VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERACE ALGORITMU PRO HLEDÁNÍ PODOBNOSTI DVOU DNA ŘETĚZCŮ

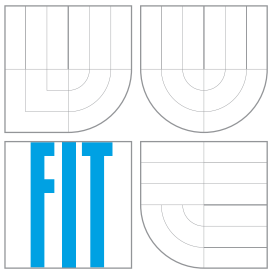
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ONDŘEJ NOSEK

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERACE ALGORITMU PRO HLEDÁNÍ PODOBNOSTI DVOU DNA ŘETĚZCŮ

HARDWARE ACCELERATION OF ALGORITHMS FOR APPROXIMATE STRING MATCHING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. ONDŘEJ NOSEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MARTÍNEK
SUPERVISOR

BRNO 2007

Zadání diplomové práce

Řešitel	**Nosek Ondřej, Bc.**
Obor	Počítačové systémy a sítě
Téma	**Hardwarová akcelerace algoritmu pro hledání podobnosti dvou DNA řetězců**
Kategorie	Počítačová architektura

Pokyny:

1. Seznamte se s technologií programovatelných hradlových polí FPGA a dostupnými nástroji pro syntézu a implementaci obvodů do FPGA hradlových polí
2. Seznamte se s algoritmy pro hledání podobnosti dvou řetězců. Mezi tyto algoritmy patří například algoritmus Smith-Waterman nebo Needleman-Wunsch.
3. Na základě předchozích dvou bodů navrhněte vhodnou hardwarovou architekturu pro urychlení algoritmu hledání podobnosti dvou DNA řetězců
4. Proveďte implementaci navrženého řešení v jazyce VHDL nebo HandleC a jeho funkci ověřte simulací.
5. Realizujte funkční prototyp navržené architektury a ověřte její správnost na kartě COMBO6.
6. V závěru diskutujte vlastnosti vytvořené implementace a možnosti dalšího pokračování projektu.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- První tři body zadání.

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 paměťovém médiu (disketa, CD-ROM), 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	28. února 2007
Datum odevzdání	22. května 2007

Licenční smlouva

Licenční smlouva v kompletním znění je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Výňatek z licenční smlouvy:

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zvěřejněním díla v databázi přístupné v mezinárodní síti:

   ☐ ihned po uzavření této smlouvy

   ☒ 1 rok po uzavření této smlouvy

   ☐ 3 roky po uzavření této smlouvy

   ☐ 5 let po uzavření této smlouvy

   ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací).

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Abstrakt

Metody pro zarovnání různých typů bioinformatických sekvencí jsou klíčovou součástí výzkumu v této oblasti. Úlohy jsou časově velmi náročné, a proto má smysl vytvořit hardwarovou platformu pro urychlení těchto výpočtů. Cílem této práce je navržení obecné architektury založené na FPGA technologii, která dokáže pracovat s několika různými druhy sekvencí. Metody, které bude navržená akcelerační karta používat budou především dynamické algoritmy Needleman-Wunsch a Smith-Waterman.

Klíčová slova

DNA řetězec, RNA, proteinový řetězec, nukleotid, aminokyselina, kodón, triplet, evoluce, geny, porovnávání řetězců, zarovnání, akcelerace algoritmu, hardware, architektura, FPGA, globální metoda, lokální metoda, algoritmus, Levenshteinova vzdálenost, Smith-Waterman, Needleman-Wunsch, pokuta, mezery, zpětný průchod, substituční matice, skóre podobnosti, výpočet, porovnávací buňka, porovnávací pravidla, řídící obvod.

Abstract

Methods for aproximate string matching of various sequences used in bioinformatics are crucial part of development in this branch. Tasks are of very large time complexity and therefore we want create a hardware platform for acceleration of these computations. Goal of this work is to design a generalized architecture based on FPGA technology, which can work with various types of sequences. Designed acceleration card will use especially dynamic algorithms like Needleman-Wunsch and Smith-Waterman.

Keywords

DNA sequence, RNA, protein sequence, nucleotide amino-acid, codon, triplet, evolution, genes, pairwise alignment, string matching, global alignment, algorithm acceleration, hardware, architecture, FPGA technology, local alignment, Levenshtein distance, Smith-Waterman algorithm, Needleman-Wunsch algorithm, penalty, gaps backtracking, substitution matrix, match score, computing, matching cell, character rule, sequencer.

Citace

Ondřej Nosek: Hardwarová akcelerace algoritmu pro
hledání podobnosti dvou DNA řetězců, diplomová práce, Brno, FIT VUT v Brně, 2007

Hardwarová akcelerace algoritmu pro hledání podobnosti dvou DNA řetězců

Prohlášení

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

.....
Ondřej Nosek
22. května 2007

Obsah

Kapitola 1

Úvod

Porovnávání řetězců je důležité v mnoha oborech, zejména v bioinformatice jsou aplikace založené na porovnání jedním z pilířů výzkumu. Je proto zásadní mít k dispozici co nejrychlejší a co možná nejpřesnější aplikaci pro tyto úkoly. Také kvůli značné rozsáhlosti zkoumaných dat.

Termínem „porovnávání“ v této práci není myšleno klasické pojetí tohoto termínu jako zjištění shodnosti (délky či obsahu) mezi řetězci, ale je tím myšlena **míra jejich podobnosti**, případně nás zajímá vymezení oblasti, ve které jsou si řetězce nejvíce podobné.

Porovnávat lze mezi sebou obecné sekvence, zaměřím se však hlavně na sekvence vyskytující se v bioinformatice, které ač mohou být odlišných druhů, metody jejich porovnávání se v základu příliš neliší. Výsledků porovnávání lze využít při zjišťování struktury a významu proteinů (bílkovin), hledání genů s určitou funkcí či sledování evolučního vývoje.

Pro obecné sekvence mají metody porovnávání využití např. v textových databázích, kde může být vznesen požadavek na vyhledání podle vzhledem podobných či zkomolených klíčů. Další uplatnění mohou najít při automatické korekci textů.

Negativní vlastností používaných postupů pro porovnávání je jejich vysoká složitost (zejména časová), proto existují jak postupy, které zpracovávají problém „hrubou silou“, tak různé heuristické metody pro urychlení výpočtu. V práci budou zpracovány ty první zmíněné, avšak obohacené o dynamický a paralelní přístup.

Kapitola 2

Teoretická část

2.1 Typy sekvencí

V této kapitole budou představeny různé typy biologických sekvencí, pro které se provádí porovnávání.

2.1.1 DNA/RNA

DNA je velmi složitá molekula, která nese genetickou informaci u živých organismů. Strukturálně má tvar pravotočivé dvoušroubovice. Skládá se ze dvou sekvencí nukleotidů, které jsou spojeny vodíkovými můstky. Sekvence jsou vůči sobě anti-paralelně svázány a obsahují navzájem komplementární informace (aby mohla být posloupnost nukleotidů nazvána „sekvencí“ musí mít délku alespoň 4).

Jednotlivé nukleotidy se skládají ze tří složek: fosfátu, deoxyribózy a nukleové báze. Právě nukleové báze jsou nositeli informace, a proto zajímavé pro aplikaci porovnávání. V DNA se v různých kombinacích vyskytují dvě skupiny bází: purinové (*adenin* – A, *guanin* – G) a pyrimidinové (*thymin* – T, *cytosin* – C). Mezi dvěma vlákny jsou vždy spojeny komplementární dvojice bází: A–T, T–A či C–G, G–C (dvěma nebo třemi vodíkovými vazbami).

Genetická informace obsažená v sekvenci je zapsaná např. `...AATGCATGC ...`

RNA je jako DNA nukleovou kyselinou, avšak s několika rozdíly. Prvním je přítomnost hydroxylových skupin, které činí molekulu méně stabilní. Nukleová báze *thymin* je zde nahrazena *uracilem* – U. Ohledně struktury, RNA netvoří jako DNA velmi dlouhou dvoušroubovici, ale spíše kratší jednoduché vlákna. Její hlavní funkcí je mimo jiné okopírování genetické informace (genu) z DNA a fyzicky ji přenést do místa, kde po té dojde k jejímu přeložení na výsledný protein. Třída RNA, pomocí které se realizuje tento přenos se označuje *mRNA* (Messenger RNA).

DNA řetězec (z příkladu výše) by se v *mRNA* přenesl jako `UUACGUACG` (díky komplementárnosti dvojic bází a záměně *thyminu* na *uracil*).

2.1.2 Proteiny

(=bílkoviny) jsou vysoce složité látky sestávající se z posloupnosti aminokyselin. Jsou podstatou (základní stavební látkou) všech živých organismů. I jednoduchý protein obsahuje

kolem 100 aminokyselin. Primární struktura proteinu je dána pořadím aminokyselin v proteinovém řetězci.

Mezi známé proteiny zařazuji např. *kolagen*, *kasein*, *hemoglobin*, *albumin*, *lepek* ... Mezi třídu proteinů patří i *enzymy*, *hormony* a *imunoglobuliny*. U několika význačných z nich bylo už zjištěno dokonce přesné složení.

Aminokyseliny. Celkem existuje 20 různých aminokyselin (těchto 20 je jen speciální podtřídou všech možných aminokyselin) a všechny proteiny jsou jejich kombinací. Molekuly aminokyselin jsou oproti DNA velmi velmi jednoduché stavební látky.

označení	zkratka	název
A	Ala	Alanin
C	Cys	Cystein
D	Asp	Kyselina asparagová
E	Glu	Kyselina glutamová
F	Phe	Fenylalanin
G	Gly	Glycin
H	His	Histidin
I	Ile	Isoleucin
K	Lys	Lysin
L	Leu	Leucin
M	Met	Methionin
N	Asn	Asparagin
P	Pro	Prolin
Q	Gln	Glutamin
R	Arg	Arginin
S	Ser	Serin
T	Thr	Threonin
V	Val	Valin
W	Trp	Tryptofan
Y	Tyr	Tyrozin

Tabulka 2.1: Přehled aminokyselin

2.1.3 Triplety

Genetický kód představuje soubor pravidel, podle kterých se genetická informace uložená v DNA (respektive RNA) převádí na sekvenci aminokyselin (proteinovou strukturu). Každá aminokyselina je pomocí tohoto kódu vyjádřena jako trojice (triplet) nukleových bází DNA. Triplety jsou též nazývány *RNA kodóny* (angl. codons).

Existují 4 různé nukleové báze, je celkem $4^3 = 64$ kodónů. Jelikož však uvažujeme pouze 20 různých aminokyselin, říkáme že genetický kód je „degenerovaný“ $\Rightarrow$ z vytvořeného proteinu nelze zpětně (jednoznačně) získat původní podobu genu.

Pomocí aminokyselin můžeme vyjádřit sekvenci `UUACGUACG` jako: `Leu/L Arg/R Thr/T`

V tabulce je kromě 20-ti standardních aminokyselin ještě vidět speciální sekvence, tzv. „stopkodóny“ (`UAG`, `UGA` a `UAA`). Slouží jako terminální symboly sekvence.

		2.pozice			
		U	C	A	G
1. pozice	U	UUU Phe/F UUC Phe/F UUA Leu/L UUG Leu/L Start	UCU Ser/S UCC Ser/S UCA Ser/S UCG Ser/S	UAU Tyr/Y UAC Tyr/Y UAA Ochre Stop UAG Amber Stop	UGU Cys/C UGC Cys/C UGA Opal Stop UGG Trp/W
	C	CUU Leu/L CUC Leu/L CUA Leu/L CUG Leu/L Start	CCU Pro/P CCC Pro/P CCA Pro/P CCG Pro/P	CAU His/H CAC His/H CAA Gln/Q CAG Gln/Q	CGU Arg/R CGC Arg/R CGA Arg/R CGG Arg/R
	A	AUU Ile/I Start AUC Ile/I AUA Ile/I AUG Met/M Start	ACU Thr/T ACC Thr/T ACA Thr/T ACG Thr/T	AAU Asn/N AAC Asn/N AAA Lys/K AAG Lys/K	AGU Ser/S AGC Ser/S AGA Arg/R AGG Arg/R
	G	GUU Val/V GUC Val/V GUA Val/V GUG Val/V Start	GCU Ala/A GCC Ala/A GCA Ala/A GCG Ala/A	GAU Asp/D GAC Asp/D GAA Glu/E GAG Glu/E	GGU Gly/G GGC Gly/G GGA Gly/G GGG Gly/G

Tabulka 2.2: Přehled kodónů a jejich mapování na aminokyseliny

Některé aminokyseliny mají ještě další funkci jako „startkodóny“, ty naopak sekvenci zahajují.

2.2 Úvod k výpočetním metodám

V této kapitole budou rozvedeny některé obecné termíny, které jsou v další práci zmíněny.

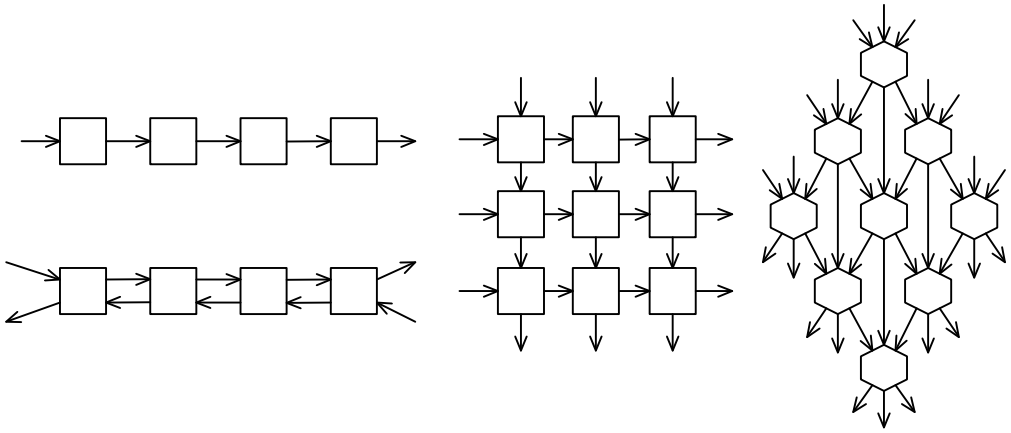
2.2.1 Dynamické programování

většina představených algoritmů využívá metody „dynamického programování“. Je to programovací technika, která řeší složitý problém rekurzivním rozkládáním na podproblémy až dokud je nelze vyřešit optimálně. Z těchto dílčích řešení se poté složí celkový výsledek. Typické problémy, řešené touto technikou, jsou mimo jiné „problém batohu“ či některé grafové a optimalizační úlohy.

