Paralelné počítačové systémy

Poznámky z prednášok

Spracované z nahraných PDF prednášok a doplnkových materiálov.

Ako tento dokument používať

Legenda stavu témy

- HOTOVÉ = téma je vysvetlená v rozsahu vhodnom na základné učenie a opakovanie.
- SKRÁTENÉ = téma je v prednáškach širšia alebo obsahuje technické detaily; v dokumente je uvedené jadro a označenie, že pri učení sa oplatí pozrieť pôvodný materiál.
- SKÚŠKOVO DÔLEŽITÉ = typická téma na definície, porovnania alebo krátke výpočty.

- **Odporúčaný postup:** najprv prejsť kapitoly 1–5, potom programovanie Pthreads/OpenMP/CUDA/OpenCL, nakoniec FPGA/HLS a doplnkové architektúry.
- **Čo sa učiť naspamäť:** definície, rozdiely medzi modelmi, taxonómie, základné vzorce, názvy direktív a synchronizačných primitív.
- **Čo chápať princípovo:** prečo komunikácia a synchronizácia znižujú ideálne zrýchlenie, prečo je prístup do pamäte často väčší problém ako počet procesorov, a prečo rôzne architektúry vyžadujú rôzne programovacie modely.

Téma	Zdroj	Stav	Poznámka
Úvod, motivácia, speedup	1. Úvod	HOTOVÉ	Základné definície, dôvody paralelizmu, výkonové požiadavky.
Konvergencia paralelných architektúr	2. Konvergencia	HOTOVÉ	Dôležité pojmy: komunikačná architektúra, programovací model, abstrakcia.
Klasifikácia, Flynn, výpočtové modely	3. Klasifikácia; 3. Výpočtové modely	HOTOVÉ + SKRÁTENÉ	PRAM a niektoré historické modely sú zhrnuté skrátene.
Návrhové otázky, komunikácia, výkon	3. Základné otázky návrhu	HOTOVÉ	Dôležité na porovnávacie otázky.
Vlastnosti paralelizmu, granularita, škálovateľnosť	4. Charakteristické vlastnosti paralelizmu	HOTOVÉ	Obsahuje aj izoefektívnosť a modely pracovnej záťaže.
Prúdové spracovanie, vektorizácia	5. Prúdové spracovanie	HOTOVÉ	Vzorce sú v praktickej forme.
Prepojovacie siete	6. Prepojovacie siete	SKRÁTENÉ	Permutačné funkcie sú uvedené len v jadre.
ILP a TLP architektúry	7. ILP; 8. TLP	HOTOVÉ	VLIW vs superskalárne, CF/DF/hybrid model.
Pthreads a OpenMP	9. Pthread; 10. OpenMP	HOTOVÉ	Praktické API, kritické sekcie, bariéry, redukcie.
GPU, CUDA, OpenCL, GPGPU	11. GPU/CUDA/OpenCL/GPGPU	HOTOVÉ + SKRÁTENÉ	OpenCL API je rozsiahle, preto je v dokumente jadro.
FPGA a High-Level Synthesis	12. FPGA; 13. HLS	HOTOVÉ + SKRÁTENÉ	Detaily nástrojov Vivado HLS sú zhrnuté skrátene.
MIMD paradigmy, dataflow, redukčné počítače	12. Paradigmy MIMD	HOTOVÉ + SKRÁTENÉ	Historické/specializované architektúry sú v prehľade.

1. Celková mapa predmetu

Paralelné počítačové systémy riešia, ako rozdeliť výpočet na viacero výpočtových jednotiek, ako zabezpečiť komunikáciu medzi nimi a ako dosiahnuť reálne zrýchlenie napriek réžii. Kľúčový rozdiel oproti sekvenčnému výpočtu je v tom, že výkon už neurčuje iba rýchlosť jedného jadra, ale aj dostupný paralelizmus, pamäťový systém, sieť, synchronizácia, granularita úloh a programovací model.

Najdôležitejšia veta na skúšku

- Paralelný počítač je súbor procesných elementov, ktoré spolupracujú pri efektívnom riešení veľkých problémov; efektívnosť závisí od rozdelenia práce, komunikácie, synchronizácie a škálovateľnosti.

Pojem	Význam	Typická skúšková pointa
Procesný element (PE)	Výpočtová jednotka: jadro, procesor, uzol, GPU multiprocesor alebo iný prvok schopný vykonávať časť úlohy.	Nepísať automaticky „procesor“; PE je všeobecnejší pojem.
Komunikácia	Prenos dát medzi PE alebo medzi PE a pamäťou.	Často znižuje reálny speedup.
Synchronizácia	Zosúladenie poradia vykonávania a prístupu k zdieľaným dátam.	Bez nej vznikajú race conditions; s ňou vzniká réžia.
Granularita	Veľkosť práce medzi synchronizáciami alebo komunikáciami.	Jemnozrnný paralelizmus má veľa réžie; hrubozrnný sa ľahšie škáluje.
Škálovateľnosť	Schopnosť systému zvyšovať výkon pri zvyšovaní počtu PE alebo veľkosti problému.	Ideálna škálovateľnosť je zriedkavá, limituje ju komunikácia a sériová časť.

2. Úvod do paralelných architektúr

2.1 Prečo paralelizmus vznikol

- Sekvenčné zvyšovanie výkonu cez vyššiu taktovaciu frekvenciu a zložitejšie jadro narazilo na technologické, energetické a tepelné limity.
- Aplikácie ako vedecké simulácie, databázy, grafika, AI, spracovanie obrazu a serverové systémy majú vysoké nároky na výpočtový výkon, pamäťovú priepustnosť alebo sieťovú priepustnosť.
- Paralelizmus sa objavuje na viacerých úrovniach: vnútri inštrukcie, medzi inštrukciami, medzi vláknami, procesmi, uzlami, GPU jadrami alebo špecializovanými obvodmi.
- Architektúra sa dnes neposudzuje len podľa ISA. Dôležité je, ako podporuje komunikáciu, synchronizáciu, programovateľnosť a škálovanie.

2.2 Speedup a efektívnosť

Pre fixnú veľkosť problému platí výkon = 1 / čas. Zrýchlenie pri použití p procesorov vyjadruje, koľkokrát je paralelné riešenie rýchlejšie ako sekvenčné.

