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 Cinquanta 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 nel mondo del supercalcolo per riferirsi all'uso simultaneo di molteplici risorse di calcolo per la risoluzione di problemi ad elevata intensità computazionale in tempi ragionevolmente brevi.

In seguito, investigheremo le cause che hanno portato alla nascita del parallelismo e a descrivere le principali difficoltá 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 impiegati.

La sostituzione di un singolo processore caratterizzato da un'elevata capacitá di calcolo, tipicamente presente nelle architetture dei 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 essere eseguiti su hardware parallelo; approfondiremo questi aspetti nei paragrafi 1.2 e 1.3.

Una tendenza introdotta da IBM nel 2001 nell'ambito della progettazione di sistemi paralleli [4] 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, ciascuno dotato di un set di registri e di una memoria *cache* dedicati; solitamente i microprocessori *multicore* sono impiegati in sistemi 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 a un'applicazione 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, garantendo un elevato *throughput* per attivitá tra loro indipendenti.

D'altro canto, tutte le applicazioni che richiedono un utilizzo intensivo di risorse computazionali, diffuse non piú solamente in ambito scientifico, 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 distinti.
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 disponibili durante l'esecuzione; risulta necessario applicare un parallelismo a livello di dati, che prevede la distribuzione dell'insieme dei dati in input tra le CPU del *cluster*, per poi far eseguire la medesima operazione, su sottoinsiemi distinti di dati, a ogni processore.

Una tipica operazione parallelizzabile a livello di dati è la somma vettoriale perché le componenti del vettore risultante vengono ottenute sommando solamente le componenti omologhe dei due vettori di partenza; possiamo intuire fin da subito che una condizione necessaria per la parallelizzazione di un qualsiasi algoritmo è l'indipendenza tra le operazioni eseguite ad un certo passo dell'esecuzione.
Per esempio, supponiamo di dover sommare due vettori di numeri reali di dimensio-

ne N avvalendoci di un sistema *dual-core*, ossia di un sistema di elaborazione dotato di un microprocessore che contiene al suo interno due *core*.
Sarebbe conveniente avviare un thread separato su ogni *core*, specializzato nella somma di due componenti omologhe dei vettori operandi; attraverso un'attenta distribuzione dei dati in input, il *thread* in esecuzione sul primo *core* sommerebbe le componenti da 1 a $\lceil \frac{N}{2} \rceil$ dei vettori di partenza, e al contempo il secondo *core* si occuperebbe della somma delle componenti da $\lceil \frac{N}{2} \rceil + 1$ a N.

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

Cogliamo l'occasione per precisare la terminologia, in parte giá utilizzata, per descrivere la componente hardware e la componente software di un generico calcolatore: l'hardware, riferendoci con questo termine esclusivamente al processore, puó essere 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 di processi cooperanti la cui esecuzione viene 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 a esecuzione parallela, o piú semplicemente software parallelo, indichiamo un programma, sequenziale o concorrente, eseguito su hardware parallelo.

1.2 Le cause a supporto del parallelismo

L'attenzione riservata al calcolo parallelo da parte della comunitá scientifica risale al 1957, anno in cui la Compagnie des Machines Bull (l'odierna Bull SAS) ha annunciato Gamma 60, equipaggiato con la prima architettura della storia in grado di offrire un supporto diretto al parallelismo, mentre, l'anno successivo, i ricercatori IBM John Cocke e Daniel Slotnick hanno per la prima volta aperto alla possibilitá di impiegare il *parallel computing* per l'esecuzione di simulazioni numeriche [5].

Anche oggi permangono applicazioni in ambito scientifico che possono essere eseguite solo su *cluster* di elaboratori oppure che richiedendo lo sviluppo di architetture specifiche di dominio (DSA, *Domain Specific Architecture*), considerate le loro ca-

ratteristiche *compute-intensive*.
Esempi di settori che hanno beneficiato dello sviluppo di architetture innovative per il calcolo parallelo sono la bioinformatica, l'elaborazione di immagini e video e il settore aerospaziale, che ha potuto contare su simulazioni numeriche sempre piú accurate.
La rivoluzione introdotta dal calcolo parallelo non si limita esclusivamente al campo scientifico; un dominio applicativo che negli ultimi due decenni ha registrato uno sviluppo senza precedenti é l'intelligenza artificiale (AI, *Artificial Intelligence*) e, in particolare, l'addestramento di modelli di AI mediante tecniche di *Machine Learning*; i successi e le evoluzioni ottenuti in questo settore, e tangibili in diversi campi di applicazione come il riconoscimento di oggetti o l'industria della traduzione, non sarebbero stati fattibili se non supportati da sistemi di calcolo sufficientemente potenti in grado di eseguire le operazioni aritmetiche richieste per svolgere compiti sempre piú complessi.
Come ulteriore esempio, possiamo citare i calcolatori dei moderni centri di calcolo, chiamati *Warehouse Scale Computer* (WSC), che costituiscono l'infrastruttura di erogazione dei moderni servizi Internet utilizzati ogni giorno da milioni di utenti, come i motori di ricerca, i *social network* e i servizi di *e-commerce*. Inoltre, la rivoluzione del *cloud computing*, ovvero l'offerta via Internet di risorse di elaborazione *"as a service"*, consente l'accesso ai WSC a chiunque sia dotato di una carta di credito.

