VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

SIMULACE A OPTIMALIZACE METOD DNA VÝPOČTŮ

SIMULATION AND OPTIMIZATION OF DNA COMPUTATIONS METHODS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. LUKÁŠ PLEVAČ**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

153917

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Plevač Lukáš, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Bioinformatika a biocomputing
Název:	**Simulace a optimalizace metod DNA výpočtů**
Kategorie:	Bioinformatika
Akademický rok:	2023/24

Zadání:

1. Seznamte se s teoretickými základy a postupy DNA výpočtů aplikovatelnými na řešení různých úloh v oblasti informatiky (např. problému hledání hamiltonovské cesty v grafu, SAT problému apod.). Vypracujte přehledovou studii na toto téma.
2. Navrhněte a ve zvoleném prostředí implementujte systém pro simulaci DNA výpočtu pro řešení vybrané úlohy. Do systému integrujte vhodné prvky poskytované moderními technologiemi za účelem zvýšení jeho efektivity a výkonnosti, např. (masivně) paralelní zpracování, HW akcelerace apod.
3. S využitím systému z předchozího bodu proveďte sadu experimentů pro různá nastavení parametrů systému a různé instance dané úlohy.
4. Vyhodnoťte schopnosti navrženého řešení s ohledem na různé ukazatele (např. výkonnost, paměťová náročnost, kvalita dosažených výsledků apod.).
5. Diskutujte vlastnosti výsledného systému a možnosti další činnosti v této oblasti.

Literatura:

- Dle pokynů vedoucího projektu.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1-3 zadání, demonstrace prototypu aplikace z bodu 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Bidlo Michal, doc. Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2023
Termín pro odevzdání:	17.5.2024
Datum schválení:	30.10.2023

Abstrakt

Tato práce se zaměřuje na vytvoření programu pro simulaci architektury výpočtů SIMD||DNA a následné využití této simulace k navrhování nových algoritmů pro tuto architekturu, jako jsou například posuvný registr, 3-Stavový celulární automat nebo LFSR registr. SIMD||DNA patří mezi výpočetní architektury v oblasti DNA výpočtů, což je netradiční způsob výpočtů zcela odlišný od dnešních elektronických počítačů. Hlavním principem DNA výpočtů je využití vlastností DNA pro zpracování informací. Tato metoda přináší výhody v energetické efektivitě a masivním paralelismu teoreticky umožňujícím překonat současné limity zpracování informací. Taktéž nabízí vyšší hustotu ukládání informací, což vedlo k vzniku nového typu úložišť známých jako DNA digital data storage. SIMD||DNA je architekturou, která se snaží provádět výpočty s takto uloženými daty.

Abstract

This work focuses on creating a program for simulating the SIMD||DNA computation architecture and subsequently utilizing this simulation to design new algorithms for this architecture, such as shift registers, 3-state cellular automata, or LFSR registers. SIMD||DNA belongs to the field of DNA computing architectures, which represent an unconventional computing method entirely different from today's electronic computers. The main principle of DNA computing involves leveraging DNA properties for information processing. This method offers advantages in energy efficiency and massive parallelism, theoretically capable of surpassing current limits in information processing. It also provides higher information storage density, leading to the emergence of a new type of storage known as DNA digital data storage. SIMD||DNA is an architecture aimed at performing computations with data stored in this manner.

Klíčová slova

SIMD||DNA, DNA počítání, SAT, DNA operace, Toehold mediated strand displacement

Keywords

SIMD||DNA, DNA computing, SAT, DNA operations, Toehold mediated strand displacement

Citace

PLEVAČ, Lukáš. *Simulace a optimalizace metod DNA výpočtů*. Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Michal Bidlo, Ph.D.

Simulace a optimalizace metod DNA výpočtů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. MICHALA BIDLA, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Lukáš Plevač
15. května 2024

Poděkování

Chtěl bych tímto poděkovat vedoucímu práce doc. Ing. MICHALU BIDLOVI, Ph.D. za vedení a poskytnutí cenných rad a nápadů, které mi pomohly tuto práci vytvořit.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Hlavní myšlenkou DNA počítání je použití DNA a jeho vlastností pro zpracování informací (provádění výpočtů). Zjednodušeně se dá říci, že se snaží z křemíku výpočet převést na uhlík, z mikročipů na molekuly. Schopnosti organických molekul lze v počítačích využít jako náhradu za přepínací primitiva [16].

U současných technologií počítačů jsou zřejmé limity v jejich minimalizaci (jak energeticky tak i fyzicky). Z tohoto důvodu se již delší dobu uvažuje o zmenšení komponent počítačů na molekulární úroveň. Počítače s takovými komponentami by mohly být mnohem menší než je tomu dnes a zároveň by mohly být i energeticky úspornější. Mezi hlavní způsoby tohoto zmenšení v dnešní době patří kvantové počítače a DNA počítače [16].

V současnosti se po celém světě pokračuje ve zkoumání těchto cest. DNA počítání jako jedna z cest, má velké výhody ve své energetické nenáročnosti, která je doprovázena masivním paralelismem, který teoreticky umožňuje překonat současné limity zpracování informací. Díky těmto vlastnostem je DNA počítání teoreticky schopno vyřešit NP-úplné problémy v polynomiálním čase, což je zásadní zvrat, který nám teoreticky umožní vypočítat problémy, které bychom s dnešní technologií řešily stovky let. Další nemenší výhodou je hustota záznamu informací, která je několikanásobně vyšší než u současných technologií, což vedlo k vytvoření nového typu úložišť, která dnes označujeme jako DNA digital data storage. Tento zvrat má i svou stinnou stránku, kterou je schopnost prolomení většiny současných šifer, které stojí právě na složitosti NP-úplných problémů [16].

DNA počítače jsou v dnešní době nádobou s materiálem (DNA), který provádí výpočet. Lze říci, že pouze „vložíme vstup do zkumavky", pak chvíli počkáme a zase „ze zkumavky přečteme" výstup. Tento proces může dělat člověk nebo lze postup automatizovat použitím, vhodných konvenčních technologií [16].

Hlavní naděje v budoucnost DNA počítání je založena na dvou fundamentálních vlastnostech, kterými jsou masivní paralelismus mezi DNA vlákny a Watson-Crick komplementarita. Tyto vlastnosti zajišťují fungování celého mechanizmu DNA počítání. Algoritmy tohoto DNA počítače na těchto vlastnostech staví [16].

Spoustu výpočetních problémů lze vyřešit hrubou silou. Tím je myšleno, že projdeme všechny možnosti našeho řešení. Například když se snažíme prolomit heslo, zkusíme všechny jeho kombinace, kterých mohou být miliardy. Tento způsob nicméně u spousty problémů s dnešní technologií není možný, protože by trval příliš dlouho a my nemůžeme tak dlouho čekat. V případě DNA počítání hustota záznamu informací v DNA a jednoduchost vytvoření kopií DNA pomocí PCR může vyústit v prohledávání všech možných řešení našeho problému díky masivnímu paralelismu mezi DNA vlákny. V našem příkladě s prolamováním hesla by to znamenalo možnost „vyzkoušení všech kombinací hesla najednou"[16].

Průkopníkem tohoto oboru byl Leonard Adleman, který jako první v roce 1994 demonstroval výpočetní potenciál DNA. Ve své práci On Constructing A Molecular Computer popsal způsob řešení hamiltonovské cesty v grafu v polynomiálním čase [4]. Tento postup používal především zmíněné principy komplementarity a masivního paralelismu. Po této práci bylo představeno řešení různých problémů pomocí DNA. Dalším významným řešením bylo řešení SAT problému pomocí Liptonova algoritmu, který byl představen v práci On the computational power of DNA [9]. Tento algoritmus ukázal sílu DNA počítaní, které dokázalo, že by bylo schopné prolomit šifrovací algoritmus DES v polynomiálním čase. Nicméně pro šifry založené na složitosti NP-úplných problémů bylo později dokázáno, že spotřeba materiálu u těchto metod roste příliš rychle a způsobuje nemožnost aplikovat takovéto algoritmy na reálné šifry [16].

DNA počítání se v dnešní době dělí do několika směrů podle metod, které používají na úpravu řetězců. Mezi tyto metody patří metody využívající nahrazení řetězců [45], metody Chemical reaction networks (CRNs) [42] [44], metody využívající bílkoviny [7] [8] a metody využívající algoritmické samosestavení [40]. Každý z těchto směrů se lehce liší v tom, jak pracuje s DNA řetězci, nicméně všechny tyto metody staví na principu komplementarity a principu masivního paralelismu. Tyto metody lze kombinovat mezi sebou, například větší část algoritmu bude využívat metody nahrazení řetězců, ale pro pár specifických situací použije bílkoviny pro úpravu DNA.

SIMD||DNA provádí výpočet podobně jako architektura SIMD u paralelních počítačových systémů. Architektura SIMD (Single Instruction, Multiple Data) staví na základu provedení jedné instrukce nad mnoha daty paralelně. Podobně funguje i SIMD||DNA, která používá DNA řetězce jako instrukce a jiné DNA řetězce jako datové registry, které instrukční řetězce upravují. Aby SIMD||DNA mohlo fungovat správně, používá ještě kromě principu komplementarity a principu masivního paralelismu mezi DNA vlákny navíc i princip nahrazení řetězců [45].

Za pomoci tohoto principu je SIMD||DNA schopno řešit libovolné úlohy nad zadanými daty, pokud mu dáme vhodnou reprezentaci dat a posloupnost instrukcí na jejich zpracování. Tento systém je turingovsky výpočetně úplný, což je dokázáno implementací celulárního automatu s pravidlem 110 v původním článku od autorů Boya Wang, Cameron Chalk a David Soloveichik, který představil SIMD||DNA [45]. Tento koncept byl dále rozšířen o implementaci 3-znakového Turingova stroje ve článku Simulationg 3-Symbol Turing Machines with SIMD||DNA od autorů David Doty a Aaron Ong [14]. Z tohoto důvodu je možné s tímto modelem počítat jako s výpočetně úplným a je proto pro něj velmi vhodné začít vytvářet optimalizované specializované algoritmy, jako je například LFSR registr, které by tuto výpočetní architekturu mohly rozšířit.

Cílem této práce je navrhnout simulátor pro architekturu SIMD||DNA a následně jej využít pro implementaci algoritmů v SIMD||DNA, a to konkrétně posuvného binární registru, třístavového celulárního automatu a LFSR registru. Dále v rámci simulátoru bude nutné implementovat vhodnou paralelizaci pomocí OpenMP, dopočtení pravděpodobnosti chyb DNA operací a hardwarovou akceleraci pomocí FPGA, která v případě nutnosti může být nahrazena akcelerací na GPU pomocí OpenACC nebo OpenMP. Prioritou práce je návrh nových algoritmů pro SIMD||DNA.

Kapitola 2

Základy molekulárni biologie

Jaký je význam molekulární biologie? Molekulární biologie vychází z oblastí genetiky a biochemie. Tento termín má několik interpretací. Někteří ji definují jako snahu pochopit biologické jevy na molekulární úrovni. Avšak tato interpretace může být zavádějící, jelikož se tím příliš neliší od oblasti biochemie. Jiný výklad ji definuje jako zkoumání struktury a funkce genů na molekulární úrovni. Zaměření na vysvětlování genetických mechanismů je hlavním cílem této kapitoly. Na začátku této kapitoly se podíváme na hlavní počátky této disciplíny, začínaje prvními genetickými experimenty Gregora Mendela v polovině devatenáctého století.

2.1 Stručná historie

2.1.1 Začátky genetiky

V roce 1865 Gregor Mendel zveřejnil své poznatky o dědičnosti vlastností hrachu zahradního [26]. Před Mendelovým výzkumem se vědci domnívali, že dědičnost vzniká smísením vlastností rodičů u potomstva. Mendel však dospěl k závěru, že dědičnost probíhá částicově. To znamená, že každý rodič předává potomkovi určité částice nebo genetické jednotky, které dnes označujeme jako geny. Díky pečlivému počítání počtu potomků s konkrétním fenotypem (například žlutá semena, bílé květy) Mendel učinil důležité obecné závěry. Termín fenotyp pochází z řeckého slova phenomenon, což označuje vzhled.

Mendel zjistil, že gen může existovat v různých formách nazývaných alely. Například hrách může mít buď žlutá nebo zelená semena. Jedna alela genu pro barvu semene vede k žlutým semenům, druhá k zeleným. Navíc jedna alela může být dominantní vůči druhé, recesivní alele. Mendel prokázal dominanci alely pro žlutá semena, když zkřížil hrách zelený s hrachem žlutým. Všichni potomci první generace měli žlutá semena. Nicméně, když tyto žluté hrachy byly samooplozeny, objevily se opět některé hrachy se zelenými semeny.

Mendel došel k závěru, že alela pro zelená semena musela být zachována v první generaci, i když se to neprojevilo ve zbarvení těchto hrachů. Žlutý rodič přispěl gametou s genem pro žlutá semena. Zelený rodič gametou s genem pro zelená semena. Všechny hrachy první generace dostaly jednu alelu pro žlutá semena a jednu alelu pro zelená semena. Neztratily tedy alelu pro zelená semena, ale kvůli dominanci žluté barvy byla všechna semena žlutá. Nicméně, když byly tyto hrachy samooplozeny, vytvořily gamety obsahující stejný počet alel pro žluté a zelené barvy, což umožnilo znovu objevení se zeleného fenotypu [43] [47].

2.1.2 Po roce 1900

Až do roku 1900 byla práce Gregora Mendela zavrhována a ignorována do doby, kdy v roce 1900 tři botanici, nezávisle dospěli k podobným závěrům, a tím teorii znovuobjevili [39]. V této době se objevila představa, že chromozomy nesou geny. Tato teorie se nazývá Chromozomová teorie dědičnosti. Byl to zásadní nový krok v genetickém myšlení. Geny již nebyly nehmotnými faktory. Teď byly pozorovatelnými objekty v jádře buňky. Někteří genetikové, zejména Thomas Hunt Morgan, zůstávali skeptičtí k této myšlence. Ironií osudu je že, právě Morgan v roce 1910 poskytl první definitivní důkaz pro chromozomovou teorii, důkazem který provedl na octomilce obecné [30].

Teorie chromozomální dědičnosti zásadně změnila náš pohled na přenos vlastností mezi generacemi. Geny, nositelé dědičných informací, skutečně leží na chromozomech, což jsou struktury v buňkách, jež obsahují DNA.

Tato teorie vysvětluje, proč určité vlastnosti často putují společně, jelikož jsou umístěny na stejném chromozomu. Nicméně, proces rekombinace během dělení buněk (meiózy) hraje klíčovou roli v rozmanitosti genetického materiálu. Při tomto procesu se genetický materiál mezi chromozomy může přeskupovat a vytvářet nové kombinace alel, které nebyly přítomné v původních rodičovských chromozomech. Pravděpodobnost rekombinace mezi dvěma geny je ovlivněna fyzickou vzdáleností mezi nimi na chromozomu, to znamená čím jsou dále, tím je pravděpodobnější, že dojde k rekombinaci.

Tento princip vazby a rekombinace poskytuje základ pro porozumění genetickým vzorům dědičnosti a tomu, jak se vlastnosti mohou dědit nezávisle nebo společně v závislosti na jejich relativním umístění na chromozomech [43] [47].

2.1.3 Začátky Molekulární Genetiky

V roce 1869 objevil Friedrich Miescher v jádře buňky směs látek, kterou nazval nuklein [28] [29]. Hlavní složkou nukleinu je deoxyribonukleová kyselina (DNA). Do konce devatenáctého století chemici získali obecnou strukturu DNA a související sloučeninu kyselinu ribonukleovou (RNA).

V této době ukázali biochemikové, že všechny živé organismy provádějí nespočet chemických reakcí, a že tyto reakce jsou urychlovány, neboli katalyzovány bílkovinami. Mnoho těchto reakcí probíhá postupně, takže jeden chemický produkt se stává výchozím materiálem, neboli substrátem pro další reakci.

Spojitost mezi geny a bílkovinami prokázali v roce 1941 George Beadle a E. L. Tatum [6]. Použili k tomu plíseň Neurospora jako svůj experimentální systém. Jejich genetické experimenty ukázaly, že vadný gen dává vadnou bílkovinu. Jinými slovy jeden gen se zdál být zodpovědný za tvorbu jedné bílkoviny.

V roce 1953 Watson a Crick [46] navrhli, že DNA je dvojitá šroubovice, tedy se jedná o dva řetězce DNA spojené dohromady. Důležité je, že báze každého řetězce jsou uvnitř šroubovice a báze jednoho řetězce se páruje s bází druhého řetězce velmi specifickým způsobem. DNA má pouze čtyři různé báze: adenin, guanin, cytosin a thymin, které zkracujeme jako A, G, C a T. Kdekoliv nalezneme A v jednom řetězci, vždy nalezneme T v druhém, kdekoliv nalezneme G v jednom řetězci, vždy nalezneme C v druhém. Tyto dva řetězce jsou tedy komplementární. Pokud známe sekvenci bází jednoho, automaticky známe sekvenci druhého. Tato komplementarita umožňuje, že se DNA může replikovala. Dva řetězce se oddělí a bílkoviny pro ně vytvoří nové partnery používající staré řetězce jako šablony a následující pravidla párování bází podle Watsona a Cricka (A s T, G s C).

Francis Crick původně předpokládal, že každý ribozom byl schopen vytvořit pouze jeden druh bílkoviny - ten, který je zakódován v něm. François Jacob a Sydney Brenner měli jiný názor a to ten, že ribozomy jsou nespecifické stroje na translaci, které mohou vytvářet neomezený počet různých bílkovin podle instrukcí v RNA, které čtou ribozomy. Experimenty ukázaly, že tato představa je správná.

Marshall Nirenberg a Gobind Khorana v 60. letech odhalili základy genetického kódu [33]. Ukázali, že 3 báze tvoří kodon, který reprezentuje jednu aminokyselinu z 64 možných kombinací (4^3 - čtyři různé typy nukleotidů na třech pozicích). Tři z těchto kodonů slouží jako signály pro zastavení překladu. Z aminokyselin se později skládají bílkoviny.

Genetici od sedmdesátých let dokázali izolovat a klonovat geny do nových organismů. Klonované geny nejen poskytují výzkumníkům dostatek materiálu pro studium, ale mohou být i využity k produkci nových bílkovin [43] [47].

2.1.4 Projekt lidského genomu

Koncem 20. století se započal projekt lidského genomu [11]. Cílem tohoto projektu bylo osekvenovat (sekvenování je způsob přečtení DNA/RNA jako řetězec bází) celý lidský genom (genom je souhrn všech genů a jiných genetických materiálů určitého organismu). Lidský genom čítá okolo tří miliard nukleotidových párů, takže tento projekt byl velmi finančně i časově náročný. Vyvrcholením tohoto projektu byla publikace z roku 2001, která obsahovala 2,7 miliard nukleotidových párů lidské DNA. Dokončení tohoto projektu bylo významným milníkem v tomto oboru a byl umožněn především díky výraznému pokroku v technologii sekvenování, robotice a informatice. Tento projekt odstartoval spoustu dalších sekvenačních projektů, které již kromě modelových organizmů zkoumaly genomy rozličných organizmů jako je například včela či komár. Geny díky možnostem sekvenování mohou být nyní snadno studovány na molekulární úrovni a dokonce i současně. Tento způsob studia vytvořil nový obor jménem Genomika. Z důvodu potřeby studování sekvencí genů a jejich funkcí vznikly rozsáhlé databáze obsahující tyto informace [47].

2.2 DNA jako genetický materiál

2.2.1 Replikace DNA

Genetický materiál organizmu je v přírodě přenášen během buněčného dělení z mateřské buňky do jejich dceřiných buněk. Tento proces je založen na schopnosti replikace dvojvláknové DNA. Replikace je jako proces neobvykle přesná, nicméně i přesto může dojít k chybě.

Samotný proces replikace je založen na principu komplementarity vláken. Vlákna uvnitř dvojšroubovice jsou pohromadě držena celkem slabými vodíkovými vazbami, které lze poměrně snadno rozdělit. Pokud dojde k oddělení vláken od sebe mohou tato oddělená vlákna sloužit jako předlohy (templáty) pro syntézu nových vláken pomocí principu komplementarity. Při replikaci se na nukleotidy na templátech postupně přidávají jim komplementární nukleotidy a tím vznikají nové dvojvláknové molekuly, které budou kopií původní molekuly.

Proces replikace neprobíhá samostatně, ale je katalizován enzymy. Enzymy jsou specifické typy bílkovin, které katalizují (urychlují) chemické reakce [43] [47].

2.2.2 Genová exprese

Molekula DNA obsahuje informaci k ovládání aktivit buňky. Tato informace je zakódována v sekvenci nukleotidů, které tvoří genom. V genomu je tato informace uložena uvnitř kódující části v blocích, které označujeme geny. Jedna molekula DNA může obsahovat tisíce různých genů. Geny obsahují informaci na sestavení bílkovin, které ovládají aktivity buňky. Každá bílkovina se skládá z jednoho nebo více řetězců aminokyselin. Tyto řetězce nazýváme polypeptidy. Sekvence aminokyselin v polypeptidu je určena sekvencí základních kódujících jednotek v genu, které nazýváme kodony. Kodony jsou trojice nukleotidů, které v DNA leží za sebou. Každý kodon kóduje jednu aminokyselinu v polypeptidu.

Genová exprese je proces který vede k vytvoření polypeptidu. Jedná se o proces skládající se ze dvou částí. Nejprve je informace z DNA přeložena do RNA pomoci principu komplementarity. Tento proces se nazývá transkripce a jeho výstup se nazývá transkript. Tento traskript může být upraven post-transkripčními úpravami, jako je sestřih či editace, při kterých může dojít k úpravě RNA. Dokončená molekula se nazývá mRNA.

Druhá fáze převede mRNA na sekvenci aminokyselin. Tato fáze se nazývá translace. V této fázi funguje mRNA jako templát pro syntézu polypeptidu. Každý z kodonu daného genu je nyní přítomen v sekvenci mRNA a určuje začlenění aminokyseliny v rámci polypeptidu. Kodony se nyní čtou v pořadí, ve kterém jsou na mRNA a přidávají se do polypeptidu. Po dokončení syntézy se polypeptid oddělí od mRNA a začně se sbalovat do své trojrozměrné podoby, aby mohl vykonávat činnost, ke které je určen [43] [47].

Ústřední dogma molekulární biologie

Z průběhu genové exprese plyne ústřední dogma molekulární biologie, které říká, že z DNA může být syntetizována DNA (replikace), nebo může být transkripcí změněna na RNA. RNA může být změněna na polypeptid nebo zpět na DNA pomocí reverzní transkripce (tento princip je důležitý pro některé viry, mezi které patří i vir způsobující AIDS [20]). Nicméně nelze převést polypeptid na RNA či DNA [43] [47].

2.2.3 Mutace

Replikace DNA je sice velmi přesný proces, ale i přesto je možné že se stane chyba. Takováto chyba je sice velmi málo pravděpodobná, ale i přesto můžeme kolem sebe pozorovat její následky. Takové změny mohou vést k poškození nebo záměně informace uložené v genech. Molekuly DNA mohou být také poškozeny elektromagnetickým zářením nebo chemikáliemi. Přestože takovéto změny mohou být opraveny opravnými procesy, které si buňky vyvinuly, v průběhu opravného procesu může dojít k nedokonalému výsledku. Sekvence nukleotidů mohou být těmito vlivy odstraněny, zdvojeny, nebo mohou být přeuspořádány. Tyto typy změn označujeme jako mutace. Geny, které byly změněny mutacemi, nazýváme mutantními geny. Tyto mutantní geny často vedou k změnám fenotypu jedince. Vzhledem k tomu, že se mutace v organizmech v průběhu generací hromadí, mohou vést tyto mutace k rozdílům mezi organizmy. Lze proto říci, že mutace jsou důležité pro vývoj organizmů v průběhu generací a stojí za diverzitou prostředí kolem nás [43] [47].

2.3 Struktura DNA a RNA

2.3.1 DNA

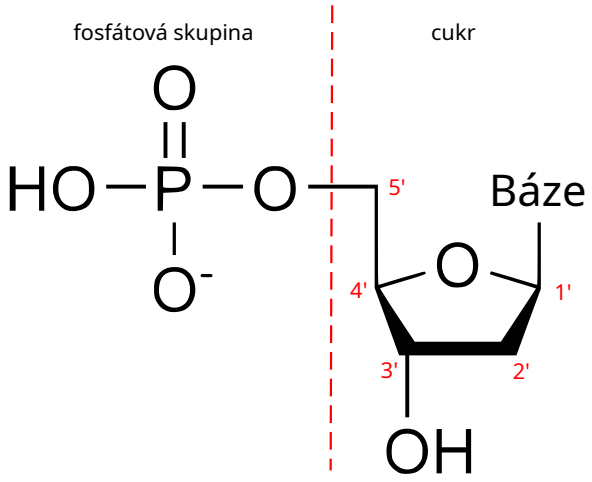
DNA je důležitou molekulou uvnitř živých buněk (in vivo) a její struktura umožňuje dvě základní funkce. Těmi jsou schopnost zápisu (kódování) postupu výroby bílkovin a vytvoření přesné kopie sebe sama (replikace), která může být přenesena na potomka. Tyto dvě vlastnosti jsou nezbytné pro fungování a přežití buněk. DNA (DeoxyriboNucleic Acid) molekula se skládá z monomerů, přesněji z deoxyribonukleotidů, přičemž každý deoxyribonukleotid se skládá z tří komponent [43] [16]:

- cukr (2-deoxyribóza),
- fosfátová skupina,
- nukleová báze.

Pro zjednodušení naší terminologie budeme nazývat deoxyribonukleotid nukleotidem. Cukr v nukleotidu má pět uhlíkových atomů, z důvodu zpřehlednění se tyto atomy uhlíku označují od 1' do 5'. Fosfátová skupina je napojena na 5' uhlík a nukleotidová báze na 1' uhlík. Uvnitř cukru je hydroxylová skupina (OH), která je připojena na 3' atom uhlíku. Struktura deoxyribonukleotidu je na obrázku 2.1. Rozdílné nukleotidy se liší pouze v jejich nukleotidové bázi, což je rozděluje na dvě skupiny, přičemž každá z těchto skupin má dva zástupce [16].

- puriny - Adenin (A) a Guanin (G),
- pyrimidiny - Citosin (C) a Thymin (T).

Protože se jednotlivé nukleotidy od sebe liší jen v nukleotidové bázi, označujeme je jednoduše jako A, G, C, nebo T tedy podle jejich nukleotidové báze [43] [47] [16].



Obrázek 2.1: Chemická struktura deoxyribonukleotidu s označenými atomy uhlíku

2.3.2 RNA

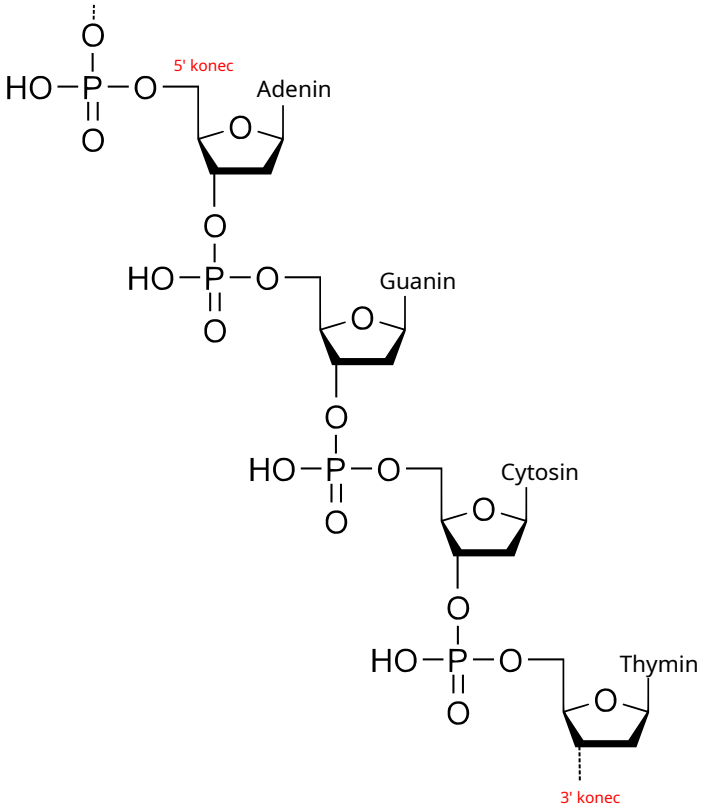
RNA je další molekula, která je zásadní pro živé buňky. Její struktura je velmi podobná struktuře DNA. RNA je složena z ribonukleotidů na rozdíl od DNA, která je složena z deoxyribonukleotidů. Ribonukleotidy se od deoxyribonukleotidů liší v následujících aspektech [43] [16]:

- Obsahují ribózu jako cukr, který se liší od 2-deoxyribózy,
- Thymin je zaměněn za Uracil, která se značí jako U. Takže možné báze v RNA jsou A, U, C a G.