2.2.2 Systolické pole

je to speciální architektura, která se sestává z velkého množství výpočetních jednotek spojených do jedné velké struktury. Tato může mít např. tvar lineárního nebo dvourozměrného pole. Existují však i sestavení do hexagonální struktury. Tato je zajímavá tím, že je možné ji mapovat jako lineární strukturu, do které data vstupují z protilehlých částí (bidirectional). Schematické ilustrace jsou vidět na obrázku 2.1.

Systolické algoritmy, které na těchto architekturách pracují jsou velice efektivní díky masivnímu paralelismu a taktéž díky tomu, že mezi bloky není potřeba vzájemná komunikace. Výpočetní jednotka pouze zpracuje předaná data a pošle je svým sousedům. Tyto algoritmy mají dvě hlavní části: definice výpočtu (zahrnující lokální operace nad předanými



Obrázek 2.1: Různé typy systolických struktur

	A	**T**	**C**	**G**
A	1	0	0	0
T	0	1	0	0
C	0	0	1	0
G	0	0	0	1

Tabulka 2.3: Matice identity

daty) a definice datového toku mezi jednotkami. Typické úlohy, na kde se tohoto schématu využívá jsou např. maticové operace (násobení matic), řetězcové operace a třeba LZ komprese dat. Jsou publikovány práce i na konstrukci obecných násobiček. Jak bude ukázáno později systolická architektura je vhodná pro úlohy porovnávání řetězců.

2.2.3 Porovnávací matice

Porovnávací matice, nebo též „substituční“ či „skórovací“, jsou častou implementací porovnávacích funkcí pro jednotlivé znaky. Porovnávací funkce slouží k ohodnocení shodnosti nebo záměny (tzv. mutace) dvou vstupních znaků. Používají se, když už se algoritmus porovnávání řetězců nespokojí s obyčejným testem shodnosti vstupních znaků. V bioinformatických řetězcích je obvyklé, že jednotlivé mutace se vyskytují s různými pravděpodobnostmi a toto bývá ve složitějších maticích promítnuto. Dalším hlediskem, které při konstrukci matic hrají roli je rozsah vstupní abecedy. Dvě hlavní používané abecedy DNA a proteiny mají 4, respektive 20 znaků. Pokud se sestaví kompletní tabulka pro všechny kombinace znaků, budou mít matice 16 a 400 polí, což je velký rozdíl v rozsahu.

Jejich použití je znevýhodněno vyšší složitostí a spotřebou systémových prostředků (hradla v hardwaru, paměť v softwaru). Matice mohou být implementovány buďto jako „pevné“ (tzn. hodnoty v matici jsou trvale nastaveny), nebo uživatelsky nastavitelné (to přináší další nároky na design systému).

Nejjednodušší je „matice identity“ 2.3, která shodu znaků oceňuje jedním bodem, a neshodnost nulou. Matici v této podobě nemá význam implementovat - realizuje se jednoduše „testem na shodnost“ s fixním ohodnocením.

	A	C	G	T
A	2	0	-2	-3
C	0	1	-1	-1
G	-2	-1	3	-2
T	-3	-1	-2	2

Tabulka 2.4: Příklad matice podobnosti: asymetrická

	A	**T**	**C**	**G**
A	1	-5	-5	-1
T	-5	1	-1	-5
C	-5	-1	1	-5
G	-1	-5	-5	1

Tabulka 2.5: Příklad matice podobnosti: symetrická

Hodnoty v matici se vhodně nastaví v závislosti na prováděné úloze. Existují tak jak matice symetrické (např. porovnání znaků A s T dává stejný výsledek jako opačná kombinace, tedy T s A) tak asymetrické (záleží na pořadí porovnávaných znaků, $T \rightarrow G \neq G \rightarrow T$). Ilustrace obou typů jsou k vidění v tabulkách 2.4 a 2.5. Pro využití v praxi jsou mnohem výhodnější symetrické matice, kvůli své téměř poloviční úspoře prostoru. Není to přesná polovina, neboť hodnoty na diagonále zůstávají součástí matice (ale lze ji odstranit za předpokladu, že shodnost znaků bude zpracovávána odděleně, nebo bude shodnost ohodnocena konstantní hodnotou pro všechny znaky). U plných proteinových matic tak lze uspořit až 190 paměťových míst.

Případné úspory na systémových prostředcích při použití matic podobnosti by se dalo dosáhnout seskupením znaků do skupin a matice by obsahovala ohodnocení ne pro jednotlivé znaky, ale ohodnocení pro celou skupinu. Pro DNA sekvence by se nabízela varianta zařazení nukleotidů do dvou skupin, jak ukazuje 2.6.

	AT	**CG**
AT	1	-2
CG	-1	1

Tabulka 2.6: Upravená matice podobnosti pro skupiny vstupních znaků

Složitým maticím se říká „parametrizované“. Jsou využívány hlavně pro proteinové řetězce a jsou obvykle sestaveny pro určité charakteristické úlohy, i když existují i více univerzální univerzální typy.

Základem pro jejich sestavení je databáze řetězců. Poté se sleduje, jak často se v nich vyskytují mezi jednotlivými znaky záměny. Tyto jsou potom promítnuty do matice. Výsledek může být reprezentován i desetinnými čísly, což není pro hardwarové řešení ideální a proto se převádí na celočíselné.

Existují dvě hlavní třídy parametrizovaných matic:

- PAM — Jsou založeny na globálním porovnávání navzájem podobných proteinových sekvencí. PAM1 je matice postavená na srovnáních sekvencí lišících se o 1%. Vyšší

řády PAM jsou odvozeny od PAM1. Často se vyskytují typy PAM100, PAM120 a PAM250.

- BLOSUM — Tyto jsou svázány s lokálním porovnáváním. Význačné jsou např. BLOSUM45, BLOSUM62, BLOSUM80. BLOSUM45 znamená, že byla vypočítaná ze sekvencí nelišících se více než o 45%. Z toho vyplývá, že pro porovnávání velmi podobných sekvencí se více hodí BLOSUM80.

2.3 Metody porovnávání řetězců

Ačkoliv jsou řetězce popsané v předchozím oddílu odlišné co do vzhledu a jejich významu, mohou být reprezentovány stejným způsobem a tím pádem algoritmicky zpracovávány těmi samými metodami (jen s drobnými změnami). Na všech typech se díky evolučnímu vlivu mohou obecně uplatnit tyto rozdíly mezi dvěma sekvencemi (a to v libovolném množství či pořadí):

- mutace
- vložení znaku/sekvence
- smazání znaku/sekcence

Každá taková modifikace řetězce má svou pravděpodobnost výskytu a podle ní je jí přiřazena určitá hodnota penalizace. Podstatou procesu porovnání je najít takovou sekvenci operací (vložení, smazání, mutace) aplikovaných na porovnávaný řetězec, při které získáme druhý. Pokud taková sekvence není jedna (a při porovnávání dlouhých řetězců je jich velmi mnoho), zajímá nás ta nejoptimálnější z nich - celkové skóre, dané souhrnem všech penalizací a bonusů za shodné části je nejpříznivější. Společné sekvenci se říká „zarovnání“ (alignment). Zarovnání se hledá buďto na společných podřetězcích, nebo na celé jejich délce. Jako výsledek často postačí i jen skóre porovnání.

Existuje mnoho metod pro porovnávání od jednoduchých až po sofistikované. Mezi základní (a jednodušší) můžeme zařadit např. Levenshteinovu vzdálenost či Hammingovu vzdálenost. Tyto se ovšem příliš nehodí pro použití v bioinformatice (avšak budou dále popsány, neboť usnadní pochopení složitějších metod) a v této oblasti se využívají hlavně postupy porovnávání, které lépe zohledňují jevy jako je vkládání nebo mazání znaků ze sekvencí. Budiž zde nazývány „algoritmy pro porovnání s mezerami“.

Pozn. Různé metody budou demonstrovány na velmi krátkých dvojicích DNA sekvencí: `AATCTATA` a `AAGATA`, případně `GCGA` a `CGGC`.

2.3.1 Zarovnávání bez mezer

Při tomto postupu se oba dva řetězce zarovnávají jako celek (nesmí být nikterak „roztrženy“ na více částí). Do celkového skóre se promítá pouze to, jestli si právě porovnávané znaky z obou řetězců odpovídají, či nikoliv. Celkové skóre pro zarovnání sekvencí *str1* a *str2* je tedy dáno vztahem 2.1.

$$\sum_{i=1}^{n} = \begin{cases} \text{skóre shody}, & \text{pokud } str1_i = str2_i \\ \text{skóre neshody}, & \text{pokud } str1_i \neq str2_i \end{cases} \tag{2.1}$$

kde n je délka té delší ze sekvencí $str1$ a $str2$. Skóre shody a skóre neshody mohou být zvolené konstantní hodnoty, nebo jsou definovány maticí, tzv. „maticí podobnosti“.

		A	**A**	**T**	**C**	**T**	**A**	**T**	**A**
	0	1	2	3	4	5	6	7	8
A	1								
A	2								
G	3								
A	4								
T	5								
A	6								

Tabulka 2.7: Levenshteinova vzdálenost: inicializovaná matice

Zde je uveden příklad všech možných zarovnání demonstračních řetězců metodou „bez mezer“. Pro ohodnocení byla použita matice 2.4 (s tím. že při porovnání znaku s prázdným místem se připočte 0).

```
AATCTATA   AATCTATA   AATCTATA
AAGATA      AAGATA      AAGATA
```

Ohodnocení jednotlivých případů:

$$\begin{aligned} &2+2-2+0+2+2+0+0=6 \\ &0+2-3+0-3-3-3+0=-10 \\ &0+0-3+0-2+2+2+2=1 \end{aligned} \tag{2.2}$$

Hammingova vzdálenost

Velmi jednoduchý postup porovnávání, který se aplikuje pouze na shodně dlouhé řetězce. Výsledkem algoritmu je skóre, které udává minimální počet záměn znaků, které jsou potřeba pro převedení z jednoho řetězce na druhý.

Už podmínka na stejnou délku vstupních řetězců ho tak vylučuje z použití v bioinformatice.

Levenshteinova vzdálenost

Nebo též „editační vzdálenost“. Je to algoritmus porovnávání dvou řetězců, kde skóre porovnání je dáno minimálním počtem operací potřebných pro převedení jednoho řetězce na druhý. Mezi povolené operace patří vložení znaku, smazání znaku či jeho záměna za jiný. Pokud je skóre rovné nule, pak jsou oba řetězce shodné. Jde v podstatě o zobecnění Hammingovy vzdálenosti.

Mějme matici A o velikostech $(m+1) \times (n+1)$. Vertikální indexy je indexován jeden z řetězců, horizontálními druhý. Vždy ale až od pozice 1 (délka řetězců je m a n). Nultý řádek popř. sloupec je vyhrazen pro inicializační hodnoty.

Dále se činnost algoritmu řídí následujícím vztahem:

$$A_{m,n} = \text{minimum} \begin{cases} A_{m-1,n} + 1 & //\text{smazání} \\ A_{m,n-1} + 1 & //\text{vložení} \\ A_{m-1,n-1} + \begin{cases} 1 & \text{pokud } str1_m \neq str2_n \\ 0 & \text{pokud } str1_m = str2_n \end{cases} & //\text{záměna} \end{cases} \tag{2.3}$$

Postupně (to jest řádek po řádku) se prochází tabulka a vyplňují se hodnoty podle vzorce 2.3. Výsledkem je tabulka 2.8.

		A	**A**	**T**	**C**	**T**	**A**	**T**	**A**
	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
A	2	1	0	1	2	3	4	5	6
G	3	2	1	1	2	3	4	5	6
A	4	3	2	2	2	3	3	4	5
T	5	4	3	2	3	2	3	3	4
A	6	5	4	3	3	3	2	3	**3**

Tabulka 2.8: Levenshteinova vzdálenost: vyplněná tabulka

		A	**A**	**T**	**C**	**T**	**A**	**T**	**A**
	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
A	2	1	0	1	2	3	4	5	6
G	3	2	1	1	2	3	4	5	6
A	4	3	2	2	2	3	3	4	5
T	5	4	3	2	3	2	3	3	4
A	6	5	4	3	3	3	2	3	3

Tabulka 2.9: Levenshteinova vzdálenost: zarovnání

Číslo v pravém dolním rohu říká, že řetězce mezi sebou můžeme převést za pomoci tří operací. Chceme-li dodatečně vědět, jaké jsou to operace, musíme projít tzv. „zpětnou cestou“ na začátek (do levého horního rohu). K tomu je potřeba si v průběhu výpočtu u každé buňky uchovávat stav, ze které předchozí (levé, horní, levé horní) buňky jsme dostali výsledek. Poté můžeme projít cestou a přitom sestavit posloupnost operací.

V tabulce 2.9 je uvedena jedna z nich. V tomto případě jsou max. 3 možné (s tímto ohodnocením). z cest:

Seznam odpovídajících operací při převodu `AATCTATA` na `AAGATA`:

1. záměna $T \rightarrow G$
2. smazání C
3. smazání T

2.3.2 Zarovnání s mezerami

Mezery (anglicky „gaps“) v sekvencích nám umožňují rozdělit sekvence na více částí. Díky tomu můžeme simulovat vložení nebo smazání znaku ze sekvence. Tato rodina algoritmů už je složitější.

Mezera se ve výpisu zarovnání označuje pomlčkou. Z pohledu jednoho řetězce pomlčka znamená smazání, z pohledu druhého představuje toto místo naopak vložení příslušného znaku. Proto se často při porovnávání podobně dlouhých řetězců neliší penalizace za vložení a smazání.

V následujícím příkladě může v řetězci dojít celkem k 28-mi různým pozicím mezer, zobrazeny jsou vybrané tři z nich:

$$\begin{array}{lll} \texttt{AATCTATA} & \texttt{AATCTATA} & \texttt{AATCTATA} \\ \texttt{AAG--ATA} & \texttt{AA-G-ATA} & \texttt{AA--GATA} \end{array} \tag{2.4}$$

Pro určité zvolené pozice mezer se vypočte skóre jejich podobnosti vztahem 2.5:

$$\sum_{i=1}^{n} = \begin{cases} \text{pokuta mezery}, & \text{pokud } str1_i = \text{'–' nebo } str2_i = \text{'–'} \\ \text{skóre shody}, & \text{pokud není mezera a } str1_i = str2_i \\ \text{skóre neshody}, & \text{pokud není mezera a } str1_i \neq str2_i \end{cases} \tag{2.5}$$

kde n je délka té delší ze sekvencí $str1$ a $str2$.

