Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra ... (DOPLŇTE)

Bakalářská práce

Doplňte název práce

Doplňte Vaše jméno a tituly

Vedoucí práce: Doplňte jméno vedoucího práce

6. dubna 2014

Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstraňte tento příkaz.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či spracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. dubna 2014 .....

Odkaz na tuto práci

Doplňte Vaše příjmení, Doplňte Vaše křestní jméno/jména. *Doplňte název práce*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstrakt

V několika větách shrňte obsah a přínos této práce v češtině. Po přečtení abstraktu by se čtenář měl mít čtenář dost informací pro rozhodnutí, zda chce Vaši práci číst.

Klíčová slova Nahraďte seznamem klíčových slov v češtině oddělených čárkou.

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords Nahraďte seznamem klíčových slov v angličtině oddělených čárkou.

Obsah

Seznam obrázků

Úvod

Kapitola 1

Úvod do problematiky

1.1 Použití násobení matic

TODO: kde se pouziva nasobeni matic

1.2 Matice

Matice **A** typu $(m,\ n)$ je mn uspořádaných prkvů z množiny **R**. O prvku $a_{r,s} \in \mathbf{R}, r \in \{1, 2, \ldots, m\}, s \in \{1, 2, \ldots, n\}$ říkáme, že je na r-tém řádku a s-tém sloupci matice **A**. Matici **A** zapisujeme do řádků a sloupců takto:

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \tag{1.1}$$

Matici **M** typu $(m,\ n)$, kde všechny její prvky jsou rovny nule, nazýváme *nulovou maticí.*

O matici typu $(m,\ n)$ budeme říkat, že je m široká a n vysoká. Pokud o matici řekneme že má velikost n, myslíme tím, že je typu $(n,\ n)$.

1.3 Vektor

Matici **V** typu (1, n) nazveme vektorem.

TODO=popsat vektory poradne FIXME=muzu to takhle zjednodusit?

1.4 Násobení matic

Buď **A** matice typu (m,n) s prvky $a_{i,j}$ a **B** matice typu (n,p) s prvky $b_{j,k}$. Definujeme součin matic $\mathbf{A} \cdot \mathbf{B}$ jako matici **C** typu (m,p) s prvky $c_{i,k}$ které vypočteme jako:

$$c_{row,col} = \sum_{k=1}^{N} a_{row,k} b_{k,col} \tag{1.2}$$

Výsledek součinu matic se nezmění, pokud matice doplníme o libovovolný počet nulových řádků a nebo sloupců. Této vlastnosti můžeme využít pro získání potřebných rozměrů:

1. Při násobení matice A typu (m,n) s maticí B typu (o,p), kde $n \neq o$.
2. Pokud potřebujeme matice stejné velikosti.
3. Pokud potřebujeme matice určité velikosti, například $2^{\mathbf{N}}$.

1.5 Složitosti

TODO: popsat notace

1.6 Řídké matice

Matice, které obsahují velké množství nulových prvků, nazýváme řídké. Nebudeme přesně uvádět kolik procent z celkového počtu prvků musí být nulových, abychom matici nazývali řídkou. Stejně jako řídkou matici můžeme uložit do formátu pro husté matice, můžeme hustou matici uložit do formátu pro řídké matice.

Řídkost matice budeme vyjadřovat pomocí nnz (Number of NonZero elements), tedy počtem nenulových prvků z celkových mn, pro matici A typu (m, n).

TODO: typy ridkych matic (pasova, atd, pattern, real)

1.7 Numerická stabilita

TODO: numerická stabilita (viz strassen?)

1.8 Optimalizace kódu

Dnešní překladače umí velice dobře optimalizovat vygenerovaný kód. Pokusy o nějaké mikrooptimalizace program spíše zpomalí.

Je vhodné používat funkce standartních knihoven, protože bývají optimalizované přímo v assembleru.

1.8.1 Rozděl a panuj

divide, conquer, combine

1.8.2 Rozbalování cyklů

1.8.3 AoS -> SoA

1.8.4 Loop tiling

Kapitola 2

Algoritmy násobení matic

2.1 Podle definice

Základním algoritmem násobení dvou matic je podle definice. Ve třech for cyklech postupně vybíráme řádky matice A, sloupce matice B a v N krocích násobíme. N je jak šířka matice A, tak i výška matice B.

