

# Cache-oblivious algoritmy a ich vizualizácia

Ladislav Pápay

Vedúci: Mgr. Jakub Kováč, PhD.

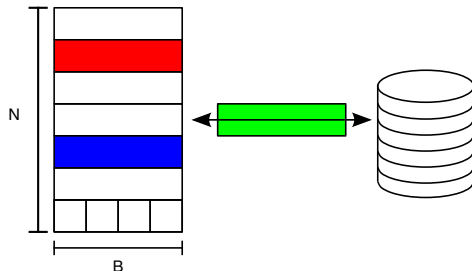
16.04.2014

# Ciele práce

- ▶ Vysvetlenie pamäťového modelu
- ▶ Prehľad cache-oblivious algoritmov a DŠ
- ▶ Vizualizácia vybraných DŠ

# External-memory model

- ▶ Cache celkovej veľkosti  $M$
- ▶ Obsahuje  $N = \frac{M}{B}$  blokov veľkosti  $B$
- ▶ Disk (neobmedzenej) veľkosti
- ▶ Počíta sa počet prenesených blokov (operácie v cache sú zadarmo)



# External-memory model

- ▶ Známe tiež ako cache-aware (kontrast voči cache-oblivious)
- ▶ Poznáme  $B$  a  $M$ , každá operácia čítania/zápisu na disk je explicitne vykonaná algoritmom
- ▶ Ak je cache plná musí vybrať, ktorý blok zahodí

# Problémy

- ▶ Treba explicitne čítať a zapisovať bloky, riešiť asociatívnosť cache
- ▶ Čo ak sa zmenia parametre  $B$  alebo  $M$ ?
- ▶ Čo v prípade viac úrovní? Treba pre každú susediacu dvojicu poznať  $B$ ,  $M$  a spravovať bloky.

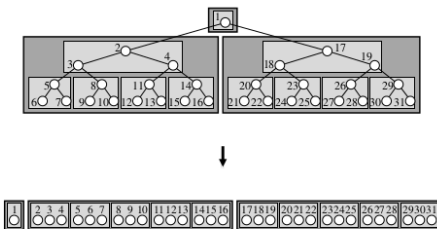
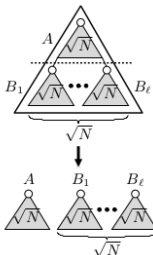
Úroveň	Veľkosť	Odozva
L1	$\approx 128\text{KiB}$	$4\text{clk} \approx 1\text{ns}$ (pri 3.5GHz)
L2	$\approx 1\text{MiB}$	$10\text{-}12\text{clk} \approx 3\text{ns}$ (pri 3.5GHz)
L3	$\approx 8\text{MiB}$	$20\text{-}50\text{clk} \approx 10\text{ns}$ (pri 3.5GHz)
RAM	$\approx 4\text{GiB}$	$\approx 100\text{ns}$
HDD	$\approx 1\text{TiB}$	$\approx 0.1\text{-}10\text{ms}$

# Cache-oblivious model

- ▶ Rovnaká architektúra, presun blokov prebieha automaticky
  - ▶ Predpokladá optimálne nahrádzanie blokov (offline), ale FIFO/LRU je len konštantne horšie
- ▶ Chceme (asymptoticky) rovnaký počet presunov ako optimálny cache-aware algoritmus, ale bez znalosti parametrov  $B$  a  $M$

# Statický strom

- ▶ Binárny vyhľadávací strom
- ▶ V pamäti uložený podľa *van Emde Boas* usporiadania
  - ▶ Rozdelíme uprostred na horný podstrom a  $\sqrt{N}$  spodných podstromov veľkosti  $\sqrt{N}$
  - ▶ Tie rekurzívne rozdelíme a uložíme do súvislého bloku pamäte



