Građa računala

Paralelizam - single i multi-core procesori

Tipovi paralelizma

- Paralelizam u hardveru
  - single-core - pipelining, superscalar, VLIW
  - SIMD instructions, Vector processors, GPU (primjer: MMX, SSE)
  - multi-core - SMP, distributed-memory MP
  - multicomputer (klasteri)
- Paralelizam u softveru
  - Instruction-level parallelism
  - Task-level parallelism
  - Data parallelism
  - Transaction level parallelism

Instruction-level parallelism

- Više instrukcija iz istog toka instrukcija se izvršavaju u isto vrijeme
- Generiran i upravljan od strane hardvera (superskalarni procesori) ili kompajlera (VLIW)
- Pristup koji je u praksi limitiran podatkovnim i kontrolnim međuovisnostima

Thread-level ili task-level paralelizam

- Više threadova ili sekvenci instrukcija iz iste aplikacije izvršavaju se konkurentno
- Generiran od strane kompajlera (korisnika), upravljan od strane kompajlera i hardvera
- U praksi, limitiran zbog overheada u komunikaciji i sinhronizaciji, kao i karakteristika različitih algoritama

Data-level paralelizam

- Instrukcije iz jednog toka operiraju konkurentno na više podataka
- U praksi, limitiran od strane nepravilnih uzoraka manipulacije podataka i bandwidth-a memorije

Transaction-level paralelizam

- Više threadova/procesa iz različitih transakcija se izvršavaju konkurentno
- U praksi limitiran zbog overheada konkurentnosti

Osnovni pojmovi - ponavljanje

Procesori koriste slijedeće tehnologije za poboljšanje performansi:

- pipelining (razbijanje instrukcija u manje dijelove);
- superskalarnost (neovisno izvršavanje instrukcija u različitim dijelovima procesora);
- out-of-order izvršavanje (redoslijed izvršavanja je drugačiji od programskog koda)
- problem: svaka od ovih metoda znači dodatnu kompleksnost hardvera

Superpipelined i Superscalar procesori

U praksi, pokazalo se da je bolje producirati superskalarne procesore, često sa vrlo dubokim *pipelineom*, što je puno bolje od superpipelined procesora:

- problemi sa povećanim frekvencijama rada procesora
- neke operacije ili moduli se teško razbijaju na pipeline operacije
- potreba za balansiranjem logike u različitim fazama pipelinea

Ali:

- Scheduling instrukcija je vrlo kompleksan
- Postoje i druge metode - unutrašnji paralelizam u instrukcijskom toku, kompleksnost, grananje kroz VLIW (Very Long Instruction Word)

VLIW

- instruction-level paralelizam - programi kontroliraju paralelno izvršavanje instrukcija
- koristi kompajler za razrješavanje svih problema
- potprogrami odlučuju paralelni tijek instrukcija i rješavaju konflikte
- povećava se kompleksnost kompajlera, ali u isto vrijeme se smanjuje kompleksnost hardvera

VLIW mogućnosti

- procesori imaju više funkcionalnih jedinica, dohvaćaju potrebne instrukcije iz instrukcijske cache memorije koja ima VLIW organizaciju
- Više neovisnih operacija se grupira zajedno u jednu VLIW instrukciju, koja se inicijalizira u istom ciklusu
- Svaka operacija dobiva svoju neovisnu funkcionalnu jedinicu
- Sve funkcionalne jedinice imaju zajedničke registre
- Instrukcijske riječi su tipično dugačke 64-1024 bitova, ovisno o broju izvršnih jedinica i duljine koda koji je potreban za kontrolu svake funkcionalne jedinice
- scheduling instrukcija i paralelno slanje riječi se radi statički (kompajler)
- kompajler provjerava međuovisnosti prije schedulinga paralelnih instrukcija

VLIW prednosti

- Smanjenje kompleksnosti hardvera
- Smanjenje potrošnje energije (zbog manje kompleksnosti hardvera)
- Pošto se međuovisnost provjerava u fazi kompajliranja, dekodiranje, problematike instrukcija i dekodiranje se značajno pojednostavljuju
- Potencijalni porast frekvencije
- Funkcionalne jedinice su pozicionirane u skladu sa potrebama kompajlera

VLIW problemi

- Kompleksni kompajleri koje je teško dizajnirati
- Povećana veličina programa
- Potreban je veći memorijski bandwidth, kao i bandwidth registara
- neočekivani događaji (npr.cache miss) mogu dovesti do zastoja u pipelineu
- ako VLIW instrukcije imaju praznina, trošimo memorijski prostor i bandwidth instrukcija za ništa

