VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

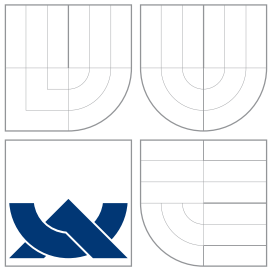
AKCELERACE ALGORITMŮ PRO POROVNÁNÍ BIOLOGICKÝCH SEKVENCÍ S VYUŽITÍM FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

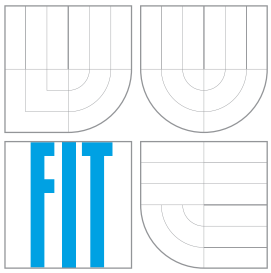
AUTOR PRÁCE
AUTHOR

PATRIK BECK

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

AKCELERACE ALGORITMŮ PRO POROVNÁNÍ BIOLOGICKÝCH SEKVENCÍ S VYUŽITÍM FPGA

AKCELERATION OF ALGORITHMS FOR APPROXIMATE STRING MATCHING USING FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PATRIK BECK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MARTÍNEK

BRNO 2007

Abstrakt

Táto práca sa zaoberá implementáciou hardwarového zariadenia, ktoré porovnáva biologické sekvencie. Pri porovnávaní využíva algoritmy Smith-Waterman a Needleman-Wunsch. Zariadenie slúži ako akcelerátor bioinformatických algoritmov na vyššej úrovni. Príkladom využitia može byť analýza ľudského genómu, porovnávanie proteínu s databázou, odhaľovanie dedičných informácií. Dosiahnuté zrýchlenie sa v závislostí na danej úlohe, oproti bežnému PC pohybuje v niekoľkých rádoch.

Klíčová slova

FPGA, približné porovnávanie reťazcov

Abstract

This work describes implementation of hardware device for approximate string matching of biological sequences. Matchnig is performed using Smith-Waterman and Needleman-Wunsch algorithms. Device can be used as an accelerator for bioinformatics algorithms on higher level. This accelerator can be used for human genome analysis, matching proteins against a database, revealing inheritance information. Depending on task character, the acceleration speed up achieves several orders of magniture in comparison with conventional computers.

Keywords

FPGA, approximate string, matching

Citace

Patrik Beck: Akcelerace algoritmů pro porovnání biologických sekvencí s využitím FPGA, bakalářská práce, Brno, FIT VUT v Brně, 2007

Akcelerace algoritmů pro porovnání biologických sekvencí s využitím FPGA

Prohlášení

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

.....
Patrik Beck
10. května 2007

Poděkování

Ďakujem riešiteľom projektu liberouter za odbornú pomoc.

Zadanie

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 biologických sekvencí.
4. Na základě předchozích dvou bodů navrhněte vhodnou hardwarovou architekturu pro urychlení algoritmu hledání podobnosti dvou biologických sekvencí.
5. Proveďte implementaci navrženého řešení v jazyce VHDL a jeho funkci ověřte simulací.
6. Realizujte funkční prototyp navržené architektury a ověřte její správnost na kartě COMBO6X.
7. V závěru diskutujte vlastnosti Vaší implementace a možnosti dalšího pokračování projektu.

Obsah

Kapitola 1

Úvod

Bioinformatika je rýchlo sa rozvíjajúca veda, ktorá vznikla spojením biológie a informatiky. Zaoberá sa realizáciou náročných výpočtov, ktoré sú potrebné na poli biológie. V tejto práci je popísaný návrh a implementácia hardwarového urýchlenia algoritmov *Smith-Waterman* a *Needleman-Wunsch*, ktoré sa používajú pre *približné porovnávanie reťazcov*.

Biológia skúma deje v živých organizmoch. Interakcie medzi jednotlivými proteínmi sú skúmané až na molekulárnej úrovni. Proteíny v živej bunke vznikajú postupnou *syntézou* z genetického kódu. Proteíny riadia všetky funkcie v bunke, rovnako katalizujú vznik iných proteínov. Biológovia dnes majú k dispozícii technológie k získavaniu genetického kódu zo živých organizmov. Tento proces sa nazýva sekvencovanie.

Technológia genetického sekvencovania sa rýchlo rozvíja. Vedci majú dnes k dispozícii databázy s veľkým množstvom DNA sekvencií. Sekvencie je nutné klasifikovať, preto sú často medzi sebou porovnávané. Vyhľadávajú sa funkčné podreťazce zvané gény, ktoré sú schopné ovplyvniť tvorbu proteínov. DNA sekvencie sú opakovane porovnávané, vyhľadávajú sa v nich podreťazce za účelom hľadania napríklad dedičných chorôb, prípadne odhaľovania evolučného stromu. Výpočty, ktoré biológia vyžaduje sú často pamäťovo, ale hlavne časovo veľmi náročné.

Pri analýze genetického kódu sú bežné metódy hľadania exaktnej zhody nedostačujúce. Genetický kód je vplyvom mutácií a dedenia na mnohých miestach pozmenený. Takto aj takmer podobné podreťazce môžu mať miestami znak pozmenený, chýbajúci, prípadne sa znaky vymenia. Preto sa používa *približné porovnávanie* reťazcov. V praxy sa využíva viacero algoritmov vykonávajúcih približné porovnávanie. Vyššie spomenuté algorimy *Needleman-Wunsch* a *Smith-Waterman* majú ale kvadratickú zložitosť. Ľudský genóm má niekoľko miliónov báz. Približné porovnánie dvoch ľudských genómov algorimom Needleman-Wunsch je v súčastnosti na štandardnom PC prakticky nereálne. Pre realizáciu približného porovnávania sa preto využívajú buď iné prístupy, napríklad heuristické algoritmy *FASTA* a *BLAST*(ktoré však nezaručujú optimálne zarovnanie), alebo sa vyššie zmienené algoritmy realizujú na gridoch[1], prípadne sú *hardwarovo urýchľované*.

Úlohy súvisiace s porovnávaním môžu mať rôzny charakter, nejedná sa vždy o porovnanie celých dvoch genómov. Može sa napríklad vyhľadávať krátky podreťazec v celom genóme. Hľadajú sa miesta, ktoré sa najviac podobajú na daný podreťazec. Prípadne sa konkrétny kratší reťazec hľadá v databáze. Môže sa jednať aj o hľadanie korelácií akýchkoľvek podreťazcov v genóme. Úlohy sa takisto môžu líšiť aj vo veľkosti použitej abecedy, na kódovanie znaku DNA reťazca stačia 2 bity, avšak na kódovanie proteínu je ich

[1] veľké množstvo počítačov v sieti riešiacich tú istú úlohu

potrebných 5. Preto je tento hardwarový urýchlovač parametrizovateľný a prispôsobiteľný veľkej skupine úloh.

Hardwarový urýchlovač nemusí vždy realizovať danú úlohu celú. Môže byť využitý pre urýchlenie približných porovnaní pre algoritmy na vyššej vrstve. Napríklad pri hľadaní korelácií podreťazcov v jednom genóme, kedy sa podreťazce porovnávajú medzi sebov. Rovnako pri urýchlovaní simulácií, napríklad simulácia *PCR*(Polymerase chain reaction). Metóda PCR enzymaticky amplifikuje(replikuje) úsek DNA reťazca. Pri simulácii je veľmi často potrebné približné porovnanie podreťazcov.

Existuje viacero prácí zaoberajúcich sa hardwarovým urýchlovaním približného porovnávania reťazcov. Avšak ich širšiemu nasadeniu bráni ich úzka špecializácia na konkrétnu úlohu. Napríklad práca pánov Yu, Kwong, Lee a Leng [11] je zameraná na to, aby čo najoptimálnejšie realizovala porovnávanie DNA sekvencií, pričom vylučuje použitie inej veľkosti abecedy. Ich realizácia takisto využíva FPGA programovateľné pole. Snahou riešenia, ktoré prezentuje táto práca je široká parametrizovatelnosť designu [2], aby bolo možné dosiahnuť čo najvačšie zrýchlenie a zároveň design udržať vo forme šablony, ktorá bude prispôsobiteľná čo najviac úlohám. Zhrnutím doterajších prác sa zaobera samostatná kapitola 3.

V nasledujúcej kapitole 2 je vysvetlený princíp urýchľovaných algoritmov Smith-Waterman a Needleman-Wunsch. Takisto je popísaná paralelizovateľnosť a rozdelenie na podúlohy. Kapitola 3 je venovaná podobným prácam, sú zhrnuté dosiahnuté výsledky na poli urýchlovania približného porovnávania reťazcov, ako aj princípy a rôzne prístupy k riešeniu tejto problematiky. V kapitole 4 je uvedený princíp funkcie hardwarovej implementácie, ktorej je táto práca venovaná. V kapitole 5 sú uvedené dosiahnuté výsledky. Klady a zápory tohoto riešenia sú diskutované v kapitole 4.6.

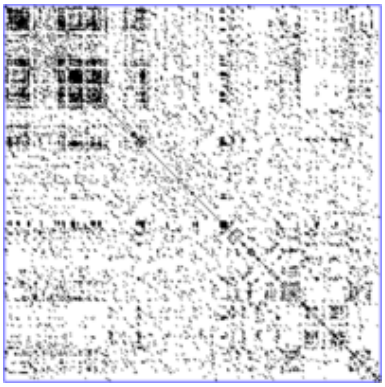
Kapitola 2

Približné porovnávanie reťazcov

Táto kapitola popisuje bioinformatický prístup k porovnávaniu reťazcov. Sú prezentované algoritmy *Smith-Waterman* a *Needleman-Wunsch*, ďalej algoritmy *BLAST* a *FASTA*. Tieto algoritmy patria medzi základné nástroje, ktoré sa používajú pre klasifikáciu biologických sekvencií. Hardwarový akcelerátor prezentovaný v tejto práci počíta pomocou algoritmov Needleman-Wunsch a Smith-Waterman. Akým spôsobom sú implementované, je vysvetlené až v následnej kapitole 4. Kapitola čerpá z knihy Fundamental Concepts of Bioinformatics [7].

Biológia potrebuje pri analýze nových sekvencií zistiť, či sa už nachádzajú niekde v databáze. Alebo či existuje daná sekvencia v genóme. Toto prehľadávanie komplikuje fakt, že biologické sekvencie sú často modifikované vplyvom mutácie.

