**1. Teorie grup: Grupoidy, pologrupy, monoidy a grupy. Podgrupy, cyklické grupy a jejich generátory.**

NI-MPI

* **Grupoid** – uspořádaná dvojice
  + = libovolná neprázdná množina
  + = binární operace nad (tzv. kule)
* **Pologrupa** – grupoid, pro který je asociativní
* **Monoid** – pologrupa, ve které existuje neutrální prvek
  + V monoidu existuje právě jeden neutrální prvek
* **Grupa** – monoid, ve kterém ke každému existuje inverzní prvek
  + V grupě má každý prvek právě 1 inverzní prvek
* **Abelovská grupa** – grupa, kde je komutativní

Uzavřená grupoid asociativní pologrupa monoid inverze grupa komutativní Abel. grupa

* **Podgrupa grupy** je je grupa
  + V každé grupě s alespoň 2 prvky existují alespoň 2 podgrupy
    - **Triviální podgrupy**
    - Ostatní podgrupy jsou **vlastní podgrupy**
  + **Průnik** podgrup je podgrupa
  + **Kritérium** podgrupovosti: je podgrupa , právě když
  + **Neutrální prvek** podgrupy je roven neutrálnímu prvku grupy
  + **Inverze** prvku v podgrupě je stejná, jako inverze stejného prvku v grupě
* **Řád grupy** – počet prvků množiny –
  + Podle řádu se dělí na **konečné a nekonečné grupy** (nekonečná )
  + **Lagrangeova věta** – buď podgrupa konečné grupy , potom žád dělí řád
  + **Sylowova věta** – buď grupa konečného řádu a číslo prvočíselný dělitel čísla . Pokud dělí (pro přirozené), pak grupa obsahuje podgrupu řádu
* je podgrupou obsahující
* **Grupa generovaná množinou** – podgrupa grupy
  + Množina je **generující množina** grupy
  + Pro jednoprvkovou množinu zavádíme značení , = generátor
  + je nejmenší podgrupa
  + Všechny prvky lze získat pomocí „**grupového obalu**“
  + **Generátor** = prvek, jehož **mocněním** dostaneme všechny prvky grupy
* Grupa je rovna právě když a jsou nesoudělná čísla
* **Cyklická grupa** – existuje prvek
  + = **generátor** cyklické grupy
  + Buď prvek grupy . Pokud existuje , pak **nejmenší**  s touto vlastností je **řád prvku** . Pokud takové neexistuje, řád prvku je nekonečno
    - Řád se značí
    - Řád prvku je roven řádu grupy
    - je cyklická, právě když , kde je liché prvočíslo a
* **Jak najít všechny generátory**
  + Je-li cyklická grupa řádu a nějaký její generátor, potom je také generátor tehdy, a jen tehdy, když a jsou nesoudělná
  + V cyklické grupě řádu je počet generátorů roven
    - = **Eulerova funkce** – každému přiřazuje počet přirozených čísel menších než , které jsou s ním nesoudělná
    - Takže pro prvočíslo je cyklická grupa řádu a má generátorů
    - Algoritmus na to není
* Libovolná podgrupa cyklické grupy je opět cyklická grupa
* **Malá Fermatova věta** – pro libovolné a libovolné
  + Z důsledku Lagrangeovy věty: , kde je neutrální prvek
* **TODO přidat příklad**

**2. Tělesa a okruhy: Základní definice a vlastnosti. Konečná tělesa. Okruhy polynomů, ireducibilní polynom.**

NI-MPI

* **Okruh** – , kde je neprázdná množina a binární operace na ní a platí:
  1. je **abelovská grupa** = **aditivní grupa** okruhu
     + Neutrální prvek = **nulový prvek** – značí se
     + Inverzní prvek vůči k  značíme
     + Lze definovat odčítání:
  2. je monoid = **multiplikativní monoid** okruhu
     + Je-li komutativní, je komutativní okruh
     + Neutrální prvek = **jednička**, značení 1
  3. Platí **distributivní zákon:**
* Základní vlastnosti okruhu
  + Násobení nulovým prvkem dává nulový prvek
  + Levý i pravý distribuční zákon pro odečítání:
* **Obor integrity** – okruh, ve kterém neexistují dělitelé nuly
  + **Dělitelé nuly** = nenulové prvky
* **Těleso** – okruh kde je abelovská grupa
  + Tuto grupu nazýváme **multiplikativní grupou** tělesa T
  + Pokud pro z tělesa platí , potom , nebo
    - Každé těleso je oborem integrity
* **Zobrazení** z okruhu/tělesa do okruhu/tělesa je **homomorfismus** těchto okruhů/těles, jestliže je homomorfismem příslušných aditivních a multiplikativních grupoidů/grup a platí
  + Je-li navíc **bijekce** (prosté a na), jedná se o **izomorfismus** těchto okruhů/těles
  + Tělesa a nazýváme izomorfní, právě když existuje izomorfismus . V tomto případě je těleso izomorfní s tělesem
* **Konečné těleso** – těleso, které má konečný počet prvků
* **Řád tělesa** = počet prvků tělesa
* Základní příklad konečného tělesa – množina s operacemi **modulo prvočíslo** 
  + **-** aditivní grupa , multiplikativní grupa
    - - řád
      * Každý nenulový prvek je její **generátor**
      * Je grupou i pro neprvočíselné
    - - řád – není prvočíslo
      * Je **cyklická**
      * Počet generátorů závisí na řádu, je roven
* **Řád konečného tělesa** musí být mocnina prvočísla , kde je prvočíslo a je kladné celé číslo
  + Všechna tělesa řádu jsou **navzájem izomorfní**
* **Galois field** – těleso s prvky -
  + Prvočíslo = charakteristika tělesa
  + – aditivní grupa
    - Řád
    - Neutrální prvek
    - Pro není cyklická
  + – multiplikativní grupa
    - Řád
    - Neutrální prvek:
    - Inverzi lze nalézt pro každý prvek s REA v polynomiálním čase
    - Je vždy cyklická
* **Polynom nad okruhem**
  + Nad okruhem
  + … koeficienty polynomu
  + … **formální proměnná** polynomu
  + Pokud pro existuje , pak největší z  = **stupeň polynomu** , značeno
  + … nulový polynom – nedefinovaný stupeň
  + Abychom mohli dělat operace s polynomy, potřebujeme je umět s jejich koeficienty – lze vybudovat okruh polynomů nad libovolným okruhem (i tělesem)
* **Okruh polynomů** – množina všech polynomů nad okruhem spolu s operacemi **sčítání** a **násobení** definovanými předpisy

kde , tvoří okruh polynomů nad okruhem -

* + **Násobení** polynomů: buď těleso a nenulové polynomy. Platí
  + **Dělení** polynomů: buď těleso a nenulové polynomy. Pak existují jednoznačně určené polynomy takové, že

kde je buď nulový, nebo má stupeň ostře menší než stupeň

* + **Bézoutova rovnost** pro polynomy: Buďte a nenulové polynomy nad tělesem . Pak existují polynomy tak, že:
  + Buď těleso a polynom stupně . Prvek je **kořen polynomu** právě tehdy, když:

kde je stupně

* **Ireducibilní polynom** – buď stupně alespoň 1. Řekneme, že je ireducibilní nad okruhem , jestliže :
  + Mějme celé a prvočíslo . Označme počet monických polynomů stupně ireducibilních nad . Potom
    - **Monický polynom** – má za koeficient u nejvyšší mocniny jedničku
    - – **Möbiova funkce** definovaná pro celé :

**3. Funkce více proměnných: gradient, Hessián, definitnost matic, extrémy funkcí více proměnných bez omezení a s rovnostními omezeními.**

NI-MPI

**4. Integrál funkcí více proměnných (Darbouxova konstrukce).**

NI-MPI

**5. Numerická matematika: reprezentace čísel v počítači, chyby vznikající při výpočtech s pohyblivou řádovou čárkou, podmíněnost a stabilita numerických algoritmů.**

