

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: Doplněte jméno vedoucího práce

6. dubna 2014



---

## Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstráňte 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

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2014 Doplňte Vaše křestní jméno/jména Doplňte Vaše příjmení. Všechna práva vyhrazena.

*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.*

### **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** Nahradte seznamem klíčových slov v češtině oddělených čárkou.

---

## Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

**Keywords** Nahradte seznamem klíčových slov v angličtině oddělených čárkou.



---

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Úvod do problematiky</b>	<b>3</b>
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
<b>2 Algoritmy násobení matic</b>	<b>7</b>
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 . . . . .	15
2.8 Rychlé násobení řídkých matic . . . . .	15
2.9 Další algoritmy pro řídké maticice . . . . .	16
<b>3 Formáty uložení řídkých matic</b>	<b>19</b>
3.1 COO - Coordinate list . . . . .	19
3.2 CSR - Compressed sparse row . . . . .	19
3.3 BSR - Block Sparse Row . . . . .	19
3.4 Quadtree . . . . .	19
3.5 ? . . . . .	19

<b>4</b>	<b>Modifikace formátu quadtree</b>	<b>21</b>
<b>5</b>	<b>Analýza a návrh</b>	<b>23</b>
<b>6</b>	<b>Realizace</b>	<b>25</b>
6.1	MatrixMarket . . . . .	25
6.2	Optimalizace . . . . .	25
6.3	? Design implementace . . . . .	25
6.4	Měření . . . . .	25
	<b>Závěr</b>	<b>27</b>
	<b>Literatura</b>	<b>29</b>
<b>A</b>	<b>Seznam použitých zkratk</b>	<b>31</b>
	<b>Seznam obrázků</b>	<b>31</b>
<b>B</b>	<b>Obsah přiloženého CD</b>	<b>35</b>

---

## Seznam obrázků

2.1	Strassen (převzato z wikipedie, předělat?) . . . . .	11
2.2	Strassen stability . . . . .	14
2.3	Matice orsirr_1 před vynásobením . . . . .	17
2.4	Matice orsirr_1 po vynásobením sama se sebou . . . . .	18



---

# Úvod





# Úvod do problematiky

## 1.1 Použití násobení matic

TODO: kde se používá násobení matic

## 1.2 Matice

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

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

Matice  $\mathbf{M}$  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

Matice  $\mathbf{V}$  typu  $(1, n)$  nazveme vektorem.

TODO=popsat vektory poradně FIXME=muzu to takhle zjednodušit?

## 1.4 Násobení matic

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

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

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  $2^N$ .

### 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 řídkých 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 standardních 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  $\rightarrow$  SoA

1.8.4 Loop tiling



## 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  $row \leftarrow 0$  to  $A.height$  do           ▷ řádky
3:     for  $col \leftarrow 0$  to  $B.width$  do           ▷ sloupce
4:        $sum \leftarrow 0$ ;
5:       for  $i \leftarrow 0$  to  $A.height$  do
6:          $sum \leftarrow sum + A[row][i] * B[i][col]$ ;
7:       end for
8:        $C[row][col] \leftarrow sum$ ;
9:     end for
10:  end for
11: end procedure
```

---

Z pseudokódu je vidět, že ve dvou for cyklech provádíme  $N$  násobení a  $N$  sčítání. 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-TRANPOSE( $A, B, C$ ) ▷  $A, B, C$  jsou matice
2:    $B \leftarrow \text{transpose}(B)$ 
3:   for  $rowA \leftarrow 0$  to  $A.height$  do ▷ řádky
4:     for  $rowB \leftarrow 0$  to  $B.height$  do ▷ sloupce
5:        $sum \leftarrow 0$ ;
6:       for  $i \leftarrow 0$  to  $A.height$  do
7:          $sum \leftarrow sum + A[rowA][i] * B[i][rowB]$ ;
8:       end for
9:        $C[rowA][rowB] \leftarrow sum$ ;
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 Compressed 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  $r \leftarrow 0$  to  $A.height$  do ▷ řádky matice  $A$  i  $B$ 
3:     for  $cA \leftarrow 0$  to  $A.width$  do ▷ sloupce matice  $A$ 
4:       for  $cB \leftarrow 0$  to  $B.width$  do ▷ sloupce matice  $B$ 
5:          $C[r][cA] \leftarrow C[r][cA] + A[r][cA] * B[r][cB]$ ;
6:       end for
7:     end for
8:   end for
9: end procedure
```

---

## 2.4 Rekurzivní násobení

Pro matice  $A$  i  $B$  o stejné velikosti  $2^N$  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.