2.3.3 Spojování nukleotidů

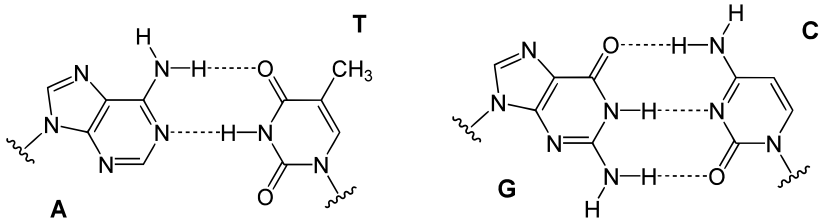
Spojování jednotlivých nukleotidů může probíhat dvěma rozdílnými způsoby [43] [16].

1. Fosfátová skupina jednoho nukleotidu se může napojit na 3' atom cukru jiného nukleotidu. Tím vzniká fosfodiesterová vazba, což je velmi silná vazba dvou molekul a vzniká pomocí ní primární struktura DNA. Při tomto spojení nám na jedné straně zůstane volný 3' atom cukru a na druhé straně nám zůstane volný 5' atom cukru. Tyto rozdílné konce dávají molekule směr. Můžeme tedy mluvit o 5' - 3' směru a 3' - 5' směru. Tento směr je velmi důležitý pro pochopení operací nad DNA a je i velmi důležitý pro DNA počítání. Na obrázku 2.2 je znázorněna molekula skládající se z 4 nukleotidů, která ve směru 5' - 3' reprezentuje sekvenci AGCT a ve směru 3' - 5' sekvenci TCGA. Podle konvence se pořadí nukleotidů zapisuje směrem 5' - 3', takže se jedná o AGCT [43] [16].



Obrázek 2.2: Chemická struktura jednoho vlákna DNA AGCT

2. Báze jednoho nukleotidu reaguje s bází druhého nukleotidu a vzniká spojení pomocí vodíkové vazby. Vodíková vazba je slabou vazbou a díky tomu může být snadněji přetržena. Spojování bází mezi sebou probíhá podle pravidel pro spojování bází, protože ne všechny báze budou mezi sebou interagovat. Pravidla pro párování jsou následující: A s T a C s G, jiné páry vzniknout nemohou. Tento párovací princip se nazývá Watson-Crick komplementarita a je pojmenován po Jonas D. Watson a Francis H. C. Crick, kteří odvodili dvojitou šroubovici jako strukturu DNA v roce 1953. Princip tohoto párování je ukázán na obrázku 2.3 [43] [16].



(a) Párování A s T (b) Párování G s C

Obrázek 2.3: Párování jednotlivých nukleotidů pomocí vodíkové vazby. Převzato z [48]

2.3.4 Jednovlákna a dvojvlákna

Jak víme DNA připomíná svým tvarem dvojšroubovici. Ze sekce 2.3.3 již víme, že části DNA zvané nukleotidy, mají schopnost spojovat se a díky tomu vytvořit vetší celek. Pomocí fosfodiesterové vazby jsme schopni skládat nukleotidy za sebe a tím vytvářet řetězce. Pomocí vodíkové vazby jsme schopni skládat nukleotidy naproti sobě a díky tomu jsem schopni k jednomu řetězci vytvořit druhý řetězce, který s ním bude spojen. Takže nyní jsme schopni poskládat dvojvlákno DNA pomocí těchto pravidel. Například pokud budeme chtít sestavit DNA sekvenci 5' - ACG tak již víme, že u 5' konce bude A které bude spojeno pomocí fosfodiesterové vazby s C a to bude spojeno pomocí fosfodiesterové vazby s G, které bude mít 3' konec. Také víme že komplementární řetězec k ACG je TGC. Díky tomu víme, že druhý řetězec, který se pomocí vodíkových vazeb s prvním spojí bude TGC. Jen je ještě nutné zmínit, že komplementární řetězec bude mít přesně opačné konce (opačný směr), než zdrojový. To znamená že se bude jednat o 3' - TGC [43] [16].

Je důležité poznamenat, že vodíkové vazby jsou tak slabé, že by dva nukleotidy u sebe neudržely. Z tohoto důvodu je nutné, aby řetězce, které u sebe drží vodíkovými vazbami, byly dostatečně dlouhé a tím došlo k zesílení této vazby. Tomuto jevu se říká kumulativní efekt vodíkových vazeb mezi komplementárními bázemi DNA [43] [16].

DNA stejně jako RNA může existovat ve formě dvojvlákna (**d**ouble **s**tranded) nebo jednovlákna (**s**ingle **s**tranded). V přírodě se častěji setkáme s dvojvláknovým DNA (dsDNA) a jednovláknovým RNA (ssRNA) ostatní variaty jsou méně časté (ssDNA a dsRNA) [43] [16].

2.3.5 Dvojšroubovice DNA

V části 2.3.4 jsme vytvořily dvojvláknovou DNA, nicméně tato lineární podoba DNA řetězců je stále velkým zjednodušením reality, protože v DNA molekule jsou dva řetězce kolem sebe otáčeny a tím vznikne DNA dvojšroubovice zobrazená na obrázku 2.4 [16].

Obrázek 2.4: Dvojšroubovice DNA. Obrázek vytvořen programem PyMol

In vivo je situace mnohem komplikovanější, protože velmi dlouhá DNA molekula se musí vejít do malé buňky. Například pro většinu bakterií platí, že jejich „rozvinutá" DNA je 10000x vetší než samotná bakterie. Z tohoto důvodu se buňky naučily DNA sbalovat a tím ji udělat menší. Toto balení je u složitějších buněk dokonce děláno hierarchicky v několika fázích. Nicméně pro DNA počítaní není nutné nad DNA uvažovat takto komplexně a můžeme se na něj dívat jako na dva lineární řetězce. Také je nutné poznamenat, že některé bakterie nemají DNA jako lineární dvojšroubovici, ale mají ji kruhovou, toho dosáhly tak, že 5' konec spojili s 3' koncem pomocí fosfodiesterové vazby [43] [16] [47].

2.4 Nástroje pro studium DNA

2.4.1 Měření délky DNA molekuly

Délka molekuly jednovláknové DNA odpovídá počtu nukleotidů v molekule. Pokud molekula obsahuje 12 nukleotidů, tak poté říkáme, že její dálka je 12 mer (12 monomers). Délku dvojvláknové DNA měříme v počtu bázových párů (počet nukleotidů dělený dvěma). Tedy pokud z 12 mer jednovlákna uděláme pomocí komplementarity dvojvlákno bude měřit 12 bp (base pairs). Pokud by naše vlákno mělo 12 000 bp, můžeme psát, že se jedná o 12 kbp [43] [16].

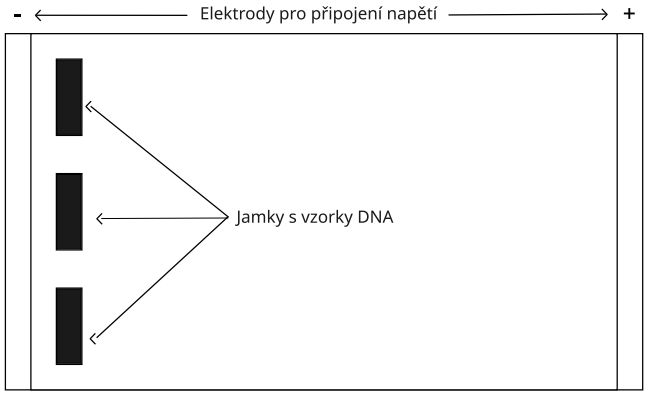
Pro změření délky DNA je možné použít gelovou elektroforézu. Elektroforéza je technika založena na faktu, který říká že, DNA je jako molekula záporně nabita a díky tomu, pokud molekuly DNA budou vsazeny do elektrického pole, budou se pohybovat ke kladnému pólu. Přičemž délka molekuly DNA je přímo úměrná jejímu elektrickému náboji a zároveň síla, která je nutná k rozpohybování molekuly, je přímo úměrná její délce. V ideální situaci se

tyto dvě síly vyruší a všechny molekuly se budou pohybovat stejnou rychlostí. Z tohoto důvodu potřebujeme gel, abychom tuto rovnováhu narušily [43] [16].

Metoda [16]

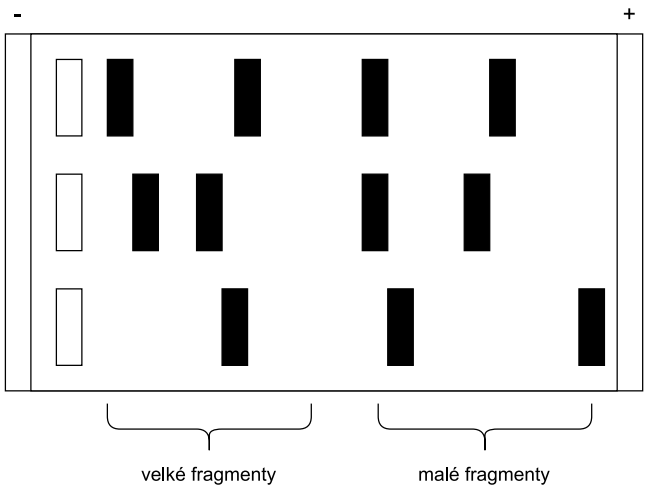
1. Gelový prášek je zahříván a tím vzniká gel, který je umístěn do skleněného nebo plastového kontejneru. Před zchladnutím se do gelu hřebínkem udělají drobné dírky u záporného pólu.

2. Po zchladnutí se do jamek od hřebínku umístí opravdu malé množství našich DNA vzorků (do jedné dírky lze umístit i více vzorků, ale nebude možné rozpoznat, který z nich je ten kratší) a poté aktivujeme elektrické pole.



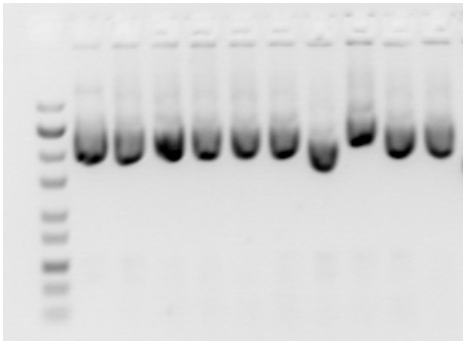
Obrázek 2.5: Schéma zařízení pro gelovou elektroforézu DNA

3. Nyní všechny DNA molekuly cestují od záporného pólu ke kladnému. Menší molekuly cestují rychleji, protože jim gel méně brání v pohybu než molekulám velkým. Jakmile dojde první molekula ke kladnému konci deaktivujeme elektrické pole.



Obrázek 2.6: Schématický nákres přílkadu dokončené elektroforézy nad různě dlouhými vzorky DNA

4. Je jasné že nejmenší molekuly v daném časovém okamžiku docestovaly nejdále a větší tolik cesty neurazily. Nyní tedy stačí jen zjistit délky molekul. Pro tento výpočet můžeme použít délku dobře známých fragmentů. Některé řádky mohou být použity jako kalibrační a potom může být jejich vzdálenost porovnána se vzdáleností molekuly s neznámou velikostí. Na obrázku 2.7 je umístěna kalibrační cesta. Bez kalibrační cesty by bylo velmi složité poznat délku neznámých fragmentů, proto typicky alespoň jeden řádek odpovídá kalibrační cestě.



Obrázek 2.7: Fotografická ukázka dokončeného procesu gelové elektroforézy s kalibrační cestou (vlevo)

Protože DNA nemá žádnou barvu a v gelu bychom ji nemohly vidět, je nutné ji nějak obarvit před tím, než jej umístíme do gelu [43] [16].

Molekuly DNA zůstávající v gelu po elektroforéze lze zachránit, pokud je to nutné. Například pokud vyřízneme malou část DNA z gelu a tento gel pak zamrazíme (pomocí kapalného vodíku), gel se rozpadne. Následně stačí pomocí centrifugy a speciálního filtru DNA odfiltrovat [16].

2.4.2 Hledání konkrétních sekvencí (metoda FISH)

Tato technika slouží na hledání konkrétních sekvencí v DNA pomocí principu komplementarity. Pomocí této techniky tedy můžeme říci, zda se nějaká sekvence nukleotidů v DNA nachází nebo ne [16].

meteda [16]

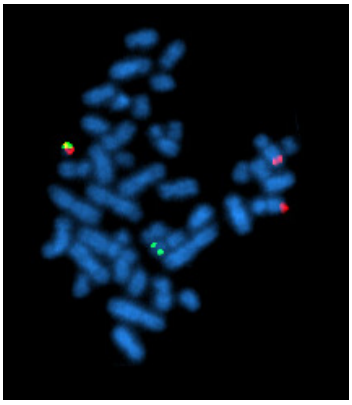
Nechť sekvence S je sekvencí/sekvencemi, ve které/kterých hledáme, a sekvence a, je sekvencí kterou hledáme.

1. Pokud je S dvojvláknová provedeme denaturaci S (viz sekce 3.1.1).
2. Pokud je a dvojvláknová provedeme denaturaci a.
3. Vytvoříme $K(a)$, která obsahuje komplementarní sekvenci k sekvenci a.
4. Nakopírujeme $K(a)$ do velkého počtu (tyto sekvence se nazývají DNA sondy). Tato část je zajištěna pomocí techniky PCR, která je popsána v sekci 3.1.7.
5. Filtrujeme S pomocí $K(a)$ - Za pomoci komplementarity se $K(a)$ naváže na hledané podsekvence v S a ostatní vynechá.

6. Nyní máme jednovláknovou sekvenci, která je místy dvojvláknová a právě v místech, kde je dvojvláknová, se nachází hledaná sekvence.

7. Následně záleží na postupu dané metody a také, zda jsme pracovali s mnoha malými sekvencemi, mezi kterými jsme hledali nějaké konkrétní, nebo zda jsme měli dlouhou sekvenci a v ní jsme hledali nějakou podčást. V případě hledání konkrétní sekvence je možné dvojvlákna přemístit do jiného kontejneru, tam DNA denaturovat a odstranit filtr. Následně dostaneme jen hledané sekvence nebo žádné, pokud hledané sekvence v S nejsou. Pokud máme sekvenci, ve které hledáme podsekvenci, je vhodné zkusit tyto dvojřetězce vystřihnout pomocí bílkovin nebo na sekvence $K(a)$ lze umístit fluorescentní barvu (Metoda FISH).

Tato metoda je použitelná jak na jednu dlouhou sekvenci, ve které hledáme podsekvence, tak na spoustu malých sekvencí, mezi kterými hledáme, zda tam je jedna konkrétní nebo ne. V případě metody FISH je možné vyhledávat několik sekvencí současně a každou obarvit jinou barvou (typicky maximálně pět barev). V takovém případě je možné porovnávat pozice jednotlivých podsekvencí vůči sobě, jako je tomu na obrázku 2.8 [38].



Obrázek 2.8: Obrázek vytvořený metodou FISH ukazující umístění genů BCR a ABL.

2.4.3 Čtení DNA (sekvenování)

Ne vždy si vystačíme se zjištěním délky DNA nebo s informací, zda se konkrétní sekvence v DNA vyskytuje nebo ne. V některých případech je nutné znát celou sekvenci DNA. To bylo i cílem projektu lidského genomu, který byl zmíněn v části 2.1.4. Sekvenování DNA (též sekvenace či sekvencování) je souhrnný termín pro biochemické metody, jimiž se zjišťuje pořadí nukleových bází (A, C, G, T) v sekvencích DNA. V dnešní době je několik metod sekvenování DNA. V poslední době se do popředí dostává hlavně pyrosekvenování a další metody, jež slibují zrychlení i zlevnění celého procesu. Níže je popis vybraných metod [16].

- Pyrosekvenování - Představuje jednu z novějších metod pro sekvenování DNA vycházející ze syntézy nových DNA sekvencí. Pro tuto metodu jsou nezbytné různé enzymy včetně DNA polymerázy, ATP sulfurylázy, luciferázy a apyrázy, spolu se substráty adenosinfosfosulfátem a luciferinem. V průběhu pyrosekvenování jsou postupně vkládány nukleotidy různých typů. Když se jeden nebo více nukleotidů daného typu začlení do vznikajícího řetězce, uvolní se světelné záření. Tento světelný efekt vzniká díky enzymatické reakci, kde se na začátku uvolní pyrofosfát z nově začleněného nukleotidu

a na konci dochází k oxidaci luciferinu a spotřebě ATP luciferázou. Tento záblesk je detektory zachycen a analyzován jako odpovídající nukleotid, který byl právě začleněn do řetězce [5].

- Technologie nanopórů - Využívají tenké otvory na molekulární úrovni, skrz které prochází zkoumaný řetězec DNA. V těchto nanopórech jsou umístěny mikroskopické elektrody. Díky specifickému vlivu každého ze čtyř nukleotidů na tok elektronů mezi těmito elektrodami lze rychle určit pořadí nukleotidů procházejících skrz nanopór [32].

Kapitola 3

DNA počítání

DNA počítání je oblastí, kde molekulární struktury DNA nejen uchovávají genetické informace, ale jsou i využívány k provádění výpočtů. Tato kapitola představí koncepty a techniky, které umožňují využití DNA jako výpočetní platformy. Ukáže základní DNA operace, na kterých DNA počítání staví, a představí možnosti jeho simulace.

3.1 Operace s DNA

Operace, které můžeme s DNA provádět plynou z jeho chemické struktury a základně se je dá rozdělit na operace katalyzované enzymy a operace, které se provádějí bez potřeby přítomnosti enzymů. Mezi operace, které se vyhnou potřebě enzymů, patří denaturace a renaturace. Všechny ostatní operace představené v této části vyžadují přítomnost specifických enzymů [16].

3.1.1 Denaturace a renaturace

V DNA se nachází dvě základní vazby a to vodíková (vazba mezi jednotlivými vlákny DNA) a fosfodiesterová (vazba mezi nukleotidy na jednom vlákně). Vodíková vazba je mnohem slabší než fosfodiesterová, což nám umožňuje rozdělit DNA vlákna od sebe bez toho, než bychom porušili samotná vlákna [16].

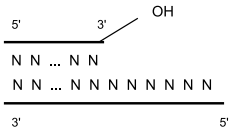
Jedním způsobem, jak provést toto rozpojení, je zahřívání dokud se nerozpojí, to znamená, dokud se nerozpadne na dvě vlákna. Tomuto jevu se říká denaturace. Teplota tání DNA je okolo 85°C až 95°C. Teplota nutná pro denaturaci DNA je závislá na počtu CG vazeb v DNA, čím více jich bude, tím vyšší teplotu potřebujeme. Pokud DNA řetězec bude obsahovat příliš CG vazeb, je možné, že nebude možné jej denaturovat. Toto je způsobeno typem vazby mezi C a G, která používá tři vodíkové vazby oproti AT, které používá pouze dvě, což způsobuje vyšší teplotní odolnost. Pokud přestaneme zahřívat a necháme DNA zchladnout, opět se spojí pomocí principu komplementarity, tomuto jevu říkáme renaturace. Renaturaci a denaturaci lze také navodit i pomocí některých chemických látek, jako je například sůl, tedy pokud náš roztok bude příliš slaný, nemusí při zchlazení dojít k renaturaci [16] [43].

3.1.2 Prodlužování

Třída enzymů jménem polymeráza je schopna přidat nukleotid do existující DNA molekuly. Důležitou vlastností je, že DNA polymeráza dokáže rozšiřovat DNA pouze ve směru 5‘→3‘. Tento enzym pro provedení přidání potřebuje:

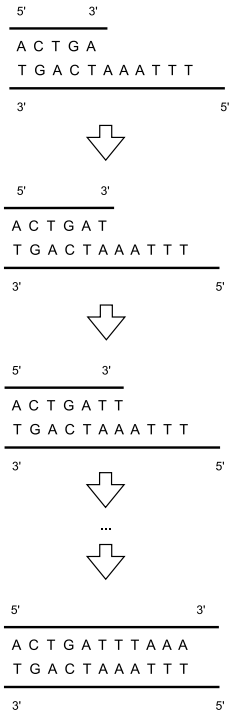
- template - pomocí komplementarity určuje nukleotidy, které budou přidány,
- primer - již existující sekvence, která je připojena k části template na jejím 3‘ konci.

Na obrázku 3.1 je ukázáno DNA, které je připraveno na prodloužení. Dolní vlákno je template, horní je primer. N na obrázku představuje libovolný nukleotid, který na daném místě může být při dodržení komplementarity [16] [43].



Obrázek 3.1: DNA připraveno na prodloužení

Polymeráza nyní bude opakovaně rozšiřovat primer vlákno na 3‘ konci pomocí nukleotidů, které musí být dostupné v médiu, ve kterém se reakce provádí. Zároveň tyto nukleotidy musí být komplementy k nukleotidům v template řetězci [16] [43]. Tento proces je ukázán na obrázku 3.2.



Obrázek 3.2: Průběh prodlužování DNA

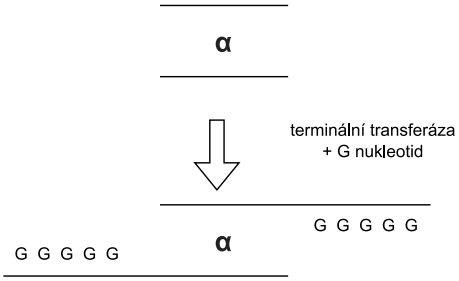
Pokud chceme vytvořit druhé vlákno k jednovláknu, které zatím nemá primer, můžeme uměle primer dodat a pomocí polymerázy dosyntetyzovat zbytek řetězce. Těmto uměle

dodaným primerům se říká Oligonukleotidy a jsou často používány v genetickém inženýrství jako primery. Směr syntézy bude 5‘ → 3‘, jak určila příroda, a i ve kterém in vivo DNA syntéza probíhá [16] [43].

Primer je možné vytvořit chemicky jako krátké jednovláknové DNA, které se pomocí komplementarity spojí s naším template řetězcem. Chemická syntéza DNA vlákna, která přidává nukleotid po nukleotidu, probíhá ve směru 3‘ → 5‘. 3‘ konec molekuly je pevně uchycen k pomocnému úchytu. Takže je pro nový nukleotid dostupný pouze 5‘ konec, na který se může připojit nový nukleotid s 3‘ koncem. Dobře načasované puštění 3‘ konce vlákna a 5‘ konce nukleotidu zajistí přidání nukleotidu do vlákna pomocí fosfodiesterové vazby. Tento postup je dobře automatizovatelný, proto již existuje spousta syntetizačních zařízení [16] [43].

Transferáza

Všechny typy polymeráz potřebují pro prodlužování 3’ konec, tedy potřebují primer řetezec. Nicméně exitují speciální polymerázy, které nevyžadují template řetězec, takovou polymerázou je terminální deoxynukleotidyl transferáza [16]. Ukázka její funkce je na obrázku 3.3.



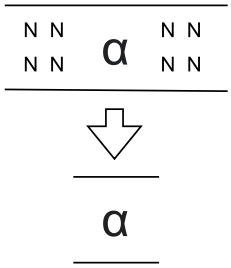
Obrázek 3.3: Úprava terminální deoxynukleotidyl transferázou

3.1.3 Skracování

DNA nukleáza je enzym, který zkracuje DNA. DNA nukleázy se dělí na endonukleázy a exonukleázy. Endonukleázy štěpí fosfodiesterové vazby uvnitř řetězce nukleové kyseliny. Naproti tomu exonukleázy štěpí nukleovou kyselinu od jejich konců a to tak, že štěpením fosfodiesterové vazby postupně odštěpují jednotlivé nukleotidy. Zjednodušeně řečeno endonukleázy štěpí uvnitř DNA vlákna a exonukleázy štepí konce DNA vlákna [43].

Exonukleázy jsou na rozdíl od polymeráz více flexibilní, proto najdeme exonukleázy takové, které odstraňují nukleotidy z 3’ konce a jiné, které odstraňují z 5’ konce. Také najdeme exonukleázy, které se specializují na dvojvláknové DNA, jiné na jednovláknové a některé zvládnou obě varinaty [16].

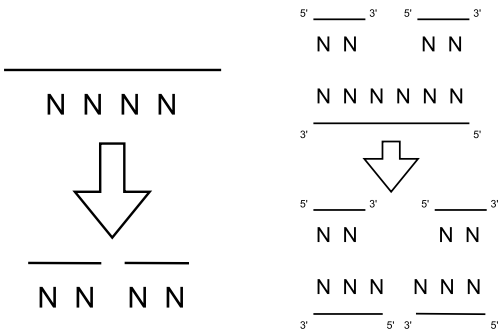
Pokud chceme zkrátit DNA vlákno, exonukleázy pro to budou vhodnou volbou. Vzhledem k diverzitě exonukleáz je možné vybrat takovou, které nám vlákno zkrátí námi vyžadovaným způsobem [16] [43].



Obrázek 3.4: Aktivita exonukleázy Bal31

3.1.4 Stříhání

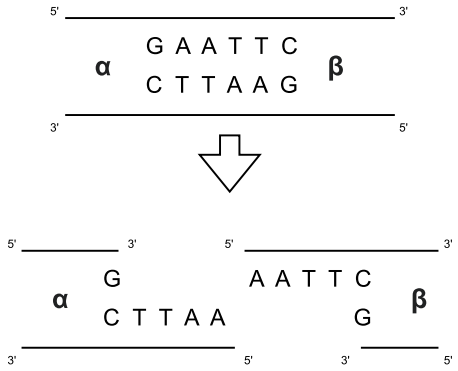
Endonukleázy štěpí fosfodiesterové vazby uvnitř řetězce nukleové kyseliny. Lze říci, že endonukleázy rozstřihnou vlákno na dvě či více částí. Stejně jako exonukleázy jsou i endonukleázy velmi specializovanými enzymy. Jednotlivé endonukleázy se liší v tom jak, kde a co budou stříhat. Například endonukleáza S1 je schopna stříhat pouze celojednovláknovou případně jednovláknovou DNA, která je místy dvojvláknová. Samotné místo střihu u S1 není specifické a může k němu dojít kdekoli. Nicméně tam, kde dojde ke střihu, musí být DNA ve formě jednořetězce [16] [43].



(a) Jednovlákno (b) Dvojvlákno

Obrázek 3.5: Aktivita endonukleázy S1

Opakem jsou enzymy, které dokáží rozpoznat specifickou sekvenci. Tyto enzymy najdou tuto specifickou sekvenci, pro kterou jsou určeny, a provedou střih DNA v daném místě. Příkladem takového enzymu je enzym EcoRI, který rozpozná sekvenci 3'-GAATTC a rozdělí dvojvláknovou DNA tak, že každé vlákno bude mít AATT [16].



Obrázek 3.6: Aktivita endonukleázy EcoRI

3.1.5 Spojování a vkládání

DNA může být spojena (zřetězena) za pomoci procesu zvaného ligace, který je pojmenován po enzymu DNA ligáza. Tato operace je prováděna různými způsoby [43].

Enzym EcoRI, který byl zmíněn v části 3.1.4, vytvoří dvě DNA dvojvlákna, která mají přečnívající lepivé (komplementární k sobě) konce, to lze vidět na obrázku 3.6. Pokud se tato vlákna příliš přiblíží k sobě, mohou se opět spojit pomocí vodíkových vazeb, nicméně toto spojení nebude tak pevné jako před sestřihem. Důvodem je, že vlákna nejsou spojena pomocí fosfodiesterové vazby. V místech spojení můžeme pozorovat trhliny v DNA. Těmto trhlinám v řetězci se říká nick. Tyto trhliny je enzym ligáza schopen zacelit tak, že spojí 5' konce s 3' konci. Při použití endonukleáz je možné tímto způsobem DNA i přeskupit [16] [43].

Práce DNA ligázy je v této situaci celkem lehká, protože 5' a 3' konce drží blízko sebe díky vodíkovým vazbám. Horší situace nastane, pokud oba řetězce nebudou mít žádné přesahující konce. Poté máme takzvané tupé konce. DNA Ligáza je schopna spojit vlákna i v tomto případě, ale nebude snadné tuto operaci provést vzhledem k tomu, že ligáza potřebuje mít 5' konec blízko 3' konce a v současné situaci se zde nenacházejí vodíkové vazby, které by vlákna přidržovaly. Z tohoto důvodu nebude tato operace příliš efektivní v porovnání s předchozí situací [16] [43].

Další možností spojení tupých konců je použití enzymu transferáza, který byl zmíněn v části 3.1.2. Pomocí tohoto enzymu k tupým koncům přidáme přečnívající lepivé konce a následně budeme pokračovat jako u prvního příkladu [16].

Spojováním řetězců lze provádět i vkládání nových sekvencí do DNA. Pomocí endonukleázy rozštěpíme DNA v místě, kam chceme vložit novou část, následně mezi rozštěpená vlákna vložíme vlákno, které chceme vložit, a pomocí ligáz všechna vlákna zpět spojíme [16] [43].

