

MASARYKOVA UNIVERZITA  
FAKULTA INFORMATIKY



# **Dekonvolúcia biomedicínskych obrazov na GPU**

DIPLOMOVÁ PRÁCA

**Michal Kuderjavý**

Brno, jar 2013

## **Prehlásenie**

Prehlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

**Vedúci práce:** RNDr. Pavel Karas

## **Pod'akovanie**

Rád by som pod'akoval vedúcemu mojej práce RNDr. Pavlovi Karasovi, za jeho cenné rady, pripomienky, komentáre a trpezlivosť počas písania mojej práce. Taktiež ďakujem mojej rodine za ich podporu v priebehu vypracovania tejto práce.

## **Zhrnutie**

Cieľom práce bolo vybrať vhodné iteratívne aj neiteratívne algoritmy a implementovať ich na grafickej karte a na systémoch s viacerými grafickými kartami. Jednotlivé algoritmy boli v práci porovnané z hľadiska presnosti, rýchlosti a kvality výstupu.



## **Kľúčové slová**

dekonvolúcia, konvolúcia, Fourierova transformácia, fluorescenčná mikroskopia, konfokálna mikroskopia, CUDA, Wienerova dekonvolúcia, ICTM, EM-MLE, Huygens

## Obsah

|       |                                                                 |    |
|-------|-----------------------------------------------------------------|----|
| 1     | <b>Základné pojmy</b>                                           | 5  |
| 1.1   | <i>3D obraz</i>                                                 | 5  |
| 1.2   | <i>Fluorescenčná mikroskopia</i>                                | 6  |
| 1.3   | <i>Konfokálna mikroskopia</i>                                   | 8  |
| 1.4   | <i>Fourierova transformácia</i>                                 | 8  |
| 1.5   | <i>Konvolúcia</i>                                               | 10 |
| 1.6   | <i>Rozptylová funkcia</i>                                       | 11 |
| 1.7   | <i>Šum</i>                                                      | 12 |
| 1.8   | <i>CUDA</i>                                                     | 13 |
| 2     | <b>Dekonvolúcia</b>                                             | 15 |
| 2.1   | <i>Wienerova dekonvolúcia</i>                                   | 15 |
| 2.2   | <i>EM-MLE</i>                                                   | 16 |
| 2.3   | <i>ICTM</i>                                                     | 17 |
| 3     | <b>Dekonvolúcia na GPU</b>                                      | 19 |
| 3.1   | <i>Príprava obrázkov pre rýchlu konvolúciu</i>                  | 19 |
| 3.2   | <i>Pseudokódy dekonvolučných metód</i>                          | 22 |
| 3.2.1 | <i>Wienerova dekonvolúcia</i>                                   | 22 |
| 3.2.2 | <i>EM-MLE</i>                                                   | 23 |
| 3.2.3 | <i>ICTM</i>                                                     | 25 |
| 4     | <b>Dekonvolúcia na systémoch s viacerými grafickými kartami</b> | 28 |
| 4.1   | <i>Motivácia</i>                                                | 28 |
| 4.2   | <i>Možnosti</i>                                                 | 28 |
| 4.3   | <i>Funkcia dot a funkcia sum_squared</i>                        | 29 |
| 4.4   | <i>Rýchla konvolúcia</i>                                        | 29 |
| 4.4.1 | <i>Decimation in frequency</i>                                  | 30 |
| 4.4.2 | <i>Signal and kernel tiling</i>                                 | 31 |
|       | <i>Signal tiling</i>                                            | 32 |
|       | <i>Kernel tiling</i>                                            | 32 |
|       | <i>Signal and kernel tiling</i>                                 | 32 |
| 4.4.3 | <i>Porovnanie a výber vhodného spôsobu</i>                      | 33 |
| 4.4.4 | <i>Wienerova dekonvolúcia</i>                                   | 34 |
| 4.4.5 | <i>EM-MLE a ICTM</i>                                            | 34 |

---

|     |                                     |    |
|-----|-------------------------------------|----|
| 5   | <b>Popis programu</b>               | 36 |
| 5.1 | <i>Stručný popis</i>                | 36 |
| 5.2 | <i>Dodatočný padding obrázkov</i>   | 37 |
| 5.3 | <i>Dodatočné buffre na GPU</i>      | 37 |
| 5.4 | <i>Výmena okrajov</i>               | 38 |
| 6   | <b>Experimenty</b>                  | 40 |
| 6.1 | <i>Použité miery</i>                | 40 |
| 6.2 | <i>Príprava</i>                     | 41 |
| 6.3 | <i>Porovnanie presnosti výpočtu</i> | 41 |
| 6.4 | <i>Porovnanie rýchlosti výpočtu</i> | 44 |
| 6.5 | <i>Porovnanie kvality výstupu</i>   | 46 |
| 7   | <b>Záver</b>                        | 48 |
| A   | <b>1. typ dekonvolvovaných dát</b>  | 51 |
| B   | <b>2. typ dekonvolvovaných dát</b>  | 54 |
| C   | <b>PSF</b>                          | 57 |

## Úvod

Získavanie obrazu v oblasti biológie a medicíny je náročná úloha. Kvalita výsledného obrazu býva v praxi limitovaná mnohými faktormi. Obraz je po prechode optickou sústavou, ktorá ho sníma často rozmazaný a obsahuje šum. To je dané hlavne fyzikálnymi obmedzeniami snímania, ktoré spôsobujú, že nezískame skutočný obraz. Prechod obrazu optickou sústavou snímacieho zariadenia je v oblasti spracovania obrazu modelovaný ako konvolúcia [1].

Dekonvolúcia je opačný proces ku konvolúcii, teda snaha o získanie originálneho, skutočného obrazu z toho, ktorý sme nasnímali. Obecne môžeme povedať, že získať presný skutočný obraz nie je možné, keďže tieto obrazy obsahujú šum, ktorý je vo svojej podstate náhodný. Rozmazanie obrazu tiež nie je operácia, ktorú by sme mohli úplne zvrátiť. Z týchto dôvodov je dekonvolúcia zložitá a často aj výpočtovo náročná úloha.

Fakt, že výkon grafických kariet (GPU) je v dnešnej dobe oveľa väčší, než výkon procesorov (CPU) je už obecné známy [2]. Grafické karty tento väčší výkon dosahujú hlavne masívnym paralelizmom. Aby bolo možné tento potenciál grafických kariet využiť, je potrebné mať úlohu, ktorá je dobre paralelizovateľná a dostatočne výpočtovo náročná. Túto podmienku spĺňa veľa úloh z oblasti spracovania obrazu a dekonvolúcia nie je žiadnou výnimkou.

Cieľom tejto práce je vybrať vhodné iteratívne a neiteratívne algoritmy na výpočet dekonvolúcie a implementovať ich na GPU za účelom zrýchlenia výpočtu oproti CPU algoritmom. Druhým cieľom práce je implementovať tieto algoritmy na systémoch s viacerými grafickými kartami a dosiahnuť tak ešte väčšiu rýchlosť výpočtu, ale aj zníženie pamäťovej náročnosti na jednotlivých GPU, keďže pamäť na GPU je obecné menšia, než CPU pamäť. Vďaka tomu budeme schopní dekonvolvovať na GPU veľké obrázky.

Úvodná kapitola práce nás zoznamuje so základnými pojmami, ktoré budeme v práci používať. Ďalšia kapitola sa potom bližšie venuje procesu dekonvolúcie a konkrétnejšie popisuje, s akými problémami sa v tejto oblasti obecné stretávame. Následne sú stručne popísané tri známe a používané dekonvolučné metódy, ktoré sú implementované na GPU v rámci tejto

---

práce.

Samotné jadro práce začína kapitolou 3, ktorá sa venuje dekonvolúcii na GPU a uvádza pseudokódy jednotlivých metód, počítaných na GPU. V nasledujúcej kapitole sa venujeme dekonvolúcii počítanej na systémoch s viacerými grafickými kartami. V tejto kapitole popisujeme spôsoby, pomocou ktorých môžeme jednotlivé metódy počítať s využitím viacerých grafických kariet, porovnávame ich a vyberáme ten najvhodnejší pre danú metódu.

Kapitola 5 obsahuje popis programu, ktorý je súčasťou tejto práce a niekoľko poznámok, ktoré súvisia s implementáciou jednotlivých metód. V kapitole 6 potom porovnávame presnosť, rýchlosť a kvalitu výstupu jednotlivých metód medzi sebou a voči komerčnému softwaru Huygens, ktorý sa bežne používa v oblasti spracovania obrazu. Táto kapitola obsahuje diskusiu dosiahnutých výsledkov. Záverečná kapitola obsahuje celkové zhodnotenie práce.

## Kapitola 1

### Základné pojmy

#### 1.1 3D obraz

Pre účely diplomovej práce rozumieme pod pojmom 3D obraz šedotónny trojrozmerný obraz, uložený v počítači ako trojrozmerné pole (matica) intenzít, nasnímaných v jednotlivých bodoch obrazu, tvoriacich pravidelnú mriežku. Matematicky pod týmto pojmom rozumieme funkciu intenzity  $I(x, y, z)$  [3]:

$$\begin{aligned} I(x, y, z) \\ x \in \{x_1, \dots, x_{nx}\}, \quad x_i = x_{min} + (i - 1) \cdot \Delta x, \quad x_i \in \mathbb{R}, \quad \Delta x \in \mathbb{R} \\ y \in \{y_1, \dots, y_{ny}\}, \quad y_j = y_{min} + (j - 1) \cdot \Delta y, \quad y_j \in \mathbb{R}, \quad \Delta y \in \mathbb{R} \\ z \in \{z_1, \dots, z_{nz}\}, \quad z_k = z_{min} + (k - 1) \cdot \Delta z, \quad z_k \in \mathbb{R}, \quad \Delta z \in \mathbb{R} \\ I \in [I_{min}, \dots, I_{max}], \quad I \in \mathbb{R} \end{aligned} \quad (1.1)$$

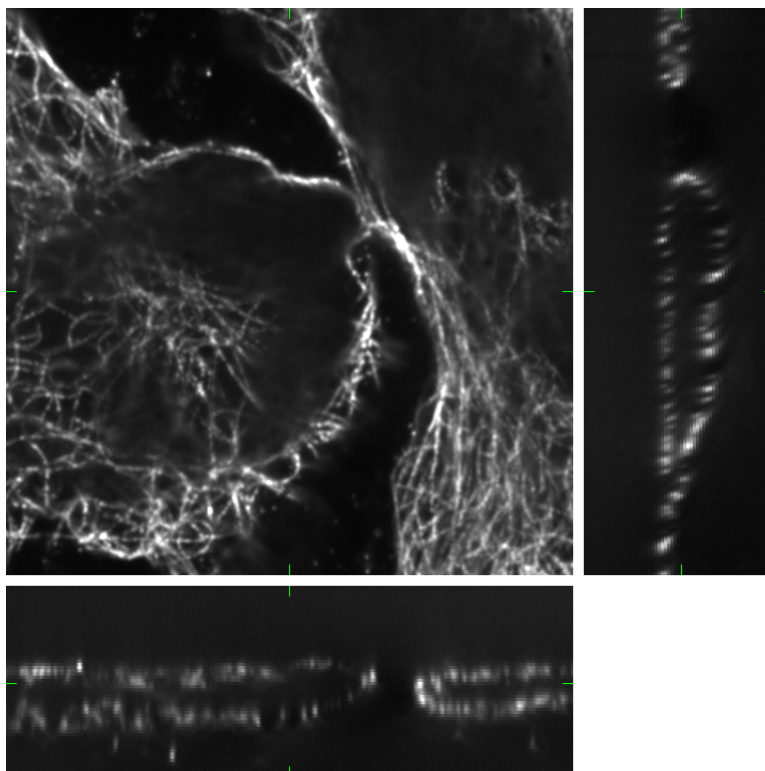
kde  $x_{min}, y_{min}, z_{min}$  predstavujú minimálne hodnoty súradníc jednotlivých osí a  $\Delta x, \Delta y, \Delta z$  predstavujú rozdiel (odstup) medzi súradnicami jednotlivých bodov. Intenzita  $I_{min}$  a  $I_{max}$  je minimálna, resp. maximálna hodnota intenzity v obrázku. Tu už nepredpokladáme pravidelné rozostupy intenzít a vo všeobecnosti je intenzita definovaná ako reálne číslo.

Takýto 3D obraz je možné nasnímať niekoľkými spôsobmi. Môžeme napríklad snímaný objekt fyzicky rozseknúť na jednotlivé 2D rezy a tie potom postupne nasnímať a zrekonštruovať z nich pôvodný 3D objekt<sup>1</sup>. Pochopiteľne, pri použití tohto spôsobu je snímaný objekt zničený. Medzi menej invazívne spôsoby snímania 3D obrazu patrí napríklad bežne používaná magnetická rezonancia (MRI), alebo počítačová tomografia (CT). V oblasti mikroskopie môžeme 3D obraz získať napríklad fluorescenčnou alebo konfokálnou mikroskopiou. Týmto dvom spôsobom sú, vzhľadom k téme práce, venované samostatné podkapitoly.

Vo väčšine prípadov je veľkosť bodu nasnímaného obrazu v jednotlivých osách rozdielna. Veľkosť bodu v ose  $x$  a  $y$  je daná hlavne použitým

---

1. Takýmto spôsobom bolo napríklad nasnímané celé mužské a ženské telo v rámci projektu The Visible Human Project



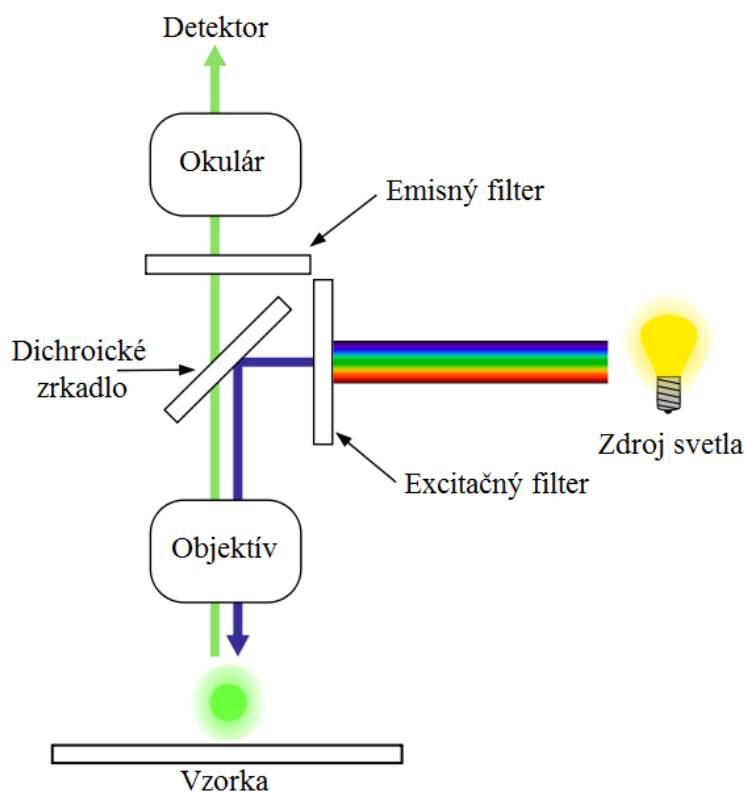
Obr. 1.1: 3D obrázok nasnímaný konfokálnym mikroskopom. Zobrazené sú rezy v rovine  $xy$ ,  $yz$  a  $xz$  týmto obrázkom.

zväčšením objektívu, zatiaľ čo veľkosť bodu v ose  $z$  je daná hlavne krokom posunu (zvyčajne sa obraz sníma po jednotlivých rezoch v rovine  $xy$ ). V osi  $z$  má tak obraz často najhoršie vlastnosti a malé rozlíšenie. Príklad 3D obrazu nasnímaného konfokálnym mikroskopom je na obrázku 1.1, kde sú zobrazené rezy v rovine  $xy$ ,  $yz$  a  $xz$  týmto obrázkom<sup>2</sup>. Rozmery jedného bodu (voxelu) sú dané konkrétnou konfiguráciou mikroskopu a v tomto prípade je to  $67 \times 67 \times 200$  nm.

## 1.2 Fluorescenčná mikroskopia

Fluorescenčná mikroskopia patrí do oblasti optickej, resp. svetelnej mikroskopie. Fluorescenčná mikroskopia využíva jav zvaný fluorescencia. Je to emisia svetla látkou, ktorá práve absorbovala svetlo, pričom platí, že emi-

2. Všetky ostatné 3D obrázky budú v práci zobrazované týmto spôsobom



Obr. 1.2: Zloženie fluorescenčného mikroskopu.

tované svetlo má nižšiu energiu, a teda aj väčšiu vlnovú dĺžku. Samotné snímané objekty túto vlastnosť nemajú, preto sú farbené fluorescenčným farbivom, ktoré túto vlastnosť má. Jav fluorescencie nám umožňuje rozlíšiť medzi svetlom, ktoré používame na ožiarenie objektu a svetlom, ktoré sa od daného objektu odráža – má totiž inú vlnovú dĺžku. Na rozlíšenie medzi týmito svetlami s rozdielnou vlnovou dĺžkou sa používa dichroické zrkadlo, ktoré odráža svetlo s kratšou vlnovou dĺžkou ako istá daná hranica a naopak, prepúšťa svetlo s dlhšou vlnovou dĺžkou ako daná hranica. Základná schéma fungovania fluorescenčného mikroskopu je zobrazená na obrázku 1.2<sup>3</sup>.

