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

19. dubna 2014

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

# 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 19. 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

**Úvod** **1**

**1 Úvod do problematiky** **3**
1.1 Použití násobení matic . . . 3
1.2 Matice . . . 3
1.3 Vektor . . . 3
1.4 Násobení matic . . . 3
1.5 Složitosti . . . 4
1.6 Řídké matice . . . 4
1.7 Numerická stabilita . . . 4
1.8 Optimalizace kódu . . . 4

**2 Algoritmy násobení matic** **7**
2.1 Podle definice . . . 7
2.2 Násobení transponovanou maticí . . . 7
2.3 Násobení po řádcích . . . 8
2.4 Rekurzivní násobení . . . 9
2.5 Strassenův algoritmus . . . 10
2.6 Rychlé algoritmy . . . 14
2.7 Algoritmus podle definice upravený pro řídké matice . . . 14
2.8 Rychlé násobení řídkých matic . . . 18
2.9 Další algoritmy pro řídké matice . . . 18

**3 Formáty uložení řídkých matic** **19**
3.1 COO - Coordinate list . . . 20
3.2 CSR - Compressed sparse row . . . 21
3.3 BSR - Block Sparse Row . . . 23
3.4 Quadtree . . . 23
3.5 ? . . . 23

**4 Modifikace formátu quadtree** **25**

**5 Analýza a návrh** **27**
5.1 Práce s řídkými maticemi v moderním software . . . . . . . . . 27

**6 Realizace** **29**
6.1 MatrixMarket . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
6.2 Optimalizace . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
6.3 ? Design implementace . . . . . . . . . . . . . . . . . . . . . . 31
6.4 Měření . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

**Závěr** **33**

**Literatura** **35**

**A Seznam použitých zkratek** **37**

**Seznam obrázků** **37**

**B Obsah přiloženého CD** **41**

# Seznam obrázků

2.1 Strassen (TODO: převzato z wikipedie: předělat?) . . . . . . . . . . 11
2.2 Ukázka numerické stability Strassenova algoritmu . . . . . . . . . . . 14
2.3 Matice orsirr_1 před vynásobením . . . . . . . . . . . . . . . . . . . . 16
2.4 Matice orsirr_1 po vynásobením sama se sebou . . . . . . . . . . . . 17

3.1 Matice uložená ve formátu CSR . . . . . . . . . . . . . . . . . . . . . . 22

6.1 Matice vygenerovaná generátorem . . . . . . . . . . . . . . . . . . . . 31

# Seznam tabulek

3.1 Matice uložená ve formátu COO . . . . . . . . . . . . . . . . . . . . . 20

# 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.

## 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 libovolný 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, mistrovskou metodu

## 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 obecne (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.

TODO: priklady, zdroje

### 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

```
procedure MMM-transpose(A, B, C)                    ▷ A,B,C jsou matice
    B ← transpose(B)
    for rowA ← 0 to A.height do                     ▷ řádky
        for rowB ← 0 to B.height do                 ▷ sloupce
            sum ← 0;
            for i ← 0 to A.height do
                sum ← sum + A[rowA][i] ∗ B[i][rowB];
            end for
            C[rowA][rowB] ← sum;
        end for
    end for
end procedure
```

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

## 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

```
procedure MMM-by-rows(A, B, C)                      ▷ A,B,C jsou matice
    for r ← 0 to A.height do                        ▷ řádky matice A i B
        for cA ← 0 to A.width do                    ▷ sloupce matice A
            for cB ← 0 to B.width do                ▷ sloupce matice B
                C[r][cA] ← C[r][cA] + A[r][cA] ∗ B[r][cB];
            end for
        end for
    end for
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

**procedure** MMM-RECURSIVE$(A, B, C, ay, ax, by, bx, cy, cx, n)$
    **if** $n = 1$ **then**
        $C[cy][cx] \leftarrow C[cy][cx] + A[ay][ax] \cdot B[by][bx]$;
        *return*;
    **end if**
    **for all** $r \in \{0, n/2\}$ **do**
        **for all** $c \in \{0, n/2\}$ **do**
            **for all** $i \in \{0, n/2\}$ **do**
                `MMM-recursive`$(A, B, C, ay + i, ax + r, by + c, bx + i, cy + c, cx + r, n/2)$;
            **end for**
        **end for**
    **end for**
**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.

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

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

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

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

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.

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

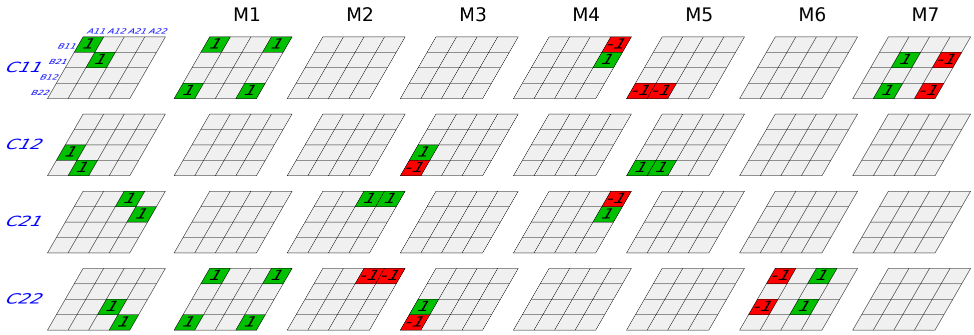
Protože platí, že $a = 8, b = 2, r = \log_2 8, n^r = n^{\log_2 8} = n^3 = \Omega(1)$, tak asymptotická složitost podle mistrovké metody je `MMM-recursive`(n) $= O(n^3)$.

## 2.5 Strassenův algoritmus

V roce 1969 Volker Strassen v časopise Numerische Mathematik publikoval článek [5], 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 náročnejší 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 [3] 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 (TODO: 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 bloku prvků o velikosti n v maticích 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