Jedným zo spôsobov, ako zistiť či sa sekvencie na niektorých miestach podobajú je výpočet *bodovej matice(dot matrix)*. Znaky obidvoch sekvencií sú nanesené na kraje matice, na ľavú stranu a na horný okraj. Na mieste, kde sú znaky zhodné je bodka. Diagonálne usporiadanie bodov ukazuje podobnosť reťazcov v daných miestach(príklad na obrázku 2.1). Táto metóda ma nevýhodu veľkého šumu. Napríklad pri porovnávaní DNA reťazcov existujú iba 4 znaky. Čo znamená veľa náhodných zhodných znakov a teda veľký šum.



Obrázok 2.1: Bodová matica(dot matrix)

V praxi sa využívajú skôr metódy, ktoré sú schopné určiť zarovnanie dvoch sekvencií a číselne ho ohodnotiť.

2.1 Výpočet globálneho zarovnania

Zarovnanie sekvencií je párovanie znakov z prvej sekvencie na znaky z druhej. Pri párovaní sekvencií AATCTATA a AAGATA máme pri nepovolení medzier tri možnosti ako ich spárovať.

$$\begin{array}{lll} AATCTATA & AATCTATA & AATCTATA \\ AAGATA & AAGATA & AAGATA \end{array}$$

Pri povolení dvoch medzier pre kratšiu sekvenciu dostávame až 28 možností zarovnania týchto dvoch sekvencií. Toto sú iba 2 z nich.

$$\begin{array}{cccccccc} A & A & T & C & T & A & T & A \\ A & A & G & - & A & T & - & A \end{array} \qquad \begin{array}{cccccccc} A & A & T & C & T & A & T & A \\ A & A & - & G & - & A & T & A \end{array}$$

Všetky zarovnania sa dajú ohodnotiť podľa skórovacieho systému. Často používaným systémom je -1 pri medzere, 1 pri zhodnom znaku a 0 pri substitúcii. Toto nastavenie hodnotenia reflektuje fakt, že k substitúcii dochádza v prírode častejšie ako k pridaniu(zmazaniu) znaku. Všetky dvojice znakov sa ohodnotia podľa skórovacie systému a výsledné skóre sa dostane sčítaním. Ak porovnávané sekvencie sú T a R, výsledné skóre zarovnania sa dá vypočítať takto.

$$\sum_{i=1}^{n} \begin{cases} -1 & ak\ T_i\ alebo\ R_i\ je\ medzera \\ 0 & ak\ T_i \neq R_i \\ 1 & ak\ T_i = R_i \end{cases}$$

Pre uvedený príklad sú skóre 1 a 3.

Optimálne zarovnanie sa dá vypočítať ohodnotením všetkých možných zarovnaní. Tento postup je pochopiteľne výpočetne veľmi náročný. Pri reťazcoch dĺžky 100 a 95 a povolení len 5 medzier je približne 55 miliónov možných zarovnaní.

Pre tento výpočet sa používajú princípy *dynamického programovania*, kde je problém rozdelený do niekoľkých podproblémov. Ak vezmeme ako príklad sekvencie ACAGTAG a ACTCG. Pri zarovnávaní prvých dvoch znakov máme v podstate iba 3 možnosti: (A,A), (A,-) a (-,A). Každá z týchto možností má svoje skóre, ktoré sa pričíta k zvyšku zarovnaných znakov, ktoré už inak neovplyvňuje. Takže možnosti sú skóre 1 + zarovanie CAGTAG a CTCG, skóre -1 + zarovnanie ACAGTAG a CTCG a skóre -1 + CAGTAG a ACTCG. Pri hľadaní optimálneho zarovnania sa zvolí možnosť s najvyšším skóre. Pri zarovnávaní ďalších 2 znakov sú zase iba 3 možnosti. Využitie tohoto princípu znamená redukciu zložitosti na *kvadratickú*. Postup výpočtu sa dá zakresliť do matice.

		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

Skóre v prvom riadku a prvom stĺpci reprezentujú skóre pri vložení medzier na začiatok. Skóre na pozícii 2,2(1), reprezentuje výber zarovnania prvých dvoch znakov ako (A,A). Bolo vyberaté z troch možností: (-,A) znamená skóre z prvku 2,1(zhora) plus penalta za medzeru, (A,-) znamená skóre 1,2(zľava) plus penalta za medzeru, posledná možnosť skóre z prvku 1,1(vľavo hore) plus ohodnotenie zhody. Z týchto hodnôt bolo vybraté maximum(teda 1). Týmto spôsobom sú vypočítané všetky prvky matice.

Veľkosť matice je $n+1$ a $m+1$, m a n reprezentujú dĺžky porovnávaných reťazcov. Prvok matice MAT_{ij} je definovaný(dá sa vypočítať) susednými prvkami $MAT_{(i-1)j}$, $MAT_{i(j-1)}$ a $MAT_{(i-1)(j-1)}$.

Definícia hodnôt prvkov v matici:

$$MAT_{ij} = max \begin{cases} MAT_{(i-1)(j-1)} & +match \quad ak \quad U_i = V_j \\ MAT_{(i-1)(j-1)} & +sub \quad\quad ak \quad U_i \neq V_j \\ MAT_{i(j-1)} & +ins \\ MAT_{(i-1)j} & +del \end{cases}$$

Parametre *ins*, *del*, *sub* a *match* sú ohodnotenie za vloženie znaku, chýbajúci znak, substitúciu a zhodu znakov. Vloženie znaku a chýbajúci znak sú medzery.

Prvok matice v pravom dolnom rohu je výsledné skóre pre optimálne zarovnanie daných reťazcov. Zarovnanie sa dá z matice vypočítať hľadaním spätnej cesty, ktorou sa k výslednému skóre prišlo. Postupuje sa postupne od prvku vpravo dole vždy smerom, odkiaľ bolo skóre propagované, až sa dojde do ľavého horného rohu. Príklad dohľadania spätnej cesty je možné vidieť na obrázku 2.2. Optimálnych zarovnaní môže by všeobecne niekoľko.

Tento postup bol prvý raz formulovaný v článku [8] z roku 1970. Algoritmus sa nazýva Needleman-Wunsch a patrí k najpoužívanejšim v bioinformatike.

		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

Obrázok 2.2: Príklad spätného dohladania cesty

$$\begin{array}{ccccccc} A & C & - & - & T & C & G \\ A & C & A & G & T & A & G \end{array}$$

Týmto spôsobom je možné počitať *globálne zarovnanie*, to znamená, že sú zarovnané celé reťazce. Pri hľadaní podreťazca v reťazci to môže byť nedostačujúce kôli veľkým penaltám za medzeri na začiatku a na konci.

2.2 Semi-globálne a lokálne zarovnanie

Pri zarovnávaní reťazcov algoritmom Needleman-Wunsch sa vždy porovnávajú celé reťazce. Toto znemožnuje vyhladávanie kratšej sekvencie v dlhom reťazci.

$$\begin{array}{cccccccccccc} A & C & A & T & C & T & A & T & A & G & T \\ - & - & - & T & C & T & A & - & - & - & - \end{array}$$

Jednoduchou modifikáciou algoritmu Needleman-Wunsch je možne eliminovať vplyv začiatočných a koncových medzier. V prvom riadku a stĺpci matice budú samé nuly(eliminácia začiatočných medzier), pri počítaní posledného riadku a posledného stĺpca sa zrušia penalty za medzery(eliminácia koncových medzier). Tento postup výpočtu sa nazýva *semi-globálne zarovnanie*.

Pri hľadaní krátkeho reťazca v dlhšom reťazci, pričom sa tam môže vyskytovať viackrát, je semi-globálne zarovnanie nedostačujúce. Jeho obmedzením je, že hľadá len jeden výskyt. Ďalšou modifikáciou algoritmu Needleman-Wunsch je pridanie 4 možnosti pri určovaní prvku matice. Tou štvrtou možnosťou je nula. Týmto spôsobom sú eliminované všetky krajné medzery. Po vypočítaní matice sa v nej nájde maximum, a odtiaľ sa dohľadá spätná cesta. Takto sa dá získať zarovnanie dvoch podreťazcov. Týchto lokálnych maxím môže byť v celej matici niekoľko na rôznych miestach. Umožnuje nájsť všetky výskyty podobných podreťazcov v reťazcoch. Tento algoritmus prezentovali Temple Smith a Michael Waterman v roku 1981 [9]. Slúži k výpočtu *lokálneho zarovnania*. Príkladom výpočtu je matica na obrázku 2.3.

		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	0	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

Obrázok 2.3: Výpočet algoritmom Smith-Waterman

2.3 Heuristické prístupy

Približné porovnávanie reťazcov sa často používa pri prehľadávaní databáz. Analyzovaná biologická sekvencia je porovnávaná s množstvom už klasifikovaných sekvencií, s cieľom nájsť jej podobné sekvencie, a tak zistiť jej vlastnosti. Algoritmy Smith-Waterman a Needleman-Wunsch sa stávajú z dôvodu kvadratickej zložitosti nepoužiteľné.

Jednou z ciest ako riešiť túto situáciu je hľadanie nových spôsobov analýzy. Používanými heuristickými algorimami sú *BLAST* a *FASTA*. Ich nevýhodou je, že sú menej presné a nie je zaručené, že nájdu všetky príbuzné sekvencie.

BLAST sa využíva pre prehľadávanie databáz. V prvom kroku rozdelí hľadaný reťazec na prekrývajúce sa podreťazce pevnej dĺžky(najčastejšie dĺžky 4).

$$ATTACGAC \rightarrow ATTA, TTAC, TACG, ACGA, CGAC$$

Po tejto úprave sú podreťazce obsahujúce často sa vyskytujúce znaky zahodené a zvyšok podreťazcov je v databázach vyhľadávaný metódou presnej zhody. Po nájdení presnej zhody sa hľadanie na danom mieste rozšíry do oboch smerov. Posledné verzie *BLASTu* už pri rozšírenom vyhľadávaní akceptujú aj medzery.