3.1.6 Úprava DNA

Enzymy, které upravují chemické složení jednotlivých nukleotidů, jsou velmi užitečné při ovládání většího množství operací nad DNA. Methyltransferázy jsou enzymy, které se používají in vivo jako partner k restrikční endonukleáze. Hlavním úkolem restrikční endonukleázy je chránit hostitele (bakterie) před útočníky (viry). Restrikční endonukleáza, pokud najde DNA útočníka, provede jeho degradaci (rozstříhá jej). Nicméně hostitelská buňka může mít ve své vlastní DNA sekvence podobné útočníkově a restrikční endonukleáza by jej

mohla rozstříhat. Proto je nutné upravit tuto část tak, aby nebyla restrikční endonukleáze dostupná. Tuto úpravu provádí methyltransferázy [16] [43].

Methyltransferáza se naváže na jeden nukleotid v restrikční části a upraví jej. Tím se stává restrikční část pro restrikční endonukleázu nedostupnou [16] [43].

CRISPR

CRISPR je technika genetického inženýrství, kterou lze modifikovat DNA. Je založena na zjednodušené verzi bakteriální CRISPR - Cas9, kterou si bakterie také vyvinuly na ochranu proti virům. Dodáním Cas9 spojené se syntetickou vodicí RNA (gRNA) je možné DNA štěpit na požadovaném místě, což umožňuje odstranit stávající nukleotidy nebo přidat nové nukleotidy in vivo [43] [25].

3.1.7 Amplifikace

Polymerázová řetězová reakce (PCR) je metodou sloužící pro získání velkého množství kopií daných DNA vláken z menšího počtu zdrojových vláken, tedy pro amplifikaci vláken DNA. Tato metoda je hojně používána v DNA počítání ale i v jiných oborech jako je například zdravotnictví [43].

PCR je založeno na opakované řízené denaturaci dvojřetězcové DNA a následné renaturaci osamocených řetězců se specifickými oligonukleotidy. Tyto oligonukleotidy slouží jako primery pro syntézu nových DNA vláken. Amplifikace DNA probíhá v opakujících se cyklech [36].

Metoda [36] [1]

1. Denaturace - zahřátím roztoku s DNA dojde k rozpadu vodíkových spojů mezi řetězci, čímž se dsDNA přemění na dvě ssDNA.

2. Hybridizace - roztok se lehce ochladí. Molekuly ssDNA se začnou renaturovat. Pokud bude v roztoku nadbytek specifických oligonukleotidů dojde k jejich spojení s komplementární sekvencí rychleji než ke spojení dlouhých ssDNA vláken, jejichž koncentrace je mnohem nižší. Teplota pro tento krok je kritická a závisí na použitém páru primerů.

    - Pokud bude teplota příliš nízká, budou primery nasedat na sekvence, se kterými jsou komplementární jen částečně.
    - Pokud bude teplota příliš vysoká, budou primery nasedat jen na malou část sekvencí, se kterými jsou komplementární, a výsledného produktu bude málo.

3. Elongace - roztok lehce ohřejeme natolik, aby si již primery nenasedaly na ssDNA řetězce, ale ne natolik, aby začalo docházet k denaturaci. Oligonukleotidy, které dosedly na ssDNA (templát) v předchozím kroku, slouží v tomto kroku jako primery pro DNA polymerázu. Od 3' konce primeru začíná syntéza nového řetězce komplementárního k templátu, jak je popsáno ve sekci 3.1.2.

4. Opakujeme krok 1 až 3 dokud nemáme dostatek řetězců.

Tato metoda při každém cyklu zdvojnásobí počet DNA molekul. Nárůst počtu DNA molekul je tedy k počtu cyklů v exponenciální závislosti. Pokud máme před zahájením PCR N molekul, tak po m cyklech PCR budeme mít $N \cdot 2^m$ molekul DNA [1].

3.2 Počátky DNA počítání - Adlemanův experiment

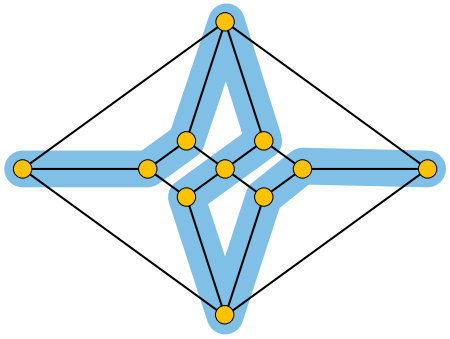
Adlemanův experiment [3], který je pojmenovaný po vědci Leonardu Adlemanovi, odhalil možnosti využití biologických molekul konkrétně DNA jako prostředku pro provádění výpočtů. Leonard Adleman tímto experimentem demonstroval potenciální možnosti DNA v oblasti počítačových výpočtů tím, že poprvé úspěšně řešil problém hamiltonovské cesty pomocí manipulace s molekulami DNA. Tento experiment nejenže otevřel nové perspektivy pro výzkum v oblasti bioinformatiky a nanotechnologií, ale také poskytl nový pohled na to, jak můžeme využít přírodních jevů k řešení složitých matematických problémů. Tato kapitola se zabývá detaily tohoto experimentu a jeho důsledky pro další vývoj v oblasti počítačových věd [16] [19].

3.2.1 Hamiltonovská cesta

Hamiltonovská cesta je v teorii grafů cestou, která prochází každým vrcholem v grafu právě jednou. Tedy, je to posloupnost vrcholů v grafu, kde každý vrchol je navštíven právě jednou a přitom sousedí s předchozím a následujícím vrcholem v cestě [31].

Hamiltonovské cesty jsou základním konceptem v matematické teorii grafů a mají důležité aplikace v různých oblastech, jako jsou logistika, plánování tras, biologie a informatika. Najít hamiltonovskou cestu v grafu může být v některých případech složitý problém zvláště v případech velkých a složitých grafů [31].

Problém nalezení hamiltonovské cesty v grafu patří mezi takzvané NP-úplné problémy. To znamená, že není znám efektivní algoritmus, který by v polynomiálním čase našel řešení pro libovolnou instanci tohoto problému [31].



Obrázek 3.7: Ukázka hamiltonovské cesty (zvýrazněna modře) v grafu

3.2.2 Algoritmus nalezení hamiltonovské cesty v grafu

Algoritmus, který tento experiment používá, využívá nedeterminizmus k získání řešení v polynomiálním čase. Díky tomu je tento algoritmus principielně schopen vypočítat výsledek rychleji něž algoritmy, které se na řešení tohoto problému používají v konvenčních počítačích. Algoritmus tedy staví na nedeterministických rozhodnutích, které zásadně algoritmus urychlují [16] [19].

Nechť je $G = (V, E, \epsilon)$ orientovaným grafem[1], ve kterém hledáme hamiltonovskou cestu p, která začíná ve vrcholu v_{in}, končí ve vrcholu v_{out} a má právě n vrcholů. Poté je algoritmus pro zjištění p následující [19]:

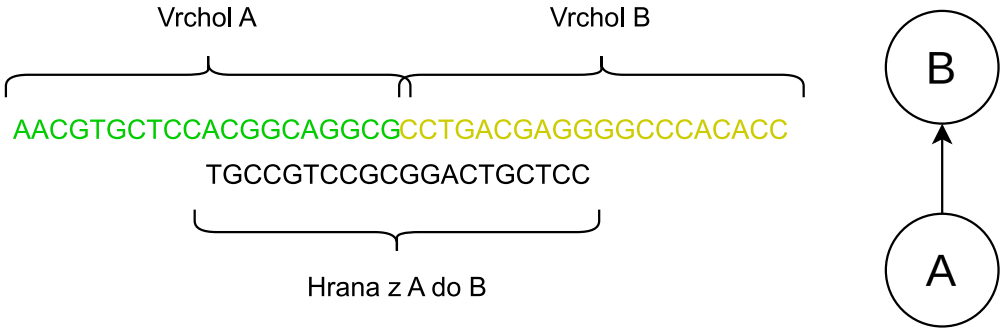
[1] Je možné aplikovat i na neorientovaný graf tak, že neorientovaný graf převedeme na orientovaný.

1. Generování dostatečného počtu náhodných cest uvnitř G.
2. Zamítnutí všech cest, které nezačínají v v_{in} a nekončí v v_{out}.
3. Zamítnutí všech cest, které nemají právě n vrcholů.
4. Zamítnutí všech cest, pro které platí $\exists v; v \in V \wedge v \notin p$ (existuje vrchol v uvnitř G, který není v p)

Jedná se o stochastický algoritmus, který byl použit z důvodu vhodnosti vykonávání příslušných kroků molekulami DNA.

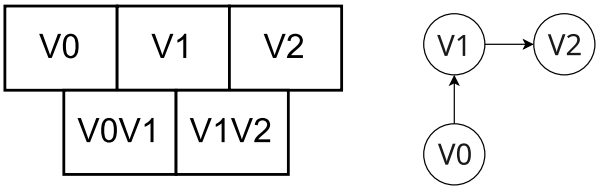
3.2.3 Kódování problému

Před provedením samotného výpočtu je nutné vymyslet vhodnou reprezentaci pro problém tak, aby bylo možné využít vlastností DNA pro provedení výpočtu. Adleman ve svém experimentu zakódoval každý vrchol pomocí jednovláknové DNA (ssDNA) o délce 20 nukleotidů. Tyto sekvence byly voleny náhodně tak, aby žádný vrchol nemohl mít stejnou 20nukleotidovou sekvenci jako jiný vrchol. Následně každá hrana orientovaného grafu byla zakódována pomocí ssDNA. Toto ssDNA bylo složeno z nukleotidů, které jsou komplementy k druhé polovině ssDNA vrcholu, ze kterého hrana vychází, a z komplementů k první polovině nukleotidů kódující vrchol, do kterého hrana vede. Na obrázku 3.8 je zobrazení zakódování dvou vrcholů A (**AACGTGCTCCACGGCAGGCG**) a B (**CCTGACGAGGGGCCCACACC**) a jedné hrany, která vede z vrcholu A do vrcholu B (**TGCCGTCCGCGGACTGCTCC**) [19].



Obrázek 3.8: Příklad zakódování hrany A→B pomocí DNA

Toto kódování využívá masivního paralelismu, který plyne z vhodného využití komplementarity bází. Z obrázku 3.8 je patrné, že pokud existuje hrana mezi vrcholy A a B dojde k spojení obou vrcholů do jednoho delšího vlákna DNA, které vznikne díky vazbě hrany na vrcholy pomocí komplementarity bází. Také je patrné, že okraje našeho nového vlákna jsou stále volné pro párování a dokonce i během párování hrany AB byly dostupné k párování, to umožňuje párování více hran ve stejný okamžik. Díky tomu je možné aby byly hledány celé náhodné cesty v grafu masivně a paralelně. Na obrázku 3.9 je ukázána možná náhodná cesta v grafu s vrcholy (V0, V1, V2) a hranami (V0 →V1, V1 →V0, V1 →V2). Digram na obrázku je zjednodušen nahrazením 20nukleotidové sekvence jmény vrcholů a hran [19].



Obrázek 3.9: Ukázka cesty v Adlemanově experimentu

3.2.4 Aplikace algoritmu nalezení hamiltonovské cesty v grafu

První krok algoritmu provádí náhodné generování cest v grafu G. To je zajištěno pomocí amplifikace zakódovaných vrcholů a hran pomocí PCR. Když jejich počet bude dostatečně velký, smísíme jednotlivá vlákna reprezentující hrany s vlákny, které reprezentují vrcholy. Po smísení dojde k párování komplementárních bází mezi vrcholy a hranymi, jak bylo popsáno ve sekci 3.2.3. Protože každá molekula ssDNA bude po smísení na jiném místě v tekutině, bude mít ve svém blízkém okolí rozdílné vrcholy a hrany, což povede k vytvoření náhodných cest v grafu. Pokud bude hran a vrcholů dostatečně mnoho, je pravděpodobné, že pokud graf obsahuje hamiltonovskou cestu, bude tato cesta mezi náhodnými cestami vytvořenými v tomto kroku [19] [3].

Druhý krok provádí filtraci sekvencí pomocí vstupního a výstupního vrcholu. Provedení tohoto úkonu je založeno na využití volných konců vláken, které jsou připraveny na párování hrany. Vytvoříme sekvenci, která bude komplementární k tomuto volnému konci a umístíme na ni magnet. Tuto sekvenci použijeme (jak bylo popsáno v části 2.4.2) pro nalezení vláken, které začínají nebo končí touto specifickou sekvencí. Tyto naše sondy se pomocí komplementarity spojí s vlákny, které odpovídají našim požadavkům, následně můžeme použít magnet pro separaci těchto vláken a odstranit zbytek vláken vypuštěním roztoku. Alternativním způsobem je amplifikace vláken pomocí PCR, u kterého použijeme specifické primery, které umožní amplifikaci jen těch řetězců, které odpovídají požadavkům [19] [3].

Třetí krok provádí filtraci náhodných cest pomocí délky. Je zřejmé, že hamiltonovská cesta musí mít právě tolik vrcholů, kolik je vrcholů v grafu. Z tohoto důvodu se provádí odfiltrování náhodných cest, které takovou délku nemají. Víme, že jednotlivé vrcholy mají délku dvaceti nukleotidů, pokud dojde ke spojení dvou vrcholů hranou, bude délka molekuly dvojnásobná, protože se skládá ze dvou vrcholů. Hrany délku neprodlužují, protože se vážou na místa ve vrcholech a tím na tomto místě tvoří dsDNA. Tedy pokud máme graf, který má šest vrcholů, budeme hledat molekuly o délce sto-dvaceti nukleotidů ($6 * 20$). Pro rozeznání délky vláken použijeme gelovou elektroforézu, která byla zmíněna v části 2.4.1. Pomocí této metody separujeme sekvence, které odpovídají našim kritériím [19] [3].

Čtvrtý krok provádí filtraci podle obsahu jednotlivých vrcholů v cestě. Provedení tohoto úkonu je závislé na existenci jednovláknové DNA se sekvencí, která je specifická pro každý vrchol. Vytvoříme vlákno, které bude komplementární k této sekvenci a umístíme jej na magnet. Toto vlákno použijeme (jak bylo popsáno v části 2.4.2) pro nalezení vláken, které obsahují touto specifickou sekvenci. Tyto naše sondy se pomocí komplementarity spojí s vlákny, které odpovídají našim požadavkům, následně můžeme použít magnet pro separaci těchto vláken a odstranit zbytek vláken vypuštěním roztoku. Tento postup provádíme postupně pro každý vrchol z grafu [19] [3].

Posledním krokem je přečtení samotné cesty, pokud existuje. Zda existuje zjistíme tak, že nám nezbyde žádné vlákno a nebudeme tedy schopni provést kroky pro přečtení cesty. Samotné přečtení cesty jsme schopni provést pomocí sekvenování DNA, které bylo pro-

psáno v části 2.4.3. Adleman v původím experimentu použil jinou metodu, při které použil PCR a gelovou elektoforézu. Jako primery jsou zde použity vlákna odpovídající prvnímu vrcholu cesty a vrcholu uvnitř cesty (libovolný vrchol grafu). DNA je mezi těmito dvěma primery amplifikováno a následně změřeno pomocí gelové elektroforézy. Takto se postupuje postupně pro všechny vrcholy grafu. Pomocí rozdílů délek jsme následně schopni zjistit pořadí jednotlivých vrcholů uvnitř cesty. Tento postup byl úspěšně aplikován v Adlemanově prvním experimentu. Zároveň se jedná o časově nejnáročnější krok celého algoritmu, kdy v Adlemanově prvním experimentu zabral tento postup sedm dní laboratorní práce [19] [3].

3.2.5 Formální zápis algoritmu

Pro sepsání formálního algoritmu popisujícího Adlemanův experiment je nutné nejdříve nadefinovat operace, které bude používat [3].

- **prefix-extract**(T, x): Nechť T je zkumavka a x je řetězec, potom vytvoříme novou zkumavku obsahující všechny řetězce z T začínající řetězcem x.
- **postfix-extract**(T, x): Nechť T je zkumavka a x je řetězec, potom vytvoříme novou zkumavku obsahující všechny řetězce z T končící řetězcem x.
- **substring-extract**(T, x): Nechť T je zkumavka a x je řetězec, potom vytvoříme novou zkumavku obsahující všechny řetězce z T obsahující podřetězec x.
- **length-separate**(T, m): Nechť T je zkumavka a m je celé nezáporné číslo, potom vytvoříme novou zkumavku obsahující všechny řetězce z T jejichž délka je rovna m. [2]
- **detect**(T): Nechť T je zkumavka, potom vrať *ano*, pokud T obsahuje alespoň jednu DNA molekulu, jinak vrať *ne*.

Nechť $G = (V, E, \epsilon)$ je orientovaným grafem s vrcholy $V = \{v_1, v_2, ..., v_n\}$, poté máme zdrojovou zkumavku T, která obsahuje množinu řetězců abecedy V, které reprezentují cesty v grafu G. Každý vrchol z V je v DNA zakódován pomocí m nukleotidů. Potom je algoritmus nalezení hamiltonovské cesty následující [19] [3].

Algorithm 1 Algoritmus pro hledání hamiltonovské cesty pomocí DNA [19]

Require: vstupní zkumavka T, graf G
  T ← prefix-extract(T, v_1)
  T ← postfix-extract(T, v_n)
  T ← length-separate$(T, m \cdot n)$
  $i \leftarrow 1$
  while $i \leq n$ **do**
    T ← substring-extract(T, v_i)
    $i \leftarrow i + 1$
  end while
  return detect(T)

[2] Je možné použít i variantu, kdy délka řetězce je $\leq m$, výsledek algoritmu změněn nebude.

3.3 Řešení problému splnitelnosti (SAT)

Výsledky Adlemanova experimentu vedly k zahájení výzkumu tohoto nově objeveného oboru. Jedním z prvních výzkumníků, který objevil potenciál DNA byl R. Lipton (1995) [24], který přinesl do oboru řešení problému splnitelnosti a jiných NP-úplných problémů. Nechť F je booleovský výraz s n proměnnými, který je v konjunktivní normální formě (CNF) [19].

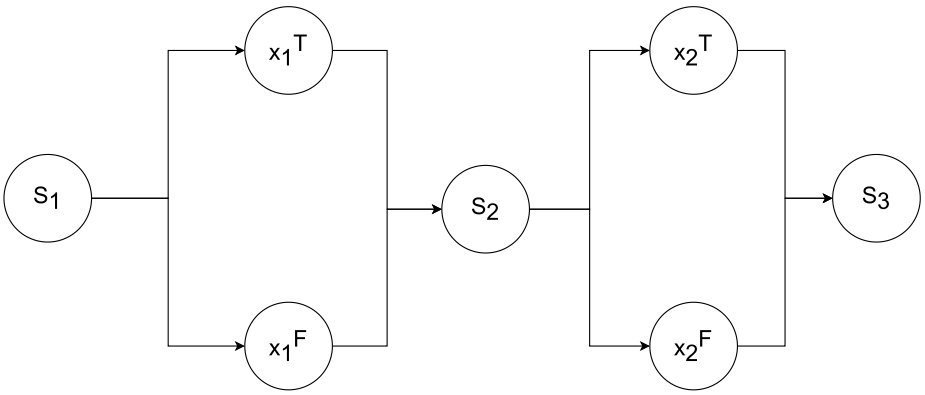
$$F = (\neg x_1 \vee x_2) \wedge (v_1 \vee \neg x_2)$$

Lipton se rozhodl reprezentovat každou proměnou x_i pomocí dvou ssDNA molekul, jedna reprezentuje její pravdivou hodnotu (x_i^T) a druhá její nepravdivou hodnotu (x_i^F). Pomocí tohoto kódování Lipton převedl SAT problém na grafový problém [19].

Pro zajištění fungování algoritmu Lipton navrhl následující operace [24] [19].

- **extract**(T, i, b): Nechť T je zkumavka, i je celé nezáporné číslo a $b \in \{0, 1\}$. Potom vytvoříme novou zkumavku obsahující všechny řetězce z T obsahující b na pozici i $x \in T; x_i = b$.
- **rem**(T, i, b): Nechť T je zkumavka, i je celé nezáporné číslo a $b \in \{0, 1\}$. Potom vytvoříme novou zkumavku obsahující všechny řetězce z T neobsahující b na pozici i $x \in T; x_i \neq b$.
- **merge**$(T_1, ..., T_n)$: Nechť T_1 až T_n je zkumavka. Potom vytvoříme novou zkumavku obsahující sjednocení všech zkumavek T_1 až T_n. $T_1 \cup ... \cup T_n$

Liptonův algoritmus pak dostane vstupní zkumavku T, která obsahuje mnoho řetězců $a_1, ..., a_n$, kde a_i reprezentuje pravdivostní hodnotu proměnné na pozici i, která může být pravda (x_i^T) nebo nepravda (x_i^F). Vstupní zkumavka má tedy téměř stejný formát jako v Adlemanově experimentu, který řeší grafový problém hamiltonovské cesty. V případě Liptonova řešení je použit speciální graf G_n, který je použit pro řešení SAT problému s n proměnnými (na obrázku 3.10) [24] [19].



Obrázek 3.10: Graf pro SAT problém výrazu F

Algoritmus 2, vyjadřující pseudokód Liptonova algortimu, iterativně vybírá ssDNA, které splňují disjunktivní klauzule. Například v případě $\neg x_1 \vee x_2$, bude vybráno pouze x_1^F a x_2^T. Po iterativním projití všech disjunktivních částí zůstanou ve zkumavce jen ty ssDNA, které splňují všechny disjunktivní klauzule a tedy i ty, pro které je výraz pravdivý [24] [19].

Algorithm 2 Liptonův algoritmus pro řešení problému SAT [19]

Require: vstupní zkumavka T, CNF F v proměnných $x_1, ..., x_n$, prázdná zkumavka T_c
  for `disjunktivní část c in F` **do**
    for `literál a in c` **do**
      $i \leftarrow id\_of\_varaible(a)$
      if $a = x_i$ **then** ▷ Proměnná bez negace tedy jen x_i
        $T' \leftarrow extract(T, i, 1)$
        $T \leftarrow rem(T, i, 1)$
        $merge(T', T_c)$
      else if $a \neq x_i$ **then** ▷ Proměnná s negací tedy $\neg x_i$
        $T' \leftarrow extract(T, i, 0)$
        $T \leftarrow rem(T, i, 0)$
        $merge(T', T_c)$
      end if
    end for
    $T \leftarrow T_c$ ▷ T_c obsahuje řešení disjunktivní klauzule
  end for
  return detect(T)

3.4 Řešení 3-SAT pomocí Split and Merge

3-SAT problém je specifickou variantou SAT problému, kde je dáno omezení na maximální počet literálů v jedné disjunktivní klauzuli. V případě 3-SAT problému je toto omezení stanoveno na maximálně tři literály v jedné disjunktivní klauzuli. Nechť F je booleovský výraz s n proměnnými, který je v konjunktivní normální formě (CNF) [10] [19].

$$F = (\neg x_1 \vee x_2) \wedge (v_1 \vee \neg x_2) \wedge (v_3 \vee \neg x_4 \vee x_5)$$

Algoritmus pro řešení 3-SAT problému pomocí DNA byl vymyšlen L. Adlemanem a jeho spolupracovníkem v roce 2002, kdy pomocí něj úspěšně vyřešily 3-SAT problém pro dvacet proměnných. Algoritmus je velmi podobný Liptonovu řešení, ale používá jinou metodu filtrování. Pro fungování algoritmu je potřebná následující operace [10] [19].

- **remove(T, $\{x_1, ..., x_m\}$)**: Nechť T je zkumavka a $\{x_1, ..., x_m\}$ je množinou řetězců. Potom vytvoříme novou zkumavku obsahující jen ty řetězce z T, které nejsou libovolným řetězcem z $\{x_1, ..., x_m\}$.

Algoritmus dostane vstupní zkumavku T, která obsahuje mnoho řetězců $a_1, ..., a_n$, kde a_i reprezentuje pravdivostní hodnotu proměnné na pozici i, která může být pravda (x_i^T) nebo nepravda (x_i^F). Vstupní zkumavka má tedy téměř stejný formát jako v případe Liptonova řešení [10] [19].

Algoritmus 3, vyjadřující pseudokód Split and Merge řešení 3-SAT problému iterativně odstraňuje ssDNA, která nesplňují disjunktivní klauzule. Například v případě $(v_3 \vee \neg x_4 \vee x_5)$, bude odstraněno x_3^F, x_5^F a x_4^T. Po iterativním projití všech disjunktivních částí zůstanou ve zkumavce jen ta ssDNA, která splňují všechny disjunktivní klauzule a tedy ta, pro která je výraz pravdivý. Odstranění je provedeno operací **remove(T, $\{x_1, ..., x_m\}$)** a lze realizovat pomocí navázání ssDNA, která jsou komplementární k odstraňovaným ssDNA, a následně

je odstraní z roztoku vytažením těchto ssDNA nebo pomocí délkové separace provedené pomocí gelové elektroforézy [10] [19].

Algorithm 3 Řešení problému 3-SAT pomocí Split and Merge [19]

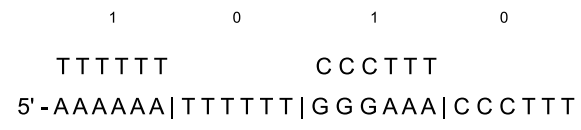
Require: vstupní zkumavka T, CNF F v proměnných $x_1, ..., x_n$
  for `disjunktivní část c in F` **do**
    for `literál a in c` **do**
      $i \leftarrow id\_of\_varaible(a)$
      if $a = x_i$ **then** ▷ Proměnná bez negace tedy jen x_i
        $T \leftarrow remove(T, \{x_i^F\})$
      else if $a \neq x_i$ **then** ▷ Proměnná s negací tedy $\neg x_i$
        $T \leftarrow remove(T, \{x_i^T\})$
      end if
    end for
  end for
  return detect(T)

3.5 Sticker systém

Sticker systém [41] byl posán L. Adlemanem a jeho spolupracovníky v roce 1996 a v současnosti je jedním z nejpopulárnějších DNA modelů. Jedná se o model, na který lze nahlížet jako na realizaci registrového automatu [19].

Sticker systém je model, který umožňuje náhodný přístup do paměti (RAM), který nevyžaduje rozšiřování řetězce. Sticker systém je model skládající se z takzvaných paměťových komplexů. Paměťový komplex je místy jednovláknové a místy dvojvláknové DNA. Na tento komplex lze nahlížet jako na reprezentaci n-bitového čísla. Každý paměťový komplex se skládá ze dvou základních ssDNA částí, kterými jsou memory stread a sticker. Memory stread je ssDNA, které se skládá z n nepřekrývajících se podčástí, na které lze nahlížet jako na reprezentaci jednoho bitu. Sticker je ssDNA, které je komplementární právě k jedné podčásti memory stread [41] [19].

Sticker se může navázat na memory stread pomocí komplemetarity a tím „zapnout“ daný bit v paměti. Pokud se sticker na daném bitu na memory stread nenachází, lze na tuto situaci pohlížet tak, že daný bit je „vypnutý“. Na obrázku 3.11 je zobrazen paměťový komplex, ve kterém je uloženo čtyřbitové číslo 1010 [41] [19].



Obrázek 3.11: paměťový komplex s čtyřbitovým číslem 1010

Sticker systém ve své základní verzi umožňuje jen základní operace, mezi které patří především práce s jednotlivými bity. Nad sticker modelem lze tedy provádět pět základních operací, které toto zajišťují [41] [19].

- **merge**$(T_1, ..., T_n)$: Nechť T_1 až T_n je zkumavka. Potom vytvoříme novou zkumavku obsahující sjednocení všech zkumavek T_1 až T_n. $T_1 \cup ... \cup T_n$

- **separate**(T, T^+, T^-, i): Nechť T je zkumavka a i je celé nezáporné číslo. Potom vytvoříme dvě nové zkumavky T^+ a T^-. T^+ bude obsahovat všechny paměťové komplexy z T, ve kterých je bit na pozici i zapnutý, zkumavka T^- bude obsahovat všechny ostatní.

- **set**(T, i): Nechť T je zkumavka a i je celé nezáporné číslo. Potom vytvoříme novou zkumavku obsahující všechny paměťové komplexy z T, kde v každém z nich bude zapnut bit na pozici i.

- **clear**(T, i): Nechť T je zkumavka a i je celé nezáporné číslo. Potom vytvoříme novou zkumavku obsahující všechny paměťové komplexy z T, kde v každém z nich bude vypnut bit na pozici i.

- **discard**(T): Nechť T je zkumavka. Potom vymažeme veškerý obsah zkumavky T.

Operace merge$(T_1, ..., T_n)$ je provedena jednoduchým smícháním obsahu všech zkumavek $T_1, ..., T_n$. Nicméně je nutné toto smíchání provést opatrně, protože pokud bychom provedly smíchání příliš hrubě, mohlo by dojít k roztrhání DNA řetězců na menší fragmenty. Dalším problémem je, že některé řetězce mohou zůstat na stěnách zkumavky [41] [19].

