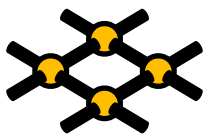
Obecnější aplikované úvahy o paralelizaci

Petr Holub
hopet@ics.muni.cz



Laboratoř pokročilých síťových technologií

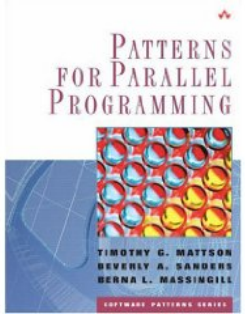
PV197
2010–11–10

Motivace

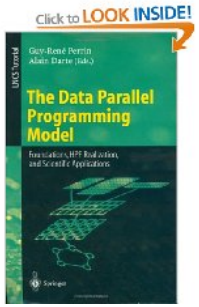
- SIMD modely paralelního programování jsou tu od 60. let
- paměťové hierarchie jsou tu od vzniku paralelních počítačů
- CUDA je v podstatě variace na SIMD model s konkrétním paměťovým modelem a omezenými možnostmi synchronizace

Literatura

- Mattson T. G., Sanders B. A., Massingill B. L., *Patterns for Parallel Programming* Addison-Weslay 2004.

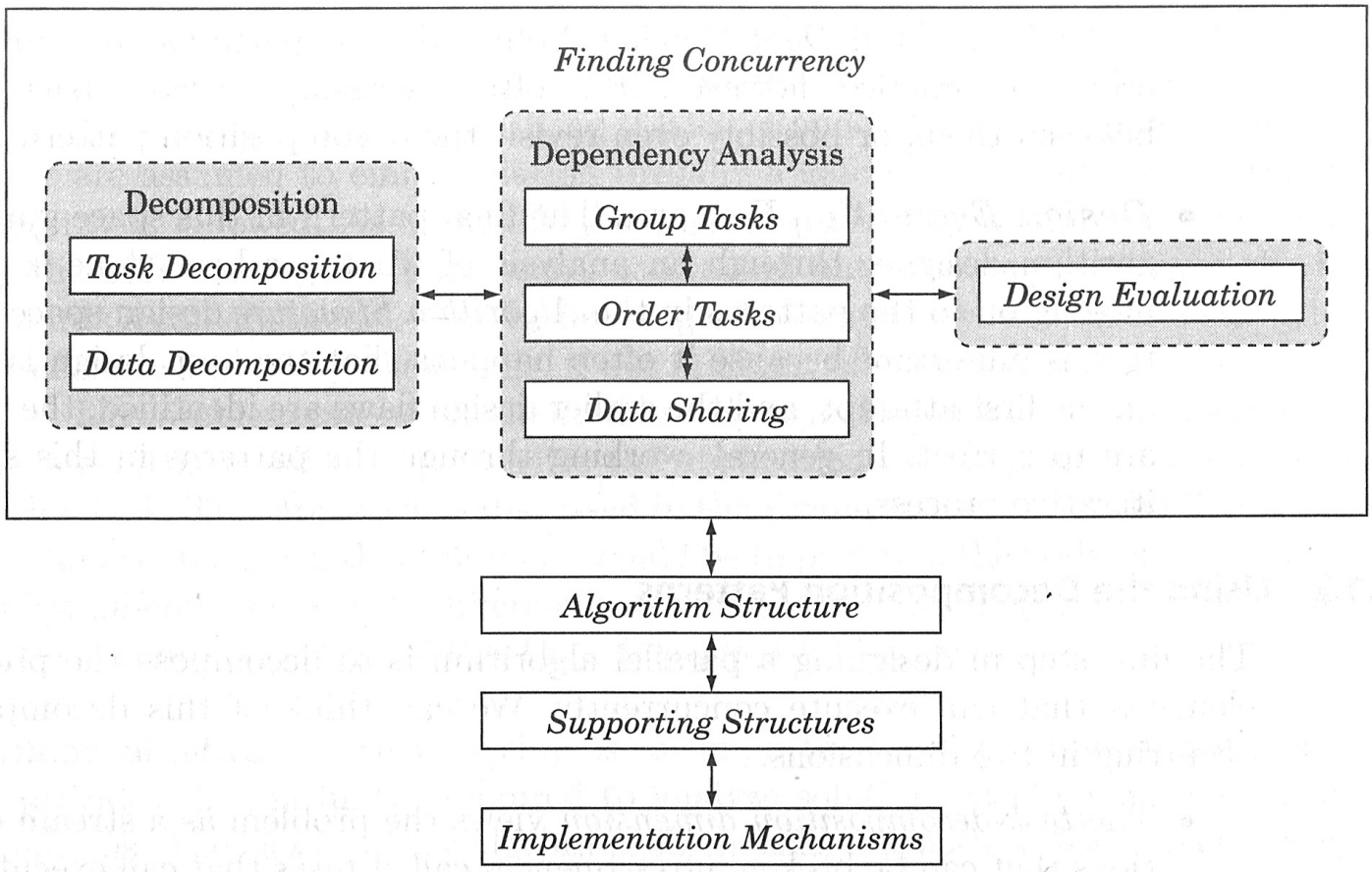


- Perrin G.-R., Darte A., *The Data Parallel Programming Model: Foundations, HPF Realization and Scientific Applications*, Lecture Notes in Computer Science, Springer 1996.



Přehled přednášky

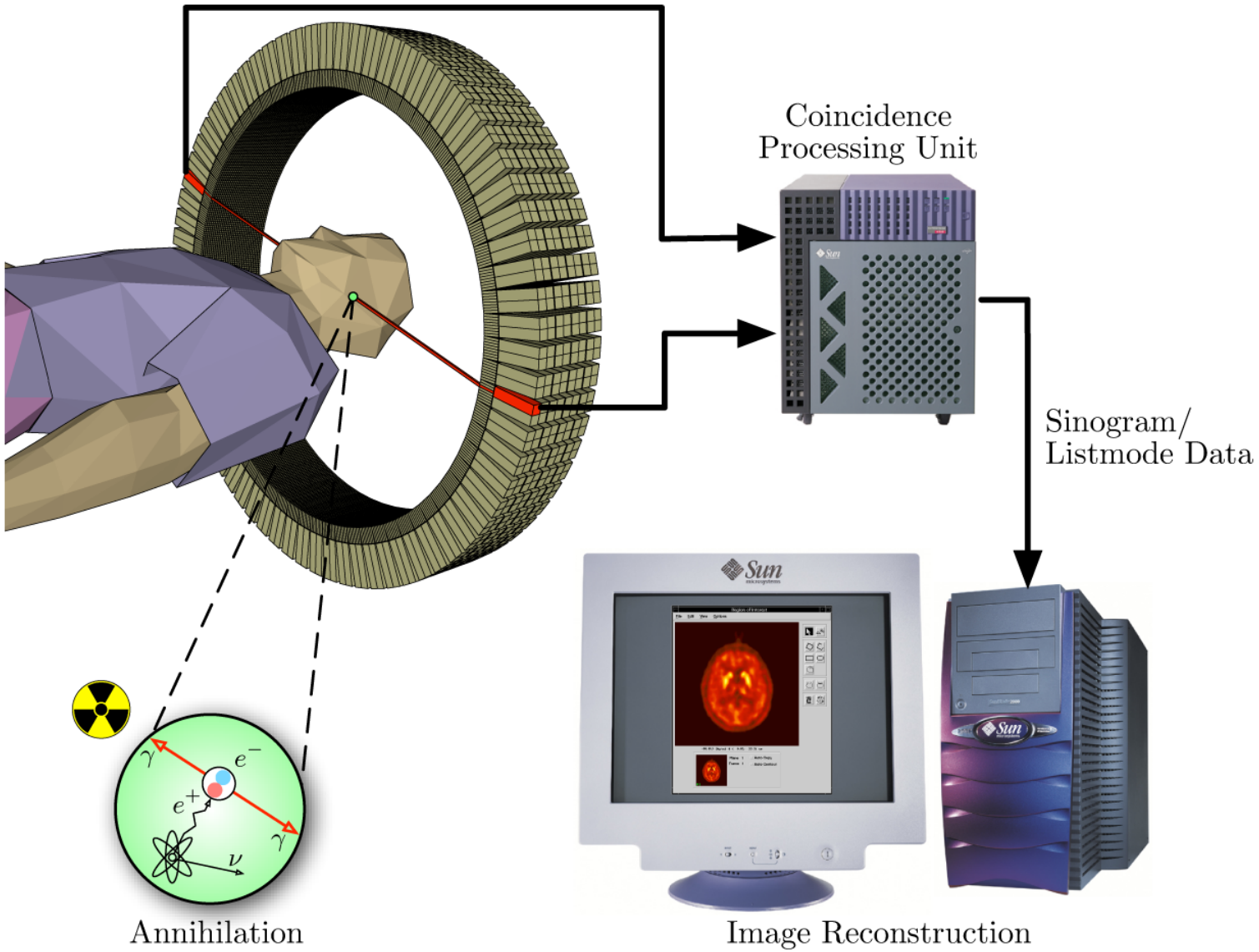
Paralelismus v návrhu



Příklady – Pozitronová emisní tomografie (EX-PET)