Vzorce

- Speedup: S(p) = T(1) / T(p). – T(1) – čas výpočtu na 1 procesore, T(p) – cas vypoctu na p procesoroch
- Efektívnosť: E(p) = S(p) / p.

- Ideálny prípad: S(p) = p a E(p) = 1. V praxi sa znižuje vplyvom sériovej časti, komunikácie, synchronizácie, nerovnomernej záťaže a pamäťových limitov.

- **Sériová časť:** časť programu, ktorá sa nedá paralelizovať. Obmedzuje maximálne zrýchlenie.
- **Komunikačná réžia:** čas potrebný na prenos dát a koordináciu medzi jednotkami.
- **Load imbalance:** niektoré jednotky majú viac práce, iné čakajú.
- **Pamäťová priepustnosť:** ak všetky jednotky čakajú na dáta z pamäte, vyšší počet jadier nepomôže.

SKÚŠKOVO DÔLEŽITÉ

- Vedieť slovne vysvetliť, prečo N procesorov neznamená automaticky N-násobný výkon.
- Vedieť rozlíšiť speedup a efektívnosť. = Speedup – kolkokrát je paralelny vypocet rychlejsi, Efektivnosť – ako dobre využívame procesory.
- Vedieť uviesť príklady aplikácií vhodných pre paralelizmus: matice, simulácie, spracovanie obrazu, vektorové operácie, nezávislé úlohy.

3. Konvergencia paralelných architektúr

Historicky sa paralelné architektúry vyvíjali rôznymi smermi: SIMD, odovzdávanie správ, zdieľaná pamäť, dataflow, systolické polia. Moderný pohľad ich spája cez vrstvený rámec, v ktorom sú dôležité programovacie modely, komunikačné abstrakcie, knižnice, OS, kompilátory a komunikačný hardvér.

SIMD – simple instruction, multiple data - riadiaca jednotka vydáva tú istú inštrukciu viacerým výpočtovým jednotkám naraz, ale každá jednotka ju vykonáva nad inými dátami.

Pojem	Význam	Typická skúšková pointa
Programovací model	To, čo používa programátor: vlákna, zdieľaná pamäť, správy, dátový paralelizmus, tasky.	Architektúra môže podporovať viac programovacích modelov.
Komunikačná abstrakcia	Logický spôsob výmeny dát: load/store cez zdieľanú adresu, send/receive, kolektívne operácie.	Moderná architektúra sa často opisuje cez komunikáciu, nie iba ISA.
Rozhranie používateľ/systém	Vrstva medzi aplikáciou a podporou OS/knižníc.	Kompilátor a knižnice mapujú model na hardvér.
Komunikačný hardvér	Sieť, zbernica, cache koherencia, pamäťové radiče.	Určuje latenciu, priepustnosť a škálovanie.

- STARÝ prístup: inštrukčno orientovaná architektúra, dôraz na ISA.
  - ISA – Instruction Set Architecture - Určuje, aké inštrukcie procesor pozná a ako ich programátor/kompilátor môže používať. (napr. ADD, SUB, LOAD, STORE…, potom ake registre existuju, ake datove typy atd..)
- NOVÝ prístup: komunikačná architektúra, dôraz na spoluprácu prvkov systému.
- Knižnice, kompilátory a OS tvoria most medzi programovacím modelom a konkrétnym hardvérom.
- Konvergencia znamená, že rôzne stroje môžu navonok podporovať podobné modely a podobné stroje sa môžu používať rôznymi modelmi.

4. Klasifikácia a výpočtové modely

4.1 Koncept uloženého programu

Klasický počítač vykonáva sekvenciu inštrukcií nad dátami uloženými v registroch a pamäti. Program je množina inštrukcií, inštrukcia obsahuje operačný kód a operandy. Von Neumannova architektúra používa spoločnú pamäť pre inštrukcie aj dáta, Harvardova architektúra ich oddeľuje.

Pojem	Význam	Typická skúšková pointa
Von Neumann	Spoločná pamäť pre program a dáta.	Jednoduchší model, možný bottleneck spoločnej pamäte.
Harvard	Oddelená pamäť/programová a dátová cesta.	Vyššia priepustnosť, časté v embedded/DSP kontexte.
ISA	Inštrukčná sada viditeľná programátorovi alebo kompilátoru.	Nie je to celý obraz paralelnej architektúry.

4.2 Flynnova taxonómia

Pojem	Význam	Typická skúšková pointa
SISD	Single Instruction, Single Data. Jeden prúd inštrukcií a jeden prúd dát.	Klasický sekvenčný počítač.
SIMD	Single Instruction, Multiple Data. Jedna inštrukcia sa vykonáva nad viacerými dátami.	Vhodné pre vektory, matice, obraz, GPU/SIMT príbuzný model.
MISD	Multiple Instruction, Single Data. Viac inštrukčných prúdov nad jedným dátovým prúdom.	Zriedkavé; typicky sa uvádzajú systolické polia alebo redundantné spracovanie.
MIMD	Multiple Instruction, Multiple Data. Viac nezávislých inštrukčných aj dátových prúdov.	Multiprocesory, multipočítače, distribuované systémy.

Ako písať porovnanie SIMD vs MIMD

- SIMD: silná synchronizácia, rovnaká operácia nad mnohými dátami, výhodné pri pravidelných dátových štruktúrach.
- MIMD: nezávislé PE, každý môže vykonávať iný program alebo inú časť programu, vhodné pre všeobecné a nepravidelné úlohy.
- SIMD je často jednoduchšie riadiť, MIMD je flexibilnejšie, ale vyžaduje riešiť komunikáciu a synchronizáciu.

4.3 Pamäťové modely

- **Zdieľaná pamäť:** procesory/vlákna pristupujú do spoločného adresného priestoru. Programovanie je prirodzenejšie, ale treba riešiť race conditions, zámky, cache koherenciu a škálovanie.
- **Distribuovaná pamäť:** každý uzol má vlastnú pamäť. Komunikácia prebieha správami. Lepšie škálovanie, ale programátor explicitne rieši výmenu dát.
- **Hybridná pamäť:** kombinácia oboch modelov, napríklad MPI medzi uzlami a OpenMP v rámci uzla.
- **OpenCL/GPU pamäťový model:** global, constant, local/shared a private memory s rozdielnou latenciou, viditeľnosťou a pravidlami synchronizácie.