Pro příklad 2.4 uvedený výše si zvolíme konstanty: 2 pro shodu, 0 pro neshodu a -1 pro smazání či vložení. Jednotlivá skóre budou vypadat jak ukazuje rovnice 2.6.

$$\begin{aligned} 2+2+0-1+0+0-1+2 &= 4 \\ 2+2-1+0-1+2+2+2 &= 8 \\ 2+2-1-1+0+2+2+2 &= 8 \end{aligned} \tag{2.6}$$

Tento postup ohodnocení však nebere ohledy na skutečnou pravděpodobnost, se kterou se mezery vyskytují v reálném světě. Mezery totiž mají tendenci se slučovat do delších skupin, než se vyskytovat odděleně. Pokud je tento fakt vztažen k uvedeným příkladům, pak z bioinformatického hlediska je 2. zarovnání mnohem nepravděpodobnější než 1. a 3. případ, v nichž jsou mezery sloučeny do jedné delší sekvence. Tento poznatek je též potřeba zohlednit v algoritmu. Byly proto vyvinuty komplexnější verze jež budou uvedeny v následujících částech.

2.3.3 Globální

Globální metoda znamená, že řetězce se mezi sebou porovnávají po celé jejich délce, z jednoho konce na druhý (end to end). Pro delší porovnávané řetězce už nelze hrubou silou zkoušet všechny možné pozice mezer k nalezení optimálního skóre (zarovnání). Mějme třeba středně dlouhý řetězec o 100 znacích. Bude-li druhý řetězec mít 98 znaků, možností kam umístit mezery bude 4950. Pokud však bude mít 95 znaků, rázem máme asi 55 miliónů možností. Takový rozsah úlohy už je časově neúnosný. Hlavním představitelem je metoda Needleman-Wunsch.

Needleman-Wunsch algoritmus

Je to dynamický algoritmus pocházející ze 70. let, který garantuje nalezení zarovnání s maximálním skóre (pro aktuální nastavená pravidla) v rozumném čase. Také používá, jako předchozí metody, pro výpočet dvourozměrného pole.

Na předpise 2.8 je vidět rekurzivní pojetí algoritmu jako dynamické metody.

$$\begin{aligned} S_{0,n} &= d * n; \\ S_{m,0} &= d * m; \\ S_{m,n} &= max(A_{m-1,n-1} + f_{(str1_m, str2_n)}, A_{m,n-1} + d, A_{m-1,n} + d) \end{aligned} \tag{2.7}$$

kde d představuje penalizaci za mezeru a $f_{(x,y)}$ je skóre z porovnání znaků v substituční matici. Tento předpis ale pracuje jen s jednoduchými penalizacemi mezer.

		A	**C**	**T**	**C**	**G**
	0	-1	-2	-3	-4	-5
A	-1	1				
C	-2					
A	-3					
G	-4					
T	-5					
A	-6					
G	-7					

Tabulka 2.10: Needleman-Wunsch: inicializovaná tabulka

Pokud se přepíše na sekvenční, je obecný předpis:

$$S_{m,n} = \text{maximum} \begin{cases} S_{m-1,n} - f_{mezera} & //\text{smazání} \\ S_{m,n-1} - f_{mezera} & //\text{vložení} \\ S_{m-1,n-1} + f_{subst}(str1_m, str2_n) & //\text{porovnání znaků} \end{cases} \tag{2.8}$$

kde $f_{subst}(str1_m, str2_n)$ Představuje funkce realizovanou substituční maticí. Více v části 2.2.3.

Pomocí mezerové funkce f_{mezera} se inicializují nultý sloupec a řádek porovnávací tabulky. Bývá definována různě. Nejčastěji jako jednoduchá $f_i = i * p$. Tedy výše penalizace za přidání mezery je po celé její délce shodná.

Funkce která bere větší ohledy na evoluční hledisko může vypadat:

$$f_{mezera} = \begin{cases} p_{init} & \text{pokud délka mezery} = 0 \\ p * \text{délka mezery} & \text{pokud jinak} \end{cases} \tag{2.9}$$

Mezerová funkce potřebuje v tomto případě pro svou činnost zadání dvou konstant — pokutu za otevření mezery a pokutu za přidání znaku do mezery, přičemž by mělo platit $|p_{init}| > |p|$. Podobná pravidla se dají aplikovat nejenom na otevření mezery, ale i její uzavření.

Algoritmus pracuje následujícím způsobem: Nejprve se inicializuje tabulka 2.10 mezerovou funkcí (jednoduchá verze s penalizací -1 za vložení mezery). Postupně se podle vzorce 2.8 vyplní políčka v tabulce 2.11 (pro porovnání znaků použita matice identity 2.3). Každá buňka si pamatuje, ze které sousední buňky převzala (a upravila) hodnotu. Výsledné skóre 2 se nachází v pravém dolním rohu.

V poslední (nepovinné) fázi výpočtu se provádí „zpětná cesta“ skrze vyplněné pole, aby se získalo „grafické zarovnání“. Cesta nemusí být jen jedna — pokud je ale požadováno zobrazení více případných zarovnání, musí si algoritmus v každém políčku pro pozdější zpětný průchod ukládat ne jednu, ale ideálně všechny směry políček, ze kterých získala svou hodnotu. Více směrů se ukládá pouze v případě, že některé hodnoty (po přičtení penalizací a bonusů) jež políčko získává od sousedů jsou shodné. To znamená že každé políčko může uložit maximálně 3 směry. Prakticky se ale nevyskytují hardwarové implementace umožňující přečtení všech zarovnání. Zpětné procházení tabulkou zvyšuje délku provádění na dvojnásobek, než pouhé zjištění skóre, proto je volitelné (při porovnání mnoha řetězců se spokojíme s výsledkem skóre a pokud je ohodnocení dostatečně dobré, vyžádá se i celé zarovnání). Zarovnání se taktéž běžně nenačítá pro velmi dlouhé sekvence (řádově tisíce znaků), ale pro kratší, kdy je určeno pro grafickou reprezentaci výsledku.

		A	**C**	**T**	**C**	**G**
	0	-1	-2	-3	-4	-5
A	-1	1	0	-1	-2	-3
C	-2	0	2	1	0	-1
A	-3	-1	1	2	1	0
G	-4	-2	0	1	2	2
T	-5	-3	-1	1	1	2
A	-6	-4	-2	0	1	1
G	-7	-5	-3	-1	0	**2**

Tabulka 2.11: Needleman-Wunsch: vyplněná tabulka

		A	**C**	**T**	**C**	**G**
	0	-1	-2	-3	-4	-5
A	-1	1	0	-1	-2	-3
C	-2	0	2	1	0	-1
A	-3	-1	1	2	1	0
G	-4	-2	0	1	2	2
T	-5	-3	-1	1	1	2
A	-6	-4	-2	0	1	1
G	-7	-5	-3	-1	0	2

Tabulka 2.12: Needleman-Wunsch: zarovnání

Podle vybrané cesty z tabulky 2.12 se sestaví zarovnání obou sekvencí. Směr cesty nahoru znamená pro řetězec ležící vodorovně smazání znaku, pohyb doleva vložení znaku a pohyb po diagonále mutaci nebo shodu.

```
AC--TCG
ACAGTAG
```
(2.10)

2.3.4 Lokální

Lokální porovnávání je proces, který hledá mezi dvěma sekvencemi jejich nejlepší společné podřetězce. Nejsou to nutně identické podřetězce (i ony mohou být rozděleny mezerami).

Smith-Waterman algoritmus

Dynamický algoritmus, vzniklý v 80. letech. Tento algoritmus je v mnohém podobný, v posloupnosti prováděných kroků je stejný, avšak v bioinformatické praxi je významnější než Needleman-Wunsch. Více hodí pro typickou aplikaci, kdy je máme poměrně krátkou sekvenci (tzv. dotazovací řetězec) a potom velmi rozsáhlou databázi bioinformatických dat. Problém zní: „Nachází se v databázi náš krátký vzorek (respektive blízký vzorek)?“ Řídí se

		A	**A**	**C**	**C**	**T**	**A**	**T**	**A**	**G**	**C**	**T**
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	1	0	0
C	0	0	0	1	1	0	0	0	0	0	2	1
G	0	0	0	0	0	0	0	0	0	1	0	1
A	0	1	1	0	0	0	1	0	1	0	0	0
T	0	0	0	0	0	1	0	2	1	0	0	1
A	0	1	1	0	0	0	2	0	3	2	1	0
T	0	0	0	0	0	1	1	3	2	2	1	2
A	0	1	1	0	0	0	2	2	4	3	2	1

Tabulka 2.13: Smith-Waterman: zarovnání

obecným vzorcem 2.11.

$$A_{m,n} = \text{maximum} \begin{cases} 0 & \\ A_{m-1,n} - f_{mezera} & //\text{smazání} \\ A_{m,n-1} - f_{mezera} & //\text{vložení} \\ A_{m-1,n-1} + f_{subst}(str1_m, str2_n) & //\text{porovnání znaků} \end{cases} \tag{2.11}$$

V provedení se od Needleman-Wunsch algoritmu liší v několika aspektech. Především nedovoluje dílčím skóre poklesnout pod nulu. Stale-li se tak, dojde k „roztržení“ cesty, a tím zahájení nového potenciálního podřetězce. Zajímavé oblasti se tak mohou vyskytnout kdekoliv v tabulce. Taktéž skóre může při průchodu cestou jak klesat, tak si stoupat, proto se při zpětném průchodu řídí jen pomocí vektorů (ne průchod až po nalezení nuly). Po vypočtení tabulky se navíc musí najít (nebo se průběžně uchovává) buňka, s nejvyšším skóre — ta označuje konec nejlepšího zarovnání a odtud se provádí zpětný průchod.

Je-li podřetězců se stejným vysokým skóre více, záleží na implementaci algoritmu, jak se s faktem vypořádá. Vrací buď jedno zvolené zarovnání, předem daný maximální počet zarovnání či všechny (nejobtížnější).

Dále se na začátku neprovádí inicializace pomocí mezerové funkce, aby se tak nepenalizovaly mezery na začátku a konci podřetězců, dochází pouze k vynulování příslušných políček.

Ilustrace porovnání sekvencí `AACCTATAGCT` a `GCGATATA` v tabulce 2.13 používá skóre +1 pro shodu a shodně -1 pro penalizaci mezery a neshodu.

Jako nejvyšší skóre se našlo 4 a odtud se zpětně prochází až na konec cesty. Nalezené zarovnání má tvar:

```
AACCTATAGCT
----TATA---
```
(2.12)

2.3.5 Ostatní metody

Semi-globální metoda

Globální metoda většinou dobře nezarovná dva poměrně délkově odlišné řetězce. Bude do něj vkládat příliš mnoho mezer, které degradují skóre. A přitom může existovat poměrně dlouhá společná sekvence, kterou algoritmus neodhalí.

Vycházíme z předpokladu, že mezery, které jsou vloženy na začátku nebo na konci řetězce, většinou nemají biologický význam a plynou spíše z nekompletnosti řetězce.

Algoritmus Needleman-Wunsch jde snadno modifikovat, aby tyto druhy mezer nepenalizoval a namísto toho je vynechal. Na začátku algoritmu se pouze neprovede inicializace tabulky podle mezerové funkce. Postačí vynulování příslušných políček. Dále je ještě potřeba při výpočtu sledovat dosažení posledního sloupce nebo řádku tabulky a poté už neměnit celkové skóre pro zarovnání až po dosažení pravého dolního rohu.

Takto vzniklý algoritmus však svými vlastnostmi nepřekonává plnohodnotný lokální Smith-Watermanův algoritmus, neboť optimálně nezjistí začátek a konec nejlepšího společného podřetězce. Jeho výhodou je zejména snadné vytvoření s již implementované Needleman-Wunsch metody.

2.3.6 FASTA

Je rychlý lokální heuristický algoritmus, který používá vyhledávací tabulku pro urychlení výpočtu. Citlivost metody a jeho rychlost jsou závislé na délce slova v tabulce. Je sice méně citlivější než Smith-Waterman, ale zato rychlejší.

2.3.7 BLAST

Tak jako *FASTA*, je i tento algoritmus lokální. Nabízí však ještě vyšší rychlost s poměrně slušnou citlivostí. Pochází z 90. let a pracuje tak, že zpracovávané řetězce rozděluje na malé úseky fixní délky a vyhledává takové podsekvence. Tyto dva algoritmy (a jejich četné modifikace) jsou nejčastěji používanými metodami pro práci nad genetickými databázemi. Zejména pro jejich rychlost se běžně používají i jejich softwarové varianty.

2.4 Používané hardwarové architektury

2.4.1 Vysoce optimalizované

Typickou hardwarovou realizací úlohy porovnávání je specializovaná aplikace fungující podle jednoho zvoleného algoritmu a zpracovávající jeden vybraný typ vstupních dat (řetězců). Právě taková zařízení jsou nejčastěji využívány pro výzkum a zpracování bioinformatických dat v rozsáhlých databázích vzorků. Jejich výhodou je samozřejmě vysoká efektivita pro vybranou úlohu, ale je s tím spojena i určitá omezenost přizpůsobovat si zpracování aktuálním požadavkům.

Často je u nich využita speciální optimalizace, která umožňuje pomocí 1-bit informace předávat informace mezi výpočetními elementy. Do této rodiny patří řešení popsané v [10] a [4].

2.4.2 Architektury využívající rekonfigurace

Využívá vlastnosti některých FPGA obvodů měnit za chodu svou konfiguraci (vnitřní zapojení). S úspěchem se jí používá např. pro změnu konstant pro penalizaci, nebo pro změnu zapojení celé části obvodu. Dynamická rekonfigurace má velký potenciál, ale je obtížná na vývoj. Pro oblast porovnávání řetězců se jí zabývá práce [1].

2.4.3 Vláknové zpracování

Snaží se o maximální využití času. Pokud např. v průběhu výpočtu některá jednotka nepracuje, nebo čeká na výsledky, je snahou v tomto přístupu ji „zaměstnat“ jiným úkolem. Vlákna využívající řešení tak mohou zpracovávat více vstupních řetězců současně. např. Více v [9].

Zajímavým přístupem prezentovaným v [2] je dopředný spekulativní výpočet za účelem zkrátit „kritickou cestu“ a zvýšit tak maximální frekvenci a propustnost. Každá buňka posílá sousedovi upravené penalizace pro substituci, vložení a smazání ještě před tím, než je zjištěno, která z nich je platná. V době, kdy je toto vypočteno, má už soused předpočítané své tři hodnoty jen se mu oznámí, která z nich je platná. Tento přístup, jak je uvedeno, zkracuje „kritickou cestu“ 2,5x s tím, že nároky na systémové prostředky vzrostly jen o 20%.

2.4.4 Generická architektura

Genericky navržené obvody umí nastavovat své parametry za běhu, a pokud jsou dobře navrženy, postihnou celou škálu úloh v jednom designu. Co získávají na univerzálnosti, to naopak ztrácí na výkonu. Je potřeba najít rozumný kompromis. Jedna z obecných architektur je popisována v [3].