NI-MPI

**6. Testování statistických hypotéz. T-testy, testy nezávislosti, testy dobré shody.**

NI-VSM

**7. Základy teorie informace a kódování, entropie.**

NI-VSM

**Entropie**

* **Entropie** = „míra neuspořádanosti“
* Uvažujeme **diskrétní náhodné veličiny**
  + Množina hodnot
  + Pravděpodobnostní funkce v  …
  + Pravděpodobnostní rozdělení …
  + Argument pravděpodobnostní funkce – o kterou veličinu se jedná
* **Entropie diskrétní náhodné veličiny**:
  + Log o základu 2,
  + Entropie závisí pouze na rozdělení veličiny
  + Je invariantní vůči transformacím:
* **Jednotky entropie** – báze logaritmu – , označuje jednotky entropie
  + … bit
  + … digit
  + … nat
  + Přechod mezi bázemi:
* Entropie **jako očekávaná míra neurčitosti**
  + lze chápat jako **střední hodnota**:
    - = **vlastní informace** = míra neurčitosti

entropie je očekávaná míra neurčitosti

* + Míra neurčitosti je vždy nezáporná a pro jisté jevy 0
  + Méně pravděpodobný jev vyšší míra neurčitosti
  + se při pozorování nezávislých jevů sčítá
* **Vlastnosti entropie**
  + **Nezápornost** entropie:
  + Entropie je konkávní funkcí rozdělení
  + V deterministických případech je entropie 0
  + Maximální rovnoměrné rozdělení – nejvyšší neurčitost
* **Sdružená entropie**  diskrétních náhodných veličin se sdruženým rozdělením :
  + Sdružená entropie diskrétního náhodného vektoru se sdruženým rozdělením :
  + Alternativně
* **Podmíněná entropie**  diskrétních náhodných veličin se sdruženým rozdělením :
  + Alternativně
    - Z

* **Řetězové pravidlo**:

… určuje, která část informace je ve veličině navíc oproti tomu, co je v

* **Relativní entropie** = Kullback-Leiblerova vzdálenost :
  + Pokud :
  + Je to „vzdálenost“ – nezáporná a 0, jen pokud
    - Ale ne opravdová, neplatí ani trojúhelníková nerovnost
    - Alternativně:
* **Vzájemná informace** :

= relativní entropie skutečného sdruženého rozdělení a rozdělení nezáv. veličin se stejnými marginálami:

* + Symetrie:
  + Z nezápornosti relativní entropie:
* **Vztah vzájemné informace a entropie**
  + Odvození přes věty o logaritmech
  + ; – vlastní informace
* **Informační nerovnost**: možná rozdělení :
  + Rovnost pouze pokud
  + Důsledky:
    - **Nezápornost vzájemné informace** – pro dvě d.n.v. :
      * Pokud **rovnost, pak jsou závislé**
      * je číselná charakteristika sdruženého rozdělení, která je schopná poznat nezávislost
    - **Maximalizace entropie** – pro d.n.v. s hodnotami z :
      * … počet prvků množiny – rovnost, pokud rovnoměrné rozdělení
      * Entropie je maximalizována **rovnoměrným rozdělením**
    - **Podmiňování redukuje entropii**:
      * **Rovnost, pokud jsou a nezávislé**
      * „informace neublíží“ – znalost n.v. může v průměru pouze redukovat neurč. v
      * Pouze v průměru, samotné může být pro nějaké větší než , ale:

**Teorie kódování**

* Jak zapsat zdrojovou zprávu, tak, aby následný přenos byl co nejefektivnější
* **D-ární abeceda** – abeceda obsahující přenositelných symbolů
* **Zpráva** je posloupnost znaků z
* Chceme co nejkratší zakódovanou zprávu
* Zobrazení z množiny do množiny konečných řetězců symbolů -ární abecedy nazýváme **kód** diskrétní náhodné veličiny
  + Obraz = **kódové slovo** příslušného prvku a jeho délku značíme
    - …řetězec symbolů z  délky
* **Střední délka** kódu náhodné veličiny s rozdělením :
  + …**délka kódového slova** příslušejícího k prvku :

* **Typy kódů**
  + **Nesingulární kód** d.n.v. – pokud je prosté zobrazení
    - Dostačující pro schopnost rekonstruovat z kódových slov jednotlivé hodnoty
    - Není dostačující pro dekódování posloupnosti hodnot (celých zpráv)
  + **Jednoznačně dekódovatelný kód** – pokud je nesingulární
    - = rozšíření kódu – zobrazení do :
      * Zápis jednotlivých kódových slov po sobě
    - Jsme schopni jednoznačně dekódovat libovolnou přijatou zprávu
  + Kód je **instantní** (prexifový), pokud žádné kódové slovo není prefixem jiného kódového slova
  + Obsah obrázku kruh, text, Písmo, diagram

    Popis byl vytvořen automatickyHierarchie kódů:
* **Kraftova nerovnost** – pro libovolný instantní kód nad -ární abecedou musí délky kódových slov splnit nerovnost:

Navíc, ke každé -tici délek, které splní tuto nerovnost, existuje instantní kód s kódovými slovy těchto délek

* + Pro jednoznačně dekódovatelné kódy analogicky (McMillanova věta)
    - Ke každému jednoznačně dekódovatelnému kódu lze sestrojit instantní kód, který má stejně dlouhá kódová slova
* **Optimální kódy**
  + **Střední délka** instantního -árního kódu d.n.v. je:
    - Rovnost, právě když
  + **Optimální kód** = kód o nejmenší střední délce
  + Uvažujme optimální instantní kód
    - **Optimálním kódem se od dolní meze dané entropií můžeme vzdálit maximálně o 1**
* **Huffmanovo kódování**
  + Algoritmus na sestrojení binárního Huffmanova kódu:

1. Spojíme 2 nejmíň pravděpodobné hodnoty nové rozdělení s o 1 menším počtem hodnot
2. Opakujeme, dokud nezůstane 1 hodnota prázdný řetěz jako kódové slovo
3. Zpětným chodem zkonstruujeme kódová slova všech původních hodnot
4. Pro hodnotu , která vznikla spojením a vytvoříme kódové slovo méně pravděpodobné hodnoty připojením 1 za kódové slovo a analogicky kódové slovo více pravděpodobné hodnoty připojením 0 za

* Tzn. Pokud a , tak a
  + Huffmanův kód je **optimální** – je-li Huffmanův kód a libovolný jednoznačně dekódovatelný kód, potom
  + Algoritmus sestrojení je hladový algoritmus, který **lokálně agreguje** 2 nejméně pravděpodobné hodnoty

**8. Markovské řetězce s diskrétním časem. Jejich limitní vlastnosti.**

NI-VSM

**9. Markovské řetězce se spojitým časem. Souvislost s Markovskými řetezci s diskrétním časem a s Poissonovým procesem.**

NI-VSM

**10. Systémy hromadné obsluhy a jejich limitní vlastnosti. Souvislost s Markovskými řetězci se spojitým časem.**

NI-VSM

**11. Význam tříd NP a NPH pro praktické výpočty.**

NI-KOP

* Třída P – rozhodovací problém patří do třídy P, jestliže pro něj existuje program pro deterministický Turingův stroj, který jej řeší v čase , kde je velikost instance a konečné číslo
* PSPACE – existuje program pro deterministický Turingův stroj, který jej řeší v **paměti** , kde je velikost instance a konečné číslo
* EXPTIME – existuje program pro deterministický Turingův stroj, který jej řeší v čase , kde je polynom ve velikosti instance