Algorithm 1 Násobení matic podle definice

```
1:  procedure MMM-definition(A, B, C)                ▷ A,B,C jsou matice
2:      for row ← 0 to A.height do                    ▷ řádky
3:          for col ← 0 to B.width do                 ▷ sloupce
4:              sum ← 0;
5:              for i ← 0 to A.height do
6:                  sum ← sum + A[row][i] * B[i][col];
7:              end for
8:              C[row][col] ← sum;
9:          end for
10:     end for
11: end procedure
```

Z pseudokódu je vidět, že ve dvou for cyklech provádíme N násobení a N sčítaní. Asymptotická složitost je tedy $O(n^2(n+n)) = O(2n^3)$. V ukázkových výpočtech je násobení pouze $N-1$ krát, to proto, že neuvádíme přičítání k nule (řádek 6).

2.2 Násobení transponovanou maticí

Pokud nám formát uložení matice nedovolí procházet prvky po sloupcích, je řešením druhou matici transponovat. Poté můžeme násobit řádky matice A s řádky transponované matice B.

Algorithm 2 Násobení transponovanou maticí

```
1: procedure MMM-TRANSPOSE(A, B, C)                    ▷ A,B,C jsou matice
2:     B ← transpose(B)
3:     for rowA ← 0 to A.height do                              ▷ řádky
4:         for rowB ← 0 to B.height do                          ▷ sloupce
5:             sum ← 0;
6:             for i ← 0 to A.height do
7:                 sum ← sum + A[rowA][i] * B[i][rowB];
8:             end for
9:             C[rowA][rowB] ← sum;
10:        end for
11:    end for
12: end procedure
```

Podobný algoritmus můžeme použít i pokud nám formát nedovolí procházet prvky po řádcích, ale pouze po sloupcích. Například v této práci neuvedený Compresed Sparse Columns.

Pro matice musí platit, že výška matice A musí být stejná jako výška matice B. (FIXME: je to opravdu tak?)

2.3 Násobení po řádcích

Další možností jak násobit dvě matice, kde nám formát uložení nedovolí procházet po sloupcích je procházet současně řádky matice A i B a přičítat jednotlivé součiny na správné místo ve výsledné matici C.

Nevýhodou tohoto řešení je velký počet přístupů do pole C. Protože k prvkům přičítáme, tedy načítáme a sčítáme, je potřeba před samotným násobením nastavit všechny prvky matice C na hodnotu nula.

Algorithm 3 Násobení po řádcích

```
1: procedure MMM-BY-ROWS(A, B, C)                      ▷ A,B,C jsou matice
2:     for r ← 0 to A.height do                    ▷ řádky matice A i B
3:         for cA ← 0 to A.width do                  ▷ sloupce matice A
4:             for cB ← 0 to B.width do              ▷ sloupce matice B
5:                 C[r][cA] ← C[r][cA] + A[r][cA] * B[r][cB];
6:             end for
7:         end for
8:     end for
9: end procedure
```

2.4 Rekurzivní násobení

Pro matice A i B o stejé velikosti $2^{\mathbf{N}}$ můžeme použít rekurzivní přístup. Tedy programovací techniku rozděl a panuj, kdy rozdělíme větší problémy na menší podproblémy.

Každou z matic rozdělíme na čtvrtiny a jednotlivé podmatice násobíme algoritmem podle definice, tedy jako matice o velikosti dva.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix} \tag{2.1}$$

Tento postup opakujeme, dokud velikostí podmatic nenarazíme na práh, tedy hodnotu, při které opustíme rekurzivní algoritmus a použijeme algoritmus lineární. V ukázkovém pseudokódu dělíme podmatice až na velikost prahu jedna, podmatice tedy obsahují pouze jeden prvek.

Algorithm 4 Rekurzivní násobení

```
1: procedure MMM-RECURSIVE(A, B, C, ay, ax, by, bx, cy, cx, n)
2:     if n = 1 then
3:         C[cy][cx] ← C[cy][cx] + A[ay][ax] · B[by][bx];
4:         return;
5:     end if
6:     for all r ∈ {0, n/2} do
7:         for all c ∈ {0, n/2} do
8:             for all i ∈ {0, n/2} do
9:                 MMM-recursive(A, B, C, ay + i, ax + r, by + c, bx + i, cy +
   c, cx + r, n/2);
10:            end for
11:        end for
12:    end for
13: end procedure
```