```
procedure MMM-strassen(A, B, C, ay, ax, by, bx, cy, cx, n)
    if n = 1 then
        C[cy][cx] ← C[cy][cx] + A[ay][ax] · B[by][bx];
        return;
    end if
    h ← n/2;                                                          ▷ čtvrtina
    m[9] ←init-matrices(9, h);                                        ▷ devět pomocných matic
    offset-add(a, a, m[8], ay, ax, ay + h, ax + h, 0, 0, h);          ▷ M1
    offset-add(b, b, m[9], by, bx, by + h, bx + h, 0, 0, h);
    MMM-strassen(m[8], m[9], m[1], 0, 0, 0, 0, 0, 0, h);
    offset-add(a, a, m[8], ay + h, ax, ay + h, ax + h, 0, 0, h);      ▷ M2
    MMM-strassen(m[8], b, m[2], 0, 0, bx, by, 0, 0, h);
    offset-sub(b, b, m[8], by, bx + h, by + h, bx + h, 0, 0, h);      ▷ M3
    MMM-strassen(a, m[8], m[3], ay, ax, 0, 0, 0, 0, h);
    offset-sub(b, b, m[8], by + h, bx, by, bx, 0, 0, h);              ▷ M4
    MMM-strassen(a, m[8], m[4], ay + h, ax + h, 0, 0, 0, 0, h);
    offset-add(a, a, m[8], ay, ax, ay, ax + h, 0, 0, h);              ▷ M5
    MMM-strassen(m[8], b, m[5], 0, 0, by + h, bx + h, 0, 0, h);
    offset-sub(a, a, m[8], ay + h, ax, ay, ax, 0, 0, h);              ▷ M6
    offset-add(b, b, m[9], by, bx, by, bx + h, 0, 0, h);
    MMM-strassen(m[8], m[9], m[6], 0, 0, 0, 0, 0, 0, h);
    offset-sub(a, a, m[8], ay, ax + h, ay + h, ax + h, 0, 0, h);      ▷ M7
    offset-add(b, b, m[9], by + h, bx, by + h, bx + h, 0, 0, h);
    MMM-strassen(m[8], m[9], m[7], 0, 0, 0, 0, 0, 0, h);
    offset-add(m[1], m[4], m[8], 0, 0, 0, 0, 0, 0, h);                ▷ c1,1
    offset-sub(m[8], m[5], m[8], 0, 0, 0, 0, 0, 0, h);
    offset-add(m[8], m[7], c, 0, 0, 0, 0, cy, cx, h);
    offset-add(m[3], m[5], c, 0, 0, 0, 0, cy, cx + h, h);             ▷ c1,2
    offset-add(m[2], m[4], c, 0, 0, 0, 0, cy + h, cx, h);             ▷ c2,1
    offset-sub(m[1], m[2], m[8], 0, 0, 0, 0, 0, 0, h);                ▷ c2,2
    offset-add(m[8], m[3], m[8], 0, 0, 0, 0, 0, 0, h);
    offset-add(m[8], m[6], c, 0, 0, 0, 0, cy + h, cx + h, h);
end procedure
```

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

V každém kroku rekurze počítáme $\Theta(n^2)$ operací na vytvoření pomocných matic.

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