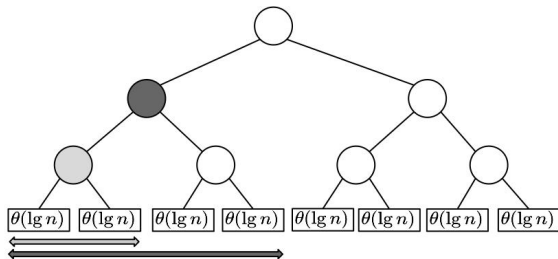
# Analýza

- ▶ Vyhľadávanie ako v klasickom BST (cache-ignorant prístup)
  - ▶ Pozrime sa na takú úroveň delenia, že sú podstromy  $\leq B$  a teda zmestia sa do najviac dvoch blokov
  - ▶ Tieto podstromy majú výšku  $< \lg B$  ale  $\geq \frac{1}{2} \lg B$
  - ▶ Vyhľadávanie prejde cestu od koreňa do listu dĺžky  $\lg N$
  - ▶ Prejdeme najviac cez  $\frac{\lg N}{\frac{1}{2} \lg B}$  podstromov a na každý treba najviac 2 pamäťové presuny
- ▶ Spolu teda  $\mathcal{O}(\log_B N)$  čo je spodná hranica pre external-memory model



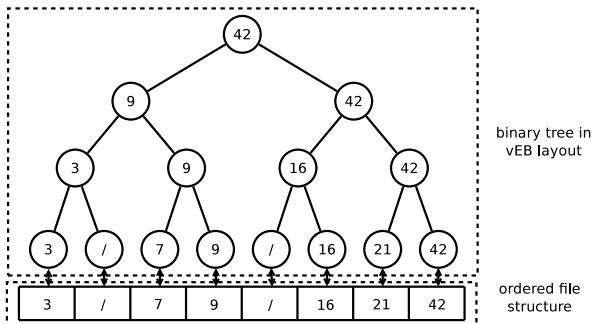
# Ordered file

- ▶ Problém: udržiavať usporiadanú postupnosť prvkov, možnosť vkladať/odstraňovať, ako pole veľkosti  $\mathcal{O}(N) \Rightarrow$  medzery  $\mathcal{O}(1)$ , vieme rýchlo prechádzať
- ▶ Riešenie: rozdelíme súvislé pole na imaginárne bloky veľkosti  $\mathcal{O}(\log N)$  a vyrobíme nad nimi imaginárny úplný binárny strom
- ▶ *Hustota vrcholu* bude  $\frac{\text{počet plných}}{\text{kapacita}}$ , udržiavame podľa istých hraníc aby nebolo príliš plné (nie je kam vložiť) ani príliš prázdne (pomalé prechádzanie)
- ▶ *Vloženie prvku*: Upravíme blok a postupujeme hore v strome, kým nenájde vrchol, ktorý má hustotu v medziach a rovnomerne prerozdelíme prvky v príslušných blokoch
- ▶ *Zmazanie prvku* podobne
- ▶ Každá operácia upraví súvislý interval veľkosti  $\mathcal{O}(\lg^2 N)$  (amortizovane, dá sa aj worst-case)

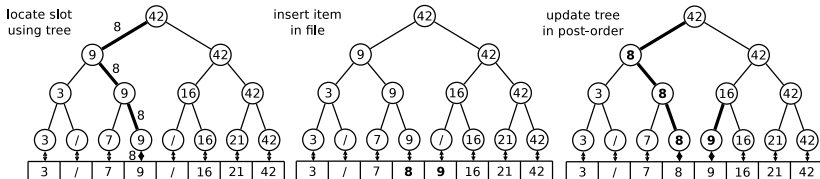


# Dynamický strom

- ▶ Skombinujeme predošlé dve štruktúry a dostaneme dynamický vyhľadávací strom
- ▶ Máme OF v ktorom sú kľúče a medzery, použijeme ako listy pre BST uložený v vEB layoute
- ▶ Vo vnútorných uzloch ukladáme maximum z podstromov



- Vyhľadávanie: rovnako ako v statickom  $\mathcal{O}(\log_B N)$
- Vkladanie
  - Nájdeme nasledujúci kľúč v OF, vložíme nový za neho
  - Musíme upraviť strom pre zmenený interval veľkosti  $\mathcal{O}(\lg^2 N)$
  - Prechod cez zasiahnuté vnútorné uzly vyžaduje  $\mathcal{O}(\frac{\lg^2 N}{B})$  pamäťových operácií
- Vkladanie a odstraňovanie spolu potrebuje  $\mathcal{O}(\log_B N + \frac{\lg^2 N}{B})$
- Dá sa zredukovať na  $\mathcal{O}(\log_B N)$  ak ukladáme  $\mathcal{O}(\log N)$  bloky



# Vizualizácie