**12. Experimentální vyhodnocení algoritmů, zejména randomizovaných.**

NI-KOP

**13. Princip lokálních heuristik, pojem globálního a lokálního minima, obrana před uváznutím v lokálním minimu.**

NI-KOP

**14. Princip genetických algoritmů, význam selekčního tlaku pro jejich funkci.**

NI-KOP

**15. Princip simulovaného ochlazování, význam parametrů a způsoby jejich řízení.**

NI-KOP

**16. Výkonnostní měřítka paralelních algoritmů, PRAM model, APRAM model, škálovatelnost.**

NI-PDP

**Výkonnostní měřítka paralelních algoritmů**

* **Paralelní časová složitost** závisí nejen na n, ale i na počtu procesorů/jader p
  + p = # procesorů = # jader = # vláken
  + **Paralelní čas**  = čas, který uplynul od začátku paralelního výpočtu do okamžiku, kdy poslední (nejpomalejší) procesor skončil výpočet
  + závisí na architektuře paralelního výpočtu hodnocení par. Algoritmu musí vždy brát v úvahu architekturu počítače
  + je meřen čítáním:
    - Výpočetních kroků – aritmetické, logické, paměťové operace
    - Komunikačních kroků – přenos a výměna dat mezi procesory
* **Paralelní cena –** 
  + Většinou statická alokace výpočetních jader
  + Výpočet začíná vytvořením vláken a ty jsou použita k výpočtu až do konce, i když některá mohou být nějakou dobu neaktivní (idle)
  + Měřením kvality je součin procesory – čas = paralelní cena
  + Sekvenční složitost =
  + **Cenová optimalita** – paralelní algoritmus má optimální cenu, pokud
    - Cena je optimální právě tehdy když
* **Paralelní zrychlení** , nebo asymptoticky
  + Paralelní zrychlení je lineární, právě když (
  + **Lineární zrychlení** = nejvyšší cíl paralelního programu – jestliže stoupne -krát, chceme, aby klesnul -krát – obtížně splnitelné
  + **Superlineární zrychlení** – výjimečně dosažitelné

1. Sekvenční algoritmus je paměťově náročnější, než kapacita paměti a souhrnná kapacita pamětí paralelního systému je dostatečná a při paralelním výpočtu ušetříme swapování mezi hlavní pamětí a diskem

* Sekvenční algoritmus znevýhodněn tím, že běží za jiných HW podmínek než paralelní

1. Anomálie při prohledávání kombinatorického stavového prostoru

* **Paralelní efektivnost** relativní vytížení jader dedikovaných paralelnímu výpočtu během výpočtu
  + Vždy (komunikační a synchronizační režie)
  + zrychlení na jádro
  + Konstanta
  + Paralelní algoritmus má konstantní efektivnost, jestliže
* **Paralelní optimalita výkonnosti** – z definic plyne, že paralelní algoritmus je cenově optimální má lineární zrychlení má konstantní efektivnost

**PRAM model**

* **PRAM** = paralelní RAM = výpočetní model
  + Množina p procesorů
    - 1 procesor vlastní lokální (soukromá) paměť + index i
  + M sdílených paměťových buněk (pole)
  + Každý p může přistoupit do jakékoliv buňky sdílené paměti v O(1) čase
  + Řešení konfliktů – explicitní ošetření
* **PRAM algoritmus**
  + Vstup = n položek v (obvykle prvních) n buňkách sdílené paměti
  + Výstup = n’ položek v n’ buňkách sdílené paměti
* Procesy provádí synchronně 3 typy instrukcí:
  + **READ** – čtení sdílené buňky
  + **LOCAL** – lokální operace
  + **WRITE** – zápis do buňky sdílené paměti
* Komunikace procesorů = READ/WRITE sdílené buňky
* PRAM algoritmus lze zapsat regulárními výrazy
* **Jednotkový** model R/L/W trvá čas 1
* **Globální** model L trvá 1 a R/W trvají konstantní čas
* Ošetřování konfliktů při přístupech do sdílené paměti
  + **EREW** – Exclusive Read Exclusive Write
    - Žádné 2 procesory nesmějí číst nebo zapisobat tutéž sdílenou pam. Buňku současně
  + **CREW** – Concurrent Read Exclusive Write
    - Současná čtení 1 b. povolena, ale v 1 okamžiku může jen 1 proces zkoušet zapsat do dané buňky
  + **CRCW** – Concurrent Read Concurrent Write
    - Povoleny současné čtení a zápisy téže buňky
    - **Priority**-CRCW-PRAM
      * Procesy mají pevné priority – dokončení zápisu povoleno procesu s nejvyšší prioritou
    - **Arbitrary** CRCW-PRAM
      * Dokončení zápisu povoleno náhodně vybranému procesu (algoritmus nesmí činit žádné předpoklady, který proces byl vybrán)
    - **Common**-CRCW-PRAM
      * Všechny procesy smí dokončit zápis, pokud jsou všechny zapisované hodnoty stejné
        + Každý a. musí zajistit splnění podmínky
        + Jinak a. není správný a stav PRAM není definován

**APRAM model**

* **APRAM** = asynchronní PRAM
  + Procesy pracují asynchronně, neexistují centrální hodiny
  + READ, WRITE, LOCAL jako PRAM
  + Nutná explicitní synchronizace – **bariérová synchronizace**
  + Doba přístupu do sdílené paměti není jednotková
* **APRAM výpočet** = posloupnost globálních fází, ve kterých procesory pracují asynchronně, oddělených bariérovou synchronizací
* Dva + procesory nemohou přistupovat do téže buňky v téže globální fázi, pokud jeden z nich do ní zapisuje
* Výkonnostní parametry:
  + Lokální operace – 1
  + Globální READ/WRITE: d
  + K po sobě jdoucích globálních R/W: d+k-1
  + Bariérová synchronizace: B(p)
* 2 možné implementace bariéry
  + **Centrální čítač**
    - Inicializovaný na 0 a na příchozí fázi, procesy přistupují ve vzájemném vyloučení

1. Proces dorazí k bariéře, zkontroluje, zda je v příchozí fázi a inkrementuje čítač
2. Je-li čítač , proces se deaktivuje
3. Jinak nastaví bariéru do odchozí fáze a aktivuje ostatní procesy
4. Poslední aktivovaný proces nastaví bariéru do příchozí fáze
   * **Binární redukční strom**
5. Každý proces dorazí k bariéře a zkontroluje, zda je v příchozí fázi
6. Čeká, až skončí redukce v jeho podstromu
7. Po jejím skončení pošle signál rodiči
8. Kořen stromu počká na redukci z obou podstromů a přespne do odchozí fáz

**Škálovatelnost**

* **Amdahlův zákon saturace paralelizace**
  + Každý sekvenční algoritmus A s časem nad daty o velikosti n se proporčně skládá z

1. Inherentně sekvenčního podílu , který může provést pouze 1 vlákno
2. Paralelizovatelného podílu
   * Nechť A je paralelizován pro pevné n pomocí vláken
   * Pak pro zrychlení A platí při vláknech ideálně:

Obsah obrázku Písmo, číslo, řada/pruh, text

Popis byl vytvořen automaticky

* + Nezávisle na tom, kolik vláken bylo použito, nemůže zrychlení přesáhnout
  + Po jisté hranici nemá přidávání procesů už smysl, bo pro něn není dost paralelní práce
  + Problém fixní velikosti poskytuje omezené množiství paralelismu a tudíž při provedení i omezuje použitelný počet paralelních vláken/jader
* **Gustafsonův zákon**
  + S rostoucím p máme úměrně navyšovat i velikost problému n
  + Pak sekvenční část trvá konstantní čas nezávisle na (V/V operace, inicializace), kdežto inherentně paralelní část bude lineárně škálovat s  v čase
  + pro monotónně rostoucí
  + **Paralelní škálovatelnost** – schopnost par. Počítače se zvětšit, pokud narůstá velikost řešeného problému
    - **Silná** – schopnost p. a. pro fixní dosáhnout lineárního zrychlení s rostoucím
      * Měří pokles efektivnosti, pokud roste a se nemění
    - **Slabá** – definuje, jak se mění par. Čas s  pro fixní
      * Měří růst takový, že při rostoucím zůstává efektivnost stejná
    - **Škálovatelnost** = schopnost p. a. držet paralelní optimalitu, pokud oba a rostou/klesají
  + **Izoefektivní funkce**
    - = asymptoticky minimální funkce taková, že



* + - = asymptoticky maximální funkce taková, že



* + Z Amdahlova zákonu vyplývá:



* + - Abychom si udrželi konstantní efektivnost, musí být procesorů alespoň
  + Z Gustafsonova zákonu vyplývá, že když velikost problému roste s  vztahem



efektivnost nebude klesat

**17. Programování nad sdílenou pamětí, programový model OpenMP, datový a funkční paralelismus, synchronizace vláken, vícevláknové algoritmy (násobení polynomů, násobení matic, řazení).**

NI-PDP

**OpenMP**

* **OpenMP** – explicitní model paralelního výpočtu, kdymá programátor plnou kontrolu a zodpovědnost za paralelní výpočet
* **Paralelní regiony** = části původně sekvenčního kódu
  + V nich pomocí fork-join mechanismu vytvářena, prováděna a ukončována paralelní vlákna

Obsah obrázku diagram, řada/pruh, Plán

Popis byl vytvořen automaticky

* Mimo par. regiony pouze 1 hlavní (master) vlákno
* Podpora pro iterační i funkční model paralelismu
* Pomocí OpenMP direktiv v kódu paralelní regiony, ve kterých bude výpočet prováděn více paralelně běžícími vlákny nebo paralelně běžícími úlohami, kdy je každá úloha prováděna 1 vláknem
* Zákaz skákat z paralelního regionu ven či dovnitř
* Kompilace s -fopenmp
* Tvorba paralelních regionů – **direktiva parallel**
  + Možné klauzule direktivy:
    - **If(podmínka)**: podmínka paralelizace regionu
    - **Num\_threads(výraz)**: počet vláken v paralelním regionu
    - **Vlastnosti(seznam proměnných)**: OpenMP vlastnosti proměnných v paralelním regionu
  + Na konci p. r. je implicitní bariéra
  + Po jejím provedení jsou nová vlákna ukončena a dál pokračuje jen hlavní vlákno 0
  + Pokud je 1 vlákno předčasně ukončeno, jsou ukončena všechna vlákna i celý program
* Vlastnosti proměnných v paralelním regionu
  + **Shared** – daná skalární proměnná (ne pole, ne struktura) je sdílená všemi vlákny
  + **Private** – daná proměnná je lokální ve vláknech – každé vlákno má nezávislou minimalizovanou instanci této proměnné
  + **Firstprivate** – proměnná je lokální ve vlákně, každé vlákno ji má inicializovanou na hodnotu, kterou měla před vstupem do p. r.
  + **Lastprivate** (pouze v paralelních cyklech) – p. je lokální ve vláknech, ale hodnota ze sekvenčně poslední iterace se po skončení p. cyklu překopíruje do proměnné hlavního vlákna procesu
  + **Default** – určuje, jakou z předchozích vlastností budou mít implicitně všechny proměnné použité v paralelním regionu
  + **Reduction** – určuje, že daná sdílená proměnná je lokálně nakopírovaná do každého vlákna a že po skončení par. Regionu se všechny lokální instance této proměnné zredukují pomocí zadaného redukčního operátoru a výsledek bude zapsán do původní sdílené proměnné
    - Musí to být skalární proměnná
    - Redukční operátory: +,\*,-,&,^,|,&&,||
    - Nelze kombinovat s direktivou task
  + **Threadprivate** – def. Globální platnost hodnost lokálních proměnných vláken v rámci celého programu napříč všemi paralelními regiony
  + Počet vláken ve všech regionech musí být stejný, proměnné si „drží“ hodnoty při přestupech mezi p. r.

**Datový a funkčí paralelismus**

* **Direktiva for** – přidělení jednotlivých iterací for cyklu uvnitř par. Regionu jednotlivým vláknům
* Na konci par. Cyklu implicitně bariéra
* Možné klauzule:
  + **Schedule**(): upřesňuje způsob přidělení iterací cyklu vláknům
  + **Collapse**(): upřesňuje paralelizaci vnořených cyklů (implicitně for jen na nejvyšší úrovni)
  + **Ordered**(): pořadí provádění iterací je stejné jako při sekvenčním provádění
  + **Nowait**(): vlákna po dokončení svých iterací nečekají na bariéře
* **Klauzule schedule –** schedule(typ) schedule(typ, chunk-size)
  + Typy klauzulí:
    - **Static** – buď jsou vláknům přiděleny staticky cyklicky bloky (=chunks) o velikosti chunk-size, nebo se přidělí rovnoměrně (n/p)
    - **Dynamic** – dynamicky přiřazuje chunky po sobě jdoucích iterací velikosti chunk-size nebo 1
    - **Guided** – vláknům dynamicky přiděleny bloky x iterací, kde
    - **Runtime** – způsob přiřazení zvolen v okamžiku spuštění dle systémové proměnné OMP\_SCHEDULE
    - Auto – přidělení it. Necháno kompilátoru/běhovému prostředí
  + **Efektivita**:
    - Schedule(**static**[,k])
      * Nejmenší režie
      * Rovnoměrné rozdělení iterací
      * Ideální, pokud mají všechny iterace stejnou výpočetní dobu
      * K ovlivňuje promíchání iterací
    - Schedule(**dynamic**[,k])
      * Vyšší režie kvůli synchronizaci
      * Vyšší k snižuje režii
      * Výhodné při kolísavé době iterací
    - Schedule(**guided**[,k])
      * Vyšší režie (synchr.)
      * Vyšší k režii snižuje
      * Výhodné při postupně rostoucí době itercí
* **Paralelizace 2-úrovňového for cyklu**
  + **Statické přidělení** vláken (pro jednoduchost)

# parallel for nebo #parallel for collapse(2)

For(…)

For(…)

Funkce()

* + Paralelizace **pouze vnitřního cyklu**
    - For(…)

#...parallel for

For(…)

Funkce()

* + - # … parallel

For(…)

#...for

For(…)

Funkce()

Nutné, je-li vnitřní smyčka datově závislá na vnější smyčce

* **Task** = úloha
* Podporuje složitější funkční paralelismus s větší režií – vhodné i pro rekurzivní algoritmy (zapouzdření kódu i dat)
* **Přidělování úloh** – typ producent-konzument
  + Vlákna jsou producenti i konzumenti
* **Úloha** = jednotka par. Výpočtu, obsahuje:
  + Ukazatel na začátek svého kódu (k provedení)
  + Vstupní data
  + Dat. Strukturu, do které vloží svůj identifikátor vlákno, jakmile danou úlohu začne provádět jeho konzument (=vlastnické vlákno)
* **#pragma omp task** způsobí:
  + Vlákno – producent vygeneruje novou úlohu a vloží ji do zásobárny úloh (=task pool)
  + Úloha čeká, než ji volné vlákno – konzument vyzvedne a provede
* **Podmíněné spouštění par. Úloh**
  + If(…) – efektivní řízení task par. Rekurzivních kódů, kdy rekurze závisí na splnění podmínky
    - Splněno – synovská úloha do task poolu
    - Nesplněno – pozastavení rodičovské úlohy a odložení do zásobárny úloh, ihned provedení nové synovské úlohy, po dokončení vyzvednutí rodiče a dokončení
  + #pragma omp **taskwait**
    - Rodičovská úloha řeká na dokončení všech synovských úloh
    - Stromová rekurze
* Volání task direktivy musí být uvnitř paralelního regionu
* Rekurzi startuje jediné vlákno
  + #pragma omp parallel num\_threads(…){

# pragma omp single

Funkce()

}

**Synchronizace vláken**

* **Synchronizační direktivy**
  + **Barrier** – místo, kam par. Vlákna daného p. r. musí dorazit a čekat na ostatní
  + **Master** – daný blok kódu smí provést pouze hlavní vlákno
  + **Single** – daný blok kódu smí provést pouze 1 libovolné vlákno
  + **Critical** – vytvoření kritické sekve
  + **Atomic** – operace nad paměťovou buňkou bude provedena jednovláknově a nepřerušitelně
  + **Flush** – propsání aktuálních hodnot daných sdílených proměnných do sdílené paměti
  + **Taskwait** – synchronizace synovských úloh s rodičovskou v task paralelismu
* **Bariéra a serializace** v par. Regionu
  + # pragma omp **single** – následující blok se smí provést pouze jednou – ostatní vlákna čekají na implicitní bariéře za single blokem
  + # pragma omp **master** – následující blok smí provést pouze hlavní vlákno – ostatní pokračují hned kódem, který je za tím
  + # pragma omp **barrier** – synchronizační bod, vlákna uspávána a probouzena, až dorazí všechna vlákna
    - Implicitně na konci par. Regionu a single
* **Kritická sekce**
  + Jedna/více částí kódu par. Regionu, které lze v 1 okamžiku provádět pouze 1 vláknem
  + Direktiva # pragma omp critical – anonymní kritická sekce
  + Několik kritických sekcí – vzájemné vyloučení vstupu vláken platí globálně pro všechny její výskyty
  + Direktiva #pragma omp critical name – pojmenovaná krit. sekce taky může být víckrát, platí to samé
* **Direktiva atomic** a její použití
  + Přístup do pam. Místa se skalárním datovým typem (integer, floating-point, … ) bude atomická operace
    - Nepřerušitelná R/W/RMW
    - Bude deterministický výsledek
  + Read, write, update, capture
  + Inkrementace - #... atomic update
  + Capture – rozšiřuje update o získání hodnoty dané pr. Před/po modifikaci
* Uživatelsky řízené předčasné ukončení par. Regionu – **direktiva cancel**
  + Provedením vydá vlákno ostatním signál k ukončení – přejde na bariéru
  + Další vlákna, která později narazí na cancel provedou totéž
  + Vlákna, která už poslední volání cancel minuly, standardně dokončují
  + # pragma omp cancel construct[if(expr)]
    - Construct [parallel, for, taskgroup, sections]

**Vícevláknové algoritmy**

* **Prohledávání kombinatorického SP**
  + NPH úloha najít určitý 1 stav ve velkém SP
  + Vstupní proměnné, stavové proměnné, výstupní proměnné, omezení, optimalizační kritérium
  + Rozhodovací vs konstruktivní vs enumerační
  + **SB-DFS** – přípustný koncový stav bez optimalizace
  + **BB-DFS** – diskrétní optimalizační problém
    - Přípustný koncový stav s max./min. cenou
  + **PP-DFS** – prohledávání v iteracích se zvyšující se hloubkou SP (např. lineární prohlubování)
    - BB-DFS do hloubky L, pokud nenašlo řešení, prohloubí se
* **Paralelní algoritmy pro PKPS**
  + Čas. Složitost PKPS je superpolynomiální
  + Paralelní prohledávání může mít anomální chování
  + Základní podmínka úspěšného par. PKPS:
    - Jádra by měla být pokud možno stále vytížena prohl. pokud možno disjunktních částí SP
* **Statické rozdělení SP**
  + Nerozlišujeme mezi procesem a vláknem
  + P CPU jader, každé p\_i provádí v 1 okamžiku jedno vlákno
  + Základní postup statického rozdělení výpočtu:
    - Master vlákno – sekvenční BFS – vygeneruje odlišných stavových prostorů s cca stejným počtem nastavených stavových proměnných
    - Prohledávání stavových podprostorů přiděleno vláknům
    - Každé vlákno (včetně hlavního) provede sekvenční DFS přiděleného SPp pomocí lokálního zásobníku
    - Výsledky lokálních PKSP předají hlavnímu vláknu – globální řešení
* **Problémy efektivnosti** statického rozdělení SP
  + P jader by mělo mít podobný výkon a parametry paměti
    - Stejně rozsáhlé podprostory se stejnou sekvenční časovou náročností
  + ALE: navracení (=backtracking) je silně datově závislé
    - Výpočet vláken se může dost lišit
    - Některá pak budou neúčinná – neefektivní
* Obsah obrázku kresba, diagram, Dětské kresby, skica

  Popis byl vytvořen automaticky**Statické rozdělení SP a anomální chování**
  + V případě prohledávání celého SP rozděleného pouze staticky může vzniknout anomální chování:
    - 4-vláknové řešení vpravo pomalejší než 2-vláknové vlevo
  + 2. příklad:Obsah obrázku řada/pruh, trojúhelník, diagram

    Popis byl vytvořen automaticky
    - V případě FSB-DFS:
      * Paralelní DFS s 2 vlákny trvá stejně jako se 4 vlákny
      * Anomálie – přidáním jader může DFS
        + Superlineárně zrychlit
        + Zpomalit
  + Pro efektivní PKSP:
    - Jemnější statická dekompozice v modelu dynamického Master-Slave
    - Doplnění o dynamické vyvažování zátěže
* **Dynamické vyvažování zátěže**
  + generuje podprostory a přiřadí je vláknům
  + – DFS pomocí lokálního zásobníku (= je aktivní)
  + Aktivní vyprázdní zásobník, ale nenajde řešení
    - Stává se nečinným, ale žádá jiná vlákna o přidělení neprozkoumaných částí jejich SP
  + se stane dárcem - = příjemce
  + Půlení zásobníku (rozdělování na k částí – požadavků) - částí
    - Neexp. stavy blízko dna/vrcholu zásobníku skrývají pravděpodobně větší/menší části SP
    - Položky nad řeznou výškou H se nepředávají
* Paralelní algoritmus dyn. **Master-Slave DFS**
  + Hlavní vlákno = Master, dalších vláken = Slaves
  + M – sekvenční BFS podprostory pro vlákna S
  + M pošle každému S 1 podprostor z množiny
  + S po přijetí podprostoru jede sekvenční DFS, nikdy se nevrací za počáteční stav svého zásobníku
  + S nepspěšně ukončí lok. PKSP požádá M o další podprostor (definován lok. Zásobníkem)
    - M má nepřidělené podprostory přidělí S další lokální PKSP
    - M odpoví negativně požádá S o ukončení aktivity
  + FSB-DFS – S nalezne řešení informuje M M oznámí všem S konec
* **Klasifikace efektivně paralelizovatelných algoritmů**
  + **Výpočetně intenzivní algoritmy –** čas procesoru strávený výpočtem nad daty je větší než čas nutný na přesun dat z paměti do CPU (PKSP pro NPH úlohy, …)
  + **Paměťově intenzivní algoritmy –** čas procesoru strávený nad výpočtem je menší, než čas nutný na přesun dat z paměti do CPU
    - Počet výpočetních operací na přenesený bajt/prvek je příliš malý
    - Typicky a. s lineární výpočetní složitostí, kde data přen. Z paměti do CPU použita jen k-krát, kde je malá konstanta
    - Skalární součin, dynamické programování, Fourierovy transformace
  + O smysluplnosti paralelizace rozhoduje typ úlohy
    - Nutná podmínka = teoretické zrychlení
    - Rozhodující je řád výpočetní složitosti nebo multiplikativní konstanta (u lin. Složitosti)
* **Zdroje neefektivity** OpenMP kódů
  + **Nevyvázěná výpočetní zátěž** pro jednotlivá vlákna – bariéra čekající vlákna nevyužitá jádra
  + Příliš **těsná synchronizace –** velký počet bariér/krit. sekcí
  + **Omezený paralelismus –** # iterací for < # vláken
  + **Vysoká režie** správy vláken – častá tvorba/zánik, schedule(dynamic)
  + **Významná sekvenční část –** z Amdahlova zákonu
  + **Neefektivní využívání keš** paměti – falešné sdílení, častý zápis
* **Falešné sdílení –** různá vlákna zapisují na různé adresy, které jsou ale natolik blízké, že jsou namapovány do stejného bloku keš paměti
  + U datového paralelismu typické
  + Zabránění vede na protichůdný požadavek, než je požadavek přístupu se třídou 1 u jednovláknových aplikací
* **Snížení dopadu falešného sdílení**
  + Vhodnější rozdělení iterací cyklu nad dostatečně velkým polem mezi vlákna je blokově rovnoměrné – schedule(static)
  + Vhodná **chunk-size** při statickém/dynamickém přidělování bloků iterací vláknům
    - Pole A začíná na adrese dělitelné cache\_line\_size, čili pole A je v paměti zarovnáno stejně jako bloky keše
  + Umělé navýšení velikosti zapisované datové struktury připojením jalové výplně (dummy data)
    - Např. každý prvek pole navýšen na velikost bloku keše + podm. Zarovnání
    - Dobré pro malá sdílená pole, kde má každé vlákno vyhrazené místo pro zápis svého lok. Výsledku velikosti blok keše

**18. Programování nad distribuovanou pamětí, programový model MPI (vícevláknové procesy, komunikátory, 2-bodové blokující a neblokující komunikační operace, kolektivní operace), paralelní násobení hustých matic, paralelní mocninná metoda.**

NI-PDP

* **MPI** = systém zasílání zpráv mezi procesy
* Komunikace procesů/vláken
  + **OpenMP –** pomocí čtení/zápisů z/do sdílené paměti, podpora pro redukci
  + **MPI –** procesy nesdílí paměť – komunikace zasíláním zpráv, všechny proměnné privátní
    - Redukce pro všechny procesy najednou:
* **Využití sdílené paměti**:
  + **Jen MPI** – na každém jádru 1 či několik MPI procesů – nedělí se o vlákna
  + **MPI + OpenMP** – výpočetní uzel MPI proces(y) pomocí OpenMP dělení na několik vláken, běžících na jádrech
  + **Hybrid** – 1 OpenMP vlákno na jádro
* **Kombinace MPI + OpenMP**
  + Inicializace Vrací v proměnné zaručenou míru spolupráce MPI s vlákny
  + Požadovaná míra spolupráce MPI s vlákny:
    - – pouze MPI, procesy se nedělí na vlákna
    - – vícevláknové procesy s omezením, že pouze hlavní vlákno může zavolat funkce MPI = jednoportový model
    - – vícevláknové procesy s om., že v daném okamžiku smí funkce MPI volat pouze 1 vlákno (volání MPI funkcí je kritická sekce) = jednoportový model
    - – vícevláknové procesy, kde všechna vlákna mohou volat funkce MPI bez omezení = všeportový model
* **Komunikátor** – určuje množinu procesů, v rámci níž probíhá komunikace
* **Intra-komunikátor** – asociovaný s konkrétní skupinou procesů
* – předdef. Intra-kom. Pro všechny MPI procesy
* **Inter-komunikátor** – 2 různé skupiny procesů
* – číslo procesu, – počet procesů
* **Komuniační MPI operace**: 2-bodové (komunikace mezi 2 procesy), kolektivní – (komunikace mezi všemi p.)
* **Základní 2-bodová kom.** – zdrojový p. - – určí cílový p., cílový p. - – určí zdrojový p.
* **Blokující komunikační operace** – příslučná MPI funkce je ukončena teprve po dosažení určitého stavu dané komunikační operace
  + Buf – ukazatel na posílaná data
  + Count – počet posílaných položek
  + Datatype – dat. Typ posílaných dat
  + Dest – číslo cíl. Procesu
  + Tag – značka procesu
  + Comm – MPI komunikátor
  + Source – číslo zdrojového procesu
  + Status – ukazatel na stavový objekt
  + Zbytek stejně jako u Send
* **Typ přenášených dat**
  + Parametr datatype – typ
  + Základní datové typy - , , …
  + Složitější -
* **Množství** přenášených dat – Lze najednou posílat víc prvků, ale stejného dat. Typu a uložené za sebou v paměti, Parametr count
* **Zdrojový a cílový proces** – Cíl – parametr dest, Zdroj – parametr source
  + Přijetí od 1 konkrétního zdroje od libovolného zdroje -
* **Značky přenášených dat – Tag** – rozeznání sémantického významu zpráv
  + Přijetí konkrétní značky libovolné značky -
* **Stavový objekt** – Proměnná typu
  + Můžeme ignorovat –
  + Struktura s položkami:
    - – číslo zdroj. Procesu zprávy
    - – značka přijaté zprávy
  + Pomocí velikost zprávy
* je blokující – ukončena až když lze modifikovat vstupní buffer
  + Realizuje standardní mód – návrat z funkce nastane, když jsou data:
    - Odeslána cílovému procesu
    - Překopírována do dočasného systémového bufferu pro pozdější odeslání
  + Kvůli odesílání je to nelokální operace
* – realizuje Buffered mode, návrat zaručeně nezávisí na připravenosti příjemce přijímat data, lokální operace
  + Pokud příjem nebyl iniciován, MPI musí odesílaná data uložit do bufferu, který si musí uživatel předtím připravit pomocí
* – Synchronous mode – není návrat, dokud není inicializace přijetí dat, nelokální operace
* – Ready mode – pokud při volání není init příjmu, vrátí chybu, nelokální operace
* **Standardní mód –** MPI rozhodne, jestli použít Buffered/Synchronous – Uživatel to neovládá
  + MPI – stanovisko, že korektní MPI program není na systémovém bufrování závislý
* Tyhle sendy jsou blokující ve smyslu, že po návratu z nich můžeme buffer odesílaných dat přepsat
* Recv je blokující ve smyslu, že po jejich ukončení jsou přijatá data uložená v bufferu a lze je číst
* Neblokující funkce **, , ,**  iniciují odeslání dat a skončí
  + může začít, až když příjemce iniciuje příjem, ostatní libovolně
* Buffer vstupních dat nelze modifikovat, dokud není dokončení komunikační operace explicitně otestováno
* Neblokující funkce iniciuje příjem dat
* Buffer není možné použít, dokud není dokončení operace příjmu dat explicitně otestováno
* Všechny neblokující funkce mají dodatečný parametr – ukazatel na proměnnou typu
  + Vstupní arg. Funkcí, které slouží pro testování/čekání na dokončení těchto komunikačních operací
  + **Testování dokončení** -
    - Neblokující, vrátí okamžitě nebo chybu
  + **Čekání na dokončení** -
    - Blokující, vrátí až tehdy, když jsou data skutečně obdržena
* U neblokujícího příjmu **stavový objekt** až z funkcí Test/Wait, NE z Irecv
  + Parametry a stavový objekt
* Neblokující operace důležité, bo umožňují překrývání volání komunikačních párů – není nutná serializace
* **/** – dokončení libovolné operace
* **/** – dokončení všech operací z množiny
* **Komunikační módy** neblokujících operací
  + Vrací okamžitě nezávisle na splnění dané podmínky
  + Na splnění podm. závisí návrat z funkcí čekání na dokončení neblokujících operací , …
* **,**  – testují příchod zprávy, aniž by zpráva byla přijata
* – nebolokující lokální funkce
  + , pokud existuje zpráva, kterou lze přijmout a která odpovídá parametrům
  + Pak vrátí argumentu stejnou hodnotu, jakou by vrátila operace
  + Jinak vrátí a je nedefinován
* může být a může být
* Sondovaná zpráva nemusí být přijata po té, co byla sondována a danou zprávu lze tedy sond. opakovaně
* – blokující nelokální funkce
  + Vrátí až poté, co existuje zpráva, kterou lze přijmout a která odpovídá parametrům a ve výstupním argumentu vrátí stejnou hodnotu, jakou by vrátila operace
* **Požadavky na implementaci** a – měly by garantovan následující:
  + Je-li zavolán jedním procesem a jiný proces zavolá s kompatibilními parametry, pak se volání úspěšně vrátí KROMĚ případů, kdy zprávu přijme konkurenční funkce
  + Pokud proces aktivně čeká pomocí a odpovídající zpráva byla vyslána, pak volání v konečném čase vrátí , pokud
    - Kompat. zprávu nepřijme konkurenční provedená jiným vl. téhož procesu
    - Taková zpráva nebyla sond. konkurenční operací provedenou jiným vl. téhož proc.
* Volání Iprobe/Probe det. zprávu, kterou by byla přij. ve stejném místě vol. f. se stejnými arg.
* Ve vícevláknových procesech je seznam příchozích zpráv sdílen a může docházet k soupeření vláken o přijetí zpráv kompatibilních s těmi, které vysondovaly přerchozími voláními Probe/Iprobe
* **Sondování s rezervací pro budoucí –** pro zajištění větší korektnosti a efektivnosti soupeření
  + Neblokující
    - Oproti Iprobe vrátí v případě, že zpráva existuje, v hodnotě argumentu message hangle na vysondovanou zprávu
  + Message je vstupem volání funkce
    - Před návr. z volání se message handle reset. Na
    - Volání s takovouto hodnotou argumentu message nic nepřijme
* Využití funkcí pro **testování příchodu zpráv**
  + Příchod „volitelných“ zpráv – ředčasné uk. výpočtu při nalezení optimálního řešení jiným procesem
  + Příjem zprávy neznámé velikosti – zjištění velikosti zprávy pomocí a , alokace bufferu, příjem dat pomocí
* MPI neposkytuje mechanismy pro řešení chyb komunikačního systému
* Chyby způsobené voláním MPI funkce s chybným argumentem, nedostatek zdrojů, …
* **Návrátová hodnota MPI funkce** – úspěch/neúspěch
  + Úspěch **,** neúspěch chybový kód
* **Chybový kód** = základ pro obsluhu chyby (=error handler) dané MPI funkce, která se při výskytu chyby zavolá před návratem
* 3 předdefinované obsluhy chyb:
  + **–** chyba násilně ukončen celý pr., k návratu chyb. funkce nedojde
    - Procesy interně zavolají
    - Implicitně asociovaná s
  + neukončí program, vrátí chybový kód funkce
    - Stav MPI výpočtu není po chybě MPI standardem definován
    - Pro diagnostiku stavu a výpis chybového hlášení
  + **–** násilné ukončení procesů spoj. s daným komunikátorem, ale ne všech
* Funkce pro **vytvoření kódu obsluhy chyby**, její navázání na komunikátory, testování vazeb a jejich zrušení:

uživatelsky naprogramované volání obsluhy

**Násobení hustých matic**

* **Násobení hustých matic**: Předpokládám klasický školní algoritmus na násobení matic a blokově-šachovnicové mapování matic
* **Naivní algoritmus**: Každý procesor potřebuje odpovídající submatice pomocí AAG
  + Na závěr se provede lokální vynásobení, časová náročnost: , paměťově neefektivní (nevleze se to do pamě: jednoho procesoru)
* Obsah obrázku text, snímek obrazovky, design, typografie

  Popis byl vytvořen automaticky**Cannonův systolický algoritmus**:
  + Přesouvá iteraPvně a synchronně submatice tak, že vždy můžu násobit
  + SystemaPcky rotuji i. sloupec a j. řádek o i/j pozic pomocí cyklický posun MPI\_Sendrecv
  + Vždy přičtu výsledek a orotuji o 1 víc, na vhodné topologii (všeportová WH Q log p ) současně
  + Výsledná složitost
* **Foxův algoritmus** – Broadcast-Multiply-Roll:
  + Nejprve se submatice pošle všem procesorům v rámci řádku i (OAB: MPI\_Bcast)
  + Obsah obrázku text, snímek obrazovky, Písmo, design

    Popis byl vytvořen automatickyNásledně se provede lokální násobení přijatých submatic
  + Na závěr se provede rotace ve sloupci k o jednu pozici nahoru (cyklický posun)
  + Časová šložitost, škálovatelnost podobná jako u Cannonova algoritmu

**Paralelní mocninná metoda**

* **Mocninná metoda**: hledá iterativně největší vlastní číslo, vhodné pro velmi řídkou matici, využití: Google PageRank
* **Algoritmus**:
  + Vytvořím nenulový počáteční vektor (typicky x = (1,1,1,1,...))
  + Vynásobím A vektorem x, vznikne vektor y = Ax (nějaký algoritmus pro řídkou MVM)
  + Spočteme normu 𝛼 vektoru y, nahradíme x normalizovaným y = x/𝛼 (paralelní redukce)
  + Vyhodnotíme kritérium konvergence, pokud není splněno, pokračujeme dál
* **Implementace v MPI**:
  + Předpokládáme řídkou matici, předem neurčená struktura
  + Procesory provádí lokální násobení, dílčí výsledky redukují (MPI\_Allreduce)
* **Náhodné mapování matice**:
  + Každý procesor potřebuje celý vektor x a vytvoří libovolný prvek vektoru y
  + Po provedení algoritmu má každý proces celý vektor y a 𝛼
  + Složitost: paměť, kde
* **Řádkové mapování matice**:
  + Každý procesor potřebuje celý vektor x, ale vektor y si již můžou rovnoměrně rozdělit
  + Matice rozdělena do p horizontálních pásů velikosti n/p
  + Získání vektoru x, složení vektoru y:
  + Rychlejší (nepotřebujeme kopírovat y do x),
  + Složitost:
* **Šachovnicové mapování matice**:
  + Procesy tvoří virtuální 2D mřížku
  + Každý procesor potřebuje jen část vektoru x (menší paměťové nároky)
  + Po lokálních MVM mají procesy příspěvek k čásP y a provedeme redukci
  + Nejpřirozenější mapování na diagonální procesy
  + Složitost:
* **Šachovnicové mapování** – rozdělení komunikátorů:
  + Potřebujeme provést paralelní redukci jen ve virtuálních řádcích matice procesů
  + – rozdělí komunikátor podle pole „barev“ (řádková souřadnice)
  + Redukci tak můžeme provádět ve všech řádcích nezávisle na sobě
  + Potřebujeme ale taky komunikátor pro diagonální procesy, nejefektivnější časově i paměťově

**19. Přímé ortogonální a hyperkubické propojovací sítě paralelních počítačů (definice, vlastnosti, vnořování).**

NI-PDP

* **Topologie** G n : nekonečná množina instancí jednoho typu grafu s parametrem n (dimenze)
* **Inkrementálně / částečně škálovatelná topologie**: definována pro všechna / některá n
* **Řídká topologie**: stupně uzlů jsou omezeny konstantou |𝐸(𝐺$ )| = 𝑂(|𝑉(𝐺$ )|)
* **Hustá topologie**: stupně uzlů jsou rostoucí funkcí n: |𝐸(𝐺$ )| = 𝜔(|𝑉(𝐺$ )|)
* **Hierarchicky rekurzivní topologie**: instance menších dimenzí jsou podgrafy instancí větších  
  dimenzí (příklad: n-rozměrná mřížka je složena kartézským součinem menších mřížek)
* **Vzdálenost uzlu u a v:** délka nejkratší cesty mezi u a v
* **Excentricita uzlu u**: nejdelší vzdálenost mezi uzlem u a libovolným uzlem v grafu
* **Průměr grafu diam(G):** největší excentricita libovolného uzlu (největší vzdálenost)
* **Poloměr grafu r(G):** nejmenší excentricita libovolného uzlu (nejkratší z nejdelších cest)
* **Regulární graf**: stupeň každého uzlu (počet sousedů) je roven nějaké konstantě k
* **Uzlový (hranový) řez 𝜿**(𝑮): množina uzlů (hran), jejichž odebráním se rozpojí souvislý graf
* **Uzlová (hranová) souvislost 𝝀(**𝑮): velikost minimálního uzlového (hranového) řezu  
  = minimální počet uzlů / hran, jejichž odebráním zruším souvislost grafu
  + Platí: pro libovolný graf uzlová souvislost ≤ hranová souvislost ≤ minimální stupeň v grafu
  + Rovnost nastává při optimální souvislosti
  + (pokud má uzel stupeň 2, tak odebráním těch 2 hran už to rozpojím, někdy stačí i méně)
* **Bisekční šířka bw** e (G): velikost nejmenšího hranového řezu grafu na 2 poloviny
  + (kolik hran musím odebrat, abych dostal dvě přibližně stejné poloviny)
* **Bipartitní graf**: graf, kde existuje obarvení vrcholů dvěma barvami tak, že koncové vrcholy každé hrany mají odlišnou barvu (rozdělím na dvě poloviny, hrany jen mezi nimi)
* **Optimální topologie**:
  + Konstantní stupeň uzlu (umožní univerzální, levné směrovače a řídkou topologii)
  + Malý průměr a malá průměrná vzdálenost (snižuje komunikační zpoždění)
  + Spodní mez průměru optimální topologie: Ω(log 𝑁) (každá řídká topologie s logaritmickým průměrem je op:mální)
* **Požadavky na topologie**:
  + **Hierarchická rekurzivita**: umožní snadnější škálovatelnost
  + **Bisekční šířka**: vhodná pro paralelní binární rozděl-a-panuj algoritmy
  + Vysoká souvislost: redundantní cesty v případě výpadků nebo přegžení uzlů / linek,
  + Možnost rozdělit velké zprávy po paralelních disjunktních cestách
* **Kartézský součin**:
  + (vezmu kartézský součin množin vrcholů)
  + Obsah obrázku text, Písmo, typografie, kaligrafie

