UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Mattia Petroni

Gensko regulatorna omrežja v domeni podatkovno pretokovnih platform procesiranja

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Nikolaj Zimic
Somentor: prof. dr. Roman Jerala

Ljubljana, 2011

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01777/2011
Datum: 05.09.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATTIA PETRONI**

Naslov: **GENSKO REGULATORNA OMREŽJA V DOMENI PODATKOVNO PRETOKOVNIH PLATFORM PROCESIRANJA**

GENE REGULATORY NETWORKS AS DATAFLOW PROCESSING PLATFORMS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Do razvoja komercialnih podatkovno pretokovnih računalnikov zaradi tehnoloških problemov kljub številnim raziskavam v osemdesetih in devetdesetih letih prešnjega stoletja ni prišlo. Po drugi strani biološki sistemi, natančneje gensko regulatorna omrežja, že sami po sebi izražajo delovanje, ki je sorodo podatkovno pretokovnemu procesiranju. Kandidat naj v svojem delu analizira podobnosti med gensko regulatornimi omrežji in podatkovno pretokovnimi računalniki. Pri tem naj poda možnosti biološke realizacije osnovnih konstruktov podatkovno pretokovnih arhitektur in navede njihove prednosti in slabosti.

Mentor:

prof. dr. Nikolaj Zimic

Somentor:

prof. dr. Roman Jerala

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Mattia Petroni,

z vpisno številko 63050390,

sem avtor diplomskega dela z naslovom:

Gensko regulatorna omrežja v domeni podatkovno pretokovnih platform procesiranja

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  prof. dr. Nikolaja Zimica

  in somentorstvom

  prof. dr. Romana Jerale

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 23.09.2011 Podpis avtorja:

Zahvala

Na tem mestu bi se zahvalil mentorju prof. dr. Nikolaju Zimicu, somentorju prof. dr. Romanu Jerali in izr. prof. dr. Mihi Mrazu za pomoč pri pripravi in izdelavi tega dela. Za številne pripombe in komentarje ter za pomoč pri pripravi besedila se zahvaljujem asistentu Mihi Moškonu. Nejcu Tomšiču bi se iskreno zahvalil za pomoč pri lektoriranju. Za moralno in finančno podporo pri večletnem študiju na Univerzi v Ljubljani pa se toplo zahvaljujem družini.

- Mattia Petroni, Ljubljana, 2011.

alla nonna Wanda che mi guarda da lassù

Kazalo

Seznam uporabljenih kratic in simbolov

CME	-	chemical master equation (slov. kemijska glavna enačba)
DF	-	dataflow (glej PP)
DNK	-	deoksiribonukleinska kislina
FFT	-	fast Fourier transform (slov. hitra Fourier-jeva transformacija)
GFP	-	green fluorescent protein (slov. zeleni fluorescentni protein)
GPP	-	graf pretoka podatkov
GRO	-	gensko regulatorna omrežja
Id	-	Irvine dataflow language (slov. PP jezik Irvine)
PP	-	podatkovno pretokovni
SPR	-	surface plasmon resonance (slov. površinska plazmonska resonanca)
SSA	-	stohastični simulacijski algoritem
VAL	-	value-oriented algorithmic language (slov. podatkovno usmerjeni algoritmični jezik)
ZNF	-	zinc finger (slov. cinkov prst)

Povzetek

Razvoj čistih *podatkovno pretokovnih* (PP) arhitektur se je ustavil že na začetku devetdesetih let prejšnjega stoletja, saj so nepremostljivi problemi, prisotni v večini le-teh pripeljali do razvoja preveč kompleksnih rešitev. Te so povzročile občuten padec zmogljivosti tovrstnih računalniških sistemov v primerjavi z računalniki temelječimi na von Neumannovi arhitekturi. Posledica tega je bil zastoj v razvoju PP računalnikov. Realizacija nekaterih prototipov je bila tako omejena zgolj na laboratorijske projekte. V zadnjem desetletju so se računalniški inženirji v sodelovanju z biotehnologi posvetili raziskovanju možnosti uporabe bioloških gradnikov za procesiranje informacij. Pri tem so se še posebej osredotočili na *gensko regulatorna omrežja* (GRO). Procesiranje informacij na podlagi bioloških platform odpira vrsto zanimivih pogledov na PP procesiranje, saj nekatere lastnosti GRO omogočajo odpravo določenih problemov, ki so zaustavili razvoj PP arhitektur na začetku devetdestih let prejšnjega stoletja. Kljub temu, da imajo procesne lastnosti GRO paralelno naravo in kažejo zelo veliko analogijo s PP računalniškimi arhitekturami pa je pred samo implementacijo takega računalnika potrebno rešiti mnogo tehnoloških, predvsem biokemijskih problemov. V tem delu je predstavljen splošen pregled temeljnih modelov PP arhitektur in njihovih analogij. Poleg tega so predstavljene metode implementacije osnovnih konstruktov PP procesiranja z uporabo GRO.

Ključne besede:

gensko regulatorna omrežja, podatkovno pretokovno procesiranje, podatkovno pretokovna arhitektura, paralelno procesiranje, stohastično modeliranje.

Abstract

The research and study on pure *dataflow* (DF) computer architectures has almost reached a dead point since early '90, when several technological issues became insurmountable. Although, the pioneering work of J.B. Dennis at MIT in early seventies, showed to engineers all the potential of the fine grained parallelism, implicit in dataflow computing, applied to several prototype machines, still many issues inside architecture model remain partially unsolved. Because of increasing complexity which grew in solving these issues, the dataflow architecture became quickly economically disadvantageous in comparison to von Neumann architecture, so it remained alive only as laboratory project. With the recently works of many computer engineers and biotechnologists, which use biological systems, precisely *gene regulatory networks* (GRNs), to build logic circuits, a new point of view arises concerning this technology and dataflow computation. Several properties of GRNs allow to view the memory system and the information processing of dynamic dataflow architecture in entirely different way, so that some constraints may fall. Although these properties may improve solutions to the main dataflow architecture issues, several technological (biochemical) problems remain to solve. The work presents a general review of the most studied models of dataflow architectures and their analogies with GRNs. At the end, a few possible implementations of fundamental constructs of dataflow computing using GRNs are presented.

Key words:

gene regulatory networks, dataflow computing, dataflow architecture, parallel computing, stochastic modeling.

Poglavje 1

Uvod

Večina današnjih računalniških sistemov temelji na von Neumannovi arhitekturi, ki predpostavlja, da procesiranje informacij poteka z izvajanjem strojnih ukazov v določenem zaporedju, pri čemer je sekvenca teh ukazov prostorsko določena s *programskim števcem* (angl. *program counter*), časovno pa z urinim signalom [28]. Taki sinhronizirani obliki procesiranja pravimo tudi *ukazno pretokovno procesiranje* (angl. *instruction flow computing*), njegovi arhitekturi pa *ukazno pretokovna arhitektura* (angl. *instruction flow architecture*), kateri glavni predstavnik je prav von Neumannov računalnik. V *podatkovno pretokovnih* (PP) računalnikih izvajanje ukazov ne poteka na podlagi vsebine programskega števca, ampak je določeno s prisotnostjo oziroma odsotnostjo operandov (podatkov). Strogega zaporednega izvajanja ukazov v PP arhitekturah torej ni. Zaporedno izvajanje ukazov predstavlja ozko grlo hitrosti procesiranja in posledično eno glavnih slabosti von Neumannove arhitekture. Po drugi strani predstavljajo PP arhitekture posebno obliko asinhronskega procesiranja z implicitnim paralelnim delovanjem. Za njih je značilna ekstremna oblika *drobno-zrnatega paralelizma* (angl. *fine-grain parallelism*), ki jo običajni von Neumannovi računalniki ne morejo dosečiti. Prav ta ugotovitev je na začetku pripeljala do razvoja prvih PP arhitektur (prof. J.B. Dennis in sodelavci iz MIT).

Čeprav je bilo v zadnjih tridesetih letih, realiziranih več različnih prototipov PP računalnikov, imajo skoraj vsi zametke v zgodnjem Dennisovem delu [13]. Razlog za to je njihova enostavnost in modularnost. V poznih sedemdesetih letih sta se razvili dve glavni družini PP arhitektur, in sicer *statične* in *dinamične*. Statične arhitekture so se razvile direktno iz Dennisovega modela, toda zgolj kot laboratorijski projekt. Zaradi strukturne neučinkovitosti (glej

poglavje 2) ni bilo realiziranih nobenih komercialnih različic PP računalnikov s statično arhitekturo. Po drugi strani so bile dinamične arhitekture uporabljene v realizaciji številnih PP računalniških sistemov na začetku osemdesetih let. Čeprav je delo profesorja Dennisa in profesorja Arvinda [2, 3, 4, 12] pripeljalo do delujočega primera računalnika z dinamično PP arhitekturo ("Monsoon"), so bile zmogljivosti tega računalnika občutno slabše v primerjavi s klasičnim von Neumannovim strojem. Razlog za to so bile *cevovodne nevarnosti*[1] in *procesorska neizkoriščenost* (angl. *load imbalance*) [25].

PP procesiranje temelji na izvajanju operacij na *grafu pretoka podatkov* (GPP) [48]. To je usmerjeni graf, v katerega vstopajo vhodni signali. Procesiranje le-teh poteka vzdolž vozlišč in povezav, rezultati procesiranja pa konvergirajo proti izhodnim povezavam grafa. Vozlišča GPP predstavljajo aritmetično logične operacije, medtem, ko povezave predstavljajo tok operandov, ki jih operacije v vozliščih zahtevajo. Operacije v vozliščih se *aktivirajo* (angl. *firing*) natanko takrat, ko so vsi operandi, ki jih operacija zahteva prisotni na vhodnih povezavah vozlišča. Pri tem načinu procesiranja torej ni vgrajenega nobenega sinhronizacijskega mehanizma temveč vse poteka asinhronsko. Izvajanje operacij je krmiljeno na osnovi prisotnosti oziroma odsotnosti operandov v usmerjenih povezavah GPP. Zaradi asinhronskega *toka* podatkov skozi vozlišča GPP, se tako procesiranje imenuje podatkovno pretokovno.

V bioloških sistemih, bolj pravilno v celicah bioloških organizmov, se stanje posameznih biokemijskih sistemov oz. *metabolizem*, regulira na osnovi prisotnosti/odsotnosti določenih kompleksnih molekul (proteini, encimi, proteaze, kinaze, itd.), ki celici omogočajo preživetje. Osnovni gradnik biokemijskih sistemov in "stroj" za sintezo kompleksnih molekul je *deoksiribonukleinska kislina* (DNK) (angl. *deoxyribonucleic acid* - DNA). Ta vsebuje dolga zaporedja nukleotidov za tvorbo aminokislinskih zaporedij, ki sestavljajo ciljne proteine. Pogoji za sintezo določenega proteina so določeni s t.i. *transkripcijskimi faktorji* (angl. *transcription factors*), ki po potrebi omogočajo (angl. *promote*) ali onemogočajo (angl. *inhibit*) *transkripcijo* mRNK molekul, ki se v procesu *translacije* prepišejo v ciljne proteine. Vezava različnih transkripcijskih faktorjev na določena vezavna mesta DNK določajo trenutno stanje t.i. regulacijskega omrežja. *Gensko regulatorno omrežje* - GRO (angl. *gene regulatory network* - GRN) je množica DNK vezavnih mest, DNK vezavnih proteinov in genov, ki sestavljajo zaokroženo celoto biokemijskih procesov.

[1] vgrajene v arhitekturi in odgovorne za več kot 6% vseh mirovnih ciklov (angl. *idle cycles*)

Dinamika v takemu omrežju je določena z regulacijo genov opazovanih proteinov, t.j. z njihovim zaviranjem (represijo) in njihovo aktivacijo. Če v splošnem gledamo na proteine kot signale, ki prenašajo informacijo o potrebi po določenem kemijskem procesu znotraj celice, in na gensko ekspresijo oziroma represijo, kot na izvajanje regulacijskega procesa, lahko takoj opazimo podobnost s podatkovnim pretokovnim procesiranjem. Množica genov z DNK vezavnimi mesti predstavlja osnovne operacije na signalnemu toku proteinov, ki se lahko vežejo ali ne, medtem ko proteini predstavljajo neposreden tok podatkov (operandov). Na GRO lahko torej gledamo kot na poseben primer podatkovno vodenega procesa.

V diplomski nalogi bomo predstavili pregled sorodnih lastnosti PP procesiranja in procesiranja informacij na podlagi GRO. V drugem poglavju je predstavljen osnutek PP arhitektur, njihove poglavitne lastnosti, slabosti in problematike pri realizaciji ter njihov dosedanji razvoj. V tretjem poglavju so predstavljene osnove GRO, sintetičnih DNK vezavnih proteinov ter možnosti snovanja logičnih vezij na podlagi le-teh. V četrtem poglavju so predstavljane podobnosti GRO omrežij s PP procesiranjem ter predlog razvoja visoko-paralelne procesne platforme na osnovi GRO z njegovimi prednosti in slabosti. V petem poglavju je opisana implementacija nekaterih osnovnih komponent PP arhitektur z GRO z uporabo stohastičnih modelov, ki temeljijo na uporabi sintetičnih DNK vezavnih proteinov.