Medzi výhody fluorescenčného mikroskopu patrí schopnosť pridať fluorescenčné farbivo len ku istým častiam snímaného objektu, takže len tieto časti budú videné v mikroskope. Môžeme navyše použiť aj viac ako jedno

3. Obrázok bol prevzatý z [https://en.wikipedia.org/wiki/Fluorescence\\_microscope](https://en.wikipedia.org/wiki/Fluorescence_microscope).



farbivo, aby sme boli schopní rozlíšiť viacero častí snímaného objektu.

Najväčší problém fluorescenčnej mikroskopie spočíva v tom, že zvyčajne trpí nedostatkom svetla, čo má za dôsledok vyššiu prítomnosť šumu alebo dlhé doby expozície. Zvyšovanie intenzity svetla však nie je vhodné riešenie tohto problému, pretože potom dochádza k poškodeniu tkaniva živých buniek (fototoxicita) a javu zvanom photobleaching, čo je vyprchanie fluorescenčného farbiva ešte predtým, ako sa podarí obrázok nasnímať. Problém nedostatku svetla je preto nutné riešiť inak ako zvyšovaním intenzity svetelného zdroja, predovšetkým napríklad nákupom kvalitných komponentov mikroskopu.

### 1.3 Konfokálna mikroskopia

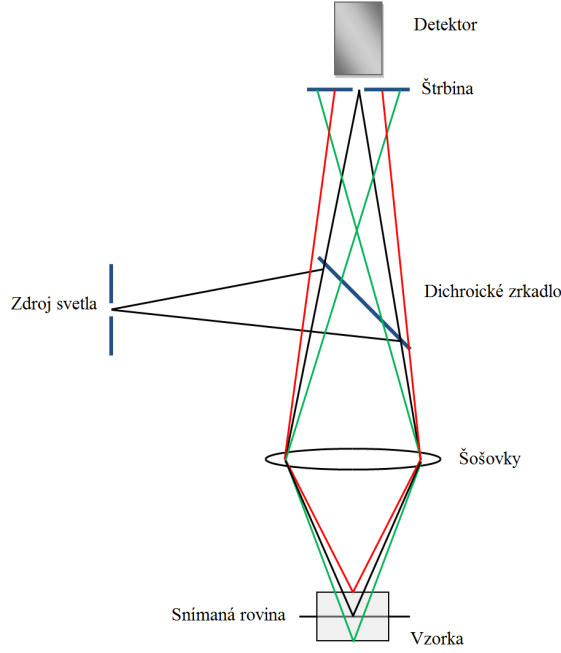
Konfokálny mikroskop patrí taktiež do oblasti optickej mikroskopie. Tento druh mikroskopu využíva tzv. konfokálny jav – pred detektorom je maličká štrbina, vďaka ktorej prejde cez túto štrbinu len zaostrená informácia a svetlo, ktoré prichádza z nezaostrených častí, prípadne iné nežiadúce svetlo, cez štrbinu neprejde. Mikroskop má tak vďaka tejto štrbine vyššiu rozlišovaciu schopnosť, no za cenu nižšej úrovne signálu, takže väčšinou sú vyžadované dlhšie doby expozície. Zloženie konfokálneho mikroskopu je znázornené na obrázku 1.3<sup>4</sup>.

Takýmto spôsobom samozrejme nasnímame len jeden bod objektu. Podľa spôsobu snímania celého obrazu rozlišujeme dva základné druhy konfokálnych mikroskopov: laserový skenovací mikroskop a mikroskop založený na princípe Nipkowho disku. Laserový skenovací mikroskop postupne sníma obraz bod po bode v jednotlivých osiach a celý obraz teda sníma relatívne pomaly. Mikroskop založený na princípe Nipkowho disku je v tomto zmysle oveľa rýchlejší – Nipkowov disk obsahuje tisícky malých (desiatky  $\mu\text{m}$ ) dier, oddelených veľkými (stovky  $\mu\text{m}$ ) medzerami. Nipkowov disk rotuje a postupne tak sníma celý obraz niekoľkonásobne rýchlejšie, ako klasický skenovací mikroskop.

### 1.4 Fourierova transformácia

Fourierova transformácia je reverzibilná matematická transformácia, ktorá prevedie obraz z priestorovej do frekvenčnej domény. To znamená, že zatiaľ čo vstupný obrázok v priestorovej doméne má v jednotlivých bodoch

4. Obrázok bol prevzatý z <http://www.tcd.ie/Physics/photonics/research/plasmon.php>.



Obr. 1.3: Zloženie konfokálneho mikroskopu.

uloženú informáciu o intenzite daného bodu v priestore, obrázok vo frekvenčnej doméne má v jednotlivých bodoch uloženú informáciu o sile rôznych frekvencií. Fourierova transformácia sa v trojrozmernom prípade počíta takto:

$$F(\omega_x, \omega_y, \omega_z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y, z) e^{-2\pi i(x\omega_x + y\omega_y + z\omega_z)} dx dy dz \quad (1.2)$$

kde  $f(x, y, z)$  je funkcia v priestorovej doméne a  $F(\omega_x, \omega_y, \omega_z)$  je funkcia vo frekvenčnej doméne. Inverzná Fourierova transformácia je v trojrozmernom prípade vyjadrená takto:

$$f(x, y, z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\omega_x, \omega_y, \omega_z) e^{2\pi i(x\omega_x + y\omega_y + z\omega_z)} d\omega_x d\omega_y d\omega_z \quad (1.3)$$

V tejto práci budeme používať hlavne diskretnú Fourierovu transformáciu, ktorá má v trojrozmernom prípade tvar:

$$\hat{I}(\omega_x, \omega_y, \omega_z) = \sum_{x=0}^{I_x-1} \sum_{y=0}^{I_y-1} \sum_{z=0}^{I_z-1} I(x, y, z) e^{-2\pi i\left(\frac{\omega_x x}{I_x} + \frac{\omega_y y}{I_y} + \frac{\omega_z z}{I_z}\right)} \quad (1.4)$$

kde  $I$  je 3D obraz veľkosti  $I_x, I_y, I_z$ . Inverzná diskretná Fourierova transformácia v trojrozmernom prípade má tvar:

$$I(x, y, z) = \frac{1}{I_x I_y I_z} \sum_{\omega_x=0}^{I_x-1} \sum_{\omega_y=0}^{I_y-1} \sum_{\omega_z=0}^{I_z-1} \hat{I}(\omega_x, \omega_y, \omega_z) e^{2\pi i \left( \frac{\omega_x x}{I_x} + \frac{\omega_y y}{I_y} + \frac{\omega_z z}{I_z} \right)} \quad (1.5)$$

Fourierova transformácia počítaná podľa (1.4) má časovú zložitosť  $O(I_x^2 I_y^2 I_z^2)$ . Fourierovu transformáciu je ale možné vypočítať aj rýchlejšie a to použitím tzv. rýchlej Fourierovej transformácie (Fast Fourier Transform – FFT). V tomto prípade sa časová zložitosť znižuje na  $O(I_x I_y I_z \cdot \log_2(I_x I_y I_z))$ .

### 1.5 Konvolúcia

Konvolúcia je matematická operácia na dvoch funkciách  $f(x, y, z)$  a  $g(k, l, m)$ , vyjadrená týmto vzťahom:

$$(f \otimes g)(x, y, z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y, z) g(k - x, l - y, m - z) dx dy dz \quad (1.6)$$

Vzorec môže byť chápaný jednoducho ako vážený priemer jednotlivých hodnôt funkcie  $f$  váhami funkcie  $g$  (zrkadlovo prevrátenými). Funkciu  $g$  nazývame konvolučné jadro, resp. kernel. Konvolúcia je veľmi častá operácia používaná v oblasti spracovania obrazu. Diskretná konvolúcia trojrozmerných obrazov  $I$  veľkosti  $I_x, I_y, I_z$  a  $P$  veľkosti  $P_x, P_y, P_z$  je daná takto:

$$(I \otimes P)(x, y, z) = \sum_{x=0}^{I_x-1} \sum_{y=0}^{I_y-1} \sum_{z=0}^{I_z-1} I(x, y, z) P(k - x, l - y, m - z) \quad (1.7)$$

Z vyššie uvedeného vzorca vidíme, že časová zložitosť výpočtu konvolúcie je  $O(I_x I_y I_z \cdot P_x P_y P_z)$ .

Existuje niekoľko spôsobov ako znížiť časovú zložitosť konvolúcie. V prípade, že konvolučné jadro je separabilné, je možné konvolúciu vypočítať postupne v jednotlivých osiach a časová zložitosť sa tak znižuje na  $O(P_x \cdot I_x I_y I_z + P_y \cdot I_x I_y I_z + P_z \cdot I_x I_y I_z)$ , čo je značná redukcia. Konvolučné jadro je separabilné v prípade, že  $g(x, y, z) = g_1(x) \otimes g_2(y) \otimes g_3(z)$  [4]. Medzi najznámejšie (umelé) separabilné konvolučné jadrá patrí napr. Gaussian<sup>5</sup>

5. Gaussian je filter, ktorý sa používa na rozmazanie obrázku.

alebo Sobel<sup>6</sup>. V obecnom prípade a hlavne vo fluorescenčnej alebo konfokálnej mikroskopii sa so separabilnými konvolučnými jadrami veľmi nestretávame.

Inou efektívnou možnosťou ako zlepšiť časovú zložitosť konvolúcie v obecnom prípade je počítanie konvolúcie podľa konvolučného teorému. Konvolučný teorém hovorí, že Fourierova transformácia dvoch konvoluovaných funkcií je ekvivalentná súčinu (bod po bode) Fourierových transformácií jednotlivých funkcií a naopak:

$$FT(f \otimes g) = FT(f).FT(g) \quad (1.8)$$

$$FT(f.g) = FT(f) \otimes FT(g) \quad (1.9)$$

kde  $FT(f)$  vyjadruje Fourierovu transformáciu funkcie  $f$ . Samotnú konvolúciu teda môžeme pomocou konvolučného teorému vyjadriť takto:

$$f \otimes g = IFT(FT(f).FT(g)) \quad (1.10)$$

kde  $IFT(f)$  vyjadruje inverznú Fourierovu transformáciu funkcie  $f$ . Použitím rýchlej Fourierovej transformácie spolu s využitím konvolučného teorému môžeme redukovať časovú zložitosť konvolúcie na  $O(S_x S_y S_z \cdot \log_2(S_x S_y S_z))$  kde  $S_x = I_x + P_x$ ,  $S_y = I_y + P_y$  a  $S_z = I_z + P_z$ . Takto počítaná konvolúcia býva označovaná aj ako rýchla konvolúcia [5] a je to obecné najvhodnejší nástroj na počítanie konvolúcie v prípade, že použité obrázky, resp. signály sú dostatočne veľké a konvolučné jadro nemá žiadne špeciálne vlastnosti.

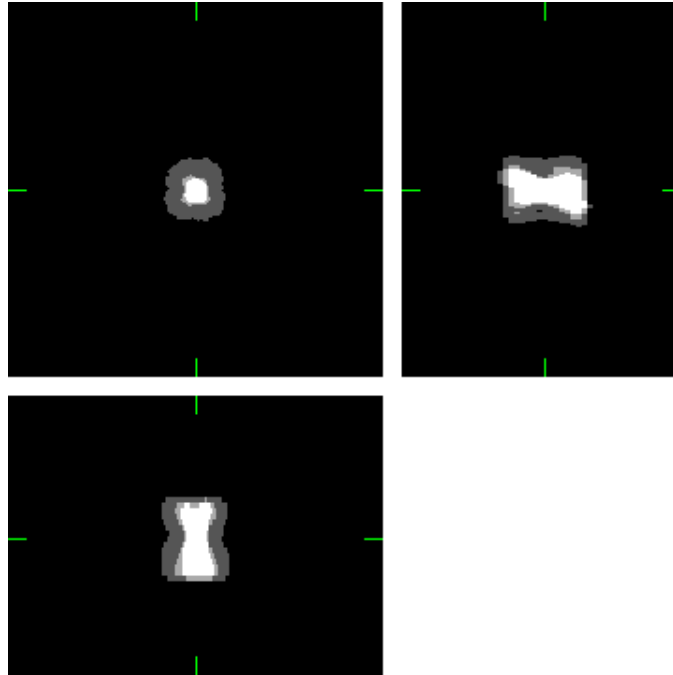
Poslednou efektívnou možnosťou počítania konvolúcie sú rekurzívne filtry. Ako už názov hovorí, takéto filtry sú definované rekurzívnymi vzorcami. Transformácia požadovaného konvolučného jadra do tejto rekurzívnej formy obecné nie je jednoduchá, môže byť nestabilná alebo nepresná. Akýkoľvek algoritmus v rekurzívnej forme sa taktiež problematcky implementuje na GPU, resp. implementácia je neefektívna a zle využíva zdroje GPU. Z týchto dôvodov v práci rekurzívne filtry nie sú využité.

## 1.6 Rozptylová funkcia

Rozptylová funkcia (Point Spread Function – PSF) je impulzná odozva optického systému na ideálny bodový zdroj svetla. Táto funkcia určuje, ako sa jeden bod snímaného obrazu zobrazuje v mikroskope. Prechod obrazu optic-

6. Sobelov filter je filter, ktorý sa používa na detekciu hrán. Aproximuje prvú deriváciu obrázku.

kou sústavou modelujeme pomocou tejto funkcie ako konvolúciu snímá-  
ného objektu s rozptylovou funkciou použitého mikroskopu. Príklad rozp-  
tylovej funkcie konfokálneho mikroskopu je na obrázku 1.4.



Obr. 1.4: Rozptylová funkcia konfokálneho mikroskopu

Môžeme vlastne povedať, že prechod optickou sústavou snímajú ob-  
jekt rozmazáva. Rozptylovú funkciu môžeme buď zmerať nasnímaním ne-  
jakého objektu, ktorý sa čo najviac podobá na ideálny bodový zdroj svetla,  
alebo spočítať teoreticky.

Pokiaľ sa rozptylová funkcia nemení v závislosti na súradniciach  $x$ ,  $y$ ,  
 $z$ , tak hovoríme, že je *pozične nezávislá*. V opačnom prípade hovoríme, že  
rozptylová funkcia je *pozične závislá*. V tejto práci budeme pre jednoduchosť  
predpokladať, že rozptylová funkcia je pozične nezávislá.

## 1.7 Šum

Šum je náhodná a nežiadúca zmena v hodnotách intenzity jednotlivých  
bodov nasnímaného obrazu. Každý reálne nasnímaný obrázok (tj. nevyt-  
vorený umelo v počítači) obsahuje šum. Poznáme niekoľko druhov šumu,  
z ktorých každý ma iný pôvod:

**Vyčítací (aditívny) šum (Readout noise)** – Tento typ šumu vzniká amplifikáciou elektronického signálu a konverziou analógového na digitálny signál. Závisí najviac na použitej elektronike, rýchlosti snímania (čím rýchlejšie snímanie, tým vyšší šum) a teplote senzoru (čím vyššia teplota, tým vyšší šum). Naopak, nezávisí na intenzite snímaného obrazu. Samotný šum môže mať rôzne distribúcie, v závislosti od toho, aké zariadenie používame. Veľmi častou distribúciou je Gaussova distribúcia, no vyskytuje sa aj Rayleighova distribúcia (v prípade radaru), či exponenciálna distribúcia (laserové snímanie).

**Výstrelkový fotónový šum (Photon shot noise)** – Fotónový šum je šum, ktorý je spôsobený fluktuáciou fotónov, ktorá plynie z kvantovej povahy týchto častíc [4]. Tento šum závisí na intenzite signálu (čím nižší signál, tým vyšší šum) a dĺžke expozície (čím dlhšia expozícia, tým nižší šum). Tento šum má Poissonovu distribúciu a pre účely tejto práce je najdôležitejší, pretože prevažuje v obrázku vtedy, ak je pri snímaní nedostatok svetla, čo ako už vieme je presne prípad fluorescenčnej alebo konfokálnej mikroskopie.

**Temný prúd (Dark current)** – Temný prúd je malý elektrický prúd, ktorý prúdi senzorom, aj keď žiadne fotóny nevstupujú do zariadenia. Tento prúd generuje náboj, ktorý sa vo výslednom obraze prejavuje ako šum. Tento šum je závislý na teplote senzoru (čím vyššia teplota, tým vyšší šum). Spolu s temným šumom sa vyskytuje aj jeho pridružený výstrelkový šum, ktorý má Poissonovu distribúciu.

## 1.8 CUDA

CUDA (Compute Unified Device Architecture) je paralelná výpočtová platforma a programovací model vytvorený firmou NVIDIA. CUDA umožňuje relatívne jednoducho a efektívne využiť zdroje grafickej karty na obecné výpočty – GPGPU<sup>7</sup>.

Samotný výpočet na grafickej karte je v CUDA realizovaný pomocou špeciálnych funkcií – *kernelov*. Kernel je funkcia, ktorá bude spustená na GPU paralelne v predom špecifikovanom počte vlákien. O vhodné rozdelenie práce medzi jednotlivými vláknami sa musí starať programátor, rovnako ako aj o prípadnú synchronizáciu vlákien. Kernely môžu pracovať s niekoľkými pamäťovými priestormi, ktoré sa líšia v rýchlosti prístupu a kapacite.