Algoritmus *FASTA* sa využíva pre hľadanie podobného podreťazca v dlhom reťazci(napríklad hľadanie génu v celom genóme). Jedná sa o *lokálne zarovnanie*. Podobne ako pri algoritme *FASTA*, aj tu je reťazec najprv rozdelený na podreťazce. Pri vyhľadávaní v DNA sekvenciách sa používaju podreťazce dĺžky 4 až 6, pri aminokyselinách 1 a 2. Pri vyhľadávaní sekvencie aminokyselín *FAMLGFIKYLPGCM*, s rozdelením na podreťazce dĺžky 1, sa najprv vytvorí tabuľka so všetkými znakmi danej abecedy:

podreťazec	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
pozícia	2	13			1	5		7	8	4	3		11							9
					6	12				10	14									

V tabuľke v stĺpci F sa nachádza 1 a 6, pretože znak F sa nachádza v hľadanom reťazci na prvej a šiestej pozícii. Pri porovnávaní reťazca s druhým reťazcom TGFIKYLPGACT je vytvorená druhá tabuľka:

1	2	3	4	5	6	7	8	9	10	11	12
T	G	F	I	K	Y	L	P	G	A	C	T
	3	-2	3	3	3	-3	3	-4	-8	2	
	10	3				3		3			

Tabuľka ukazuje relatívnu pozíciu prvkov v prvom reťazci proti pozícii v druhom reťazci. V tabuľke sa často vyskytuje číslo 3, čo znamená že reťazce môžu byť na seba zarovnané pridaním 3 medzier na začiatok druhého reťazca.

Na tomto princípe algoritmus *FASTA* nájde lokácie, kde by mohli byť reťazce podobné. V danej oblasti sa potom konkrétne zarovnaie dopočíta algoritmom Smith-Waterman.

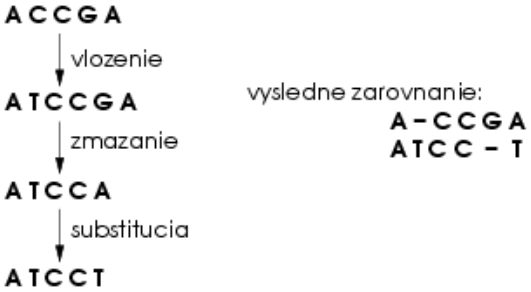
Obidva heuristické prístupy *BLAST*, *FASTA* nezaručujú nájdenie optimálneho zarovnania a tým pádom ani nezaručujú najdenie všetkých podobných sekvencií v databázach.

2.4 Možnosti paralelizácie algoritmov Smith-Waterman a Needleman-Wunsch

Hlavnou výhodou urýchlovania v hardwary je možnosť vykonávať viac výpočtov súbežne. Rýchlosť bežného procesoru(synchronizačná frekvencia) niekoľkonásobne prevyšuje rýchlosť FPGA čipu. Napriek tomuto faktu sú hardwarové riešenia často omnoho rýchlejšie práve vďaka svojej špecifickosti a orientovanosti na daný problém, ktorý riešia paralelne. Ako príklad sa dá uviesť grafický akcelerátor, ktorý je bežnou súčastou pracovnej stanice. Okrem iného dokáže veľmi výkonne násobiť matice, čo sa pri grafických transformáciách často využíva. Hardwarová akcelerácia sa využíva iba pri paralelizovateľnom probléme.

Na zarovnávanie reťazcov sa dá pozerať aj ako na postupnosť editáčných krokov, ktoré z prvého reťazca vytvoria druhý. Editačné kroky sú: substitúcia znaku, vloženie a vynechanie znaku. Týmto editačným krokom sú pridelené penalizácie. Súčet editačných penalizácií pri transformovaní prvého reťazca na druhý reťazec sa nazýva *editačná vzdialenosť*.

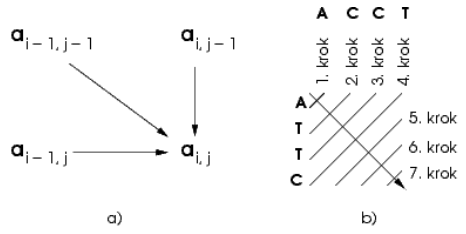
Editačný proces z DNA sekvencie ACCGA na sekvenciu ATCCT je znázornený na obrázku 2.4.



Obrázok 2.4: Editačný proces

Pri počítaní editačnej vzdialenosti je algoritmus Needleman-Wunsch jemne pozmenený. Všetky penalty sú kladné, a pri počitaní sa vyberá minimum z možných hodnôt. Výsledné zarovnanie zostáva rovnaké, avšak skóre sú iba kladné, čo zjednodušuje implementáciu.

Nech znaky v reťazcoch sú $R_{0..n}$ a $T_{0..m}$, prvky vypočítanej matice sú a_{nm}. Algorimy Smith-Waterman a Needleman-Wunsch majú podobný znak v tom, že každá bunka matice sa dá vypočítať zo susedných troch, $a_{i,j} \leftarrow a_{i-1,j-1}, a_{i,j-1}, a_{i-1,j}$. Závislosť hodnôt popisuje obrázok 2.5. V prípade, že sú k dispozícii dané susedné prvky, je možné počítať. Pred začatím výpočtu sú k dispozíci iba data pre výpočet jednej bunky, $a_{0,0}$. Po vypočítaní tejto bunky, už sú počitateľné bunky dve: $a_{0,1}$ a $a_{1,0}$. V n-tom kroku je možné počitať bunky: $a_{i,j}$, kde $i + j = n$. Je potrebných $min(n, m)$ samostatne pracujúcich výpočetných jednotiek(PE, implemetácia je popísaná 4.1), aby bola paralelizácia plne využitá. Výpočet zarovnania dvoch reťazcov je teda paralelizovateľný, je možné vykonávať viac výpočtov zároveň. Pri popísanej paralelizácii je časová zložitosť $n + m - 1$, teda lineárna. Popísaný princíp je znázornený v obrázku 2.6.



Obrázok 2.5: závislosť buniek matice

Obrázok 2.6: paralelizácia výpočtu

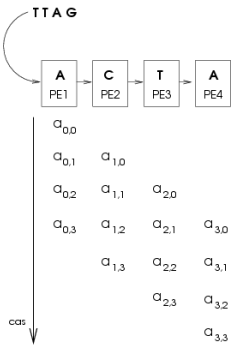
Pri následnom výpočte zarovnania nedochádza k problému nedostatku dát, celá matica je už vypočítaná. Napriek tomu nastáva problém pri spätnom dohľadávaní. Od prvku $a_{n,m}$ smerom k prvku $a_{0,0}$ sa môže cesta rôzne vetviť. Optimálnych zarovnaní môže všeobecne existovať niekoľko. Je potrebný vopred neznámy počet výpočetných jednotiek, pričom nastáva aj problém s prístupom k dátam z vypočítanej matice.

Uloženie skóre z matice kvôli výpočtu spätnej cesty tiež nie je triviálne. Ako výhodná možnosť sa ukazuje neukladať skóre(je nezaujímavé, dôležitý je iba prvok $a_{n,m}$), ale ukladať odkial bolo skóre propagované. Táto možnosť sa dá kódovať na 3 bity.

Všeobecne používaný hardwarový koncept výpočtu matice je realizovaný pomocou *systolického poľa*. Počítacie jednotky(Processing Elemet, PE) sú usporiadané v poli. Každá z jednotiek realizuje výpočet jedného stĺpca matice. Medzivýsledky si procesné elementy medzi sebou vymieňajú.

Dva porovnávané reťazce sú označené ako *referenčný* a *testovací*.

PE potrebuje k výpočtu 3 skóre a jeden znak z každého reťazca. Kedže počíta jeden stĺpec, zhora dole, znak z referenčného reťazca je stále ten istý. Skóre z prvku $a_{i,j-1}$ je vlastne ním samým vypočítaný predošlý výsledok. Testovací reťazec je nasúvaný zľava a PE si ho spolu so skóre posúvajú ďalej. Tento proces v čase zobrazuje obrázok 2.7.



Obrázok 2.7: Práca systolického poľa v čase

V prvom kroku počíta prvý PE prvok matice $a_{0,0}$. Má k dispozícii všetky skóre a prvý znak z každého reťazca. Po vypočítaní prvku si jeho hodnotu uloží pre svoj nasledujúci výpočet, a zároveň túto hodnotu spolu so znakom T propaguje ďalej nasledujúcemu PE. V nasledujúcom kroku už prvý PE počíta znak $a_{0,1}$, druhý PE počíta znak $a_{1,0}$. Vždy po vypočítaní prvku matice, PE propaguje vypočítané skóre a znak nasúvaného(testovacieho) reťazca ďalšiemu PE. Výsledné skóre zarovnania obidvoch reťazcov je posledná vypočítaná hodnota posledným PE. Časovú zložitosť výpočtu sa dá týmto spôsobom zredukovať na lineárnu. Pri dĺžkach reťazcov n a m je to $n + m - 1$. Tento princíp sa nazýva jednosmerné systolické pole procesných elementov [5].

U algoritmu Smith-Waterman nastáva problém, kde začat so spätným dohľadávaním. Kedže sa jedná o lokálne zarovnanie, reťazce môžu byť podobné kdekoľvek. Bežne je zaujímavých k najpresnejších zarovnaní. Začiatok týchto zarovnaní reprezentuje vysoké skóre. Od bunky s lokálne najvyšším skóre sa pokračuje podobne ako pri Needleman-Wunsch podla propagácie skóre. Tieto miesta by sa dali identifikovať už počas výpočtu 4.6.1.

Kapitola 3

Hardwarové implementácie v iných prácach

V tejto kapitole sú uvedené práce a dosiahnuté výsledky z minulosti. Dôležité je, všímať si či sa zaoberajú iba výpočtom matice alebo aj zarovnaním reťazcov(vysvetlené 2.4). Ďalej je podstatná použiteľnosť architektúry, do akej miery je parametrizovateľná. O výkone vypovedá hodnota BUPs(billions of updates per second), teda koľko miliónov výpočtov architektúra vykoná za sekundu(výpočtom sa myslí jedna bunka matice).

3.1 Prvé hardwarové urýchlenia

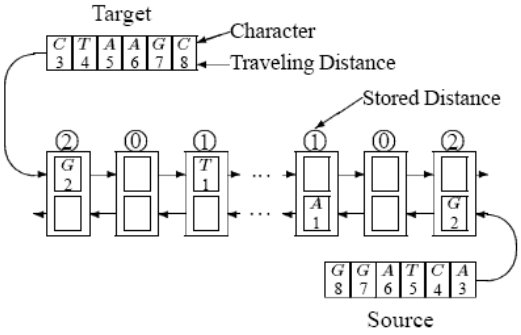
Prvá hardwarová architektúra [6] realizujúca výpočet podobnosti a zároveň aj zarovnania reťazcov bola realizovaná na platforme SPLASH, jedná sa o koprocesor pre Sun VME bus. Obsahuje 32 Xilinx XC3090 FPGA čipov. Autori publikácie realizujú výpočet pomocou *obojsmerného* systolického poľa procesných elementov(PE).