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

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:      $C[cy][cx] \leftarrow C[cy][cx] + A[ay][ax] \cdot B[by][bx]$ ;
4:     return;
5:   end if
6:   for all  $r \in \{0, n/2\}$  do
7:     for all  $c \in \{0, n/2\}$  do
8:       for all  $i \in \{0, n/2\}$  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.

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

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

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

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

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 mistrovské metody je **MMM-recursive**(n) =  $O(n^3)$ .

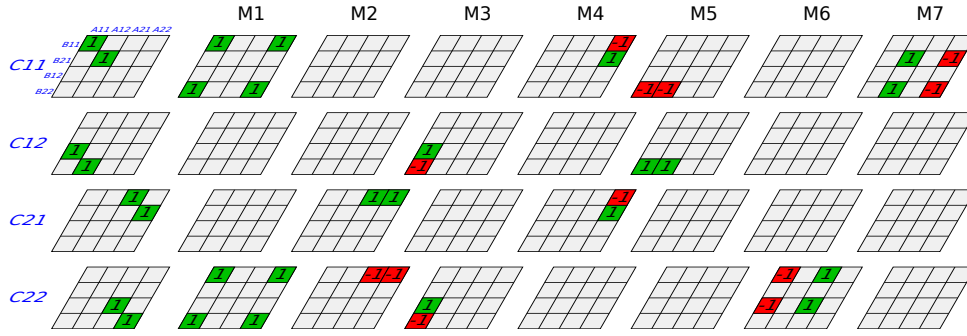
## 2.5 Strassenův algoritmus

V roce 1969 Volker Strassen v časopise Numerische Mathematik publikoval článek [4], ve kterém jako první představil algoritmus násobení dvou matic s menší asymptotickou složitostí než algoritmus 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 [2] 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 seskládány. 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 následovně:

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

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

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

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

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

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

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

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

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

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  $x$ . Parametry 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:    $m[9] \leftarrow \text{init-matrices}(9, h)$ ; ▷ 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.

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 mistrovské metody je  $\text{MMM-strassen}(n) = O(n^{\log_2 7}) \approx O(n^{2.8})$ .

Stejně jako v předešlém algoritmu, i zde demonstrujeme výpočet levého horního prvku matice z násobení dvou matic o velikosti dva. Místo parametrické matice použijeme konkrétní desetinná čísla, abysme ukázali numerickou stabilitu Strassenova algoritmu. Pro ukázkou 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 následovně:

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

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:

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

$$M_1 = (30.234 + 10.456) \cdot (0.8912 + 9.999) = 443.12 \quad (2.17)$$

$$\dots \quad (2.18)$$

$$M_4 = 10.456 \cdot (0.7891 - 0.8912) = -1.067 \quad (2.19)$$

$$M_5 = (30.234 + 0.5678) \cdot 9.999 = 307.98 \quad (2.20)$$

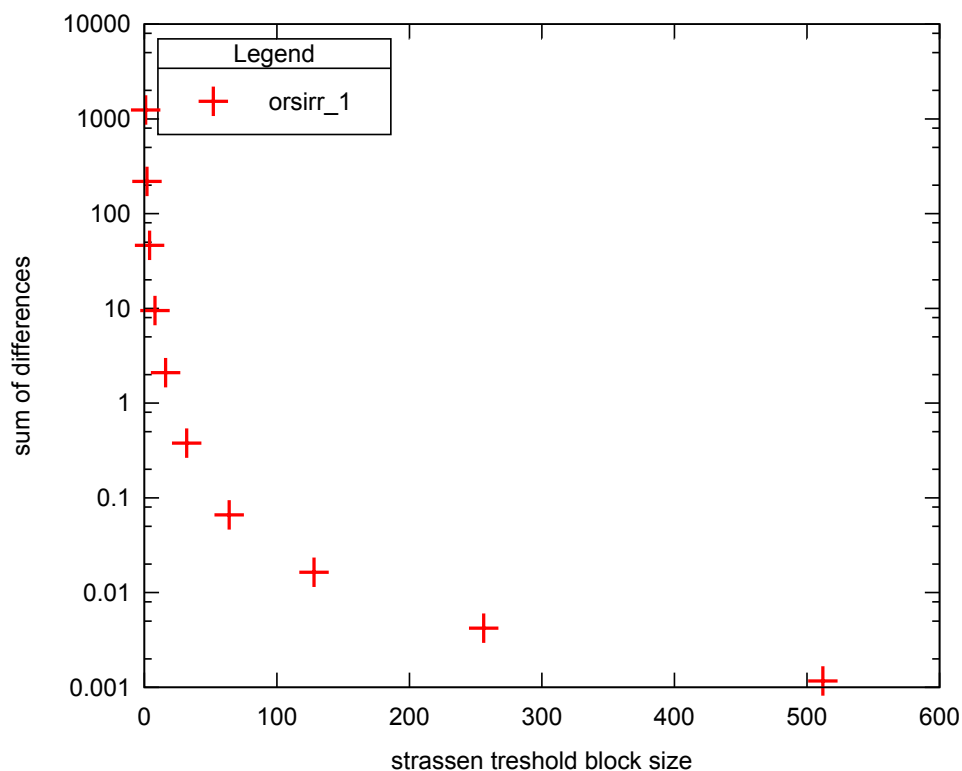
$$\dots \quad (2.21)$$

$$M_7 = (0.5678 - 10.456) \cdot (0.7891 + 9.999) = -106.67 \quad (2.22)$$

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

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 práhy a sečetli všechny rozdíly mezi výsledky. Násobení probíhalo ve dvojitě desetinné přesnosti, tedy v datovém typu `double` jazyka C99. Z grafu je vidět exponenciální růst chyby.



Obrázek 2.2: Strassen stability

Strassenův algoritmus lze ještě vylepšit. Algoritmům na stejném principu se říká Strassen-like. Pro sedm operací násobení je možné snížit počet sčítání a odečítání. Pro jednoduchost zde ovšem uvádíme originální 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 [7][6], 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 [3], 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 2.4. Kdyby nastal nejhorší možný případ, počet operací násobení při použití algoritmu podle definice pro řídké matice by byl  $1030 \times 6858 = 7063740$ . Pro tento případ ovšem stačí 46976 operací násobení. Oproti nejhoršímu možnému případu nastalo  $7063740 - 46976 = 7016764$  situací, kdy jeden ze dvou prvků byl nulový a operace násobení nemusela být provedena. Při násobení algoritmem podle definice pro husté matice by v 1092680024 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  $m = 6858$  na  $m = 23532$ . 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