4.4 PRAM a abstraktné modely

- PRAM je teoretický model paralelného stroja s viacerými procesormi a spoločnou pamäťou.
- Varianty riešia, či viaceré procesory môžu súčasne čítať/zapisovať to isté pamäťové miesto: EREW, CREW, ERCW, CRCW.
- PRAM je užitočný na analýzu algoritmov, ale ideálna realizácia je veľmi náročná: univerzálne prepojenie p procesorov s m pamäťovými miestami vyžaduje zložitú sieť prepínačov.

SKRÁTENÉ
- Detaily PRAM modelov a formálne algoritmické analýzy sú v tomto dokumente zhrnuté. Pri otázke z PRAM si treba pozrieť pôvodné slajdy s variantmi EREW/CREW/ERCW/CRCW a dôvodom náročnosti realizácie.

5. Základné otázky návrhu paralelnej architektúry

Pri návrhu paralelnej architektúry nestačí samotná taxonómia ani samotný programovací model. Dôležité sú architektonické vlastnosti, ktoré ovplyvňujú kompilátor, knižnice, operačný systém, runtime a výsledný výkon aplikácií.

- **Alokácia zdrojov:** koľko PE, aký výkon PE, koľko pamäte, aká sieť, aké cache.
- **Pomenovanie dát:** ako sa identifikujú objekty a adresy v systéme; či je adresný priestor zdieľaný alebo distribuovaný.
- **Komunikácia:** load/store, message passing, kolektívne operácie, explicitná alebo implicitná komunikácia.
- **Synchronizácia:** zámky, bariéry, semafory, atomické operácie, udalosti, podmienkové premenné.
- **Replikácia a konzistencia:** cache, lokálne kópie dát, koherencia a konzistenčný model.
- **Výkon:** latencia, priepustnosť, škálovanie, lokálnosť dát a overhead runtime systému.

Typická odpoveď na návrhovú otázku
- Najprv identifikuj programovací model.
- Potom pomenuj komunikačnú abstrakciu.
- Potom vysvetli, ako sa prenášajú dáta a ako sa synchronizuje.
- Nakoniec uveď výkonnostné dôsledky: latencia, priepustnosť, škálovateľnosť, lokálnosť, réžia.

6. Charakteristické vlastnosti paralelizmu

6.1 Implicitný a explicitný paralelizmus

- **Implicitný paralelizmus:** architektúra alebo kompilátor využíva paralelizmus bez toho, aby ho programátor explicitne riadil; príkladom je pipeline, superskalárne vykonávanie, predikcia vetvenia.
- **Explicitný paralelizmus:** programátor používa vlákna, procesy, OpenMP direktívy, Pthreads, CUDA kernely alebo OpenCL work-itemy.
- **Časový paralelizmus:** prekrývanie fáz v čase, typicky prúdové spracovanie.
- **Priestorový paralelizmus:** viac fyzických jednotiek pracuje súčasne, napríklad viac jadier alebo viac funkčných jednotiek.

6.2 Granularita a profil paralelizmu

- **Jemnozrnný paralelizmus:** malé úlohy, častá synchronizácia. Typicky ILP, vektorové operácie, GPU work-itemy. Výhoda: veľa paralelizmu; nevýhoda: vysoká réžia.
- **Strednozrnný paralelizmus:** vlákna alebo tasky s väčšou jednotkou práce. Typicky OpenMP slučky, Pthreads.
- **Hrubozrnný paralelizmus:** samostatné procesy/uzly, menej synchronizácie, väčšie bloky práce. Typicky distribuované systémy a MIMD.
- **Profil paralelizmu:** rozloženie dostupného paralelizmu počas výpočtu. Ak je paralelizmus dostupný len v malej časti programu, speedup bude obmedzený.

6.3 Škálovateľnosť

- Škálovateľnosť závisí od komunikačnej réžie, uniformity operácií, dátových štruktúr, lokálnosti dát, pomeru výpočtu ku komunikácii a od toho, či algoritmus umožňuje zväčšovať problém s počtom procesorov.
- Modely pracovnej záťaže: fixed-load, fixed-time, fixed-memory a medzistavy medzi nimi.
- Ak rastie počet procesorov, často musí rásť aj veľkosť problému, aby zostala efektívnosť prijateľná.

SKÚŠKOVO DÔLEŽITÉ

- Vedieť vysvetliť rozdiel medzi časovým a priestorovým paralelizmom.
- Vedieť vysvetliť, prečo jemnozrnný paralelizmus nemusí byť rýchlejší, ak synchronizačná réžia prevýši výpočet.
- Vedieť uviesť, že statické a pravidelné algoritmy sú vhodné pre SIMD/pipeline, dynamické a nepravidelné skôr pre MIMD.

7. Prúdové spracovanie, zreťazenie a vektorizácia

7.1 Podstata zreťazenia

Zreťazenie rozkladá výpočet na viac fáz, ktoré sa vykonávajú v samostatných segmentoch. Kým jedna inštrukcia je v neskoršej fáze, ďalšia môže byť v skoršej fáze. Tým sa zvyšuje priepustnosť, nie nevyhnutne latencia jednej úlohy.

Kľúčový rozdiel

- Latencia jednej inštrukcie môže zostať rovnaká alebo byť aj vyššia.
- Priepustnosť po naplnení pipeline sa zvyšuje, pretože výsledky začnú vychádzať pravidelne.

- Typické stupne: F(fetch), D(decode), E(execute), S(store/writeback).
- Ak má výpočet k stupňov a spracúva sa veľa úloh, ideálne zrýchlenie sa približuje ku k.
- Zreťazenie je výhodné pri pravidelnom prúde podobných operácií alebo inštrukcií.

7.2 Základné vzťahy

Vzorce pre pipeline

- Sekvenčný čas pre n úloh pri k fázach približne: $T_s = n \cdot k \cdot T_k$.
- Prúdový čas ideálne: $T_p = (k + n - 1) \cdot T_k$.
- Zrýchlenie: $S = T_s / T_p$; pre $n \to \infty$ sa blíži ku k.
- Pri superskalárnom zreťazení sa uvažuje šírka m a počet inštrukcií začínajúcich v cykle p; výpočet je širší, ale závislosti a konflikty bránia ideálu.