Poglavje 2

Podatkovno pretokovno procesiranje in arhitekture

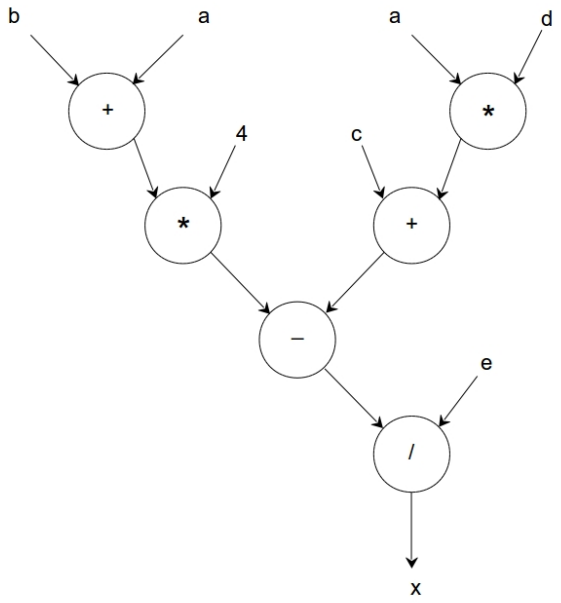
Podatkovno pretokovno ali podatkovno vodeno procesiranje temelji na načelih *funkcionalnosti* in *asinhronosti* [46]. Načelo funkcionalnosti trdi, da procesiranje oziroma računanje poteka modularno po sklopih oziroma z izvajanjem povezanih funkcij. Te se izvršijo samo takrat, ko so na voljo vsi potrebni podatki oziroma operandi - načelo asinhronosti. Opisani računski model predstavimo z *grafom pretoka podatkov* - GPP (angl. *dataflow graph*), ki ponazarja pretok operandov skozi množico operacij oz. funkcij. Istočasno graf implicitno prikazuje vse podatkovne odvisnosti med operacijami. V modelu grafa ni prisotna nobena oblika sinhronizacije pri izvajanju operacij, kot je skupni urin signal. Tej obliki izvajanja pravimo asinhrono izvajanje operacij.
Računsko reševanje problemov se prične z gradnjo GPP. Ta je tudi osnovni strojni jezik PP računalnikov in lahko vsebuje mnogo neodvisnih operacij, ki se lahko izvajajo vzporedno (več o strukturi grafa v razdelku 2.1). Računalniška arhitektura, ki implementira izvajanje GPP imenujemo *podatkovno pretokovna arhitektura* (PP arhitektura) in se bistveno razlikuje od običajne von Neumannove arhitekture računalniških sistemov.

2.1 Graf pretoka podatkov

Graf pretoka podatkov (GPP) je posebna oblika *Petrijeve mreže* (angl. *Petri net*) [39]. V tem grafu vozlišča predstavljajo operatorje, usmerjene povezave pa smer pretoka vrednosti operandov oziroma žetonov (angl. *data tokens*) skozi graf. Žetoni so nosilci operandov, ki lahko predstavljajo realne, celoštevilske, logične ali znakovne vrednosti. Kot primer si oglejmo kako enostavni aritmetični izraz predstavimo z GPP. Recimo, da želimo rešiti naslednjo enačbo:

$$x = ((a + b) * 4 - (c + a * d))/e. \tag{2.1}$$

Aritmetične izraze lahko zapišemo z binarnim drevesom, ki enolično predstavlja GPP:



Slika 2.1: Primer GPP za aritmetični izraz iz enačbe 2.1.

GPP v obliki drevesa izpostavlja dve zanimivi lastnosti iz vidika paralelnega procesiranja informacij. Posamezni nivoji drevesa predstavljajo neodvisne operacije, ki se lahko izvajajo vzporedno, saj le-te nimajo podatkovnih odvisnosti (na sliki 2.1 prvi nivo sestavljata dve sosedni operacij, $a + b$ in $a * d$[1], pri čemer

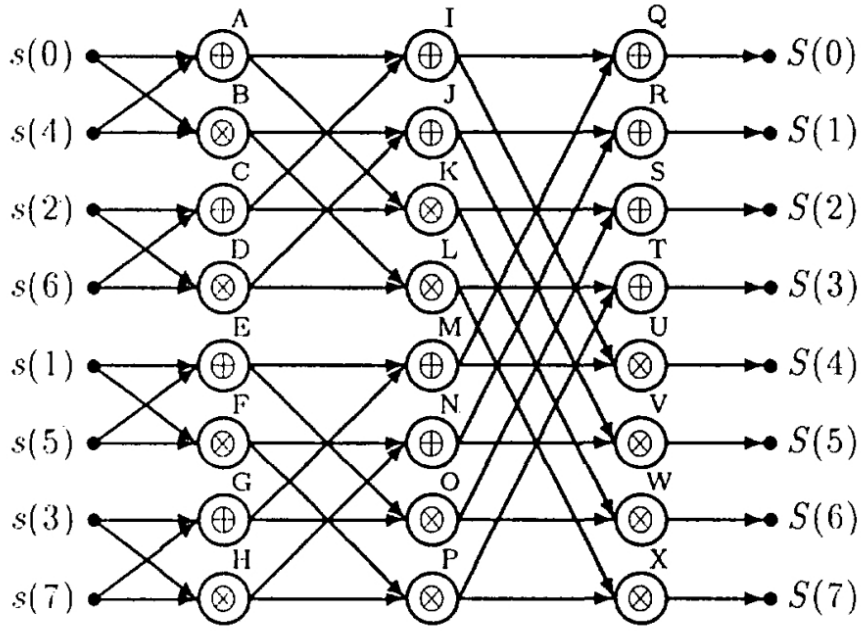
[1]žetona a in d, ali a in b pravimo *sosednja žetona*

se obe lahko izvajata istočasno). Taki paralelnosti rečemo *prostorska paralelnost*. Poleg tega lahko posamezne nivoje grafa izvajamo v cevovodni obliki, če imamo opravka z večkratno uporabo grafa. Taki paralelnosti rečemo *časovna paralelnost*. Podoben graf tvorijo prevajalniki visoko programskih jezikov pri procesu odkrivanja podatkovnih odvisnosti. Paralelnosti, ki jo implicitno predstavlja GPP, pravimo tudi drobno-zrnati paralelizem. Tak paralelizem je težko realizirati v von Neumannovi arhitekturi. V razdelku 2.2 bomo videli, kakšne so arhitekture, ki že implicitno podpirajo drobno-zrnati paralelizem.

Podatkovno vodeno procesiranje ne zahteva posebnih mehanizmov globalne sinhronizacije, kot je recimo urin signal v centralni procesni enoti von Neumannovega računalnika. Po drugi strani je potreben natančen mehanizem za krmiljenje podatkov skozi GPP. Podrobnosti takih mehanizmov lahko bralec dobi v [9, 29]. Operacije v vozliščih GPP se aktivirajo, ko na vhodne povezave prispejo vsi potrebni operandi. Takrat pravimo, da se vozlišče *vžge* (angl. *fire*). Operacija se izvede tako, da vzame operande iz vhodnih povezav (angl. *token fetch*), sprocesira funkcijo nad njimi in zapiše rezultat na izhodne povezave vozlišča (angl. *result token*). Opisani mehanizem skriva določene probleme, ki se pojavijo pri uporabi cevovoda nad GPP in pri nekaterih primerih cikličnih grafov. V razdelku 2.1.2 bodo na kratko predstavljeni posamezni mehanizmi, ki so implementirani v najbolj poznanih PP arhitekturah.

PP procesiranje je bilo na začetku sedemdesetih let prejšnjega stoletja zelo odmevno. Razlog za to je bila njegova računska zmogljivost, v implementaciji nekaterih osnovnih algoritmov iz področja digitalnega procesiranja signalov. Eden izmed najbolj eklatantnih primerov pri katerem so računalničarji videli ogromno korist v uporabi PP arhitektur je bil algoritem *hitre Fourier-jeve transformacije* (angl. *fast Fourier transform* - FFT). Slika 2.2 prikazuje graf algoritma FFT z osmimi točkami.

Vidimo, da lahko graf algoritma FFT skoraj neposredno pretvorimo v obliko GPP. Pogoj za to je implementacija kompleksnega seštevanja in kompleksnega množenja v posameznih vozliščih grafa. Poleg tega lahko izvajanje algoritma FFT izvedemo bodisi cevovodno pri računanju FFT dolgega signala, bodisi paralelno, pri čemer lahko procesiramo osem vozlišč naenkrat na vsakem nivoju grafa. Čeprav se do danes na trgu ni pojavil noben splošno namenski PP računalnik so bile v zmogljivih *digitalnih signalnih procesorjih* (angl. *digital signal processor* - DSP) uspešno implementirane mnoge posebnosti podatkovno vodenega procesiranja, na primer v družini C66x firme Texas Instruments [52].



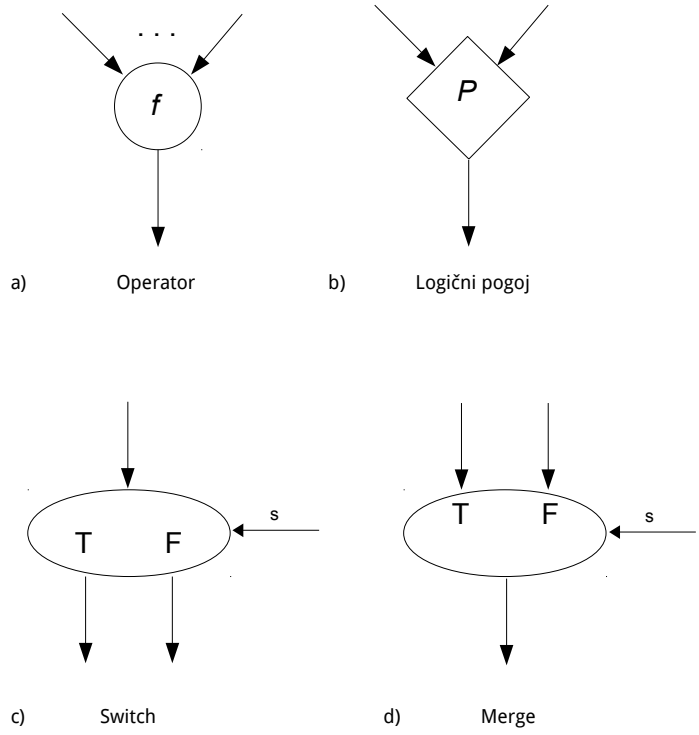
Slika 2.2: Primer dobro obnašajočega GPP za FFT z 8 točkami (slika je povzeta po [48]).

GPP-ji na sliki 2.1 in na sliki 2.2 so predstavljeni v poenostavljeni obliki. Za implementacijo bolj splošnih algoritmov so potrebni dodatni kontrolni mehanizmi in operatorji, ki jih je potrebno implementirati v GPP.

2.1.1 Osnovni konstrukti

Operatorji oz. vozlišča v grafu pretoka podatkov so lahko dveh vrst, in sicer *kontrolni* in *funkcionalni*. Kontrolne operatorje uporabljamo za krmiljenje pretoka žetonov skozi določene predele grafa GPP. Taki operatorji običajno vsebujejo vhodne povezave, ki nosijo logične kontrolne vrednosti (v programskih jezikih te vrednosti označujemo s tipom *boolean*). Funkcionalne operatorje potrebujemo za procesiranje aritmetično logičnih vrednosti žetonov v grafu GPP. Operator seštevanja je tipičen primer funkcionalnega operatorja. Operatorja *switch* in *merge* pa spadata med kontrolne operatorje (pravimo jim tudi *pogojni konstrukti*). Na množico operatorjev podatkovno pretokovne arhitekture lahko gledamo kot na nabor ukazov v von Neumannovem računalniku. Na sliki 2.3 so prikazani osnovni operatorji PP arhitektur: *funkcijski operator*,

logični operator (angl. *predicate*), *preklopni operator* (angl. *switch*) in *operator združevanja* (angl. *merge*).



Slika 2.3: Osnovni kontrolni in funkcionalni operatorji.

Slika 2.3 a) prikazuje primer funkcionalnega operatorja *f*, ki lahko predstavlja poljubno aritmetično logično operacijo (seštevanje, odštevanje, množenje, itd.) na dveh ali več operandih. Slika 2.3 b) prikazuje predikatni operator, ki se običajno uporablja za primerjanje logičnih ali številskih vrednosti. Predikatni (oz. logični) operator daje na izhodno povezavo žeton z vrednostjo *boolean* (angl. *true/false token*). Slika 2.3 d) prikazuje operator merge. Osnovno delovanje je naslednje: vhodni kontrolni signal oziroma žeton *s* določa kateri izmed dveh vhodnih žetonov se prepiše na izhodno povezavo. V primeru, da ima *s* vrednost "1" se na izhodno povezavo prepiše žeton, ki čaka na vhodni povezavi T. V nasprotnem primeru se na izhodno povezavo prepiše žeton, ki čaka na vhodni povezavi F. Podobno delovanje ima kontrolni operator switch (slika 2.3 c), toda v obrnjeni obliki. Kontrolni žeton določa na katero izhodno povezavo (T ali F), se bo vhodni žeton prepisal. Operatorja merge in switch

bosta predmet implementacije z gensko regulatornimi omrežji v poglavju 5.