Operace separate(T, T^+, T^-, i) zajišťuje rozdělení DNA do dvou zkumavek podle hodnoty bitu na pozici i. To lze zajistit pomocí krátkých oligonukleotidů komplementárních právě k podčásti na pozici i, které jsou připojeny ke stěně zkumavky. Paměťové komplexy se pomocí komplementarity naváží na tyto oligonukleotidy a v případě vypuštění obsahu zkumavky zůstanou nalepeny na její stěně. Oligonukleotidy musí mít s touto podčástí slabší vazbu (méně komplementárních bází) než samotný sticker, aby bylo možné po operaci paměťový komplex obnovit do původní formy [41] [19].

Operace set(T, i), která mění bit na pozici i na zapnutý, je prováděna přidáním velkého množství sticker. Pokud podčást na pozici i je již zapnuta, nic se nestane, protože sticker se nebude mít kam vázat. Pokud bude daná podčást vypnuta, sticker se naváže právě na tuto podčást a zapne ji tím [41] [19].

Operace clear(T, i) je nejproblematičtější ze všech operací. Denaturace bohužel v tomto případě nelze použít, protože by v takovém případě došlo k oddělení všech sticker a tím by systém vše zapomněl. Naštěstí tato operace může být vynechána bez ztráty úplnosti tohoto systému. Například místo vypnutí bitu odstraněním sticker, můžeme natavit jiný bit tak, že bude symbolizovat, že současný bit je vypnut [19].

3.6 SIMD||DNA

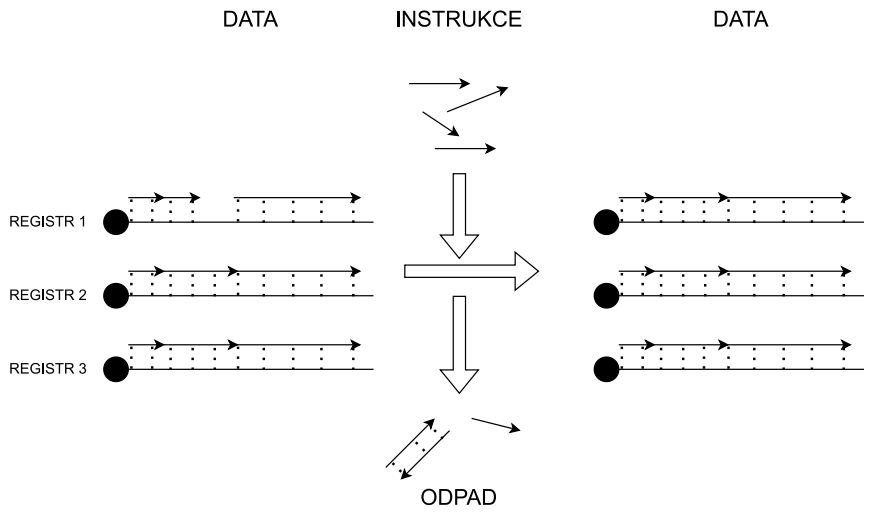
Všechny myšlenky v následující sekci pocházejí z článku SIMD||DNA: Single Instruction, Multiple Data Computation with DNA Strand Displacement Cascades, který model SIMD||DNA předtavuje [45].

Poznámka: Vzhledem k narůstající komplexitě modelů je zaveden pojem **doména**, který reprezentuje sekvenci nukleotidů, která není dostatečně dlouhá na to, aby vodíkové vazby udržely vlákno obsahující jednu tuto sekvenci pomocí komplementarity na vláknu jiném. Nicméně pokud vytvoříme vlákno obsahující dvě domény za sebou, bude síla vodíkových spojení již dostatečná. Komplementární doména k doméně je pak značena pomocí $*$. Pokud máme doménu A, pak její komplementární doménou je doména A^*. Ortogonální doména k doméně A je poté doména která není komplementární k doméně A.

SIMD||DNA je modelem, který se snaží zpracovávat data uložená v DNA. DNA je jako úložiště dat velmi vhodné právě díky hustotě záznamů, která podle studií umožní uložit až 455 exabajtů do jednoho gramu hmoty [15]. Existují již i experimenty, které úspěšně uložily text, obrázky a videa o velikostech v řádech stovek megabajtů do DNA [34]. Modely, které se používají na ukládání dat do DNA, jsou často velice podobné sticker systému uvedenému v části 3.5. Vzhledem k problému s operací clear taková datová úložiště nepočítají s prováděním operací nad daty, protože by data po provedení několika operací mohly velmi rychle zvětšit svůj objem. SIMD||DNA se snaží tomuto problému předejít pomocí systému strand displacement. Tento systém bohužel není kompatibilní s tradičními DNA úložišti, nicméně nám umožní provést operaci clear, která v základním sticker systému je velmi problematická. Aby operace clear byla možná, je nutné lehce upravit reprezentaci dat a zavést nové metody úprav podčástí paměťového komplexu, kterým se v tomto modelu říká buňky registru.

Registr je v SIMD||DNA ssDNA, které je upevněno k magnetu, aby bylo možné nad ním provádět všechny operace. Těchto registrů se ve zkumavce může vyskytovat více a všechny budou upevněny magnetem. Jednotlivé operace jsou v tomto modelu prováděny pomocí jednoho či více ssDNA, které jsou zde nazývány instrukčními vlákny. Při aplikaci instrukce jsou tato instrukční vlákna přidána do roztoku s registry a tím dojde k aplikaci instrukce. Jednotlivé instrukce se aplikují nad všemi registry paralelně, z tohoto důvodu je v názvu modelu SIMD tedy single instruction multiple data (jedna instrukce nad více daty).

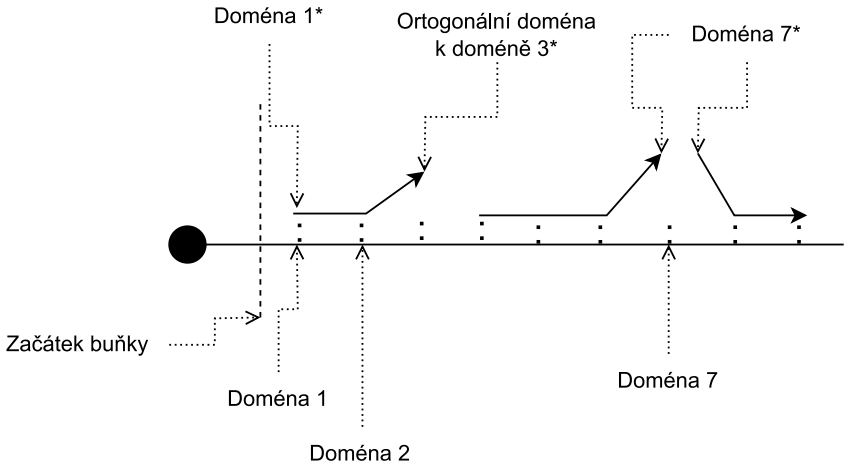
Provedení jedné instrukce lze rozdělit do tří částí. V první části vložíme instrukční vlákna do roztoku s registry. V druhé části instrukční vlákna reagují s registry a upravují data v nich. Tato reakce vytvoří ssDNA a dsDNA, která jsou odpadem této operace. Z tohoto důvodu je v třetí části postupně vypuštěn roztok ze zkumavky a tím jsou všechny odpadní řetězce odstraněny. Registry ve zkumavce zůstanou díky magnetům na jejich koncích. Po dokončení je možné zkumavku připravit na další instrukci.



Obrázek 3.12: Schéma SIMD||DNA systému

3.6.1 Kódování dat

Data jsou uložena ve vícevláknovém komplexu, který se nazývá registr, registr se skládá z jednoho dlouhého ssDNA, které se nazývá dolní řetězec a je obdobou memory stread ze Sticker systému. Druhou částí jsou krátká ssDNA, která se nazývají horní řetězce a jsou obdobou sticker. Dolní vlákno se skládá z několika částí zvaných buňky, tyto buňky jsou naskládány ve vláknu za sebou. Každá buňka se skládá z několika domén, přičemž všechny buňky jsou si ekvivalentní z pohledu jejich doménové reprezentace. Konfigurace horních řetězců nad buňkou určuje uložená data. Tato konfigurace lze chápat jako binární kódování podobně jako v případě sticker systému. Nicméně díky volnější definici horního řetězce je možné do jedné buňky uložit například i ternární číslo.



Obrázek 3.13: Notace řetězců uvnitř jedné buňky. Tečkované čáry reprezentují vodíkové můstky, rovné čáry reprezentují vlákna DNA a šipky reprezentují směr vláken DNA. Černý kruh reprezentuje magnet na konci registru. Krátká vzdálenost mezi vláknem a vodíkovým můstek značí existenci spojení mezi vlákny a dlouhá vzdálenost jeho absenci.

3.6.2 Instrukce

Instrukce jsou množinou ssDNA, která se ve velké koncentraci přidá do roztoku s registry. Přidání těchto vláken může vyvolat tři základní reakce. Pomocí těchto reakcí je možné vytvořit výpočetně úplné systémy [45]. Těmito reakcemi jsou:

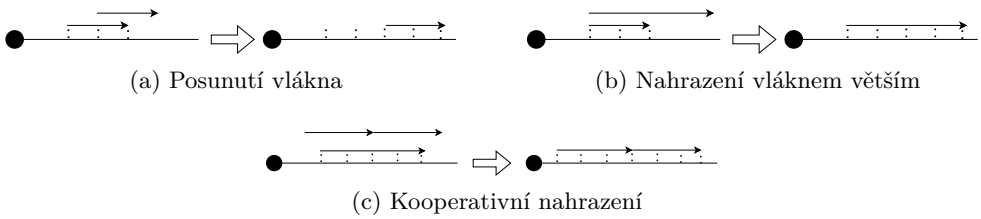
- **Přidání**: Instrukční vlákno může být pomocí komplementarity navázáno na spodní vlákno, pokud jsou k tomu potřebné síly dostatečně velké (existují-li alespoň dvě volné domény na spodním vláknu, které jsou komplementární k doménám na instrukčním vláknu). Důležitou poznámkou je, že navázání instrukčního vlákna může vyvolat částečné oddělení vlákna, které již na spodním vlákně navázáno bylo.

(a) Přidání vlákna na volné domény

(b) Přidání vlákna na volné domény a vytlačení vlákna starého

Obrázek 3.14: Ukázka reakce přidání

- **Nahrazení**: Tato operace přidá nové horní vlákno do registru a zároveň vyvolá oddělení vlákna, které bylo horním vláknem před provedením reakce. Toto je zajištěno párováním s volnou doménou v registru, která se nazývá toehold. Po navázání na toehold nové vlákno vytlačí staré vlákno, pokud nové vlákno má dostatek komplementárních domén k spodnímu vláknu. Přesněji se musí jednat o situaci, kdy nové vlákno má maximálně o jednu komplementární doménu méně než vlákno staré, případně má komplementárních domén více. K nahrazení dojde díky 3-way branch migration. Je také možné, aby více instrukčních řetězců spolupracovalo na nahrazení horního vlákna.



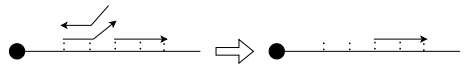
(a) Posunutí vlákna

(b) Nahrazení vláknem větším

(c) Kooperativní nahrazení

Obrázek 3.15: Ukázka reakce nahrazení

- **Oddělení**: Tato reakce vyvolá odstranění horního vlákna z registru bez zavedení nového vlákna na registr. Instrukční vlákno, které je komplementární k hornímu vláknu s převisem, je schopné toto vlákno z registru odstranit. Protože převis není spojen se spodním vláknem, je volný pro párování. Na tento převis se může párovat instrukční vlákno, které, pokud bude plně komplementární k hornímu vláknu, provede spojení s ním, tím odpojí toto vlákno od spodního vlákna.



Obrázek 3.16: Ukázka reakce oddělení

Aby následující reakce fungovaly správně, je nutné, aby jednotlivé domény byly k sobě ortogonální, tedy aby se mohly k sobě vázat domény, které jsou k sobě komplementární, jiné domény se na sebe nesmějí vázat.

Bylo zjištěno [45], že odpad, který vznikne těmito operacemi, již dál neinteraguje s registry díky tomu, že je v roztoku obsažen v malém množství, zatímco instrukční řetězce jsou přidávány v množství velkém.

Existují posloupnosti instrukcí, které jsou schopny nad daty vyvolat nedeterministické chování, tedy je možné, že po provedení instrukcí by se registr mohl nacházet ve dvou rozdílných stavech. Publikace [45], která představuje SIMD||DNA, se touto množinou posloupností instrukcí nezabývá a uvažuje jen deterministickou podmnožinu možných posloupností instrukcí.

3.6.3 Programy

V SIMD||DNA se programem nazývá sekvence instrukcí. V původní práci představující SIMD||DNA byly představeny dva programy: celulární automat s pravidlem 110 a binární čítač. Každý program v SIMD||DNA může definovat svou reprezentaci dat. Takže, i když oba tyto programy pracují s řetězcem binárních čísel, oba používají jinou reprezentaci pro jednotlivé bity. Reprezentace dat je zvolena tak, aby operace, které je možné nad SIMD||DNA provádět, bylo možné použít pro řešení problému, který je řešen programem.

3.6.4 Celulární automat s pravidlem 110

Celulární automat s pravidlem 110 je jednodimenzionální celulární automat s jednoduchým souborem pravidel, která určují, jak se vyvíjí stav každé buňky v průběhu času. Pravidla jsou definována funkcí $c_i(t+1) = f(c_{i-1}(t), c_i(t), c_{i+1}(t))$, kde $ci(t)$ představuje stav buňky i v čase t.

Konkrétní pravidla pro automat s pravidlem 110 jsou potom následující:

$$f(0,0,0) = 0; f(0,0,1) = 1$$
$$f(0,1,0) = 1; f(0,1,1) = 1$$
$$f(1,0,0) = 0; f(1,0,1) = 1$$
$$f(1,1,0) = 1; f(1,1,1) = 0$$

Na tato pravidla je možné pohlížet tak, že 0 se mění na 1 pouze tehdy, pokud stav souseda napravo od něj je 1, a 1 se mění na 0 pouze tehdy, pokud jsou oba sousedé nalevo a napravo 1.

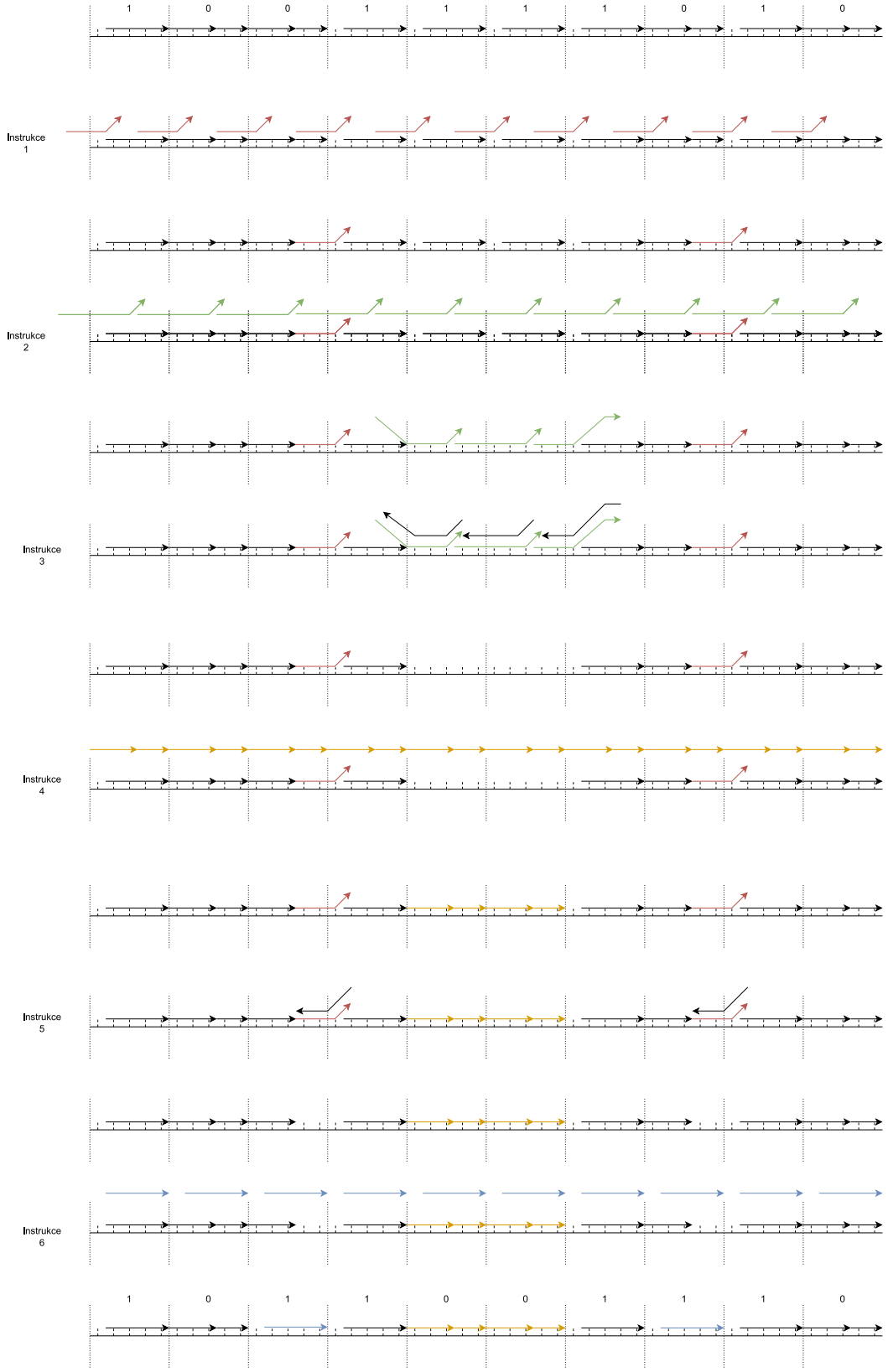
Tedy:

- Pokud je $c_i(t) = 0 \wedge c_{i+1}(t) = 1$ potom $c_i(t+1) = 1$ jinak $c_i(t+1) = 0$.
- Pokud je $c_i(t) = 1 \wedge c_{i+1}(t) = 1 \wedge c_{i-1}(t) = 1$ potom $c_i(t+1) = 0$ jinak $c_i(t+1) = 1$.

Tento jednoduchý soubor pravidel vede k složitým vzorům, pokud je iterativně aplikován na počáteční konfiguraci buněk. Automat s pravidlem 110 je známý svou turingovskou kompletností, která znamená, že může simulovat univerzální Turingův stroj a provádět libovolné výpočty s odpovídající počáteční konfigurací [12].

Instrukce implementující jednu iteraci celulárního automatu jsou zobrazeny na obrázku 3.17. Každá buňka ve stavu 0 je plně pokryta dvěma horními vlákny, jedno je délky tři domény a druhé je délky dvě domény. Každá buňka ve stavu 1 je částečně pokryta pěti-doménovým horním jednovláknem, které nepokrývá nejlevější doménu buňky. Program se skládá ze šesti instrukcí, kde první instrukce označí řetězec 01. Poté druhá a třetí instrukce vymažou vnitřní jedničky v libovolném řetězci, kde jsou alespoň tři po sobě jdoucí jedničky. Instrukce čtvrtá vyplní mezery, které vytvořily instrukce tři a dva, nulami. Poslední dvě instrukce provedou odstranění značek, které vytvořila instrukce jedna, a změní dříve označenou nulu na jedničku. Důkaz korektnosti programu rule 110 je následující:

- Během instrukce jedna nemohou instrukční řetězce vytlačit vlákno reprezentující stav 1, pokud se jedná o posloupnost 01, pak ve stavu 0 bude vytlačeno jedeno z vláken, jinak nebude vytlačen žádný řetězec.
- Uvnitř instrukce dva je řetězec na buňce i vytlačen kooperativně pouze tehdy, pokud jsou volné domény na obou stranách řetězce. Pokud na obou stranách bude uložena jednička, budou tyto domény volné, díky reprezentaci čísla jedna.
- Instrukce tři provede oddělení řetězců, které vytvořila instrukce dva, takže každá buňka, jejíž horní vlákna byla vytlačena instrukcí dva, je nyní volná.
- Instrukce čtyři nastaví všechny tyto volné buňky do stavu 0.
- Instrukce pět odstraní označení, které vytvořila instrukce jedna.
- Instrukce šest naváže vlákno reprezentující stav 1 do míst s volnou doménou, která vznikla instrukcí pět.



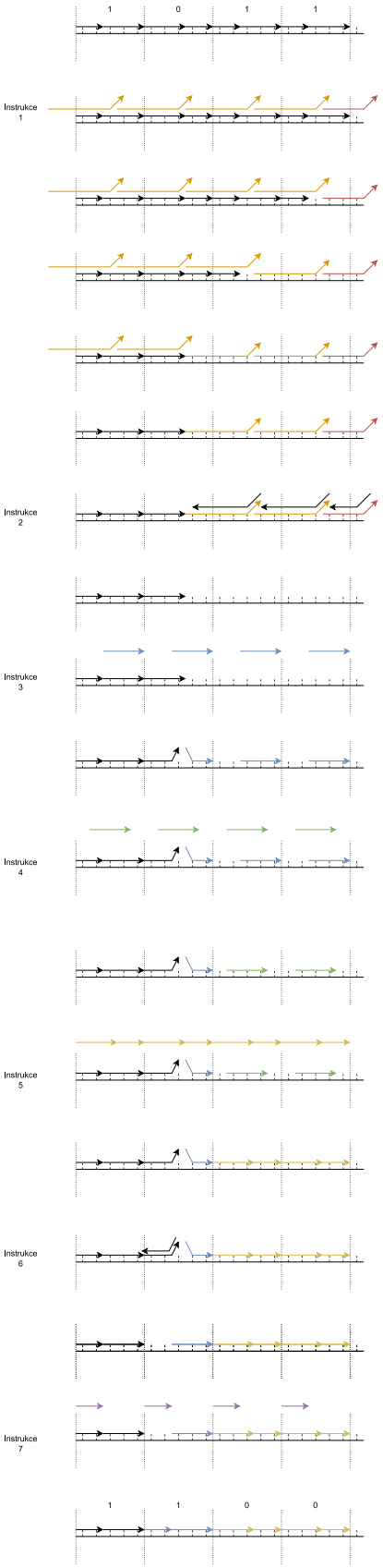
Obrázek 3.17: Ukázka běhu programu pro celulární automat s pravidlem 110. Jednotlivé barvy slouží k rozeznání rozdílných vláken.

3.6.5 Binární čítač

Program pro binární čítání provádí funkci $f(s) = s+1$. Pro binární čítání je typické to, že se všechny jedničky mění na nuly od nejméně významného bitu k nejvíce významným bitům až do první nuly, která se změní na jedničku. Všechny bity významnější než nejpravější nula zůstávají stejné. Například $f(1011) = 1100$, nebo $f(1000) = 1001$. V případě přetečení přepíšeme celý registr nulami. Na což se dá dívat tak, že na nejvyšším bitu (bit o jedno vyšší než máme velikost binárního čísla), který není součástí našeho čísla je konstantní nula tedy $f(1111) = 0000$.

Instrukce implementující přičtení jedničky k registru jsou na obrázku 3.18. Každá buňka ve stavu 0 je plně pokryta dvěma vlákny, přičemž jedno pokrývá první tři domény a druhé pokrývá zbylé dvě domény. Každá buňka ve stavu 1 je plně pokryta dvěma vlákny, přičemž jedno pokrývá první dvě domény a druhé pokrývá zbylé tři domény. K pravému okraji nejpravější buňky je přidána jedna extra doména, která slouží k iniciování operace. Program se skládá ze sedmi instrukcí, přičemž první dvě instrukce vymažou všechny jedničky mezi nejpravější buňkou a nejpravější buňkou ve stavu 0. Tato nejpravější buňka ve stavu 0 je označena pomocí instrukce tři. Změnu volných buněk na stav 0 provedeme instrukcemi čtyři a pět. Poslední dvě instrukce provedou změnu označené nuly na jedničku. Důkaz korektnosti programu binárního čítače je následující:

- Instrukce jedna zahajuje sérii reakcí od nejméně významného bitu n po nejpravější nulu, tato reakce probíhá sekvenčně. Nejprve instrukční řetězec s přečnívající doménou vytlačí řetězec pokrývající domény čtyři a pět. Pokud je nejméně významný bit jedna, po této vytlačovací reakci se stane doména tři volnou. Poté doména tři slouží jako otevřená doména pro zahájení další vytlačovací reakce s instrukčním řetězcem s přečnívající doménou. Podobné reakce na vytlačování pokračují až po buňku, která obsahuje nulu.
- Přidání vláken instrukce dva odpojí vlákna instrukce jedna z registru, díky tomu buňky od prvního nulového bitu do nejméně významného bitu zůstanou volné.
- Instrukce tři zapíše na volná místa vlákno, které slouží pro oddělení části vlákna reprezentující nulu.
- Instrukce čtyři posune tato přidaná vlákna instrukcí tři o jednu doménu doleva, pokud to lze, což otevře volnou doménu pro kooperativní vytlačení instrukcí pět.
- Instrukce pět kooperativním vytlačením nastaví nuly do buněk, které byly připraveny instrukcí čtyři.
- Instrukce šest a sedm odstraní označenou nulu a zapíší místo ní jedničku.



Obrázek 3.18: Ukázka běhu programu binárního čítače

3.7 Počítačová simulace DNA výpočtů

Počítačová simulace je metodou modelování reálného světa nebo hypotetických situací za využití počítače. Cílem je zkoumat a analyzovat chování systému a jeho fungování. Tato technika se stala klíčovým nástrojem při vytváření modelů přírodních systémů v oblastech např. fyziky, chemie a biologie. Kromě toho nachází uplatnění i při modelování systémů v oblastech jako je ekonomie a společenské vědy a přispěla k hlubšímu porozumění fungování inženýrských systémů [18] [37].

Tradičně se systémy formálně modelují prostřednictvím matematických modelů, které se snaží nalézt analytická řešení, umožňující předvídat chování systému na základě určených parametrů a počátečních podmínek. Počítačová simulace se stává stále častěji doplňkem nebo alternativou pro modelování systémů, u kterých není možné použít analytického řešení. Existuje mnoho různých typů počítačových simulací, které sdílejí snahu vytvářet příklady reprezentativních scénářů pro modely, u nichž není možné vytvořit kompletní výčet všech možných stavů [18] [37].

Simulátory, které slouží pro simulaci DNA počítání, jsou založeny na simulaci/predikci sekundární a terciální struktury DNA a RNA, která je určena párováním jednotlivých bází a reakcemi mezi strukturami vzniklými tímto párováním. Toto modelování lze provádět prakticky třemi různými způsoby, jedním z nich je možnost modelovat strukturu pomocí sekvenční podobnosti DNA/RNA se strukturou, u které známe její sekvenci. Další možností je vytvořit simulátor, který bude počítat energii jednotlivých částí DNA a bude odhadovat strukturu, která je energeticky nejvýhodnější. Poslední metodou je metoda, která je kombinací dvou výše uvedených metod. Tato metoda rozstříhá sekvence na menší podčásti, pro které hledá podobné sekvence, u kterých zná strukturu a následně pomocí heuristiky a dopočítání energií tyto podstruktury spojí do větší struktury [23] [13] [35].

V případě simulace DNA počítání se nejčastěji setkáme s dopočítáváním energie jednotlivých částí a následné hledání energeticky nejvýhodnější struktury. Podle aplikace simulátoru je možné rozeznat rozdíly v podrobnosti modelů. V případě některých simulací nám pouze stačí zjistit, zda na nějaké části vlákna DNA bude přilepeno jiné vlákno DNA a složitější struktura nás nezajímá. V takovém případě simulace řeší především problém zarovnání jednotlivých sekvencí vláken a problém, zda dané vlákno bude něčím vytlačeno nebo ne. Příkladem takového simulátoru je například simd-dna [2], který slouží pro simulaci SIMD||DNA architektury, Visual DSD nebo SDC-Sim, který je předmětem této práce. V případě nanotechnologií, které jsou algoritmicky sestavovány je situace komplikovanější. V takovém případě nás zajímá i prostorové uspořádání výsledného produktu. V takovém případě kromě párování bází a řešení problému zarovnání musíme navíc řešit i interakce takto párovaných vláken mezi sebou [23] [2].