7.3 Problémy pipeline

- **Štrukturálne konflikty:** viaceré fázy potrebujú rovnaký hardvérový zdroj.
- **Dátové závislosti:** inštrukcia potrebuje výsledok predchádzajúcej inštrukcie. Riešenie: forwarding, stall, preusporiadanie.
- **Riadiace závislosti:** vetvenia menia tok programu. Riešenie: predikcia vetvenia, oneskorené vetvenie, špekulatívne vykonávanie.
- **Naplnenie a vyprázdnenie pipeline:** pri krátkych úsekoch sa výhoda znižuje.

7.4 Vektorizácia

- Vektorizácia prevádza opakované skalárne operácie na operácie nad vektormi dát.
- Typické vhodné operácie: nezávislé prvkové sčítanie, násobenie, SAXPY, matice, obrazové filtre.
- Nevhodné sú slučky s dátovou závislosťou medzi iteráciami, nepravidelným prístupom do pamäte alebo vetveniami závislými od dát.

SKRÁTENÉ

- Detailné stavové grafy inicializácie a niektoré transformačné pravidlá vektorizácie sú v dokumente zhrnuté na princíp. Pri skúške s kreslením schémy pipeline si pozri pôvodné slajdy prednášky 5.

8. Prepojovacie siete

Prepojovacia sieť zabezpečuje komunikáciu medzi procesormi a pamäťami alebo medzi procesormi navzájom. Je kľúčová pre škálovanie, pretože aj ideálne paralelný algoritmus môže byť pomalý, ak sieť nedokáže preniesť dáta dostatočne rýchlo.

Pojem	Význam	Typická skúšková pointa
Jednoduchá zdieľaná zbernica	Jedna komunikačná cesta časovo zdieľaná viacerými jednotkami.	Jednoduchá, lacná, nevhodná pre veľa súbežných komunikácií.
Viacnásobná zbernica	Viac zberníc alebo viacprístupové jednotky.	Lepšia priepustnosť, stále obmedzená škálovateľnosť.
Prepojovacia sieť	Všeobecná sieť N vstupov a M výstupov.	Používa sa vo výkonných paralelných systémoch.
Blokujúca sieť	Niektoré požadované spojenie nemožno realizovať pre konflikt s iným spojením.	Treba vedieť definíciu blokovania.
Neblokujúca/prestaviteľná sieť	Spojenia možno realizovať bez blokovania alebo po prestavení ciest.	Vyššia zložitosť a cena.
Permutačná sieť	Realizuje permutácie vstupov na výstupy.	Shuffle, inverse shuffle, exchange/cube sú typické funkcie.

- Komunikačné modely: procesor–pamäť (P–M) a procesor–procesor (P–P).
- Podľa typu prepojenia: jednostranné a obojstranné siete.
- Podľa smerovania: jednosmerné a obojsmerné siete.
- Dôležité metriky: latencia, priepustnosť, bisection bandwidth, počet prepínačov, priemer/diameter siete, blokovanie.

SKRÁTENÉ

- Matematické definície permutačných funkcií, napr. perfect shuffle, inverse shuffle, exchange a cube, sú uvedené iba orientačne. Ak bude skúška obsahovať výpočet permutácie bitového indexu, treba prejsť pôvodnú prednášku 6.

9. ILP architektúry

ILP znamená Instruction-Level Parallelism, teda paralelizmus na úrovni inštrukcií. Cieľom je vykonať viac inštrukcií za jednotku času bez toho, aby programátor explicitne vytváral vlákna.

- **Skalárny procesor:** v ideálnom prípade dokončí jednu jednoduchú inštrukciu za strojový cyklus, využíva pipeline.
- **Superskalárny procesor:** má viac funkčných jednotiek a môže vydávať/vykonávať viac inštrukcií v jednom cykle; paralelizmus hľadá hardvér dynamicky.
- **VLIW:** Very Long Instruction Word. Viac operácií je zakódovaných v jednej dlhej inštrukcii; paralelizmus plánuje kompilátor staticky.
- **RISC/CISC:** RISC má jednoduchšie inštrukcie vhodné pre pipeline; CISC má zložitejšie inštrukcie, často sa interne rozkladajú na mikrooperácie.

VLIW vs superskalárne procesory

- Superskalárny: hardvér zisťuje závislosti, plánuje a vydáva inštrukcie dynamicky. Výhoda: lepšia adaptácia na runtime. Nevýhoda: zložitejší hardvér.
- VLIW: kompilátor pripraví paralelné operácie do jednej dlhej inštrukcie. Výhoda: jednoduchší hardvér. Nevýhoda: závislosť od kompilátora a kompatibility kódu s konkrétnou implementáciou.

9.1 Limity ILP

- Dátové závislosti medzi inštrukciami obmedzujú, čo možno vykonávať súčasne.
- Vetvenia a neznáme adresy pamäte komplikujú plánovanie.
- Pamäťová latencia môže spôsobiť čakanie napriek voľným funkčným jednotkám.
- ILP je jemnozrnný paralelizmus; na väčšie zrýchlenia sa kombinuje s TLP, SIMD alebo GPU paralelizmom.

10. TLP architektúry a multithreading

TLP znamená Thread-Level Parallelism. Využíva paralelizmus medzi vláknami alebo procesmi. Oproti ILP ide o strednozrnný až hrubozrnný paralelizmus.

- **Multiprocesorový čip:** viac jadier na jednom čipe.
- **Multivláknový procesor:** prekrýva vykonávanie inštrukcií rôznych vlákien, aby sa lepšie využili jednotky a zakryli latencie.
- **SMT:** Simultaneous Multithreading umožňuje, aby sa v jednom cykle využívali zdroje procesora inštrukciami z viacerých vlákien.
- **UMA:** Uniform Memory Access, približne rovnaký prístupový čas do pamäte.
- **NUMA:** Non-Uniform Memory Access, prístup do lokálnej pamäte je rýchlejší ako do vzdialenej.
- **MPP:** Massively Parallel Processor alebo multiprocesory s odovzdávaním správ; typicky distribuovanejší model.