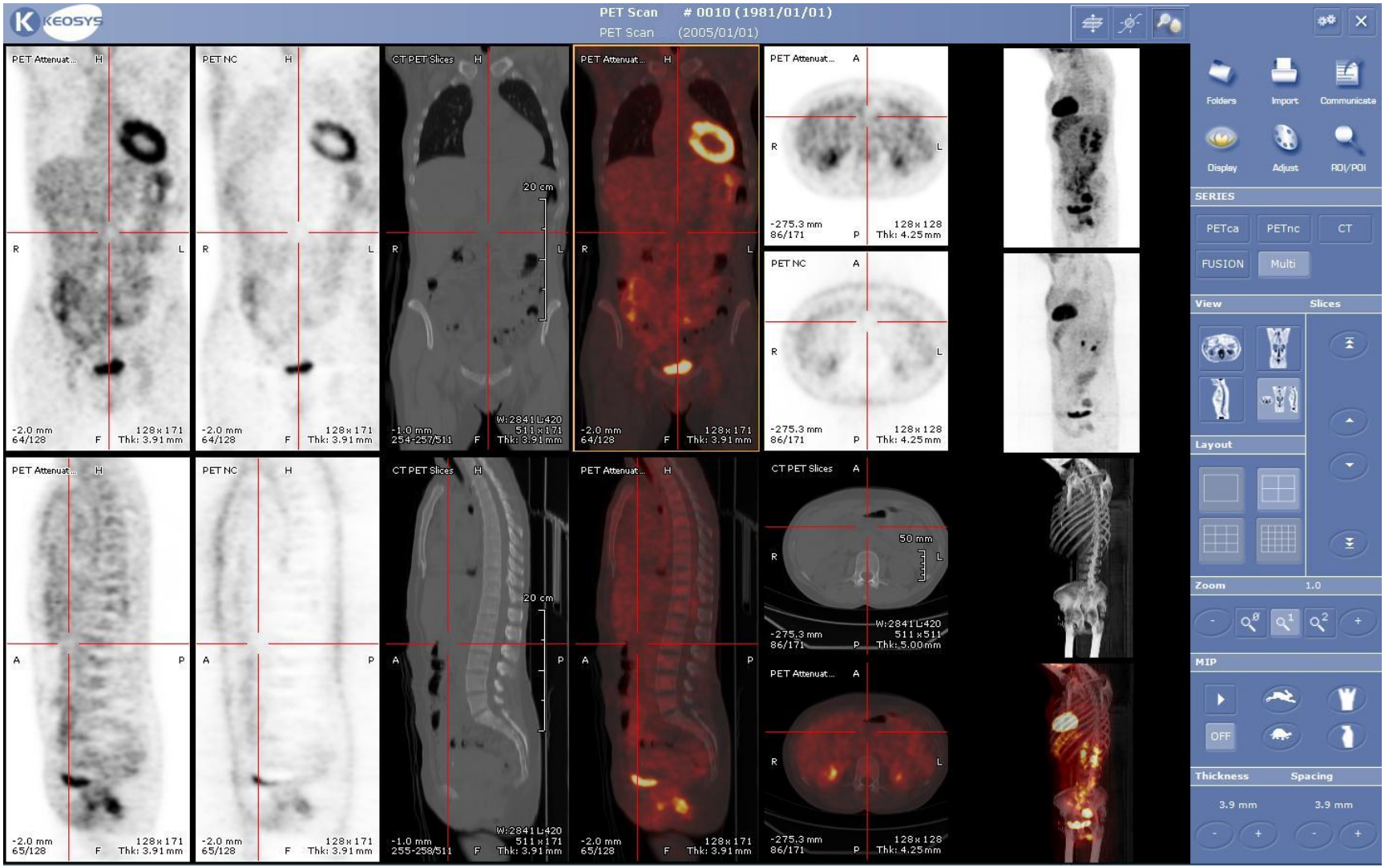
- Distribuce radioaktivního materiálu v lidském těle
  - vyšetření postupu metabolismu, např. pro onkologická vyšetření
  - rozptyl záření při průchodu tkáněmi
    $\Longrightarrow$ nízké rozlišení obrazu
  - absorpce záření není v organismu homogenní
    $\Longrightarrow$ absolutní hodnoty nejsou užitečné

Příklady – Pozitronová emisní tomografie (EX-PET)



Zdroj: **http://en.wikipedia.org/wiki/File:PET-schema.png**

Příklady – Pozitronová emisní tomografie (EX-PET)



Zdroj:
http://en.wikipedia.org/wiki/File:Viewer_medecine_nucleaire_keosys.JPG

Příklady – Pozitronová emisní tomografie (EX-PET)

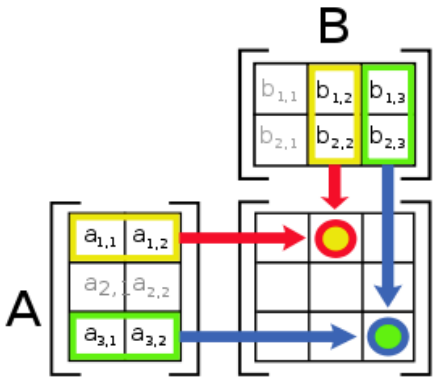
- Model těla
  - rozsáhlý model popisující jednotlivé tkáně
  - absorbční koeficienty pro jednotlivé „buňky“ modelu
- Simulace průchodu záření lidským tělem
  - Monte Carlo raytracing
  - náhodně zvolené body v lidském těle emitují záření (obvykle γ)
  - sledují se rozptyl a absorpce paprsku
  - zajímají nás paprsky dopadající na detektor
- Možnosti paralelizace
  - po paprscích $\Longrightarrow$ dekompozice na úlohy
  - po buňkách $\Longrightarrow$ dekompozice na data

Příklady – Lineární algebra (EX-LA)

- Násobení matic a vektorů

$$C = A \cdot B$$

$$C_{ij} = \sum_{k=1}^{N} A_{ik} B_{kj}$$

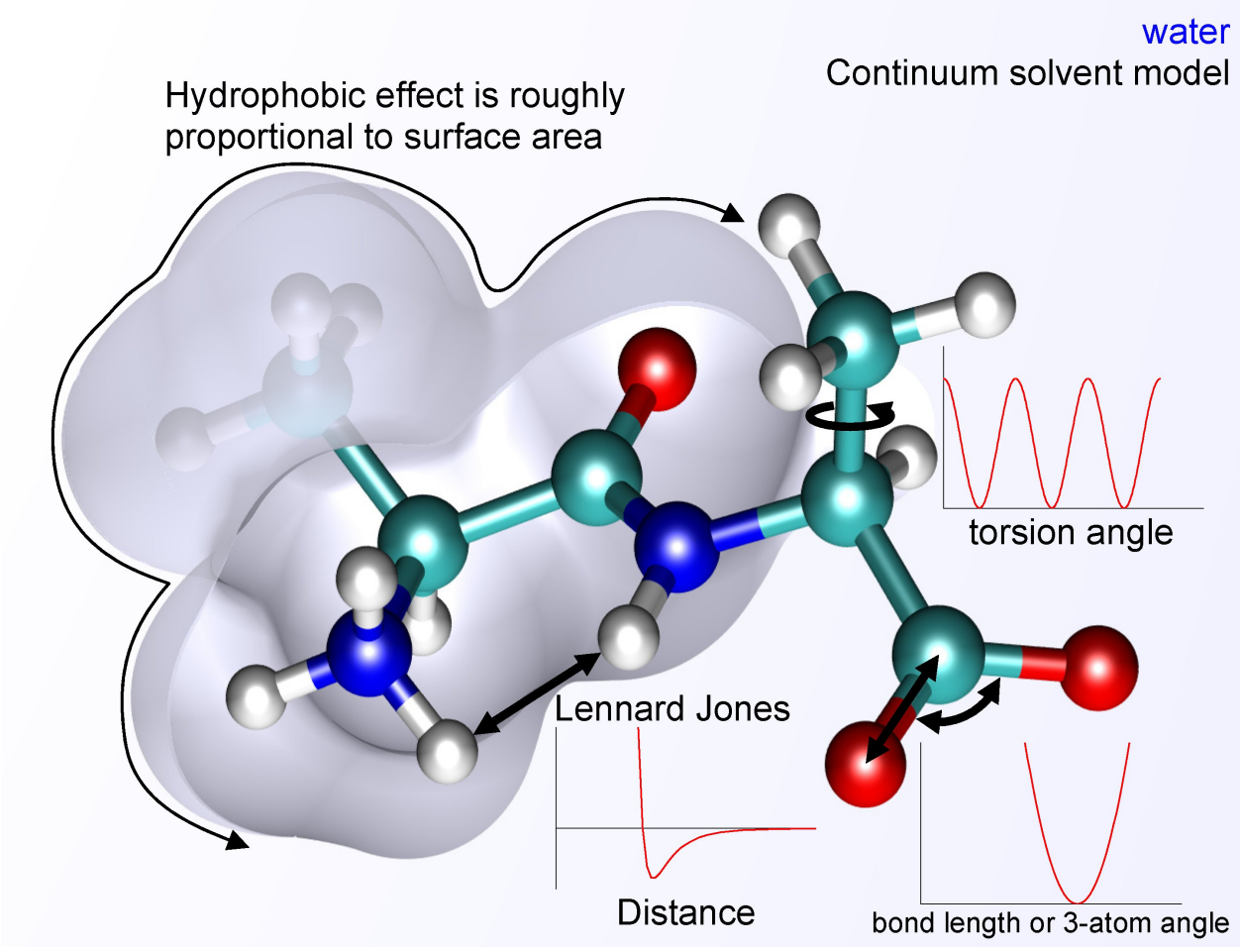


  - $\mathcal{O}(N)$ násobení a $\mathcal{O}(N-1)$ sčítání pro každý prvek $\Longrightarrow$ $\mathcal{O}(N^3)$ operací
  - existují efektivnější algoritmy
- Možnosti paralelizace
  - pro jednotlivé výsledné buňky $\Longrightarrow$ dekompozice na úlohy
  - pro jednotlivé oblasti zdrojových matic $\Longrightarrow$ dekompozice na data