Na podlagi opisanih operatorjev lahko definiramo pojem *dobro obnašajočih GPP* (angl. *well-behaved dataflow graph*) [3]. Dobro obnašajoči GPP je posebna vrsta PP grafa, pri katerem imajo vsa vozlišča grafa sledečo lastnost: vsak žeton na posamezni vhodni povezavi vozlišča povzroči dostavo natanko enega žetona na posamezno izhodno povezavo vozlišča. Vsi GPP grafi, ki vsebujejo samo funkcionalne operatorje, so dobro obnašajoči GPP (glej sliko 2.1). GPP, ki vsebuje najmanj en kontrolni operator ni dobro obnašajoči graf [6].

V PP arhitekturah imajo žetoni v GPP obliko paketov. Ti so sestavljeni iz množice informacij kot so na primer operacijske kode ukazov, operandov in njihovih naslovov v glavnem pomnilniku, naslovov izhodnih rezultatov in *bitov prisotnosti* (angl. *presence bits*). Oblika in sestava paketov sta neposreden izraz PP arhitekture. Primer paketa lahko vidimo na sliki 2.7.

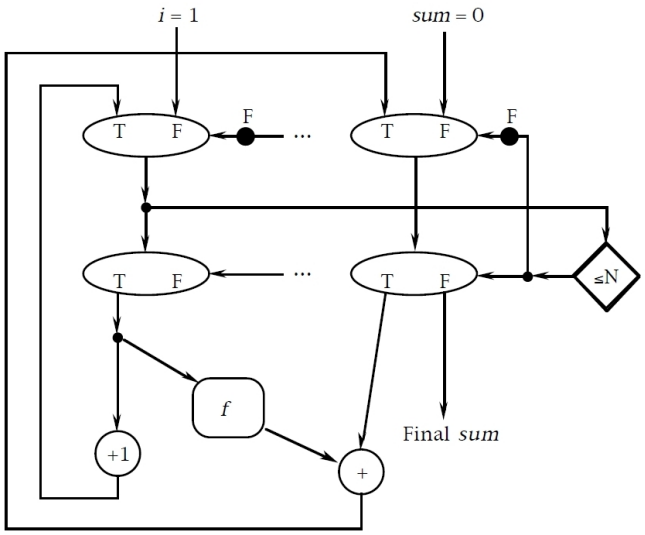
2.1.2 Reševanje problema povratnih povezav

V grafu pretoka podatkov lahko implementiramo zanke z operatorjema switch in merge, ter z uporabo povratnih povezav. Povratne povezave spremenijo aciklični graf v ciklični. Primer cikličnega grafa lahko vidimo na sliki 2.4, ki poleg tega vsebuje tudi varianto strukture operatorja merge, t.j. merge operator z negiranim kontrolnim signalom s. Graf na sliki 2.4 implementira enačbo 2.2. Zanimivost na tem grafu je operator *f*. Ta je označen z obliko podgrafa (nestandardni operator), t.j. simbol, ki nam pove, da se v tem vozlišču izvrši kompleksna operacija, ki zahteva razširitev v podgrafu.

$$\sum_{i=1}^{n} f(i) \tag{2.2}$$

Podrobnejša analiza pokaže, da graf na sliki 2.4 vsebuje celo vrsto skritih problemov. Prvič: kako se ciklični graf obnaša v primeru, ko je čas, ki je potreben za izvajanje določene operacije na nekem vozlišču, daljši od tistega na nekem sosednem vozlišču? Predpostavimo, da se operator f na sliki 2.4 izvaja z daljšim časom kot operator "*+1*"? Drugič, ali se števec prehodov zanke i lahko poveča do največje možne vrednosti preden se lahko vsota funkcij *f(i)* izračuna?

Na sliki 2.4 lahko opazimo, da se števec prehodov zanke i poveča predčasno



Slika 2.4: Primer zanke za izračun vsote funkcij iz enačbe 2.2.

v primeru, da je računanje vrednosti *f(i)* počasnejše glede na inkremetalno operacijo *+1*. Posledica tega je začetek novega prehoda zanke še preden se izračuna vsota *sum = sum + f(i)*. To lahko povzroči predčasni izhod zanke z napačnim rezultatom. Temu problemu pravimo *problem predčasnega izhoda zanke*. Tekom razvoja PP arhitektur se je razvilo pet glavnih modelov računanja z GPP, ki so ga v večji ali manjši meri učinkovito rešili [47, 48].

Model 1. Prvi dopolnjeni model računanja z GPP zahteva implementacijo mehanizma, ki prepreči začetek novega prehoda zanke pred končanjem prejšnjega prehoda. Ta model so inženirji takoj opustili zaradi slabega izkoristka oziroma zaradi skoraj odsotne paralelnosti [3, 48].

Model 2. V drugem modelu GPP je dovoljena prisotnost samo enega žetona na posameznih povezavah GPP. Poleg tega se zahteva spremembo mehanizma aktiviranja vozlišč (angl. *firing rule*): vozlišče se lahko sproži oziroma vžge samo takrat, ko na izhodnih povezavah ni žetonov. V statičnih arhitekturah v poznih sedemdesetih letih se je ta mehanizem implementiral s potrditvenimi signali (angl. *acknowledgment signal*). Glavna slabost te rešitve je bila podvojitev povezav v GPP, saj potrditveni signali potujejo po posebej namenjenih povezavah.

Model 3. Tretji model implementira transformacijo cikličnih grafov v aciklične. Temu procesu pravimo tudi *razvoj zanke*. Njegovi glavni slabosti sta potreba po velikemu glavnemu pomnilniku in po sprotnemu dinamičnemu razvoju grafa (t.j. generiranje vozlišč in povezav) v primeru, da je potreben predčasni izstop iz zanke.

Model 4. Četrti model zahteva dopolnjevanje žetonov s trenutno vrednostjo števca obhodov zanke. Ta skupaj s še nekaterimi dodatnimi informacijami o vozlišču kot je recimo naslov operacije, kateri je žeton namenjen, tvorijo enolično *oznako* žetona[2]. Posledično je mehanizem aktiviranja vozlišč spremenjen: operacija na vozlišču se izvede natanko takrat, ko imajo žetoni na vhodnih povezavah enako oznako. Označevanje žetonov (angl. *token tagging*) omogoča učinkovito implementacijo rešitve *problema prekrivanja indeksov* [48]. Izkoristek paralelizma je mnogo večji v primerjavi s prejšnjimi modeli. Glavna slabost je, da ta proces zahteva večji pretok podatkov skozi GPP in torej nekaj dodatnih mehanizmov za njihovo krmiljenje [48].

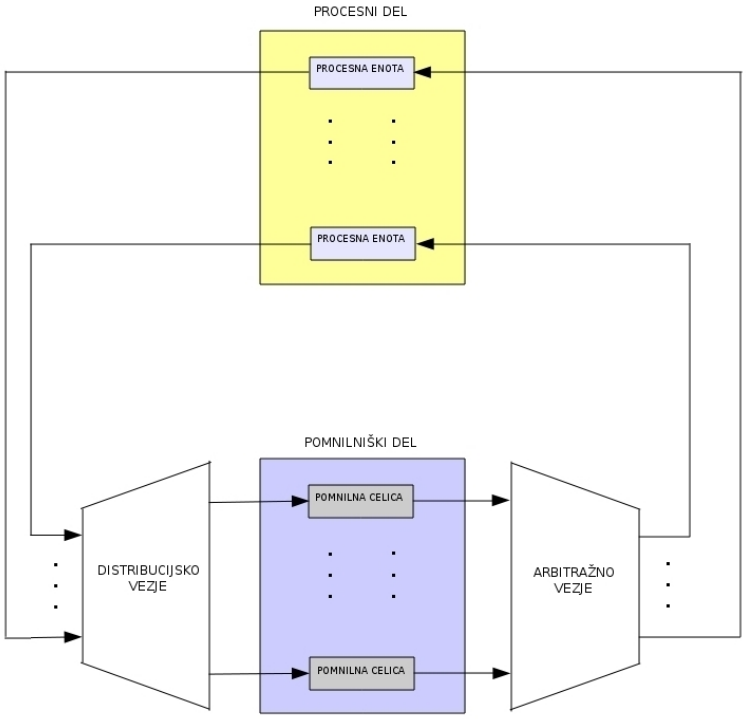
Model 5. V tem primeru je GPP spremenjen tako, da vse povezave grafa implementirajo čakalno vrsto (angl. *FIFO queue*). Žetoni se porabijo v istem vrstnem redu, kot so prispeli v čakalni vrsti. Ta model zahteva implementacijo čakalnih vrst na vsaki povezavi grafa kar je lahko zelo težko izvedljivo, vendar je pri tem izkoristek paralelizma enako dober kot pri četrtem modelu.

2.2 Podatkovno pretokovne arhitekture

Graf pretoka podatkov je strojni jezik (angl. *machine language*) vseh podatkovno pretokovnih arhitektur. GPP običajno predstavimo s seznamom ali s poljem paketov (to so nosilci operandov in ukazov). Vsak paket vsebuje operacijsko kodo ukaza, *zastavice prisotnosti* posameznih operandov (angl. *presence bit flag*), operande in naslov (oziroma naslove), na katerega se shranijo rezultati operacije. Medtem ko v dinamičnih arhitekturah paketi vsebujejo še sestavljeno oznako, v statičnih arhitekturah paketi vsebujejo dodatno polje, v katerega se zapišejo prevzeti potrditveni signali.

[2]angl. *labeled* ali *tagged token*. Arhitekture, ki so se razvile po tem modelu GPP so dinamične in se imenujejo *tagged token dataflow architectures*.

Če za trenutek ponovno preletimo splošni mehanizem delovanja vozlišč[3], lahko opazimo, da ta mehanizem določa naslednji osnovni ukazni cikel: (1) detekcija aktivnega vozlišča v GPP (oz. zbiranje operandov), (2) jemanje ukaza iz vozlišča (angl. *instruction fetch*), (3) izvajanje ukaza (angl. *instruction execute*) in (4) ustvarjanje izhodnih žetonov. Ta ukazni cikel je skupen vsem PP računalnikom, vendar se posamezne arhitekture lahko močno razlikujejo v načinu izvajanja [3].



Slika 2.5: Osnovna zgradba podatkovno pretokovne arhitekture. Shema je povzeta po [13].

Podatkovno pretokovno arhitekturo lahko predstavimo z abstraktnim modelom, ki ga sestavljajo *procesni del, pomnilni del, distribucijsko vezje* in *arbitražno vezje* (glej sliko 2.5). Pri tem je lahko procesni del sestavljen iz poljubnega števila procesnih enot, pomnilni pa iz poljubnega števila pomnilnih

[3]Operacija znotraj vozlišča se sproži natanko tedaj, ko so operandi na voljo v vseh vhodnih povezavah. Med izvajanjem se operandi porabijo. Po končani operaciji se na izhodne povezave zapiše rezultat.

celic. Pomnilne celice hranijo pakete, medtem ko procesne enote vsebujejo aritmetično logične enote in logiko za dekodiranje paketov potrebno za izvajanje operacij v vozliščih GPP (druga in tretja stopnja osnovnega ukaznega cikla). Pomembni komponenti sta še *distribucijsko* in *arbitražno vezje* (angl. *distribution* and *arbitration network*). Distribucijsko vezje ugotovi kateri paketi v pomnilnih celicah čakajo na operande. Ob prihodu rezultatov oziroma operandov iz procesnih enot vezje dodeli te operande ustreznim (čakajočim) paketom v pomnilniku (četrta stopnja osnovnega ukaznega cikla). Arbitražno vezje ugotovi kateri paketi oziroma katera vozlišča so ugodna za izvršitev (kateri paketi v pomnilnih celicah vsebujejo vse operande), nato jih prevzame iz pomnilnih celic in jih dodeli procesnim enotam za izvršitev (prva stopnja osnovnega ukaznega cikla) [13, 46].

