CACHE-OBLIVIOUS ALGORITMY

Dnešní procesory jsou řádově rychlejší než paměti (10cm je hrozná dálka...)
→ zavedeme keš (cache), která je menší a rychlejší (blíž procesoru)
a necháme ji pamatovat si často používaná data.

Zjednodušený princip keše: (jen čtení)

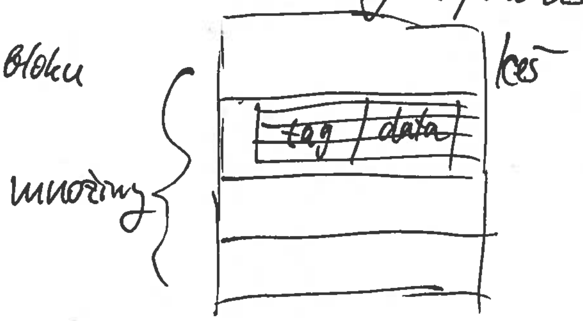
- pracuje po blocích velikosti B (typicky 64 bytů), blok si pamatuje celý/vůbec ("cache line")
- adresu dělíme | tag | 6 | (tag, pozice v bloku)
- keš ukládá dvojice (tag, obsah bloku)
- požadavek na čtení:
  - hledáme v keši blok s daným tagem ("asociativní paměť")
  - pokud tam není, čteme z hlavní paměti a uložíme do keše
  - dojde-li místo, něco vyhodíme
    - typicky nejdéle nepoužitý blok (LRU = Least-Recently Used)

Zápisy:

- nechceme kešovat částečné bloky ⇒ blok nejdříve přečteme
- zapamatujeme si, že je modifikovaný
- kdykoli později zapíšeme do paměti (write back), nejpozději při vyhození z keše

Omezená asociativita:

- plně asociativní keš je příliš složitá ⇒ rozdělíme keš na množiny, adresa určí množinu (řekněme 16 bloků), uvnitř množiny už plně asociativní
- adresu dělíme | tag | index množiny | pozice v bloku |



- příklad: 32b adresy
  64B bloky
  64 KiB keš → 1024 bloků → 64 množin → index má 6 bitů
  16-cestná (množiny velikosti 16)

20	6	6
tag	index	pozice

! pozor, adresy lišící se o násobek 4 KiB (2^{12}) mají stejný index – "aliasing"
– projevuje se např. při procházení matice $2^k \times 2^k$ po sloupcích
(efektivně daleko menší keš)

Víceúrovňové keše

typické parametry
L1 32 KiB
L2 256 KiB
L3 8 MiB
↓ větší, dál od procesoru, pomalejší

Tzv. paměťová hierarchie – můžeme do ní počítat i disky, síťová úložiště apod.

Příklady – jednoduchý program procházející pole ~~bloků~~ položek pevné velikosti ②
↳ z bloku vždy přečte 32b hodnotu

- obr. 1: ~~bloky~~ položky velikosti 64B – je vidět přechod L1/L2/L3/RAM
- obr. 2: vliv velikosti ~~bloků~~ položek – 64 a 128 se v keši chovají stejně (v RAM jinak)
  16 je lepší (více ~~bloků~~ položek na 1 blok keše)
- obr. 3: více velikostí položek – proč mají 64 a 128 stejná rozlišení?
  – vliv MMU, TLB … za to může aliasing
- obr. 4: sekvenční/náhodný přístup, čtení/zápis
  - sekvenční/náhodný: v keši stejně rychlé, v RAM se výrazně liší
  - zápis je pomalejší (např. proto, že nejdřív čteme)

MODELOVÁNÍ KEŠÍ

RAM s interní a externí pamětí: I/O model

- externí paměť: neomezeně bloků velikosti B (jednotku vhodně zvolíme), je pomalá
- interní paměť velikosti M (obvykle si představujeme M/B bloků velikosti B)
- v interní umíme počítat + přenášíme bloky mezi int/ext pamětí
- nová míra složitosti: # I/O (přesunů bloků)
  - obvykle BÚNO počítáme jen čtení (ale pozor na výstup ⇒ vstup)