7. GPGPU – General Purpose GPU

CUDA podporuje aj systémy s viacerými grafickými kartami. Jednotlivé grafické karty tak môžu byť využívané na jeden výpočet. Akúkoľvek komunikáciu medzi nimi musí explicitne zabezpečiť programátor. Komunikácia medzi dvoma grafickými kartami je najrýchlejšia, ak sú zapojené na jednom PCI-E prepínači [9]. V prípade, že to tak nie je, musí sa prenos dát urobiť cez CPU pamäť.

## Kapitola 2

### Dekonvolúcia

Problém dekonvolúcie môžeme vo frekvenčnej doméne chápať nasledovným spôsobom [1]:

$$\hat{I}(x, y, z) = \hat{O}(x, y, z)\hat{P}(x, y, z) + \hat{N}(x, y, z) \quad (2.1)$$

kde  $P$  predstavuje rozptylovú funkciu optickej sústavy,  $N$  je aditívny šum,  $O$  je obraz snímaného objektu pred prechodom optickou sústavou a  $I$  je obraz snímaného objektu po prechode optickou sústavou. Cieľom dekonvolúcie je čo najpresnejšie určiť obraz  $O$ , ak poznáme obraz  $I$ . V prípade, že poznáme aj rozptylovú funkciu  $P$ , tak hovoríme o neslepej (non-blind) dekonvolúcii. Pokiaľ rozptylovú funkciu nepoznáme, potom hovoríme o slepej (blind) dekonvolúcii. V tejto práci sa zaoberáme metódami neslepej dekonvolúcie.

Intuitívne riešenie problému dekonvolúcie je možné získať nasledujúcou úvahou. Keďže konvolúcia je podľa konvolučného teorému vlastne násobenie vo frekvenčnej doméne, môže sa zdať, že dekonvolúciu je možné urobiť jednoduchým delením vo frekvenčnej doméne. Takýmto spôsobom ale nie je možné získať presne pôvodný obraz, pretože konvolúcia niektoré frekvencie z obrazu nenávratne odstraňuje, čo je obecný problém v dekonvolúcii. Navyše tento postup zlyháva pri prítomnosti akéhokoľvek šumu [1]. V praxi je problém dekonvolúcie tzv. *ill-posed problem*, čo znamená, že neexistuje unikátne a stabilné riešenie [1].

#### 2.1 Wienerova dekonvolúcia

Wienerova dekonvolúcia patrí do kategórie neslepých neiteratívnych metód dekonvolúcie. Počíta s prítomnosťou aditívneho šumu v obraze a bola navrhnutá tak, aby minimalizovala strednú kvadratickú chybu  $E$  medzi



vstupným a výstupným obrazom:

$$E = \sum_x \sum_y \sum_z e^2(x, y, z) \quad (2.2)$$

$$e(x, y, z) = I(x, y, z) - O(x, y, z) \quad (2.3)$$

kde  $I(x, y, z)$  je vstupný obraz a  $O(x, y, z)$  je výstupný obraz. Samotná Wienerova dekonvolúcia sa vo frekvenčnej doméne počíta takto:

$$\hat{O}(x, y, z) = \frac{\hat{P}^T(x, y, z)\hat{I}(x, y, z)}{|\hat{P}(x, y, z)|^2 + \frac{S_N(x, y, z)}{S_O(x, y, z)}} \quad (2.4)$$

kde  $S_N$  a  $S_O$  sú spektrálne hustoty (power spektra) šumu a obrazu a  $P^T$  je transpozícia rozptylovej funkcie, ktorá sa v tomto kontexte počíta takto<sup>1</sup>:

$$P^T(x, y, z) = P(P_x - x, P_y - y, P_z - z) \quad (2.5)$$

Pretože Wienerova dekonvolúcia je neiteratívna metóda, je veľmi rýchla, no má niekoľko vážnych nedostatkov. Najväčším problémom je tvorba tzv. ringing artefacts<sup>2</sup>. Ďalším problémom je nutnosť znalosti spektrálnej hustoty šumu, ktorý je prítomný v obraze. Toto je obecné problematické určiť. Napriek týmto nedostatkom je Wienerova dekonvolúcia použiteľná metóda v oblasti rekonštrukcie obrazu.

## 2.2 EM-MLE

Metóda EM-MLE je kombináciou dvoch štatistických metód – Expectation Maximization a Maximum Likelihood Estimation. Jedná sa o obľúbenú a často používanú dekonvolučnú metódu v oblasti mikroskopie. Metóda predpokladá prítomnosť šumu s Poissonovou distribúciou v obraze. Patrí do kategórie neslepých iteratívnych metód dekonvolúcie, vďaka čomu môžeme zaviesť nelineárne obmedzenia, ako napríklad nezápornosť výsledku. Počíta sa nasledujúcim spôsobom:

$$O^{n+1}(x, y, z) = P_C \left( O^n(x, y, z) \left( \frac{I(x, y, z)}{(P \otimes O^n)(x, y, z)} \otimes P^T(x, y, z) \right) \right) \quad (2.6)$$

1. Predpokladáme indexovanie od nuly a cyklickosť signálu, tj. ak  $x = 0$ , potom  $P_x - x = 0$ .  
2. Ringing artefacts sú artefakty, ktoré sa objavujú na ostrých prechodoch v obrázku. Vizualne vyzerajú ako „duchovia“ blízko hrán.

kde  $P_C$  je operátor, ktorý oreže negatívne hodnoty svojich argumentov na nulu:

$$P_C(O(x, y, z)) = \begin{cases} O(x, y, z) & \text{ak } O(x, y, z) \geq 0 \\ 0 & \text{inak} \end{cases} \quad (2.7)$$

Metóda EM-MLE je známa aj ako Richardson-Lucy metóda.

### 2.3 ICTM

Metóda ICTM (Iterative Constrained Tikhonov Miller) je neslepá iteratívna metóda, ktorá predpokladá prítomnosť aditívneho Gaussovského šumu v obraze. Metóda sa snaží minimalizovať nasledujúci funkcionál:

$$\Phi(O(x, y, z)) = \frac{1}{2} (\|(P \otimes O)(x, y, z) - I(x, y, z)\|^2 + \lambda \|(C \otimes O)(x, y, z)\|^2) \quad (2.8)$$

kde  $\| \cdot \|$  je Euklidovská norma. Regularizačný parameter  $\lambda$  predstavuje kompromis medzi dôležitosťou prvého vzťahu, reprezentovaného  $\|P \otimes O(x, y, z) - I(x, y, z)\|^2$  a vyhladzovacieho vzťahu  $\|C \otimes O(x, y, z)\|^2$ .  $C$  je regularizačná matica, ktorá zvyrazňuje časti  $O$ , ktoré je potrebné penalizovať. Populárna voľba  $C$  je Laplacianov filter<sup>3</sup>.

Podobne ako v prípade metódy EM-MLE, aj tu môžeme zaviesť nelineárne obmedzenie nezápornosti výsledku dekonvolúcie. Minimalizáciu robíme iteratívne pomocou algoritmu konjugovaných gradientov [6]. Pre jednoduchosť budú odteraz vypustené parametre funkcií  $(x, y, z)$ . Odhad obrázku  $O^{n+1}$  určíme takto:

$$O^{n+1} = P_C(O^n + \alpha^n D^n) \quad (2.9)$$

kde smer hľadania  $D^n$  je vypočítaný pomocou metódy konjugovaných gradientov takto:

$$D^n = R^n + \frac{\|R^n\|^2}{\|R^{n-1}\|^2} \cdot D^{n-1} \quad (2.10)$$

Gradient  $R$  v  $n$ -tej iterácii je určený takto:

$$R^n = O^n \otimes A - B \quad (2.11)$$

kde

$$A = P^T \otimes P + \lambda C^T \otimes C \quad (2.12)$$

$$B = P^T \otimes I \quad (2.13)$$

3. Laplacianov filter je populárnym filtrom na detekciu hrán v oblasti spracovania obrazu. Aproximuje druhú deriváciu obrázku.

Veľkosť kroku v každej iterácii  $\alpha$  je dôležitá hodnota, ktorá ovplyvňuje rýchlosť konvergenzie metódy. Určí sa takto:

$$\alpha^n = -\frac{\text{dot}(D^n T(O^n), R^n)}{\text{dot}(D^n T(O^n), A \otimes D^n T(O^n))} \quad (2.14)$$

kde  $\text{dot}$  je operátor podobný skalárnemu súčinu:

$$\text{dot}(M, N) = \sum_x \sum_y \sum_z M(x, y, z) \cdot N(x, y, z) \quad (2.15)$$

a

$$T(O(x, y, z)) = \begin{cases} 1 & \text{ak } O(x, y, z) \geq 0 \\ 0 & \text{inak} \end{cases} \quad (2.16)$$

Odvodenie tohto vzťahu pre  $\alpha$  je možné nájsť v [7].

## Kapitola 3

### Dekonvolúcia na GPU

V tejto kapitole si uvedieme pseudokódy jednotlivých dekonvolučných metód počítaných na GPU. Ešte predtým sa ale pozrieme bližšie na rýchlu konvolúciu a ukážeme si, aké kroky je potrebné urobiť predtým, ako budeme rýchlu konvolúciu počítat'.

#### 3.1 Príprava obrázkov pre rýchlu konvolúciu

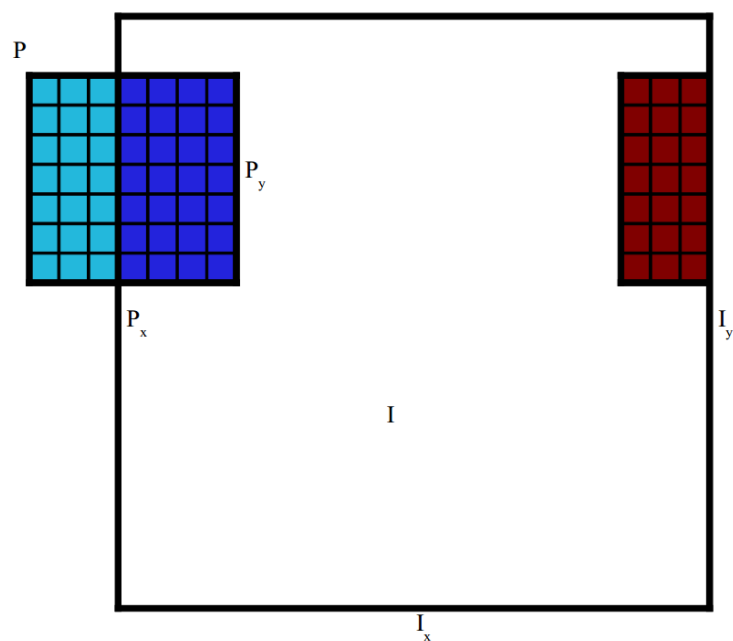
V sekcii 1.5 sme si vysvetlili výhody výpočtu konvolúcie pomocou konvolučného teorému s použitím rýchlej Fourierovej transformácie. To nám umožňuje počítat' konvolúciu aj pre relatívne veľké obrázky v dohľadnom čase. Táto kapitola sa hlbšie venuje rýchlej konvolúcii a jej implementačným problémom, ktoré je potrebné poznať a vyriešiť.

Prvým problémom, ktorý treba vyriešiť je to, že veľkosť konvolvovaného obrázku a kernelu musí byť rovnaká. To vyplýva z (1.9). Druhým problémom je, že aj v prípade rovnakej veľkosti obrázku a kernelu výsledkom nebude konvolúcia definovaná podľa (1.7), ale tzv. cyklická konvolúcia [5], pretože Fourierova transformácia je periodická [8]. Vstupný obrázok sa teda po prechode Fourierovou transformáciou správa ako periodický signál, čo má za následok nesprávne hodnoty pri okrajoch obrázku. Prehľadne je táto situácia znázornená v priestorovej doméne na obrázku 3.1.

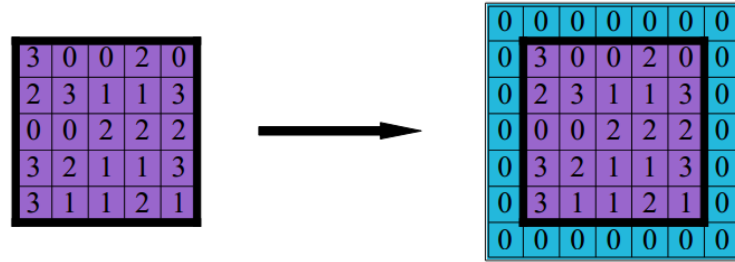
Jednoduchým riešením oboch popisovaných problémov je tzv. *padding*. Padding je doplnenie obrázku nulami istým spôsobom tak, aby obidva obrázky mali rovnakú veľkosť. To vyrieši prvý problém. Druhý problém vyriešime správnou veľkosťou paddingu, ktorá musí byť minimálne  $S_x$ ,  $S_y$  a  $S_z$ , ktoré sú dané ako:

$$S_x = I_x + P_x - 1, \quad S_y = I_y + P_y - 1, \quad S_z = I_z + P_z - 1 \quad (3.1)$$

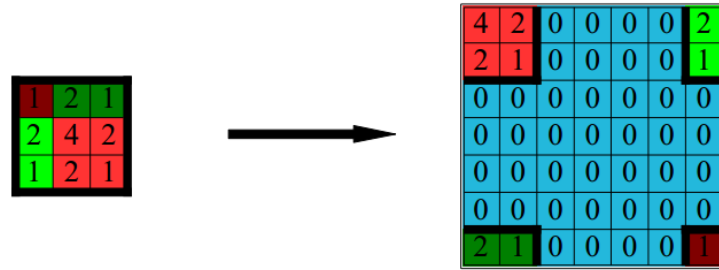
Samotných spôsobov paddingu, teda ako a kde budú do obrázkov doplnené nuly, existuje niekoľko. V tejto práci bol zvolený spôsob znázornený na obrázku 3.2 a 3.3 a to z toho dôvodu, že obrázok ostáva v takej istej relatívnej pozícii voči paddingu aj po konvolúcii. To sa bude veľmi hodiť pri



Obr. 3.1: Ukážka nesprávneho počítania cyklickej konvolúcie pri okrajoch obrázka. Ak sa kernel cyklickej konvolúcie priloží na okraj obrázka, tak ako chýbajúce hodnoty (svetlo modrá časť) budú použité hodnoty z druhej strany obrázka (tmavo červená časť) – obrázok je chápaný ako periodický (torus). Pretože v skutočnosti obrázok periodický nie je, svetlo modrá časť by mala obsahovať nuly.



Obr. 3.2: Ukážka paddingu pre vstupný obrázok. Padding je pridaný rovnomerne po stranách tak, aby obrázok ostal v strede. Je znázornený svetlo modrou farbou.



Obr. 3.3: Ukážka paddingu pre konvolučné jadro, ktoré je cyklicky posunuté tak, aby centrálny element bol v bode  $[0, 0]$ .

výpočte iteratívnych dekonvolučných metód, ktoré používajú konvolúciu niekoľkokrát po sebe. Nebudeme tak musieť robiť padding alebo kopírovať dáta po každej iterácii, ale bude stačiť nastaviť padding na nuly pred každou konvolúciou. Znázornené je to na obrázku 3.4.

Posledná vec, ktorú je treba vyriešiť, je skutočnosť, že výstupom konvolúcie podľa (1.6) je obrázok veľkosti  $S_x, S_y, S_z$ . Toto je obecné nežiadúca vlastnosť konvolúcie, pretože keď napríklad chceme rozmazať obrázok Gaussovým filtrom, tak by bolo vhodné dostať na výstupe obrázok rovnako veľký ako ten vstupný, nie väčší. To sa v praxi rieši jednoducho tak, že sa prebytočná informácia zahodí. V našom prípade to znamená, že sa padding buď kompletne zruší, resp. oreže (v prípade, že už ďalej nebudeme počítat a chceme obrázok odoslať na výstup) alebo sa jednoducho opätovne nastaví na nuly (v prípade, že ešte budeme ďalej počítat – obr. 3.4).

|   |    |    |    |    |    |    |
|---|----|----|----|----|----|----|
| 3 | 6  | 3  | 2  | 4  | 2  | 0  |
| 8 | 19 | 15 | 10 | 14 | 11 | 3  |
| 7 | 20 | 23 | 20 | 24 | 22 | 8  |
| 5 | 15 | 21 | 23 | 28 | 26 | 10 |
| 9 | 23 | 24 | 21 | 26 | 24 | 9  |
| 9 | 22 | 20 | 15 | 18 | 15 | 5  |
| 3 | 7  | 6  | 5  | 6  | 4  | 1  |

Obr. 3.4: Ukážka výsledku konvolúcie obrázkov 3.2 a 3.3. Všimnime si, že v prípade, že chceme urobiť ďalšiu konvolúciu, stačí nám nastaviť padding na nuly a nemusíme robiť žiadne dátové presuny.