Pro ilustraci jako příklad uvádíme výpočet horního levého prvku v násobení dvou matic o velikosti 2^2. Pro větší přehlednost značíme prvky malým písmem z názvu matice a indexy o jejich pozicích.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} \end{pmatrix} = \tag{2.2}$$

$$\begin{pmatrix} \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} + \begin{pmatrix} a_{1,3} & a_{1,4} \\ a_{2,3} & a_{2,4} \end{pmatrix} \cdot \begin{pmatrix} b_{3,1} & b_{3,2} \\ b_{4,1} & b_{4,2} \end{pmatrix} & \cdots \\ \cdots & \cdots \end{pmatrix} = \tag{2.3}$$

$$\begin{pmatrix} \begin{pmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & \cdots \\ \cdots & \cdots \end{pmatrix} + \begin{pmatrix} a_{1,3}b_{3,1} + a_{1,4}b_{4,1} & \cdots \\ \cdots & \cdots \end{pmatrix} & \cdots \\ \cdots & \cdots \end{pmatrix} = \tag{2.4}$$

$$\begin{pmatrix} \begin{pmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} + a_{1,3}b_{3,1} + a_{1,4}b_{4,1} & \cdots \\ \cdots & \cdots \end{pmatrix} & \cdots \\ \cdots & \cdots \end{pmatrix} \tag{2.5}$$

Kvůli režii rekurzivního dělení v praxi nezmenšujeme podmatice až na velikost jedna. Vhodný práh velikosti podmatice je například takový, co se vejde do L1 cache.

Asymptotická složitost je samozřejmě stejná jako u algoritmu podle definice. Asymptotickou složitost rekurzivního algoritmu můžeme spočítat pomocí mistrovské metody.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 8T(n/2) + \Theta(1) & \text{if } n > 1 \end{cases}$$

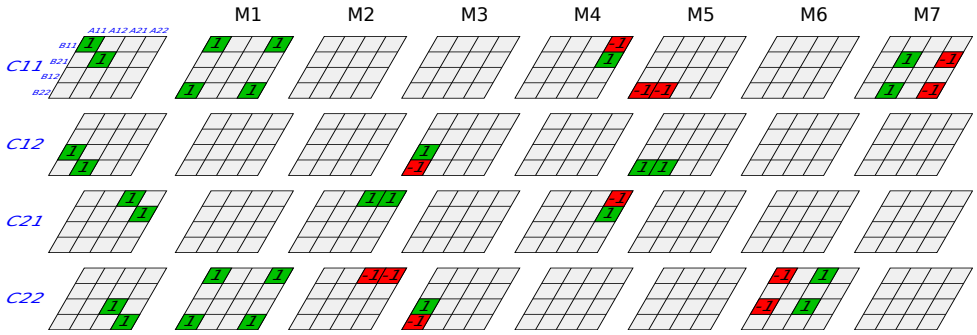
Protože platí, že $a = 8, b = 2, r = \log_2 8, n^r = n^{\log_2 8} = n^3 = \Omega(1)$, tak asymptotická složitost podle mistrovké metody je `MMM-recursive`(n) $= O(n^3)$.

2.5 Strassenův algoritmus

V roce 1969 Volker Strassen v časopise Numerische Mathematik publikoval článek [3], ve kterém jako první představil algoritmus násobení dvou matic s menší asymptotickou složitostí než algotimus podle definice, tedy $O(n^3)$.

Algoritmus je založen na myšlence, že sčítání je operace méně náročnější než operace násobení. Respektive dvě matice umíme sečíst nebo odečíst v složitosti $O(n^2)$, ale vynásobit v $O(n^3)$.

Volker Strassen tedy využil jisté symetrie [1] v násobení dvou matic A a B o velikosti dva a výslednou matici C seskládal pomocí sedmi pomocných matic. Obrázek 2.1 ukazuje, z čeho se pomocné matice skládají a jak jsou do výsledné matice seskládany. V ilustračních maticích o velikosti čtyři ukazujeme, které sčítance pomocná matice do výsledku přičítá a které odečítá.



Obrázek 2.1: Strassen (převzato z wikipedie, předělat?)

Zápis Strassenova algoritmu vypadá následnovně:

$$A \cdot B = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \tag{2.6}$$

$$M_1 = (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2}) \tag{2.7}$$

$$M_2 = (A_{2,1} + A_{2,2}) \cdot B_{1,1} \tag{2.8}$$

$$M_3 = A_{1,1} \cdot (B_{1,2} - B_{2,2}) \tag{2.9}$$

$$M_4 = A_{2,2} \cdot (B_{2,1} - B_{1,1}) \tag{2.10}$$

$$M_5 = (A_{1,1} + A_{1,2}) \cdot B_{2,2} \tag{2.11}$$

$$M_6 = (A_{2,1} - A_{1,1}) \cdot (B_{1,1} + B_{1,2}) \tag{2.12}$$

$$M_7 = (A_{1,2} - A_{2,2}) \cdot (B_{2,1} + B_{2,2}) \tag{2.13}$$

$$C = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix} \tag{2.14}$$

V pseudokódu používáme procedury `offset-add` respektive `offset-sub`. Slouží ke sčítání respektive odečítání matic bloku prvků od nějakého offsetu y a x. Paremetry obou funkcí jsou: `offset-*`$(A, B, C, ay, ax, by, bx, cy, cx, n)$.