SIMD (Single-Instruction, Multiple-Data)

- višeprocesorska mašina sposobna izvršavati istu instrukciju na procesoru ali raditi na različitim podacima
- ovaj se pristup tipično koristi u znanstvenim izračunima pošto takva vrsta računarstva obično ima puno vektorskih i matričnih operacija
- Informacije se mogu slati na sve elemente procesiranja, organizirani podatkovni elementi vektora mogu biti razdijeljeni u više setova (N setova za N elemenata procesiranja), pa svaki element procesiranja može procesirati jedan set podataka

ARM SIMD instrukcije (NEON)

"Arm Neon technology is an advanced Single Instruction Multiple Data (SIMD) architecture extension for the A-profile and R-profile processors. Neon technology is a packed SIMD architecture. Neon registers are considered as vectors of elements of the same data type, with Neon instructions operating on multiple elements simultaneously. Multiple data types are supported by the technology, including floating-point and integer operations. Neon technology is intended to improve the multimedia user experience by accelerating audio and video encoding and decoding, user interface, 2D and 3D graphics, and gaming. Neon can also accelerate signal processing algorithms and functions to speed up applications such as audio and video processing, voice and facial recognition, computer vision, and deep learning."

Source: ARM Developer website, https://developer.arm.com/Architectures/Neon

Intel SIMD instrukcije (Instruction Set Extensions)

- MultiMedia eXtension (1996) - MMX (Pentium MMX, Pentium II)
- Streaming SIMD Extensions - SSE (Pentium III), SSE2 (Pentium 4), SSE3 (Pentium 4 with HT), SSE4 (Intel Silvermont, 2013)
- Advanced Vector eXtensions - AVX (2008.), AVX2 (2013), AVX-512 (2013)

Vektorski procesori

"Although what constitutes a vector processor has changed over the years, their key characteristic is that they can operate on arrays or vectors of data, while conventional CPUs operate on individual data elements or scalars."

- Peter S. Pacheco, Matthew Malensek: An Introduction to Parallel Programming (Second Edition), 2022.

Tipični dijelovi vektorskih procesora

- Vektorski registri - sposobni pohraniti vektor operanda i izvršavati operacije nad njima. Duljina je fiksirana na razini sustava, obično je između 4 i 128 64-bitna elementa
- Vektorizirane i pipeline jedinice - izvršavaju istu operaciju na svaki vektorski element, ili, u slučaju operacija tipa zbrajanje, ista operacija se izvršava nad parom susjednih elemenata dva vektora. Dakle, radi se o SIMD operacijama
- Interleaved memorija - ima više banki memorije, kojima se može pristupati manje ili više neovisno. Nakon pristupa jednoj banci, postoji kašnjenje nakon kojeg se može ponoviti pristup, ali različitoj banci se može pristupiti puno ranije. Dakle, ako su elementi vektora distribuirani kroz više banki, kašnjenje kod load-store operacija će biti blizu 0.
- pristup memoriji "u koracima", hardverski mehanizam za sakupljanje i komadanje - program pristupa elementima vektora koji su locirani na fiksnim intervalima u memoriji. Npr. prvi, peti, deveti itd (stride = 4). Sakupljanje/komadanje je zapisivanje (sakupljanje) ili čitanje (komadanje) elemenata vektora koji su locirani na varijabilnim intervalima - npr. prvi, drugi, četvrti, osmi, i slično. Tipični vektorski sustavi imaju poseban hardver za akceleraciju ovih operacija.

Prednosti vektorskih procesora

- brzina, jednostavno korištenje za dosta aplikacija
- kompajleri su jako dobri u smislu detektiranja koda koji se može vektorizirati, petlje koje se ne mogu vektorizirati, informacije zašto i slično - olakšava odluke o redizajnu aplikacije ako je potreban
- velika brzina memorije, svaki podatak koji se se i koristi, što je u suprotnosti sa spekulativnim izvršavanjem, cache memorijom i sličnim tehnologijama koje ne koriste svaki podatak