    Popis byl vytvořen automaticky
  + (Hrana vede tehdy, pokud vedla původně v jednom nebo druhém grafu)

Obsah obrázku řada/pruh, diagram

Popis byl vytvořen automaticky

* + komutativní a asociativní operace (nezáleží na pořadí a uzávorkování)
* **Uzlová symetrie**:
  + pro každé dva vrcholy existuje automorfismus (prosté, na) 𝑓 takový, že 𝑓(𝑢. ) = 𝑢0
  + (Jsem schopný na sebe namapovat libovolné dva vrcholy, je jedno, z jakého úhlu se dívám)
* Věta: každý graf vzniklý kartézským součinem uzlově symetrických grafů je uzlově symetrický
  + (Idea důkazu: pokud byly původní uzlově symetrické, existují automorfismy f1 , f2 , které mi mapují na sebe uzly v G1 , G2 . Pokud provedu kartézský součin, hrana vede pouze pokud vedly v původním G1 nebo G2 , takže to namapování bude sedět)
* Každý uzlově symetrický graf je regulární (jinak by mi neseděly stupně uzlů)
* **Vzdálenosti** v uzlově symetrickém grafu
  + Průměr je stejný jako poloměr, jelikož všechny uzly mají stejnou excentricitu (vychází z vlastnosn automorfismu)
* **Optimální topologie** a **uzlová symetrie**:
  + Umožní snazší návrh paralelních a komunikačních algoritmů – je jedno, odkud začnu

N-rozměrná hyperkrychle

* **Ortogonální topologie**:

Obsah obrázku řada/pruh, diagram, Vykreslený graf, Písmo

Popis byl vytvořen automaticky

* + Sestavena kartézským součinem cest délky 1
  + Vrcholy: {0, 1}n – binární n-Pce
  + Hrana: vede mezi vrcholy, pokud se liší právě v jednom bitu (0000 -> 0001, 0010, 0100, 1000)
  + Počet vrcholů:
  + Počet hran:
  + Průměr: n (nejdále jsou například 0000 a 1111)
  + Stupeň: n (n negací = n hran)
  + Bisekční šířka:  (chci rozseknout na dvě n-1 krychle, ty mají 2n-1 vrcholů ke spojení)
* Vlastnosti:
  + Regulární graf (vše stupeň n)
  + Hustá topologie (logaritmický stupeň uzlů – n = log(2 n )
  + Optimální souvislost (uzlová i hranová souvislost je rovna stupni uzlu – n)
  + Bisekční šířka je největší možná (2n-1 = 2n /2 = N/2)
  + Vyvážený bipartitní graf (skládá se ze dvou podkrychlí, ty stačí obarvit)
  + Hamiltonovský graf (každý n-bitový Grayův kód je hamiltonovská kružnice Qn )
* **Hamiltonovský graf**: lze projít tak, že je každý vrchol navšgven právě jednou
* **Grayův kód**: kód, který v každém kroku invertuje jeden bit, „zrcadlový kód“
  + (0000 -> 0001 -> 0011 -> 0010 -> 0110 -> 0111 -> 0101 -> 0100 ...)
  + Uzlově symetrická (kartézský součin P2 = Q 1 , která je uzlově symetrická)
  + Existuje 2$ ⋅ 𝑛! různých automorfismů, k! různých nejkratších cest ve vzdálenosP k
  + Počet uzlů ve vzdálenosP i od zadaného uzlu: ( $2 ) – průměrná vzdálenost je horní celá část
* **Směrování**: minimální e-cube směrování: zprava doleva (podle dimenzí)
* Využití:
  + Není řídký graf, škálovatelné jen po mocninách 2 (částečně škálovatelná)
  + Základní testovací topologie, maximálně použitá jako podsíť

N-rozměrná mřížka

* **Ortogonální topologie:**
  + Sestavena kartézským součinem mřížek nižších dimenzí: 𝑀(𝑧. , 𝑧0 , ... , 𝑧$ ) = 𝑀(𝑧. ) × ⋯ 𝑀(𝑧$ )

Obsah obrázku řada/pruh, diagram, čtverec, Barevnost

Popis byl vytvořen automaticky

* + **Vrcholy**: (𝑎. , 𝑎 0 , ... , 𝑎 $ ), kde ai je vždy od 0 do příslušného z n
  + **Hrany**: hrana vede mezi vrcholy, pokud se liší v právě jedné dim o jedničku (012 -> 022, 112, 002, 011)



(V každé dimenzi je zi – 1 hran, a rozprostře se to po ploše rovné součinu ostatních dimenzí)

* Bisekční šířka: Ω(𝑁/max (𝑧2 )) (chci říznout přes nejdelší hranu)
* v případě, že je délka nejdelší hrany sudá, bez Ω
* **Vlastnosti:**
  + Není regulární graf (pokud z i není stejné pro každé i) => není uzlově symetrická
  + Nejčastější 2-D (rovinné) a 3-D (kubické) mřížky
  + **Hierarchicky rekurzivní** (krychle dána součinem mřížek nižších dimenzí)
  + **Optimální souvislost** (uzlová i hranová souvislost rovna stupni uzlu)
  + **Bipartitní** graf (skládá se ze dvou podmřížek, ty stačí obarvit), ale ne nutně vyvážený
  + Hamiltonovský graf, pokud alespoň jedna hrana sudou délku
* **Směrování:** dimenzně uspořádané směrování (XY, XYZ směrování v 2D, 3D mřížkách)

N-rozměrný toroid

* **Ortogonální topologie**
  + Sestaven kartézským součinem toroidů nižších
  + Dimenzí: 𝐾(𝑧. , 𝑧0 , ... , 𝑧$ ) = 𝐾(𝑧. ) × ⋯ × 𝐾(𝑧$ )

Obsah obrázku diagram, řada/pruh, snímek obrazovky, Paralelní

Popis byl vytvořen automaticky

* + Vrcholy: (𝑎. , 𝑎 0 , ... , 𝑎 $ ), kde ai je vždy od 0 do z n
  + Hrany: hrana vede mezi vrcholy, pokud se jejich
  + XOR liší v právě jedné dimenzi o jedničku (012 -> 022, 112, 002, 011, 312, 010)

Obsah obrázku text, Písmo, snímek obrazovky, řada/pruh

Popis byl vytvořen automaticky

* + (V podstatě mřížka, akorát spojím i krajní hrany)
  + (jako mřížka, ale ještě musím seknout toroidové hrany)
* **Vlastnosti:**
  + **Regulární a uzlově symetrický** graf (kartézský součin kružnic – ty jsou uzlově symetrické)
  + Průměr je poloviční vůči stejně velké mřížce, souvislost a bisekční šířka je dvojnásobná
  + **Hierarchicky rekurzivní** (na rozdíl od mřížek ale nelze rozložit na stejnorozměrné toroidy)
  + Hamiltonovský a vyvážený bipartitní graf
* **Směrování:** podobně jako v mřížkách / hyperkrychlích, postupně podle dimenzí (XY, XYZ)
* **Využití**: nejúspěšnější komerční topologie pro masivně paralelní počítače
* Porovnání: hyperkrychle nejdražší, toroid uprostřed, mřížka nejlevnější

**20. Paralelní algoritmy pro redukci, prefixový součet a segmentový prefixový součet na PRAM, v ortogonálních, hyperkubických a obecných topologiích, aplikace.**

NI-PDP