Algorithm 5 Strassenův algoritmus

1: **procedure** MMM-STRASSEN($A, B, C, ay, ax, by, bx, cy, cx, n$)
2: **if** $n = 1$ **then**
3: $C[cy][cx] \leftarrow C[cy][cx] + A[ay][ax] \cdot B[by][bx]$;
4: *return*;
5: **end if**
6: $h \leftarrow n/2$; ▷ čtvrtina
7: $m[9] \leftarrow$`init-matrices`$(9, h)$; ▷ devět pomocných matic
8: `offset-add`$(a, a, m[8], ay, ax, ay + h, ax + h, 0, 0, h)$; ▷ M1
9: `offset-add`$(b, b, m[9], by, bx, by + h, bx + h, 0, 0, h)$;
10: `MMM-strassen`$(m[8], m[9], m[1], 0, 0, 0, 0, 0, 0, h)$;
11: `offset-add`$(a, a, m[8], ay + h, ax, ay + h, ax + h, 0, 0, h)$; ▷ M2
12: `MMM-strassen`$(m[8], b, m[2], 0, 0, bx, by, 0, 0, h)$;
13: `offset-sub`$(b, b, m[8], by, bx + h, by + h, bx + h, 0, 0, h)$; ▷ M3
14: `MMM-strassen`$(a, m[8], m[3], ay, ax, 0, 0, 0, 0, h)$;
15: `offset-sub`$(b, b, m[8], by + h, bx, by, bx, 0, 0, h)$; ▷ M4
16: `MMM-strassen`$(a, m[8], m[4], ay + h, ax + h, 0, 0, 0, 0, h)$;
17: `offset-add`$(a, a, m[8], ay, ax, ay, ax + h, 0, 0, h)$; ▷ M5
18: `MMM-strassen`$(m[8], b, m[5], 0, 0, by + h, bx + h, 0, 0, h)$;
19: `offset-sub`$(a, a, m[8], ay + h, ax, ay, ax, 0, 0, h)$; ▷ M6
20: `offset-add`$(b, b, m[9], by, bx, by, bx + h, 0, 0, h)$;
21: `MMM-strassen`$(m[8], m[9], m[6], 0, 0, 0, 0, 0, 0, h)$;
22: `offset-sub`$(a, a, m[8], ay, ax + h, ay + h, ax + h, 0, 0, h)$; ▷ M7
23: `offset-add`$(b, b, m[9], by + h, bx, by + h, bx + h, 0, 0, h)$;
24: `MMM-strassen`$(m[8], m[9], m[7], 0, 0, 0, 0, 0, 0, h)$;
25: `offset-add`$(m[1], m[4], m[8], 0, 0, 0, 0, 0, 0, h)$; ▷ c1,1
26: `offset-sub`$(m[8], m[5], m[8], 0, 0, 0, 0, 0, 0, h)$;
27: `offset-add`$(m[8], m[7], c, 0, 0, 0, 0, cy, cx, h)$;
28: `offset-add`$(m[3], m[5], c, 0, 0, 0, 0, cy, cx + h, h)$; ▷ c1,2
29: `offset-add`$(m[2], m[4], c, 0, 0, 0, 0, cy + h, cx, h)$; ▷ c2,1
30: `offset-sub`$(m[1], m[2], m[8], 0, 0, 0, 0, 0, 0, h)$; ▷ c2,2
31: `offset-add`$(m[8], m[3], m[8], 0, 0, 0, 0, 0, 0, h)$;
32: `offset-add`$(m[8], m[6], c, 0, 0, 0, 0, cy + h, cx + h, h)$;
33: **end procedure**

Výpočet asymptotické složitosti vypočteme podobně jako u `MMM-recursive`, tedy mistrovskou metodou.