Protože platí, že $a = 7, b = 2, r = \log_2 7, n^r = n^{\log_2 7} = \Omega(n^2)$, tak asymptotická složitost podle mistrovké metody je `MMM-strassen`(n) $= O(n^{\log_2 7}) \approx O(n^{2.8})$.

Stejně jako v předešlém algorimu, i zde demonstrujeme výpočet levého horního prvku matice z násobení dvou matic o velikosti dva. Místo parametrické matice použijeme konkrétní desetinná čísla, abychom 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:

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

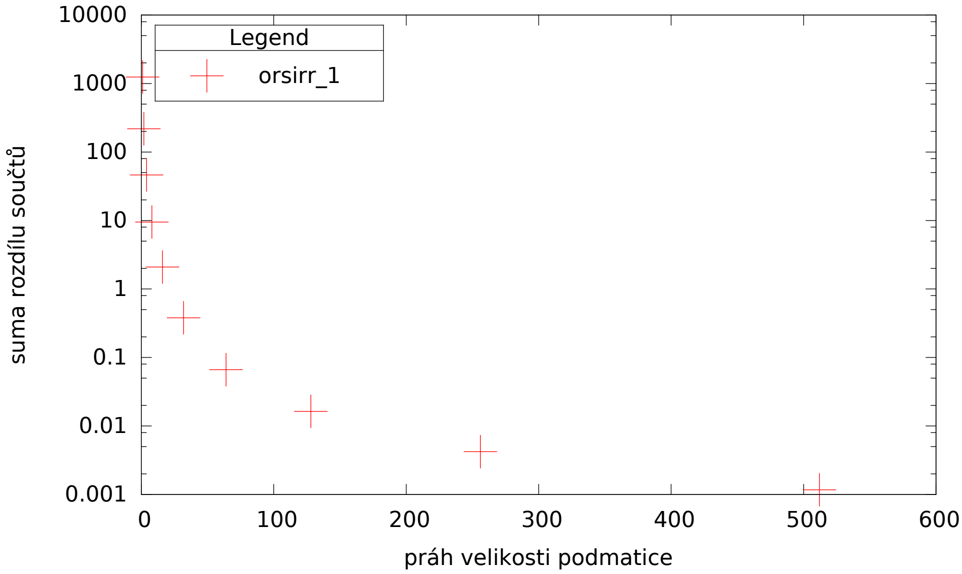
Správný výsledek je $30.234 \times 0.8912 + 0.5678 \times 0.7891 = 26.9445408 + 0.44805098 = 27.39259178$.

Nyní výpočet provedeme pomocí Strassenova algoritmu:

$$\begin{aligned}
\begin{pmatrix} 30.234 & 0.5678 \\ 0.9123 & 10.456 \end{pmatrix} \cdot \begin{pmatrix} 0.8912 & 0.3456 \\ 0.7891 & 9.999 \end{pmatrix} \qquad (2.16)\\
M_1 = (30.234 + 10.456) \cdot (0.8912 + 9.999) = 443.12 \qquad (2.17)\\
\dots \qquad (2.18)\\
M_4 = 10.456 \cdot (0.7891 - 0.8912) = -1.067 \qquad (2.19)\\
M_5 = (30.234 + 0.5678) \cdot 9.999 = 307.98 \qquad (2.20)\\
\dots \qquad (2.21)\\
M_7 = (0.5678 - 10.456) \cdot (0.7891 + 9.999) = -106.67 \qquad (2.22)\\
\begin{pmatrix} M_1 + M_4 - M_5 + M_7 & \dots \\ \dots & \dots \end{pmatrix} = \begin{pmatrix} 27.403 & \dots \\ \dots & \dots \end{pmatrix} \qquad (2.23)
\end{aligned}$$

Jak můžeme vidět, zatímco u algoritmu podle definice jsme pouze ztratili desetinnou přesnost, výsledek Strassenova algoritmu se lišil už v prvním desetinném čísle.

Pro reálnou představu stability Strassenova algoritmu jsme provedli experiment, ve kterém jsme vynásobili dvě stejné matice (matice orsirr_1, oříznuta na velikost 1024) 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 2.2 je vidět exponencionální závislost mezi velikostí prahu a celkovou chybou.




Obrázek 2.2: Ukázka numerické stability Strassenova algoritmu

Strassenův algorimus lze ještě vylepšit. Algoritmům na stejném principu se říká Strassen-like [1]. 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 algoritmus s $\omega = 3$ je podle definice. Strassenův algoritmus se sedmi násobeními má $\omega \approx 2.807354$. Jeden z nejrychlejších algoritmů je algoritmus Virginie Williamsové [8][7], 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 domněnky na základě teorie grup [4], že $\omega = 2$ skutečně platí, ale přímý důkaz ještě neexistuje.