10.1 CF, DF a hybridný model

Pojem	Význam	Typická skúšková pointa
CF model	Control Flow: poradie vykonávania riadi programový čítač a riadiace inštrukcie.	Klasický von Neumannov model.
DF model	Data Flow: inštrukcia sa vykoná, keď sú dostupné jej operandy; poradie v programe nie je rozhodujúce.	Vhodné na vyjadrenie prirodzeného paralelizmu závislostného grafu.
Hybrid CF/DF	Vlákna bežia vo vnútri podľa CF, medzi vláknami sa používa dataflow alebo explicitné riadiace operátory.	FORK, JOIN, komunikácia cez registre/pamäť.

SKÚŠKOVO DÔLEŽITÉ

- ILP = paralelizmus inštrukcií v rámci jedného toku.
- TLP = paralelizmus medzi vláknami/procesmi.
- CF sa riadi poradím inštrukcií a programovým čítačom; DF sa riadi dostupnosťou dát.

11. MIMD architektúry a paradigmy

- MIMD architektúry spájajú viac nezávislých PE. Každý môže mať vlastnú kópiu programu a pracovať nad vlastným dátovým prúdom.
- Multiprocesory používajú zdieľanú pamäť. Delia sa na UMA, NUMA a COMA.
- Multipočítače používajú distribuovanú pamäť a odovzdávanie správ.
- SPMD znamená Single Program Multiple Data: každý proces vykonáva rovnaký program, ale nad inou časťou dát; správanie sa môže líšiť podľa ranku/id.
- MPMD znamená Multiple Program Multiple Data: rôzne procesy môžu vykonávať rôzne programy.

11.1 Dataflow, redukčné počítače a špecializované modely

- **Dataflow počítače:** inštrukcie pasívne čakajú na príchod operandov; výpočet je riadený tokom dát. Dobre vyjadrujú paralelizmus, ale praktická implementácia je náročná.
- **Redukčné počítače:** výpočet je chápaný ako redukcia výrazov; prirodzené pri funkcionálnom programovaní.
- **Systolické polia:** pravidelné pole PE, cez ktoré rytmicky prúdia dáta; vhodné pre špecifické algoritmy ako matice a signálové spracovanie.
- **MSIMD/MIMD-SIMD hybridy:** kombinujú viacero SIMD podsystémov alebo MIMD riadenie so SIMD akcelerátormi.

SKRÁTENÉ

- Špecializované historické paradigmy MIMD/SIMD, redukčné počítače a niektoré dataflow varianty sú zhrnuté koncepčne. Pri detailnej otázke je potrebné pozrieť slajdy z prednášky „Paradigmy MIMD“.

12. Programovanie zdieľanej pamäte: Pthreads

Pthreads je POSIX API pre programovanie vlákien v jazyku C/C++ na Unix-like systémoch. V jednom procese môže existovať viac vlákien, ktoré zdieľajú adresný priestor, ale každé má vlastný zásobník a tok riadenia.

Pojem	Význam	Typická skúšková pointa
Proces	Inštancia bežiaceho programu s vlastným adresným priestorom.	Ťažší ako vlákno.
Vlákno	Ľahší tok riadenia v rámci procesu. Vlákna zdieľajú globálne dáta procesu.	Zdieľanie zjednodušuje komunikáciu, ale prináša race conditions.
Race condition	Výsledok závisí od nepredvídateľného poradia prístupu vlákien k zdieľaným dátam.	Typická chyba v paralelnom programe.
Kritická sekcia	Kód, ktorý pristupuje k zdieľanému zdroju a nesmie sa vykonávať súčasne viacerými vláknami.	Chráni sa mutexom, semaforom alebo atomikou.
Mutex	Mutual exclusion lock.	Jeden thread vstúpi, ostatné čakajú.
Bariéra	Miesto, kde všetky vlákna čakajú, kým dorazia ostatné.	Synchronizuje fázy výpočtu.
Podmienková premenná	Mechanizmus čakania na splnenie podmienky.	Používa sa s mutexom, napr. producer-consumer.

12.1 Základné funkcie

```
#include <pthread.h>

pthread_t thread;
pthread_create(&thread, NULL, thread_function, arg);
pthread_join(thread, NULL);

pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);
pthread_mutex_lock(&mutex);
/* critical section */
pthread_mutex_unlock(&mutex);
pthread_mutex_destroy(&mutex);
```

- pthread_create vytvorí nové vlákno a spustí funkciu.
- pthread_join čaká na dokončenie vlákna.
- Argumenty vlákna sa často odovzdávajú cez ukazovateľ; treba si dávať pozor na životnosť dát.
- Globálne premenné sú zdieľané medzi vláknami; lokálne premenné sú na zásobníku konkrétneho vlákna.

12.2 Typické synchronizačné problémy

- **Producer-consumer:** producent vkladá dáta do bufferu, konzument ich odoberá. Treba riešiť prázdny/plný buffer a vzájomné vylúčenie.
- **Read-write locks:** viac čitateľov môže čítať naraz, zapisovateľ potrebuje exkluzívny prístup.
- **Deadlock:** vlákna čakajú navzájom na zdroje a žiadne nepokračuje.
- **Thread safety:** funkcia/knižnica je bezpečná pri súbežnom volaní viacerými vláknami.

Časté chyby v Pthreads

- Zabudnutý pthread_join a predčasné ukončenie main.
- Predanie adresy lokálnej premennej, ktorá sa zmení v slučke.
- Prístup ku globálnej premennej bez mutexu.
- Zamknutie mutexu a návrat z funkcie bez odomknutia.
- Nesprávne použitie condition variable bez while kontroly podmienky.