Il fattore fondamentale dietro all'adozione a un'adozione di massa delle architetture multiprocessore é la riduzione del consumo di energia elettrica da parte dei sistemi di elaborazione; infatti, l'alimentazione e il raffreddamento delle centinaia di server presenti in un centro di calcolo moderno costituiscono una componente di costo non trascurabile, che risente solo marginalmente dalla disponibilitá di sistemi di raffreddamento dei microprocessori in grado di dissipare una grande quantitá di energia.

Il consumo di energia elettrica dei microprocessori viene misurato in Joule (J) ed é quasi interamente rappresentato dalla dissipazione di energia dinamica da parte dei transistori CMOS (*Complementary Metal Oxide Semiconductor*), essendo la tecnologia dominante impiegata nella realizzazione dei moderni circuiti integrati. Un transistore assorbe prevalentemente energia elettrica durante la commutazione alto-basso-alto del suo stato di uscita, secondo la formula

$$E = V^{+2} \cdot C_L$$

dove V^+ rappresenta la tensione di alimentazione e C_L la capacitá di carico del transistore.
La potenza dissipata P_D, assumendo che la frequenza di commutazione dello stato del transistore sia pari a f, é quindi data da

$$P_D = f \cdot E = f \cdot C_L \cdot V^{+2} \propto f_C$$

dove f_C é la frequenza di *clock* del circuito, esprimibile come funzione di f.

In passato, i progettisti di circuiti integrati hanno tentato di contenere l'assorbimento di energia da parte dei microprocessori riducendo la tensione di alimentazione V^+ di circa il 15% ad ogni nuova generazione di CPU fino al raggiungimento del limite inferiore di 1 V; al contempo, la diminuzione della tensione di alimentazione ha favorito la crescita delle correnti di dispersioni interne al transistore, tanto che nel 2008 circa il 40% della potenza assorbita da un transistore era dovuta alla dispersione di corrente.

In figura 1.1, possiamo notare come fino alla prima metá degli anni Ottanta del secolo scorso, la crescita annua delle prestazioni dei processori si attestava al 25%, per poi passare al 52% grazie a importanti innovazioni nella progettazione e nell'organizzazione dei calcolatori; infine dal 2002, si sta continuando a registrare una crescita delle prestazioni pari 3.5% annuo a causa del raggiungimento dei limiti relativi la potenza assorbita.

La presenza di queste limitazioni tecnologiche ha accelerato la ricerca di nuove architetture per la realizzazione dei microprocessori, culminata con lo sviluppo del primo processore *multicore* Power4 di IBM nel 2001 e il successivo lancio dei primi modelli destinati al mercato *consumer multicore* nel 2006 da parte di Intel e AMD. In futuro, l'aumento delle prestazioni dei microprocessori sará verosimilmente segnato dall'aumento del numero di *core* montati su un singolo *chip* piuttosto che dall'aumento della frequenza di clock dei processori *single core*.

1.3 Sfide della progettazione di software parallelo

Una caratteristica fondamentale posseduta da ogni programma a esecuzione parallela è la scalabilitá, ovvero la capacitá del sistema software di incrementare le proprie prestazioni in funzione della potenza di calcolo richiesta in un preciso istante [1]; ció permette incidentalmente di realizzare sistemi tolleranti ai guasti e ad elevata disponibilitá.

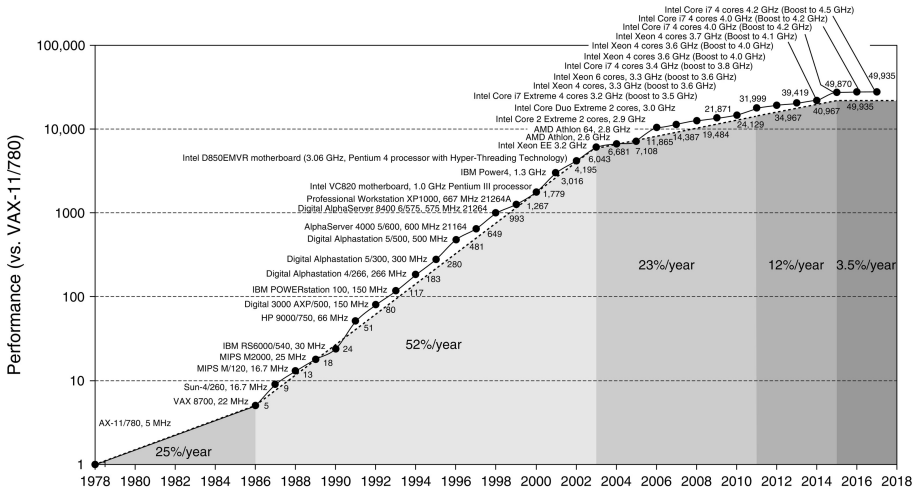

Figura 1.1: Crescita nelle prestazioni dei processori dal 1978 al 2018; il grafico riporta le prestazioni dei processori disponibili sul mercato relative al VAX11/780 misurate attraverso i *benchmark* SPECint.

Fonte: J.L. Hennessy, D.A. Patterson, Computer Architecture: A quantitative Approach. Ed. 6. Waltham, MA:Elsevier, 2017

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] D. A. Patterson and J. L. Hennessy, *Struttura e progetto dei calcolatori*, 5th ed. Bologna: Zanichelli, 2022, A cura di Alberto Borghese.

[3] P. Spirito, *Elettronica Digitale*, 3rd ed. McGraw-Hill Education, 2021.

[4] 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.

[5] G. V. Wilson, "The History of the Development of Parallel Computing," Virginia Tech/Norfolk State University, Interactive Learning with a Digital Library in Computer Science, 1994.