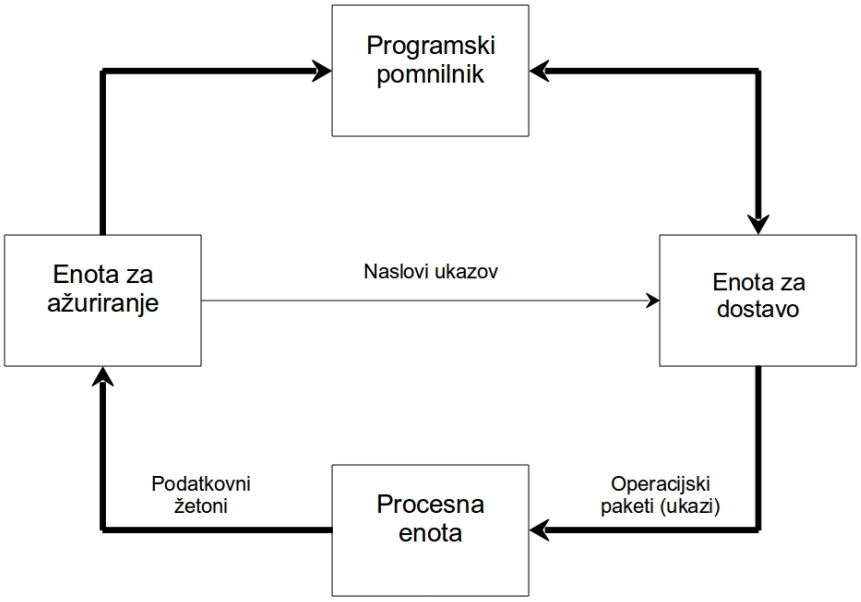
V abstraktnem modelu imamo opravka z velikim omrežjem povezav (angl. *packet communication system*) med distribucijskim vezjem, procesnim delom, arbitražnim vezjem in pomnilniškim delom [13]. V idealnem modelu PP arhitektur omenjeno omrežje sicer ne predstavlja problemov pri procesiranju. Med razvojem različnih PP arhitektur pa so odkrili, da velika omrežja povzročijo veliko zakasnitev zaradi bolj kompleksnega krmiljenja procesnih enot in bolj kompleksnega upravljanja s pomnilnimi celicami. Latenca pretoka podatkov skozi taka omrežja je v nekaterih primerih lahko tako velika, da postane paralelizem neuporaben [12, 29].

V nadaljevanju sta predstavljeni dve družini PP arhitektur, *statične* in *dinamične* (glej razdelek 2.2.1 in 2.2.2). Obe arhitekturi izhajata iz osnovnega modela na sliki 2.5. Statične arhitekture so se razvile kot posledica razvoja modela št. 2 iz razdelka 2.1.2, dinamične pa na osnovi ideje dinamičnega dodeljevanja pomnilnika paketom in na osnovi modela št. 4 iz razdelka 2.1.2 [3].

2.2.1 Statične arhitekture

Ta družina arhitektur izhaja neposredno iz Dennisovega predloga [13, 15]. Dennisov predlog predstavlja model sestavljen iz štirih glavnih enot: *programskega pomnilnika* (angl. *program memory*), *enote za dostavo* (angl. *fetch unit*), *procesne enote* (angl. *processing unit*) in *enote za ažuriranje* (angl. *update unit*). Organizacijo teh enot prikazuje slika 2.6.

Programski pomnilnik vsebuje *ukazne pakete* (angl. *instruction templates*), ki

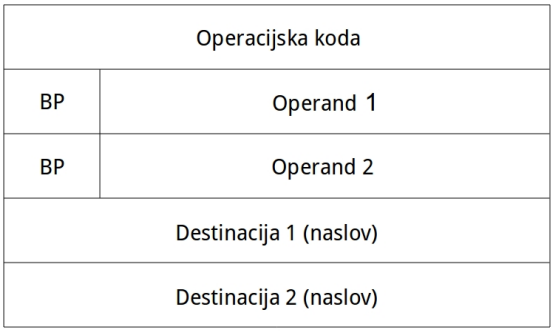


Slika 2.6: Organizacija osnovnega modela statične PP arhitekture. Slika je povzeta po [48].

predstavljajo vozlišča GPP. Vsak ukazni paket vsebuje operacijsko kodo, polja za operande in naslov destinacije rezultatov (slika 2.7). Operandi so na voljo in torej se lahko vozlišče izvrši takrat, ko so zastavice prisotnosti postavljene. Enota za ažuriranje je zadolžena za detekcijo aktivnih paketov[4] v programskem pomnilniku. Ko je ta pogoj izpolnjen, enota pošlje naslov ukaza (oz. paketa) enoti za dostavo. Ta enota jemlje pakete na prevzetih naslovih iz programskega pomnilnika, briše postavljene zastavice prisotnosti posameznih paketov in dodeljuje pakete procesni enoti, ki jim rečemo tudi *operacijski paketi*. Procesna enota izvrši operacijo, ki jo zahteva operacijska koda znotraj paketa in sestavi izhodne pakete. Enota za ažuriranje jemlje izhodne pakete in posamezne rezultate zapiše na naslove, ki jih določa posamezni paket (slika 2.7). Na koncu postavi še ustrezne zastavice prisotnosti v polju operandov.

Vidimo, da ima opisani model statične arhitekture enostaven mehanizem za detekcijo aktivnih vozlišč v grafu t.j. z bitom prisotnosti (BP). Tekom razvoja tega modela so se pojavile težave pri uporabi poljubnih cikličnih GPP, zato je ta model omogočal izvajanje samo dobro obnašajočih grafov. Tem restrikcijam je sledila rešitev, ki je zahtevala implementacijo mehanizma pri katerem je

[4]To so paketi, ki vsebujejo vse potrebne operande za izvrševanje operacije. To so tisti paketi, ki imajo postavljene vse zastavice prisotnosti.



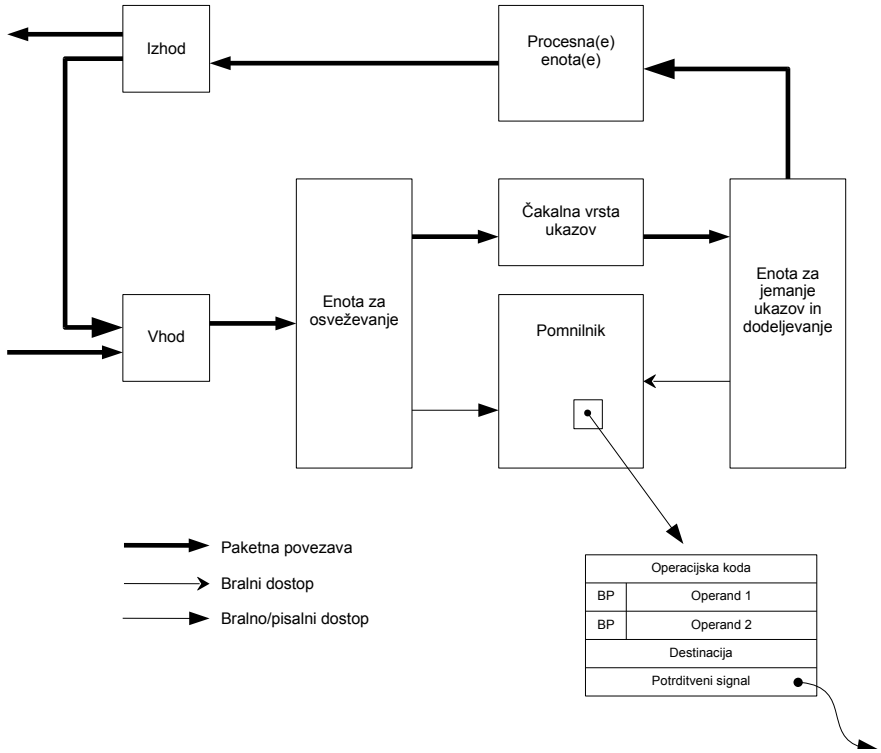
Slika 2.7: Osnovna zgradba paketa statične PP arhitekture [48]

graf omogočal prisotnost samo enega žetona v posamezni povezavi (angl. *one-token-per-arc restriction*) [3]. Nadaljnje izboljšave so izkoristile načelo uporabe potrditvenih signalov (drugi model GPP v razdelku 2.1.2). Cena, ki so jo za to plačali razvijalci je bila v skoraj podvojenem številu povezav v GPP, saj potrditveni signali potujejo po posebnih povezavah. Dennis in sodelavci [3] so prišli do razvoj modela arhitekture, ki je predstavljen na sliki 2.8. Ta vsebuje vse potrebne izboljšave za uporabo cikličnih grafov. Kljub temu se lahko nekateri subtilni problemi še vedno pojavijo [3]).

Če si ogledamo podrobnosti modela iz slike 2.8, vidimo, da se paketi nahajajo v pomnilniku, naslovi aktivnih paketov pa v čakalni vrsti ukazov. Enota za jemanje ukazov in dodeljevanje odstrani prvi element čakalne vrste, z njim dostopa do pomnilnika (angl. *activity store*) in jemlje ustrezni pripadajoči paket. Zatem dekodira vsebino paketa, briše bite prisotnosti operandov in sestavi operacijski paket, ki je namenjen procesni enoti. Operacijski paket vsebuje dekodirano operacijsko kodo, operande in seznam naslovov v glavnem pomnilniku, na katere je potrebno zapisati rezultat operacije.

Procesna enota po prevzemu operacijskega paketa izvrši ustrezno operacijo in zapiše rezultate v izhodni paket. Izhodne pakete prebere enota za osveževanje, ki je zadolžena za naslavljanje in zapisovanje rezultatov na posamezne lokacije v glavnem pomnilniku. Te lokacije določa seznam naslovov v izhodnem paketu procesne enote. Enota za osveževanje zapiše potrditveni signal in postavi ustrezne bite prisotnosti v polja operandov paketov, ki so zahtevali rezultat ravno izvršene operacije. Nato pregleda ali so v pomnilniku prisotni aktivni paketi[5]

[5]Aktivni paketi oziroma aktivna vozlišča grafa so v tem modelu tisti paketi, ki poleg vseh potrebnih operandov vsebujejo tudi potrditveni signal. To zagotavlja, da v izhodnih



Slika 2.8: Notranja organizacija statične podatkovno pretokovne arhitekture. Slika je povzeta po [3].

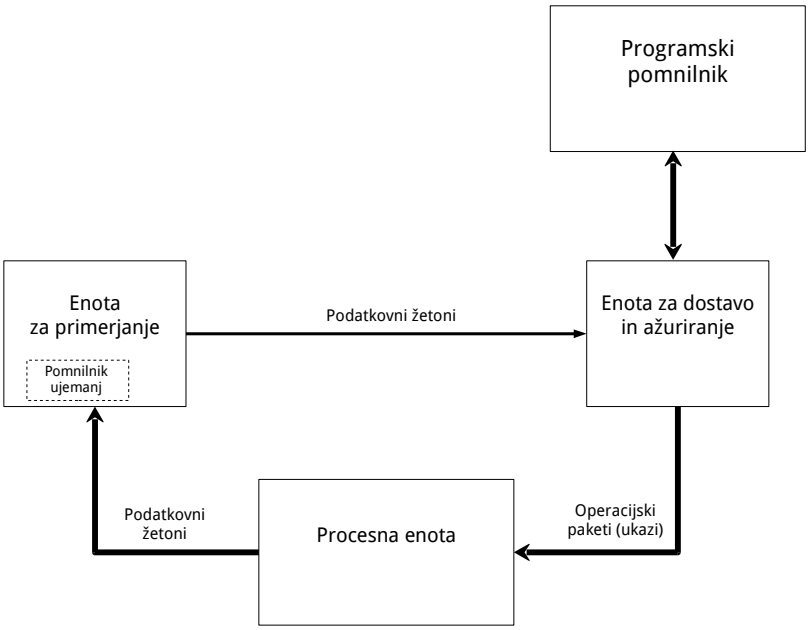
ter naslove le-teh postavi v čakalno vrsto ukazov. Uporaba potrditvenih signalov lahko pripomore k izboljšanju zmogljivosti arhitekture, ker omogoča cevovodno izvajanje operacijskih paketov.

Uporaba cevovoda lahko privede do dodatnih težav kot so na primer cevovodne nevarnosti, ki lahko povzročijo ciklično izvajanje t.i. *praznih paketov* (angl. *idle tokens*). Poleg tega lahko opisani model pod določenimi pogoji preide v t.i. *mrtvo zanko* (angl. *deadlock*). Razlog tega je potrebno iskati v mehanizmu potrditve paketov v cevovodni implementaciji [3]. Opisane težave so pripeljale raziskovalne ekipe do razvoja prvih modelov dinamičnih PP arhitektur [3].

povezavah grafa ni prisotnih nobenih žetonov.

2.2.2 Dinamične arhitekture

Dinamične arhitekture so se razvile po predlogih raziskovalcev na MIT in na univerzi v Manchestru [3, 24]. Osnovni model dinamičnih arhitektur je prikazan na sliki 2.9. Model je sestavljen iz *enote za primerjanje*[6] (angl. *matching unit/store*), *procesne enote* (angl. *processing unit*), *enote za dostavo in ažuriranje* (angl. *fetch unit*) in *programskega pomnilnika* (angl. *program memory*).



Slika 2.9: Osnovni model dinamične podatkovno pretokovne arhitekture po [48].

Enota za primerjanje, ki ni nič drugega kot velik asociativni pomnilnik v katerem se nahajajo žetoni, ki čakajo na sosede, prejme žeton[7], ki prihaja iz procesne enote (t.j. rezultat neke operacije). Takoj po prihodu žetona enota dekodira njegovo oznako in v *pomnilniku ujemanj* (angl. *matching store*) preveri, če je prisoten drugi žeton z isto oznako[8]. V primeru, da takega žetona

[6] t.j. čisti asociativni pomnilnik v katerem se hranijo žetoni, ki čakajo na sosede

[7] Žetoni so v dinamičnih arhitekturah nosilci operandov in enolične oznake (četrti model v razdelku 2.1.2), medtem ko so ukazi sestavljeni iz operacijske kode in iz seznama naslovov vozlišč, ki zahtevajo kopijo rezultata operacije. Primer ukaza je prikazan na sliki 2.10.

