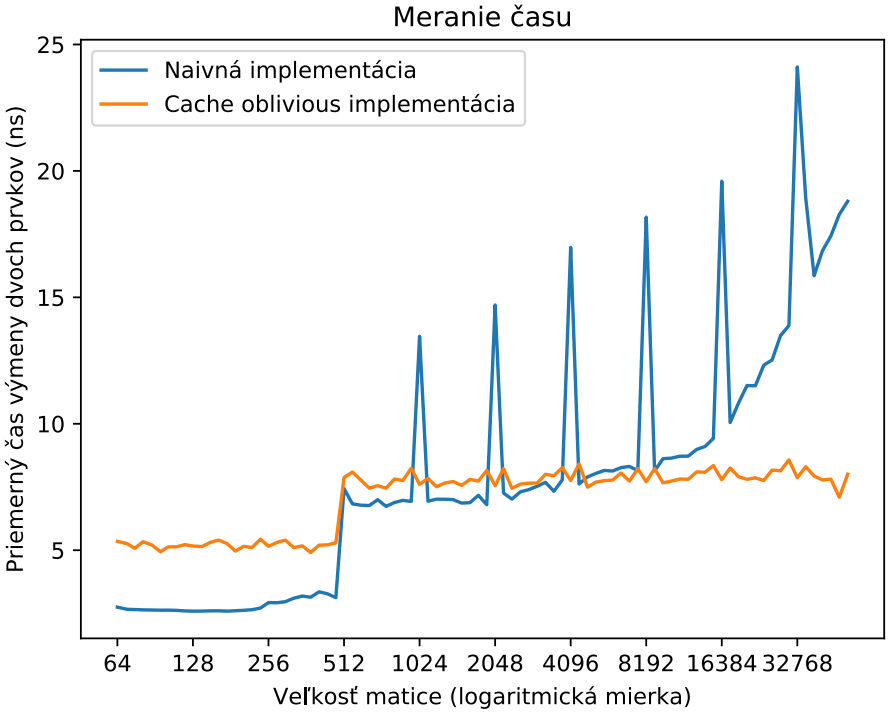
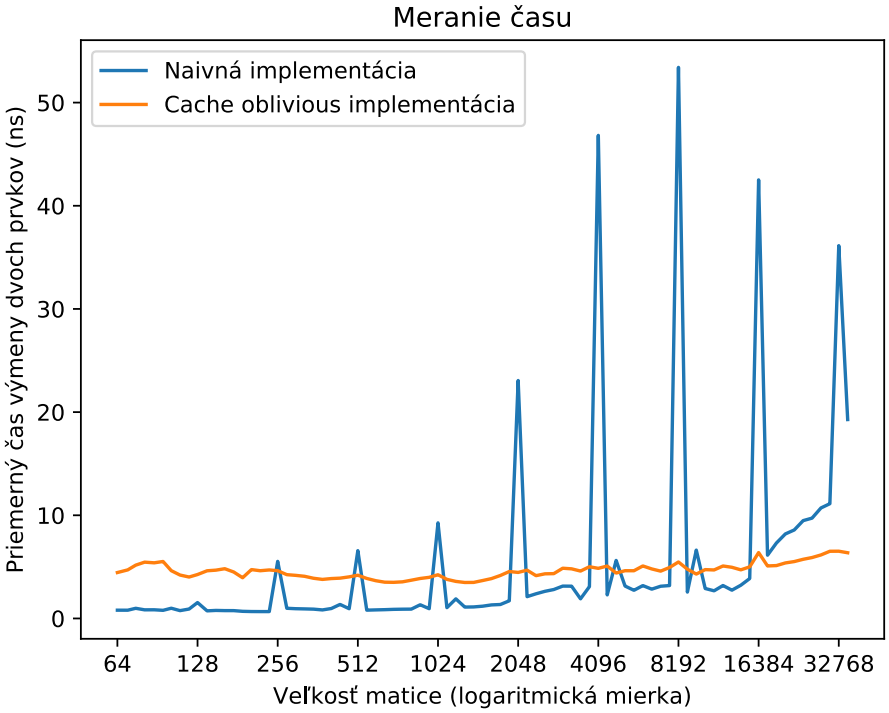
1 Transpozícia matice

1.1 Meranie času

Ako prvé uvádzam meranie na mojom počítači: Intel Core i5-6600K@4.4GHz, 6MB cache, 16GB RAM, Windows 10.



