INGEGNERIA INFORMATICA (LM75)

(Lecce - Università degli Studi)

Insegnamento ALGORITMI PARALLELI

GenCod A006812

Docente titolare Massimo CAFARO

Docenti responsabili dell'erogazione
Massimo CAFARO, MARCO PULIMENO

Insegnamento ALGORITMI PARALLELI

Anno di corso 2

Insegnamento in inglese PARALLEL ALGORITHMS

Lingua ITALIANO

Settore disciplinare ING-INF/05

Percorso PERCORSO COMUNE

Corso di studi di riferimento
INGEGNERIA INFORMATICA

Tipo corso di studi Laurea Magistrale

Sede Lecce

Crediti 9.0

Periodo Secondo Semestre

Ripartizione oraria Ore Attività frontale: 81.0

Tipo esame Orale

Per immatricolati nel 2022/2023

Valutazione Voto Finale

Erogato nel 2023/2024

Orario dell'insegnamento
https://easyroom.unisalento.it/Orario

BREVE DESCRIZIONE DEL CORSO

Il corso fornisce un'introduzione moderna alla progettazione, analisi ed implementazione di algoritmi sequenziali e paralleli. In particolare, il corso si basa su un approccio pragmatico alla programmazione parallela di algoritmi message-passing attraverso il linguaggio C e la libreria MPI.

PREREQUISITI

Analisi Matematica I e II, teoria della probabilità. Capacità di programmazione inl linguaggio C/C++.

OBIETTIVI FORMATIVI

Knowledge and understanding.
Gli studenti devono avere un solido background con un ampio spettro di conoscenze di base sugli algoritmi sequenziali e paralleli:

- gli studenti devono possedere gli strumenti cognitivi di base per pensare in modo analitico, creativo, critico e curioso, e possedere le capacità di astrazione e di risoluzione dei problemi necessarie per affrontare sistemi complessi;
- devono avere una solida conoscenza della progettazione e dell'implementazione di algoritmi efficienti sequenziali e paralleli;
- devono possedere gli strumenti per analizzare le risorse utilizzate dagli algoritmi;
- devono possedere un catalogo dei più noti ed efficienti algoritmi sequenziali e paralleli per problemi computazionali di base.
-

Applying knowledge and understanding.
Al termine del corso lo studente dovrebbe essere in grado di:

- Descrivere e utilizzare le principali tecniche di progettazione di algoritmi sequenziali;
- Progettare, dimostrare la correttezza e analizzare la complessità computazionale di algoritmi sequenziali;
- Comprendere le differenze tra diversi algoritmi che risolvono lo stesso problema e riconoscere quale sia migliore in condizioni diverse;
- Descrivere e utilizzare gli algoritmi sequenziali di base;
- Descrivere e utilizzare le strutture dati di base; conoscere l'esistenza di strutture dati avanzate;
- Comprendere la differenza tra algoritmi sequenziali e paralleli;
- Progettare, implementare e analizzare algoritmi paralleli basati sul message-passing in C/C++ utilizzando la libreria MPI;
- Descrivere e utilizzare algoritmi paralleli di base.

Making judgements. Gli studenti sono guidati ad apprendere criticamente tutto ciò che viene loro spiegato in classe, a confrontare diversi approcci alla soluzione di problemi algoritmici e a identificare e proporre, in modo autonomo, la soluzione più efficienti.

Communication. È fondamentale che lo studente sia in grado di comunicare con un pubblico vario e composito, non culturalmente omogeneo, in modo chiaro, logico ed efficace, utilizzando gli strumenti metodologici acquisiti e le proprie conoscenze scientifiche e, in particolare, il lessico specialistico. Il corso promuove lo sviluppo delle seguenti abilità dello studente: capacità di esporre in termini precisi e formali un modello astratto di problemi concreti, individuandone le caratteristiche salienti e scartando quelle non essenziali; capacità di descrivere e analizzare una soluzione efficace per un dato problema.

Learning skills. Gli studenti devono acquisire la capacità critica di rapportarsi, con originalità e autonomia, alle problematiche tipiche degli algoritmi sequenziali e paralleli e, in generale, alle questioni culturali legate ad altri ambiti simili. Dovranno essere in grado di sviluppare e applicare autonomamente le conoscenze e i metodi appresi in vista di un eventuale proseguimento degli studi a livello superiore (dottorato) o nella più ampia prospettiva di auto-miglioramento culturale e professionale dell'apprendimento permanente. Pertanto, gli studenti devono essere in grado di passare a forme espositive diverse dai testi di partenza per memorizzare, riassumere per sé e per gli altri e diffondere le conoscenze scientifiche.

METODI DIDATTICI	Il corso si propone di mettere gli studenti in grado di astrarre modelli e problemi algoritmici formali da problemi computazionali concreti e di progettare soluzioni algoritmiche efficienti per questi ultimi. A tal fine si utilizzerà il seguente metodo di insegnamento. Ogni problema computazionale sarà introdotto motivandolo con esempi concreti. La presentazione di ogni argomento sarà divisa in quattro parti: 1. Descrizione del problema computazionale concreto. 2. Modellazione del problema reale mediante un problema astratto. 3. Risoluzione del problema astratto attraverso un algoritmo ottenuto con l'applicazione delle tecniche generali di progettazione di algoritmi introdotte nel corso. 4. Analisi delle risorse utilizzate dall'algoritmo. Il corso consiste in lezioni frontali ed esercitazioni in aula. Ci saranno lezioni teoriche finalizzate all'apprendimento delle tecniche di base per la progettazione e l'analisi degli algoritmi, e una parte delle lezioni dedicata alle esercitazioni in cui si illustrerà, con dovizia di esempi, come le conoscenze teoriche acquisite possano essere utilizzate per risolvere problemi algoritmici di interesse pratico e implementare algoritmi paralleli in linguaggio C/C++ attraverso la libreria MPI.

MODALITA' D'ESAME

L'esame consiste in una prova scritta e nell'implementazione di un prototipo di software parallelo. La prova scritta (2 ore, 21 punti su 30) verte su argomenti teorici relativi alla progettazione e all'analisi di algoritmi sequenziali e paralleli, al fine di verificare la conoscenza e la comprensione della materia da parte dello studente. Per superare la prova scritta, gli studenti dovranno ottenere almeno 9 punti su 21. Il progetto di software parallelo (9 punti su 30) ha lo scopo di verificare la pratica della programmazione parallela e la capacità dello studente di implementare algoritmi paralleli teorici o di parallelizzare un algoritmo sequenziale. Per superare la prova di programmazione parallela, gli studenti devono ottenere almeno 4 punti su 9. Sia la prova scritta che l'implementazione del prototipo di software parallelo sono obbligatori. Gli studenti devono superare sia la prova scritta sia la prova di programmazione parallela; l'esame è superato solo se la somma dei punti ottenuti nella prova scritta e in quella di programmazione parallela è maggiore o uguale a 18 punti.
Il problema da risolvere in parallelo (o un algoritmo sequenziale da implementare in parallelo) viene assegnato su richiesta dello studente. Il progetto di software parallelo assegnato può essere discusso solo se la prova scritta è stata superata con successo; inoltre, il progetto di software parallelo deve essere obbligatoriamente discusso in uno degli appelli della stessa sessione in cui lo studente supera la prova scritta. A tal fine, il progetto (codice e relativa documentazione) deve essere inviato al docente via email almeno tre giorni prima dell'esame. Non saranno accettati progetti inviati oltre il termine indicato.

La relazione relativa al progetto assegnato deve essere strutturata come segue.

1. Introduzione. Lo studente deve fornire una descrizione accurata del progetto assegnato, comprensiva dell'analisi dell'algoritmo sequenziale che risolve il problema affrontato nel progetto. Qualora lo studente lo ritenga importante ai fini della comprensione, possono essere presenti pseudo-codice, esempi, grafici, figure, istanze applicative etc;

2. Design parallelo. Si tratta di uno studio preliminare volto ad evidenziare le opportunità per il parallelismo insite nel problema e/o nell'algoritmo sequenziale di partenza. Partendo dallo pseudo-codice occorre quindi determinare le funzionalità, le parti di codice e/o le strutture dati più opportune che possono essere prese in considerazione per la parallelizzazione. In questa fase devono essere considerati e discussi design alternativi, relativi a diverse strategie di parallelizzazione, motivando opportunamente perché alcune operazioni si prestano ad una parallelizzazione efficace ed altre no. Inoltre, devono essere riportati i risultati attesi, inclusa l'analisi della complessità parallela nel caso peggiore. È richiesta massima chiarezza espositiva in merito alla strategia di parallelizzazione, pertanto le strutture dati utilizzate devono essere descritte integralmente, motivando le scelte adottate per minimizzare il peso delle comunicazioni e della sincronizzazione. La valutazione analitica teorica della scalabilità della versione parallela è obbligatoria, e deve essere eseguita sia utilizzando la isoefficienza che la funzione di scalabilità.