### 3.2 Pseudokódy dekonvolučných metód

Nadefinujme si najprv niekoľko funkcií, ktoré budeme používať:

- $image\_pad(image, x, y, z)$  – vráti obrázok  $image$  s paddingom urobeným podľa obrázku 7a tak, že výsledná veľkosť obrázka  $image$  je  $x, y, z$ .
- $kernel\_pad(kernel, x, y, z)$  – vráti  $kernel$  s paddingom urobeným podľa obrázku 7b tak, že výsledná veľkosť kernelu  $kernel$  je  $x, y, z$ .
- $cut\_pad(image, x, y, z)$  – odstráni padding z obrázku  $image$  tak, že výsledná veľkosť obrázku  $image$  je  $x, y, z$ . Funkcia  $cut\_pad$  predstavuje inverznú funkciu voči funkcii  $image\_pad$ .
- $fft(image)$  – urobí rýchlu Fourierovu transformáciu obrázku  $image$  na GPU.
- $ifft(image)$  – urobí inverznú Fourierovu transformáciu obrázku  $image$  na GPU.
- $transpose(image)$  – urobí transpozíciu obrázku podľa (2.5).

#### 3.2.1 Wienerova dekonvolúcia

Proces počítania Wienerovej dekonvolúcie je veľmi podobný procesu počítania rýchlej konvolúcie s tým rozdielom, že vo frekvenenej doméne sa robí iný výpočet. Pseudokód tejto metódy je zobrazený v algoritme 1.  $I$  je vstupný obrázok,  $P$  je PSF mikroskopu a  $e$  je pomer spektrálnej hustoty šumu a obrazu.

**Algoritmus 1** Wienerova dekonvolúcia na GPU

---

```

1: function WIENER( $I, P, e$ )
2:    $(S_x, S_y, S_z) = (I_x + P_x - 1, I_y + P_y - 1, I_z + P_z - 1)$ 
3:    $P^T \leftarrow \text{transpose}(P)$ 
4:    $I \leftarrow \text{image\_pad}(I, S_x, S_y, S_z)$ 
5:    $P \leftarrow \text{kernel\_pad}(P, S_x, S_y, S_z)$ 
6:    $P^T \leftarrow \text{kernel\_pad}(P^T, S_x, S_y, S_z)$ 
7:
8:   skopírovanie  $I, P, P^T$  do GPU pamäte
9:
10:   $\hat{I} \leftarrow \text{fft}(I)$ 
11:   $\hat{P} \leftarrow \text{fft}(P)$ 
12:   $\hat{P}^T \leftarrow \text{fft}(P^T)$ 
13:   $\hat{O} \leftarrow \frac{\hat{P}^T \cdot \hat{I}}{|\hat{P}|^2 + e}$ 
14:   $O \leftarrow \text{ifft}(\hat{O})$ 
15:
16:  skopírovanie  $O$  do CPU pamäte
17:
18:   $O \leftarrow \text{cut\_pad}(O, I_x, I_y, I_z)$ 
19:  return  $O$ 
20: end function

```

---

**3.2.2 EM-MLE**

V kapitole 3.1 sme si vysvetili výhody zvoleného spôsobu paddingu obrázkov. Vďaka tomu sme schopní počítať konvolúciu opakovane, bez akéhokoľvek prenosu dát. Musíme jedine po každej konvolúcii nastaviť padding späť na nuly, keďže počítanie konvolúcie vytvára nadbytočné dáta, ktoré v tomto prípade nepotrebujeme. Na tento účel použijeme funkciu *set\_zero\_pad*:

- *set\_zero\_pad(image)* – nastaví padding obrázku *image* na nuly.

Pseudokód metódy EM-MLE je znázornený v algoritme 2.  $I$  je opäť vstupný obrázok,  $P$  je PSF mikroskopu a hodnota *num\_iterations* je počet iterácií algoritmu.



**Algoritmus 2** EM-MLE dekonvolúcia na GPU

---

```

1: function EM_MLE( $I, P, num\_iterations$ )
2:    $(S_x, S_y, S_z) = (I_x + P_x - 1, I_y + P_y - 1, I_z + P_z - 1)$ 
3:    $P^T \leftarrow transpose(P)$ 
4:    $I \leftarrow image\_pad(I, S_x, S_y, S_z)$ 
5:    $P \leftarrow kernel\_pad(P, S_x, S_y, S_z)$ 
6:    $P^T \leftarrow kernel\_pad(P^T, S_x, S_y, S_z)$ 
7:
8:   skopírovanie  $I, P, P^T$  do GPU pamäte
9:
10:   $O \leftarrow I$ 
11:   $\hat{P} \leftarrow fft(P)$ 
12:   $\hat{P}^T \leftarrow fft(P^T)$ 
13:
14:  for  $i \leftarrow 1, num\_iterations$  do
15:     $O \leftarrow EM\_MLE\_ITER(I, O, \hat{P}, \hat{P}^T)$ 
16:  end for
17:
18:  skopírovanie  $O$  do CPU pamäte
19:
20:   $O \leftarrow cut\_pad(O, I_x, I_y, I_z)$ 
21:  return  $O$ 
22: end function

```

---

} GPU výpočet

**Algoritmus 3** Iterácia metódy EM-MLE

---

```

1: function EM_MLE_ITER( $I, O, \hat{P}, \hat{P}^T$ )
2:    $W \leftarrow ifft(fft(O) \cdot \hat{P})$ 
3:    $W \leftarrow \frac{I}{W}$ 
4:    $set\_zero\_pad(W)$ 
5:
6:    $W \leftarrow ifft(fft(W) \cdot \hat{P}^T)$ 
7:    $W \leftarrow P_C(O \cdot W)$ 
8:    $set\_zero\_pad(W)$ 
9:
10:  return  $W$ 
11: end function

```

---

### 3.2.3 ICTM

Ešte pred tým, ako uvidíme pseudokód pre metódu ICTM si opäť nadefinujeme niekoľko dodatočných funkcií:

- $gpu\_full\_convolution(image, kernel)$  – funkcia skopíruje obrázok  $image$  a kernel  $kernel$  do GPU pamäte, spočíta ich konvolúciu, výsledok skopíruje späť do CPU pamäte a vráti výsledný obrázok. Funkcia nezahadzuje padding<sup>1</sup>.
- $sum\_squared(image)$  – funkcia vráti  $\sum_x \sum_y \sum_z image(x, y, z)^2$ .

3D Laplacián, ktorý je v texte označený ako  $LAP$ , je definovaný takto (zobrazené sú jednotlivé rezy v rovine  $xy$ ):

$$LAP_{z=0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad LAP_{z=1} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -6 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad LAP_{z=2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Pseudokód metódy ICTM je zobrazený v algoritme 4.  $I$  je vstupný obrázok,  $P$  je PSF mikroskopu, hodnota  $num\_iterations$  je počet iterácií algoritmu a  $\lambda$  je regularizačný parameter metódy definovaný v podkapitole 2.3. Všimnime si tiež, že konvolučné jadro  $A$  počítané podľa (2.12) má väčšiu veľkosť, ako PSF. Z toho dôvodu je potrebný padding na riadku 2 v algoritme 4 väčší, ako v predošlých metódach.

1. Toto je jeden z prípadov, kde je nežiadúce zahadzovať padding, keďže obsahuje dôležité informácie pre správny výpočet dekonvolúcie.

**Algoritmus 4** ICTM dekonvolúcia na GPU

---

```

1: function ICTM( $I, P, num\_iterations, \lambda$ )
2:    $(S_x, S_y, S_z) = (I_x + 2P_x - 2, I_y + 2P_y - 2, I_z + 2P_z - 2)$ 
3:    $P^T \leftarrow transpose(P)$ 
4:    $PP \leftarrow gpu\_full\_convolution(P, P^T)$ 
5:    $L \leftarrow gpu\_full\_convolution(LAP, LAP^T)$ 
6:    $A \leftarrow PP + \lambda.L$ 
7:
8:    $I \leftarrow image\_pad(I, S_x, S_y, S_z)$ 
9:    $P^T \leftarrow kernel\_pad(P^T, S_x, S_y, S_z)$ 
10:   $A \leftarrow kernel\_pad(A, S_x, S_y, S_z)$ 
11:
12:  skopírovanie  $I, P^T, A$  do GPU pamäte
13:
14:   $O \leftarrow I$ 
15:   $\hat{A} \leftarrow fft(A)$ 
16:
17:   $B \leftarrow ifft(fft(O).fft(P^T))$ 
18:   $set\_zero\_pad(B)$ 
19:
20:   $D^{old} \leftarrow ifft(fft(O).\hat{A})$ 
21:   $set\_zero\_pad(D^{old})$ 
22:
23:   $\bar{R}^{old} \leftarrow sum\_squared(O)$ 
24:
25:  for  $i \leftarrow 1, num\_iterations$  do
26:     $(O, D, \bar{R}) \leftarrow ICTM\_ITER(O, \hat{A}, B, D^{old}, \bar{R}^{old})$ 
27:     $D^{old} \leftarrow D$ 
28:     $\bar{R}^{old} \leftarrow \bar{R}$ 
29:  end for
30:
31:  skopírovanie  $O$  do CPU pamäte
32:
33:   $O \leftarrow cut\_pad(O, I_x, I_y, I_z)$ 
34:  return  $O$ 
35: end function

```

---

GPU výpočet

---

**Algoritmus 5** Iterácia metódy ICTM

---

```

1: function ICTM_ITER( $O, \hat{A}, B, D^{old}, \bar{R}^{old}$ )
2:    $R \leftarrow \text{ifft}(\text{fft}(O) \cdot \hat{A}) - B$ 
3:    $\text{set\_zero\_pad}(R)$ 
4:    $\bar{R} \leftarrow \text{sum\_squared}(R)$ 
5:
6:    $D \leftarrow R + \frac{\bar{R}}{\bar{R}^{old}} \cdot D^{old}$ 
7:    $DT \leftarrow D \cdot T(O)$ 
8:
9:    $\alpha_1 \leftarrow \text{dot}(DT, R)$ 
10:   $\alpha_2 \leftarrow \text{dot}(DT, \text{ifft}(\text{fft}(DT) \cdot \hat{A}))$ 
11:   $\alpha \leftarrow -\frac{\alpha_1}{\alpha_2}$ 
12:
13:   $O \leftarrow P_C(O + \alpha D)$ 
14:
15:  return ( $O, D, \bar{R}$ )
16: end function

```

---

## Kapitola 4

# Dekonvolúcia na systémoch s viacerými grafickými kartami

### 4.1 Motivácia

V tejto kapitole preskúmame možnosti implementácie dekonvolučných metód na systémoch s viacerými grafickými kartami. Použitie viacerých grafických kariet si kladie za cieľ jednak zlepšenie časovej zložitosti algoritmov, a taktiež aj zlepšenie pamäťovej náročnosti, čo je v prípade GPU výpočtov podstatná vec, keďže na GPU nemáme väčšinou k dispozícii také veľké množstvo pamäte ako na CPU<sup>1</sup>.

### 4.2 Možnosti

Keďže každé GPU má samostatnú pamäť a neexistuje nijaká „centrálna“ pamäť, v ktorej by mohli byť obrázky uložené a ku ktorej by mohli jednotlivé GPU pristupovať, je potrebné dáta medzi jednotlivé GPU rozdeliť. Ideálny spôsob by bol, ak by každé GPU mohlo operovať na svojej časti dát, nezávisle na ostatných GPU, a po skončení výpočtu by sa výsledky jednoducho skopírovali späť do CPU pamäte, kde by boli spojené do výsledného obrázku. V skutočnosti to ale tak jednoduché nie je. Ak sa pozrieme na pseudokódy 1, 2 a 4 jednotlivých dekonvolučných metód, môžeme si všimnúť, že sa tam vyskytujú operácie, ktoré sa dajú rozdeliť do dvoch skupín v závislosti od toho, či ich vyhodnotenie závisí na ich okolí alebo nie.

V prvej skupine sú operácie, ktoré sú vykonávané bod po bode a nezávisia ani na svojom okolí, ani na celom obrázku. Sem patria napríklad operácie súčinu alebo podielu dvoch obrázkov bod po bode. Svojou charakteristikou pripomínajú SIMD<sup>2</sup> programovací model – vykonávajú jednu

---

1. Napríklad karty Tesla od firmy NVIDIA, ktoré patria medzi najlepšie grafické karty na vedecké využitie, a ktoré sú bežne používané vo vedeckých aplikáciách, majú 6 GB pamäte. Naproti tomu nie je problém zostrojiť domáci počítač s 64 GB CPU pamäte za oveľa nižšiu cenu.

2. SIMD – Single Instruction Multiple Data

jednoduchú operáciu na celom obrázku. Každá grafická karta tak môže vykonávať takéto operácie len na svojej časti dát a k ostatným dátam pristupovať nemusí. To je veľká výhoda a pokiaľ by to takto platilo pre všetky operácie v jednotlivých dekonvolučných metódach, bola by implementácia na viacerých GPU priamočiara a dobre by škalovala pri zvyšovaní počtu použitých GPU.

Na druhej strane, operácie ako počítanie funkcie *dot* (2.15), funkcia *sum\_squared* (v kapitole 3.2.3) a hlavne rýchla konvolúcia (1.10) závisia na celom obrázku a pre výpočet výsledku týchto operácií je nutné pristúpiť ku všetkým bodom obrázku. V tomto prípade už implementácia nie je tak jednoduchá ako v predošlom prípade. V nasledujúcich podkapitolách si preberieme najprv jednoduchý prípad funkcie *dot* a *sum\_squared*, a potom sa zameriame na počítanie rýchlej konvolúcie, kde je situácia zložitejšia.

### 4.3 Funkcia *dot* a funkcia *sum\_squared*

Tieto dve funkcie sú zaradené v tej istej podkapitole z toho dôvodu, že sa v podstate jedná o takmer tú istú operáciu. V prípade funkcie *dot* sa najprv vypočíta súčin dvoch obrázkov bod po bode a následne sa urobí súčet všetkých bodov novovzniknutého obrázku. Funkcia *sum\_squared* počíta súčet druhých mocnín všetkých bodov obrázku. Dalo by sa teda povedať, že

$$\text{dot}(I, I) = \text{sum\_squared}(I) \quad (4.1)$$

Tieto funkcie môžeme na viacerých GPU implementovať jednoduchým spôsobom. Každé GPU vyhodnotí výsledok danej funkcie na svojej časti dát a následne rozošle tento výsledok na všetky ostatné GPU. Ide o jediné číslo, takže spomalenie výpočtu vplyvom prenosu dát nie je v tomto prípade veľké. Každé GPU má teda výsledok funkcie na jednotlivých častiach obrázku, a pretože ide o sumy, je možné ich jednoducho spočítať a dostať tak celkový výsledok týchto funkcií.

### 4.4 Rýchla konvolúcia

Obecne existujú tri spôsoby ako rozdeliť obrázok na viacero častí a rozdeliť tak výpočet rýchlej konvolúcie na viacero grafických kariet. Ide o *decimation in time* (DIT), *decimation in frequency* (DIF) a *signal and kernel tiling* [5]. Všetky tieto tri spôsoby boli preskúmané v [5] a záverom bolo, že spôsob DIT je vo všetkých ohľadoch menej efektívny ako DIF a neexistuje žiadny dôvod, prečo by mal byť použitý namiesto DIF. Z tohto dôvodu si v nasledujúcich podkapitolách popíšeme len zvyšné dva spôsoby.

#### 4.4.1 Decimation in frequency

DIF využíva nasledujúcu skutočnosť: Ak máme obrázok  $I$  veľkosti  $I_x, I_y, I_z$ , kde  $I_z$  je párne číslo, potom môžeme vytvoriť dva menšie obrázky  $I_R$  a  $I_S$  takto:

$$\begin{aligned} I_R(x, y, z') &= I(x, y, z') + I(x, y, z' + I_z/2) \\ I_S(x, y, z') &= (I(x, y, z') - I(x, y, z' + I_z/2))e^{-\frac{2\pi iz'}{I_z}} \end{aligned} \quad (4.2)$$

kde  $i$  je komplexná jednotka a  $z' \in \{0, \dots, I_z/2 - 1\}$ . Potom Fourierove transformácie  $\hat{I}_R$  a  $\hat{I}_S$  obrázkov  $I_R$  a  $I_S$  a Fourierova transformácia  $\hat{I}$  celého obrázku  $I$  majú nasledovný vzťah:

$$\begin{aligned} \hat{I}_R(x, y, z') &= \hat{I}(x, y, 2z') \\ \hat{I}_S(x, y, z') &= \hat{I}(x, y, 2z' + 1) \end{aligned} \quad (4.3)$$

Nevýhoda tohto spôsobu je, že obrázky  $I_R$  a  $I_S$  už nie sú reálne obrázky, ale komplexné obrázky. Mnohé knižnice, ktoré počítajú rýchlu Fourierovu transformáciu majú k dispozícii špeciálne funkcie, optimalizované pre reálne dáta. Tieto funkcie využívajú známu vlastnosť Fourierovej transformácie reálnych obrázkov, ktorá je vyjadrená nasledujúcim vzťahom:

$$\hat{I}(x, y, z) = \hat{I}^*(I_x - x, I_y - y, I_z - z) \quad (4.4)$$

Znamená to, že pri výpočte Fourierovej transformácie reálneho obrázku  $\hat{I}$  potrebujeme určiť len polovicu koeficientov – zvyšné sú rovnaké. Ak sa ale rozhodneme použiť DIF, potom tieto optimalizované funkcie už použiť nemôžeme, pretože ako sme už povedali, obrázky  $I_R$  a  $I_S$  sú komplexné obrázky.

