# 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

***Algoritmi Sequenziali***

Introduzione agli algoritmi sequenziali. Tecniche di progettazione di algoritmi. Design ed analisi. Modello RAM. Decrease and conquer. Russian Peasant multiplication. Il problema dell'ordinamento. Insertion Sort. Analisi di Insertion Sort. Correttezza. Assertions. Invariants. Correttezza di Insertion Sort. Ordini di grandezza e notazioni asintotiche. Uso delle notazioni asintotiche. Classi di funzioni. Analisi del running time di un algoritmo usando le notazioni asintotiche. Determinare la complessità computazionale di un algoritmo. Divide and conquer. Merge Sort. Equazioni di ricorrenza. Recursion tree. Risoluzione di equazioni di ricorrenza. Metodo di sostituzione. Albero di ricorsione. Master theorem e sue estensioni. Teoremi di Akra-Bazzi. Metodo di Iterazione. Cambio di variabile. Ricorrenze lineari di ordine costante con coefficienti costanti. Divide and conquer. Merge Sort. Binary search. Interpolation search. Powering a number. Calcolo dell'ennesimo numero di Fibonacci. Algoritmo di Strassen per la moltiplicazione matriciale. Algoritmo di Karatsuba per la moltiplicazione di due numeri. Minimo e massimo simultanei tramite divide and conquer oppure mediante algoritmo iterativo. Majority element: divide and conquer algorithm. Algoritmo di Boyer and Moore. The hiring problem. Analisi nel caso peggiore. Analisi probabilistica di un algoritmo deterministico (average-case analysis). Variabili aleatorie indicatrici. Media di una variabile aleatoria indicatrice. Algoritmi randomizzati. Caratteristiche e differenze di un algoritmo randomizzato rispetto ad un algoritmo deterministico. Algoritmi randomizzati Monte Carlo e Las Vegas. Esempio di analisi di un algoritmo randomizzato: expected e worst-case running time. Algoritmo randomizzato di Freivald (Monte Carlo Matrix Multiplication Checker) per matrici con entries a valere su un campo GF(2). Correttezza dell'algoritmo di Freivald. Aumento della probabilità che l'output sia corretto da una probabilità costante ad una esponenzialmente elevata. Generalizzazione per matrici con entries a valere su un campo GF(d). QuickSort. Partizionamento. Analisi nel caso peggiore. Analisi nel caso medio. Considerazioni introduttive sulla complessita' dell'algoritmo nel caso medio. Un esempio di analisi nel caso medio errata. Disuguaglianza di Jensen. Analisi dettagliata del caso medio tramite variabili aleatorie indicatrici. Paranoid Quicksort. Analisi nel caso medio di Paranoid Quicksort. Transform and Conquer: instance simplification, representation change, problem reduction. Heaps. Max-Heap e Min-Heap. Max-Heapify. Build-Max-Heap. HeapSort. Priority Queues. Lower bound per algoritmi di ordinamento per confronti. Ordinamento in tempo lineare. Counting sort. Radix sort. Bucket sort. Statistiche d'ordine. Selezione in tempo atteso lineare. Selezione in tempo lineare nel caso peggiore. Introduzione alla programmazione dinamica. Matrix Chain Multiply. Optimal substructure. Overlapping subproblems. Top-down divide and conquer approach with memoization. Bottom-up iterative approach. Longest Common Subsequence. Elementi fondamentali della programmazione dinamica. Knapsack problem. 0-1 and fractional knapsack. Algoritmo basato su programmazione dinamica per 0-1 knapsack. Algoritmi pseudo-polinomiali. Strategia greedy. Greddy choice property. Activity selection problem. Algoritmo basato su programmazione dinamica per activity selection. Algoritmo greedy per activity selection. Confronto tra approccio greedy e programmazione dinamica. Algoritmo greedy applicato a Knapsack 0-1 e fractional knapsack. Minimum Spanning Tree. Algoritmo di Prim. Paths in graphs. Shortest paths. Optimal substructure. Triangle inequality. Negative weight cycles. Single-source shortest paths. Dijkstra's algorithm. Correctness and analysis. Unweighted graphs and breadth-first search. Correctness and analysis. Belmann-Ford algorithm. Correctness and analysis. Shortest paths in Directed Acyclic Graphs: topological sort and depth-first search. All-pairs shortest paths. Dynamic programming algorithm. Floyd-Warshall algorithm. Correctness and analysis. Transitive closure of a directed graph. Flow networks. Maximum flow problem. Flow cancellation. Net flow. Equivalence between positive and net flow. Properties of flow. Cuts. Flow across a cut. Capacity of a cut. Upper bound on the maximum flow value. Residual network. Residual capacities. Augmenting paths. Max-flow min-cut theorem. Ford-Fulkerson algorithm. Analysis. Pseudo-polinomiality of Ford-Fulkerson. Edmonds-Karp algorithm. Monotonicity lemma. Counting flow augmentations. Analysis of Edmonds-Karp. Maximum matching problem. Maximum and maximal matching. 2-approximation greedy algorithm