Příklady – Molekulová dynamika (EX-MD)

- Problém modelování velkých molekul *per se*
  - neznáme přesné analytické řešení (dál než pro H_2^+)
  - lepší aproximativní modely kvantové mechaniky škálují $\mathcal{O}(N^5)$ – $\mathcal{O}(N!)$
- Molekulová mechanika
  - model pružinek, oscilátorů a elektrostatických potenciálů
  - problém parametrizace
  - problém chemické reaktivity (vznik/zánik vazeb, složitější interakce)
  - teoreticky „vadné“
  - prakticky používané
- Problém modelování v čase
  - interakce molekul v čase
  - konformační chování, např. při interakci léčivo–biomolekula

Příklady – Molekulová dynamika (EX-MD)



Zdroj: **http://en.wikipedia.org/wiki/File:MM_PEF.png**

Příklady – Molekulová dynamika (EX-MD)

- Mechanismus výpočtu

```
1  N : constant Positive;

3  atoms : array ( 1 .. 3, N) of Real;  --  positions
   velocities : array ( 1 .. 3, N) of Real;
5  forces : array ( 1 .. 3, N) of Real;
   neighbors : array (N) of List_of_Atoms;
7
   loop over time steps
9          vibrational_forces (in atoms, in out forces);
           rotational_forces (in atoms, in out forces);
11         neighbor_list (in atoms, out neighbors);
           non_bonded_forces (in atoms, in neighbors, in out forces);
13         update_atom_position_and_velocities (in out atoms,
                        in out velocities, in forces);
15         physical_properties (in atoms, in velocities, in forces);
   end loop;
```

Příklady – Molekulová dynamika (EX-MD)

- Problém N těles (N-body problem)
  - teoreticky všechna tělesa interagují se všemi
    $\Longrightarrow$ N^2 interakcí
  - problém při výpočtu `non_bonded_forces()`
  - řešení pomocí *cut-off* metody
    - pod definovanou vzdálenost se interakce ignoruje
    - funguje dobře pouze pro interakce $1/r^2$
- Možnosti paralelizace
  - jak na úlohy, tak na data

Dekompozice na úlohy

- *Pohled:* dekompozice na úlohy, které mohou běžet paralelně
- *Aspekty:*
  - *flexibilita* – přenositelnost na různé platformy
    - parametrizace velikosti úloh vzhledem k počtu výpočetních jednotek a jejich architektuře (např. paměťovým modelům)
  - *efektivita* – schopnost využití paralelismu vs. jeho režie
    - rozumný poměr mezi režií úlohy (vlákna, procesu) a délkou výpočtu
    - dostatečný počet úloh
  - *jednoduchost* – ladění, udržování kódu
    - přenos úloh ze sekvenčního prostředí (přístup OpenMP)
- Řešení
  - rozdělení na úlohy s minimálními závislostmi/synchronizací
  - rovnoměrné rozdělení úloh mezi výpočetní jednotky (load-balancing)
  - snažit se najít co nejvíce paralelních úloh, spojit se dají později
  - příklady:
    - volání funkcí
    - nezávislé smyčky
    - dekompozice dat pro jednotlivé úlohy

Dekompozice na úlohy – EX-PET

- Potřebujeme vygenerovat až ~ 10^6 trajektorií
- Přiřadíme každou trajektorii jedné úloze
- Data
  - informace o trajektorii (read-write)
  - model těla (read-only)
- Vlastnosti
  - \+ velké množství úloh
  - \+ jednoduché, přímočaré na ladění a údržbu
  - – potřeba kopírovat model těla, který může být velmi rozsáhlý
  - – problém malého počtu výpočetních operací v porovnáním s množstím dat potřebných z paměti
  - vhodné na architektury, kde výpočetní jednotky mají velkou a rychlou paměť (příp. i sdílenou na čtení)

Dekompozice na úlohy – EX-LA

- Přiřadíme každou buňku
- Data
  - jeden řádek a jeden sloupec z matic
- Vlastnosti
  - \+ velké množství úloh
  - \+ jednoduché, přímočaré na ladění a údržbu
  - – problém špatného komplikovaného přístupu ke sloupcům
    - matici lze transponovat
- Možnost seskupení úloh
  - opakované využití oblastí paměti
  - vede spíše na dekompozici podle dat

Dekompozice na úlohy – EX-MD

- Informace o problému
  - `neighbor_list()` …trvá dlouho
    - za předpokladu dostatečně husté simulace v časové doméně nemusíme počítat v každém kroku
  - `physical_properties()` …rychlé
  - `non_bonded_forces()`
    - SIMD smyčka – pro konkrétní atom počítáme příspěvek síly od jednotlivých započítávaných sousedů
    - samotný jeden výpočet je jednoduchý
    - může zahrnovat všechny atomy
  - `vibrational_forces()`,`rotational_forces()`
    - má smysl paralelizovat per atom

Dekompozice na data

- *Pohled:* dekompozice na data, která mohou být zpracovávána paralelně
  … v případě CUDA navíc i v režimu SIMD
- *Aspekty:*
  - *flexibilita*
    - možnost různé granularity dělení dat
  - *efektivita*
    - problém závislostí mezi bloky dat při různé granularitě dělení
      $\Longrightarrow$ řešení závislostí by nemělo škálovat rychleji než výpočet
      $\Longrightarrow$ řešení závislostí by nemělo trvat déle než výpočet
    - problém vyvažování zátěže na výpočetních jednotkách
    - SIMD/SIMT: lze pro danou dekompozici provést výpočet stejnými operacemi, nebo bude výpočet divergovat?
  - *jednoduchost*
    - datová dekompozice vede často na velmi zapeklité závislosti
- Řešení:
  - rozdělení polí – např. geometrické dekompozice
  - rekurzivní datové struktury
  - optimalizace vzhledem k použitému paměťovému modelu

Dekompozice na data – EX-PET

- Rozdělení modelu těla na segmenty
  - počet segmentů ~ počtu výpočetních jednotek
  - každá jednotka má svoji část modelu v blízké paměti
- Výpočet
  - každá výpočetní jednotka počítá trajektorii a absorpci ve svém segmentu
  - musí určit, které další výpočetní jednotce předat výpočet
  - problém balancování využití jednotek
    - v případě absence práce může vygenerovat novou náhodnou emisi
- Vlastnosti
  - \+ efektivní práce s pamětí
  - – musí řešit předávání trajektorie mezi výpočetními jednotkami

Dekompozice na data – EX-LA

- Rozdělení na bloky řádků matice C
  - využití segmentu matice A jednou
  - opakované využití celé matice B
- Rozdělení na bloky (podmatice) matice C
  - dosažení jemnější granularity
- Vlastnosti
  - \+ granularizovatelná paralelizace
  - \+ možnost dostatečně malých podoblastí, které se vejdou do cache/rychlé paměti
  - – musí správně vybírat data po sloupcích
    - možnost transpozice matice B
- Reálné implementace
  - ScaLAPACK – dělení na bloky
  - PLAPACK
    - $y = Ax$
    1. rozdělí vektory y a x
    2. odvodí rozdělení A