Tento problém rieši tzv. *komplexifikácia* [5], ktorá je založená na nasledujúcej myšlienke: ak máme obrázok  $I$  veľkosti  $I_x, I_y, I_z$ , kde  $I_x$  je párne číslo, potom môžeme vytvoriť komplexný obrázok  $I_C$  polovičnej veľkosti v osi  $x$  takto:

$$I_C(x', y, z) = I(2x', y, z) + iI(2x' + 1, y, z) \quad (4.5)$$

kde  $x' \in \{0, \dots, I_x/2 - 1\}$ . Fourierove transformácie  $\hat{I}$  a  $\hat{I}_C$  majú potom nasledujúci vzťah:

$$\begin{aligned} \hat{I}(x', y, z) &= \frac{1}{2} \left( \alpha_+(x', y, z) - i\alpha_-(x', y, z)e^{-\frac{2\pi ix'}{I_x}} \right) \\ \hat{I}(x' + I_x/2, y, z) &= \frac{1}{2} \left( \alpha_+(x', y, z) + i\alpha_-(x', y, z)e^{-\frac{2\pi ix'}{I_x}} \right) \end{aligned} \quad (4.6)$$

kde

$$\alpha_{\pm}(x', y, z) = \hat{I}_C(x', y, z) \pm \hat{I}_C^*(I_x/2 - x', y, z) \quad (4.7)$$

Výpočet týchto vzťahov budeme nazývať *rekombinácia*. V prípade, že komplexné dáta sú uložené v počítači tradičným spôsobom (jednotlivé body sú uložené v poli za sebou a najprv je uložená reálna časť bodu, potom komplexná), tak vyššie uvedený obrázok  $I_C$  môžeme vytvoriť jednoduchým pretypovaním datového typu poľa, v ktorom je daný obrázok  $I$  uložený. Týmto spôsobom sa vyhneme akýmkoľvek dátovým prenosom, či výpočtom [5].

Použitím komplexifikácie teda dosiahneme časovú úsporu tým, že pracujeme s obrázkami polovičnej veľkosti a vykonávame tak o polovicu menej operácií. To je podobná úspora ako v prípade použitia špeciálnych funkcií optimalizovaných pre reálne dáta.

Analyzujme časovú zložitosť výpočtu, ktorý počíta jedno GPU, množstvo pamäte, ktorú potrebujeme alokovať na jednom GPU a potrebný počet pamäťových prenosov medzi jedným GPU a CPU. Počet GPU, na ktorých bude výpočet realizovaný označme  $d$ . Veľkosť Fourierovej transformácie, ktorú počíta jedno GPU je<sup>3</sup>:

$$R = \frac{(I_x + P_x)(I_y + P_y)(I_z + P_z)}{d} \quad (4.8)$$

Časová zložitosť výpočtu, ktorý počíta jedno GPU potom je:

$$O(R \cdot \log(R)) \quad (4.9)$$

Množstvo pamäte, ktorú potrebujeme alokovať na každom GPU je:

$$4R \quad (4.10)$$

Celkový počet pamäťových prenosov medzi GPU a CPU je:

$$3R \quad (4.11)$$

#### 4.4.2 Signal and kernel tiling

Tento spôsob rozdelenia výpoču rýchlej konvolúcie je založený na rozdelení vstupného obrázku aj kernelu na niekoľko častí. Je však možné rozdeliť len obrázok a kernel ponechať nerozdelený alebo naopak. V ďalších podkapitolách si preto ukážeme najprv samostatné spôsoby *signal tiling* a *kernel tiling*.

3. Predpokladáme tu a aj v ďalšom texte, že  $I_x, I_y, I_z \gg 1$  a  $P_x, P_y, P_z \gg 1$



### Signal tiling

Signal tiling je založený na myšlienke, že rozdelíme obrázok  $I$ , ktorý chceme konvolvovať s kernelom  $P$  na  $m$  menších častí  $I_j, j \in \{1, \dots, m\}$ , vypočítame na týchto častiach samostatne konvolúciu s  $P$ , a potom časti spojíme dohromady. Aby sme zaistili, že výsledok konvolúcie bude správny, je nutné pridať ku jednotlivým častiam  $I_j$  aj nejaké dáta od susedných častí. Týmto dátam budeme hovoriť *okraje* časti  $I_j$ . Veľkosť okraja je rovná celkovej veľkosti kernelu  $P$  [5]. V prípade, že sa časť  $I_j$  nachádza v rohu, alebo na strane obrázka  $I$ , tak okraj obsahuje samé nuly (a predstavuje padding, ktorý je potrebný pre korektné vypočítanie konvolúcie – viz. kapitola 3.1). Po vypočítaní konvolúcie jednotlivých častí sa okraje zahodia, všetky časti sa spoja a vytvoria výsledok konvolúcie  $I \otimes P$ .

### Kernel tiling

Kernel tiling je veľmi podobný spôsob ako signal tiling, ale v tomto prípade delíme kernel  $P$  a nie vstupný obrázok  $I$ . Konvolúcia obrázkov  $I$  a  $P$  je vyhodnotená nasledujúcim spôsobom: kernel  $P$  je rozdelený na  $n$  častí  $P_k, k \in \{1, \dots, n\}$  bez akýchkoľvek okrajov, resp. redundantnej informácie. Následne urobíme padding obrázka  $I$  a kernelov  $P_k$  tak, aby ich výsledná veľkosť bola rovnaká. Potom vyhodnotíme jednotlivé konvolúcie  $I \otimes P_k$ . Spojenie výsledkov konvolúcie realizujeme tak, že všetky časti sčítame dohromady, pričom každá časť je ešte predtým posunutá voči prvej časti  $I_1$  o posun, ktorý odpovedá posunu jednotlivých častí  $P_k$  voči prvej časti  $P_1$ .

Výsledný obrázok ešte orežeme o padding a dostaneme tak výsledok konvolúcie  $I \otimes P$ .

### Signal and kernel tiling

Môžeme samozrejme obidva spôsoby skombinovať dohromady a rozdeliť tak aj obrázok  $I$  aj kernel  $P$ . Následne sa vypočíta konvolúcia jednotlivých častí  $I_j \otimes P_k$ , zahodia sa okraje jednotlivých častí a obrázky sa spoja dohromady rovnakým spôsobom ako v prípade kernel tilingu. Veľkosť Fourierovej transformácie, ktorú počíta jedno GPU je:

$$T = \left( \frac{I_x}{m} + \frac{P_x}{n} \right) \left( \frac{I_y}{m} + \frac{P_y}{n} \right) \left( \frac{I_z}{m} + \frac{P_z}{n} \right) \quad (4.12)$$

Časová zložitosť výpočtu, ktorý počíta jedno GPU je:

$$O(T \cdot \log(T)) \quad (4.13)$$

#### 4. DEKONVOLÚCIA NA SYSTÉMOCH S VIACERÝMI GRAFICKÝMI KARTAMI

|                             | časová<br>zložitosť                                                          | množstvo<br>pamäte    | počet<br>dátových prenosov |
|-----------------------------|------------------------------------------------------------------------------|-----------------------|----------------------------|
| DIF                         | $O\left(\frac{Q}{d} \cdot \log\left(\frac{Q}{d}\right)\right)$               | $4\frac{Q}{d}$        | $3\frac{Q}{d}$             |
| signal and kernel<br>tiling | $O\left(\frac{Q}{\sqrt{d}} \cdot \log\left(\frac{Q}{\sqrt{d}}\right)\right)$ | $2\frac{Q}{\sqrt{d}}$ | $3\frac{Q}{\sqrt{d}}$      |

Tabuľka 4.1: Porovnanie časovej zložitosti, množstva pamäte a počtu dátových prenosov spôsobu DIF a spôsobu signal and kernel tiling.

Množstvo pamäte, ktorú potrebujeme alokovať na každom GPU je:

$$2T \quad (4.14)$$

Celkový počet pamäťových prenosov medzi GPU a CPU je:

$$3T \quad (4.15)$$

Optimálny prípad nastane, ak počet častí, na ktoré je rozdelený obrázok  $I$  a kernel  $P$  je rovnaký, teda ak  $m = n$  [5]. Potrebný počet GPU je potom  $m^2$ .

#### 4.4.3 Porovnanie a výber vhodného spôsobu

Predstavili sme si dva odlišné prístupy na počítanie rýchlej konvolúcie na systémoch s viacerými grafickými kartami. Teraz tieto poznatky využijeme na výber najvhodnejšieho spôsobu pre každú dekonvolučnú metódu, ktorej sa v tejto práci venujeme. Najprv ale porovnajme jednotlivé spôsoby výpočtu rýchlej konvolúcie na viacerých grafických kartách.

Nech  $m = n = \sqrt{d}$ , kde  $d$  je opäť počet GPU, na ktorých je konvolúcia počítaná a nech  $Q = (I_x + P_x)(I_y + P_y)(I_z + P_z)$ . Tabuľka 4.1 porovnáva časové zložitosti, množstvo potrebnej pamäte a počet dátových prenosov všetkých metód. Z tabuľky vidíme, že časová zložitosť metódy DIF je aspoň  $\sqrt{d}$  krát menšia, ako časová zložitosť metódy signal and kernel tiling. Podobne je to aj s počtom pamäťových prenosov, kde počet pamäťových prenosov je presne  $\sqrt{d}$  krát menší v prípade metódy DIF. Čo sa týka množstva potrebnej pamäte, tak v tomto prípade platí, že metóda DIF potrebuje  $\frac{1}{2}\sqrt{d}$  krát menej pamäte ako metóda signal and kernel tiling<sup>4</sup>.

4. V obecnom prípade teda spôsob DIF vyžaduje menej pamäte, no pre  $d = 2$  je to naopak. Pre  $d = 4$  je potrebná pamäť rovnaká pre obidva spôsoby.

Vidíme teda, že pre výpočet rýchlej konvolúcie na viacerých grafických kartách je metóda DIF lepšia, ako metóda signal and kernel tiling. Podobný záver bol vyvodený aj v [5].

##### 4.4.4 Wienerova dekonvolúcia

Ako sme už hovorili v kapitole 3.2.1, Wienerova dekonvolúcia je z technického hľadiska veľmi podobná rýchlej konvolúcii – líši sa len rozdielnym výpočtom vo frekvenčnej doméne. Tento výpočet je však vlastnosťami zhodný s jednoduchým súčinom obrázkov, ako to je v prípade rýchlej konvolúcie – opäť totiž robíme operácie bod po bode.

Z tohto dôvodu je najvhodnejšia metóda pre počítanie Wienerovej dekonvolúcie na viacerých grafických kartách metóda DIF, rovnako ako to je v prípade rýchlej konvolúcie.

##### 4.4.5 EM-MLE a ICTM

V prípade týchto dvoch metód je situácia zložitejšia. Nemôžeme sa pozerať len na porovnanie v sekcii 4.4.3, pretože tam analyzujeme výpočet len jednej rýchlej konvolúcie. Metódy EM-MLE a ICTM ale potrebujú konvolúciu vyhodnotiť niekoľkokrát, pretože ide o iteratívne metódy. Navyše, tieto metódy obsahujú aj operácie v priestorovej doméne, ktoré sa striedajú s konvolúciou. Nevyhneme sa teda opakovanému prepočítavaniu obrázkov z priestorovej do frekvenčnej domény.

Keďže dátové prenosy medzi jednotlivými GPU sú veľmi drahé [9], potrebujeme zvoliť takú metódu, ktorá má počet a veľkosť týchto dátových prenosov minimálny. Metóda DIF sa na tento účel teda už nehodí, pretože vyžaduje, aby sa výsledok konvolúcie „zložil“ na nejakej výpočtovej jednotke (na jednom GPU alebo na CPU). To by vyžadovalo veľké dátové prenosy pred a po konvolúcii a v prípade, že výsledok spojíme na GPU aj dodatočnú pamäť, čím by sme sa zbavili akýchkoľvek výhod použitia viacerých GPU na výpočet dekonvolúcie. Podobne to dopadne aj s metódou kernel tiling. V tomto prípade opäť potrebujeme výsledok konvolúcie „zložiť“ z viacerých častí, čo opäť vyžaduje veľké dátové prenosy. Pri použití metódy signal tiling potrebujeme pred každou konvolúciou len vymeniť, resp. „aktualizovať“ okraje jednotlivých častí. Veľkosť týchto okrajov je daná veľkosťou konvolučného jadra a čím je jadro menšie vzhľadom k veľkosti vstupného obrázku, tým je táto metóda efektívnejšia.

Tabuľka 4.2 sumarizuje dodatočný počet dátových prenosov medzi GPU, ktoré je nutné urobiť pred každou konvolúciou. Pri metóde DIF uvažujeme

#### 4. DEKONVOLÚCIA NA SYSTÉMOCH S VIACERÝMI GRAFICKÝMI KARTAMI

|                                     | <b>dodatočné<br/>dátové prenosy</b> |
|-------------------------------------|-------------------------------------|
| <b>DIF</b>                          | $4dR$                               |
| <b>signal tiling</b>                | $mP_xP_yP_z$                        |
| <b>kernel tiling</b>                | $2(n-1)I_xI_yI_z$                   |
| <b>signal and kernel<br/>tiling</b> | $2m(n-1)I_xI_yI_z + mP_xP_yP_z$     |

Tabuľka 4.2: Dodatočné množstvo pamäte a dátových prenosov jednotlivých metód na iteratívne počítanie konvolúcie.

jeme, že jednotlivé časti sú spojené na CPU<sup>5</sup>. Čo sa týka ostatných metód, tak vzhľadom k tomu, že napríklad v CUDA 4 [9] existuje možnosť tzv. *peer-to-peer prístupu*, to znamená možnosť čítať dáta priamo z pamäte iného GPU, nie je potrebná žiadna dodatočná pamäť. Pokiaľ táto technológia nie je prístupná, je viacero možností, akými implementovať jednotlivé metódy, preto pre jednoduchosť uvažujeme, že technológia peer-to-peer prístupu je k dispozícii.

Ak vezmeme do úvahy tieto analýzy, tak vidíme, že použitie samostatnej metódy signal tiling je najlepšia voľba, pretože tento spôsob používa spomedzi ostatných spôsobov minimálny počet dátových prenosov, ktoré sú pre rýchlosť výpočtu kritické. Navyše, vzhľadom k tomu, že PSF mikroskopu je obecné výrazne menšia, ako obrázky, ktoré potrebujeme dekonvolvovať, je efektívnosť tohto spôsobu v praktických aplikáciách ešte väčšia. Z týchto dôvodov bol pre dekonvolučné metódy EM-MLE a ICTM v aplikácii, ktorá je súčasťou tejto práce, použitý práve spôsob signal tiling.

5. Prípadnú dodatočnú pamäť na CPU v tejto analýze nepovažujeme za podstatnú.

## Kapitola 5

### Popis programu

#### 5.1 Stručný popis

Aplikácia je naprogramovaná v jazyku C++ a umožňuje počítať v práci popisované dekonvolučné metódy – Wienerovu dekonvolúciu, EM-MLE dekonvolúciu a ICTM dekonvolúciu na 3D obrázkoch. Tieto 3D obrázky dokáže aplikácia načítať a uložiť vo formáte \*.ics, čo je často používaný formát na ukladanie biomedicínskych obrázkov. Na manipuláciu s týmito obrázkami bola použitá knižnica libics [10].

Program umožňuje počítať jednotlivé dekonvolučné metódy buď na CPU alebo na GPU. Na výpočet je možné využiť aj viacero GPU (pokiaľ sú prítomné v systéme), a to buď dve GPU alebo štyri GPU. Výpočet je možné urobiť v dátovom type float<sup>1</sup>, alebo double<sup>2</sup>. V prípade CPU implementácie je možné použiť aj typ long double<sup>3</sup>.

Na počítanie rýchlej Fourierovej transformácie v CPU implementácii bola využitá jedna z najpopulárnejších knižníc na tento účel – knižnica FFTW [11]. Samotné počítanie rýchlej Fourierovej transformácie pomocou tejto knižnice je schopné využiť aj viacero jadier procesoru, pokiaľ sú programu zadané odpovedajúce parametre a použitý procesor má viacero jadier. Všetky ostatné časti kódu využívajú na výpočet len jedno vlákno (a teda len jedno jadro procesoru).

GPU implementácia je vytvorená v CUDA. Na výpočet rýchlej Fourierovej transformácie v GPU implementácii bola použitá knižnica CUFFT<sup>4</sup> [12], a na počítanie niektorých operácií v metóde ICTM, knižnica CUBLAS<sup>5</sup>

---

1. 4 bajtový dátový typ pre desatinné čísla.

2. 8 bajtový dátový typ pre desatinné čísla.

3. Typ long double má obecnú rozdielnú presnosť na rôznych architektúrach. V tejto práci predpokladáme, že je to 16 bajtový dátový typ pre desatinné čísla.

4. Knižnica, ktorá dokáže počítať rýchlu Fourierovu transformáciu na GPU v CUDA. Nedokáže automaticky na výpočet použiť viacero GPU.

5. Knižnica CUBLAS je implementácia knižnice BLAS (Basic Linear Algebra Subprograms) v CUDA. Knižnica umožňuje počítať bežné operácie lineárnej algebry na GPU. Podobne ako CUFFT, ani táto knižnica nedokáže automaticky využiť na výpočet viacero GPU.

[13].

Program je dodaný aj v spustiteľnej podobe a je možné ho spustiť na unixovom systéme v príkazovom riadku príkazom:

```
./deconvolution
```

Po spustení tohto príkazu program vypíše návod na obsluhu.