13. Programovanie zdieľanej pamäte: OpenMP

OpenMP je API pre paralelné programovanie zdieľanej pamäte. V C/C++ sa používa cez direktívy #pragma omp, runtime funkcie a environment premenné. Výhodou je, že mnohé sekvenčné slučky možno paralelizovať s malými zmenami zdrojového kódu.

```
#include <omp.h>

#pragma omp parallel num_threads(4)
{
    int tid = omp_get_thread_num();
    int n   = omp_get_num_threads();
}

#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < n; i++) {
    sum += a[i];
}
```

Pojem	Význam	Typická skúšková pointa
parallel	Vytvorí tím vlákien, ktoré vykonajú blok kódu.	Fork-join model.
parallel for	Rozdelí iterácie for cyklu medzi vlákna.	Najtypickejšia direktíva.
private	Každé vlákno má vlastnú kópiu premennej.	Zabráni nechcenému zdieľaniu.
shared	Premenná je zdieľaná medzi vláknami.	Treba synchronizovať zápisy.
reduction	Bezpečne kombinuje lokálne výsledky do jedného.	Vhodné pre sum, product, min, max.
critical	Len jedno vlákno naraz vykoná blok.	Jednoduché, ale môže byť pomalé.
atomic	Chráni jednoduchú operáciu nad zdieľanou premennou.	Menšia réžia ako critical, ale obmedzené použitie.
barrier	Vlákna čakajú, kým všetky dorazia.	Na konci parallel for býva implicitná bariéra.
schedule	Určuje rozdelenie iterácií: static, dynamic, guided.	Dôležité pri nerovnomernej záťaži.

OpenMP vs Pthreads

- Pthreads je nízkoúrovňové API: explicitne vytváraš vlákna, zámky a synchronizáciu.
- OpenMP je vyššia úroveň: programátor označí paralelné oblasti a slučky, runtime sa stará o tím vlákien a rozdelenie práce.
- Pthreads dáva viac kontroly, OpenMP je často rýchlejšie na implementáciu.

- Nepomiešavať rôzne typy vzájomného vylúčenia pre tú istú kritickú sekciu.
- Vzájomné vylúčenie nemusí garantovať férovosť.
- Vnárané zámky a kritické sekcie môžu spôsobiť deadlock alebo výraznú réžiu.
- Predvolený spôsob rozdelenia iterácií býva blokový, ale OpenMP podporuje viac plánovaní.

14. GPU, GPGPU a CUDA

14.1 Prečo GPU

GPU je vysoko paralelný, multivláknový, many-core procesor s vysokou výpočtovou priepustnosťou a pamäťovou priepustnosťou. Viac tranzistorov je venovaných dátovému spracovaniu ako riadeniu a cache, preto GPU exceluje pri masívne dátovo-paralelných úlohách.

- Vhodné úlohy: matice, vektory, spracovanie obrazu, simulácie, deep learning, veľa nezávislých rovnakých operácií.
- Nevhodné úlohy: silne sekvenčné algoritmy, veľa vetvenia, nepravidelné prístupy do pamäte, častá synchronizácia medzi blokmi.
- Efektívny GPU program musí mať dostatok paralelných vlákien a minimalizovať komunikáciu, synchronizáciu a pomalé prístupy do globálnej pamäte.

14.2 CUDA model

Pojem	Význam	Typická skúšková pointa
Host	CPU časť programu. Spúšťa kernel, alokuje a kopíruje dáta.	Serial control + orchestration.
Device	GPU. Vykonáva kernel vo veľkom počte vlákien.	Masívny dátový paralelizmus.
Kernel	Funkcia vykonaná paralelne mnohými vláknami.	Volá sa z host kódu.
Grid	Všetky vlákna spustené jedným kernel launchom.	Grid obsahuje bloky.
Block	Skupina vlákien, ktoré môžu spolupracovať.	Vlákna v bloku sa môžu synchronizovať a zdieľať shared memory.
Thread	Jedna inštancia vykonania kernelu.	Identifikácia cez threadIdx a blockIdx.
Global memory	Pomalšia pamäť viditeľná pre všetky bloky a host.	Minimalizovať nekoalescentné prístupy.
Shared memory	Rýchla pamäť zdieľaná v rámci bloku.	Kľúčová optimalizácia.
Registers/private	Najrýchlejšie dáta konkrétneho vlákna.	Obmedzený počet, ovplyvňuje occupancy.

```
__global__ void saxpy(float a, float *x, float *y, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) y[i] = a * x[i] + y[i];
}

// host launch
int blockSize = 256;
int gridSize = (n + blockSize - 1) / blockSize;
saxpy<<<gridSize, blockSize>>>(a, d_x, d_y, n);
```

CUDA pamäťové API

- cudaMalloc(): alokácia v device global memory.
- cudaMemcpy(): prenos host ↔ device alebo device ↔ device.
- Kernel launch: <<<grid, block>>> určuje počet blokov a vlákien v bloku.

SKÚŠKOVO DÔLEŽITÉ
- Vlákna v jednom bloku môžu spolupracovať cez shared memory a synchronizáciu.
- Vlákna z rôznych blokov sa počas jedného kernelu bežne nesynchronizujú.
- GPU používa SIMT/SIMD-like model: mnoho vlákien vykonáva podobný kód nad rôznymi dátami.

15. OpenCL

OpenCL je otvorený štandard pre paralelné programovanie heterogénnych zariadení: CPU, GPU, FPGA a iných akcelerátorov. Program má host časť a kernel časť. Host pripraví platformu, zariadenie, kontext, command queue, pamäťové objekty a spúšťa kernely.

Pojem	Význam	Typická skúšková pointa
Platform	Implementácia od výrobcu, ktorá poskytuje OpenCL zariadenia.	Najprv sa zisťujú platformy.
Device	Konkrétne zariadenie: GPU, CPU, FPGA.	Vyberá sa podľa dostupnosti a vlastností.
Context	Prostredie, v ktorom existujú zariadenia a pamäťové objekty.	Väzba host programu na device.
Command queue	Front príkazov: copy, kernel execution, sync.	Môže byť blocking/non-blocking.
Program object	Zostavený zdrojový kód kernelov.	Build pre konkrétne device.
Kernel object	Konkrétna kernel funkcia pripravená na spustenie.	Nastavujú sa argumenty.
NDRange	Indexový priestor výpočtu.	CUDA grid analógia.
Work-item	Jedna inštancia kernelu.	CUDA thread.
Work-group	Skupina work-itemov.	CUDA block.