## 2.7 Algoritmus podle definice upravený pro řídké matice

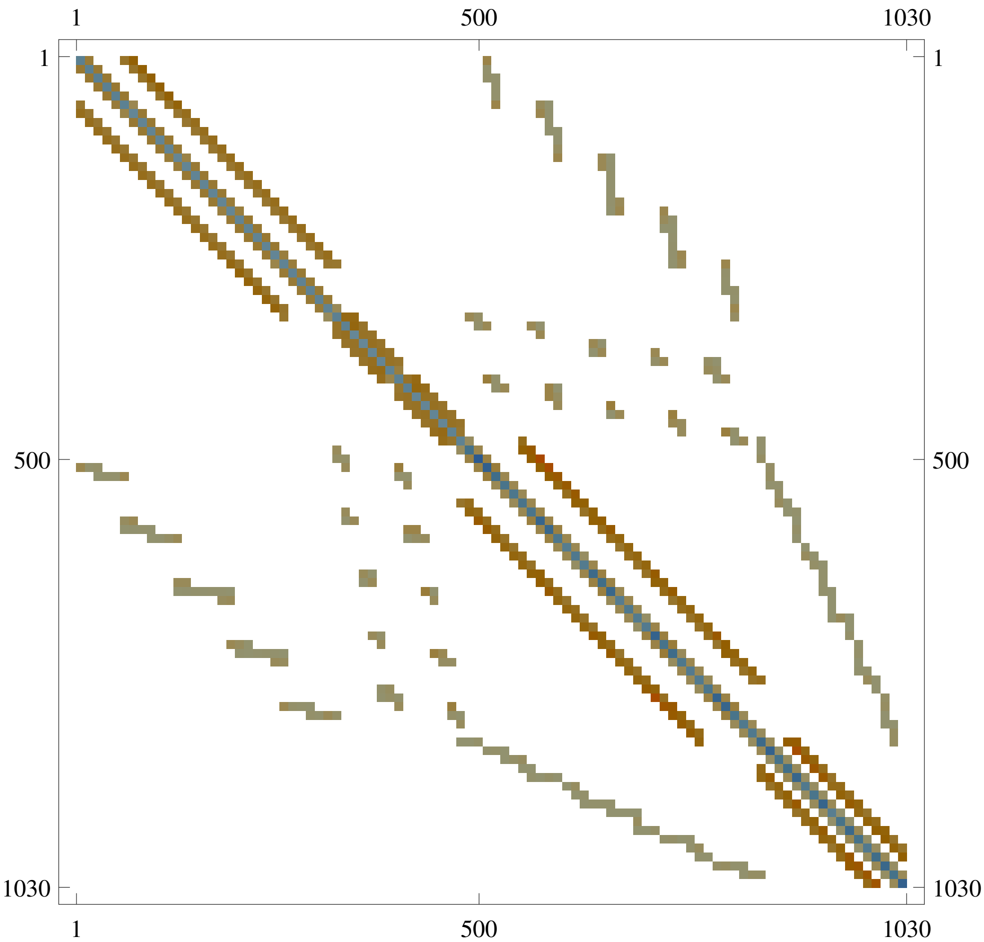
Pokud násobíme řídké matice $A$ a $B$ o velikosti $N$, můžeme vynechat násobení takových dvou prvků, z nichž je alespoň jeden nulový. Označme $nnzr_{M,i}$

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:

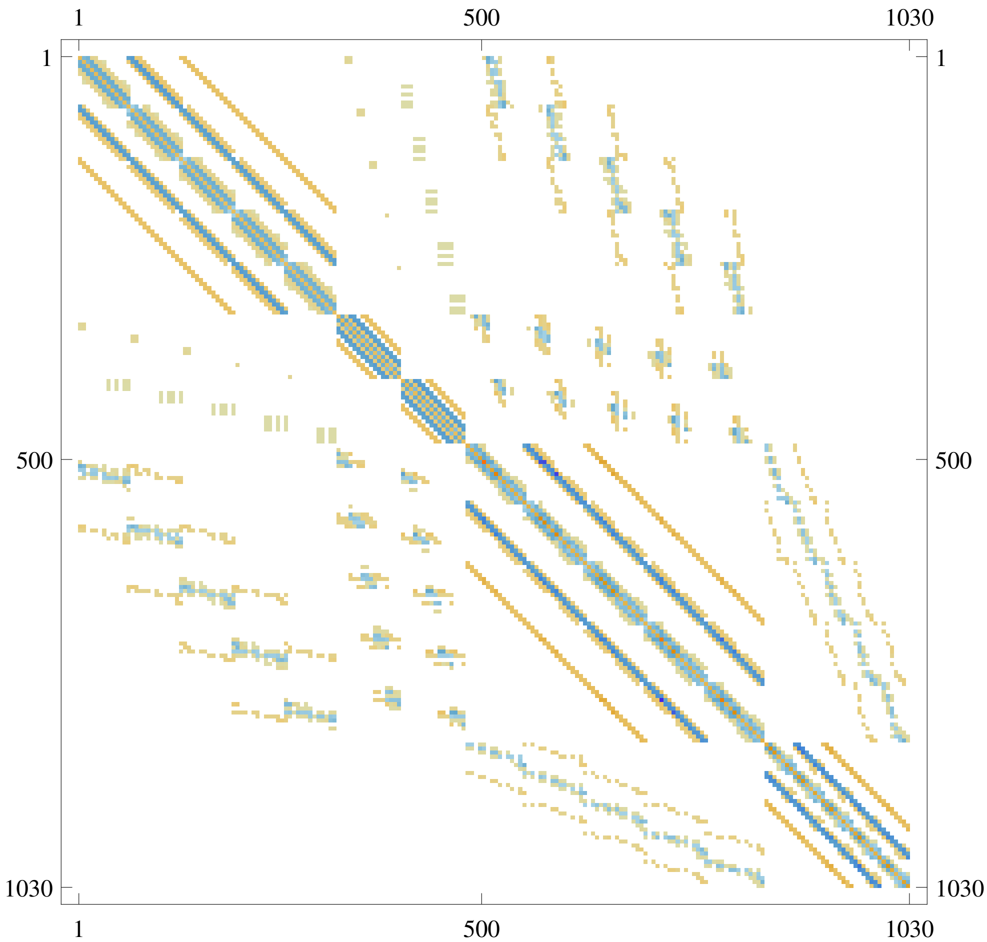
$$\sum_{i=1}^{N} nnzr_{A,i} nnzc_{B,i}$$

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 bez jediného nulového prvku samozřejmě platí, že $m = n^2$. Nutno podotknout, že $O(mn)$ je nejhorší případ, kdy se všechny nenulové prvky matice $A$ budou násobit se všemy nenulovými prvky matice $B$.

Pro reálnou představu demonstrujeme násobení dvou stejných matic. Opět se jedná o dvě matice orsirr_1. Matice orsirr_1 má velikost $n = 1030$ a $m = 6858$ nnz. Rozmístění nnz před respektive po vynásobení ukazují obrázky 2.3 respektive 6.1.




Obrázek 2.3: Matice orsirr_1 před vynásobením




Obrázek 2.4: Matice orsirr_1 po vynásobením sama se sebou

Kdyby nastal nejhorší možný případ, počet operací násobení při použití algoritmu podle definice pro řídké matice by byl $1030 \times 6858 = 7.063740 \times 10^6$. Pro tento případ ovšem stačí $4.6976 \times 10^4$ operací násobení. Oproti nejhoršímu možnému případu nastalo $7.063740 \times 10^6 - 4.6976 \times 10^4 = 7.016764 \times 10^6$ situací, kdy jeden ze dvou prkvů byl nulový a operace násobení nemusela být provedena. Při násobení algoritmem podle definice pro husté matice by v $1.092680024 \times 10^9$ případech operace násobení nemusela být provedena, protože alespoň jeden ze dvou prvků by byl nula. Po vynásobení matice orsirr_1 sama se sebou, stoupl její počet nnz z 6858 na 23532. V tomto případě to bylo především z důvodu, že všechny prvky z diagonály matice jsou nenulové, každý prvek se tedy započítá.