Reťazce sú do poľa vkladané z oboch strán(obrázok 3.1) a pri ich stretnutí začína výpočet. Pri každom PE je statická pamäť pre uloženie skóre. Po vypočítaní celej matice začína druhá fáza výpočtu(zarovnanie). Reťazce sú nasúvané do poľa reverzne. Skóre sú vyčítavané zo statických pamätí, opätovne dochádza k porovnávaniu znakov, zatiaľ čo PE si podávajú *značku* určujúcu, ktorý znak je práve zarovnávaný. Počas výpočtu generujú jednoduchý bitstream, ktorý reprezentuje pohyb *značky*. Ten je následne dekódovaný na zarovnanie reťazcov.

Je viacero možností ukladania vypočítaného skóre(kvôli neskoršiemu výpoču zarovnania) [4]. Ako použiteľné sa javia 3 možnosti, ukladať skóre ako hodnotu. Nevýhodou tohoto riešenia je pomerne široká dátová cesta, ktorá teoreticky môže pretiecť. Ďalšou možnosťou je ukladať 3 bitový vektor, ktorý ukazuje, od ktorých z troch susedných buniek matice(obrázok 2.5) bolo skóre propagované. Poslednou možnosťou je ukladať iba prírastok oproti susednému prvku. Toto riešenie môže byť kódované jedným, prípadne viacerými bitmi(v závislosti na penalizačných parametroch).

Obmedzením tohoto riešenia je obojsmerné systolické pole PE. Na SPLASH zariadení má veľkosť 248 PE. Čo limituje dĺžku porovnávaných reťazcov na 123. Takisto autori neuvažovali použitie inej abecedy, veľkosť použitých Xilinx FPGA čipov je značne limitujúca. Rýchlosť výpočtu je 0.37 BUPs.

Tí istí autori neskôr implementovali výpočet na novšej architektúre [5]. Výpočet implementovali aj *obojsmerným* systolickým polom PE aj *jednosmerným*. V práci pojednávajú

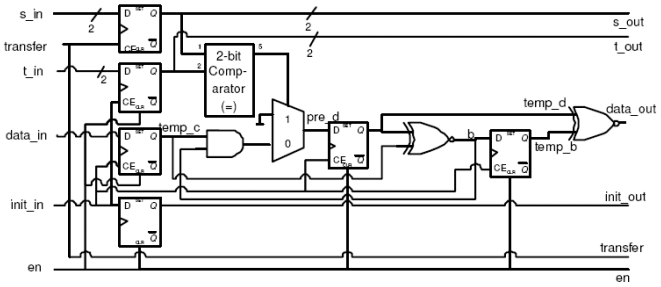


Obrázok 3.1: obojsmerné pole PE

o výhodách a nevýhodách. Prišli k záveru, že jednosmerné pole je efektívnejšie. Pre porovnanie 2 reťazcov dĺžky n a m, je potrebných $2max(m+1, n+1)$ PE, pričom pri jednosmernom poli stačí $max(n, m)$ PE. Implementácia dosahuje výkonu 43 BUPs. Podobne ako predošlá implementácia [6], aj tu je veľmi nízka modularita. Nie je možné meniť ani šírku abecedy, ani penalizačné parametre.

3.2 Silná optimalizácia na jednu úlohu

Autor Yu a kolektív vo svojej práci [11] prezentovali veľmi silne optimalizovaný design pre riešenie algoritmu Smith-Waterman nad DNA sekvenciami. Využíva sa platforma FPGA. Riešenie nepodporuje výpočet zarovnania, iba podobnosti. PE je implementované pomocou elementárnych prvkov technológie FPGA, čo umožňuje vysokú optimalizáciu. 2 PE sú mapované do 6 Virtex SLICEs. Na druhej strane nie je možné používať vačšiu šírku abecedy ako 2 bity. Rovnako nie je možné meniť penalizačné parametre. Na FPGA čipe XCV1000E je 4032 PE, pracovná frekvencia až 202 MHz, dosiahnutý výkon 814 BUPs.

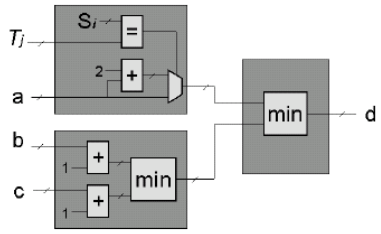


Obrázok 3.2: optimalizovaný PE

3.3 Porovnávanie s využitím run-time rekonfigurácie

Technológia FPGA poskytuje možnosť rekonfigurácie [3]. FPGA čip je konfigurovateľné hradlové pole. Pred použitím je do neho nahratá informácia o nastavení hradiel(konfigurácia)

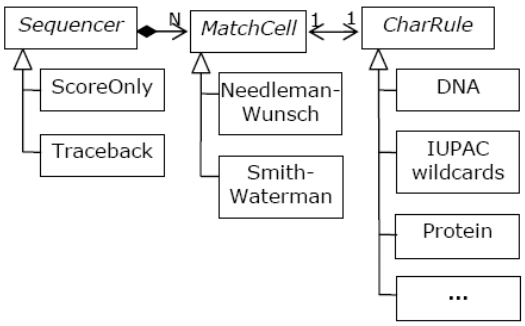
v podobe *bitstreamu*. Novou myšlienkou je nahrať vstupné reťazce, rovnako ako aj penalizačné konštanty do FPGA v podobe konfigurácie. Autori chcú po vypočítaní jednej konkrétnej úlohy prepisom časti konfigurácie do FPGA nahrať ďalšiu úlohu. Penalizačné parametre a jeden z reťazcov by boli priamo súčasťou logiky procesných elementov. Tento princíp umožňuje PE viac optimalizovať, zaberie menej zdrojov a je schopný bežať na vyššej frekvencii. Avšak súčastná podoba FPGA hradlových polí nie je na to pripravená. Rekonfigurovať iba časť designu, zatial čo zvyšok počíta je prakticky nemožné. Autori [3] o tejto problematike iba pojednávajú, fungujúca implementácia prezentovaná nie je. Na obrázku 3.3 je zobrazená schéma PE, ktorý má jeden z reťazcov nahratý ako konštantu priamo v konfigurácii. Teoretický výsledok prezentovaný autormi na FPGA čipe xc2v6000 má 11 000 PE, beží na frekvencii 280 MHz. Výkon teoretickej architektúri by bol 3225 BUPs.



Obrázok 3.3: PE s penalizačnými parametrami a jedným reťazcom nahratým v konfigurácii

3.4 Všeobecné architektúry

Viacero prác sa snaží o čo najvšeobecnejšiu architektúru. Využíva sa podobný prístup ako pri tvorbe softwaru, napríklad využívanie UML diagramov [2]. PE je implementované všeobecne pre obidva algoritmi Smith-Waterman a Needleman-Wunsch. Sú možné dva typy priechodu, len výpočet skóre, alebo aj spätné dohľadanie cesty. Problém je rozdelený do podúloh [10]. Tieto riešenia nevykazujú vysoký výkon, PE sú niekedy až príliž komplikované, čo znižuje maximálnu frekvenciu, ako aj počet PE na čipe. Výkon sa pohybuje okolo 2 až 10 BUPs.



Obrázok 3.4: Všeobecný prístup k riešeniu [2]

3.5 Porovnanie architektúr

Pri implementácií hardwarového urýchlovania sú možné dve hlavné cesty: buď vysoký výkon s malou parametrizovateľnosťou [11], alebo nízky výkon, ktorý je schopný riešiť veľkú škálu úloh [2]. Pri implementovaní treba voliť podľa reálnych biologických úloh: ktoré parametre nemusia byť nastavovateľné, ktoré by mali byť nastavovateľné, ale stačí pred syntézou[1] designu. A parametre voliteľné za behu. Využitie run-time rekonfigurácie je zatial len ťažko uskutočniteľné.

Prehľad parametrov súčastných architektúr je v tabuľke 3.5. Sú uvedené hardwarové urýchľovače postavené na technológiách ASIC a FPGA. Pre porovnanie je uvedený aj cluster staníc Alpha. Posledné dva riadky 2 riadky obsahujú teoretické hodnoty. Kedže porovnávanie s využitím run-time rekonfugurácie(viď 3.3) zatiaľ nebolo implementované.

	počet PE na 1 zariadení	**počet zariadení**	**BUPs**
Celera - *Alpha cluster*	1	800	*250*
BIPS - *ASIC*	16	16	*0,2*
Gene Matcher 2 - *ASIC*	192	16	*0,2*
SWASAD - *ASIC*	64	4	*3,2*
Paracel - *ASIC*	192	144	*276*
TimeLogic - *FPGA*	6	160	*50*
Splash - *FPGA XC3090* [6]	8	32	*0,37*
Splash 2 - *FPGA XC4010* [5]	14	272	*43*
SAMBA - *ASIC*	4	128	*1,28*
Yu a kol. - *FPGA XCV1000E* [11]	4032	1	*814*
Van Court - všeobecný prístup - *FPGA* [2]	cca 100..200	1	*cca 2..10*
Hokiegene - run-time rekonfigurácia - *FPGA*	7000	1	*1260*
JBits - run-time rekonfigurácia - *FPGA* [3]	11000	1	*3225*

Tabuľka 3.1: Zhrnutie súčastných hardwarových architektúr

[1] proces prekladu zdrojového VHDL kódu na konfiguráciu pre čip

Kapitola 4

Implementácia v FPGA

Táto kapitola sa zaoberá výberom hardwarovej platformy, implementačných nástrojov a implementačneho jazyka. Popisuje princípy návrhu architektúry. Zbežne sa venuje návrhu a funkčnosti kľúčových komponent v designe. Pojednáva aj o metódach implementácii niektorých riešení, ktoré budú implementované v ďalšej práci.

Prezentované existujúce architektúry sú často orientované a optimalizované na jednu úlohu [11], čo bráni ich širšiemu nasadeniu. Prípadne zbytočne veľká všeobecnosť znižuje dosiahnuteľný výkon. V tejto práci je implementovaný design, ktorého cieľom je vysoký výkon, ale hľavne použiteľnosť. Kód implementácie je napísaný genericky. Pred syntézou[1] je možné design "nastaviť", aby počítal zadanú úlohu čo najvýkonnejšie. Vytvorená konfigurácia pre čip bude teda špecializovaná na konkrétnu úlohu. Ale táto špecializácia je nastaviteľná pomocou generických parametrov pred syntézou.

