Università degli Studi di Bergamo

SCUOLA DI INGEGNERIA
Corso di Laurea Triennale in Ingegneria Informatica
Classe n. L-8 Ingegneria dell'Informazione (D.M. 270/04)

Introduzione al calcolo parallelo in MATLAB®

Candidato:
Thomas Fabbris
Matricola 1086063

Relatore:
Chiar.mo Prof. Fabio Previdi

Anno Accademico 2024–2025

Indice

Introduzione

I primi progettisti di calcolatori, negli anni '50 del secolo scorso, hanno avuto sin da subito l'intuizione di interconnettere una moltitudine di calcolatori tradizionali, al fine di ottenere un sistema di elaborazione sempre più potente. Questo sogno primordiale ha portato alla nascita dei *cluster* di elaboratori trent'anni dopo e allo sviluppo delle architetture *multicore* a partire dall'inizio del 2000. Oggi la maggior parte delle applicazioni in ambito scientifico, tra cui quelle impiegate nella risoluzione di problemi di analisi numerica su larga scala, possono funzionare solo disponendo di sistemi di calcolo in grado di fornire una capacità di elaborazione molto elevata.

Nel capitolo 1 di questo lavoro di tesi, ci concentreremo sul concetto di calcolo parallelo e sulle principali sfide da affrontare durante la scrittura di software eseguito su più processori simultaneamente, tra cui spicca una crescita delle prestazioni non proporzionale al miglioramento apportato al sistema di elaborazione, un risultato espresso quantitativamente dalla legge di Ahmdal.

Nel corso del capitolo 2, analizzeremo i principali costrutti di programmazione parallela messi a disposizione dall'ambiente di calcolo numerico e programmazione MATLAB®, nonché le scelte di progettazione fondamentali che hanno influenzato le attuali caratteristiche del linguaggio dedicate alla scrittura di programmi ad esecuzione parallela.

Nel capitolo 3, descriveremo dal punto di vista formale il metodo di Jacobi, un metodo iterativo dell'analisi numerica per la risoluzione approssimata di sistemi di equazioni lineari, dopo aver introdotto alcune nozioni di algebra lineare la cui conoscenza è necessaria per un'adeguato sviluppo dell'argomento. Successivamente, proporremo un'implementazione parallela dell'algoritmo dietro al metodo di Jacobi, sfruttando le potenzialità fornite dagli *array* globali, utili per aumentare il livello di astrazione di un programma parallelo. Infine, ci occuperemo dell'analisi dei risultati ottenuti dall'esecuzione dell'algoritmo su problemi di grandi dimensioni.

Capitolo 1

Calcolo parallelo: sfida o opportunitá?

L'obiettivo di questo capitolo é dare una definizione precisa di calcolo parallelo, un termine spesso impiegato all'interno del mondo del supercalcolo per riferirsi all'uso simultaneo di molteplici risorse di calcolo per la risoluzione di problemi ad elevata intensità computazionale, tipici delle applicazioni sviluppate in ambito scientifico, in tempi ragionevolmente brevi.

Andremo, poi, ad investigare le cause che hanno portato alla nascita del parallelismo e a descrivere le principali difficolt'radizionalmente incontrate dai programmatori durante l'implementazione di programmi ad esecuzione parallela.

1.1 Introduzione al calcolo parallelo

L'idea alla base del calcolo parallelo é che gli utenti di un qualsiasi sistema di elaborazione possono avere a disposizione tanti processori quanti ne desiderano, per poi interconnetterli a formare un sistema multiprocessore, le cui prestazioni sono, con buona approssimazione, proporzionali al numero di processori montati.

La sostituzione di un singolo processore caratterizzato da un'elevata capacitá di calcolo, tipicamente presente nelle architetture di sistemi *mainframe*, con un insieme di processori piú efficienti dal punto di vista energetico permette di raggiungere migliori prestazioni per unitá di energia, a condizione che i programmi eseguiti siano appositamente progettati per sfruttare pienamente la potenza di calcolo di ogni singolo processore; approfondiremo questi aspetti nei paragrafi 1.2 e 1.3.