TODO: mistrovska metoda

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7T(n/2) + \Theta(1) & \text{if } n > 1 \end{cases}$$

Stejně jako v předešlém algorimu, i zde demonstrujeme výpočet levého horního prvku matice z násobení dvou matic o velikosti dva. Místo parame-

trické matice použijeme desetinná čísla, abysme ukázali numerickou stabilitu Strassenova algoritmu. Pro ukázku budeme uvažovat počítač, který u čísel ukládá pouze pět cifer, znaménko a desetinnou čárku.

Pomocí algoritmu podle definice, by takový počítač vypočítal součin dvou matic následnovně:

$$\begin{pmatrix} 30.234 & 0.5678 \\ 0.9123 & 10.456 \end{pmatrix} \cdot \begin{pmatrix} 0.8912 & 0.3456 \\ 0.7891 & 9.999 \end{pmatrix} = \begin{pmatrix} 27.392 & \dots \\ \dots & \dots \end{pmatrix} \tag{2.15}$$

Správný výsledek je $30.234 \times 0.8912 + 0.5678 \times 0.7891 = 26.9445408 + 0.44805098 = 27.39259178$.

Nyní výpočet provedeme pomocí Strassenova algoritmu:

$$\begin{pmatrix} 30.234 & 0.5678 \\ 0.9123 & 10.456 \end{pmatrix} \cdot \begin{pmatrix} 0.8912 & 0.3456 \\ 0.7891 & 9.999 \end{pmatrix} \tag{2.16}$$

$$M_1 = (30.234 + 10.456) \cdot (0.8912 + 9.999) = 443.12 \tag{2.17}$$

$$\dots \tag{2.18}$$

$$M_4 = 10.456 \cdot (0.7891 - 0.8912) = -1.067 \tag{2.19}$$

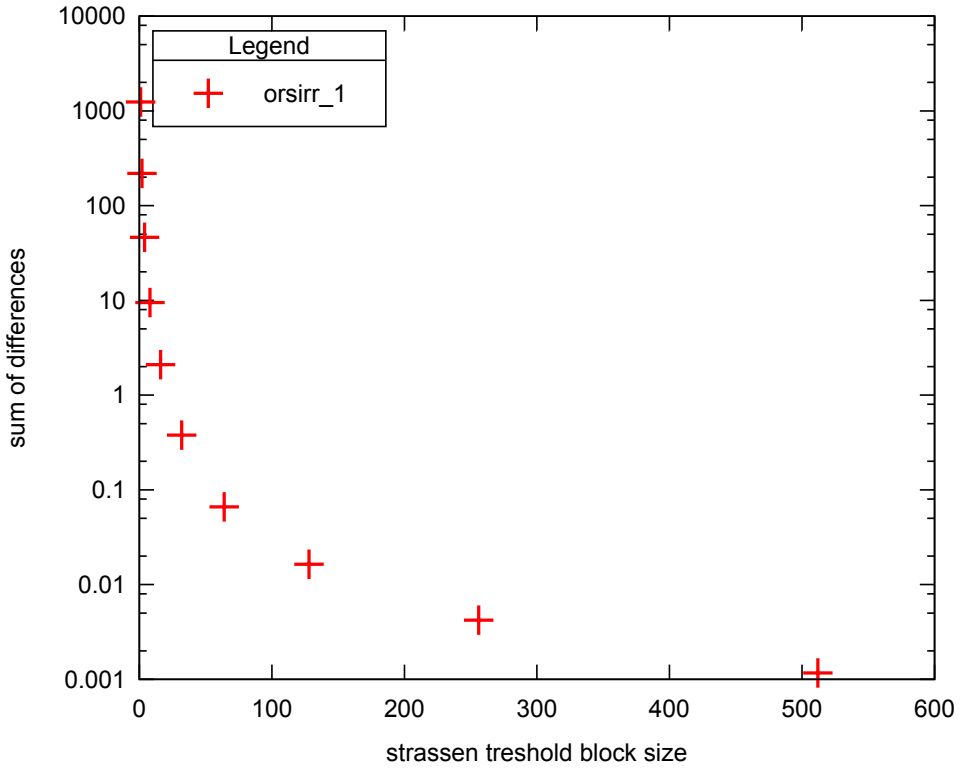
$$M_5 = (30.234 + 0.5678) \cdot 9.999 = 307.98 \tag{2.20}$$

$$\dots \tag{2.21}$$

$$M_7 = (0.5678 - 10.456) \cdot (0.7891 + 9.999) = -106.67 \tag{2.22}$$

$$\begin{pmatrix} M_1 + M_4 - M_5 + M_7 & \dots \\ \dots & \dots \end{pmatrix} = \begin{pmatrix} 27.403 & \dots \\ \dots & \dots \end{pmatrix} \tag{2.23}$$

Pro reálnou představu stability Strassenova algoritmu jsme provedli experiment, ve kterém jsme vynásobili dvě stejné matice (matice orsirr_1, oříznuta na 1024x1024) algoritmem podle definice a Strassenovým algoritmem s různými prahy a sečetli všechny rozdíly mezi výsledky. Násobení probíhalo ve dvojté desetinné přesenosti, tedy v datovém typu `double` jazyka C99. Z grafu je vidět exponencionální růst chyby.