Dekompozice na data – EX-MD

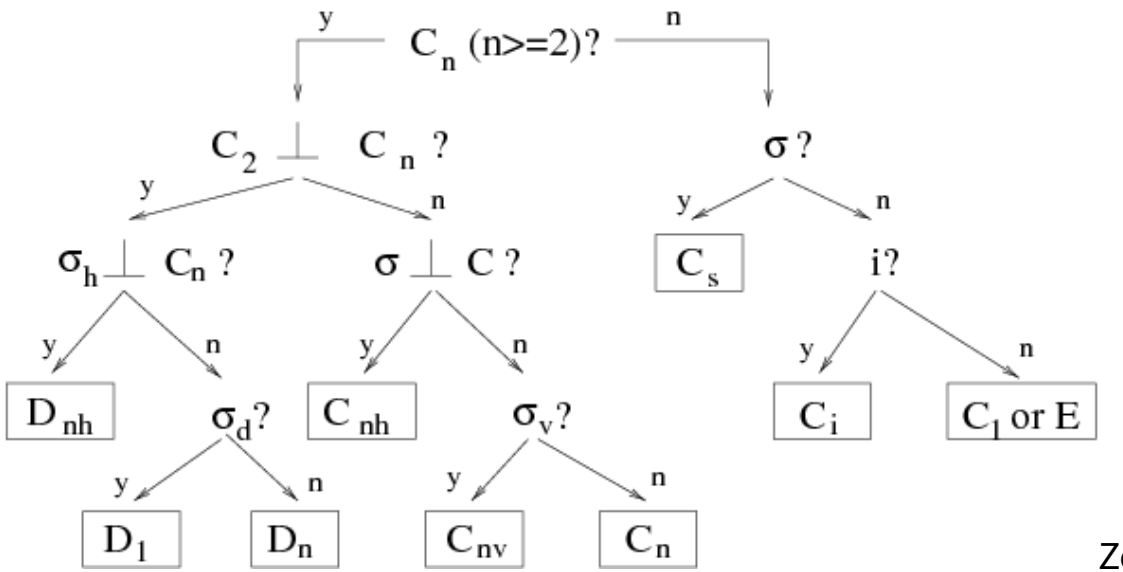
```
atoms : array ( 1 .. 3, N) of Real;  -- positions
velocities : array ( 1 .. 3, N) of Real;
forces : array ( 1 .. 3, N) of Real;
neighbors : array (N) of List_of_Atoms;

loop over time steps
        vibrational_forces (in atoms, in out forces);
        rotational_forces (in atoms, in out forces);
        neighbor_list (in atoms, out neighbors);
        non_bonded_forces (in atoms, in neighbors, in out forces);
        update_atom_position_and_velocities (in out atoms,
                        in out velocities, in forces);
        physical_properties (in atoms, in velocities, in forces);
end loop;
```

- Samostatná datová dekompozice není přímočará
  - lépe je využít rozpad datových struktur podle úloh
  - polohy atomů – jsou potřeba všude
  - rychlosti atomů – pouze u aktualizace
  - síly – lze počítat lokálně a následně sčítat
- Symetrie v datech
  - $f_{ij} = -f_{ji}$ … třetí Newtonův zákon
  - snížení množství dat na polovinu

Dekompozice na data – EX-MD

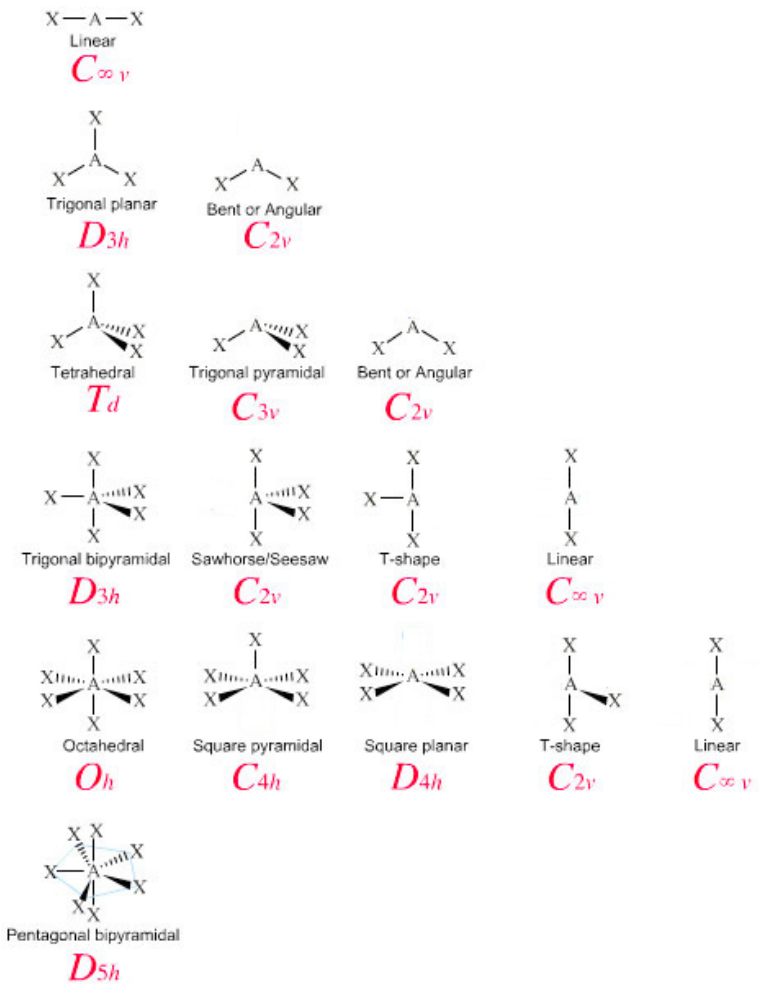
- Využívání symetrie molekul
  - nalezení grupy symetrie



Zdroj:
http://en.wikibooks.org/wiki/Introduction_to_Mathematical_Physics/N_body_problem_in_quantum_mechanics/Molecules

Dekompozice na data – EX-MD

- Využívání symetrie molekul
  - nalezení grupy symetrie



Zdroj:
http://chemistry.ncssm.edu/labs/molgeom/pointsym.jpg

Seskupování úloh

- Hledání struktur/hierarchie v úlohách
  - skupiny úloh zjednodušují plánování a snižují chybovost
  - časové struktury/synchronizace
    - striktní souslednost
    - pipelining
    - současný běh – ko-plánování
    - zcela nezávislé úlohy
  - závislosti mezi úlohami (možnost vyvažování zátěže výpočetních jednotek)
- Postupy
  - hierarchie dekompozice
  - sdílení omezení mezi úlohami
  - slučování skupin: větší skupiny vedou na efektivnější plánování
- EX-LA
  - seskupování do bloků
  - skládání výsledné matice

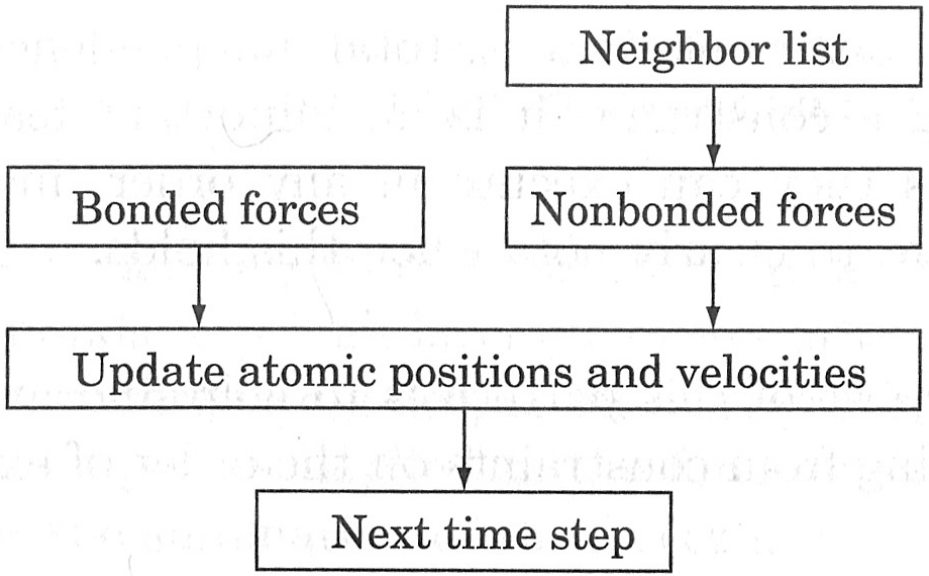
Seskupování úloh – EX-MD

- Seskupování podle činností
  - výpočet sousedů
  - aktualizace působících sil
    - výpočty vibračních sil
    - výpočty rotačních
    - výpočet nevazebných interakcí
  - aktualizace poloh a rychlostí
  - výpočet vlastností

Pořadí úloh

- Jaké jsou závislosti mezi úlohami?
  - časové závislosti
  - datové závislosti
- Přístupy
  - minimální nutná omezení – nevynucovat uspořádání, kde to není třeba
  - čekání na potřebná data
  - čekání na dokončení potřebných I/O operací
  - vyjádření nezávislosti
    - z důvodu bezpečnosti a udržovatelnosti vyjadřovat explicitně

Řazení úloh – EX-MD



- nevazebné síly potřebují znát započítávané sousedy
- pozice atomů se mohou aktualizovat až jsou vypočítané síly
- výpočet vlastností potřebuje pozice atomů a jejich rychlosti