for maximal matching. Approximation algorithms. Maximum matching on bipartite graphs. Introduzione alla complessita' computazionale concreta. Presentazione informale delle classi di complessita' P, NP e NPC. Problemi astratti e concreti. Problemi decisionali. Codifica di un problema. Linguaggi formali. Caratterizzazione formale delle classi P, NP e NPC. Riduzioni in tempo polinomiale. Il problema P = NP e sue implicazioni. Problemi NP-Completi. Dimostrazioni di NP-Completeness. Circuit-SAT. SAT. 3-CNF-SAT. Clique. Vertex cover. Tipi di computazione: deterministica, randomizzata, nondeterministica. Computing models: RAM e Turing Machine. Nondeterminismo. Church-Turing thesis. Linguaggi decidibili, semidecidibili e indecidibili. Extended Church-Turing thesis. Sequential computing thesis. Upper bounds e lower bounds. Classi di complessita' di base. Relazioni tra RAM e Turing Machine. Classi di complessita' L, NL, P, NP, POLYLOGSPACE, PSPACE, EXP. Riduzioni e completezza. P-Completeness. Tesi del calcolo parallelo. Problemi altamente parallelizzabili. Classe Polylogspace. Classe NC. Problemi difficilmente parallelizzabili: i problemi P-Completi.

***Algoritmi Paralleli***

Introduction to parallel computing. Concurrency, parallelism and Bernstein's conditions. Modern scientific method. Evolution of supercomputing. Modern parallel supercomputers. Data Parallelism, functional parallelism and pipeline. Data and functional parallelism example: a generic clustering algorithm. Programming parallel computers. Extending sequential compilers. Extending sequential programming languages. Adding a parallel language layer on top of a sequential programming language. Design and implement a new parallel language and its compiler. MPI and OpenMP. Interconnection networks. Shared and switched medium. Static (direct) and dynamic (indirect) topologies. Switch's functionalities. Latency and its four components: software overhead, channel delay, switching delay, contention time. Evaluation of network topologies: bisection bandwidth, bisection width, diameter, number of edges per switch node and edge length. 2D mesh, linear network, binary tree, fat tree, hypertree, butterfly, hypercube, shuffle-exchange. Computer vettoriali. Processor array. Multiprocessori. Multiprocessori centralizzati (UMA). Cache coherence. Write-invalidate protocol. Sincronizzazione. Multiprocessori distribuiti (NUMA). Directory based protocol. Multicomputer. Asymmetrical e symmetrical multicomputers. Commodity clusters. Network of Workstations. Tassonomia di Flynn. Task-Channel model. Metodologia PCAM. Partitioning. Domain e functional decomposition. Comunicazione. Comunicazione locale e globale, strutturata e non strutturata. Agglomerazione. Granularity. Surface to volume effect. Communication/computation ratio. Mapping. Static and dynamic mapping decision tree. Case studies. Boundary value problem. Determinare il massimo. Reduction. The n-body problem. Gather. All-gather. Adding data input. Scatter. message-Passing model. Libreria MPI. Circuit-SAT: parallelizzazione tramite MPI. User-Defined Reductions. Derived datatypes. Benchmarking. How to install the MPI library on linux, macOS and Windows. Crivello di Eratostene. Parallelizzazione. Allocazione a blocchi. Broadcast in MPI. Sieve performance enhancements. Eliminare i numeri pari, replicare il calcolo per eliminare le comunicazioni, riorganizzare i cicli per migliorare il cache hit rate. Analysis and benchmarking. All-pairs shortest path problem. Dynamic 2-D arrays in C. Parallel algorithm design: Partitioning. Communication. Agglomeration e mapping. Rowwise e Columnwise block striped decompositions. Point-to-point communication in MPI. Block row matrix I/O. Analysis and benchmarking. Overlapping communication and computation. Analisi delle prestazioni. Communication overhead. Total overhead. Work. Cost. Degree of concurrency. Maximum and average degree of concurrency. Critical Path Length, also known as span or depth. Parallelism. Work Law. Span Law. Brent's theorem. Execution time components. General speedup formula. Optimal number of processors. Super-linear speedup. Another definition of overhead, based on the speedup. Efficiency. Amdahl's effect. Amdahl's Law. Gustafson' Law. Karp-Flatt metric. Isoefficiency Metric. Scalability. Strongly scalable and weakly scalable algorithms. Quasi-scalable algorithms and the scaling zone. Scalability function. Cost-Optimality (Work-Efficiency) and Isoefficiency. Scaling down dei processori. Impatto della mancanza di cost-optimality. Minimum Parallel execution Time.