Simulaci DNA počítání s vyšší přesností je možné provést stochastickou nebo deterministickou metodou. Stochastická metoda má velké výhody spočívající v rychlosti simulace, nicméně tyto metody neumožňují vypočíst chybovost jednotlivých operací nad DNA. Pokud chceme vypočíst chybovost operací nad DNA je nutné použít deterministickou metodu. V případě stochastických metod se často setkáváme s algoritmem Gillespie algorithm (Gillespie, 1977 [17]), který slouží ke generování cest chemickou reakční sítí, která reprezentuje možné reakce nad DNA. Deterministické simulátory řeší obvykle diferenciální rovnice, které popisují síly v médiu s DNA. Nejčastěji tyto simulátory používají metodu Rungeova–Kuttova, kterou používají na řešení diferenciálních rovnic [23].

3.8 Aplikace počítačové simulace DNA výpočtů

Simulátory DNA výpočtů slouží k modelování a analýze různých DNA výpočtových modelů a mohou být rozsáhle využívány i pro simulaci různých chemických systémů. Zvláště vynikají při návrhu nových systémů, protože nabízejí vizualizace, které usnadňují ladění programů. Tímto způsobem zjednodušují a zlevňují vědecký pracovní postup a zvyšují efektivitu a produktivitu při návrhu a analýze DNA výpočtových modelů. Umožňují uživatelům simulovat systémy před jejich fyzickým sestavením v laboratoři, což šetří čas a laboratorní zdroje. Simulátory rovněž umožňují zkoumat systémy, které by jinak nebylo možné sestavit z technických nebo jiných důvodů. Schopnost zápisu DNA algoritmů do programovacího jazyka usnadňuje provádění formální verifikace algoritmů bez dodatečných informací a výpočtů. Simulace a kontrola modelů poskytují důkaz správnosti implementace DNA výpočtu [23].

Kapitola 4

Návrh simulátoru DNA výpočtů SDC-Sim

Tato práce se zaměřuje na návrh, implementaci a aplikaci simulátoru DNA výpočtů, který má simulovat SIMD||DNA systémy, které byly popsány v části 3.6. Hlavní důraz bude kladen na ověření, zda lze pomocí těchto technik skutečně řešit uvedené problémy, měření výkonnosti simulátoru, jeho optimalizaci a implementaci jeho akcelerátoru.

Zvolený název simulátoru má odkazovat na stěžejní princip v SIMD||DNA systémech. Tímto principem je řetězová reakce (kaskáda) nahrazení DNA řetězců, anglicky **S**trand **D**isplacement **C**ascade. Tato reakce je pozorovatelná například na programu binárního čítače v sekci 3.6.5, ve chvíli, kdy dochází k vyvolání nové reakce díky dokončení jiné reakce, která nové reakci vytvořila vhodné podmínky. Díky podpoře **S**trand **D**isplacement **C**ascade reakce je schopen simulátor kromě SIMD||DNA systému simulovat i nemalou část obecných **S**trand **D**isplacement **C**ascade systémů.

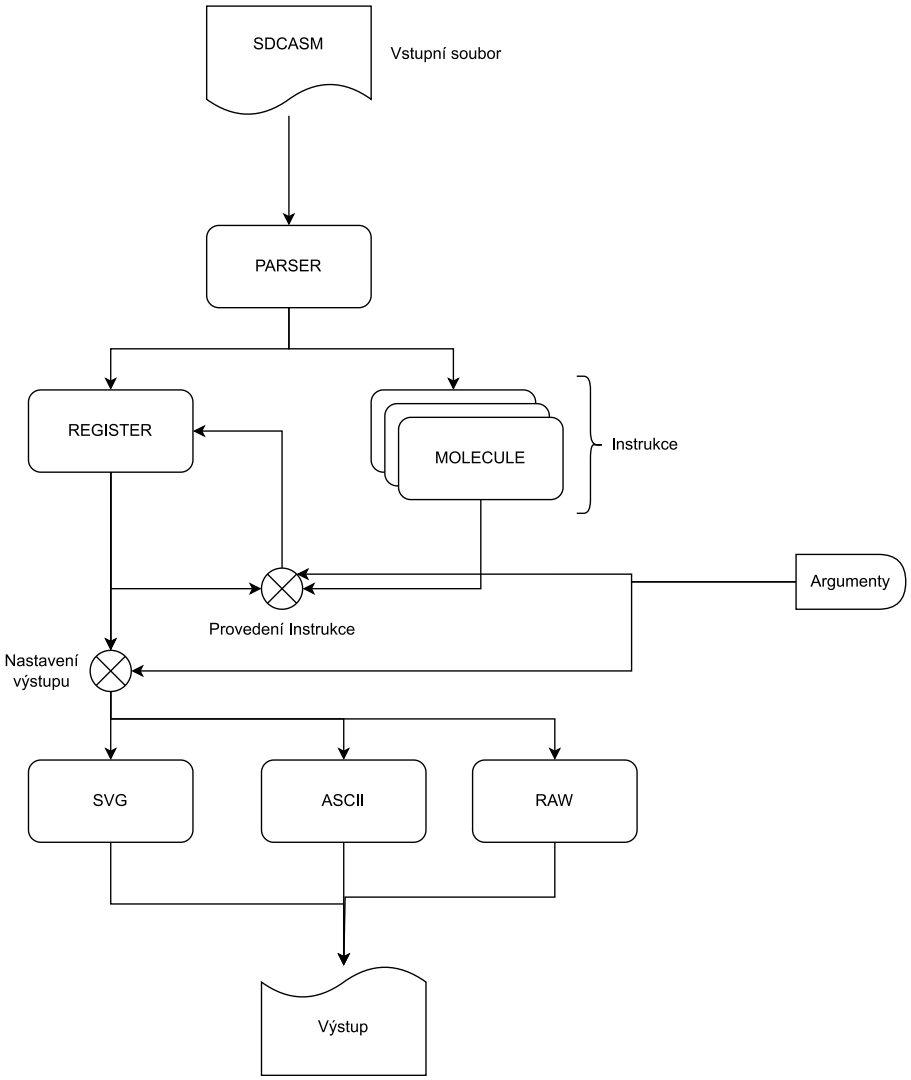
Pro implementaci simulátoru SIMD||DNA byl zvolen programovací jazyk C++ v prostředí operačního systému Linux. Výsledná aplikace bude pouze konzolová a nebude poskytovat grafické uživatelské rozhraní.

Simulátor se bude skládat z několika komponent, z nichž každá bude mít svou logickou funkci. Diagram jednotlivých komponent je ukázán na obrázku 4.1. Simulátor pomocí argumentu obdrží vstupní soubor, který bude obsahovat systém, který má simulovat. Tento soubor bude využívat speciální jazyk pro popis SIMD||DNA algoritmu, který bude inspirován jazykem DSD [22], který je využíván v simulátoru Visual DSD[1]. Ostatní parametry simulátoru poté specifikují, jakým způsobem má být simulace provedena. Vstupní soubor bude při spuštění simulátoru zpracován komponentou parser, která jeho obsah zpracuje a předá komponentám zajišťující simulaci.

Vstupní soubor bude po svém zpracování rozdělen na objekty dvou rozdílných typů. Jen z typů bude Registr, ten bude reprezentovat datovou část SIMD||DNA. Registr bude abstrakcí nad molekulou, která bude přidávat molekule funkce, které budou potřebné pro práci s registrem, jako bude například aplikace instrukce. Druhým typem objektu bude molekula. Parser vrátí pole polí molekul, které budou reprezentovat instrukční vlákna. Jedna instrukce bude moci být složena z více instrukčních vláken, z tohoto důvodu bude zvolena reprezentace polem polí, kde první dimenze bude reprezentovat jednotlivé instrukce a druhá jednotlivá instrukční vlákna v rámci instrukcí. Instrukce budou moci být aplikovány na registr pomocí funkce pro aplikaci instrukce.

[1] https://ph1ll1ps.github.io/project/visualdsd/

Aplikace instrukcí bude řízena hlavním programem, který bude iterativně aplikovat instrukce podle požadavků, které budou sděleny argumenty. Hlavní program také bude moci vyvolat vizualizační část, která bude podporovat zobrazení pomocí SVG, ASCII art nebo v surovém formátu.



Obrázek 4.1: Blokové schéma simulátoru

4.0.1 Reprezentace molekul DNA

Molekula bude v simulátoru reprezentovat strukturu vzniklou spojením několika vláken pomocí vodíkových vazeb. Molekula bude tedy strukturou, která bude schopna reprezentovat registr i instrukční vlákno v SIMD||DNA modelu. Molekula bude složena z vláken, která budou složena z domén. Vlákna budou ukládat domény pomocí pole. Jednotlivá vlákna budou spojena pomocí ukazatelů.

Reprezentace domén

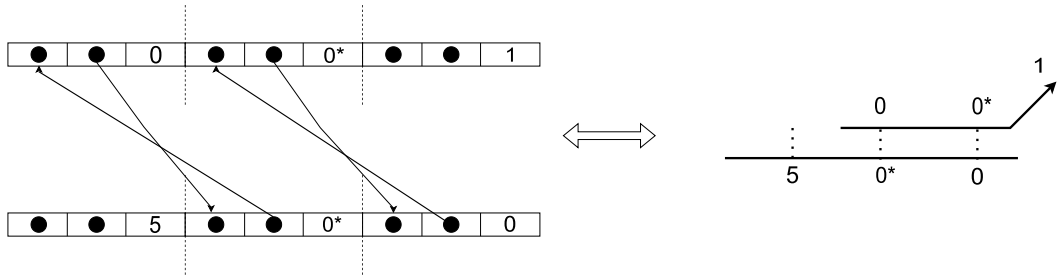
Jednotlivé domény budou reprezentovány pomocí šestnácti bitového čísla, které umožňuje reprezentovat i komplementaritu domény. Z šestnácti bitů je možné použít pouze patnáct bitů pro reprezentaci domény. Simulátor tedy bude umožňovat použití až $2^{15} = 32768$ domén. V případě nutnosti bude možné změnit počet bitů použitých na reprezentaci domén. V binárním čísle bude nejvýznamnější bit reprezentovat komplementaritu domény. Pokud budeme chtít získat komplementární doménu k doméně provedeme její binární negaci. Díky této jednoduché operaci, která je jednoduše implementovatelná v hardwaru, bude zajištěn rychlý výpočet komplementární domény. Pokud tedy budeme mít doménu 0, která bude binárně zapsána jako 0000000000000000, tak její komplementární doménou bude doména $0*$, která bude reprezentována jako 1111111111111111. Samotná doména bude v simulátoru abstrahována třídou, která bude abstrahovat tento datový typ.

Reprezentace vláken

Každé vlákno bude reprezentováno pomocí pole struktur. Tato struktura bude obsahovat tři položky. Jednou z položek bude ukazatel na vlákno, dále ukazatel na partnera a doména. Ukazatel na vlákno bude odkazovat vždy na objekt vlákna, ve kterém se tato struktura bude vyskytovat. Ukazatel na partnera bude sloužit pro reprezentaci spojení s jiným vláken. Doména pak bude reprezentovat doménu na dané pozici ve vlákně. Vlákno bude abstrahováno pomocí třídy, která bude obsahovat funkce umožňující práci s ním.

Reprezentace vodíkových můstků

Vodíkové můstky, které realizují spojení dvou vláken pomocí principu komplementarity, budou v simulaci reprezentovány pomocí křížového propojení dvou vláken za pomoci ukazatele na partnera. Ukazatel na partnera, který bude přítomen u každé domény, bude sloužit jako ukazatel na doménu, se kterou je spojen komplementaritou. Pokud nebude spojen s žádnou doménou, bude tento ukazatel nastaven na NULL. Tato reprezentace je ukázána na obrázku 4.2.



Obrázek 4.2: Reprezentace vodíkových můstků

Abstrakce molekuly DNA

Molekula DNA bude v této reprezentaci množina všech vláken, která jsou mezi sebou spojena vodíkovými můstky. Aby byla zajištěna abstrakce nad nízkoúrovňovými ukazateli, budou všechna vlákna, která budou mezi sebou spojena, uložena do objektu třídy, která bude reprezentovat molekulu. Tato třída bude obsahovat funkce na práci s molekulou, tedy funkce jako je přidání vlákna, jeho odstranění nebo spojení dvou vláken.

4.0.2 Parametry simulátoru

Uživatel bude moci ovlivňovat průběh simulace pomocí parametrů, které bude zadávat jako argumenty při spuštění simulátoru. Tyto parametry budou schopny ovlivňovat průběh simulace a výstup simulátoru. Jednotlivé argumenty budou následující:

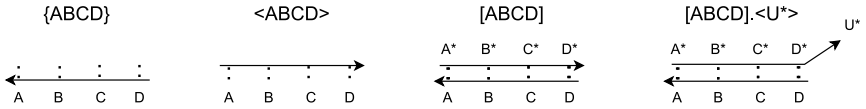
- **spaceing** - bude určovat velikost mezery mezi doménami v ASCII art výstupu. Tento parametr bude tedy použit, jen když bude zvolen ASCII art výstup, v jiných případech bude ignorován. Základně bude tento parametr nastaven na jednu mezeru.
- **format** - bude určovat typ výstupu. Bude muset být vybrán vždy jeden z množiny {ascii, raw, svg, dummy, assembly}. Základně bude nastaven na ascii.
- **all** - bude zapínat výpis jednotlivých kroků simulace (instrukce po instrukci). Tento parametr bude velmi užitečný při ladění SIMD||DNA programů. Základně bude tato funkce vypnuta.
- **decode** - povolí dekódování buněk registru pomocí maker, která budou definována v sdcasm souboru. Základně bude tato funkce vypnuta.
- **break** - umožní uživateli specifikovat, na kolikáté iteraci se simulace má zastavit. Tento parametr bude určen především pro ladění algoritmů pro architekturu SIMD||DNA a bude umožňovat zastavit simulaci v libovolné iteraci. Základně bude tento parametr nastaven na ∞.
- **time** - umožní specifikovat čas, po který budou registry vystaveny instrukčním vláknům. Tento čas tedy může zásadně ovlivňovat dobu simulace i výsledek simulace. Základně bude tento parametr nastaven na 30 v případě simulace na úrovni domén a na 999999 v případe simulace na úrovni nukleotidů.
- **nucleotides** - povolí provádění simulace za použití nukleotidů místo domén, což zásadně zpřesní výsledek simulace. Více o významu tohoto parametru je v části 4.0.5. Základně bude tato funkce vypnuta.
- **silent** - Omezí výstup simulátoru na nezbytné minimum.
- **gpu** - Povolí provedení simulace na nukleotidové úrovni na GPU akcelerátoru.
- **colors** - umožní specifikovat barevné schéma pro barvení výstupního svg obrázku. Schéma musí být z množiny {domain, chain, black}.
- **strands** - umožní specifikovat kvantitu přidávaných instrukčních řetězců, při simulaci na úrovni nukleotidů. Základně bude tento parametr nastaven na 100.
- **temperature** - umožní specifikovat teplotu, při simulaci na úrovni nukleotidů. Základně bude tento parametr nastaven na $25°C$.

4.0.3 Jazyk souborů SDCASM

Jazyk, který bude použit pro reprezentaci souborů SDCASM, bude inspirován jazyky DSD a x86 assembly. Každý soubor se bude skládat z maximálně čtyř sekcí, kterými budou datová sekce, sekce maker, sekce domén a sekce instrukcí. Datová sekce bude reprezentovat registry v architektuře SIMD||DNA. Sekce maker bude sloužit pro zjednodušení zápisu instrukcí a registrů pomocí jednoduché substituce slov. Poslední sekcí bude sekce instrukcí, která bude reprezentovat programy SIMD||DNA.

4.0.4 Jazyk pro popis molekul

V rámci této DP jsem navrhnul jazyk pro popis programů SIMD||DNA, který vychází z jazyku DSD [22] a adaptuje jej do prostředí architektury SIMD||DNA. Hlavní změnou oproti DSD je rozdělení zápisu na datovou a instrukční část a úprava jazyku pro popis molekul DNA, tak aby umožňoval popis dlouhého vlákna DNA, které má na sobě několik kratších DNA vláken, které nejsou vždy s ním plně spojeny. Jazyk pro popis molekul je silně inspirován jazykem DSD, který využívá tři typy závorek pro reprezentaci různých typů řetězců. Hranaté závorky [] jsou použity pro reprezentaci dsDNA, složené závorky **{}** jsou použity pro reprezentaci spodního vlákna typu ssDNA a závorky složené ze znaků **<>** jsou použity pro reprezentaci horního vlákna typu ssDNA. Jednotlivé domény, ze kterých je daný řetězec sestaven, se píší mezi ukončovací a otevírací závorku v pořadí, v jakém jsou ve vlákně. Domény jsou reprezentovány libovolným alfanumerickým znakem. Pokud je alfanumerický znak následován znakem *****, jedná se o reprezentaci komplementární domény k doméně, která je reprezentována alfanumerickým znakem. Jazyk pro popis molekul bude obohacen navíc o znak **.**. Znak **.** bude sloužit ke specifikaci, ke kterému dsDNA se má připojit horní řetězec ssDNA. Na obrázku 4.3 je příklad zápisu některých molekul DNA.



Obrázek 4.3: Zápis některých molekul jazykem SDCASM souborů

Syntaxe souboru SDCASM

Začátek každé sekce bude reprezentován pomocí znaku **:**. Tomuto znaku bude předcházet slovo, které bude reprezentovat název sekce. Název sekce bude muset být z množiny $\{define, data, instructions, domains\}$. Znak **:** je vždy následován novým řádkem, na kterém pokračují informace specifické pro danou sekci.

Sekce *define* (sekce maker) bude obsahovat makra, která budou použita v ostatních sekcích. V jazyce bude každé makro odděleno novým řádkem a jméno od hodnoty bude odděleno mezerou. Syntaxe definování makra bude tedy následující **<jméno> <hodnota>**. Kdekoli, kde bude použito **<jméno>**, bude toto jméno nahrazeno **<hodnota>**. Pokud bude zapnuta možnost **decode**, budou tato makra použita i na dekódování hodnot buněk. Toto dekódování bude provedeno nahrazením všech částí, které budou odpovídat **<hodnota>**, hodnotou **<jméno>**.

Sekce *data* (datová sekce) bude reprezentovat počáteční stav registrů. Každý registr bude zapsán na svém vlastním řádku a ukončen novým řádkem. Syntaxe zápisu molekuly reprezentující registr bude inspirována jazykem DSD a je blíže popsána v části 4.0.4.

Sekce *instructions* (sekce instrukcí) bude reprezentovat posloupnost instrukcí, která bude realizovat SIMD||DNA program. Každá instrukce bude zapsána na svém vlastním řádku a ukončena novým řádkem. Instrukce se bude skládat z několika molekul (minimálně jedné), které budou odděleny mezerou. Syntaxe zápisu molekuly bude inspirována jazykem DSD a je blíže popsána v části 4.0.4.

Sekce *domains* (sekce domén) bude obsahovat nukleotidové realizace jednotlivých domén. Tenta sekce stejně jako sekce *define* nebude povinná a nebude se muset ve funkčním kódu vyskytovat. V jazyce bude každá definice domény oddělena novým řádkem a jméno od hodnoty bude odděleno mezerou. Syntaxe definování domény bude tedy násle-

dující **<jméno> <hodnota>**. **<jméno>** bude reprezentovat jméno domény, které bude použito ve zbytku programu. Hodnota pak bude reprezentovat sekvenci nukleotidů, které budou realizovat danou doménu. Komplementy domény nebude nutné definovat, protože je simulátor vypočte sám. Povolené nukleotidy v hodnotě budou A, C, T, G a U, které jsou prvním písmenem skutečných bází nukleotidů DNA a RNA.

Každý řádek bude moci navíc obsahovat znak #, který bude reprezentovat jednořádkový komentář. Všechny znaky, které budou na řádku po tomto znaku, budou ignorovány.

```
#
# RULE 110 cellular automaton implementation in DNA|SIMD
# @autor Lukas Plevac <xpleva07@vutbr.cz>
# @date 11.21.2023
#

domains:
    A~CACATAC
    B CTTTACA
    C TCTCCT
    D ACACACA
    E CAAAACT
    F TCAAATC
    G TCCACT

define:
    0 [ABC][DE]
    1 {A}[BCDE]

data: # 0(6)
    1001111010

instructions:
    {D*E*A*F*} # mark 01
    {D*E*A*B*C*G*} # mark 11
    {DEABCG} # remove mark 11
    {A*B*C*} {D*E*} # write 0
    {DEAF} # remove mark 01
    {B*C*D*E*} # write 1
```

Obrázek 4.4: Obsah SDCASM souboru, který obsahuje implementaci celulárního automatu.

4.0.5 Algoritmus pro aplikaci instrukce na úrovni domén

Algoritmus 4 pro aplikaci instrukce bude algoritmem, který se bude snažit najít nejlepší zarovnání pro instrukční řetězce na registrovém spodním vlákně i vláknech horních. Algoritmus postupně vypočítá všechna zarovnání a následně eliminuje ta zarovnání, na kterých by se instrukční řetězce nemohl párovat. Algoritmus po odstranění vláken, která se nemohou párovat musí vypočíst, zda nějaká vlákna budou oddělena nebo ne. Po dokončení oddělení algoritmus dokončí jednu iteraci a pokusí se aplikovat opět stejné instrukční řetězce. Tyto operace bude opakovat do té doby, něž vyprší čas pro rekci. Pokud by bylo možné na nějaké místo párovat více instrukčních řetězců, simulátor zvolí ten řetězec, jehož párovací síla je nejvyšší.

Algorithm 4 Algoritmus pro aplikaci instrukce na doménové úrovni

Require: Instrukce v podobě pole molekul I a registr R
  $time \leftarrow 0$
  while $time < reaction\_time$ **do**
    R ← do-all-bidings(I, R) ▷ Připojí vlákna na všechna komplementární místa
    R ← remove-removed(R) ▷ Odstraní vlákna, která reagovala reakcí oddělení
    R ← remove-replaced(R) ▷ Odstraní vlákna, která byla odstraněna reakcí nahrazení
    R ← filter-stable-binded(R) ▷ Ponechá řetězce držící se na min. dvou doménách
    $time \leftarrow time + 1$
  end while
  return R

4.0.6 Algoritmus pro aplikaci instrukce na úrovni nukleotidů

Algoritmus 5 je pseudokódem představujícím stochastickou simulaci, která je založena na Metropolisově algoritmu [27] [21].

Fyzikálními objekty v simulaci reakcí DNA molekul jsou zde domény, které jsou reprezentovány sekvencí nukleotidů. Všechny DNA molekuly v simulaci jsou v teplotní rovnováze s teplotou T.

Důležitou vlastností simulovaného termodynamického systému je Gibbsova volná energie, která je definována jako $G = H_b - TS_u$, kde H_b je celková vazebná energie, která je záporná, a S_u je entropie nespárovaných bází, která je pozitivní. Dále Boltzmannova pravděpodobnost, která říká, že systém bude v částicovém stavu C, je následující $\frac{1}{Z}e^{-G(C)/(k_BT)}$, kde $Z = \sum c^{e^{-G(C)/k_BT}}$ a k_B je Boltzmannova konstanta [21].

Uvažujme oba extrémní limity teploty. Pokud bude teplota maximální, budou všechny domény DNA vláken denaturované, protože entropie v Gibbsově volné energii nabývá vysokých hodnot a tím se stává vazebná energie nepodstatnou. Naopak pokud teplota bude nejnižší, budou ssDNA formovat dsDNA, protože hodnota entropie bude tak nízká, že entropie nespárovaných bází se stane nepodstatnou a zůstane pouze vazebná energie [21].

Protože hodnota entropie v Gibbsově volné energii klesá, s teplotou dochází k formování dsDNA, které nejsou vždy plně komplementární, protože v systému není energie, která by bránila formování takových dsDNA. Z tohoto důvodu je nutné systém SIMD||DNA provozovat v takovém rozsahu teplot, ve kterém nedojde k denaturaci celého systému a zároveň nebude docházet k navazování domén, které k sobě nejsou komplementární.

Proto skládání ssDNA do dsDNA nebo rozkládání dsDNA do ssDNA bude určeno termodynamikou vazebné energie a entropie v Gibbsově volné energii [21].

Algorithm 5 Algoritmus pro aplikaci instrukce na nukleotidové úrovni

Require: Instrukce v podobě pole molekul I, registr R a množství instrukčních vláken I_c

```
time ← 0
R ← R ∪ copyAndAmplify(I,I_c)          ▷ Vytvoření I_c kopií instrukčních vláken
while time < reaction_time do
    IStrand ← randomStrand(R)          ▷ Získá náhodné vlákno z registru
    JStrand ← randomStrand(R)          ▷ Získá náhodné vlákno z registru
    IDomain ← randomDomain(IStrand)    ▷ Získá náhodnou doménu z vlákna
    JDomain ← randomDomain(JStrand)    ▷ Získá náhodnou doménu z vlákna
    if random() < 0.5 then             ▷ S 50% pravděpodobností se zvolí renaturace
        if random() <= BindProbability(IDomain, JDomain) then
            R ← renature(IDomain, JDomain, R)   ▷ Provede renaturaci dvou domén
        end if
    else                               ▷ S 50% pravděpodobností se zvolí denaturace
        if random() <= UnBindProbability(IDomain, JDomain) then
            R ← denature(IDomain, JDomain, R)   ▷ Provede denaturaci dvou domén
        end if
    end if
    if time modulo 1024 = 1023 then
        R ← DeterministicBind(R)
    end if
    time ← time + 1
end while
R ← elution(R)   ▷ Provede odstranění vláken, která jsou spojena jen jednou doménou
return R
```

V algoritmu 5 je použita funkce DeterministicBind(), která slouží k deterministickému dokončení nahrazení vlákna vláknem jiným. Toto nahrazení se totiž řídí jinými pravděpodobnostmi, než jsou v simulátoru použity. Obecně platí, že když má dojít k nahrazení vlákna vláknem jiným, stane se tato reakce s pravděpodobností blízké 1. Z tohoto důvodu stačí najít deterministicky všechny řetězce, které se mají této reakce účasnit a deterministicky reakci provést.

Hodnota 1024 byla zvolena podle výsledků experimentů s různými hodnotami a byla vybrána jako nejvhodnější.

Pravděpodobnosti akceptace A denaturace a renaturace domén vyházejí ze stejné rovnice, která je následující $A = min(1, e^{-\Delta G/(k_B T)})$, kde $\Delta G = G(C_{new}) - G(C)$, kde C_{new} je nová konfigurace a C je současná konfigurace [27].

Když dvě ssDNA vytvoří dsDNA, je možné ΔG vypočítat jako $\Delta G = \Delta H - T\Delta S$, kde ΔH může být snadno vypočítáno jako suma vazebných energií jednotlivých párů bází a ΔS jako záporná suma entropií všech bází, které byly původně volné. Jako entropie jedné volné báze byla použita hodnota 23 cal/(deg MBP), vazebné energie jednotlivých párů jsou pak v tabulce 4.1 [21].

V případě, že dojde k rozpadu dsDNA na dvě ssDNA, vychází se ze stejné rovnice jako u opačné reakce, pouze dojde ke změně znamének u ΔS a ΔH.

Bázový pár	Vazebná energie (kcal/MBP)
G-C	-9.0
A-T	-7.2
Ostatní	-5.4

Tabulka 4.1: vazebné energie jednotlivých párů (Klumb and Ackermann, 1971) [21]

4.0.7 Automatické testy

K simulátoru bude sestavena množina automatických testů, které budou simulátor testovat na dvou úrovních abstrakce. Jednou z úrovní budou unit testy. Tyto testy budou testovat funkčnost jednotlivých bloků, jako je molekula, doména a podobně. Dále budou testovat chování těchto bloků a správnost výstupů jejich funkcí. Druhou úrovní budou celkové testy, které budou testovat správnost fungování celého simulátoru. Tyto testy se budou skládat z vstupu v podobě SDCASM souboru a souboru, který bude obsahovat očekávaný výstup simulátoru. Tyto testy budou sestaveny podle otestovaných programů a reakcí, které na SIMD||DNA byly zkoušeny v původní publikaci. Tedy se bude jednat o simulaci celulárního automatu s pravidlem 110 a binárního čítače. Tyto testy mohou být doplněny o ručně vytvořené testy, které budou nejdříve ověřeny jinými simulátory, které umožňují simulovat systémy typu SIMD||DNA.

4.0.8 Paralelizace pomocí OpenMP

Hlavní výhodou DNA výpočtů oproti konvenčním technikám je jejich masivní paralelismus. Avšak při sekvenční simulaci na počítači tato výhoda mizí a výpočet trvá velmi dlouho. Z tohoto důvodu bude do simulátoru přidán paralelismus na úrovni jader CPU. Je evidentní, že na současných počítačích není možné dosáhnout tak masivního paralelismu, jaký je možný při provádění DNA operací. Z tohoto důvodu tedy simulace těchto systémů i po zavedení paralelizmu bude pomalá.

OpenMP představuje speciální API, které umožňuje paralelní programování v jazycích C, C++ a Fortran na většině operačních systémů a procesorových architekturách. Tento standard pro programování počítačů se sdílenou pamětí zahrnuje soubor direktiv pro překladač, knihovních procedur a proměnných prostředí, které ovlivňují způsob provádění aplikace.