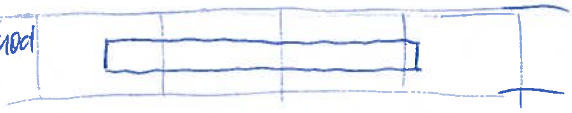
Model s keší

- opět int+ext paměť, ~~ale~~ program adresuje tu externí, automaticky se kešuje v interní
- předpokládáme optimální obsluhu keše (to je dosti naivní, ale časem dokážeme, že LRU je obvykle jen konst-krát horší)
- opět měříme I/O složitost
- zná program parametry keše? – cache-aware (zná M, B)
  – cache-oblivious (nezná nic)

↳ Chceme najít algoritmus (C/A, nebo lépe C/O), který má dobrou časovou, prostorovou i I/O složitost pro každé "rozumné" parametry keše.

SCAN POLE

sekvenční průchod polem



v I/O | C/A zarovnáme začátek → $\lceil N/B \rceil$ bloků

v C/O to neumíme → $\lceil N/B \rceil + 1$ bloků

- otočení pole: 2 proložené scany ⇒ $O(N/B + 1)$ [! pozor, +1 je nutné]
  ↑ potřebujeme keš na ≥2 bloky najednou (typický předpoklad)
- slévání setříděných posloupností: 3 scany ⇒ $O(N/B + 1)$ (má-li keš aspoň 3 bloky)
- Mergesort v nerekurzivní podobě: $\log N \times$ slévání ⇒ $O((N/B + 1) \log N)$
  ↑ pozor, záleží na uložení v paměti; v nerek. verzi je to jasné
  lze "vysunout ven": $N/B \leq 1$ ⇔ vstup má ≤1 blok, takže vše spočítáme v keši
- lze zlepšit: (M/B)-cestný Mergesort (sléváme $O(M/B)$ posl. pomocí haldy … to se stále ukešuje)
  to je ovšem C/A

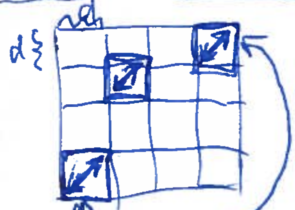
$\#I/O \in O\left(\frac{N}{B} \cdot \log_{M/B} N + 1\right)$ čas $O\left(N \cdot \log_{M/B} N \cdot \log \frac{M}{B}\right) = O(N \log N)$

– téže složitosti lze dosáhnout i C/O algoritmem (Funnelsort), ale je složitý ③
– dolní odhad (dokonce pro permutování!) odpovídá ⇒ alg. je optimální

TRANSPOZICE MATICE

– matice uložena v paměti souvisle po řádcích … zatím tvaru $N \times N$
  – průchod po řádcích je scan ⇒ přečte $\lceil N^2/B \rceil + 1$ bloků
  – průchod po sloupcích: pokud $N \gg M/B$, skoro nic se nekešuje (do dalšího sloupce v keši zůstane jen M/B bloků)
  – transponování dle definice kombinuje obojí ⇒ je pomalé

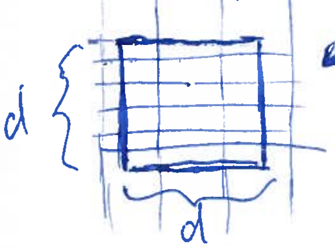
Cache-aware alg.:



- matici rozdělíme na ~~bloky~~ dlaždice $d \times d$ (d určíme později, na krajích ~~částečné bloky~~ neúplné dlaždice)
- dlaždice na diagonále transponujeme
- ty mimo diag. navíc prohazujeme

① pokud $B \mid N$, zvolíme $d := B$ a vše zarovnáme na bloky
… potřebujeme $M \geq B^2 + O(1)$ … tedy štíhlou keš "tall cache"
Tehdy na 1 dlaždici potřebujeme B čtení z paměti ⇒ celkem $\left(\frac{N}{B}\right)^2 \cdot B = \underbrace{\frac{N^2}{B}}_{\text{evidentně optimální}}$