15.1 Pamäťový model OpenCL

Pojem	Význam	Typická skúšková pointa
Global memory	Prístupná hostu aj work-itemom; veľká, pomalšia.	Používa sa pre hlavné vstupy/výstupy.
Constant memory	Read-only pre work-itemy, zapisuje host.	Vhodné pre konštantné parametre.
Local memory	Zdieľaná v rámci work-group.	Analógia CUDA shared memory.
Private memory	Súkromná pre work-item.	Analógia registrov/lokálnych premenných vlákna.

Terminológia OpenCL vs CUDA
- OpenCL device = GPU
- work-item = thread
- work-group = thread block
- NDRange = grid
- local memory = shared memory
- private memory = local/register memory v CUDA terminológii
- compute unit = multiprocessor

- OpenCL programovanie je prenosnejšie, ale verbóznejšie ako CUDA.
- CUDA je úzko spätá s NVIDIA ekosystémom a má integrovaný C-like model cez NVCC.
- V oboch modeloch je výkon silne určený pamäťovým modelom a organizáciou vlákien.

SKRÁTENÉ
- OpenCL API má veľa detailov: retain/release, event objekty, build logy, subdevices, image/sampler objekty, optimalizácie matíc, histogramov a filtrov. V tomto dokumente je jadro potrebné na skúšku; detailné programátorské použitie je v doplnkovom OpenCL materiáli.

16. GPGPU algoritmické vzory

- **Map:** každý prvok výstupu sa počíta nezávisle z jedného alebo viacerých vstupov. Ideálny prípad pre GPU.
- **Stencil/filter:** každý prvok závisí od okolia, napr. konvolučný filter. Dôležitá je lokálna pamäť/cache.
- **Reduction:** veľa prvkov sa redukuje na jeden výsledok: suma, maximum, minimum, priemer. Robí sa stromovo vo viacerých krokoch.
- **Scan/prefix sum:** výpočet prefixových súčtov; základ pre kompakciu, triedenie a ďalšie paralelné algoritmy.
- **Scatter/gather:** gather číta z viacerých miest, scatter zapisuje na viaceré miesta. Scatter môže spôsobovať konflikty a vyžadovať atomiky.
- **Histogram:** veľa vlákien zvyšuje počítadlá; vznikajú race conditions, riešením sú atomiky alebo lokálne histogramy s následnou redukciou.

Princíp redukcie na GPU
- Nesnažiť sa vypočítať jeden výsledok jedným vláknom.
- Rozdeliť vstup na bloky, redukovať lokálne v blokoch, potom redukovať čiastkové výsledky.
- Operácia musí byť asociatívna alebo približne vhodná na stromové zlučovanie.

17. FPGA

FPGA znamená Field Programmable Gate Array. Je to programovateľné hradlové pole, ktoré umožňuje vytvoriť vlastný digitálny obvod. Na rozdiel od CPU alebo mikrokontroléra samo od seba nevykonáva fixný inštrukčný program; po konfigurácii sa správa ako hardvér navrhnutý pre danú úlohu.

Pojem	Význam	Typická skúšková pointa
LUT	Look-Up Table, základný programovateľný logický prvok.	Implementuje logické funkcie.
Flip-flop/register	Pamäťový prvok pre sekvenčnú logiku.	Umožňuje pipeline a stavové automaty.
BRAM	Bloková pamäť v FPGA.	Lokálne ukladanie dát.
DSP blok	Špecializovaný blok pre násobenie, MAC a signálové operácie.	Výhodné pre numerické algoritmy.
Interconnect	Programovateľné prepojenia medzi blokmi.	Často limitujú frekvenciu a routovanie.
Bitstream	Konfiguračný súbor, ktorý nastaví FPGA.	Výsledok syntézy/place&route.

- Výhody: vysoký paralelizmus na úrovni hardvéru, nízka latencia, energetická efektívnosť pre špecifické úlohy, možnosť pipeline na mieru.
- Nevýhody: náročnejší návrh, dlhší build, obmedzené zdroje, nižšia pracovná frekvencia než CPU/GPU v absolútnych hodnotách, potreba HW myslenia.
- FPGA sa často používa pre streamové spracovanie, sieťové aplikácie, DSP, embedded systémy, akcelerátory a prototypovanie.

18. High-Level Synthesis (HLS)

High-Level Synthesis prevádza algoritmický opis vo vyššej úrovni abstrakcie, typicky C/C++, do RTL návrhu pre FPGA/ASIC. Cieľom je skrátiť návrhový cyklus a umožniť HW/SW partitioning bez ručného písania celého RTL.

- **Klasický RTL flow:** špecifikácia → ručné RTL → logická syntéza → gate-level netlist.
- **HLS flow:** algoritmus/I/O správanie → behavioral synthesis → RTL → logická syntéza.
- **Výhody HLS:** vyššia abstrakcia, menší zdrojový kód, rýchlejší návrh, jednoduchšie modelovanie zložitých slučiek a pamäťových prístupov.
- **Riziko HLS:** ak programátor nerozumie hardvéru, výsledok môže byť neefektívny alebo nesplniť timing/resource constraints.

Pojem	Význam	Typická skúšková pointa
Pipelining	Prekrytie iterácií/fáz v hardvéri.	Kľúčové pre throughput.
Loop unrolling	Rozbalenie slučky na viac paralelných operácií.	Zvyšuje výkon, ale spotrebuje viac zdrojov.
Array partitioning	Rozdelenie poľa na viac pamäťových bánk.	Zvyšuje paralelný prístup do pamäte.
Initiation Interval (II)	Počet cyklov medzi štartom dvoch po sebe idúcich iterácií pipeline.	II=1 je často cieľ.
Latency	Čas do prvého výsledku.	Nie je to to isté ako throughput.
Resource utilization	Spotreba LUT, FF, BRAM, DSP.	Optimalizácia výkon vs zdroje.

SKRÁTENÉ

- Detaily Vivado HLS, konkrétne pragmy a príklady optimalizácie sú v dokumente zhrnuté. Pri praktickej otázke o HLS treba pozrieť doplnkové slajdy s behavioral synthesis, SW/HW partitioning a optimalizačnými direktívami.