Nedostatci vektorskih procesora

- ne vole neuredne podatke
- imaju jako tanku liniju skalabilnosti, tj. sposobnost da odrade veće zadaće
- što su vektori veći, to je teže dizajnirati sustav koji ih može korstiti
- skalabilnost se postiže većim brojem vektorskih procesora, ne duljinom vektora
- trenutni sustavi imaju ograničenu podršku za operacije na vrlo kratkim vektorima, dok su procesori koji rade sa dugačkim vektorima rijetki i proizvedeni po narudžbi, i stoga vrlo skupi

Grafičke kartice

- koriste točke, linije i trokute za predstavljanje površine fizičkog objekta
- koriste pipeline grafičkog procesora za pretvaranje interne reprezentacije u niz piksela koji se mogu prikazivati na ekranu
- više faza pipelinea == programibilne faze
- ponašanje programibilnih dijelova pipelinea speficira se korištenjem funkcija koje se zovu shader funkcije
- shader funkcije su obično jako kratke, često samo par linija C koda
- shader funkcije su implicitno paralelne, pošto se mogu primjeniti na više elemenata (npr. na vertice, točke gdje se spajaju dva ili više segmenta linija) u nizu grafike

Grafičko procesiranje - I

- SIMD procesiranje je osnova grafičkog procesiranja pošto se primjena shader funkcije na elemente grafike odvija u više faza
- ovakvo procesiranje postiže se kroz primjenu velikih količina ALU-a na svakoj jezgri GPU procesora

Grafičko procesiranje - II

- Procesiranje jedne slike traži ogromne količine podataka - često i stotine megabajta
- GPU-ovi zbog toga moraju održavati veliku brzinu i efikasnost premještanja podataka, a u svrhu izbjegavanja zastoja koriste hardverski multithreading - neki GPUovi su sposobni spremiti stanje više od 100 pauziranih threadova za svaki pokrenuti thread
- Broj threadova ovisi o količini resursa (npr. registara) koje trebaju shader funkcije
- Negativna strana - potrebna je velika količina threadova koja procesira podatke da bi ALU bio efikasno iskorišten, pa GPU-ovi imaju relativno loše performanse na malim problemima

Dodatne karakteristike GPU-ova

- GPU-ovi nisu striktno SIMD sustavi
- Iako ALUovi koriste SIMD paralelizam, GPU-ovi mogu imati veliku količinu jezgri koje mogu izvršavati neovisne instrukcije
- vrlo popularni za generalno računarstvo visokih performansi
- razvijeno nekoliko programskih jezika koje koristimo za iskorištavanje njihovih mogućnosti - za kombinaciju CPU+GPU postoji OpenCL, a za iskorištavanje GPU-a samog CUDA

Ubrzanje, skalabilnost

- Skalabilnost - dodavanje X puta više resursa da bismo dobili blizu X puta bolje performanse
  - Resursi su obično procesori, ali može biti memorija, ili bandwidth
  - Obično znači da ako dodamo X puta više procesora želimo dobiti cca X puta veće performanse
- Postoji nekoliko modela skalabilnosti
  - Ograničeni problemom
  - Ograničeni vremenom
- Adekvatni model ovisi o potrebama korisnika

Skaliranje ograničeno problemom

- Veličina problema je fiksna, cilj je smanjiti vrijeme izvršavanja
- Povećavamo broj procesora i količinu memorije
- "Ubrzanje" se definira kao

$$S_{PC} = \frac{\text{Time(1 processor)}}{\text{Time(p processors)}}$$

- Primjer: simulacija vremenske prognoze koja se ne završi u razumnom vremenu

Skaliranje ograničeno vremenom

- Zadana je maksimalna količina vremena za izvršavanje, fiksno
- Cilj je smanjiti veličinu problema
- Broj procesora i količina vremena se povećava
- "Ubrzanje" se definira kao

