

# Paralelné algoritmy, část č. 1

František Mráz

Kabinet software a výuky informatiky, MFF UK, Praha

Paralelné algoritmy, 2011/2012

# Obsah

# Úvod

## Prečo paralelné počítanie

- zvyšovanie výkonu sekvenčných procesorov naráža na hranice možností
- vývoj hardware
  - už v 80. rokoch 20. stor. sa používali vektorové počítače a boli stavané multiprocessorové systémy (nCUBE, Connection Machine (prvý s  $2^{16}$  procesormi, Intel Paragon, ...))
  - v 90. rokoch viacjadrové procesory

## Môže paralelizácia pomôcť?

- Riešenie akých úloh môže paralelizácia zrýchliť?
- Aké zrýchlenie je teoreticky možné? Pre všetky úlohy, alebo iba pre niektoré?
- $\Rightarrow$  potrebujeme dobrý model!

# Obsah semestrálnej prednášky

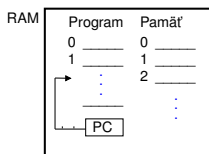
- 1 Model PRAM. Vychádza zo sekvenčného modelu RAM, ale môže byť príliš silný!
- 2 Základné techniky paralelných výpočtov.
- 3 Základné paralelné algoritmy – hlavne grafové.
- 4 Aké sú hranice paralelných výpočtov – ťažko paralelizovateľné úlohy.

# Literatúra

- 1 Ian Parberry: Parallel Complexity Theory, Pitman Publishing, (John Wiley & Sons), 1987.
- 2 Alan Gibbons, Wojciech Rytter: Efficient Parallel Algorithms, Cambridge University Press, 1988.
- 3 Joseph Jájá: An Introduction to Parallel Algorithms, Addison Wesley, Massachusetts, 1992.

# Random Access Memory – RAM

Počítač s náhodným prístupom (do pamäte).

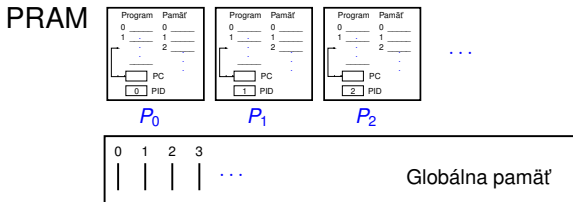


- **Program** = pevná postupnosť inštrukcií
- **Programový čítač (PC)** – register obsahujúci číslo inštrukcie, ktorá sa bude vykonávať.

- **Pamät'** = spočetne mnoho registrov; každý z nich môže obsahovať ľubovoľne veľké celé číslo.
- **Inštrukcie** programu:
  - $r_i :=$  konštanta;
  - $r_i := r_j \circ r_k$ , kde  $\circ$  je binárna operácia, napr.  $+$ ,  $-$ ,  $*$ ,  $\text{mod}$ ,  $\text{div}$ ;
  - $r_i := r_j$ ,  $r_i := r_j \dots$  **nepriama adresácia**;
  - halt;
  - goto  $m$  if  $r_i > 0$ .
- **Vstup**:  $r_0$  obsahuje počet čísel,  $r_1, \dots, r_{r_0}$  obsahujú vstupné údaje;
- **Výstup**: dtto

# Parallel RAM – PRAM

- postupnosť procesorov – RAM počítačov; každý z nich má navyše číslo PID (**P**rocessor **I**dentification number) uložené v špeciálnom registri; procesory pracujú synchronne
- Spoločná globálna pamäť; jej registre budeme označovať s pruhom  $\bar{r}_0, \bar{r}_1, \bar{r}_2 \dots$
- Inštrukcie – tak ako RAM
  - + tie isté inštrukcie na prácu s globálnou pamäťou a
  - + inštrukcia  $r_i := \text{PID}$
- Vstup a výstup v počiatočnom úseku globálnej pamäte.



# PRAM

## Riešenie konfliktov

- Pri čítaní

**ER** Exclusive Read – konflikty pri čítaní zakázané,

**CR** Concurrent Read – konflikty povolené.

- Pri zápise

**EW** – Exclusive Write – konflikty pri zápise zakázané,

**CW** – Concurrent Write – konflikty povolené, ale čo s nimi?

**PRIORITY** zápis sa podarí procesoru s najnižším PID

**ARBITRARY** zápis sa podarí ľubovoľnému procesoru

**COLLISION** podarí sa zapísať značku, že nastala kolízia

**COMMON** zápis sa podarí, ale všetci musia zapisovať to isté číslo

**W-PRAM** zápis sa podarí, ale všetci musia zapisovať rovnaké číslo 1

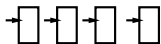
...

