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

3. 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 3. 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 $\mathbf{A}$ typu $(m,\ n)$ je mn uspořádaných prkvů z množiny $\mathbf{R}$. 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 $\mathbf{A}$. Matici $\mathbf{A}$ zapisujeme do řádků a sloupců takto:

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

Matici $\mathbf{M}$ 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

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

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

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:

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

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).

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 $O(1)$.

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

TODO, i priklad?

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 šírka matice A, tak i výška matice B.

Algorithm 1 Násobení matic podle definice

```
procedure MMM-DEFINITION(A, B, C)                ▷ A,B,C jsou matice
    for row ← 0 to A.height do                   ▷ řádky
        for col ← 0 to B.width do                ▷ sloupce
            sum ← 0;
            for i ← 0 to A.height do
                sum ← sum + A[row][i] * B[i][col];
            end for
            C[row][col] ← sum;
        end for
    end for
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.

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.

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

```
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.

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

Pro ilustraci jako příklad uvádíme výpočet horního levého prvku v násobení dvou matic o velikosti 2^2.

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

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

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

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

Kvůli režii rekurzivního dělení nezmenšujeme matice až na velikost dva. Vhodná velikost matice je například taková, co se vejde do L1 cache.

Algorithm 4 Rekurzivní násobení

```
procedure MMM-recursive(A, B, C, ax, ay, bx, by, cx, cy, s)
    if s = 1 then
        C[cy][cx] ← C[cy][cx] + A[ay][ax] · B[by][bx];
        return;
    end if
    for all r ∈ {0, s/2} do
        for all c ∈ {0, s/2} do
            for all i ∈ {0, s/2} do
                MMM-recursive(A, B, C, ax + i, ay + r, bx + c, by + i, cx + c, cy + r, s/2);
            end for
        end for
    end for
end procedure
```

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

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

Protože platí $a = 8, b = 2, r = \log_8 2, n^r = n^{\log_8 2} = 3 = O(n^2)$, je složitost skutečně `MMM-recursive`(n) $= O(n^3)$.

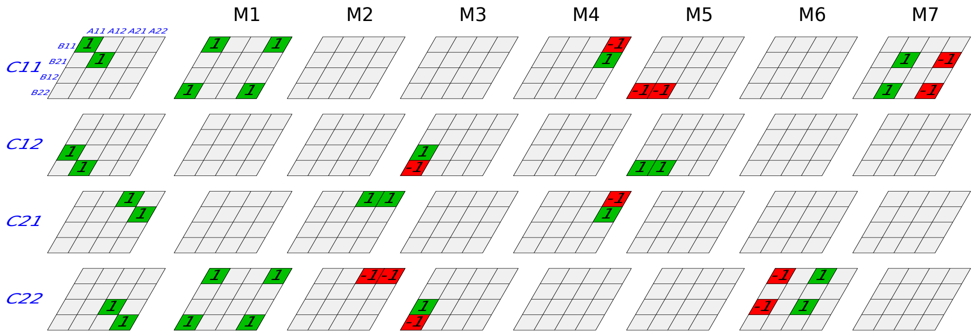
2.5 Strassenův algoritmus

V roce 1969 Volker Strassen v časopise Numerische Mathematik publikoval článek [2], ve kterém jako první představil algoritmus násobení dvou matic s menší složitostí než $O(n^3)$.

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 $O(n^3)$.

Volken 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 o podmaticích velikosti čtyři, které sčítance přičítají a které odečítají.

Zápis Strassenova algoritmu vypadá následnovně:



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

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

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

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

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

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

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

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

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

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

Strassenův algoritmus pořebuje sedm operací násobení

TODO

Strassenův algoritmus má lepší asymptotickou složistost $O(n^{2.808})$, to však za cenu numerické stability. Z definice tento algoritmus od $n > 655$ potřebuje méně instrukcí než algoritmus podle definice[zdroj]

TODO: zdroj: Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7. 2007, Revised Papers | Front Cover Ralf Lämmel, Joost Visser, João Saraiva | Springer, Oct 8, 2008 - Computers - 521 pages

Numerickou stabilitu Strassenova algoritmu demonstrujeme na výpočtu jednoho prvku z násobení dvou matic o velikosti dva. Pro ukázku budeme uvažovat počítač, který u čísel ukládá pouze tři destinná čísla.