② obecný případ: nelze zajistit zarovnání



na každém řádku 1 blok navíc
⇒ do keše se musí vejít $d^2 + dB$ prvků
Nastavíme $d := \alpha B$. Potřebujeme $d^2 + dB \leq M$

$$\alpha^2 B^2 + \alpha B^2 \leq M$$
$$(\alpha^2 + \alpha) B^2 \leq M$$

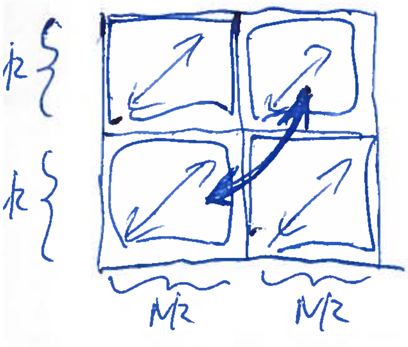
Jelikož $B^2 \leq M$, stačí $\alpha^2 + \alpha \leq 1$
⇒ zvolíme $\alpha := \frac{1}{2}$: $\alpha^2 + \alpha = 3/4$
⇒ zbude dalších $O(1)$ bloků na režii

↳ na 1 dlaždici potřebujeme $O(B)$ čtení,
celkem $(\lceil N/B \rceil)^2 \cdot O(B) = O\left(\frac{N^2}{B}\right)$, což je optimální.
Časová složitost je stále $\Theta(N^2)$

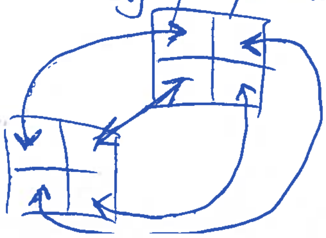
- stejným způsobem můžeme zařídit, aby se do keše vešlo $O(1)$ dlaždic
- Ono štíhlá cache funguje každá s $M = \Omega(B^2)$

Cache-oblivious alg.:

Neznáme vhodné d ⇒ dělíme rekurzivně na čtvrtiny



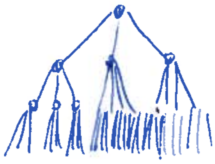
- Zatím předpokládáme $N = 2^k$ ⇒ všechny dlaždice jsou shodné čtverce
- Vznikají podproblémy typu "transponuj a prohoď":



4 podproblémy velikosti $N/2$, dělením trávíme čas $O(1)$

- rekurzi zastavíme na triviálních případech
- strom rekurze: $O(N^2)$ listů, vnitřních vrcholů $\le$ # listů $\Rightarrow$ časová složitost $O(N^2)$

mají 3 nebo 4 syny



← na i-té hladině podproblémy velikosti $N/2^i$ $\Rightarrow$ hloubka $\log N$

- je jich nejvýše 4^i $\Rightarrow$ listů je nejvýše $4^{\log N} = N^2$

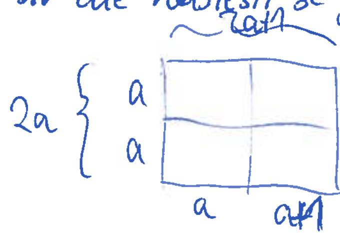
- I/O složitost … "zaostříme" na nejvyšší hladinu, kdy se dlaždice vejdou do keše — všude pod tím též

  podle předchozího $d \le \frac{B}{2}$

  - tedy: $\frac{N}{2^i} \le \frac{B}{2}$, ale $\frac{N}{2^{i-1}} > \frac{B}{2}$ … $\frac{N}{2^i} \le \frac{B}{2} < \frac{N}{2^{i-1}}$ … $\frac{2N}{B} \le 2^i < \frac{4N}{B}$
  - Na této hladině je $\le 4^i$ podproblémů, což je $\Theta\left(\frac{N^2}{B^2}\right)$
  - V každém $\Theta(B)$ I/O $\Rightarrow$ celkem $\Theta\left(\frac{N^2}{B^2}\cdot B\right) = \Theta\left(\frac{N^2}{B}\right) + 1$

    v celém jeho podstromu se vše vejde do keše
    $\Rightarrow$ stačí to načíst jen 1× a podstrom spočítat v keši