Ďalej uvádzam meranie na počítači v labe: Intel Core i7-6700@3.4 GHz, 8MB cache, 16GB RAM, linux.



Môžeme pozorovať niekoľko vecí:

- Na prvom grafe pozorujeme rovnako pri naivnom aj rekurzívnom algoritme veľký skok pri veľkosti matice 512. Vtedy má matica 512*512*4=1MB, čo nie je veľkosť žiadnej z keší (L1 má 32 kB, L2 256 kB, L3 6 MB). Je tiež zvláštne, že skok nastane pri *oboch* algoritmoch. Bohužiaľ sa mi nepodarilo nájsť odpoveď na to, prečo to tak je. Na počítači v labe som nič také nenameral, a tak si myslím, že je to niečo špecifické pre implementáciu procesoru, prekladača, alebo operačného systému, a ďalej sa tým v analýze nebudem zaoberať.

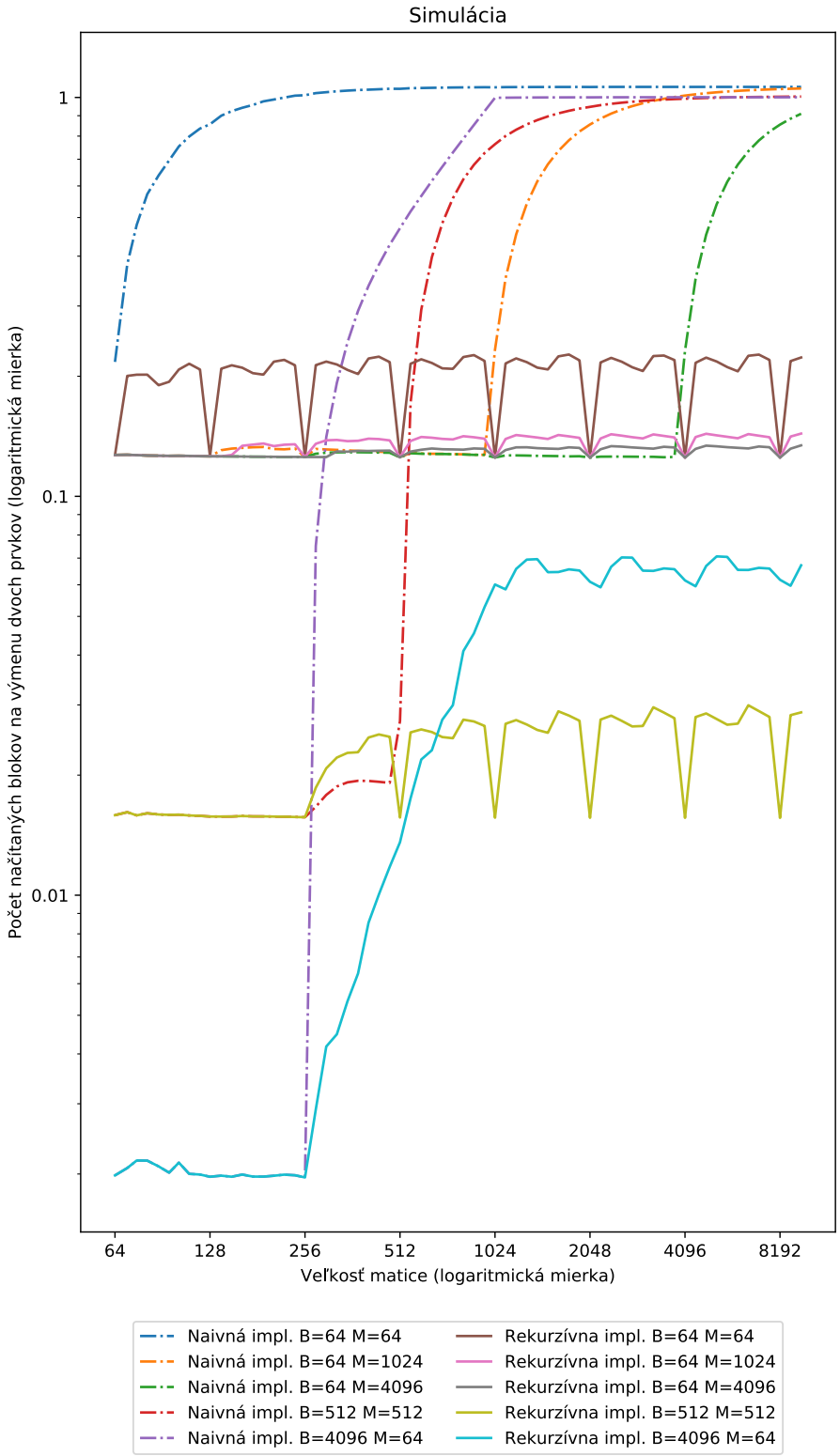
- Priemerný čas jednej výmeny rekurzívneho cache-oblivious algoritmu (podľa očakávania) nezáleží na veľkosti matice.