[8] Dekodiranje s pomočjo konteksta oznake ugotovi,koliko operandov potrebuje ukaz, kateremu je žeton namenjen. V primeru, da ukaz zahteva več kot dva operanda (to je bolj

ni, se prejeti žeton zapiše v pomnilnik ujemanj skupaj z ostalimi čakajočimi žetoni. Če v pomnilniku ujemanj že obstaja žeton z isto oznako, se ta skupaj s prejetim žetonom prenese v enoto za dostavo in ažuriranje. Enota za dostavo in ažuriranje prevzame ukaz[9] iz naslova v programskem pomnilniku, ki ga enolično določa oznaka v žetonu. Enota za dostavo in ažuriranje s prevzetim ukazom je odgovorna za sestavljanje t.i. operacijskega paketa, ki vsebuje operande in operacijsko kodo ukaza oziroma vozlišča ter potrebne podatke za kodiranje oznake. Po končanem sestavljanju enota za dostavo in ažuriranje dostavi operacijski paket procesni enoti. Procesna enota dekodira operacijski paket, izvrši operacijo vozlišča in sestavi oznako izhodenga žetona. Rezultat delovanja procesne enote je žeton, ki vsebuje rezultat v obliki operanda s pripadajočo oznako. Tega procesna enota na koncu dostavi enoti za primerjanje. Žetoni se neposredno prenesejo iz izhoda procesne enote v enoto za dostavo in ažuriranje (angl. *matching stage bypass*), če ukaz zahteva eno-operandno operacijo. Pazljive analize najbolj uporabljenih podatkovno pretokovnih programov za dinamične arhitekture so ugotovile, da so take bližnjice prisotne v skoraj 40% vseh žetonov [4].

Slika 2.10: Osnovna zgradba ukaza dinamične podatkovno pretokovne arhitekture.

Na sliki 2.9 vidimo, da se v dinamičnih arhitekturah ukazi in operandi nahajajo v dveh različnih pomnilnikih in se operacijski paketi tvorijo samo zaradi

izjema v PP procesiranju), enota za primerjanje shrani število prisotnih čakajočih žetonov z enako oznako. Ob prihodu zadnjega žetona, se vsi žetoni (trije ali več) prenesejo iz pomnilnika ujemanj v enoto za dostavo in ažuriranje.

[9]Od tu naprej se beseda "ukaz" nanaša na podatkovno strukturo iz slike 2.10. Taki strukturi pravimo tudi *ukazna celica* (angl. *instruction cell)*. Ta neposredno predstavlja vozlišče v GPP. Operacija v vozlišču je neposredno določena z operacijsko kodo. Vozlišča katerim so namenjeni rezultati operacije so definirani s seznamom njihovih naslovov.

potrebe izvajanja v procesni enoti. V statičnih arhitekturah se ukazi in operandi nahajajo v istem pomnilniku.

Žetoni so v dinamičnih arhitekturah sestavljeni iz operanda in oznake. Oznaka je sestavljena iz naslednjih komponent: *naslov ukaza* v programskem pomnilniku (n), *polje konteksta* (c), ki enolično definira iz katerega konteksta se žeton sklicuje na določen ukaz (ta podatek lahko vsebuje tudi število operandov, ki jih operacija potrebuje), *števec prehodov zanke*[10] *(i)* (angl. *loop counter*), ki enolično definira indeks prehoda zanke, v katerem se žeton procesira ter *številka*[11] *vrat* (p), oziroma številka vhodne povezave vozlišča, ki ga določa naslov (n). Glede na to lahko žeton enolično predstavimo z zapisom (več o tem v [1, 50]), ki ga določa naslednja peterica:

$$\langle c, i, n, data \rangle_p \tag{2.3}$$

Če predpostavimo, da izvajamo prvo vozlišče iz slike 2.1 lahko zapišemo prva dva žetona v formalni obliki:

$$\langle c, 0, n, b \rangle_1 \qquad in \qquad \langle c, 0, n, a \rangle_2 \tag{2.4}$$

Ukaz, ki se nahaja na naslovu n v programskem pomnilniku bi lahko imel naslednjo formalno obliko:

$$\{add, \langle 2, 1 \rangle, m\}, \tag{2.5}$$

kjer je *add* simbol operacije seštevanja, *2* število operandov, ki ga operacija zahteva, *1* številka vhodne povezave (vrat) vozlišča, kateri je namenjen rezultat in m naslov tega vozlišča (ukaza). Rezultat izvajanja vozlišča bi lahko bil žeton z obliko:

$$\langle c, 0, m, a + b \rangle_1 \tag{2.6}$$

Več o obliki zapisa žetonov in o formatu ukazov lahko bralec dobi v [1].
Bolj podroben model dinamične PP arhitekture je prikazan na sliki 2.11. Slika prikazuje t.i. MIT *tagged token* arhitekturo, ki so jo razvili Arvind in sodelavci

[10] V primeru gnezdenih zank je možno zapisati vse števce prehodov zank v verigi v obliki $i_1.i_2.i_3$ *itd.* Število pik določa število gnezdenih zank.

[11] številka 1 označuje levo vhodno povezavo vozlišča, medtem ko številka 2 označuje desno vhodno povezavo vozlišča

[2, 3, 5]. Model iz slike 2.11 vsebuje tudi krmilnike vhodno/izhodnih sistemov in pomnilnike, ki se uporabljajo za procesiranje posebnih podatkovnih struktur, to so *I-strukture* [3]. Vidimo tudi, da je s tako arhitekturo dodajanje več procesnih enot v t.i. večprocesorsko polje preprosto. Izvajalni cikel tega modela je povsem enak tistemu na sliki 2.9, ki poleg tega omogoča tudi učinkovito implementacijo cevovoda [3].

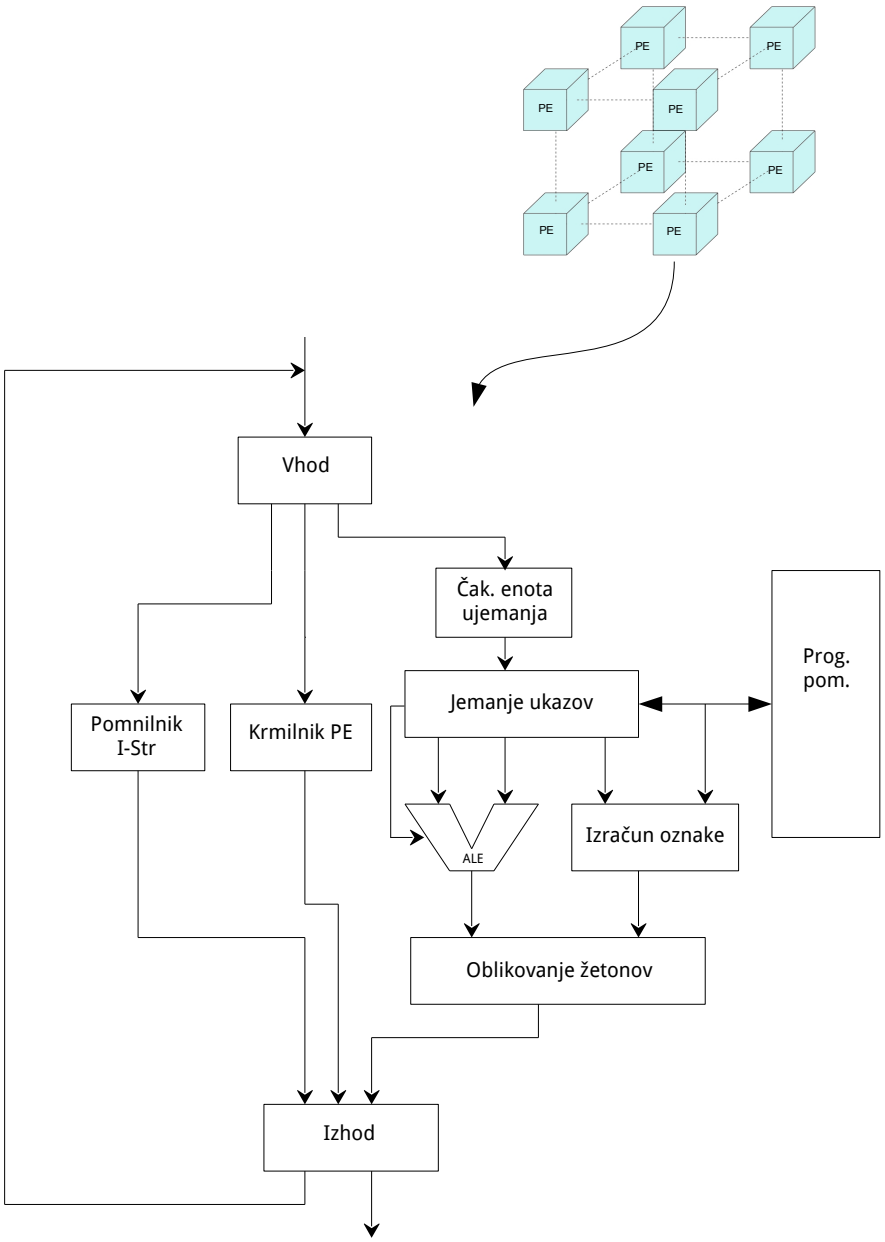
Model na sliki 2.11 dopolnjuje osnovni model na sliki 2.9 pri implementaciji povratnih povezav v GPP in torej posredno tudi zank. Pri osnovnem modelu smo zaradi enostavnosti predpostavili, da lahko ta izvaja samo dobro obnašajoče grafe. Razširitev modela zahteva implementacijo cikličnih grafov in torej tudi programskih zank. Podrobnosti o rešitvah, ki uporabljajo formalni jezik žetonov (kot v zapisu 2.3) za implementacijo preklopnih operatorjev in operatorjev združevanja ter o bolj naprednih rešitvah z I-strukturami, lahko bralec dobi v [3, 50].

Enota za primerjanje žetonov je ključna komponenta v dinamični arhitekturi, vsebuje kompleksno logiko za dekodiranje oznake žetonov in velik asociativni pomnilnik[12] za shranjevanje žetonov.

Če računalnik na sliki 2.9 izvaja določen GPP, pri katerem se v enoti za primerjanje začne kopičiti preveč čakajočih žetonov, se lahko pomnilnik ujemanj hitro napolni, vse dokler ne pride do preliva (angl. *memory overflow*). Žetoni lahko zapustijo pomnilnik ujemanj samo ob prihodu žetona z isto oznako. Če takih žetonov ni se pomnilnik napolni s čakajočimi žetoni. V tem primeru sistem preide v mrtvo zanko. Pri tem se učinkovitost procesiranja drastično zmanjša. Mrtva zanka se pojavi, ker se noben žeton ne pojavi v enoti za dostavo in ažuriranje. Jemanje ukazov iz programskega pomnilnika se tako ustavi. Računalnik preide v t.i. *mrtvi cikel* (angl. *idle state*), iz katerega ne pride več. Preprečevanje mrtvega cikla lahko rešimo z dodajanjem posebnih *žetonskih medpomnilnikov* (angl. *token buffer*), s katerimi dosežemo določeno minimalno količino aktivnih žetonov v krogu. Pogoj za to je uporaba dovolj velikega asociativnega pomnilnika ujemanj in dovolj velikega medpomnilnika, da je verjetnost pojava mrtve zanke sprejemljivo majhna [3]. To je bilo v zadnjih tridesetih letih praktično nemogoče zagotoviti[13]. Motivacija za razvoj PP arhitektur je uporaba velikih programov oz. velikih GPP, v katerih je izkoriščanje paralelnosti maksimalno. Veliki paralelni

[12] t.j. pomnilnik ujemanj v katerem poteka dostop preko vsebine oznake

[13] izključno zaradi uporabljene tehnologije



Slika 2.11: Podrobnosti multiprocesorskega sistema, ki sestavlja množico procesnih elementov (PE). Vsak procesni element je neodvisen računalnik z dinamično podatkovno pretokovno arhitekturo. Slika je povzeta po [3].

programi (torej zelo široki GPP) zahtevajo velik asociativni pomnilnik ujemanj. Premajhen pomnilnik lahko pripelje računalnik iz slike 2.9 v mrtvi cikel. Učinkovitega PP stroja z dinamično arhitekturo torej ni mogoče realizirati brez soočenja s problematiko realizacije velikega asociativnega pomnilnika. Z zmanjšanjem velikosti asociativnega pomnilnika ujemanj[14] se zmogljivost PP računalnika z dinamično arhitekturo drastično zmanjša (zaradi pojava mrtvih zank). Pri izbiri srednje velikega asociativnega pomnilnika z izvajanjem srednje-širokega GPP so zmogljivosti PP stroja manjše v primerjavi z von Neumannovim strojem [25]. V nadaljevanju bomo pokazali kako lahko z izbiro drugačne tehnološke platforme (t.j. biološki sistemi) teoretično premostimo problematiko asociativnega pomnilnika.