- zbývá dořešit případ obecného N: tehdy jsou některé matice obdélníkové … ale naštěstí se jejich rozměry vždy liší jen o 1



VYHLEDÁVÁNÍ v setříděném poli (obecně: statická DS pro množinu)

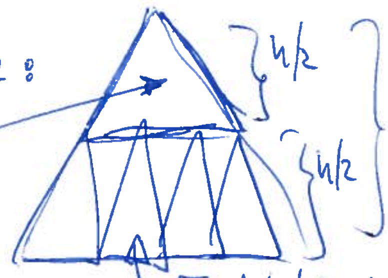
- binární vyhledávání – vzdálenosti mezi "sondami" do pole jsou postupně $\frac{N}{2^i}$
  - jen posledních $\log B$ je ve stejném bloku
  - celkem $\Theta(\log N - \log B + 1) = \Theta(\log \frac{N}{B} + 1)$ přístupů do paměti, $\Theta(\log N)$ času.
- binární strom uložený v poli po patrech (jako halda) se chová stejně špatně
- C/A: (a,b)-strom – zvolíme a,b tak, aby se 1 vrchol vešel do $\Theta(1)$ bloků.
  - hloubka vyjde $\log_B N$
  - celkem $\Theta(\log_B N + 1) = \Theta\left(\frac{\log N}{\log B} + 1\right)$ přístupů, $\Theta\left(\frac{\log N}{\log B}\cdot \log B\right) = \Theta(\log N)$ času.
  - lépe to nejde: z 1 bloku získáme $\le \log B$ bitů informace (kam lze zařadit x mezi prvky v bloku?)

    může být korelovaná s již známými věcmi

    potřebujeme $\log N$ bitů $\Rightarrow$ musíme se podívat do $\Omega\left(\frac{\log N}{\log B}\right)$ bloků.
- C/O řešení – myšlenka: $\sqrt{N}$-ární strom … má 2 patra, v $\forall$ vrcholu stejná struktura pro $\sqrt{N}$ prvků
  → rekurze hloubky $\log \log N$, ~~…~~ na dně binární vrcholy
- ukážeme ekvivalentní popis pomocí binárních stromů.

Van-Emde Boasovo uložení bin. stromů

- mějme úplný bin. strom hloubky $h = 2^\ell$ ← to je dost silný předpoklad, časem ho odstraníme
  - ↑ to je BÚNO
- rozřízneme v hloubce $h/2$:

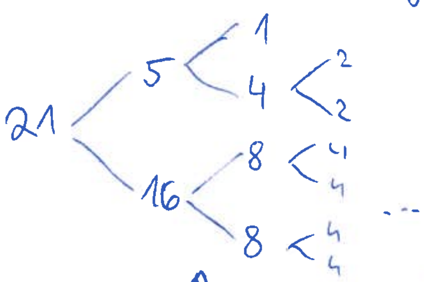


horní strom s $\sqrt{N}$ vrcholy (řádově) (funguje jako kořen (a,b)-stromu)

$\sqrt{N}$ dolních stromů o $\sqrt{N}$ vrcholech ← (listy (a,b)-stromu)

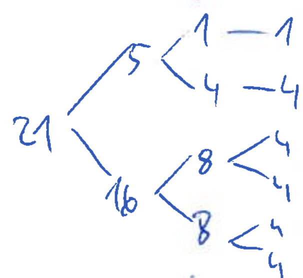
→ v paměti nejprve horní strom, pak v libovolném pořadí dolní stromy, vše rekurzivně rozkládáme dál
↓
hloubka dekompozice $\log h = \log\log N$