Una tendenza introdotta da IBM nel 2001 nell'ambito della progettazione di sistemi paralleli [2] è il raggruppamento di diverse unitá di calcolo all'interno di un singolo circuito integrato; in questo contesto, i processori montati su un singolo *chip* di silicio vengono chiamati *core*. Il microprocessore *multicore* risultante appare al sistema operativo in esecuzione sull'elaboratore come l'insieme di N processori, ognuno dei quali dotato di un set di registri e di una memoria *cache* dedicati; solitamente si tratta di sistemi multiprocessore a memoria condivisa, in cui i *core* condividono lo stesso spazio di indirizzamento fisico.
Il funzionamento di questa categoria di sistemi multiprocessore si basa sul parallelismo a livello di attività (o a livello di processo): più processori sono impiegati per svolgere diverse attivitá simultaneamente e ciascuna attivitá corrisponde ad applicazioni a singolo *thread*. In generale, ogni *thread* esegue un'operazione ben definita e *thread* differenti possono agire sugli stessi dati o su insiemi di dati diversi.

D'altro canto, tutte le applicazioni che richiedono un utilizzo intensivo di risorse computazionali, oggi diffuse non piú esclusivamente in ambito scientifico ma anche in settori come quello dei motori di ricerca o dell'*hosting* di siti Web, possono essere eseguite solo su *cluster* di elaboratori, una tipologia di sistemi multiprocessore che si differenzia dai microprocessori *multicore* per il fatto di essere costituita da un insieme di calcolatori completi, chiamati nodi, collegati tra loro per mezzo di una rete LAN (*Local Area Network*).
In ogni caso, il funzionamento di un sistema di elaborazione parallelo si basa sull'uso congiunto di processori diversi.
Per sfruttare al meglio le potenzialitá offerte dai *cluster* di elaboratori, i programmatori di applicazioni devono sviluppare programmi a esecuzione parallela efficienti e scalabili a seconda del numero di processori a disposizione durante l'esecuzione; risulta necessario applicare il parallelismo a livello di dati, un approccio che prevede la distribuzione di un insieme di dati di partenza sulle CPU del *cluster* per poi eseguire la medesima operazione, ma su un insieme di dati diverso, su ogni processore.

Una tipica operazione parallelizzabile a livello di dati è la somma vettoriale poichè ogni componente del vettore risultante viene ottenuta sommando le corrispondenti componenti omologhe dei due vettori di partenza, in modo indipendente dalle altre; possiamo sin da subito intuire che una condizione necessaria per la parallelizzazione di un qualsiasi algoritmo è l'indipendenza tra le operazioni eseguite in un certo passo. Ad esempio, supponiamo di dover sommare due vettori di numeri reali di dimensione N avvalendoci di un sistema *dual-core*, ossia di un sistema parallelo dotato di un microprocessore con due core distinti. Sarebbe conveniente avviare un

thread separato su ogni *core* specializzato nella somma di due componenti omologhe dei vettori operandi; attraverso una opulata distribuzione dei dati, otterremmo che il *thread* sul primo *core* somma le componenti omologhe dei vettori di partenza da 1 a $\lfloor \frac{N}{2} \rfloor$, mentre il secondo *core* si occuperebbe della somma delle componenti da $\lceil \frac{N}{2} \rceil$ a N.

In realtà, la rigida classificazione proposta tra parallelismo a livello di attivitá e parallelismo a livello di dati non ritrova un diretto riscontro nella realtá, in quanto sono stati sviluppati programmi che sfruttano entrambi gli approcci al fine di massimizzare le prestazioni.

Cogliamo l'occasione per precisare la terminologia, in parte giá utilizzata, impiegata per descrivere la componente hardware e la componente software di un sistema di elaborazione: l'hardware, descrivendo con questo termine il processore del sistema, puó essere classificato in seriale, come nel caso di un processore *single core*, o parallelo, come nel caso di un processore *multicore*, mentre il software viene detto sequenziale o concorrente, a seconda della presenza o meno di diversi processi cooperanti la cui esecuzione é influenzata dagli altri processi presenti nel sistema.
Naturalmente, un programma concorrente puó essere eseguito sia su hardware seriale che su hardware parallelo.
Infine, con il termine programma ad esecuzione parallela, o piú semplicemente software parallelo, indicheremo un programma (sequenziale o concorrente) eseguito su hardware parallelo.

1.2 Le cause alla nascita del parallelismo

1.3 Sfide della progettazione di software parallelo

Una caratteristica fondamentale posseduta da ogni programma a esecuzione parallela è la scalabilitá, ovvero la capacitá di un sistema software di incrementare le proprie prestazioni in funzione della potenza di calcolo richiesta in un preciso istante [1], che consente di ottenere architetture tolleranti ai guasti assieme a un'elevata disponibilitá del sistema.

Bibliografia

[1] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, "Scale-up x Scale-out: A Case Study using Nutch/Lucene," in *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–8.

[2] J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5–25, dec 2001.