Poleg problema realizacije velikega asociativnega pomnilnika na hitrosti procesiranja v PP računalnikih neposredno vplivajo časovne zakasnitve (latence), ki se pojavijo pri dekodiranju oznak v enoti za primerjanje. Poleg teh zakasnitev so prisotne še dodatne zakasnitve zaradi dostopa do žetonov iz ene enote v drugo in zaradi dostopa ukazov v programskem pomnilniku (glej povezave modela na sliki 2.9). Vse te časovne zakasnitve lahko povzročijo občuten padec zmogljivosti procesiranja do te mere, da paralelnost nima več pozitivnega učinka.

Podrobne analize problematike velikega asociativnega pomnilnika so pripeljale raziskovalno skupino iz MIT (med njimi tudi prof. Arvind) do razvoja dinamične arhitekture z eksplicitnim pomnjenjem žetonov (angl. *explicit token store architecture*) [9]. Osnovno delovanje tega računalnika temelji na uporabi aktivnih okvirjev, ki se statično dodelijo v času prevajanja programa in na masivni uporabi I-struktur kot osnovne podatkovne strukture v višjih programskih jezikih. Podrobnosti arhitekture in organizacije ter podrobnosti dodeljevalnih in razvrščevalnih strategij, ki se uporabljajo v tem modelu lahko bralec dobi v [9, 29]. Poglavitni rezultat razvoja te arhitekture je bil eden izmed prvih delujočih podatkovno pretokovnih računalnikov na silicijevem čipu, t.j. računalnik Monsoon.

[14]Uporabljena CMOS tehnologija je do danes omogočala gradnjo relativno majhnih asociativnih pomnilnikov. Veliki asociativni pomnilniki so poleg tega zelo dragi.

2.2.3 Pomnilnik v podatkovno pretokovnih arhitekturah

Glavni pomnilnik v von Neumannovi arhitekturi je *pasiven*, kar pomeni, da naredi samo tisto kar od njega zahteva centralna procesna enota [28]. V nasprotju s tem so pomnilniki v statičnih in v dinamičnih PP arhitekturah *aktivni*, t.j. opravljajo določene operacije brez zunanjih ukazov, čeprav se te operacije pričnejo šele v prisotnosti prejetih žetonov iz zunanjih vodil. Primeri operacij so lahko iskanje, brisanje ali označevanje (t.j. postavljanje bitov na določenih naslovih). V večini primerov te operacije zahtevajo implementacijo posebnih mehanizmov krmiljenja pomnilnika. Primer takega pomnilnika je asociativni pomnilnik v enoti za primerjanje iz slike 2.9. Osnovni model arhitekture zahteva, da asociativni pomnilnik implementira naslednje operacije:

1. dekodiranje prejetega žetona skupaj z njegovo oznako,
2. preverjanje ali se v pomnilniku nahaja žeton z isto oznako in
3. sestavljanje paketa namenjenega enoti za dostavo in ažuriranje[15].

Načelo asinhronosti zahteva, da se vse te operacije izvedejo v določenem maksimalnem času. Maksimalni čas je tu vsota časa, ki ga potrebuje procesna enota za izvrševanje operacije določenega aktiviranega ukaza in časa, ki ga potrebuje enota za dostavo in ažuriranje za sestavo aktiviranih ukazov. V primeru, da se operacije asociativnega pomnilnika izvedejo v času, ki je daljši od maksimalnega časa lahko pride do izgube žetonov. Rešitev tega problema zahteva implementacijo medpomnilnika na vhodni povezavi enote za primerjanje kot je prikazano v modelu na sliki 2.9.

Ena izmed glavnih slabosti PP arhitektur je upravljanje spremenljivk in podatkovnih struktur. Večina modelov PP arhitektur predpostavlja uporabo žetonov kot nosilcev podatkovnih vrednosti v GPP. V vseh modelih arhitektur pa imajo žetoni zelo kratko "*življenjsko dobo*", ker so porabljeni skozi vozlišča GPP, kjer se neprestano ustvarjajo in brišejo. S takim podatkovnim objektom je posebno težko realizirati statične spremenljivke ali poljubne podatkovne strukture. V realnih okoliščinah se večkrat srečujemo z algoritmi, ki zahtevajo uporabo kompleksnih podatkovnih struktur (kot so drevesa, povezani

[15] tukaj je paket definiran kot par žetonov z enako oznako

seznami, grafi, kopice itd). V modelih PP arhitektur predstavlja problem že sama implementacija tabele z desetimi elementi. Tega problema se je zavedal že Dennis [13]. Od takrat so se razvile različne rešitve, ki v večji ali manjši meri poenostavljajo implementacijo podatkovnih struktur z uporabo GPP in z uporabo posebno namenjenih krmilnikov in pomnilnikov. Na tem mestu ne bomo opisovali podrobnosti saj jih bralec lahko dobi v [3, 29]. Več o upravljanju podatkovnih struktur v PP arhitekturah lahko bralec dobi v [29].

2.2.4 Primerjava statičnih in dinamičnih arhitektur

Največja prednost dinamičnih arhitektur v primerjavi s statičnimi je višja zmogljivost zaradi večjega števila žetonov v posameznih povezavah GPP, kar zagotavlja večji izkoristek implicitne paralelnosti. Statične arhitekture imajo po drugi strani bolj enostaven model. Njihova realizacija je preprosta, medtem ko sta glavni slabostih podvojeni tok podatkov, saj je potrebna implementacija potrditvenih žetonov na vsaki povezavi GPP in pomanjkanje podpore za programske konstrukte, kot so klici podprogramov (angl. *procedure call*) [50].

Glavne slabosti dinamičnih arhitektur so skrite v sami strukturi in realizaciji. Te arhitekture zahtevajo uporabo velikega čistega asociativnega pomnilnika, v katerem je potrebno shraniti žetone, ki čakajo na "soseda" z isto oznako. Problem je v tem, da velikih asociativnih pomnilnikov z današnjo CMOS tehnologijo preprosto ne moremo realizirati [28]. V bolj razvitih modelih dinamičnih PP arhitektur [49] se temu problemu izognemo z uporabo drugačnih pomnilnikov, ki izkoriščajo posebne vrste *zgoščevalnih mehanizmov* (angl. *hashing techniques*) [50]. Po drugi strani so ti pomnilniki počasnejši v primerjavi z asociativnimi pomnilniki.

2.3 Programski jeziki

Sočasno z razvojem strojne opreme se je razvila tudi programska oprema PP računalnikov. Eden izmed prvih uporabljenih PP programskih jezikov je bil VAL (angl. *Value-Oriented Algorithmic Language*). VAL je razvila raziskovalna ekipa pod vodstvom prof. Dennisa [13]. VAL je jezik, ki je bil zasnovan na statični PP arhitekturi Dennisovega dela [13], zato v osnovi ne podpira dinamičnih arhitektur. VAL je v nasprotju z večino tedanjih programskih

jezikov funkcionalni jezik, čeprav je močno tipiziran in zaradi tega vsebuje velike podobnosti z jezikom LISP.

Algoritem 2.1 Primer zapisa funkcije v jeziku VAL.

```
function enacba(a, b, c, d, e: real) : real;
      let
            x : real := ( (a + b) * 4 - (c + a*d) ) / e;
      in
            x
      end
end
```

Višji programski jezik PP računalnikov, kot je na primer VAL, predstavlja formalni zapis procedure, ki jo neposredno implementira GPP. Prevajalnik jezika VAL prevede zapis iz višjega programskega jezika v seznam paketov, ki predstavlja GPP v programskemu pomnilniku. Posebni nalagalnik je odgovoren za postavitev prvih aktivnih paketov seznama v čakalni vrsti ukazov (slika 2.8). Primer programa v jeziku VAL je prikazan v algoritmu 2.1, pripadajoči graf pa na sliki 2.1. Iz primera je razvidno, da sintaksa jezika omogoča hitro komponiranje funkcij. Poleg tega jezik VAL podpira klasične podatkovne tipe, in sicer celoštevilska predznačena števila, števila v plavajoči vejici, boolean vrednosti in znake. Več o sintaksi jezika in o osnovnih algoritmih zapisanih v tem jeziku lahko bralec dobi v [14, 18, 50].

Algoritem 2.2 Primer implementiranega algoritma iz slike 2.4 v jeziku Id.

```
(initial i <-- 1; sum <-- 0;
       while i <= N do
           new i <-- i+1;
           new sum <-- sum + f(i);
 return sum)
```

Tudi za dinamične arhitekture je bilo razvitih nekaj višjih programskih jezikov, med temi pa je nujno omeniti jezik *Id* (angl. *Irvine dataflow language*). Id je višji programski jezik, ki ga je razvil Arvind v sodelovanju z raziskovalci na univerzi Irvine. Jezik je osnovan na arhitekturi iz slike 2.11 in je bil večkrat

adaptiran za potrebe razvoja drugih dinamičnih arhitektur. Primer zapisa procedure v jeziku Id prikazuje algoritem 2.2. Na tem mestu ne bomo posebej predstavljali posamezne značilnosti in podrobnosti tega jezika, saj bralec lahko to dobi v [5, 38].

2.4 Prednosti in slabosti podatkovno pretokovnih arhitektur

Skupna šibka točka vseh PP arhitektur predstavlja problem upravljanja s kompleksnimi podatkovnimi strukturami in način dodeljevanja virov posameznim PP programom. Bralec lahko dobi podrobno analizo problematike v [29]. V primeru multiprocesorskega PP sistema z večjim številom procesnih elementov (t.j. omrežje PP računalnikov) se pojavi še problem maksimalnega izkoriščanja procesnih elementov glede na optimalno izvrševanje GPP na multiprocesorskem sistemu [47, 48].

PP računalniki (s statično ali z dinamično arhitekturo) se niso nikoli pojavili na tržišču zaradi številnih problemov in težav, ki smo jih spoznali v razdelkih 2.2.1, 2.2.2 in 2.2.3. Po drugi strani so bile izvedene številne meritve za primerjavo zmogljivosti PP računalnikov s klasičnimi von Neumannovimi stroji, ki so pokazale definitivno premoč slednjih [4, 25]. To še ne pomeni, da so PP računalniki slabši oziroma, da je osnovni model PP računalnika napačen. V primeru izvajanja zelo širokih acikličnih dobro obnašajočih GPP dinamične arhitekture dosegajo vrhunsko stopnjo paralelnosti, ki jih von Neumannovi računalniki nikakor ne morejo doseči. V primeru izvajanja strogo zaporednega algoritma in torej zelo ozkega GPP, so zmogljivosti dinamičnega PP računalnika neprimerno manjše v primerjavi z von Neummanovim računalnikom [4]. Ob enaki zmogljivosti, so von Neumannovi računalniki cenejši v primerjavi s PP računalniki [20, 28]. Razlog za to je potrebno iskati v sami realizaciji. PP računalniki zahtevajo kompleksnejšo upravljanje s pomnilnikom (angl. *memory management*) v primerjavi z von Neummanovimi računalniki, kar lahko občutno poveča stroške implementacije [20].

V nadaljevanju bomo pokazali, da je z uporabo povsem drugačne platforme oziroma s povsem drugačno tehnologijo teoretično možno odpraviti nekatere glavne slabosti modelov PP računalnikov.

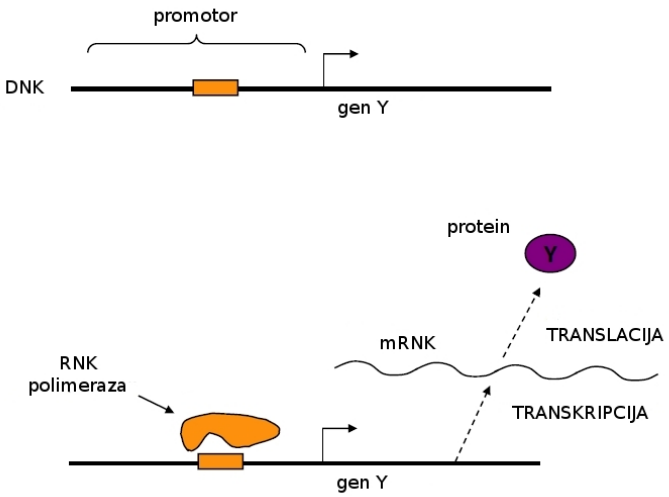
Poglavje 3

Gensko regulatorna omrežja

Gensko regulatorna omrežja - GRO (angl. *gene regulatory networks* - GRNs) omogočajo realizacijo logičnih vezij v celicah živih organizmov na osnovi dinamike izražanja *deoksiribonukleinske kisline* - DNK (angl. *deoxyribonucleic acid* - DNA) [27]. GRO je množica DNK vezavnih mest, DNK vezavnih proteinov in genov, ki sestavljajo zaokroženo celoto biokemijskih (regulatornih) procesov. Genska regulacija poteka na podlagi posebnih proteinov, ki jim pravimo *transkripcijski faktorji* (angl. *transcription factors*).
V tem poglavju bodo predstavljeni osnovni regulatorni procesi, ki omogočajo realizacijo logičnih vezij z uporabo GRO. Bralec si lahko o tej temi bolj podrobno prebere v [11].