- Najčastejšie používané: CREW, COMMON, PRIORITY



# Priradenie programov procesorom

## SIMD – Single Instruction Multiple Data



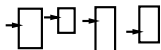
- zhodné programy a zhodné PC
- k riadeniu skokov iba globálne podmienky

## Uniformné



- zhodné programy, rovnaký počet vykonaných inštrukcií
- rôzne PC

## MIMD – Multiple Instruction Multiple Data



- rôzne programy
- rôzne PC

- simulácia SIMD  $\rightarrow O(1)$  uniformné
- simulácia SIMD  $\leftarrow O(\text{dĺžka pgmu})$  uniformné

# Miery zložitosti paralelných výpočtov

## Časová zložitosť jednej inštrukcie

- jednotková cena inštrukcie
- logaritmická cena inštrukcie = počet spracovaných bitov

uvažujeme jednotkovú (synchronizácia), ale je treba dať pozor na veľkosť spracovávaných čísel; predpokladáme model SIMD, resp. uniformný

$T(n)$  čas výpočtu na vstupe dĺžky  $n$ .

$S(n)$  maximálny počet registrov pamäte potrebných pri výpočte nad vstupom veľkosti  $n$ ,

$P(n)$  počet aktivovaných procesorov,

$W(n)$  dĺžka slova = dĺžka najdlhšieho slova (počet bitov čísla) zapísaného v priebehu výpočtu do nejakého registra

# Aktivácia procesorov

- iniciálna = “všetci naraz” – nerealistické
- lenivá (angl. “lazy”) = “prvý budí ostatných”

# Instrukčná sada

množina aritmetických inštrukcií

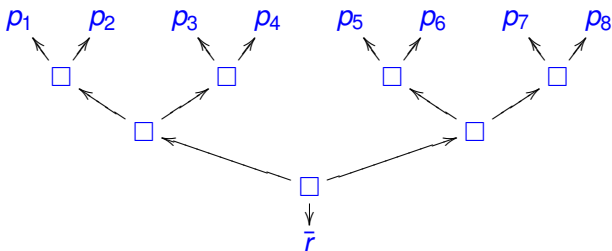
- minimálna:  $+$ ,  $-$ ,  $\text{div } 2$
- **obmedzená**:  $+$ ,  $-$ ,  $\text{div } 2$ ,  $\lfloor x2^y \rfloor$  (aritmetické posuny, “shifts”)
- úplná:  $+$ ,  $-$ ,  $\text{div } 2$ ,  $\lfloor x2^y \rfloor$ ,  $*$ ,  $\text{div}$
- rozšírená:  $+$ ,  $-$ ,  $\text{div } 2$ ,  $\lfloor x2^y \rfloor$ ,  $*$ ,  $\text{div}$ ,  $x^y$

Väčšinou budeme používať obmedzenú sadu inštrukcií

# Simulácia COMMON na EREW PRAM

Predpokladajme, že v určitom kroku výpočtu môže do istého registra  $\bar{r}$  zapisovať jeden alebo viacej z procesorov  $p_1, p_2, \dots, p_n$ . Zmeňte algoritmus tak, aby nedochádzalo k zapisovacím konfliktom!

- pridáme stromovú štruktúru – každý procesor v strome dohliada na najviac 2 procesory pod ním, či nechcú zapisovať do registra  $\bar{r}$



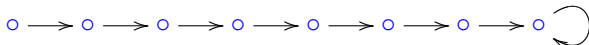
- ak áno, tak ohlásí do vyššieho poschodia, že chce zapisovať hodnotu  $h$
- spomalenie  $O(\log n)$

# Simulácia COMMON na EREW PRAM – pokračovanie

- ako získať pomocné procesory – napr. pôvodným procesorom prideliť procesory s párnym PID a pomocným s nepárnym PID
- podobne sa dá “rozriediť” globálna pamäť → v podstate nepotrebujeme lokálne pamäte

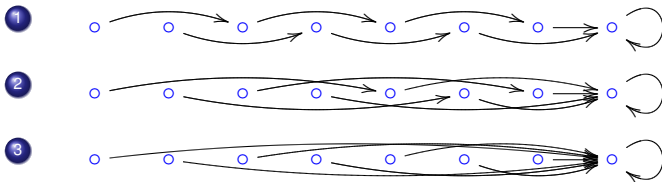
# Dynamicky alokovaná pamäť

väzby medzi procesormi môžu byť vytvorené dynamicky. Napr. lineárny spojový zoznam:



- predpokladáme, že každý prvok  $k$  zoznamu má pridelený procesor a procesor má odkaz na následníka  $P(k)$
- **repeat  $\log n$  times**

**if  $P(P(k)) \neq P(k)$  then  $P(k) := P(P(k))$**



# Sekvenčné modely výpočtov (algoritmov)

- 1930 – Alan Turing – Turingov stroj (TS): rôzne modely líšiac sa počtom hláv, pásov, . . . .  
zložitosť výpočtov:
  - časová –  $SEQTIME(n)$
  - priestorová –  $SPACE(n)$
- neskôr mnoho ďalších: Markovove algoritmy, PL-programy, čiastočne rekurzívne funkcie, RAM (s logaritmickou cenou inštrukcie), . . .  
sú rovnako silné, líšia sa najviac polynomiálne
- **Tvrdenie:** Ak TS rieši daný problém veľkosti  $n$  v čase  $T(n)$ , tak tento problém môže byť vyriešený na ľubovoľnom inom sekvenčnom modeli M v čase  $O([T(n)]^k)$  pre vhodnú konštantu  $k$  závislú iba na modeli M
- **Dôsledok:** Trieda P úloh riešiteľných na TS (RAME s logaritmickou cenou inštrukcie) v polynomickej čase je pojem robustný voči zmene modelu z tejto triedy.



# Sekvenčná a paralelná téza

- **Sekvenčná téza:** Všetky “rozumné” sekvenčné modely algoritmov sú polynomiálne zviazané.  
Pozn.: RAM s jednotkovou cenou inštrukcie nie je “rozumný”.
- **Paralelná téza:** Všetky “rozumné” paralelné modely sa líšia iba polynomiálne a ich časová zložitosť je polynomiálne viazaná na priestorovú zložitosť sekvenčných algoritmov.

$$\text{PARTIME}(T(n)) \leq \text{SPACE}(T(n)^k)$$

$$\text{SPACE}(T(n)) \leq \text{PARTIME}(T(n)^{\tilde{k}})$$

$k$  a  $\tilde{k}$  konštanty

$\text{SPACE}(f(n))$  úlohy riešiteľné sekvenčne v priestore  $O(f(n))$ ,  $n$  je veľkosť úlohy

$\text{PARTIME}(f(n))$  úlohy riešiteľné v paralelnom modeli v čase  $O(f(n))$

# Použitie paralelnej tézy

- kritérium “rozumnosti” modelu
- ak model spĺňa paralelnú tézu:

**Otvorený problém:**  $P \stackrel{?}{=} \text{POLYLOGSPACE}$

**POLYLOGSPACE** označuje triedu úloh riešiteľných v polylogaritmickom priestore  $= \bigcup_k \text{SPACE}(\log^k n)$

iná formulácia problému: *Dá sa každá úloha riešiteľná v “rozumnom čase” (v  $P$ ) exponenciálne zrýchliť na paralelnom modele?*

- ak poznáme sekvenčnú priestorovú zložitosť problému, tak môžeme odhadovať paralelný čas

# Optimálne a efektívne paralelné algoritmy

nech  $n$  označuje veľkosť problému

**NC** – Nick Pippenger's Class

trieda úloh riešiteľných v polylogaritmickej čase s polynomiálnym počtom procesorov (vzhľadom k  $n$ )

- nech  $S$  je problém, pre ktorý najlepší sekvenčný algoritmus má časovú zložitosť  $T(n)$ . Paralelný algoritmus  $A$  riešiaci problém  $S$  v čase  $t(n)$  s  $p(n)$  procesormi sa nazýva

optimálny ak

- $t(n) = \text{polylog}(n)$  – exponenciálne zrýchlenie
- $p(n) \cdot t(n) \in O(T(n))$  – práca vykonaná algoritmom  $A$  je ekvivalentná sekvenčnej zložitosti.

efektívny ak

- $t(n) = \text{polylog}(n)$  – exponenciálne zrýchlenie
- $p(n) \cdot t(n) \in O(T(n)) \cdot \text{polylog}(n)$  – práca vykonaná algoritmom  $A$  je ekvivalentná sekvenčnej zložitosti.

pojmem efektívneho paralelného algoritmu je robustný vzhľadom k “rozumným modelom”, pretože jednotlivé paralelné modely sa dajú navzájom simulovať v polylogaritmickej čase