Parametre, ktoré sú známe ešte pred syntézou sú šírka abecedy, približné dĺžky porovnávaných reťazcov a editačné penalty. Z týchto parametrov je potom nutné vypočítať ďalšie parametre designu. Toto bude bližšie popísané v sekcii 4.5. Výpočet týchto údajov môže byť automatizovaný [1]. Takže v konečnom dôsledku je možné podľa parametrov bioinformatickej úlohy vygenerovať generické parametre pre design, ktorý sa následne syntetizuje. Takto získaná konfigurácia sa nahrá do FPGA čipu a bude vykonávať výpočet danej úlohy. Nie je pri tom potrebný skúsený VHDL vývojár.

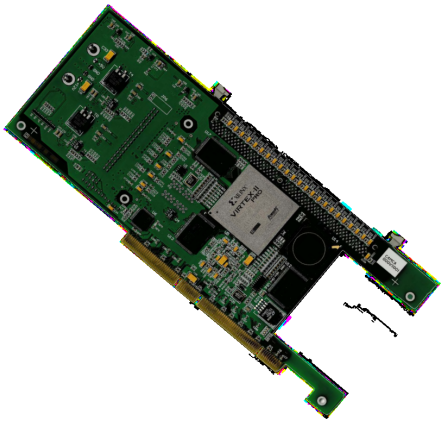
Dáta, ktoré sú nahrávané do karty už po syntéze a konfigurácii čipu sú obidva porovnávané reťazce a riadiace príkazy.

Zvolenou platformou pre implementáciu urýchľovania je karta *combo6x*(obrázok 4.1). Obsahuje výkonný FPGA čip *Xilinx Virtex II PRO - XC2VP50* s pomerne vysokou kapacitou. Karta combo6x komunikuje s hosťovským počítačom cez PCI-X zbernicu. Priepustnosť PCI-X zbernice je limitujúci faktor, ktorý má vplyv na celkový výkon. Celkové riešenie ale nie je na danej platforme závislé a nebude komplikované ho v budúcnosti používať na novšej karte *combo6E*, ktorá zatiaľ nie je k dispozícii. Karta *combo6E* ku komunikácii využíva PCI Express zbernicu, ktorá má neporovnateľne vyššiu priepustnosť.

4.1 Procesný element

Srdcom hardwarového urýchlovača je procesný element(PE). Vykonáva porovnávanie dvoch znakov. Procesných elementov je v designe niekoľko, sú organizované do systolického poľa 4.2.

[1] proces prekladu zdrojového VHDL kódu na konfiguráciu pre čip



Obrázok 4.1: karta combo6x

Princíp činnosti systolického poľa je popísaný vyššie 2.4. Na obrázku 4.2 je schéma komponenty.

Pred začatím výpočtu je PE inicializovaný. Nastavením signálu *INIT_I* do logickej jednotky sa do registru *reg_vs*(vertical string) zapíše znak z referenčného reťazca, ktorý sa počas celého výpočtu nebude meniť. Súčastne sa do skóre registru zapíše inicializačné skóre. Tento register uchováva vždy predošlý výsledok. Signál *INIT_I* je registrovaný a v ďalšom takte propagovaný nasledujúcemu PE, ktoré bude inicializované o takt neskôr.

Porovnávanie znakov vykonáva jednotka *mcell*(matching cell), vstupy A, B, C označujú skóre susedných prvkov v tabuľke 2.5, $A = a_{i-1,j-1}$, $B = a_{i,j-1}$, $C = a_{i-1,j}$. Výstup D je vypočítané skóre, ktoré je v ďalšom takte spätne privádzané ako signál B. Signály A a C sú privádzané od susedného PE(v prípade prvého PE od komponenty *SYS_CTRL* 4.3). Znak testovacieho reťazca HS(horizontal string), je takisto privádzaný zo susedného PE a zároveň je registrovaný pre použitie v ďalšom takte pre nasledujúce PE.

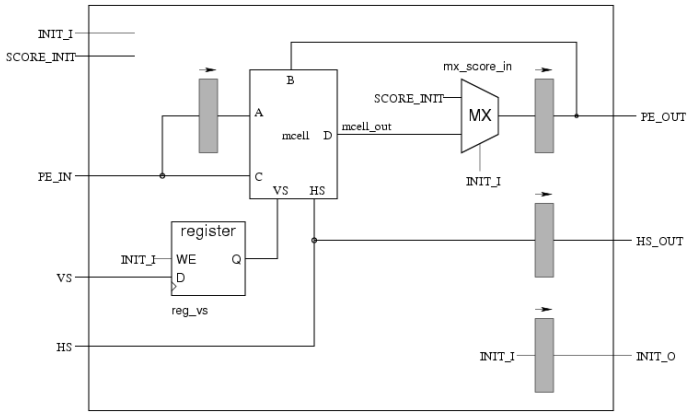
Jednotka *mcell* je jednoduchá, vyberá minimum z hodnôt A, B, C. K hodnote A pripočíta substitučnú penaltu v prípade, že sú signály *HS* a *VS* nezhodné 2.1.

Komponenta PE žiadnym spôsobom neuchováva skóre z predošlých výpočtov. Pamätá si iba posledne vypočítané skóre, pretože iba to potrebuje pre ďalší výpočet. Výsledkom zarovnávania je teda iba výstup z jedného PE po vypočítaní posledného prvku matice. Tento postup neumožnuje spätné dohľadávanie cesty. Komponenta PE sa bude musieť v ďalšej práci rozšíriť. Je potrebné doimplementovať jeden z troch spôsobov 2.4 ukladania skóre počas výpočtu, aby bol možný výpočet zarovnania pri spätnom priechode.

4.2 Systolické pole

Systolické pole (SA) je jednoka samostatne realizujúca celý výpočet, v celom designe sa ich môže nachádzať viac. SA má vlastný konečný automat realizujúci jeho riadenie, s okolím komunikuje pomocou registrov.

Komponenta obsahuje generický počet *procesných elementov* (PE, viď 4.1), ktoré sú usporiadané v poli, preto sa nazýva systolické pole. Súčasťou komponenty sú ďalej dve pamäte pre uloženie zadaných reťazcov(SMEM, viď 4.4). Registre pre komunikáciu s okolím: dva pre dĺžky reťazcov, prikazový register, register pre uloženie výsledku. Jednou z najkomplikovanejších komponent je *SYS_CTRL*(systolic controler, viď 4.3), ktorý riadi beh výpočtu

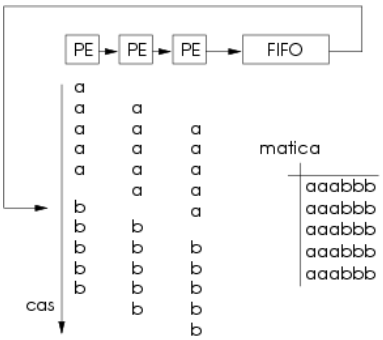


Obrázok 4.2: Bloková štruktúra komponenty PE

v poli PE. Reťazce sú pomenované ako *referenčný* a *testovací*. Ich zamenenie neovplyvní výsledok, ale architektúra je stavaná tak, že predpokladá dhlhší reťazec ako *testovací*. Ak je jeden z reťazcov dlhší musí byť nahratý do druhej SMEM(string memory 2).

Medzi posledným PE a prvým PE je zapojené *FIFO*[2](je vnúry komponenty *SYS_CTRL* 4.3). Toto zapojenie umožňuje, pri výpočete reťazcov dlhších ako počet PE, uchovanie medzivýsledkov a následné pokračovanie výpočtu. Nasúvanie *testovacieho reťazca* je potom opakované. Princíp ukazuje obrázok 4.3.

Na obŕazku 4.4 je možné vidieť blokovú štruktúru komponenty. V schéme je vidieť generický stromový multiplexor, ktorý prepojuje výstupy všetkých PE a register pre uloženie výsledku. V prípade, že je referenčný reťazec kratší ako počet PE, môže byť výsledok(editačná vzdialenosť) vygenerovaný ktorýmkoľvek PE. Z tohoto dôvodu je nutné všetky výstupy multiplexovať.



Obrázok 4.3: Priebeh výpočtu v čase s využitím FIFO

4.2.1 Riadenie systolického poľa

Systolické pole je riadené registrami, jeho ovládanie zabezpečujú jednoduché zápisy. Adresy jednotlivých registrov sú uvedené 7.1. Postup pre zadanie výpočtu je nasledovný.

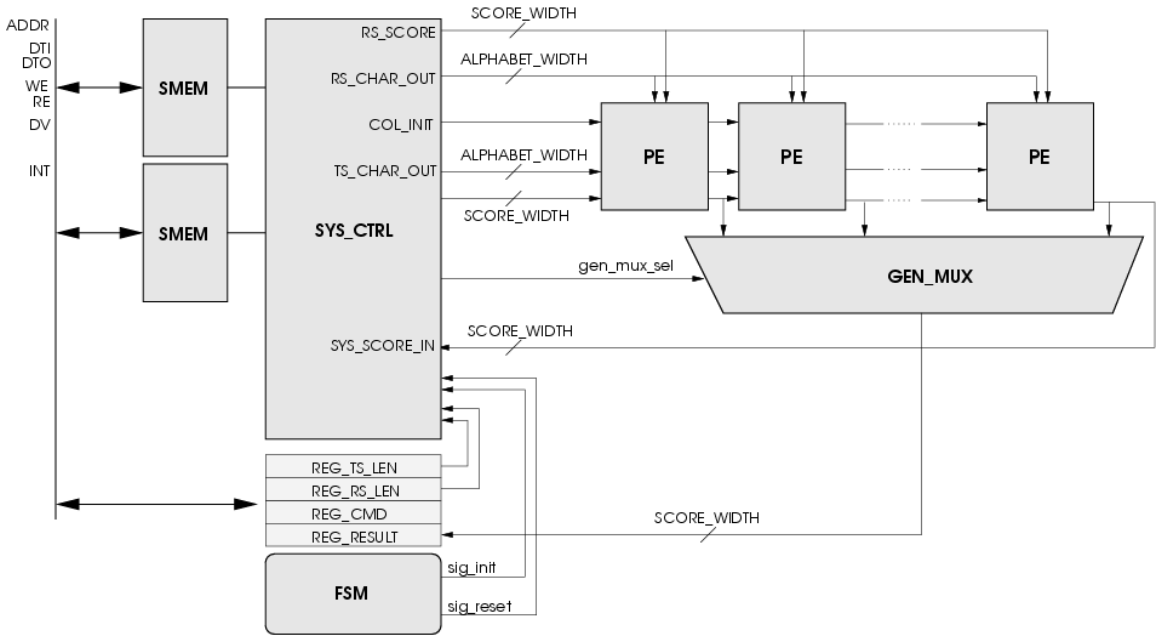
[2]First In, First Out - komponenta, z ktorej sú hodnoty vyčítavané v rovnakom poradí, ako boli do nej vložené

1. **Nahranie úlohy**
   Ako prvé je nutné nahrat reťazce do pamätí pre ne určených. Kapacita pamätí je genericky ovplyvniteľná. Následne je nutné nahrať dĺžky reťazcov do zodpovedajúcich registrov.