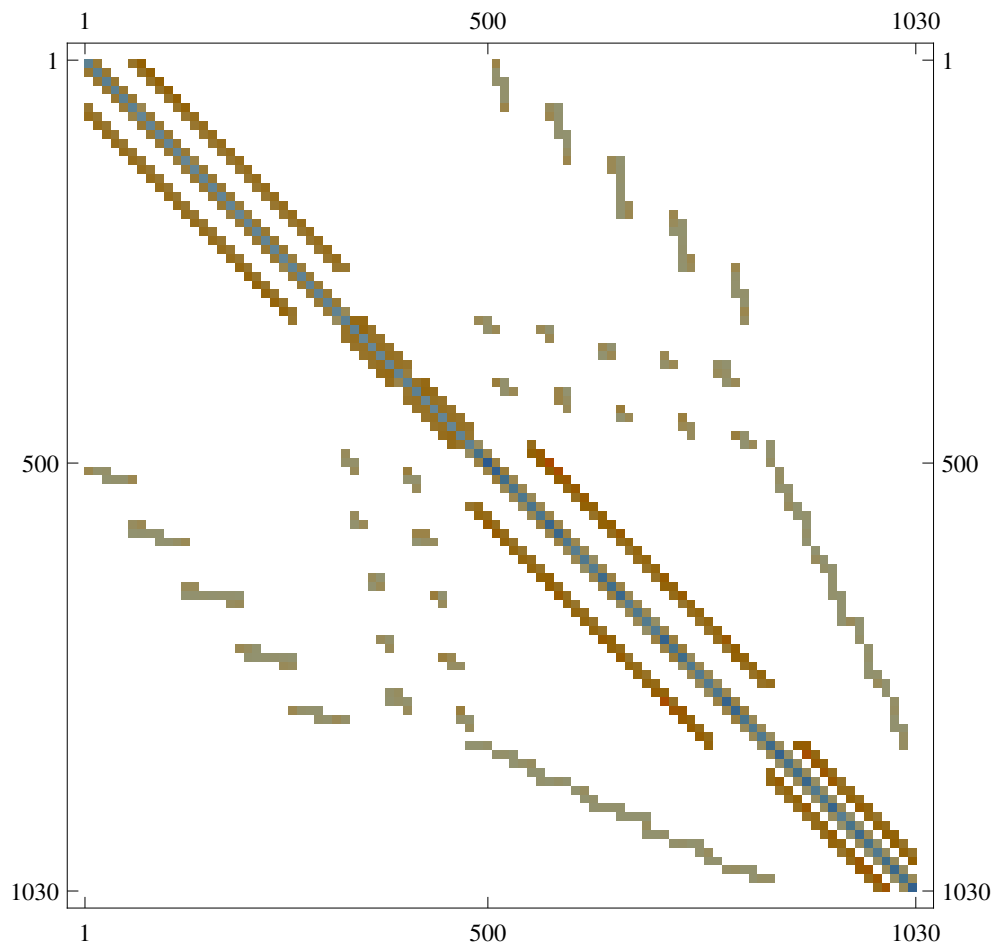
Strassenův algoritmus  $\omega = 2.8$  od  $nnz < n^{1.8}$  a algoritmus Virginie Williamsové  $\omega = 2.3$  od  $nnz < n^{1.37}$  jsou asymptoticky stejně rychlé jako algoritmus podle definice upravený pro řídké matice  $O(mn)$ , kdy  $m = nnz$ . Na příklad, pro  $n = 1000$  je tato hranice pro Strassenův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

Raphael Yuster a Uri Zwick ukázali algoritmus [8] 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 upraveným pro řídké

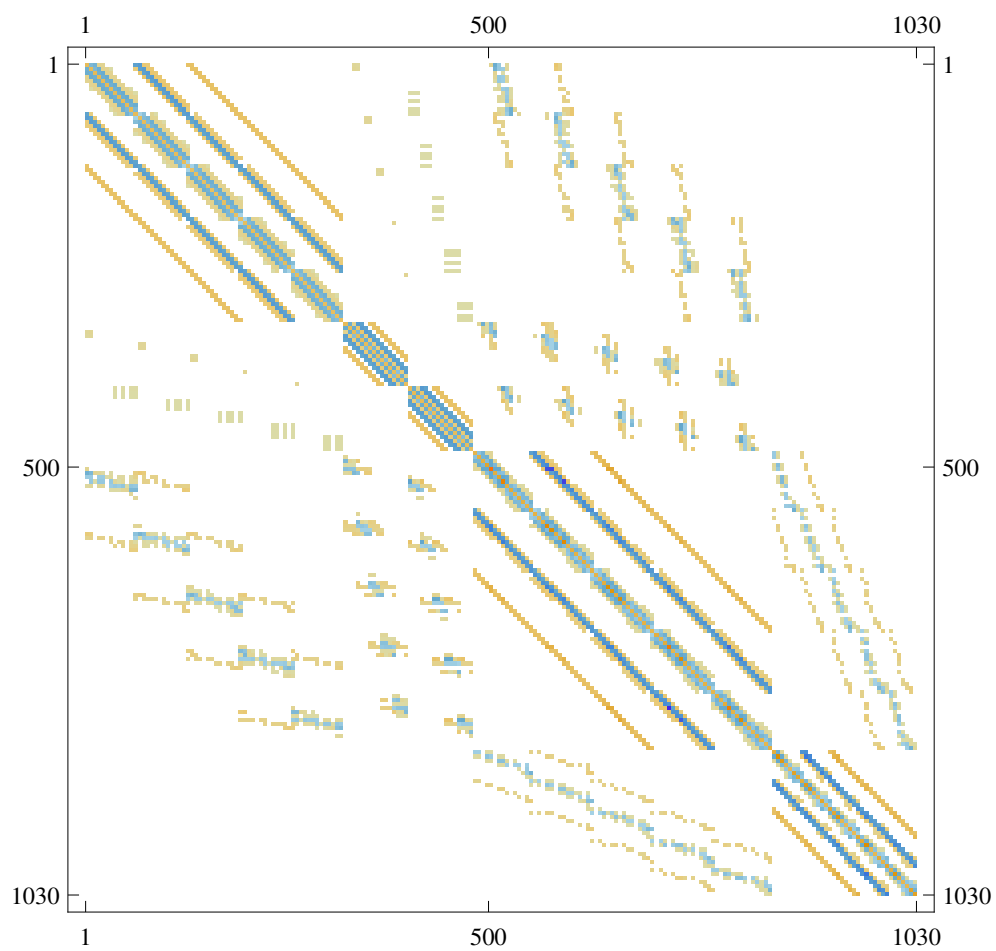
matice  $O(mn)$ . Husté permutace vynásobí v té době nejznámějším nejrychlejším algoritmem a to od Dona Coppersmitha a Shmuela Winograda s asymptotickou složitostí  $\omega = 2.3$  [1].

