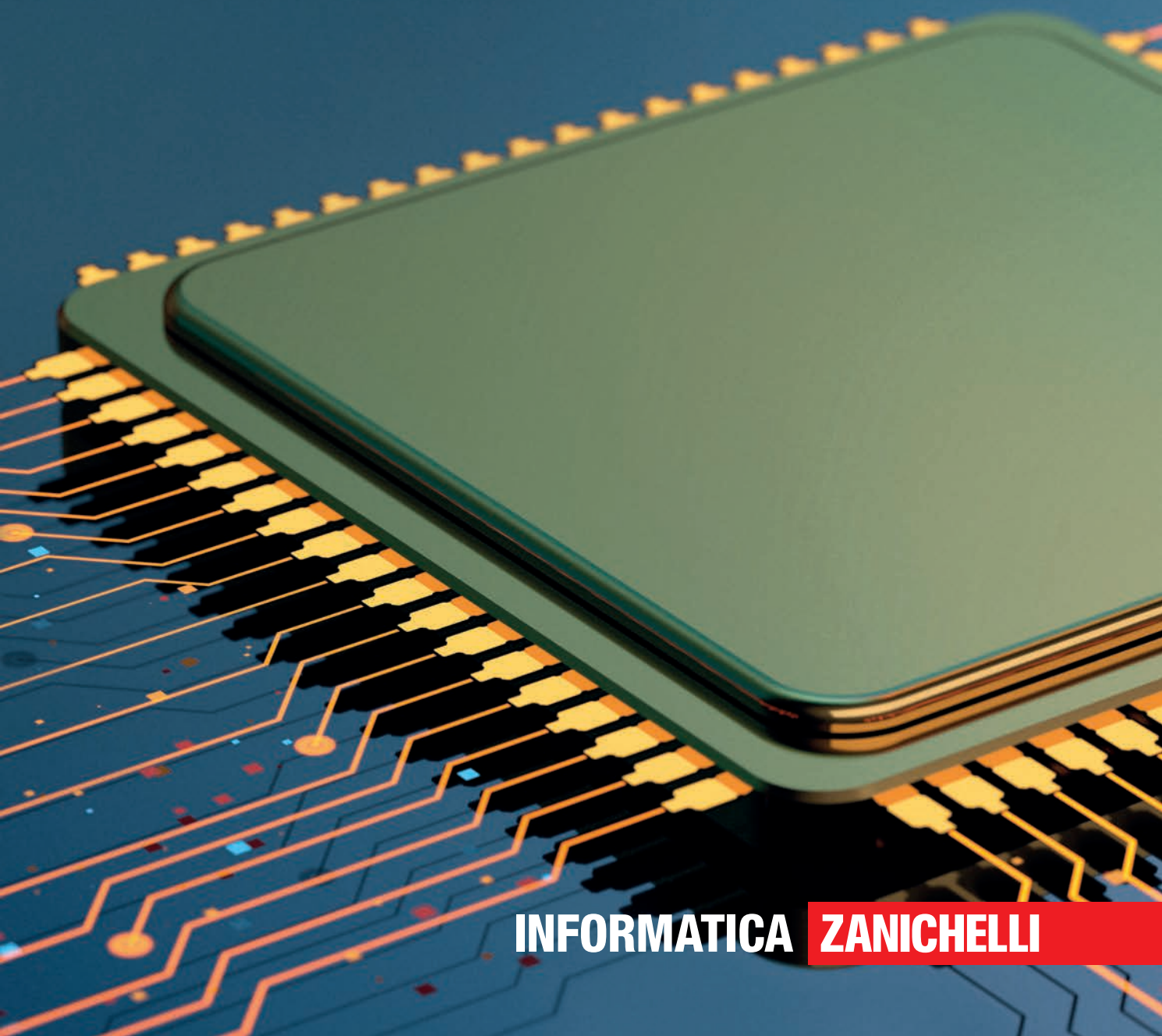


David A. Patterson John L. Hennessy

Struttura e progetto dei calcolatori

Progettare con RISC-V

Seconda edizione italiana a cura di Alberto Borghese



INFORMATICA **ZANICHELLI**

David A. Patterson John L. Hennessy

Struttura e progetto dei calcolatori

Progettare con RISC-V

Seconda edizione italiana a cura di Alberto Borghese

Se vuoi accedere alle risorse online riservate

1. Vai su **my.zanichelli.it**
2. Clicca su *Registrati*.
3. Scegli *Studente*.
4. Segui i passaggi richiesti per la registrazione.
5. Riceverai un'email: clicca sul link per completare la registrazione.
6. Cerca il tuo codice di attivazione stampato in verticale sul bollino argentato in questa pagina.
7. Inseriscilo nella tua area personale su **my.zanichelli.it**

Se sei già registrato, per accedere ai contenuti riservati ti serve solo il codice di attivazione.

Titolo originale: *Computer Organization and Design RISC-V Edition*, second edition
Copyright © 2021 Elsevier Inc. All Rights Reserved.
Published by arrangement with Elsevier Inc., 230 Park Avenue South, New York, USA.
Morgan Kauffman is an imprint of Elsevier.

This edition of *Computer Organization and Design RISC-V Edition*, second edition, ISBN 978-0-12-820331-6 by **David Patterson** and **John Hennessy** is published by arrangement with Elsevier Inc.
Questa traduzione di *Computer Organization and Design RISC-V Edition*, second edition, ISBN 978-0-12-820331-6 di **David Patterson** e **John Hennessy** è pubblicata con l'autorizzazione di Elsevier Inc.

Questa traduzione è stata realizzata da Zanichelli editore ed è di sua esclusiva responsabilità.
Professionisti e ricercatori devono sempre basarsi sulla propria esperienza e sulle proprie conoscenze nel valutare e utilizzare qualsiasi informazione, metodo, composto chimico o esperimento qui descritto.

Nessuna responsabilità è assunta di fronte alla legge da Elsevier, dagli autori, redattori o collaboratori rispetto alla traduzione o in relazione a qualsiasi infortunio e/o danno arrecato a persone o proprietà a causa di prodotti difettosi, negligenza o altro, o in seguito all'uso o al funzionamento di qualsiasi metodo, prodotto, istruzione o idea descritti nel testo qui presente.

RISC-V and the RISC-V logo are registered trademarks managed by the RISC-V Foundation, used under permission of the RISC-V Foundation. All rights reserved.

This publication is independent of the RISC-V Foundation, which is not affiliated with the publisher, and the RISC-V Foundation does not authorize, sponsor, endorse or otherwise approve this publication.

All material relating to the ARM technology has been reproduced with permission from ARM Limited, and should only be used for education purposes. All ARM-based models shown or referred to in the text must not be used, reproduced or distributed for commercial purposes, and in no event shall purchasing this textbook be construed as granting you or any third party, expressly or by implication, estoppel or otherwise, a license to use any other ARM technology or know how. Materials provided by ARM are copyright ©ARM Limited (or its affiliates).

© 2023 Zanichelli editore S.p.A., via Innerio 34, 40126 Bologna [19966]
www.zanichelli.it

Traduzione: Alberto Borghese

Diritti riservati

I diritti di pubblicazione, riproduzione, comunicazione, distribuzione, trascrizione, traduzione, noleggio, prestito, esecuzione, elaborazione in qualsiasi forma o opera, di memorizzazione anche digitale e di adattamento totale o parziale su supporti di qualsiasi tipo e con qualsiasi mezzo (comprese le copie digitali e fotostatiche), sono riservati per tutti i paesi. L'acquisto della presente copia dell'opera non implica il trasferimento dei suddetti diritti né li esaurisce.

Fotocopie e permessi di riproduzione

Le fotocopie per uso personale (cioè privato e individuale, con esclusione quindi di strumenti di uso collettivo) possono essere effettuate, nei limiti del 15% di ciascun volume, dietro pagamento alla S.I.A.E. del compenso previsto dall'art. 68, commi 4 e 5, della legge 22 aprile 1941 n. 633. Tali fotocopie possono essere effettuate negli esercizi commerciali convenzionati S.I.A.E. o con altre modalità indicate da S.I.A.E.

Per le riproduzioni ad uso non personale (ad esempio: professionale, economico, commerciale, strumenti di studio collettivi, come dispense e simili) l'editore potrà concedere a pagamento l'autorizzazione a riprodurre un numero di pagine non superiore al 15% delle pagine del presente volume.

Le richieste vanno inoltrate a:

Centro Licenze e Autorizzazioni per le Riproduzioni Editoriali (CLEARedi),
Corso di Porta Romana 108, 20122 Milano
e-mail: autorizzazioni@clearedi.org e sito web: www.clearedi.org

L'autorizzazione non è concessa per un limitato numero di opere di carattere didattico riprodotte nell'elenco che si trova all'indirizzo

www.zanichelli.it/chi-siamo/fotocopie-e-permessi

L'editore, per quanto di propria spettanza, considera rare le opere fuori del proprio catalogo editoriale. La loro fotocopia per i soli esemplari esistenti nelle biblioteche è consentita, anche oltre il limite del 15%, non essendo concorrenziale all'opera. Non possono considerarsi rare le opere di cui esiste, nel catalogo dell'editore, una successiva edizione, né le opere presenti in cataloghi di altri editori o le opere antologiche. Nei contratti di cessione è esclusa, per biblioteche, istituti di istruzione, musei e archivi, la facoltà di cui all'art. 71-ter legge diritto d'autore.

Per permessi di riproduzione, diversi dalle fotocopie, rivolgersi a ufficiocontratti@zanichelli.it

Licenze per riassunto, citazione e riproduzione parziale a uso didattico con mezzi digitali
La citazione, la riproduzione e il riassunto, se fatti con mezzi digitali, sono consentiti (art. 70 bis legge sul diritto d'autore), limitatamente a brani o parti di opera, a) esclusivamente per finalità illustrative a uso didattico, nei limiti di quanto giustificato dallo scopo non commerciale

perseguito. (La finalità illustrativa si consegue con esempi, chiarimenti, commenti, spiegazioni, domande, nel corso di una lezione); b) sotto la responsabilità di un istituto di istruzione, nei suoi locali o in altro luogo o in un ambiente elettronico sicuro, accessibili solo al personale docente di tale istituto e agli alunni o studenti iscritti al corso di studi in cui le parti di opere sono utilizzate; c) a condizione che, per i materiali educativi, non siano disponibili sul mercato licenze volontarie che autorizzano tali usi. Zanichelli offre al mercato due tipi di licenze di durata limitata all'anno accademico in cui le licenze sono concesse:

A) licenze gratuite per la riproduzione, citazione o riassunto di una parte di opera non superiore al 5%. Non è consentito superare tale limite del 5% attraverso una pluralità di licenze gratuite.
B) licenze a pagamento per la riproduzione, citazione, riassunto parziale ma superiore al 5% e comunque inferiore al 40% dell'opera. Per usufruire di tali licenze occorre seguire le istruzioni su www.zanichelli.it/licenzeeducative

L'autorizzazione è strettamente riservata all'istituto educativo licenziatario e non è trasferibile in alcun modo e a qualsiasi titolo.

Garanzie relative alle risorse digitali

Le risorse digitali di questo volume sono riservate a chi acquista un volume nuovo: vedi anche al sito www.zanichelli.it/contatti/acquisti-e-recesso le voci

Informazioni generali su risorse collegate a libri cartacei e Risorse digitali e libri non nuovi.

Zanichelli garantisce direttamente all'acquirente la piena funzionalità di tali risorse.

In caso di malfunzionamento rivolgersi a assistenza@zanichelli.it

La garanzia di aggiornamento è limitata alla correzione degli errori e all'eliminazione di malfunzionamenti presenti al momento della creazione dell'opera.

Zanichelli garantisce inoltre che le risorse digitali di questo volume sotto il suo controllo saranno accessibili, a partire dall'acquisto, per tutta la durata della normale utilizzazione didattica dell'opera. Passato questo periodo, alcune o tutte le risorse potrebbero non essere più accessibili o disponibili: per maggiori informazioni, leggi my.zanichelli.it/fuoricatalogo

Soluzioni degli esercizi e altri svolgimenti di compiti assegnati

Le soluzioni degli esercizi, compresi i passaggi che portano ai risultati e gli altri svolgimenti di compiti assegnati, sono tutelate dalla legge sul diritto d'autore in quanto elaborazioni di esercizi a loro volta considerati opere creative tutelate, e pertanto non possono essere diffuse, comunicate a terzi e/o utilizzate economicamente, se non a fini esclusivi di attività didattica.

Diritto di TDM

L'estrazione di dati da questa opera o da parti di essa e le attività connesse non sono consentite, salvi i casi di utilizzazioni libere ammessi dalla legge.

L'editore può concedere una licenza.

La richiesta va indirizzata a tdm@zanichelli.it

Realizzazione editoriale: Epitesto, Milano

Copertina:

– Progetto grafico: Falcinelli & Co., Roma

– Immagine di copertina: © MF3d/Getty Images

Prima edizione italiana: giugno 2019

Seconda edizione italiana: maggio 2023

Ristampa: **prima tiratura**

5 4 3 2 1 2023 2024 2025 2026 2027

Realizzare un libro è un'operazione complessa, che richiede numerosi controlli: sul testo, sulle immagini e sulle relazioni che si stabiliscono tra essi. L'esperienza suggerisce che è praticamente impossibile pubblicare un libro privo di errori. Saremo quindi grati ai lettori che vorranno segnalarceli. Per segnalazioni o suggerimenti relativi a questo libro scrivere al seguente indirizzo:

Zanichelli editore S.p.A.

Via Innerio 34 - 40126 Bologna - fax 051293322

e-mail: linea_universitaria@zanichelli.it - sito web: www.zanichelli.it

Prima di effettuare una segnalazione è possibile verificare se questa sia già stata inviata in precedenza, identificando il libro interessato all'interno del nostro catalogo online per l'Università.