## 2.8 Rychlé násobení řídkých matic

Strassenův algoritmus $\omega = 2.8$ od $nnz < n^{1.8}$ a algoritmus Virginie Williamsové $\omega = 2.3$ od $nnz < n^{1.37}$ jsou asymptoticky stejně rychlé jako algoritmus podle definice upravený pro řídké matice $O(mn)$. Například pro $n = 1000$ je tato hranice pro Strassenův algoritmus 251189 (40 %) nnz a pro algoritmus Virginie Williamsové 12882 (78 %) nnz z celkových možných 1000000 prvků.

Raphael Yuster a Uri Zwick ukázali algoritmus [9] s asymptotickou složitostí $O(m^{0.7}n^{1.2} + n^{2+o(1)})$, který rozdělí permutace řádků a sloupců na řídké a husté. Řídké permutace násobí algoritmem podle definice v úpravě pro řídké matice $O(mn)$. Husté permutace vynásobí v té době nejznámějším nejrychlejším algoritmem a to od Dona Coppersmitha a Shmuela Winograda s asymptotickou složitostí $\omega = 2.3$ [2].

## 2.9 Další algoritmy pro řídké matice

Algoritmů násobení řídkých matic je mnoho. Často se odvíjejí od typu řídkých matic a formátu v jakém jsou uloženy. Příkladem může být formát, ve kterém se ukládají diagonály [6].

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 jednotliýchh 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.

Protože řídké matice můžeme rozdělit do mnoha kategorií a provádět nad nimi mnoho operací, existuje hodně formátů, jak řídkou matici efektivně uložit a pracovat s ní.

### 3.0.1 Modifikace řídké matice

Formáty uložení řídkých matic můžeme také rozdělit podle toho, zda-li je možné do nich přidávat nebo odebírat prvky.

Při násobení matic $C = A \cdot B$ se matice $A$ ani $B$ nemění. V této práci budeme předpokládat, že matice $C$ bude hustá a formáty umožujícími přidávání nebo odebírání prvků nebudou součástí práce. Stejně tak při násobení matice $A$ vektorem $B$ je výsledek $C$ vektor.

### 3.0.2 Uspořádanost řídké matice

Dalším kritériem pro rozdělení formátů je uspořádanost nenulových prvků v řídké matici. Pro uspořádané prvky bude efektivnější takový formát, který využije určitý vzor. V řídkých maticích takovým vzorem může být například diagonála, nebo blok prvků. Efektivně lze za vzor považovat i prvky v řádku, nebo ve sloupci.

Uspořádáním může být také symetrie matice, kdy nám stačí uložit pouze polovinu matice. Zpravidla řídké matice bývají symetrické podle hlavní diagonály.