2. **Spustenie výpočtu**
   Ak sú data pre výpočet pripravené, stačí zapísať do *reg_cmd* hodnotu 1. Ak práve prebiehal iný výpocet, dochádza k jeho vyresetovaniu.

3. **Získanie výsledku**
   Po istom čase bude možné z registru *reg_result* vyčítať výsledok. Po dokončení výpočtu sa generuje prerušenie(jeho generovanie sa dá ovplyvniť genericky). Prerušenie je zhodené po vyčítaní ktoréhokoľvek registru.

Nevýhodou tohoto riešenia je ťažkopádne riadenie zo softwaru. PCI tranzakcie častokrát trvajú dlhšie ako samotný výpočet. V budúcnosti bude potrebné implementovať viac autonómne riadenie.



Obrázok 4.4: Bloková štruktúra komponenty SYS_ARRAY, *systolické pole*

4.3 Systolický kontrolér

Úlohou systolického kontroléru je zásobovať pole PE znakmi z obidvoch porovnávaných reťazcov a správnym skóre. Ďalšou úlohou je identifikovať výsledok, ktoré PE ho vypočítalo a kedy.

Systolický kontrolér má k sebe pripojený výstup z posledného PE(schéma 4.4). Ak je referenčný reťazec dlhší ako počet PE, sú skóre ukladané do FIFA(vo vnútri kontroléru), odkiaľ sú opäť nasúvané do prvého PE, keď dopočíta stĺpec. Toto umožnuje výpočet reťazcov dlhších ako je počet PE. Tento princíp spätnej väzby je podrobnejšie popísaný v sekcii 4.2.

Obsahom komponenty je niekoľko čítačov spolu s ich riadením. Jeden čítač číta adresu pre vyčítanie znakov referenčného reťaza(reťazec je uložený v SMEM), pri reťazci dlhšom ako počet PE je pozastavovaný a opätovne spustený, až keď je prvé PE pripravené znova počítať. Podobne ďalší čítač je určený pre adresovanie znakov testovacieho reťazca, nie je nikdy pozastavovaný, ale beží viackrát v prípade referenčného reťazca dlhšieho ako je počet PE v poli. Od týchto dvoch čítačov sa odvíja riadenie FIFA(read a write signály), výber vsupného skóre pre prvý PE, ako aj multiplexor výberu výsledku a zápis do registru výsledku.

FIFO komponenta môže byť realizovaná poskladaním Block RAM komponent[3], alebo distribučne pomocou LUT komponent[3]. Výber realizácie je genericky ovplyvniteľný. Pri rôznych realizáciách má ale FIFO rôzne správanie. Pri čítaní dát z Block RAM FIFO komponenty, sú dáta opozdené o 2 až 3 takty od nahodenia čítacieho signálu. Pri použití FIFA poskladaného z LUT komponent, sú dáta k dispozícii hneď. Tento fakt komplikuje celkové riadenie. Komponenta SYS_CTRL sa tomuto faktu prispôsobuje vkladaním registrov za výstup z FIFO komponenty.

Podobný problém nastáva pri využívaní pamätí s reťazcami, ktoré môžu byť tiež realizované obidvoma spôsobmi. Je riešený podobne, vyčitací signál spolu s adresou je vystavovaný o dva takty skôr. Potom v prípade použitia distribučnej pamäti z LUT, sú dáta za výstupom z pamäti opozdené.

Závažný problém ešte nastáva pri referenčnom reťazci o jedna dlhšom ako je počet PE a zároveň rovnako dlhom testovacom reťazci. Za týchto, i keď ojedinelých podmienok, je skóre z FIFO komponenty potrebné skôr ako stihne prebehnúť jeho zápis a vyčítanie. Tento ojedinelý prípad je detekovaný a pri jeho uskutočnení je FIFO komponenta obídená. Skóre je s istým oneskorením napojené priamo na vstup systolického poľa.

4.4 Pamäť na reťazce

Reťazce sú počas výpočtu uložené v komponente SMEM, ktorá je súčasťou komponenty SYS_ARRAY. Jej kapacita je genericky ovplyvniteľná. Komponenta SMEM má dve prístupové rozhrania. Prvé dokáže aj čítať aj zapisovať, je pripojené na *local bus*. Toto rozhranie má pevnú šírku 32 bitov a je využívané na zadávanie požiadavok z hosťovského počítača. Čítanie reťazcov späť do PC je možné, ale využívané iba pri ladení.

Druhé rozhranie je napojené na komponentu SYS_CTRL, dátová šírka sa rovná šírke abecedy. Šírka adresy je rozšírená na zodpovedajúcu hodnotu. Z tohoto rozhrania je možné iba čítať. Obrázok 4.5 znázorňuje princíp implementácie.

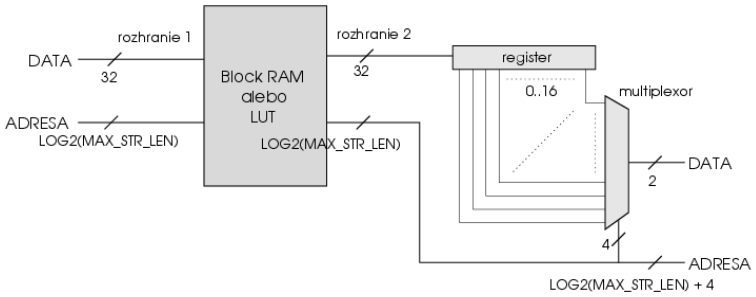
Samotná pamäť pre uloženie reťazca može byť, v závislosti na generickom parametry SMEM_TYPE(viď 4.5), realizovaná buď pomocou Block RAM komponent, alebo distribuovaná s využitím LUT komponent.

4.5 Štruktúra designu

Design je z veľkej časti generický. Jedným z hlavných cieľov bola možnosť riešiť vačšiu škálu bioinformatických úloh. Hlavné generické parametre sú:

- *ALPHABET_WIDTH* - Počet bitov pre kódovanie abecedy, všeobecne je možné použiť akúkoľvek abecedu. Napríklad pri porovnávaní DNA reťazcov sú 2 bity dostačujúce.

[3]jedna z elementárnych komponent technológie FPGA



Obrázok 4.5: Činnosť komponenty SMEM

- *SCORE_WIDTH* - Počet bitov na ukladanie skórovacej matice, závisí na predpokladanej dĺžke porovnávaných reťazcov, ako aj na veľkostí penalizácií.
- *SYSTOLIC_COUNT* - Počet systolických polí v designe.
- *SYSTOLIC_SIZE* - Počet procesných elementov (PE) 4.1. V každom systolickom poli.
- *FIFO_SIZE* - Veľkosť FIFA pre dočasné ukladanie *skóre*, viď 4.2.
- *MAX_STR_LENGTH* - Kapacita komponenty SMEM, aká je maximálna uložiteľná veľkosť reťazca
- *SMEM_BRAM* - ak nastavené na *TRUE*, komponenta SMEM použije pre uloženie reťazca Block RAM pamäť, inak využije LUT
- *FIFO_BRAM* - ak nastavené na *TRUE*, FIFO v spätnej väzbe(viď. 4.2) v komponente SYS_CTRL použije pre uloženie skóre Block RAM pamäť, inak využije LUT
- *sub_penalty* - substitučná penalta(viď. 2.1), penalizácia za vymenenie znaku
- *up_penalty* - penalta za chýbajúci znak(viď. 2.1), penalizácia za vynechanie znaku
- *left_penalty* - penalta za znak navyše(viď. 2.1), penalizácia za prebytočný znak

Hodnoty generických parametrov sa dajú vypočítať automaticky [1]. Vstupom je popis úlohy, šírka používanej abecedy, približné dĺžky reťazcov a editačné penalty. Automaticky sú potom pre design vypočítané parametre, ako šírka skóre, počet PE v poli, počet polí v designe a podobne. Cieľom je aby komponenta PE nebola o nič komplikovanejšia ako musí byť(šetrenie zdrojov). Takto sa zmenší spotreba zdrojov pre jedno systolické pole. Týchto polí sa potom umiestni do designu čo najviac. Kedže sú schopné pracovať nezávisle, dojde k zvýšeniu výkonu.

Pri nastavovaní generických parametrov je nutné vedieť ako ovplyvnia možnosti designu, aké reťazce budú počitateľné.

Maximálna dĺžka počitateľného reťazca je nasledovná.

$$\min(\text{FIFO\_SIZE} + \text{SYSTOLIC\_SIZE} - 3, \text{MAX\_STR\_LENGTH})$$

Prvá podmienka vyplýva z počtu skóre, ktoré je schopná udržať kapacita FIFO. Druhá podmienka je jednoduchá, vyplýva z maximálnej veľkosti, ktorú poskytuje pamäť na reťazec.

Výpočet šírky skóre(SCORE_WIDTH) pri ktorom ani v najhoršom prípade nedôjde k pretečeniu. Premenná R symbolizuje maximálnu dĺžku reťazca.