Per comunicazioni di tipo commerciale: universita@zanichelli.it

Stampa: Grafica Ragno

Via Lombardia 25, 40064 Tolara di Sotto, Ozzano Emilia (Bologna)

per conto di Zanichelli editore S.p.A.

Via Innerio 34, 40126 Bologna

Indice generale

Prefazione	IX	Misura delle prestazioni delle istruzioni	30
		Equazione classica di misura delle prestazioni	31
1 Il calcolatore: astrazioni e tecnologia	1	1.7 La barriera dell'energia	35
1.1 Introduzione	1	1.8 Metamorfosi delle architetture: il passaggio dai sistemi uniprocessore ai sistemi multiprocessore	37
Tipi di calcolatori e loro caratteristiche	3	1.9 Un caso reale: la valutazione del Core i7 Intel	41
Benvenuti nell'era post-PC	5	Benchmark SPEC per la CPU	41
Che cosa si può imparare da questo libro	6	Benchmark SPEC sull'assorbimento di potenza	42
1.2 Sette grandi idee sull'architettura dei calcolatori	8	1.10 Come andare più veloci: la moltiplicazione di matrici in Python	43
Utilizzo delle astrazioni per semplificare il progetto	8	1.11 Errori e trabocchetti	45
Rendere veloci le situazioni più comuni	8	1.12 Note conclusive	48
Prestazioni attraverso il parallelismo	9	Organizzazione del testo	49
Prestazioni attraverso la pipeline	9	1.13 Inquadramento storico e approfondimenti	49
Prestazioni attraverso la predizione	9	1.14 Aiuto allo studio	49
Gerarchia delle memorie	9	Risposte ai quesiti	51
Affidabilità e ridondanza	10	1.15 Esercizi	53
1.3 Che cosa c'è dietro un programma	10	Risposte alle domande di autovalutazione	56
Da un linguaggio ad alto livello al linguaggio dell'hardware	11	2 Le istruzioni: il linguaggio dei calcolatori	57
1.4 Componenti di un calcolatore	13	2.1 Introduzione	57
Attraverso lo specchio	15	2.2 Operazioni svolte dall'hardware del calcolatore	60
Touchscreen	16	2.3 Operandi dell'hardware del calcolatore	62
Dentro la scatola	16	Operandi allocati in memoria	64
Un posto sicuro per i dati	19	Operandi immediati o costanti	67
Comunicare con gli altri calcolatori	19		
1.5 Tecnologie per la produzione di processori e memorie	21		
1.6 Prestazioni	24		
Definizione delle prestazioni	25		
Misurare le prestazioni	27		
Prestazioni della CPU	29		

2.4 Numeri con e senza segno	69	Codifica delle istruzioni x86	146
Riepilogo	74	Conclusioni sull'x86	148
2.5 Rappresentazione delle istruzioni nel calcolatore	75	2.20 Un caso reale: le altre istruzioni dell'architettura RISC-V	148
Campi delle istruzioni RISC-V	77	2.21 Come andare più veloci: la moltiplicazione di matrici in C	149
2.6 Operazioni logiche	82	2.22 Errori e trabocchetti	151
2.7 Istruzioni per prendere decisioni	84	2.23 Note conclusive	152
Cicli	86	2.24 Inquadramento storico e approfondimenti	155
Scorciatoie per il controllo dei confini di vettori e matrici	88	2.25 Aiuto allo studio	155
Costrutto <i>case/switch</i>	89	Risposte ai quesiti	156
2.8 Supporto hardware alle procedure	90	2.26 Esercizi	157
Utilizzo di più registri	91	Risposte alle domande di autovalutazione	162
Procedure annidate	93		
Allocazione dello spazio nello stack per nuovi dati	96		
Allocazione dello spazio nello heap per nuovi dati	96		
2.9 Comunicare con le persone	99		
Caratteri e stringhe in Java	101		
2.10 Indirizzamento RISC-V di un campo immediato e di un indirizzo ampio	104	3 L'aritmetica dei calcolatori	163
Operandi immediati ampi	104	3.1 Introduzione	163
Indirizzamento nei salti	105	3.2 Somme e sottrazioni	163
Riassunto delle modalità di indirizzamento del RISC-V	108	Riepilogo	166
Come decodificare il linguaggio macchina	108	3.3 Moltiplicazione	167
2.11 Parallelismo e istruzioni: la sincronizzazione	111	Versione sequenziale dell'algoritmo della moltiplicazione e sua implementazione hardware	168
2.12 Tradurre e avviare un programma	114	Moltiplicazione di numeri dotati di segno	170
Compilatore	114	Moltiplicazione veloce	171
Assemblatore	114	Moltiplicazione nel RISC-V	171
Linker	117	Riepilogo	172
Loader	120	3.4 Divisione	172
Librerie a caricamento dinamico	120	Un algoritmo della divisione e l'hardware che lo implementa	174
Come avviare un programma Java	122	Divisione di numeri dotati di segno	176
2.13 Un esempio riassuntivo in linguaggio C	123	Una divisione più veloce	177
Procedura <i>scambia</i>	124	Divisione nel RISC-V	177
Procedura <i>ordina</i>	125	Riepilogo	177
2.14 Confronto tra vettori e puntatori	130	3.5 Numeri in virgola mobile	180
Versione della procedura <i>azzerà</i> che utilizza vettore e indice	131	Rappresentazione in virgola mobile	181
Versione della procedura <i>azzerà</i> che utilizza i puntatori	132	Eccezioni e Interrupt	182
Confronto tra le due versioni di <i>azzerà</i>	133	Standard IEE 754 per la virgola mobile	182
2.15 Approfondimento: compilazione del C e interpretazione di Java	134	Addizione in virgola mobile	186
2.16 Un caso reale: le istruzioni dell'architettura MIPS	134	Moltiplicazione in virgola mobile	190
2.17 Un caso reale: le istruzioni dell'architettura Armv7 (32 bit)	135	Istruzioni in virgola mobile nel RISC-V	193
Modalità di indirizzamento	137	Aritmetica accurata	200
Comparazioni e salti condizionati	137	Riepilogo	202
Caratteristiche uniche dell'ARM	138	3.6 Parallelismo e aritmetica dei calcolatori: parallelismo a livello di parola	204
2.18 Un caso reale: le istruzioni dell'architettura ARMv8 (64 bit)	139	3.7 Un caso reale: le estensioni SIMD per lo streaming e le estensioni avanzate dell'x86 per il calcolo vettoriale	204
2.19 Un caso reale: le istruzioni dell'architettura x86	140	3.8 Come andare più veloci: il parallelismo a livello di parola applicato alla moltiplicazione di matrici	206
Evoluzione dell'Intel x86	140	3.9 Errori e trabocchetti	208
Registri e modalità di indirizzamento dell'x86	142	3.10 Note conclusive	211
Operazioni su numeri interi dell'x86	144	3.11 Inquadramento storico e approfondimenti	212

3.12 Aiuto allo studio	212	4.14 Argomenti avanzati: un'introduzione alla progettazione digitale con un linguaggio di progettazione dell'hardware e un modello di pipeline e approfondimenti sulla pipeline	323
Risposte ai quesiti	214		
3.13 Esercizi	215	4.15 Errori e trabocchetti	323
Risposte alle domande di autovalutazione	219	4.16 Note conclusive	324
4 Il processore	220	4.17 Inquadramento storico e approfondimenti	325
4.1 Introduzione	220	4.18 Aiuto allo studio	325
Un'implementazione di base del RISC-V	221	Risposte ai quesiti	326
4.2 Convenzioni del progetto logico	224	4.19 Esercizi	327
Metodologia di temporizzazione	225	Risposte alle domande di autovalutazione	336
4.3 Realizzazione di un'unità di elaborazione	227	5 Grande e veloce: la gerarchia delle memorie	338
Progettazione di un'unità di elaborazione unificata	232	5.1 Introduzione	338
4.4 Uno schema semplice di implementazione	234	5.2 Tecnologie delle memorie	343
Unità di controllo della ALU	235	Tecnologia SRAM	343
Progettazione dell'unità di controllo principale	235	Tecnologia DRAM	344
Funzionamento dell'unità di elaborazione	242	Memorie flash	346
Completamento dell'unità di controllo	245	Memorie a disco	346
Perché oggi non si utilizzano più implementazioni a singolo ciclo?	245	5.3 Principi base delle memorie cache	348
4.5 Un'implementazione multi-ciclo	246	Accesso alla cache	351
4.6 Introduzione alla pipeline	246	Gestione delle miss della cache	357
Progettazione dell'insieme di istruzioni per architetture dotate di pipeline	251	Gestione delle scritture	358
Hazard nelle pipeline	251	Un esempio di memoria cache: il processore FastMATH Intrinsity	360
Hazard sul controllo	255	Riepilogo	362
Riepilogo sulla pipeline	258	5.4 Come misurare e migliorare le prestazioni di una cache	362
4.7 Unità di elaborazione con pipeline e unità di controllo associata	260	Riduzione delle miss di una cache utilizzando un posizionamento più flessibile dei blocchi	366
Rappresentazione grafica delle pipeline	270	Come trovare un blocco nella cache	371
Unità di controllo della pipeline	273	Come scegliere il blocco da sostituire	372
4.8 Hazard sui dati: propagazione o stallo	276	Ridurre la penalità di miss utilizzando una cache multilivello	373
Hazard sui dati e stalli	284	Ottimizzazione software mediante elaborazione a blocchi	375
4.9 Hazard sul controllo	287	Riepilogo	380
Ipotizzare che il salto condizionato non venga preso	288	5.5 Affidabilità delle gerarchie delle memorie	380
Ridurre i ritardi associati ai salti condizionati	288	Definizione di malfunzionamento	381
Predizione dinamica dei salti	291	Codice di Hamming per la correzione di errori singoli e identificazione di errori doppi (SEC/DED)	382
Riepilogo sulla pipeline	293	5.6 Macchine virtuali	386
4.10 Le eccezioni	294	Requisiti del monitor di una macchina virtuale	388
Gestione delle eccezioni nelle architetture RISC-V	295	Mancanza di supporto alle macchine virtuali da parte dell'architettura dell'insieme di istruzioni	389
Eccezioni e loro gestione nella pipeline	296	Protezione e architettura dell'insieme delle istruzioni	389
4.11 Parallelismo a livello di istruzioni	300	5.7 Memoria virtuale	390
Concetto di speculazione	301	Come individuare la posizione di una pagina e come ritrovarla	394
Parallelizzazione statica dell'esecuzione	302	Page fault	395
Processori dotati di parallelizzazione dinamica dell'esecuzione	306	Memoria virtuale per un insieme ampio di indirizzi virtuali	398
Efficienza energetica e pipeline avanzate	312	Che cosa succede in scrittura?	400
4.12 La pipeline del Core i7 Intel e del Cortex-A53 ARM	313	Come rendere più veloce la traduzione degli indirizzi: il TLB	400
Cortex-A53 ARM	313	TLB del processore FastMATH Intrinsity	402
Le prestazioni della pipeline dell'A53	315		
Core i7 6700 di Intel	316		
Prestazioni del Core i7	319		
4.13 Come andare più veloci: parallelismo a livello di istruzioni e moltiplicazione di matrici	321		

Integrazione della memoria virtuale, dei TLB e delle cache	405	6.4 Multithreading hardware	472
Meccanismi di protezione basati sulla memoria virtuale	406	6.5 I multicore e gli altri multiprocessori a memoria condivisa	476
Gestione delle miss del TLB e dei page fault	408	6.6 Introduzione alle GPU	479
Riepilogo	411	Introduzione alle architetture GPU di NVIDIA	481
5.8 Schema comune per le gerarchie delle memorie	413	Strutture di memoria delle GPU NVIDIA	483
Domanda 1: dove può essere posizionato un blocco?	413	La prospettiva delle GPU	484
Domanda 2: come si individua un blocco?	414	6.7 Architetture specifiche di dominio	486
Domanda 3: quale blocco deve essere sostituito in caso di miss della cache?	415	6.8 Cluster, calcolatori per centri di calcolo e altri multiprocessori a scambio di messaggi	489
Domanda 4: come vengono gestite le scritture?	416	Calcolatori per grandi centri di calcolo	491
Le tre C: un modello intuitivo per comprendere il comportamento delle gerarchie delle memorie	417	6.9 Introduzione alle topologie delle reti di calcolatori	494
5.9 Come utilizzare una macchina a stati finiti per controllare una cache semplificata	419	Implementazione delle topologie di rete	497
Una cache semplificata	419	6.10 Come comunicare con il mondo esterno: le reti dei cluster	497
Macchine a stati finiti	420	6.11 Benchmark per i multiprocessori	497
FSM per il controllore semplificato della cache	422	Modelli delle prestazioni	500
5.10 Parallelismo e gerarchie delle memorie: coerenza delle cache	423	Modello roofline	502
Schemi di base per garantire la coerenza	425	Confronto tra due generazioni di Opteron	503
Protocolli di snooping	425	6.12 Un caso reale: il confronto tra il supercalcolatore TPuv3 di Google e il Cluster di GPU Volta di NVIDIA	507
5.11 Parallelismo e gerarchie delle memorie: i dischi RAID	427	Confronto tra addestramento e inferenza nelle DNN	508
5.12 Argomenti avanzati: come implementare i controllori delle cache	427	La rete di un supercalcolatore DSA	509
5.13 Due casi reali: la gerarchia delle memorie del Cortex-A53 ARM e del Core i7 Intel	428	Un nodo di calcolo di una DSA	509
Prestazioni della gerarchia delle memorie del Cortex-A53 e del Core i7	430	Aritmetica nelle DSA	512
5.14 Un caso reale: il resto del sistema RISC-V e le istruzioni speciali	433	Confronto tra DSA TPuv3 e GPU Volta	512
5.15 Come andare più veloci: blocchi di cache e moltiplicazione tra matrici	434	Prestazioni	513
5.16 Errori e trabocchetti	435	6.13 Come andare più veloce: processori multipli e moltiplicazione di matrici	516
5.17 Note conclusive	440	6.14 Errori e trabocchetti	518
5.18 Inquadramento storico e approfondimenti	441	6.15 Note conclusive	521
5.19 Aiuto allo studio	441	6.16 Inquadramento storico e approfondimenti	523
Risposte ai quesiti	443	Bibliografia	523
5.20 Esercizi	446	6.17 Aiuto allo studio	524
Risposte alle domande di autovalutazione	457	Risposte ai quesiti	525
6.18 Esercizi	525	Risposte alle domande di autovalutazione	532
6	458	Indice analitico	533
6.1 Introduzione	458	Manuale di riferimento RISC-V	540
6.2 Le difficoltà nel creare programmi a esecuzione parallela	460	Appendici	
6.3 SISD, MIMD, SIMD, SPMD e processori vettoriali	465	Appendice A The Basics of Logic Design	
SIMD negli x86: le estensioni multimediali	467	Appendice B Mapping Control to Hardware	
Architetture vettoriali	467	Appendice C La grafica e il calcolo con la GPU	
Confronto tra architetture vettoriali e scalari	469	Appendice D Survey of Instruction Set Architectures	
Processori vettoriali ed estensioni multimediali	470		

Prefazione

La cosa più bella che possiamo sperimentare è il mistero; esso è la fonte della vera arte e della vera scienza.