## 3.1 COO - Coordinate list

Formát COO, česky seznam souřadnic, je základní formát řídkých matic. Ke každému nenulovému prvku ukládá jeho souřadnice `y` a `x`. Implementovat tento formát můžeme například jako tři pole, jedno s hodnotami prvků, druhé s `y` souřadnicemi a třetí s `x` souřadnicemi.

Pro ukázku v tomto formátu uložíme matici o velikosti $n = 8$ s $nnz = 5$ nenulovými prvky.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{4} & \mathbf{5} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.1}$$

| | | | | | |
|---|---|---|---|---|---|
| values[5] | 1 | 2 | 3 | 4 | 5 |
| y-coords[5] | 1 | 1 | 4 | 7 | 7 |
| x-coords[5] | 0 | 1 | 1 | 5 | 6 |

Tabulka 3.1: Matice uložená ve formátu COO

Jak můžeme vidět, délka polí je závislá pouze na $nnz$. Pro velké matice s malým počtem neuspořádaných nenulových prvků je tento formát velmi efektivní. Pokud by bylo prvků velké množství, informace o uložení `y` nebo `x` souřadnic by byla často redundatní.

Formát COO je velmi jednoduchý a přímočarý. Při procházení jeho prvků nám stačí jedna iterace přes tři stejně dlouhé pole. Tím je tedy například algoritmus násobení řídké matice ve formátu COO s vektorem velmi jednoduchý:

**Algorithm 6** Násobení matice COO s vektorem

**procedure** COO-MVM($COO, V, C$)
  **for** $i \leftarrow 0$ `to` $COO.nnz$ **do**
    $V.v[COO.r[i]] \leftarrow V.v[COO.r[i]] + COO.v[i] * V.v[COO.c[i]]$;
  **end for**
**end procedure**

Při násobení dvou matic ale narazíme na problém. Při této operaci se každý prvek násobí dvakrát. Potřebujeme způsob, jak se v matici vrátit zpátky na určité místo. Takový naivní algoritmus pro násobení dvou COO matic by byl složitý. Museli bychom si pamatovat začátky řádků a neustále kontrolovat, jestli jsme nepřesáhli další řádek. Lepším řešením je dopředu si předpočítat, kde který řádek začíná a končí. Předpočítáme si tedy pole, nazvané *row_pointers*, o délce $M.height+1$, obsahující indexy začátků a konců řádek.

Při procháchezní matice se tak v poli s prvky mezi indexy $row\_pointers[i]$ a $row\_pointers[i+1]$ nachází prkvy na řádku $i$. Proto je pole právě o jedna delší než výska matice, abychom mohli určit konec posledního řádku.

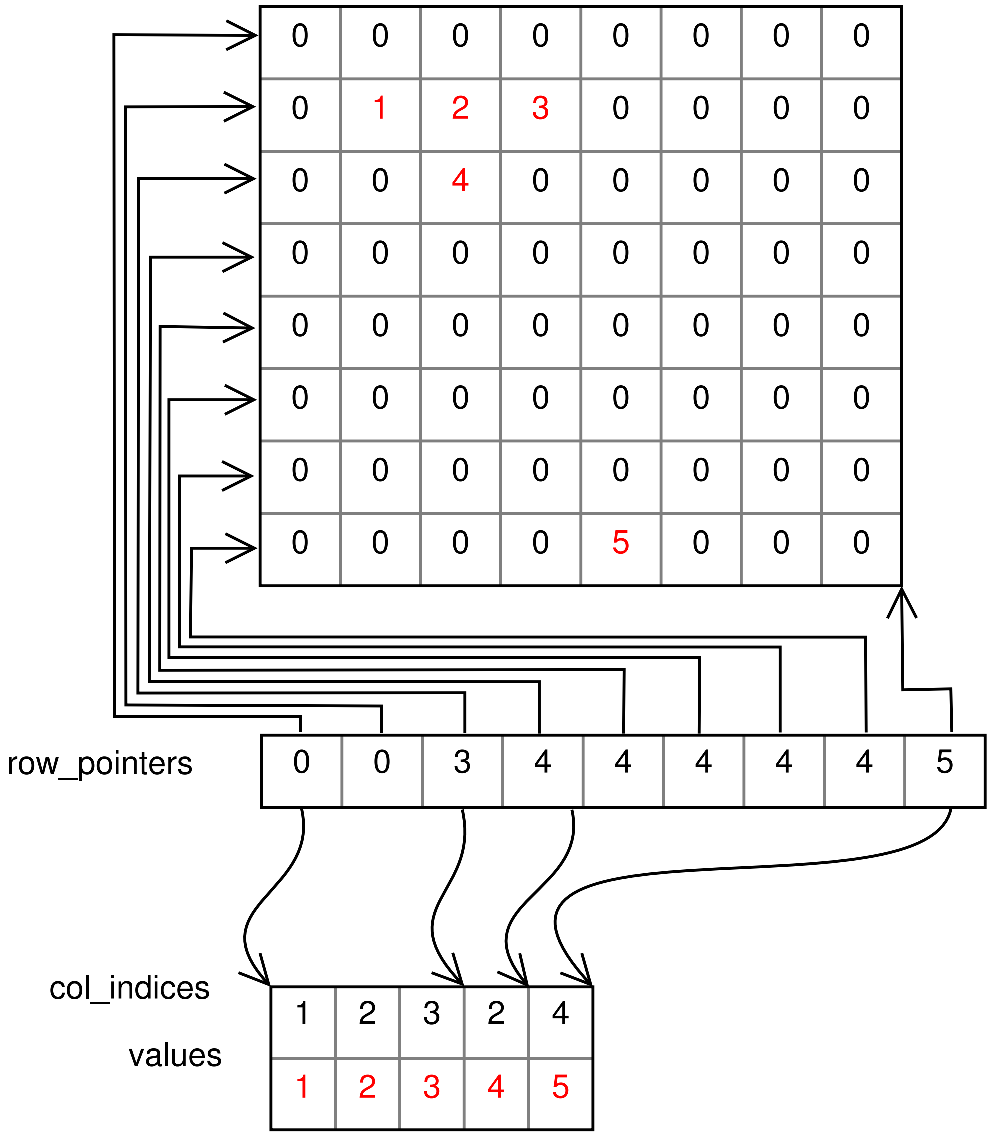
**Algorithm 7** Násobení dvou CSR matic