3. Implementazione. Il design parallelo adottato deve essere implementato utilizzando il linguaggio di programmazione C o, eventualmente, C++ (in modo da usare le strutture dati presenti nella STL del C++). Il codice deve essere necessariamente commentato in modo adeguato. Nel caso in cui lo studente verifichi l'esistenza di strategie di parallelizzazione multiple per lo stesso problema assegnato, è possibile fornire una implementazione per ciascuna strategia, discutendo vantaggi e svantaggi di ogni strategia. Si noti che il codice parallelo non deve essere riportato integralmente nel testo della relazione, in quanto il codice del progetto - sorgenti e Makefile per il build (in alternativa è possibile usare CMake) - deve in ogni caso essere consegnato a parte. Tuttavia, la relazione può includere alcuni snippet di codice relativi alla parti più critiche ed interessanti.

4. Debug e test. Si raccomanda di effettuare un debug e test del codice parallelo che consenta di

verificare, ragionevolmente, l'assenza di bugs e la correttezza dell'algoritmo parallelo in relazione all'output prodotto. Si invita lo studente a progettare e verificare gli unit tests ritenuti idonei a valutare la bontà del codice. Lo studente è esplicitamente avvisato che l'implementazione di una strategia di parallelizzazione non corretta e/o la presenza di bugs che mandano in crash l'applicazione parallela a runtime comportano il non superamento dell'esame, mentre la presenza di bugs che inficiano la correttezza dell'algoritmo comporta una penalizzazione relativa al punteggio che sarà attribuito.

5. Analisi delle prestazioni e della scalabilità. Lo studente deve analizzare le prestazioni dell'implementazione sviluppata in termini di tempo di esecuzione, occupazione di memoria (se necessario), speedup ed efficienza. Deve essere valutata sia la strong scalability (Amdahl) che la weak scalability (Gustafson) ove possibile.

6. Sintesi. La relazione deve essere quanto più possibile sintetica, ma senza che ciò vada a discapito della chiarezza espositiva.

7. Valutazione del progetto. Il progetto sarà valutato con un punteggio relativo ad una scala che va da 1 a 9 punti. Il voto sarà definito solo al termine della discussione del progetto. La complessità del problema assegnato sarà presa in considerazione per la valutazione; gli studenti che si occupano di problemi più semplici devono quindi aspettarsi una minore indulgenza nella valutazione rispetto agli studenti che si occupano di problemi più difficili.

La valutazione terrà inoltre conto di:

- Chiarezza ed efficacia della presentazione;
- Profondità, correttezza ed originalità dell'analisi teorica;
- Capacità tecniche e documentazione dell'implementazione;
- Numero e qualità delle strategie multiple parallele, ove presenti;
- Pensiero critico nella valutazione e nell'analisi delle prestazioni;

- Significatività dei risultati: dato che la programmazione parallela si occupa principalmente di prestazioni, un buon progetto deve anche fornire un significativo speedup.

8. Plagio. Lo studente è avvisato esplicitamente che il plagio è gravissimo, ed è considerato molto seriamente. L'uso di fonti esterne di qualsiasi genere (internet, libri, dispense, lavori pregressi di altri studenti etc) è consentito solo per contributi marginali nel progetto (ad esempio, per la descrizione iniziale del progetto assegnato), ed ogni singola occorrenza deve essere esplicitamente citata e riportata nella relazione. La violazione di questo codice di comportamento non sarà in alcun modo tollerata, e comporta l'immediato annullamento dell'esame ed il deferimento dello studente alla commissione disciplinare di ateneo, con avvio del relativo procedimento disciplinare secondo quanto riportato del Titolo V (PROCEDIMENTI DISCIPLINARI E SANZIONI) del Regolamento di Ateneo per gli studenti (D.R. 672 del 5/12/2017, entrato in vigore in data 8/12/2017).

APPELLI D'ESAME

ALTRE INFORMAZIONI UTILI

Ricevimento Studenti
Su appuntamento; contattare il docente via e-mail o al termine degli incontri di classe.

PROGRAMMA ESTESO

Parallel Algorithms
Introduction. Parallel Architectures. Parallel algorithm design. Message-Passing Programming. Sieve of Erathostenes. Floyd all-pairs shortest path algorithm. Performance analysis. Matrix-vector multiplication. Document classification. Matrix multiplication. Linear Systems. Finite Difference Methods. Sorting.

Parallel Programming

Message-Passing programming using MPI.

Sequential Algorithms

Introduction. Order of growth. Analysis of algorithms. Decrease and conquer. Divide and conquer. Recurrences. Randomized algorithms. Transform and conquer. Dynamic programming. Greedy algorithms. Single Source Shortest Paths. Dijkstra algorithm. Breadth-First Search. Bellman-Ford Algorithm. Complexity and computability. NP-Completeness. The transition from sequential to parallel computing. Parallel complexity.

TESTI DI RIFERIMENTO

Introduction to Algorithms. Fourth edition. Cormen, Leiserson, Rivest, Stein. The MIT Press

Parallel Programming in C with MPI and OpenMP (International Edition). Michael J. Quinn. McGraw-Hill