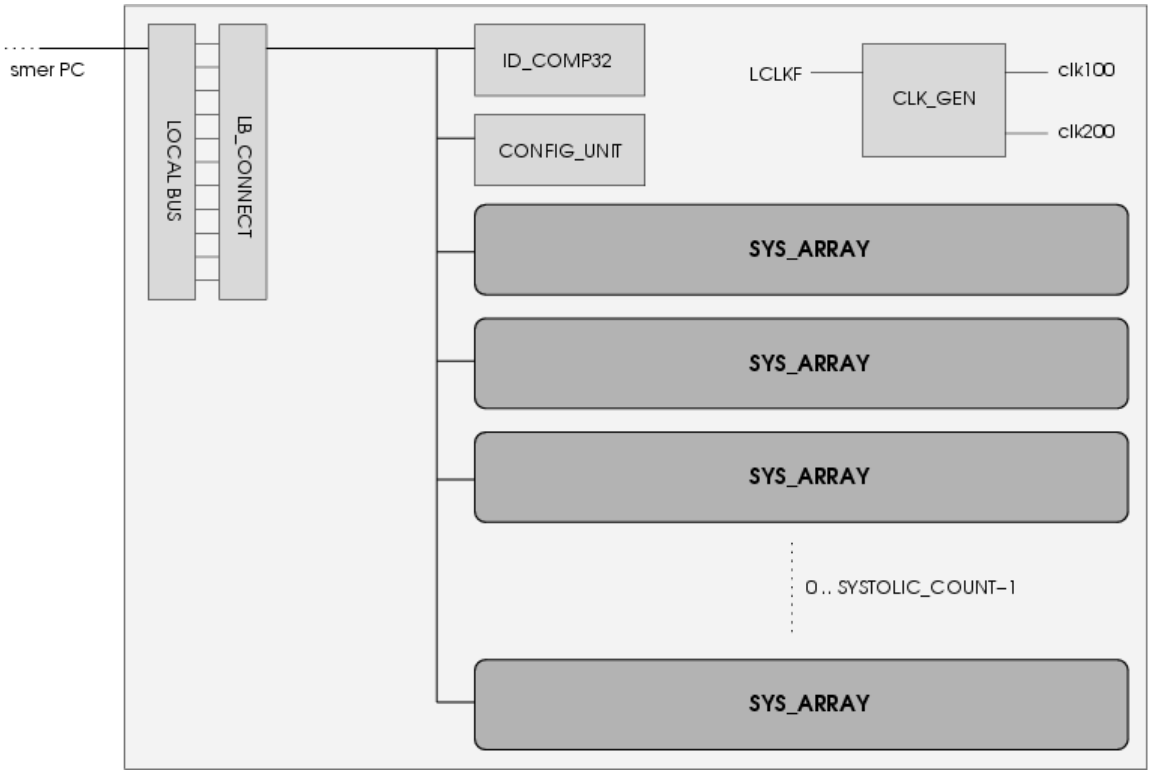
$$\text{SCORE\_WIDTH} = \log_2(\max(\text{sub\_penalty}, \text{up\_penalty}, \text{left\_penalty}).R + 1)$$

Ak RS symbolizuje dĺžku referenčného reťazca(nahrávaného do PE počas inicializácie 4.1). TS, dĺžku testovacieho reťazca putujúceho cez systolické pole a SS reprezentujúci generický parameter SYSTOLIC_SIZE. Potom dĺžka výpočtu v počte taktov je nasledovná.

$$\text{takty} = \begin{cases} (TS+1).(RS \div SS) + (TS + (RS \mod SS)) & \text{pre } RS \mod SS \neq 0 \\ (TS+1).(RS \div SS) + SS - 1 & \text{pre } RS \mod SS = 0 \end{cases}$$

Operácia $\div$ znamená celočíselné delenie, operácia mod zvyšok po celočíselnom delení.

Blokovú štruktúru znázornuje obrázok 4.6. Komponenta *ID_32COMP* je mapovaná na začiatok adresového priestoru a jej úlohou je identifikácia designu. Ovládač dokáže zistit aký design je nahratý v karte pomocou vyčítania údajov z adries ID komponenty. Komponenta *CONFIG_UNIT* je veľmi podobná identifikačnej, má v sebe generické parametre designu. Ovládač je teda schopný zistit koľko systolických polí ma k dispozíci a podobne. Najdôležitejšie sú komponenty *SYSTOLIC_ARRAY*. Jedná sa o *systolické pole*, ktoré realizuje samotný výpočet.



Obrázok 4.6: Bloková štruktúra designu

4.6 Ďalšia práca

Aktuálna implementácia má niekoľko nedostatkov. Riadenie systolických polí je riešené zápismi do kontrolných registrov 4.2.1, čo znamená že urýchlovač musí byť riadený softwarom. Tento princíp znamená minimálne 5 zápisových tranzakcií pre výpočet jednej podobnosti a šiesta čítacia tranzakcia pre získanie výsledku. Karta combo6x je pripojená cez PCI-X zbernicu. Takéto riadenie je veľmi pomalé. V budúcnosti bude potrebné požiadavky na výpočet kopírovať do combo6x karty pomocou DMA(viď 4.6.2), rovnako ako zaistiť možnosť viac autonómneho riadenia výpočtu. Pri realizácii bude potrebné vytvorit podporu zo strany softwaru vo forme *ovládača*. V budúcnosti bude design upravený pre nasadenie na novej karte *combo6E*(zatiaľ nie je k dispozícii), ktorá bude pripojená k hosťovskému počítaču pomocou PCI Expres systémovej zbernice. Toto umožní zväčšenie transportnej kapacity medzi bioinformatickou aplikáciou a akcelerátorom.

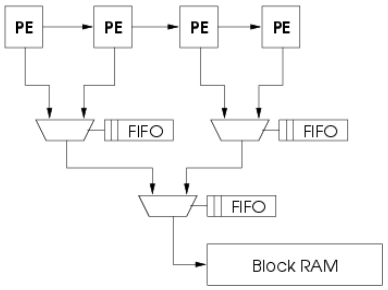
Implementovaný design nepodporuje možnosť výpočtu zarovnania reťazcov. Táto úloha je v bioinformatike často vyžadovaná, a preto by mala byť doimplementovaná. Niektoré komponenty by mohli byť optimalizované, napríklad skóre by nemuselo byť reprezentované hodnotou, ale rozdielom susedných hodnôt 2.4 [4].

4.6.1 Výpočet zarovnania u algoritmu Smith-Waterman

Algoritmus Smith-Waterman nepočíta podobnosť reťazcov *globálne*(ako Needleman-Wunsch), ale *lokálne*(viď 2.2). Výsledná matica určuje kde sú si reťazce najviac podobné. Koniec lokácie, kde sú podreťazce najpodobnejšie sa dá v matici identifikovať podľa vysokej hodnoty(lokálne maximum). Toto veľmi komplikuje spätné dohľadávanie zarovnania, začínať výpočet sa môže z viacerých miest. Podobných podreťazcov môže byť všeobecne niekoľko.

Ako možnosť riešenia sa ukazuje počas výpočtu matice tieto miesta identifikovať a po výpočte celej matice od nich začať spätné dohľadávanie zarovnania(obrázok 2.3). Každý PE počíta jeden stĺpec, pri poklese *skóre*(nastalo lokálne maximum) generuje o tejto udalosti *správu*. Zber správ je komplikovaný, z dôvodu veľkého počtu PE, ktoré nemôžu zároveň zapisovať do jednej pamäti. Posielanie správ by preto bolo zberané postupne *multiplexovacími jednotkami*, ktoré majú pri sebe FIFO, ktoré v prípade prijatia dvoch spŕav jednu uloží. Návrh riešenia popisuje obrázok 4.7.

Po dokončení výpočtu sa vyberie n najvyšších skóre, od ktorých sa vypočíta zarovnanie. PE jednotky môžu v najhoršom prípade generovať správu každý druhý takt. Takto veľký objem je navrhnutou architektúrou len ťažko spracovateľný.



Obrázok 4.7: Zber správ o lokálnych maximách

Pravdepodobnosť generovania správ pri náhodných reťazcoch

Správa(o nájdení lokálneho maxima) je generovaná v prípade výskytu nezhody porovnávaných znakov, pričom musela predchádzať minimálne jedna zhoda. Pri veľkosti abecedy a, je pravdepodobnosť zhody dvoch znakov $\frac{1}{a}$ a pravdepodobnosť nezhody $\frac{a-1}{a}$. Pravdepodobnosť, že PE bude generovať správu pri náhodných reťazcoch je $p_{pe} = \frac{1}{a}.\frac{a-1}{a} = \frac{a-1}{a^2}$. Po dosadení veľkosti abecedy pre DNA sekvencie, je táto pravdepodobnosť $0,1875$.

Z tohoto údaju je možné dopočítať aká musí byť veľkosť FIFO komponent na jednotlivých stupnoch multiplexovacieho stromu(obrázok 4.7), aby s istou pravdepodobnosťou nepretieklo. Z markovových modelov vychádza, že pre ustálený stav je pravdepodobnosť vyplnenia FIFA n prvkami:

$$p_n = p_{n-1}.\frac{p_{pe}^2}{(1-p)^2}$$

$$p_n = \frac{p_{pe}^{2n}}{(1-p_{pe})^{2n}} - \frac{p_{pe}^{2n+1}}{(1-p_{pe})^{2n+1}}$$

Ustálený stav nastáva, keď $\frac{p_{pe}^2}{(1-p_{pe})^2} < 1$. Toto platí pre všetky $a \in N$. Pravdepodobnosť pretečenia pri FIFE o veľkosti n prvkov, sa dá vypočítať ako suma pravdepodobností nekonečného geometrického radu, začínajúc prvkom pravdepodobnosťou zaplnenia n.

$$p_{overflow} = \frac{p_n}{1 - \frac{p_{pe}^2}{(1-p)^2}}$$

Pri použití abecedy o veľkosti 4(DNA sekvencie), FIFO o dvoch prvkoch pretečie s pravdepodobnosťou $0,00214$. Z uvedeného modelu je možné určiť pravdepodobnosť, že *multiplexovacia jednotka* bude posielať *správu* ďalej, do ďalšieho stupňa multiplexovania. Jedná sa o pravdepodobnosť, že FIFO je zaplnené aspoň jedným prvkom, plus pravdepodobnosť príchodu jednej až dvoch správ z vyššieho stupňa multiplexorového stromu.

$$p_{sprava} = \frac{p_{pe}^2}{(1-p_{pe})^2} + (1 - \frac{p_{pe}^2}{(1-p_{pe})^2}).(p_{pe}^2 + 2.p_{pe}.(1-p_{pe}))$$

S týmto údajom je rovnakým spôsobom možné určiť veľkosti FIFO a pravdepodobnosť pretečenia pre nasledujúce stupne *multiplexovacích jednotiek*. Prvý stupeň, pri použití doterajšej abecedy, generuje *správu* s pravdepodobnosťou $0,375$. Je potrebné FIFO veľkosti 4 prvky, aby s pravdepodobnosťou 0.967 nepretieklo.

Pravdepodobnosť generovania *správy* druhou úrovňou je $0,75$, čo znamená, že stav nebude ustálený a zabraňuje nasadiť doteraz používaný pravdepodobnostný model. Ukazujú sa dve možnosti riešenia tohoto problému.