OpenMP poskytuje implementaci paralelizmu na více úrovních. Nejnižší úrovní je podpora SIMD jednotek uvnitř procesoru a nejvyšší úrovní je podpora paralelizmu na úrovni vláken. OpenMP v nejnovějším standartu také podporuje externí akcelerátory jako jsou grafické karty nebo FPGA.

Implementace paralelizace pomocí více vláken je založena na vytváření podvláken. Hlavní vlákno (master thread) vytváří podle potřeby skupinu podvláken (subthread), do které jsou úkoly určené pro výpočet rozděleny. Tato vlákna pak pracují paralelně. Každé vlákno je identifikováno unikátním ID, které lze získat pomocí funkce, kterou nám API nabízí. ID vlákna je datového typu integer, přičemž hlavní vlákno vždy nese ID 0.

OpenMP má několik vrstev, přičemž uživatel komunikuje s OpenMP prostřednictvím aplikace. Tuto aplikaci vytváří programátor, který má možnost využít direktivy překladače, knihovní procedury nebo proměnné prostředí z programovací vrstvy. Všechny tyto prvky ovlivňují běhové knihovny v systémové vrstvě, které jsou implementovány specificky pro daný operační systém. Programovací vrstva je tak nezávislá na operačním systému, zatímco

systémová vrstva je na něm závislá. Poslední vrstvou, která má výrazný vliv na rychlost paralelizace skrze počet a takt jader, je hardware.

API

Direktivy pro jazyky C a C++ v rámci OpenMP jsou definovány pomocí pragma preprocesorového příkazu. Zápis direktiv je citlivý na velikost písmen, a proto začíná každá OpenMP direktiva **#pragma omp**. Platnost každé direktivy OpenMP je omezena na následující strukturovaný blok instrukcí. Každou z těchto direktiv lze dále upravit pomocí různých klauzulí.

```
#pragma omp <direktiva> [ klauzule ]
```

Jednotlivé direktivy nám umožňují specifikovat typ paralelizace, případně specificitu nějakého příkazu, například nutnost bariéry, či nutnost atomičnosti operace. Direktivy, které určují typ paralelizace jsou **simd** a **parallel**.

- **simd** specifikuje, že daná oblast má být provedena na SIMD jednotkách
- **parallel** specifikuje, že daná oblast má být provedena pomocí paralelních vláken

Na klauzule je možné se dívat jako na argumenty pro direktivy. Klauzule slouží pro specifikaci, jakým způsobem danou direktivu provést, případně zda je nutné před jejím provedením něco provést. Například zde najdeme informace jak přistupovat k proměnným, způsob mapování paměti, typ redukce atd.

Jednotlivé direktivy je možné také kombinovat buď v rámci jedné pragmy nebo zanořením v jednotlivých blocích pomocí více pragmat.

4.0.9 Paralelizace pomocí OpenACC

OpenACC představuje speciální API, které umožňuje stejně jako openMP paralelní programování v jazycích C, C++ a Fortran na většině operačních systémů a procesorových architekturách. Tento standard pro programování počítačů se sdílenou pamětí zahrnuje soubor direktiv pro překladač, knihovních procedur a proměnných prostředí, které ovlivňují způsob provádění aplikace.

OpenACC je v mnoha ohledech podobné openMP a rozdílů je mezi nimi opravdu málo. Nejmarkantnějším rozdílem je, že OpenMP se dlouho zaměřovalo na paralelizaci na CPU nikoly na GPU a jiných akcelerátorech, o což se naopak snažilo OpenACC. Z tohoto důvodu je u starších GPU podpara pro OpenACC ale nikoly pro OpenMP, které přišlo s GPU paralelizací později. Během vývoje bude použito GPU Tesla P4, které je generace Pascal. Kompilátor NVHPC bohužel nepodporuje openMP standard pro generaci Pascal, podporuje pouze openACC. Z tohoto důvodu bude paralelizace na GPU provedena pomocí OpenACC a na CPU pomocí OpenMP.

API

Direktivy pro jazyky C a C++ v rámci OpenACC jsou definovány pomocí pragma preprocesorového příkazu. Zápis direktiv je citlivý na velikost písmen, a proto začíná každá OpenACC direktiva **#pragma acc**. Platnost každé direktivy OpenACC je omezena na následující strukturovaný blok instrukcí. Každou z těchto direktiv lze dále upravit pomocí různých klauzulí.

```
#pragma acc <direktiva> [ klauzule ]
```

Jednotlivé direktivy nám umožňují specifikovat typ paralelizace případně specificitu nějakého příkazu například nutnost bariéry, či nutnost atomičnosti operace. Direktivy, které určují typ paralelizace, jsou **kernel** a **parallel**.

- **kernel** specifikuje, že daná oblast má být automaticky zparalelizována a provedena na akcelerátoru. Překladač se zde sám rozhoduje jaké smyčky zparalelizuje a kde vloží redukci, atomické instrukce a datové transfery.
- **parallel** specifikuje, že daná oblast bude prováděna paralelně a způsob paralelizace bude dále určen podle pragmat uvnitř bloku. Při správně provedené paralelizaci je typicky tento typ rychlejší.

Další nedílnou součástí jsou direktivy pro specifikaci typu paralelizace smyček, pomocí které kompilátoru sdělujeme, že se na daném řádku nalézá smyčka a jakým způsobem s ní má zacházet.

- **loop** specifikuje, že daná oblast je smyčkou, a že má být zparalelizována
- **loop seq** specifikuje, že daná oblast je smyčkou, a že nemá být zparalelizována

Neméně důležitou částí jsou klauzule určující datové transfery.

- **copyin(var1, var2, ...)** specifikuje, že daná proměnná má být před spuštěním paralelní sekce přenesena z paměti počítače do GPU.
- **copyout(var1, var2, ...)** specifikuje, že daná proměnná má být po dokončení paralelní sekce přenesena z GPU zpět do paměti počítače.
- **create(var1, var2, ...)** specifikuje, že daná proměnná má být před spuštěním paralelní sekce vytvořena v paměti GPU.
- **delete(var1, var2, ...)** specifikuje, že daná proměnná má být po dokončení paralelní sekce odstraněna z paměti GPU.

Na klauzule je možné se dívat jako na argumenty pro direktivy. Klauzule slouží pro specifikaci, jakým způsobem danou direktivu provést případně, zda je nutné před jejím provedením něco provést. Například zde najdeme informace jak přistupovat k proměnným, způsob mapování paměti, typ redukce atd.

Jednotlivé direktivy je možné také kombinovat buď v rámci jedné pragmy nebo zanořením v jednotlivých blocích pomocí více pragmat.

Kapitola 5

Experimentální výsledky

5.1 Návrh SIMD||DNA programů

Všechny níže uvedené programy jsou novými programy, které byly navrhnuty v rámci této diplomové práce za pomoci simulátoru, který byl také předmětem této diplomové práce. Všechny zmiňované programy jsou v této sekci popsány a ukázány výsledky jejich simulace na doménové úrovni. Zdrojové kódy těchto programů jsou součástí příloh této práce.

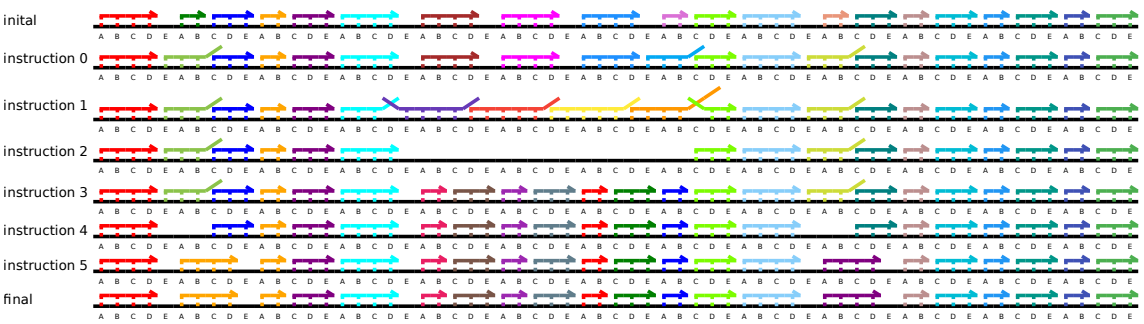
5.1.1 Rule28

Program realizující celulární automat s pravidlem 28 je založen na programu celulárního automatu s pravidlem 110. Pracuje úplně na stejném principu jako program pro celulární automat 110 pouze s tím rozdílem, že je pozměněna reprezentace stavu 1. Reprezentace stavů jsou na obrázku 5.1.

(a) stav 0 (b) stav 1

Obrázek 5.1: Reprezentace stavů v programu rule28

Instrukce implementující jednu iteraci celulárního automatu jsou zobrazeny na obrázku 5.2. Program se skládá ze šesti instrukcí, kde první (nultá) instrukce označí řetězec 10. Poté druhá a třetí instrukce vymažou vnitřní jedničky v libovolném řetězci, kde jsou alespoň tři po sobě jdoucí jedničky. Instrukce čtvrtá vyplní nulami mezery, které vytvořily instrukce tři a dva. Následující a zároveň poslední dvě instrukce provedou odstranění značek, které vytvořila instrukce jedna, a změní dříve označenou nulu na jedničku.



Obrázek 5.2: Běh programu implementující jednu iteraci celulárního automatu rule 28

5.1.2 Selected NOT

Selected NOT je podprogramem programu Shift left. Selected NOT slouží pro provedení negace binární hodnoty, která bude vybrána takzvaným selektorem. Selektor je místo na DNA vlákně registru, které umožňuje párovat speciální vlákno DNA. Podle přítomnosti vlákna se následně rozhodne, zda dojde k negaci nebo ne.

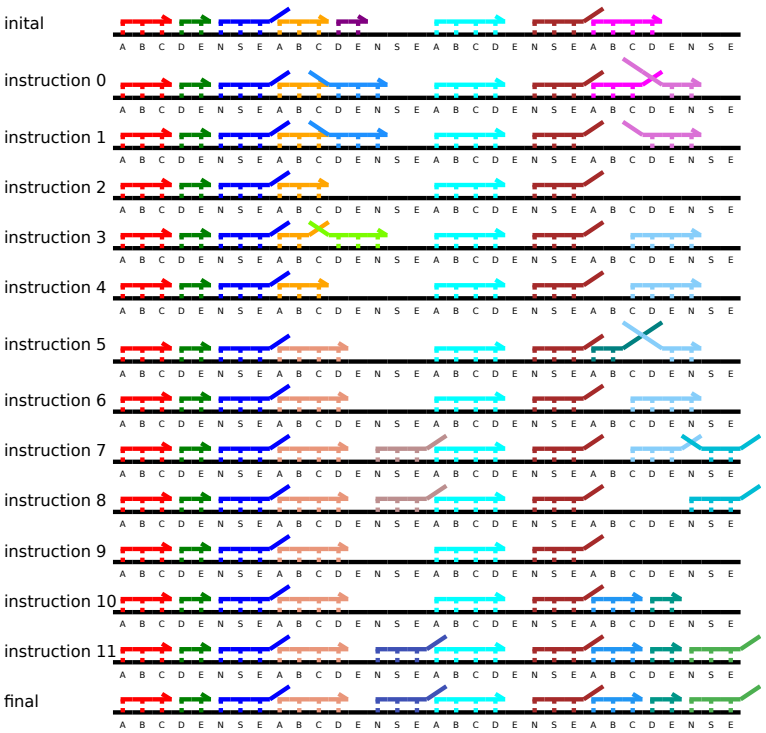
V tomto programu existují čtyři validní stavy buňky registru. První dva reprezentují binární hodnoty 1 a 0, zbylé dva stavy reprezentují tyto binární hodnoty se zapnutým not selektorem. Not selektor je reprezentován posledními třemi doménami v každé buňce registru. Všechny čtyři stavy jsou zobrazeny na obrázku 5.3.

(a) stav 0 (b) stav 1 (c) stav 0 NOT (d) stav 1 NOT

Obrázek 5.3: Reprezentace stavů v programu Selected NOT

Jak lze pozorovat na obrázku 5.3, buňky které mají být negovány mají poslední tři domény volné. Tyto tři volné domény slouží na vyvolání reakce, která provede samotnou negaci.

Program funguje následovně. Nultá instrukce zaznačí buňky určené k negaci tak, že pokud hodnota, která má být negována, bude logická jedna, dojde i k částečnému odvázání vlákna reprezentující logickou jedničku. V případě, že se jedná o logickou nulu, dojde k plnému nahrazení druhého vlákna v reprezentaci logické nuly. První a druhá instrukce odstraní logickou jedničku, která byla částečně odvázána instrukcí nultou, a také odstraní vlákno, které bylo nultou instrukcí. Třetí instrukce slouží pro označení místa, na které má být zapsána jednička (mark1). Pokud místo bude úplně prázdné, tedy jedná se o logickou jedničku, která má být negována na nulu, bude tato instrukce plně spárována s registrem, pokud by se jednalo o logickou nulu, dojde k neúplnému spárování kvůli prvnímu vláknu v logické nule, které bude touto instrukcí částečně odpárováno. Čtvrtá instrukce odstraní mark1 všude, kde se nedokonale spároval, tedy na místě, na kterém je logická nula. Instrukce pátá přímo zapíše logickou jedničku. Místo, na kterém se nachází jedno vlákno logické nuly, bude plně nahrazeno vláknem logické jedničky. Pokud by se na místě nacházel mark1 dojde k nedokonalému napárování. Šestá instrukce provede odpárování všech nedokonale napárovaných logických jedniček, které vznikly předchozí instrukcí. Instrukce sedmá až desátá provádějí odstranení mark1 a zapsání logické nuly. Poslední instrukce vypne všechny not selektory. Průběh tohoto programu je ukázán na obrázku 5.4.



Obrázek 5.4: Běh programu Selected NOT

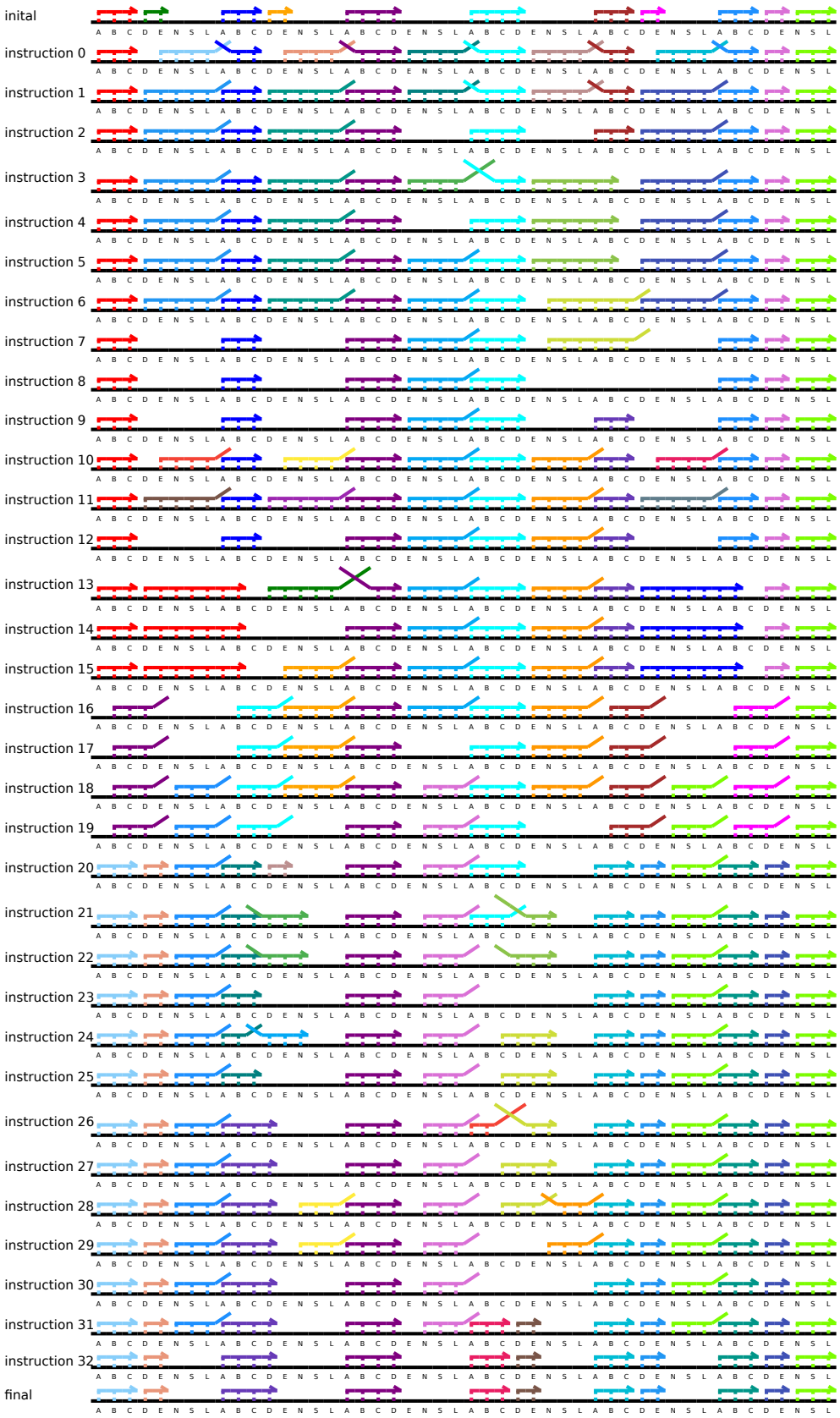
5.1.3 Shift left

Shift left je jednoduchým programem, který implementuje binarní posun o jedna do leva. Tento program využívá podprogram Selected NOT, který je popsán v části 5.1.2. Program nejdříve zjistí, které bity mají být negovány, nastaví podle toho not selektory a následně spustí Selected NOT podprogram. Shift left program má pouze dva validní vstupněvýstupní stavy, jimiž jsou logická jedna a logická nula. Reprezentace těchto stavů vychází z podprogramu Selected NOT a je uvedena na obrázku 5.5. Aby vše fungovalo správně, je nutné mít nejvíce na levo nulu, která bude mít vypnut not selektor.

(a) stav 0 (b) stav 1

Obrázek 5.5: Reprezentace stavů v programu Shift left

Nultá instrukce odpáruje druhé vlákno ze stavu nula a vytvoří tím jednu volnou doménu. Pokud by se v buňce nacházela místo nuly jednička, dojde k párování instrukce na volnou doménu E a žádné vlákno nebude nahrazeno. První instrukce nahradí předchozí instrukci tam, kde je volná doména D tedy tam, kde došlo k odpárování vlákna reprezentující nulu. Druhá instrukce odstraní všechny zbylé instrukční vlákna instrukce nula, která nebyla nahrazena předchozí instrukcí. Instrukce třetí částečně odpáruje první vlákno reprezentující nulu, pokud je napravo od ní nula, pokud je na pravé straně jednička dojde k úplnému napárování na volné místo. Čtvrtá instrukce provede odstranění předchozí instrukce v případě, že je na pravé straně jednička. Pátá instrukce se napáruje na not selector, aby nebyl ovlivněn jinou instrukcí, protože došlo k detekci stavu dvou jedniček po sobě (nebude docházet k negaci). Instrukce šest až devět řeší odstranění všech instrukčních vláken, která jsme napárovali předchozími instrukcemi s výjimkou instrukce, která vypínala not selektor. Instrukce desátá se snaží napárovat na volné místo mezi buňkami (pokud je napravo jednička, nezůstane po napárování instrukce žádné volné místo, ale pokud je napravo nula, zůstane po napárování jedna volná doména). Instrukce jedenáctá provede nahrazení předchozí instrukce všude tam, kde je volná doména způsobena nulou. Instrukce dvanáctá odstaní všechny výskyty instrukce jedenácté. Instrukce třináctá provede částečné odpárování nuly, pokud je nalevo jedna a plně se napáruje, pokud je nalevo jednička. Instrukce čtrnáctá odpáruje všechny vlákna předchozí instrukce, která se plně nenapárovala. Instrukce patnáctá provede uzamčení not selektoru, aby nemohl být ovlivněn jinou instrukcí, protože byla detekována situace, kdy nula je následována jedničkou (tedy chceme, aby došlo k negaci). Instrukce šestnáctá provede odstranění instrukce třinácté. Instrukce sedmnáctá provádí odstranění dočasného not selektoru, který jsme umístili v případě, že jsme detekovali dvě jedničky. Instrukce osmnáctá se napáruje na not selektor a brání negaci buněk tam, kde jsme detekovali dvě nuly nebo dvě jedničky po sobě. Instrukce devatenáctá odstraní všechny předchozí instrukce, které by se mohly ještě na registru vyskytovat s výjimkou instrukce minulé. Instrukce dvacátá provede opětovné zapsání nuly, aby došlo k opravě stavu nula, který mohl být částečně poškozen předchozími instrukcemi (díky vhodné reprezentaci stavů nula nemůže nahradit zapsanou jedničku). Následující instrukce jsou instrukcemi podprogramu Selected NOT, který je popsán v části 5.1.2. Ukázka běhu simulace programu Shift left je na obrázku 5.6.



Obrázek 5.6: Běh programu Shift left

5.1.4 Mul 2

Mul 2 je programem, který realizuje simulaci třístavového celulárního automatu, který provádí násobení čísla zapsaného v unárním formátu číslem dva. Tento program slouží k demonstraci schopnosti SIMD||DNA systému pracovat i s celulárními automaty, které mají více jak dva stavy, přímo pomocí reprezentace každého stavu v SIMD||DNA systému bez nutnosti simulování třístavového celulárního automatu automatem dvoustavovým. Tento program má tři stavy, jejichž reprezentace je zvolena tak, aby bylo možné snadno provádět nutné porovnání levého a pravého souseda. Celulární automat, který bude simulován má vždy na svém začátku konfiguraci, která se převážně skládá ze stavu dva a stavu nula. Pomocí stavu dva je unárně zapsáno číslo, které má být vynásobeno dvěma. Nuly slouží jako volné místo pro dvojnásobek původního čísla v unární reprezentaci. Zjednodušeně by se fungování celulárního automatu dalo popsat tak, že pokud by dvojku následovala nula, budou oba tyto stavy nahrazeny stavem jedna. Pokud by stav jedna byl následován stavem dva, dojde k posunu stavu dva dopředu. Tento slovní popis je popsán následující přechodovou funkcí, kde X představuje libovolný stav:

$$f(0,2,X) = 1; f(X,1,2) = 2$$
$$f(X,0,2) = 1; f(1,2,X) = 1$$
$$f(X,0,0) = 0; f(X,1,1) = 1$$
$$f(X,0,1) = 0; f(2,2,X) = 2$$

Vzhledem k tomu, že program bude muset detekovat všechny přechody mezi stavem dva a okolím, je stav dva zvolen tak, aby bylo z obou stran možné k němu navázat vlákna. Ostatní stavy jsou navrhnuty tak, aby při detekci okolí bylo možné detekovat, který stav se stavem dva sousedí. Každý stav je definován na buňce o velikosti šesti domén a stav dva je jediný stav, který má nějaké domény volné. Reprezentace jednotlivých stavů je na obrázku 5.7.

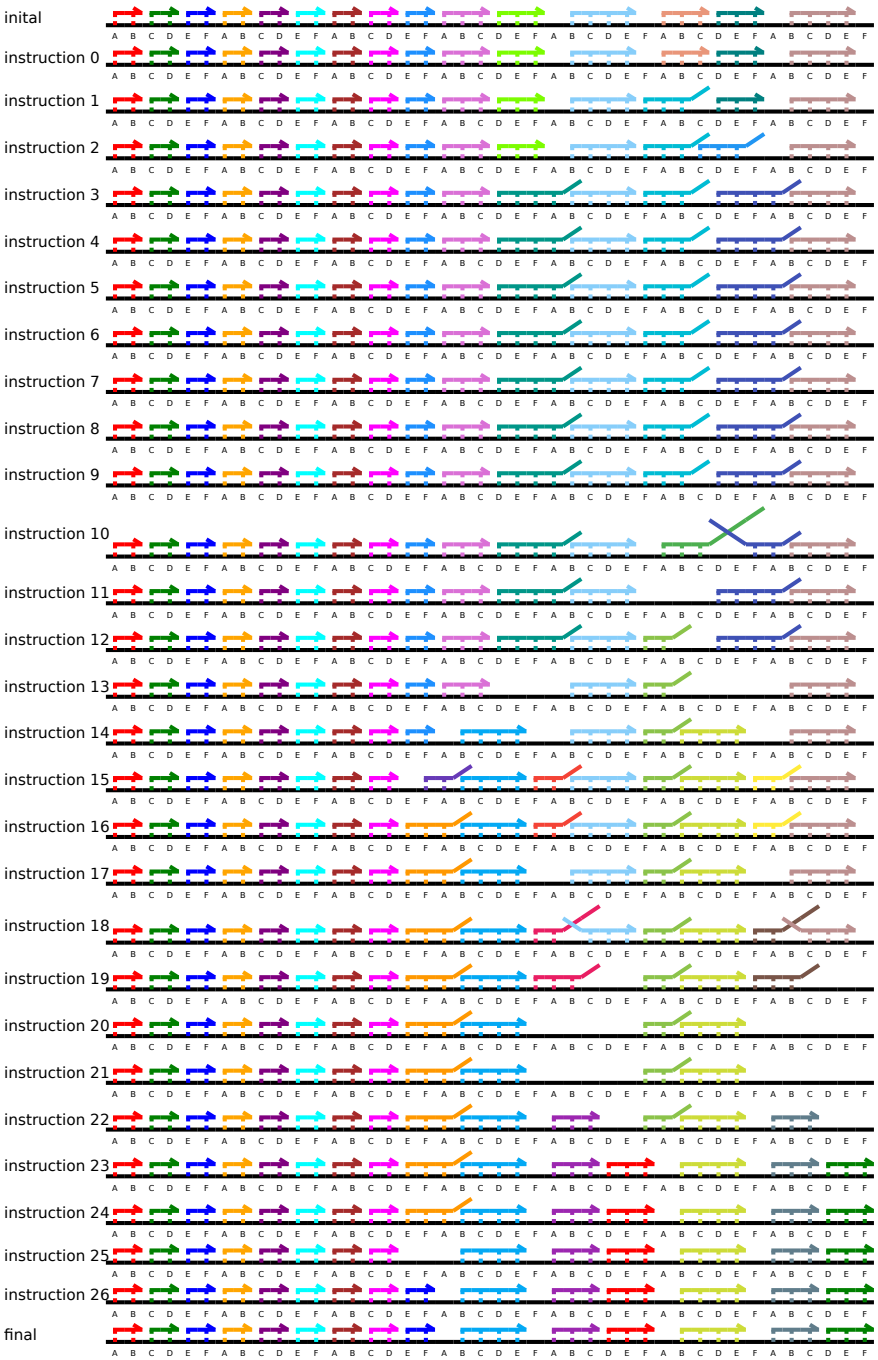
(a) stav 0 (b) stav 1 (c) stav 2

Obrázek 5.7: Reprezentace stavů v programu Mul 2

Program pak funguje následovně. Nultá až třetí instrukce provádí označení přechodů nula-dva, dva-dva, dva-nula a jedna-dva. Instrukce čtvrtá až šestá provádí odstranění označení přechodu nula-dva. Instrukce sedmá až devátá provádí odstranění stavu dva z přechodu nula-dva. Instrukce desátá provádí označení tohoto místa značkou označující zápis jedničky.

Instrukce jedenáctá a dvanáctá provádí zaznačení přechodu stavu dva na jedna, aby nedošlo k přepsání této změny následujícími instrukcemi. Instrukce třináctá provádí odstranění značky přechodu jedna-dva. Instrukce čtrnáctá zapíše na všechna místa stav dva, tento zápis půjde provést pouze na místa, která byla vytvořena předchozí instrukcí, všechna ostatní místa jsou obsazena značkami. Instrukce patnáctá provádí částečné odpárování stavu nula, pokud je nula v přechodu nula-dva. Instrukce šestnáctá vytvoří značky na místech, na kterých došlo k reakci, kterou provedla instrukce předchozí. Instrukce sedmnáctá provádí odstranění všech vláken instrukce patnácté. Instrukce osmnáctá až dvacátá provedou odstranění stavu dva v přechodu nula-dva. Instrukce dvacátá první provádí odstranění značky,

která značí zápis jedničky. Instrukce dvacátá druhá až dvacátá třetí provádí zapsání jedničky, která půjde zapsat pouze na místa, která vznikla reakcí předchozí instrukce. Instrukce dvacátá čtvrtá až dvacátá pátá provedou odstranění značek, které zůstaly na registru. Poslední instrukce provádí opravu reprezentace nuly, která mohla být v průběhu programu poškozena. Simulace tohoto programu je na obrázku 5.8.