3.1 Osnovni pojmi

Dinamika delovanja DNK molekule ima vrsto zanimivih lastnosti v kontekstu procesiranja informacij. Zanimivi so procesi genske regulacije, ki povzročijo zaviranje ali pospeševanje ekspresije določenih *genov*. DNK je nosilec informacij o strukturi oziroma sestavi *proteinov* v celici. Te informacije so zakodirane v genih, ki se nahajajo vzdolž DNK verige. Sinteza proteina se začne z vezavo *RNK polimeraze* na mesto v DNK verigi, ki mu pravimo *promotor*. Vsakemu genu pripada promotor. RNK polimeraza je začetnik procesa *transkripcije* (angl. *transcription*), ki prepiše vsebino gena v molekulo *mRNK*. Molekula mRNK vsebuje zaporedje prepisanih nukleotidov, ki se v procesu *translacije* (angl. *translation*) v *ribosomih* sintetizira v aminokislinsko zaporedje in s tem v protein. Opisani postopek sinteze je skiciran na sliki 3.1.



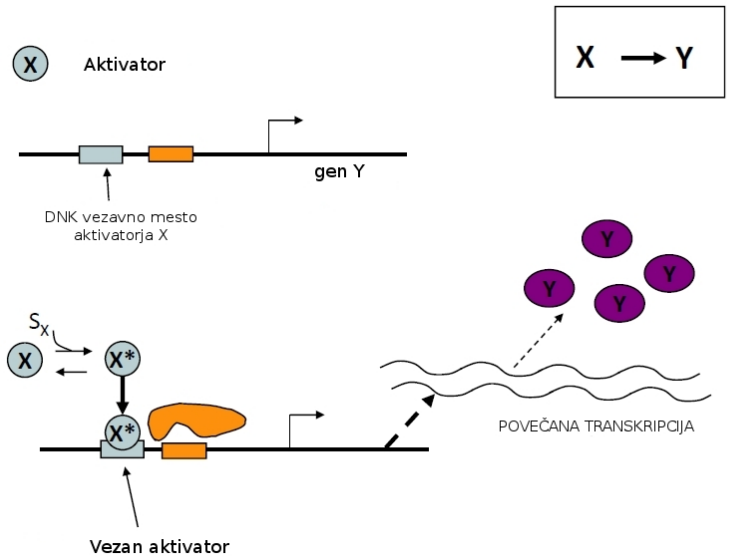
Slika 3.1: Poenostavljen prikaz postopka sinteze proteinov. Slika je povzeta po [26]

Sliko 3.1 lahko dopolnimo z upoštevanjem posebnih mehanizmov za nadzor genske ekspresije. Pri tem igrajo poglavitno vlogo transkripcijski faktorji. To so posebni kontrolni proteini, ki zavirajo ali pospešujejo proces transkripcije. Transkripcijske faktorje lahko glede na njihov vpliv razdelimo na *represorje* in *aktivatorje*.

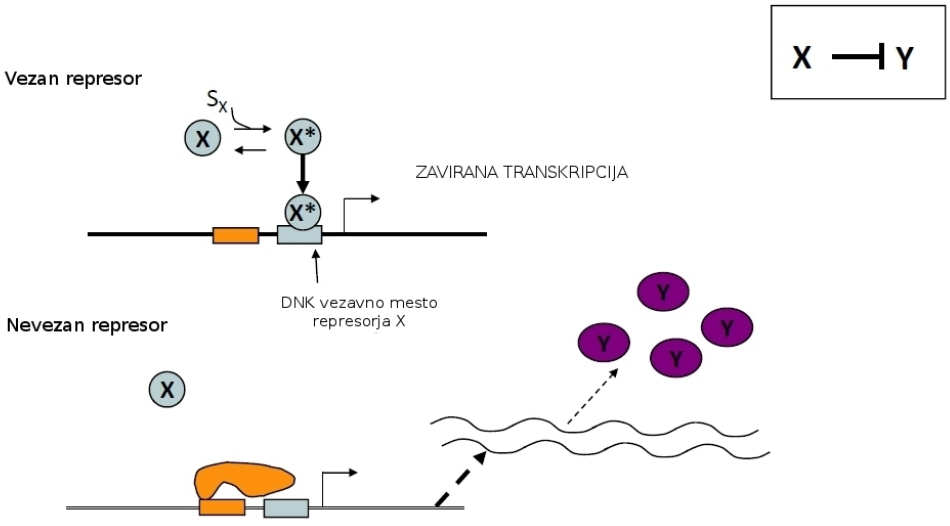
3.2 Kontrola genske ekspresije

Nadzor nad postopkom sinteze določenega proteina poteka v glavnem preko dveh mehanizmov, in sicer *aktivacije* in *represije*. Pri aktivaciji se DNK vezavni protein - *aktivator* veže na promotor in s tem začne proces transkripcije soležnega gena. Pri represiji se DNK vezavni protein - *represor* veže na represorsko mesto in s tem zavre proces transkripcije oziroma utiša ekspresijo gena. Proces aktivacije je skiciran na sliki 3.2, medtem ko je proces represije prikazan na sliki 3.3.

GRO so sestavljena iz množice DNK segmentov prikazanih na sliki 3.2 in 3.3. Pri tem pripadajoči geni kodirajo informacijo o DNK vezavnih proteinih, ki imajo neposreden vpliv na regulacijo ostalih genov v omrežju.



Slika 3.2: Poenostavljen prikaz dinamike aktivatorjev na določenem DNK segmentu [26].



Slika 3.3: Poenostavljen prikaz dinamike represorjev na določenem DNK segmentu [26].

Na proteine lahko gledamo kot na signale oziroma logične vrednosti, na GRO pa kot na logične strukture in generatorje signalov. Slika 3.3 prikazuje dva temeljna dogodka. V prvem primeru je vhodni signal oziroma represor X prisoten, kar povzroči inhibicijo gena in zaviranje transkripcije, ter posledično odsotnost izhodnega signala - proteina Y v sistemu. V drugem primeru pa ni prisotnih molekul represorja X, zato se gen lahko izraža. Koncentracija proteina Y se posledično poveča. Opisano delovanje spominja na NOT vrata oziroma negacijo. Več o tovrstnih realizacijah logičnih struktur si lahko bralec prebere v [10].

3.3 Realizacija logičnih vezij z gensko regulatornimi omrežji

Načrtovanje in snovanje logičnih vezij z GRO predstavlja posebno poglavje, ki presega namen te diplomske naloge. Bralec si lahko več prebere o osnovah tehnologije in o modeliranju logičnih sistemov z GRO v [10, 11, 27, 33, 41, 44].

Tehnologija rekombinantne DNK omogoča sestavljanje poljubnega zaporedja genov in DNK vezavnih mest v določenem vektorju oz. plazmidu. V prejšnjem desetletju je bilo načrtovanje velikih GRO zelo težko zaradi omejenega nabora naravnih represorjev. Po odkritju sintetičnih DNK vezavnih proteinov [45] imajo biotehnologi na voljo večjo izbiro represorjev. Gensko regulatorna omrežja s sintetičnimi DNK vezavnimi proteini, kot so *cinkovi prsti* (angl. *zinc-fingers*) [45] so zanimiva zaradi stabilnosti in učinkovitosti delovanja. Sintetičnih DNK vezavnih proteinov oziroma sintetičnih represorjev je veliko (mnogo več v primerjavi z naravnimi represorji) zato se z njihovo uporabo odpirajo možnosti sestavljanja velikih GRO. Potrebno je posvetiti pozornost velikosti procesnih elementov, ker smo tukaj omejeni s sposobnostjo celice za sprejem večje količine DNK.

Gensko regulatorna omrežja nudijo številne prednosti. GRO teoretično omogočajo paralelno izvajanje velikega števila logičnih operacij saj lahko določeno logično vezje implementiramo na več različnih mestih vektorja. Poleg tega lahko v celici zagotovimo prisotnost več vektorjev hkrati. Zaradi možnosti izkoriščanja velike vgrajene paralelnosti, GRO predstavljajo eno izmed najbolj zanimivih platform za razvoj računalniških sistemov.

GRO delujejo popolnoma asinhrono. Če želimo zagotoviti določeno stopnjo sočasnosti, potrebujemo sinhronizacijske signale. Te lahko generiramo z uporabo genskih oscilatorjev. Primer realizacije takega oscilatorja je predstavljen v [17]. Primeri modeliranja in načrtovanja različnih vrst oscilatorjev na osnovi sintetčnih GRO ter podrobnejše matematične analize stabilnosti sistemov, pa v [36, 51].

Pri načrtovanju velikih GRO lahko postane njihova realizacija problematična. Zanima rešitev predstavlja razširitev GRO na večcelične sisteme, pri katerih se za medcelično komunikacijo uporablja t.i. *quorum sensing*[1] [30]. Quorum sensing predstavlja samo enega izmed možnih načinov medcelične komunikacije. Podroben opis metod in načinov implementacije takih rešitev presega namen tega diplomskega dela. Vredno je omeniti, da modeliranje večceličnih sistemov zahteva dopolnjeno matematično predstavitev dinamike GRO [53].

[1]Tukaj imamo v mislih bakterijske celice. Te imajo v primerjavi s sesalskimi hitrejši odziv na določene zunanje signale. Pri njih je sinteza proteinov hitrejša saj ne vsebujejo jedra. Transkripti potujejo neposredno v citoplazmi brez potrebe po prehodu skozi jedro celice. Neupoštevanje takih podrobnosti lahko močno vpliva na *in silico* modeliranje dinamike GRO.

Poglavje 4

Gensko regulatorna omrežja kot podatkovno pretokovna procesna platforma

Dinamiko gensko regulatornih omrežij (GRO) lahko matematično opišemo z uporabo časovnih Petrijevih mrež [19]. V 2. poglavju smo omenili, da so tudi GPP posebna oblika Petrijeve mreže. Če gledamo na GRO kot na graf, v katerem vozlišča predstavljajo posamezni DNK segmenti, povezave pa DNK vezavne proteine (oziroma represorje ali aktivatorje) se izkaže, da imata GPP in GRO veliko skupnih lastnosti.
V tem poglavju bodo predstavljene analogije podatkovno vodenega procesiranja s procesiranjem informacij na nivoju GRO. V razdelku 4.2 bo predstavljen predlog hipotetične procesne platforme na osnovi *sintetičnih GRO*. Prednosti in slabosti te procesne platforme bodo podrobneje opisane v razdelku 4.3.

4.1 Arhitekturne podobnosti gensko regulatornih omrežij z grafi pretoka podatkov

Logika v ozadju GRO temelji na represiji in aktivaciji posameznih DNK segmentov, ki implementirajo logična vrata. Preklopi potekajo povsem asinhrono, njihova hitrost pa je odvisna od številnih biokemijskih lastnosti uporabljenih gradnikov kot so hitrosti transkripcije in translacije, bližina ribosomskih mest, mobilnost proteinov v citoplazmi, konstante difuzije molekul, stopnja kooperativnosti, itd. Ko je koncentracija DNK vezavnega proteina v GRO do-

volj visoka, se sprožijo regulacijske reakcije, katerih rezultat je povečana ali zmanjšana koncentracija določenega DNK vezavnega proteina (*izhod vezja*). Mehanizem preklopa torej deluje glede na prisotnost določene koncentracije reaktanta (v tem primeru DNK vezavnih proteinov). Opisani mehanizem spominja na pravilo *vžiga*[1] vozlišč v grafih pretoka podatkov, vendar le-temu ni popolnoma enak (več o tem v razdelku 4.3). Na DNK vezavne proteine lahko gledamo kot na posebno vrsto "*operandov*" oziroma "*žetonov*", na DNK segmente, ki implementirajo določeno logično operacijo na podlagi represije ali aktivacije genov pa kot na operacije v vozliščih pri PP procesiranju. Poleg tega oba modela procesiranja delujeta asinhrono in omogočata izkoriščanje maksimalne implicitne paralelnosti prisotne v računskih postopkih. V GRO se lahko transkripcija in translacija izvajata na več mestih hkrati in se zato regulacija vrši paralelno na več DNK segmentih istočasno. Brez posebnih zadržkov lahko trdimo, da je procesiranje informacij na osnovi GRO sorodno PP procesiranju [54].

4.2 Programski model gensko regulatornih omrežij

V razdelku 4.1 smo ugotovili, da si lahko DNK vezavni protein v GRO interpretiramo kot žeton v GPP. Ugotovitev je lahko zavajujoča saj so žetoni v GPP lahko nosilci celoštevilskih operandov, ki so predstavljeni z aritmetiko v fiksni vejici oziroma z več-bitno besedo. DNK vezavni proteini po drugi strani lahko predstavljajo zgolj eno logično vrednost (eno-bitno besedo). Temu problemu se izognemo tako, da predpostavimo procesiranje z GPP, ki vsebuje samo eno-bitne operande. Zaradi velikostnih omejitev je realizacija večjih logičnih struktur danes skoraj nemogoča, zato se bomo v nadaljevanju omejili na razvoj eno-bitnih in dvo-bitnih operandov. V primeru uporabe GPP z eno-bitnimi operandi lahko trdimo, da so žetoni povsem ekvivalentni DNK vezavnim proteinom v GRO. Ta trditev omogoča neposredno primerjavo in uporabo lastnosti GPP tudi v GRO. Predpostavimo, da želimo izvajati določen GPP[2], ki vsebuje izključno eno-bitne operande in enostavne logične operacije na osnovi GRO. Poleg tega predpostavljamo, da:

[1] t.j. v grafu pretoka podatkov se vozlišče aktivira natanko takrat, ko so prisotni vsi potrebni operandi na vhodnih povezavah vozlišča

[2] na tem mestu bomo predpostavili uporabo izključno acikličnih dobro obnašajočih GPP

1. Vsaki povezavi GPP pripada v GRO določen DNK vezavni protein.
2. Vsako vozlišče GPP se prevede v GRO v množici DNK segmentov, ki implementirajo logično operacijo vozlišča.