19. Rýchle porovnania, ktoré sa oplatí vedieť

- **SISD vs SIMD vs MIMD:** SISD je sekvenčný model; SIMD je jeden inštrukčný prúd nad mnohými dátami; MIMD je viac nezávislých prúdov. SIMD je vhodné pre pravidelné dátové paralelné úlohy, MIMD pre všeobecnejšie úlohy.
- **ILP vs TLP:** ILP hľadá paralelizmus medzi inštrukciami, zvyčajne implicitne hardvérom alebo kompilátorom. TLP používa vlákna/procesy a je viditeľnejšie na úrovni programu.
- **Pipeline vs viac funkčných jednotiek:** Pipeline je časový paralelizmus: fázy sa prekrývajú. Viac FJ je priestorový paralelizmus: viac operácií sa vykonáva naraz.
- **Pthreads vs OpenMP:** Pthreads poskytuje nízkoúrovňovú kontrolu nad vláknami. OpenMP používa direktívy a runtime, je vhodné na rýchlu paralelizáciu slučiek.

- **CUDA vs OpenCL:** CUDA je NVIDIA-oriented model s NVCC a jednoduchším ekosystémom na NVIDIA GPU. OpenCL je otvorený a prenosnejší, ale API je rozsiahlejšie.
- **CPU vs GPU:** CPU je optimalizované na nízku latenciu a komplexné riadenie; GPU na vysokú priepustnosť pri masívnom dátovom paralelizme.
- **GPU vs FPGA:** GPU je programovateľný many-core akcelerátor s vysokou priepustnosťou. FPGA vytvára hardvér na mieru, často s nízkou latenciou a dobrou energetickou efektívnosťou, ale s náročnejším vývojom.
- **Zdieľaná vs distribuovaná pamäť:** Zdieľaná pamäť zjednodušuje prístup k dátam, ale rieši koherenciu a race conditions. Distribuovaná pamäť škáluje lepšie, ale vyžaduje explicitné správy.

20. Skúškové otázky na aktívne opakovanie

1. Definuj paralelný počítač a vysvetli úlohu komunikácie a synchronizácie.
2. Prečo sa paralelné architektúry stali nevyhnutné napriek rastu taktovacej frekvencie?
3. Vysvetli speedup a efektívnosť. Prečo sa reálny speedup líši od ideálneho?
4. Porovnaj SISD, SIMD, MISD a MIMD. Uveď príklady použitia.
5. Porovnaj zdieľanú, distribuovanú a hybridnú pamäť.
6. Čo je PRAM a prečo je jeho ideálna realizácia náročná?
7. Čo znamená komunikačná architektúra a prečo nestačí opis cez ISA?
8. Vysvetli časový a priestorový paralelizmus.
9. Čo je granularita paralelizmu a ako ovplyvňuje réžiu?
10. Vysvetli princíp pipeline a rozdiel medzi latenciou a priepustnosťou.
11. Aké sú hlavné typy hazardov v pipeline?
12. Porovnaj superskalárny a VLIW procesor.
13. Čo je TLP a aký je rozdiel medzi CF a DF modelom?
14. Čo sú UMA, NUMA a COMA?
15. Čo je race condition a ako ju riešiť v Pthreads/OpenMP?
16. Aký je rozdiel medzi mutexom, semaforom, bariérou a condition variable?
17. Vysvetli OpenMP direktívy parallel, parallel for, critical, atomic, barrier a reduction.
18. Vysvetli CUDA pojmy host, device, kernel, grid, block, thread.
19. Prečo sa vlákna z rôznych CUDA blokov nemajú bežne synchronizovať v rámci jedného kernelu?
20. Porovnaj OpenCL work-item/work-group/NDRange s CUDA thread/block/grid.
21. Vysvetli rozdiel medzi global, local/shared, private a constant memory.
22. Čo je reduction a prečo sa na GPU robí vo viacerých krokoch?
23. Čo je FPGA a ako sa líši od CPU/GPU?
24. Čo je HLS a aké sú jeho výhody a riziká?
25. Porovnaj GPU a FPGA ako akcelerátory.

21. Jednostranový ťahák pojmov

Pojem	Význam	Typická skúšková pointa
Speedup	T(1)/T(p)	Čím väčší, tým rýchlejšie oproti sekvenčnému.
Efektívnosť	S(p)/p	Koľko z ideálneho výkonu procesorov sa využilo.
SIMD	Jedna inštrukcia, viac dát	Vektory, matice, obraz, GPU-like.
MIMD	Viac inštrukcií, viac dát	Multiprocesory, multipočítače.

Pipeline	Prekrytie fáz	Zvyšuje throughput.
ILP	Instruction-Level Parallelism	Superskalárne, VLIW, pipeline.
TLP	Thread-Level Parallelism	Vlákna, procesy, multicore.
Race condition	Neurčitý výsledok pri súbežnom prístupe	Riešiť synchronizáciou.
Mutex	Vzájomné vylúčenie	Chráni kritickú sekciu.
Barrier	Čakanie všetkých vlákien	Synchronizácia fáz.
Reduction	Zlučovanie viacerých hodnôt do jednej	Sum, max, min; OpenMP reduction; GPU stromovo.
CUDA block	Skupina threadov	Môžu zdieľať shared memory.
OpenCL work-group	Skupina work-itemov	Analógia CUDA block.
FPGA	Programovateľné hradlové pole	Hardvér na mieru.
HLS	High-Level Synthesis	C/C++ → RTL.

Použité materiály

- 1. Úvod.pdf
- 2. Konvergencia paralelných architektúr.pdf
- 3. Klasifikácia.pdf
- 3. Výpočtové modely.pdf
- 3. Základné otázky návrhu.pdf
- 4. Charakteristické vlastnosti paralelizmu.pdf
- 5. Prudove spracovanie.pdf
- 6. Prepojovacie siete.pdf
- 7. ILP architektúry.pdf
- 8. TLP architektúry.pdf
- 8.-doplnujuce-memoryLogix.pdf
- 9. Pthread.pdf
- 10. OpenMP.pdf
- 11. Graphic Processing Units.pdf
- 11.-doplnujuce-CUDA.pdf
- 11.-doplnujuce-gpgpu.pdf
- 11.-doplnujuce-OpenCL__English.pdf
- 12. FPGA.pdf
- 12. Paradigmy MIMD.pdf
- 13-doplnujuce-High-Level Synthesis.pdf