## 5.2 Dodatočný padding obrázkov

V kapitole 3.1 sme stanovili potrebnú veľkosť paddingu tak, aby výsledná veľkosť obrázku, pripraveného na rýchlu konvolúciu, bola:

$$S_x = I_x + P_x - 1, \quad S_y = I_y + P_y - 1, \quad S_z = I_z + P_z - 1 \quad (5.1)$$

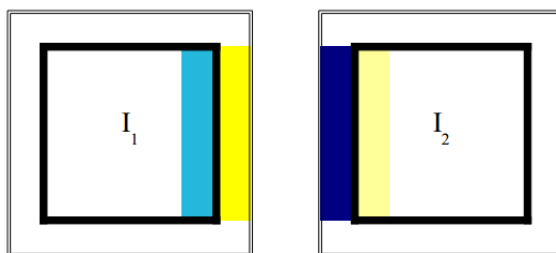
kde  $I$  a  $P$  sú konvolvované obrázky a  $I_x, I_y, I_z$  a  $P_x, P_y, P_z$  sú ich veľkosti v jednotlivých osiach. Hoci knižnice FFTW a CUFFT vedia počítať rýchlu Fourierovu transformáciu ľubovoľných rozmerov v časovej zložitosti  $O(S_x S_y S_z \log(S_x S_y S_z))$  [11, 12], pre isté rozmery je výpočet oveľa pomalší, ako pre iné. To je dané tým, že najefektívnejšie algoritmy z pohľadu celkového počtu operácií, pracujú len nad obrázkami špecifických rozmerov. V prípade týchto knižníc je použitý najefektívnejší algoritmus, ak majú obrázky v každej osi rozmery nasledujúceho tvaru:

$$S_i = 2^a 3^b 5^c 7^d \quad (5.2)$$

kde  $a, b, c, d \in \mathbb{N}$ . V praxi sa osvedčilo používať rozklad len na mocniny 2 a 3. Z tohto dôvodu je padding vstupného obrázku a kernelu zväčšený tak, aby výsledná veľkosť bola tohto tvaru. V praxi obecné platí aj to, že čím menej týchto prvočísel obsahuje veľkosť  $S_i$  vo svojom rozklade, tým rýchlejšia Fourierova transformácia býva. Z tohto dôvodu je v programe nastaviteľné, aké najväčšie prvočíslo bude veľkosť  $S_i$  vo svojom rozklade obsahovať.

## 5.3 Dodatočné buffre na GPU

Pri porovnávaní spôsobov na počítanie konvolúcie na systémoch s viacerými grafickými kartami v sekcii 4.4.5 sme povedali, že vďaka technológii peer-to-peer prístupu v CUDE [9] nepotrebujeme žiadnu ďalšiu pamäť na GPU v prípade metódy signal tiling. V práci však táto technológia nebola použitá z nasledujúcich dôvodov: technológia sa dá použiť len vtedy, ak sú



Obr. 5.1: Výmena okrajov medzi dvoma GPU. Svetlejší odtieň farby označuje tú časť, ktorá sa kopíruje (zdroj), tmavší odtieň príslušnej farby označuje miesto, na ktoré je časť skopírovaná, tj. napríklad svetlomodrá časť sa kopíruje na miesto tmavomodrej časti. Všetky prenosy sú asynchrónne.

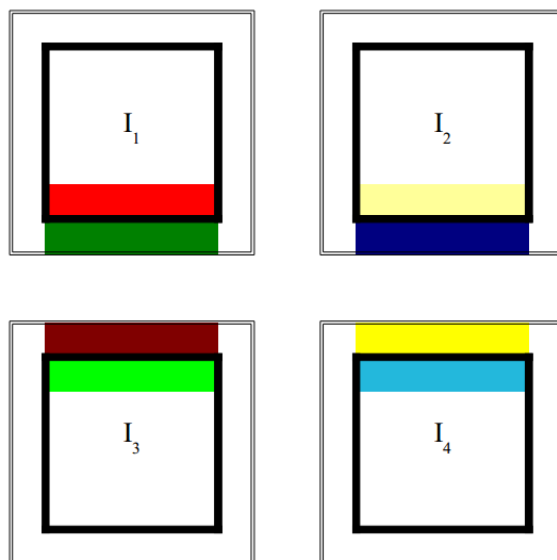
jednotlivé GPU pripojené na jednom PCI-E prepínači. Server *kudu*, na ktorom bol tento program vyvíjaný, však túto podmienku nespĺňa pre všetky GPU. Navyše, experimentálne bolo overené, že použitie tejto technológie je o niečo pomalšie, ako *peer-to-peer kópia*<sup>6</sup>.

Program preto používa dodatočnú pamäť, ktorá je použitá na výmenu dát medzi GPU. Každé GPU má dve dodatočné polia, každé veľkosti okrajov, ktoré potrebuje odoslať susedným GPU. Jedno pole slúži na extrahovanie dát z centrálného poľa a na odoslanie dát do susedného GPU, druhé pole slúži na prijímanie dát od susedného GPU.

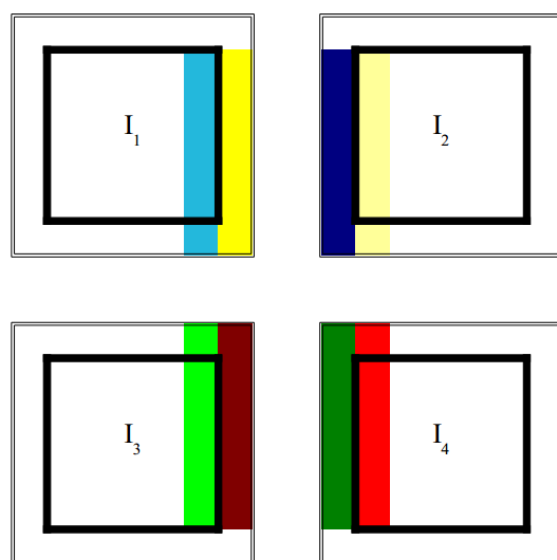
## 5.4 Výmena okrajov

Samotná výmena okrajov prebieha nasledujúcim spôsobom. V prípade použitia dvoch GPU je obrázok rozdelený na dve časti buď v ose  $x$  alebo  $y$ , v závislosti od toho, v ktorej osi je väčší rozdiel medzi veľkosťou obrázka a veľkosťou PSF. Výmena okrajov pre 2 GPU je znázornená na obrázku 5.1. V prípade štyroch GPU sa obrázok vždy rozdelí na dve časti v ose  $x$  a na dve časti v ose  $y$ , dokopy teda na štyri časti. V tomto prípade výmena prebieha v dvoch krokoch, ktoré sú znázornené na obrázkoch 5.2a a 5.2b. Všetky prenosy, ktoré sú znázornené na jednom obrázku prebiehajú asynchrónne.

6. V prípade, že karty nie sú zapojené na jednom PCI-E prepínači, tak *peer-to-peer kópia* je automaticky urobená cez CPU pamäť [9], čo je pomalšie, ako v prípade priamej kópie medzi kartami.



(a) Prvá časť výmeny okrajov medzi štyrmi GPU.



(b) Druhá časť výmeny okrajov medzi štyrmi GPU.

Obr. 5.2: Výmena okrajov medzi GPU. Svetlejší odtieň farby označuje tú časť, ktorá sa kopíruje (zdroj), tmavší odtieň príslušnej farby označuje miesto, na ktoré je časť skopírovaná, tj. napríklad svetlomodrá časť sa kopíruje na miesto tmavomodrej časti. Všetky prenosy na jednom obrázku sú asynchrónne.



## Kapitola 6

### Experimenty

V tejto kapitole sa venujeme experimentálnemu porovnaniu jednotlivých dekonvolučných metód z hľadiska presnosti, rýchlosti a z hľadiska kvality výstupu. Následne implementované metódy porovnávame voči komerčnému softwaru Huygens [14], ktorý sa bežne používa v oblasti spracovania obrazu, a teda aj na dekonvolúciu obrázkov. Z popisovaných dekonvolučných metód program Huygens počíta len EM-MLE dekonvolúciu.

#### 6.1 Použité miery

Ako miery na porovnanie dvoch obrázkov boli zvolené tri miery: *priemerná kvadratická chyba*<sup>1</sup>, *maximálna kvadratická chyba* a miera SSIM [15]. Nech  $\Omega$  je doménou obrázkov  $F$  a  $G$  (keďže obrázky majú rovnakú veľkosť predpokladáme, že majú aj rovnakú doménu). Priemerná kvadratická chyba je potom daná ako:

$$MSE(F, G) = \frac{1}{S_x S_y S_z} \sum_{(x,y,z) \in \Omega} (F(x, y, z) - G(x, y, z))^2 \quad (6.1)$$

kde obrázky  $F$  a  $G$  sú porovnávané obrázky rovnakej veľkosti  $S_x, S_y, S_z$ . Priemerná kvadratická chyba je populárna a intuitívna miera, ktorá bude schopná dobre zmerať napríklad rozdiely medzi presnosťou jednotlivých metód.

Maximálna kvadratická miera nám určuje, aká najväčšia chyba je v obraze prítomná a je daná ako:

$$MaxSE(F, G) = \max_{(x,y,z) \in \Omega} (F(x, y, z) - G(x, y, z))^2 \quad (6.2)$$

Poslednou mierou je miera SSIM. Táto miera rieši nedostatky miery MSE medzi ktoré patrí napríklad skutočnosť, že obrázky môžu byť roz-

---

1. Mean Square Error – MSE

dielne, no mieru MSE môžu mať veľmi podobnú alebo dokonca až rovnakú. Miera SSIM sa počíta takto:

$$S(F, G) = f(l(F, G), c(F, G), s(F, G)) \quad (6.3)$$

$$SSIM(F, G) = \frac{1}{S_x S_y S_z} \sum_{(x,y,z) \in \Omega} S(F, G) \quad (6.4)$$

kde  $f$  je kombinačná funkcia,  $l(F, G)$  je funkcia miery jasú,  $c(x, y)$  je funkcia miery kontrastu a  $c(F, G)$  je funkcia miery štruktúry obrázkov  $F$  a  $G$ . Definíciu týchto funkcií je možné nájsť v [15]. Pre účely tejto práce je podstatné, že miera SSIM má isté veľmi dobré vlastnosti ako napríklad vlastnosť ohraničenia  $S(F, G) \leq 1$  a jedinečného maxima ( $SSIM(F, G) = 1$  práve vtedy a len vtedy, ak  $F(x, y, z) = G(x, y, z)$  pre všetky  $x, y, z$ ). Túto mieru budeme používať hlavne na porovnanie výsledkov dvoch rôznych dekonvolučných metód.

## 6.2 Príprava

Experimenty používali dva typy vstupných obrázkov. Obrázky typu A sú zobrazené v prílohe A, obrázky typu B sú zobrazené v prílohe B.

PSF mikroskopu bola ešte pred testami orezaná o nulové hodnoty na okrajoch. Tento krok neporušuje správnosť výsledkov a vďaka nemu sú jednotlivé metódy rýchlejšie. Obzvlášť dôležitý je tento krok pre implementácie, ktoré využívajú viacero GPU, kde by dochádzalo k zbytočnej výmene príliš veľkých okrajov jednotlivých častí. Obidve PSF sú zobrazené v prílohe C.

Všetky nasledujúce experimenty boli urobené na počítači tejto konfigurácie: štvorjadrové CPU Intel Xeon 2.4Ghz, 4×GPU NVIDIA Tesla 2090. Obrázky, ktoré boli dekonvolvované používajú typ `int162`. Parameter  $e$  pre Wienerovu dekonvolúciu mal hodnotu 0.001, parameter  $\lambda$  pre metódu ICTM mal hodnotu 0.02. Počet CPU vlákien, ktoré počítali Fourierovu transformáciu bol 8.

## 6.3 Porovnanie presnosti výpočtu

Presnosť výpočtu porovnáваме na päťici obrázkov každého typu – A a B. Pre účely tohto testu však boli jednotlivé obrázky orezané na veľkosť  $480 \times$

2. `int16` je 16 bitový dátový typ určený na ukladanie celých čísel.

480×40, a to z toho dôvodu, aby bolo možné vyhodnotiť všetky metódy v dátových typoch float aj double<sup>3</sup>.

Samotný test prebiehal nasledujúcim spôsobom: pre každú dekonvolučnú metódu sme vypočítali výsledný obrázok v dátovom type long double. Tento obrázok predstavoval tzv. *ground truth*, teda referenčný obrázok, voči ktorému sme porovnávali výsledky počítané v dátovom type float a double na CPU a GPU. Výsledky týchto meraní sumarizuje tabuľka 6.1 a 6.2. Uvedené hodnoty sú priemerné hodnoty z piatich meraní rôznych obrázkov príslušnej sady. Počet iterácií metód EM-MLE a ICTM je 40.

| Výpočtová jednotka | Wiener               |                      | EM-MLE               |        | ICTM                 |        |
|--------------------|----------------------|----------------------|----------------------|--------|----------------------|--------|
|                    | MSE                  | MaxSE                | MSE                  | MaxSE  | MSE                  | MaxSE  |
| CPU                | $2.2 \times 10^{-6}$ | $6.4 \times 10^{-4}$ | $4.0 \times 10^{-6}$ | 0.0422 | $6.5 \times 10^{-6}$ | 0.0051 |
| 1 GPU              | $1.6 \times 10^{-6}$ | 0.0013               | $1.1 \times 10^{-5}$ | 0.0902 | $1.2 \times 10^{-5}$ | 0.0085 |
| 2 GPU              | 0.3149               | 18.2599              | $1.2 \times 10^{-5}$ | 0.227  | $4.1 \times 10^{-6}$ | 0.0019 |
| 4 GPU              | 0.315                | 18.263               | $1.0 \times 10^{-5}$ | 0.1924 | $3.3 \times 10^{-6}$ | 0.0011 |

(a) Presnosť typu float.

| Výpočtová jednotka | Wiener                |                       | EM-MLE                |                       | ICTM                  |                       |
|--------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
|                    | MSE                   | MaxSE                 | MSE                   | MaxSE                 | MSE                   | MaxSE                 |
| CPU                | $7.4 \times 10^{-24}$ | $1.8 \times 10^{-21}$ | $1.1 \times 10^{-23}$ | $8.1 \times 10^{-20}$ | $1.4 \times 10^{-22}$ | $7.5 \times 10^{-20}$ |
| 1 GPU              | $8.7 \times 10^{-24}$ | $9.6 \times 10^{-21}$ | $8.6 \times 10^{-23}$ | $8.3 \times 10^{-19}$ | $4.3 \times 10^{-23}$ | $2.9 \times 10^{-20}$ |
| 2 GPU              | 0.3149                | 18.2621               | $1.2 \times 10^{-22}$ | $2.9 \times 10^{-18}$ | $2.9 \times 10^{-23}$ | $1.9 \times 10^{-20}$ |
| 4 GPU              | 0.3149                | 18.2621               | $1.2 \times 10^{-22}$ | $1.9 \times 10^{-28}$ | $3.0 \times 10^{-23}$ | $2.1 \times 10^{-20}$ |

(b) Presnosť typu double.

Tabuľka 6.1: Porovnanie presnosti jednotlivých dekonvolučných metód, počítaných pre orezané obrázky typu A.

3. Pokiaľ by sme obrázky neorezali, neboli by sme schopní vypočítať niektoré metódy kvôli nedostatku pamäte.

| Výpočtová jednotka | Wiener               |                      | EM-MLE               |        | ICTM                 |                      |
|--------------------|----------------------|----------------------|----------------------|--------|----------------------|----------------------|
|                    | MSE                  | MaxSE                | MSE                  | MaxSE  | MSE                  | MaxSE                |
| CPU                | $5.6 \times 10^{-7}$ | $5.1 \times 10^{-5}$ | $4.6 \times 10^{-7}$ | 0.0032 | $4.7 \times 10^{-7}$ | $1.3 \times 10^{-4}$ |
| 1 GPU              | $2.2 \times 10^{-7}$ | $7.6 \times 10^{-5}$ | $1.4 \times 10^{-6}$ | 0.0092 | $1.6 \times 10^{-6}$ | $4.1 \times 10^{-4}$ |
| 2 GPU              | 463.439              | 12391.981            | $1.5 \times 10^{-6}$ | 0.0247 | $1.2 \times 10^{-6}$ | $5.8 \times 10^{-5}$ |
| 4 GPU              | 463.449              | 12392.288            | $2.0 \times 10^{-6}$ | 0.0479 | $1.2 \times 10^{-6}$ | $9.2 \times 10^{-5}$ |

(a) Presnosť typu float.

| Výpočtová jednotka | Wiener                |                       | EM-MLE                |                       | ICTM                  |                       |
|--------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
|                    | MSE                   | MaxSE                 | MSE                   | MaxSE                 | MSE                   | MaxSE                 |
| CPU                | $1.4 \times 10^{-24}$ | $9.6 \times 10^{-23}$ | $1.6 \times 10^{-24}$ | $1.6 \times 10^{-20}$ | $6.4 \times 10^{-23}$ | $1.5 \times 10^{-20}$ |
| 1 GPU              | $1.1 \times 10^{-24}$ | $7.5 \times 10^{-22}$ | $1.3 \times 10^{-23}$ | $8.1 \times 10^{-20}$ | $3.6 \times 10^{-24}$ | $1.0 \times 10^{-21}$ |
| 2 GPU              | 463.443               | 12392.217             | $1.7 \times 10^{-23}$ | $1.8 \times 10^{-19}$ | $2.2 \times 10^{-24}$ | $4.7 \times 10^{-22}$ |
| 4 GPU              | 463.452               | 12393.862             | $1.8 \times 10^{-23}$ | $2.5 \times 10^{-19}$ | $2.8 \times 10^{-24}$ | $3.8 \times 10^{-22}$ |

(b) Presnosť typu double.