Obrázek 2.2: Strassen stability

Strassenův algorimus lze ještě vylepšit. Algoritmům na stejném principu se říká Strassen-like. Pro sedm operací násobení je monžné snížit počet sčítání a odečítání. Pro jednoduchost zde ovšem uvádíme originalní algoritmus.

2.6 Rychlé algoritmy

Po tom, co Strassen ukázal, že existují rychlejší algoritmy než $O(n^3)$, ještě rychlejší algoritmy než ten jeho na sebe nenechaly dlouho čekat. Složitost násobení matic pro jednoduchost označíme jako $O(n^\omega)$.

Nejpomalejší algorotmus s $\omega = 3$ je podle definice. Strassenův algoritmus se sedmi násobeními má $\omega \approx 2.807354$. Jeden z nejrychlejších algoritmů je algoritmus Virginie Williamsové [5][4], pro který je $\omega < 2.3727$.

Hranicí nejlepší možné složitosti může být $\omega = 2$, protože každý prvek z matice musíme nějak započítat. Existují z velké části podložené doměnky na základě teroie grup [2], že $\omega = 2$ skutečně platí, ale přímý důkaz ještě neexistuje.

2.7 Algoritmus podle definice upravený pro řídké matice

Pokud násobíme řídké matice A a B o velikosti N, můžeme vynechat násobení takových dvou prvků, z nichž je alespoň jeden nulový. Označme $nnzr_{M,i}$ jako počet nenulových prvků v i-tém řádku matice M a $nnzc_{M,i}$ jako počet nenulových prvků v i-tém sloupci matice M. Protože násobíme každý řádek s každým sloupcem, bude celkový počet operací násobení dán vzorcem:

$$\sum_{i=1}^{N} nnzr_{A,i} nnzc_{B,i}$$

Složitost tohoto algoritmu pro násobení dvou matic A a B o velikosti n tedy můžeme vyjádřit jako $O(mn)$, kde $m = max(nnz(A), nnz(B)$. Pro husté matice platí, že $m = n^2$. Nutno podotknout, že $O(mn)$ je nejhorší případ, kdy všechny nenulové prvky matice A budou násobit se všemy nenulovými prvky matice B.

2.8 Rychlé násobení řídkých matic

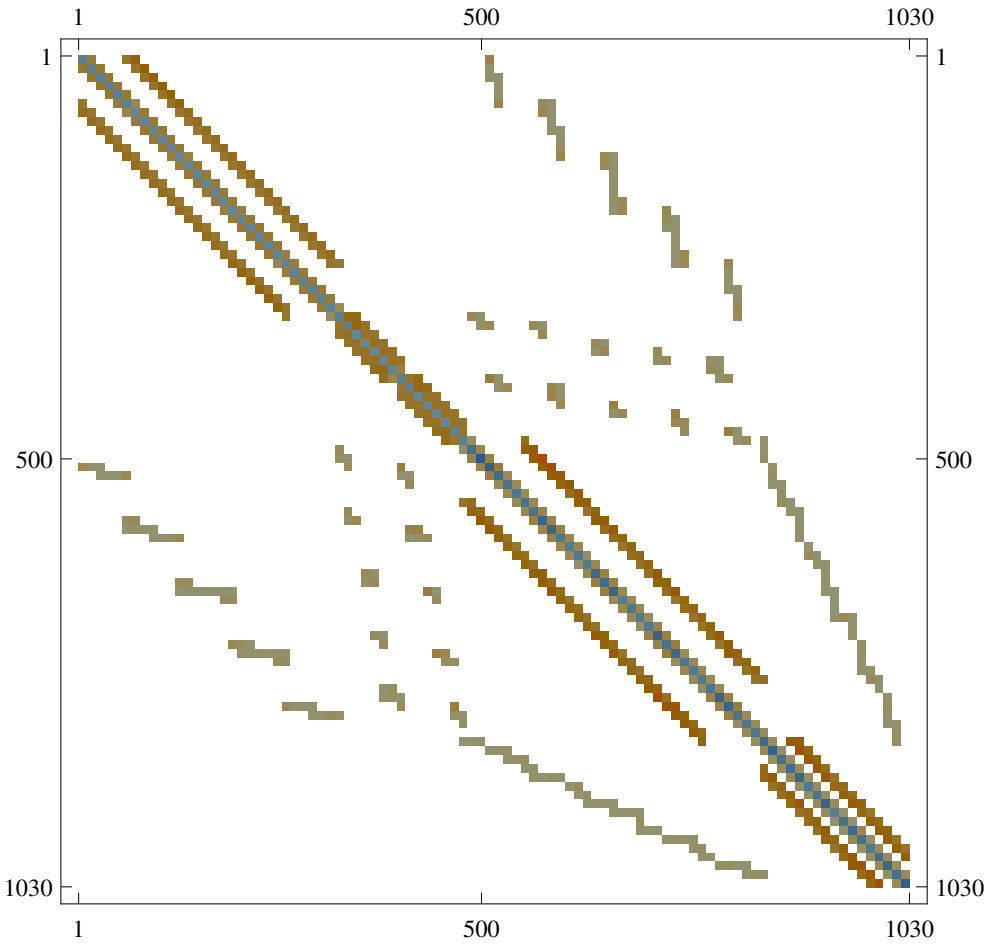
TODO: ukazat co z matice se kolikrat zapocita