Sdílení dat

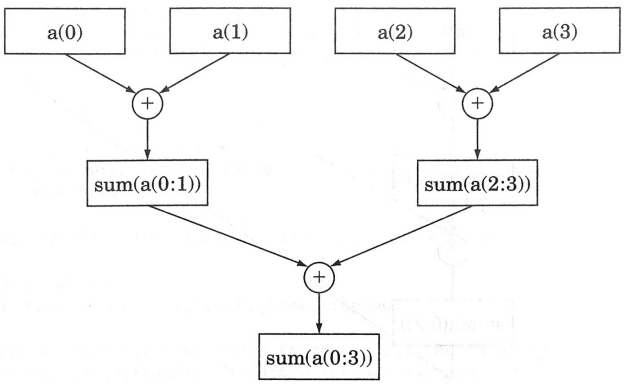
- Definování přístupu k datům u souběžných jednotek
  - lokální data
  - globální data
  - vyměňovaná data mezi úlohami
    - např. okrajové buňky u konečné diferenční metody
  - relaxování přístupu ke globálním datům
    - ne vždy je třeba globální synchronizace
    - vytváření lokálních kopií a jejich synchronizace
    - minimalizace překryvů přestrukturováním úloh
  - pipelining – paralelizace výpočtu a synchronizace/komunikace

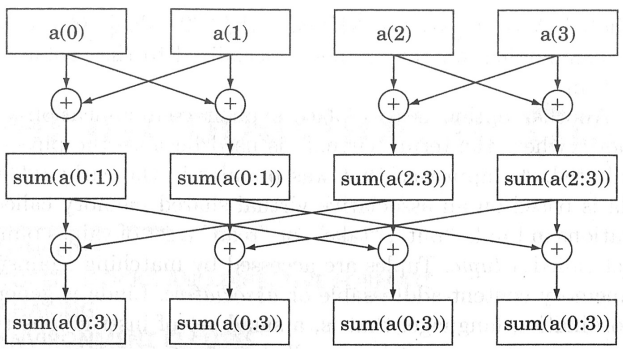
Sdílení dat

- Způsoby přístupu
  - jen na čtení
    - možno cacheovat/replikovat
  - efektivně lokální
    - přístup na čtení i zápis jen k disjunktnímu bloku dat
    - možnost pozdější redukce
    - složitější výběr prvků
  - na čtení i zápis
    - přístup do globální paměti vs. distribuce aktualizací

Sdílení dat

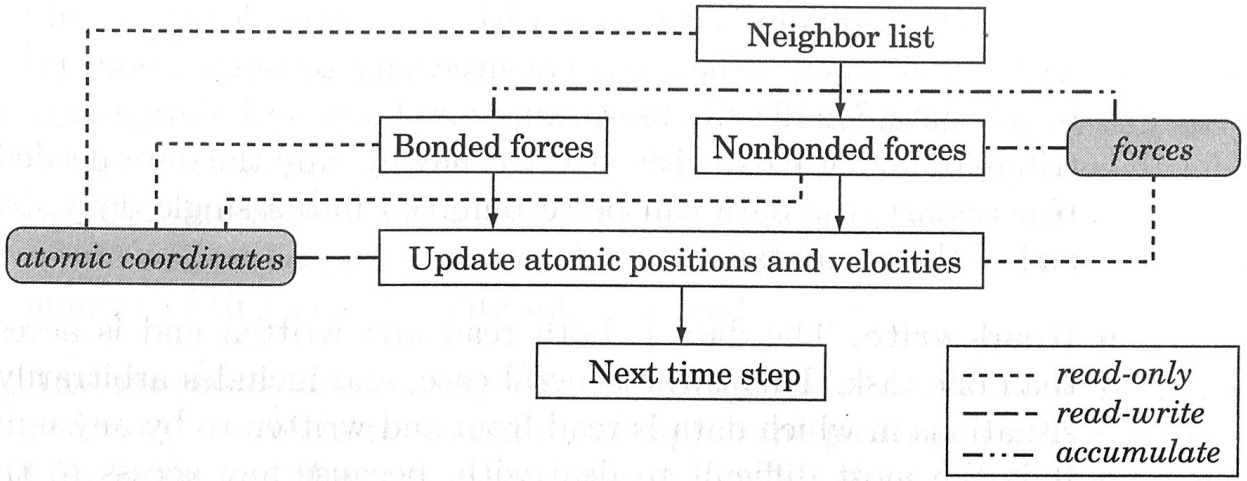
- Způsoby přístupu
  - akumulace/redukce
    - paralelizace asociativních operací





    - implementace pro CUDA:
      `http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf`
  - jeden zapisuje, ostatní čtou
    - lokální kopie pro zapisujícího s distribucí čtoucím

Sdílení dat – EX-MD



- možnost pravidelné replikace pozic atomů
- použití paralelních redukcí při sčítání sil
- průběžný výpočet sousedů s periodickým přenosem do výpočtu nevazebných sil

Analýza návrhu

- Zpětná analýza návrhu
  - poslední krok samotného návrhu … neobjevovat chyby návrhu až hluboko v implementace
  - máme:
    - dekompozice podle úloh
    - dekompozice podle dat
    - analýzu struktury dekompozice
- Úvahy
  - Je návrh vhodný pro cílovou platformu? (CUDA)
    - máme dost paralelní práce pro všechny multiprocesory?
    - nebudou nám výpočetní stromy divergovat pro SIMD/SIMT?
    - máme dobře namapované úlohy na mřížku?
    - využívá úloha dobře paměťové hierarchie (registry/lokální paměť/globální paměť/cachování read-only dat, zarovnávané adresování, konflikty)?
    - umíme pipeliningem (pomocí spouštění jiných warpů) maskovat latenci pamětí a synchronizaci? máme pro to dost vláken?
  - Je návrh stabilní, laditelný a udržitelný?
    - řeší dobře hraniční případy? (např. vysoce asymetrické matice)

Analýza návrhu – EX-LA

- minulá přednáška:

Implementace	rychlost	rel. Δ	abs. Δ
Naivní implementace, thready 1×128	3.9 GFlops		
Naivní implementace	36.6 GFlops	9.4×	9.4×
Blokový přístup	198 GFlops	5.4×	51×
Bloky 32×16 pracující s daty 32×16	235 GFlops	1.19×	60×
Odstranění ADD instrukce	276 GFlops	1.17×	71×
Jen jeden blok ve sdílené paměti	375 GFlops	1.36×	96×

- Poučení:
  - blokovou paralelizací jsme dosáhli největšího zrychlení

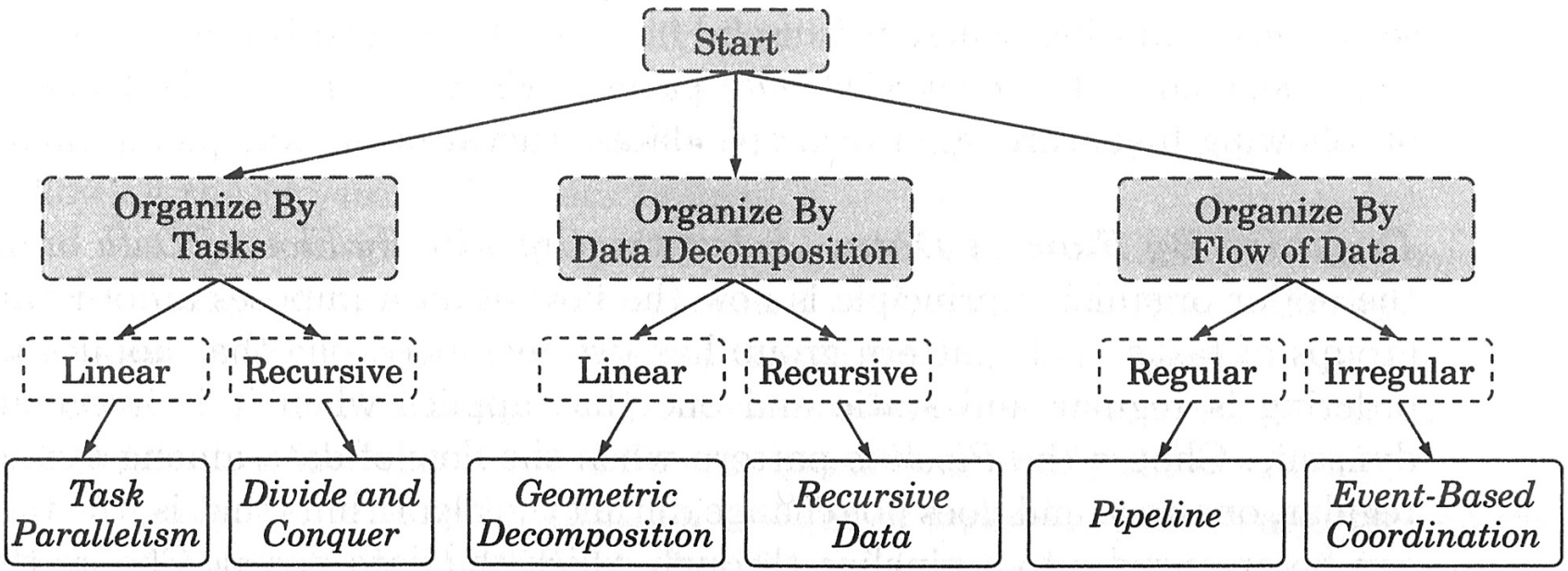
Analýza návrhu – EX-PET

- Umíme dekomponovat na segmenty modelu
  - \+ škáluje na téměř neomezený počet výpočetních jednotek
  - – problém s předáváním stop
- Umíme dekomponovat na jednotlivé stopy
  - \+ škáluje na téměř neomezený počet výpočetních jednotek
  - – přístup k rozsáhlému modelu
  - eventuální částečné zavedení lokality
  - intuitivnější