Použiť pravdepodobnostný model, ktorý nepredpokladá ustálený stav a zväčšiť FIFO, tak aby po istý počet taktov s istou pravdepodobnosťou stíhalo ukladať správy. Po ukončení výpočtu by sa strom postupne vyprázdnil. Nevýhodou je relatívne dlhý čas, kým sa odošlú všetky správy a nie je možné začať nový výpočet.

Druhá možnosť je od istej úrovne dať možnosť *multiplexovacej jednotke* odosielať viac správ naraz. Zväčšením dátovej šírky by bolo možné toto docieliť. Nevýhodou sú komplikovanejšie multiplexovacie jednotky a komplikované spracovávanie správ na konci stromu. Táto otázka ešte nebola vyriešená, uvedený princíp nie je implemetovaný. Je to jedna z možností rozšírenia funkcionality.

4.6.2 Návrh autonómneho riadenia a využitia DMA prenosov

Autonómne riešenie operácií zatiaľ v akcelerátore nie je implementované. V tejto sekcii je uvedená jedna z možností riešenia. Implementované bude v ďalšej práci spolu s ovládačom.

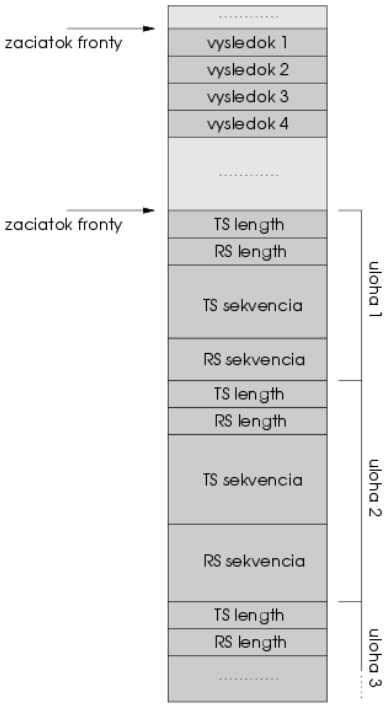
Riadenie systolického poľa je v momentálnej implementácii ťažkopádne, je nutných minimálne 5 tranzakcií na PCI-X zbernici, pre výpočet porovnania dvoch reťazcov 4.2.1. Tento fakt prudko znižuje výkonnosť celého zariadenia.

Hardwarové zariadenia na zberniciach typu PCI využívajú k rýchlemu prenosu DMA(Direct Memmory Access, priamy prístup do pamäte). O prenos dát sa stará hardwarové zariadenie samo, vyčítaním dát z hlavnej pamäti počítača, prípadne zápisom do hlavnej pamäti počítača. Tento princíp umožňuje vylúčiť procesor z tranzakcií, čo znamená, že môže vykonávať inú činnosť. Dáta sú takisto kopírované po väčších celkoch.

Cieľom je, aby reťazce boli pripravené v hlavnej pamäti počítača, odkiaľ si ich urýchlovač sám vyčíta, spracuje a výsledky opäť sám zapíše späť do pamäte.

Ovládač bežiaci na hlavnom počítači pripraví úlohy pre akcelerátor do formátu, ktorý hardwarový akcelerátor očakáva. Zápisom do riadiacich registrov určí adresu dát v hlavnej pamäti a dá povel na vykonanie úloh. Akcelerátor samostatne distribuuje dáta do systolických polí. Po vypočítaní výsledkov ich prekopíruje do hlavnej pamäte, kde ich očakáva ovládač. Oznámenie o pripravenosti výsledkov v hlavnej pamäti môže hardwarový akcelerátor dvoma spôsobmi. Buď vyvolaním prerušenia, alebo nastavením kontrolných bitov v hlavnej pamäti.

Úložište dát v hlavnej pamäti, aj zadanie úloh, aj výsledky. Môžu byť implementované vo forme kruhovej fronty. Jeden zo spôsobov organizácie dát je na obrázku 4.8.



Obrázok 4.8: návrh organizácie dát v pamäti

Tento spôsob riadenia systolických polí je výrazne autonómnejší a umožňoval by dosiahnuť väčší výkon. Limitujúcim faktorom však naďalej zostáva pomerne nízka priepustnosť

systémovej zbernice PCI-X. Tento problém sa dá riešiť jedine použitím modernejšej karty *combo6E*, ktorá momentálne nie je k dispozícii.

Kapitola 5

Dosiahnuté výsledky

Implementácia bola otestovaná na karte *combo6x*(obrázok 4.1), ktorá komunikuje s hosťovským počítačom cez PCI-X zbernicu. V tabuľke 5 sú uvedené teoretické výsledky, ktoré zanedbávajú čas nutný k transportu dát po systémovej zbernici a čas potrebný na riadenie. Ako aplikácie sú uvedené reálne bioinformatické úlohy, ktoré sa líšia v šírke abecedy, penalizačných parametroch a dĺžke porovnávaných sekvencií. Ďalej sa môže jednať o porovnávanie vždy dvoch rôznych párov reťazcov, alebo o porovnanie jednej sekvencie proti databáze.

Vysvetlenie jednotlivých stĺpcov tabulky:

- SP – počet systolických polí na čipe
- PE – počet procesných elementov(PE) v jednom systolickom poli
- Frekv. – maximálna frekvencia na ktorej je design syntetizovateľný
- BUPs – milióny updatov za sekundu, koľko prvkov matice je vypočítaných za sekundu (Pentium 4 3.2 GHz má približne 0,05 BUPs)
- Čas – čas potrebný na realizáciu daného výpočtu

Generické parametre pre design boli generované automaticky pre danú úlohu [1], nie je k tomu nutný skúsený hardwarový designer. Vygenerovanie designu pre danú úlohu je automatizované.

applikácia	SP	PE	Frekv.	BUPs	Čas
VPCR+PRIMEX	81	10	241	195.2	353 us
VPCR microarrays	1	40	241	9.6	12.4 s
Oligo vs. Genóm	1	80	209	16.7	14.3 s
Gén vs. Genóm	1	270	152	41.0	19.29 min
Proteín+PRIMEX	79	8	177	111.6	5.57 min
Proteín vs. Databáza	1	270	133	35.9	1.12 min
Proteínové uhly	6	71	137	58.3	46 s

Tabuľka 5.1: Výsledky s použitím čipu Virtex II xcv2p50-7

V porovnaní so softwarovým riešením(0,05 BUPs) dochádza k urýchleniu o niekoľko rádov. Aplikácia Virtual Polymerase chain reaction + PRIMEX pobeží s prezentovaným hardwarovým urýchlením približne 4000-krát rýchlejšie než bez hardwarového urýchlenia.

Kapitola 6

Záver

Pri implementácii hardwarového urýchlovača bolo použité FPGA hradlové pole od firmy Xilinx(Virtex II PRO - XC2VP50) na karte *combo6x*. Design bol úsepešne simulovaný nastrojom *ModelSim SE 6.0e*. Pri syntetizovaní boli použité nástroje *Preciosion Synthesis 2006a.112* od firmy Mentor Graphics a ďalšie nástroje. V práci sú prezentované všeobecne známe algoritmy používané pre približné porovnávanie reťazcov Smith-Waterman a Needleman-Wunsch 2.1. V sekcii 2.4 je popísaná možnosť ich paralelizácie. Bližšie detaily implementácie sú zhrnuté v kapitole 4.

Hardwarová architektúra je navrhnutá s cieľom urýchliť výpočet algoritmov Smith-Waterman a Needleman-Wunsch na čo najširšom množstve reálnych bioinformatických úloh [1]. Štruktúra návrhu riešenia čerpá princípy aj z iných prác [4]. Riešenie bolo testované na karte *combo6x*.

Jedným z nedostatkov uvedeného riešenia a takisto priestorom pre ďalšiu prácu je implementovanie viac autonómneho riadenia operácií. Softwarové riadenie po systémovej zbernici je pomalé a brzdí celú architektúru. Takisto je pre širšie využitie nutné rozšíriť funkcionalitu, hlavne podporu výpočtu zarovnania reťazcov.

Kapitola 7

Prílohy

7.1 Adresovanie komponent

Adresy jednotlivých komponent zo strany *local bus*.

adresa			komponenta	
00000000	-	00000030	ID_32COMP	
		0		NEG register
		4		Project ID, SW major, SW minor
		8		HW major, HW minor
		20		Project text
00000400	-	0000041C	CONFIG_UNIT	
		400		ALPHABET_WIDTH
		404		MAX_STR_LENGTH
		408		SCORE_WIDTH
		40C		SYSTOLIC_COUNT
		410		SYSTOLIC_SIZE
		414		FIFO_SIZE
00000800	-	00001810	SYSTOLIC_ARRAY 1	
		800		string memmory 1
		1000		string memmory 2
		1800		string 1 length
		1804		string 2 length
		1808		command register
		180C		result register
00002800	-	00003810	SYSTOLIC_ARRAY 2	
00004800	-	00005810	SYSTOLIC_ARRAY 3	
00006800	-	00007810	SYSTOLIC_ARRAY 4	

Literatúra

[1] P. Beck, O. Fučík, M. Lexa, and T. Martínek. Automatic generation of circuits for approximate string matching. In *Design and Diagnostics of Electronic Circuits and Systems*, pages 203–208, 2007.

[2] Tom Van Court and Martin C. Herbordt. Families of fpga-based accelerators for approximate string matching. *Microprocess. Microsyst.*, 31(2):135–145, 2007.

[3] 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, September 2002. Proceedings of the 12th International Workshop on Field-Programmable Logic and Applications, FPL 2002. Lecture Notes in Computer Science 2438.

[4] Dzung T. Hoang. A systolic array for the sequence alignment problem. Technical Report CS-92-22, Brown University, 1992.

[5] Dzung T. Hoang. Searching genetic databases on splash 2. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 185–191, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[6] Dzung T. Hoang and Daniel P. Lopresti. FPGA implementation of systolic sequence alignment. In Herbert Grünbacher and Reiner W. Hartenstein, editors, *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, pages 183–191. Springer-Verlag, Berlin, 1992.

[7] Dan E. Krane and Michael L. Raymer. *Fundamental Concepts of Bioinformatics*. Benjamin Cummings, september 2002.

[8] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–535, 1969.

[9] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[10] Yoshiki Yamaguchi, Tsutomu Maruyama, and Akihiko Konagaya. High speed homology search with FPGAs.

[11] C. W. Yu, K. H. Kwong, Kin-Hong Lee, and Philip Heng Wai Leong. A smith-waterman systolic cell. In *FPL*, pages 375–384, 2003.