Od Od $m < n^1.37$ se algoritmus Virginie Williamsové $O(n^2.3727)$ stává stejně asyptoticky rychlý jako algoritmus podle definice upravený pro řídké matice $O(mn)$.

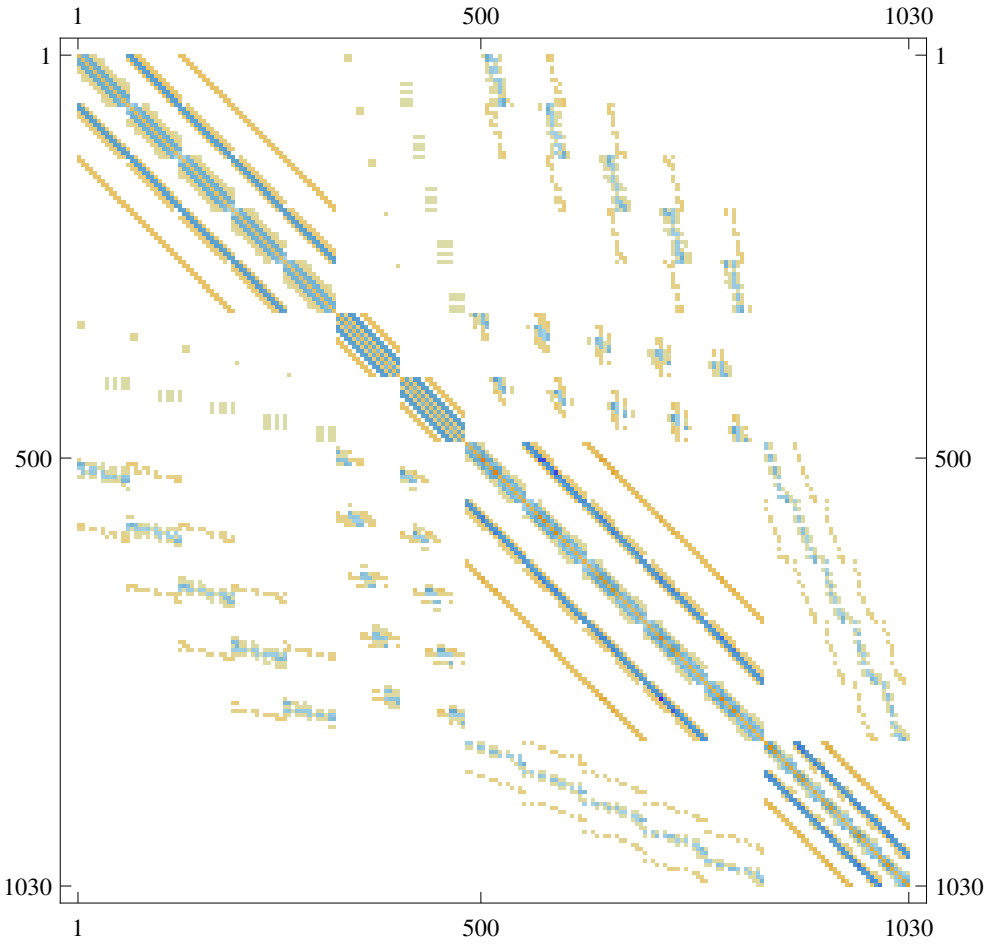
Strassenův algoritmus $\omega = 2.8$ od $m < n^1.8$ a algoritmus Virginie Williamsové $O(n^2.3727)$ od $m < n^1.37$ jsou stejně asymptoticky rychlé jako algoritmus podle definice upravený pro řídké matice $O(mn)$. Na příklad, pro $n = 100$ je v husté matici 10000 prvků. Pokud by z nich by