Minimum Cost-Optimal Parallel Execution Time. Moltiplicazione matrice-vettore. Sequential algorithm and its complexity. Design, analysis, and implementation of parallel programs based on Rowwise block striped, and Columnwise block striped decompositions. Replication of vectors. All-gather. MPI_Allgatherv. All-to-all. MPI_Scatterv. MPI_Gatherv. MPI_Alltoallv. Moltiplicazione matrice-vettore: algoritmo basato su decomposizione checkerboard block. Design ed analisi. Creazione di communicators con topologia cartesiana in MPI. Creazione di communicators mediante MPI_Comm_split. Document classification. Parallel algorithm design. Manager-worker paradigm (master-slave). Creating communicators. Non-blocking communications. Additional MPI functions. Moltiplicazione matriciale. Sequential algorithms: Iterative row-oriented and Recursive block-oriented. Parallel algorithms: Rowwise block striped decomposition and Cannon's algorithm. DNS Algorithm. SUMMA algorithm. Fox's matrix multiplication algorithm. Dense linear systems. Special matrices. Back substitution algorithm. Parallel row-oriented algorithm. Parallel column-oriented algorithm. Gaussian elimination algorithm. Avoiding numerical issues: partial pivoting. Parallel row-oriented algorithm MPI reduction operators MPI_MAXLOC and MPI_MINLOC. Parallel column-oriented algorithm. Pipelined row-oriented algorithm. Sparse linear systems. Jacobi and Gauss-Seidel methods. Conjugate Gradient Method. Parallel algorithm based on rowwise block striped decomposition. Ordinary and partial differential equations. Examples of Phenomena Modeled by PDEs. Solving PDEs. Linear Second-order PDEs. Difference Quotients. Finite difference methods. Vibrating string problem. Discretization. Parallelization. Ghost cells. Analysis. Replication of computations done by neighbouring processes to reduce communications. Steady state heat distribution problem. Parallel algorithm based on rowwise block striped and checkerboard block decompositions. Analysis. Sorting problem. Sequential Quicksort algorithm. Parallel Quicksort. Complexity and isoefficiency analysis. HyperQuicksort. Complexity and isoefficiency analysis. Parallel sorting by regular sampling. Complexity and isoefficiency analysis. Analysis of parallel Floyd's algorithm based on checkerboard block decomposition.

***Parallel Programming***

Message-Passing programming using the MPI library.

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