Kapitola 3

Praktická část

3.1 Návrh vhodné architektury

Z předchozího přehledu je vidět, že v porovnávacích aplikacích máme na výběr z mnoha postupů a ty lze navíc ještě ladit mnoha parametry. V návrhu musí být zohledněno, zdali chceme vytvořit obecnou architekturu podporující široké spektrum nastavení a režimů činnosti, nebo se spokojíme s konkrétní aplikací pro určitou úlohu.

Aplikace dbající o obecnost řešení jsou omezeny v tom, že mají značně omezen prostor pro optimalizaci a urychlování. Na druhou stranu specializované implementace mohou být (a bývají) kvalitně vyladěny aby co nejvíce využívaly výhod svých platforem a dosahovaly tak vyšších výkonů. Tyto převyšují výkonnost softwarových implementací prováděných např. na PC i o několik řádů.

Co se týče používaných výpočetních platforem, existují různá řešení. Pro tuto práci byla zvolena implementace na FPGA (jedná se o čip s programovatelnými hradlovými poli) akcelerační kartě pro obecné použití. Konkrétně typ s čipem Spartan3 od firmy Xilinx. Z dalších uvádím např. aplikace na ASIC čipech (obvody vyrobené na míru podle návrhu zákazníka), výpočetní serverové clustery, nebo víceprocesorové systémy. FPGA přístup se z této nabídky jeví jako relativně levné, dostupné a přitom jako rozumně výkonné řešení.

Materiály o dosud vytvořených aplikacích založených na FPGA technologii uvádějí, že provádějí své výpočty několikrát až několik desítek krát rychleji než softwarová řešení.

3.2 Požadavky na řešení

Aplikace by měla umožňovat na začátku nastavení do vybrané konfigurace, od které se bude odvíjet typ zadávaných a výstupních dat.

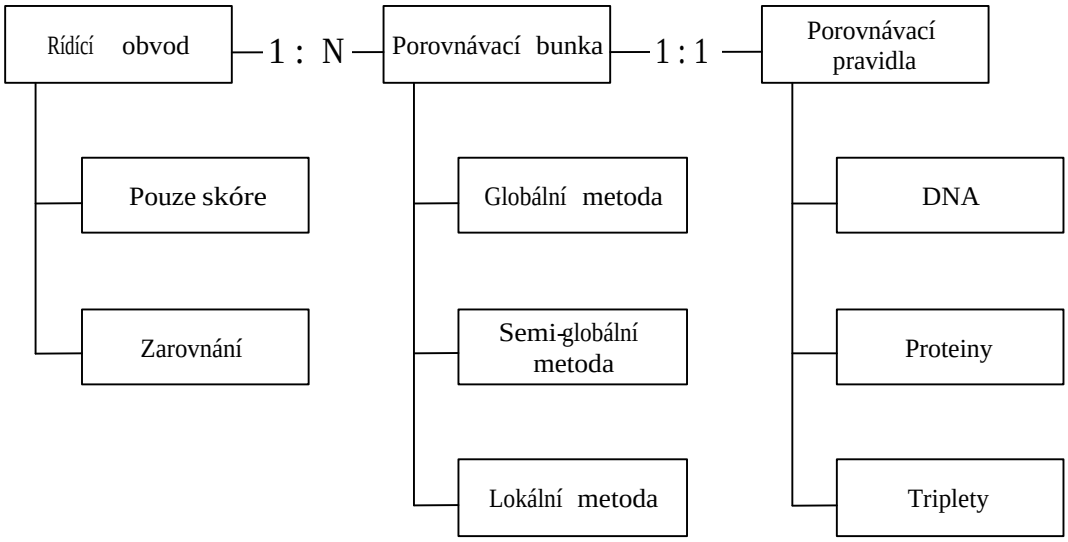
Zpracovávat se budou následující typy sekvencí:

- DNA/RNA sekvence (oba typy 4 různé znaky)
- Proteinové sekvence (20 různých aminokyselin)

Podporované režimy činnosti:

- Globální zarovnání sekvencí (postaveno na Needleman-Wunsch algoritmu)
- Lokální zarovnání (podle Smith-Waterman algoritmu)

Typ poskytovaného výsledku:



Obrázek 3.1: Kardinalita a vztahy mezi funkčními bloky

- Celkové skóre porovnání
- Výpis nejlepšího nalezeného zarovnání
- Případně výpis více nalezených zarovnání

Také by mělo být umožněno uživatelské nastavení pravidel porovnání zvoleného druhu sekvencí: Toto zahrnuje hodnoty skóre pro shodu prvků v sekvenci, dále pokutu za neshodu prvků (u DNA/RNA je zamýšleno nastavení pomocí matice podobnosti). Penalizace pro vložení či smazání prvku je reprezentována hodnotami parametrů jako jsou „zahájení mezery“, nebo „pokračování v mezeře“ (bodové postihování mezer může být rovněž řízeno vhodnou funkcí).

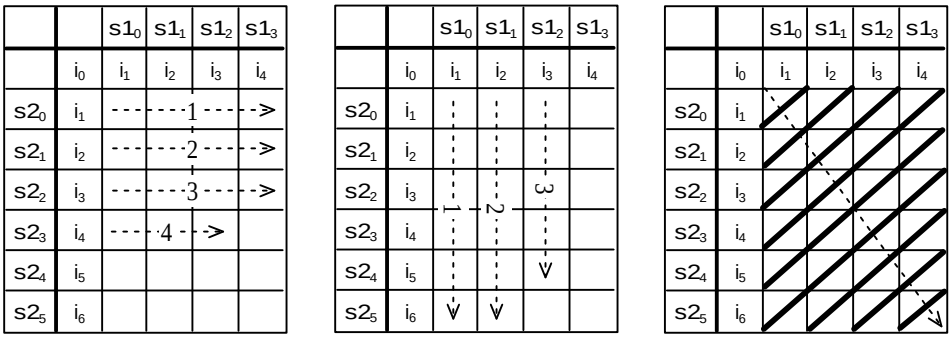
Vhodné by bylo zařadit systém vyrovnávacích pamětí. Např. pro načtení více porovnávaných sekvencí. Umožnilo by se tak v jednom čase vkládání dat pro zpracování a v druhé fázi pouze načítání výsledků, přičemž výpočet neustále probíhá až do zpracování celého obsahu paměti.

3.3 Schéma architektury

První nástin požadavků na architekturu je vidět na tomto schématu 3.1. Obrázek znázorňuje obecné schéma a zejména vztahy mezi hlavními součástmi aplikace. Stěžejní je fakt, že hlavní výpočetní síla je soustředěna v buňkách.

Výpočetní buňky se chovají podle svého nastavení (globální, lokální). Buňka jako vstup dostane dva porovnávané znaky a dále tři čísla ze sousedních buněk (představující jimi vypočtené skóre). Z nich se podle nastavených pravidel získá aktuální skóre. Potom se musí postarat o distribuci nového skóre jejím třem sousedům. Distribuují se i oba vstupní znaky, ty ovšem jen dvěma sousedům (doprava a dolů).

Aby mohla buňka zpracovávat skóre, musím mít přístup ke konfiguračním datům (hodnoty penalizací pro vkládání a mazání), nebo tu jsou přímo uloženy. O vlastní porovnání dvou znaků se stará jiná část.



Obrázek 3.2: Postup výpočtu v buňkách

Dalším údajem, který si každá buňka musí pamatovat je, které ze tří dílčích skóre bylo nakonec vybráno pro výpočet aktuálního skóre. Této informace se využije ve volitelné 2. fázi výpočtu, kdy se provádí zpětný průchod polem buněk a sestavuje se zarovnání.

Porovnávací pravidla. Tato jednotka pracuje společně s buňkami — každá obsahuje jednu tuto jednotku. Dostane na vstup dva znaky a jako výsledek vrací skóre porovnání (buňce nezáleží na informaci jestli znaky jsou nebo nejsou shodné). Pro každý typ sekvencí existují odlišná skórovací pravidla. Mohou mít podobu např. konstant (např. X — když jsou znaky shodné, Y — když nikoliv), komplexnější (avšak taky náročnější na prostor na čipu) je matice podobnosti.

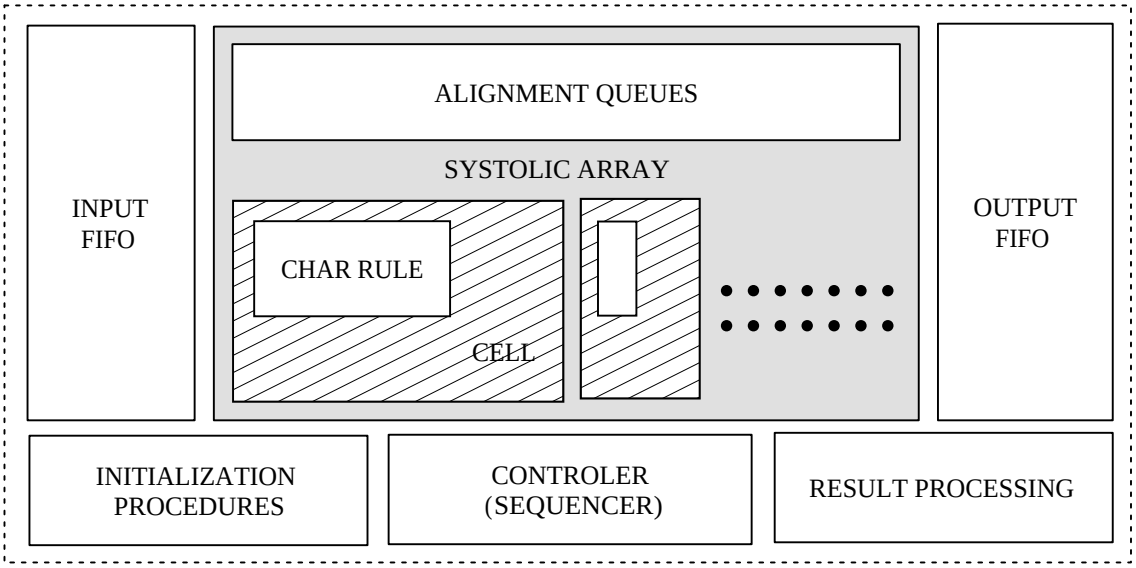
Řídící obvod dohlíží na správné časování výpočtu (podle délky vstupu), zásobuje buňky vstupními daty, spravuje paměť výsledků a spouští jednotlivé fáze výpočtu. Obecně lze fáze výpočtu stanovit:

1. Celková inicializace obvodu — uvedení do počátečního stavu
2. Konfigurace — nastavení do požadovaného režimu (typ výpočtu, druh zpracovávaných sekvencí, parametry pro skórování formát výsledků, ...), stačí ji udělat jen jednou.
3. Inicializace tabulky podle mezerové funkce (pokud se provede pro celou délku tabulky, není nutno opakovat až do další změny konfigurace)
4. Načtení vstupních řetězců
5. Výpočet skóre
6. Výpočet zarovnání (pokud je zapnuto)
7. Uložení výsledků

3.4 Možnosti urychlení

Hardwarové řešení porovnávání slibuje oproti softwarovému značné urychlení. Jaké jsou tedy cesty k dosažení co nejrychlejšího výpočtu?

Výpočet tabulky trvá v algoritmu největší dobu a taky změna v přístupu představuje největší zdroj urychlení. Díky datové závislosti jedné buňky pouze na třech vedlejších, se může vypočítat celá diagonála současně. Dostáváme tak oproti sekvenčnímu softwarovému



Obrázek 3.3: Hierarchie jednotek

řešení s časovou složitostí průchodu tabulkou buněk $O(m*n)$ složitost o třídu nižší: $O(m+n-1)$. S využitím hardwarové paralelizace se tak z kvadratické přechází na téměř lineární složitost. Postup je znázorněn na obrázku 3.2.

3.5 Design a jeho komponenty

Jako výpočetní architektura byla zvolena systolická struktura uspořádaná do lineárního pole. Jelikož používané algoritmy jsou předepsány pro dvourozměrné pole, musel se návrh odpovídajícím způsobem přetransformovat na jednorozměrné – lineární. Skládá se z několika součástí, které budou na následujících stránkách jednotlivě popsány. Původní návrh dvourozměrného pole je nastíněn v příloze.

3.5.1 Reprezentace znaků

Je v běžném binárním kódování. V architektuře nejsou definována pevná pravidla pro překlad znaků, proto je lze libovolně stanovit, aby to vyhovovalo nastavené datové šířce. Např. pro DNA: A = 0, C = 1, G = 2, T = 3 a podobně pro proteiny.

3.5.2 Jednotka vyhodnocení znaků

Označována jako `CHAR RULE`. V navržené architektuře je vždy implementována jako vyhledávací RAM tabulka s datovou šířkou a kapacitou podle vstupních znaků. Tento přístup ale rozhodně není úsporný k systémovým prostředkům FPGA čipu, protože spotřebovává mnoho komponent. Výhodou je, že tabulka je plně modifikovatelná za běhu a umožňuje tak změnit strategii ohodnocování.

V konfiguraci pro DNA řetězce má kapacitu 16 znaků. Pro proteiny s 20-ti různými znaky reprezentovanými na 5-ti bitech by to bylo už $2^5 * 2^5 = 1024$ paměťových míst. Pro syntézu byla alespoň provedena částečná redukce prostoru, aby nebyl zaplněn celý (5+5)-ti

bitový adresový prostor: celková adresa do tabulky se skládá ze dvou dílčích položených za sebe. V binární reprezentaci 10011|10011 = 627. Nemůže se vyskytnout adresa vyšší.

3.5.3 Výpočetní buňka (element)

Hlavní částí navrženého designu buňky je čtyřvstupový přepínač, který ma na vstupu hodnoty ze tří virtuálních směrů. Funkce `MAX` na jeho výstup přepíná tu, představující nejvyšší bonus (nejnižší penalizaci). Výstup funkce `MAX` je hodnota, která se uchovává v zásobníku pro případný pozdější zpětný průchod. Čtvrtý vstup představuje nulu, jak je předepsáno v 2.11. Nula se započítává v případě Smith-Waterman algoritmu. Přepnutí do režimu Needleman-Wunsch je v rámci této jednotky realizováno tak, že přepínací vodič je spojen s nejvýznamnějším bitem signálu „nula“ a jeho logická 1 změní hodnotu v tomto signálu na minimální záporné číslo na dané datové šířce. Všechny porovnání a operace totiž probíhají ve znaménkové aritmetice.

Obsahy registrů `C_M` a `C_N` znamenají penalizaci za vložení a smazání znaku a jsou nastavitelné. Přičítají se každý cyklus k hodnotám `A_M` a `A_N` předávaným ze sousedních buněk. Jelikož je buňka přetransformována z dvourozměrného předpisu, je hodnota `A_M` předávána jako horní soused pomocí zpětné vazby.

Buňka má v po celou dobu výpočtu uložen jeden znak z „dotazovacího řetězce“, který tvoří jeden ze vstupů do jednotky pro porovnávání znaků. Druhý znak, který patří do jiné sekvence znaků (databázová sekvence) se předává každý cyklus mezi sousedy.

Speciální (inicializační) registr `A_M` je potřeba pouze pro první krok výpočtu, kdy je nastaven (a polem se dále pohybuje) signál `INIT`, který obsah registru `A_M` přivede na vstup sčítačky.

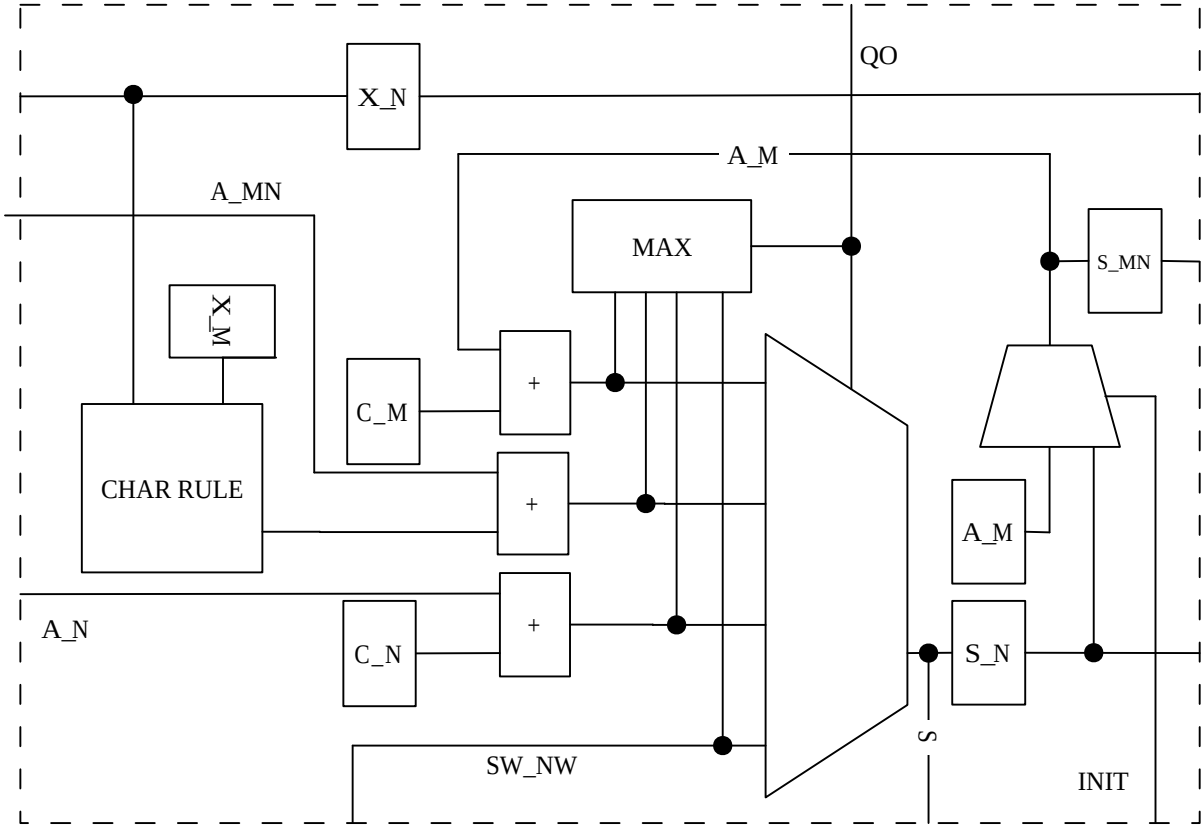
Jednotka je implementovaná pouze s jednoduchou (lineární) penalizací mezer – není definován žádný stav jestli už mezera byla v předchozím kroku vložena či nikoliv. Funkčnost pro vylepšenou mezerovou penalizaci umožňuje rozdělit penalizaci za mezeru na dvě složky: pokuta za otevření mezery a pokuta za pokračování v mezeře. Ta první z nich v této jednoduché buňce chybí a je to na škodu Smith-Waterman algoritmu. Je potřeba přidat ještě dva další registry `G_M` a `G_N`, kde budou uloženy konstantní (v době výpočtu) pokuty za otevření mezery. Dále musí být k dispozici jednobitové informace, z dvou sousedních buněk, jestli u nich byla vložena mezera. Na základě vstupních informací zvolíme hodnotu penalizace a výsledky předáme dalším sousedům.

3.5.4 Implementace maxima

Uvažovaná funkce `MAX` se nachází v každé buňce, kde rozhoduje, která z (již upravených!) vstupních hodnot bude výstupní, tedy:

$$S_{m,n} = \mathrm{MAX} \begin{cases} S_{m-1,n} \\ S_{m,n-1} \\ S_{m-1,n-1} \\ 0 \end{cases} \tag{3.1}$$

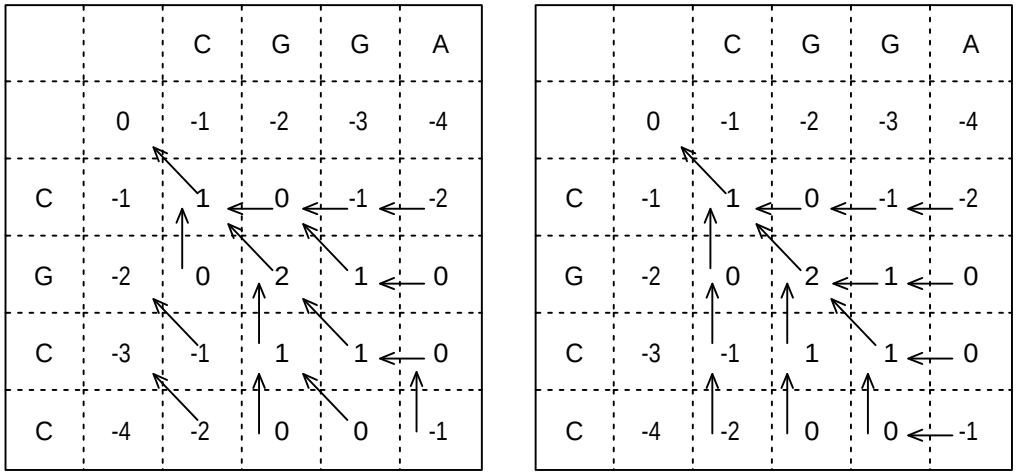
Algoritmus přesně nestanovuje pravidla, a tak je implementace volná. Zde jsou ukázány dva různé přístupy a příklady výsledků pro ně. Rozdíly mezi nimi se neuplatňují v případě, že je počítáno pouze skóre podobnosti. Nastávají pro buňku, která si pamatuje jen jeden směr pro zpětnou cestu a má „rozhodnout“ mezi více stejnými vstupy. Zpětná cesta je tak částečně ovlivnitelná implementací maxima.



Obrázek 3.4: Schéma lineární výpočetní buňky

	1	2
Needleman-Wunsch	CGGA- CGC-C	CGG-A CGCC-
Smith-Waterman	CG-- CG--	CG-- CG--

Tabulka 3.1: Výsledná zarovnání pro rozdílné implementace funkce `MAX`



Obrázek 3.5: Needleman-Wunsch, odlišné implementace maxima

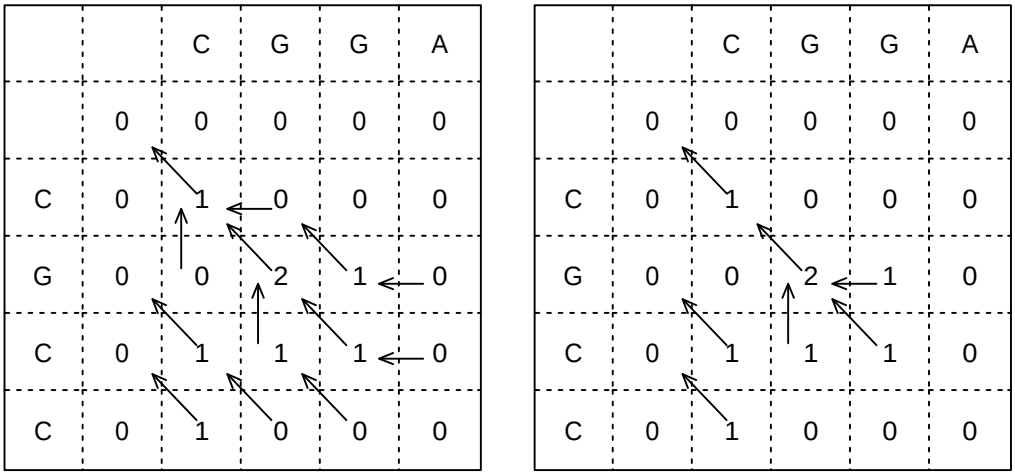
```
comp <= comp2 when (comp2 >= comp1) else comp1;
comp1 <= sum_m when (sum_m >= zero) else zero;
comp2 <= sum_mn when (sum_mn >= sum_n) else sum_n;

comp <= comp2 when (comp2 > comp1) else comp1;
comp1 <= sum_m when (sum_m > zero) else zero;
comp2 <= sum_mn when (sum_mn > sum_n) else sum_n;
```

Zatímco první část VHDL kódu „preferuje“ výběr hodnoty `sum_mn`, což je hodnota od levého horního souseda, druhá dává nejvíce přednost výběru signálu `zero` (jeho význam vysvětlen v 3.5.3). Příklady jsou provedeny na sekvencích `CGCC` a `CGGA`. Porovnávací matice je 2.5, mezerová penalizace s pokutou -1 je lineární. Výsledná zarovnání pro oba algoritmy a implementace maxima jsou vypsány v 3.1.

3.5.5 Jednotka zpracování výsledků

Toto není funkční blok nacházející se na konci systolického pole, nýbrž blok samostatný pro každou jednu buňku. Tato jednotka předá správný výsledek sousedské buňce podle aktuálního nastavení. Může pracovat v následujících režimech:

- **Smith-Waterman**: sousedovi předává buďto své vypočtené skóre `S` (a jeho index – pozici v zásobníku), nebo skóre `MAX_M` (a index) svého předchůdce. Podle toho, které je větší.



Obrázek 3.6: Smith-Waterman, odlišné implementace maxima

- **Needleman-Wunsch**: předává dále vždy jen svoje skóre a index (i když index tentokrát není potřeba pro další zpracování, protože zajímavá je z hlediska výsledků pouze pravá krajní buňka).
- **Přeposílač**: vždy posílá dál hodnotu, kterou mu předal soused. V tomto režimu se nachází v případě, že buňka není součástí dotazovacího řetězce a díky ní se dostanou výsledky aktivních buněk na pravý kraj k finálnímu zpracování.
- **Čtení zásobníku**: signálem `LOAD_ALIGNMENT` se vstup jednotky přepne k výstupu zásobníku a v příštím cyklu bude číst z něj.

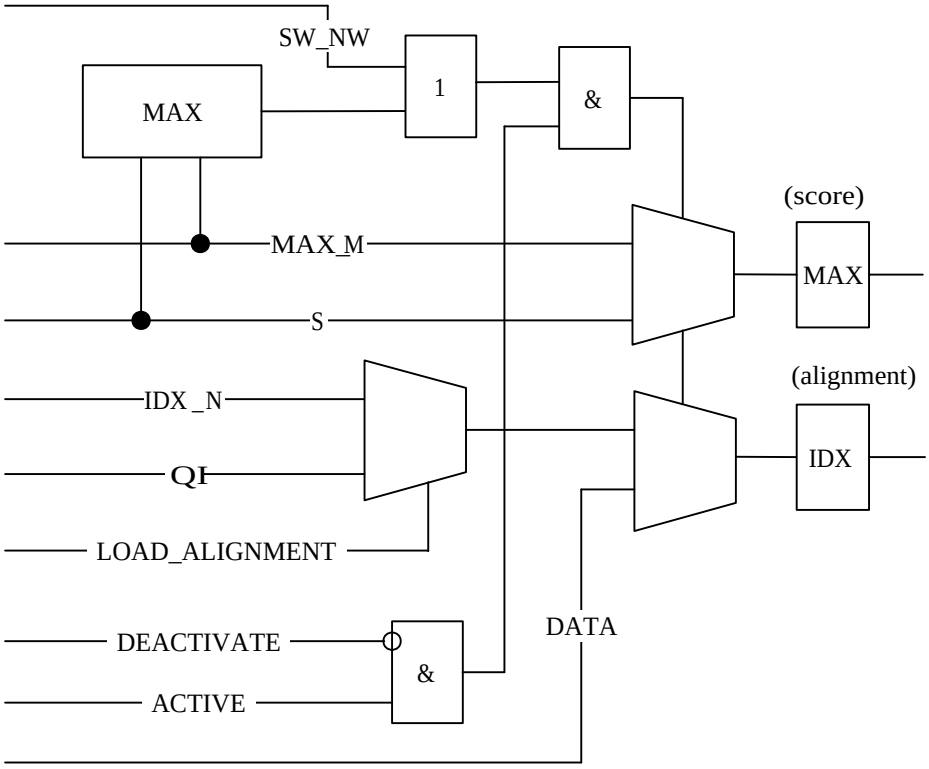
Schéma jednotky je k prozkoumání zde 3.7. Kromě lokálního signálu `ACTIVE` je jednotka řízena hlavně globálními signály, které umožňují okamžité přepnutí režimu všech buněk. Např. `DEACTIVATE` způsobí změnu módu na „přeposílač“ a spolu se signálem `LOAD_ALIGNMENT` mohou kompletně (vyjma zásobníkových signálů) řídit načítání zarovnání.

3.5.6 Vstupní a výstupní fronty

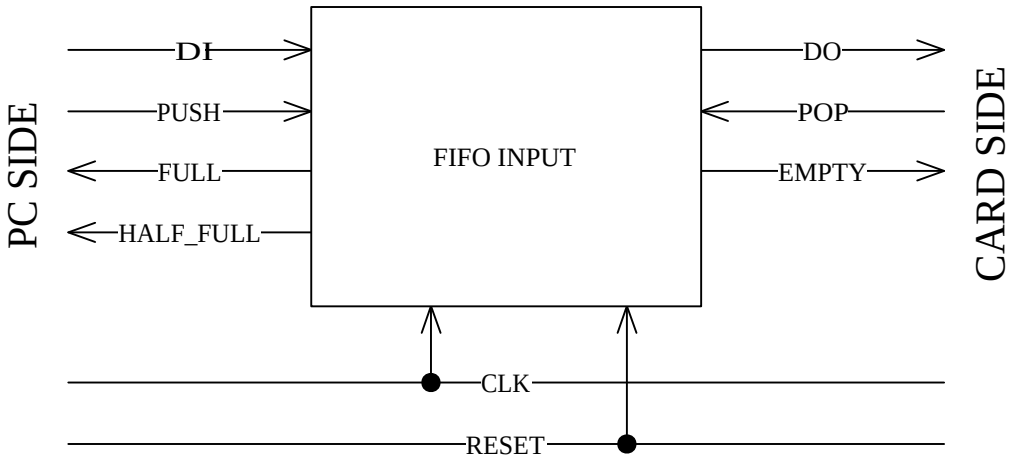
Slouží jako vyrovnávací paměť pro uložení nejen řetězců na zpracování, ale i inicializačních dat a výsledků. Jedna z front je použita i jako interní pro předávání dat mezi sekvencerem a jednotkou pro zpracování finálního výsledku.

Nejpoužívanější frontou je vstupní (obrázek 3.8). Plní se ze strany počítačové sběrnice a její výstup směřuje přímo k výpočetním elementům (tedy přesněji k nejlevějšímu elementu). Umožňuje simultální čtení i zápis. Obě činnosti jsou synchronizované společným hodinovým signálem. Jejich konstrukce umožňuje snadné zřetězení bloků, a tím navýšení kapacity.

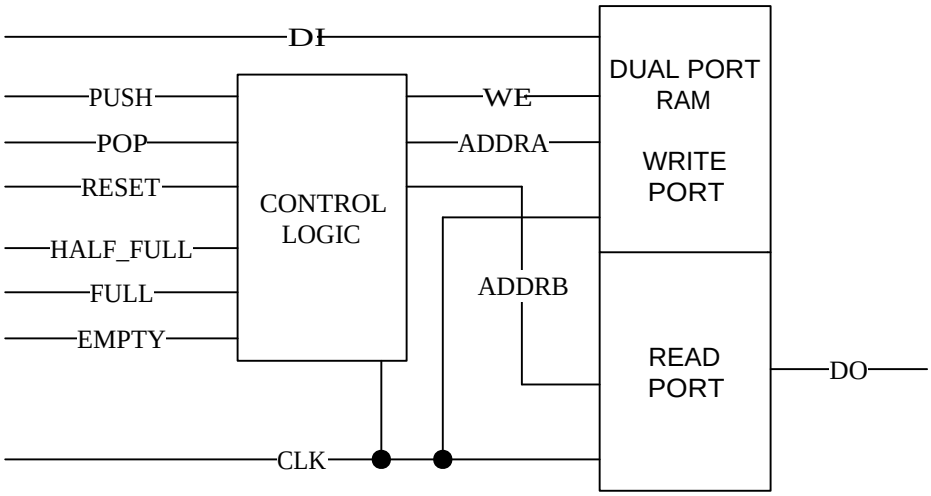
Pro její implementaci je na kartě použita vestavěná bloková paměť, konkrétně její dvouportová varianta. Kontrolní logika zahrnuje 2 čítače uchovávající adresu prvku na začátku fronty a adresu kam bude zapsán další vložený prvek. Data se do paměti ukládají cyklicky (po přetečení adresy se ukládá opět od začátku). Aby se při cyklickém zápisu předešlo přepsání platných prvků (nebo vrácení neplatných prvků), uchovává se aktuální počet prvků ve frontě ve 3. čítači (vratném) a z něj jsou odvozeny příznaky „plnosti“ (`FULL`)



Obrázek 3.7: Jednotka zpracování výsledků



Obrázek 3.8: Vstupní fronta



Obrázek 3.9: Schéma fronty

a „prázdnosti“ (`EMPTY`) fronty. Nejsou to jen signály pro vnější použití, ale pracuje se s nimi i uvnitř obvodu a díky nim jsou ošetřovány mezní stavy. Schéma fronty je na obrázku 3.9.

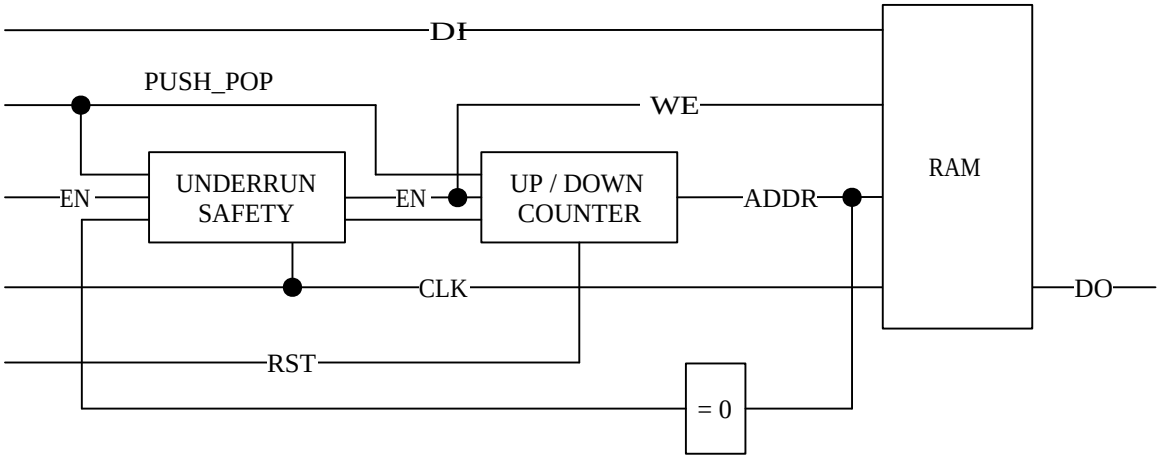
Aktivní pomocný příznak `HALF_FULL` indikuje polovinu (a více) zaplněné vstupní paměti. Umožňuje tak obslužnému programu zjistit přibližný stav fronty a při poklesu pod jednu polovinu rovnou (bez dalších dotazů na stav) odeslat bezpečné množství (tj. minimálně polovinu kapacity) dat do karty. Významně se tak omezuje doba spotřebovaná na „pooling“, kdy se obslužný program opakovaně dotazuje na stav fronty, protože dotazy se mohou provádět v delších intervalech. V této oblasti se přímo nabízí potenciální vylepšení funkčnosti, a to použití přerušovacího systému PC: karta sama při poklesu zaplnění fronty pod polovinu vyšle požadavek o další data.

Výhodné by bylo mít frontu řízenou dvěma časovými doménami – dva hodinové signály. Jeden pro zapisovací port a druhý pro čtecí. Bylo by tak možné nezávisle zapisovat (a číst) data různými rychlostmi. Např. při trvalém odebírání dat z fronty, ověřit na straně obslužného počítače její stav a případně ji rychle doplnit novými daty. Bohužel toto řešení klade velký důraz na vzájemnou synchroznizaci obou domén, a to v řídící části, která se stará o příznaky „plnosti“ nebo „prázdnosti“.

Nabízí se ale snadnější řešení zvýšení rychlosti zápisu. Protože na systémové sběrnici (po níž je akcelerační karta spojena s počítačem) se přenáší data obvykle v šířkách 32 nebo 64 bitů (PCI sběrnice) a zpracovávané znaky mají šířku několikanásobně menší, je možné do fronty zapisovat v plné sběrnicové šířce několik znaků naráz (vhodně uspořádaných, aby se omezilo předzpracování na PC) a mezi frontu a systolické pole zařadit jednotku, která je bude postupně rozdělovat.

3.5.7 Zásobníky dílčích výsledků

Jsou určeny k uchování průběžných hodnot „směru“ během postupu výpočtu. Řízení zásobníku je na rozdíl od vstupní a výstupní fronty navrženo pro ovládání z jednoho místa, tzn. uvnitř výpočetního elementu. V modelu má každá buňka svůj vlastní zásobník a zajišťuje jeho plnění i vyprazdňování. Proto pro implementaci postačí jednoportová bloková paměť. Schéma na obrázku 3.10. Adresu pro čtení a zápis obsluhuje vnitřní vratný čítač. Ten



Obrázek 3.10: Schéma zásobníku

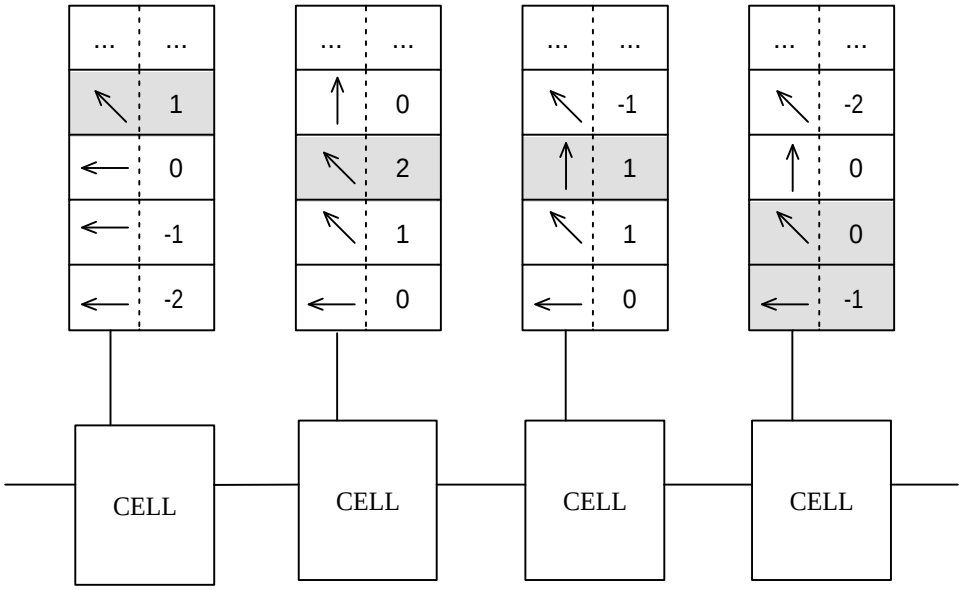
v případě zápisu ukazuje na aktuální volné paměťové místo a v následujícím cyklu dojde k zapsání hodnoty na něj. Zásobník nemá ochranu proti přetečení a zapisování tak může probíhat cirkulárně (k tomu dojde když je zpracovávaný databázový řetězec příliš dlouhý)

Cílové FPGA čipy bohužel nedisponují takovým množstvím blokových pamětí (z jakých jsou synchronní zásobníky postaveny), bylo od modelu 1:1 upuštěno a zásobníky musejí být sdílené. Sdílení zásobníku více buňkami je možné, protože buňky produkují pouze dvoubitovou informaci a datová šířka blokových pamětí může být až 36 bitů. Implementovat velké bloky paměti pomocí distribuované RAM není řešením. Jednak je jí relativně málo, a navíc je tato distribuovaná paměť potřeba pro jednotky porovnávání znaků.

Řízení všech spojených zásobníků je globální a při výpočtu se do nich ukládají souběžně data všech buněk. Ukládají se tak informace o zarovnání všech řetězců aktuálně procházejících polem. Společné řízení taky znamená souběžné čtení všech uložených zarovnání – číst a zpracovávat ho je však komplikované (avšak na této architektuře možné), že bylo stanoveno omezení jen jednoho ukládaného a načítaného zarovnání v čase. Každý nový řetězec vstupující do pole předem „vyresetuje“ zásobník a pak do něj ukládá své hodnoty. Pokud máme po skončení „skórovací“ části algoritmu zájem o načtení zarovnání, nesmí být do pole vyslán už žádný další řetězec, dokud není zarovnání přečteno. Tím se v tomto případě připravíme o možnost řetězení výpočtu, který jinak v první, skórovací, půli může probíhat. Ilustrace ukládání na zásobník pro oba přístupy (zásobník samostatný a sdílený) je vidět na obrázcích 3.11 a 3.12. Vztahuje se k příkladu uvedenému na 3.5.

Jednodušší na provedení než nastíněná komplikovaná verze by bylo stanovit, že zarovnání mohou být ve frontách ukládána „za sebe“. Což znamená, že v jednom okamžiku by sice mohl být v poli zpracováván pouze jeden řetězec, u něhož máme zájem o celé zarovnání, avšak výpočet skóre ostatních by mohl nerušeně probíhat zřetězeným způsobem. Zarovnání by se potom pro jednotlivé řetězce načítalo ve zpětném pořadí. Sekvencer by si mezitím musel pamatovat pozici (a případně hodnotu) maxima všech řetězců čekajících na načtení zarovnání.

Čtení zarovnání blokuje celou kapacitu pole, protože se na transportu dat ze zásobníků na konec pole podílí jednotky pro zpracování výsledků (kterou má každý element) a ty posílají zarovnání po stejných vodičích jako dílčí maxima a indexy. Tomu by se šlo vyhnout vytvořením druhé (čtecí) brány pro zásobníky a na ně by byla napojena extra lo-



Obrázek 3.11: Ukládání zarovnání do samostatných zásobníku

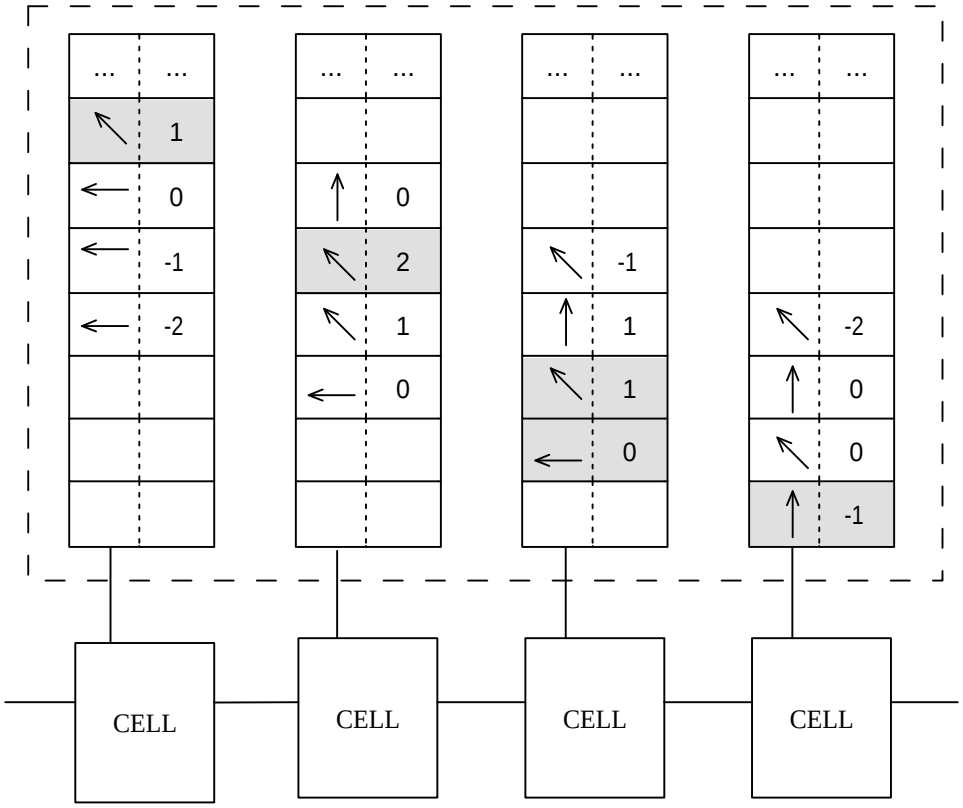
gika starající se jen o načítání zarovnání. Realizováno by to bylo pravděpodobně velmi širokým multiplexerem, nebo jejich stromem a na dlouhém poli by to spotřebovalo značné systémové prostředky. Zřetězeným načítáním zarovnání se zpětný průchod sice zpomalí, ale předpokládá se, že tato úloha není oproti hledání skóre a maxima tak moc využívána a ani se neprovádí pro tak dlouhé sekvence. Na druhou stranu se znovuvyužívá již implementovaná struktura.

3.5.8 Systolické pole

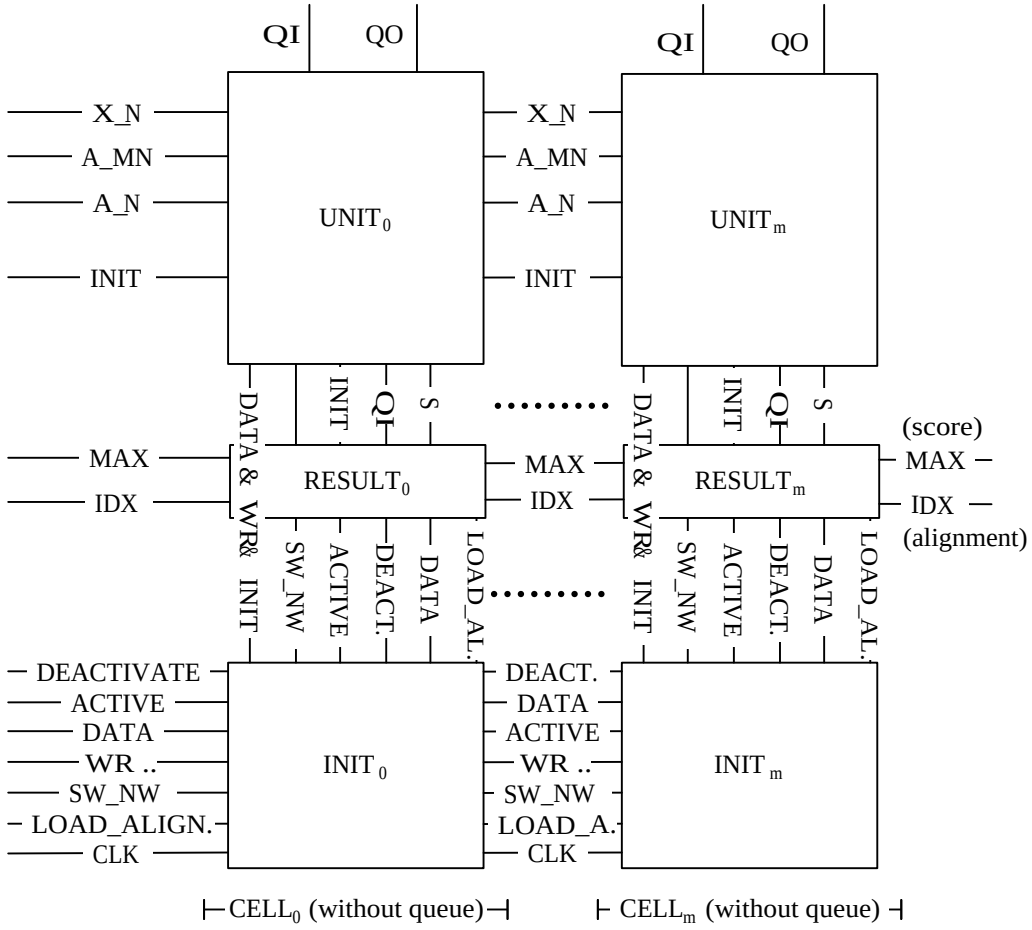
Ze schématického obrázku 3.13 je vidět, že celé systolické pole je složeno s vedle sebe pospojovaných výpočetních buněk. Každá buňka má následující 4 hlavní součásti:

- Výpočetní část
- Jednotka pro zpracování výsledků
- Inicializační a řídící jednotka
- Zásobník

Celé pole je navrženo tak, aby co nejvíce datových přenosů bylo realizováno pouze mezi sousedními buňkami. K těmto přenosům byla přidána jedna společná sběrnice `DATA`, která je určena jako datový vstup pro nastavitelné registry, a pokud není právě používána přenáší se po ní fáze výpočtu.



Obrázek 3.12: Ukládání zarovnání do sdíleného zásobníku



Obrázek 3.13: Lineární pole buněk

Inicializační a řídící jednotka představuje prakticky jen uzel, kde jsou napojeny společné řídící vodiče vedoucí do ostatních jednotek. Výjimkou je signál `ACTIVE`, který odlišuje „aktivní“ od „přeposílací“ buňky a tvoří jej posloupnost registrů.

3.5.9 Inicializační součásti

Tato část popisuje, jak je systolické pole inicializováno a připravováno na výpočet. Inicializace zahrnuje více procedur s různou složitostí. Některá nastavení se mezi jednotlivými úlohami mohou měnit často, některá vydrží po celou dobu neměnná. U každé procedury je uveden postup a její časová složitost. Hodnota `T` představuje dobu periody hlavního hodinového signálu.

1. Nastavení **konstant pro penalizace mazání a vkládání znaku** `C_M` a `C_N`. Velmi jednoduchý postup, nastaví se požadovaná hodnota na stejnojmennou bránu `C_M` a potvrdí se signálem `SET_C_N` aktivním po dobu `2T`. To samé pro druhou konstantu. V obvodu dojde k zápisu do registrů v každé buňce skrze společnou sběrnici.

2. **Dotazovací sekvence** se načítá ze vstupní fronty, a před započetím procedury již musí být zapsány a to v pořadí „koncem napřed“. Pro náš průvodní příklad vstupuje

dotazovací řetězec CGCC do fronty v pořadí nejprve C, C, G a nakonec C. Dále se musí nastavit odpovídající hodnota QUERY_LENGTH podle délky sekvence. Poté už se může aktivovat signál SET_QUERY na dobu nejméně (QUERY_LENGTH+2)T. Znaky se z fronty načtou do pole a v době, kdy jsou na správném místě se zapíší do X_M registrů v buňkách. Spolu s daty ještě polem putuje ACTIVE signál, kterým se nastaví prvních QUERY_LENGTH buněk jako aktivních a ostatní jako „přeposílače". Celé pole projde za dobu (ELEMENTS+2)T, ale není nutné čekat až se dostane na konec.

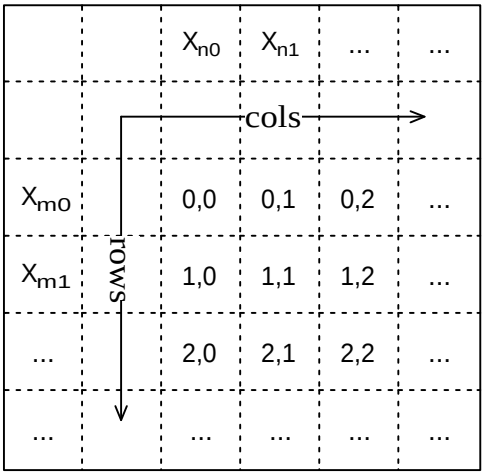
3. Hodnota **registrů pro první krok** výpočtu. Než se použije tento proces, musí být nastaven registr C_M a taky brána QUERY_LENGTH. Tyto paměťové prvky jsou označeny jako A_M a jejich hodnota je při výpočtu načtena každou buňkou pouze jednou – na začátku. Nastavením SET_GAP_PENALTY po dobu alespoň (N+1)T, obvod od 0 přičítá C_M a číslo zapisuje přes společnou sběrnici do buněk. Jelikož je při „probuzení" A_M registrů použit signál INIT postupující skrze pole, nesmí být v době (ELEMENTS+1)T spuštěn žádný nový výpočet. Je to z důvodu, že na tento signál reaguje jednotka pro ukládání finálního výsledku a uložila by tak do fronty neplatný výsledek.

4. „Oznámení" o použití **metody** Needleman-Wunsch či Smith-Waterman se provede pouhým nastavením nebo vynulováním bitu SW_NW. Po jednom hodinovém cyklu se změna projeví v celém obvodu.

5. **Penalizace pro porovnání znaků**, neboli nastavení porovnávací matice. Je to nejzdlouhavější procedura kvůli velikosti matice. Přístup k matici pro zápis je pro každé políčko zvlášť, ale adresa políčka se k matici musí dostat stejným způsobem jako znaky při výpočtu, tzn. sekvenčně. Zápis do matice se děje pro QUERY_LENGTH buněk zároveň. Do fronty se nahraje adresa (znak, který představuje adresu), a poté se využije existující procedury SET_QUERY s nastaveným doplňkovým bitem PENALTY_WRITE_MODE. Ten způsobí, že z fronty je dokola čtena pouze jediná hodnota adresy a ta se zapíše do X_M registrů buněk jako první část adresy. Následujícím krokem je nahrání druhé části adresy do fronty (fronta ovšem může být předem zásobena) a spuštění procesu SET_PENALTY s předem definovanou hodnotou penalizace na bráně PENALTY. Druhá adresa prochází polem po stejných vodičích a ve vhodném okamžiku dojde k zapsání hodnoty ze sběrnice do porovnávací matice. Časová náročnost obou kroků je 2*(N+2)T. Výhodou ovšem je, že pokud hodláme zapsat do matice celý sloupec, provádí se dále už jen druhá polovina postupu. Časová spotřeba zápisu sloupce do matice je ((N+2)+ROWS*(N+2))T. A nakonec pro celou matici COLS*((N+2)+ROWS*(N+2))T. Pozn.: po modifikaci penalizačních hodnot se musí opět provést nastavení dotazovacího řetězce, neboť tento byl předchozím postupem přepsán.

3.5.10 Řízení výpočtu

Na správně načasovaný průběh výpočtu „dohlíží" sekvencer. Je ovládán vstupním signálem RUN. Po správně provedené inicializaci a nastavení parametrů pro úlohu se naplní vstupní fronta znaky „databázového řetězce" a ohlásí se jeho délka na bránu DATABASE_LENGTH. Obvod po obdržení spouštěcího impulzu zkopíruje do „prováděcí" fronty délku řetězce, vynuluje zásobníky a začíná načítat znaky z fronty do pole. Zásobníky jsou implicitně zapnuté pro zápis zarovnání. Výpočet jednoho řetězce trvá (QUERY_LENGTH+DATABASE_LENGTH+1)T a po minimálně tuto dobu musí být aktivní bit RUN. Po uplynutí této doby se může zpracovat další sekvence (pokud ovšem nepožadujeme z právě prováděného řetězce zarovnání) a využít tak výhody zřetězení výpočtu.



Obrázek 3.14: Souřadnice poskytované jako výsledek Smith-Waterman algoritmem

Prováděcí frontu čte na konci pole jednotka pro zpracování finálního výsledku a podle hodnoty si potom odpočítá, kdy přijde výsledek. Nejdříve však čeká na `INIT` pulz, kterým je zahájen každý řetězec. Dále se postupuje podle metody. Při jednodušším Needleman-Wunsch algoritmu se pouze ve správný čas zkopíruje do výstupní fronty skóre (pravý dolní roh tabulky). Pokud se počítá podle Smith-Waterman algoritmu, musí se z hodnot vystupujících z pole určit ta maximální a k ní přiřadit souřadnice odpovídající souřadnice. Tato trojice je opět zkopírována do fronty. Sémantika poskytovaného výsledku je vidět na obrázku 3.14.

Od doby zahájení výpočtu se bude po `(ELEMENTS+DATABASE_LENGTH+3)T` cyklech nacházet skóre ve výstupní frontě. Pokud nebyl mezitím zahájen jiný řetězec, je právě teď možnost získat zarovnání. Řídící signál `GET_ALIGNMENT` spustí proceduru.

Všechny buňky se nastaví do režimu „přeposílačů“, Čtení ze zásobníků probíhá tak, že každá buňka si z něj načte jednu hodnotu do svého výstupního registru v dalších cyklech se data posouvají směrem k jednotce pro zpracování finálního výsledku umístěné na konci pole.

Úvodním krokem je získání konce zarovnání. V metodě Needleman-Wunsch se nachází v pravém dolním rohu a načte se proto po fixním počtu kroků. Algoritmus Smith-Waterman může mít maximum umístěno kdekoliv v tabulce. Z předchozích kroků máme uloženy souřadnice a podle nich si koncovou hodnotu načteme. Pokud není umístěna na vrcholu zásobníku, vybírají se řádky naprázdno a hodnoty se „zahazují“.

Dále se už postupuje podle načteného údaje. Nabývá 4 hodnot – 3 směry a jedna nulová hodnota. Pokud je načtena nula, zpětný průchod končí. To samé platí i pro situaci, kdy se dostaneme na okraj virtuální tabulky se zarovnáním. Na obrázku 3.12 je stínováním ukázáno zarovnání a je z ní poznat, jak dále probíhá načítání podle vyznačených směrů. 2 bitové směry se stejně jako maxima ukládají do výstupní fronty (každá ze 4 součástí výsledku má přiřazenu určitou část bitové šířky fronty).

Kapitola 4

Výsledky

4.1 Syntéza

Syntéza do existujícího FPGA obvodu proběhla pro 3 vybrané čipy. Třetí z nich byl vybrán zejména kvůli výkonnostnímu srovnání s [3], jelikož má k zde implementovanému designu nejblíže (a samozřejmě používá tento čip).

- Spartan3 XC3S200 (součást školní karty COMBO6)
- Spartan3 XC3S2000 (model s více programovatelnými hradly a větší pamětí)
- Virtex-II Pro XC2VP30 (typ s moderními prvky umožňující např. i umístění procesorů PowerPC na čipu, umožňuje dosáhnout vyšší frekvence než řada Spartan3)

Jejich základní parametry jsou uvedeny v tabulce 4.1.

Samotná syntéza probíhala v softwaru Precision RTL Synthesis 2005 of firmy Mentor Graphics. Jako omezení (constraint) byla nastavena frekvence - 50 Mhz pro čipy Spartan3, 100 Mhz pak pro Virtex-II. Pro nastavené parametry byla spuštěna vždy vícekrát, aby se zjistil maximální počet výpočetních elementů možných umístit na čip.

Globální generické parametry implementovaného designu ovlivňující syntézu jsou:

- Datová šířka znaků zpracovávaných sekvencí: 2 a 5 bitů
- Datová šířka ostatních dat (skóre, penalizace, indexy): 6 – 16 bitů
- Počet výpočetních elementů - proměnný
- Kapacita zásobníků (závisí také na jejich šířce): při syntéze zůstávala fixní – 2^8 slov
- Datová šířka zásobníků (čím vyšší, tím více elementů sdílí zásobník): pohybovala se v rozmezí 2 – 32 bitů
- Kapacita vstupní a výstupní fronty: při syntéze fixně 2^8 slov

Výkonnost architektury je v tabulkách uváděna v násobcích jednotky `CUPS` (Cell Updates Per Second). Udává kolikrát za jednotku času je obvod schopen změnit stav výpočetních buněk. Je jednou z typických vlastností systolických struktur.

$$\text{výkonnost} = ELEMENTS \cdot f_{max} \quad [CUPS] \tag{4.1}$$

Type	Slices	Block RAM [kb]	Distributed RAM [kb]
XC3S200	1920	216	30
XC3S2000	20480	720	320
XC2VP30	13696	2448	428

Tabulka 4.1: Přehled základních parametrů vybraných FPGA obvodů

typ sekvence	datová šířka	počet elementů	max. frekvence [MHz]	výkonnost [GCUPS]
DNA	6b	32	48,981	1,567
	8b	26	47,787	1,242
protein	6b	6	48,112	0,289
	8b	4	47,337	0.189

Tabulka 4.2: Syntéza pro čip Spartan XC3S200

typ sekvence	datová šířka	počet elementů	max. frekvence [MHz]	výkonnost [GCUPS]
DNA	8b	296	44,020	13,029
	10b	256	42,255	10,817
	12b	212	40,719	8,632
protein	8b	52	37,022	1,925
	10b	42	36,061	1,514
	12b	36	34,988	1,260
	16b	26	33,783	0,878

Tabulka 4.3: Syntéza pro čip Spartan XC3S2000

typ sekvence	datová šířka	počet elementů	max. frekvence [MHz]	výkonnost [GCUPS]
DNA	8b	196	89,622	17,566
	10b	154	83,640	12,881
	12b	140	82,433	11,541
	16b	108	77,991	8,423
protein	8b	38	79,542	3,023
	10b	30	76,664	2,300
	12b	26	73,828	1,920
	16b	18	70,244	1,264

Tabulka 4.4: Syntéza pro čip Virtex XC2VP30

Vysyntetizované parametry byly porovnány s řešením, které uvádí [3]. Tato práce byla zvolena protože má k zde implementovanému designu nejblíže svou obecnou funkčností. Citovaná práce uvádí, že na stejném FPGA čipu Virtex-II Pro bylo dosaženo v DNA porovnávání výkonnosti 7,5 GCUPS (pro porovnávání znaků podle „RAM table“) a pro proteinové sekvence 2,23 GCUPS (tentokrát podle „Fixed table“). Není však uvedena datová šířka. Přesto, uváděné výkonnosti obou řešení jsou si velmi blízké.

4.2 Možná vylepšení a další vývoj

Zde je uveden seznam cílů na které je vhodné se zaměřit při dalším zlepšování architektury. Vyplývá ze zjištění ve fázi implementace a testování, že některé části byly ne příliš dokonale navrženy, nebo se nepodařilo dodržet časový rozvrh.

- Upravit výpočetní buňku tak, aby uměla vyhodnocovat vstupní data s využitím vylepšené mezerové penalizace. S tím souvisí i změny v inicializační části. Tato funkce buněk je velmi důležitá (a využívaná) hlavně pro Smith-Waterman algoritmus. Bohužel prvky, jež je nutné přidat pro novou funkčnost zasahují do „kritické cesty“ a dojde tím k určitému zpomalení výpočtu.

- V jednotce pro zpracování znaků umožnit i jiné způsoby ohodnocení: např. zadání uživatelské funkce, metoda nazývaná „IUPAC Wildcards“, slučování skupin znaků a přiřazení jedné penalizace celé skupině. Návrhy z tohoto bodu by mohly pomoci zbavit se rozsáhlých matic a tím pádem uvolnit systémové prostředky pro více výpočetních jednotek.

- Za vstupní frontu je dobré zařadit jednotku, která z ní vybírá slova o šířce datové sběrnice a rozděluje je na znaky (ty mají datovou šířku několikrát menší). Znaky se sekvenčně posílají do systolického pole. S frontou je spjata i chybějící funkce, která by zajistila pauzu v při prázdné vstupní frontě při běhu algoritmu.

- Zdokonalení jednotky pro řízení výpočtu, aby byla schopná ukládat a posléze načítat zarovnání pro více řetězců. Také by měla poznat že předchozí řetězec byl zpracován a sama zahájit další (dávkové zpravování).

- Celkové zrychlení zvýšení maximální operační frekvence např. pomocí zřetězení výpočtu v rámci buněk. Další cestou k vyšším výkonům by mohlo být vícevláknové zpracování, nebo i zmíněné spekulativní dopředné výpočty.

- Velký potenciál taktéž skýtá dynamická rekonfigurace, avšak jejím negativem je obtížný návrh.

Kapitola 5

Závěr

Byly nastudovány dostupné materiály zabývající se problémem porovnávání DNA a proteinových řetězců. Podle zadání jsem se seznámil nejen se současnými trendy, ale i historickými pokusy o řešení zadaného problému.

Na základě těchto znalostí byla navržena generická hardwarová architektura založená na systolické struktuře provádějící zpracování sekvencí. Podle návrhu implementovaná architektura byla úspěšně simulována a umí k výpočtu použít oba doporučené algoritmy Smith-Waterman i Needleman-Wunsch.

Výsledné řešení je velice flexibilní co do schopnosti zpracovávat různé úlohy, ale v rychlosti zaostává za specializovanými obvody. Přesto že jde o obecnou architekturu, poskytuje zajímavé výkonnostní výsledky.

Literatura

[1] Mauricio Ayala-Rincón, Ricardo P. Jacobi, Luis G. A. Carvalho, Carlos H. Llanos, and Reiner W. Hartenstein. Modeling and prototyping dynamically reconfigurable systems for efficient computation of dynamic programming methods by rewriting-logic. In *SBCCI '04: Proceedings of the 17th symposium on Integrated circuits and system design*, pages 248–253, New York, NY, USA, 2004. ACM Press.

[2] Hans-Martin Blüthgen and Tobias Noll. A Programmable Processor for Approximate String Matching with High Throughput Rate. In *ASAP '00: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, page 309. IEEE Computer Society, 2000.

[3] Tom Van Court and Martin C. Herbordt. Families of FPGA-Based Algorithms for Approximate String Matching. In *ASAP '04: Proceedings of the Application-Specific Systems, Architectures and Processors, 15th IEEE International Conference on (ASAP'04)*, pages 354–364. IEEE Computer Society, 2004.

[4] Steven A. Guccione and Eric Keller. Gene Matching using JBits. In Manfred Glesner, Peter Zipf, and Michel Renovell, editors, *Field-Programmable Logic and Applications*, pages 1168–1171. Springer-Verlag, Berlin, 2002. Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications, FPL 2002. Lecture Notes in Computer Science 2438.

[5] Pascale Guerdoux-Jamet, Dominique Lavenier, Charles Wagner, and Patrice Quinton. Design and Implementation of a Parallel Architecture for Biological Sequence Comparison. In *Euro-Par '96: Proceedings of the Second International Euro-Par Conference on Parallel Processing*, pages 11–24, London, UK, 1996. Springer-Verlag.

[6] Dzung T. Hoang. A Systolic Array for the Sequence Alignment Problem. Technical report, Providence, RI, USA, 1992.

[7] David L. Osterbur. Similarity Searching: BLAST, BLink and DART all around. `http://www.ncbi.nlm.nih.gov/Class/NAWBIS/Modules/Similarity/simsrch1.html`, 2006.

[8] WWW stránky. WikipediA, The Free Encyclopedia. `http://www.wikipedia.org`.

[9] Yoshiki Yamaguchi, Tsutomu Maruyama, and Akihiko Konagaya. High Speed Homology Search with FPGAs. In *Pacific Symposium on Biocomputing*, pages 271–282, 2002.

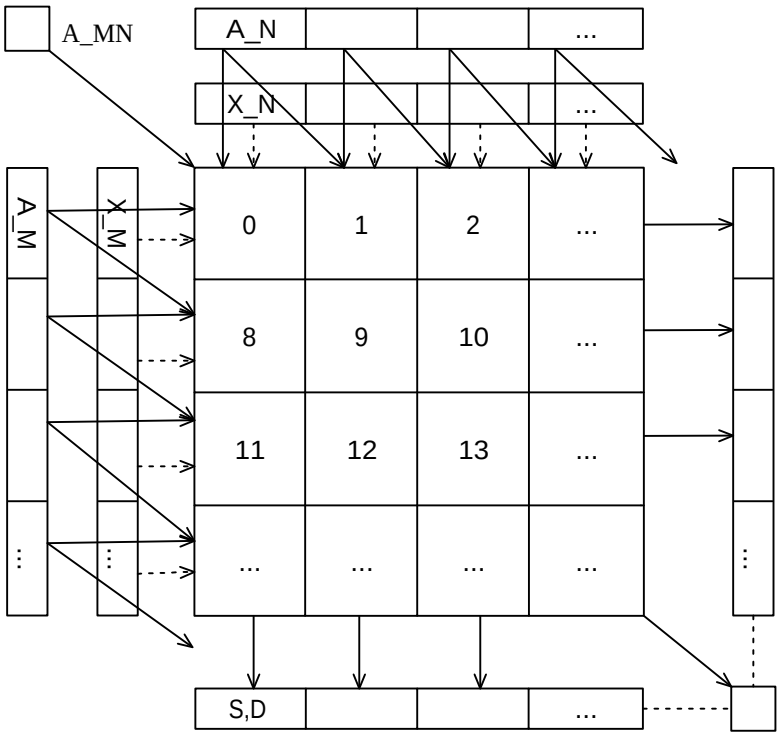
[10] C. W. Yu, K. H. Kwong, Kin-Hong Lee, and Philip Heng Wai Leong. A Smith-Waterman Systolic Cell. 2003. 3-540-40822-3.

Dodatek A

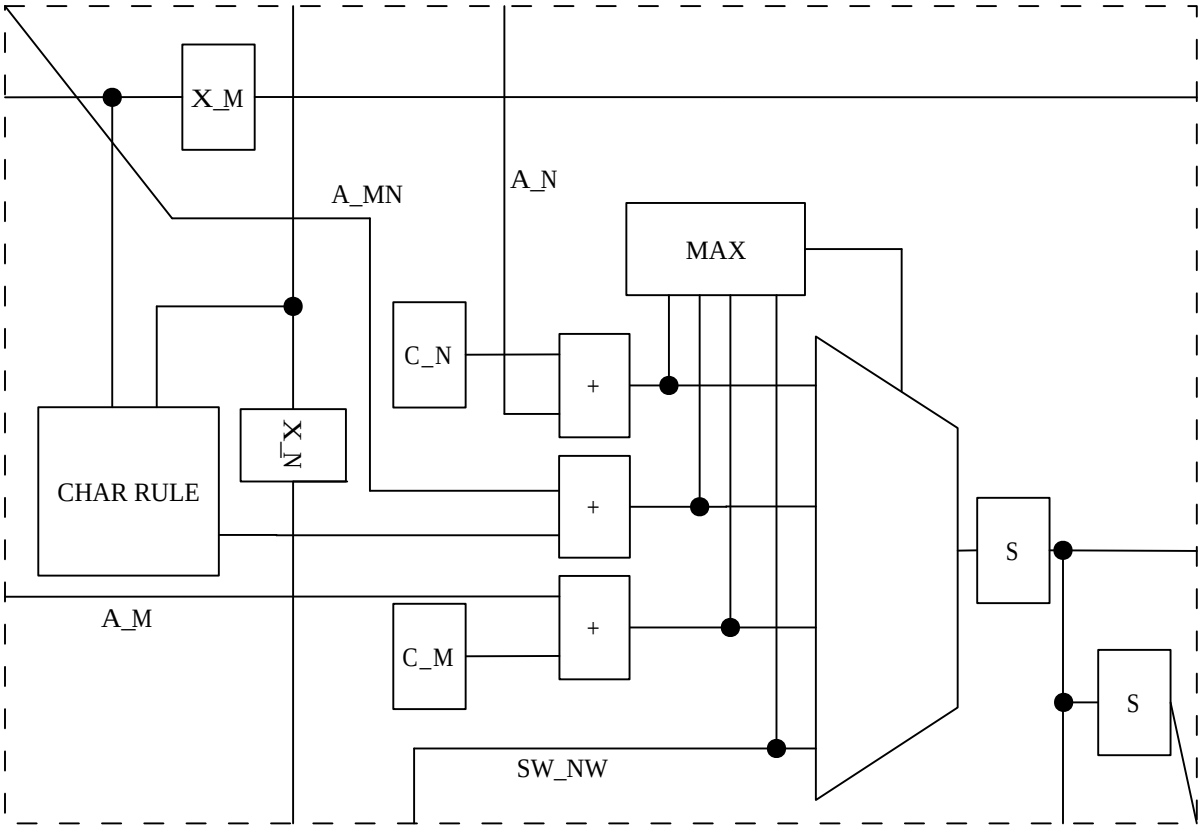
Dvourozměrné systolické pole

Pokusně bylo realizováno funkční obvodové řešení celého dvourozměrného systolického pole. Bylo však zjištěno, že to není příliš vhodná architektura z několika důvodů. Nicméně byla základem pro implementaci lineárního pole a posloužila pro vyzkoušení různých postupů.

- Více vzájemných spojení mezi buňkami — složitější propojení
- Je potřeba mnoho elementů i pro zpracování krátkých sekvencí
- Nevyřešené řetězení úloh



Obrázek A.1: Dvourozměrné pole buněk se svými inicializačními a výstupními obvody



Obrázek A.2: Jednoduchá výpočetní buňka pro dvourozměrné systolické pole