

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF COMPUTER SYSTEMS

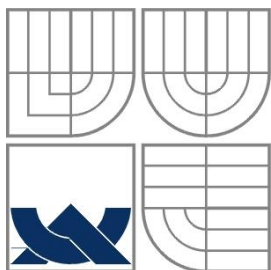
AKCELERACE ALGORITMŮ PRO HLEDÁNÍ  
PALINDROMŮ A OPAKUJÍCÍCH SE STRUKTUR

DIPLOMOVÁ PRÁCE  
MASTER'S THESIS

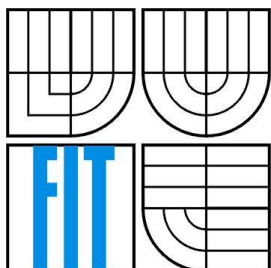
AUTOR PRÁCE  
AUTHOR

BC. JAN VOŽENÍLEK

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF COMPUTER SYSTEMS

# AKCELERACE ALGORITMŮ PRO HLEDÁNÍ PALINDROMŮ A OPAKUJÍCÍCH SE STRUKTUR

ACCELERATION OF METHODS FOR SEARCHING PALINDROMES AND REPETITIVE  
STRUCTURES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

BC. JAN VOŽENÍLEK

VEDOUCÍ PRÁCE

SUPERVISOR

ING. TOMÁŠ MARTÍNEK

BRNO 2010

## Zadání diplomové práce

Řešitel: **Voženílek Jan, Bc.**

Obor: Počítačové systémy a sítě

Téma: **Akcelerace algoritmů pro hledání palindromu a opakujících se struktur**  
**Acceleration of Methods for Searching Palindroms and Repetitive Structures**

Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s technologií programovatelného hardware (FPGA) od firmy Xilinx.
2. Prostudujte si základní metody z oblasti analýzy biologických sekvencí. Dále pak algoritmy pro hledání palindromu a opakujících se sekvencí.
3. Na základě předchozích dvou bodů navrhnete vhodnou architekturu pro technologii FPGA, která bude sloužit k akceleraci hledání palindromů a opakujících se biologických sekvencí.
4. Realizujte funkční prototyp navržené architektury v jazyce VHDL nebo HandleC a ověřte její funkčnost na kartě COMBO.
5. V závěru diskutujte dosažené výsledky a možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části diplomového projektu je požadováno:

- První tři body.

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 ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

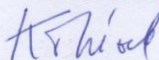
Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo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

Vedoucí: **Martínek Tomáš, Ing.**, UPSY FIT VUT

Datum zadání: 21. září 2009

Datum odevzdání: 26. května 2010

**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
Fakulta informačních technologií  
Ústav počítačových systémů a sítí  
612 66 Brno, Božetěchova 2



doc. Ing. Zdeněk Kotásek, CSc.  
vedoucí ústavu

## **Abstrakt**

Veškerá genetická informace živých organismů je uložena v DNA. Zkoumání její struktury a funkce představuje důležitou oblast výzkumu moderní biologie. Jednou ze zajímavých struktur, vyskytujících se v sekvencích DNA, jsou také palindromy. Na základě jejich výzkumu se předpokládá, že hrají důležitou roli při interpretaci informace uložené v DNA, jelikož se často vyskytují v okolí důležitých genů. Jejich hledání je složitější díky výskytu mutací (změn v posloupnosti prvků DNA), což zvyšuje časovou složitost algoritmů. Proto má smysl zabývat se jejich paralelizací a akcelerací. Rozborem metod pro hledání palindromů a návrhem akcelerační architektury se zabývá tato práce. Výpočet pomocí hardwarové jednotky implementované v čipu FPGA na kartě ml555 může být až 6 667krát rychlejší oproti nejlepšímu známému softwarovému řešení využívajícímu sufixová pole.

## **Abstract**

Genetic information of all living organisms is stored in DNA. Exploring of its structure and function represents an important area of research in modern biology. One of the interesting structures occurring in DNA are palindromes. Based on the research they are expected to play an important role in interpreting the information stored in DNA, because they are often observed near important genes. Palindromes searching is complicated by the presence of mutations (changes in sequences of DNA elements), which increases the time complexity of algorithms. Therefore it is reasonable to study their parallelization and acceleration. The objective of this work is a study of palindromes searching methods and acceleration architecture design. The hardware unit implemented in a chip with FPGA technology placed on ml555 board can speed up the calculation up to 6 667 times in comparison with the best-known software method relying on suffix arrays.

## **Klíčová slova**

Hledání přibližných palindromů, hardwarová akcelerace, VHDL, FPGA.

## **Keywords**

Approximate palindrome detection, hardware acceleration, VHDL, FPGA.

## **Citace**

Voženílek Jan: Akcelerace algoritmů pro hledání palindromů a opakujících se struktur, diplomová práce, Brno, FIT VUT v Brně, 2010

# Akcelerace algoritmů pro hledání palindromů a opakujících se struktur

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Tomáše Martínka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....  
Jan Voženílek  
26. 5. 2010

## Poděkování

Děkuji Tomáši Martínkovi za vedení práce a řešitelům projektu Liberouter za odbornou pomoc.

© Jan Voženílek, 2010.

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Obsah

Obsah.....	1
1 Úvod .....	2
2 Teoretický rozbor.....	3
2.1 Analýza DNA a RNA .....	5
2.2 Zarovnání sekvencí .....	6
2.3 Definice a význam palindromů .....	9
2.4 Algoritmy pro hledání palindromů.....	13
2.5 Technologie FPGA .....	19
3 Hardwarová akcelerace .....	21
3.1 Předávání dat, zřetěžené zpracování .....	21
3.2 Architektura hardwaru .....	23
3.3 Rozhraní akcelerační jednotky .....	29
3.4 Struktura softwaru.....	31
3.5 Parametry akcelérátoru .....	34
Závěr .....	37

# 1 Úvod

Lidstvo se již od nepaměti snaží pochopit fungování živých organismů na všech úrovních. Jednou z důležitých oblastí moderní biologie je výzkum struktury a funkce deoxyribonukleové kyseliny (DNA), která je nositelem veškeré genetické informace každého jedince. Tento obor se začal rozvíjet především po objevu struktury DNA Watsonem a Crickem v roce 1953 (poprvé sestavili známý model dvoušroubovice), díky kterému bylo možné začít genetický kód vyjadřovat pomocí čtyř znaků A, C, G a T, což jsou počáteční písmena názvů základních stavebních prvků DNA.

Výzkum ukázal [1], že hlavní funkci organismu určují pouze některé úseky DNA, které jsou označovány jako geny. Právě v jejich blízkosti byl zjištěn výskyt specifických struktur nazývaných palindromy, což jsou obecně řetězce, které se čtou stejně z jednoho i druhého konce. V DNA se palindromy podílejí na regulaci překlada genetického kódu z genů do proteinů (bílkovin). Konkrétně mohou označovat hranice genů nebo dokonce určovat, zda bude informace z daného genu použita k tvorbě bílkoviny či nikoli.

V DNA dochází poměrně často k malým změnám v posloupnosti prvků řetězce (mutacím), a to jak v rámci buněčných procesů manipulujících s genetickou informací, tak i působením vnějších vlivů (radiace, některých chemikálií či specializovaných virů). Při zkoumání řetězců je nutné k výskytu těchto změn přihlížet. Existuje řada algoritmů, které provádí analýzu sekvencí DNA. Tato práce se věnuje průzkumu postupů pro hledání přibližných palindromů (tedy palindromů s výskytem mutací), především pak možnostmi jejich paralelizace a následné akcelerace výpočtu.

Hlavními problémy, spojenými s analýzou DNA, jsou kromě výskytu mutací také velký objem dat (pro uložení kompletního souboru lidské DNA je zapotřebí přes 700 MB paměťového prostoru), a to jak vstupních tak i výstupních. U hledání přibližných palindromů se objevuje požadavek na možnost definování jejich kvality. Návrh akcelerační architektury byl proto proveden také s ohledem na co možná největší flexibilitu, díky které lze jednoduše připravit jednotku přesně odpovídající požadavkům konkrétní úlohy.

Následující kapitola obsahuje detailnější popis biologických struktur a rozbor problematiky analýzy řetězců, dále pak zavedení klasifikace přibližných palindromů, jejich význam v sekvencích nukleových kyselin a principy některých algoritmů pro jejich hledání. Ve třetí kapitole je popsán návrh akcelerační jednotky a také její výkonnostní parametry, především zrychlení výpočtu oproti nejlepšímu známému softwarovému řešení (používajícímu sufixová pole). V závěru je shrnuta celá práce včetně možností jejího dalšího pokračování.

## 2 Teoretický rozbor

Řetězec je obecně definován jako posloupnost prvků  $w$  z dané množiny  $\Sigma$  nazývané abeceda. Biologickými řetězci se nejčastěji rozumí posloupnost dusíkatých bází v sekvencích nukleových kyselin [2]. Šroubovice DNA obsahuje dvě vlákna, která jsou tvořena pětičlennými cukry (spojenými fosfáty), na něž jsou navázány nukleové báze. Schéma 2.1 ukazuje postupně všechny stavební prvky vláken nukleových kyselin, a to cukry ribózu (tvořící základ RNA) a deoxyribózu (její ekvivalent v DNA liší se nepřítomností OH skupiny na druhém uhlíku) a dále kyselinu fosforečnou, ze které je odvozena fosfátová skupina tvořící spojovací mezičlánek jednotlivých sacharidů. U nich je také naznačeno číslování uhlíků v řetězci, které je důležité při určování orientace vláken v molekule (tučnější spoje ilustrují prostorové uspořádání cyklu, nepopsané vrcholy představují atomy uhlíku).

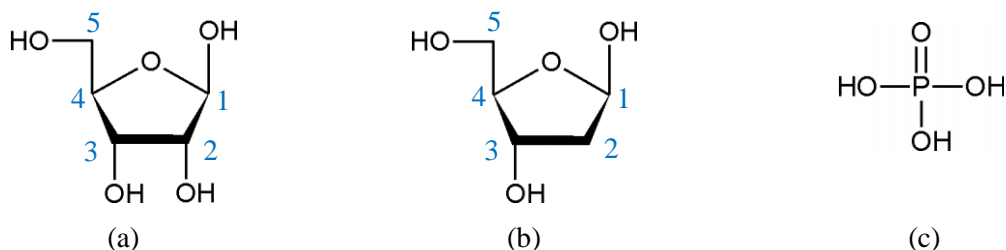


Schéma 2.1: Strukturální vzorce (a) ribózy, (b) deoxyribózy a (c) kyseliny fosforečné.

Genetickou informaci kódují nukleové báze, respektive jejich posloupnost (tedy tzv. primární struktura). V DNA se vyskytují čtyři různé dusíkaté sloučeniny: od purinu odvozené adenin a guanin a z pyrimidinu vzniklé cytosin a thymin. Jejich strukturální vzorce jsou uvedeny ve schématu 2.2 (purin a pyrimidin odpovídají čistým heterocyklům bez dalších navázaných skupin). Běžný [2] je zápis vláken jako sekvence počátečních písmen názvů bází, odtud tedy  $\Sigma = \{A, G, C, T\}$ .

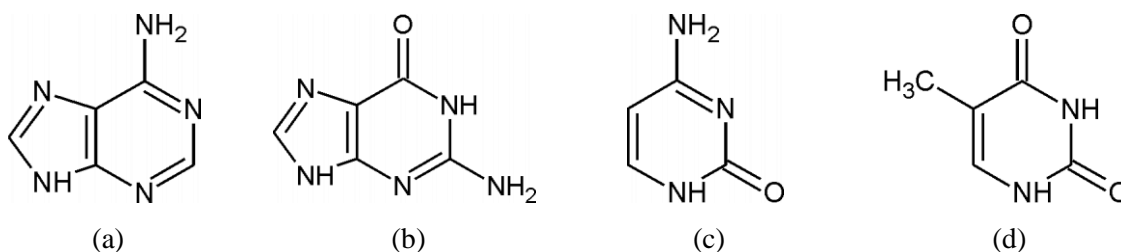


Schéma 2.2: Strukturální vzorce nukleových bází (a) adeninu, (b) guaninu, (c) cytosinu a (d) thyminu.

Tímto zápisem lze obecně popsat celý řetězec, pojem báze je proto často zaměňován s pojmem nukleotid, který označuje jeden stavební prvek vlákna nukleové kyseliny – tedy cukr s dusíkatou bází



navázanou místo OH skupiny na prvním uhlíku a fosfátem (bez kterého by se jednalo pouze o nukleosid [1]) navázaným stejným způsobem na pátém uhlíku. Do vlákna se nukleotidy spojují vazbou přes kyslík z fosfátu na třetí uhlík sacharidového cyklu (dochází opět k nahrazení původní OH skupiny).

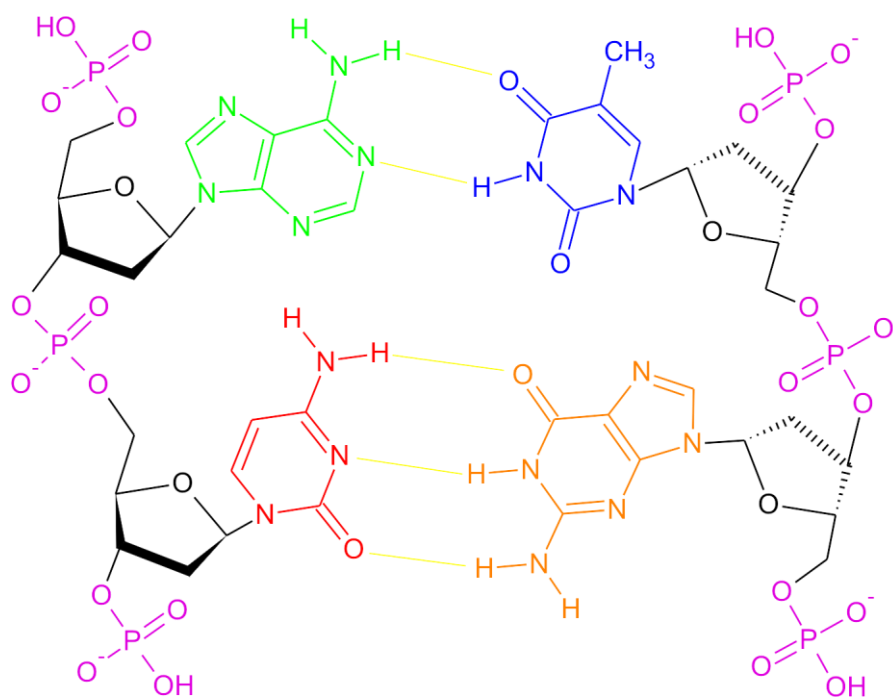


Schéma 2.3: Strukturální vzorec části DNA, žlutě jsou znázorněny vodíkové můstky mezi vlákny.

Z chemických a fyzikálních vlastností následně vyplývá, že obě vlákna jsou spojena právě přes nukleové báze pomocí vodíkových můstků (což je relativně slabá vazba spojující atom vodíku s vysoce elektronegativním atomem, typicky kyslíkem či dusíkem), přičemž adenin tvoří komplementární bázi s thyminem a cytosin s guaninem. Schéma 2.3 obsahuje krátký úsek DNA, přičemž báze adenin, guanin, cytosin a thymin jsou zobrazeny zeleně, oranžově, červeně a modře v tomto pořadí a žluté linie naznačují zmíněné vodíkové můstky. Fialovou barvu mají fosfátové zbytky – díky zápornému náboji na kyslíku je výsledná dvoušroubovice na povrchu silně negativně nabitá.

Na základě číslování atomů uhlíku v cukrech ilustrovaného ve schématu 2.1 lze u fragmentu vlákna vlevo označit horní konec jako 5' a dolní jako 3', u vlákna vpravo pak přesně naopak. Ve schématu 2.4 je zobrazena výsledná obecně známá prostorová struktura DNA [1], ve které lze rozpoznat jak atomy fosforu (znázorněny fialově), tak cykly v sacharidech (pětúhelníky obsahující červený kyslík) a konečně i dusíkaté báze (atomy dusíku jsou modře). Tento tvar, udržovaný a podporovaný vodíkovými můstky, umožňuje bázím v celé molekule zaujmout energeticky velmi výhodnou pozici.

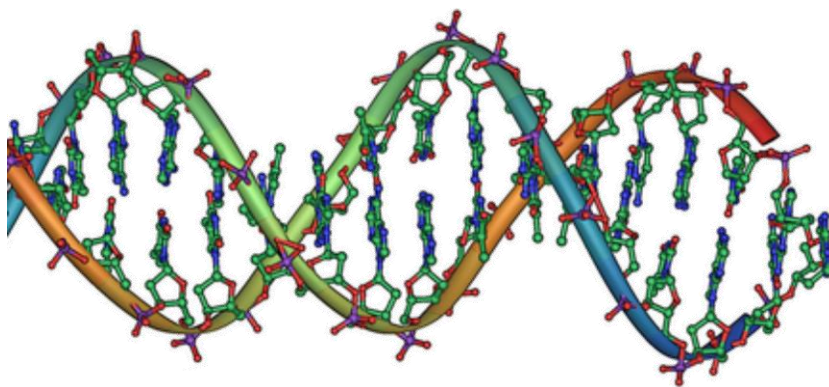


Schéma 2.4: Ilustrace pravotočivé dvoušroubovice DNA.

Sekvence RNA (ribonukleové kyseliny) jsou velmi podobné, od DNA se kromě záměny deoxyribózy za ribózu liší navíc výskytem nukleové báze uracilu, který chemicky odpovídá thyminu bez methylové  $\text{CH}_3$  skupiny (abeceda se mírně pozmění na  $\Sigma = \{A, G, C, U\}$ ). Na rozdíl od DNA ji však tvoří pouze jediné vlákno. Stejně síly, které formují molekuly DNA do pravotočivé dvoušroubovice, působí také u RNA. Dusíkaté báze mají stále tendenci tvořit komplementární páry – vzhledem k absenci druhého vlákna však dochází k tvorbě vodíkových můstků mezi nukleotidy jediného řetězce. Vznikají pak jednodušší i složitější struktury, které se souhrnně označují [1] jako sekundární struktura RNA (primární strukturou se rozumí prostá posloupnost bází).

Výše popsané chování se vztahuje na celé úseky řetězce. Obě spárované části vlákna musí tedy být tvořeny navzájem komplementárními podsekvencemi. Taková posloupnost nukleotidů se anglicky nazývá inverted repeat (její první polovina odpovídá zbylé části zapsané v opačném pořadí a invertované ve smyslu párování bází). Obecně ji lze označit také jako palindrom. Konkrétní příklady útvarů sekundární struktury RNA jsou uvedeny v kapitole věnující se palindromům spolu s jejich přesnou definicí v biologických sekvencích.

Kromě nukleových kyselin jsou analýze podrobovány i další řetězce, jako například posloupnosti aminokyselin (existuje 20 základních), které jednoznačně identifikují bílkovinu (protein). Tímto výčtem je pokryt prakticky celý buněčný proces [1] od transkripce (přepisu) dvouvláknové DNA dočasným rozvolněním vodíkových můstků a komplementárním navázáním volných nukleotidů do jednovláknové RNA. Ta je následně v jiné části buňky (ribozomu – transkripce probíhá přímo v jádře) využita právě k tvorbě řetězce aminokyselin, čímž vzniká bílkovina, základní stavební prvek živých organismů.

## 2.1 Analýza DNA a RNA

Prvním krokem při analýze biologických sekvencí bývá typicky zjištění primární struktury, která obecně představuje složení molekuly na úrovni atomů a (kovalentních) vazeb mezi nimi. V případě

nukleových kyselin, které mají známou [1] stavbu kostry vlákna (realizovanou velmi pevnou fosfodiesterovou vazbou, odtud anglický termín phosphodiester backbone) a dusíkatých bází, se pak jedná o určení posloupnosti nukleotidů v řetězci. K tomu se dnes používá několik laboratorních metod souhrnně označovaných jako sekvenování DNA a RNA. Jejich principy se v některých ohledech značně liší, v jiných jsou naopak takřka totožné (např. rozřídění sekvencí za pomoci gelové elektroforézy).

Výsledkem sekvenování je vždy posloupnost dusíkatých bází v nukleové kyselině (celé či pouze její části), kterou lze dále analyzovat. Největší význam má analýza DNA pravděpodobně v lékařství, kde lze s její pomocí předem odhalit náchylnost jedince k určitému typu chorob a zefektivnit jejich prevenci. Nemoci již probíhající lze lépe diagnostikovat a dokonce i použít léky vyráběné cíleně proti účinkům specifických genů (částí DNA). Do budoucna se uvažuje i o genové terapii (která se začíná rozvíjet u rostlin v podobě geneticky modifikovaných potravin se zvýšenou odolností vůči škůdcům), kdy dokonalá znalost struktury nukleových kyselin je jejím nezbytným předpokladem. Další využití nalézá analýza DNA v kriminalistice (kam zasahuje i forenzní genetika zahrnující určování otcovství) či fylogenetice, která na základě osekvenovaných částí DNA hledá podobnosti mezi živými organismy – tedy genetickou příbuznost všech druhů od jednobuněčných bakterií po člověka.

Velký význam má hledání palindromů v sekvencích RNA z pohledu určování její sekundární struktury. Ta spolu s terciární (uspořádáním a orientací objektu v prostoru) pomáhá chápat principy fungování buněčných procesů, jelikož celkový tvar molekuly ovlivňuje i její další chemické i fyzikální vlastnosti (příkladem může být molekula transferové RNA – tRNA, která díky svému trojlístkovému tvaru může přenášet a navazovat aminokyseliny do proteinového řetězce při translaci – viz dále). V DNA se pak palindromy vyskytují blízko důležitých genů a předpokládá se [3], že hrají podstatnou roli v regulaci jejich interpretace v rámci proteosyntézy. Konkrétně se jedná o určitou vlastnost enzymů, pro které slouží palindromatické úseky DNA jako řídicí sekvence označující začátek či konec genu.

## 2.2 Zarovnání sekvencí

S analýzou biologických řetězců velmi úzce souvisí problematika zarovnání sekvencí. Při buněčných procesech často dochází k pozměnění některých úseků, a to na základě mnoha různých vlivů. Tyto změny, souhrnně označované jako mutace, pak přispívají k variabilitě živých organismů a značně komplikují porovnávání biologických dat. Také při hledání palindromů je nutné brát tyto jevy v potaz (viz následující část popisující přibližné palindromy) a neomezovat analýzu dat na určování přesné shody.

Na každém místě v řetězci nukleové kyseliny může obecně dojít ke třem základním mutacím [2]. Těmi jsou záměna báze za jinou (přičemž u některých úloh můžeme chtít rozlišovat substituci

tranzicí – výměnu za stejný typ báze – a transverzí – nahrazení pyrimidinové báze purinovou a naopak), inserce (vlození jednoho či několika prvků) a delece (proces opačný – tedy vyjmutí libovolného počtu znaků). Jelikož záměna báze v sekvenci nukleotidů je v přírodě mnohem častější než zbývající dvě možnosti (znamenající ve výsledku výskyt mezery v jednom z řetězců), pracují základní metody zarovnání s oběma sekvencemi jako s nedělitelnými.

## 2.2.1 Jednoduché zarovnání pracující pouze se záměnou

V nejjednodušším případě je hledání vhodného zarovnání pouze otázkou nalezení počtu znaků z kratší sekvence, které mají být vynechány před započítáním samotného párování. Tento přístup většinou umožňuje vyhodnotit všechny možné varianty, což se provádí jednoduchým výpočtem za použití dvou konstant – „bonusu“ za každý vzniklý komplementární pár a „pokuty“ pro ostatní páry – jejichž suma dává výsledné skóre. Ve schématu 2.5 je příklad ohodnocení sekvencí CAAGUCAUG a GUGAAG, pro bonus 1 a pokutu 0 by výsledné skóre činilo 4, 2, 0 a 2 (v pořadí zleva doprava).

CAAGUCAUG	CAAGUCAUG	CAAGUCAUG	CAAGUCAUG
GUGAAG---	-GUGAAG--	--GUGAAG-	---GUGAAG

Schéma 2.5: Čtyři možná zarovnání krátkých sekvencí uvažující pouze záměnu znaku.

## 2.2.2 Zarovnání povolující vkládání mezer do řetězce

Zavedením možnosti vložení či vyjmutí znaku na libovolné pozici v řetězci velmi rapidně narůstá počet možných zarovnání. Pro sekvence použité v předchozím příkladu jednoduchého zarovnání existovaly čtyři možnosti jak spárovat znaky. Povolněním umístění tří mezer na libovolná místa v kratší sekvenci (vkládat mezery do delšího řetězce zde nemá velký smysl) se toto číslo zvyšuje na 28. Ve schématu 2.6 jsou uvedeny čtyři vybrané varianty.

CAAGUCAUG	CAAGUCAUG	CAAGUCAUG	CAAGUCAUG
GUG-A--AG	G-U-G-AAG	G-U--GAAG	GU---GAAG

Schéma 2.6: Čtyři vybrané varianty zarovnání dvou sekvencí s možností vložení mezer.

Zohlednit při vyhodnocování podobnosti vyjmuté či přidané znaky lze provést zavedením další „pokutující“ konstanty – tentokrát za vložení mezery. Do výsledku se započítá v případě, že v jednom z řetězců se na zkoumané pozici nachází speciální znak zastupující mezeru. Přidáním této konstanty (o hodnotě např. -1) k dříve uvedeným vzniká vztah pro výpočet ohodnocení podobnosti řetězců (označovaný jako skórovací funkce, formálně zapsaný ve schématu 2.7 – obecně zkoumanou shodu znaků lze ve specifických případech nahradit komplementaritou), na jehož základě jsou příklady uvedené ve schématu 2.6 postupně zleva doprava ohodnoceny skórem 1, 0, 1 a 1. Druhá varianta se

tedy jeví jako nejméně pravděpodobná mutace ze zkoumaných možností při porovnání na základě uvedených kritérií.

$$\sum_{i=1}^n \begin{cases} \text{pokuta za mezeru; } (S1_i = "-") \vee (S2_i = "-") \\ \text{bonus za shodu; } (S1_i = S2_i) \wedge (S1_i \neq "-") \wedge (S2_i \neq "-") \\ \text{pokuta za odlišnost; } (S1_i \neq S2_i) \wedge (S1_i \neq "-") \wedge (S2_i \neq "-") \end{cases}$$

Schéma 2.7: Skórovací funkce pro výpočet ohodnocení zarovnání dvou řetězců  $S1$  a  $S2$ .

V úvahách o vkládání mezer lze jít ještě dál. Statisticky je totiž mnohem častější jediné vložení či vyjmutí souvislé sekvence znaků, než vícenásobné kratší mutace. Tuto skutečnost lze promítnout do hodnocení rozdělením pokuty za vložení mezery na dvě části: za počáteční mezeru (započetí nové podsekvence mezer v jednom ze zarovnávaných řetězců) a za trvající mezeru (prodloužení již existující série). Pokud použijeme tyto konstanty (o hodnotách např. -2 a -1 v tomto pořadí) na příklad ze schématu 2.6, změní se výsledné skóre na -1, -3, -1 a 0. Zavedením tohoto kritéria dosáhlo poslední zarovnání nejvyššího skóre, zatímco před rozdělením pokuty za mezeru nebylo možné z hodnot jednoznačně určit nejlepší variantu. Důvodem je právě přítomnost jediné delší mezery (chybějící sekvence znaků) namísto více kratších.

### 2.2.3 Globální a lokální zarovnání

Kromě rozlišení, zda jsou mezery vloženy bezprostředně za sebou nebo izolovaně, se lze zajímat také o to, zda se vyskytují uvnitř nebo vně řetězce. Praktické je to zejména v případech, kdy se délky zkoumaných sekvencí podstatně liší a navíc je kratší z nich velmi podobná některému úseku delší. Například při porovnávání CAGCUACGAC s GAUGC se jako neoptimálnější varianta jeví vložení tří mezer na začátek a dvou na konec kratšího řetězce, jelikož poté všechny jeho znaky tvoří komplementární dvojice s nukleotidy delší sekvence, jak ilustruje schéma 2.8(a). Pokutováním těchto okrajových mezer se citelně sníží výsledné skóre. U některých typů úloh tedy nejsou penalizovány mezery na začátku a konci řetězců. Tímto se od globálního zarovnání přechází k pologlobálnímu (též semiglobálnímu [2]).

Ovšem ani semiglobální zarovnání nemusí být pro potřeby porovnávání postačující. Při hledání palindromů jsou předmětem zájmu kratší úseky velmi dlouhých sekvencí, které odpovídají definici palindromu. Podobná situace nastává při hledání podobných podřetězců ve velmi objemných datech (např. nalezení částí DNA společných pro člověka a šimpanze). Semiglobální zarovnání v takovém případě vyprodukuje velké množství pokut a nalezení zajímavých úseků bude velmi obtížné.

Zde se používá tzv. lokální zarovnání, které přesně odpovídá požadavkům uvedeného příkladu. Jeho cílem je nalézt nejpodobnější (případně nejkomplementárnější) podsekvence z obou

zkoumaných řetězců, dochází tedy vlastně k jejich oříznutí. Optimální semiglobální a lokální zarovnání dvou stejných vstupů (AAUGCAGAUUC a ACUGGUCU) je pro ilustraci uvedeno ve schématu 2.8(b) a (c). Obě možnosti sice při použití dříve uvedené skórovací funkce dosáhnou stejného ohodnocení (4), ovšem pouze druhý přístup odhalí ty části obou sekvencí, které jsou dokonale komplementární.

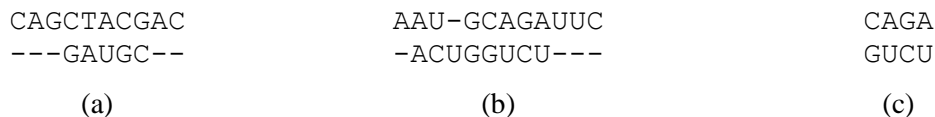


Schéma 2.8: Příklad zarovnání (a), (b) semiglobálního a (c) lokálního.

## 2.3 Definice a význam palindromů

Nad každou abecedou jsou palindromy definovány jako řetězce, které jsou totožné ke svým reverzním formám (kdy jsou prvky uspořádány v opačném pořadí). Palindromy v literárních abecedách se tedy vyznačují tím, že při čtení zleva i zprava tvoří stejný text. Mezi běžně uváděné příklady patří slova „radar“, „tahat“ či „nezařazen“, existují však i celé palindromatické věty (při jejichž posuzování se většinou neberou v úvahu mezery mezi slovy, interpunkce, délky samohlásek a rozdíl v i/y): „Kobyly má malý bok.“ „Jelenovi pivo nelej.“

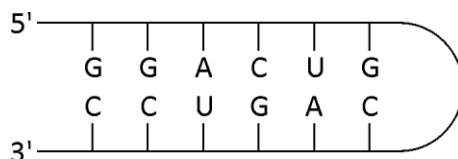


Schéma 2.9: Stopka (stem) v RNA tvořená palindromatickým řetězcem dusíkatých bází.

Poněkud praktičtější význam mají palindromy vyskytující se v biologických sekvencích bází nukleových kyselin. V tomto případě je definice palindromu [4] mírně odlišná – sekvence je označena jako palindromatická, pokud je shodná s reverzním řetězcem, ovšem ve formě komplementárních bází. Vyskytne-li se např. v řetězci RNA posloupnost GGACUGCAGUCC, jedná se o palindrom, jelikož reverzní řetězec je komplementární k původní sekvenci. Navíc je první polovina nukleotidů komplementární ke zbylým bázím v opačném pořadí. Tato vlastnost je významná především z pohledu určování sekundární struktury RNA (což je krok následující bezprostředně po určení posloupnosti dusíkatých bází – primární struktury), jelikož palindromatické úseky vytvářejí tzv. stopky (angl. stem). Ty se následně vlivem působení dalších sil stáčí do dvoušroubovice velmi podobné té, jaká je známá [1] z uspořádání molekuly DNA. Schéma 2.9 ilustruje stopku tvořenou

bázemi uvedené sekvence (5' a 3' je rozlišení konců vlákna – konvencí je zápis vlákna právě v tomto směru).

Kromě sekundární struktury RNA jsou palindromatické sekvence nukleotidů zajímavé i z pohledu dalších buněčných procesů. Týká se to zejména jisté redundance informace (z půlky palindromu lze velmi jednoduše odvodit zbytek řetězce) a tím i možnosti použití mechanismů opravy chyb vzniklých při replikaci genetického kódu. Palindromy v DNA se navíc vyznačují podobnou vlastností jako palindromy literární – poskytují stejnou informaci při interpretaci z obou konců. Z pohledu buněčných dějů tedy není (na rozdíl od nepalindromatických struktur) rozhodující, zda je daný úsek DNA při transkripci přepisován do jednovláknové RNA z 5' nebo 3' konce řetězce – výsledná sekvence je vždy stejná.

### 2.3.1 Sudé a liché palindromy

Všechny palindromy uvedené v předcházejícím textu jsou dle běžně používané klasifikace [4] označovány jako přesné (angl. exact). Navíc byly uvažovány pouze palindromy maximální, tedy takové, které při rozšíření o jeden znak na obou svých koncích již netvoří přesný palindrom (eliminují se tím opakování se stejným středem, ale menším počtem znaků, např. centrální podřetězec UGCA z předchozího příkladu je zajisté také palindromem).

Jak již bylo uvedeno, nad literárními abecedami se za palindrom považuje takový řetězec  $w$ , který se čte stejně zleva i zprava, čili je shodný s řetězcem  $w^R$  (což zde označuje sekvenci znaků zapsanou v opačném pořadí), tedy  $w = w^R$ . Tato rovnice platí i pro biologická data,  $w^R$  v tomto případě zastupuje reverzní řetězec komplementárních bází. Právě díky komplementaritě lze každý biologický palindrom vyjádřit také vzorem  $w.w^R$ , který lépe vystihuje schopnost těchto úseků vláken RNA tvořit stopky. Vzhledem k celkovému počtu prvků v palindromu ( $2 \cdot |w|$ ) jsou sekvence tohoto typu označovány jako sudé palindromy.

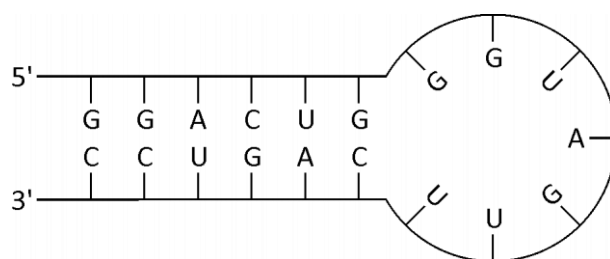


Schéma 2.10: Smyčka (loop) v RNA vložená do přesného palindromu.

Oproti nim existují i liché palindromy, které odpovídají vzoru  $w.c.w^R$ , kde  $c$  značí tzv. centrální prvek, který netvoří komplementární dvojici. Při určování sekundární struktury RNA se zkoumají i takové liché palindromy, kde  $|c| > 1$ , tedy kdy mezi řetězec  $w$  a  $w^R$  je vložen celý další

nepalindromatický podřetězec. Tato sekvence nukleotidů pak tvoří tzv. smyčku (angl. loop) a její délka spolu s délkou opakující se části (konkrétně jejich poměr) jsou podstatnými parametry při určování kvality daného palindromu. Ve schématu 2.10 je opět stopka tvořená palindromem GGACUGCAGUCC, tentokrát však doplněna o smyčku GGUAGUU.

Sekundární struktura tRNA obsahuje hned tři podobné smyčky [1], ve kterých je vždy čtyři až pět spárovaných bází doplněno o podřetězec tvořený sedmi (případně osmi) nukleotidy. Centrální trojice prostřední smyčky (tzv. antikodon) jednoznačně určuje, která aminokyselina je danou tRNA přenášena (transportována – odtud označení transferová RNA), přičemž jedné aminokyselině může odpovídat více antikodonů na tRNA. Obecná sekundární struktura tRNA je naznačena ve schématu 2.11 – lze zde pozorovat hned několik palindromatických úseků.

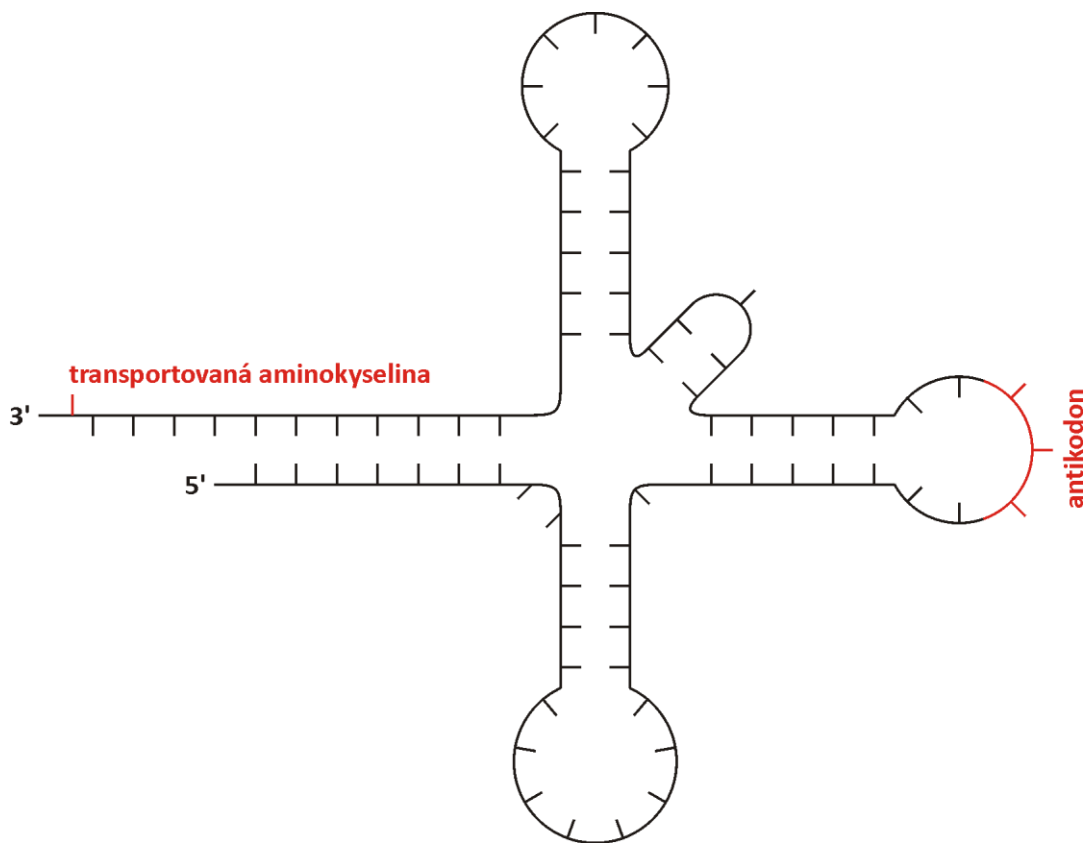


Schéma 2.11: Obecná sekundární struktura tRNA.

Vlastní proteosyntéza, při které vzniká výsledná bílkovina, probíhá na ribozomu, a to komplementární interakcí antikodonu tRNA s kodonem (opět trojicí nukleotidů) na vlákně mRNA (z angl. messenger – informační RNA), která je produktem již zmíněné transkripce DNA. Při tomto procesu dojde k dočasnému vytvoření vodíkových můstků mezi tripletem bází mRNA a tRNA, navázání právě přenášené aminokyseliny do polypeptidického řetězce proteinu, přerušení vazebných



interakcí a posunu mRNA v ribozomu o tři nukleotidy (toto vlákno je vlastně sekvenčně čteno a interpretováno po trojicích znaků).

## 2.3.2 Přibližné palindromy

Výskyt přesných palindromů v reálných vláknech RNA a DNA je omezen na velmi krátké úseky. Naprosto běžné jsou naopak drobné odchylky v jednom či druhém zkoumaném řetězci, které lze primárně rozdělit na záměnu (vzniká neshoda – angl. mismatch) a vložení či vynechání znaku (angl. insertion, deletion). V prvním případě dojde k nahrazení nukleotidu v jednom z podřetězců za jiný, který již není komplementární s odpovídající bází zkoumaného reverzovaného úseku. Výsledkem druhé uvedené transformace je pak deformace struktury RNA anglicky označovaná jako bulge, která je tvořena buď vloženým elementem do jedné části sekvence, nebo nukleovou bází, jejíž komplementární prvek byl odstraněn ze spárovaného vlákna. Tuto nepravidelnost může opět obecně tvořit i více nukleotidů.

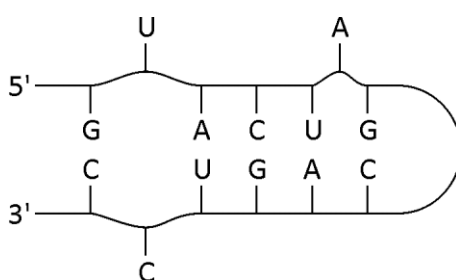


Schéma 2.12: Deformace (bulge) vzniklé záměnou (vlevo) a vložení (vynecháním) báze (vpravo).

Při sekvenčním postupu určování sekundární struktury RNA je často nemožné rozpoznat, zda neshoda dvou zarovnaných nukleotidů vznikla záměnou jedné z bází, či vložení (respektive vynecháním) prvku do jednoho (z druhého) vlákna. Působením fyzikálních sil totiž i při záměně vzniká deformace, tentokrát oboustranná. Tato struktura je zobrazena ve schématu 2.12 za použití řetězce z předchozích příkladů. Zde došlo jednak k záměně guaninu (druhý nukleotid v pořadí) za uracil, čímž se na opačné straně vyčlenil z vlákna i cytosin, a dále zřejmě ke vložení adeninu. Bez znalosti původní struktury lze však vzniklou nepřesnost v párování interpretovat také jako vynechání příslušné komplementární báze (zde by se jednalo o uracil) na opačné straně stopky.

Všechny uvedené deformace včetně smyček mají také význam při dalším určování struktury RNA, jelikož nespárované nukleové báze mohou mezi sebou nadále tvořit vodíkové můstky a vytvořit tak např. spojení dvou smyček (angl. kissing hairpins) či smyčky s nespárovanou částí vlákna atp. Pro úlohu hledání palindromů je však podstatnější zavedení zmíněné klasifikace změn v řetězcích, na jejímž základě je v algoritmech rozlišována kvalita přibližného palindromu nejčastěji v závislosti na poměru počtu a typu chyb na celkové délce řetězce.

## 2.4 Algoritmy pro hledání palindromů

Již základní rozhodovací problém, zda daný řetězec je či není palindrom, může být výpočetně velmi náročný. Ve formálních jazycích je množina všech palindromatických řetězců nad příslušnou abecedou typicky jazyk bezkontextový, který není přijímán deterministickým zásobníkovým automatem. Odtud lze odvodit později uvedenou prostorovou (paměťovou) náročnost algoritmů, která se odvíjí od faktu, že střed palindromu (ať už lichého či sudého) nelze zjistit dříve, než je zpracována celá vstupní sekvence.

Celková náročnost dále stoupá se zobecněním na hledání všech přibližných palindromů (určitých parametrů) v řetězci. Zde již nelze efektivně použít přímočaré přístupy, které za určitých podmínek postačují pro určení, zda na vstupu je či není palindrom (např. uvažovat postupně jako střed palindromu každý jednotlivý znak a to navíc dvakrát – pro lichý a sudý palindrom). Vzniklo proto několik algoritmů, které pracují s různými datovými strukturami pro efektivnější přístup k datům, přičemž některé již z principu nabízejí možnost paralelizace, což je pro hardwarovou akceleraci klíčový předpoklad.

### 2.4.1 Dynamické programování

Řešení úlohy metodou dynamického programování je možné vždy, pokud lze tuto úlohu rozdělit na menší celky a jejich výsledky (které algoritmus uchovává v paměti) opakovaně použít při vyhodnocování původního problému. Typickými příklady, které se v této souvislosti uvádějí [5], jsou rekurzivní algoritmy jako výpočet faktoriálu či Fibonacciho čísel. Jejich definice jsou uvedeny ve schématu 2.13 – do paměti lze vždy uložit všechny již vyhodnocené výsledky a při dalším dotazu tak přímo vrátit příslušnou hodnotu, případně pokračovat ve výpočtu od poslední uložené pozice.

$$\begin{array}{ll} 0! = 0 & Fibb(0) = 0 \\ 1! = 1 & Fibb(1) = 1 \\ n! = n \cdot (n-1)! & Fibb(n) = Fibb(n-1) + Fibb(n-2) \end{array}$$

(a) (b)

Schéma 2.13: Rekurzivní definice výpočtu (a) faktoriálu a (b) Fibonacciho čísla.

Podobným způsobem lze sestavit také čistě rekurzivní algoritmus pro hledání palindromů nad řetězcem, respektive pro nalezení optimálního zarovnání sekvence a její reverzované formy s povoleným vkládáním mezer do obou z nich (za podmínky, že nikdy nedojde ke spárování dvou mezer). Algoritmus zpracovává vstupy po znacích a postupně je zkracuje. V každém kroku může nastat jedna z následujících variant: 1) zarovnání znaků z obou sekvencí, 2) vložení mezery do

původního řetězce, 3) vložení mezery do reverzované posloupnosti. V prvním případě se pokračuje se vstupy zkrácenými o první znak, ve zbylých dvou je pak odebrán prvek jen z jedné sekvence. Takto se pokračuje, dokud není dosaženo dvou prázdných řetězců na vstupu (ukončující podmínka rekurze).

Na základě výše uvedených možností vzniká strom (kde každý vnitřní uzel má právě tři následovníky), který uchovává všechna možná zarovnání a musí zajisté obsahovat také optimální spárování a tedy i nejkvalitnější palindrom obsažený ve vstupní sekvenci. Ten odpovídá zpětnému průchodu stromem za použití skórovací funkce (viz schéma 2.7), pomocí které je nalezeno nejlépe ohodnocené zarovnání (cesta s nejmenším výsledným skóre). Časová složitost takového postupu je však exponenciální, konkrétně  $O(3^n)$  v závislosti na délce vstupu, což jej činí prakticky nepoužitelným. Mnoho stejných dvojic podřetězců je totiž vyhodnocováno opakovaně a tedy zbytečně.

Pro potřeby uchování mezivýsledků používá dynamické programování dvourozměrné pole, čili tabulku, jejíž sloupce odpovídají postupně zleva doprava jednotlivým prvkům zkoumané sekvence a řádky pak odshora dolů prvkům reverzovaného řetězce (jedná se o typickou datovou strukturu pro vyhledávání podle dvou indexů – přístup k libovolnému prvku je možný v konstantním čase [4]). Vedlejší diagonála takové matice pak představuje všechny možné pozice centrálních prvků lichých palindromů obsažených ve vstupu. To samé platí pro sousední horní rovnoběžnou diagonálu a sudé palindromy. Inicializace struktury spočívá ve vyplnění všech těchto prvků (středů potenciálních palindromů) hodnotou nula, která obecně indikuje, že na této diagonále (tentokrát souběžné s hlavní diagonálou) se nevyskytla žádná chyba oproti definici přesného palindromu.

$$D = \min \begin{cases} A; \text{ shoda znaků} \\ A + \text{ pokuta za substituci; neshoda znaků} \\ B + \text{ pokuta za vložení} \\ C + \text{ pokuta za vyjmutí} \end{cases}$$

Schéma 2.14: Výpočet parciálního skóre D z hodnot A, B a C přičtením pokut a výběrem minima.

Vlastní výpočet, spočívající v zaplnění pravé dolní trojúhelníkové matice daty, probíhá stejným způsobem jako při výpočtu skóre zarovnání dvou sekvencí (např. algoritmy Needleman-Wunsch a Smith-Waterman [2]). Pro každou buňku tabulky je nutno uvažovat následující možnosti: 1) vyjmutí znaku ze sekvence je reprezentováno přičtením pokuty k hodnotě nalevo od aktuální pozice, 2) analogicky vložení znaku odpovídá inkrementaci horní sousední hodnoty o příslušnou pokutu, a 3) dle příslušných znaků na okrajích tabulky se určí buď bonus za shodu, či pokuta za rozdílnost prvků (ekvivalentní záměně nukleotidu) a ta je přičtena k ohodnocení buňky vlevo nahoře ve směru hlavní diagonály. Jelikož cílem algoritmu je nalézt co nejkvalitnější palindromy, je z uvedených možností vybrána nejmenší hodnota, která je následně přiřazena zpracovávané buňce. Tímto

způsobem (formálněji popsaným ve schématu 2.14) jsou vypočteny všechny potřebné hodnoty parciálních skóre. Z uvedeného postupu vyplývá, že pro výpočet hodnoty každé buňky je nutno mít k dispozici tři ohodnocení (běžně používaná notace je naznačena také v pravé dolní části schématu 2.15(a) – z hodnot A, B, C je získáno skóre D) a příslušné znaky na okrajích tabulky pro účely porovnání.

Schéma 2.15 dále ukazuje datovou strukturu pro řetězec MISSISSIPPI (který obsahuje několik palindromů). Vlevo 2.15(a) je tabulka po inicializaci (zeleně podbarvené buňky) s naznačeným postupem výpočtu (červená linie – fakt, že v jednom kroku lze spočítat všechny hodnoty na vedlejší diagonále, je velmi důležitý pro paralelizaci a tedy i následnou hardwarovou akceleraci). Ten probíhá přesně dle stěžejní myšlenky dynamického programování, tedy od nejjednodušších podproblémů (v tomto případě jsou vlastně ohodnoceny všechny potenciální palindromy tvořené dvěma znaky) k vyřešení celku. Ve schématu vpravo 2.15(b) je pak výsledek výpočtu (bylo použito uniformní pokuty pro vložení, vynechání i záměnu znaku, a to hodnoty 1).

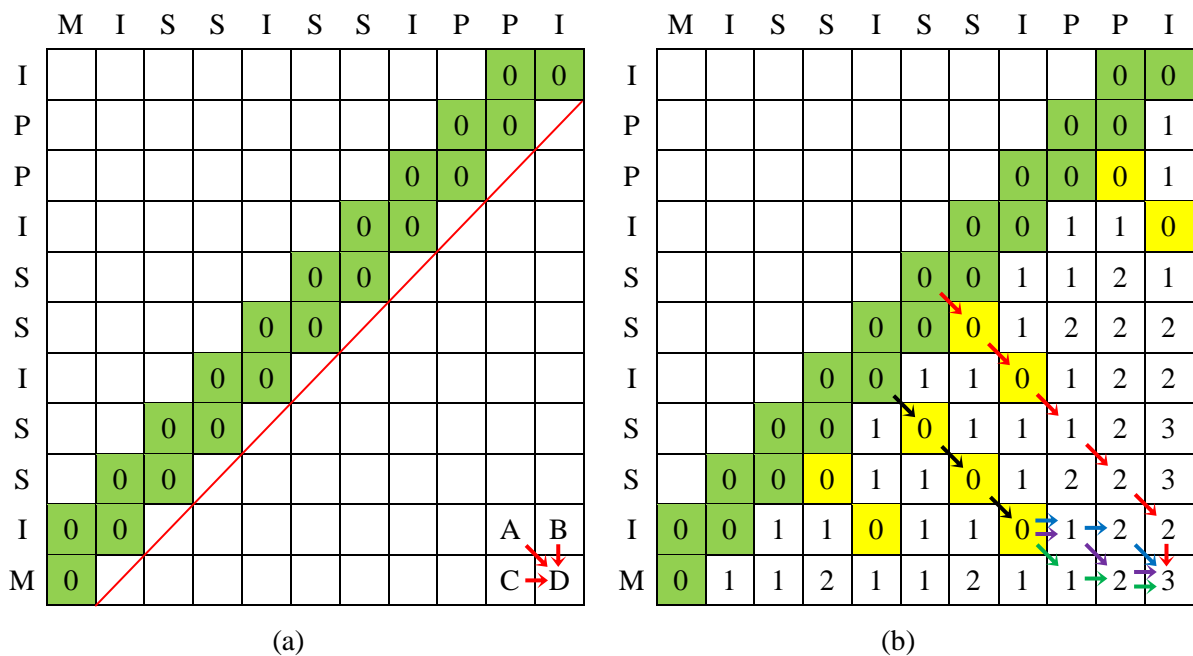


Schéma 2.15: Tabulka parciálních skóre (a) po inicializaci, (b) po skončení výpočtu.

Každé parciální skóre v tabulce lze interpretovat jako ohodnocení optimálního zarovnání (nejlepší možný výsledek) podřetězce mezi znaky danými číslem sloupce a řádku (tedy začátek dle písmena v reverzované posloupnosti a konec odpovídající indexu v původní sekvenci). Minimální skóre pro globální zarovnání plné délky obou řetězců je proto v pravém dolním rohu tabulky (dle předchozího popisu od prvního po poslední písmeno – analogicky s výpočtem ohodnocení podobnosti řetězců algoritmem Needleman-Wunsch [2]), zde se jedná o hodnotu tři. Rekonstrukci tohoto zarovnání lze provést zpětným průchodem, začínajícím v pravém dolním rohu a postupujícím zpět

k vedlejším diagonálám inicializovaným na hodnotu nula. V každém kroku je nutno analyzovat, jakým způsobem bylo dosaženo aktuální hodnoty (viz schéma 2.14) a podle toho provést spárování znaků (posun po diagonále) či vložení mezery do původní (reverzované) posloupnosti v případě vertikálního (horizontálního) pohybu.

Ve schématu 2.15(b) mohlo hned v prvním kroku dojít ke všem třem možnostem (započtení pokuty za jednu z mezer či za záměnu znaku – porovnává se „M“ a „I“) a je proto nutné sledovat všechny možné větve. Pokud cesta končí přímo na vedlejší diagonále (černé šipky), je výsledný přibližný palindrom lichý s centrálním prvkem odpovídajícím příslušnému znaku v tomto sloupci a řádku. Dospěje-li hledání až na sousední horní diagonálu (červené šipky), je sestaven sudý palindrom. Schéma 2.16 ilustruje všechna čtyři optimální zarovnání (barvy odpovídají šipkám ve schématu 2.15(b) – černá cesta zároveň odpovídá nejdelšímu přesnému palindromu ISSISSI), všechna s ohodnocením 3. Přibližný palindrom 2.16(a) obsahuje přesný palindrom ISSI (se začátkem na pátém znaku sekvence), díky spárování obou zbývajících písmen „I“ je výsledné skóre stejné jako ve zbývajících případech.



Schéma 2.16: Čtyři výsledná optimální globální zarovnání.

Při analýze dlouhých sekvencí je žádoucí použití lokálního zarovnání namísto globálního. Tím lze nalézt přesné i přibližné palindromy uvnitř řetězce bez ohledu na jeho délku a ohodnocení nezarovnaných částí. Výše popsaná interpretace parciálního skóre na základě dvou indexů v řetězci může být nahrazena popisem každé buňky tabulky pomocí identifikace diagonály, na které leží (určuje střed palindromu), a vzdálenosti od vedlejší diagonály. Ta udává poloměr, tj. délku poloviny palindromu (počet znaků nalevo i napravo od středu – při posunu po diagonále jsou spárovány dva znaky). Hodnota je stále minimálním skóre optimálního zarovnání takového palindromu.

Každá diagonála rovnoběžná s hlavní diagonálou tedy reprezentuje jeden potenciální palindrom. Pro získání všech přesných palindromů je nutno uvažovat pouze ta pole, které obsahují ohodnocení nula (tedy žádná chyba – ve schématu 2.15(b) jsou celkem čtyři takové palindromy podbarvené žlutě, tento výsledek navíc zároveň odpovídá výsledku hledání nejdelší společné předpony LCP popsané v sekci věnující se sufixovým stromům a polím). Na základě pozměněné interpretace lze v uvedeném příkladu identifikovat čtyři přesné palindromy, a to dvakrát ISSI (se začátkem na druhém a pátém znaku zkoumané sekvence), dále IPPI a nejdelší ISSISSI.

Pro detekci přibližných palindromů z uvedené matice je nutno definovat jejich kvalitu. Nejjednodušším způsobem je určit hodnotu  $k$ , která udává počet chyb, které se v daném palindromu

vyskytly. Z výzkumu biologických sekvencí [3] ovšem vyplývá mimo jiné i požadavek na toleranci většího počtu chyb s rostoucí délkou palindromu. Zavádí se proto relativní hodnota  $e$ , která udává podíl počtu chyb  $k$  vůči délce palindromu  $l$ . V datové struktuře dynamického programování odpovídají ohodnocení jednotlivých buněk právě parametru  $e$  (platí pro uvedenou uniformní penalizaci všech změn v řetězci – při různých hodnotách by bylo nutné zavést upravený parametr  $e_s$ , který by rozdílné pokuty reflektoval). Je-li tedy úloha definována jako hledání všech přibližných palindromů s danou maximální hodnotou relativní chyby  $e$ , dojde k tzv. exportu (tj. uložení souřadnic) na všech buňkách, jejichž hodnota přesáhla daný práh  $e$  a zároveň v předchozím kroku (zpět po diagonále) nebyl tento práh ještě překročen. Dále se exportují i pozice, které představují palindrom, jehož ohodnocení je stále v daných mezích, ale přitom nemůže být dále rozšiřován (výpočet skončil). Při použití tohoto mechanismu dojde k exportu všech předpokládaných přibližných palindromů (pro hodnotu  $e = 0$  pak pouze přesných), ale obvykle také k exportu na obou jeho sousedních diagonálách (které mají jen nepatrně horší skóre – jde o právě exportovaný palindrom s přidaným znakem na svém konci či začátku). Je žádoucí tato data eliminovat, jelikož nemají žádnou přidanou hodnotu a navíc zvyšují objem výstupu.

Analýza reálných sekvencí [3] také ukázala, že s narůstající délkou potenciálního palindromu klesá jeho kvalita exponenciálně. Není tedy nutné počítat celou trojúhelníkovou matici, ale pouze několik diagonál rovnoběžných s vedlejší diagonálou (což v důsledku omezuje maximální možnou délku nalezených palindromů). Tím se snižuje počet buněk, jejichž hodnotu algoritmus počítá, čímž dochází ke snížení jeho časové náročnosti (která bude odvozena později).

## 2.4.2 Metody používající sufixové stromy a sufixová pole

Přípony (sufixy) sekvence tvoří množina všech podřetězců, počínaje samotným vstupem, vzniklých postupným odebíráním vždy prvního znaku sekvence až do získání prázdného řetězce. Tuto množinu lze reprezentovat za použití různých datových struktur, které umožňují rychlé a efektivní operace (typicky s lineární časovou složitostí ve vztahu k délce řetězce) nad vstupní sekvencí.

Sufixový strom reprezentuje vstupní řetězec formou orientovaného stromu, přičemž každá jeho hrana je ohodnocena jedním symbolem ze vstupní abecedy a z jednoho uzlu nevede více hran, které by byly ohodnoceny stejným znakem. Každá cesta tímto stromem (dle orientace od kořene k listům, jejichž počet odpovídá délce vstupu) pak tvoří právě jeden sufix vstupního řetězce. Jsou používány i optimalizace počtu uzlů, při nichž jsou redukovány uzly mající právě jednoho potomka. Nová hrana je pak ohodnocena podřetězcem vzniklým konkatenací původních znaků (vzniká tzv. Patricia trie).

Schéma 2.17 ilustruje takovýto optimalizovaný strom reprezentující sufixy vzorové sekvence (symbol  $\varepsilon$  je použit k vyjádření nejkratšího sufixu  $C$ , tedy posledního znaku původní sekvence). Sufixové stromy lze sestavit v lineárním čase [5]. Jedna z metod používá další hrany přidané do grafu, tzv. sufixové spoje (angl. suffix links), které vedou od každého vnitřního uzlu (kromě

kořenového) k takovému, který reprezentuje sufix vzniklý odebráním prvního znaku (v optimalizované verzi i více prvních znaků) z podřetězce kódovaného výchozím uzlem. Další typ algoritmů se vyznačuje zpracováváním vstupní sekvence znak po znaku a průběžnou (on-line) tvorbou stromu [6].

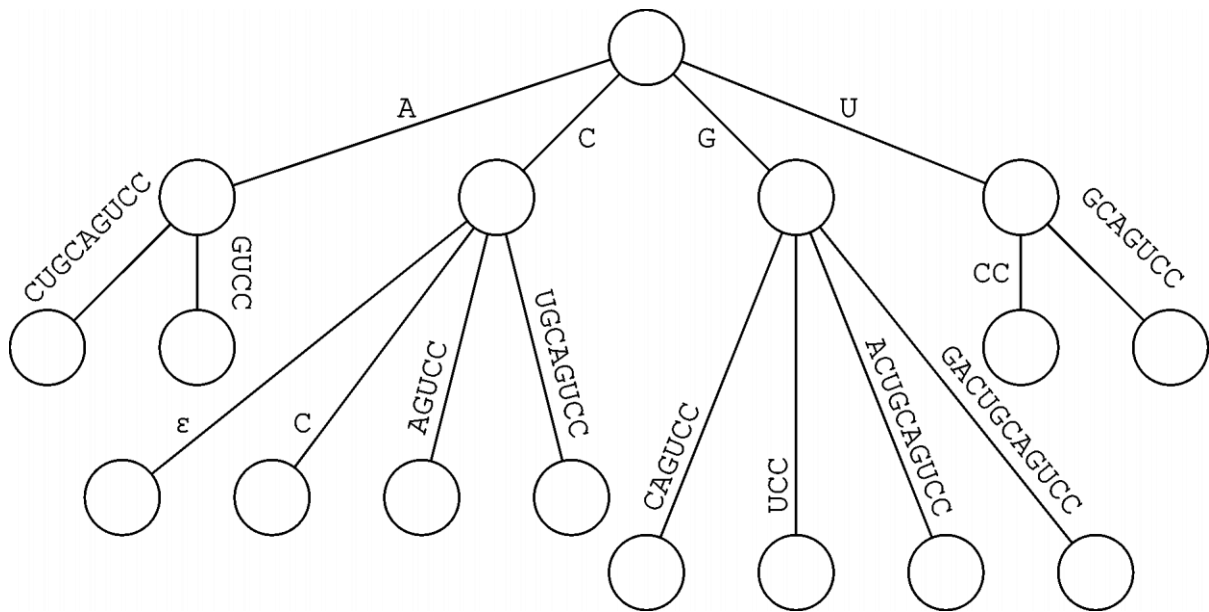


Schéma 2.17: Sufixový strom reprezentující vzorovou sekvenci.

V případě sufixového pole se jedná o jednorozměrné pole indexů, které označují pozici (index) prvního znaku přípony v původním řetězci. Smyslem pole je lexikografické (slovníkové) uspořádání všech možných přípon sekvence dané pořadím indexů v poli. Schéma 2.18 ukazuje rozklad sekvence GGACUGCAGUCC na sufixy, jejich lexikografické uspořádání a přiřazení hodnot indexů. Třetí sloupec udává délku nejdelší společné předpony vzhledem k předchozímu sufixu (LCP – viz dále). Časová složitost vytvoření sufixového pole se pohybuje od  $O(n^2 \log n)$  pro nejjednodušší metodu založenou na postupném porovnávání a řazení sufixů až po nejefektivnější lineární  $O(n)$  složitost, publikovanou např. v [7]. Existuje také řada algoritmů pro převod sufixového stromu na pole a obráceně.

Často používaným mezikrokem pro operace nad řetězci (včetně některých metod pro hledání přibližných palindromů) je nalezení nejdelší společné předpony (angl. longest common prefix – LCP) dvou sekvencí. Sufixový strom umožňuje provedení této operace v konstantním čase [5], u pole se pak často ukládá ke každému prvku hodnota LCP vztahená na předchozí sufix (viz schéma 2.18 – minimum z těchto hodnot pak udává délku LCP nad množinou všech sufixů). Úlohu nalezení všech nejdelších možných přesných palindromů v řetězci lze pak řešit s lineární časovou složitostí, při akceptaci mezer (vzniklých vložením či vynecháním znaku) délky  $g$  se složitostí  $O(gn)$  a konečně s tolerancí  $k$  chyb (záměn znaků) se složitostí  $O(kn)$ .

<b>sufix</b>	<b>index</b>	<b>LCP</b>
ACUGCAGUCC	2	0
AGUCC	7	1
C	11	0
CAGUCC	6	1
CC	10	1
CUGCAGUCC	3	1
GACUGCAGUCC	1	0
GCAGUCC	5	1
GGACUGCAGUCC	0	1
GUCC	8	1
UCC	9	0
UGCAGUCC	4	1

Schéma 2.18: Sufixové pole indexů a hodnot LCP.

Symetricky k LCP existuje také nejdelší společná přípona neboli prodloužení (angl. longest common extension – LCE), tedy maximální počet totožných znaků následujících po dvou různých prvcích sekvence. Sufixový strom řeší tento problém v konstantním čase pro libovolné dvě počáteční pozice [5]. Potom není složité sestavit algoritmus [8, 11], který s lineární časovou složitostí nalezne všechny maximální palindromy ve vstupním řetězci.

Z uvedených vlastností lze odvodit jednu z nevýhod těchto struktur – výměnou za nízkou (většinou lineární) časovou složitost při jejich sestavování a následných operacích nad řetězci mají vyšší než lineární prostorovou složitost právě z důvodu předzpracování vstupní sekvence do formy datové struktury (sufixové stromy navíc narůstají se zvyšující se mocností abecedy). Paměťová náročnost dále stoupá, je-li použit algoritmus, který při hledání přibližných palindromů v řetězci  $w$  pracuje se vstupem  $w.w^R$ , který má dvojnásobnou délku oproti originálu.

## 2.5 Technologie FPGA

Nástrojem zvoleným pro realizaci akcelerační jednotky je FPGA (z angl. field-programmable gate array), což je čip obsahující pole programovatelných logických bloků a propojů. Jeho hlavní výhodou je rekonfigurovatelnost, tedy možnost kompletní změny vykonávané funkce. K jejímu zápisu se používají speciální programovací jazyky pro popis hardwaru (angl. hardware description language – HDL), mezi něž patří především v USA rozšířený Verilog a převážně evropský VHDL („V“ je první písmeno anglické zkratky VHSIC – velmi rychlé integrované obvody), který je použit i pro implementační část této práce.

Největším světovým distributorem čipů je firma Xilinx, která také poskytuje software pro syntézu, čili převod kódu v programovacím jazyce na přesný popis konfigurace konkrétního čipu. Vysyntetizovaný design, který bývá většinou uložený v paměti typu EEPROM či FLASH, je



nahráván do FPGA při každém zapnutí napájení přes sériové rozhraní. Lze tak mít připraveno více designů s různými parametry (z pohledu programovacího jazyka VHDL nazývanými generiky, čili konstantami pro každou jednotku), např. několika rozdílnými nastaveními kvality exportovaného palindromu (délka a počet chyb) nebo počtem výpočetních elementů, které mají vliv na výsledné zrychlení oproti běžnému řešení na univerzálním procesoru.

## 3 Hardwarová akcelerace

Hlavní výhodou výpočtů prováděných ve speciálním hardwaru oproti použití univerzálního procesoru je možnost paralelizace. Ta ve srovnání se sekvenčním softwarem může vykazovat značné zrychlení, které závisí na celé řadě parametrů od vlastností konkrétního použitého hardwaru až po možnosti implementovaného algoritmu. V předchozí části byly uvedeny dva typy algoritmů použitelné pro hledání přibližných palindromů, a to dynamické programování a použití datových struktur založených na sufixech řetězce. Právě prvně jmenovaná metoda je vhodná pro paralelizaci, jelikož v datové struktuře tvořené tabulkou lze vyhodnocovat více polí nezávisle na sobě. V konkrétní úloze hledání přibližných palindromů, která začíná inicializací vedlejší diagonály a sousední horní diagonály s ní rovnoběžnou na skóre nula, je teoreticky možné v jednom kroku vyplnit všechny buňky tabulky, které leží na sousední spodní rovnoběžné diagonále. To vše kvůli závislosti ve výpočetním vzorci (nutná znalost tří předcházejících hodnot).

Časová složitost algoritmu dynamického programování pro úlohu hledání přibližných palindromů při sekvenčním řešení (typicky software na jednojádrové architektuře – v jednom kroku je vypočteno ohodnocení jedné buňky) je  $O(n(n-1)/2)$ , kde  $n$  je délka vstupního řetězce. Teoretická složitost paralelního řešení schopného v každém jednom kroku vypočítat hodnoty všech buněk na vedlejší diagonále je pak  $O(n-1)$ , tedy lineární. Pokud je délka hledaných palindromů shora omezená (z důvodu již uvedené velmi malé pravděpodobnosti výskytu dlouhých opakování v biologických sekvencích), dochází k vyhodnocení jen daného počtu  $k$  diagonál a časová složitost paralelního řešení se stává konstantní  $O(k)$ , tedy indiferentní vůči délce analyzované sekvence.

### 3.1 Předávání dat, zřetězené zpracování

Výpočet ohodnocení všech buněk na vedlejší diagonále matice dynamického programování lze v jediném kroku provést za předpokladu, že pro každou z nich je k dispozici jeden výpočetní prvek, který má k dispozici všechna potřebná data (tedy dvojici znaků pro porovnání, trojici parciálních skóre ze sousedních buněk a hodnoty pokut). Tato podmínka pak platí pro každý jeden krok. Pro dosažení optimální propustnosti celého systému se jeví jako výhodné použít koncepci zřetězeného zpracování (angl. pipeline), která sice obecně zvyšuje zpoždění výstupu oproti vstupu (latenci – ta však pro úlohu hledání přibližných palindromů není kritická), na druhou stranu ale může zjednodušit problém zásobování výpočetních prvků (označovaných také jako procesní elementy – PE) daty nutnými k provedení výpočtu.

Obecný časový diagram zřetězeného zpracování je naznačen ve schématu 3.1 – každá vstupní data procházejí postupně všemi stupni linky. Lze odtud vyčíst zmíněnou zvýšenou latenci (odpovídající délce zřetězené linky) a také odvodit vylepšenou propustnost. Po naplnění linky daty

( $k$  cyklů = počet stupňů) je nový výsledek na výstupu k dispozici v každém dalším kroku výpočtu. Pro velké množství vstupních dat v poměru k délce linky je navíc možné počáteční prodlevu zanedbat a zrychlení z pohledu propustnosti systému označit za  $k$ -násobné oproti běžnému sekvenčnímu přístupu (v tomto smyslu použití jediného procesního elementu).

krok:	1	2	3	4	5	6	7	8	9	10
vstup 1	PE 1	PE 2	PE 3	PE 4						
vstup 2		PE 1	PE 2	PE 3	PE 4					
vstup 3			PE 1	PE 2	PE 3	PE 4				
vstup 4				PE 1	PE 2	PE 3	PE 4			
vstup 5					PE 1	PE 2	PE 3	PE 4		
vstup 6						PE 1	PE 2	PE 3	PE 4	
vstup 7							PE 1	PE 2	PE 3	PE 4

Schéma 3.1: Obecný časový diagram zřetěženého zpracování pro čtyři stupně linky.

Pro konkrétní případ hledání přibližných palindromů je nutné určit sémantické přiřazení stupňů zřetěžené linky jednotlivým buňkám tabulky dynamického programování. Pro vyhodnocení celé vedlejší diagonály v jednom kroku je nutné alokovat pro každý sloupec matice jeden procesní element – výpočet začne ve všech najednou a dle jejich pozice v systému končí s prodlevou jednoho kroku. Toto chování je naznačeno ve schématu 3.2(a), význam přiřazení PE i prvnímu sloupci (kde neprobíhá výpočet) spočívá ve vzájemném propojení a předávání dat mezi prvky linky.

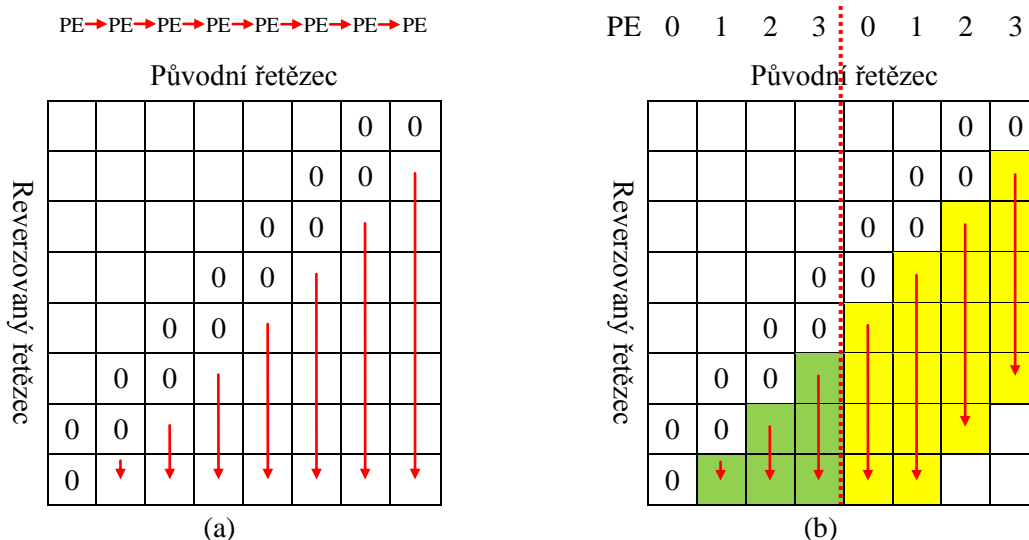


Schéma 3.2: (a) přiřazení PE z pipeline ke sloupcům tabulky, (b) výpočet  $k$  anti-diagonál po fázích.

Po inicializaci linky mají všechny procesní elementy právě ta data, která jsou nutná pro provedení prvního kroku výpočtu, tedy příslušné znaky pro sloupec a řádek tabulky a dále tři nulové

hodnoty (A, B, C) představující parciální skóre sousedních buněk. Data pro další krok je vždy možné získat následujícím způsobem: znak pro sloupec zůstává totožný po celou dobu výpočtu, znak pro aktuální řádek měl v předešlém kroku k dispozici procesní element zpracovávající sloupec tabulky nalevo. Lze jej tedy v rámci zapojení linky předat (proto je PE alokovan i pro první sloupec matice). Stejně tak lze předat i parciální skóre D vypočtené tímto prvkem – v následujícím kroku bude představovat hodnotu C v sousedním elementu. Hodnota A odpovídá hodnotě C z předchozího kroku, stačí ji proto uchovat zpožděnou o jeden cyklus. Poslední potřebnou položku – skóre B – vypočetl daný PE jako výsledek D v rámci předchozího vyhodnocení, může ji proto také uchovat a znovu použít pro určení dalšího parciálního skóre.

Z praktického hlediska je nutné uvažovat i případy, kdy není k dispozici dostatek procesních elementů (fyzické a prostorové omezení). Pro zpracování libovolně dlouhého vstupu je tabulka dynamického programování rozdělena na několik sekcí – sdružených sloupců, kdy každému je opakovaně alokovan právě jeden PE tak, jak je naznačeno ve schématu 3.2(b) pro dvě sekce a čtyři PE (je zde také ilustrováno zastavení výpočtu po vyhodnocení pěti vedlejších diagonál). Data z posledního sloupce jedné sekce jsou zpětnou vazbou předána opět na vstup další sekce. Kvůli tomuto omezení je složitost celé akcelerační architektury sublineární (přesněji  $O(kn/p)$ , kde  $k$  je maximální délka palindromu,  $n$  počet znaků vstupní sekvence a  $p$  odpovídá počtu PE – podíl  $n/p$  tedy udává, na kolik sekcí je tabulka rozdělena), stále však dochází k výraznému urychlení výpočtu.

## 3.2 Architektura hardwaru

Návrh architektury pro čip FPGA vychází z předchozí práce [9] zabývající se porovnáváním řetězců na základě podobnosti, která též při akceleraci využívá principů dynamického programování. Základní jednotkou je procesní element, který zajišťuje kromě výpočtu ohodnocení buňky také předávání dat potřebných pro řízení celého řetězce procesních elementů umístěných v systolickém poli (angl. systolic array – SA). Jelikož výpočet začíná ve všech PE zároveň (na rozdíl od porovnávání řetězců na podobnost, kde je spouštěn postupně od prvního), je nutno každému PE dodat příslušný znak pro potřeby porovnání ještě před začátkem výpočtu. To lze splnit pomocí paměti připojené paralelně ke všem PE.

### 3.2.1 Systolické pole a jeho řízení

Základem výpočtu je zřetěžená linka procesních elementů, schéma 3.3 pak ukazuje zapojení celého pole. Výstup linky je přes vyrovnávací jednotku (FIFO, k dispozici je několik typů dle použité paměti) přiveden opět na její vstup, kde je pro první krok výpočtu multiplexován s nulami v případě skóre a statusu, respektive s prvním prvkem řetězce, který je dále předáván jako vertikální znak. Právě tato zpětná vazba umožňuje architektuře zpracovat libovolně dlouhé řetězce, jelikož počet znaků není omezen množstvím procesních elementů.

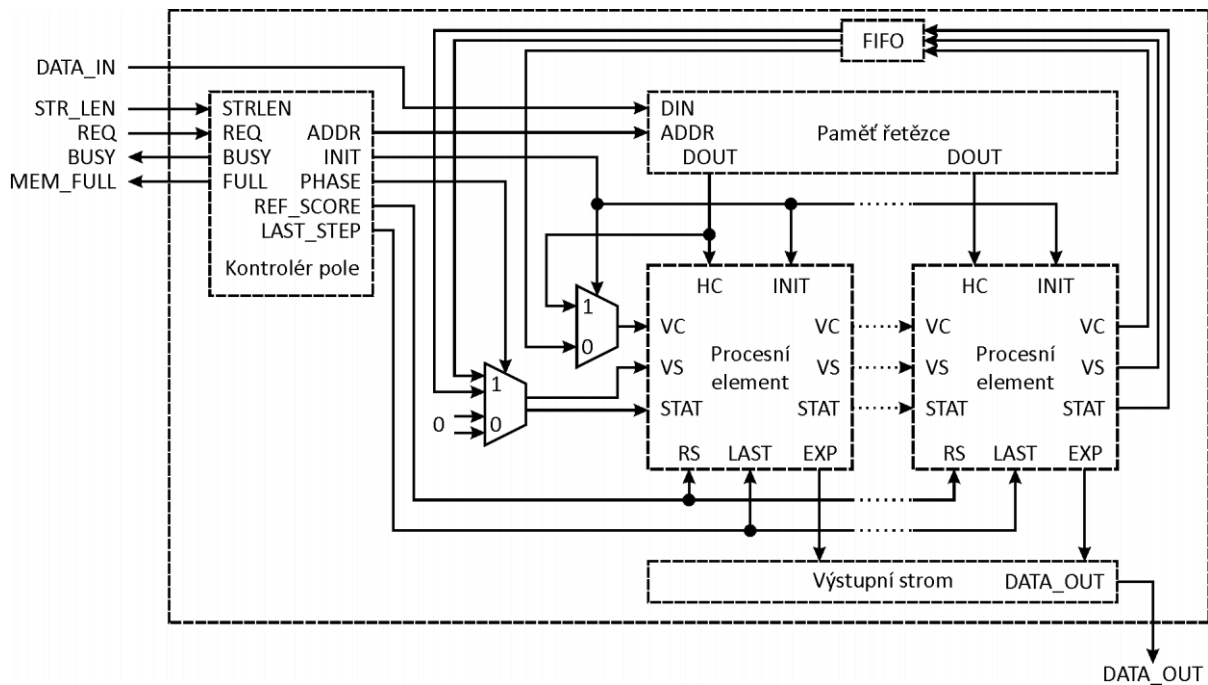


Schéma 3.3: Blokové schéma systolického pole (jednotka SA).

Zásobování pole procesních elementů daty je prováděno z paměti řetězce, která má dva porty – zápisový (který může mít obecně libovolnou datovou šířku dle potřeb konkrétního zapojení) a čtecí (ten je paralelně připojen ke všem PE, přičemž datová šířka každého výstupu je shodná s datovou šířkou jednoho znaku). Velikost paměti je vhodné nastavit tak, aby v průběhu výpočtu bylo možno načítat další data a po skončení jedné fáze (vyhodnocení sekce tabulky) bez časové prodlevy začít okamžitě navazující výpočet (pro každý PE lze tedy uložit alespoň dva znaky řetězce – s jedním se pracuje, druhý je možno načítat).

Export dat na výstup systolického pole zajišťuje stromová architektura (v jednotce output tree, viz schéma 3.4), která obsahuje vyrovnávací FIFO (z posuvných registrů – minimální nutný počet položek lze formálně odvodit [10]) pro každý výstup z PE a dále stromovou architekturu z registrů a multiplexorů, která sdružuje všechny datové kanály do jednoho výstupu. Data jsou tvořena číslem procesního elementu a informací, ve kterém kroku kolikáté fáze k exportu došlo. Z těchto údajů lze jednoduše zpětně určit pozici v tabulce dynamického programování, na které se nalézá přibližný palindrom.

Generování společných signálů pro inicializaci a řízení procesních elementů a nastavování adresy paměti zajišťuje kontrolér systolického pole. Jedná se o konečný automat (FSM), který zpracovává vstupy SA (délka řetězce a požadavek na začátek hledání palindromů) a zpětně generuje výstupy informující nadřazenou jednotku – typicky automat zajišťující převod vstupního protokolu na konkrétní signály vstupující do systolického pole – o probíhajícím výpočtu (příznak BUSY) a zaplnění paměti (příznak MEM\_FULL – slouží především k pozastavení vstupního toku dat na

sběrnici DATA\_IN). Kontrolér také poskytuje informace o aktuálním kroku výpočtu stejně jako o pořadí probíhající fáze. Ty jsou použity v exportovaných datech výstupní jednotky.

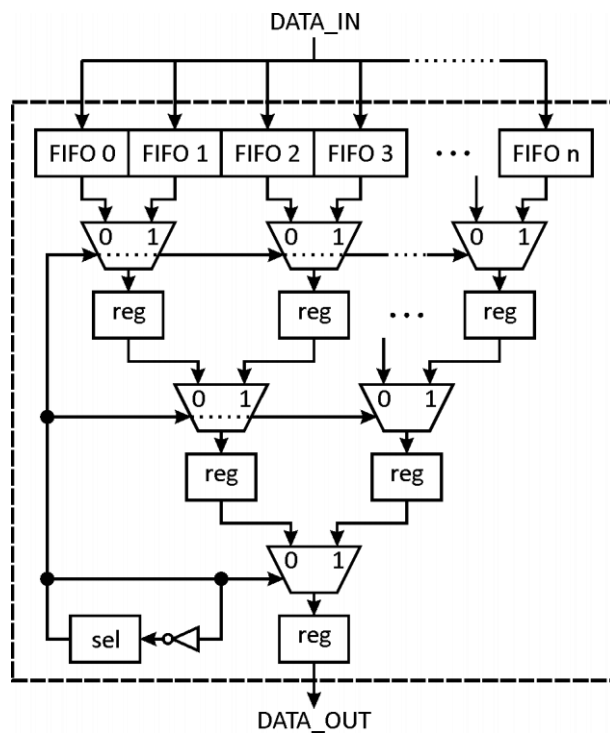


Schéma 3.4: Blokové schéma výstupního stromu (jednotka output tree).

### 3.2.2 Procesní element s porovnávací buňkou

Rozhraní procesního elementu obsahuje několik skupin signálů. Základem jsou vstupy z předchozího PE v rámci zřetězené linky (ve schématu 3.5 po levé straně jednotky) a výstupy do dalšího PE (vpravo). Přes ně jsou postupně předávány jednak odpovídající vertikální znak (z řetězce po levé straně tabulky – znak, který je v daném kroku zpracováván jako vertikální v jednom PE, bude v dalším kroku potřebný pro sousední jednotku), aktuální ohodnocení pole (skóre C pro porovnávací buňku) a jednobitový status, který udává, zda hodnota D v příslušném PE přesáhla nastavený práh. Ten je ve formě aktuálně povoleného počtu chyb (referenčního skóre závislého na tom, kolik kroků již proběhlo – tuto hodnotu generuje kontrolér systolického pole) obsažen ve skupině signálů, které jsou společné pro všechny PE.

Ve schématu 3.5 jsou signály společné všem PE zakresleny na horní a spodní straně jednotky. Kromě již zmíněného REF\_SCORE jsou jimi jednobitové příznaky prvního (inicializačního) a posledního kroku výpočtu daného PE a výstupní indikace exportu pozice palindromu. Posledním vstupem je znak z řetězce na horním okraji tabulky dynamického programování, který je sice zapojen paralelně ke všem PE, ale pro každý z nich obsahuje jinou hodnotu z paměti.

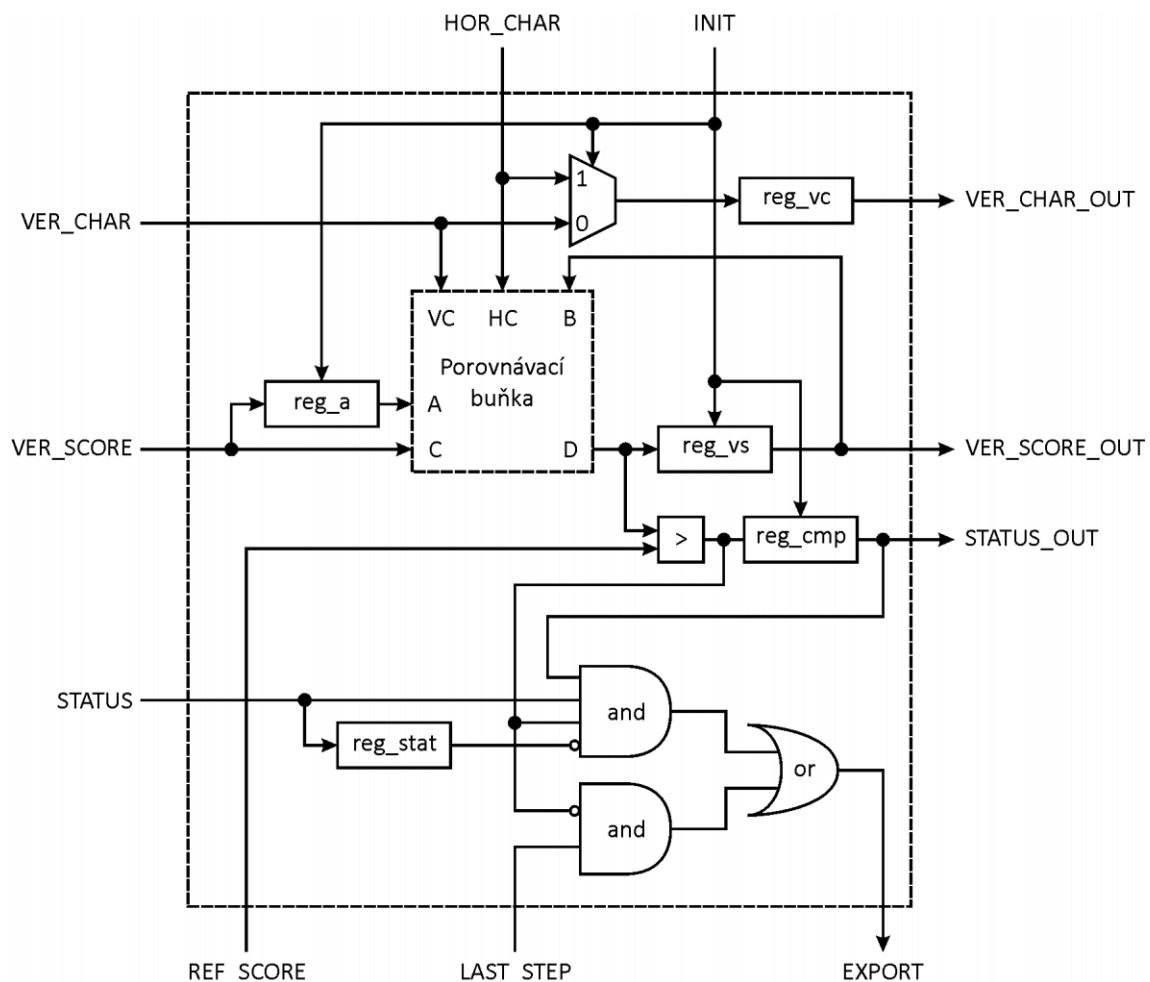


Schéma 3.5: Blokové schéma procesního elementu (jednotka PE).

Při inicializaci (aktivní vstup **INIT**) jsou vynulovány registry zřetěženého zpracování, přičemž do registru uchovávajícího aktuální vertikální znak je uložen vstup **HOR\_CHAR** (v příštím kroku bude pro sousední PE platit za vertikální znak). Během výpočtu je multiplexor na vstupu registru přepnut a probíhá klasické posouvání znaku přes zřetěženou linku. Kromě obou znaků potřebuje porovnávací buňka hodnoty vstupů **A**, **B** a **C**. Skóre **B** z levé sousední buňky je jednoduše její výstup, který zpožděný o jeden krok výpočtu udává také hodnotu **A** (v tabulce po diagonále). Podobně vstup **C** lze získat zaregistrováním výstupu **D** porovnávací buňky přímo uvnitř procesního elementu.

Logika rozhodující o exportu palindromu pracuje se všemi čtyřmi hodnotami **A**, **B**, **C**, **D** ve formě jednobitových příznaků, které jsou aktivní pro ty buňky tabulky, jejichž ohodnocení přesáhlo referenční skóre. K exportu pak dojde v případě, že všechny hodnoty kromě **A** mají vyšší hodnotu než je tento vstup (to odpovídá právě takové diagonále, kde je obsažen palindrom a došlo k jeho nežádoucímu rozšíření o nadbytečný znak – navíc tento mechanismus eliminuje exporty na sousedních diagonálách). Zároveň je nutno exportovat i ty palindromy, jejichž ohodnocení nepřesáhlo

danou mez a výpočet přitom končí. To zajišťuje signál LAST\_STEP, který je logicky vynásoben s příznakem aktuální buňky.

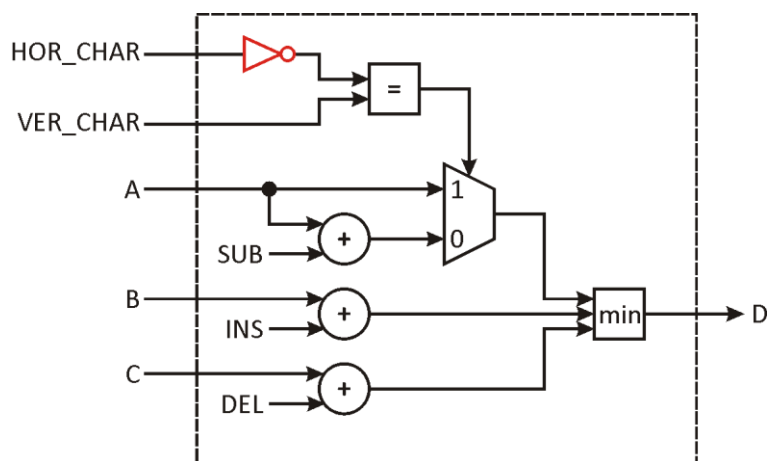


Schéma 3.6: Blokové schéma porovnávací buňky (jednotka mcell).

Funkci výběru minima ze tří možných hodnot (se započtenými pokutami za záměnu, vložení či vynechání znaku) zajišťuje kombinační obvod umístěný v jednotce nazvané mcell (z angl. matching cell – porovnávací buňka). Její zapojení ukazuje schéma 3.6 (vstupy A, B, C a výstup D odpovídají dříve uvedenému modelu výpočtu, vstupy HOR\_CHAR a VER\_CHAR pak obsahují binární reprezentaci znaků na horním a levém okraji tabulky dynamického programování). Konstanty SUB, INS a DEL představují již zmíněné pokuty.

Zavedením drobné úpravy lze jednoduše použít jednotku i pro hledání palindromů v biologických sekvencích bez nutnosti předchozího generování reverzního řetězce komplementárních nukleotidů (v tabulce dynamického programování po levé straně): Při vhodně zvoleném kódování lze komplementární znak generovat přímo v porovnávací buňce, a to použitím operace negace NOT. Pro sekvence dusíkatých bází v nukleových kyselinách je mocnost abecedy čtyři, každý znak lze tedy uchovat pomocí dvou bitů. Budou-li kódy 00, 01, 10 a 11 odpovídat znakům A, C, G a T (případně U) v uvedeném pořadí, pak operace NOT nad každým z nich vrací vždy odpovídající komplementární bázi. Je-li tedy právě jeden ze vstupních znaků negován (červený invertor ve schématu 3.6), je výsledkem porovnání příznak komplementarity, nikoli shody.

### 3.2.3 Nastavitelné parametry jednotky

Pro snadnější konfigurovatelnost hardwarové jednotky jsou všechny použité konstanty umístěny v jediném souboru, kde je implementován tzv. package (balík) nazvaný dna\_pkg. Kromě parametrů uvedených v tabulce ve schématu 3.7 je možné změnou jediné hodnoty určit typ paměti, který se použije pro realizaci jednotky FIFO zajišťující zpětnou vazbu z konce zřetězené linky na její začátek



(viz zapojení procesních elementů v systolickém poli), konkrétně lze využít distribuovanou paměť (složenou z look-up tabulek), blockRAM bloky v FPGA nebo posuvné registry (odpovídá hodnotám 0, 1 a 2 v uvedeném pořadí). Dále je zde definována např. také funkce binárního logaritmu (nezbytná pro určení počtu bitů nutných k adresování daného počtu položek) a další pomocné hodnoty využívané ve více podkomponentách systému. Hlavním smyslem tohoto přístupu je jednotná lokace všech parametrů a také možnost tento soubor dynamicky generovat při napojení na komplexnější či uživatelsky transparentnější systém (jakým může být webové rozhraní).

Mezi některými parametry existují určité závislosti. Týká se to zejména datových šířek: vstupní datová šířka musí být celým násobkem datové šířky jednoho znaku. Tím je dán počet znaků v každém vstupním bloku dat, na kterém je stejným způsobem závislý počet procesních elementů (načtení odpovídajícího počtu znaků musí být zarovnaná operace kvůli práci s paměť řetězce). Jelikož datový výstup hardwarového TX bufferu platformy NetCope (viz dále) obsahuje 64 bitů (tedy 32 nukleotidů, každý uložený na dvou bitech), znamená tato vlastnost relativně silné omezení. Lze však použít jeden z prostředků vývojové platformy, a to sadu hardwarových komponent pro práci s protokolem FrameLink. Mezi ně patří také jednotka označovaná jako transformer, která umožňuje měnit datovou šířku protokolu na jakoukoli mocninu čísla 2 v rozmezí 8 až 128 bitů a to libovolným směrem (z menší na větší i naopak). Datová šířka vstupního toku dat může být tedy snížena až na 8 bitů, při analýze sekvencí nukleotidů, kde každá vstupní data obsahují minimálně 4 znaky řetězce, musí být proto počet procesních elementů celým násobkem čtyř.

<b>parametr</b>	<b>význam</b>
<b>SUB_PENALTY</b>	hodnota pokuty za záměnu znaku
<b>INS_PENALTY</b>	hodnota pokuty za vložení znaku
<b>DEL_PENALTY</b>	hodnota pokuty za vynechání znaku
<b>COMPLEMENTARY</b>	příznak označující sekvenci nukleotidů (vkládá invertor do mcell)
<b>PE_ITEMS</b>	počet procesních elementů v systolickém poli
<b>BUFFER_SIZE</b>	velikost jednotky FIFO na vstupu output_tree (viz [8])
<b>MIN_RADIUS</b>	minimální poloměr exportovaných palindromů
<b>MAX_RADIUS</b>	maximální poloměr exportovaných palindromů
<b>MATCH_PER_ERR</b>	parametr $e$ (udává požadovanou kvalitu přibližného palindromu)
<b>MAX_PHASES</b>	maximální počet fází výpočtu
<b>DATA_WIDTH</b>	datová šířka vstupního toku dat v bitech
<b>CHAR_WIDTH</b>	datová šířka jednoho znaku v bitech
<b>OUTPUT_DATA_WIDTH</b>	datová šířka výstupního toku dat v bitech

Schéma 3.7: Nastavitelné parametry hardwarové jednotky.

Pro dosažení větší flexibility lze odděleně nastavit i výstupní datovou šířku, jelikož je z pohledu složitosti softwarové aplikace vhodné, aby každá exportovaná data (pozice přibližného palindromu v tabulce dynamického programování) byla uložena v jednom slově rámce protokolu FrameLink, a to i za cenu nevyužití většiny z dostupných bitů. Výstupní datový tok je totiž malý v porovnání se vstupem a toto plýtvání tudíž nemá negativní efekt. Jelikož vstup RX bufferu má opět šířku 64 bitů, je v případě nastavení rozdílné hodnoty znovu použita jednotka transformující protokol FrameLink na potřebný rozsah.

### 3.3 Rozhraní akcelerační jednotky

Analyzovaný řetězec (vstup, nejčastěji v textové formě) může být již v softwarové aplikaci převeden na vhodnou binární reprezentaci. V případě sekvencí nukleotidů postačují k reprezentaci jednoho znaku dva bity. Z důvodu omezení velikosti paketu (celý vstup nelze vždy poslat najednou, většinou bude rozdělen na více částí) je vhodné přidat před samotná data řídicí informaci o délce analyzované sekvence. Není tedy nutno explicitně označovat konec posloupnosti speciálním znakem (skupinou bitů), jak tomu je u některých komunikačních protokolů.

Součástí hardwarové akcelerace není zpětný průchod maticí dynamického programování hledající optimální zarovnání řetězce a jeho reverzované formy. Výstupem jsou pozice buněk tabulky, dané pořadím příslušného procesního elementu ve zřetěžené lince (spolu s číslem sekce identifikuje sloupec matice) a aktuálním krokem výpočtu (určuje vedlejší diagonálu), na kterých došlo k exportu palindromu (popsanému v rámci teoretického rozboru algoritmu).

#### 3.3.1 Platforma NetCope

Namísto výběru konkrétní cílové karty osazené čipem FPGA byla zvolena platforma NetCope, která slouží k rychlému vývoji primárně síťových aplikací (jsou na ní postaveny všechny výstupy výzkumné aktivity CESNETu – projektu Liberouter). Lze ji však použít i jako čistě akcelerační platformu, která poskytuje abstraktní vrstvu mezi operačním systémem (její součástí jsou i ovladače a další knihovny pro tvorbu softwaru) a FPGA (zejména podpora DMA přenosů přes sběrnici PCIe). Díky jednotnému rozhraní hardwarové aplikace je pak abstrakce od fyzické karty úplná. V současné době je tato platforma podporována nad celou rodinou karet Combo (včetně nejnovějších označovaných jako V2) a využít ji lze i s kartou ml555 od výrobce čipů, firmy Xilinx.

Z pohledu obecné akcelerace výpočtu nemá smysl zabývat se síťovým rozhraním platformy (nebude použito – viz dva nezapojené vstup-výstupní buffery ke konektorům desetigigabitového Ethernetu ve schématu 3.8). Naopak velmi podstatné je rozhraní směrem k systémové sběrnici PCIe a dále k softwarové vrstvě. Výstup vysílacího (TX) a vstup přijímacího (RX) bufferu v hardwaru tvoří sběrnice používající protokol FrameLink (což je zjednodušený protokol LocalLink používaný firmou Xilinx). Mezi buffery přímého přístupu do paměti a PCIe probíhá přenos dat pomocí rychlé interní

sběrnice implementované přímo v FPGA. Ta je převedena na komunikační protokol koncové jednotky systémové sběrnice, která je zde přítomna ve formě IP Core od Xilinxu. Ve schématu 3.8 je celková architektura platformy NetCope od systémové sběrnice až po síťové rozhraní s naznačeným zapojením akcelerační jednotky.

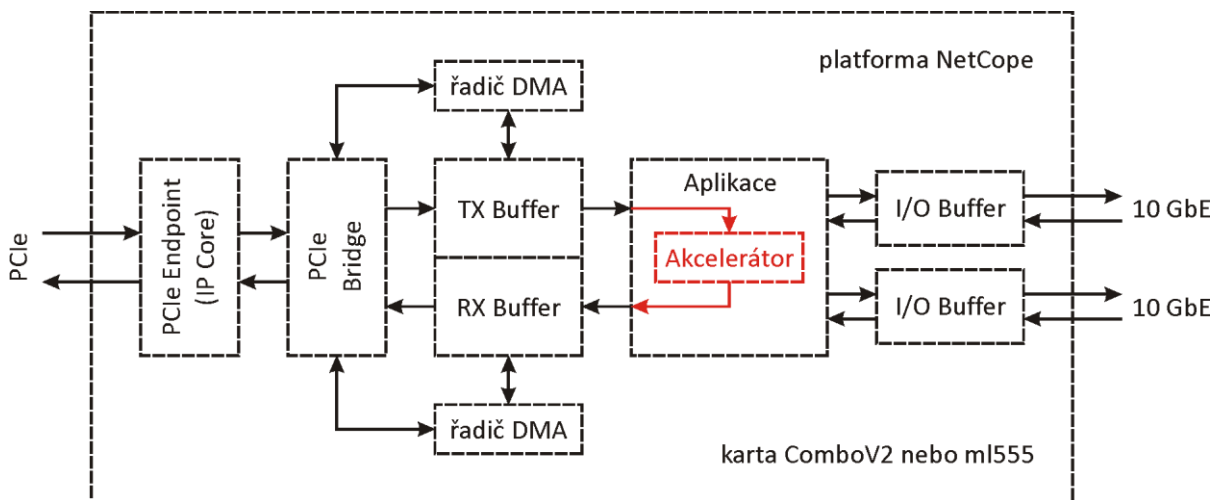


Schéma 3.8: Obecná architektura hardwarové části platformy NetCope.

Z pohledu hierarchicky nejvyšší obálky zapojené do platformy NetCope je tedy jasně dané rozhraní – vstupem i výstupem musí být protokol FrameLink obsahující analyzovanou sekvenci na vstupu a exportovaná data na výstupu. Návrh však probíhal nezávisle na rozhraní daném platformou a uvnitř obálky jsou proto dva jednoduché stavové automaty (FSM), které provádějí převod mezi FrameLinkem a obecnými signály systolického pole (popsanými dříve). Lze tedy jednoduše implementovat další sadu automatů a jednotku použít v prostředí jiného aplikačního protokolu.

Přenos dat mezi pamětí RAM a akcelerační kartou probíhá za pomoci dvojice kruhových bufferů, z nichž jeden je alokovan přímo v čipu FPGA (hardwarový – viz RX a TX buffer ve schématu 3.8) a druhý v paměťovém prostoru jádra systému (softwarový – viz odpovídající buffery ve schématu 3.9). Mezi nimi je pak prováděn přesun dat, a to s využitím čtyř ukazatelů – vždy dvojice udávající adresu začátku (StartPointer) a konce (EndPoint) souvislého datového bloku v každém z kruhových bufferů. Všechny čtyři hodnoty pak kontroluje příslušný DMA řadič umístěný na čipu a na jejich základě iniciuje DMA přenosy. V případě vysílání dat do karty je řadič informován o nových datech v paměti RAM prostřednictvím ovladače (driveru), který posune hodnotu ukazatele SwEndPoint. Pokud je v hardwarovém bufferu dostatek místa na jejich uložení (řadič tuto informaci zjistí z rozdílu příslušných ukazatelů), dojde k jejich přesunu do čipu a zároveň posunutí SwStartPointeru (uvolnění místa v RAM) a HwEndPointu, čímž je TX DMA buffer informován o velikosti zapsaných dat, která následně bez dalšího zásahu řadiče odešle do aplikace. Poté zpětně informuje řadič o uvolněném místě změnou hodnoty HwStartPointer.

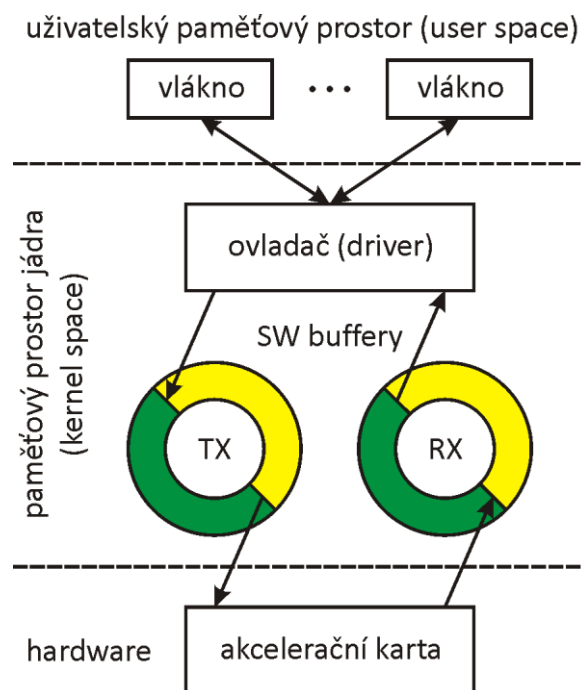


Schéma 3.9: Princip přenosu dat mezi softwarem a kartou.

Přenos dat přijatých do hardwarového bufferu probíhá na stejném principu, pouze v opačném směru. Při změně velikosti dat v RAM (uvolnění po odeslání do čipu či přijetí nových dat z karty) může být také generováno přerušení informující o dané skutečnosti. Obecně je každý směr považován za samostatný DMA kanál s vlastní dvojicí bufferů. Navíc lze těchto kanálů použít více odděleně (např. při umístění více samostatných jednotek do aplikačního jádra). Základní konfigurací pro hardwarovou akceleraci výpočtu je jeden TX a jeden RX kanál pro vyslání dat k analýze a zpětný příjem jejího výsledku. S daty může manipulovat i více než jeden proces či vlákno (program tak může používat jedno vlákno pro zápis a druhé pro čtení dat), vícenásobný přístup ke kruhovým bufferům v SW je koordinován přímo ovladačem a není jej tedy nutné řešit v rámci uživatelské aplikace.

### 3.4 Struktura softwaru

Kromě nízkoúrovňových ovladačů nabízí platforma NetCope také softwarovou knihovnu pro jazyk C, která umožňuje pohodlnou a efektivní implementaci aplikací pracujících nad akcelerační kartou. Hlavním konceptem, použitým pro dosažení optimální rychlosti přenosu dat, je zamezení nadbytečnému kopírování dat mezi pamětí a aplikací, které běžně provádí procesor (odtud její název `libsze2` – z angl. `straight zero copy data version 2 library`). Data jsou připravována přímo v paměti, odkud jsou následně v rámci DMA přenosu poslána do karty.

Před započítím komunikace s hardwarem je provedeno načtení dat (řetězce k analýze) do paměti, a to ze souboru zadaného parametrem při spouštění (jde o jediný parametr, který je

vyžadován k běhu aplikace – seznam všech parametrů je uveden ve schématu 3.10). V případě, že se jedná o sekvenci nukleotidů (zadáno také příznakem z příkazové řádky), jsou tato data navíc transformována do takové podoby, aby každý znak kódovaly pouze dva bity (dle dříve uvedeného konceptu vhodného pro zjištění komplementarity za použití funkce NOT před porovnáním). Výsledkem jsou tedy binární data v paměti (zarovnaná tak, aby počet znaků byl násobkem počtu procesních elementů v hardwaru), počet znaků a jejich celková velikost v bajtech, která později slouží při jejich odesílání.

parametr	význam
<b>-f</b> <i>název_souboru</i>	relativní cesta k souboru se vstupní sekvencí (povinný)
<b>-b</b>	příznak značící sekvenci nukleotidů (použije se dvoubitové kódování)
<b>-d</b> <i>název_zařízení</i>	cesta k zařízení (výchozí je /dev/szedataII0)
<b>-h</b>	zobrazí nápovědu a ukončí program

Schéma 3.10: Parametry příkazové řádky akceptované softwarovou aplikací.

Inicializační část programu dále zahrnuje připojení k zařízení specifikovanému parametrem funkce `szedata_open` (odpovídá argumentu `-d` příkazové řádky), která jako návratovou hodnotu poskytuje ukazatel na strukturu `szedata` popisující zařízení (handle). Ten je následně předáván všem funkcím pracujícím nad rozhraním jako první parametr. Před započítím komunikace je nutné určit, které z jednotlivých kanálů chce aplikace obsluhovat. Bitové masky pro oba směry (TX a RX) společně s údaji, po jaké době mají být buffery kontrolovány na nová data (na principu vyzývání – pollingu), jsou předány funkci `szedata_subscribe`. Po jejím úspěšném provedení lze všechna požadovaná rozhraní aktivovat voláním funkce `szedata_start` (opět s ukazatelem na strukturu `szedata` jako parametrem).

Jelikož zasilání a příjem dat lze provádět nezávisle na sobě, bylo přikročeno k použití dvou vláken (knihovna `pthread`) kdy každé obsluhuje právě jeden směr komunikace. Jsou vytvořena po předzpracování dat ze souboru a inicializaci zařízení, po jejich úspěšném skončení (průběh je popsán v následujících sekcích; zápisové končí dřív, jelikož po odeslání všech dat k analýze způsobí latence zřetěžené linky v hardwaru zpoždění výsledku) je již pouze volána funkce `szedata_close`, která ukončí práci se zařízením.

### 3.4.1 Zápis dat

Jakmile je analyzovaná sekvence připravena v paměti a hardwarové zařízení inicializováno, je možno zahájit přenos dat do akcelerační karty. Knihovna `libsze2` poskytuje pro tento účel skupinu funkcí, z nichž jako nejvhodnější z hlediska této aplikace se jeví kombinace dvou funkcí nazvaná

`szedata_prepare_and_try_write_next`, která v sobě obsahuje přípravu dat v odesílacím bufferu ovladače a následný pokus o jejich odeslání (který může selhat v případě nedostatku místa v hardwarovém TX bufferu). Jako parametry využívá kromě nezbytného ukazatele na strukturu `szedata` a čísla TX kanálu, do kterého má data vyslat, ještě dvojici ukazatelů na začátek dat a údaje o jejich velikosti v bajtech. První z bloků blok dat, označovaný jako `hw_data`, bude v protokolu FrameLink na čipu FPGA představovat hlavičku, která pro potřeby této aplikace předává akcelerační jednotce informaci o délce analyzované sekvence a je tedy přidáván pouze k prvnímu z paketů. Druhý blok (`sw_data`) pak obsahuje binární reprezentace znaků řetězce.

Vzhledem k omezené velikosti bufferu v hardwaru (platforma NetCope používá hodnotu 4 kB korespondující s velikostí stránky systémové paměti) není vždy možné odeslat údaj o počtu znaků sekvence i její obsah v jediném paketu. Hlavičku (s délkou řetězce uloženou na 32 bitech) obsahuje pouze první z odeslaných rámců protokolu FrameLink, všechny další přenášejí pouze data k analýze (funkci je jako délka bloku `hw_data` předána nula). Pro potřeby rozdělení dlouhého řetězce na více odesílaných paketů je také vhodné určit jeho maximální možnou velikost. V této práci byla zvolena hodnota 2 048 bajtů odpovídající právě polovině velikosti hardwarového bufferu. Po přenosu jednoho paketu do karty je tedy zahájeno odesílání dat do akcelerační jednotky, zatímco na pozadí je připravován a přenášen další paket. Jedná se proto o efektivnější řešení než přenosy maximálního možného objemu dat, po kterém by bylo nutné pozastavit zápis až do úplného vyprázdnění HW bufferu, čímž by vznikaly prodlevy při vysílání dat do hardwarové části aplikace.

### 3.4.2 Čtení dat a jejich interpretace

Také pro získání dat přijatých hardwarovým RX bufferem existuje v rámci knihovny `libsze2` jednoduchá funkce nazvaná `szedata_read_next`, která ze zařízení (daného opět ukazatelem na strukturu `szedata`) přeneše jeden paket a vrátí ukazatel na jeho začátek v paměti a také velikost nového bloku dat. V jejím rámci je však proveden také jednoduchý rozbor začátku paketu, kde se v prvních dvou bajtech očekává velikost celého rámce protokolu FrameLink a v následujících dvou bajtech pak velikost hlavičky (hodnota nula zde udává, že přijatá data byla bez hlavičky). Dodržet uvedený formát by však z pohledu této práce představovalo zbytečnou komplikaci (byla by nutná implementace hardwarové jednotky umístěné na vstupu RX bufferu vkládající potřebné údaje – které navíc mohou být proměnné – před každý rámeček dat), a proto bylo za použití prostředků knihovny `libsze2` použito alternativní řešení.

To spočívá ve využití dvojice funkcí `szedata_rx_lock_data` pro získání bloku dat a `szedata_rx_unlock_data` pro jeho uvolnění. Obě pracují se strukturou `szedata_lock` – první poskytuje ukazatel na ni jako návratovou hodnotu (pokud jsou nějaká data k dispozici; parametrem je kromě identifikace zařízení také číslo kanálu, odkud se data čtou), druhá použije stejný ukazatel k uvolnění paměti a přístupu k zařízení. Prvky struktury jsou jednak ukazatel na přijatá data

v paměti a jejich velikost v bajtech, dále pak ukazatel na další strukturu stejného typu. Ten je využit v případě, že došlo k přechodu přes hranici kruhového bufferu v paměti – aplikace proto musí tento ukazatel zkontrolovat a případně zpracovat i další odkazovanou strukturu v pořadí (zmiňovaná funkce `szedata_read_next` toto provádí interně).

Interpretace dat je specifická pro každou implementaci programu. Z hlediska této práce obsahuje každá datová jednotka (její velikost se liší v závislosti na více faktorech, v základní konfiguraci jde o osm bajtů) číslo výpočetního kroku, identifikaci procesního elementu a číslo fáze, kdy došlo k exportu přibližného palindromu. Při znalosti pozic těchto údajů (závisí na implementaci v hardwarové jednotce) lze operacemi bitových posunů jednoduše získat všechny uvedené hodnoty. S nimi lze pak v rámci softwarové aplikace dále pracovat, především z nich určit přesnou pozici palindromu v původní sekvenci.

## 3.5 Parametry akcelerátoru

Pro ohodnocování výkonnostních parametrů vytvořeného prototypu akcelerační jednotky je nutné přesně stanovit, jaká architektura je použita a dále ke kterému referenčnímu řešení se výsledky vztahují. V následujícím textu tedy bude uvažována implementace za pomoci platformy NetCope nad kartou ml555 (osazené FPGA čipem označeným jako xc5vlx50t-1 ff1136 umístěné v konektoru systémové sběrnice PCIe x8 počítače se čtyřjádrovým procesorem Xeon pracujícím na frekvenci 1,6 GHz a 2 GB systémové RAM paměti), pro srovnání s neakcelerovaným řešením byly pak převzaty parametry softwaru [11] uvedené v článku [3] – tedy implementace algoritmu hledání přibližných palindromů s využitím sufixových polí (operace LCP) v jazyce C testované na stroji s procesorem Xeon na frekvenci 3 GHz a 4 GB paměti. V této souvislosti je také uveden parametr BSps použitý pro srovnání výkonnosti, který udává počet kroků v miliardách, jež jsou provedeny během jedné sekundy (angl. billion steps per second). Referenční softwarové řešení dosahuje hodnoty  $p_{SW} = 0,03$  BSps na vzorku dat obsahujícím deset tisíc znaků.

### 3.5.1 Propustnost platformy bez akcelerátoru

Před porovnáním zrychlení výpočtu s referenčním sekvenčním řešením byl proveden test propustnosti platformy NetCope bez akcelerační jednotky, pouze se zpětnou vazbou (loopback – výstup TX bufferu vede přímo na vstup RX bufferu). Jeho cílem bylo určit maximální možnou rychlost komunikace mezi aplikací nad knihovnou libsz2 a kartou ml555 přes systémovou sběrnici. Software odesílal opakovaně malý vzorek dat (konkrétně 2 kB, tedy jeden paket zabírající polovinu HW bufferu) a to do celkové velikosti 128 MB a 1 GB (aby byla vyloučena závislost na objemu dat). Čtení pak probíhalo pouze po blocích s čítačem přijatých bajtů (bez zpracování dat) pro rozpoznání konce testu. Čas byl získáván ze systému voláním knihovny funkce `gettimeofday`, přesnost

rozdílu mezi časem těsně před začátkem odesílání a bezprostředně po přijetí posledního bloku dat by se měla dle specifikace pohybovat v řádu mikrosekund, což je postačující rozlišení.

Na základě deseti naměřených hodnot (pět pro každý z použitých objemů dat) byla určena propustnost přibližně 5,88 Gb dat za sekundu. Teoretickým maximem z pohledu karty ml555 je při výchozí frekvenci čipu 125 MHz a šířce interní sběrnice i datové cesty mezi oběma HW buffery 64 bitů hodnota 7,45 Gb za sekundu. Dosažený výsledek je ovlivněn režii systémové sběrnice, pro potřeby akcelerace výpočtu je postačující.

### 3.5.2 Zrychlení oproti sekvenčnímu řešení

Přímé srovnání hodnot BSps není možné. Hardwarová jednotka musí pro nalezení všech palindromů maximální délky  $l$  provést  $2l(n/n_{PE})$  kroků, kde  $n$  je délka vstupu a  $n_{PE}$  počet procesních elementů (podíl  $n/n_{PE}$  tedy udává počet sekcí tabulky). Počet kroků, které k získání výsledku potřebuje softwarová aplikace, je závislý na vstupní sekvenci – obecně je časová složitost  $O(kn)$  pro  $k$  povolených chyb. V nejlepším případě (palindrom na každé pozici – např. všechny znaky stejné) stačí  $2n$  kroků ( $k = 1$ ). Pokud však řetězec neobsahuje ani jeden palindrom, musí být provedeno všech  $2kn$  kroků, kde  $k = l$  (což odpovídá hardwarové implementaci). V závislosti na vlastnostech analyzované sekvence se tedy  $k$  pohybuje v rozmezí 1 až  $l$ .

$$\begin{aligned}
 T_{SW} &= \frac{2kn}{P_{SW}} & T_{HW} &= \frac{2k \cdot \frac{n}{n_{PE}}}{P_{HW}} & \alpha &= \frac{T_{SW}}{T_{HW}} = \frac{k}{l} \cdot \frac{P_{HW} \cdot n_{PE}}{P_{SW}} \\
 \text{(a)} & & \text{(b)} & & \text{(c)} &
 \end{aligned}$$

Schéma 3.11: Čas potřebný pro výpočet (a) v softwaru, (b) v hardwaru; (c) celkové zrychlení.

Vztahy pro určení času výpočtu softwarové i hardwarové implementace jsou uvedeny ve schématu 3.11(a) a 3.11(b). Zrychlení je pak rovno jejich podílu – po úpravě je možné rozdělit vztah na dva zlomky, a to poměr počtu povolených chyb k maximální délce hledaných palindromů a konstantu závislou na parametrech obou řešení [3]. Na základě této rovnice již lze provést srovnání obou přístupů.

V závislosti na počtu procesních elementů v systolickém poli, který je omezen fyzickými parametry čipu FPGA, a známé pracovní frekvenci jednotky (pro kartu ml555 vždy 125 MHz, jeden hodinový cyklus trvá 8 ns) lze formálně výpočtem odvodit hodnotu BSps pro každou konkrétní konfiguraci hardwarové architektury. Jedná se o počet PE (odpovídá počtu buněk tabulky dynamického programování, jejichž ohodnocení lze určit v jednom kroku) v miliardách násobený frekvencí (či dělený periodou),  $P_{HW} = n_{PE}f = n_{PE}/T$ . Již s jedním procesním elementem lze tedy



teoreticky dosáhnout výkonnosti 0,125 BSps. Reálně se podařilo vytvořit a na kartě ml555 otestovat design, který obsahoval  $n_{PE} = 40$  procesních elementů, což vede na hodnotu  $p_{HW} = 5$  BSps. Kvalita palindromů a další parametry (maximální počet sekcí tabulky) byly nastaveny tak, aby splňovaly požadavky (délka zkoumané sekvence i nalezených palindromů) kladené na referenční softwarovou implementaci.

Hodnota konstanty ze vztahu 3.11(c) je tedy 6 666,67 pro uvedený počet procesních elementů. Výsledné zrychlení pak závisí už pouze na poměru  $k/l$  a tedy vlastnostech vstupní sekvence. Schéma 3.12 obsahuje zrychlení dosažitelné pro tři různé typy vstupů – řetězec bez palindromů, úsek DNA a posloupnost náhodných znaků. Poměr počtu chyb k délce palindromů byl převzat z článku [3].

typ sekvence	poměr $k/l$	zrychlení
bez palindromů	1	6 667
DNA	0,535	3 567
náhodná	0,536	3 573

Schéma 3.12: Zrychlení akcelerovaného výpočtu pro různé typy vstupních sekvencí.

Největšího zrychlení je tedy dosaženo při analýze sekvencí, které neobsahují žádné palindromy. S akcelerovaným řešením je doba trvání výpočtu 6 667krát menší než při použití softwarové implementace. Poměr počtu chyb k maximální délce palindromů je přibližně stejný v řetězcích DNA a náhodných posloupnostech (i když DNA obsahuje více palindromů [3]), výpočet v hardwaru pro tyto sekvence je více než 3 500krát rychlejší než v softwaru.

Pro vstupy obsahující samé palindromy ( $k = 1$ ) může nastat i situace, kdy je softwarová aplikace rychlejší, než akcelerovaná jednotka. Ta totiž musí provést  $l$  kroků, zatímco při použití sufixového pole a operace LCP postačuje jen jediný cyklus výpočtu. Při dosazení hodnoty 1 za počet chyb i zrychlení (tedy stejná výkonnost obou přístupů) do vztahu 3.11(c) lze určit práh  $l = 6 667$ , tedy maximální délku hledaných palindromů, při které ještě má smysl použít akcelerační jednotku (dojde ke zrychlení výpočtu). Po překročení této hodnoty již pro uvedený typ sekvencí dosahuje lepších výsledků softwarová implementace.

# Závěr

Analýza biologických sekvencí hraje důležitou roli ve zkoumání živých organismů. Časová složitost některých metod včetně hledání přibližných palindromů je kromě velkého objemu dat zvyšována také výskytem mutací v posloupnostech nukleotidů tvořících DNA. Průzkumem těchto algoritmů se zaměřením na možnost jejich paralelizace a akcelerace se zabývá první část práce. V rámci ní jsou popsány postupy pro nalezení všech přibližných palindromů v řetězci, a to jednak algoritmy využívající sufixové stromy a sufixová pole a následně metoda dynamického programování.

Na základě studia jednotlivých přístupů byla navržena hardwarová architektura (popsaná v druhé části práce) akcelerující výpočet založený na principech dynamického programování. Ta byla následně implementována v jazyce VHDL za využití prostředků vývojové platformy NetCope a vytvořený design byl otestován v čipu FPGA xc5v1x50t, kterým je osazena karta ml555 firmy Xilinx připojená na sběrnici PCIe (varianta x8) počítače s procesorem Xeon (4 jádra na frekvenci 1,6 GHz) s 2 GB paměti RAM.

Výkonnost systému byla srovnána s nejlepším známým softwarovým řešením [11], konkrétně s implementací algoritmu pro hledání přibližných palindromů za použití sufixových polí v jazyce C testovaného na počítači s procesorem Xeon (na frekvenci 3 GHz) s 4 GB paměti RAM. Akcelerovaná jednotka dosahuje zrychlení až 6 667 oproti softwarové implementaci. Vzhledem k odlišnému přístupu obou porovnávaných řešení je nutné přihlédnout také k vlastnostem zkoumaných sekvencí, od kterých se odvíjí míra využití výhod sufixových polí. Pro řetězce DNA přesahuje zrychlení systému hodnotu 3 500, což je vzhledem k použitému čipu velmi slušný výsledek.

Jelikož je jednotka koncipována jako aplikace pro platformu NetCope, je nezávislá na konkrétní kartě a čipu. Při použití karet z rodiny ComboV2, nad kterými je uvedená platforma primárně vyvíjena, lze očekávat zlepšení dosažených výsledků vzhledem k vyšší frekvenci čipu FPGA a především dostupnosti více než trojnásobku zdrojů pro fyzickou realizaci akcelerační jednotky oproti kartě ml555.

Vytvořenou architekturu lze dále rozšiřovat, a to např. přidáním dalších filtračních mechanismů pro export přibližných palindromů či implementací zpětného průchodu maticí dynamického programování za účelem zjištění optimálního zarovnání. V úvahu připadá také použití sufixových polí přímo v hardwarové jednotce. Díky širokým možnostem pro definování kvality hledaných palindromů je možné i vytvoření nadřazeného systému, který by uživateli umožňoval obecné nastavení parametrů na základě požadavků pro konkrétní úlohu, a to naprosto transparentně k použitému akceleračnímu zařízení.

# Literatura

- [1] **Bruce, Alberts a Martin, Raff.** *Essential Cell Biology*. New York: Garland Science, 2003. 978-0815334804.
- [2] **Krane, Dane E. a Raymer, Micheal L.** *Fundamental Concepts of Bioinformatics*. San Francisco: Benjamin Cummings, 2003. 0-8053-4633-3.
- [3] *Hardware Acceleration of Approximate Palindromes Searching.* **Martínek, Tomáš a Lexa, Matej.** Taipei: IEEE Computer Society, 2008. The International Conference on Field-Programmable Technology. stránky 65-72. 978-1-4244-2796-3.
- [4] **Porto, Alexandre H. L. a Barbosa, Valmir C.** Finding Approximate Palindromes in Strings. *Pattern Recognition*. Listopad 2002, Sv. 35, 11, stránky 2581-2591.
- [5] **Böckenhauer, Hans-Joachim a Bongartz, Dirk.** *Algorithmic Aspects of Bioinformatics*. Berlin: Springer, 2007. 978-3-540-71912-0.
- [6] **Ukkonen, Esko.** On-line Construction of Suffix Trees. *Algorithmica*. Zář 1995, Sv. 14, 3, stránky 249-260.
- [7] *Simple Linear Work Suffix Array Construction.* **Kärkkäinen, Juha a Sanders, Peter.** Eindhoven: Lecture Notes In Computer Science, 2003. stránky 943-955. 0302-9743.
- [8] **Gusfield, Dan.** *Algorithms on Strings, Trees And Sequences: Computer Science And Computational Biology*. New York: Cambridge University Press, 1997. 0-521-58519-8.
- [9] *Automatic Generation of Circuits for Approximate String Matching.* **Martínek, Tomáš, a další.** Krakow: IEEE Computer Society, 2007. IEEE Design and Diagnostics of Electronic Circuits and Systems. stránky 203-208. 1-4244-1161-0.
- [10] *Architecture Model for Approximate Palindrome Detection.* **Martínek, Tomáš, Lexa, Matej a Voženilek, Jan.** Liberec: IEEE Computer Society, 2009. IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems. stránky 90-95. 978-1-4244-3339-1.
- [11] **Miranda, R. de Castro a Ayala-Rincón, Mauricio.** A Modification of the Landau-Vishkin Algorithm Computing Longest Common Extensions via Suffix Arrays. *Lecture Notes in Computer Science*. 2005, 3594, stránky 210-213.

# Seznam příloh

Příloha 1: CD se zdrojovými texty a touto zprávou v elektronické podobě (formát .doc a .pdf).