$$S_{TC} = \frac{\text{Work(p processors)}}{\text{Work(1 processor)}}$$

- Primjer: simulacija vremenske prognoze sa poboljšanim, finijim rasterom

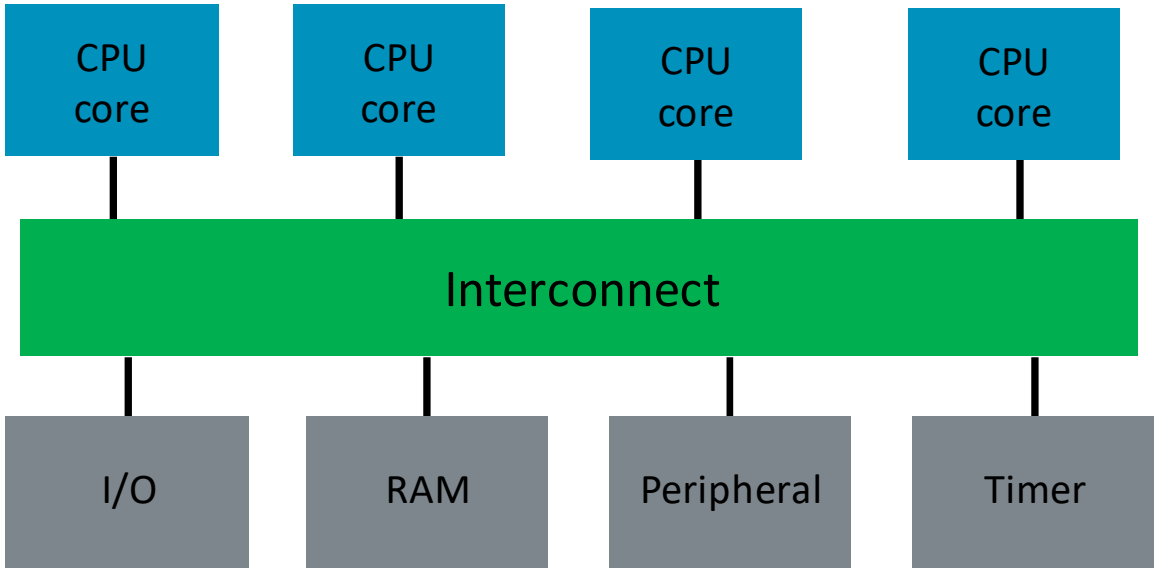
Motivacija za razvoj višejezgrenih/ višeprocesorskih sustava

- Moore-ov zakon
  - Obično temeljni zakon za ubrzanje single-core procesora
  - Obično kroz bolje spekulativno izvršavanje, koje na žalost ne daje linearno poboljšanje performansi
- Ovakva shema nije više moguća pošto Dennardovo skaliranje više ne vrijedi
  - Kakav je smisao razviti procesor koji koristi veću količinu energije bez većeg poboljšanja performansi?
- Višejezgrene arhitekture su efikasniji način korištenja tranzistora
  - Napredak u performansama dolazi kroz paralelizam, thread ili process-level paralelizam
  - Multi-core procesori su postojali i prije, ali su ih ovi problemi pogurnuli u mainstream
- Koji su izazovi kod komunikacije između više jezgri? Kako komuniciraju?

Multi-core arhitekture

- Više jezgri na jednom čipu
- Jezgre su spojene kroz nekakav oblik sabirnice
- Jezgre dijele neke komponente - npr. memorijsko sučelje ili neku razinu cache memorije

Primjer multi-core sustava

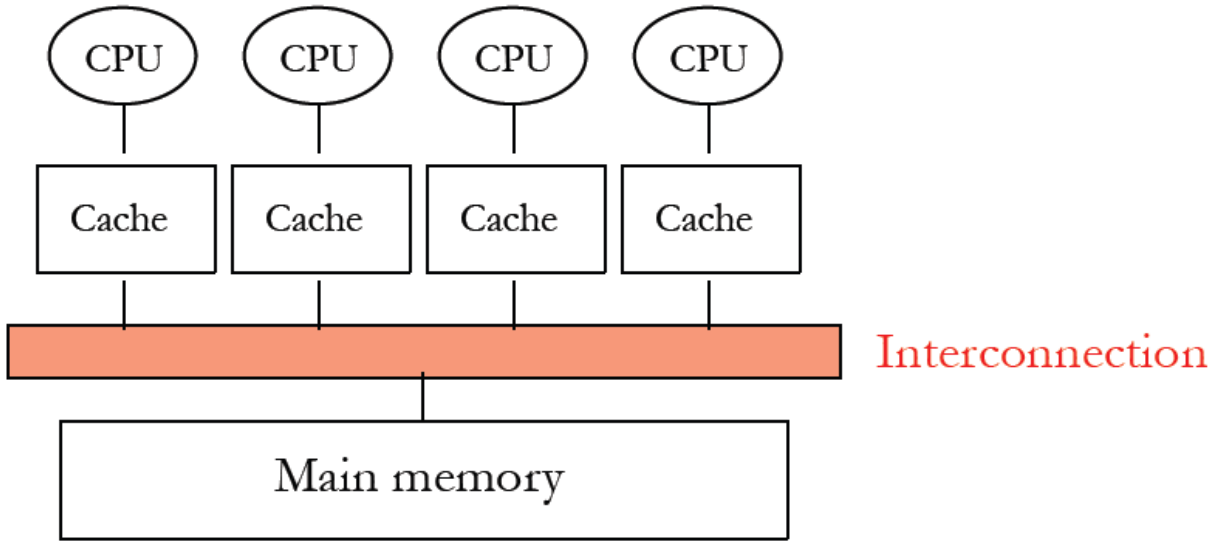


Multi-core

- Jezgre povezane i mogu surađivati
- Puno izazova:
  - Kako jezgre međusobno komuniciraju?
  - Kako se sinhroniziraju podaci?
  - Kako osiguravamo da jezgre ne dobe zastarjele podatke koje su modificirale druge jezgre?
  - Kako jezgre vide redoslijed događaja koji dolaze iz drugih jezgri?
- Obično ove koncepte razmatramo kroz razmatranje:
  - zajedničke memorije i prenošenja poruka
  - cache koherencije
  - konzistentnosti memorije

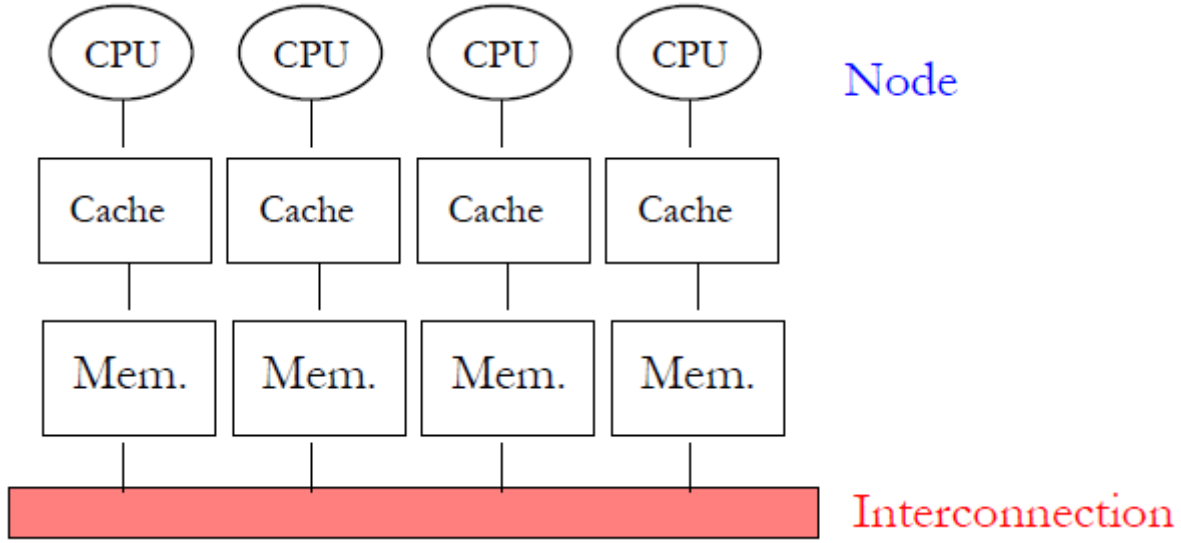
UMA/SMP

- Model zasnovan na slijedećoj fizičkoj organizaciji procesora i memorije:
  - fizički centralizirana memorija, uniformni pristup
  - sva memorija je na jednakoj "udaljenosti" od svih procesora - zbog toga se često zove i simetrično multiprocesiranje (SMP)
  - brzina rada memorije je fiksna i mora zadovoljiti potrebe svih procesora - problem skaliranja na veći broj procesora
  - Koristi se danas - npr.u sustavima sa jednim fizičkim podnožjem za procesore



NUMA

- Model zasnovan na slijedećoj fizičkoj organizaciji procesora i memorije:
  - fizički distribuirana memorija, nije uniformna
  - dio memorije je alociran za svaki procesor
  - pokušava se napraviti sustav kod kojeg je većina zahtjeva prema memoriji lokalnog tipa
  - pristup lokalnoj memoriji je brži od pristupa remote memoriji
  - zajedno, to znači da ako imamo većinom lokalne pristupe memoriji, brzina rada memorije se povećava linearno sa brojem procesora
  - koristi se kod multi-socket sustava (serverski sustavi)



Hvala na pažnji!