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

$$\begin{pmatrix} 0.1234 & 0.5678 \\ 0.9123 & 0.4567 \end{pmatrix} \cdot \begin{pmatrix} 0.8912 & 0.3456 \\ 0.7891 & 0.2345 \end{pmatrix} = \begin{pmatrix} 0.5579 & \dots \\ \dots & \dots \end{pmatrix} \tag{2.15}$$

Správný výsledek je $0.1234 \times 0.8912 + 0.5678 \times 0.7891 = 0.10997408 + 0.44805098 = 0.55802506$.

Nyní výpočet provedeme pomocí Strassenova algoritmu:

$$\begin{pmatrix} 0.1234 & 0.5678 \\ 0.9123 & 0.4567 \end{pmatrix} \cdot \begin{pmatrix} 0.8912 & 0.3456 \\ 0.7891 & 0.2345 \end{pmatrix} \tag{2.16}$$

$$M_1 = (0.1234 + 0.4567) \cdot (0.8912 + 0.2345) = 0.6530 \tag{2.17}$$

$$\dots \tag{2.18}$$

$$M_4 = 0.4567 \cdot (0.7891 - 0.8912) = -0.0466 \tag{2.19}$$

$$M_5 = (0.1234 + 0.5678) \cdot 0.2345 = 0.1620 \tag{2.20}$$

$$\dots \tag{2.21}$$

$$M_7 = (0.5678 - 0.4567) \cdot (0.7891 + 0.2345) = 0.1137 \tag{2.22}$$

$$\begin{pmatrix} M_1 + M_4 - M_5 + M_7 & \dots \\ \dots & \dots \end{pmatrix} = \begin{pmatrix} 0.5581 & \dots \\ \dots & \dots \end{pmatrix} \tag{2.23}$$

TODO: pro strassena ja potreba dodelat i ostatni prvky

Strassenův algorimus lze ještě vylepšit. Pro sedm operací násobení je monžné snížit počet sčítání a odečítání. Pro jednoduchost zde ovšem uvádíme originalní algoritmus.

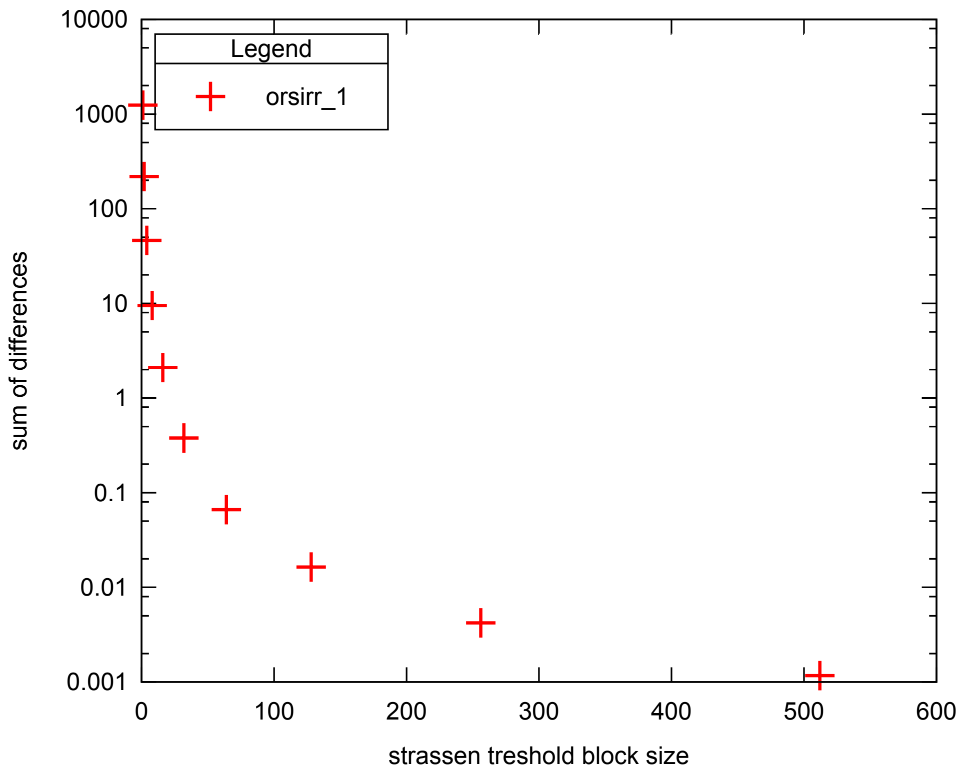
Tady jsem zkusil matici z MM:

2.6 Coppersmith-Winograd algoritmus

TODO: rict ze ma dobrou O(n2735477), ale velkou konstantu a je v praxi zatim nepouzitelnej

2.7 Další algoritmy

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 2.2: Strassen stability

Algorithm 5 Strassenův algoritmus

procedure MMM-STRASSEN$(A, B, C, ax, ay, bx, by, cx, cy, s)$
  if $s = 1$ **then**
    $C[cy][cx] \leftarrow C[cy][cx] + A[ay][ax] \cdot B[by][bx];$
    $return;$
  end if
  $h \leftarrow s/2;$
  $M[9] \leftarrow initMatrices(9, h);$ ▷ devět pomocných matic
  $offsetAdd(M[8], 0, 0, A, ax, ay, A, ax + h, ay + h, h);$ ▷ M1
  $offsetAdd(M[9], 0, 0, B, bx, by, B, bx + h, by + h, h);$
  `MMM-strassen`$(M[8], M[9], M[1], 0, 0, 0, 0, 0, 0, h);$
  $offsetAdd(M[8], 0, 0, A, ax, ay + h, A, ax + h, ay + h, h);$ ▷ M2
  `MMM-strassen`$(M[8], B, M[2], 0, 0, bx, by, 0, 0, h);$
  $offsetSub(M[8], 0, 0, B, bx + h, by, B, bx + h, by + h, h);$ ▷ M3
  `MMM-strassen`$(A, M[8], M[3], ax, ay, 0, 0, 0, 0, h);$
  $offsetSub(M[8], 0, 0, B, ax, ay + h, B, ax, ay, h);$ ▷ M4
  `MMM-strassen`$(A, M[8], M[4], ax + h, ay + h, 0, 0, 0, 0, h);$
  $offsetAdd(M[8], 0, 0, A, ax, ay, A, ax + h, ay, h);$ ▷ M5
  `MMM-strassen`$(M[8], B, M[5], 0, 0, bx + h, by + h, 0, 0, h);$
  $offsetSub(M[8], 0, 0, A, ax, ay + h, A, ax, ay, h);$ ▷ M6
  $offsetAdd(M[9], 0, 0, B, bx, by, B, bx + h, by, h);$
  `MMM-strassen`$(M[8], M[9], M[6], 0, 0, 0, 0, 0, 0, h);$
  $offsetSub(M[8], 0, 0, A, ax + h, ay, A, ax, ay, h);$ ▷ M7
  $offsetAdd(M[9], 0, 0, B, bx, by + 2, B, bx + h, by + h, h);$
  `MMM-strassen`$(M[8], M[9], M[7], 0, 0, 0, 0, 0, 0, h);$
  $offsetAdd(M[8], 0, 0, M[1], 0, 0, M[4], 0, 0, h);$ ▷ C1,1
  $offsetSub(M[8], 0, 0, M[8], 0, 0, M[5], 0, 0, h);$
  $offsetAdd(C, cx, cy, M[8], 0, 0, M[7], 0, 0, h);$
  $offsetAdd(C, cx + h, cy, M[3], 0, 0, M[5], 0, 0, h);$ ▷ C1,2
  $M[8] \leftarrow offsetAdd(C, cx, cy + h, M[2], 0, 0, M[4], 0, 0, h);$ ▷ C2,1
  $M[8] \leftarrow offsetSub(M[8], 0, 0, M[1], 0, 0, M[2], 0, 0, h);$ ▷ C2,2
  $M[8] \leftarrow offsetAdd(M[8], 0, 0, M[8], 0, 0, M[3], 0, 0, h);$
  $M[8] \leftarrow offsetAdd(C, cx + h, cy + h, M[8], 0, 0, M[6], 0, 0, h);$
end procedure

Kapitola 3

Formáty uložení řídkých matic

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

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
```