```
procedure CSR-MMM(A, B, C)
    arp ← InitArray(A.nnz + 1);
    brp ← InitArray(B.nnz + 1);
    for i ← 0 to A.nnz do          ▷ předpočítání prvků v řádcích matice A
        arp[A.r[i] + 1] = arp[A.r[i] + 1] + 1;
    end for
    for i ← 0 to A.height do
        arp[i + 1] = arp[i + 1] + arp[i];
    end for
    for i ← 0 to B.nnz do          ▷ předpočítání prvků v řádcích matice B
        brp[B.r[i] + 1] = brp[B.r[i] + 1] + 1;
    end for
    for i ← 0 to A.height do
        brp[i + 1] = brp[i + 1] + brp[i];
    end for
    for i ← 0 to A.height do                                      ▷ násobení
        for ac ← arp[r] to arp[r + 1] do
            for bc ← brp[A.c[ac]] to brp[A.c[ac] + 1] do
                C.v[r][A.c[bc]] ← C.v[r][A.c[bc]] + A.v[ac] ∗ B.v[bc];
            end for
        end for
    end for
end procedure
```

V části s násobením již pole s `y` souřadnicemi nepotřebujeme. Lepší formát uložení řídkých matic pro násobení by byl takový, který namíto pole s `y` souřadnicemi obsahuje předpočítané začátky a konce řádků. Takovým formátem je CSR.

## 3.2 CSR - Compressed sparse row

Problém efektivnosti formátu COO pro větší množství prvků řeší formát CSR, česky komprimované řídké řádky. Formát CSR obsahuje pole *row_pointers*, které ukládá informace o tom, kolik se v daném řádku nachází prvků. K poli s hodnotami je další pole *col_indicies*, přiřazující ke každému prvku informaci o sloupci.




Obrázek 3.1: Matice uložená ve formátu CSR

Jak je vidět z ilustrace 3.1, řádek s více prvky je uložen efektivně. Díky prázdným řádkům se nezdá pole *row_pointers* rozumně využité.

Při násobení matice CSR s vektorem potřebujeme o jeden for cyklus více než v případě násobení matice COO s vektorem. Důvodem je ztráta informace o řádku prvku.

**Algorithm 8** Násobení matice CSR s vektorem

**procedure** CSR-MVM($CSR, V, C$)
    **for** $i \leftarrow 0$ `to` $CSR.h$ **do**
        **for** $ci \leftarrow CSR.rp[i]$ `to` $CSR.rp[i+1]$ **do**
            $C.v[r] \leftarrow C.v[r] + CSR.v[ci] * V.v[A.ci[ci]]$;
        **end for**
    **end for**
**end procedure**

Násobení dvou CSR matic je stejné jak v případě násobení dvou COO matic popsaném v sekci COO - Coordinate list.

**Algorithm 9** Násobení dvou COO matic

**procedure** CSR-MMM($A, B, C$)
    **for** $i \leftarrow 0$ `to` $A.height$ **do** ▷ násobení
        **for** $ac \leftarrow A.rp[r]$ `to` $A.rp[r+1]$ **do**
            **for** $bc \leftarrow B.rp[A.ci[ac]]$ `to` $B.rp[A.ci[ac]+1]$ **do**
                $C.v[r][B.ci[bc]] \leftarrow C.v[r][B.ci[bc]] + A.v[ac] * B.v[bc]$;
            **end for**
        **end for**
    **end for**
**end procedure**

## 3.3 BSR - Block Sparse Row

`https://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/mklman/GUID-9FCEB1C4-670D-4738-81D2-F378013412B0.htm`

## 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

## 5.1 Práce s řídkými maticemi v moderním software

### 5.1.1 SciPy.sparse

SciPy je knihovna skriptovacího jazyka Python.

ZDROJ: http://docs.scipy.org/doc/scipy/reference/sparse.html

### 5.1.2 Wolfram Mathematica

Wolfram Mathematica je TODO

Pomocí Wolfram Mathematicy vizualizujeme řídké matice z formátu `.mtx`. MatrixMarket nabízí matice v souborech `.mtx.gz` Wolfram Mathematica umí pracovat i s těmito komprimovanými soubory. Příkaz pro importování řídké matice a její vizualizaci je `Import["/tmp/matrix2.mtx", "Graphics"]`

ZDROJ: http://reference.wolfram.com/mathematica/ref/format/MTX.html

TODO: moznosti implementace, zvolene reseni, ostatni reseni

Kapitola 6

# Realizace

### 6.0.3 Pseudokódy

(XXX: ?) Pro pseudokódy v této práci platí, že pole jsou indexovaná od nuly a for cyklus $fori \leftarrow 0to10$ bude iterovat do devátého prvku.

## 6.1 MatrixMarket

TODO: posat format matrixmarket

### 6.1.1 Generátor řídkých matic

**Algorithm 10** Generování řídkých matic

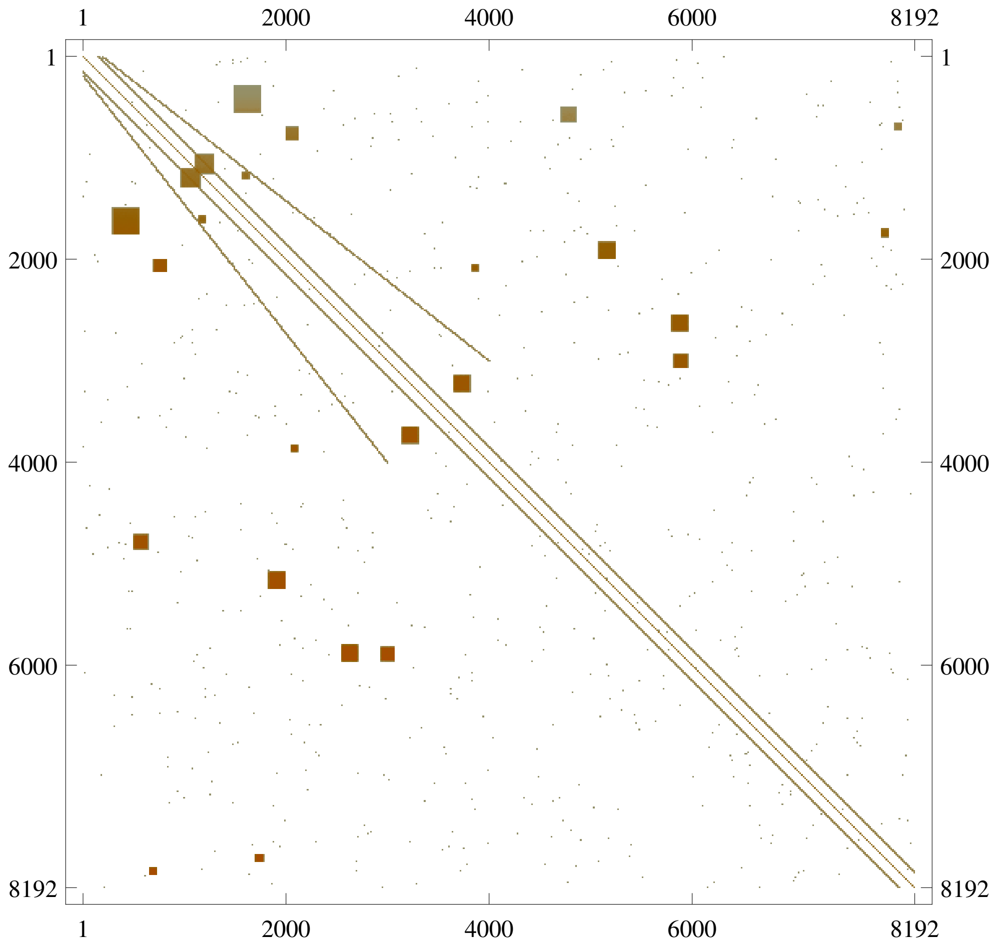
```
procedure SparseMatrixGenerator(file, width, height, ItemList)
    MtxWrapper ← InitMtxWrapper();
    MtxWrapper.PositionVector ← InitVector();
    for all Item ∈ ItemList do
        MtxWrapper.addItem(Item.y, Item.x, Item.properties);
        if Item.type == Mirrored then
            MtxWrapper.addItem(Item.x, Item.y, Item.properties);
        end if
    end for
    MtxWrapper.PositionVector.sort();
    MtxWrapper.PositionVector.removeDuplicates();
    MtxWrapper.write(file);
end procedure
```

Generátor řídkých matic byl implementován v jednom souboru. Lze spouštět s následujícími parametry:

- `-c` matice bude obsahovat hlavní diagonálu

- -H <celé číslo> výška matice
- -i <typ,a,b,c,d,...> seznam objektů, které se do matice přidají
  - diagonal,ay,ax,by,bx,sparsity prvky v přímce od bodu [ax,ay] do bodu [bx,by] s řídkostí sparsity
  - block,ay,ax,by,bx,sparsity blok prvků v obdelníku ohraničiného body [ax,ay] a [bx,by] s řídkostí sparsity
- -n <celé číslo> velikost matice
- -o <soubor> cílový soubor (lze použít i stdout)
- -s <desetinné číslo> řídkost matice (sparsity)
- -S <desetine číslo> startovací číslo
- -W <celé číslo> šířka matice
- -h zobraz nápovědu
- -o <soubor> cílový soubor (lze použít i stdout)
- -v vypisuj průběh generování (verbose)

$./tests/bin/matrix_generator-n8192-s0.00001-imdiagonal, 150, 0, 8192, 8042, 0.95, mdiagonal, 2$
$o/tmp/matrix2.mtx$




Obrázek 6.1: Matice vygenerovaná generátorem

## 6.2 Optimalizace

TODO: popsat moznosti optimalizace

## 6.3 ? Design implementace

TODO: C, testy

## 6.4 Měření

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

# Závěr

# Literatura


[1] Cenk, M.; Hasan, M. A.: On the Arithmetic Complexity of Strassen-Like Matrix Multiplications. *IACR Cryptology ePrint Archive*, ročník 2013, 2013: str. 107.

[2] Coppersmith, D.; Winograd, S.: Matrix Multiplication via Arithmetic Progressions. *J. Symb. Comput.*, ročník 9, č. 3, 1990: s. 251–280.

[3] 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.

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

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

[6] Tvrdík, P.; Šimeček, I.: A new diagonal blocking format and model of cache behavior for sparse matrices.

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

[8] 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.

[9] 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

2.1 Strassen (TODO: převzato z wikipedie: předělat?) . . . . . . . . . . 11
2.2 Ukázka numerické stability Strassenova algoritmu . . . . . . . . . . . 14
2.3 Matice orsirr_1 před vynásobením . . . . . . . . . . . . . . . . . . . 16
2.4 Matice orsirr_1 po vynásobením sama se sebou . . . . . . . . . . . . 17

3.1 Matice uložená ve formátu CSR . . . . . . . . . . . . . . . . . . . . . 22

6.1 Matice vygenerovaná generátorem . . . . . . . . . . . . . . . . . . . 31

# Seznam algoritmů


1 Násobení matic podle definice . . . . . . . . . . . . . . . . . . . . 7
2 Násobení transponovanou maticí . . . . . . . . . . . . . . . . . . . 8
3 Násobení po řádcích . . . . . . . . . . . . . . . . . . . . . . . . 8
4 Rekurzivní násobení . . . . . . . . . . . . . . . . . . . . . . . . 9
5 Strassenův algoritmus . . . . . . . . . . . . . . . . . . . . . . . 12
6 Násobení matice COO s vektorem . . . . . . . . . . . . . . . . . 20
7 Násobení dvou CSR matic . . . . . . . . . . . . . . . . . . . . . 21
8 Násobení matice CSR s vektorem . . . . . . . . . . . . . . . . . . 23
9 Násobení dvou COO matic . . . . . . . . . . . . . . . . . . . . . 23
10 Generování řídkých matic . . . . . . . . . . . . . . . . . . . . . . 29
*

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
```