### 2.9 Další algoritmy pro řídké maticice

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 [5].



Obrázek 2.3: Matice orsirr\_1 před vynásobením



Obrázek 2.4: Matice orsirr\_1 po vynásobením sama se sebou



## 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 souřadnicích jednotlivých prvcích. Za druhé, ztrácíme možnost přístupu k prvku na libovolných souřadnicích 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 dvojité přesnosti, bude zabírat  $M \times N \times \text{sizeof}(\text{double}) = 100 \times 100 \times 8 = 80000\text{B} = 80\text{kB}$ . Pokud zvolíme řídký formát matice, kde ke každému elementu uložíme i jeho  $x$  a  $y$  souřadnici, tak do 80kB uložíme  $80000 / (\text{sizeof}(\text{int}) + \text{sizeof}(\text{int}) + \text{sizeof}(\text{double})) = 80000/16 = 5000$  elementů. Pokud matice obsahuje více jak 50 % nulových elementů, vyplatí se nám ji uložit do řídkého formátu.



## 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



## 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



## Realizace

### 6.1 MatrixMarket

TODO: popsat 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] Coppersmith, D.; Winograd, S.: Matrix Multiplication via Arithmetic Progressions. *J. Symb. Comput.*, ročník 9, č. 3, 1990: s. 251–280.
- [2] 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.
- [3] Stothers, A. J.: *On the Complexity of Matrix Multiplication*. diploma thesis, University of Edinburgh, 2010.
- [4] Strassen, V.: Gaussian Elimination is not Optimal. *Numerische Mathematik*, ročník 13, č. 4, Prosinec 1967: s. 354–355.
- [5] Tvrdík, P.; Šimeček, I.: A new diagonal blocking format and model of cache behavior for sparse matrices.
- [6] Williams, V. V.: Breaking the Coppersmith-Winograd barrier. 2011.
- [7] 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.
- [8] Yuster, R.; Zwick, U.: Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, ročník 1, č. 1, 2005: s. 2–13.



## Seznam použitých zkratk

**GUI** Graphical user interface

**XML** Extensible markup language

---

## Seznam obrázků

2.1	Strassen (převzato z wikipedie, předělat?) . . . . .	11
2.2	Strassen stability . . . . .	14
2.3	Matice orsirr_1 před vynásobením . . . . .	17
2.4	Matice orsirr_1 po vynásobením sama se sebou . . . . .	18



---

## 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
*		





## 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 L <sup>A</sup> T <sub>E</sub> X
	text .....	text práce
	thesis.pdf .....	text práce ve formátu PDF
	thesis.ps .....	text práce ve formátu PS