Tabuľka 6.2: Porovnanie presnosti jednotlivých dekonvolučných metód, počítaných pre orezané obrázky typu B.

Na základe týchto výsledkov môžeme konštatovať, že dátový typ float je dostačujúci pre počítanie všetkých dekonvolučných metód – miery MSE a MaxSE nadobúdajú veľmi nízke hodnoty. V prípade Wienerovej dekonvolučnej metódy sa ale objavila jedna anomália – výpočty pre 2 alebo 4 GPU majú MSE a MaxSE podstatne väčšie. Táto anomália súvisí s dodatočným paddingom, ktorý je potrebný pre výpočet na viacerých GPU<sup>4</sup>. Keďže padding obsahuje nulové hodnoty, pravdepodobne sú výsledky ovplyvnené výpočtom Wienerovej dekonvolúcie vo frekvenčnej doméne, kde je prítomné delenie, ktoré môže ovplyvňovať výsledok práve v prípadoch, keď delíme hodnotami blízkymi nule. Ak umelo pridáme dodatočný padding aj pre výpočet na 1 GPU či CPU, dostávame rovnaké<sup>5</sup> výsledky ako pre výpočet na 2, resp. 4 GPU.

4. V tomto prípade ide o padding, ktorý je nutné urobiť napríklad z toho dôvodu, aby bolo možné obrázok rozdeliť na dve rovnaké časti, tj. veľkosť musí byť párna.

5. Rovnaké v tom zmysle, že prípadné rozdiely sú veľmi malé, podobne ako v prípade ostatných dekonvolučných metód.

### 6.4 Porovnanie rýchlosti výpočtu

V tejto podkapitole porovnáme najprv rýchlosti jednotlivých implementácií na orezaných obrázkoch, podobne ako v teste presnosti v predošlej sekcii<sup>6</sup>. V tabuľke 6.3 môžeme nájsť výsledky tohto merania. Uvedené časy sú v sekundách a sú opäť vytvorené ako priemer piatich rôznych meraní na obrázkoch jednej sady.

| Výpočtová jednotka | Dátový typ  | Wiener |     | EM-MLE |       | ICTM  |       |
|--------------------|-------------|--------|-----|--------|-------|-------|-------|
|                    |             | A      | B   | A      | B     | A     | B     |
| CPU                | float       | 1.9    | 1.8 | 75.6   | 75.4  | 220.8 | 222.5 |
|                    | double      | 2.5    | 2.4 | 82.6   | 79.3  | 227   | 229.6 |
|                    | long double | 4.6    | 4.6 | 134.6  | 132.9 | 376.9 | 379.5 |
| 1 GPU              | float       | 1.3    | 1.3 | 3.4    | 3.4   | 8.9   | 8.9   |
|                    | double      | 1.9    | 1.9 | 6.1    | 6.1   | 16.1  | 16.1  |
| 2 GPU              | float       | 1.8    | 1.8 | 2.8    | 2.8   | 7.8   | 8.1   |
|                    | double      | 2.5    | 2.6 | 4.9    | 4.9   | 13.8  | 13.8  |
| 4 GPU              | float       | 2.6    | 2.3 | 2.7    | 2.7   | 8     | 7.9   |
|                    | double      | 3.2    | 3.1 | 4.6    | 4.5   | 13.6  | 13.7  |

Tabuľka 6.3: Porovnanie rýchlosti implementovaných metód. Uvádzame dve hodnoty pre každú metódu – jednu pre obrázky typu A, druhú pre obrázky typu B.

Z tabuľky vidíme, že GPU implementácie rýchlosťou predčili CPU implementácie a v prípade iteratívnych metód dokonca viac než 20-krát. To dokazuje, že problém dekonvolúcie je dobre paralelizovateľný a vhodný na implementáciu na GPU.

Rýchlosť implementácií, ktoré využívajú viacero GPU oproti tým, ktoré využívajú len jedno GPU je rôzna v závislosti od konkrétnej dekonvolučnej metódy. V prípade Wienerovej dekonvolúcie je spomalenie dané hlavne tým, že proces rozdelenia a komplexifikácie obrázku prebieha na CPU, kde neboli využité ďalšie jadrá procesoru ani vektorové inštrukcie. V prípade iteratívnych metód sú rozdiely dané tým, že okraje jednotlivých častí obrázku<sup>7</sup> sú v prípade metódy EM-MLE menšie ako v prípade metódy ICTM, kvôli väčšej veľkosti konvolučného jadra  $A$  použitého v metóde ICTM.

Ďalší test rýchlosti bol urobený na neorezaných obrázkoch z prílohy A

6. Orezané obrázky používame opäť z toho dôvodu, aby všetky metódy mali dostatok pamäte.

7. Svoju úlohu tu zohráva aj padding, ktorý celkovú veľkosť okrajov zväčšuje.

a B. Na týchto obrázkoch porovnáваме rýchlosť programu, ktorý je súčasťou tejto práce s rýchlosťou softwaru Huygens. Experimenty urobené v Huygensi ale používali neorezanú PSF. Z tohto dôvodu prikladáme v tabuľke 6.4a výsledky našich metód, ktoré počítali s neorezanou PSF a v tabuľke 6.4b výsledky, ktoré počítali s orezanou PSF.

| Výpočtová jednotka | Dátový typ  | Wiener   |          | EM-MLE |        | ICTM     |          |
|--------------------|-------------|----------|----------|--------|--------|----------|----------|
|                    |             | A        | B        | A      | B      | A        | B        |
| CPU                | float       | 51.1     | 50.2     | 565.9  | 526.3  | 3582.4   | 3512.7   |
|                    | double      | 38.4     | 38       | 646.5  | 621.8  | 1355.5   | 1289.4   |
|                    | long double | 37.3     | 37.4     | 1056.1 | 1021.8 | 4859.3   | 4947.2   |
| 1 GPU              | float       | –        | –        | –      | –      | –        | –        |
|                    | double      | –        | –        | –      | –      | –        | –        |
| 2 GPU              | float       | 11.7     | 11.8     | 20.7   | 20.2   | –        | –        |
|                    | double      | –        | –        | –      | –      | –        | –        |
| 4 GPU              | float       | 15.1     | 15.1     | 27.5   | 27.3   | –        | –        |
|                    | double      | 20       | 20.2     | –      | –      | –        | –        |
| Huygens            | double      | nepočíta | nepočíta | 158.7  | 168.7  | nepočíta | nepočíta |

(a) Rýchlosť výpočtu jednotlivých metód pri počítaní s neorezanou PSF.

| Výpočtová jednotka | Dátový typ  | Wiener |      | EM-MLE |       | ICTM   |        |
|--------------------|-------------|--------|------|--------|-------|--------|--------|
|                    |             | A      | B    | A      | B     | A      | B      |
| CPU                | float       | 10.9   | 10.6 | 415.1  | 390.8 | 971.3  | 921.9  |
|                    | double      | 14     | 13.5 | 432.3  | 407.7 | 1072.4 | 1007.2 |
|                    | long double | 25.2   | 25.2 | 750.8  | 708   | 1809.6 | 1635   |
| 1 GPU              | float       | 6.8    | 6.8  | 19.3   | 18.7  | –      | –      |
|                    | double      | –      | –    | –      | –     | –      | –      |
| 2 GPU              | float       | 7.7    | 7.6  | 14.8   | 14.4  | 30.3   | 29.5   |
|                    | double      | 11.4   | 11.4 | –      | –     | –      | –      |
| 4 GPU              | float       | 11.1   | 11.1 | 12.1   | 11.8  | 38.2   | 36.7   |
|                    | double      | 15.7   | 15.8 | 21.3   | 20.7  | –      | –      |

(b) Rýchlosť výpočtu jednotlivých metód pri počítaní s orezanou PSF.

Tabuľka 6.4: Porovnanie rýchlosti výpočtu implementovaných metód a softvéru Huygens. Počet iterácií bol v rozmedzí 36-40. Uvedené hodnoty sú v sekundách. Pokiaľ hodnota nie je uvedená, znamená to, že na výpočet nebolo k dispozícii dostatok pamäte.

Z tabuliek vidíme, že v prípade implementácií, ktoré používajú viacero GPU je dôležitá veľkosť PSF, ktorá udáva aké veľké dátové prenosy musia byť vykonané. Pokiaľ teda môžeme PSF orezať, mali by sme tak urobiť – jednak tým znižujeme pamäťové nároky jednotlivých metód a zároveň urýchlujeme počítanie dekonvolučných metód na viacerých GPU. Celkovo sa ale ukázalo, že naše implementácie na GPU podľa očakávania rýchlosťou niekoľkonásobne prekonal softvér Huygens.

Posledný test sa venuje už len jednotlivým GPU implementáciám a ukazuje, ktoré časti výpočtu zaberajú koľko času. Výsledky sú v tabuľke 6.5, počítané v dátovom type float pre obrázky typu A.

| Časť výpočtu                 | Wiener |       | EM-MLE |       | ICTM  |       |
|------------------------------|--------|-------|--------|-------|-------|-------|
|                              | 2 GPU  | 4 GPU | 2 GPU  | 4 GPU | 2 GPU | 4 GPU |
| Padding                      | 4.9    | 5.8   | 5.2    | 5.5   | 5.5   | 5.7   |
| Prenos dát medzi CPU a GPU   | 1.8    | 1.6   | 1.3    | 0.9   | 1.2   | 1     |
| Prenos dát medzi GPU         | –      | –     | 0.5    | 1.3   | 2.2   | 9.1   |
| Rozdelenie a komplexifikácia | 0.8    | 3.4   | –      | –     | –     | –     |
| Výpočet                      | 0.2    | 0.2   | 7.7    | 4.4   | 21.4  | 22.4  |
| Celková doba                 | 7.7    | 11    | 14.7   | 12.1  | 30.3  | 38.2  |

Tabuľka 6.5: Doba strávená v jednotlivých častiach výpočtu.

## 6.5 Porovnanie kvality výstupu

Kvalitu výstupu budeme porovnávať na reálnych obrázkoch. Výsledky dekonvolúcie jednotlivých dekonvolučných metód a výsledky z Huygensa sa nachádzajú v prílohe A a B (v prílohe sa nachádza len jeden obrázok z každého typu, ostatné sú v priloženom DVD). Porovnanie miery SSIM voči výsledkom z Huygensa je v tabuľke 6.6.

| Typ<br>obrázkov | Wiener | EM-MLE | ICTM   |
|-----------------|--------|--------|--------|
| Typ A           | 0.1849 | 0.3135 | 0.3314 |
| Typ B           | 0.0377 | 0.1075 | 0.092  |

Tabuľka 6.6: Porovnanie miery SSIM jednotlivých dekonvolučných metód voči výsledkom z Huygensu.

Z tabuľky by sa mohlo zdať, že naše metódy dopadli v porovnaní s Huygensom zle, no ak sa pozrieme na výsledky v prílohe A a B a na ostatné obrázky (v priloženom DVD) tak vidíme, že hoci Huygens dáva kvalitnejšie výsledky, naše metódy dosahujú porovnateľné výsledky. Nízke hodnoty miery SSIM sú dané predovšetkým tým, že Huygens je schopný zachovať farbu pozadia, čo naše metódy tak dobre nezvládajú.

Z našich metód sa ukazuje ako najlepšia metóda EM-MLE. Vizuálne dosahuje veľmi podobné výsledky ako Huygens a má z implementovaných metód najmenej artefaktov, čo je vidieť hlavne v osi  $z$ . Medzi jej ďalšie výhody patrí nižšia pamäťová náročnosť ako metóda ICTM a takisto aj menšia veľkosť konvolučného jadra, čo má význam hlavne pre implementácie, ktoré využívajú viacero GPU.



## Kapitola 7

### Záver

Dekonvolúcia je náročná a rozsiahla problematika v oblasti spracovania obrazu. V práci sme popisovali a implementovali tri dekonvolučné metódy – Wienerovu dekonvolúciu, metódu EM-MLE a metódu ICTM. Ide o metódy neslepej dekonvolúcie, teda o metódy, pri ktorých máme k dispozícii PSF optickej sústavy, ktorá bola použitá na snímanie obrazu.

Tieto metódy sme implementovali na CPU, GPU a na systéme s viacerými GPU. Najviac priestoru sme venovali výberu vhodného algoritmu na počítanie rýchlej konvolúcie, čo je veľmi častá operácia, ktorá sa vyskytuje v týchto metódach. Jednotlivé metódy sme porovnali z hľadiska presnosti, rýchlosti a kvality výstupu. Implementácie boli porovnané aj voči softwaru Huygens.

Zistili sme, že z hľadiska presnosti výpočtu postačuje dátový typ float. Rozdiely oproti výpočtu v dátovom type long double sú zanedbateľné. Operácie v dátovom type float sú oproti ostatným typom na GPU najrýchlejšie. Ďalšia výhoda tohto typu spočíva v menšom množstve potrebnej pamäte oproti dátovému typu double.

Z hľadiska rýchlosti sa ukázalo, že problém dekonvolúcie je veľmi dobre a efektívne implementovateľný na GPU, keďže mnoho operácií je vykonávaných bod po bode a rýchla Fourierova transformácia je dobre paralelizovateľná a rýchlo vypočítateľná na GPU. Implementácie, ktoré používajú viacero GPU sú zaujímavé hlavne z dôvodu ušetrenia pamäte, a teda možnosti počítania dekonvolúcie väčších obrázkov. Obecne sú rýchlejšie len vtedy, ak sú obrázky dostatočne veľké (teda problém je dostatočne výpočetne náročný) a PSF je dostatočne malá (v tom prípade sú dátové prenosy minimálne).

Porovnanie kvality výstupu so softwarom Huygens dopadlo v prospech Huygensu, no výsledky implementácie, ktorá je súčasťou tejto práce, sú s Huygensom porovnateľné a celkovo veľmi podobné. Z hľadiska rýchlosti počítania bol Huygens podľa očakávania niekoľkonásobne pomalší. Je to dané, ako sme už spomenuli, hlavne tým, že problém dekonvolúcie je veľmi dobre paralelizovateľný, a teda vhodný pre implementáciu na GPU.

## Literatúra

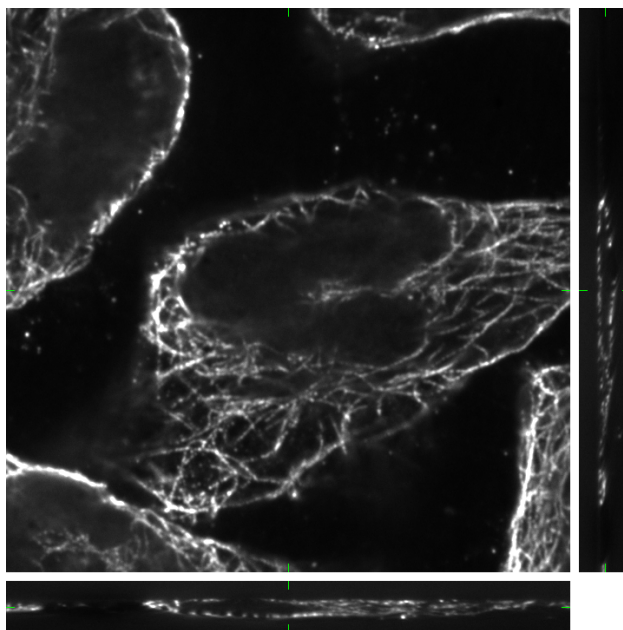
- [1] Starck, J., L., Murtagh, F.: *Astronomical Image and Data Analysis*, Springer, 2006, ISBN 978-3-540-33024-0, 335s..
- [2] *Non-graphic computing with graphics processors* [online], 2008, Dostupné na: <http://ixbtlabs.com/articles3/video/cuda-1-p1.html> [cit. 2013-05-24].
- [3] Kozubek, M.: *Prezentácia ku predmetu: Digitální zpracování obrazu* [online], Masarykova univerzita, 2009, Dostupné na: [https://is.muni.cz/auth/el/1433/podzim2009/PV131/um/8943284/lecture01\\_2009.pdf?studium=531487](https://is.muni.cz/auth/el/1433/podzim2009/PV131/um/8943284/lecture01_2009.pdf?studium=531487) [cit. 2013-05-24].
- [4] Pratt, W. K.: *Digital Image Processing: PIKS Inside*, Wiley-Interscience, 2001, ISBN 978-0471374077, 656 s..
- [5] Karas, P., Svoboda, D.: *Algorithms for Efficient Computation of Convolution* [online], Design and Architectures for Digital Signal Processing, InTech, 2012, Dostupné na: <http://www.intechopen.com/books/design-and-architectures-for-digital-signal-processing/algorithms-for-efficient-computation-of-convolution> [cit. 2013-05-24].
- [6] Horová, I., Zelinka, J.: *Numerické metody*, Masarykova univerzita, 1999, ISBN 80-210-2202-7, 230s..
- [7] Verveer, P., J.: *Computational and Optical Methods for Improving Resolution and Signal Quality in Fluorescence Microscopy*, Delft University of Technology, 1998, Dostupné na: [http://www.tnw.tudelft.nl/fileadmin/Faculteit/TNW/Over\\_de\\_faculteit/Afdelingen/Imaging\\_Science\\_and\\_Technology/Research/Research\\_Groups/Quantitative\\_Imaging/Publications/PHD\\_Theses/doc/thesis\\_verveer.pdf](http://www.tnw.tudelft.nl/fileadmin/Faculteit/TNW/Over_de_faculteit/Afdelingen/Imaging_Science_and_Technology/Research/Research_Groups/Quantitative_Imaging/Publications/PHD_Theses/doc/thesis_verveer.pdf) [cit. 2013-05-24].