tady [6] bka

Existuje mnoho dalších algoritmů násobení matic. Zmíníme algoritmy vyhledávající vzory v řídkých maticích, například diagonály, co násobí jednotlivé vzory mezi sebou. **?muzu rict, ze to delat nebudu?** V této práci se zabýváme pouze algoritmy a formáty uložení pro (odkaz do sekce s typama ridkych matic).



Obrázek 2.3: orsirr_1



Obrázek 2.4: orsirr_1 po vynásobení sama se sebou

Kapitola 3

Formáty uložení řídkých matic

Formáty uložení řídkých matic obecně ukládají jednotlivé elementy zvlášť a tedy nemusí ukládat ty nulové. To ale přináší řadu nevýhod. Za prvé se musí ukládat informace o souřadicních jednotliých prvcích. Za druhé, ztrácíme možnost přístupu k prvku na libovolných šouřadnicích v čase $\Theta(1)$, protože prvky nemáme přímo indexované podle jejich umístění v řádku a sloupci.

upravitelne vs neupravitelne

3.1 COO - Coordinate list

3.2 CSR - Compressed sparse row

3.3 BSR - Block Sparse Row

3.4 Quadtree

3.5 ?

TODO: tady jsem chtel spocictat kdy se vyplati mit ridkou matici, ale lepsi bude tabulka

Pokud například uložíme matici o rozměrech 100x100 v dvojté přestnosti, bude zabírat `M x N x sizeof(double) = 100 x 100 x 8 = 80000B = 80kB`. Pokud zvolíme řídký formát matice, kde ke každému elementu uložíme i jeho x a y souřadnici, tak do 80kB uložíme `80000 / (sizeof(int)+sizeof(int)+sizeof(double)) = 80000/16= 5000` elementů. Pokud matice obsahuje více jak 50 % nulových elementů, vyplatí se nám ji uložit do řídkého formátu.

Kapitola 4

Modifikace formátu quadtree

je to samostatnej bod v zadani tak by to mohla byt cela chapter

TODO: popsat nevyhody quadtree a obrazkama ukazat jak to udelat lip neco jako quadtree loop unrolling

Kapitola 5

Analýza a návrh

? bud to nechapu nebo tuhle chapter smazu

XXX: napsat o tom, jak jsem pouzil spoustu knihoven. napr libc nebo math. napsat o tom, jak jsem chtel scipy sparse, ale chtel jsem to co nejjednodussi

Kapitola 6

Realizace

6.1 MatrixMarket

TODO: posat format matrixmarket, ve kterem budu vsechno delat

6.2 Optimalizace

TODO: rict ze budeme verit -O3, ale popsat transformaci cyklu a loop-unrolling

6.3 ? Design implementace

TODO: popsat OOP v C, moje testy

6.4 Měření

TODO: popsat jak to budu měřit, tedy cas z omp a cachegrind/callgrind (mozna solaris)

Závěr

Literatura

[1] Gates, A. Q.; Kreinovich, V.: Strassen's Algorithm Made (Somewhat) More Natural: A Pedagogical Remark. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, ročník 73, 2001: s. 142–145.

[2] Stothers, A. J.: *On the Complexity of Matrix Multiplication*. diploma thesis, University of Edinburgh, 2010.

[3] Strassen, V.: Gaussian Elimination is not Optimal. *Numerische Mathematik*, ročník 13, č. 4, Prosinec 1967: s. 354–355.

[4] Williams, V. V.: Breaking the Coppersmith-Winograd barrier. 2011.

[5] Williams, V. V.: Multiplying matrices faster than coppersmith-winograd. In *STOC*, editace H. J. Karloff; T. Pitassi, ACM, 2012, ISBN 978-1-4503-1245-5, s. 887–898.

[6] Yuster, R.; Zwick, U.: Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, ročník 1, č. 1, 2005: s. 2–13.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

Seznam obrázků

Seznam algoritmů

*

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS