VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

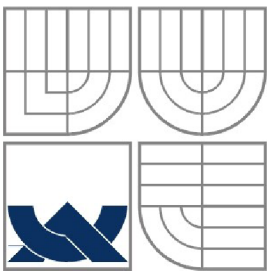
IMPLEMENTACE ALGORITMU PRO HLEDÁNÍ PODOBNOSTÍ DNA ŘETĚZCŮ V FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

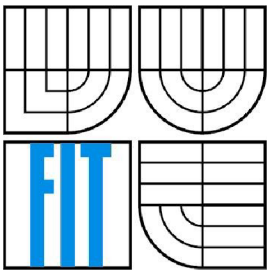
AUTOR PRÁCE
AUTHOR

Bc. Martin Pařenica

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

IMPLEMENTACE ALGORITMU PRO HLEDÁNÍ PODOBNOSTÍ DNA ŘETĚZCŮ V FPGA

APPROXIMATE STRING MATCHING ALGORITHM IMPLEMENTATION IN FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Martin Pařenica
AUTHOR

VEDOUCÍ PRÁCE — Dr. Ing. Otto Fučík
SUPERVISOR

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Pařenica Martin, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Implementace algoritmu pro hledání podobností DNA řetězců v FPGA**
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s problematikou hledání podobností v DNA sekvencích a technologií rekonfigurovatelných hradlových polí FPGA.
2. Prostudujte možnosti implementace vybraných algoritmů v hradlových polích FPGA s využitím desky COMBO6 vyvinuté v rámci projektu Liberouter.
3. Navrhněte a implementujte vybraný algoritmus v FPGA. Pro návrh uvažujte využití jazyka VHDL.
4. Navrhněte vhodný způsob implementace vybraného algoritmu s důrazem na rychlost zpracování.
5. Vybraný algoritmus implementujte a funkčnost demonstrujte na vhodném příkladě.

Literatura:

- http://www.xilinx.com
- http://www.liberouter.org

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění prvních tří bodů 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: **Fučík Otto, Dr. Ing.**, UPSY FIT VUT
Datum zadání: 28. února 2006
Datum odevzdání: 22. května 2007

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

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Bc. Martin Pařenica**
Id studenta: 49511
Bytem: Sušilova 475, 769 01 Holešov
Narozen: 07. 07. 1983, Přílepy
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Implementace algoritmu pro hledání podobností DNA řetězců v FPGA
Vedoucí/školitel VŠKP: Fučík Otto, Dr. Ing.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

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 zveřejněním 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.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Tato práce popisuje způsoby porovnání nukleotidových řetězců s využitím párového a vícenásobného porovnání. V práci jsou popsány algoritmy párového porovnávání pro hledání nad daty v databázích a nebo algoritmy využívající dynamické programování. Dále jsou popsány způsoby vícenásobného porovnání. Mezi základními algoritmy je uvedeno dynamickým programováním a nebo algoritmy, které s využitím určité míry nepřesnosti postupně sestavují porovnání. Teoretickou část práce uzavírá popis technologie FPGA. Další část práce, praktická část, je věnována implementaci jednoho z vícenásobných algoritmů. Závěrečná část shrnuje vlastnosti vybraného algoritmu.

Klíčová slova

DNA, párové porovnání, prohledávání databází, BLAST, FASTA, dynamické programování, Needleman-Wunsch, Smith-Waterman, vícenásobné porovnání, CLUSTAL, fylogenetické stromy, FPGA.

Abstract

This paper describes sequence alignment algorithms of nucleotide sequences. There are described pairwise alignment algorithms using database search or dynamic programming. Then in the paper is description of dynamic programming for multiple sequences and algorithm that builds phylogenetic trees. At the end of the first part of the paper is the description of technology FPGA. In the second part that is more practical is described implemntation of the choosen one algorithm. This part includes also examples of some multiple alignments.

Keywords

DNA, pairwise alignment, database search, BLAST, FASTA, Needleman-Wunsch, Smith-Waterman, multiple alignment, dynamic programming, CLUSTAL, phylogenetic tree, FPGA.

Citace

Martin Pařenica: Implementace algoritmu pro hledání podobností DNA řetězců v FPGA, diplomová práce, Brno, FIT VUT v Brně, 2007

Implementace algoritmu pro hledání podobností DNA řetězců v FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Dr. Ing. Otto Fučíka.
Další informace mi poskytl Ing. Tomáš Martínek.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Pařenica
15. května 2007

Poděkování

Děkuji svému vedoucímu Dr. Ing. Ottu Fučíkovi za cenné rady a odborné vedení při vývoji porovnávacího systému. Dále děkuji Ing. Tomáši Martínkovi za konstruktivní připomínky k implementaci algoritmu.

Obsah

1 Úvod

1.1 Motivace

Porovnávaní lidských genetických vzorů řeší také oblast genetické daktyloskopie, která porovnává jen některé části DNA kódu. V této oblasti má porovnání řetězců DNA své prvopočátky. Porovnávané části řetězců DNA jsou sekvence, kterými jsou jednotliví jedinci od sebe odlišní a na základě těchto sekvencí jsou lidé jednoznačně identifikovatelní. V daktyloskopii je pro identifikaci využíváno 13 sekvencí kódu DNA. Tyto sekvence tvoří specifický popis jedince, tzv. DNA fingerprint. Identifikace na základě DNA se v kriminalistice s úspěchem využívá delší dobu. Díky této metodě bylo ozřejměno a dokázáno již mnoho trestných činů a usvědčena celá řada pachatelů. Prvním a tudíž velmi známým případem, je případ z roku 1986. V tomto roce byla zpětně využita Jeffersova metoda k objasnění případu vraždy 15 leté Lindy Mannové z listopadu 1983. Tento případ zůstal neobjasněn až roku 1986, kdy byla nalezena další zavražděná 15 letá Dawn Ashorthová. Obě dívky byly znásilněny a poté zavražděny podobným způsobem. Podezření padlo na totožného pachatele, který byl následně usvědčen na základě porovnání DNA získaného z jeho krve a DNA získaného z genitálií zavražděných dívek [2, 10].

Znakem poslední doby je značný pokrok v bioinformatice a s tím související růst množství biologických dat, zejména pak genetických, které je potřeba zpracovávat. Probíhá velké množství testů a experimentů s cílem zjištění významu jednotlivých kódů zakódovaných v geneticky významné molekule deoxyribonukleové kyseliny (zkráceně tzv. DNA). Analýza DNA sekvencí se provádí v mnoha vědeckých oborech. Možností využití porovnávání v této oblasti je nespočetně: hledání léků, dispozice nositelů určitých typů sekvencí DNA k určitým nemocem, a to jak vrozeným tak získatelným v průběhu života jedince, hledání dědičných chorob atd.

Analýza dat je v počítačích, a to i v těch nejvýkonnějších velice zdlouhavá. Je proto snaha tyto analýzy paralelizovat, a tím výpočetní dobu zkrátit. Ani rozložením výpočtu mezi velké množství počítačů se nedosáhne podobné výpočetní výkonnosti jako při použití speciálního obvodu. Výhoda specializovaného obvodu spočívá v tom, že se porovnání v hardwarovém řešení provádí paralelně. Náročnost tohoto porovnání je v hardwaru lineární, ve srovnání s řešením v softwaru, pro který je náročnost výpočtu kvadratická, a tedy časově mnohem náročnější. Pokud je použito řešení například s FPGA čipem, je možné získat vysoký výpočetní výkon, paralelnost, nízkou spotřebu, relativní flexibilitu a také úsporu prostoru.

Porovnání dvou řetězců lze realizovat několika způsoby. Nejjednodušším z nich je porovnání jednotlivých nukleotidů 1:1. Je to metoda, která je nejjednodušší, časově i paměťově nejméně náročná, avšak také nejméně přesná. V důsledku toho, že je v přírodě vlivem mutací a evoluce

vysoká míra chybovosti v genetickém kódu, je tato jednoduchá metoda porovnání řetězců nevhodná. Opačných kvalit než výše jmenovaný algoritmus mají metody, které při porovnávání přiřazují srovnání určité skóre značící míru shody řetězců. Toto skóre je vypočítáváno na základě přičítávání penalt za vložení mezer, mazání znaků a nebo nahrazení nukleotidu. Tyto metody jsou velice časově a paměťově náročné. Podrobnější popis těchto metod je v kapitole č. 2. Porovnávání se řeší často po dvou řetězcích DNA, ale existují i varianty porovnávání více sekvencí najednou. Jak bylo zmíněno výše, je pro tyto úlohy vhodný výpočetní potenciál rekonfigurovatelných obvodů.

V této práci budou dále popsány možnosti vícenásobného porovnávání, základní algoritmy používané v této oblasti s rozborem jejich výsledků a náročnosti na výpočetní systém. Dále bude popsán způsob realizace algoritmu vícenásobného porovnání s využitím metody UPGMA kombinované s algoritmem Needleman-Wunch. Tato upravená metoda se blíží některými vlastnostmi algoritmu clustal. V hradlovém poli FPGA bude následně tento algoritmus implementován.

1.2 Cíl práce

Cílem této práce je popsat algoritmy pro porovnávání nukleotidových řetězců a porovnat výkonnost a kvalitu jednotlivých zástupců algoritmů. Hlavním cílem této práce je návrh a implementace algoritmu pro vícenásobné porovnávání řetězců DNA. Při návrhu byl kladen důraz na rychlost výpočtu a možnost snadné úpravy parametrů algoritmu pro porovnávání libovolných typů řetězců, jako například DNA, proteinů a nebo obyčejného textu.

1.3 Struktura práce

Práce je rozdělena do kapitol, podkapitol a článků. Následující kapitola je zaměřena na technologie porovnávání řetězců, a to jak dvojic tak vícenásobných sekvencí. Následující kapitola srovnává výkonnosti implementací. Kapitola č. 4 popisuje technologii FPGA. Následující kapitola popisuje nejhodnotnější část této práce a sice návrh a implementaci vybraného algoritmu vícenásobného porovnání. Dále 6. kapitola demonstruje funkci algoritmu na některých vybraných řetězcích. Poslední kapitolou je závěrečné shrnutí této práce.

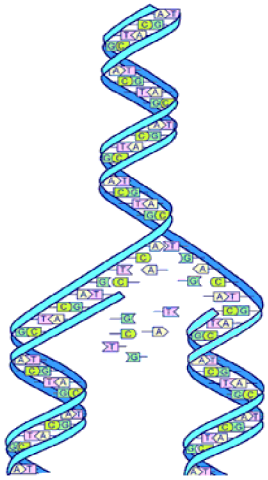
2 Porovnání řetězců

2.1 Základní pojmy

2.1.1 DNA

Molekula DNA (Deoxyribonukleová kyselina) je specifická svými jedinečnými vlastnostmi. Jedná se o molekulu, která nese veškerou genetickou informaci o každém organismu. Je pro každý živý organismus specifická a přítomná v celém těle jedince (rostliny), ve všech jeho buňkách. Její životnost je i několik staletí. Molekula se skládá ze dvou polynukleotidových řetězců, které jsou pravotočivě zatočeny do takzvané šroubovice (obrázek č. 1). Řetězec je složen z nukleotidů, které odlišují svým pořadím genetický význam.

Nukleotidy jsou 4: adenin (A), thymin (T), cytosin (C) a guanin (G). Nukleotidy tvoří komplementární páry adenin – thymin a cytosin – guanin. Souvislý úsek molekuly DNA tvoří gen. Soubor genů tvoří genom. Lidský genom představuje asi 30.000 genů, což odpovídá 3 miliardám bází.



Obr. 1: DNA
(převzato z [33])

DNA má vysokou rozlišovací schopnost. Lze ji tedy použít nejen k rozlišení živočišného druhu, ale také k identifikaci jednotlivých jedinců. DNA je makromolekula s velkým obsahem informací. Pravděpodobnost 2 stejných osob se stejnou DNA je přibližně $1{:}10^{18}$ [19]. K identifikaci

konkrétního jedince stačí jen jedna neporušená buňka, která může být odebrána i dlouhou dobu po jeho smrti.

Přestože je DNA k identifikaci/verifikaci vhodná, bylo by nejvhodnější vytvořit databázi osob, a poté využívat záznamů v databázi. Vysoká rozlišovací schopnost je kompenzována ne zrovna vysokou ochotou lidí dobrovolně odevzdávat své DNA. Neustále totiž hrozí vysoká možnost zneužití genetického kódu. Genetický kód v sobě nese nejen jednoznačnou identifikaci osoby, ale i popis charakterových vlastností, zdravotních informací čítajících možnosti vrozených vad a nemocí a tak podobně. Nevýhoda identifikace na základě DNA spočívá také v tom, že se nejedná o masovou biometrii. To znamená, že ji nelze využít například k identifikaci osob procházejících nebo vstupujících do budovy. Identifikaci na základě DNA lze provést jen u omezeného počtu osob v delším časovém horizontu (s využitím současných technologií).

2.1.2 Sekvencování DNA

Jedná se o stanovení primární struktury DNA. Prvním krokem je získání dostatečného počtu DNA z malého množství vzorků, jedná se tedy o rozmnožení, lépe řečeno kopírování DNA. Dále je nutné použít fluorescenční značky, a poté s využitím laserového detektoru napojeného na počítač během elektroforézy vyhodnotit data [23]. Každá odražená vlnová délka značí jiný nukleotid. Tímto postupem se struktura DNA přečte, a poté zapíše do sekvenční podoby.

2.1.3 Databáze DNA

Pro lékařské a vědecké účely existují databáze, v nichž je často použit porovnávací systém s algoritmy typu BLAST nebo FASTA. Tyto databáze jsou vhodné pro vyhledávání podobností v určitých sekvencích a nebo obráceně a pro vědce jsou zároveň cenným zdrojem sekvencí biologických dat. Jednou z nejznámější databází je EMBL, která je spravována společností stejného jména.

Z pohledu kriminalisty, by měla existovat databáze se záznamy všech osob a jejich DNA. S takovým bezpečnostním přístupem by se všechny trestné činy daly snadno vyřešit. Pokud by se na místě činu vyskytla stopa, na níž je nalezeno DNA (jako například vlas, krev, sliny, atd.) lze takové stopy DNA uchovat pro další analýzu. DNA se poté pomocí sekvencování přečte a v databázi je poté možné spustit porovnávání.

Z pohledu ochránců lidských práv a svobod by ovšem bylo nejlepší takovou databázi vůbec nevytvářet. Existují totiž opodstatněné obavy týkající se zneužívání dat v databázi. DNA je řetězec, který o svém nositeli prozradí mnoho osobních informací. I když se pro kriminalistické účely používají části DNA, které nekódují genetické vlastnosti osob, existuje přesto obava z jejich zneužití.

Při snímání se ukládá celá molekula DNA, v důsledku čehož by tedy zneužití nebylo obtížné [2]. Z DNA lze například zjistit možné dědičné nemoci, náchylnost na zákeřné nemoci, ale i potenciální projevy chování osoby. Osobní informace nelze využívat bez zajištění jejich bezpečnosti proti zneužití.

Existují však i možnosti využití DNA a identifikace bez omezení soukromí nevinných osob. Je jím využití syntetického DNA, které by bylo možné využít pro značkování zvláště cenného zboží jako například aut, drahokamů, uměleckých děl a podobně [17]. Nebezpečí narušení soukromí (genetickým popisem a uchováním dat v centrální databázi) se v těchto oblastech naprosto vylučuje a výhod využití je několik. DNA je dostatečně mikroskopická, aby nerušila a nebyla viditelná pouhým okem. A dále, hlavní výhodou DNA je její vysoká rozlišitelnost, takže pro identifikace/verifikaci předmětů by bylo DNA výhodnou značkou. Bez obavy z narušení soukromí by se mohly vytvořit databáze objektů, v kterých by se mohlo vyhledávat. Například pomocí algoritmů zmíněných v této práci.

2.1.4 Globální vs. lokální porovnání

Globální porovnání je vhodné pro řetězce o stejné nebo podobné délce. Lokální porovnání se používá pro rozdílné délky řetězců, kdy požadavky úvodní a koncové mezery nehrají roli [7].

Příklad (převzato z [14]):

Mějme řetězce T a S, T:FTFTALILLAVAV S:FTALLLAAV, porovnáme-li je

- globálně, dostaneme:

```
FTFTALILLAVAV
FT--AL-LLA-AV
```

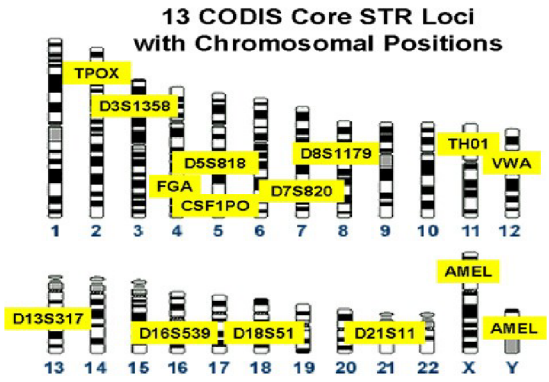
- lokálně, dostaneme:

```
FTFTALILL-AVAV
--FTAL-LLAAV--
```

Globální porovnávání se využívá v algoritmu Needleman-Wunch a v algoritmu Smith-Waterman je zase využíváno lokální porovnávání. Při stejné délce řetězců je výsledek lokálního porovnání podobný porovnání globálnímu.

2.2 Párové porovnání

DNA poskytuje dostatečný prostor pro porovnávání živočišných i rostlinných druhů. Z důvodu dostatečně objemného množství dat v něm uložených, poskytuje DNA také klíč k rozpoznávání jedinců v rámci druhu. Pro jednoznačnou identifikaci osob je používán systém, který vybere z DNA 13 úseků, které danou osobu jednoznačně popisují. Tyto úseky jsou později popsány určitou sekvencí hodnot, ty se zapisují do pásku ve formě svislých čar. Poté je možné od sebe odlišit dvě různé osoby pouhým pohledem na pásky. Tento způsob identifikace je využíván například v systému CODIS využívaným FBI – ukázka na obrázku č. 2 (CODIS je akronym Combined DNA Index System).



Obr. 2: CODIS (převzato z [34])

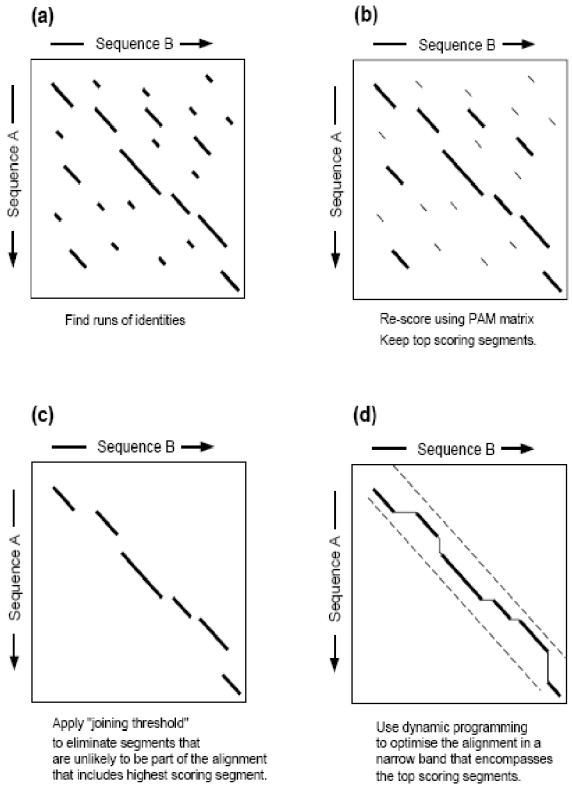
Porovnávání sekvencí DNA je výpočetně náročná úloha, zejména pokud je algoritmus implementován softwarově - platí se zde totiž určitá daň za flexibilitu softwarového systému. Porovnávání vzorků DNA lze akcelerovat v hardwaru. Mezi nejčastěji používané algoritmy pro porovnávání řetězců DNA patří FASTA, BLAST a nebo Smith-Waterman. Algoritmus Needleman-Wunsch tvoří základ Smith-Watermanova algoritmu, který je vhodný pro lokální porovnání [5]. Needleman-Wunschův algoritmus je na druhou stranu vhodný pro porovnávání globální. Algoritmy FASTA a BLAST jsou mnohem rychlejší než algoritmus Needleman-Wunsch nebo Smith-Watermanův algoritmus. Nevýhodou algoritmů FASTA a BLAST je, že používají heuristické filtrování záznamů databáze. Proto se často používají nejprve tyto rychlé algoritmy pro vymezení určité skupiny DNA, u které se poté provádí podrobné porovnání. Toto zúžení se provádí z důvodu velké náročnosti Smith-Watermanova a Needleman-Wunschova algoritmu na procesorový čas a systémovou paměť. Tento přístup je však problematický, protože se při heuristické filtraci mohou algoritmy teoreticky dopustit chyby. Tato chyba by mohla způsobit, že

by například biolog pátrající po novém typu léku proti zákeřné nemoci nenalezl potřebnou shodu v sekvenci DNA/proteinu. Shody by dosáhl teprve tehdy, pokud by použil přesnější metody.

Algoritmus Smith-Watermanův hledá lokální shodu a ve srovnání s algoritmy heuristickými je spolehlivý. Hledání v databázi pomocí tohoto algoritmu je zdlouhavé, ale jak již bylo zmíněno dříve, k urychlení porovnání lze implementovat část algoritmu v hardwaru a akcelerovat tak výpočet. Implementováním algoritmu Smith-Waterman v hardwaru lze docílit výkonu srovnatelného s 800 Alpha stanicemi. (viz tabulka č. 1) Popis těchto algoritmů je v kapitole 2.2.1 až 2.2.4.

2.2.1 FASTA

Původně vznikl jako FASTP přednostně určený pro hledání podobných sekvencí v proteinech. David J. Lipman a William R. Pearson s ním poprvé přišli roku 1985. O tři roky později se FASTP adaptovala i na porovnávání DNA sekvencí. Na rozdíl od toho je FASTA určená pro porovnávání jakékoliv posloupnosti aminokyselin a lze jí porovnávat protein, DNA a libovolné peptidové sekvence s libovolným řetězcem aminokyselin. FASTP znamená „fast protein“ a FASTA je zkrácení „fast all“, což volně přeloženo, znamená rychlé porovnání proteinů/všeho [22].

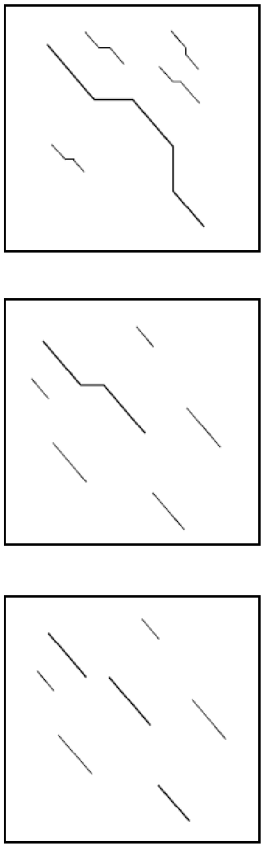


Obr. 3: Algoritmus FASTA (převzato z [13])

Tento algoritmus funguje tak, že porovnává vstupní sekvenci a hledá podobnost se sekvencemi v databázi. Podobnost se hledá lokálně. Algoritmus prohledává řetězce v databázi a srovnává je se vstupním řetězcem. Porovnání probíhá po několika znacích najednou – v celých slovech. To, jak je dlouhé slovo, o tom rozhoduje parametr algoritmu, tzv. ktup (udává počet znaků ve slově). Tento parametr se nastavuje menší pro porovnávání proteinových sekvencí (ktup = 2) a pro dlouhé nukleotidové kyseliny větší (ktup = 6). Při dlouhých porovnávacích sekvencích (větších ktup) se stává algoritmus méně citlivý, ale na druhou stranu je hledání rychlejší. Pro maximální citlivost algoritmu se volí ktup = 1. Postup je takový (viz obrázek č. 3), že se nejprve vytvoří posloupnost různě dlouhých sobě si odpovídajících sekvencí. Z těchto sekvencí se vybere 10 nejlepších shod, tedy 10 nejdelších shodných souvislých řetězců. Tyto řetězce se znovu přehodnotí a vytvoří se řetězec, který bude nejvíc podobný vzorovému, s tím že se penalizují rozdílné znaky a vkládání mezer. Pro optimální ohodnocení lze použít Smith-Watermanův algoritmus [21].

FASTA je rychlejší než výpočet podobnosti pomocí dynamického programování (algoritmy Needleman-Wunch, Smith-Waterman), protože porovnání je omezeno jen na známé části se shodnými posloupnostmi aminokyselin.

2.2.2 BLAST



Obr. 4: Srovnání algoritmů (převzato z [35])

BLAST je zkratka algoritmu Basic Local Alignment Search Tool. Poprvé jej popsali ve společné práci roku 1990 Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers a David J. Lipman. Jedná se o algoritmus pro porovnávání biologických sekvenčních dat, typologicky proteinů a DNA.

Porovnání probíhá u dat umístěných v databázi a umožňuje zobrazovat procentuální podobnost vstupního řetězce s řetězci v databázi. BLAST hledá ve vstupním řetězci podsekvence shodné s podsekvencemi v řetězcích uložených v databázi. Prohledávání dat v databázi probíhá vysokou rychlostí (rychlost je pro tento algoritmus upřednostňována před kvalitou výsledku) [3]. V porovnání s algoritmem Smith-Waterman je algoritmus asi 50krát rychlejší.

Algoritmus má tři fáze. V první fázi se vstupní řetězec rozdělí na krátká slova, většinou na slova o délce 4 znaků. Rozdělení probíhá vždy tak, že se nejdříve vezme první čtveřice znaků od prvního znaku vstupu a vytvoří se z nich slovo. Poté se vezme další čtveřice znaků od druhého znaku a vytvoří se slovo další. V druhé fázi se postupně načítají jednotlivá slova a

porovnávají se se záznamy v databázi. Jakmile je nalezena shoda slova s podsekvencí řetězce aminokyseliny v databázi, algoritmus se pokouší za pomoci všech slov shodu rozšířit. Rozšíření shody probíhá všemi směry. Tato část algoritmu se označuje jako třetí část. Toto je posloupnost kroků algoritmu, který nedovoluje mezery [8].

V poslední době se však používá verze algoritmu, který přidává čtvrtou fázi. V této fázi probíhá rozšíření shodných podsekvencí povolující mezery. BLAST má několik specifických variant, například BLASTP – vhodný pro hledání podobností mezi proteiny, BLASTN nebo BLASTX jsou vhodné pro hledání podobností mezi proteiny a nukleotidovými kyselinami. PSI-BLAST ukládá výsledky hledání do speciální ohodnocovací matice, která má zvláštní použití pro metody vizualizace a odhadování struktury aminokyselin. Nejčastěji používanou verzí tohoto algoritmu je BLAST verze 2,0. Tento algoritmus umožňuje při hledání vkládat mezery do řetězce a tím eliminovat změny, které se do řetězců zanáší vlivem evoluce.

Největším kladem tohoto algoritmu je jeho vysoká rychlost, která je však na úkor přesnosti. Ukázka srovnání výkonu s algoritmem FASTA a s přesným algoritmem Smith-Waterman (obrázek č. 4).

2.2.3 Needleman-Wunsch

Tento algoritmus má ve svém názvu jména svých tvůrců. Roku 1970 jej vymysleli Saul Needleman a Christian D. Wunsch [12]. Algoritmus se používá pro hledání podobnosti na úrovni globální a je první aplikací dynamického programování v porovnávání biologických sekvencí.

Dynamické programování rozdělí problém na podproblémy, jejichž výsledky jsou rychleji vypočítané, a poté se z částečných výsledků získává konečný výsledek. Tento způsob výpočtu je v bioinformatice vhodný, protože se zde počítá s velkým množstvím možných porovnání. Algoritmus vyplňuje tabulku částečných výsledků a vypočítává skóre porovnání [1]. Na ose x (vertikální směr) jsou zaneseny znaky jednoho řetězce a na ose y (horizontální směr) znaky řetězce druhého. Jakmile je tabulka vyplněna celá, hledá se cesta tabulkou z pravého dolního rohu do levého horního rohu. Porovnání dvou řetězců poté odpovídá cestě touto tabulkou čtenou z levého horního rohu do pravého dolního. Diagonální část cesty odpovídá shodě v řetězcích - nukleotidech. Horizontální část cesty v tabulce odpovídá mezeře v řetězci zapsaného u osy y. Vertikální část cesty v tabulce odpovídá mezeře v řetězci zapsaného u osy x.

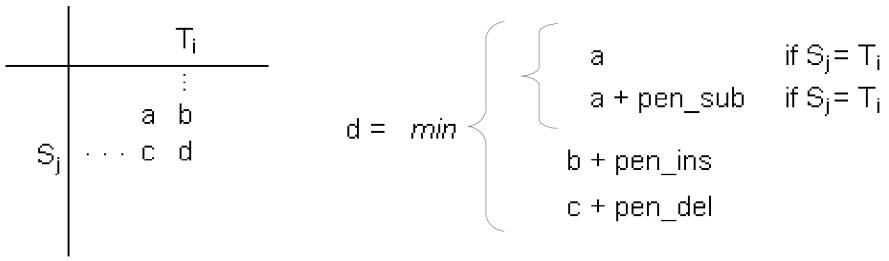
Před vlastním vypočítáváním mezivýsledků musí proběhnout inicializace tabulky. Protože tabulka popisující porovnání řetězců o délce m a n má velikost $(m+1) \times (n+1)$, je nutné první sloupec a první řádek tabulky inicializovat aritmetickou řadou rovnou součtům penalt za mezery [11]. Výpočet částečných výsledků v tabulce se řídí následujícími pravidly:

a) přečteme hodnotu zleva a přičteme k ní penaltu za smazání nukleotidu

b) načteme hodnotu shora a přičteme k ní hodnotu penalty za vložení mezery

c) přečteme hodnotu z levé horní diagonály a přičteme k ní skóre za shodu respektive penaltu za nahrazení.

Z těchto tří variant vybereme tu, jež nám dává minimální hodnotu, kterou zapíšeme do aktuálního pole tabulky (viz obrázek č. 5). Tímto způsobem vyplníme celou tabulku a hodnota, kterou zapíšeme do pravého dolního rohu je skóre optimálního porovnání řetězců [6]. Při zpětné cestě tabulkou čteme porovnání a to tak, že z každé pozice zjišťujeme kudy se mohlo původně tabulkou jít. Při směru diagonálním dochází ke shodě řetězců a při jiném se vkládá mezera. Pokud je cesta vertikální/horizontální mezera se vkládá do řetězce ve směru osy *x/y* [11]. Jakmile se dostaneme do levého horního rohu, přečetli jsme porovnání řetězců. Během zpětného vyhodnocování můžeme přijít na více možných cest tabulkou, optimální porovnání má v tom případě více variant.



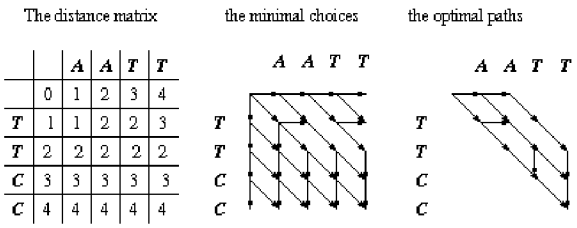
Obr. 5: Schéma algoritmu dynamického programování

2.2.4 Smith-Waterman

Roku 1981 byl popsán vědci Templem Smithem a Michaelem Watermanem [16]. Tento algoritmus se používá pro hledání lokální podobnosti dvou nukleotidů nebo sekvencí proteinů. Často se používá k hledání relativně krátkého řetězce v dlouhém vzorovém řetězci (v řádech tisíců až milionů znaků). Například máme podsekvenci, která nás zajímá a hledáme ji v celém řetězci deoxyribonukleové kyseliny. Pro tento typ problému není globální metoda porovnávání vhodná, protože ta každou mezeru ohodnocuje určitou penaltou. Vhodnou metodou je metoda lokálního porovnávání, tedy algoritmus Smith-Waterman.

Pro lokální porovnávání je použito dynamického programování, které bylo použito i u Needleman-Wunchova algoritmu s úpravou výpočtů částečných výsledků zapisovaných do tabulky. Pokud by se měla do tabulky zapsat záporná hodnota, zapíše se na místo záporné hodnoty hodnota 0. Jakmile je tabulka sestavena, a jsou vyplněny všechny její buňky - tzn. máme vypočteny všechny částečné výsledky - postupujeme tak, že najdeme v tabulce buňku s nejvyšší hodnotou a od této buňky poté hledáme cestu stejným postupem jako v případe Needleman-Wunchova algoritmu.

Hledáme do té doby, dokud nedojdeme k hodnotě v buňce 0 [21]. Tato cesta popisuje nejlepší lokální shodu řetězců. Opět platí, že porovnání může mít a často také má více variant.



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

Obr. 6: Algoritmus Needleman-Wunsch
(převzato z [36])

Obrázek č. 6 popisuje proces porovnávání dvou řetězců AATT a TTCC. V první třetině obrázku je vidět vyplnění tabulky s hodnotami mezivýsledků, ve druhé je znázorněn pomocí šipek směr přenosu hodnot do jednotlivých buněk tabulky a v poslední je pak znázorněna cesta porovnání. Je vidět, že od posledního znaku k prvnímu vede více cest nejlepších porovnání.

Dostane-li při vstupu Smith-Watermanův algoritmus řetězce o značně rozdílných délkách – porovnává lokálně. Porovnávají-li se ale řetězce o podobných délkách, pak je výsledek podobný globálnímu porovnání. Tuto vlastnost, Needleman-Wunchův algoritmus nemá. Proto se v praxi častěji implementuje algoritmus Smith-Watermanův. Výpočetní doba tohoto algoritmu roste s délkou řetězců kvadraticky. Je-li implementováno dynamické programování v hardware, pak je složitost problému lineární. Z tohoto důvodu lze implementací v hardware docílit velkého zrychlení.

2.3 Vícenásobné porovnání

V předchozí kapitole byla popsána problematika porovnávání dvou řetězců. Vícenásobné porovnání, které je předmětem této kapitoly, lze chápat tak, že základní párové porovnání je speciální případ porovnání vícenásobného.

Pokud se podíváme na metody dynamického programování z předchozí kapitoly, řeší problém v dvoudimenzionálním prostoru. V tabulce je na jedné ose řetězec zdrojový a na druhé pak řetězec testovaný. Pokud porovnáváme více řetězců, potřebujeme porovnat každý nukleotid řetězce s každým nukleotidem v řetězcích ostatních. Toho dosáhneme tím, že vytvoříme tabulku o více rozměrech. Na obrázku č. 7 je znázorněna ukázka tabulek částečných výsledků pro dva a tři řetězce. Lze si všimnout souvislosti mezi počtem porovnávaných řetězců a dimenzí tabulky. Pro dva řetězce je použito tabulky a pro tři řetězce tabulky tvaru krychle. Pokud by bylo třeba porovnat čtyři řetězce, pak by bylo použito hyper-krychle a tak dále. Při porovnání dvou řetězců je vybíráno skóre do aktuální buňky ze tří buněk, při porovnávání tří řetězců je vybíráno ze sedmi. Obecně je vybíráno ze $(2^n - 1)$ buněk, kde n je počet porovnávaných řetězců.

Obr. 7: Tabulka mezivýsledků pro dva a tři řetězce (převzato z [40])

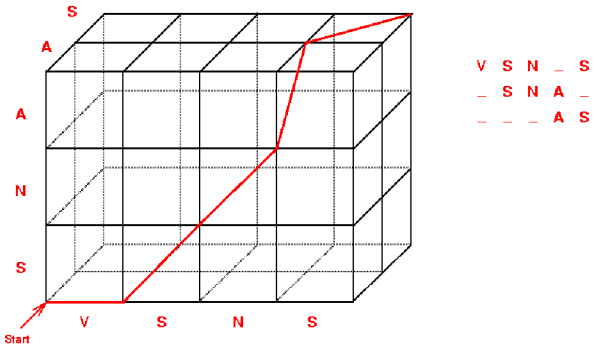
Výpočetní komplexnost tohoto způsobu porovnání je natolik výpočetně náročná, že ani výkonnost superpočítačů nebo síť pracovních stanic neposkytuje dostačující výkon pro porovnávání více než dvaceti řetězců [21].

Exaktní algoritmy na větším počtu sekvencí nejsou z výkonnostního hlediska možné, proto byla hledána cesta v metodách využívajících heuristiky. Nevýhodou heuristických metod je, že nezaručují nalezení nejlepšího porovnání, ale pouze porovnání blízkého nejlepšímu. Výsledky těchto metod nejsou tedy přesné, avšak jsou určitým způsobem aproximovány. Nejznámějším zástupcem heuristických algoritmů je algoritmus CLUSTAL.

Pro další metodu sestavování porovnání více řetězců jsou využívány takzvané vzdálenostní matice. Vzdálenostní matice, je matice obsahující evoluční vzdálenosti mezi genetickými řetězci. Tato metoda je nejčastěji používána pro vytváření fylogenetických stromů (někdy také nazývaných - evoluční stromy), což jsou stromy znázorňující evoluční podobnost řetězců. Z biologického hlediska se jedná o zobrazení podobnosti genetických kódů mezi organismy.

2.3.1 Dynamické programování

Základní varianty tohoto způsobu porovnání byly popsány v předchozí podkapitole, přesněji v článcích 2.2.3 a 2.2.4. Tyto články popisovaly speciální případy, kdy se porovnávaly právě dva řetězce. Dynamické programování je možné použít pro libovolný počet řetězců použitím vícerozměrné tabulky částečných výsledků. Vícenásobné porovnání pomocí dynamického programování je pro výpočet velice náročné. Je tomu tak z toho důvodu, že pro n řetězců je třeba vytvořit n rozměrnou tabulku. V této tabulce se poté vypočítávají hodnoty jednotlivých buněk stejným způsobem, jak v algoritmech Needleman-Wunsch a Smith-Waterman [21].



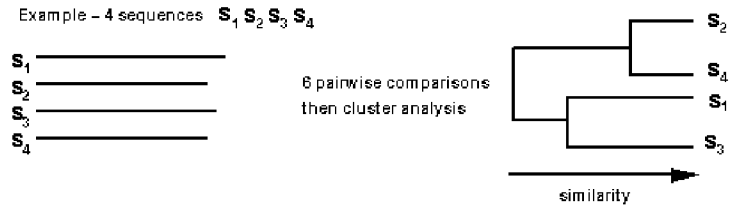
Obr. 8: Ukázka porovnání tří obecných řetězců (převzato z [41])

Metoda dynamického porovnávání pro více řetězců je charakteristická svou přesností, ale také svými vysokými nároky na výpočetní výkon a paměťový prostor. Je tedy vhodná pro případy, kdy je třeba zaručit nalezení nejlepšího porovnání.

2.3.2 CLUSTAL

Tento algoritmus se používá pro vícenásobné porovnání sekvencí a má tři fáze:

1) párové porovnání po dvojicích řetězců
2) sestavení fylogenetického stromu
3) vlastní vícenásobné porovnání.

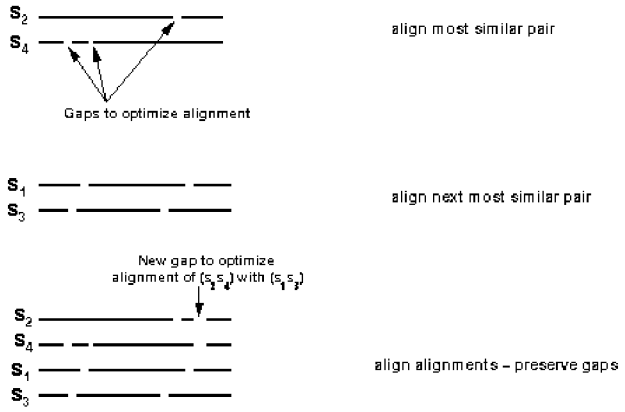


Obr. 9: Sestavení fylogenetického stromu (převzato z [41])

V první fázi se porovnává každá sekvence s každou. Pro toto porovnání lze použít rychlý přibližný výpočet a nebo přesný výpočet pomocí algoritmu dynamického programování [26]. Z těchto porovnání každého řetězce s každým se vypočte vzdálenostní matice. Ta značí evoluční vzdálenost mezi porovnávanými řetězci.

Ve druhé fázi se vytváří pomocí Neighbor-Joining metody sestaví evoluční strom na základě hodnot, které byly získány v předchozím kroku. Jelikož je strom bez kořene, kořen se umístí metodou „mid-point“. Na obrázku č. 9 je znázorněn výsledek po párovém porovnání a následném sestavení stromu.

Posledním krokem je vlastní porovnání. To se provádí tak, že se porovnají dva nejpodobnější řetězce. K určení pořadí vstupu řetězců slouží evoluční strom, který byl sestaven v předchozím kroku. Po prvním porovnání se přidá řetězec třetí. Ten se porovná s předchozím porovnáním, a tak dále, dokud nejsou porovnány všechny řetězce (viz obrázku č. 10).



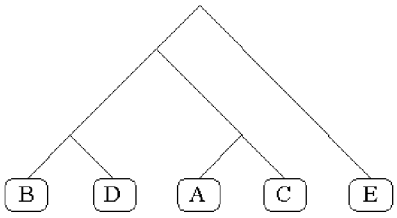
Obr. 10: Postupné porovnávání řetězců
(převzato z [41])

2.3.3 UPGMA

UPGMA je akronym vzniklý ze slovního složení „Unweighted Pair Group Method with Arithmetic mean“. Jedná se o algoritmus, který vznikl v počátcích šedesátých let minulého století [21]. Jedná se o statistický způsob porovnávání jednotlivých řetězců a vytváření vzdáleností matice s následným sestavením fylogenetického stromu.

Vstupem nemůže být libovolný řetězec – základním předpokladem tohoto algoritmu je, aby délky řetězců byly pokud možno stejně dlouhé a vykazovaly minimální rozdílnosti v nukleotidech. Během porovnávání se tak řeší pouze počet neshodných nukleotidů.

Postup je takový, že se pro řetězce vytvoří trojúhelníková matice k jejich srovnání. Matice se vytvoří tak, že se pro každý prvek matice vypočítá vzdálenostní rozdíl jednotlivých řetězců. Jakmile je výpočet celé matice hotov, vybere se nejmenší hodnota matice a řádky/sloupce řetězců s nejmenší hodnotou jsou v další vzdálenostní matici sloučeny s průměrem předchozích hodnot. Výpočet se poté dále opakuje. Řetězce, které byly sloučeny jsou si navzájem nejpodobnější a zapíší se proto do stromu tak, že mají společného přímého předka (jím je právě sloučený řádek/sloupec). Některé hodnoty z předchozí matice lze zapsat i do matice nové, a tím ušetřit výpočetně náročný krok výpočtu podobnosti. Algoritmus probíhá do doby, dokud nejsou všechny řetězce zapsané v evolučním stromu, který udává podobnost řetězců. K vlastnímu porovnání jednotlivých řetězců se používá porovnání přímé, tj. porovnání sobě si odpovídajících nukleotidů.



Obr. 11: UPGMA

(převzato z [37])

2.4 Srovnání algoritmů

2.4.1 Srovnání vícerozměrných algoritmů

Algoritmy dynamického programování jsou, co se týče přesnosti, ze všech výše uvedených algoritmů nejpřesnější. Při jakémkoliv rozměru a jakékoliv míře podobností mezi řetězci algoritmy naleznou vždy řešení nejoptimálnější. Tato přesnost je vyvážena nároky na výpočetní systém, které rostou s délkou řetězců a jejich počtem. Přesněji je pro vypočtení porovnání potřeba

$$\tau = t\, m^n$$

τ jednotek času, kde m je rovno délce sekvencí, n je počet porovnávaných řetězců a t je doba, za kterou je vypočten jeden podvýpočet (tedy jedna buňka tabulky).

Dále popisovaným algoritmem je algoritmus, který již není tak přesný jako algoritmy používající dynamického programování, ale jeho přesnost je pro většinu porovnání dostačující. Pro vypočtení jednoho porovnání je třeba

$$\tau_1 = t_1 * ((n - 1)\, n / 2) * m^2$$

τ_1 časových jednotek. t_1 je doba, po kterou trvá výpočet jedné buňky. U tohoto algoritmu je potřeba dále sestavit strom (dobu sestavení stromu zanedbáme) a setříděné řetězce dále porovnat. Pokud setřídění dvou řetězců trvá t_2, pak setřídění všech řetězců bude hotovo za τ_2

$$\tau_2 = t_2 * (n - 1)$$

Posledním z uvedených algoritmů vícenásobného porovnání je algoritmus UPGMA, který je svými nároky nejméně náročný. K realizaci porovnávacího bloku dvou nukleotidů u tohoto algoritmu stačí nejméně výpočetních prvků. Výpočet počtu shod lze provést teoreticky okamžitě. Pokud i v tomto kroku zanedbáme dobu sestavení stromu, pak doba výpočtu tohoto algoritmu je rovna době redukce evoluční matice.

Z výše popsaných algoritmů byl k implementaci vybrán algoritmus, který je méně náročný na výpočetní výkon než algoritmy Smith-Watermanův a Needleman-Wunchův. První byl vybrán algoritmus UPGMA, který je ale pro svou jednoduchost pro praktické použití nevhodný. Jeho nevýhoda spočívá v tom, že pokud máme dva identické algoritmy a u jednoho z nich přemístíme nukleotid z konce sekvence na začátek, dostaneme dvě sekvence, které jsou podle algoritmu naprosto rozdílné. Proto byl implementován algoritmus, který je variací UPGMA a CLUSTAL. Postupně se načítají řetězce po dvojicích a porovnávají se metodou dynamického programování. Skóre shody dvojic řetězců se ukládá do matice výsledků. Z této matice lze poté sestavit evoluční strom shodným postupem jako při algoritmu UPGMA.

2.4.2 Odhad náročnosti na umístění v konkrétním FPGA

Implementace bude provedena v jazyku VHDL a poté syntetizována do FPGA čipu použitého v kartě COMBO 6 [50]. Na této kartě je umístěn FPGA čip od firmy Xilinx – XC2V 3000, který v sobě obsahuje pole o 3.584 CLB blocích. Podle odhadu, který je podložen údaji ze zdroje [39], předpokládám, že do pole o takovém počtu CLB bloků se vejde 2.866 bloků PE. Potom pro implementaci dynamického programování bude možné mít čtyř-rozměrnou tabulku a v ní porovnávat 4 řetězce o velikosti 14 nukleotidů. Pro porovnání větších řetězců je třeba použít čip obsahující více CLB bloků a nebo použít FIFO paměť a data ve výpočetní tabulce překrývat. Toto řešení bude vhodné pro krátké úseky DNA.

Druhý způsob vícenásobného porovnání CLUSTAL, v první fázi porovnává každý řetězec s každým řetězcem s využitím algoritmu Smith-Waterman. V FPGA tedy může probíhat porovnávání i delších sekvencí (protože se jedná o porovnávání dvoudimenzionální). Předpokládám, že v tabulce částečných výsledku bez použití FIFO paměti bude možné porovnávat řetězce o rozměrech 53 x 53 nukleotidů.

Způsob, který byl uveden v podkapitole UPGMA je co se týče přesnosti nejméně kvalitní, ale na druhou stranu nejrychlejší. Jedno porovnání zabere z uvedených algoritmů nejmenší plochu FPGA čipu. SLICE bloky obsažené v CLB blocích je možné využít při porovnávání jako Look-up table. Tímto přístupem je možné v každém SLICE realizovat jedno porovnání nukleotidů. Předpokládám, že v jeden okamžik bude možné porovnat 14.336 nukleotidů.

Podle odhadů se zdá, že čip osazený v desce COMBO 6 je pro implementaci vhodný, protože nabízí postačující počet výpočetních jednotek, je dostatečně rychlý a pro implementaci porovnávacího algoritmu nabízí postačující rychlost. Jelikož karta COMBO-PTM obsahuje FPGA čip Spartan3 – X3S200 s 3840 CLB bloky a navíc byla při implementaci dostupnější, byl algoritmus implementován v ní.

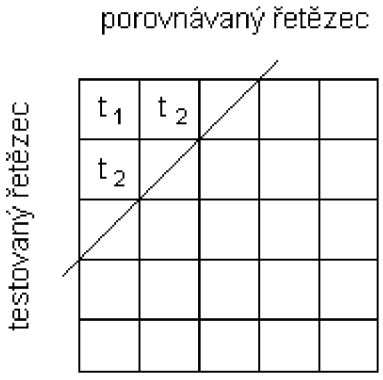
Výše uvedené odhady velikostí jsou pouze přibližné, protože v odhadech není zahrnuta realizace potřebné přídavné logiky, ale pouze realizace prostorově nejnáročnějších PE.

2.5 Implementace SW/NW

Algoritmus Smith-Waterman a algoritmus Needleman-Wunsch jsou pro implementaci v hardwaru vhodné. Je to z důvodu možného přenesení dynamického programování do tabulky výpočetních prvků a z důvodu nutnosti realizace výpočtu v systolickém systému. Základem systolického systému je pole procesních prvků, které postupně vypočítávají hodnocení. Každý z prvků komunikuje jen s prvky v jeho okolí. Systolické systémy mohou být složeny z pole výpočetních serverů a nebo mohou být implementovány ve specializovaných obvodech typu ASIC nebo FPGA. Implementaci v hardwaru usnadňuje také to, že procesní elementy jsou zpravidla všechny implementované stejně. Na obrázku č. 12 je znázorněno systolické pole. V čase t_1 probíhá výpočet jednoho elementu, v čase t_2 probíhá výpočet dvou elementů a tak dále jak je naznačeno na obrázku. Celý výpočet trvá

$$(m_1 + m_2 - 1) * t$$

jednotek času, kde t je doba mezi přijmutím dat procesním elementem a odesláním výsledku do sousedního elementu a m_1 a m_2 jsou délky porovnávaných řetězců. Výpočetní krok je také někdy označován termínem výpočetní vlna, pro svou názornost bude tohoto označení použito i v této práci. V hardwaru se toto pole realizuje tak, že se vytvoří vektor procesních elementů, do něhož jsou ze strany zaslána data a z čela postupně zasílána data druhá. Pro každý sloupec systolického pole je tedy vytvořen jeden procesní element. V každém kroku procesní element načte data z levého elementu stejné úrovně, z levého horního a z horního. V hardware každý procesní element zastupuje celý svůj sloupec, proto se posílají jen data která byla vypočtena v levém procesním elementu. Pokud je řeč o datech, data jsou zakódována ve vektoru signálů, který popisuje typ aminokyseliny. V případě DNA stačí pro popis 2 bity (je třeba kódovat 4 druhy nukleotidů) a v případě proteinů je k zakódování potřeba 5 bitů (je třeba zakódovat minimálně 20 aminokyselin).



Obr. 12: Systolické pole

3 Shrnutí současného stavu

3.1 Srovnání výkonu v SW a HW

Jak již bylo popsáno v kapitole zaměřené na algoritmy párového porovnání, složitost algoritmů dynamického programování je v softwaru mocninná s mocnitelem rovného počtu porovnávaných řetězců. V hardwaru je tento problém lineární, protože se již při návrhu takového obvodu vytvoří potřebný počet procesních elementů (zařízení vypočítávající skóre aktuální buňky tabulky) tak, aby složitost byla lineární.

Pro zajímavost, je v tabulce č. 1 uvedeno srovnání výkonnosti. V prvním řádku tabulky je uvedena výkonnost 800 stanic Alpha, které vypočtou za vteřinu 250 Byte (zkráceně B) dat. Pokud srovnáme výkonnost s posledním řádkem, u kterého je výkonnost jednoho čipu obsahujícího 11 tisíc výpočetních bloků, docházíme k závěru, že je toto číslo téměř 13 krát vyšší než je tomu u Alpha stanic. Tím se uspoří velká část energie na provoz 800 stanic, chlazení místnosti, úspora prostoru a co je významné – ušetří se finance.

	Procesorů na jedno zařízení	**Počet zařízení**	**Updatů za vteřinu**
Celera (Alpha cluster)	1	800	250 B
Paracel (ASIC)	192	144	276 B
TimeLogic (FPGA)	6	160	50 B
Splash 2 (XC 4010)	14	272	43 B
JBits (XCV1000-6)	4.000	1	757 B
JBits (XC2V6000-5)	11.000	1	3.225 B

Tabulka 1: Srovnání výkonnosti (hodnoty z [39])

3.2 Srovnání implementací v hardware

K dnešnímu dni bylo vytvořeno velké množství akceleračních implementací, za účelem zrychlení výkonnosti porovnávacího algoritmu. Jedním z prvních pokusů byl pokus o zrychlení výpočtu skóre a to tak, že se uvažovala jen shoda/neshoda a v případě neshody se připočetlo ke skóre 2. Tuto úpravu zavedl Lipton a Lopresti r. 1985. Následoval pokus Hoanga T. a Loprestiho P. s architekturou pojmenovanou SPLASH. Jejich první verze této architektury měla 8 procesních elementů (PE), těchto 8 PE tvořilo jeden z 32 obvodů. Výpočetní výkon byl velice nízký – 0,37 B updatů/s (zkráceně BUDps). Další verze, na které pracoval již jen Hoang T. se objevila o rok později, tedy roku 1993, a ta měla již 14 PE a skládala se z 272 zařízení (viz. tab č. 1). Zvýšení počtu procesních elementů vedlo také ke zvýšení počtu BUDps na více než desetinásobek – na 43 BUDps. Dalším pokusem o zvýšení výkonnosti bylo implementování akcelerace do ASIC obvodu. Zvýšení výkonu se pánům Hanovi T. a Parameswaranovi S. nepodařilo, výkonnost této architektury pojmenované SWASAD byla 3,2 BUDps. Velkým skokem ve výkonnosti byla až architektura FPL2003, kterou navrhli a implementovali C. W. Yu a K.H.Kwong. Tato architektura obsahovala 4032 PE na čipu jehož interní hodiny byly 202 MHz a díky tomu dosahovala 814 BUDps. S rostoucí hustotou tranzistorů v čipech je možné do čipů umisťovat stále více PE, např. S. A. Guccione a E. Keller vytvořili architekturu JBits, která obsahuje dostatečně velký čip Xilinx do něhož implementovali asi 11.000 PE. Čip má vnitřní frekvenci 280 MHz a proto zvládá 3225 BUDps [49].

Výkonnost reprogramovatelných čipů neustále roste, díky neustálému zvyšování hustoty tranzistorů na ploše čipu a díky zvyšování jejich frekvencí. Zásadní problémy jsou ale dva. První spočívá v tom, že každé porovnání potřebuje jiné nastavení obvodu. Přesněji, každý biolog srovnávající takové řetězce má jiné požadavky na podobnost srovnávaných řetězců – co se hodnot penalt a délky srovnávaných řetězců týče. Tento problém řeší obecné architektury, které se snaží vytvořit platformu která by byla automaticky rekonfigurovatelná podle zadaných parametrů.

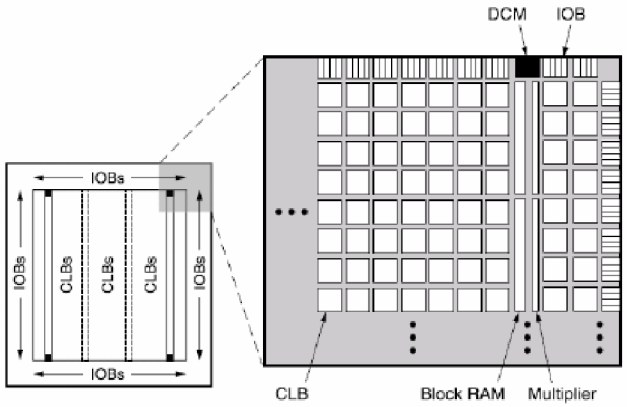
Podstatou druhého problému je, že pokud by bylo potřeba porovnávat více řetězců a byla požadována absolutní přesnost, pak nemohou stačit ani ty nejmodernější procesory. Důvodem je, že například pro porovnání 5 řetězců o délce 100 nukleotidů jedním průchodem by bylo potřeba v hardwaru implementovat 10^8 PE. Pro algoritmus, kdy se postupně porovnávají dvojice řetězců a zapisují se vypočítaná skóre do tabulky, by pro stejné řetězce bylo potřeba jen 100 PE a paměť na 10 výsledných skóre. Z nich by bylo poté možné sestavit evoluční strom, případně některé z řetězců samostatně porovnat. Tato práce se zabývá takovým přístupem, kdy se po dvojicích porovnají řetězce a následně se pomocí softwaru sestaví evoluční strom popisující vícenásobné porovnání.

4 FPGA

4.1 Základy technologie FPGA

FPGA je akronym vzniklý z názvu technologie „Field Programmable Gate Array". Jedná se o polovodičovou technologii programovatelného obvodu obsahující programovatelné funkční bloky a programovatelné vzájemné propojení. Vnitřní konfiguraci FPGA je možné v řádech několika milisekund měnit a tím i měnit funkci FPGA. To, jakou funkci FPGA vykonává, definuje jeho vnitřní propojení komponent složených ze základních logických hradel. Základní logická hradla tvoří základní stavební prvek funkčních jednotek a paměťových bloků.

FPGA obsahuje velké množství polí CLB (Configurable Logic Block; řádově několik tisíc) a dále obsahuje vestavěné funkčních bloky. Funkčními bloky jsou obvody specializované pro určitou funkci. Jsou to bloky paměti (Block RAM), rychlé násobičky (Multiplier), vstupně/výstupní bloky (IOB), vstupně/výstupní port pro komunikaci vysokou rychlostí (RocketIO; rychlost přenosu až 3,125Gb/s), platformu PowerPC a blok úpravy hodin (Digital Clock Manager zkráceně DCM). Block RAM se používá jako paměť dat, paměť typu FIFO, ke konstruování konstrukcí konečných automatů a podobně. Blok Multiplier lze použít pro násobení a nebo pro paralelní posuvy [31]. CLB bloky jsou tvořeny SLICE bloky, carry logikou a pomocnou logikou pro aritmetické operace [32]. Ukázka struktury FPGA je znázorněna na obrázku č. 13.



Obr. 13: FPGA (převzato z [38])

Konfigurace FPGA se popisuje nejčastěji pomocí jazyka Verilog nebo VHDL. Jedná se o HDL jazyky (Hardware Description Language), tedy jazyky popisující hardwarové zapojení. Jazyk VHDL je častěji používaný v Evropě a v této práci je také použit. Zkratka VHDL vznikla použitím zkratky HDL a prvního písmene označení typu obvodů Very High Speed Integrated Circuit.

Obvod FPGA má tu výhodu, že jej lze modelovat a simulovat. Díky možnosti testování a simulaci výsledného obvodu mohou být chyby nalezeny již ve fázi návrhu, a poté v krátkém čase odstraněny. Tato flexibilita ve srovnání s obvody ASIC přináší tu výhodu, že je možné výsledné zařízení s FPGA rychleji uvést na trh. Nevýhodou ve srovnání s ASIC obvody je vyšší spotřeba a menší výkonnost. Mezi oblasti, ve kterých se FPGA používá nejčastěji patří: prototypování ASIC obvodů, DSP, SDR, medicína, počítačové vidění, rozpoznávání řeči, kryptografie, bioinformatika, emulace hardwaru, a další oblasti, kde je potřeba určitou část výpočtu akcelerovat a zároveň si ponechat možnost budoucích úprav [47].

5 Implementace

Během studia problematiky porovnávání DNA řetězců docházelo k určitému vývoji v pohledu na problematiku a budoucí zaměření této práce. V počátku bylo cílem práci směrovat na porovnání párových sekvencí. Zejména pak, zhodnotit výkonnost FPGA v porovnání se softwarovým řešením. Představa o další práci byla implementovat v FPGA dynamické programování s určitými specifiky. Například možnost srovnávání DNA řetězců i proteinových řetězců s výpočtem porovnání přímo v FPGA. Stejný algoritmus by se implementoval v jazyku C++ a výsledky obou řešení by se porovnaly jak v rychlosti výpočtu tak i ceně provozu a flexibilitě.

Aby byl rozdíl ve výkonu co největší, bylo by zajímavé implementovat algoritmus Smith-Waterman v HW a porovnat s řešení v software. Pro demonstraci flexibility FPGA čipu by bylo vhodné ponechat možnost změny hodnot přiřazovaných porovnávaným nukleotidům ve formě skóre a nebo v případě negativních hodnot - penalt. Během zkoumání se dospělo k závěru, že hodnoty skóre/penalt mají vliv pouze na to, jaký řetězec je označen za podobný a ne, jak bylo prvně mylně myšleno, na zvýšení výkonu. Hodnota skóre, se kterou se v algoritmu počítá není parametrem, který by určoval kompromis mezi rychlosti a přesností, ale jedná se o statistický koeficient. Hodnoty penalt jsou pravděpodobnosti, že v daném nukleotidovém řetězci dojde k určitému jevu. Například pravděpodobnost vložení nukleotidu je vyšší, než záměna nukleotidu za jiný, a proto je penalta za vložení znaku příznivější.

Během studia oblasti porovnávání DNA sekvencí bylo zjištěno, že DNA lze porovnávat nejen podle sekvence nukleotidů, ale i podle tvaru dvojšroubovice. Protože nikde nebylo nalezeno hardwarové porovnání na základě tvaru, byla to zajímavá a neprozkoumaná cesta. Porovnávání by spočívalo v tom, že by se ze spojených nukleotidů vytvořily vektory, a ty by se následně porovnávaly mezi sebou.

Později byly nastudovány materiály o možnosti vícenásobného porovnání. Tento způsob je pro počítače velice výkonově náročný, čímž jsou jejich softwarová řešení pomalá a možná pouze do určitého počtu řetězců. Po podrobnějším nastudování vícenásobných porovnávacích algoritmů bylo rozhodnuto. Implementován bude jeden z vícenásobných algoritmů a poté umístěn do hardwaru.

Z těchto algoritmů byl pro implementaci vybrán algoritmus UPGMA pro jeho rychlost. Jeho přesnost byla zvýšena využitím dynamického programování. Přesný popis implementace je v kapitole č. 2.

5.1 Úvod

Algoritmus, který je implementován v této práci je ve své podstatě Needleman-Wunch algoritmus, který vypočítává globální porovnání. Tento algoritmus je využíván jako dvourozměrný - porovnává se vždy dvojice sekvencí (v této práci dvojice nukleotidů řetězců DNA). Po vypočtení skóre, charakterizujícího shodu dvojice řetězců, je skóre uloženo v paměti výsledků. Po vypočtení první dvojice řetězců se načte další z dvojic řetězců, dokud se tímto způsobem neporovnají všechny řetězce. Jakmile je vypočteno skóre každé z dvojic řetězců, může být sestaven evoluční strom. Sestavení těchto stromů je proces, který je shodný se sestavením stromu algoritmem UPGMA (viz kapitola 2.3.3). Tento algoritmus je spolehlivější pro delší řetězce, které se od sebe významně neliší.

Implementování celého algoritmu v hardwaru by bylo nevýhodné, protože závěrečné sestavení fylogenetického stromu již není možné paralelizovat, a navíc by taková logika v obvodu FPGA zabrala velké množství prostředků. Z tohoto důvodu je v hardwaru vhodné implementovat pouze akceleraci dynamického programování, které se jak již bylo zmíněno výše realizuje systolickým polem.

Implementace celého programového kódu je realizována genericky, to znamená, že pomocí editací hodnot v jednom souboru (tzv. package) lze měnit parametry implementovaného algoritmu a tak například místo porovnání DNA provádět porovnání proteinů nebo textů nebo lze pomocí parametrů změnit šířku pamětí a nebo hodnoty penalt a podobně.

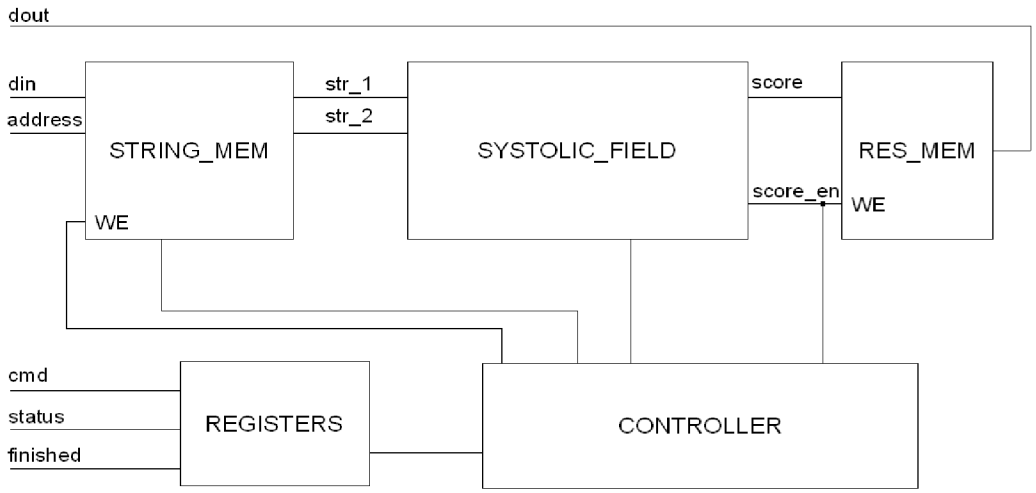
Obr. 14: Karta COMBO-PTM (převzato [48])

V FPGA čipu uloženém na desce COMBO-PTM (obrázek č. 14) se spustí výpočet skóre a přes konektor sběrnice PCI deska komunikuje s hostitelským zařízením, ve kterém se spustí program ovládající výpočet. Program slouží pro nahrání sekvence nukleotidů do paměti FPGA čipu a později (po skončení výpočtu) k přečtení obsahu paměti obsahující výsledná skóre. Program slouží také

pro zasílání příkazů typu spuštění, přerušení, znovu rozběhnutí a zastavení výpočtu. Program využívá konstrukcí zdrojových kódu a knihoven určených pro předmět Návrh externích adaptérů a vestavěných systémů (NAV) vyučovaném na Fakultě informačních technologií VUT Brno [48].

5.2 Struktura komponenty v FPGA

Obvod vytvořený v čipu FPGA je složen z několika nezávislých bloků. Tyto bloky lze chápat jako jednotlivé logické bloky výše popsaného algoritmu. Každý blok vykonává určitou část z implementovaného algoritmu: paměť porovnávaných řetězců, systolické pole, paměť výsledků, blok řízení a registry (viz obrázek č. 15).

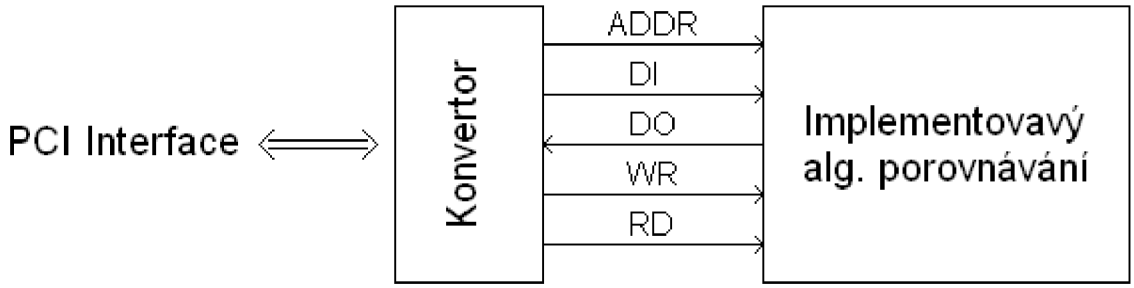


Obr. 15: Struktura komponenty

Tyto komponenty jsou pospojovány do celku, který je z vnějšího prostředí řízen příkazy zapisovanými do registru příkazů. Ovládající program může v případě potřeby přečíst stav, neboli zjistit kolik dvojic řetězců již bylo porovnáno či výpočet úplně ukončit. Pokud je výpočet všech dvojic řetězců hotov, je zároveň nastavena hodnota v registru značícího konečný stav výpočtu. A teprve poté si program může přečíst výsledky. Čtení a nebo zápis dat probíhá přes rozhraní PCI. Signály sběrnice PCI jsou převedeny v konvertoru signálů a poté připojeny k implementované komponentě provádějící porovnání řetězců (obrázek č. 16).

Konvertor signálů se skládá z čipu PLX a dále z komponent implementovaných v šabloně pro předmět NAV. Implementace této konverze bylo využito i při umisťování implementovaného

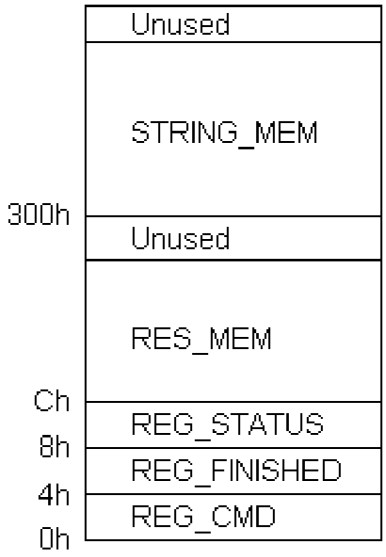
algoritmu do čipu FPGA karty COMBO-PTM, z toho důvodu, že implementace konverzní komponenty usnadňuje komunikaci se zařízeními v čipu FPGA. Užitím tohoto rozhraní, je možné k zařízení přistupovat pomocí operací čtení (signály ADDR, RD a DO) a zápisu dat (signály ADDR, WR, DI). Při čtení dat z implementované komponenty se data dostanou z komponenty přes konvertor signálů na sběrnici PCI a dále k ovládajícímu programu. Pokud jsou do čipu zapisována data, jsou zapsána do místa definovaného adresou v programu, a to tak, že data jsou po PCI sběrnici přenesena přes konvertor a pak dále přenesena na rozhraní implementované komponenty.



Obr. 16: Připojení implementované komponenty

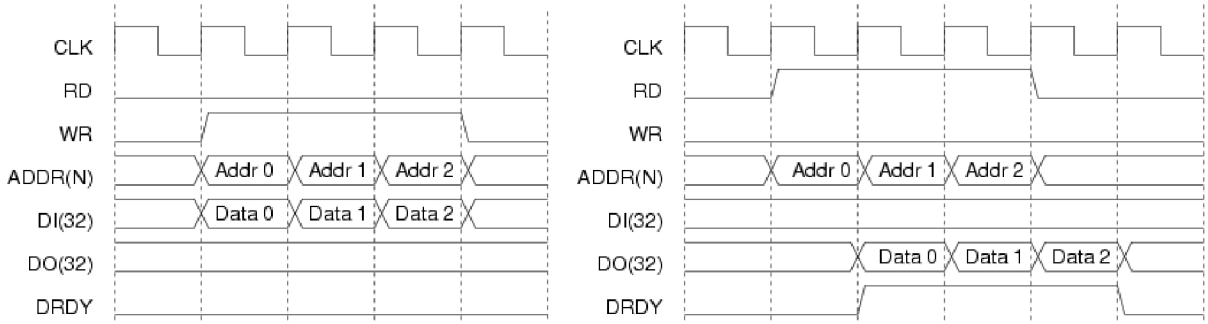
5.2.1 Adresování

Adresování entit v definovaném obvodu popisuje schéma paměťového prostoru (obrázek č. 17). Na tomto schématu si lze všimnout, že adresový prostor je zde rozdělen na dvě části. Do dolní části adresového prostoru jsou namapovány registry a rozlišení, zda se jedná o přístup k paměti řetězců a nebo paměti výsledných skóre. Namapování registrů je realizováno prostřednictvím dolních tří adres adresového prostoru, tedy prostřednictvím adres 0 až 8 hexadecimálně. Od 4. položky adresového prostoru, tedy adresy C hexadecimálně začíná adresový prostor určený pro paměť výsledků. Paměť zdrojových řetězců je zpřístupněna v paměti od adresy 300 hexadecimálně. Velikost a využití pamětí zdrojových sekvencí a pamětí výsledků závisí na délce porovnávaných sekvencí, jejich počtu, a také na typu porovnávaných řetězců (zda se jedná o řetězce DNA, proteiny a nebo obyčejný text).



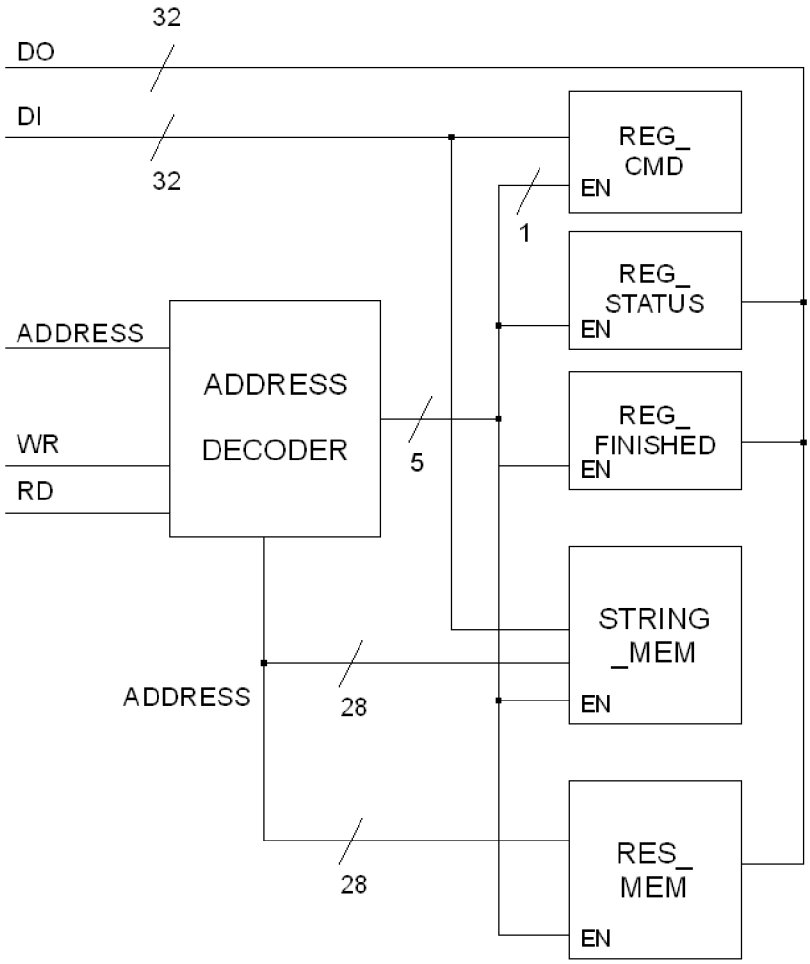
Obr. 17: Adresový prostor

Výběr místa z něhož bude čteno a nebo do něhož bude zapisováno řeší adresový dekodér, který je řešen tradičním způsobem (viz obrázek č. 19). Adresový dekodér přijme adresu a podle toho, co je obsaženo v adrese rozhodne, která z komponent dostane povolení k zápisu nebo ke čtení (podle vybrané komponenty). Pro povolení zápisové nebo čtecí operace slouží signál EN, který je veden z dekodéru do každého zařízení (v obrázku zaznačeno jako svazek signálů – kvůli přehlednosti). To, zda se má provádět zapisování a nebo čtení dat značí hodnota na vstupních signálech WR a RD. Vstup a výstup dat je 32 bitový, proto je potřeba při zápisu dat do registru příkazu zapsat jen spodní dva bity. Obráceně je při čtení registru statutu a registru příznaku ukončení třeba rozšířit na 32 bitů doplněním o hodnoty '0'. Protože dolní 4 bity adresového vektoru jsou použity pro výběr funkčního bloku, může být adresový prostor obou pamětí adresován maximálně 28 bity. (Pokud by byl program implementován v 64 bitovém procesoru, bylo by k dispozici o 32 bitů více.) Ukázka procesu čtení a



Obr. 18: Komunikační protokol přenosu dat (zápis a čtení; převzato z [48])

zápisu dat je na obrázku č. 18. Při čtení dat je signálem DRDY upozorněno zařízení, že data jsou připravena k přečtení.

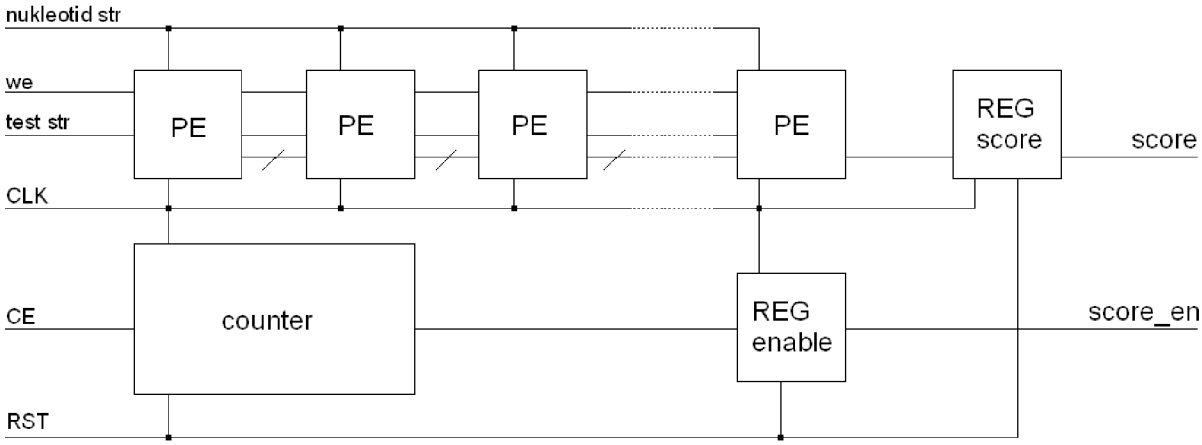


Obr. 19: Adresový dekodér

5.2.2 Implementace systolického pole

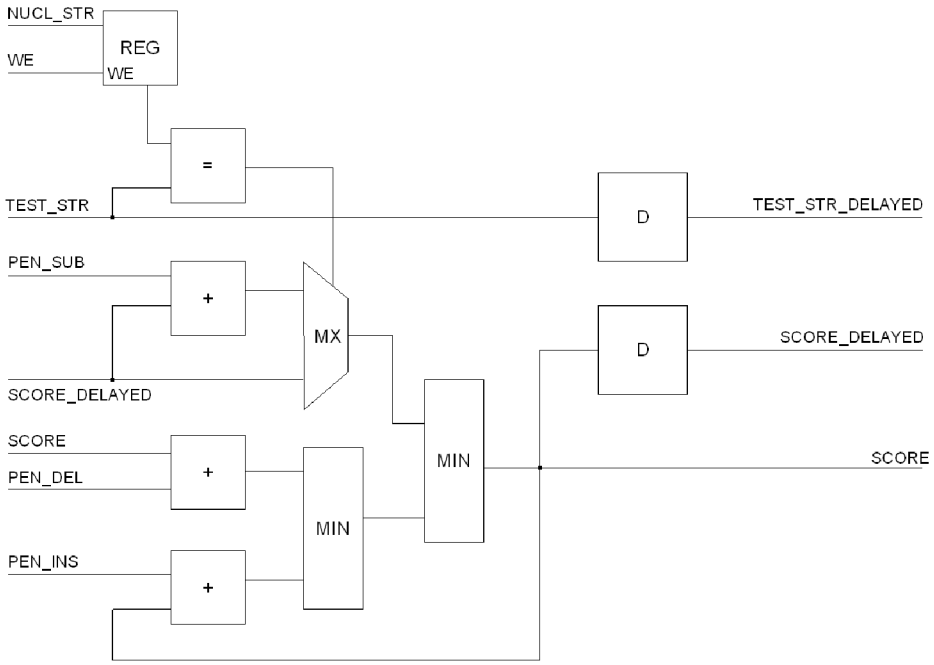
Systolické pole se skládá z pole výpočetních bloků (procesních elementů), čítače a registrů. Výpočetní bloky jsou stejné konstrukce, protože všechny vykonávají stejnou funkci. Každý z výpočetních bloků komunikuje jen s okolními výpočetními jednotkami. Sousední procesní elementy si předávají jeden z porovnávaných řetězců (v obrázku *test str*), vypočítané skóre (*score*) a skóre jež bylo vypočteno v předchozí výpočetní vlně (*score delayed*). Čítač slouží pro signalizaci platného výsledku, tedy platné hodnoty skóre porovnávaných řetězců. Do registru výsledného skóre je z posledního procesního elementu pravidelně přenášena hodnota skóre (*score*). Jakmile je nastavena hodnota v registru signalizujícího aktuálnost výsledku (*REG enable*) je povolen zápis do paměti a data se předávají dále.

Dvourozměrné systolické pole se konstruuje jako by mělo o jeden rozměr méně (podrobnosti viz kap. 2.5), proto ke konstrukci stačí sekvence procesních elementů seřazených sekvenčně za sebe. To, jak systolické pole vypadá schématicky, je znázorněno na obrázku č. 20. Počet procesních elementů v systolickém poli, bez překrývání dat, je shodný s počtem nukleotidů v řetězci.



Obr. 20: Schéma systolického pole

Jak bylo uvedeno již výše, systolické pole obsahuje procesní elementy, jejichž počet závisí na počtu porovnávaných nukleotidů v jednom průchodu a všechny elementy jsou svou vnitřní strukturou identické. Procesní elementy jsou sestaveny ze základních elementů, a to ze sčítaček, registrů, multiplexoru a jednotek pro výběr minimální hodnoty. Schéma jednoho z procesních elementů je uvedeno na obrázku č. 21.



Obr. 21: Procesní element

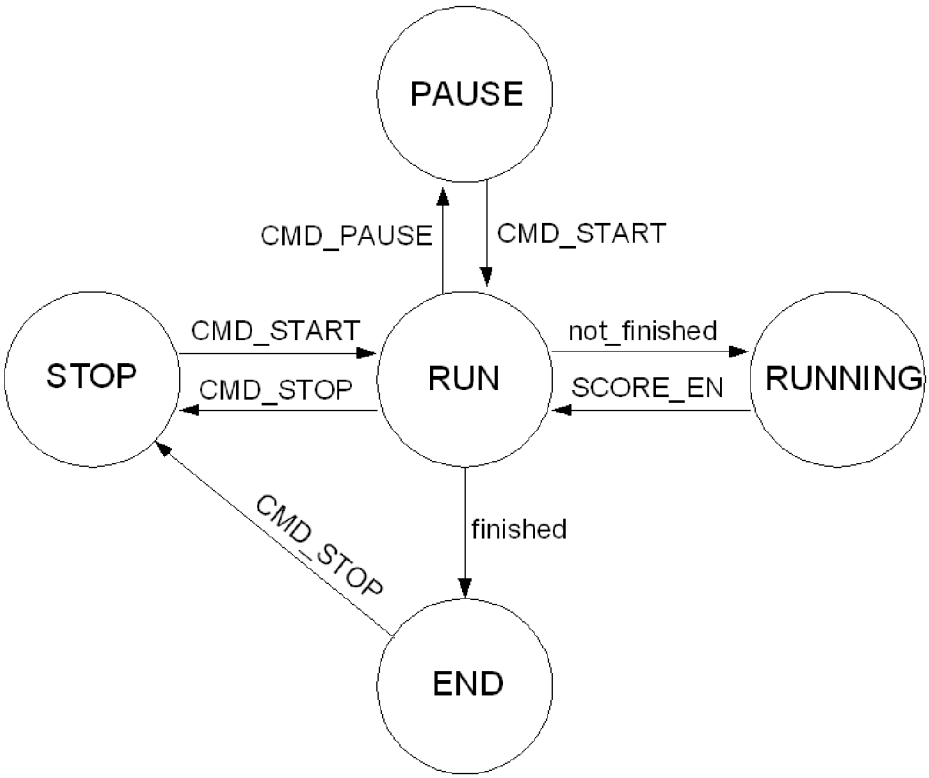
Do procesního elementu je přiveden nukleotid NUCL_STRING. Kód značící typ nukleotidu je uložen do registru. Zápis do tohoto registru je podmíněn signálem WE povolujícím zapsání do registru. Hodnota v registru je porovnána s hodnotou přivedenou druhým signálem označeným jako TEST_STRING. Podle výsledné hodnoty porovnání je řízen multiplexor, který přepíná mezi přímým přeposláním hodnoty z předchozí výpočetní vlny a inkrementací hodnoty skóre o penaltu za substituci. Skóre které bylo vypočteno předchozím procesním elementem je sečteno s penaltou za smazání znaku. A skóre, které bylo v předchozím běhu vypočteno stejným procesním elementem je sečteno s penaltou za vložení znaku. Z těchto dvou součtů je vybrána hodnota, která je nejmenší, a poté je porovnána s hodnotou, která je výstupem z multiplexoru. Následně je opět vybrána ta, která je z dvojice menší. Výsledná hodnota, je hodnotou, která je výstupem tohoto procesního elementu a posílá se do elementu následujícího. Tato výsledná hodnota bude zároveň použita také při následující výpočetní vlně (jako hodnota z horního řádku tabulky).

Jakmile jsou vypočteny všechny buňky porovnávací tabulky, neboli proběhly všechny výpočetní vlny systolického pole, je výpočet ukončen. K ukončení dojde po $(n + m - 1)$ výpočetních vlnách, kde m, n jsou délky porovnávaných řetězců. Při ukončení je posláno skóre z posledního procesního elementu do paměti výsledků a zároveň je generována hodnota povolující zápis do této paměti.

5.2.3 Řadič

Toto zařízení řídí běh celého hardwarového akcelerátoru, a zároveň jsou tomuto kontroléru zasílány příkazy ze softwaru. Kontrolér je vlastně konečný automat (přesněji automat Moorova typu) o pěti stavech, řízení řadičem je znázorněno na obrázku č. 22. Schéma řízení má počáteční stav „STOP“ a koncový stav „END“. Pojmenování stavů je odvozeno od vnitřní funkce algoritmu. A sice ve stavu „STOP“ se nahrají data do paměti, stav „RUN“ značí běh zařízení, „RUNNING“ zase vypočítávání jednoho ze skóre, stav „PAUSE“ charakterizuje stav, ve kterém je pozastaveno vypočítávání, je to stav, ze kterého může zařízení pokračovat ve svém běhu. Naopak stav „END“ je posledním stavem výpočtu, ve kterém se zařízení nachází jakmile je výpočet ukončen.

Z počátečního stavu („STOP“) zakresleného schématu se zařízení může dostat jen do stavu běhu („RUN“), a to pomocí příkazu CMD_START. Jakmile je zařízení rozběhnuto, pravidelně přechází do stavu, ve kterém vypočítává (tedy stav „RUNNING“). Po vypočtení skóre páru řetězců je výpočet ukončen, zapíší se data do paměti výsledků a zařízení přejde do stavu „RUN“. V tomto stavu automat inkrementuje adresy porovnávaných řetězců, a poté znovu spustí výpočet. Tento postup se opakuje dokud nepřijde příkaz CMD_PAUSE, CMD_STOP a nebo dokud nejsou vypočtena všechna skóre všech dvojic řetězců. Příkaz CMD_PAUSE přeruší výpočet a zařízení přejde do stavu, kdy je možné číst výsledky porovnávání. Software se doví kolik párů řetězců bylo porovnáno díky registru s uloženým stavem, tedy uloženým počtem porovnaných řetězců. Software, tedy může přečíst tolik výsledných skóre kolik je uvedeno v registru. Ze stavu „PAUSE“ je možné se vrátit do stavu běhu „RUN“ a pokračovat ve vypočítávání. Pokud se uživatel rozhodne, že výpočet trvá dlouho a zároveň jej nezajímají spočtené výsledky, může příkazem CMD_STOP výpočet přerušit a přepnout zařízení do počátečního stavu. Číst výsledky je kromě stavu „PAUSE“ možné i ve stavu „END“, do kterého zařízení přejde pokud jsou již všechny řetězce porovnány. Jakmile je zařízení v tomto konečném stavu, pak lze z akceleračního hardware číst všechny výsledky porovnání a tedy zjišťování počtu porovnaných párů je zbytečné. Software může přečíst všech $(n*(n-1))/2$ výsledků, kde n je počet porovnávaných řetězců. O tom, že se zařízení nachází ve stavu „END“ je software informován, pokud si přečte hodnotu registru REG_FINISHED. Z koncového stavu lze přejít do počátečního stavu příkazem CMD_STOP. V automatu, z důvodu přehlednosti, nejsou znázorněny hrany, které způsobují cyklení nad shodným uzlem. Dále platí, že signál resetující zařízení nastaví všechny prvky do počátečního stavu a automat řízení stejně tak, tedy do stavu „STOP“.



Obr. 22: Automat řízení

Příkazy, které se požívají v obslužném programu, ale které je možné použít při přímém zápisu na adresu zařízení jsou uvedeny v následujícím výčtu:

příkaz	**kód**
CMD_STOP	00
CMD_START	01
CMD_PAUSE	10

Pokud je třeba zařízení zadat příkaz, například příkaz, který zařízení spustí, je třeba na adresu 0 h zapsat data končící posledními dvěma bity požadovaným kódem, tedy „01“. Zapíšeme-li na adresu 0 h data 1 h (nebo i 78A1 h) zařízení se spustí a začne vypočítávat jednotlivá skóre.

Pořadí, v němž jsou řetězce porovnávány charakterizuje následující schéma. Čísla v tomto schématu značí identifikátory řetězců a *num_strings* počet porovnávaných řetězců.

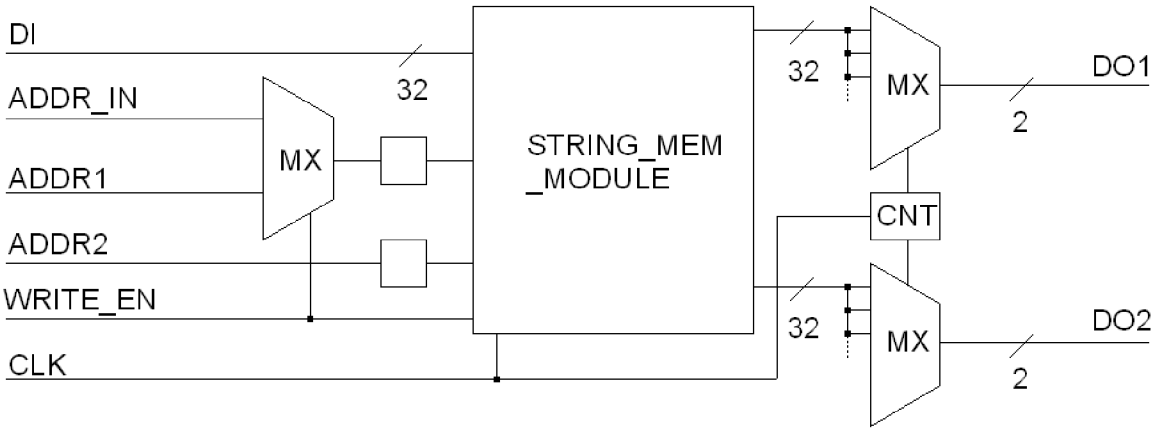
číslo řetězce	porovnávaný řetězec
0	1, 2, 3, 4 … (num_strings – 1)
1	2, 3, 4 … (num_strings – 1)
2	3, 4 … (num_strings – 1)
3	4 … (num_strings – 1)
4	… (num_strings – 1)
…	…
num_strings – 2	(num_strings – 1)

5.2.4 Paměť řetězců

Tato paměť slouží k uložení řetězců, jež mají být porovnány. Velikost této paměti závisí na počtu porovnávaných řetězců, na délce jednotlivých řetězců a na typu porovnávaných dat. Například porovnávání 8 řetězců DNA o délce 16 nukleotidů zabere v paměti 32 Byte. Je tomu tak proto, že nukleotidy lze zakódovat po dvou bitech, proto jeden řetězec zabere 32 bitů. A tedy 8 řetězců zabere úměrně více, což odpovídá výše uvedeným 32 Byte. Data jsou v paměti porovnána po 32 bitech. Pokud by sekvence z předchozího příkladu měla o jeden nukleotid více, pak by v paměti díky porovnání zabírala 64 Byte.

Tato paměť je řešena jako dvou-portová paměť s asynchronním čtením a synchronním zápisem. Dvou-portová znamená, že je u ní možné v jednom časovém okamžiku zapisovat a nebo najednou číst dvoje data. Této vlastnosti je u paměti při implementaci vybraného algoritmu třeba, protože je potřeba v jednom okamžiku porovnávat vždy dva nukleotidy, obecně prvky řetězce. Data jsou z paměti zdrojových řetězců čtena po dvou bitech odzadu, tj. od nejnižšího nukleotidu. Např. z paměti obsahující řetězec ATTG je do systolického pole nejprve načten nukleotid G, poté T, dále T a jako poslední A.

Adresa číslo jedna je multiplexovaná, to znamená, že pomocí třetího signálu se rozhoduje která adresa bude vstupem. To záleží na tom, zda je povolen zápis do pamětí či nikoliv. Pokud se do pamětí zapisuje více než 32 bitů je potřeba zajistit automatickou inkrementaci adresy, to je znázorněno v obrázku č. 23 čtvercovou komponentou bez popisku. Zápis do paměti je možný pouze s hodnotou 1 na signálu povolujícího zápis – WE. Tento signál je roven 1 pouze ve stavu „STOP“ řídícího automatu. Data je potřeba do následující komponenty – systolického pole – posílat po jednotlivých nukleotidech, proto je potřeba data na výstupu multiplexovat po dvou bitech (šířka vysílaných dat je určena typem porovnávaných dat).

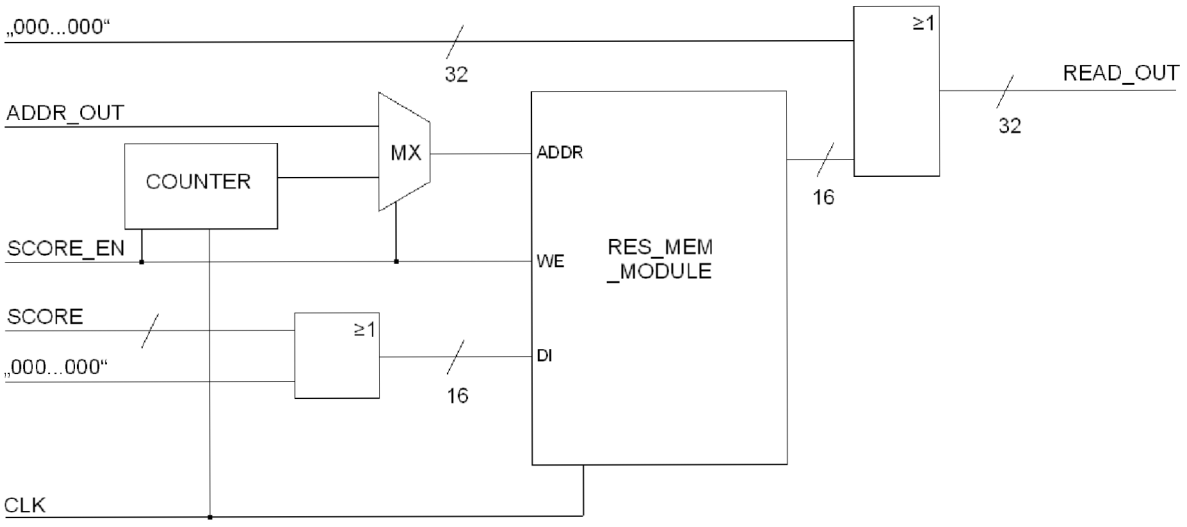


Obr. 23: Paměť porovnávaných řetězců

5.2.5 Paměť výsledků porovnání

Počet položek této paměti je definován počtem porovnávaných řetězců. Každá z položek této paměti je zarovnána na násobky 16 bitů. Do každé z položek je potřeba uložit výsledné skóre, které může nabýt maximální hodnoty, kterou lze vypočítat tak, že vynásobíme počet nukleotidů (prvků řetězce) s nejnižší z penalt (nejnižší z penalt je dostačující, protože při výběru vybíráme minimum).

K realizaci tohoto typu paměti je dostačující jednoportová varianta. S tímto typem pamětí je nutné multiplexovat adresu a rozhodovat tak, zda se jedná o adresu určující místo zápisu dat (adresa generovaná čítačem – viz. obrázek č. 24) a nebo místo odkud budou data čtena (adresa ADDR_OUT). Data paměti a tedy skóre porovnání dvojic řetězců je možné číst pouze ve stavu řadiče „PAUSE“ a „END“. Zápis hodnot je do paměti prováděn jen ve stavu „RUNNING“. Jakmile je výpočet hotov, je vystavená hodnota skóre povolena k zápisu nastaveným signálem SCORE_EN. Při zapisování dat do paměti, je z důvodu zarovnaných 16 bitových hodnot potřeba vstupní skóre rozšířit na požadovaných 16 bitů. Jelikož jsou výstupy komponenty na 32 bitové je potřeba čtené skóre rozšířit na stejnou hodnotu, pro rozšíření signálů slouží hradla OR.



Obr. 24: Paměť výsledků

6 Experimentální ověření

Funkčnost systému byla ověřena na vybraných sekvenčních datech. V následující části bude popsáno několik vícenásobných porovnání krátkých DNA sekvencí s využitím implementovaného algoritmu.

6.1 Demonstrace na příkladech

6.1.1 Příklad 1

Prvním příkladem porovnání budou řetězce, které se vzájemně liší jen s minimálním rozdílem avšak náhodně. Pro prezentaci takových typů řetězců byly vybrány následující čtyři sekvence:

A: CTCG

B: TCGA

C: CTGC

D: GTTG

Pokud tyto sekvence překódujeme podle tabulky č. 2, pak dostaneme binární sekvence uvedené ve třetím sloupci tabulky č. 3.

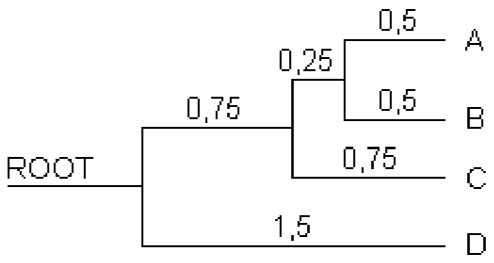
A	00
T	01
C	10
G	11

Tabulka 2: Kódování nukleotidů

	Řetězec nukleotidů	**Binární řetězec**
A sekvence	CTCG	10011011
B sekvence	TCGA	01101100
C sekvence	CTGC	10011110
D sekvence	GTTG	11010111

Tabulka 3: Zakódovaná sekvence

Pokud tyto sekvence vložíme do implementovaného systému dostaneme několik skóre porovnání. Na jejich základě lze poté sestavit evoluční strom. Srovnáním výše uvedených sekvencí dostaneme strom uvedený na obrázku č. 25.



Obr. 25: Evoluční strom

6.1.2 Příklad 2

V tomto příkladě bude prezentováno vícenásobné porovnání rozdílných sekvencí. Srovnávanými sekvencemi budou následující:

A: ATCG

B: CCCC

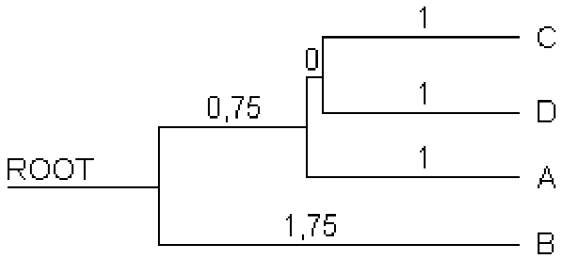
C: TGGT

D: TGAA

Řetězce jsou navzájem různé, ale jeden z nich je výrazně nepodobný ostatním. Po zakódování výše uvedených řetězců dostaneme binární sekvence uvedené v tabulce č. 4.

	Řetězec nukleotidů	**Binární řetězec**
A sekvence	ATCG	00011011
B sekvence	CCCC	10101010
C sekvence	TGGT	01111101
D sekvence	TGAA	01110000

Tabulka 4: Porovnávané sekvence



Obr. 26: Evoluční strom

Odlišnost řetězce B je v evolučním stromu znázorněném na obrázku č. 26 samostatnou větví stromu. Zajímavý jev nastal u zbývajících tří řetězců, ty jsou díky své vzájemné podobnosti ohodnoceny stejnou mírou shody, a proto také vzdálenost od společného předka pro řetězce A, C, D je shodná – rovná '1'.

6.1.3 Příklad 3

Tento příklad je zaměřen na porovnání podobných řetězců. Mezi sekvencemi jsou dva řetězce, které jsou identické. Vybranými řetězci jsou následující:

A: TAGC

B: AAGC

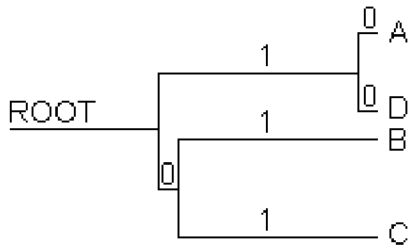
C: ACGC

D: TAGC

Přepis značek nukleotidů do binárního tvaru je uveden v tabulce č. 5. Evolučního strom, tedy výsledné vícenásobné porovnání řetězců, je znázorněn na obrázku č. 27. Z tvaru fylogenetického stromu je patrné, že řetězce A, D byly rozpoznány jako shodné, a proto vzdálenost k jejich společnému předku je nulová. Podobnost tří neidentických řetězců je zřejmá na první pohled, což dokládá také tvar evolučního stromu.

	Řetězec nukleotidů	**Binární řetězec**
A sekvence	TAGC	01001110
B sekvence	AAGC	00001110
C sekvence	ACGC	00101110
D sekvence	TAGC	01001110

Tabulka 5: Porovnávané sekvence



Obr. 27: Evoluční strom

6.2 Syntéza

Implementovaný algoritmus vícenásobného porovnání byl syntetizován (s využitím šablony k předmětu NAV) programem *Precision Synthesis 2005b.110.* Umístění bylo provedeno do čipu Spartan3 - X3S200 od firmy Xilinx. Při testování maximální velikosti systolického pole bylo zjištěno, že maximální velikost porovnávaných řetězců je 50 nukleotidů (viz tabulka č. 6). Počet porovnávaných řetězců je omezen velikostí pamětí. Hodnoty vypočítaných skóre v kartě COMBO-PTM se neshodují s očekávanými výsledky.

Délka řetězců	**Počet obsazených**	**Procentuálně**
4 nukleotidy	286	14,9 %
8 nukleotidů	339	17,6 %
16 nukleotidů	553	28,8 %
32 nukleotidů	1091	56,8 %
50 nukleotidů	1890	98,4 %
51 nukleotidů	1925	100,2 %

Tabulka 6.: Využití FPGA čipu

6.3 Diskuse výsledků

V podkapitolách 6.1.1 až 6.1.3 byly uvedeny příklady vícenásobného porovnání DNA řetězců implementovaného algoritmu. Z výsledků je zřejmé, že tento algoritmus je vhodný pro podobné sekvence, v nichž se lépe porovnávají a určují vzájemné podobnosti.

Výsledky potvrzují tvrzení, že algoritmus není vhodný pro porovnání významně odlišných sekvencí. Ale díky své rychlosti je vhodný pro vymezení určitého okruhu řetězců, nad nimiž je poté možné provést přesnější porovnání algoritmem CLUSTAL a nebo přímo některou z metod dynamického programování.

Pro porovnání není možné nalézt jedno správné porovnání, kromě řešení, která jsou triviální [40]. To, že pro každé netriviální řešení existuje více možností porovnání, je zřejmé a je dáno množstvím evolučních změn provedených na řetězci. To zapříčiňuje, že vícenásobné porovnání může mít více evolučních stromů.

7 Závěr

V této práci byly popsány algoritmy pro párové porovnání DNA řetězců. U jednotlivých algoritmů byla také hodnocena jejich výkonnost a přesnost. Dále byly popsány algoritmy pro vícenásobné porovnání, a to jak algoritmy u nichž je prioritní přesnost, tak i ty, u nichž je přední jejich rychlost. Byla také popsána technologie a struktura FPGA. Teoretická část této práce byla vytvořena jako součást semestrálního projektu a byla dále rozvíjena. Praktický rozvoj této práce je zachycen ve druhé části. V této části je popsán způsob implementace s podrobnějším popisem konstrukcí jednotlivých bloků implementovaného algoritmu.

Funkčnost algoritmu a správnost vypočtených hodnot byla ověřena simulacemi. Na základě skóre porovnaných řetězců, získaných simulacemi, bylo prezentováno i několik ilustračních příkladů. Implementace byla syntetizována a akcelerace na čipu je funkční, výsledky skóre se však neshodují s výsledky získaných simulacemi.

Další možný vývoj této práce by mohla být implementace další logiky pro umožnění porovnání libovolně dlouhých řetězců. Pro případnou akceleraci rychlosti, by mohlo být vhodné převedení algoritmu na sekvenční.

Literatura

[1] Advanced Dynamic Programming Tutorial, http://www.sbc.su.se/~pjk/molbioinfo2001/dynprog/adv_dynamic

[2] Big Brother Awards: nové technologie databáze DNA, http://www.bigbrotherawards.cz/nove_technologie_databaze_dna.html

[3] Why BLAST?, http://www.ncbi.nlm.nih.gov/Class/NAWBIS/Modules/Similarity/simsrch8.html

[4] But Not BLAST, http://www.ncbi.nlm.nih.gov/Class/NAWBIS/Modules/Similarity/simsrch8c4.html

[5] Smith-Waterman (S-W), http://www.ncbi.nlm.nih.gov/Class/NAWBIS/Modules/Similarity/simsrch8b.html

[6] Similarity Searching Slide Template, http://www.ncbi.nlm.nih.gov/Class/NAWBIS/Modules/Similarity/simsrch8a.html

[7] Pairwise local/global alignment, http://www.ebi.ac.uk/2can/tutorials/protein/align.html

[8] BLAST, http://blast.wustl.edu/doc/blast.html

[9] BLAST and FASTA benchmarks, http://www.beowulf.org/archive/2002-April/006743.html

[10] DNA, http://www.spsmvbr.cz/cesky/os_stranky/jedlicka/dna/dna.html

[11] Dynamic Programming, http://www.sbc.su.se/~pjk/molbioinfo2001/dynprog/dynamic.html

[12] Needleman-Wunsch algorithm, http://en.wikipedia.org/wiki/Needleman-Wunsch_algorithm

[13] Protein Sequence Alignment and Database Scanning, http://www.compbio.dundee.ac.uk/ftp/preprints/review93/review93.pdf

[14] Sequence alignment, http://en.wikipedia.org/wiki/Sequence_alignment

[15] Smith-Waterman algorithm, http://www.maths.tcd.ie/~lily/pres2/sld009.htm

[16] Smith-Waterman algorithm, http://en.wikipedia.org/wiki/Smith-Waterman_algorithm

[17] Existují principiálně nové možnosti identifikace osob, věcí, případně i zvířat?, http://www.mvcr.cz/casopisy/kriminalistika/2001/01_02/suchanek.html

[18] Bioinformatics and Computational Molecular Biology Algorithms and Hardware, http://coen.boisestate.edu/ssmith/biohw/SWEquations/SWEquations.htm

[19] Studijní opora předmětu BIO, https://www.fit.vutbr.cz/study/courses/BIO/private/BIO_Studijni_opora.pdf

[20] Michel-Jean Claverie, Cedric Notredame; *Bioinformatics for dummies*, For Dummies, 2003, s. 247-314.

[21] Dan E. Krane, Michael L. Raymer; *Fundamental Concepts of Bioinformatics*, The Benjamin/Cummings Publishing Company, 2003, s. 1-96.

[22] FASTA, http://en.wikipedia.org/wiki/FASTA

[23] Princip sekvencování DNA, http://orion.sci.muni.cz/kgmb/bioinformat/princip_seq.pdf

[24] Sequence alignment, http://en.wikipedia.org/wiki/Sequence_alignment

[25] Clustal, http://en.wikipedia.org/wiki/Clustal

[26] CLUSTAL W Algorithm, http://bimas.dcrt.nih.gov/clustalw/clustalw.html

[27] UPGMA, http://en.wikipedia.org/wiki/UPGMA

[28] UPGMA, http://pubmlst.org/software/analysis/start/manual/upgma.shtml

[29] UPGMA, http://www.icp.ucl.ac.be/~opperd/private/upgma.html

[30] Pokročilé číslicové systémy - Úvod, https://www.fit.vutbr.cz/study/courses/PCS/private/prednasky/01_uvod/01_uvod.pdf

[31] FPGAs in Perspective, https://www.fit.vutbr.cz/study/courses/PCS/private/prednasky/03_technologie_fpga/fpl2003.pdf

[32] Pokročilé číslicové systémy - Technologie FPGA https://www.fit.vutbr.cz/study/courses/PCS/private/prednasky/03_technologie_fpga/fpga_techn.pdf

[33] Image:Dna-split.png, http://commons.wikimedia.org/wiki/Image:Dna-split.png

[34] Image:Codis profile.jpg, http://commons.wikimedia.org/wiki/Image:Codis_profile.jpg

[35] Protein sequence comparison and Protein evolution, http://people.virginia.edu/~wrp/papers/ismb2000.pdf

[36] Similarity: Smith-Waterman, http://www.med.nyu.edu/rcr/rcr/course/sim-sw.html

[37] UPGMA, http://www.cs.ecu.edu/~hochberg/spring2006/UPGMA.gif

[38] FPGA, https://www.fit.vutbr.cz/study/courses/PCS/private/prednasky/03_technologie_fpga/fpga_techn.pdf

[39] Gene Matching Using JBitsc, http://www.ccm.ece.vt.edu/hokiegene/papers/GeneMatching.pdf

[40] Multiple Alignments, http://www.math.nus.edu.sg/~matky/classes/MA5264/MA5264_11_Multiple_Alignments.pdf

[41] Multiple Alignment, http://journal-ci.csse.monash.edu.au/ci/vol04/mulali/mulali.html

[42] Algorithmische Bioinformatik, http://www.bio.ifi.lmu.de/lehre/WS2004/VLG_Algo_2/Material/041111_VII_1_MultAli.pdf

[43] Multiple Sequence Alignments, http://www.bisb.uni-bayreuth.de/People/ullmann/Lectures-WS05/lecture-muti-align.pdf

[44] Multiple Sequence Alignment, http://www.cs.wfu.edu/~burg/Courses/Spring04/CSC391-691/course-materials/MultipleSequenceAlignment.ppt

[45] Dydel, S. , Bala, P.: Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices, N. Copernicus University, Torun.

[46] Masuno S., Maruyama T., Yamaguchi Y., Konagaya A.: Multidimensional Dynamic Programming for Homology Search on Distributed Systems, University of Tsukuba, RIKEN Genomic Sciences Center, Tsukuba, Yokohama.

[47] Field-programmable gate array, http://en.wikipedia.org/wiki/FPGA

[48] Handbook, https://www.fit.vutbr.cz/study/courses/NAV/private/handbook/index.html

[49] Prezentace na ustavu UPSY 16.12.2005, http://merlin.fit.vutbr.cz/DNA/doc/upsy-16.12.2005.ppt

[50] COMBO6, http://www.liberouter.org/card_combo6.php

[51] COMBO-PTM, http://www.liberouter.org/card_comboptm.php

Seznam zkratek

deoxyribonukleová kyselina	DNA
adenin	A
thymin	T
cytosin	C
guanin	G
procesní element	PE
Byte update/sekundu	BUDps
Configurable Logic Block	CLB
Digital Clock Manager	DCM
Hardware Description Language	HDL
Návrh externích adaptérů a vestavěných systémů	NAV

Seznam příloh

Příloha 1. CD – obsahující elektronickou verzi práce, zdrojové kódy a dokumentaci k programům (programu pro ovládání akcelerátoru a VHDL kódu)