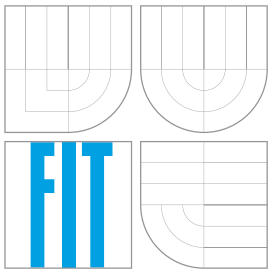
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VYHODNOCENÍ VAZEB MEZI PÁRY KONTAKTŮ INTRACEREBRÁLNÍCH SIGNÁLŮ EEG

EVALUATION OF RELATIONSHIPS BETWEEN PAIRS OF CONTACTS IN INTRACEREBRAL EEG

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN HRABOŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. KAROLÍNA KUPKOVÁ

BRNO 2016

Zadání diplomové práce/18966/2015/xhrabo06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Hraboš Martin, Bc.**
Obor: Bioinformatika a biocomputing
Téma: **Vyhodnocení vazeb mezi páry kontaktů intracerebrálních signálů EEG**
Evaluation of Relationships between Pairs of Contacts in Intracerebral EEG
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s metodami měření a vyhodnocení konektivity EEG signálu z hlubokých mozkových elektrod. Proveďte rešerši metod, které se v současné době používají a zhodnoťte je i z pohledu jejich výpočetní náročnosti.
2. Navrhněte a otestujte vhodné metody pro implementaci výpočtu vazby mezi vybranými páry kontaktů u intracerebrálních záznamů EEG. K dispozici budete mít záznamy EEG se vzorkováním 5kHz, délkou záznamu do 30 minut a maximálně 200 kanály.
3. Srovnejte a diskutujte rozdíly jednotlivých metod z pohledu informace, kterou přinášejí a z pohledu jejich výpočetní náročnosti.
4. Nejvýhodnější metodu implementujte jako zásuvný modul pro nástroj SignalPlant.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti budoucího rozšíření.

Literatura:

- Antony, A. R. et al. Functional Connectivity Estimated from Intracranial Predicts Surgical Outcome in Intractable Temporal Lobe Epilepsy. *PLoS One.* 2013, 8(10).
- Ortega, G. J. et. al. Synchronization clusters of interictal activity in the lateral temporal cortex of epileptic patients: intraoperative electrocorticographic analysis. *Epilepsia*. 2008, 49(2).
- Mormann, F. et al. Mean phase coherence as a measure for phase synchronization and its application to the EEG of epilepsy patients. *Physica D: Nonlinear Phenomena.* 2000, 144(3-4).

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kupková Karolína, Ing.**, UITS FIT VUT
Konzultant: Plešinger Filip, Ing., Ph.D., AV ČR
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Táto práca popisuje analýzu zvolených metód mozgovej konektivity. Tieto metódy vyhodnocujú väzby a závislosti medzi signálmi získanými pomocou hĺbkových intrakraniálnych elektród. Súčasťou tejto práce je aplikácia – zásuvný modul, ktorý určuje závislosti medzi signálmi pomocou Pearsonovych korelačných koeficientov. Pri výpočte týchto koeficientov je použítá akcelerácia pomocou GPU.

Abstract

This thesis describes selected methods of brain connectivity analysis. It was created an application, as a part of this thesis – plugin for evaluating relationships and dependencies between signals calculated as Pearson correlation coefficients. Computation of these coefficients is accelerated by GPU.

Kľúčové slová

Mozgová konektivita, intrakraniálne elektródy, elektroencefalografia, EEG, GPGPU, OpenCL, CUDA, grafická akcelerácia.

Keywords

Brain connectivity, intracranial electrodes, electroenecefalography, EEG, GPGPU, OpenCL, CUDA, graphics acceleration.

Citácia

HRABOŠ, Martin. *Vyhodnocení vazeb mezi páry kontaktů intracerebrálních signálů EEG*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kupková Karolína.

Vyhodnocení vazeb mezi páry kontaktů intracerebrálních signálů EEG

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pani Ing. Karolíny Kupkovej. Ďalšie informácie mi poskytli Ing. Filip Plešinger, Ph.D. a Ing. Petr Klimeš. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Hraboš
24. mája 2016

Poďakovanie

Špeciálne poďakovanie patrí Ústavu přístrojové techniky Akademie věd ČR, konkretne osobám Ing. Filip Plešinger, Ph.D a Ing. Petr Klimeš.

Obsah

Zoznam skratiek a symbolov

AMD Advanced Micro Devices – výrobca mikroprocesorov, integrovaných obvodov a grafických kariet (predtým *ATI*)

API Application Programming Interface – rozhranie pre programovanie aplikácií

Coh Coherence – koherencia

Corr Correlation – lineárna korelácia

CPU Central Processing Unit – hlavný procesor počítača

CSV Comma-separated Values – formát súboru s hodnotami oddelenými čiarkami

CT Computer Tomography – počítačová tomografia

CUDA Platforma pre paralelné výpočty (NVIDIA)

DCM Dynamic Causal Modeling – dynamické kauzálne modelovanie

DLL Dynamic-link Library – dynamicky linkovateľné knižnice

ECG/EKG Electrocardiography – elektrokardiografia

EEG Electroencephalography – elektroencefalografia

fMRI Functional Magnetic Resonance Imaging – funkcionálna magnetická rezonancia

GC Granger causality – Grangerova kauzalita

GPGPU General-purpose computing on graphics processing units – výpočet pomocou GPU

GPU Graphics Processing Unit – grafický procesor

HDF5 Hierarchical Data Format – formát pre ukladanie dát rôznych dátových typov

Hz Hertz – jednotka frekvencie

iEEG Intracranial electroencephalography – intrakraniálna elektroencefalografia

MDS Multidimensional scaling – multidimenzionálne škálovanie

MEG Magnetoencephalography – magnetoencefalografia

MIT Mutual Information – vzájomná informácia

MRI Magnetic Resonance Imaging – magnetická rezonancia

mV miliVolt – jednotka elektrického napätia

$NCorr$ Nonlinear Correlation – nelineárna korelácia

$NVCC$ NVIDIA CUDA Compiler Driver – nástroj pre preklad kernelov pre CUDU

$NVIDIA$ Výrobca procesorov do grafických kariet

$OpenCL$ Open Computing Language – štandard pre paralelné programovanie

PCA Principal Component Analysis – analýza hlavných komponent

PET Positron Emission Tomography – pozitrónová emisná tomografia

PTX Parallel Thread Execution – preložený kernel pre CUDU

SEM Structural Equation Modeling – štrukturálne modelovacie rovnice

TE Transfer of Entropy – prenos entropie

TMS Transcranial Magnetic Stimulation – transkraniálna magnetická stimulácia

Kapitola 1

Úvod

Elektroencefalografické vyšetrenie (EEG) patrí k najdôležitejším diagnostickým metódam v neurológii. Skúma elektrickú aktivitu mozgu vyšetrovanej osoby. Na makroskopickej úrovni bolo prvou neinvazívnou metódou skúmania mozgovej aktivity "zaživa". Od svojho počiatku v roku 1929 prešlo mnohými zmenami, vylepšeniami, stále však je priestor na výskum a rozvoj. Jednou zo špecifikácií, ktorou sa zaoberá aj táto diplomová práca, je vyšetrenie pomocou hlbokých mozgových elektród – intrakraniálna elektroencefalografia.

Intrakraniálne EEG (iEEG, alebo aj invazívne EEG) nachádza svoje uplatnenie najmä v radoch epileptických pacientov. Takto postihnutým ľuďom už častokrát niet inej nádeje ako podstúpiť toto vyšetrenie. Klasické EEG s povrchovými elektródami nedokáže zachytiť hlbokú mozgovú aktivitu, preto nastupuje iEEG, ktorého elektródy sa zavádzajú cez lebku hlboko do mozgu. Fokálna epilepsia, ktorej záchvaty pochádzajú z ohraničených ložísk – epileptogénne zóny, sú bežné pre viac ako 50% epileptický pacientov. Aj napriek veľkému pokroku farmakológie stále viac ako 30% pacientov trpí týmito záchvatmi. V tomto prípade sa ukazuje iEEG ako jediné možné riešenie ako detekovať zdroje záchvatov a prípadne ich eliminovať. Epileptogénne zóny sa občas podarí detekovať aj neinvazívnymi vyšetreniami ako sú napr. MRI[1], PET[2], či video-EEG. Pri prípadnom neúspechu týchto metód nastupujú invazívne metódy postavené na princípe implantovania elektród priamo do mozgu. Intrakraniálne elektródy dokážu zaznamenať signály z malých zhlukov neurónov, ktoré negenerujú dostatočne silné signály, aby ich bolo možné detekovať povrchovými elektródami. Zaznamenávajú aktivitu mozgovej kôry, ktorá nie je skrytá hlboko v mozgu a nie je detekovateľná inými monitorovacími technikami. Taktiež nedochádza k veľkému rušeniu signálu okolitými vplyvmi. Nevýhodou je, že takto je možné získať údaje len z oblastí nachádzajúcich sa relatívne blízko k elektróde. Ďalšou nevýhodou je, že existuje potenciálne riziko infekcie pri aplikovaní elektród do mozgu.

Na prvý pohľad nie príliš príjemná predstava tohto vyšetrenia vyžadujúca chirurgický zákrok však výrazne uľahčuje lokalizovanie a ohraničenie epileptogénnych ložísk a následné stanovenie účinnej terapie. Mnoho pacientov sa obáva tohto zákroku, pretože úspešnosť operácie býva asi 50%. Pri komplikáciach spojených s operáciou môže dôjsť až k odstráneniu časti mozgu. Výsledky úspešnej operácie sú však smerodajné pri ďalšej liečbe. Naďalej pretrváva strach pacientov a odahaduje sa, že len menej ako 3% pacientov, ktorí by z toho mali nejaký úžitok, podstúpia tento zákrok. Samotný zákrok pozostáva z rutinných predoperačných vyšetrení. Potom sa podajú anestetiká a neurochirurg navŕta malú dierku do lebečnej

[1] Magnetická rezonancia
[2] Pozitrónová emisná tomografia

kosti, cez ktorú vloží elektródy. V závislosti od konkrétneho vyšetrenia sa obvykle zavádza dve až šesť takýchto elektród. Potom ako sú aplikované elektródy pacient vykonáva bežné činnosti a očakáva sa epileptický záchvat. Všetka mozgová aktivita je zaznamenaná. Keď je získaný dostatočný objem údajov, dáta sa vyhodnotia a pacient sa vracia k bežnému životu[3].

Keďže ide o jednoduché zariadenie, ktoré generuje veľké množstvo signálov, je nutné tieto signály spracovávať pomocou výpočtovej techniky. Rozvoj vedy a techniky umožňuje stále viac spresňovať namerané hodnoty, zrýchľovať výpočty, porovnávať výsledky, aplikovať rôzne filtre, vizualizovať dáta, optimalizovať algoritmy, atď. Stále je teda priestor na skúmanie, zlepšovanie, experimentovanie, hodnotenie a v poslednom rade aplikovanie na pacientov, ktorým to môže výrazne zlepšiť kvalitu ich života.

V kapitole (2) sú uvedené ciele práce, čomu sa práca venuje a ako je možné dosiahnuť definovaných cieľov. Kapitola (3) popisuje problematiku práce. Sú tu zhrnuté dôležité odborné termíny, ktoré sa objavujú v nasledujúcom texte. Charakteristika a analýza jednotlivých metód mozgovej konektivity je uvedená v kapitole (4). Metódy sú analyzované aj z pohľadu výpočtovej náročnosti. Výsledkom tejto analýzy je návrh vhodnej metódy pre implementáciu. Kapitola (5) popisuje vybrané technológie, pomocou ktorých bude implementovaná navrhnutá metóda. Jej návrh je uvedený v kapitole (6). Následne je aplikácia implementovaná na základe návrhu aplikácie. Detailne je implementácia popísaná v kapitole (7). Testovaniu vytvorenej aplikácie sa venuje kapitola (8). Na konci práce (9) sa nachádza zhrnutie odvedenej práce, zhodnotenie výsledkov a navrhovaný ďalší možný vývoj projektu.

Tejto práci predchádzal semestrálny projekt, v rámci ktorého boli získané teoretické informácie nutné k ďalšiemu postupu práce a lepšiemu pochopeniu problematiky. Výsledok tohto projektu je začlenený do diplomovej práce, konkrétne v kapitolách (3) a (4). Prínosom tohto projektu bolo hlavne oboznámenie sa s problematikou a analýzou mozgovej konektivity.

[3] údaje o vyšetrení sú z Royal Hallamshire Hospital

Kapitola 2

Ciele práce a postup

2.1 Ciele práce

Cieľom tejto práce je zoznámiť sa s aktuálnymi výpočtovými metódami pre analýzu mozgovej konektivity. Dáta poskytnuté k tomuto účelu budú zaznamenané pomocou hĺbkových intrakraniálnych elektród. Treba zvoliť najvhodnejšiu metódu a pokúsiť sa ju optimalizovať tak, aby sa skrátil efektívny čas výpočtu. Takto optimalizovaná metóda bude potom implementovaná ako zásuvný modul pre už existujúcu aplikáciu SignalPlant[1].

2.2 Postup práce

Na začiatku je potrebné sa zoznámiť s problematikou mozgovej konektivity a metódami, ktoré túto konektivitu analyzujú. Táto tématika je zhrnutá v kapitolách (3) a (4). Pri spracovávaní bola použitá adekvátna literatúra, ktorá sa tejto problematike venuje. Je uvedený prehľad metód a je uvedená aj ich výpočtová zložitosť. Postup práce možno rozdeliť do nasledujúcich etáp:

- Získanie teoretických poznatkov z odbornej literatúry,
- Na základe získaných informácií je vytvorená analýza problematiky,
- Výsledkom tejto analýzy je návrh vhodnej metódy,
- Na základe návrhu vhodnej metódy je navrhnutá aplikácia,
- Aplikácia je implementovaná v súlade s návrhom aplikácie,
- Napokon je aplikácia otestovaná. Výsledky testov sú zhodnotené,

[1] SignalPlant – https://signalplant.codeplex.com/

Kapitola 3

Uvedenie do problematiky

Elektroencefalografia je základná elektrofyziologická metóda pre vyšetrenie vnútornej mozgovej aktivity. EEG registruje časopriestorové zmeny mozgových biopotenciálov, ktoré vznikajú na základe kontinuálnej aktivity dráždiacich membrán na synapsiách. Kladné a záporné náboje vytvárajú dipóly, ktoré sú obvykle kolmé k mozgovému povrchu. Snímacie elektródy potom zaznamenávajú rozdiely potenciálov medzi jednotlivými oblasťami [18]. Elektróda slúži na prepojenie medzi vodivou tekutinou v tkanivách a vstupným zosilňovačom zaznamenávacieho prístroja. Elektródy sa umiestňujú buď na povrch lebky alebo v našom prípade do lebečnej dutiny.

Ľudský mozog obsahuje miliardy neurónov, ktoré sú navzájom prepojené synapsiami. Mozog pre svoju činnosť využíva elektrochemické deje, pričom generuje bioelektrický signál. Bioelektrický signál je jednosmerný, rozpätie hodnôt bioelektrických mozgových potenciálov je na úrovni 5 až $210mV$ a rýchlosť generovania výbojov v cykloch za sekundu je 0,5 až $40Hz$. Namerané prúdy sú však veľmi nízke, preto je nutné ich prístrojovo zosilniť, vyfiltrovať a odstrániť nežiadúci šum. Kov v prostredí elektrolytu uvoľňuje kladne nabité ióny a sám sa pritom nabíja záporne[1] [30].

3.1 Typy elektród

Intrakaniálne elektródy sú obvykle vyrobené z nehrdzavejúcej ocele alebo platiny. Oba materiály sú kompatibilné s monitorovacou technikou MRI.

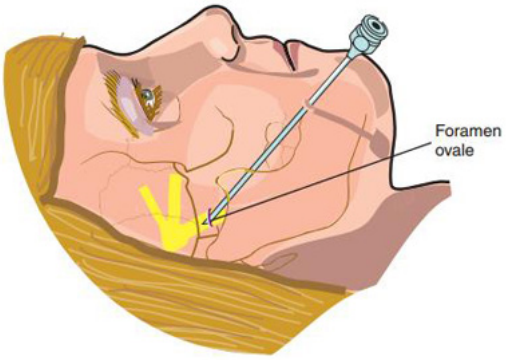
3.1.1 Foramen ovale elektródy

Semi-invazívne elektródy foramen ovale[2] sa používajú ako alternatíva k invazívnym monitorovacím technikám pri vyšetrovaní pacientov s epilepsiou spánkového laloku. Elektródy sú vkladané cez foramen ovale. Nedetekujú priamo mozgovú aktivitu hypokampu[3], ale zaznamenávajú spánkové elektrické výboje, ktoré môžu pochádzať z hypokampu. Výhodou týchto elektród je, že si nevyžadujú chirurgický zákrok. Obvykle sa podávajú anestetiká na eliminovanie bolesti. Nepríjemnosť spôsobená aplikovaním týchto elektród je však porovnateľná s ostatnými typmi intrakraniálnych elektród [22].

[1] Elektrický vodič, v ktorom sa elektrický náboj prenáša pohybom iónov

[2] Foramen ovale – jeden z väčších otvorov v lebke nachádzajúci v klinovej kosti, cez ktorý prenikajú nervy

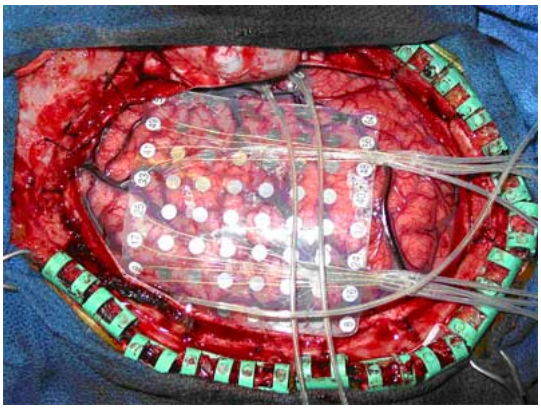
[3] Časť veľkého mozgu



Obr. 3.1: Foramen ovale elektróda

3.1.2 Subdurálne elektródy

Subdurálne elektródy pozostávajú zo série diskov pripevnených k tenkej plastovej fólii. Sú navrhnuté tak, aby držali na povrchu mozgovej kôry a zároveň nepoškodili mozog. Elektródy sú zoradné do pásov. Do vnútrolebečnej dutiny sú vložené cez vytvorený otvor a umiestnené podľa potreby buď pod spánkové alebo čelné laloky alebo pozdĺž stredového žľabu medzi hemisférami. Typicky sa používajú pásy veľkosti 1×4 až 1×8 cm. Mriežky potom rozmerov od 2×4 do 8×8 cm. V praxi sa častejšie používa zoradenie elektród do štvorcovej alebo obdĺžnikovej mriežky, aby sa pokryla väčšia oblasť mozgu. Takýmto spôsobom sa mapujú napr. oblasti mozgu kde sa nachádza centrum motoriky, reči, atď. V niektorých prípadoch sa používalo umiestnenie mriežkových elektród na obe polovice mozgu. Avšak takáto aplikácia intrakraniálnych elektród bola už často spájaná s neakceptovateľnou morbiditou. Aj pri tomto type elektród sa vyskytuje riziko infekcie. Infekcia sa vyskytuje približne pri 2 až 3% pacientov. Menej ako 1% tvorí riziko spôsobenia krvácania do mozgu [4].



Obr. 3.2: Subdurálne elektródy [6].

3.1.3 Hĺbkové mozgové elektródy

V porovnaní s ostatnými typmi elektród hĺbkové elektródy prenikajú cez mozgovú hmotu a dostávajú sa hlbšie do mozgovej dutiny. Pri ich implantovaní je nevyhnutné použiť aj iné

monitorovacie techniky, spravidla CT[4] alebo MRI. Môžu byť umiestnené na pár miestach – v mozgových lalokoch. Používajú sa dva typy hĺbkových elektród: kanylové[5] a drôtové. Každá elektróda obsahuje pozdĺž celej svojej dĺžky 6 až 12 kontaktných bodov v rozmedzí 5 až 10 mm. Je možné použiť pružné alebo pevné elektródy. Pružné elektródy sa považujú za bezpečnejšie, pretože pri náraze do ciev sa ohnú a nepoškodia cievy. Najčastejšie skúmanými oblasťami mozgu hĺbkovými elektródami sú spánkový a čelový lalok. Riziká spojené s týmto typom vyšetrenia tvoria ako aj u predošlých techník krvácanie do mozgu a vnútrolebečná infekcia. Riziko infikovania sa je približne 5%. Infekcia sa potom prejavuje ako meningitída[6] alebo zriedkavejšie ako encefalitída [21].

Obr. 3.3: Hĺbkové mozgové elektródy [7].

[4]Počítačová tomografia
[5]Kanyla – tenká trubica, ktorá sa zavádza do tela, obvykle na získanie tekutín alebo informácií
[6]Zápal mozgových blán

3.2 Mozgová konektivita

Mozgová konektivita zahrňuje nasledovné vzory:

- Štrukturálna konektivita – anatomické spoje,
- Funkcionálna konektivita – štatistické závislosti,
- Efektívna konektivita – kauzálne pôsobenie,

medzi rozličnými oddielami v rámci mozgu. Jednotlivé oddiely môžu byť tvorené buď jednotlivými neurónmi, populáciou neurónov, alebo anatomicky oddelenými oblasťami mozgu. Vzor konektivity je potom tvorený štrukturálnymi spojmi ako sú synapsie, mozgové vlákna, alebo je vzor reprezentovaný štatistickými, alebo kauzálnymi vzťahmi myslené ako súvislosti, krížové vzťahy alebo toky informácií. Nervová aktivita je teda obmedzená mozgovou konektivitou. Tá nám v konečnom dôsledku objasňuje, ako nervy a nervové siete spracovávajú informácie [23].

Myšlienka, že nervový systém je sieť vzájomne prepojených neurónov má dlhú a významnú históriu v neurovede. Anatomické štúdie mozgovej bunkovej štruktúry, bunkových obvodov dlhých vláknových systémov priniesli značné množstvo detailných informácií o usporiadaní štruktúry mozgu. Aktuálna snaha mapovať ľudský mozog so zvyšujúcou sa presnosťou a rozlíšením nadobudla v poslednej dobe nové rozmery. Technologický pokrok v neinvazívnom neuro-monitorovaní otvára nové smery študovania štruktúr a funkcie ľudského mozgu [8], [3].

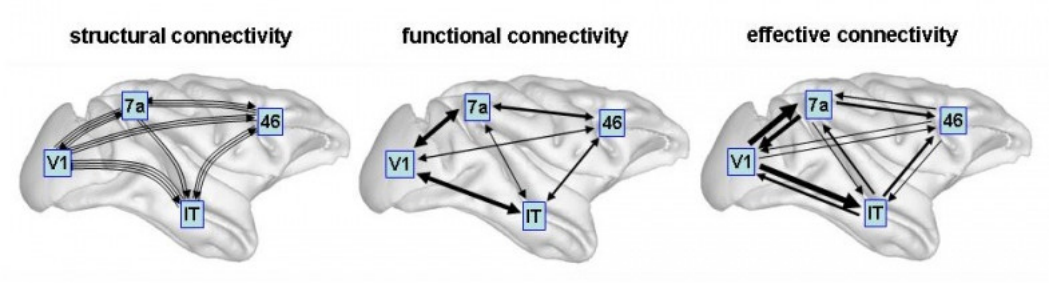
Mozgovú konektivitu možno študovať a analyzovať širokou škálou sieťových analytických postupov. Mnohé z nich je možné aplikovať aj na mapovanie a popisovanie iných biologických sietí, napr. bunkový metabolizmus, génovú reguláciu, ekológiu a iné. **Teória grafov,** konkrétne teória orientovaných grafov má osobitný význam pri spracovávaní štrukturálnej, funkcionálnej aj efektívnej konektivite na všetkých úrovniach. Grafy pozostávajú z vrcholov (korešpondujúce s neurónmi alebo mozgovými regiónmi) a hrán (korešpondujúce so synapsiami alebo mozgovými vláknami). V najjednoduchšej forme môže byť graf popísaný maticou spojení alebo maticou susedností s binárnymi prvkami, ktoré popisujú prítomnosť alebo neprítomnosť hrany medzi pármi vrcholov. Vrcholy môžu byť prepojené buď priamo, alebo nepriamo cez spojenia vytvorené viacerými hranami. Efektivita týchto nepriamych spojení potom závisí od celkovej dĺžky spojenia. Za vzdialenosť medzi dvomi vrcholmi sa počíta najkratšia vzdialenosť medzi nimi. Celkový priemer všetkých vzdialeností sa nazýva **charakteristická dĺžka cesty**. V týchto grafoch môžu byť ďalej skúmané stupne vrcholov, podgrafy, zhlukové koeficienty, stupne korelácií, dĺžky ciest atď. popri iných metrikách z teórie grafov. V mnohých prípadoch vyžaduje štatistické vyhodnotenie návrh odpovedajúcej nulovej hypotézy[7], vrátane vhodnej voľby náhodných grafových modelov. Ďalšiu zaujímavú skupinu spracovávania grafov tvoria zhlukové techniky, ktoré sú aplikovateľné na všetky typy dátových množín mozgovej konektivity. Existuje množstvo zhlukovacích algoritmov, vrátane tých, ktoré sú založené na analýze hlavných komponent (PCA[8]) alebo multidimenzionálnom škálovaní (MDS[9]). Okrem techník používaných v teórii grafov na skúmanie topologických funkcií mozgových sietí existujú aj metódy, ktoré sa

[7]Nulová hypotéza (H_0) – Tvrdenie, ktoré obvykle vyjadruje „žiadny alebo nulový rozdiel“ medzi testovanými súbormi dát. Alternatívna hypotéza (H_1) potom popiera túto hypotézu.

[8]Principal Component Analysis – metóda slúžiaca na dekoreláciu dátových množín, uľahčuje jej skúmanie a vizualizáciu. Používa sa napr. v teórii rozpoznávania, kompresii atď.

[9]Multidimensional scaling – prostriedok na vizualizáciu miery podobnosti jednotlivých prvkov množiny. Používa sa na redukciu dimenzionality dát.

zameriavajú na 3-dimenzionálnu štruktúru mozgovej konektivity. Tieto techniky zahrňujú morfometrické metódy, napr. na skúmanie miery dĺžky vláken alebo ich objem [29].



Obr. 3.4: Vzory konektivity [23].

3.2.1 Štrukturálne mozgové siete

Štrukturálna konektivita popisuje anatomické spojenia množinou nervových prvkov. V ľudskom mozgu sa týmito spojeniami myslia hlavne spojenia bielej hmoty s kortikálnymi a subkortikálnymi regiónmi. V štúdiach ľudského neuro-monitorovania je štrukturálnou mozgovou konektivitou obvykle myslená množina neorientovaných spojení, keďže smer aktuálne nie je možné spoľahlivo rozoznávať.

Analýzy vzorov štrukturálnej mozgovej konektivity umožňujú kvantifikáciu širokej škály sieťových charakteristík. Výsledky ukazujú, že mozgová kôra sa skladá z husto vzájomne spárovaných mozgových oblastí, ktoré sú globálne prepojené. Tieto vzory konektivity sú buď úplne pravidelné, alebo úplne náhodné. V konečnom dôsledku ide o kombináciu aspektov oboch týchto extrémov [23].

3.2.2 Funkcionálne a efektívne mozgové siete

Štúdie vzorov funkcionálnej konektivity (založené na súvislostiach alebo koreláciach) medzi mozgovými regiónmi ukázali, že funkcionálne mozgové siete preukazujú atribúty „malého sveta[10]“ reflektujúce podstatu organizácie anatomických spojení. Detailnejšia analýza teórie grafov funkcionálnej mozgovej konektivity pomáha identifikovať funkcionálne centrá, ktoré sú husto poprepájané a sú charakteristické zvýšeným tokom informácií.

Efektívna mozgová konektivita bola skúmaná pomocou rôznych techník. Kovariančné modelovanie umožnilo identifikáciu význačných rozdielov v efektívnej konektivite medzi danými množinami mozgových regiónov. Grangerova kauzalita bola na EEG aplikovaná taktiež ako aj fMRI a poskytla informácie o orientovaných interakciách medzi nervovými elementami v behaviorálnych a kognitívnych úlohách. Kombinácia transkraniálnej magnetickej stimulácie (TMS[11]) s funkcionálnym neuromonitorovaním umožňuje použitie lokalizovaných odchýliek mozgových sietí kým sú zamestnané vykonávaním nejakej úlohy. Napríklad kombinácia TMS a EEG s vysokou hustotou elektród odhalili prekvapujúce zmenšenie rozsahu

[10] Sieť malého sveta (small-world network) – typ matematického grafu, kde medzi uzlami sa nachádza relatívne málo hrán, ale všetky uzly sú napriek tomu dobre dostupné z hociktorého iného uzlu

[11] Transcranial magnetic stimulation – neinvazívna metóda používaná na stimuláciu malých mozgových regiónov

v oblasti kortikálnej efektívnej konektivity počas non-REM[12] fázy spánku v porovnaní s prebúdzaním [23].

3.2.3 Vzťahy medzi štrukturálnou, funkcionálnou a efektívnou konektivitou

Vzťahy medzi štrukturálnou, funkcionálnou a efektívnou konektivitou v mozgovej kôre predstavujú v dnešných dňoch dôležitú výzvu v neurovede. Dva potenciálne princípy, ktoré spájajú tieto rozdielne postupy skúmania mozgovej konektivity sú segregácia a integrácia. Segregácia sa odkazuje na existenciu špecializovaných neurónov a mozgových regiónov organizovaných do oddelených neurónových populácií a spojených dokopy, aby tvorili segregované kortikálne oblasti. Komplementárny princíp – integrácia, vyhľadáva koordinovanú aktiváciu distribuovaných neurónových populácií, teda umožňuje objavenie súvislých kognitívnych a behaviorálnych stavov. Súhra segregácie a integrácie v mozgových sieťach generuje informácie, ktoré sú simultánne vysoko odlišné a vysoko integrované.

Aplikácia sieťových analyzovacích techník umožňuje porovnávanie vzorov mozgovej konektivity získaných zo štrukturálnych a funkcionálnych štúdií. Napríklad objavenie atribútov *malého sveta* vo vzoroch funkcionálnej konektivity odvodených od fMRI, EEG a MEG[13] štúdií vyvoláva otázku, ako čo najpresnejšie mapovať funkcionálne spojenia na štrukturálne spojenia. Objavuje sa taký návrh pohľadu, kde sú štrukturálne vzory konektivity naozaj veľkým omedzením dynamiky kortikálnych obvodov a systémov, ktoré sú ukryté pod funkcionálnou a efektívnou konektivitou. Je pravdepodobné, že aspoň niektoré štrukturálne charakteristiky mozgových regiónov sa odrážajú v jej funkcionálnych interakciách. Napríklad oblasti štrukturálnych centier by mali udržiavať väčší počet funkcinálnych vzťahov. Výpočtový model štruktúry mozgovej kôry vo veľkej mierke ukázal čiastočnú korešpondenciu medzi štrukturálnymi a funkcionálnymi centrami aj v menšej časovej mierke (v rádoch milisekúnd až sekúnd). V budúcnosti bude pravdepodobne v skúmaní zahrnutá aj paralelná analýza štrukturálnej konektivity ľudského mozgu a vzorov funkcionálnej a efektívnej konektivity zaznamenanej pri rôznych podmienkach [23], [11].

[12] Non-rapid eye movement – fáza spánku, pri ktorej oči nie sú v žiadnom alebo len vo veľmi malom pohybe
[13] MEG – magnetoencefalografia

Kapitola 4

Analýza metód konektivity

Bolo použité množstvo rozličných metód na charakterizáciu funkcionálnej konektivity medzi uzlami v sieťach rozličných foriem (EEG, fMRI,...) Odkedy rozličné metriky funkcionálnej analýzy podávajú rozličné výsledky pre rovnakú dátovú množinu, je dôležité vedieť kedy a ako môžu byť použité. Toto určenie môže byť niekedy veľmi obtiažné, preto hrajú v tomto kontexte dôležitú rolu simulované dáta. Výmena informácií sa objavuje na viacerých úrovniach v mozgu, od interakcií medzi proteínmi v bunkách až po komunikáciu medzi kortikálnymi regiónmi. Informácia môže byť kódovaná viacerými spôsobmi (chemicky, elektricky) a meraná róznymi metódami. Pri odhadovaní funkcionálnej konektivity boli posudzované a testované rôzne metódy založené na fMRI simulovaných signáloch, matematických modeloch a modeloch nervovej hmoty. Tieto metódy ďalej obsahujú metódy postavené na korelácii, koherencii, vzájomnej informácii, prenose entropie, orientovanej koherencii, Grangerovej kauzalite, generalizovanej synchronizácii a Bayesovských sieťových metódach. Metódy pre vyhodnocovanie efektívnej konektivity obsahujú dynamické kauzálne modelovanie (DCM) a štrukturálne modelovacie rovnice (SEM) [25].

Uvažujeme metriky konektivity z nasledujúcich rodín:

- Korelácia,
- Koherencia,
- Grangerova kauzalita,
- Nelineárna korelácia,
- Vzájomná informácia,
- Prenos entropie.

4.1 Prehľad metód

Prehľad metód je uvedený v tabuľke (4.1). Jedná sa o výber metód, ktoré sa aktuálne používajú na analýzu mozgovej konektivity. Ďalej je uvedená špecifikácia jednotlivých metód.

		Bez modelu	S modelom
Čas	Lineárny	Korelácia	Granger
		CorrU	GC
		CorrD	CondGC
	Nelineárny	Nelineárna korelácia	Prenos entropie
		NCorr	TEU
		Vzájomná informácia	TED
		MITU	
		MITD	
Frekvencia	Lineárna	Koherencia	
		CohF	
		CohW	

Tabuľka 4.1: Prehľad metód – čiernym písmom sú označené metódy neorientovanej konektivity, červenou farbou metódy orientovanej konektivity [25].

4.1.1 Korelácia (Corr)

Metódy z rodiny korelácie sú založené na Pearsonovom korelačnom koeficiente. Budeme rozlišovať dva druhy korelačných metód: CorrD a CorrU. Sufixy *D (direct)* a *U (undirect)* označujú orientovanú, resp. neorientovanú konektivitu.

Uvažujme, že máme množinu n simultánnych meraní intrakraniálnej mozgovej aktivity. Jednotlivé merania označme $x_i(t)$, kde index $i \in \{1,2,...,N\}$ označuje i–tú stopu a $t \in \{1,2,...,T\}$ označuje časový priebeh. Maticu funkcionálnej konektivity medzi dvomi signálmi i a j definujeme ako maticu Pearsonových korelačných koeficientov medzi $x_i(t)$ a $x_j(t)$:

$$C_{ij} = \frac{\sum_{t=1}^{T}(x_i(t) - \overline{x_i})(x_j(t) - \overline{x_j})}{\sqrt{\sum_{t=1}^{T}(x_i(t) - \overline{x_i})^2}\sqrt{\sum_{t=1}^{T}(x_j(t) - \overline{x_j})^2}}, \tag{4.1}$$

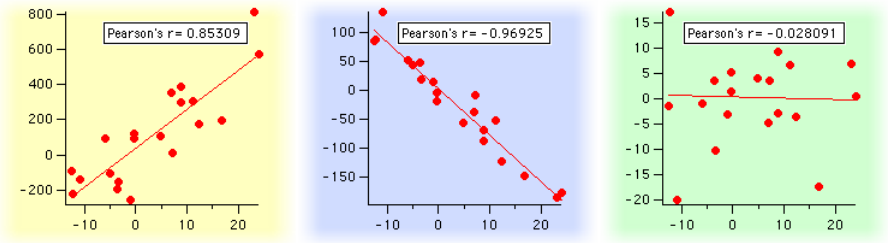
kde $\overline{x_i}$ je priemerná hodnota veličiny $x_i(t)$ vypočítaná ako $\overline{x_i} = \frac{1}{T}\sum_{t=1}^{T} x_i(t)$. Analogicky vypočítame priemernú hodnotu $\overline{x_j}$. Výsledná hodnota koeficientu C_{ij} leží v rozmedzí -1 až 1 vrátane. Tento koeficient reflektuje stupeň lineárnej závislosti dvoch dátových množín. Hodnota $+1$ znamená, že medzi dvomi množinami je absolútna pozitívna lineárna závislosť. Hodnota -1 znamená, že medzi dvomi množinami je absolútna negatívna lineárna závislosť. Hodnota 0 znamená, že medzi dvomi množinami nie je žiadna lineárna závislosť. Obr. (4.1) ukazuje príklady Pearsonových koeficientov [19], [2].

4.1.2 Koherencia (Coh)

Koherencia počíta konektivitu založenú na vzájomných koreláciach vo frekvenčnej doméne. CohF využíva Fourierovú transformáciu, CohW vlnovú (Waveletovu) transformáciu.

Majme dva signály x a y. Funkciu koherencie definujeme nasledovne:

$$C_{xy}(\omega) = \frac{P_{xy}(\omega)}{\sqrt{P_{xx}(\omega)P_{yy}(\omega)}} \tag{4.2}$$



Obr. 4.1: Príklady Pearsonových koeficientov [28].

kde P_{xx} a P_{yy} sú výkonové spektrá[1] signálov x a y a P_{xy} je vzájomné výkonové spektrum[2] týchto signálov, ω je frekvencia. Výpočet týchto spektier je definovaný nasledovne:

$$P_{xx}(\omega) = |\hat{x}(\omega)|^2 = \hat{x}(\omega)\overline{\hat{x}(\omega)} \tag{4.3}$$

$$P_{xy}(\omega) = \hat{x}(\omega)\overline{\hat{y}(\omega)} \tag{4.4}$$

kde $\overline{x}$ je komplexne združený signál x a

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt \tag{4.5}$$

je Fourierova transformácia.
Namiesto Fourierovej transformácie možno použiť aj vlnovú transformáciu:

$$\hat{x}(s,\tau) = \int_{-\infty}^{\infty} x(t)\psi^*_{s,\tau}(t)dt \tag{4.6}$$

kde ψ je tzv. materská vlnka a $\psi_{s,\tau}$ jej roztiahnutie(s) a posunutie v čase (τ), ψ^* je komplexne združená funkcia k ψ [9].

4.1.3 Grangerova kauzalita (GC)

Metódy Grangerovej kauzality počítajú orientovanú konektivitu (j → i) založenú na myšlienke, že minulosť informácie j pomáha predikovať budúcnosť informácie i s väčšou presnosťou, ako keby sa uvažovala iba minulosť informácie i. Uvažujeme dve metódy analýzy Grangerovej kauzálnej konektivity: GC a CondGC.

Uvažujme dva stochastické procesy $X_1(t)$ a $X_2(t)$. Budúce hodnoty procesu $X_1(t+1)$ môžu byť predpovedané dvomi rozdielnymi dátovými množinami:

1. použitím iba minulých hodnôt $X_1(t)$,
2. použitím minulých hodnôt $X_1(t)$ i $X_2(t)$ (prípadne i $X_3(t)priCondGC$).

[1] Power spectrum
[2] Cross-power spectrum

Uvažujme, že X_1 a X_2 sú reprezentované jednotlivými autoregresívnymi modelmi:

$$X_1(t) = \sum_{j=1}^{m} a_j X_1(t-j) + \varepsilon_{11}(t) \tag{4.7}$$

$$X_2(t) = \sum_{j=1}^{m} b_j X_2(t-j) + \varepsilon_{22}(t) \tag{4.8}$$

Vzájomný prediktor $X_1(t)$ môže byť definovaný ako:

$$X_1^*(t) = \sum_{j=1}^{m} a_j^* X_1(t-j) + \sum_{j=1}^{m} b_j^* X_2(t-j) + \varepsilon_{12}(t) \tag{4.9}$$

Potom, ak odchýlka predikčnej chyby $\delta_{12}^2(\varepsilon_{12})$ je menšia ako odchýlka $\delta_{12}^2(\varepsilon_{11})$, tak to značí kauzálnu interakciu z $X_2(t)$ do $X_1(t)$. Rozmer tejto kauzality je definovaný ako $F_{x_2 \to x_1} = ln\left(\frac{\delta_{12}^2}{\delta_1^2}\right)$ Ak $\delta_1^2 = \delta_{12}^2$, potom je veľkosť kauzality z X_2 do X_1 rovná 0. Podobne definujeme aj kauzalitu opačného smeru, teda $F_{x_1 \to x_2}$ z X_1 do X_2.

Ďalej môžme uvažovať, že máme viac ako dve množiny časovo menných dát. Ak X_1 kauzálne ovplyvňuje X_2 a X_2 ďalej kauzálne ovplyvňuje X_3, tak potom X_1 nepriamo kauzálne opvlyvňuje X_3. Priama a nepriama kauzalita medzi dvomi časovo mennými množinami môže byť definovaná *podmienenou Grangerovou kauzalitou:* $F_{x_1 \to x_2|x_3} = ln\left(\frac{\delta_{1,3}^2}{\delta_{1,2,3}^2}\right)$[5].

4.1.4 Nelineárna korelácia (NCorr)

Pre množinu dvoj-dimenzionálnych dát $\{(x_i,y_i)|i \in 1,...,n\}$, kde x a y sú množiny dát prevažne v nelineárnom vzťahu, môžme použiť napr. metódu najmenších štvorcov alebo použiť iné spôsoby pre získanie korešpondujúceho lineárneho regresného modelu. Viď obrázok (4.2), ktorý prezentuje nelineárnu regresiu pomocou metódy najmenších štvorcov.

Uvažujme, že nelineárny matematický model množiny $\{(x_i, y_i)\}$ získaný metódou najmenších štvorcov je $\hat{y} = f(x)$. Ak tento model dostatočne zodpovedá testovacej množine dát, potom musí pre množinu $\{(y_i,\hat{y}_i)|i \in 1,...,n\}$ existovať lineárny regresný model:

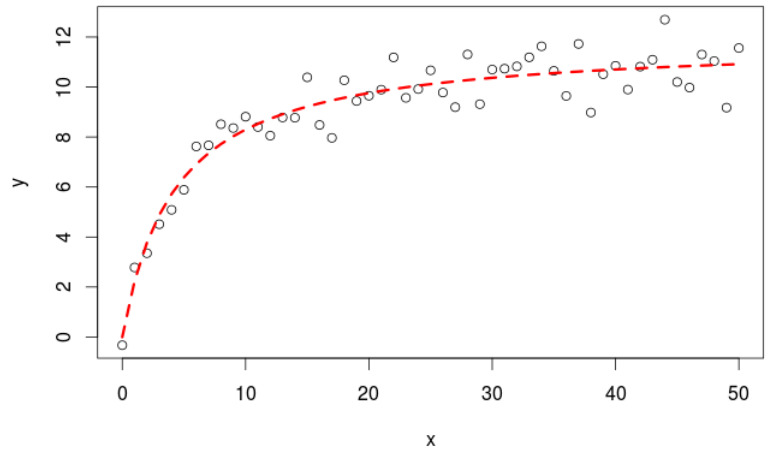
$$\hat{z} = A + By$$

Potom podobne ako v rovnici (4.1), pri substitúcii:

$$\{(y_i,\hat{y}_i)|i \in 1,...,n\} \equiv \{(y_i,z_i)|i \in 1,...,n\}$$

odvodíme vzťah pre získanie korelačného koeficientu r [26]:

$$r = \frac{\sum_i (y_i - \overline{y})(z_i - \overline{z})}{\sqrt{\sum_i (y_i - \overline{y})^2}\sqrt{\sum_i (z_i - \overline{z})^2}} \tag{4.10}$$



Obr. 4.2: Nelineárna regresia metódou najmenších štvorcov [10].

4.1.5 Vzájomná informácia (MIT)

Vzájomná informácia je metrika odvodená od teórie informácie, ktorá kvantitatívne určuje informácie získané pozorovaním dvoch náhodných veličín. Meria lineárne a nelineárne závislosti, ktoré môžu existovať medzi dvomi dátovými množinami. MITD je metóda pre orientovanú konektivitu a MITU pre neorientovanú konektivitu.

Uvažujme, že $X = x_i$ a $Y = y_j$ reprezentujú dve množiny náhodných pozorovaní s distribúciou pravdepodobností $P_X(x_i)$ a $P_Y(y_j)$. Priemerné množstvo informácie získanej meraním X zo Shannonovej informačnej teórie je:

$$H(X) = -\sum_{x_i} P_X(x_i) \log P_X(x_i) \tag{4.11}$$

Ďalej vyjadríme vzťah pre podmienenú neurčitosť veličiny X danú náhodnou veličinou Y:

$$\begin{aligned} H(X|Y) &= \sum_{y_j} P_Y(y_j) H(X|Y = y_j) = \\ &= -\sum_{x_i,y_j} P_{XY}(x_i, y_j) \log\left(\frac{P_{XY}(x_i,y_j)}{P_Y(y_j)}\right) = \\ &= H(X,Y) - H(Y) \end{aligned} \tag{4.12}$$

kde $H(X,Y) = -\sum_{x_i,y_j} P_{XY}(x_i, y_j) \log[P_{XY}(x_i, y_j)]$.
Vzájomná informácia (MI) je potom daná:

$$MI(X,Y) = H(X) - H(X|Y) = H(X) + H(Y) - H(X,Y) \tag{4.13}$$

Zo vzťahov (4.11) a (4.12) potom môžme vyjadriť vzájomnú informáciu dvoch veličín X a Y ako:

$$MI(X,Y) = \sum_{x_i,y_j} P_{XY}(x_i,y_j) log\left(\frac{P_{XY}(x_i,y_j)}{P_X(x_i)P_Y(y_j)}\right) \tag{4.14}$$

Vzájomná informácia je symetrická, platí teda: $MI(X,Y) = MI(Y,X) \geq 0$ Hodnota 0 znamená, že medzi veličinami X a Y nie je žiadna závislosť. Maximálna hodnota, ktorá môže byť získaná, sa rovná vlastnej hodnote jedného zo signálov. Existujú aj také verzie tejto metódy, ktoré normalizujú výsledné hodnoty do intervalu $\langle 0,1 \rangle$ [20].

4.1.6 Prenos entropie (TE)

Entropia je fyzikálna veličina, ktorá meria mieru neurčitosti (náhodnosť, neusporiadanosť, neporiadok). Metódy založené na prenose entropie sú menej časovo náročné ako metódy Grangerovej kauzality. BTED počíta orientovanú konektivitu, BTEU neorientovanú konektivitu. Prenos entropie je metrika teórie informácie odvodená od metriky vzájomnej informácie. Prenos entropie rozširuje túto metriku o možnosť odhadu podmienenej pravdepodobnosti medzi dvomi procesmi meniacimi sa v čase.
Prenos informácie z Y do X ($T_{Y \to X}$) vyjadríme nasledovne:

$$T_{Y \to X} = \sum_{t} p(x_{t+1}, x_t, y_t) \log \left(\frac{p(x_{t+1}|x_t, y_t)}{p(x_{t+1}|x_t)} \right) \tag{4.15}$$

kde index t označuje priebeh v čase a $p(x|y)$ označuje podmienenú pravdepodobnosť javu x za predpokladu, že nastal jav y.
Na rozdiel od vzájomnej informácie pri prenose entropie neplatí symetrickosť [16]:

$$T_{Y \to X} \neq T_{X \to Y}$$

4.2 Porovnanie metód

Uviedli sme si metódy, ktoré sa používajú na vyhodnotenie väzieb medzi signálmi EEG. Teraz ich zhodnotíme podľa toho, akú informáciu prenášajú a aká je ich výpočtová náročnosť.

Metódy z rodiny Corr nám poskytnú výsledok vo forme reálnych koeficientov v rozsahu $\langle -1, 1 \rangle$. Nás budú zaujímať najmä kladné koeficienty, ktoré určujú mieru závislosti dvoch signálov. Záporné hodnoty udávajú taktiež mieru závislosti dvoch signálov, ale koeficient -1 dosiahneme iba v prípade, ak bude jeden zo signálov invertovaný. V prípade, že budeme pri porovnávaní signálov uvažovať aj časové posuvy, bude výsledkom množina korelačných koeficientov, z ktorej môžeme skonštruovať nový signál o rozmere definovaného časového úseku. Výhodou tejto metódy je jednoduchosť implementácie ako aj jednoduchosť interpretácie výsledkov.

Koherenčné metódy (Coh) poskytujú podobne ako metódy rodiny Corr výsledky v podobe reálnych koeficientov, tento raz však v intervale $\langle 0,1 \rangle$, kde vyššia hodnota koeficientu značí väčšiu podobnosť spracovávaných signálov. Nevýhodou oproti korelačným metódam je nutná transformácia signálov pomocou Fourierovej alebo vlnovej transormácie (časť 4.1.2).

Metódy nelineárnej korelácie majú rovnaký výstup ako metódy rodiny Corr – reálne koeficienty v intervale $\langle -1,1 \rangle$. Získame teda rovnakú alebo podobnú množinu informácií. Výpočet nelineárnych korelačných koeficientov je však náročnejší ako pri korelačných metódach, čo potvrdzujú aj výsledky výpočtovej náročnosti (časť 4.3). Tento fakt spôsobuje predovšetkým nutnosť tvorby lineárneho regresného modelu a následný výpočet Pearsonových korelačných koeficientov.

Doteraz uvedené metódy poskytuju pomerne jednoducho interpretovateľné výsledky. Vždy sme získali výsledky vo forme reálnych koeficientov, ktoré značili mieru závislosti

medzi signálmi. Pri metódach rodiny MIT to už také jednoduché nie je. Vzájomná informácia určuje, aké množstvo informácie získané z jednej náhodnej veličiny obsahuje druhá náhodná veličina. Vyššia hodnota znamená väčšiu závislosť, nižšia hodnota menšiu závislosť a hodnota nula úplnú nezávislosť. V závislosti na zvolenom základe logaritmu pri výpočte (viď rovnica 4.14) potom dostaneme výstup vo forme bitov ($\log_2$), natov ($\ln_e$) alebo banov ($\log_{10}$)[3]. Aby sme rozumeli výsledkom tejto metódy, musíme si definovať pojmy *entropia* a *podmienená entropia*. Entropia ($H(X)$) je miera neurčitosti. Čím vyššia je entropia, tým vyššia je miera neurčitosti medzi náhodnými veličinami [15]. Môžeme ju vypočítať podľa vzťahu uvedenom v rovnici (4.11). Podmienená entropia ($H(X|Y)$) je priemerná miera neurčitosti náhodnej veličiny X určená pozorovaním druhej náhodnej veličiny Y. Vypočítame ju podľa rovnice (4.12). Pri metódach prenosu entropie (TE) dostávame výsledky v podobnej forme ako pri metódach MIT. Nevýhodou týchto metód je nutnosť tvorby pravdepodobnostného modelu a komplikovaná interpretácia výsledkov.

Grangerova kauzalita je efektívny spôsob ako analyzovať kauzálne pôsobenie medzi modelmi. Keďže rozmer kauzality môžme vypočítať pomocou prirodzeného logaritmu (viď časť 4.1.3), výslednú silu kauzálneho pôsobenia medzi dvomi modelmi dostaneme v intervale $(0, \infty)$, kde hodnota nula značí nezávislosť modelov. Odhadnúť však do akej miery dané signály spolu korelujú je dosť obtiažne, lebo výsledné hodnoty sa teoreticky môžu blížiť k nekonečnu. Spoľahlivo môžeme tvrdiť len to, že nižšia hodnota znamená menšiu vzájomnú podobnosť. V teste výpočtovej náročnosti skončila rodina týchto metód na poslednom mieste, preto nie je veľmi vhodné používať tieto metódy pri vyhodnocovaní funkcionálnej konektivity a nebudeme sa teda zaoberať jej optimalizáciou.

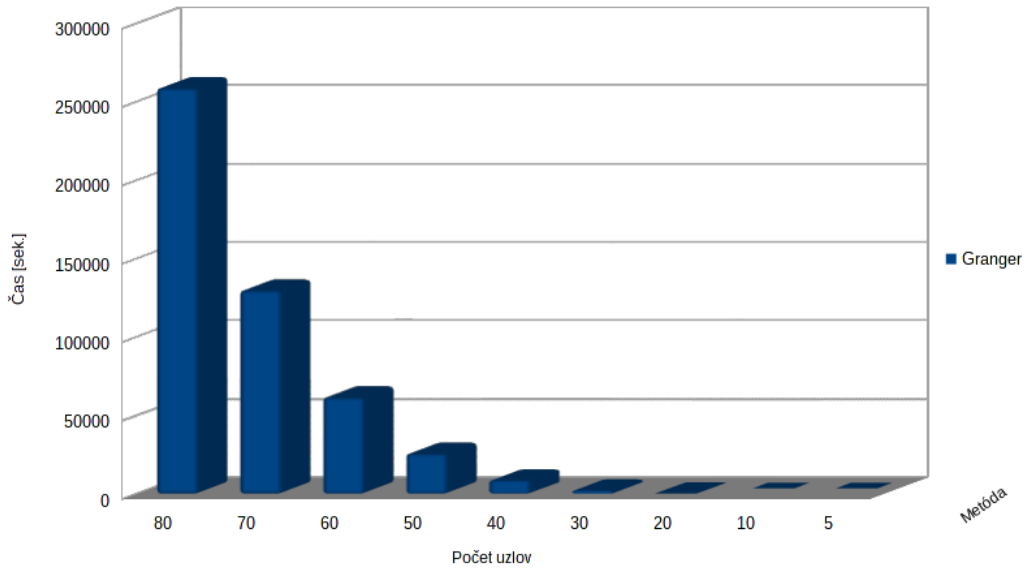
4.3 Analýza časovej náročnosti

Výpočtová náročnosť môže byť limitujúcim faktorom pri vykonávaní analýzy mozgových sietí, najmä pokiaľ sa objem týchto sietí zväčšuje. Meraný bol výpočtový čas každej rodiny metód na procesore Intel Xeon E5–2690 (8 jadier, 2.90 GHz). Počet uzlov sa postupne zvyšuje od 5 do 80. Obrázok (4.3) zobrazuje výpočtový čas v sekundách pre rodinu metód GC. Na obr. (4.4) je potom zobrazený výpočtový čas pre ostatné analyzované metódy. Časová zložitosť pre GC je $O(n^7)$. Pre metódy z ostatných rodín je časová zložitosť $O(n^2)$, kde n je počet uzlov. Časová zložitosť uvedených metód môže byť zoradená od najnižšej po najvyššiu: Corr, MIT, TE, Coh, NCorr, GC.

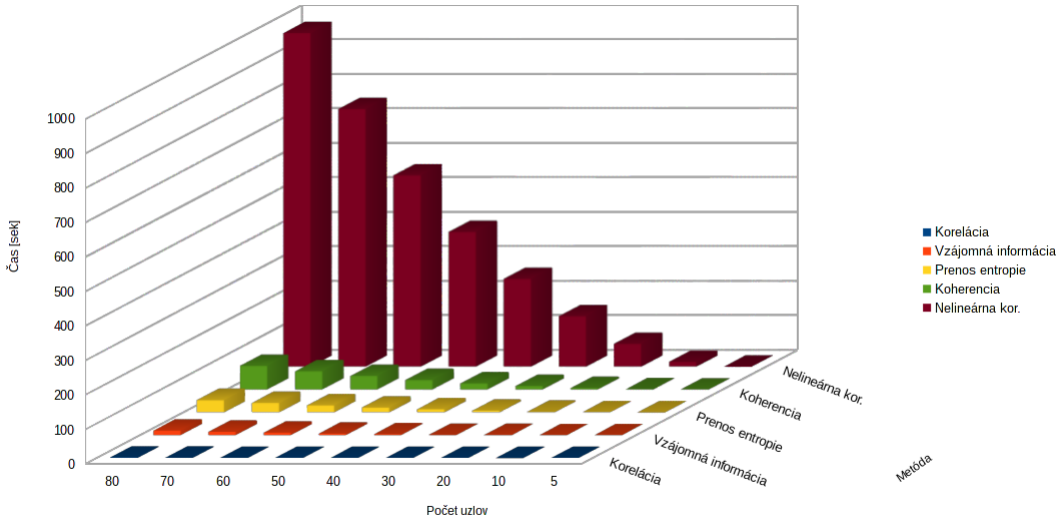
Metóda → Počet uzlov ↓	**CORR**	**MIT**	**TE**	**COH**	**NCORR**	**GC**
80	1,8084	13,79	36,285	69,428	968,87	258852,96
70	1,4014	10,65	28,05	53,682	748,57	129912,94
60	1,0456	7,91	20,855	39,936	556,27	61624,480
50	0,64	5,00	14,7	28,19	391,97	25746,9
40	0,3956	3,07	9,585	18,444	255,67	8693,92
30	0,2164	1,67	5,51	10,698	147,37	2135,26
20	0,0884	0,67	2,475	4,952	67,07	294,24
10	0,0116	0,07	0,48	1,206	14,77	10,18
5	0,1	0,004	0,11	0,5	0,56	0,002

Tabuľka 4.2: Testovacie dáta [25].

[3] 1 bit = $\ln_e 2$ natov = $\log_{10} 2$ banov



Obr. 4.3: Výpočtový čas pre Grangerovu metódu [25].



Obr. 4.4: Výpočtový čas pre rôzne rodiny metód konektivity [25].

4.4 Návrh vhodnej metódy

Na základe porovnania jednotlivých metód sa ako najvhodnejšia metóda pre implementáciu ponúka metóda lineárnej korelácie (Corr). Jej výpočet je pomerne jednoduchý a zároveň pomerne rýchly vzhľadom k ostatným metódam. Pri uvažovaní časových posuvov je možné výpočet značne paralelizovať, čo určite ocenia používatelia takejto implementácie. Pri tejto metóde nie je nutná tvorba pravdepodobnostých modelov ani transformácia signálov.

Kapitola 5

Technologická analýza

Určite budeme požadovať, aby výsledná aplikácia bola postavená na pevných technologických základoch s využitím overených technológií. Musíme sa zamerať na to, ktoré technológie pri implementácii použiť, určiť na akých prístrojoch a systémoch bude aplikácia prevádzkovaná a aká skupina užívateľov ju bude používať. Keďže sa nejedná o samostatnú desktopovú aplikáciu ale o rozšírujúci modul už existujúcej aplikácie, je potrebné prispôsobiť tento výber technológií tak, aby technológie boli kompatibilné s pôvodnou aplikáciou SignalPlant.

Samotná aplikácia SignalPlant je implementovaná v jazyku C# s využitím vývojového prostredia *MS Visual Studio*. Aplikácia beží pod operačným systémom Windows 7 a vyšší. Budeme teda pokračovať podobným spôsobom aj pri vývoji nového rozšírenia. Hlavnou požiadavkou na vytvorenú aplikáciu je, aby bola rýchlejšia oproti existujúcim implementáciam a optimalizovaná s využitím maximálneho možného výkonu počítača. To znamená, snažíme sa efektívne urýchliť výpočet rozdelením jednotlivých krokov výpočtu medzi výpočtové jednotky CPU[1] a GPU[2]. Túto funkcionalitu nám poskytnú frameworky *OpenCL*[3] a *CUDA*.

5.1 OpenCL

OpenCL je slobodné, voľne šíriteľné API[4] umožňujúce paralelné výpočty s využitím GPU, CPU a ostatných koprocesorov. Hlavnou výhodou použitia OpenCL je akcelerácia v paralelnom spracovávaní programu. OpenCL využíva všetky výpočtové zdroje (viacjadrové CPU a GPU) ako rovnocenné výpočtové jednotky a prideľuje im rozličné úrovne pamäte [1]. Je možné ho používať na zariadeniach od veľkých výrobcov, ako sú AMD, NVIDIA, IBM a Intel. Existujú väzby na bežne používané programovacie jazyky aj keď samotné OpenCL je postavané na základoch jazyka C so štandardom C99.

OpenCL používa vlastnú terminológiu na popis jednotlivých elementov. Vysvetlíme si základné pojmy, ako sú:

- **Kernel,**
- **Program,**

[1] *Central Processing Unit* – Hlavný procesor počítača – vykonáva inštrukcie alebo dáta programu.
[2] *Graphics Processing Unit* – Procesor slúžiaci na výpočet grafických informácií pre zobrazovacie účely.
[3] OpenCL – The Open Computing Language.
[4] Application Programming Interface – rozhranie pre programovanie aplikácií.

- **Pracovná jednotka (work-item),**
- **Pracovná skupina (work-group),**
- **Pracovný front (work-queue),**
- **Front príkazov (command-queue),**
- **Zariadenie,**
- **Kontext.**

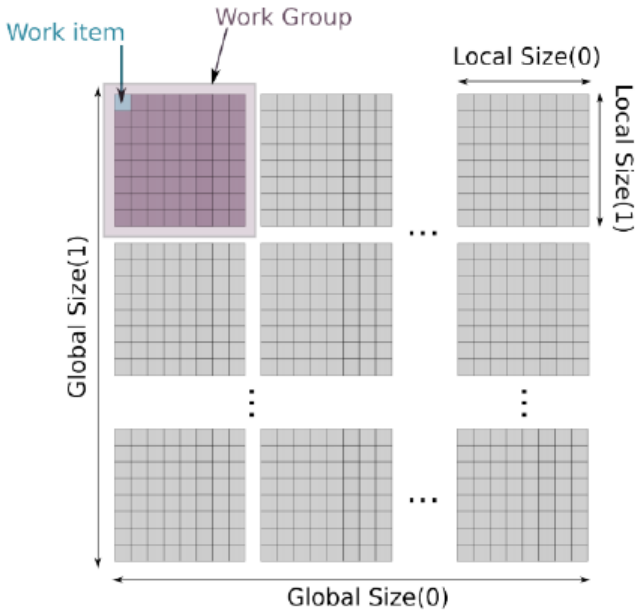
5.1.1 Terminológia

Kernel je funkcia deklarovaná v programe a spustená na niektorom OpenCL zariadení. Kernely sú identifikované kľúčovým slovom `__kernel`.
Program je kolekcia kernelov.
Pracovná jednotka predstavuje vlákno, na ktorom je spustený kernel.
Pracovná skupina je kolekcia pracovných jednotiek. Každá pracovná jednotka patrí do nejakej pracovnej skupiny.
Pracovný front je kolekcia pracovných skupín.
Front príkazov obsahuje príkazy, ktoré sa majú vykonať. Každému zariadenia prislúcha jeden front príkazov.
Zariadenie je súbor výpočtových jednotiek. Takýmto zariadením býva spravidla GPU alebo viacjadrové CPU.
Kontext je prostredie, v ktorom sú spúšťané kernely. Je to taktiež oblasť, kde je definovaná správa pamäti a synchronizácia.

5.1.2 Výpočtový model

OpenCL aplikácie sa skladajú zo sekvenčných častí, ktoré sú spracovávané pomocou CPU a z paralelných častí – známe ako *kernely*. Tieto kernely môžu byť spúšťané na zariadeniach kompatibilných s OpenCL (CPU a GPU), pričom je zabezpečená synchronizácia medzi sekvenčným kódom a kernelmi. V OpenCL je každému nezávislému výpočtovému elementu priradená *pracovná jednotka*. Pracovné jednotky sú ďalej hierarchicky organizované do nezávislých *pracovných skupín*. (viď obr. 5.1) Každej pracovnej jednotke je pridelený unikátny identifikátor v rámci pracovnej skupiny [24], [12], [13].
Spustenie OpenCL prebieha nasledovne:

1. Najskôr CPU hostiteľ definuje výpočtovú N-dimenzionálnu výpočtovú doménu nad nejakým pamäťovým regiónom. Každá zložka tejto domény predstavuje jednu pracovnú jednotku, ktorá spúšťa kernel,
2. Hostiteľ definuje zoskupenie jednotlivých pracovných jednotiek do pracovných skupín, ktoré sú následne umiestnené do *pracovných frontov,*
3. Hardvér následne nahrá dáta z pamäte do GPU a spustí každú pracovnú skupinu z pracovných frontov.



Obr. 5.1: Organizácia pracovných jednotiek do pracovných skupín [24].

5.1.3 Pamäťový model

OpenCL definuje kontext, ktorý zahŕňa hostiteľa, jedno alebo viac zariadení, fronty príkazov a pamäť v rámci kontextu. V jednom kontexte môže byť viacero zariadení, avšak všetky majú prístup k pamäti definovanej hostiteľom. Na jednom zariadení môžu spúšťať paralelný kód viaceré pracovné skupiny, čím pochopiteľne môže dôjsť ku konfliktom vo vyhradenej pamäti. Napokon, v rámci jednej pracovnej skupiny spúšťaju jednotlivé pracovné jednotky svoj kód, čím taktiež môže dochádzať ku konfliktom v pamäti. Z týchto dôvodov bol vytvorený pamäťový model pozstávajúci z nasledujúcich štyroch častí:

1. **Pamäťové regióny** – odlišné pamäte viditeľné hostiteľovi a zariadeniam, ktoré zdieľajú rovnaký kontext,
2. **Pamäťové objekty** – objekty definované OpenCL API a ich spravovanie hostiteľom a zariadeniami,
3. **Zdieľaná virtuálna pamäť** – virtuálny adresový priestor dostupný hostiteľovi a zariadeniam v rámci kontextu,
4. **Konzistenčný model** – pravidlá definujúce, ktoré hodnoty sú dostupné, ak sa viacero jednotiek snaží o prístup do pamäte, plus definovanie obmedzení poradia pamäťových operácií a synchronizácie.

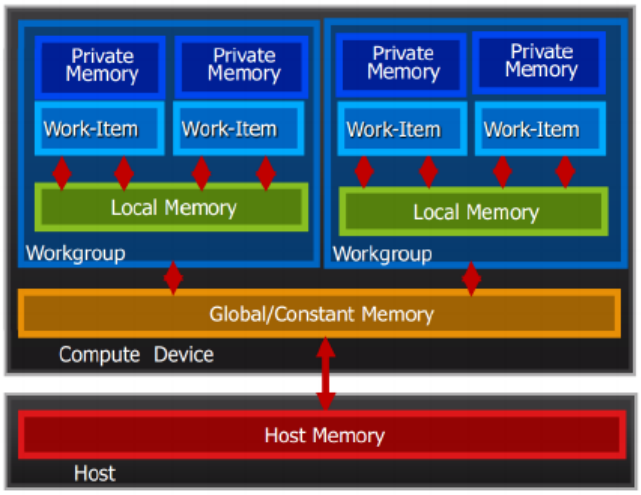
Pamäť zariadenia pozostáva zo štyroch pamäťových regiónov (obr. 5.2):

1. **Globálna pamäť** – umožňuje čítanie/zápis všetkým pracovným jednotkám vo všetkých pracovných skupinách bežiacich na ľubovoľnom zariadení v rámci kontextu,

2. **Konštantná pamäť** – oblasť globálnej pamäte, ktorá zostáva nemenná počas vykonávanie inštancie niektorého z kernelov,

3. **Lokálna pamäť** – pamäťový región prístupný pracovnej skupine. Táto oblasť je použítá na vytvorenie premenných, ktoré sú zdieľané medzi pracovnými jednotkymi v rámci jednej pracovnej skupiny,

4. **Privátna pamäť** – pamäť prístupná jednej konkrétnej pracovnej jednotke. Premenné definované v tejto oblasti nie sú prístupné žiadnej inej pracovnej jednotke [14].



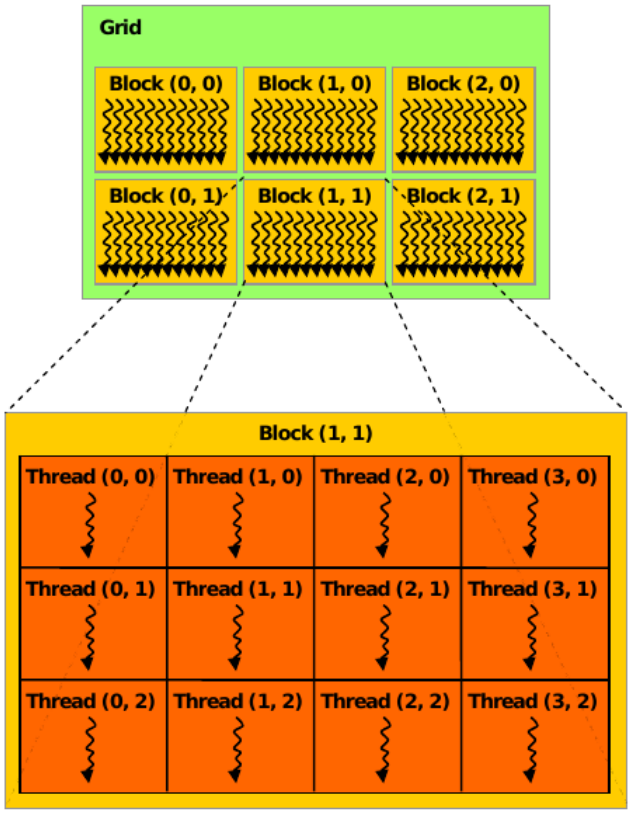
Obr. 5.2: Pamäťový model [24].

5.2 CUDA

CUDA je výpočtová platforma umožujúca masívne paralelné spracovanie dát na grafickom hardvéri. Narozdiel od OpenCL je CUDA podporovaná len na zariadeniach od výrobcu NVIDIA.

5.2.1 Programovací model

Základom pri programovaní aplikácií pre grafický hardvér je opäť *kernel*. Každý kernel je spustený jedným *vláknom (thread)*. Každému vláknu je pridelený unikátny identifikátor, ktorý je prístupný pomocou premennej *threadIdx*. Vlákna sú potom organizované do *blokov (block)*. Každý blok obsahuje dostupný počet vláken na základe grafického hardvéru. Bloky sú ďalej organizované do 1-dimenzionálnej alebo 2-dimenzionálnej *mriežky (grid)*.

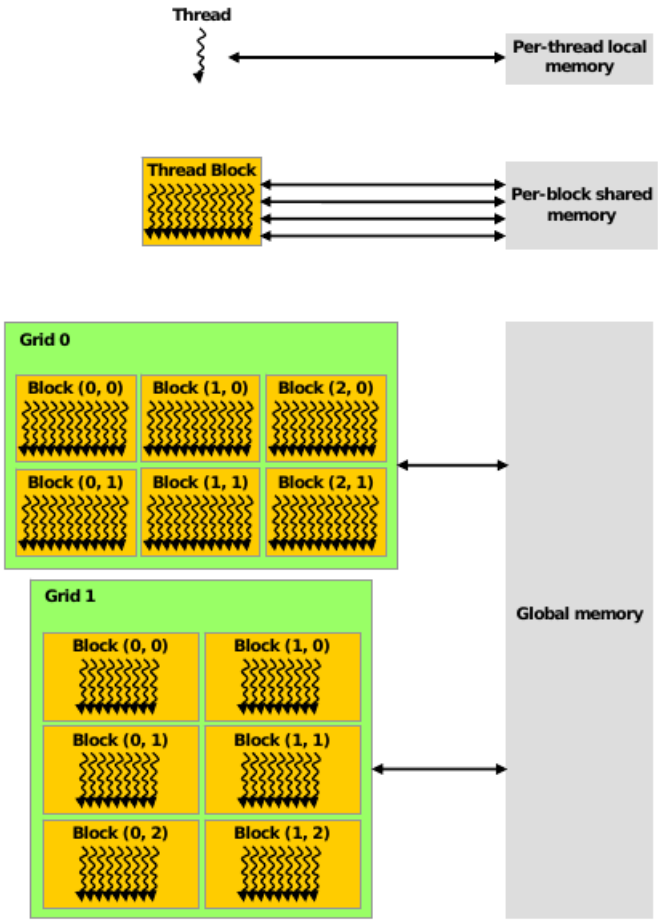


Obr. 5.3: Programovací model [17].

5.2.2 Pamäťový model

Jednotlivé CUDA vlákna môžu počas vykonávania výpočtu pristupovať k rôznym pamäťovým oblastiam:

- **Privátna pamäť** – adresový priestor dostupný konkrétnemu vláknu,
- **Pamäť bloku** – zdieľaná pamäť medzi vláknami daného bloku,
- **Globálna pamäť** – pamäť dostupná všetkým vláknam,
- **Konštantná pamäť** – pamäť umožňujá iba čítanie, nie zápis,
- **Pamäť pre textúry** – pamäť umožňujá iba čítanie, nie zápis. Používaná napr. pri spracovávaní videa.



Obr. 5.4: Pamäťový model [17].

Kapitola 6

Návrh aplikácie

6.1 Formalizácia požiadaviek na aplikáciu

Ešte pred samotným návrhom aplikácie si definujeme, čo vlastne od aplikácie očakávame. A to z hľadiska užívateľského aj technologického. Aplikácia bude rozširujúcim modulom už existujúcej aplikácie SignalPlant. Skupinu užívateľov budú teda tvoriť užívatelia tohto nástroja. SignalPlant je voľne šíriteľný softvér určený na analýzu, skúmanie a spracovávanie signálov. Primárne je však používaný na analýzu EKG a EEG signálov. Je vyvinutý a stále vyvíjaný Ústavom prístrojovej techniky Akadémie vied Českej republiky[1]. Je navrhnutý tak, aby ho bolo možné rozširovať rozširujúcimi modulmi (pluginy) pomocou dynamicky linkovaných knižníc (dll). V súčasnej dobe je možné SignalPlant používať iba na operačnom systéme Windows.

Aplikácia bude počítať Pearsonove korelačné koeficienty medzi dátovými množinami. V našom prípade budú tieto množiny tvorené signálmi, ktoré boli nasnímané pomocou intrakraniálnych elektród. Všeobecne však bude možné spracovávať akékoľvek signály, ktoré je možné spracovávať aplikáciou SignalPlant.

Predpokladajme, že máme n signálov. Budeme teda chcieť vypočítať korelačné koeficienty medzi všetkými signálmi. Definujme si operáciu $\bigotimes$, ktorá vyjaduje korelačný koeficient medzi dvomi signálmi. Môžeme uvažovať, že medzi dvomi signálmi S_1, S_2 platí komutatívnosť: $S_1 \bigotimes S_2 = S_2 \bigotimes S_1$. Koreláciu $S_1 \bigotimes S_1$, teda signál sám so sebou nemá zmysel uvažovať (z matematického hľadiska bude vždy takýto korelačný koeficient = 1). Výsledná množina korelačných koeficientov teda bude mať rozmer $\frac{n^2-n}{2}$.

Pre ilustráciu si uveďme príklad výslednej korelačnej matice pre 5 signálov ($n = 5$). Dostaneme $\frac{5^2-5}{2} = 10$ výsledných korelačných koeficientov pre každú kombináciu signálov.

$$C_{S_1,S_2} = \begin{pmatrix} C_{1,1} & C_{1,2} & C_{1,3} & C_{1,4} & C_{1,5} \\ C_{2,1} & C_{2,2} & C_{2,3} & C_{2,4} & C_{2,5} \\ C_{3,1} & C_{3,2} & C_{3,3} & C_{3,4} & C_{3,5} \\ C_{4,1} & C_{4,2} & C_{4,3} & C_{4,4} & C_{4,5} \\ C_{5,1} & C_{5,2} & C_{5,3} & C_{5,4} & C_{5,5} \end{pmatrix} \Rightarrow \begin{pmatrix} & & & \\ C_{2,1} & & & \\ C_{3,1} & C_{3,2} & & \\ C_{4,1} & C_{4,2} & C_{4,3} & \\ C_{5,1} & C_{5,2} & C_{5,3} & C_{5,4} \end{pmatrix} \tag{6.1}$$

Ďalej budeme požadovať, aby bolo možné každé dva analyzované signály rozdeliť na men-

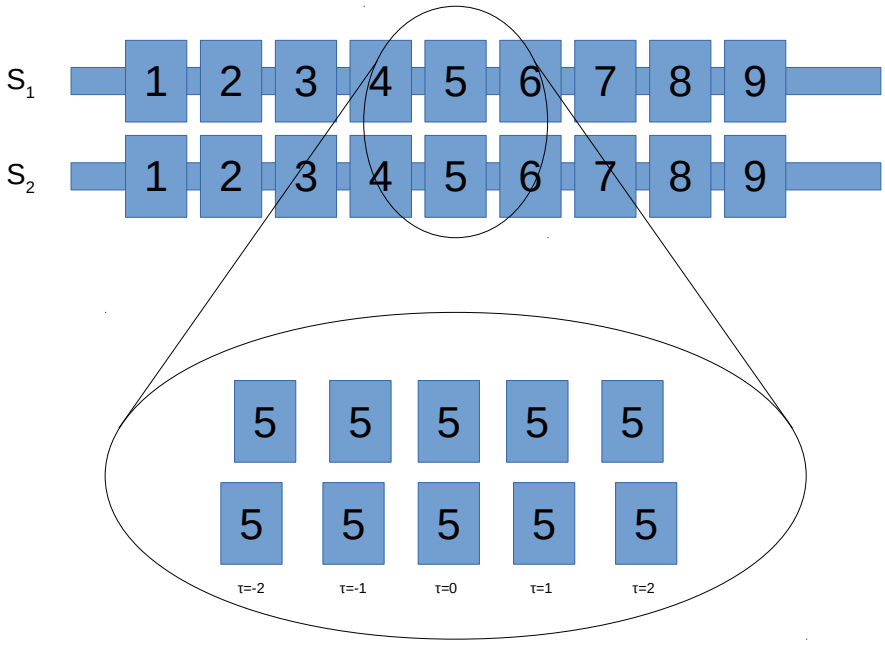
[1] Akademie věd České republiky, Ústav přístrojové techniky – https://www.isibrno.cz/

šie celky – okienka (sliding window). Veľkosť okienka a posun okienka v rámci signálu bude definovať užívateľ. Neporovnávname teda medzi sebou celé signály, ale iba časti signálov definované veľkosťou okienka. Pre dvojicu analyzovaných signálov potom nezískame jednu korelačnú hodnotu, ale množinu korelačých hodnôt, ktorej mohutnosť závisí od veľkosti okienka a definovaného posunu.

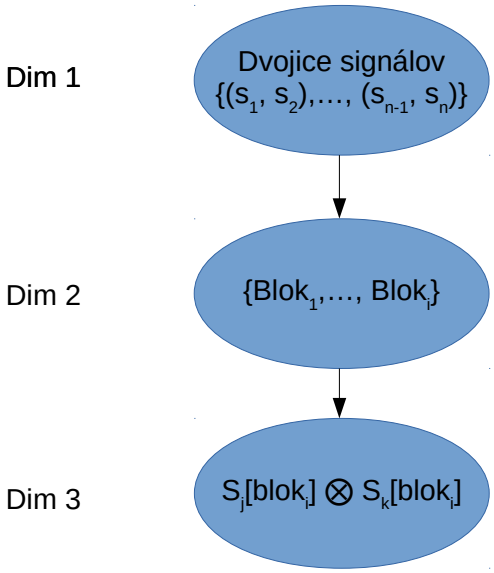
Pri porovnávaní okienok budeme ďalej uvažovať aj ich časové posuvy. Hodnota, o akú sa má daná dvojica okienok medzi sebou posunúť, bude taktiež definovaná užívateľom. Pre každú dvojicu okienok teda nedostaneme jednu korelačnú hodnotu, ale opäť množinu hodnôt, ktorej mohutnosť závisí od definovaného časového posunu. Týmto sa nám situácia značne komplikuje.

Keď si zhrnieme vyššie uvedené požiadavky, zistíme, že sa dostávame do troch dimenzií. Prvú dimenziu tvoria jednotlivé signály, druhú dimenziu tvoria hodnoty získané koreláciou každej dvojice okienok a tretiu dimenziu tvoria hodnoty získané koreláciou okienok v čase. Obrazne je táto situácia znázornená na obrázkoch (6.1) a (6.2). Na obr. (6.3) je znázornená šírka a posuv okienka.

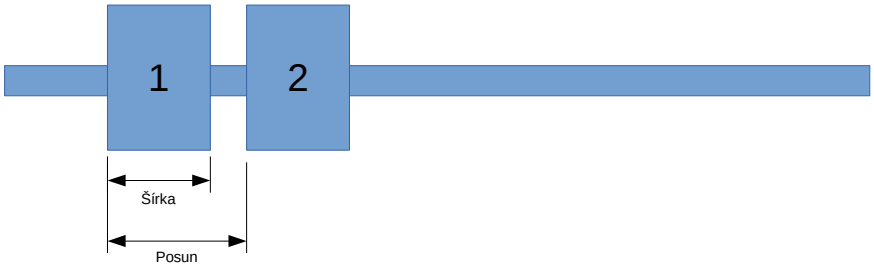
Na získenie korelačných koeficientov z takto definovanej úlohy budeme potrebovať veľké množstvo výpočtových jednotiek. Ak uvažujeme, že procesor priemerného počítača obsahuje štyri jadrá, výpočet môže trvať veľmi dlho. Preto je vhodné výpočet korelačných koeficientov akcelerovať pomocou grafického hardvéru, ktorý ponúka viac paralelných výpočtových jednotiek ako procesor.



Obr. 6.1: Výpočet korelácií – znázornenie výpočtu korelácií medzi signálmi S_1 a S_2. Čísla 1-9 označujú poradie okienok. Hodnota τ udáva, o koľko jednotiek sa má posunúť okienko druhého signálu oproti okienku prvého signálu. Znázornená je situácia výpočtu korelácií v bloku č. 5.



Obr. 6.2: Dimenzie – 1. dimenziu tvoria dvojice signálov, 2. dimenziu tvoria jednotlivé okienka (bloky) každej dvojice signálov, 3. dimenziu tvoria posuvy v rámci každého bloku každej dvojice signálov.



Obr. 6.3: Šírka a posuv okienka.

Pre zhrnutie si uvedieme, aké sú vstupy a výstupy aplikácie:
Užívateľské vstupy:

- Množina signálov,
- Šírka okienka,
- Posuv okienok,
- Časový posuv v rámci okienka.

Výstupy:

- 3-dimenzionálne dáta (korelačné koeficienty).

6.2 Výpočet

Návrh efektívneho výpočtu tvorí kľúčovú časť pri implementácii. Ako sme si uviedli v časti (6.1), budeme potrebovať množstvo výpočtových jednotiek. Aplikácia teda bude akcelerovaná pomocou grafického hardvéru. Pre výpočty na grafickom hardvéri je možné použiť rozhrania OpenCL a CUDA. CUDA je však obmedzená len na grafický hardvér od spoločnosti NVIDIA, naproti tomu OpenCL je podporované väčšou škálou výrobcov grafického hardvéru, ale aj procesorov. Keďže budeme chcieť pokryť čo najväčší počet užívateľov, je vhodné implementovať výpočet korelačných koeficientov nasledujúcimi spôsobmi:

1. **Čisto procesorovo** – výpočet bude síce pomalý, umožníme tým však výpočet na starších počítačoch. Môže byť použiteľné pre menej rozsiahle dáta,
2. **S využitím OpenCL** – výpočet akcelerovaný pomocou rozhrania OpenCL dostupnom na bežne používanom hardvéri (AMD, NVIDIA, Intel),
3. **S využitím CUDA** – výpočet akcelerovaný pomocou rozhrania CUDA (iba NVIDIA)

Aplikácia by teda mala automaticky detekovať dostupný hardvér a v prípade dostupnosti viacerých výpočtových platforiem ponúknuť užívateľovi možnosť výberu vhodnej platformy.

6.2.1 Výpočet pomocou OpenCL a CUDA

Paralelizovať budeme výpočet korelačných koeficientov v rámci jednotlivých okienok. Ak bude užívateľom zadaná hodnota posuvu napr. 1 sekunda, znamená to, že pri frekvencii $5kHz$ budeme potrebovať $2 \times 5000 + 1 = 10001$ výpočtov, teda hodnota τ bude v intervale $\langle -5000, 5000 \rangle$. Každé vlákno bude počítať korelácie okienok s inou hodnotou τ. V prípade, že bude požadovaných viac výpočtov ako je počet dostupných vláken, musí sa proces opakovať.

Kroky výpočtu:

1. Vytvorenie dvoch blokov dát zo signálov na základe zadanej šírky okienka a posunu medzi okienkami,
2. Skopírovanie blokov dát zo systémovej pamäte do pamäte GPU,
3. Spustenie paralelných výpočtov,
4. Skopírovanie dát z pamäti GPU do systémovej pamäte,
5. Posun na ďalšie okienko. Ak sme na poslednom okienku, pokračuje sa krokom (6.), inak sa opakuje cyklus od kroku (1.),
6. Vizualizácia dát.

6.2.2 Výpočet pomocou procesoru

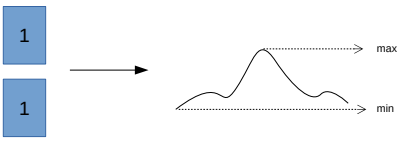
Keďže sa v súčasnej dobe používajú viac-jadrové procesory, môžme sa pokúsiť našu úlohu optimalizovať aj paralelizáciou na procesore. Paralelizovať budeme, rovnako ako pri GPU akcelerácii, výpočet korelačných koeficientov pri posuve dvoch okienok. Je potrebné vytvoriť vlákna, pričom každé z nich bude počítať korelácie medzi okienkami pri inom parametri

τ. Počet vláken použitých pri výpočte umožníme zadať užívateľovi. Pri implementácii pomocou procesoru odpadá réžia spôsobená prenosom dát medzi CPU a GPU.

6.3 Vizualizácia

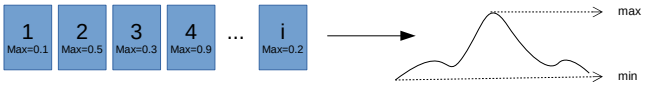
Výstupom výpočtu je množina 3-dimenzionálnych dát. Je teda potrebné dôkladne prepracovať návrh vizualizácie, aby boli výsledky prehľadné, dobre pochopiteľné a kompletné. Ukážeme postup vizualizácie od najnižšej úrovne (3. dimenzia).

Na úrovni tretej dimenzie máme 2 bloky dát, ktoré medzi sebou posúvame v čase a pre každý posuv získame korelačný koeficient. Z týchto hodnôt je následne možné zrekonštruovať nový signál, ktorý znázorní korelácie medzi týmito blokmi v čase $-\tau$ až τ (obr. 6.4). Charakteristickou a najdôležitejšou hodnotou bude maximálna hodnota korelačných koeficientov. Ďalej je vhodné zaznemenať minimálnu hodnotu a mediánovú hodnotu.



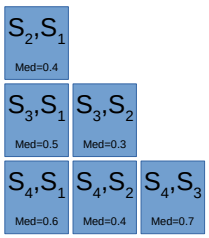
Obr. 6.4: Dimenzia 3.

Na úrovni druhej dimenzie máme množinu všetkých blokov. Každý blok bude charakterizovaný obdĺžnikom, ktorého farba bude určená maximálnou hodnotou korelačných koeficientov tretej dimenzie. Z týchto maximálnych hodnôt môžeme opäť zrekonštruovať novy signál (obr. 6.5).



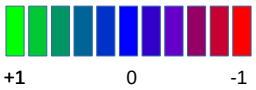
Obr. 6.5: Dimenzia 2.

Na úrovni prvej dimenzie máme množinu spracovávaných signálov. Treba teda vizualizovať 2-D maticu dát (dvojice signálov) uvedenú v (6.1). Každá dvojica signálov bude reprezentovaná obdĺžnikom, ktorého farba bude určená mediánovou hodnotou korelačných koeficientov druhej dimenzi (obr. 6.6).



Obr. 6.6: Dimenzia 1.

Pre zobrazenie hodnôt korelačných koeficientov budeme používať nasledujúcu farebnú schému. Zelená farba značí silnú pozitívnu koreláciu, modrá farba nulovú koreláciu a červená farba silnú negatívnu koreláciu (obr. 6.7).



Obr. 6.7: Farebná škála.

6.4 Export údajov

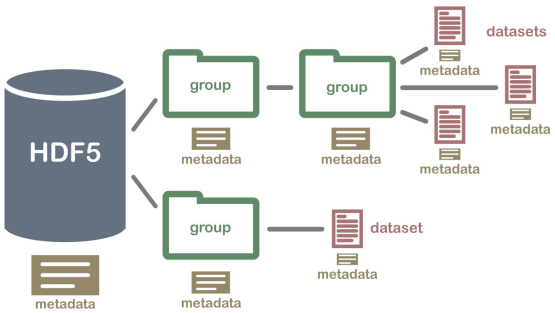
Po úspešnom ukonečení výpočtov budeme chcieť výsledky uchovať pre prípadné ďalšie spracovanie alebo opätovné zobrazenie. Preto je potrebné zabezpečiť uloženie údajov do súboru a zabezpečiť kompatibilitu s aplikáciou SignalPlant tak, aby dokázala otvoriť nami vygenerované výsledky. SignalPlant pre ukladanie údajov do súboru používa primárne binárny formát *HDF5*. Je však možné použiť aj jednoduchší textový formát *CSV*.

6.4.1 HDF5 formát

HDF5 (Hierarchical Data Format) je dátový formát pre ukladanie a organizovanie dát rôznych dátových typov. Dáta sú v tomto súbore organizované do skupín (Group) a množín (Dataset).

- **Skupina** je niečo ako priečinok v bežnom súborovom systéme. Môže obsahovať ďalšie skupiny alebo dátové množiny,
- **Dátová množina** predstavujé samotné dáta. Analogicky, súbor v bežnom súborovom systéme.

Vnútornú štruktúru súboru znázorňuje obr. (6.8). Výhodou tohto formátu je možnosť ukladania obrovského množsta dát a ich organizácia do skupín, zapísanie dát rôznych dátových typov, možnosť pridať popisné informácie – metadáta. Nevýhodou je potreba špeciálneho softvéru, ak chceme dáta zobrazovať alebo upravovať [27].



Obr. 6.8: Štruktúra HDF5 súboru [27].

6.4.2 CSV formát

CSV (Comma-separated Values) je jednoduchý textový formát pre ukladanie hodnôt oddelených čiarkou. Jeho výhodou je možnosť zobraziť a editovať údaje v akomkoľvek textovom editore. Nevýhodou je väčšia veľkosť súboru oproti formátu HDF5.

Príklad CSV súboru:

```
Name, EmailAddress, FirstName, LastName, Password
adamsta0109, terrya@mail.com, Terry, Adams, 1091990
beebeab0211, annb@mail.com, Ann, Beebe, 2894302
cannocc0328, chrisc@mail.com, Chris, Cannon, 9432456
```

Kapitola 7

Implementácia

Táto kapitola detailne popisuje implementáciu aplikácie podľa definovaných požiadaviek a podľa návrhu uvedeného v kapitole (6).

7.1 Integrácia do aplikácie SignalPlant

Aplikácia SignalPlant je pripravená na integráciu zásuvných modulov. Je zabezpečená detekcia a automatické načítanie vytvorených dynamických knižníc (dll) umiestnených v definovanom priečinku. K dispozícii je pripravená šablóna na tvorbu nových modulov. Celá aplikácia je napísaná v programovacom jazyku C# s využitím grafickej knižnice *Window Forms* pre tvorbu užívateľského rozhrania a s využitím vývojoveho prostredia *Microsoft Visual Studio.*

7.2 Výpočet

Výpočet tvorí najdôležitejšiu časť celej aplikácie. Všetky výpočty sú združené do abstraktnej triedy `Computation.` Táto trieda obsahuje všetky dôležité údaje potrebné pri výpočte:

- Vstupné signály – `List<List<float>> data`,
- Názvy signálov – `List<string> labels`,
- Vzorkovacia frekvencia – `float frequency`,
- Veľkosť okienka – `long windowSize`,
- Posuv okienka – `long windowOffset`,
- Posuv v rámci okienka – `long offset`,

Od triedy `Computation` sú ďalej odvodené triedy `GPUComputation` a `CPUComputation`, ktoré implementujú výpočet v závislosti na zvolenej výpočtovej platforme. Po nastavení vstupných parametrov sa zavolá metóda `compute(),` ktorá je implementovaná v odvodených triedach. Tým sa spustí výpočet v závislosti na zvolenej platforme.

7.2.1 OpenCL

Pri implementácii výpočtu pomocou OpenCL bola použitá knižnica *Cloo*, ktorá poskytuje väzbu rozhrania OpenCL na jazyk C#. Pri programovaní paralelných výpočtov s využitím GPU je treba určiť, ktorá časť kódu bude vykonávaná na CPU, a ktorá na GPU. V návrhu aplikácie sme určili, že paralelizovaný bude výpočet posuvov v rámci okienka. Všetko ostatné bude vykonávať procesor. Výpočet bežiaci na GPU je popísaný v časti (7.2.3). V triede `GPUComputation` sú definované 2 metódy spojené s výpočtom pomocou OpenCL: `ComputeOpenCL()` a `ComputeBlockCL()`. Samotný výpočet prebieha nasledovne:

1. Zavolá sa metóda `ComputeOpenCL()`,
2. Skompiluje sa kód kernelu a inicializuje sa zariadenie,
3. Vytvorí sa dvojica signálov,
4. Vytvorí sa dvojica blokov dát na základe zadanej šírky okienka a posuvu,
5. Pre každú dvojicu takto vytvorených blokov sa zavolá metóda `computeBlockCL()`,
6. V metóde `computeBlockCL()` sa pomocou `new ComputeBuffer<float>()` vytvoria objekty reprezentujúce pole hodnôt v pamäti zariadenia a následne sa sem skopírujú dáta zo systémovej pamäte,
7. Vytvorí sa kernel a nastavia sa jeho argumenty:
   - Vstupné dáta – `xBuff` a `yBuff`,
   - Výstupné dáta – `resBuff`,
   - Posuv okienok – `offset`,
   - Veľkosť okienka – `windowSize`,
   - Počet už získaných výsledných hodnôt – `processed`,
8. Spustí sa kernel s maximálnym možným počtom vláken – `maxWorkGroupSize`,
9. Každé vlákno spustí výpočet s inou hodnotou posuvu (τ). Výsledok uloží do poľa `resBuff`. V prípade, že sme ešte nezískali všetky korelačné koeficienty, pokračuje sa bodom (7.), inak bodom (10.),
10. Získané pole hodnôt sa uloží a inkrementuje sa index začiatku okienka. Ak už sú spracované celé signály, pokračuje sa bodom (11.), inak sa cyklus opakuje od bodu (4.),
11. Ak sú spracované všetky dvojice signálov, výpočet sa ukončí, inak sa pokračuje bodom (3.).

7.2.2 CUDA

Implementácia pre CUDU je veľmi podobná implementácii pre OpenCL. Väzbu na programovací jazyk C# zabezpečuje knižnica *ManagedCuda.* Hlavným rozdielom oproti OpenCL je, že sa kernel nekompiluje za behu aplikácie. Program pre kernel je potrebné preložiť pomocou nástroja *NVCC*[1], ktorý vygeneruje súbor *PTX*[2], čo je vlastne medzikód s vlastnou sadou inštrukcií pre grafický hardvér. Kód CUDA kernelu je takmer identický s kódom OpenCL kernelu. Rozdiely sú uvedené v časti (7.2.3). V triede `GPUComputation` sú definované 2 metódy spojené s výpočtom pomocou CUDA: `ComputeCUDA()` a `ComputeBlockCUDA()`. Samotný výpočet prebieha nasledovne:

1. Zavolá sa metóda `ComputeCUDA()`,
2. Načíta sa kernel, inicializuje sa zariadenie a nastaví sa maximálny možný počet vláken.
3. Vytvorí sa dvojica signálov,
4. Vytvorí sa dvojica blokov dát na základe zadanej šírky okienka a posuvu,
5. Pre každú dvojicu takto vytvorených blokov sa zavolá metóda `computeBlockCUDA()`,
6. V metóde `computeBlockCUDA()` sa pomocou `new CudaDeviceVariable<float>()` vytvoria objekty reprezentujúce pole hodnôt v pamäti zariadenia a pomocou metódy `CopyToDevice()` sa sem skopírujú dáta zo systémovej pamäte,
7. Vytvorí sa kernel a nastavia sa jeho argumenty:
   - Vstupné dáta – `x` a `y`,
   - Výstupné dáta – `res`,
   - Posuv okienok – `offset`,
   - Veľkosť okienka – `windowSize`,
   - Počet už získaných výsledných hodnôt – `processed`,
8. Spustí sa kernel s maximálnym možným počtom vláken – `maxWorkGroupSize`,
9. Každé vlákno spustí výpočet s inou hodnotou posuvu (τ). Výsledok uloží do poľa `resBuff`. V prípade, že sme ešte nezískali všetky korelačné koeficienty, pokračuje sa bodom (7.), inak bodom (10.),
10. Získané pole hodnôt sa uloží a inkrementuje sa index začiatku okienka. Ak už sú spracované celé signály, pokračuje sa bodom (11.), inak sa cyklus opakuje od bodu (4.),
11. Ak sú spracované všetky dvojice signálov, výpočet sa ukončí, inak sa pokračuje bodom (3.).

[1] NVCC – Nvidia Cuda Compiler Driver
[2] PTX – Parallel Thread Execution

7.2.3 Kernel

Kernel je program, ktorý je spúštaný na GPU. Každé vlákno na GPU je identifikované svojím ID a spúšťa práve jeden kernel. Obr. (7.1) znázorňuje výpočet korelačných koeficientov pri veľkosti okienka = 10 a uvažovaným posuvom v rámci okienka = 2. Teda výsledkom bude $2 \times 2 + 1 = 5$ korelačných koeficientov. Obr. (7.2) znázorňuje, ktoré vlákno zapisuje na ktorý index pri ktorej hodnote τ. Kód pre kernel je uvedený v prílohách (C) a (D). Podstatný rozdiel tvorí hlavne definovanie funkcií. Pri OpenCL sa funkcia začína kľúčovým slovom `__kernel`, pri CUDA je to `__global__`. Pri OpenCL sa kľúčovým slovom `__global` pred parametrom funkcie definuje, že premenná bude v globálnej pamäti.

Ako vstupné parametre dostáva kernel tieto hodnoty:

- Dvojicu blokov signálov – `float* x` a `float* y`,
- Uvažovaný posuv blokov – `long offset`,
- Veľkosť blokov – `long size`,
- ID vlákna – `long myIdx` – táto hodnota sa pripočítava k ID vláknu v prípade, že blok treba spracovať vo viacerých iteráciach.

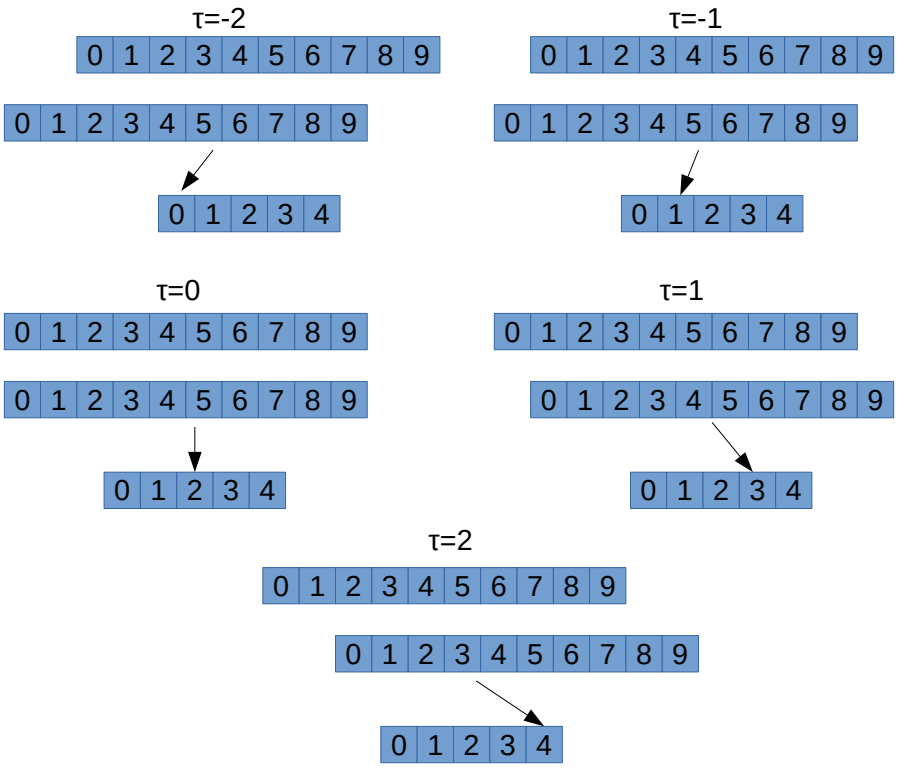
Výstupné parametre:

- Pole korelačných koeficientov – `float* result.`

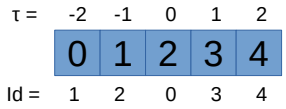
7.2.4 CPU

Výpočet na procesore je implementovaný v triede `CPUComputation.` V rámci tejto triedy je implementovaná ešte trieda `ChildCompute,` v ktorej je implementovaný výpočet pre konkrétnu hodnotu posuvu okienka. Je možné definovať počet vláken, ktoré sa majú použiť pri výpočte. Vysoký počet vláken však nemusí zaručiť získanie výsledkov v kratšom čase kvôli nutnej réžii pri vytváraní vláken. Výpočet na procesore prebieha nasledovne:

1. Vytvorí sa dvojica signálov,
2. Vytvorí sa dvojica blokov dát na základe zadanej šírky okienka a posuvu,
3. Pre každú dvojicu takto vytvorených blokov sa zavolá metóda `computeBlockCPU()`,
4. V metóde `computeBlockCPU()` sa vytvorí inštancia triedy `childData` a nastavia sa parametre: dáta oboch blokov, šírka okienka, posuv v rámci okienka a počet vláken,
5. Vytvoria sa vlákna na základe definovaného počtu vláken,
6. Každé vlákno spustí výpočet s inou hodnotou posuvu (τ). Výsledok uloží do poľa `result` v objekte `childData.` V prípade, že sme ešte nezískali všetky korelačné koeficienty, pokračuje sa bodom (5.), inak bodom (7.),
7. Získané pole hodnôt sa uloží a inkrementuje sa index začiatku okienka. Ak už sú spracované celé signály, pokračuje sa bodom (8.), inak sa vytvorí nová dvojica blokov a cyklus sa opakuje od bodu (2.),
8. Ak sú spracované všetky dvojice signálov, výpočet sa ukončí, inak sa pokračuje bodom (1.).



Obr. 7.1: Posuvy v rámci okienka pri hodnote posuvu = 2 a veľkosti okienka = 10.

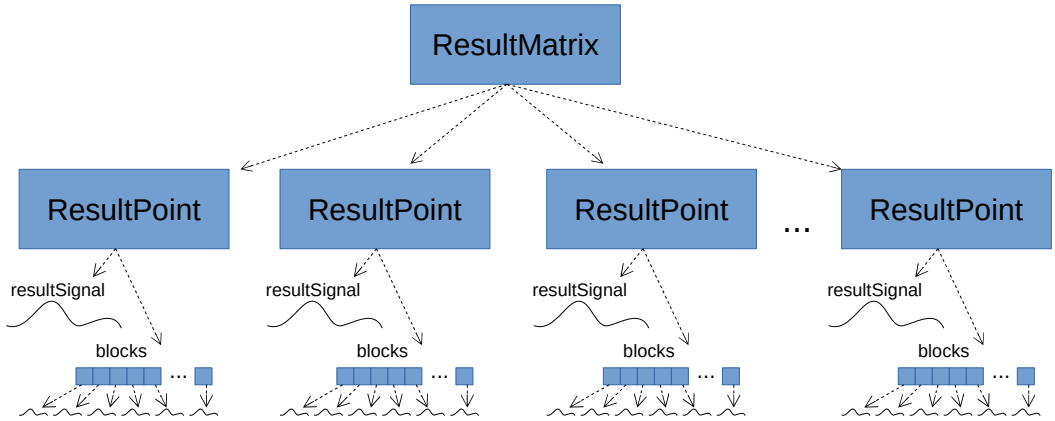


Obr. 7.2: Zápis na konkrétny index v závislosti na ID vlákna a hodnote τ.

7.3 Dátová štruktúra

Pretože dostávame výsledky v troch dimenziách, výsledná dátová štruktúra je trochu zložitá. V návrhu aplikácie (kapitola 6) sme si uviedli čo predstavuje ktorá dimenzia. Teraz je nutné reprezentovať jednotlivé dimenzie pomocou vhodných dátových štruktúr. Výsledná dátová štruktúra je implementovaná v triedach `ResultMatrix` a `ResultPoint`.

`ResultMatrix` obsahuje list objektov typu `ResultPoint`, ktoré predstavujú jednotlivé kombinácie signálov. Každý `ResultPoint` ďalej obsahuje list blokov – `blocks` a signál zrekonštruovaný z týchto blokov – `resultSignal`. Ďalej sa v tejto triede nachádzajú informácie potrebné pre vizualizáciu dát (viď časť 7.4). Schéma výslednej dátovej štruktúry je znázornená na obr. (7.3).



Obr. 7.3: Výsledná dátová štruktúra.

7.4 Vizualizácia výsledných dát

Vizualizácia dát je zabezpečená pomocou dvoch okien. Prvým oknom rozumieme okno celého pluginu, druhým oknom potom detail zvolenej dvojice signálov.

7.4.1 Hlavné okno

V hlavnom okne sa nachádzajú ovládacie prvky, pomocou ktorých je možné nastaviť vstupné parametre pre výpočet. Ďalej sa tu nachádzajú informácie o aktuálne prebiehajúcom výpočte (statusbar) a prvky umožňujúce export údajov (uloženie do súboru, kopírovanie do schránky). Z hľadiska vizualizácie samotných dát je dôležitý panel, na ktorom sa zobrazujú výsledky korelácie. Pre každú kombináciu signálov sa vykreslí štvorec s farbou vypočítanou na základe mediánu zo všetkých blokov danej dvojice signálov. Tento výpočet je realizovaný v metóde `CalculateColor()`. Všetky vykreslené štvorce sú organizované do matice uvedenej v rovnici (6.1). Hlavné okno teda zabezpečuje vizualizáciu dát prvej dimenzie (obr. 7.4).

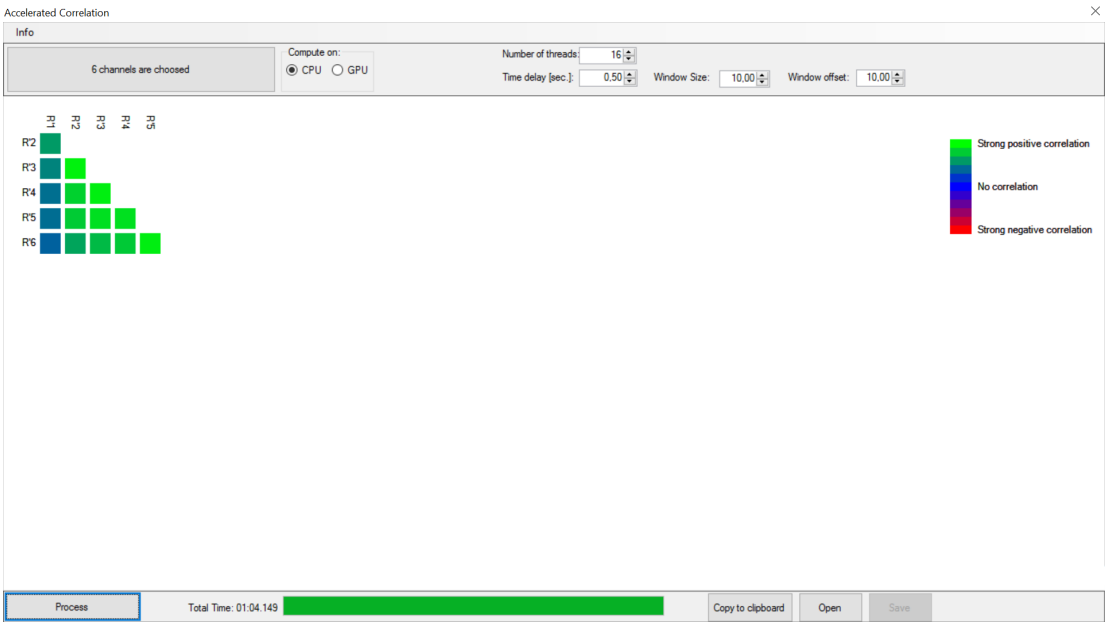
7.4.2 Okno s detailmi

Okno s detailmi je implementované v triede `SignalPlotter`. Táto trieda zabezpečuje vykreslenie každého objektu triedy `ResultPoint`, teda vizualizáciu druhej a tretej dimenzie. Na každý prvok matice v hlavnom okne je možné kliknúť. Po kliknutí sa otvorí okno s detailmi. V tomto okne je v hornej časti zobrazený signál vypočítaný z maximálnych hodnôt každého bloku – `resultSignal`. Okrem samotného signálu sú tu zobrazené aj informácie ako minimálna, maximálna a mediánova hodnota tohto signálu. Tento signál predstavuje vizualizáciu druhej dimenzie.

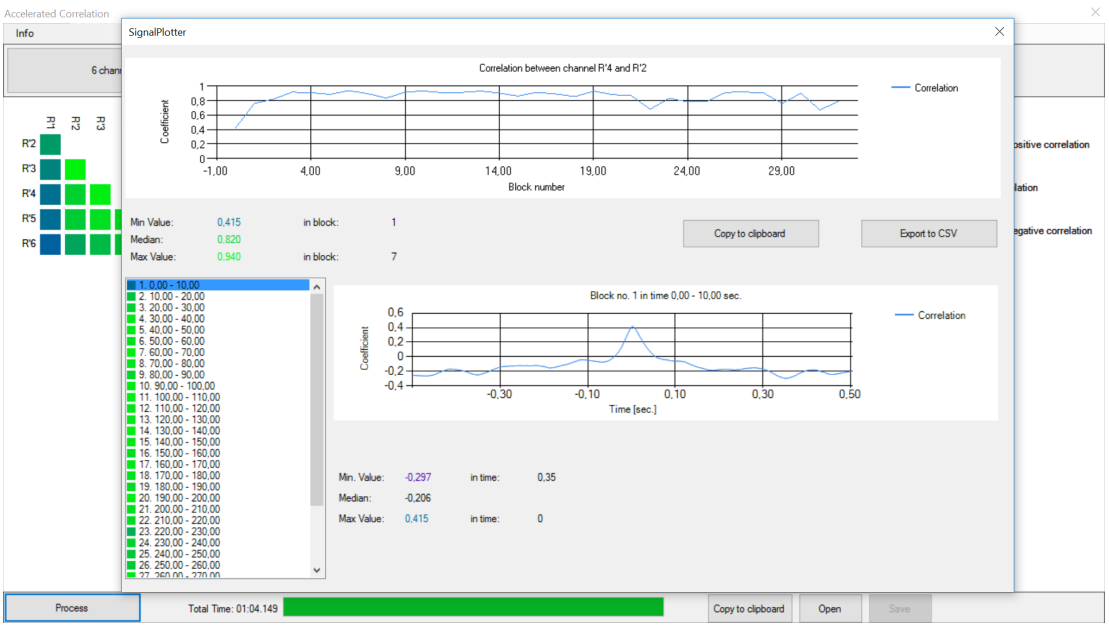
Tretia dimenzia je vizualizovaná v dolnej časti okna. V ľavej časti je umiestnený *listBox*[3], v ktorom sa nachádzaju jednotlivé bloky. Každý blok je reprezentovaný vnorenou triedou `ListBoxItem`. Sú tu uložené vypočítané korelačné koeficienty – `values`, farba vypočítaná na základe maximálnej hodnoty korelačného koeficientu – `color`, index bloku – `blockIndex`

[3]ListBox – prvok grafického užívateľského rozhrania, ktorý umožňuje zvoliť prvok alebo prvky zo zoznamu elementov.

a údaje o začiatku a konci bloku – `value`. Po kliknutí na blok sa v pravej časti zobrazí signál vypočítaný z hodnôt korelačných koeficientov príslušného bloku, údaje o maximálnej, minimálnej a mediánovej hodnote (obr. 7.5).



Obr. 7.4: Hlavné okno.



Obr. 7.5: Okno s detailmi.

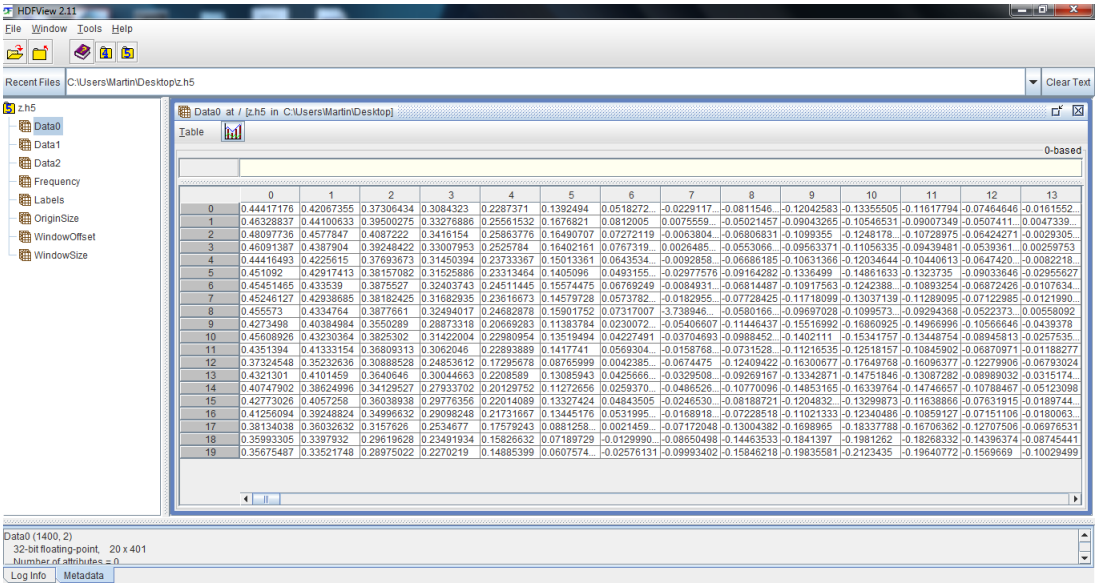
7.5 Export údajov

Ukladanie a načítanie vypočítaných hodnôt je implementované v triede `SignalWriter`. Podporované sú dátové formáty `HDF5` a `CSV`. Popis týchto formátov je uvedený v kapitole (6). Do HDF5 súboru sa ukladá celá dátová štruktúra (všetky 3 dimenzie). Takto vytvorený súbor je možné opätovne otvoriť implementovanou aplikáciou, nie však samotným SignalPlant-om. Do CSV súboru nie je možné uložiť 3-D dátovú štruktúru, je však možné uložiť 2-D štruktúry. Užívateľ si preto preto môže zvoliť, či chce uložiť výsledný signál (zrekonštruovaný z maximálnych hodnôt blokov) pre všetky kombinácie signálov alebo uložiť všetky bloky zvolenej dvojice signálov. Takto vytvorené súbory je možné načítať aplikáciou SignalPlant, nie už však implementovaným pluginom (dáta nie sú kompletné, vždy chýba jedna dimenzia).

7.5.1 HDF5

Zápis do HDF5 súboru je realizovaný pomocou metódy `WriteH5`. Táto metóda dostane na vstup celú dátovú štruktúru – objekt `ResultMatrix`. Pre každú dvójicu signálov sa potom vytvorí *DataSet* (viď obr. 6.8), do ktorého sa zapíše 2-rozmerné pole hodnôt – korelačné koeficienty pre každý blok. Okrem samotných hodnôt sa vytvoria aj DataSety pre informácie ako sú vzorkovacia frekvencia, veľkosť okienka, posuv okienok a veľkosť pôvodných signálov.

Takto vytvorený súbor je možné opäť načítať a zobraziť informácie v ňom obsiahnuté. Načítanie dát prebieha pomocou metódy `ReadH5()`, ktorá z nich vytvorí objekt typu `ResultMatrix`. Na obr. (7.6) je zobrazená štruktúra vygenerovaného HDF5 súboru otvorenom v prehliadači *HDFfView*.



Obr. 7.6: Štruktúra HDF5 súboru.

7.5.2 CSV

Pre zapís dát do formátu CSV sú implementované dve metódy:

- `WriteCSV(ResultMatrix,...)` – metóda pre zápis zrekonštruovaných signálov z maximálnych hodnôt blokov. Na jednotlivé riadky sú zapísané signály reprezentujúce všetky kombinácie signálov (viď obr. 7.7),
- `WriteCSV(ResultPoint,...)` – metóda pre zápis signálov vytvorených z korelačných koeficientov každého bloku pre zvolenú kombináciu signálov. Na každom riadku sú zapísané korelačné krivky reprezentujúce jednotlivé bloky (viď obr. 7.8). Pri zápise však musia byť časové hodnoty na osi x posunuté do hodnoty 0. Napr. pri nastavenom časovom posuve 1 sekunda dostávame výslednú krivku s x-ovými hodnotami v intervale $\langle -1,1 \rangle$. SignalPlant však nedokáže načítať signály so zápornými časmi, preto sa pôvodný interval posunie do intervalu $\langle 0,2 \rangle$.

```
'Block number','II_x_I','II_x_LA','II_x_RA','II_x_LL','I_x_LA','I_x_RA','I_x_LL','LA_x_RA','LA_x_LL','RA_x_LL'
'hh:mm:ss.mmm','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv'
'00:00:00.000',0.8465691,0.1214434,0.173838,0.9058139,0.4163355,0.2827196,0.6833018,0.5533431,0.3562937,0.3220313
'00:00:00.001',0.8410652,0.1177264,0.1795842,0.9107823,0.4189748,0.3336089,0.6860306,0.5827022,0.3553019,0.3107218
'00:00:00.002',0.8452817,0.114221,0.1773594,0.9044106,0.4056656,0.3392704,0.6871951,0.6159897,0.3533336,0.3187411
'00:00:00.003',0.8407917,0.1096673,0.1750276,0.9106643,0.4167511,0.3427753,0.6867971,0.5979246,0.3467722,0.3022817
'00:00:00.004',0.8394153,0.1457881,0.1858101,0.904784,0.4493946,0.3393064,0.6758831,0.5856352,0.3653019,0.3211292
'00:00:00.005',0.8412349,0.1170873,0.1322565,0.9037665,0.4135518,0.2907068,0.6794483,0.5925813,0.3404697,0.2791512
'00:00:00.006',0.8495405,0.157315,0.1910369,0.9060696,0.4397153,0.3422533,0.6895238,0.6151809,0.3648471,0.3271755
'00:00:00.007',0.8506231,0.1486388,0.194971,0.9072284,0.436294,0.3505981,0.6962496,0.6121503,0.3520369,0.3307847
'00:00:00.008',0.8411841,0.1208556,0.1873753,0.9076578,0.4274444,0.3404225,0.6831151,0.6040449,0.3585883,0.3273122
'00:00:00.009',0.8407813,0.1441629,0.1618699,0.902681,0.4394219,0.300638,0.6789532,0.5744279,0.3744107,0.3060913
'00:00:00.010',0.849089,0.1116753,0.1841643,0.9070622,0.3936471,0.3268835,0.6961546,0.6027296,0.3777215,0.3283297
'00:00:00.011',0.8468853,0.1606991,0.1626485,0.9039906,0.4685819,0.2968698,0.6847069,0.5859467,0.3604289,0.3121086
'00:00:00.012',0.8378057,0.1353908,0.1888615,0.907908,0.4584548,0.2614093,0.6773586,0.5235332,0.3689024,0.3314668
'00:00:00.013',0.8387518,0.1569922,0.1666311,0.9012842,0.4728877,0.3023209,0.6737162,0.586741,0.3560985,0.3177257
'00:00:00.014',0.8561743,0.1538418,0.1616535,0.9055542,0.4439524,0.2894719,0.7012095,0.5680171,0.3690282,0.3210031
'00:00:00.015',0.8399566,0.1714078,0.1813031,0.9041288,0.4891019,0.3186194,0.6752023,0.5968607,0.3577116,0.3276283
'00:00:00.016',0.8532252,0.195683,0.2229122,0.9070597,0.5000065,0.2949342,0.6903555,0.5572855,0.3587098,0.3484204
'00:00:00.017',0.8334284,0.159699,0.1636026,0.9015518,0.4892586,0.2830196,0.6629542,0.5203064,0.3911008,0.3053323
'00:00:00.018',0.8458339,0.1852244,0.1827077,0.9046655,0.4939564,0.3033613,0.6787972,0.5092462,0.3682868,0.3055452
'00:00:00.019',0.8872594,0.1908494,0.2834312,0.9088269,0.4424337,0.2957274,0.7438239,0.5510795,0.3685777,0.4159728
```

Obr. 7.7: Štruktúra CSV súboru s kombináciami signálov.

```
'Elapsed time','Block0','Block1','Block2','Block3','Block4','Block5','Block6','Block7','Block8','Block9','Block10','Block11','Block12','Block13','Block14','Block15','Block16','Block17','Block18','Block19'
'hh:mm:ss.mmm','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv','uv'
'00:00:00.000',-0.03597253,-0.06235755,-0.06477665,-0.07406763,-0.07264105,-0.05710057,-0.06590101,-0.0781339,-0.06057541,-0.07564285,-0.07269117,-0.02505479,-0.05331952,-0.03959292,-0.04429137
'00:00:00.001',-0.0330162,-0.0594444,-0.06225658,-0.07163192,-0.06969775,-0.05467876,-0.06335861,-0.07555502,-0.05813548,-0.07341568,-0.07074504,-0.02258986,-0.05068263,-0.03702231,-0.04146417,-
'00:00:00.002',-0.03003602,-0.05646658,-0.05963822,-0.06907446,-0.06666992,-0.05216565,-0.06065362,-0.0727879,-0.05549845,-0.07100247,-0.06860242,-0.01996097,-0.04792476,-0.03427863,-0.03848501
'00:00:00.003',-0.02697404,-0.0533911,-0.0568772,-0.06634396,-0.06348335,-0.04951585,-0.05777413,-0.06982589,-0.05264217,-0.06835023,-0.06623354,-0.01708514,-0.04497004,-0.03130849,-0.03534732,-
'00:00:00.004',-0.02390088,-0.05024993,-0.05401218,-0.06346007,-0.06019408,-0.04673247,-0.05470824,-0.06668173,-0.04963104,-0.06547997,-0.06368702,-0.01403475,-0.04192028,-0.02821855,-0.0320585
'00:00:00.005',-0.02088614,-0.04709067,-0.05108064,-0.06053214,-0.05689606,-0.0439079,-0.05157724,-0.06348846,-0.04651818,-0.06246461,-0.06096612,-0.01091135,-0.03886944,-0.02505746,-0.02878304
'00:00:00.006',-0.01791193,-0.04394467,-0.04812532,-0.05759577,-0.05364113,-0.04109029,-0.04842556,-0.06022698,-0.04336545,-0.05935236,-0.05816748,-0.007759549,-0.03580985,-0.02192893,-0.025507
'00:00:00.007',-0.01493518,-0.04079089,-0.04512665,-0.05460691,-0.05040538,-0.03825071,-0.04525768,-0.05692569,-0.04014208,-0.05614077,-0.05528382,-0.004536193,-0.03275405,-0.01875,-0.02221604,-
'00:00:00.008',-0.01199946,-0.03769805,-0.0421569,-0.0516588,-0.04722963,-0.03543476,-0.042088,-0.05359616,-0.0369166,-0.05284088,-0.05230044,-0.001326601,-0.02972074,-0.01558544,-0.01891593,-0.
'00:00:00.009',-0.009089293,-0.03462461,-0.0391927,-0.04869977,-0.04409641,-0.03260858,-0.03893299,-0.0502284,-0.03366854,-0.04950987,-0.04927548,0.001898955,-0.02672405,-0.0125193,-0.01567172,
```

Obr. 7.8: Štruktúra CSV súboru s blokmi.

Kapitola 8

Testovanie

Testovanie bolo zamerané na porovnanie výpočtových časov na rôznych hardvérových platformách. Testy prebiehali na nasledujúcich počítačových konfiguráciach:

- CPU: Intel Core i3-2330M 2.20 GHz, GPU: NVIDIA GeForce 410M + Intel HD 3000, OS: Windows 7 64-bit, RAM: 4GB
- CPU: Intel Core i5-4210U 2.40 GHz, GPU: AMD Radeon R5 M255 + Intel HD 4400, OS: Windows 7 64-bit, RAM: 8GB
- CPU: Intel Core i7-5500U 3.00 GHz, GPU: AMD Radeon R9 M375 + Intel HD 5500, OS: Windows 10 64-bit, RAM: 8GB

Pri testovaní boli zvolené dáta s ohľadom na výkon dostupných počítačov, aby sem dostali výsledky v rozumnom čase a aby teda bolo možné pozorovať rozdiel medzi jednotlivými výpočtovými platformami. Testované dáta boli tvorené intrakraniálnymi EEG záznamami – spolu 6 signálov s dĺžkou 5 minút a frekvenciou 500 Hz.

Testovanie aplikácie prebiehalo v 2 fázach:

1. Aplikácia bola testovaná na rôznych počítačoch s odlišným hardvérom pri rovnakom nastavení parametrov (časť 8.1):
   - Časový posuv = 0,5 sekundy,
   - Veľkosť okienka = 10 sekúnd,
   - Posun okienka = 10 sekúnd,
   - Počet CPU vláken = 16.
2. Aplikácia bola testovaná na jednom počítači s rôznym nastavením parametrov (časť 8.2).

8.1 Prvá fáza

V tejto fáze sa meral výpočtový čas na rôznom hardvéri pri rovnakom nastavení parameterov. Pre každú platformu bolo vykonaných 10 meraní (tabuľka 8.1). Napokono boli výpočítané priemerné hodnoty z nameraných hodnôt a určené zrýchlenie vzhľadom k CPU výpočtového času rovnakej počítačovej konfigurácie (tabuľka 8.2).

CPU	**Intel Core i3-2330M 2.20 GHz**	4:24.578	6:16.121	6:19.326	6:52.359	6:03.456	5:33.705	5:06.220	4:52.000	4:33.815	5:21.543
	Intel Core i5-4210U 2.40 GHz	1:46.706	2:00.121	2:06.999	2:03.393	1:58.385	1:54.195	2:10.231	1:58.031	1:52.389	2:00.984
	Intel Core i7-5500U 3.00 GHz	42.980	43.658	44.812	57.060	47.340	43.401	43.222	43.833	45.322	45.974
OpenCL CPU	**Intel HD 3000**	6.571	7.902	8.008	7.935	7.533	10.026	9.067	10.315	9.568	8.783
	Intel HD 4400	2.535	2.019	1.918	1.955	2.016	1.910	1.892	2.006	1.905	1.896
	Intel HD 5500	5.842	6.247	6.130	6.239	6.004	5.319	5.398	6.097	6.169	6.287
OpenCL GPU	**NVIDIA GeForce 410M**	10.567	12.166	12.787	12.516	16.522	17.016	15.169	15.227	15.446	15.350
	AMD Radeon R5 M255	3.945	3.780	3.818	3.816	3.776	3.775	3.761	3.766	3.757	3.763
	AMD Radeon R9 M375	6.493	6.897	5.111	4.298	4.814	4.573	4.581	5.292	4.293	4.666
CUDA	**NVIDIA GeForce 410M**	10.329	11.590	12.180	9.788	14.611	9.381	13.090	10.559	14.525	8.436

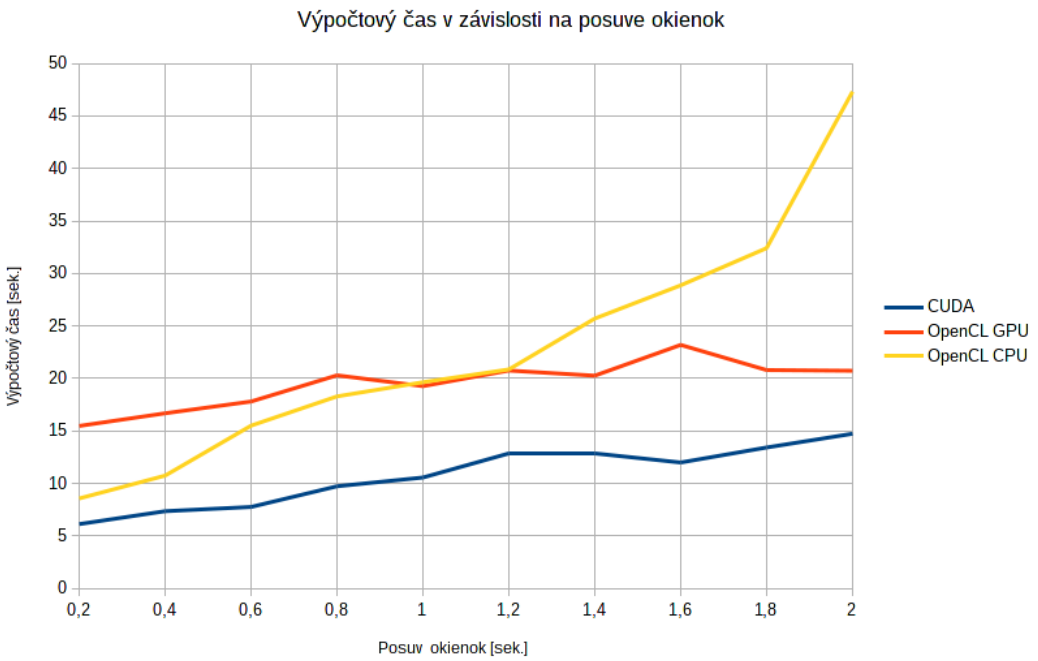
Tabuľka 8.1: Výpočtové časy pre rôzny hardvér. Časy sú uvedené vo formáte [min]:[sek].[milisek], resp. [sek].[milisek].

Platforma	Priemerný čas	Zrýchlenie vzhľadom k CPU
Intel Core i3-2330M 2.20 GHz	5:30 min.	-
Intel HD 3000 (OpenCL)	8.571 sek.	38x
NVIDIA GeForce 410M (OpenCL)	14.277 sek.	23x
NVIDIA GeForce 410M (CUDA)	11.449 sek.	29x
Intel Core i5-4210U 2.40 GHz	1:59 min.	-
Intel HD 4400 (OpenCL)	2.005 sek.	59x
AMD Radeon R5 M255 (OpenCL)	3.796 sek.	31x
Intel Core i7-5500U 3.00 GHz	45.760 sek.	-
Intel HD 5500 (OpenCL)	5.973 sek.	8x
AMD Radeon R9 M375 (OpenCL)	5.102 sek.	9x

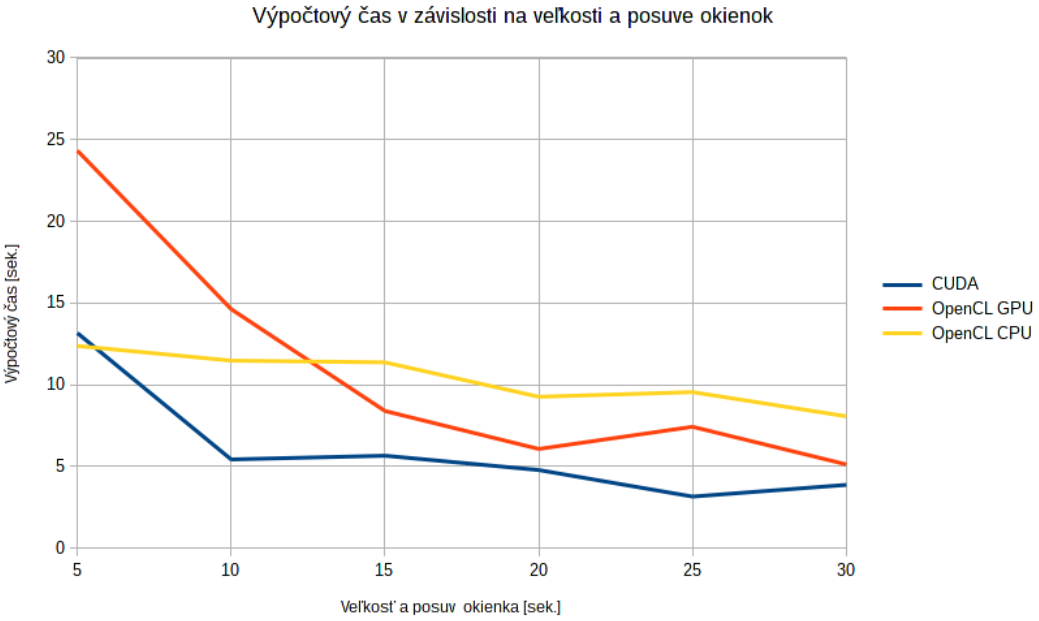
Tabuľka 8.2: Priemerné výpočtové časy a zrýchlenie. Farebne sú odlíšené jednotlivé počítačové konfigurácie. Zrýchlenie je vypočítané vzhľadom k CPU času rovnakej konfigurácie.

8.2 Druhá fáza

V druhej fáze bol testovaný výpočtový čas v závislosti na veľkosti a posune okienka a posuve medzi okienkami. Pri tomto teste už nebol uvažovaný výpočet čisto na procesore, pretože pri niektorých kombináciach parametrov by výpočet trval príliš dlho. Na obr. (8.1) je znázornená závislosť výpočtového času od zadanej hodnoty posuvu medzi okienkami. Na obr. (8.2) je znázornená závislosť výpočtového času od veľkosti a posunu okienka. Tieto výpočty prebiehali na hardvéri NVIDIA GeForce 410M (OpenCL GPU a CUDA) a Intel HD 3000 (OpenCL CPU).



Obr. 8.1: Závislosť výpočtového času od hodnoty posuvu medzi okienkami.



Obr. 8.2: Závislosť výpočtového času od veľkosti a posunu okienka.

Kapitola 9

Záver

Táto diplomová práca uvádza a analyzuje problematiku mozgovej konektivity. Na vyhodnotenie mozgovej konektivity existuje množstvo efektívnych či menej efektívnych metód. Najvhodnešie metódy analyzuje aj táto práca. Význam analýzy mozgovej konektivity je hlavne v neurochirurgii, kde pomocou vhodných metód môžeme u pacientov s epileptickými ochoreniami určiť epileptogénne zóny. Problémom pri týchto metódach môže byť ich výpočtová náročnosť. Preto bola v rámci tejto práce navrnutá a implementovaná vhodná metóda – založená na Pearsonovom korelačnom koeficiente, pomocou ktorej je výpočet efektívnejší a rýchlejší. Toto urýchlenie je zabezpečené implementáciou pomocou grafickej akcelerácie, kde dosahujeme podstatné zrýchlenie vzhľadom k rovnakej implementácii na procesore (viď zhodnotenie výsledkov – tabuľka. 8.2). Samotná aplikácia však nie je výhradne určená len pre analýzu záznamov získaných pomocou intrakraniálneho EEG. Je možné ňou analyzovať aj iné signály používané v medicínskych oblastiach (napr. ECG) alebo všeobecne akékoľvek signály.

Aplikácia poskytuje veľmi dobré časové výsledky v porovnaní s rovnakou implementáciou na CPU. To však neznamená, že nie je priestor pre ďalší vývoj aplikácie. Aplikáciu je možné ďalej akcelerovať s využitím maximálnej dostupnej pamäte grafického zariadenia. To možno dosiahnuť napr. tak, že sa bude simultánne spracovávať viac blokov signálu alebo viac signálov súčasne. Aplikáciu ocenia používatelia SignalPlant-u, ktorým sa pomocou tohto zásuvného modulu otvárajú nové možnosti spracovávania signálov.

Samotnej diplomovej práci predchádzal semestrálny projekt, ktorý sa venoval problematike mozgovej konektivity z teoretického hľadiska. Táto časť bola prevzatá a začlenená do diplomovej práce a na jej základe bol postavený návrh metódy pre analýzu mozgovej konektivity, ktorá bola následne implementovaná.

Prínosom tejto práce bolo získanie nových informácií jednak z oblasti neurochirurgie, neurológiee a biomedicíny, ale aj získanie nových poznatkov pri programovaní aplikácií pomocou grafickej akcelerácie.

Literatúra

[1] ADVANCED MICRO DEVICES. *Introduction to OpenCL™ Programming*. 2010.

[2] ANTHONY Arun R. et al. Functional Connectivity Estimated from Intracranial EEG Predicts Surgical Outcome in Intractable Temporal Lobe Epilepsy. *PLoS One*, 2013, vol. 8, iss. 10.

[3] BANDETTINI, Peter. Functional MRI: a confluence of fortunate circumstances. *Neuroimage*, 2012, vol. 61, s. A3–A11.

[4] BLUME, Howard. The Surgical Treatment of Epilepsy. In: *The Comprehensive Evaluation and Treatment of Epilepsy*, San Diego, CA: Academic Press, 1997, s. 197–206, ISBN 978-0-12-621355-3.

[5] COBEN, Robert a Iman MOHAMMAD-REZAZADEH. Neural connectivity in Epilepsy as measured by Granger Causality. *Frontiers in Human Neuroscience*, 2015, vol. 9, iss. 194, ISSN 1662-5161.
Dostupné z: http://www.frontiersin.org/human_neuroscience/10.3389/fnhum.2015.00194/abstract

[6] DIPATRI Arthur J. a Tord D. ALDEN. *Surgical Treatment of Epilepsy in Children* [online], 2005.
Dostupné z:
https://www2.luriechildrens.org/ce/online/article.aspx?articleID=103

[7] DOE, B.. Depth, strip and grid electrode placement for eeg monitoring in partial seizures. 2012.

[8] FRISTON, Karl J.. Modalities, modes, and models in functional neuroimaging. *Science*, 2009, vol. 326, s. 399–403.

[9] GOLIŃSKA, Agnieszka Kitlas. Coherence function in biomedical signal processing: a short review of applications in Neurology, Cardiology and Gynecology. *Studies in Logic, Grammar and Rhetoric*, 2011, vol. 25, iss. 38, s. 73–82.

[10] HERTZOG Lionel. *First steps with Non-Linear Regression in R* [online].
Dostupné z:
http://www.r-bloggers.com/first-steps-with-non-linear-regression-in-r/

[11] HONEY C. J., R. KÖTTER, M. BREAKSPEAR a O. SPORNS. Network structure of cerebral cortex shapes functional connectivity on multiple time scales. *Proc Natl*, 2007, vol. 104, s. 10240–10245.

[12] KANTER David. *Introduction to OpenCL* [online], 2010.
Dostupné z: http://www.realworldtech.com/opencl/

[13] KHRONOS Group. *Introduction and Overview* [online], 2010.
Dostupné z: https://www.khronos.org/assets/uploads/developers/library/overview/OpenCL-Overview-Jun10.pdf

[14] KHRONOS OpenCL Working Group. *The OpenCL Specification* [online], 2011.
Dostupné z: https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf

[15] LATHAM, P.E. a Y. ROUDI. Mutual information. *Scholarpedia*, 2009, vol. 4, iss. 1, str. 1658.

[16] MADULARA, M.D. et al. EEG Transfer Entropy Tracks Changes in Information Transfer on the Onset of Vision. *International Journal of Modern Physics: Conference Series*, 2012, vol. 17, s. 9–18.

[17] NVIDIA CORPORATION. *NVIDIA CUDA™ Programming Guide*. Santa Clara, USA, 2009.

[18] POKORNÝ, Jan. *Elektroencefalografie* [online], Kladno, ČR: Fakulta biomedicínského inženýrství, ČVUT.
Dostupné z: http://fbmi.cvut.cz/files/nodes/657/public/EEG.pdf

[19] PRESS, William H. et. al. *Numerical Recipes in C*, kapitola 14.5 Linear Correlation. Cambridge, VB: Cambridge University Press, druhé vydání, 1992.

[20] RAMAMAND, Pravitha, M.C. BRUCE a E.N. BRUCE. Mutual Information Analysis of EEG Signals Indicates Age-Related Changes in Cortical Interdependence during Sleep in Middle-aged vs. Elderly Women. *Journal of clinical neurophysiology*, 2010, vol. 27, iss. 4, s. 274–284.

[21] SPENCER, Dennis D.. Depth electrode implantation. In: *Surgical treatment of the epilepsies*, editace J. E. Jr, New York: Raven, 1987, s. 603–607.

[22] SPERLING, Michael R.. Clinical Challenges in Invasive Monitoring in Epilepsy Surgery. *Epilepsia*, 1997, vol. 38, iss. 4, s. S6–S12.

[23] SPORNS, Olaf. Brain connectivity. 2007, vol. 2, iss. 10, str. 4695.

[24] TOMPSON, Jonathan a Kristofer SCHLACHTER. An Introduction to the OpenCL Programming Model. *Person Education*, 2012.

[25] WANG, H. E., C. G. BÉNAR, P. P. QUILICHINI, K. J. FRISTON, V. K. JIRSA a C. BERNARD. A systematic framework for functional connectivity measures. *Frontiers in Neuroscience*, 2014, vol. 8, s. 1–22.

[26] WANG, Ting a Shiqiang ZHANG. Study on Linear Correlation Coefficient and Nonlinear Correlation Coefficient in Mathematical Statistics. *Studies in Mathematical Sciences*, 2011, vol. 3, iss. 1, s. 58–63.

[27] WASSER, Leah A. *About: Hierarchical Data Formats - What is HDF5?* [online], Boulder, Colorado, USA, 2015.
Dostupné z: http://neondataskills.org/HDF5/About

[28] WAVEMETRICS. *Convolution and Correlation* [online].
Dostupné z: https://www.wavemetrics.com/products/igorpro/dataanalysis/signalprocessing/convolution.htm

[29] WEN, Quan a Dmitri B. CHKLOVSKII. Segregation of the Brain into Gray and White Matter: A Design Minimizing Conduction Delays. *PLoS Comput Biol*, 2005, vol. 1.

[30] ŽALUD, Václav. *Elektroencefalografická vyšetření* [online], Plzeň, ČR: Ústav patologické fyziologie LF, UK v Praze.
Dostupné z: https://www.lfp.cuni.cz/patofyziologie/materialy/eeg.ppt

Prílohy

Zoznam príloh

Príloha A

Obsah CD

`CorrCL/` – adresár obsahujúci zdrojové kódy, z ktorých je možné zostaviť aplikáciu.
`lib/` – adresár s vytvorenou knižnicou a s knižnicami nutnými pre beh aplikácie.
`tex/` – adresár so zdrojovým textom technickej správy.
`manual.pdf` – užívateľská príručka.
`corrcl.pdf` – technická správa.
`README.txt` – popis aplikácie.
`INSTALL.txt` – popis inštalácie aplikácie.

Príloha B

Manuál

1. Najprv je nutné nainštalovať a zaregistrovať aplikáciu SignalPlant podľa pokynov (https://signalplant.codeplex.com/),
2. Po nainštalovaní aplikácie je potrebné skopírovať obsah adresáru `lib/` do adresáru `plugins/`,
3. Aplikácia sa spustí kliknutím na `signalplant.exe`,
4. Po spustení aplikácie treba vybrať signály pomocou menu `Open -> File`,
5. Po načítaní signálov treba spustiť plugin pomocou menu `Plugins -> Analysis -> Accelerated Correlation`,
6. Kliknutím na `Choose Channels` treba vybrať signály, ktoré chceme spracovávať,
7. Nastavíme parametre spracovania – CPU, GPU, windowSize, windowOffset, Time Delay, atď.,
8. Klikneme na `Process` a počkáme, kým získame výsledky,
9. Kliknutím na jednotlivé farebné štvorce získame detaily každej dvojice signálov.

Príloha C

Kód kernelu pre OpenCL

```
__kernel void Pearson(__global float* x,
                      __global float* y,
                      __global float* result,
                      const long offset,
                      const long size,
                      const long myIdx)
{
        int idx = get_global_id(0) + myIdx;
        int resultIdx = 0;
        int localIdx = 0;

        if (idx == 0) {
                resultIdx = offset;
                localIdx = 0;
        } else if (idx <= offset) {
                resultIdx = (idx - offset)*(-1);
                localIdx = idx;
        } else if (idx > offset) {
                resultIdx = idx;
                localIdx = idx - offset;
        }

        if (resultIdx >= 2 * offset + 1)
                return;

        float meanX;
        float meanY;
        float sumX;
        float sumY;
        float sumXX;
        float sumYY;
        float sumXY;
        meanX = 0.0;
        meanY = 0.0;
        sumX = 0.0;
        sumY = 0.0;
        sumXX = 0.0;
        sumYY = 0.0;
        sumXY = 0.0;

        if (idx <= offset) {
```

```
                int i=0;
                float actX=0.0;
                float actY=0.0;
                for (i=0; i < size - localIdx; i++) {
                        actX = x[i];
                        actY = y[i+localIdx];
                        sumX += actX;
                        sumXX += actX*actX;
                        sumY += actY;
                        sumYY += actY*actY;
                        sumXY += actX*actY;

                }

                meanX = sumX / i;
                meanY = sumY / i;
                float res = ((sumXY / i) - (meanX * meanY)) / ((sqrt((
                    sumXX / i) - (meanX * meanX))) * (sqrt((sumYY / i) -
                    (meanY * meanY))));
                if (isnan(res))
                        res = 0.0;
                result[resultIdx] = res;

        } else if (idx > offset) {
                int i=0;
                float actX=0.0;
                float actY=0.0;
                for (i=0; i < size - localIdx; i++) {
                        actX = x[i+localIdx];
                        actY = y[i];
                        sumX += actX;
                        sumXX += actX*actX;
                        sumY += actY;
                        sumYY += actY*actY;
                        sumXY += actX*actY;
                }

                meanX = sumX / i;
                meanY = sumY / i;
                float res = ((sumXY / i) - (meanX * meanY)) / ((sqrt((
                    sumXX / i) - (meanX * meanX))) * (sqrt((sumYY / i) -
                    (meanY * meanY))));
                if (isnan(res))
                        res = 0.0;
                result[resultIdx] = res;
        }
}
```

Príloha D

Kód kernelu pre CUDA

```
__global__ void Pearson(float* x,
                        float* y,
                        float* result,
                        const long offset,
                        const long size,
                        const long myIdx)
{
        int idx = (blockIdx.x * blockDim.x + threadIdx.x)+ myIdx;

        int resultIdx = 0;
        int localIdx = 0;

        if (idx == 0) {
                resultIdx = offset;
                localIdx = 0;
        } else if (idx <= offset) {
                resultIdx = (idx - offset)*(-1);
                localIdx = idx;
        } else if (idx > offset) {
                resultIdx = idx;
                localIdx = idx - offset;
        }

        if (resultIdx >= 2 * offset + 1)
                return;

        float meanX;
        float meanY;
        float sumX;
        float sumY;
        float sumXX;
        float sumYY;
        float sumXY;
        meanX = 0.0;
        meanY = 0.0;
        sumX = 0.0;
        sumY = 0.0;
        sumXX = 0.0;
        sumYY = 0.0;
        sumXY = 0.0;
```

```
        if (idx <= offset) {
                int i=0;
                float actX=0.0;
                float actY=0.0;
                for (i=0; i < size - localIdx; i++) {
                        actX = x[i];
                        actY = y[i+localIdx];
                        sumX += actX;
                        sumXX += actX*actX;
                        sumY += actY;
                        sumYY += actY*actY;
                        sumXY += actX*actY;

                }

                meanX = sumX / i;
                meanY = sumY / i;
                float res = ((sumXY / i) - (meanX * meanY)) / ((sqrt((
                    sumXX / i) - (meanX * meanX))) * (sqrt((sumYY / i) -
                    (meanY * meanY))));
                if (isnan(res))
                        res = 0.0;
                result[resultIdx] = res;

        } else if (idx > offset) {
                int i=0;
                float actX=0.0;
                float actY=0.0;
                for (i=0; i < size - localIdx; i++) {
                        actX = x[i+localIdx];
                        actY = y[i];
                        sumX += actX;
                        sumXX += actX*actX;
                        sumY += actY;
                        sumYY += actY*actY;
                        sumXY += actX*actY;
                }

                meanX = sumX / i;
                meanY = sumY / i;
                float res = ((sumXY / i) - (meanX * meanY)) / ((sqrt((
                    sumXX / i) - (meanX * meanX))) * (sqrt((sumYY / i) -
                    (meanY * meanY))));
                if (isnan(res))
                        res = 0.0;
                result[resultIdx] = res;
        }
}
```