Analýza návrhu – EX-MD

- Umíme dekomponovat podle jednotlivých atomů podle příspěvků sil
  - pro menší počty procesorů můžeme shlukovat úlohy
- Počítání nevazebných interakcí bere většinu času
  - umíme zmenšit díky symetriím
  - owner-computes filter – rozdělení, kdo v dané iteraci počítá a kdo lelkuje
  - můžeme hledat efektivnější algoritmy na cut-off
  - … námět na další přednášku :-)

Volba vzorů


Start
Organize By Tasks
Organize By Data Decomposition
Organize By Flow of Data
Linear
Recursive
Linear
Recursive
Regular
Irregular
Task Parallelism
Divide and Conquer
Geometric Decomposition
Recursive Data
Pipeline
Event-Based Coordination

Paralelismus úloh

- Dekompozice podle úloh
  - potřeba vygenerovat dostatečné množství úloh
- Řešení závislostí:
  - odstranitelné závislosti

```
int ii=0, jj=0;

for (int i=0; i<N; i++) {
        ii = ii+1;
        d[ii] = dlouhy_vypocet(ii);
        jj = jj+i;
        a[jj] = jiny_dlouhy_vypocet(jj);
}
```

```
for (int i=0; i<N; i++) {
        d[i] = dlouhy_vypocet(i);
        a[i*(i+1)/2] = jiny_dlouhy_vypocet(i*(i+1)/2);
}
```

  - rozdělitelné závislosti: lokální výpočet + redukce
  - jiné typy závislostí: řešení sdílením dat

Paralelismus úloh

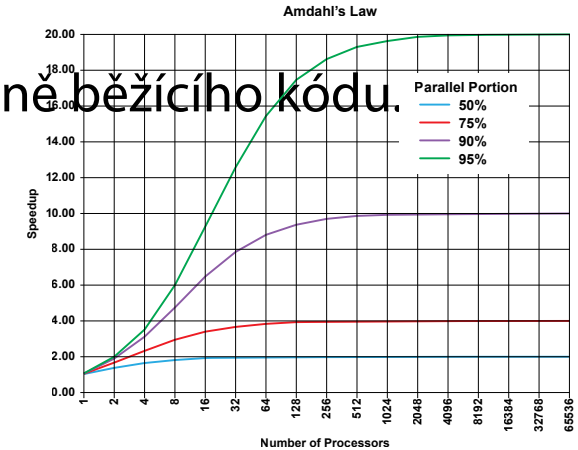
- Plánování úloh:
  - statické: vygenerování úloh na počátku (např. paralelizace smyček předem dané velikosti)
  - dynamické
    - dynamické generování úloh do centrální fronty
    - work stealing
- Idiomatické typy:
  - embarrassingly parallel tasks
  - úlohy nad replikovanými daty (1. vykopírování dat do lokálních proměnných, 2. výpočet, 3. seskládání výsledků)
- Příklady:
  - zpracování obrazu: pokud je zpracování pro oblasti nezávislé na okolí
  - raytracing, PET: dekompozice na jednotlivé paprsky
  - molekulová dynamika: dekompozice na jednotlivé atomy při výpočtu příspěvků sil; dvouelektronové integrály u kvantových metod

Rozděl a panuj

- Klasický přístup pro rekurzivní úlohy
- Problém se sekvenční režií dělení na podúlohy

$$S(P) = \frac{T_{tot}(1)}{T_{tot}(P)} = \frac{1}{1 - f + \frac{f}{N}}$$

kde f je podíl paralelně běžícího kódu.



Zdroj: http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

- Příklady: mergesort, diagonalizace matic (hledání vlastního vektoru a vlastních hodnot – hledání matice Q takové že $Q^T \cdot T \cdot Q$ je diagonální), rychlá multipólová metoda (FMM)

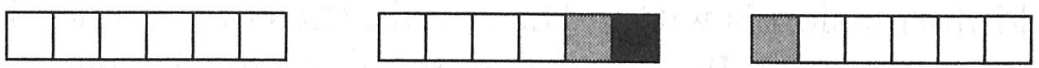
Geometrická dekompozice

- Hledání geometrického dělení zpracovávaných dat
  - data dělitelná na velké množství nezávislých segmentů
  - řešení hranic mezi segmenty – ghost boundary
- Příklad – vedení tepla:

$$\frac{\partial U}{\partial t} = \alpha \frac{\partial^2 U}{\partial x^2}$$

```
ukp1[i] = uk[i] + (dt/(dx*dx))*(uk[u+1] + uk[u-1] - 2*uk[i]);
```

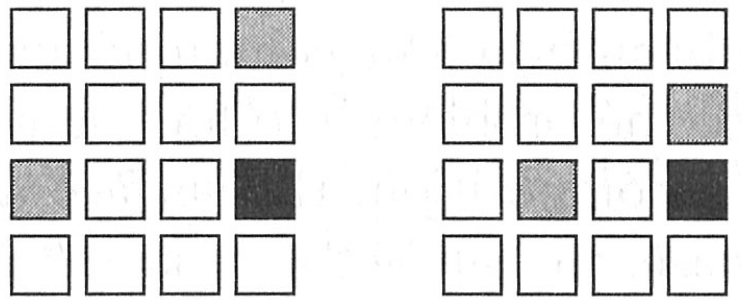
update requiring only local info



update requiring information from another chunk

Geometrická dekompozice

- Příklad – násobení matic:



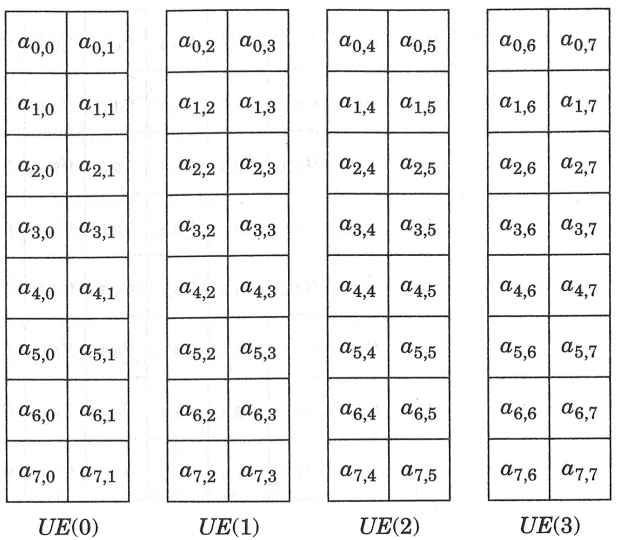
- Kroky:
  - rozklad dat
    - poměr mezi množstvím vyměňovaných dat a všech dat
    - dělení matice na 4ks: sloupce ($5N/2$) vs. čtverce ($2N$)
  - výměna dat před novým krokem výpočtu
  - výpočet na vlastním segmentu

Geometrická dekompozice

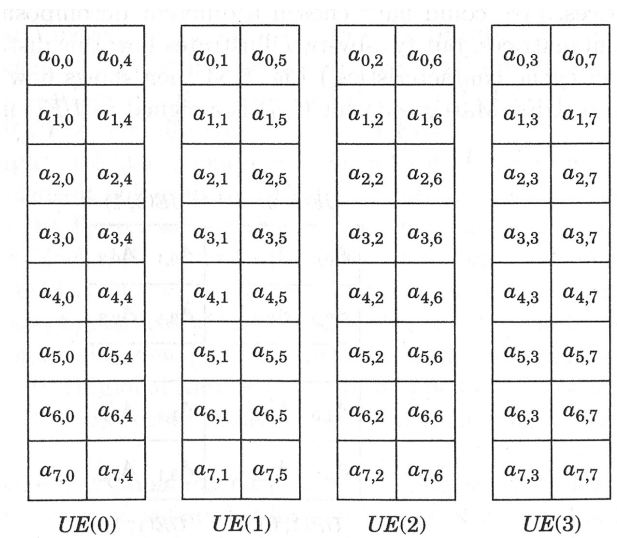
- Rozložení úloh přes procesory:
  - minimalizace režie výměn dat
  - efektivita využití úloh (např. při eliminacích matic mohou procesory hladovět $\Longrightarrow$ cyklické přidělování bloků)

  1D dekompozice:

dekompozice



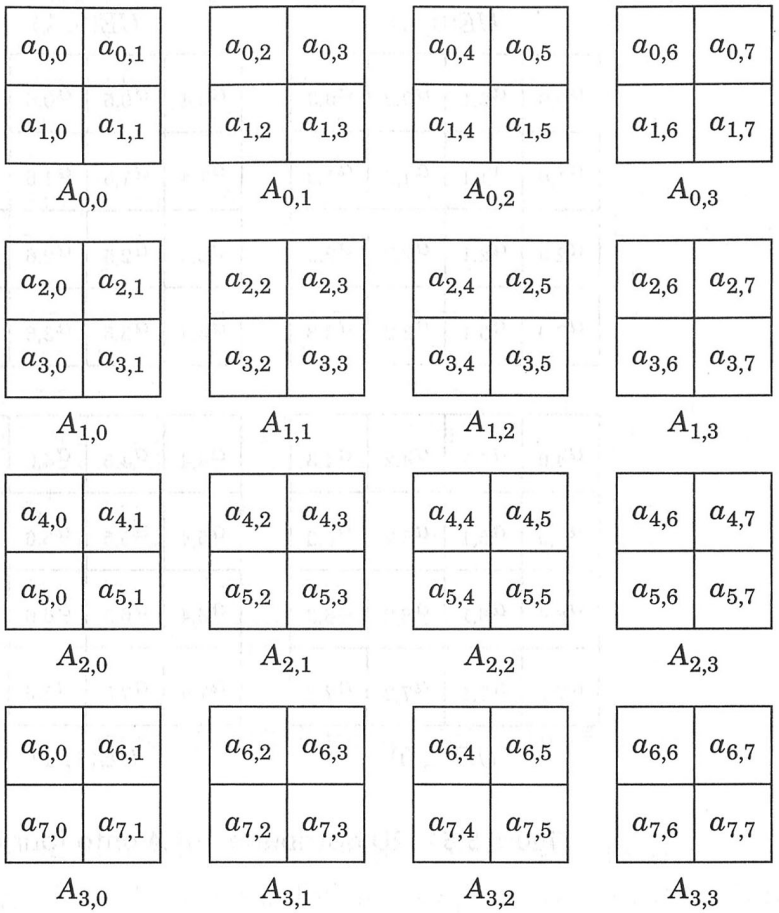
přiřazení ($j \mod 4$)



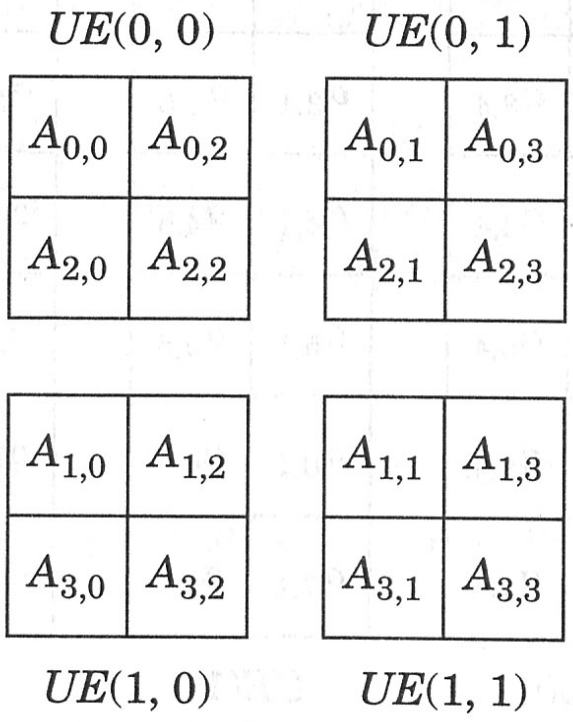
Geometrická dekompozice

- 2D dekompozice:

dekompozice



přiřazení ($i \mod 2, j \mod 2$)



Geometrická dekompozice

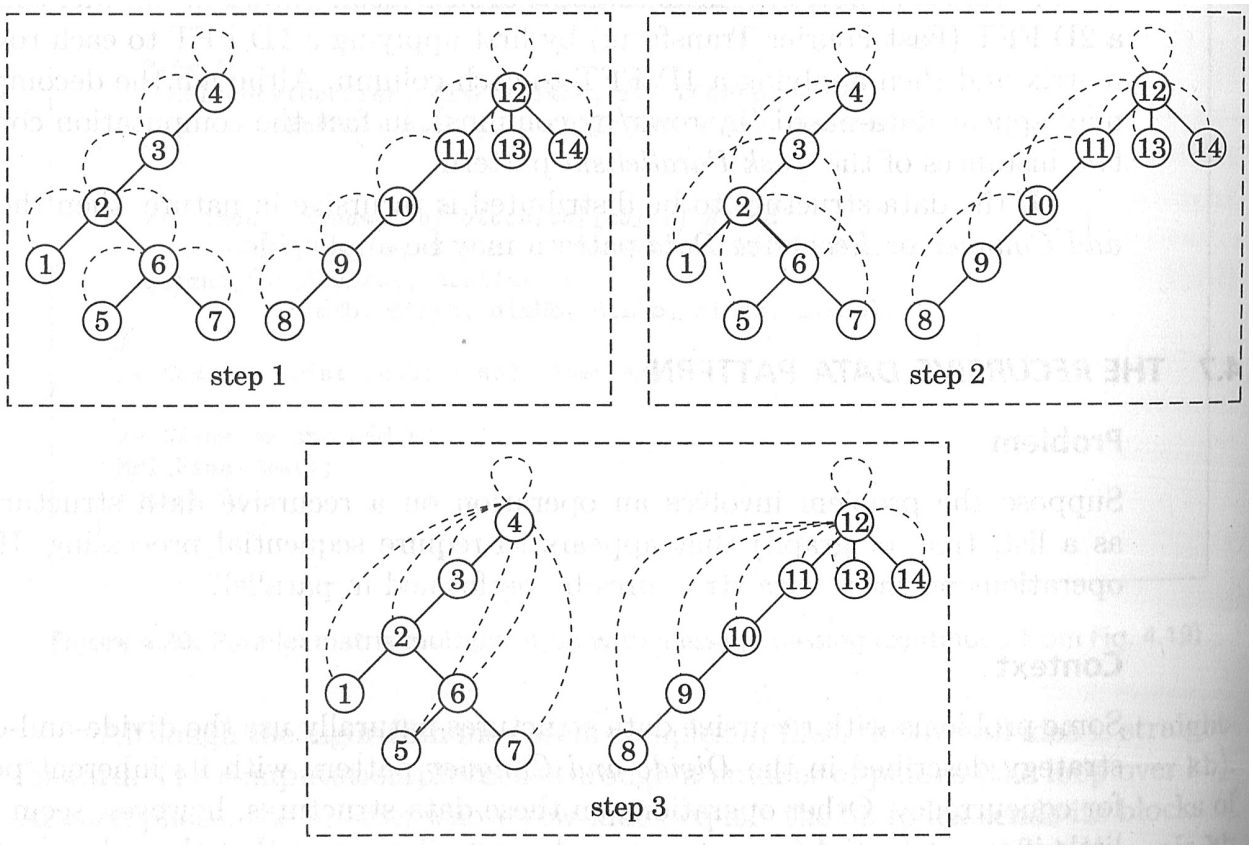
- Maskování latence komunikace – překryv komunikace a výpočtu
  1. neblokující volání zahajující komunikaci (distribuci ghost boundary)
  2. výpočet vnitřních prvků
  3. čekání na dokončení komunikace
  4. výpočet hodnot okrajových prvků, závisejících na ghost boundary

Rekurzivní pohled na data

- Pohled na data, kde nejde přímočaře použít přístup rozděl a panuj z povahy algoritmu
- Příklad: les stromů (každý uzel si drží svého předchůdce, kořen je sám sobě předchůdcem), hledáme pro každý uzel kořen
  - sekvenčně: z každého kořene provedeme DFS – $\mathcal{O}(N)$
- Paralelně:
  - zavedeme si pro každého předchůdce-předchůdce
  - výpočet opakujeme až do konvergence
  - postupujeme na všech N uzlech paralelně
  - celkově $\mathcal{O}(N \log N)$, ovšem při paralelním provedení efektivně jen $\mathcal{O}(\log N)$
- Zhoršení celkové složitosti je pro tuto dekompozici typické
- Potřeba sdílet výsledky po každém kroku (lockstep) (podpora např. v řadě SIMD strojů a High-Performance Fortranu)

Rekurzivní pohled na data

- Kroky:



Rekurzivní pohled na data

- Pohled na data, kde nejde přímočaře použít přístup rozděl a panuj z povahy algoritmu
- Příklad: les stromů (každý uzel si drží svého předchůdce, kořen je sám sobě předchůdcem), hledáme pro každý uzel kořen
  - sekvenčně: z každého kořene provedeme DFS – $\mathcal{O}(N)$
- Paralelně:
  - zavedeme si pro každého předchůdce-předchůdce
  - výpočet opakujeme až do konvergence
  - postupujeme na všech N uzlech paralelně
  - celkově $\mathcal{O}(N \log N)$, ovšem při paralelním provedení efektivně jen $\mathcal{O}(\log N)$
- Zhoršení celkové složitosti je pro tuto dekompozici typické
- Potřeba sdílet výsledky po každém kroku (lockstep) (podpora např. v řadě SIMD strojů a High-Performance Fortranu)

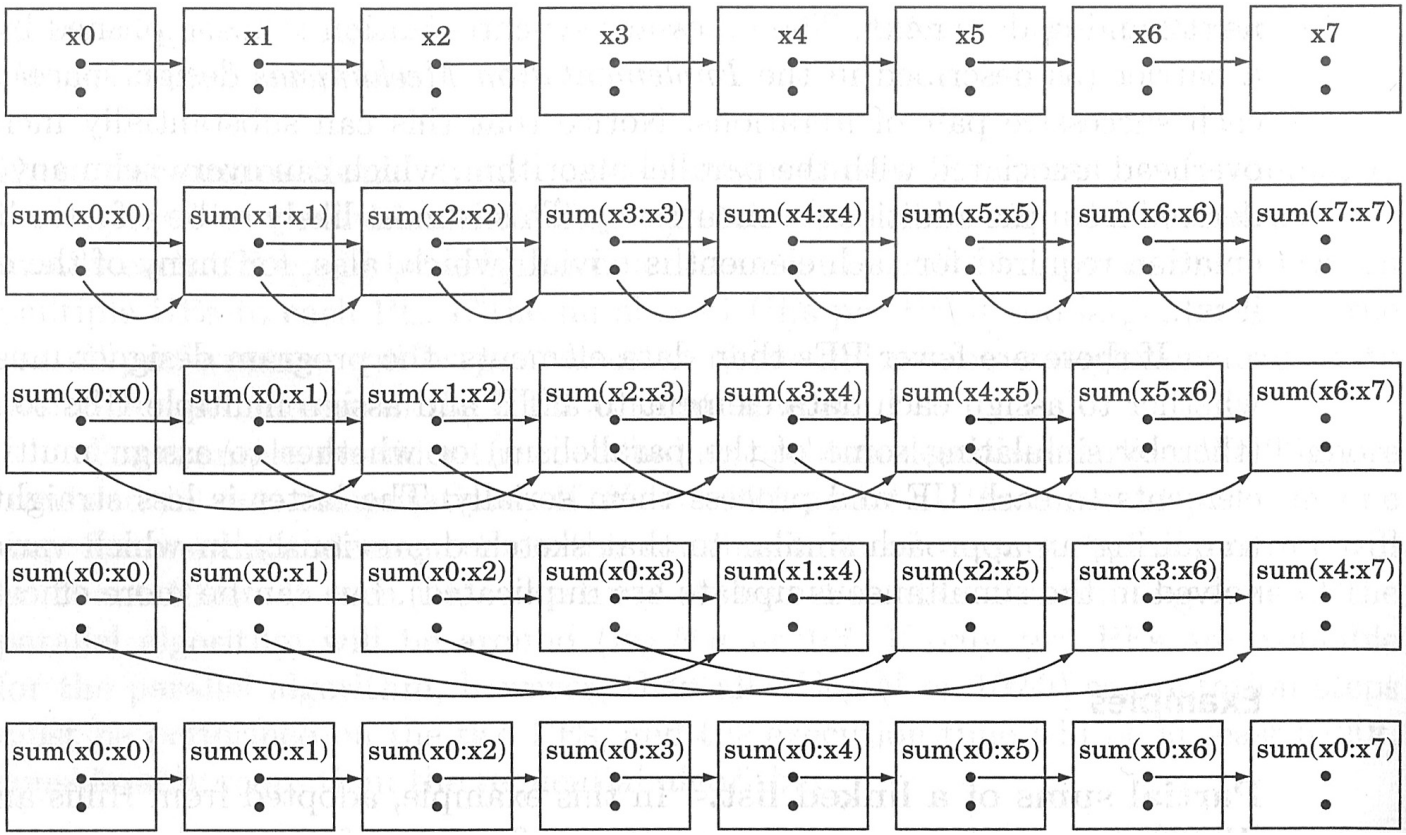
Rekurzivní pohled na data

- Součty prefixů
  - potřebujeme sečíst pro *n*-tý prvek všech 1 až *n* předchůdců
  - opět rekurzivní zdvojování

```
1 for all k in parallel {
        temp[k] = next[k];
3       while(temp[k]) {
                x[temp[k]] = x[k] + x[temp[k]];
5               temp[k] = temp[temp[k]];
        }
7 }
```

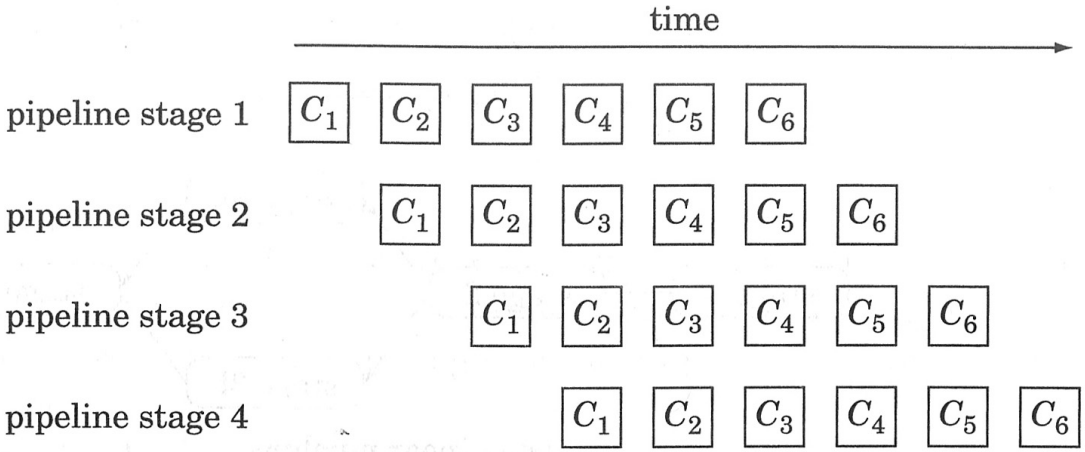
Rekurzivní pohled na data

- Co se děje:



Pipeline

- … známe z procesorů



- Příklady: obecně pravidelné zpracování
  - pipeline instrukcí
  - pipeline na úrovni paralelizace smyček (vektorové stroje)
  - zpracování signálů sérií filtrů
  - rozdělení grafického zpracování na „filtry“
  - vyjádření závislostí na *n* předchozích krocích

Pipeline

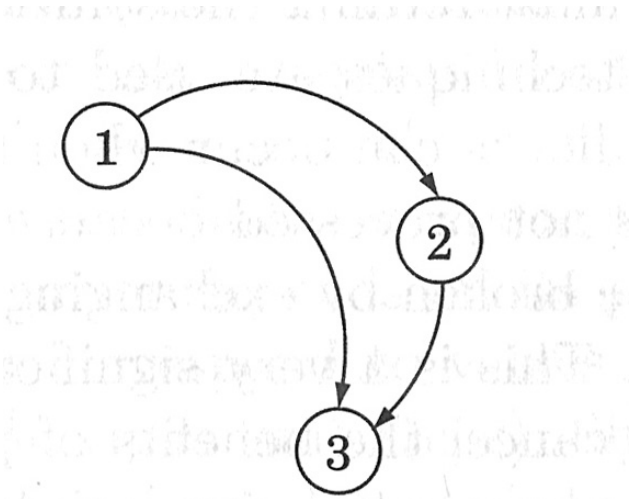
- Efektivita:
  - počet fází pipeline (škálovatelnost)
  - rovnoměrnost výpočetní náročnosti fází
  - plnění a dojíždění pipeline vs. celkový běh
  - průtok dat vs. latence zpracování
    - napojování pipeline, resp. vytvoření co nejdelší pipeline
  - všechny prvky by měly projít pravidelně stejnou sérií zpracování
- Reprezentace toku dat
  - sdílená paměť, soubory – uspořádaná fronta pro každou fázi
  - distribuovaná paměť – bufferované zasílání zpráv
    - v případě vyšší latence přenosu zpráv možno agregovat více požadavků (prodlužuje naplnění pipeline)
  - agregace dat, pokud jich každá fáze potřebuje více (dtto)
- Chyby ve zpracování
  - separátní komponenta sbírající informace chybách

Pipeline

- Plánování úloh
  - možnost/nutnost slučování fází: využívání cache, registrů
  - možnost paralelizace uvnitř fází
  - možnost více instancí jedné fáze
    - pokud nevadí přeuspořádání
- Příklady:
  - zpracování dat ve frekvenční doméně – DFT
    - provedení DFT/FFT
    - aplikace jednotlivých filtrů
    - inverzní DFT/FFT
  - např. zpracování telemetrických dat z Hubblova teleskopu, z radarů AWACS
  - častá implementace v HW – zpracování signálů v reálném čase
  - zpracování HTTP požadavků – filtrování Java Servlety

Událostmi řízená paralelizace

- Nepravidelná a asynchronní obdoba pipeline
- Nepravidelné rozdělené a zpracování vstupů – nevhodné pro SIMD/SIMT



- Problém přeuspořádání událostí
- Obecné komunikační schéma s možností deadlocků