Albert Einstein, *What I Believe*, 1930

Guida al libro

Crediamo che lo studio dell'informatica e dell'ingegneria informatica debba non solo riguardare i principi su cui è fondata l'elaborazione, ma anche riflettere lo stato delle conoscenze attuali in questi campi. Crediamo anche che chi legge, qualunque sia la branca dell'informatica nella quale lavora, possa apprezzare i paradigmi secondo i quali sono organizzati i sistemi di elaborazione, perché sono essi a determinarne funzionalità e prestazioni, e a decretarne in ultima analisi il successo.

La tecnologia richiede oggi che i professionisti di tutte le branche dell'informatica conoscano sia il software sia l'hardware, la cui interazione a tutti i livelli è la chiave per capire i principi fondamentali dell'elaborazione. Inoltre, le idee che stanno alla base dell'organizzazione e della progettazione dei calcolatori valgono sia nell'ambito informatico sia in quello dell'ingegneria elettronica e sono le stesse sia che il vostro interesse principale sia il software sia che sia l'hardware. Per questo motivo, nel testo l'enfasi viene posta sulla relazione tra hardware e software e vengono approfonditi i concetti che stanno alla base dei calcolatori delle ultime generazioni.

Il passaggio recente dalle architetture uniprocessore ai multiprocessori multicore ha confermato quanto sia corretta questa prospettiva, che abbiamo adottato già nella prima edizione di questo libro. Fino a poco tempo fa i programmatori potevano fare affidamento sul lavoro dei progettisti delle architetture e dei compilatori e dei produttori dei chip, per rendere più veloci o più efficienti dal punto di vista energetico i propri programmi senza il bisogno di apportare alcuna modifica. Questa era è finita: affinché un programma possa essere eseguito più velocemente, deve diventare un programma parallelo. Anche se l'obiettivo di molti ricercatori è fare sì che i programmatori non si accorgano della natura parallela dell'hardware per il quale scrivono i loro programmi, ci vorranno molti anni prima che ciò divenga effettivamente

possibile. Crediamo che nel prossimo decennio la maggior parte dei programmatori dovrà capire a fondo il legame tra hardware e software perché i programmi vengano eseguiti in modo efficiente sui calcolatori paralleli.

Questo libro è rivolto principalmente a coloro che, pur avendo scarse conoscenze del linguaggio assembler e della logica digitale, vogliono capire i concetti di base dell'organizzazione degli elaboratori, e che sono interessati a capire il modo in cui si progetta un elaboratore, come funziona e perché si ottengono determinate prestazioni.

L'altro testo degli stessi autori sulle architetture

Qualcuno conoscerà il testo degli stessi autori *Computer Architecture: A Quantitative Approach*, chiamato anche "Hennessy Patterson", mentre questo libro viene solitamente chiamato "Patterson Hennessy". Avevamo scritto quel libro con l'obiettivo di descrivere i principi su cui sono basate le architetture degli elaboratori utilizzando un robusto approccio ingegneristico per illustrare tutti i compromessi che si rendono necessari tra costi e prestazioni. In quel libro avevamo utilizzato un metodo basato su esempi e misure, effettuate su architetture disponibili sul mercato, per consentire di fare esperienza con la progettazione di sistemi realistici: l'obiettivo era dimostrare che le architetture degli elaboratori possono essere studiate utilizzando metodologie quantitative invece di un approccio descrittivo. Quel libro era stato pensato per i professionisti coscienti che volevano approfondire nei dettagli il funzionamento dei calcolatori.

La maggior parte delle persone che leggeranno questo libro non ha intenzione di diventare un progettista di calcolatori. Tuttavia, le prestazioni e l'efficienza energetica dei sistemi software prodotti nel prossimo futuro saranno enormemente influenzate da quanto bene i progettisti software avranno capito le strutture hardware di base che sono presenti in un calcolatore. Perciò, i progettisti dei compilatori e dei sistemi operativi, così come i programmatori di database e della maggior parte delle applicazioni, hanno bisogno di conoscere bene i principi fondamentali in base ai quali funziona un calcolatore, presentati in questo libro. Analogamente, i progettisti hardware devono capire a fondo come i loro progetti andranno a influenzare le applicazioni software.

Ci siamo quindi resi conto che questo libro doveva essere molto di più di una semplice estensione del materiale contenuto nell'altro testo, per cui ne abbiamo riesaminato a fondo il contenuto e lo abbiamo modificato adattandolo a chi leggerà questo libro. Il risultato ha avuto così tanto successo che abbiamo eliminato tutto il materiale introduttivo dalle versioni successive di *Computer Architecture* e, quindi, ora la sovrapposizione dei contenuti tra i due libri è minima.

Perché un'edizione RISC-V?

La scelta dell'architettura dell'insieme di istruzioni è chiaramente critica per un libro di testo sulle architetture degli elaboratori. Non volevamo un insieme di istruzioni che richiedesse la descrizione di caratteristiche non fondamentali e barocche per chi si avvicina per la prima volta alle architetture, per quanto l'insieme di istruzioni sia popolare. Idealmente, il primo insieme di istruzioni dovrebbe fungere da modello, più o meno come il primo amore: sorprendentemente si ricorderanno entrambi con passione.

Dato che ci sono così tante possibili scelte oggi, per la prima edizione del nostro testo *Computer Architecture: A Quantitative Approach* abbiamo inventato il nostro personale insieme di istruzioni in stile RISC. Data la popolarità crescente, l'eleganza e la semplicità dell'insieme delle istruzioni MIPS siamo passati a utilizzare i processori MIPS per la prima edizione di questo libro.

di testo e per le edizioni successive dell'altro libro. Il MIPS è stato di grande utilità per noi, i nostri lettori e le nostre lettrici.

Sono trascorsi molti anni da quando siamo passati al MIPS, e anche se miliardi di chip contenenti processori MIPS vengono ancora prodotti, questi si trovano tipicamente inseriti (*embedded*) in dispositivi dove l'insieme delle istruzioni è praticamente invisibile. E, quindi, da un po' di tempo è diventato difficile trovare un calcolatore reale sul quale si possa scaricare un programma MIPS ed eseguirlo.

La buona notizia è che un insieme di istruzioni pubblico che aderisce da vicino ai principi RISC ha fatto il suo debutto e sta rapidamente guadagnando seguaci. Il RISC-V, che è stato sviluppato inizialmente all'Università di Berkeley, non solo ripulisce le stranezze dell'insieme delle istruzioni MIPS, ma offre un semplice, elegante e moderno esempio di quello a cui dovrebbe assomigliare un insieme di istruzioni nel 2020. Dato che non è un'architettura proprietaria, esistono anche simulatori, compilatori e debugger RISC-V "open source" facilmente reperibili e sono disponibili persino implementazioni RISC-V "open source" scritte nei linguaggi di descrizione dell'hardware. Inoltre, il 2020 ha visto l'introduzione di piattaforme hardware basate sul RISC-V che sono equivalenti al Raspberry Pi, cosa che non è successa per i MIPS. Chi studia beneficerà non solo dello studio di queste architetture RISC-V, ma saranno anche in grado di modificarle percorrendo il processo di implementazione per capire l'impatto delle modifiche che propongono sulle prestazioni, sulla dimensione del chip e sull'energia assorbita.

Questa è un'opportunità eccitante sia per l'industria dei calcolatori sia per la didattica, e per questo quando è stato scritto questo libro più di 300 società hanno aderito alla fondazione RISC-V. Questo elenco di sponsor comprende praticamente tutti i maggiori produttori tranne ARM e Intel, e comprende Alibaba, Amazon, AMD, Google, Hewlett-Packard Enterprise, IBM, Microsoft, NVIDIA, Qualcomm, Samsung e Western Digital.

È per questi motivi che abbiamo scritto l'edizione RISC-V di questo libro, e siamo passati al RISC-V anche per il volume *Computer Architecture: A Quantitative Approach*.

In questa edizione siamo passati dalla versione RISC-V a 64 bit, RV64, a quella a 32 bit, RV32: i docenti hanno trovato che la maggiore complessità dell'insieme delle istruzioni a 64 bit rendesse più difficile agli studenti la comprensione degli argomenti. Nella versione RV32, l'insieme delle istruzioni core si riduce a 10: vengono eliminate le istruzioni ld, sd, lwu, subwu, addwi, sllw, srlw, sllwiw, sllwiw, srlw e gli studenti non devono capire le operazioni che vengono eseguite sui 32 bit meno significativi dei registri a 64 bit, senza contare che nel libro possiamo largamente ignorare le parole doppie e utilizzare soltanto le parole singole. In questa edizione, inoltre, nascondiamo i formati SB e UJ, che a prima vista appaiono strani, fino al Capitolo 4. Spieghiamo il risparmio in termini di hardware dello strano ordinamento dei bit del campo immediato nei formati SB e UJ successivamente dato che quel capitolo è dove mostriamo l'hardware del cammino di elaborazione. Così come abbiamo fatto per la sesta edizione MIPS, abbiamo aggiunto un paragrafo online che mostra un'implementazione multi-ciclo della CPU, modificata per essere adattata al RISC-V. Alcuni docenti preferiscono esaminare l'implementazione multi-ciclo dopo l'implementazione a singolo ciclo prima di introdurre la pipeline.

Le uniche modifiche dell'edizione RISC-V rispetto all'edizione MIPS sono quelle associate alla modifica dell'insieme delle istruzioni, che riguarda soprattutto il Capitolo 2, il Capitolo 3, la parte sulla memoria virtuale nel Capitolo 5, e i brevi esempi VMIPS del Capitolo 6. Nel Capitolo 4, siamo passati alle istruzioni RISC-V, abbiamo modificato diverse figure, e abbiamo aggiunto alcune sezioni di *Approfondimento*, ma le modifiche sono state più semplici di quello

che temevamo. Il Capitolo 1 e la maggior parte delle Appendici sono rimaste praticamente invariate. L'estesa documentazione disponibile su web combinata con la complessità del RISC-V hanno reso difficile ottenere un'Appendice A (Gli assembleri, i linker e il simulatore SPIM) come quella della sesta edizione MIPS. In compenso, nei Capitoli 2, 3 e 5 è riportato uno stupefacente riassunto delle centinaia di istruzioni RISC-V che sono al di fuori delle istruzioni core RISC-V che abbiamo descritto in dettaglio nel resto del libro.

Le novità di questa seconda edizione

Ci sono stati sicuramente più cambiamenti nel mondo della tecnologia e del mercato dei calcolatori dalla pubblicazione della quinta edizione inglese di quanti ce ne siano stati tra la prima e la quarta edizione.

- *Il rallentamento della legge di Moore.* Dopo 50 anni nei quali il numero di transistor per chip è raddoppiato ogni due anni, oggi la predizione di Gordon Moore non è più valida. La tecnologia dei semiconduttori migliorerà ancora, ma più lentamente e in modo meno predicibile che in passato.
- *La crescita delle Architetture Specifiche di Dominio (DSA).* In parte a causa del rallentamento della legge di Moore e in parte perché il modello di crescita di Dennard non è più valido, i processori a uso generale stanno aumentando le loro prestazioni ogni anno solamente di pochi punti percentuali. Inoltre, la legge di Amdahl limita i benefici effettivi dell'aumento del numero dei processori per chip. Già nel 2020, era opinione diffusa che il filone di sviluppo fossero le DSA. Una DSA non cerca di eseguire qualsiasi programma al meglio come i processori di utilizzo generale, ma è focalizzata sull'eseguire i programmi di un certo dominio molto meglio delle CPU convenzionali.
- *Micro-architetture come superfici sicure contro gli attacchi.* Lo Spectre ha dimostrato che l'esecuzione speculativa fuori-ordine il multi-threading hardware rendono possibili gli attacchi basati sulla temporizzazione dai canali adiacenti. Inoltre, questi attacchi non sono resi possibili da banchi che possono essere corretti, ma rappresentano una sfida fondamentale per i progettisti di processori di questo tipo.
- *Insiemi di istruzioni e implementazione open source.* Le opportunità e l'impatto del software open source è arrivato anche alle Architetture dei Calcolatori. Architetture degli insiemi di istruzioni "open" quali il RISC-V consentono alle organizzazioni di costruire i loro processori, senza dover prima negoziare una licenza; questo ha consentito di sviluppare implementazioni "open source" che vengono condivise per scaricare liberamente e utilizzare queste versioni al pari delle implementazioni proprietarie del RISC-V. Il software sorgente e l'hardware "open source" costituiscono una manna dal cielo per la ricerca accademica e la didattica poiché consentono a chi studia di vedere e migliorare la robustezza della tecnologia utilizzata dall'industria.
- *Ri-verticalizzazione dell'industria della tecnologia informatica.* Il calcolo sul cloud ha fatto sì che non ci siano più di una mezza decina di società nel mondo che forniscono infrastrutture di calcolo per tutti. In modo simile all'IBM negli anni '60 e '70, queste società determinano sia lo stack software sia l'hardware che viene distribuito. Questi cambiamenti hanno portato alcune di queste società a sviluppare i loro chip di DSA e di RISC-V da mettere in campo nei loro cloud.