Obrázek 5.8: Běh programu Mul 2

5.2 Vlastnosti navržených programů

5.2.1 Rule28

Program Rule28 je navrhnut podle programu rule110 a díky tomu dědí i jeho vlastnosti. Program se skládá z šesti instrukcí, přičemž žádná z instrukcí nevyvolává kaskádu reakcí, která by byla svojí dobou závislá na vstupních datech. Z tohoto důvodu je časová složitost tohoto programu konstantní. Nicméně s rostoucím počtem zpracovávaných dat u tohoto programu roste i počet nutných kopií instrukčních vláken. Tato závislost je lineární, protože se v programu nenachází žádná kaskáda reakcí, která by byla závislá na vstupu. Na doménové úrovni je tento program bezchybný a vrací vždy správný výsledek, ale jeho reálná implementace na nukleotidové úrovni již bezchybná být nemusí. Tento program má jako jediný z navržených programů i nukleotidovou reprezentaci domén, která vychází z reprezentace u programu 110. Tato reprezentace bude zkoumána dále v této kapitole.

5.2.2 Shift left

Program Shift left se skládá z třiceti tří instrukcí, přičemž žádná z instrukcí při většině počátečních konfigurací nevyvolává kaskádu reakcí, která by byla svojí dobou závislá na vstupních datech. Z tohoto důvodu je časová složitost tohoto programu pro většinu počátečních konfigurací konstantní. Nicméně s rostoucím počtem zpracovávaných dat u tohoto programu roste i počet nutných kopií instrukčních vláken. Tato závislost je lineární. Na doménové úrovni je tento program bezchybný a vrací vždy správný výsledek, ale jeho reálná implementace na nukleotidové úrovni již bezchybná být nemusí.

U programu Shift left existuje konfigurace, která tyto vlastnosti narušuje. V případě řady nul, která je v nejhorším případe dlouhá jako celý registr, dojde k řetězové reakci, která všechny tyto nuly postupně odstraní, což bude trvat N kroků za předpokladu, že registr má N bitů. Tato reakce zhorší časovou složitost na lineární. Nicméně množství potřebných instrukčních vláken se touto reakcí nezmění, protože jejich počet je stále lineárně závislí na délce registru.

5.2.3 Mul 2

Program Mul 2 se skládá z dvaceti sedmi instrukcí. Časová složitost i náročnost na počet instrukčních vláken je schodná s programem rule28, protože se jedná implementačně o stejný systém, který se pouze liší přechodovou funkcí a počtem stavů.

5.3 Evoluční návrh SIMD||DNA programů

Součásti této diplomové práce je i jednoduchý evoluční algoritmus, který slouží pro dokázání schopnosti evolučního algoritmu navrhnout program pro SIMD||DNA systém. Použitý evoluční algoritmus využívá pouze operátor mutace a turnajový výběr jedinců do nové populace, který je doplněn o elitizmus. Použitá fitness funkce počítá počet schodných vláken na očekávaném a současném řešení a vytváří penalizace za vlákna, která nejsou plně spojena s registrem. Chromozom je reprezentován dvěma částmi. První část reprezentuje reprezentaci jednotlivých stavů a druhá pak reprezentuje instrukce.

5.3.1 Funkce fintess

Fitness je počítáno tak, aby zohledňovalo počet správných koncových stavů buněk, stabilitu vlákna a případnou neurčitost stavů. Fitness se počítá jako počet všech buněk, které skončily ve správném stavu, násobený počtem všech domén, které jsou správně volné nebo zabrané, násobený koeficientem stability. Koeficient stability se počítá jako procento stabilních vláken v celém registru. V poslední části se zjišťuje, aby se fitness nastavilo na nulu, pokud se v chromozomu stavů nacházejí stavy, které jsou svojí reprezentací nerozeznatelné od sebe.

5.3.2 Reprezentace chromozomu stavů

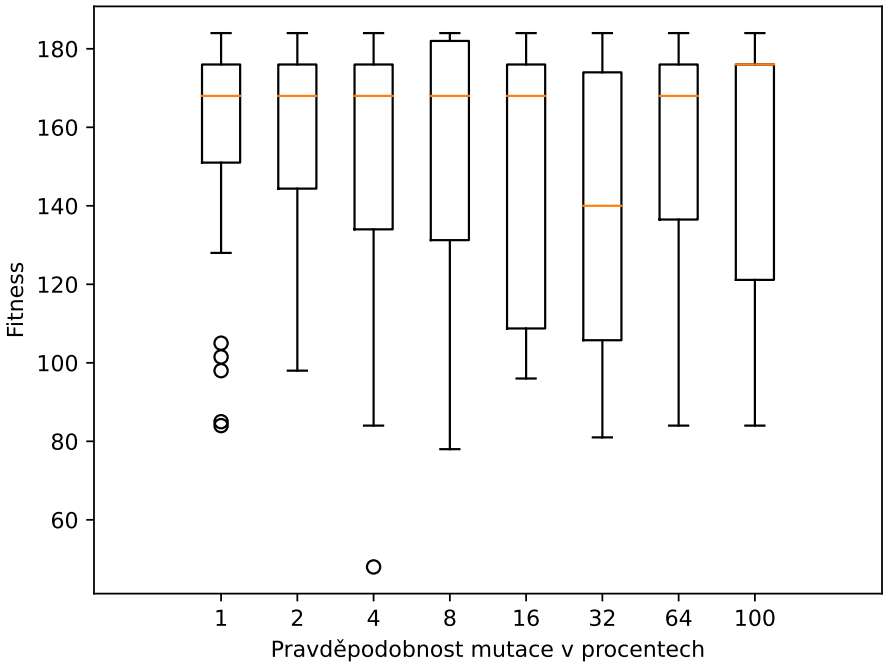
Reprezentace stavu je reprezentována polem hodnot typu int z množiny $\{0, 1, 2\}$, které je stejné délky jako je jedna buňka. Hodnoty typu int reprezentují přítomnost případně absenci vlákna. Hodnota 0 reprezentuje absenci vlákna nad doménou, 1 reprezentuje přítomnost vlákna nad doménou a 2 reprezentuje přítomnost vlákna nad doménou, přičemž vlákno nad touto doménou končí a dále nepokračuje.

5.3.3 Reprezentace chromozomu instrukcí

Instrukce jsou reprezentovány jako pětice hodnot typu int, které reprezentují obsah domén v instrukci vůči doménám v jedné buňce, která je staticky definována. Tato uspořádaná pětice je následující $(offset, len, end, complement, use)$. $offset$ definuje o kolik domén je vlákno posunuto oproti buňkám, len určuje délku vlákna, end slouží k definici domény, která není v buňkách a zároveň bude použita na konci vlákna, $complement$ je z množiny $\{0, 1\}$ a určuje, zda všechny domény ve vlákně budou komplementární nebo ne, use je z množiny $\{0, 1\}$ a určuje, zda bude vlákno ve fenotypu použito. Instrukce jsou v poli konstantní velikosti, která je definována staticky.

5.3.4 Vliv pravděpodobnosti mutace

Na obrázku 5.9 je závislost pravděpodobnosti mutace na hodnotě fitness po 20 generacích. Ostatní parametry experimentu byly, velikost populace = 8, drand = None, velikost turnaje = 2, počet mutovaných hodnot genu = 2.

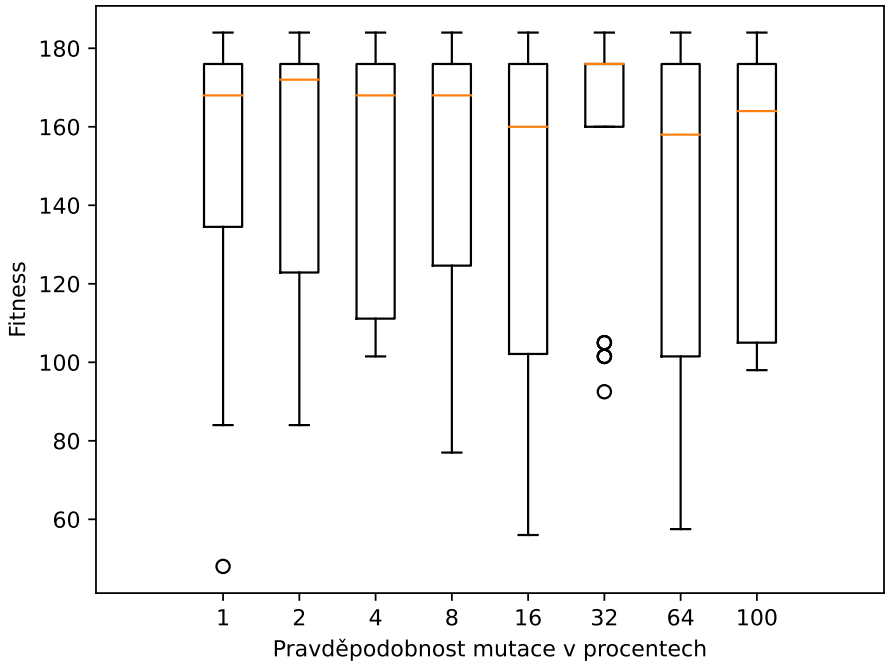


Obrázek 5.9: Hodnota fitness po 20 generacích. (Sestaveno z 30 běhů)

Z grafu plyne že nejlepší pravděpodobnosti mutace jsou v rozmezí 0.01 až 0.02. Dále lze pozorovat, že existuje několik běhů u každé pravděpodobnosti, které našli maximální lokální optimum, které je v tomto problému celkem lehce nalezitelné.

5.3.5 Vliv podpory diverzity reprezentace stavů

Na obrázku 5.10 je závislost pravděpodobnosti mutace na hodnotě fitness po 30 generacích při spuštěné podpoře diverzity reprezentace stavů. Ostatní parametry experimentu byly, velikost populace = 8, drand = None, velikost turnaje = 2, počet mutovaných hodnot genu = 2.

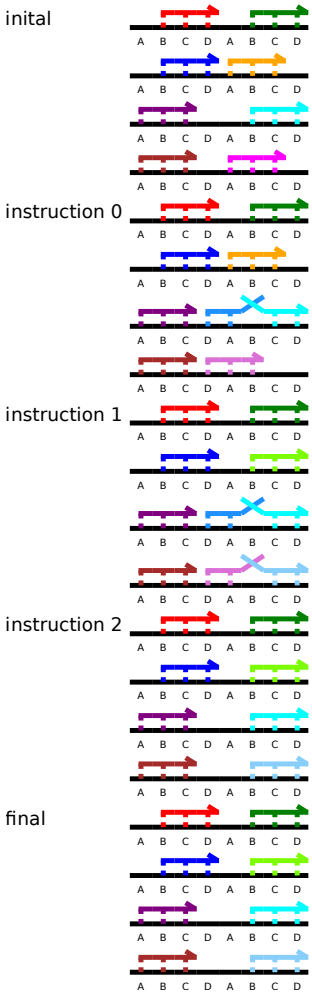


Obrázek 5.10: Hodnota fitness po 20 generacích s podporou diverzity. (Sestaveno z 30 běhů)

Graf na první pohled vypadá horší než graf 5.9. Nicméně tato implementace evolučního algoritmu po 300 generacích našla finální řešení dvakrát častěji než implementace na grafu 5.9. Důvodem, proč graf vypadá hůře, je vynucování diverzity, která zpomaluje evoluční vývoj směrem k nějakému optimu. Nicméně toto zpomalení vede k vetší diverzitě populace a menší pravděpodobnosti zaseknutí v lokálním optimu. Problém zaseknutí v lokálním optimu je způsoben především paralelním návrhem reprezentace stavu a instrukcí programu. Typicky zde dochází k problému zaseknutí na nějaké reprezentaci stavu, která neumožňuje problém plně vyřešit instrukcemi, a evoluce se zasekne na optimu této reprezentace.

5.3.6 Závěr

Experimenty provedenými v této diplomové práci bylo dokázáno, že je možné navrhnout celý SIMD||DNA program včetně reprezentace stavů. Dále se prokázalo, že podpora diverzity reprezentace stavů vede k zlepšení problému zasekávání v lokálním optimu. Na obrázku 5.11 je simulace navrženého programu, který provádí nastavení nuly do nultého bitu registru.



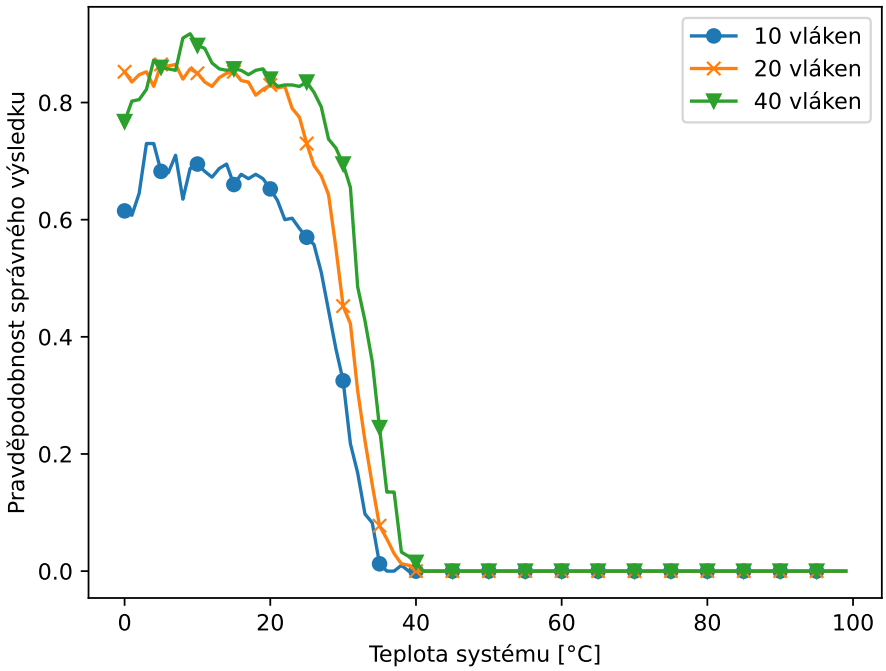
Obrázek 5.11: Běh evolučně navrženého programu

5.4 Zkoumání programů na nukleotidové úrovni

Všechny výsledky v této kapitole vznikly opakovaným spouštěním nukleotidové simulace a vyhodnocením jejích výsledků. Na jeden bod v grafu bylo použito 400 běhů simulace.

5.4.1 rule110

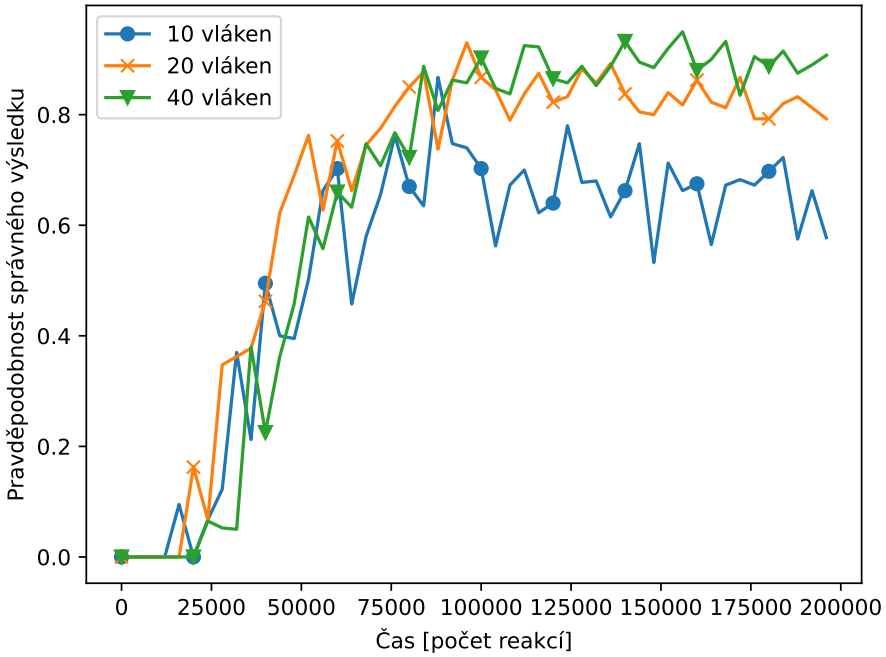
Pomocí simulátoru byl zkoumán program rule110 na nukleotidové úrovni. Byl zkoumán především vliv teploty a množství instrukčních vláken na úspěšnost dokončení výpočtu se správnou hodnotou. Na obrázku 5.12 je graf ukazující tuto závislost v tepelném rozsahu 0 °C až 100 °C a množství vláken z množiny $\{10, 20, 40\}$. Čas byl staticky nastaven na 99999 reakcí. Počáteční konfigurace byla nastavena na 1001111010.



Obrázek 5.12: Pravděpodobnost správného konce nukleotidové simulace rule110

Z grafu plyne, že program je funkční pouze v určitém tepelném rozsahu. To je způsobeno tím, že při příliš nízkých teplotách dochází k párování bází, které nejsou k sobě plně komplementární, a naopak při příliš vysokých teplotách dojde k denaturaci celého systému. U počtu vláken platí, že jich musí být dostatečně mnoho na to, aby vyžadované reakce byly dostatečně pravděpodobné a také, aby bylo dostatek vláken na jejich provedení.

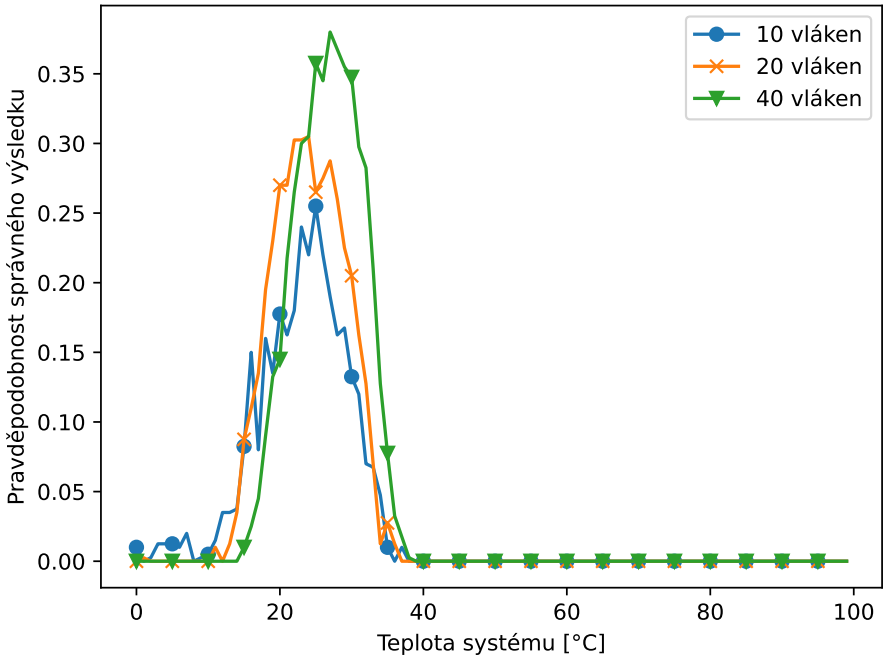
Dále byl zkoumám také vliv doby vystavení instrukčním vláknům. Předpoklad je takový, že pokud doba bude příliš krátká nedojde ke všem nutným reakcím a systém nebude funkční, pokud doba bude dostatečně dlouhá jejím prodlužování již nebude docházet k změně správnosti výstupu. Graf 5.13 ukazuje závislost času s počtem vláken na správnosti výstupu při konstantní teplotě 15°C a počáteční konfiguraci 1001111010. Data na grafu potvrzují tuto tezi.



Obrázek 5.13: Pravděpodobnost správného konce nukleotidové simulace rule110

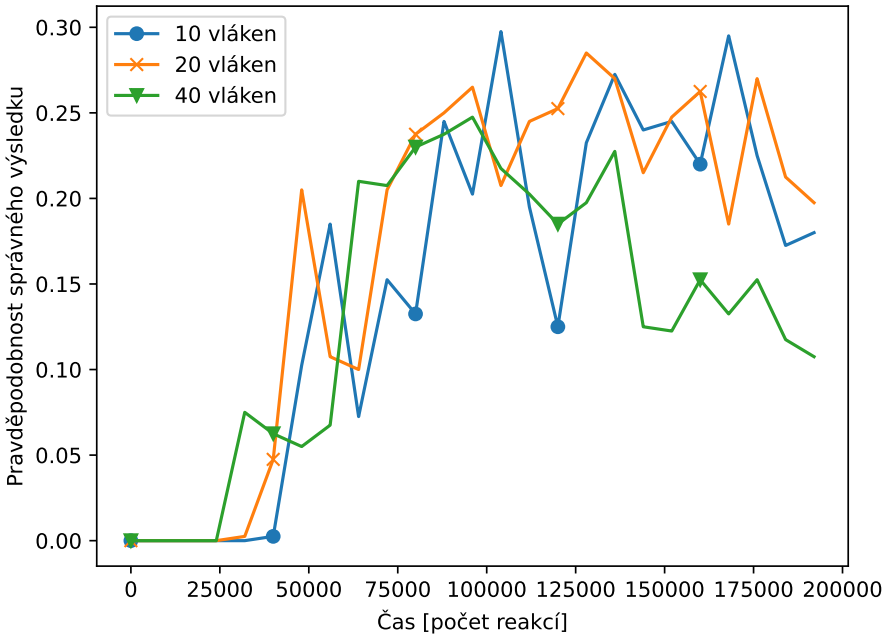
5.4.2 rule28

Stejně jako byl zkoumán program rule110, byl zkoumán i navržený program rule28. Byl tedy zkoumán vliv teploty a množství instrukčních vláken na úspěšnost dokončení výpočtu se správnou hodnotou. Na obrázku 5.14 je graf ukazující tuto závislost v tepelném rozsahu 0 °C až 100 °C a množství vláken z množiny $\{10, 20, 40\}$. Čas byl staticky nastaven na 99999 reakcí při počáteční konfiguraci 1001111010000. Bylo očekáváno, že výsledek bude velmi podobný programu rule110, protože se jedná o programy, které jsou postaveny na podobné hlavní myšlence. Tuto tezi data na grafu nepotvrzují. Rozdíl mezi grafy byl nejspíše způsoben otočením reprezentace logické jedničky a tím došlo k používání domén, které nejsou tak spolehlivé, jako jsou ty na druhé straně buňky. Tento problém by bylo možné vyřešit použitím vhodnějších reprezentací domén, pro které by platilo, že by na tomto programu vykazovaly větší teplotní odolnost než použité reprezentace z programu rule110.



Obrázek 5.14: Pravděpodobnost správného konce nukleotidové simulace rule28

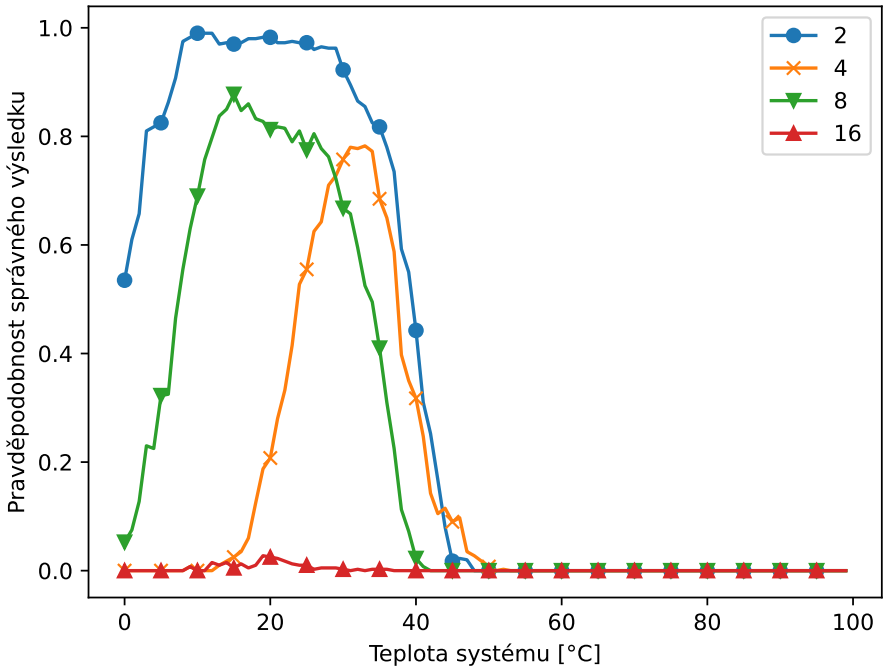
Dále byl zkoumám také vliv doby vystavení instrukčním vláknům. Předpoklad je stejný jako u programu rule110. Graf 5.15 ukazuje závislost času s počtem vláken na správnosti výstupu při konstantní teplotě 23°C a počáteční konfiguraci 100111101000.



Obrázek 5.15: Pravděpodobnost správného konce nukleotidové simulace rule28

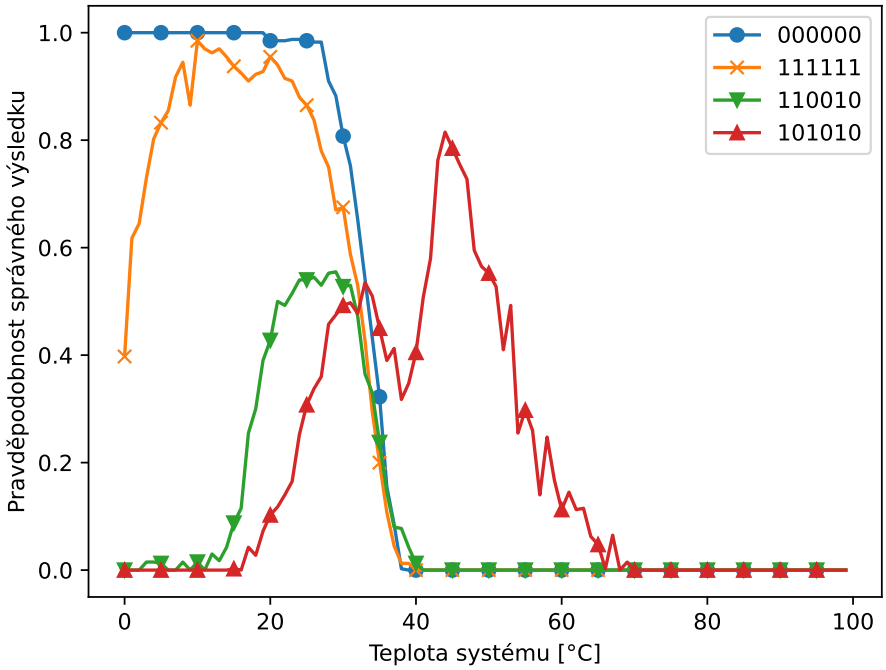
Z času nutného pro správné provedení instrukcí, je možné určit kolik zhruba reakcí instrukce vyžadují. Podle grafu 5.15 je tomu v případě rule28 tak, že instrukce vyžadují minimálně 20000 reakcí.

Zkoumán byl také vliv délky registru na úspěšnost výpočtu. Testy byly provedeny s počáteční konfigurací obsahující pouze stavy nula o délce z množiny {2,4,8,16,32}. Závislost byla zkoumána v tepelném rozsahu 0 °C až 100 °C při 100 vláknech a času 99999 reakcí. Graf 5.16 ukazuje výsledek tohoto experimentu. Data na grafu korespondují s očekávaným jevem, který způsobil snižování pravděpodobnosti úspěšného konce kvůli akumulaci chyb na delším registru. Na grafu lze také pozorovat anomálii v podobě podgrafu pro délku registru 16. Tento podgraf je mnohem nižší než bylo očekáváno. Vysvětlením této anomálie je, že 100 instrukčních vláken bylo pro tuto délku příliš málo a reakce již nebyly dostatečně pravděpodobné. Tato teorie byla následně ověřena simulací s větším počtem instrukčních vláken.