- [8] Bracewell, R., N.: *The Fourier Transform and Its Applications*, McGraw-Hill, 2000, ISBN 0-07-116043-4, 422s..
- [9] Micikevicius, P.: *Multi-GPU Programming* [online], Supercomputing, 2011, Dostupné na: <http://www.nvidia.com/docs/IO/116711/sc11-multi-gpu.pdf> [cit. 2013-05-24].
- [10] Luengo, C.: *Knižnica libics* [online], Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, 2010, Dostupné na: <http://libics.sourceforge.net/> [cit. 2013-05-24].
- [11] Frigo, M., Johnson, S., G.: *Knižnica FFTW* [online], Dostupné na: <http://www.fftw.org/> [cit. 2013-05-24].
- [12] NVIDIA Corporation: *Dokumentácia knižnice CUFFT* [online], Dostupné na: <http://docs.nvidia.com/cuda/cufft/index.html> [cit. 2013-05-24].
- [13] NVIDIA Corporation: *Dokumentácia knižnice CUBLAS* [online], Dostupné na: <http://docs.nvidia.com/cuda/cublas/index.html> [cit. 2013-05-24].
- [14] Scientific Volume Imaging Company: *Huygens* [online], Dostupné na: <http://www.svi.nl/HuygensSoftware> [cit. 2013-05-24].
- [15] Wang, Z., Bovik, A., C., Sheikh, H., R., Simoncelli, E., P.: *Image Quality Assessment: From Error Visibility to Structural Similarity* [online], IEEE Transactions on Image Processing, 2004, Dostupné na: <http://www.cns.nyu.edu/~zwang/files/papers/ssim.html> [cit. 2013-05-24].

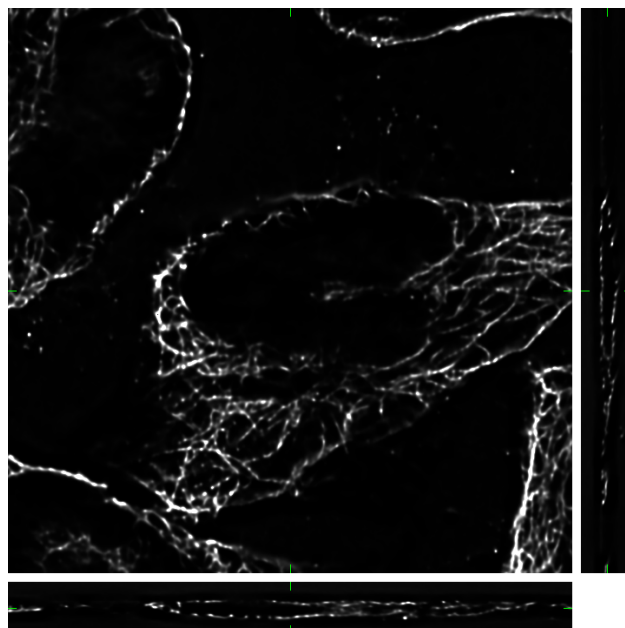
## Príloha A

### 1. typ dekonvolvovaných dát

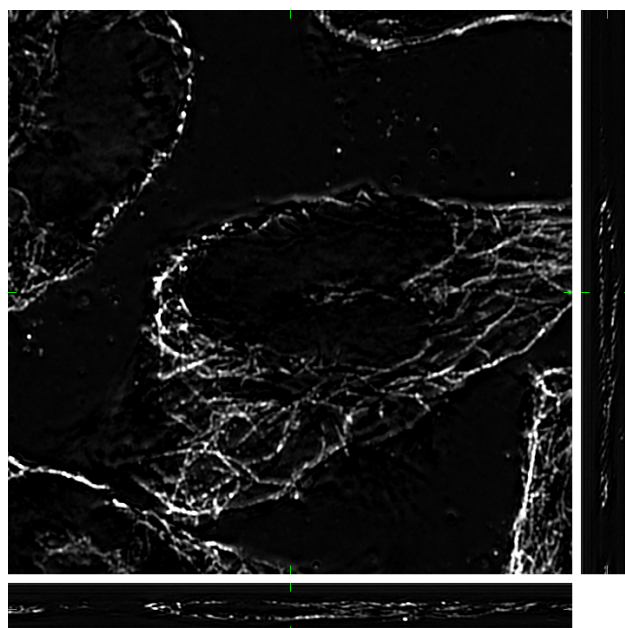
Uvádzame príklad jedného obrázku zo série dvadsiatich. Zachytené sú vlák-  
nité štruktúry na obrázku veľkosti veľkosti  $1024 \times 1024 \times 60$  (zobrazené sú  
v menšej veľkosti). Dekonvolvované Huygensom, Wienerovou dekonvo-  
lúciou, EM-MLE a ICTM metódou.



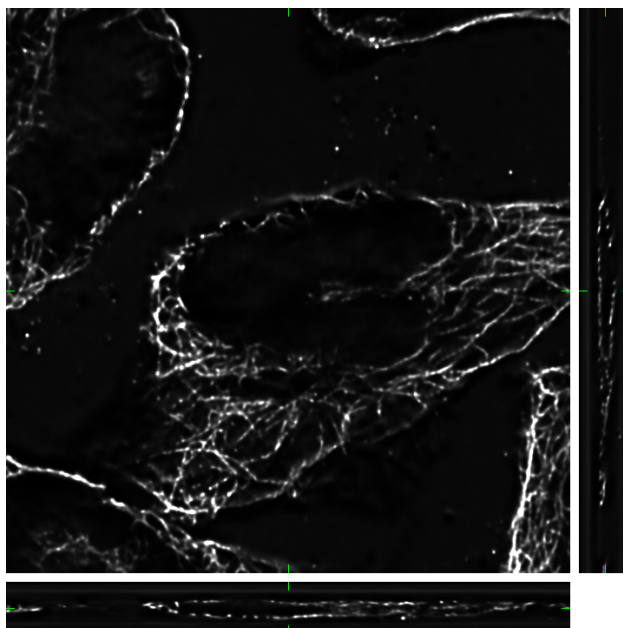
Obr. A.1: Originál



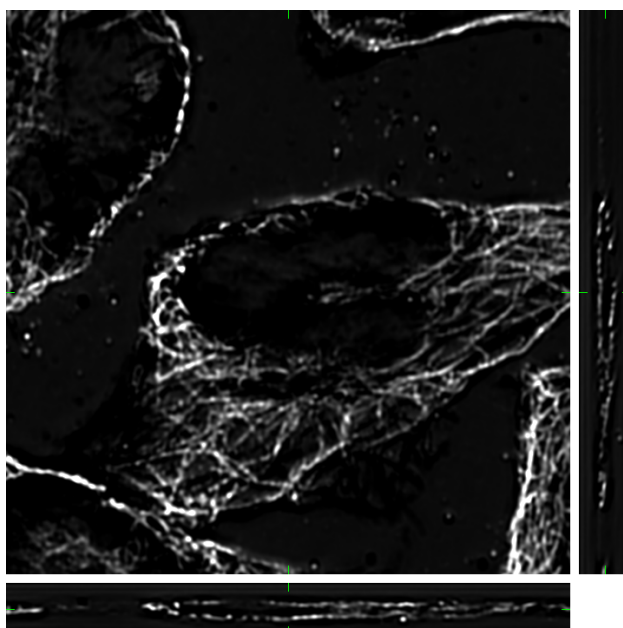
Obr. A.2: Huygens (40 iterací)



Obr. A.3: Wiener



Obr. A.4: EM-MLE (40 iterací)

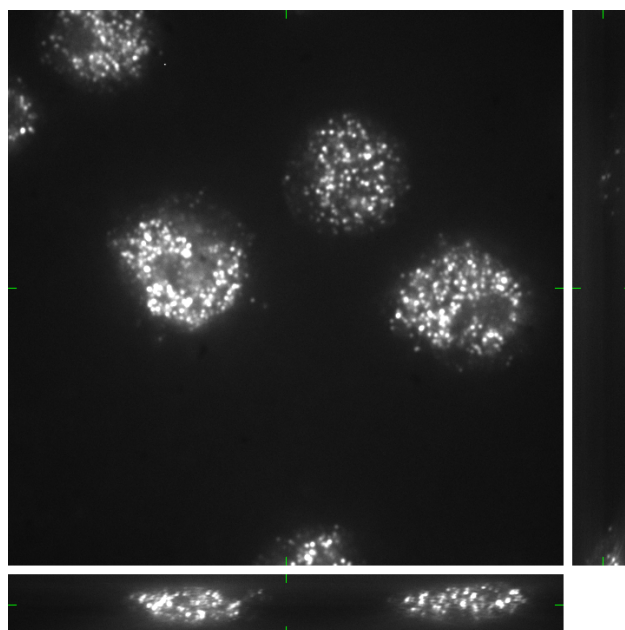


Obr. A.5: ICTM (40 iterací)

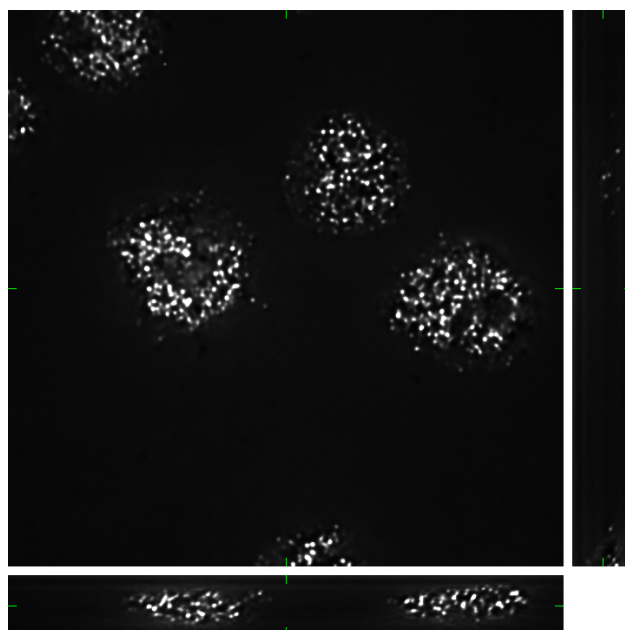
## Príloha B

### 2. typ dekonvolvovaných dát

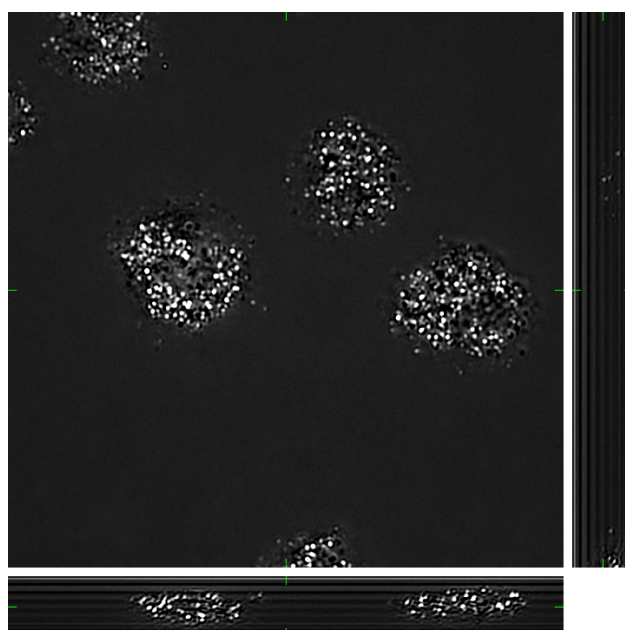
Príklad jedného obrázku guľovitých štruktúr veľkosti  $1024 \times 1024 \times 70$  (zobrazené sú v menšej veľkosti). Dekonvolvované Huygensom, Wienerovou dekonvolúciou, EM-MLE a ICTM metódou. Počet iterácií pre EM-MLE a ICTM je 36.



Obr. B.1: Originál

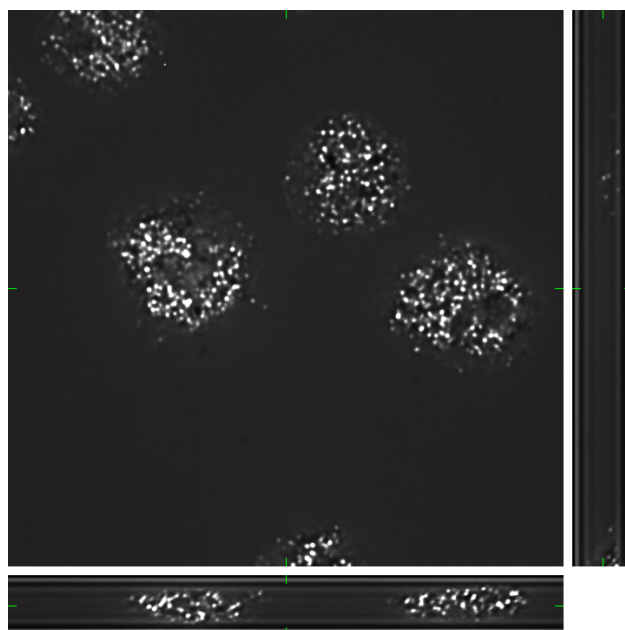


Obr. B.2: Huygens (36 iterací)

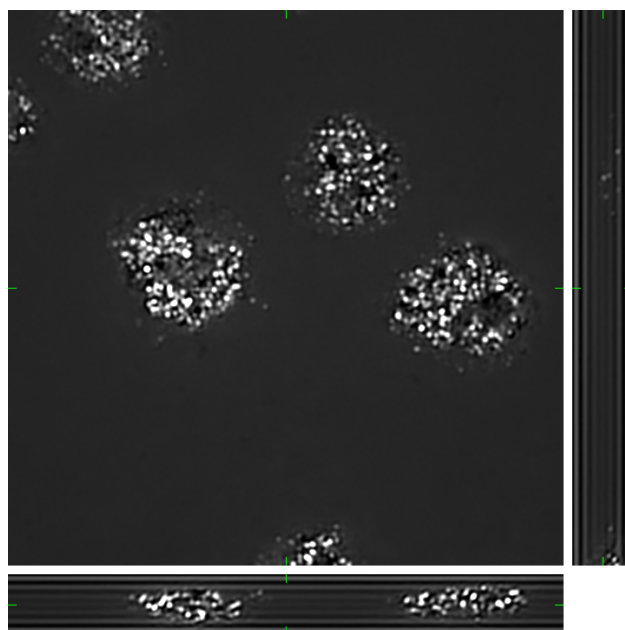


Obr. B.3: Wiener





Obr. B.4: EM-MLE (36 iterací)

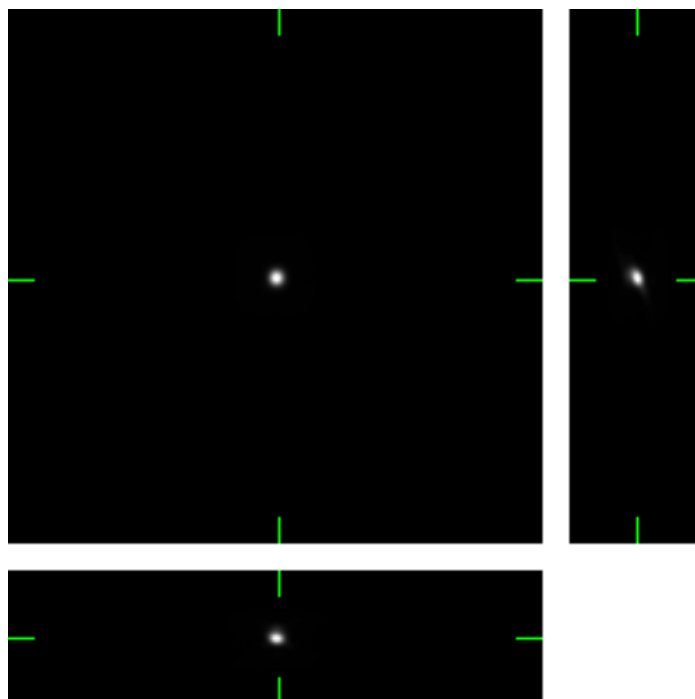


Obr. B.5: ICTM (36 iterací)

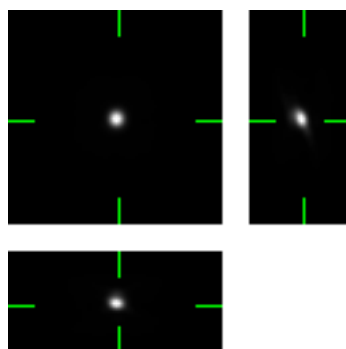
## Príloha C

### PSF

Obrázky použitej PSF. Na prvom obrázku je zobrazená originálna PSF, na druhom obrázku orezaná PSF.



Obr. C.1: Originálna PSF



Obr. C.2: Orezaná PSF