- analýza průchodu po cestě kořen-list (cvičení: implementujte rychlý přepočet mezi pozicí vrcholu ve stromu a adresou ve vEB uložení)
  - zaostříme na nejvyšší úroveň rozkladu, kdy se podstrom vejde do bloku
    → tedy podstromy hloubky h t.ž. $2^h \le B < 2^{2h}$
    $h \le \log B \le 2h$
    $h \in \Theta(\log B)$
  - cesta postupně prochází podstromy této hloubky → navštíví jich $\Theta\left(\frac{\log N}{\log B}\right)$
    → tolik je přístupů do paměti
- co když h není mocnina dvojky?
  - hloubku dolního stromu zvolíme jako $h/2$ zaokrouhleno na nejbližší vyšší mocninu dvojky (tedy horní nižší)
  - příklad:



kromě nejvyššího stromu jsou vše mocniny 2

můžu je „posunout do správné úrovně rozkladu"



pak stále funguje předchozí analýza, jen musím přičíst +1 za výjimečný nejhorejší strom

REALISTIČNOST MODELU – LRU kontra opt. strategie

Věta (Sleator, Tarjan 1985):

Nechť $s_1 \dots s_k$ je posloupnost adres, na něž přistupujeme (stačí čísla bloků v ext. paměti)

P_{OPT}, P_{LRU} jsou počty bloků v keši pro algoritmy OPT a LRU

T_{OPT}, T_{LRU} počty přenosů bloků pro oba algoritmy

Potom
$$T_{LRU} \le \frac{P_{LRU}}{P_{LRU} - P_{OPT} + 1}\, T_{OPT} + P_{OPT}$$

zmizí, pokud je keš zpočátku prázdná

⇒ ve 2x větší keši je LRU $O(1)$-krát horší (+aditivní konst.), naše c/o algoritmy 2x menší keš ovlivní jen $O(1)$-krát ⇒ celkem $O(1)$ oproti OPT

~~Důkaz~~ Důkaz: Rozdělíme posloupnost přístupů na fáze F_0 — F_z tak, aby ve fázi LRU četlo právě P_{LRU} bloků. Výjimka pro F_0, kde se čte nejvýše tolik. (tedy dělíme na fáze odzadu)

- Zaměřme se na fázi F_i ($i>0$)
  - na počátku fáze mají LRU i OPT v keši nějaký společný ~~...~~ blok **b**. (posledu, k němuž se přistoupilo v F_{i-1})
  - LRU čte P_{LRU}-krát, ukážeme, že OPT aspoň $(P_{LRU} - P_{OPT} + 1)$-krát:

    (1) Pokud LRU čte během F_i ~~...~~ blok **b**:

    ~~Muselo přistoupit k aspoň~~

    Na zač. F_i byl **b** nejnovější ⇒ muselo ho předběhnout aspoň P_{LRU} jiných bloků

    Ve fázi se tedy přistupuje k aspoň $P_{LRU}+1$ různým blokům, z nichž nejvýše P_{OPT} měl v paměti OPT ⇒ OPT čte aspoň $(P_{LRU}+1-P_{OPT})$-krát

    (2) Pokud LRU čte 2x tentýž blok: opět aspoň $P_{LRU}+1$ ~~...~~ různých bloků

    stejný argument

    (3) Jinak čte LRU P_{LRU} různých bloků, z nichž žádný není **b** … ale OPT má na začátku v paměti **b** ⇒ čte aspoň $(P_{LRU} - (P_{OPT}-1))$-krát.

- Ve fázi F_0: všechny bloky navzájem různé … ale OPT může mít až P_{OPT} z nich v keši. Tedy OPT čte aspoň $(\#\text{bloků v } F_0) - P_{OPT}$ -krát.

Pro naše účely by stačilo slabší tvrzení:

$$T_{LRU} \le \frac{P_{LRU}}{P_{LRU} - P_{OPT}} \cdot T_{LRU} + P_{OPT}$$

Pak není potřeba v důkazu rozebírat případy tak složitě – není třeba brát v úvahu **b**

Buď čteme nějaký blok 2x ⇒ potkali jsme aspoň P_{LRU} různých bloků
nebo nečteme ⇒ platí totéž

OPT měl nejvýše P_{OPT} z nich v keši ⇒ čte aspoň $(P_{LRU} - P_{OPT})$-krát