La seconda edizione inglese di questo testo (edizione RISC-V) riflette questi cambiamenti recenti; tutti gli esempi e le figure sono stati aggiornati

rispondendo alle richieste dei docenti. Inoltre, per migliorare lo studio, è stato aggiunto in ogni capitolo un Paragrafo ispirato ai libri di testo che ho utilizzato per aiutare i miei nipoti nello studio della matematica.

In questa edizione ogni capitolo ha il suo Paragrafo: *Come andare più veloce*. Iniziamo nel Capitolo 1 mostrando una versione della moltiplicazione di matrici in Python che viene eseguita così lentamente da motivare lo studio del linguaggio C per riscrivere il codice nel Capitolo 2. Nei capitoli successivi la moltiplicazione di matrici viene accelerata sfruttando il parallelismo a livello di dati, di istruzioni e di thread, e adattando gli accessi alla memoria alla gerarchia di memoria offerta dai server moderni. Questi calcolatori consentono operazioni SIMD su 512 bit, esecuzione speculativa fuori ordine, tre livelli di cache e 48 core. Tutte assieme, le quattro ottimizzazioni aggiungono solamente 21 linee di codice eppure consentono di aumentare la velocità della moltiplicazione di due matrici di quasi 50 000 volte, riducendo il tempo di esecuzione dalle 6 ore della versione in Python a meno di 1 secondo in C ottimizzato. Se fossi ancora uno studente, questo esempio pratico mi motiverebbe a utilizzare il C e a studiare i concetti hardware sottostanti presentati in questo libro.

- A partire da questa edizione, ogni capitolo contiene un Paragrafo di *Aiuto allo studio* nel quale vengono posti dei quesiti sfidanti e vengono fornite le risposte che vi aiuteranno a valutare se avete assimilato il materiale trattato nel capitolo.
- Oltre che spiegare che la legge di Moore e il modello di crescita di Dennard non sono più validi, abbiamo tolto l'enfasi sulla legge di Moore come motore dello sviluppo delle architetture, enfasi che era particolarmente accentuata nella quinta edizione inglese.
- Nel Capitolo 2 è stato aggiunto altro materiale per enfatizzare che i dati binari non hanno un significato intrinseco – è il programma a determinare il tipo di dato; questo non è un concetto facile da capire per chi inizia a studiare le architetture.
- Il Capitolo 2 contiene anche una breve descrizione dell'insieme delle istruzioni MIPS, che viene confrontato sia con l'insieme RISC-V sia con gli insiemi ARMv7, ARMv8 e x86 (è disponibile anche una versione parallela di questo testo basata sul MIPS invece che sul RISC-V, e abbiamo aggiornato anche quella versione con altre modifiche).
- I benchmark utilizzati nel Capitolo 2 sono stati aggiornati sostituendo lo SPEC2006 con lo SPEC2017.
- A richiesta dei docenti, abbiamo ripristinato la descrizione dell'implementazione multi-ciclo dei MIPS. Questa si può trovare come Paragrafo on-line del Capitolo 4, ed è stata inserita tra l'implementazione a singolo ciclo e quella in pipeline. Alcuni docenti ritengono che questi tre passaggi offrano un avvicinamento più facile alla pipeline.
- Le sezioni *Quadro d'insieme* dei Capitoli 4 e 5 sono state aggiornate alla recente mirco-architettura dell'A-53 ARM e dell'i7 6700 Skylake l'Intel.
- Il Paragrafo *Errori e Trabocchetti* dei Capitoli 5 e 6 descrive anche i trabocchetti sugli attacchi hardware alla sicurezza dello "Row Hammer" e dello "Spectre".
- Nel Capitolo 6 è contenuto un nuovo paragrafo che introduce le DSA utilizzando le TPU (Tensor Processing Unit) versione 1, di Google. La sezione "Quadro d'Insieme" del Capitolo 6 è stata aggiornata con il confronto tra il super-calcolatore con architettura DSA, TPUv3 di Google e un cluster di GPU Volta di NVIDIA.

Infine, abbiamo aggiornato tutti gli esercizi riportati nel libro.

Anche se alcuni elementi sono cambiati, abbiamo conservato le caratteristiche delle edizioni precedenti rivelatesi più utili: abbiamo mantenuto la

definizione delle parole chiave a margine del testo la prima volta che compaiono, le sezioni *Capire le prestazioni dei programmi* (dedicate alle prestazioni e a come migliorarle), le sezioni *Interfaccia hardware/software* (sui compromessi da adottare a livello di questa interfaccia), le sezioni *Quadro d'insieme* (che riepilogano i concetti principali espressi nel testo) e le sezioni *Autovalutazione*, che aiutano chi legge a valutare la comprensione degli argomenti trattati (con le relative risposte esatte alla fine di ogni capitolo). Anche questa edizione contiene nell'ultima pagina una scheda tecnica riassuntiva del RISC-V. Il contenuto della scheda è stato aggiornato e costituisce un riferimento immediato per coloro che scrivono programmi nell'assembler del RISC-V.

Le risorse multimediali

online.universita.zanichelli.it/patterson-risc2e

A questo indirizzo sono disponibili le risorse multimediali di complemento al libro. Per accedere alle risorse protette è necessario registrarsi su **my.zanichelli.it** inserendo la chiave di attivazione personale contenuta nel libro.

Libro con ebook

Chi acquista il libro nuovo può accedere gratuitamente all'ebook, seguendo le istruzioni presenti nel sito. L'ebook si legge con l'applicazione *Booktab*, che si scarica gratis da *App Store* (sistemi operativi *Apple*) o da *Google Play* (sistemi operativi *Android*). L'accesso all'ebook e alle risorse digitali protette è personale, non condivisibile e non cedibile, né autonomamente né con la cessione del libro cartaceo.

Considerazioni conclusive

Se leggerete il successivo paragrafo di ringraziamenti, vi renderete conto che abbiamo corretto moltissimi errori. E quando un libro passa attraverso diverse ristampe, si ha la possibilità di correggere un numero ancora maggiore di errori. Se doveste trovare altri errori, per cortesia, contattate direttamente l'editore attraverso la posta elettronica o la posta ordinaria, utilizzando gli indirizzi riportati nella pagina del copyright.












































































Questa edizione è la quarta dopo l'interruzione della lunga collaborazione tra Hennessy e Patterson, iniziata nel 1989. Gli impegni richiesti per dirigere una delle più importanti università del mondo hanno tolto a Hennessy il tempo necessario per lavorare alle nuove edizioni: il suo coautore, Patterson, si è sentito ancora una volta come un acrobata che si esibisce senza rete. Per questo motivo, le persone elencate nel paragrafo dei ringraziamenti e i colleghi di Berkeley hanno avuto un ruolo ancora maggiore nel dare forma al contenuto di questo libro. Ciò nonostante, uno solo è l'autore responsabile del nuovo materiale che vi apprestate a leggere.

Ringraziamenti per la seconda edizione

Siamo stati davvero fortunati a ricevere diversi contributi da molti lettori, redattori e ricercatori: ciascuno di loro ha contribuito a rendere migliore questo libro.

Siamo grati per l'assistenza di Khaled Benkrid e ai suoi colleghi di ARM Ltd. che hanno revisionato attentamente il materiale relativo agli ARM e hanno fornito interessanti suggerimenti.

Un particolare ringraziamento va a Rimas Avizensis dell'Università di Berkeley, che ha sviluppato le diverse versioni della procedura di

Capitolo o Appendice	Paragrafi	Focalizzazione sul software	Focalizzazione sull'hardware
1. Il calcolatore: astrazioni e tecnologia	Da 1.1 a 1.12		
	 1.13 (Storia)		
2. Le istruzioni: il linguaggio dei calcolatori	Da 2.1 a 2.14		
	 2.15 (Compilatori e Java)		
	Da 2.16 a 2.22		
	 2.23 (Storia)		
D. Architettura dell'Insieme delle Istruzioni RISC-V	 Da D.1 a D.16		
3. L'aritmetica dei calcolatori	Da 3.1 a 3.5		
	Da 3.6 a 3.8 (Storia)		
	Da 3.9 a 3.10 (Errori e trabocchetti)		
	 3.11 (Storia)		
A. Le basi della progettazione dei circuiti logici	 Da A.1 ad A.13		
4. Il processore	4.1 (Panoramica)		
	4.2 (Convenzioni logiche)		
	Da 4.3 a 4.4 (Un'implementazione semplice)		
	 4.5 (Implementazione multi-ciclo)		
	4.6 (Introduzione alla pipeline)		
	4.7 (Unità di elaborazione con pipeline)		
	Da 4.8 a 4.10 (Criticità, Eccezioni)		
	Da 4.11 a 4.13 (Parallelismo, Esempi reali)		
	 4.14 (Unità di controllo della pipeline in Verilog)		
	Da 4.15 a 4.16 (Errori e trabocchetti)		
	 4.17 (Storia)		
B. Mappatura delle funzioni di controllo sull'hardware	 Da B.1 a B.6		
5. Grande e veloce: la gerarchia delle memorie	Da 5.1 a 5.10		
	 5.11 (Dischi RAID)		
	 5.12 (Controllori Verilog delle cache)		
	Da 5.13 a 5.16		
	 5.17 (Storia)		
6. Processi paralleli: dai client al cloud	Da 6.1 a 6.9		
	 6.10 (Cluster)		
	Da 6.11 a 6.15		
	 6.16 (Storia)		
C. Le GPU	 Da C.1 a C.11		

Leggere con attenzione 

Leggere se si ha tempo

Riferimento Riguardare o rileggere Leggere per cultura personale 

1

Il calcolatore: astrazioni e tecnologia

La civiltà progredisce estendendo il numero delle operazioni complesse che possiamo effettuare senza dover pensare ad esse.

Alfred North Whitehead, *An Introduction to Mathematics*, 1911

1.1 | Introduzione

Benvenuti alla lettura di questo libro! È per noi un piacere mostrarvi l'eccitante mondo dei calcolatori. Questo, infatti, è tutt'altro che un mondo arido e scoraggiante, dove il progresso procede con estrema lentezza e le nuove idee finiscono per atrofizzarsi perché vengono trascurate. Tutt'altro! I calcolatori sono invece il prodotto principale di una tecnologia straordinariamente vitale, quella dell'informazione, che contribuisce per circa il 10% al prodotto interno lordo degli Stati Uniti d'America, la cui economia stessa è diventata in parte dipendente dal miglioramento così incredibilmente rapido della tecnologia dell'informazione.

In questo particolare settore industriale, l'innovazione viene introdotta con una frequenza estremamente elevata. Negli ultimi 40 anni sono stati proposti diversi nuovi calcolatori, la cui introduzione sembrava dovesse rivoluzionare l'intera industria degli elaboratori; invece queste rivoluzioni hanno avuto vita breve, ma solo perché altri calcolatori, più performanti, hanno sostituito rapidamente i calcolatori più lenti.

Questa corsa all'innovazione ha portato a progressi mai visti prima, già a partire dalla nascita del primo calcolatore elettronico, alla fine degli anni '40 del secolo scorso. Se l'industria dei trasporti avesse tenuto il passo di quella dei calcolatori, oggi si potrebbe andare da New York a Londra in circa un secondo, spendendo solo qualche centesimo di dollaro. Femandoci un momento a riflettere su come ciò modificherebbe la nostra società – si potrebbe

Autovalutazione

Le sezioni denominate *Autovalutazione* sono pensate per aiutare il lettore a valutare se abbia compreso i principali concetti introdotti nel capitolo e per mostrarne a fondo le implicazioni. Alcune domande di queste sezioni sono molto semplici, mentre altre hanno lo scopo di aprire una discussione con altre persone. Potete trovare le risposte alle singole domande alla fine del capitolo. Le domande di autovalutazione sono inserite al termine dei paragrafi, in modo da poterle evitare se siete sicuri di aver appreso quanto avete letto.

1. Il numero di calcolatori embedded venduti ogni anno supera largamente quello dei PC e persino quello dei calcolatori post-PC. Siete in grado di confermare o confutare questa affermazione basandovi sulla vostra esperienza? Provate a contare il numero di processori embedded presenti nella vostra abitazione e confrontatelo con il numero di calcolatori tradizionali. Qual è il risultato di questo confronto?
2. Come accennato precedentemente, sia il software sia l'hardware influenzano le prestazioni di un programma. Provate a pensare ad alcuni esempi pratici nei quali ognuno dei seguenti elementi può rappresentare un "collo di bottiglia" per le prestazioni:
 - algoritmo implementato;
 - linguaggio di programmazione o compilatore;
 - sistema operativo;
 - processore;
 - sistema di I/O e periferiche.

1.2 Sette grandi idee sull'architettura dei calcolatori

Presentiamo ora sette grandi idee dei progettisti dei calcolatori degli ultimi 60 anni. Queste idee sono state così innovative da durare a lungo dopo la loro implementazione nelle prime architetture: i progettisti più giovani hanno ripreso queste idee nella progettazione delle architetture più recenti. Queste idee sono dei principi ricorrenti che compariranno spesso in questo capitolo e nei capitoli successivi; per identificarle introduciamo qui i simboli che le rappresentano e i termini a esse associati. Utilizzeremo questi simboli per individuare i quasi 100 paragrafi di questo libro che descrivono le caratteristiche basate su queste idee.

Utilizzo delle astrazioni per semplificare il progetto



ASTRAZIONE

Sia i progettisti dei calcolatori sia i programmatori hanno dovuto inventare delle tecniche che li rendessero più produttivi, per evitare che il tempo riservato alla progettazione aumentasse esageratamente con il crescere delle risorse rese disponibili. Una di queste tecniche è l'**astrazione**, utilizzata per rappresentare il progetto sia hardware sia software a diversi livelli di definizione: i dettagli vengono nascosti ai livelli più bassi per offrire un modello più semplice ai livelli più alti. Utilizzeremo un disegno astratto come simbolo per questa prima grande idea.

Rendere veloci le situazioni più comuni



SITUAZIONI COMUNI

Rendere veloci le **situazioni più comuni** tende a fare aumentare le prestazioni più dell'ottimizzazione delle funzionalità utilizzate raramente. Ironicamente, le situazioni più comuni sono spesso più semplici di quelle rare e quindi di

solito sono più semplici da migliorare. Questo consiglio dettato dal buon senso prevede che sappiate quali sono le situazioni più comuni, cosa che è possibile solo attraverso un'attenta sperimentazione e analisi (Paragrafo 1.6).

Utilizzeremo il simbolo di un'auto sportiva per rappresentare quest'idea, dato che i viaggi più frequenti prevedono uno o due passeggeri, è sicuramente più facile realizzare una macchina sportiva veloce che un minivan veloce!

Prestazioni attraverso il parallelismo

Sin dagli albori, i progettisti hanno realizzato calcolatori che ottengono prestazioni più elevate eseguendo le operazioni in parallelo. Vedremo molti esempi di parallelismo in questo libro. Utilizzeremo il simbolo di un aereo a reazione con più motori per rappresentare le prestazioni attraverso il **parallelismo**.



PARALLELISMO

Prestazioni attraverso la pipeline

Una particolare forma di parallelismo è così diffusa nelle architetture da meritarsi un nome proprio: **pipeline**. Per esempio, una squadra di uomini, prima di chiamare i vigili del fuoco, può cercare di spegnere il fuoco con i secchi, come avviene tipicamente nei film western quando il cattivo appicca il fuoco: gli abitanti del paese formano una catena umana per trasportare l'acqua dalla sorgente al fuoco. Infatti, in questo modo si riesce a trasportare l'acqua dalla sorgente al fuoco più velocemente che correndo avanti e indietro. Il simbolo della pipeline è una sequenza di condotti, dove ciascun condotto rappresenta uno stadio della pipeline.



PIPELINE

Prestazioni attraverso la predizione

Seguendo il detto secondo cui "è meglio chiedere perdono che chiedere permesso", un'altra grande idea è la **predizione**. In alcuni casi, è in genere più veloce tirare a indovinare e iniziare a lavorare di conseguenza, che aspettare di sapere con certezza. Questo è vero quando il meccanismo per recuperare una predizione sbagliata non è troppo costoso e la predizione è sufficientemente accurata. Utilizzeremo la sfera di cristallo come simbolo della predizione.



PREDIZIONE

Gerarchia delle memorie

I programmatori vogliono che la memoria sia di grandi dimensioni, veloce e poco costosa, dato che la sua velocità spesso determina le prestazioni, la sua capacità limita la dimensione dei problemi che possono essere risolti e il suo costo rappresenta la voce di costo maggiore di un'architettura.

I progettisti hanno scoperto che possono soddisfare queste esigenze contrapposte con la **gerarchia delle memorie**, nella quale la memoria più veloce, piccola e con il maggior costo per bit si trova in cima alla gerarchia e la memoria più lenta, più grande e con il minore costo per bit alla base. Come vedremo nel Capitolo 5, le memorie cache forniscono al programmatore l'illusione che la memoria principale sia veloce quasi quanto la memoria in cima alla gerarchia e sia economica e grande quasi quanto quella alla base della gerarchia.

Utilizzeremo come simbolo della gerarchia delle memorie un triangolo astratto. La forma indica la velocità, il costo e la dimensione: più vicino è lo strato di memoria alla cima, maggiore sarà la sua velocità e il suo costo; più larga è la base, maggiore sarà la capacità della memoria.



GERARCHIA

ESEMPIO**Confronto di frammenti di codice**

Un progettista di compilatori deve decidere quale tra due sequenze di codice implementare per un certo calcolatore. I progettisti dell'hardware gli hanno fornito queste informazioni:

	CPI per ciascun tipo di istruzione		
	A	B	C
CPI	1	2	3

Per una certa istruzione in linguaggio ad alto livello, il progettista sta considerando due sequenze di codice in linguaggio macchina che richiedono un numero diverso di istruzioni dei tre tipi:

Sequenza di istruzioni	Numero di istruzioni in linguaggio macchina per ciascun tipo di istruzione		
	A	B	C
Sequenza 1	2	1	2
Sequenza 2	4	1	1

Quale sequenza richiede l'esecuzione di un maggior numero di istruzioni? Quale verrà eseguita più velocemente? Qual è il CPI delle due sequenze?

SOLUZIONE

Nella sequenza 1 vengono eseguite: $2 + 1 + 2 = 5$ istruzioni, mentre la sequenza 2 richiede $4 + 1 + 1 = 6$ istruzioni; perciò la sequenza 1 richiede un numero minore di istruzioni.

Per calcolare il numero totale di cicli di clock per ciascuna sequenza, possiamo utilizzare l'equazione che misura il numero di cicli di clock della CPU in funzione del numero di istruzioni e del CPI:

$$\text{Cicli di clock della CPU} = \sum_{i=1}^n (\text{CPI}_i \times C_i)$$

Da questa equazione possiamo ricavare il numero totale dei cicli di clock:

$$\text{Cicli di clock della CPU}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ cicli}$$

$$\text{Cicli di clock della CPU}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ cicli}$$

Pertanto possiamo concludere che la sequenza 2 è più veloce, anche se richiede l'esecuzione di un'istruzione in più. Dato che tale sequenza richiede meno cicli di clock ma ha un'istruzione in più, deve avere un CPI più basso. Si può calcolare il valore del CPI come:

$$\text{CPI} = \frac{\text{Cicli di clock della CPU}}{\text{Numero di istruzioni}}$$

$$\text{CPI}_1 = \frac{\text{Cicli di clock della CPU}_1}{\text{Numero di istruzioni}_1} = \frac{10}{5} = 2,0$$

$$\text{CPI}_2 = \frac{\text{Cicli di clock della CPU}_2}{\text{Numero di istruzioni}_2} = \frac{9}{6} = 1,5$$

Componente delle prestazioni	Unità di misura
Tempo di esecuzione della CPU per un dato programma	Secondi per programma
Numero di istruzioni	Istruzioni eseguite per singolo programma
Cicli di clock per istruzione (CPI)	Numero medio di cicli di clock per istruzione
Durata del ciclo di clock	Secondi per ciclo di clock

Figura 1.15 I componenti di base delle prestazioni e come vengono misurati.

QUADRO D'INSIEME

La **Figura 1.15** riporta le misure di base che vengono utilizzate per i calcolatori a livelli diversi e che cosa viene misurato in ciascun caso. Potete notare come i diversi fattori si possano combinare tra loro per formare il tempo di esecuzione di un programma, che viene misurato in secondi:


$$\begin{aligned}\text{Tempo} &= \text{Secondi/Programma} = \\ &= \frac{\text{Istruzioni}}{\text{Programma}} \times \frac{\text{Cicli di clock}}{\text{Istruzione}} \times \frac{\text{Secondi}}{\text{Ciclo di clock}}\end{aligned}$$

Occorre tenere sempre presente che il tempo è l'unica misura completa e affidabile per valutare le prestazioni di un calcolatore. Per esempio, modificare l'insieme delle istruzioni per diminuire il numero di istruzioni può portare a un'architettura con periodo di clock più lungo o a un CPI più elevato, che vanifica la riduzione del numero di istruzioni. Analogamente, poiché il CPI dipende dal tipo di istruzione eseguita, il codice che esegue il minor numero di istruzioni potrebbe non essere il più veloce. ■

Come si possono determinare i valori dei fattori che compaiono nell'equazione per il calcolo delle prestazioni? Il tempo di esecuzione della CPU può essere misurato facendo eseguire il programma, mentre la durata del ciclo di clock viene normalmente riportata nella documentazione tecnica del calcolatore; il numero di istruzioni e il CPI sono invece più difficili da ottenere. Se la frequenza di clock e il tempo di esecuzione della CPU fossero noti, sarebbe sufficiente conoscere il CPI o il numero di istruzioni per conoscere anche l'altro termine.

Il numero di istruzioni può essere determinato attraverso opportuni strumenti software di profiling che osservano l'esecuzione del programma oppure attraverso un simulatore dell'architettura. Alternativamente si potrebbero utilizzare i contatori hardware, contenuti nella maggior parte dei processori, per misurare il numero di istruzioni eseguite, il CPI medio e, spesso, l'origine stessa della perdita di prestazioni. Essendo il numero di istruzioni dipendente dall'architettura ma indipendente dalla particolare implementazione, possiamo misurarlo anche senza conoscere nei dettagli l'implementazione delle istruzioni. Il CPI, viceversa, dipende da un ampio spettro di dettagli progettuali del calcolatore, tra cui la modalità di realizzazione del sottosistema di memoria e la struttura del processore (Capitoli 4 e 5); inoltre, dipende anche dalla combinazione dei tipi di istruzioni eseguite dall'applicazione specifica. Il CPI, dunque, varia da applicazione ad applicazione, come pure tra diverse implementazioni dello stesso insieme di istruzioni.

Composizione delle istruzioni (instruction mix): una misura della frequenza dinamica delle istruzioni in uno o più programmi.

L'esempio precedente evidenzia il pericolo che si corre nell'utilizzare uno solo dei fattori, per esempio il numero di istruzioni, quando si valutano le prestazioni. Nel confrontare due calcolatori è necessario tenere conto di tutte e tre le componenti, che, combinate tra loro, forniscono il tempo di esecuzione. Se alcuni di questi fattori sono identici, come lo era la frequenza di clock nell'esempio precedente, le prestazioni dei due calcolatori si possono determinare confrontando solamente gli altri fattori. Dato che il CPI dipende dalla **composizione delle istruzioni** (*instruction mix*), occorre considerare sia il numero di istruzioni sia il CPI, anche se il clock ha la stessa frequenza. Alcuni esercizi riportati alla fine di questo capitolo vi permetteranno di capire meglio come una serie di miglioramenti del calcolatore e del compilatore influisca sulla frequenza di clock, sul CPI e sul numero di istruzioni. Nel Paragrafo 1.13 , riportiamo una misura delle prestazioni che, pur essendo di uso comune, non incorpora tutti e tre questi fattori e può quindi essere fuorviante.

Capire le prestazioni dei programmi

Le prestazioni di un programma dipendono dall'algoritmo, dal linguaggio, dal compilatore, dall'architettura e dall'hardware del calcolatore. La seguente tabella riassume come questi componenti influenzino i tre fattori dell'equazione delle prestazioni.

Componente hardware o software	Che cosa influenza?	Come?
Algoritmo	Numero di istruzioni, eventualmente il CPI	L'algoritmo determina il numero di istruzioni del programma sorgente e quindi il numero di istruzioni in linguaggio macchina che vengono eseguite dal processore. L'algoritmo può anche influenzare il CPI, favorendo l'utilizzo di istruzioni più o meno veloci. Per esempio, se l'algoritmo utilizza più operazioni di divisione, tenderà ad avere un CPI più elevato
Linguaggio di programmazione	Numero di istruzioni, CPI	Il linguaggio di programmazione influenza certamente il numero di istruzioni, dal momento che i costrutti del linguaggio ad alto livello sono tradotti in istruzioni in linguaggio macchina e queste determinano il numero di istruzioni eseguite. Il linguaggio ad alto livello può anche influenzare il CPI a seconda delle sue caratteristiche; per esempio, un linguaggio con un esteso supporto per i dati astratti (per es. Java) richiederà chiamate indirette a funzione, che sono caratterizzate da un CPI più alto
Compilatore	Numero di istruzioni, CPI	L'efficienza del compilatore influenza sia il numero di istruzioni sia il numero medio di cicli per istruzione, dal momento che il compilatore traduce le istruzioni dal linguaggio sorgente ad alto livello nelle istruzioni in linguaggio macchina. Il ruolo del compilatore può essere molto complesso e può influenzare il CPI in maniera complessa
Architettura dell'insieme delle istruzioni	Numero di istruzioni, frequenza di clock, CPI	L'architettura dell'insieme di istruzioni influenza tutti e tre i fattori delle prestazioni della CPU, dato che influenza le istruzioni richieste da una data funzione, il costo in numero di cicli di ogni istruzione e la frequenza del processore

Approfondimento. Potreste aspettarvi che il minimo CPI sia 1,0; invece, come vedremo nel Capitolo 4, alcuni processori leggono ed eseguono più istruzioni in un unico ciclo di clock. A seguito di ciò, alcuni progettisti invertono il CPI e misurano l'*IPC*, o *numero di istruzioni per ciclo di clock*. Se un processore esegue in media 2 istruzioni per ciclo di clock, esso ha un IPC di 2 e quindi un CPI di 0,5.

nuova generazione della tecnologia, e la potenza è una funzione della tensione al quadrato. Perciò, dato che in media la tensione di alimentazione è stata ridotta del 15% in ogni nuova generazione, in 20 anni la tensione è scesa da 5 V a 1 V; questo spiega come mai la potenza sia cresciuta solo di 30 volte.

Il problema oggi è che diminuendo ulteriormente la tensione di alimentazione i transistor tenderebbero a disperdere troppa corrente, come un rubinetto che non può mai essere completamente chiuso. Già il 40% della potenza assorbita da un transistor è dovuto alla dispersione di corrente; se i transistor avessero perdite ancora maggiori, i processori diventerebbero ingestibili.

Per cercare di risolvere il problema dell'assorbimento di potenza, i progettisti hanno provato a inserire dispositivi più grandi per la dissipazione del calore e meccanismi di controllo per spegnere le parti dei circuiti che non vengono utilizzate in un dato ciclo di clock. Molte altre tecniche potrebbero essere esplorate per raffreddare i chip e aumentare la potenza che può essere assorbita, per esempio fino a 300 watt; queste tecniche, però, sono troppo costose non solo per i PC e i PMD, ma anche per i server.

L'assorbimento di potenza si è quindi rivelato una barriera invalicabile e i progettisti hanno cercato nuove strade per migliorare i calcolatori, sviluppando un nuovo modo di progettare i microprocessori, diverso da quello utilizzato nei primi 30 anni.

Approfondimento. Anche se un transistor CMOS assorbe energia principalmente durante la fase di commutazione, in condizioni statiche una certa quantità di energia viene comunque dissipata a causa delle correnti di dispersione che esistono anche quando il transistor è spento; queste perdite sono responsabili del 40% dell'energia totale assorbita da un server. Perciò, aumentando il numero di transistor, aumenta l'energia assorbita anche se i transistor sono sempre spenti. Diversi accorgimenti nella progettazione dei chip e ulteriori innovazioni tecnologiche sono stati introdotti per limitare la dispersione, ma rimane difficile ridurre ulteriormente la tensione di alimentazione.

Approfondimento. L'alimentazione elettrica è una sfida per i circuiti integrati per due motivi. In primo luogo, l'alimentazione deve essere distribuita ai vari chip: i moderni microprocessori utilizzano centinaia di pin solamente per l'alimentazione e la terra! Analogamente, livelli multipli di interconnessioni tra i chip vengono utilizzati solamente per distribuire l'alimentazione e la terra a porzioni del chip. In secondo luogo, la potenza elettrica viene dissipata come calore e il calore deve essere rimosso. I chip dei server possono consumare più di 100 watt e raffreddare il chip e i sistemi attorno rappresenta una delle spese maggiori nei calcolatori dei centri di calcolo (Capitolo 6).

1.8 Metamorfosi delle architetture: il passaggio dai sistemi uniprocessore ai sistemi multiprocessore

Il limite all'assorbimento di potenza ha obbligato i progettisti a cambiare radicalmente il modo di progettare i microprocessori. La **Figura 1.17** mostra il miglioramento, negli anni, del tempo di esecuzione di un programma eseguito su processore desktop: dal 2002 il miglioramento è sceso da un fattore 1,5 per anno a un fattore 1,03 per anno.

Invece di continuare a diminuire il tempo di esecuzione del singolo programma eseguito su un processore singolo, dal 2006 tutti i produttori di desktop e server hanno iniziato a distribuire microprocessori contenenti più processori sul singolo chip; in questi microprocessori il miglioramento si verifica spesso più nel throughput che nel tempo di esecuzione. Per eliminare ogni possibile confusione tra i termini processore e microprocessore, i produttori

Fino a oggi, la maggior parte del software era come uno spartito scritto per un solista; con la nuova generazione di chip, incominciamo a fare esperienza con i duetti, i quartetti e con altre piccole formazioni musicali; ma scrivere uno spartito per una grande orchestra e coro è ben altra cosa.

Brian Hayes, *Computing in Parallel Universe*, 2007.

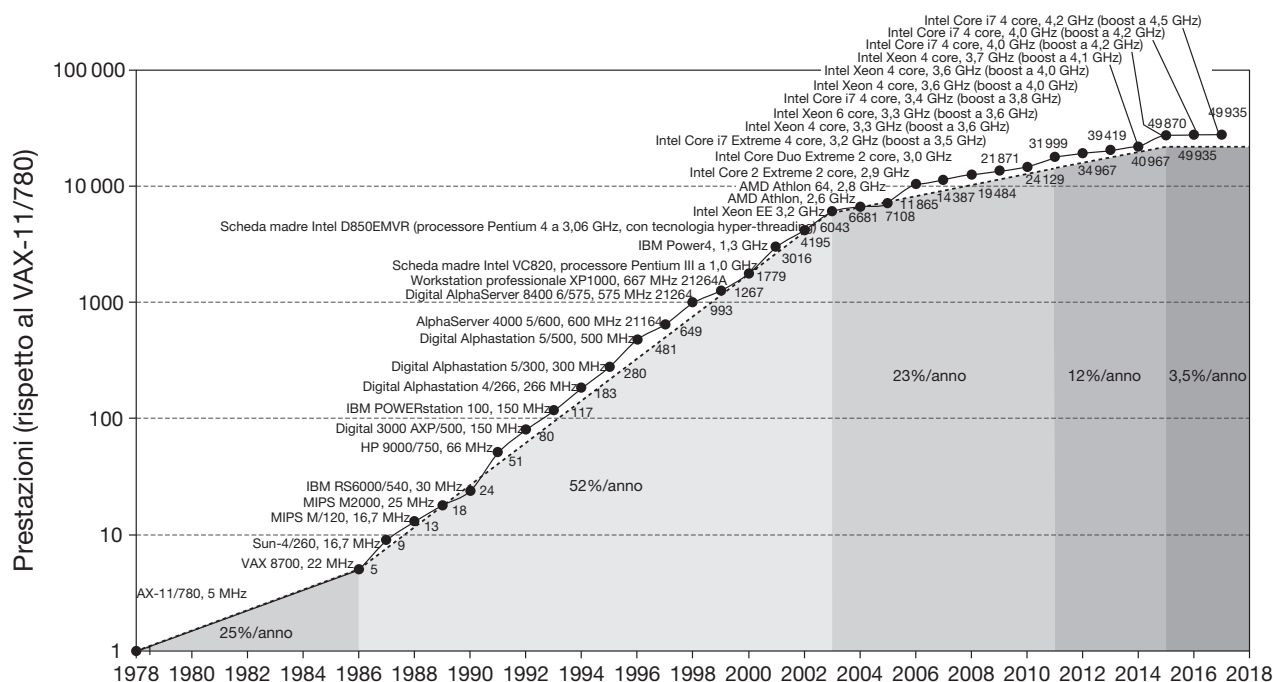


Figura 1.17 Crescita nelle prestazioni dei processori a partire dalla metà degli anni '80. Questo grafico riporta le prestazioni relative al VAX11/780, misurate attraverso i benchmark SPECint (Paragrafo 1.11). Prima della metà degli anni '80, l'aumento delle prestazioni dei processori era dovuto principalmente alla tecnologia ed era dell'ordine del 25% all'anno. L'aumento delle prestazioni nel periodo seguente è stato di circa il 52% all'anno, grazie a nuove idee nella progettazione delle architetture e nell'organizzazione dei calcolatori. Questo ha portato a un aumento delle prestazioni che nel 2002 è stata di sette volte l'aumento che si sarebbe verificato con un aumento di prestazioni del 25% all'anno. A partire dall'anno 2002 il limite sulla potenza assorbita, il parallelismo implicito delle istruzioni e la latenza della memoria hanno rallentato l'aumento delle prestazioni delle architetture monoprocesso a un 3,5% all'anno.

hanno iniziato a chiamare i processori *core*, e quindi questi microprocessori di nuova generazione vengono chiamati *multicore* (multiprocessori): un microprocessore *quadcore* è perciò un chip che contiene al suo interno quattro core (ovvero quattro processori).

In passato, i programmatori potevano confidare nelle innovazioni dell'hardware, delle architetture e dei compilatori per raddoppiare le prestazioni dei loro programmi ogni 18 mesi senza dover modificare una sola linea di codice. Oggi, invece, per ottenere significativi miglioramenti nel tempo di esecuzione dei loro programmi, devono riscrivere il codice per poter sfruttare al meglio i diversi core. Per di più, i programmatori devono continuare ad aggiornare il proprio codice per migliorarne le prestazioni ogni volta che viene aumentato il numero di core, in modo tale che i loro programmi possano essere eseguiti sempre più velocemente via via che vengono introdotti nuovi microprocessori.

Per sottolineare come i sistemi software e hardware lavorino in stretta sinergia, abbiamo inserito delle sezioni speciali denominate *Interfaccia hardware/software* all'interno del testo. In queste sezioni vengono riassunti i concetti fondamentali che stanno alla base dell'interfaccia tra hardware e software. La prima riguarda il parallelismo.

Interfaccia hardware/software

Il **parallelismo** è stato sempre un elemento critico per le prestazioni dei calcolatori, ma spesso è nascosto. Nel Capitolo 4 illustreremo la **pipeline**, una tecnica elegante per eseguire i programmi più velocemente sovrapponendo parzialmente l'esecuzione delle diverse istruzioni. La pipeline è un esempio di *parallelismo a livello di istruzioni*; in questo approccio la natura parallela dell'hardware viene

1.9 Un caso reale: la valutazione del Core i7 Intel

Ogni capitolo contiene un paragrafo dedicato a un “caso reale”, il cui scopo è mettere in relazione i concetti descritti nel libro con un calcolatore che potreste utilizzare tutti i giorni. Questi paragrafi prendono in esame le tecnologie su cui sono basati i calcolatori moderni. Nel primo esaminiamo come vengono prodotti i circuiti integrati e in che modo si misurano le prestazioni e la potenza, utilizzando come esempio il Core i7 di Intel.

Benchmark SPEC per la CPU

Un utente che tutti i giorni utilizzi gli stessi programmi è il candidato ideale per valutare le prestazioni di un nuovo calcolatore. L'insieme dei programmi che esegue rappresenta il **carico di lavoro** (*workload*) e per valutare due elaboratori l'utente dovrebbe semplicemente confrontare il tempo di esecuzione del carico di lavoro sulle due macchine. La maggior parte degli utenti, però, non si trova in questa situazione e deve affidarsi ad altri metodi per la misura delle prestazioni di una macchina, sperando che il metodo prescelto misuri effettivamente le prestazioni del nuovo calcolatore sul carico di lavoro di interesse. Solitamente si sceglie questa alternativa, valutando il calcolatore attraverso un insieme di **benchmark**, ossia programmi campione, appositamente scelti per misurare le prestazioni: i benchmark sono pensati per fornire un carico di lavoro che possa essere significativo al fine di stimare le prestazioni sui carichi di lavoro tipici. Come abbiamo già avuto modo di vedere, per **rendere veloce la situazione più comune**, occorre conoscere bene quale essa sia. Per questo motivo i benchmark giocano un ruolo importante nell'architettura dei calcolatori.

Lo SPEC (*System Performance Evaluation Cooperative*, Cooperativa per la Valutazione delle Prestazioni dei Sistemi) rappresenta uno sforzo congiunto, sovvenzionato e supportato da un certo numero di industrie produttrici di calcolatori, per creare un insieme standard di benchmark per i calcolatori moderni. I primi benchmark SPEC, nati nel 1989, erano focalizzati sulla valutazione delle prestazioni dei processori e vengono ora indicati con la sigla SPEC89. Sono seguite cinque generazioni di benchmark SPEC di cui l'ultima, SPEC CPU2017, è costituita da 10 programmi di benchmark per il calcolo intero (SPECspeed 2017 Integer) e da 13 programmi di benchmark per il calcolo in virgola mobile (SPECspeed 2017 Floating Point). I benchmark per il calcolo intero spaziano da parti di un compilatore C a un programma per giocare a scacchi e a un simulatore di calcolo quantistico. I benchmark per il calcolo in virgola mobile comprendono programmi di calcolo su griglia per la modellazione a elementi finiti, programmi che implementano metodi particellari per la dinamica molecolare e programmi di algebra lineare per la dinamica dei fluidi che implementano il calcolo con matrici sparse.

I programmi di benchmark SPEC per il calcolo intero e il loro tempo di esecuzione su un Core i7 sono riportati nella **Figura 1.18**. Vengono anche riportati i diversi fattori che determinano il tempo di esecuzione: numero di istruzioni, CPI e periodo di clock. Si noti come il CPI vari di un fattore 4 tra i vari programmi.

Per semplificare la valutazione di un calcolatore, il consorzio SPEC ha deciso di riportare anche un singolo valore che riassume i risultati ottenuti nei 10 programmi di benchmark. Per ottenere tale valore, il tempo di esecuzione viene dapprima normalizzato, dividendolo per il tempo di esecuzione misurato su un calcolatore di riferimento; questa operazione fornisce una misura, chiamata **SPECratio** (rapporto tra misure SPEC), che ha il vantaggio di assumere un valore tanto maggiore quanto più veloce è l'esecuzione (lo SPECratio è l'inverso del tempo di esecuzione). Per ottenere una valutazione

Pensavo che [i calcolatori] sarebbero stati un'idea universalmente applicabile, come lo è un libro. Ma non pensavo che si sarebbe sviluppata così velocemente, perché non prevedevo che saremmo stati in grado di integrare su un chip tanti pezzi quanti in effetti è stato possibile. Il transistor comparve inatteso. Avvenne tutto molto più rapidamente di quanto noi ci attendessimo.

J. Presper Eckert, coinventore dell'ENIAC, in un discorso del 1991.

Carico di lavoro: un insieme di programmi eseguiti su un calcolatore; può essere costituito dall'insieme dei programmi utilizzati solitamente dall'utente oppure costruito a partire da programmi reali. Un tipico carico di lavoro specifica sia i programmi che la frequenza relativa con cui vengono eseguiti.

Benchmark: programma selezionato per comparare le prestazioni dei calcolatori.



SITUAZIONI COMUNI

Descrizione	Nome	Numero di istruzioni × 10 ⁹	CPI	Periodo di clock (secondi × 10 ⁻⁹)	Tempo di esecuzione (secondi)	Tempo di riferimento (secondi)	SPECratio
Interprete Perl	perlbench	2684	0,42	0,556	627	1774	2,83
Compilatore GNU C	gcc	2322	0,670	0,556	863	3976	4,61
Pianificatore di rotte	mcf	1786	1,22	0,556	1215	4721	3,89
Libreria di simulazione di eventi discreti	omnetpp	1107	0,82	0,556	507	1630	3,21
Conversione da XML ad HTML mediante XSLT	xalancbm	1314	0,75	0,556	549	1417	2,58
Compressione video	h264	4488	0,32	0,556	813	1763	2,17
Intelligenza Artificiale: ricerca su albero Montecarlo (Scacchi)	deepsjeng	2216	0,57	0,556	698	1432	2,05
Intelligenza Artificiale: ricerca su albero alfa-beta (go)	ieela	2236	0,79	0,556	987	1703	1,73
Intelligenza Artificiale: ricerca ricorsiva di soluzioni (Sudoku)	exchange2	6683	0,46	0,556	1718	2939	1,71
Compressione dati generale	xz	8533	1,32	0,556	6290	6182	0,98
Media geometrica	—	—	—	—	—	—	2,36

Figura 1.18 I programmi che costituiscono il benchmark intero SPECspeed 2017 eseguiti su un Intel Xeon E5-2650L a 1,8 GHz. Come si è visto a pagina 31, l'equazione che calcola il tempo di esecuzione dipende da tre fattori: il numero di istruzioni (misurate qui in miliardi), il numero di cicli di clock per istruzione (CPI) e il periodo di clock, misurato in nanosecondi. Lo SPECratio (rapporto tra misure SPEC) è semplicemente il rapporto tra il tempo di esecuzione di riferimento, fornito da SPEC, e il tempo di esecuzione misurato. L'unico numero riportato in corrispondenza della voce "Media geometrica" è la media geometrica degli SPECratio. Lo SPECspeed 2017 prevede diversi file in input per i diversi programmi: perlbench, gcc, x264 e xz. Per questa figura, il tempo di esecuzione e il numero totale di cicli di clock sono la somma dei tempi di esecuzione di questi programmi per tutti gli input.

globale secondo SPECspeed 2017, viene considerata la media geometrica degli SPECratio misurati sui diversi programmi di benchmark.

Approfondimento. Quando si vogliono confrontare due calcolatori attraverso lo SPECratio, si utilizza la media geometrica in modo da ottenere lo stesso risultato indipendentemente dal calcolatore utilizzato per normalizzare i risultati. Se facessimo la media dei tempi di esecuzione normalizzati utilizzando la media aritmetica, il risultato varrebbe a seconda del calcolatore utilizzato come riferimento.

La formula della media geometrica è:

$$\sqrt[n]{\prod_{i=1}^n \text{Rapporto del tempo di esecuzione}_i}$$

dove *rapporto del tempo di esecuzione* rappresenta il rapporto tra il tempo di esecuzione dell'*i*-esimo programma degli *n* programmi di benchmark sul calcolatore da valutare e:

$$\prod_{i=1}^n a_i \text{ indica il prodotto } a_1 \times a_2 \times \dots \times a_n$$

Benchmark SPEC sull'assorbimento di potenza

Data l'importanza che l'assorbimento di energia e di potenza ricopre nei calcolatori moderni, SPEC ha creato dei benchmark specifici per misurare la potenza. Questi benchmark misurano l'assorbimento di potenza di server in condizioni reali con diversi livelli di carico di lavoro, suddivisi in intervalli con ampiezza del 10%. Nella **Figura 1.19** vengono riportate le misure relative a un server che utilizza il processore Xeon di Intel, simile al Core i7.

Il primo benchmark sull'assorbimento di potenza era presente nello SPECJBB2005, proposto per valutare le applicazioni Java per il mondo degli

% Carico di lavoro	Prestazioni (ssj_ops)	Potenza media (watt)
100%	4 864 136	347
90%	4 389 196	312
80%	3 905 724	278
70%	3 418 737	241
60%	2 925 811	212
50%	2 439 017	183
40%	1 951 394	160
30%	1 461 411	141
20%	974 045	128
10%	485 973	115
0%	0	48
Somma totale	26 815 444	2165
$\Sigma \text{ssj_ops} / \Sigma \text{potenza} =$		12 385

Figura 1.19 Il programma `ssj2008` del benchmark SPECpower eseguito su uno Xeon Platinum 8276L con due processori Intel a 2,2 GHz, doppio connettore, con 192 GB di DRAM e un disco SSD da 80 GB.

affari, e impegnava il processore, la cache e la memoria principale, nonché la Java virtual machine (macchina virtuale Java), il compilatore, il programma di pulizia della memoria (*garbage collector*) e parti del sistema operativo. Le prestazioni vengono misurate attraverso il throughput e l'unità di misura utilizzata è il numero di transazioni al secondo. Anche qui, per semplificare la valutazione, SPEC propone un singolo numero, chiamato “ssj_ops totali per watt”, che viene calcolato come segue:

$$\text{ssj_ops totali per watt} = \left(\sum_{i=0}^{10} \text{ssj_op}_i \right) / \left(\sum_{i=0}^{10} \text{potenza}_i \right)$$

dove ssj_op_i rappresenta le prestazioni per ciascuno degli intervalli con ampiezza del 10% del carico di lavoro e potenza_i rappresenta la potenza assorbita per ciascun livello di carico.

1.10 Come andare più veloci: la moltiplicazione di matrici in Python

Per dimostrare l'impatto delle idee illustrate in questo libro, ciascun capitolo contiene un paragrafo *Come andare più veloci*. In questo paragrafo viene spiegato come migliorare le prestazioni di un programma che moltiplica una matrice con un vettore. Partiamo da questo programma Python:

```
for i in xrange(n):
    for j in xrange(n):
        for k in xrange(n):
            C[i][j] += A[i][k] * B[k][j]
```

Supponiamo di utilizzare il server `n1-standard-96` del Google Cloud Engine, che è costituito da due chip Xeon Skylake di Intel, dove ogni chip ha 24 processori o core e supporta la versione 3.1 di Python. Se le matrici sono di dimensioni 960×960 , la moltiplicazione per un vettore richiede circa 5 minuti utilizzando Python 2.7. Poiché la moltiplicazione in virgola mobile scala con

il cubo della dimensione della matrice, occorrerebbero quasi 6 ore per eseguire la stessa operazione su matrici di dimensioni 4096×4096 . Anche se è veloce scrivere la moltiplicazione di matrici in Python, chi può desiderare aspettare così a lungo per avere il risultato?

Nel Capitolo 2, convertiremo questo codice della moltiplicazione dal linguaggio Python al linguaggio C; questo consentirà, da solo, di aumentare le prestazioni di un fattore 200. Infatti il livello di astrazione di un programma C è molto più vicino all'hardware di quello di Python, e questo è il motivo per cui il linguaggio C verrà utilizzato in questo libro per gli esempi di codice. La minore distanza tra il livello di astrazione del codice C e quello dell'hardware rende l'esecuzione del codice C molti più veloce del codice Python [Leiserson, 2020].

- Nel paragrafo sul parallelismo a livello di dati del Capitolo 3, utilizzeremo il parallelismo a livello di parola attraverso le funzioni intrinseche del C per aumentare le prestazioni di un fattore approssimativamente pari a 8.
- Nel paragrafo sul parallelismo a livello di istruzioni del Capitolo 4, utilizzeremo lo srotolamento dei cicli per sfruttare i cammini di elaborazione multipli e l'esecuzione fuori-ordine dell'hardware per aumentare le prestazioni di un fattore approssimativamente pari a 2.
- Nel paragrafo sull'ottimizzazione della gerarchia delle memorie del Capitolo 5, utilizzeremo la gestione a blocchi dei dati per aumentare le prestazioni di un fattore approssimativamente pari a 1,5.
- Nel paragrafo sul parallelismo a livello di thread del Capitolo 6, utilizzeremo la parallelizzazione dei cicli mediante OpenMP per sfruttare l'hardware multicore per aumentare le prestazioni di un fattore tra 12 e 17.

Gli ultimi quattro passi di ottimizzazione si basano sulla comprensione di come la vora l'hardware di un microprocessore moderno; complessivamente il codice finale richiede solo 21 linee di codice C. La **Figura 1.20** mostra che l'incremento delle prestazioni (speed-up) in scala logaritmica è di circa 50 000 volte rispetto al codice originario Python. La moltiplicazione invece di richiedere 6 ore, richiederà meno di un secondo!

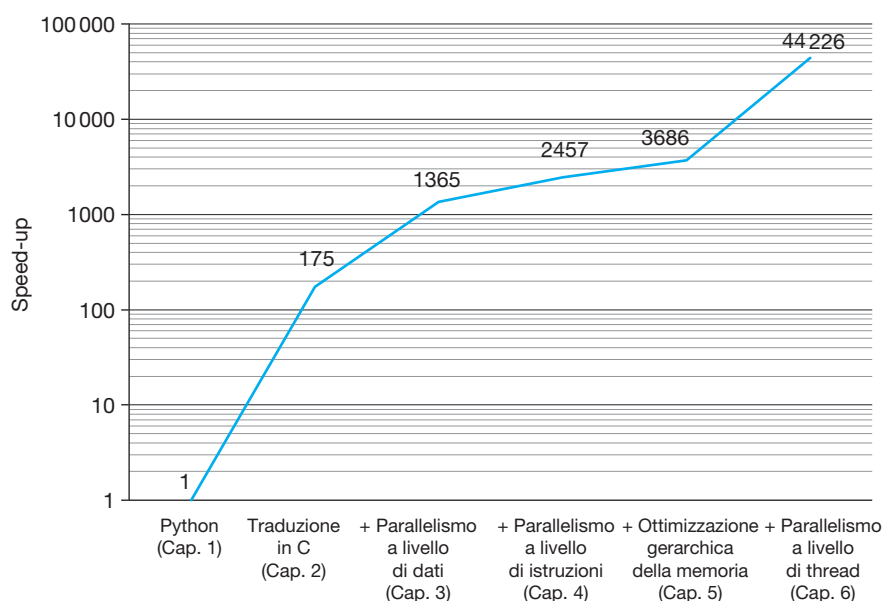


Figura 1.20 Le diverse ottimizzazioni, descritte nei prossimi Capitoli, del programma di moltiplicazione di una matrice per un vettore originariamente scritto in Python.

Approfondimento. Per rendere Python più veloce, i programmatori spesso chiamano librerie molto ottimizzate invece di scrivere loro stessi del codice Python ottimizzato. Poiché cerchiamo di illustrare la differenza intrinseca nella velocità di esecuzione tra Python e C, riporteremo i risultati della moltiplicazione tra una matrice e un vettore per un programma scritto in linguaggio Python di base; se utilizzassimo per esempio la libreria NumPy, una moltiplicazione di una matrice 960×960 per un vettore richiederebbe molto meno di un secondo rispetto ai 5 minuti del codice originario.

1.11 Errori e trabocchetti

L'obiettivo di questo paragrafo è quello di confutare alcune credenze sbagliate che si possono incontrare quando si parla di architetture degli elaboratori. Ogni capitolo di questo libro conterrà un paragrafo di questo tipo, e ogni volta che illustreremo *un errore che nasce da una credenza sbagliata* cercheremo di fornire un controesempio. Illustreremo anche alcuni *trabocchetti*, ossia errori in cui è facile incappare: questi sono spesso originati dalla generalizzazione di principi che sono veri solo in un contesto limitato. Lo scopo di questi paragrafi è quello di aiutare il lettore a evitare di cadere in questi errori nella progettazione e nella valutazione dei calcolatori. Errori e trabocchetti che riguardano il rapporto costo/prestazioni hanno tratto in inganno molti progettisti, compresi gli autori di questo libro, perciò questo paragrafo non soffre della mancanza di esempi significativi. Iniziamo da un trabocchetto che ha fatto inciampare molti progettisti e rivela una caratteristica importante della progettazione dei calcolatori.

Trabocchetto: *ci si aspetta che il miglioramento di uno dei componenti di un calcolatore produca un aumento delle prestazioni proporzionale alla dimensione del miglioramento.*

La grande idea di **rendere veloce la situazione più comune** ha un corollario che tormenta i progettisti dell'hardware e del software e che ci ricorda che l'impatto del miglioramento di una funzionalità dipende dal tempo per il quale quella funzionalità verrà utilizzata.

Lo illustriamo con un semplice esempio. Si supponga che un programma venga eseguito in 100 secondi su un dato calcolatore e che 80 di questi siano impiegati in operazioni di moltiplicazione. Di quanto bisogna migliorare la velocità di esecuzione delle moltiplicazioni se si vuole che il programma venga eseguito 5 volte più velocemente?

Il tempo di esecuzione del programma dopo il miglioramento è dato dalla seguente semplice equazione, nota come **legge di Amdahl**:

$$\begin{aligned} \text{Tempo di esecuzione} \\ \text{dopo il miglioramento} &= \frac{\text{Tempo di esecuzione affetto dal miglioramento}}{\text{Quantità di miglioramento}} + \\ &+ \text{Tempo di esecuzione} \\ &\quad \text{non affetto} \end{aligned}$$

Nel nostro caso si ottiene:

$$\text{Tempo di esecuzione dopo il miglioramento} = \frac{80 \text{ secondi}}{n} + (100 - 80 \text{ secondi})$$

Dato che vogliamo che l'esecuzione diventi cinque volte più veloce, il tempo di esecuzione dovrà scendere a 20 secondi, e quindi:

$$\begin{aligned} 20 \text{ secondi} &= \frac{80 \text{ secondi}}{n} + 20 \text{ secondi} \\ 0 &= \frac{80 \text{ secondi}}{n} \end{aligned}$$

La scienza deve iniziare dai miti e dalla discussione critica dei miti.

Karl Popper, *The Philosophy of Science*, 1957



SITUAZIONI COMUNI

Legge di Amdahl: una regola che afferma che il miglioramento delle prestazioni reso possibile da una data modifica è limitato dalla quantità tempo in cui quella modifica è effettivamente sfruttata. È la versione quantitativa della legge dei rendimenti decrescenti.

Mentre ... l'ENIAC è dotato di 18000 valvole e pesa 30 tonnellate, i calcolatori del futuro potranno avere solo 1000 valvole e pesare solamente una tonnellata e mezza.

Popular Mechanics, marzo 1949.



ASTRAZIONE

1.12 Note conclusive

Sebbene sia difficile prevedere esattamente il costo e le prestazioni dei calcolatori futuri, si può scommettere tranquillamente che saranno molto migliori degli attuali. Per giocare una parte attiva in questo progresso, i progettisti di calcolatori e i programmatori dovranno considerare un insieme di problematiche più vasto.

I progettisti sia dell'hardware sia del software costruiscono i sistemi di elaborazione per strati gerarchici, detti livelli di astrazione, in cui ciascuno strato nasconde dei dettagli al livello superiore. Questa grande idea, l'**astrazione**, è fondamentale per comprendere i calcolatori odierni, ma non significa che i progettisti debbano conoscere solo la tecnologia utilizzata nello strato di loro competenza. Forse l'esempio più importante di astrazione è l'interfaccia tra l'hardware e il software di basso livello, detta *architettura dell'insieme di istruzioni*. Se l'architettura dell'insieme di istruzioni rimane invariata, uno stesso programma sarà eseguibile su diverse implementazioni dell'architettura, che presumibilmente avranno costi e prestazioni diverse. D'altro canto, l'architettura potrebbe costituire un ostacolo all'introduzione di innovazioni che porterebbero alla modifica di questa interfaccia.

C'è un metodo affidabile per determinare e misurare le prestazioni utilizzando come metrica il tempo di esecuzione di programmi reali. Questo è legato ad altre grandezze importanti attraverso la seguente equazione:

$$\frac{\text{Secondi}}{\text{Programma}} = \frac{\text{Istruzioni}}{\text{Programma}} \times \frac{\text{Cicli di clock}}{\text{Istruzione}} \times \frac{\text{Secondi}}{\text{Ciclo di clock}}$$

Utilizzeremo più e più volte questa equazione e i suoi tre termini. Ricordatevi che i singoli fattori non determinano le prestazioni: solamente il loro prodotto costituisce una misura affidabile delle prestazioni ed è uguale al tempo di esecuzione.

QUADRO D'INSIEME

Il tempo di esecuzione è l'unica misura valida e inconfutabile delle prestazioni. Molte altre metriche sono state proposte e sono state inizialmente considerate interessanti. A volte queste metriche risultavano sbagliate fin dall'inizio perché non riflettevano il tempo di esecuzione; altre volte una metrica valida in un contesto circoscritto veniva estesa e utilizzata al di fuori di quel contesto oppure estesa senza le specifiche necessarie per utilizzarla validamente. ■

La tecnologia fondamentale dell'hardware dei calcolatori moderni è basata sul silicio. Mentre il silicio continua a sostenere una rapida evoluzione dell'hardware, nuove idee sull'organizzazione dei calcolatori hanno migliorato il rapporto prezzo/prestazioni. Le due idee fondamentali sono lo sfruttamento del parallelismo nei programmi, tipicamente ottenuto oggi grazie all'utilizzo di processori multicore, e lo sfruttamento della località nell'accesso alla **gerarchia delle memorie**, raggiunto soprattutto attraverso le memorie cache.

Il consumo di energia ha sostituito le dimensioni del chip come elemento critico nella progettazione dei microprocessori. Aumentare le prestazioni senza aumentare l'assorbimento di potenza ha obbligato i produttori di hardware a passare ai processori multicore, richiedendo quindi che i produttori di software passassero al **parallelismo**.

I diversi calcolatori sono stati sempre valutati non solo in termini di costi e prestazioni, ma anche di altri importanti fattori quali il consumo di energia,



GERARCHIA







PARALLELISMO

l'affidabilità, il costo e la scalabilità. Anche se questo capitolo era principalmente focalizzato sul costo, sulle prestazioni e sull'energia, un progetto vincente permetterà il giusto equilibrio tra i tre fattori per il mercato a cui si rivolge.

Organizzazione del testo

Alla base di queste astrazioni ci sono i cinque componenti classici di un calcolatore: unità di elaborazione dati (o datapath), unità di controllo, memoria, input e output (Figura 1.5). Questi cinque componenti servono anche come quadro di riferimento per gli altri capitoli di questo libro:


- *unità di elaborazione dati* (o datapath): Capitoli 3, 4, 6 e Appendice C 
- *unità di controllo*: Capitoli 4, 6 e Appendice C 
- *memoria*: Capitolo 5
- *input*: Capitoli 5 e 6
- *output*: Capitoli 5 e 6

Il Capitolo 4 descrive come i processori sfruttano il parallelismo implicito, mentre nel Capitolo 6 vengono descritti i processori multicore, caratterizzati da un parallelismo esplicito, che sono il nocciolo della rivoluzione del parallelismo; un processore grafico ad alto grado di parallelismo viene descritto nell'Appendice C . Il Capitolo 5 descrive come le gerarchie delle memorie sfruttano la località. Il Capitolo 2 descrive gli insiemi di istruzioni – cioè l'interfaccia tra i compilatori e il calcolatore – ed enfatizza il ruolo dei compilatori e dei linguaggi di programmazione nello sfruttare le caratteristiche dell'insieme delle istruzioni. Il Capitolo 3 mostra come i calcolatori gestiscono i numeri e realizzano le operazioni aritmetiche. L'Appendice A  descrive le tecniche di progettazione dei circuiti logici. Potete trovare tutte le Appendici online sul sito web del libro:

online.universita.zanichelli.it/patterson-risc2e

1.13 Inquadramento storico e approfondimenti

Per ogni capitolo, troverete disponibile online sul sito web del libro un paragrafo dedicato all'inquadramento storico. In questi paragrafi illustriamo lo sviluppo di un'idea anche attraverso i calcolatori che si sono succeduti negli anni, oppure descriviamo progetti particolarmente importanti; inoltre, vengono forniti riferimenti bibliografici utili per ulteriori approfondimenti.

L'inquadramento storico relativo a questo capitolo presenta lo stato dell'arte di alcune delle idee chiave presentate. Lo scopo è illustrare la storia del progresso tecnologico attraverso la storia degli scienziati e degli ingegneri che lo hanno determinato e collocare le loro conquiste nel contesto storico. Comprendendo il passato, si può capire meglio quali aspetti guideranno lo sviluppo dell'informatica nel futuro. Al termine di ogni paragrafo di inquadramento storico, vengono riportati suggerimenti per ulteriori approfondimenti; tutti i suggerimenti dei vari capitoli sono raggruppati nella sezione *Approfondimenti*, disponibile online. Il testo di questo paragrafo è disponibile come Paragrafo 1.13 online .

Una branca della scienza in piena attività è come un immenso formicaio: l'individuo quasi svanisce nell'ammasso di menti che turbinano, trasmettendo le informazioni da una parte all'altra, diffondendole alla velocità della luce.

Lewis Thomas, "Natural Science", in *The Lives of a Cell*, 1974.

1.14 Aiuto allo studio

A partire da questa edizione, abbiamo aggiunto a ogni capitolo, un paragrafo contenente degli esercizi con le relative soluzioni, che speriamo siano

per voi stimolanti e fonte di riflessione approfondita. Questi paragrafi vi consentiranno anche di verificare se avete ben assimilato gli argomenti del capitolo.

Proiettare le grandi idee delle architetture nel mondo reale. Trovare l'applicazione migliore delle sette grandi idee sull'architettura dei calcolatori in queste situazioni del mondo reale:

1. Ridurre il tempo di un bucato lavando il bucato successivo mentre il primo bucato si sta asciugando.
2. Nascondere la chiave di scorta in caso periate la vostra chiave della porta d'ingresso.
3. Controllare le previsioni del tempo per le città che attraverserete quando dovete decidere l'itinerario per un lungo viaggio invernale.
4. Coda veloce alle casse dei supermercati per i clienti che hanno acquistato non più di 10 prodotti.
5. La biblioteca di quartiere associata a una grande biblioteca centrale.
6. Un'automobile con un motore elettrico collegato a tutte e quattro le ruote.
7. Modalità opzionale di guida autonoma di un'automobile che richieda l'acquisto delle funzionalità di parcheggio automatico e navigazione.

Come si misura la velocità? Si considerino tre diversi processori: P1, P2 e P3 che eseguono lo stesso insieme di istruzioni. P1 ha un periodo di clock di 0,33 ns e un CPI di 1,5; P2 ha un periodo di clock di 0,40 ns e un CPI di 1; P3 ha un periodo di clock di 0,25 ns e un CPI di 2,2.

1. Quale processore ha la frequenza di clock più elevata? Quanto vale?
2. Qual è il computer più veloce? Se la risposta è diversa da quella alla domanda precedente, spiegare il motivo. Qual computer è il più lento?
3. Come si riflette sui benchmark la risposta alle domande (1) e (2)?

La legge di Amdahl e la parentela. La legge di Amdahl è di fatto la Legge dei Rendimenti Decrescenti che si applica sia agli investimenti sia alle architetture dei calcolatori. Questo è un esempio che serve per chiarire questa legge – Uno dei vostri fratelli è entrato in una start-up e sta cercando di convincervi a investire una parte dei vostri risparmi in essa, dichiarando che è “una cosa sicura!”.

1. Voi decidete di investire il 10% dei vostri risparmi. Quale dovrà essere il rendimento del vostro investimento nella start-up perché il vostro patrimonio complessivo raddoppi, supponendo che questo sia il vostro unico investimento?
2. Supponendo che la start-up produca effettivamente il rendimento calcolato in (1), quanta parte dei vostri risparmi dovreste investire per ottenere il 90% dell'incremento del valore della start-up? E quanto per ottenere il 95%?
3. Come sono collegati i risultati ottenuti in (1) e (2) con l'osservazione della legge di Amdahl che riguarda i computer? Cosa dice questa legge sul rapporto con i fratelli?

Prezzo e costo delle DRAM. La Figura 1.21 mostra il prezzo dei chip di DRAM dal 1975 al 2020, mentre la Figura 1.1 mostra la capacità dei chip di DRAM nello stesso periodo. Queste figure mostrano un aumento di 1 000 000 di volte della capacità (da 16 Kbit a 16 Gbit) e una riduzione di 25 000 000 di volte del prezzo per gigabyte (da 100 milioni a 4 di dollari americani). Notate che il prezzo per GiB fluttua su e giù negli anni, mentre la capacità per chip aumenta in modo stabile nello stesso periodo.

1. Potete apprezzare un rallentamento della legge di Moore nella Figura 1.21?
2. Perché il prezzo aumenta di un fattore maggiore di 25 volte rispetto

David A. Patterson, John L. Hennessy

Struttura e progetto dei calcolatori

Progettare con RISC-V

Seconda edizione italiana a cura di Alberto Borghese

Oggi ai professionisti di ogni settore dell'informatica è richiesto di conoscere sia il software sia l'hardware, la cui interazione offre la chiave per capire i principi dell'elaborazione. Ogni programma infatti, per essere eseguito più velocemente e raggiungere una maggiore efficienza energetica, deve diventare un **programma parallelo**.

Proprio su questo aspetto Patterson e Hennessy hanno posto l'enfasi fin dalla prima edizione di *Struttura e progetto dei calcolatori*, e il passaggio tecnologico dalle architetture uniprocessore ai multiprocessori multicore ha confermato quanto la prospettiva del parallelismo fosse giusta. A questo si aggiunge la scelta di dedicare una versione dell'opera all'**architettura RISC-V**, un insieme di istruzioni progettato per funzionare con cloud computing, dispositivi mobili e altri sistemi embedded, più semplice ed elegante dell'insieme di istruzioni MIPS, con anche il vantaggio di non essere un'architettura proprietaria: esistono infatti simulatori, compilatori e debugger RISC-V open-source, oltre che implementazioni RISC-V open-source scritte nei linguaggi di descrizione dell'hardware.

Questa seconda edizione presenta:

- i formati più complessi in modo graduale e nella versione a 32 bit, che riduce le istruzioni core a 10;
- la revisione della parte sul calcolo parallelo, con l'inserimento di 4 ottimizzazioni che consentono un risparmio di tempo a fronte di sole 21 linee di codice;
- la trattazione delle Architetture Specifiche di Dominio (DSA), che attraversano una fase di crescita;
- il reinserimento delle architetture multi-ciclo dei MIPS;
- più attenzione all'impatto dei software open-source sulle architetture;
- un confronto approfondito tra le architetture RISC-V e quelle MIPS, ARMv7, ARMv8 e x86.

Le rubriche *Capire le prestazioni dei programmi*, *Come andare più veloce* e *Interfaccia software/hardware* sottolineano i criteri fondamentali per chi progetta.

I paragrafi *Aiuto allo studio* e *Autovalutazione*, con risposte a fine capitolo, accompagnano chi studia. Tutti gli esercizi sono stati rinnovati.

David A. Patterson è professore emerito di Computer Science alla University of California, Berkeley. È stato direttore della Computing Research Association, presidente della Association for Computer Machinery e consulente del presidente degli Stati Uniti per le tecnologie informatiche.

John L. Hennessy è stato rettore della Stanford University, in California, dove ha insegnato Computer Science dal 1977. Nel 1984 ha fondato la società MIPS Computer Systems Inc.

Nel 2017, Patterson e Hennessy hanno ricevuto insieme il prestigioso Turing Award.

Il titolo originale dell'opera è
**Computer Organization and Design
RISC-V Edition, 2e.**

Questa traduzione è pubblicata con
l'autorizzazione di Elsevier.



ELSEVIER

Le risorse multimediali



online.universita.zanichelli.it/patterson-riscv

A questo indirizzo sono disponibili le risorse multimediali di complemento al libro. Per accedere alle risorse protette è necessario registrarsi su **my.zanichelli.it** inserendo il codice di attivazione personale contenuto nel libro.

Libro con ebook



Chi acquista il libro nuovo può accedere gratuitamente all'**ebook**, seguendo le istruzioni presenti nel sito. L'ebook si legge con l'applicazione *Booktab*, che si scarica gratis da App Store (sistemi operativi Apple) o da Google Play (sistemi operativi Android).

L'accesso all'ebook e alle risorse digitali protette è personale, non condivisibile e non cedibile, né autonomamente né con la cessione del libro cartaceo.

PATTERSON*STRUT PR CALC RISC 2E LUMK

ISBN 978-88-08-19966-9



9 788808 199669

4 5 6 7 8 9 0 1 2 (60M)