnos.
* -Resolución de casos prácticos, problemas, etc. planteados por el profesor de manera individual o en grupo
* -Exposición y discusión en clase, bajo la moderación del profesor de temas relacionados con el contenido de la materia, así como de casos prácticos
* -Elaboración de trabajos e informes de manera individual o en grupo
* -Actividades específicas de e-learning, relacionadas con el carácter semi-presencial del título,

incluyendo la visualización de contenidos grabados, actividades de auto-corrección, participación en foros, y cualquier otro mecanismo de enseñanza on-line

En la componente práctica de esta materia se desarrollarán trabajos de mejora del rendimiento de aplicaciones que podrán incluir, entre otros, el análisis de rendimiento de aplicaciones, la identificación de problemas de rendimiento, la optimización de código, y la aplicación de técnicas de paralelización.

Para estos contenidos más prácticos, se puede combinar la asistencia presencial a los laboratorios con el trabajo individual o en grupo fuera del aula a través de Aula Remota, junto con el seguimiento y tutorado de los alumnos a través de los foros y otros mecanismos de discusión. Se trabajará también con otras estrategias de e-learning, como la auto-evaluación de los trabajos realizados, todo soportado a través de Aula Global.

SISTEMA DE EVALUACIÓN

Trabajos individuales o en grupo realizados durante el curso, tanto en actividades presenciales como de e-learning: 70 %
Examen final: 30 %

Peso porcentual del Examen Final:	30
Peso porcentual del resto de la evaluación:	70

BIBLIOGRAFÍA BÁSICA

- Anthony Williams C++ Concurrency in Action. Practical Multithreading, Manning, 2012
- Bjarne Stroustrup Programming ¿ Principles and Practice. 2nd Edition, Addison-Wesley, 2014
- Carlos Oliveria Practical C++ Financial Programming, Apress, 2015
- James Reinders Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism, O'Reillly, 2007

BIBLIOGRAFÍA COMPLEMENTARIA

- Bjarne Stroustrup The C++ Programming Language. 4th Edition, Addison-Wesley, 2013
- Bjarne Stroustrup A Tour of C++, Addison-Wesley, 2013
- Gerassimos Barlas Multicore and GPU Programming: An Integrated Approach, Morgan-Kaufmann, 2014
- Kurt Guntheroth Optimized C++, O'Reilly, 2016
- Peter Gottschling Discovering Modern C++: An intensive course for scientists, engineers and programmers, Addison-Wesley, 2015