Obrázek 5.16: Pravděpodobnost správného konce nukleotidové simulace rule28 podle délky registrů

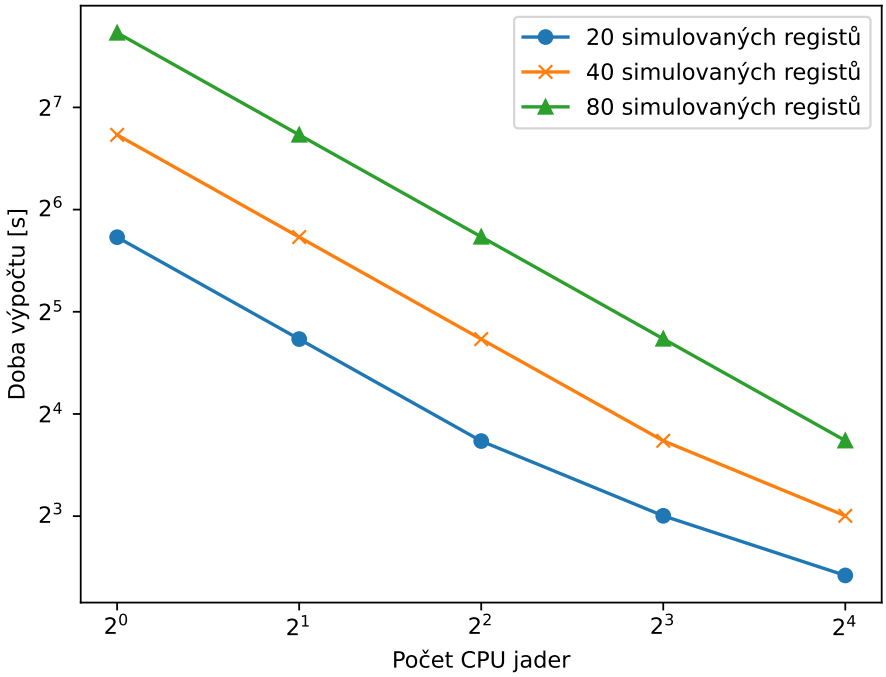
Poslední zkoumanou vlastností byl vliv počáteční konfigurace na správný výsledek výpočtu. Pro toto zkoumání byly zvoleny čtyři konfigurace (111111, 000000, 101010, 110010), které byly zkoumány v tepelném rozsahu 0 °C až 100 °C při 40 vláknech a času 99999 reakcí. Výsledky tohoto experimentu jsou v grafu 5.17. Z Grafu vyplývá, že se systém má tendenci chovat rozdílně k různým vstupům. Toto chování systému bylo očekávané, protože jednotlivé instrukce programu cílí na různé konfigurace, přičemž platí, že každá instrukce má odlišnou pravděpodobnost svého provedení. Například u konfigurace, ve které jsou samé nuly, stačí, aby instrukce nezměnily registr, a výstup bude správný. Ale například u konfigurace 110010 by toto již nestačilo. Graf dále ukazuje, že existuje teplota, pro kterou platí, že správné zpracování všech testovaných konfigurací bude podobně pravděpodobné.



Obrázek 5.17: Pravděpodobnost správného konce nukleotidové simulace rule28 podle počáteční konfigurace

5.5 Testy paralelizace

Paralelizace byla provedena na CPU pomocí openMP jak na úrovni vláken tak i na úrovni SIMD jednotek. Graf slabého škálování této implementace na procesoru 2X Intel(R) Xeon(R) CPU E5-2470 v2 @ 2.40GHz je na obrázku 5.18.



Obrázek 5.18: Slabé škálování navrženého simulátoru

Akcelerace nukleotidové simulace je možná i na GPU pomoci implementace v openACC. Tabulka zrychlení na GPU Tesla P4 oproti CPU implementaci je v tabulce 5.1. Akcelerace na GPU je na úrovni provádění jedné instrukce na jednom registru. Paralelizace na CPU je prováděna na úrovni registrů, na kterých se paralelně provádí jedna instrukce. Z tohoto důvodu má GPU sama o sobě horší výsledky než CPU, které mnohem lépe škáluje díky úrovni paralelizace. Nicméně akcelerace na GPU umožňuje akcelerovat výpočet jednoho dlouhého registru, což CPU paralelizace nezvládne, protože je implementována až o úroveň výše.

Počet registrů	CPU (Jedno jádro)	CPU (Dvacet jáder)	GPU (Tesla P4)
20	53s	2,8s	7,3s
40	106s	5,4s	14,5s
80	212s	10,7s	20,8s
160	424s	21,4s	57,1s
320	850s	42,6s	113.9s

Tabulka 5.1: Akcelerace na GPU Tesla P4

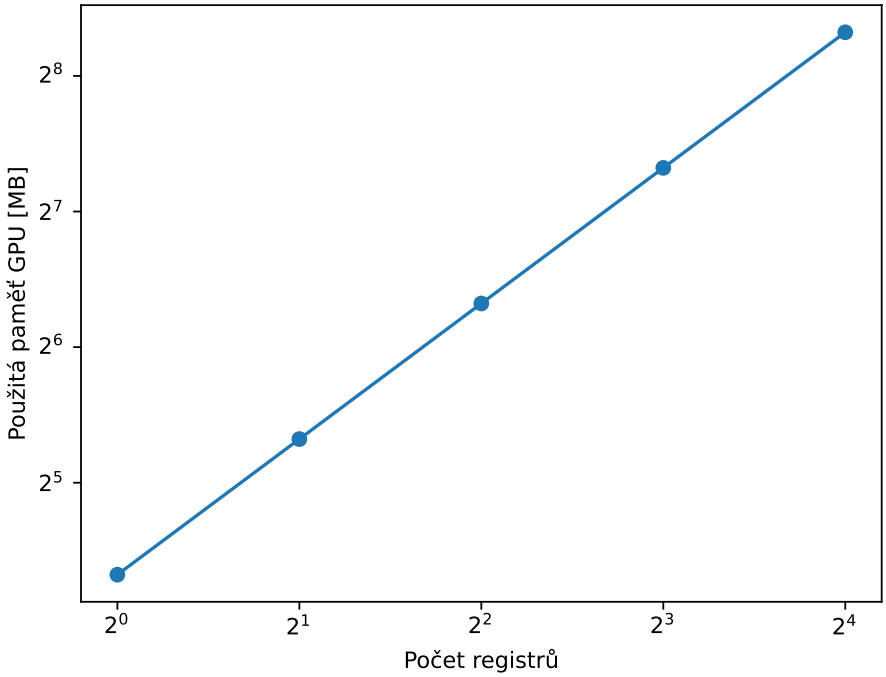
Je nutné dodat, že aplikace na GPU Tesla P4 byla mnohem energeticky efektivnější, protože grafická karta během výpočtu nespotřebovávala více než 23 W, zatímco dvě CPU spotřebovávala okolo 200 W. Tento efekt demonstruje tabulku 5.2, která ukazuje počet wattsekund na jeden registr.

Počet registrů	CPU (Jedno jádro)	CPU (Dvacet jáder)	GPU (Tesla P4)
20	212 Ws/reg	28 Ws/reg	8,39 Ws/reg
40	212 Ws/reg	27 Ws/reg	8,33 Ws/reg
80	212 Ws/reg	26,75 Ws/reg	5,98 Ws/reg
160	212 Ws/reg	26,75 Ws/reg	8,20 Ws/reg
320	212,5 Ws/reg	26,625 Ws/reg	8,18 Ws/reg

Tabulka 5.2: Energetická efektivnost akcelerace na GPU Tesla P4

5.6 Paměťová náročnost GPU implementace

Kvůli zjednodušení implementace GPU akcelerace jsou data uložena v kostce o konstantní velikosti, která je vhodně zvolena podle očekávané maximální délky registru. Souřadnice X a Y v kostce slouží k indexaci jednotlivých hodnot atributů a souřadnice Z slouží k indexaci typu atributu. Dimenze kostky jsou dále vhodně zvoleny tak, aby byly minimalizovány bankové konflikty. Kostka je složena z hodnot typu int a její velikost je určena maximálním počtem vláken v molekule, maximální délkou vlákna a počtem atributů. Počet atributů je dán implementací na pět, maximální délka vlákna je stanovena na tisíc stejně jako maximální počet vláken. Kostka reprezentující jeden registr je tedy o velikosti 1000x1000x5. Pro každý registr je nutné mít jednu kostku, takže požadavky na paměť lineárně rostou s počtem simulovaných registrů tak, jak ukazuje graf 5.19. Pro srovnání s CPU, kde je použita odlišná datová struktura, která dynamicky určuje velikost vektorů pro uložení hodnot atributů, podle skutečné velikosti vlákna, je tato implementace paměťově náročnější, nicméně je snáze na GPU implementovatelná.



Obrázek 5.19: Paměťová náročnost GPU akcelerace navrženého simulátoru

Kapitola 6

Závěr

V této práci byla navržena aplikace pro simulaci DNA výpočetní platformy SIMD||DNA. Konkrétně se práce zaměřila na simulaci programů celulárního automatu a binárního čítače. K vypracování této práce vedlo hned několik důvodů. Existuje jen velmi málo publikací, které se DNA počítáním zabývají z praktického hlediska v kombinaci s dnešními konvenčními výpočetními systémy. Do DNA počítání se vkládá velký příslib pro budoucí využití, naneštěstí se však musí vyřešit ještě mnoho překážek. SIMD||DNA je architektura, která má potenciál pro budoucí využití, a proto je vhodné se pokoušet o její zdokonalování. Z tohoto důvodu tato práce představuje simulátor, který na rozdíl od již existujících simulátorů umožňuje jednoduší zápis a přesnější simulaci, kterou je možné provádět až na úrovni nukleotidů nikoli na úrovni domén. Výsledky získané simulací výchozích problémů tedy celulárního automatu a binárního čítače ukázaly, že principy SIMD||DNA jsou navrženy a implementovány správně, a že se pomocí nich skutečně dají tyto problémy řešit. Pomocí simulátoru se také podařilo plně implementovat nové algoritmy (posuvný registr, třístavový celulární automat a celulární automat rule 28), které jsou uvedeny v části experimentálních výsledků. Dále byl úspěšně navrhnut program pomocí evolučního algoritmu. Pomocí simulátoru se podařilo ověřit funkčnost programu rule110 na nukleotidové úrovni a tato simulace byla úspěšně akcelerována pomocí GPU.

Simulátor sdcsim je dostupný na adrese https://lukas0025.github.io/sdc_sim/.

Další prací na toto téma by mohlo být rozšíření možností a experimentů s evolučním návrhem programů, dále by bylo možné zlepšit GPU simulaci zavedením streamů a překrytím výpočtu s datovým transferem. Také by bylo možné najít vhodné reprezentace domén pro navrhnuté programy a provést jejich simulaci na nukleotidové úrovni.

Literatura

[1] ŠMARDA Jan, Jiří DOŠKAŘ, Roman PANTŮČEK, Vladislava RŮŽIČKOVÁ a Jana KOPTÍKOVÁ. *Metody molekulární biologie*. 1. vyd. Brno: Masarykova univerzita, 2005. ISBN 80-210-3841-1.

[2] Aaron Ong. *SIMD-DNA: A python simulator for the SIMD||DNA model of computation*. 2024. Dostupné z: https://github.com/UC-Davis-molecular-computing/simd-dna.

[3] Adleman, L. M. Molecular Computation of Solutions to Combinatorial Problems. *Science*. 1994, sv. 266, č. 5187, s. 1021–1024. DOI: 10.1126/science.7973651. Dostupné z: https://www.science.org/doi/abs/10.1126/science.7973651.

[4] Adleman, L. M. On constructing a molecular computer. In: Lipton, R. J. a Baum, E. B., ed. *DNA Based Computers, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, April 4, 1995*. DIMACS/AMS, 1995, sv. 27, s. 1–21. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DOI: 10.1090/DIMACS/027/01. Dostupné z: https://doi.org/10.1090/dimacs/027/01.

[5] Ahmadian, A., Ehn, M. a Hober, S. Pyrosequencing: History, biochemistry and future. *Clinica Chimica Acta*. 2006, sv. 363, č. 1, s. 83–94. DOI: https://doi.org/10.1016/j.cccn.2005.04.038. ISSN 0009-8981. From Real-Time PCR to Nanotechnology: Rapid and/or High-Throughput Diagnostic Methods for Nucleic Acids. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0009898105004274.

[6] Beadle, G. W. a Tatum, E. L. Genetic Control of Biochemical Reactions in Neurospora. *Proc Natl Acad Sci U S A*. listopad 1941, sv. 27, č. 11, s. 499–506.

[7] Benenson, Y., Gil, B., Ben Dor, U., Adar, R. a Shapiro, E. An autonomous molecular computer for logical control of gene expression. *Nature*. May 2004, sv. 429, č. 6990, s. 423–429. DOI: 10.1038/nature02551. ISSN 1476-4687. Dostupné z: https://doi.org/10.1038/nature02551.

[8] Benenson, Y., Paz Elizur, T., Adar, R., Keinan, E., Livneh, Z. et al. Programmable and autonomous computing machine made of biomolecules. *Nature*. Nov 2001, sv. 414, č. 6862, s. 430–434. DOI: 10.1038/35106533. ISSN 1476-4687. Dostupné z: https://doi.org/10.1038/35106533.

[9] Boneh, D., Dunworth, C., Lipton, R. J. a Sgall, J. On the Computational Power of DNA. *Discret. Appl. Math.* 1996, sv. 71, 1-3, s. 79–94. DOI: 10.1016/S0166-218X(96)00058-3. Dostupné z: https://doi.org/10.1016/S0166-218X(96)00058-3.

[10] Braich, R. S., Chelyapov, N., Johnson, C., Rothemund, P. W. K. a Adleman, L. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*. březen 2002, sv. 296, č. 5567, s. 499–502.

[11] Collins, F. S. a Fink, L. The Human Genome Project. *Alcohol Health Res World*. 1995, sv. 19, č. 3, s. 190–195.

[12] Cook, M. A Concrete View of Rule 110 Computation. In: *CSP*. 2009. Dostupné z: https://api.semanticscholar.org/CorpusID:10266058.

[13] Das, R., Karanicolas, J. a Baker, D. Atomic accuracy in predicting and designing noncanonical RNA structure. *Nature Methods*. Apr 2010, sv. 7, č. 4, s. 291–294. DOI: 10.1038/nmeth.1433. ISSN 1548-7105. Dostupné z: https://doi.org/10.1038/nmeth.1433.

[14] Doty, D. a Ong, A. Simulating 3-Symbol Turing Machines with SIMD||DNA. In: Aspnes, J. a Michail, O., ed. *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, sv. 221, s. 14:1–14:15. LIPIcs. DOI: 10.4230/LIPICS.SAND.2022.14. Dostupné z: https://doi.org/10.4230/LIPIcs.SAND.2022.14.

[15] El Shaikh, A., Welzel, M., Heider, D. a Seeger, B. High-scale random access on DNA storage systems. *NAR Genom Bioinform*. leden 2022, sv. 4, č. 1, s. lqab126.

[16] Gheorghe Păun, A. S. *DNA Computing: New Computing Paradigms*. 1. vyd. Springer Berlin, Heidelberg, 1998. ISBN 978-3-540-64196-4.

[17] Gillespie, D. T. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*. American Chemical Society. Dec 1977, sv. 81, č. 25, s. 2340–2361. DOI: 10.1021/j100540a008. ISSN 0022-3654. Dostupné z: https://doi.org/10.1021/j100540a008.

[18] Hartmann, S. *The World as a Process: Simulations in the Natural and Social Sciences*. August 2005. Dostupné z: https://philsci-archive.pitt.edu/2412/.

[19] Ignatova, Z. a Zimmermann, K.-H. *DNA Computing Models*. Springer, prosinec 2008.

[20] Isel, C., Ehresmann, C. a Marquet, R. Initiation of HIV Reverse Transcription. *Viruses*. leden 2010, sv. 2, č. 1, s. 213–243.

[21] Kim, J. S., Lee, J.-W., Noh, Y.-K., Park, J.-Y., Lee, D.-Y. et al. An evolutionary Monte Carlo algorithm for predicting DNA hybridization. *Biosystems*. 2008, sv. 91, č. 1, s. 69–75. DOI: https://doi.org/10.1016/j.biosystems.2007.07.005. ISSN 0303-2647. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0303264707001220.

[22] Lakin, M. R., Youssef, S., Cardelli, L. a Phillips, A. Abstractions for DNA circuit design. *J R Soc Interface*. červenec 2011, sv. 9, č. 68, s. 470–486.

[23] Lakin, M. R., Youssef, S., Polo, F., Emmott, S. a Phillips, A. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics*. nov 2011, sv. 27, č. 22, s. 3211–3213. DOI: 10.1093/bioinformatics/btr543. ISSN 1367-4803. Dostupné z: https://academic.oup.com/bioinformatics/article/27/22/3211/195595.

[24] Lipton, R. J. DNA Solution of Hard Computational Problems. *Science*. 1995, sv. 268, č. 5210, s. 542–545. DOI: 10.1126/science.7725098. Dostupné z: https://www.science.org/doi/abs/10.1126/science.7725098.

[25] Marraffini, L. A. a Sontheimer, E. J. CRISPR interference: RNA-directed adaptive immunity in bacteria and archaea. *Nat Rev Genet*. březen 2010, sv. 11, č. 3, s. 181–190.

[26] Mendel, G. Versuche über Pflanzenhybriden (Pokusy s rostlinnými hybridy). *Verhandlungen des naturforschenden Vereines in Brünn (Sborník přírodovědeckého spolku v Brně)*. 1865, IV. svazek, s. 3–47.

[27] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. a Teller, E. Equation of state calculations by fast computing machines. *The journal of chemical physics*. American Institute of Physics. 1953, sv. 21, č. 6, s. 1087–1092.

[28] Miescher, F. Letter i; to wilhelm his; tübingen, february 26th, 1869. *Die histochemischen und physiologischen arbeiten von Friedrich Miescher-aus dem wissenschaftlichen Briefwechsel von F. Miescher*. FCW Vogel Leipzig, Germany. 1869, sv. 1, s. 33–38.

[29] Miescher, J. F. *Ueber die chemische Zusammensetzung der Eiterzellen*. 1871.

[30] Morgan, T. H. SEX LIMITED INHERITANCE IN DROSOPHILA. *Science*. červenec 1910, sv. 32, č. 812, s. 120–122.

[31] NEŠETŘIL J. *Teorie grafů*. Praha: SNTL - Nakladatelství technické literatury, 1979.

[32] Niedringhaus, T. P., Milanova, D., Kerby, M. B., Snyder, M. P. a Barron, A. E. Landscape of next-generation sequencing technologies. *Anal Chem*. květen 2011, sv. 83, č. 12, s. 4327–4341.

[33] Nirenberg, M. W. *Marshall W. Nirenberg Papers, 1937-2003 (bulk 1957-1997)*. 1937.

[34] Organick, L., Ang, S. D., Chen, Y.-J., Lopez, R., Yekhanin, S. et al. Random access in large-scale DNA data storage. *Nature Biotechnology*. Mar 2018, sv. 36, č. 3, s. 242–248. DOI: 10.1038/nbt.4079. ISSN 1546-1696. Dostupné z: https://doi.org/10.1038/nbt.4079.

[35] Parisien, M. a Major, F. The MC-Fold and MC-Sym pipeline infers RNA structure from sequence data. *Nature*. Mar 2008, sv. 452, č. 7183, s. 51–55. DOI: 10.1038/nature06684. ISSN 1476-4687. Dostupné z: https://doi.org/10.1038/nature06684.

[36] Pohl, G. a Shih, I.-M. Principle and applications of digital PCR. *Expert Review of Molecular Diagnostics*. Taylor & Francis. 2004, sv. 4, č. 1, s. 41–47. DOI: 10.1586/14737159.4.1.41. PMID: 14711348. Dostupné z: https://doi.org/10.1586/14737159.4.1.41.

[37] Ralston, A., Reilly, E. D. a Hemmendinger, D., ed. *Encyclopedia of Computer Science*. 4. vyd. Nature Publishing Group, červenec 2000.

[38] Redei, G. P. *Encyclopedia of Genetics, Genomics, Proteomics, and Informatics*. 3. vyd. Springer, duben 2008. Encyclopedia of Genetics, Genomics, Proteomics, and Informatics.

[39] Rheinberger, H.-J. Heredity and its entities around 1900. *Studies in History and Philosophy of Science Part A*. 2008, sv. 39, č. 3, s. 370–374. DOI: https://doi.org/10.1016/j.shpsa.2008.06.008. ISSN 0039-3681. Science and the Changing Senses of Reality CIRCA 1900. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0039368108000617.

[40] Rothemund, P. W. K., Papadakis, N. a Winfree, E. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biol*. prosinec 2004, sv. 2, č. 12, s. e424.

[41] Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N. V., Goodman, M. F. et al. A sticker-based model for DNA computation. *J Comput Biol*. 1998, sv. 5, č. 4, s. 615–629.

[42] Shah, S., Wee, J., Song, T., Ceze, L., Strauss, K. et al. Using Strand Displacing Polymerase To Program Chemical Reaction Networks. *Journal of the American Chemical Society*. 2020, sv. 142, č. 21, s. 9587–9593. DOI: 10.1021/jacs.0c02240. PMID: 32364723. Dostupné z: https://doi.org/10.1021/jacs.0c02240.

[43] Snustad Peter, D. a Simmons, M. J. *Genetika*. 2. vyd. Brno: Masarykova univerzita, 2017. ISBN 978-80-210-8613-5.

[44] Srinivas, N., Parkin, J., Seelig, G., Winfree, E. a Soloveichik, D. Enzyme-free nucleic acid dynamical systems. *Science*. 2017, sv. 358, č. 6369, s. eaal2052. DOI: 10.1126/science.aal2052. Dostupné z: https://www.science.org/doi/abs/10.1126/science.aal2052.

[45] Wang, B., Chalk, C. T. a Soloveichik, D. SIMD||DNA: Single Instruction, Multiple Data Computation with DNA Strand Displacement Cascades. In: Thachuk, C. a Liu, Y., ed. *DNA Computing and Molecular Programming - 25th International Conference, DNA 25, Seattle, WA, USA, August 5-9, 2019, Proceedings*. Springer, 2019, sv. 11648, s. 219–235. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-26807-7_12. Dostupné z: https://doi.org/10.1007/978-3-030-26807-7_12.

[46] WATSON, J. D. a CRICK, F. H. C. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature*. Apr 1953, sv. 171, č. 4356, s. 737–738. DOI: 10.1038/171737a0. ISSN 1476-4687. Dostupné z: https://doi.org/10.1038/171737a0.

[47] Weaver, R. F. *Molecular Biology*. 5. vyd. McGraw-Hill Professional, únor 2011.

[48] Wikipedia. *Base pair — Wikipedia, The Free Encyclopedia* [http://en.wikipedia.org/w/index.php?title=Base%20pair&oldid=1187784531]. 2024. [Online; accessed 28-April-2024].

Příloha A

SIMD||DNA programy

A.0.1 rule28

```
#
# RULE 28 cellular automaton implementation in DNA|SIMD
# @autor Lukas Plevac <xpleva07@vutbr.cz>
# @date 11.21.2023
#

domains:
    A~CACATAC
    B CTTTACA
    C TCTCCT
    D ACACACA
    E CAAAACT
    F TCAAATC
    G TCCACT

define:
    0 [AB][CDE]
    1 [ABCD]{E}

data:
    1001111010000

instructions: # O(6)
    {E*A*B*F*} # mark 10
    {D*E*A*B*C*G*} # mark 11
    {DEABCG} # remove mark 11
    {A*B*} {C*D*E*} # write 0
    {EABF} # remove mark 10
    {A*B*C*D*} # write 1
```

Obrázek A.1: Obsah SDCASM souboru, který obsahuje implementaci celulárního automatu rule28.

A.0.2 Selected NOT

```
#
# Selected NOT in DNA|SIMD
# @autor Lukas Plevac <xpleva07@vutbr.cz>
# @date 11.27.2023
#

define:
    # BIT NOT selector
    0 [ABC][DE][NSE].<U*>
    1 [ABCD]{E}[NSE].<U*>
    N0 [ABC][DE]{NSE}
    N1 [ABCD]{E}{NSE}

data:
    0 N0 1 N1

instructions: # O(12)
    {G*D*E*N*} # mark NOT 0 and NOT 1
    {ABCD} # remove unwraped 1
    {GDEN} # remove mark
    {C*D*E*N*} # mark write 0
    {CDEN} # remowe mark write 0 (is only posible when
           # is unvraped for second part of zero)
    {A*B*C*D*} # write 1
    {ABCD} # remove not writed 1 (is unwraped by mark write 0)
    {N*S*E*G*} # unwrap write 0 mark
    {CDEN} # remove write 0 mark
    {NSEG} # remove unwraper
    {A*B*C*} {D*E*} # write 0
    {N*S*E*U*} # lock all NOT selectors
```

Obrázek A.2: Obsah SDCASM souboru, který obsahuje implementaci SelectedNOT.

A.0.3 Shift left

```
#
# Shift left register in DNA|SIMD
# @autor Lukas Plevac <xpleva07@vutbr.cz>
# @date 11.28.2023
#

define:
    # BIT NOT selector
    0 [ABC][DE]{NSL}
    1 [ABCD]{ENSL}

data:
    # Implicit zero
    00110[ABC][DE][NSL]

instructions: # O(33)
    {E*N*S*L*A*} # unvrap DE from zero and bind here on tother site
    # onvrap first base
    {D*E*N*S*L*G*} # replace unvraper if on left site is 0
    {ENSLA} # remove all existing unvrapers free when left 1
    {E*N*S*L*A*B*} # unvrap 0 when is on right is 0 on free space
    # and fully binde here if is 1 right
    {ENSLAB} # unvrap unwraper if is 1 on right
    {E*N*S*L*I*} # bind not selector for 11 on free space
    # (only free when 1 right and 1 left) (prevent E binding to)
    {N*S*L*A*B*C*F*} # REMOVE {E*N*S*L*A*B*}
    {DENSLG} # REMOVE {D*E*N*S*L*G*}
    {NSLABCF} # REMOVE {N*S*L*A*B*C*F*}
    {A*B*C*} # write zero fisrt part back

    {E*N*S*L*J*} # try fit in free space between cells
    # (when 1 on right thare is no free base when 0 there is one free base)
    {D*E*N*S*L*G*} # replace {E*N*S*L*J*} when 0 on rigth
    {DENSLG} # REMOVE {D*E*N*S*L*G*} (now free only when have 0 on rigth)
    {D*E*N*S*L*A*B*} # unvrap 0 when is on left is 1 on
    # free space and fully binde here if is 0 right
    {DENSLAB} # unvrap {D*E*N*S*L*A*B*} when 1 on rigth

    {E*N*S*L*Y*} # pad NOT selector to prevent bind not selector here (01)
    {B*C*D*G*} {DENSLAB} # unvrap {D*E*N*S*L*A*B*} using {B*C*D*G*} and remove

    {ENSLI} # remove temp not selector for 11
    {N*S*L*U*} # bind not selector for 00

    {ENSLY} {ENSLJ} # remove PAD {E*N*S*L*Y*} and PAD {E*N*S*L*J*}
```

```
# {BCDG} remove unvraper {B*C*D*G*} not needed becasuse is replaced by 0

{A*B*C*} {D*E*} # write 0


# selected not subprogram

{G*D*E*N*} # mark NOT 0 and NOT 1
{ABCD} # remove unwraped 1
{GDEN} # remove mark
{C*D*E*N*} # mark write 0
{CDEN} # remowe mark write 0
# (is only posible when is unvraped for second part of zero)
{A*B*C*D*} # write 1
{ABCD} # remove not writed 1 (is unwraped by mark write 0)
{N*S*L*G*} # unwrap write 0 mark
{CDEN} # remove write 0 mark
{NSLG} # remove unwraper
{A*B*C*} {D*E*} # write 0
{NSLU} # remove all not selectors
```

Obrázek A.3: Obsah SDCASM souboru, který obsahuje implementaci Shift left.

A.0.4 Mul 2

```
#
# 3 state celular automaton in DNA|SIMD
# @autor Lukas Plevac <xpleva07@vutbr.cz>
# @date 21.04.2024
#

define:
    0 [AB][CD][EF]
    1 [ABC][DEF]
    2 {A}[BCDE]{F}

data: # here is all compute state to evaluate
    0001212

instructions:
    {F*A*J*} # mark 02, 22 and 20
    {F*A*B*G*} # mark 20 nothing to do here
    {C*D*E*H*} # mark 02
    {D*E*F*A*G*} # mark 12


    #
    # 02 to 11 part
    #

    {CDEH} # unmark 02

    {D*E*F*G*} # unbind 02 marker part 1
    {DEFG} # unbind 02 marker part 2

    {F*A*B*H*} # unvrap 2 in 02
    {BCDE} # unbind 2
    {FABH} # unbind unvraper

    {A*B*C*D*E*H*} # mark upgoing 1
```

```
#
# 12 to 21 part
#

{ABCDE} # when going 2 to 1 plce before other 2 mark it
{F*A*U*}

{DEFAG} # remove 12 mark

{B*C*D*E*} # write 2

{F*A*M*} # unvrap 0 when 02
{E*F*A*O*} # mark 0 in 02
{FAM} # unbind unvrap

{F*A*B*H*} # unvrap 2 in 02
{BCDE} # unbind 2
{FABH} # unbind unvraper

{ABCDEH} # unmark upgoing 1

{A*B*C*} # write 1 by small parts to avoid deplacement of 2
{D*E*F*}

{FAJ} # remove 22 marker

{EFAO} # remove mark 0 in 02
{E*F*} # write zero back
```

Obrázek A.4: Obsah SDCASM souboru, který obsahuje implementaci Mul 2

A.0.5 Evolučně nalezený program

```
define:
    0 {A}[BCD]
    1 [ABC]{D}

data:
    00
    01
    10
    11

instructions:
    {} {D*A*B*} # instruction 0
    {A*B*B*} {B*C*D*} # instruction 1
    {CDABE} {} # instruction 2
```

Obrázek A.5: Obsah SDCASM souboru, který obsahuje implementaci nulování nultého bitu v 2b registru.