- Naivný algoritmus sa správa zle, ak je počet riadkov N matice mocninou 2. To sa deje kvôli tomu, že cache procesoru nie je plne asociatívna. Pri iterovaní vnútorného cyklu postupne pristupujeme ku prvej, druhej, až poslednej N-tej bunke toho istého stĺpca matice. Keďže je matica uložená po riadkoch, tak to znamená, že pri posune o jeden riadok

dolu skočíme v pamäti o N buniek pamäte. Ak je však N násobkom počtu blokov cache, tak kvôli obmedzenej asociativite cache sa začnú vyhadzovať z cache bloky z toho istého stĺpca, hoci je zaplnený iba zlomok celkovej veľkosti cache.

- Na oboch grafoch vidíme, že naivný algoritmus je rýchly, pokiaľ sa dostatočne veľká časť matice zmestí do cache. Do cache sa nemusí zmestiť celá matica, algoritmus však potrebuje, aby sa do cache zmestil stĺpec blokov cache, a korešpondujúce riadky (potrebujeme toľko riadkov, koľko je dĺžka jedného riadku/bloku cache). Vtedy je počet cache miss minimálny. Na grafoch vidíme, že pre väčšie N už počet cache miss narastá a tým aj čas na jednu výmenu.

2 Simulátor


Simulácia
Počet načítaných blokov na výmenu dvoch prvkov (logaritmická mierka)
1
0.1
0.01
64
128
256
512
1024
2048
4096
8192
Veľkosť matice (logaritmická mierka)
Naivná impl. B=64 M=64
Naivná impl. B=64 M=1024
Naivná impl. B=64 M=4096
Naivná impl. B=512 M=512
Naivná impl. B=4096 M=64
Rekurzívna impl. B=64 M=64
Rekurzívna impl. B=64 M=1024
Rekurzívna impl. B=64 M=4096
Rekurzívna impl. B=512 M=512
Rekurzívna impl. B=4096 M=64

- Vidíme, že kým je matica menšia ako veľkosť cache, nie sú medzi algoritmami žiadne rozdiely: pre (B=64,M=64) túto situáciu graf nezachytáva, pre (B=64,M=1024) je to pre N=128, teda matica má 128*128*4 B = 64*1024 B. Pre zvyšné verzie parametre cache to nastáva pri N=256, keď cache má 256 kB

- Po tomto bode už záleží na parametroch cache. Napráklad aj naivný algoritmus pre cache (B=64,M=4096) a (B=64,M=1024) sa správa dobre, pretože sa tu uplatní vyššie popísaný princíp toho, že naivný algoritmus potrebuje, aby sa do cache zmestilo iba niekoľko stĺpcov a prislúchajúcich riadkov v závislosti od dĺžky bloku. Naproti tomu cache (B=4096,M=64) a čiastočne (B=512,M=512) nemajú dosť blokov aby poskytli naivnému algoritmu dobré správanie už pre malé veľkosti matice.

- Vidíme potvrdenie analýzy z prednášky, že naivný algoritmus potrebuje až N^2 prístupov do cache: Pre akékoľvek parametre cache sa krivky naivných algoritmov blížia k 1 s rastúcim N.

- Cache-oblivious algoritmus má menej cache missov, keď je N mocninou 2. To je pravdepodobne spôsobené tým, že podštvorce matice, ktoré sú spracovávané rekurzívne a bloky cache sú zarovnané, a teda sa nestáva, že blok, z ktorého sme potrebovali iba časť v predchodzom štvorci, načítame ešte raz v nasledujúcom (samozrejme pre štvorce väčšie ako jeden blok cache).