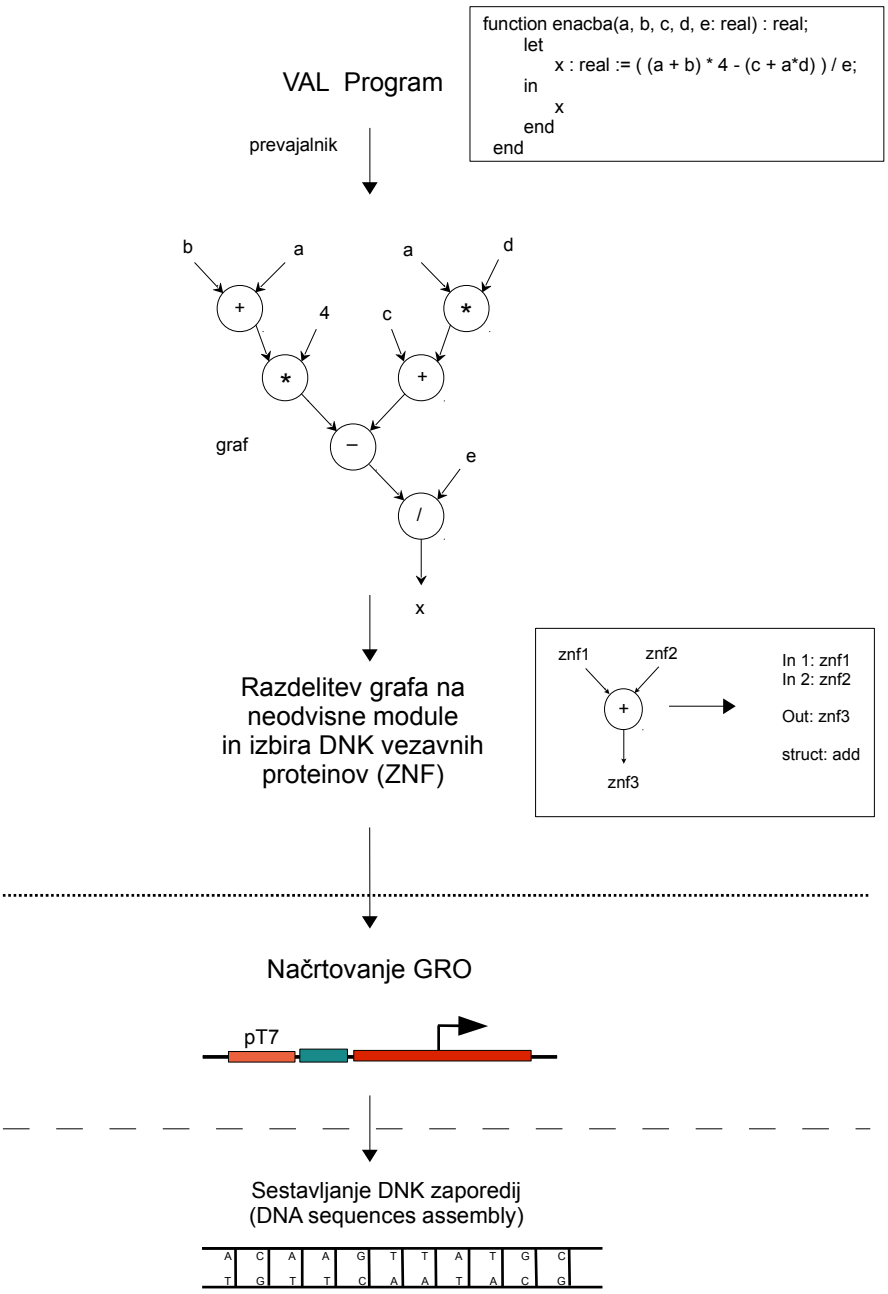
Glede na dane predpostavke, lahko posamezna vozlišča GPP realiziramo z množico DNK segmentov (glej sliko 3.3). Realizacija in sestavljanje potekata modularno, pri čemer so DNK vezavni proteini povezovalni elementi vozlišč. Vsi moduli so med seboj neodvisni, zato se lahko izvajajo hkrati. Potrebno pa je opozoriti, da je neodvisnost samo konceptualna. V realnih pogojih se lahko vozlišča izvajajo vzporedno, genska regulacija v posameznih vozliščih[3] pa se izrazi šele po določenem času. To je čas, ki ga vhodni DNK vezavni protein potrebuje, da doseže potrebno koncentracijo za pričetek represije oziroma aktivacije.

Shematski prikaz opisanega postopka oziroma implementacijo GPP[4] s hipotetičnim, sintetičnim GRO prikazuje slika 4.1. Pri tem smo predpostavljali uporabo umetnih cinkovih prstov, ki jih označujemo s kratico ZNF.

Pri sestavljanju modulov moramo biti še posebej pozorni pri izbiri DNK vezavnih proteinov za posamezne povezave. Uporaba istega DNK vezavnega proteina na več različnih mestih grafa pretoka podatkov lahko pripelje do neželjenih interferenc med DNK segmenti GRO in s tem do nepravilnega delovanja. Temu problemu se enostavno izognemo z enoličnim dodeljevanjem DNK vezavnih proteinov posameznim povezavam.

Dobra lastnost tega pristopa je izkoriščanje že uporabljenih in že testiranih modulov. Njegova glavna omejitev je velikost izvršljivega grafa, saj zaradi omejitev tehnologije rekombinantne DNK poljubno velikih GRO v enoceličnem okolju ne moremo realizirati.

[3] to so vozlišča, ki so podatkovno odvisna od višjih nivojev grafa
[4] tega dobimo iz prevoda funkcionalnega jezika - v tem primeru VAL



Slika 4.1: Programski model podatkovno vodenega procesiranja na osnovi gensko regulatornih omrežij. Črtkani vodoravni črti na tretji stopnji programskega modela predstavljata meji med računalniškim in biotehnološkim načrtovanjem. Tehnologija rekombinantne DNK [11] omogoča fizično realizacijo zadnjega nivoja (t.j. sestavljanje DNK zaporedij).

4.3 Prednosti in slabosti gensko regulatornih omrežij kot podatkovno pretokovne procesne platforme

Razvoj splošno namenskega računalniškega sistema na osnovi GRO, je trenutno izvedljiv samo hipotetično. Von Neumannov računalniški model zahteva implementacijo avtomata s kompleksno logiko za dekodiranje in izvajanje ukazov ter implementacijo velikega naslovljivega pomnilnika [28]. Z moderno tehnologijo rekombinantne DNK je to skoraj nemogoče realizirati. Brez dvoma lahko trdimo, da je pristop z von Neummanovim modelom, vsaj s trenutno tehnologijo, povsem neprimeren. Boljši pristop predstavlja procesiranje na osnovi grafov pretoka podatkov, saj le-ta odraža številne podobnosti z GRO.

Pri uporabi metode preslikovanja GPP v GRO sintetičnih DNK vezavnih proteinov (glej sliko 4.1) lahko izkoristimo tako prednosti PP procesiranja kot je paralelnost izvajanja vozlišč GPP, kot tudi prednosti procesiranja na nivoju GRO kot je modularno sestavljanje posameznih logičnih komponent.

Največje težave srečamo pri implementaciji več-bitnih logičnih struktur. Vzemimo kot primer dvo-bitni seštevalnik pri katerem običajno zahtevamo, da se oba bita vsote (rezultata) pojavita hkrati. Ta pogoj je v GRO praktično nemogoče izpolniti, saj pri kaskadni implementaciji eno-bitnih seštevalnikov, kot vhod v drugem potrebujemo ostanek prvega seštevalnika. Med dvema bitoma se tako pojavi velika zakasnitev. Latenca je lahko izjemno velika, če poleg več-bitnih struktur zahtevamo še, da se rezultat naknadno uporablja vzdolž GPP. Implementacija več-bitnih kombinatoričnih in sekvenčnih vezij na osnovi GRO zato predstavlja zanimiv izziv [40]. Velik problem predstavlja tudi predstavitev števil in implementacija bolj kompleksnih aritmetičnih operacij (deljenje, račun kvadratnega korena, itd). Glavna slabost programskega modela iz slike 4.1 je omejitev uporabe GPP z izključno eno-bitnimi operandi.

Poleg hudih restrikcij v aritmetiki se pri izvajanju GPP na osnovi GRO, pojavi poseben sinhronizacijski problem, ki mu pravimo *predčasna pojavitev rezultata*. Le-ta lahko predstavlja velik problem v večini primerov izvajanih GPP na osnovi GRO. Za njegovo reševanje potrebujemo določen mehanizem sinhronizacije kot je mehanizem vžiga v GPP. Poglejmo kaj se lahko zgodi v konkretnem primeru: predpostavimo, da smo sposobni z GRO izvajati GPP na sliki 2.1. V realnem okolju bi se vsa vozlišča grafa izvajala hkrati. Če se os-

redotočamo na operacijo deljenja z e lahko ugotovimo, da se lahko ta operacija izvaja hkrati s prvo operacijo $a + b$. Na izhodu se rezultat pojavi predčasno. Pravilni rezultat se na izhodu pojavi šele, ko se genska regulacija DNK vezavnih proteinov stabilizira v t.i. *ustaljeno stanje* (angl. *steady state*) skozi vsa vozlišča izvajanega grafa. Velik GRO, ki implementira zelo dolgi GPP lahko tako doseže ustaljeno stanje šele po zelo dolgem času. Zaradi tega se skušamo izogniti izvajanju zelo dolgih zaporednih GPP na osnovi GRO[5].

Drugi problem, ki zadeva izvajanje GPP na osnovi GRO, je stabilnost vhodnih signalov. Če izvajamo določen acikličen dobro obnašajoči GPP z modelom na sliki 4.1, so rezultati na izhodih GPP na voljo vse dokler je GRO v ustaljenem stanju. To dosežemo natanko tedaj, ko so tudi koncentracije vseh vhodnih DNK vezavnih proteinov stabilne (na grafu 2.1 to pomeni stabilnost koncentracij vhodnih DNK vezavnih proteinov a, b, c, d, e, in konstante 4). V realnih GRO lahko to dosežemo s konstitutivnim izražanjem določenih genov, ki sproti sintetizirajo vhodne DNK vezavne proteine.

Ob analizi razlik med dinamično arhitekturo procesiranja podatkov iz slike 2.9 in programskim modelom iz slike 4.1, s predpostavko izvajanja izključno acikličnih dobro obnašajočih GPP, opazimo sledeče poglavitne izboljšave:

1. Aciklični dobro obnašajoči graf pretoka podatkov lahko neposredno implementiramo v sintetičnih gensko regulatornih omrežij.

2. Potreba po *označevanju* (angl. *labeling*) žetonov odpade, ker posamezni DNK vezavni proteini v GRO enolično definirajo vhod oziroma izhod posameznih vozlišč grafa.

3. Potreba po velikem asociativnem pomnilniku odpade, ker odpade tudi potreba po implementaciji enote za primerjanje.

4. Potreba po programskem pomnilniku in enoti za dostavo in ažuriranje odpade, ker je program v celoti zapisan na DNK segmentih.

Z uporabo procesiranja na nivoju GRO smo shemo na sliki 2.9 reducirali na procesno enoto. To nam je uspelo, ker se v GRO podatki nahajajo v istem prostoru kot procesni elementi. Po drugi strani se v shemi na sliki 2.9 podatki nahajajo v različnih prostorih.

[5]to predstavlja tudi splošni način reševanja problema sinhronizacije

Implementacija zank v GRO in posledično izvajanje poljubnih GPP predstavlja problem, kateremu bi se bilo zanimivo posvetiti v dodatnih raziskavah. Za potrebe programskega modela na sliki 4.1 smo predpostavili, da lahko izvajamo izključno aciklične dobro obnašajoče GPP. Za razvoj splošno namenskega PP računskega stroja na osnovi GRO, ki bi bil ekvivalent Turingovemu stroju pa je implementacija zank nujno potrebna.

Podatkovno pretokovne značilnosti GRO ne predstavljajo edinega pogleda na podatkovno pretokovno procesiranje v celici. Bralec lahko dobi osnovne primere PP procesiranja na nivoju mikro-reaktorjev v [37].

Poglavje 5

Implementacija osnovnih konstruktov podatkovno pretokovnega procesiranja z gensko regulatornimi omrežji

Realizacija računalniških sistemov in kompleksnih logičnih vezij za procesiranje informacij na osnovi GRO že več kot desetletje predstavlja enega izmed poglavitnih izzivov sintezne biologije. Razvoj enostavnih logičnih vezij z uporabo naravnih DNK vezavnih proteinov in zanesljivih DNK vezavnih mest je v zadnjem desetletju dosegel izjemne rezultate (glej kot primer [17]). Zaradi omejenega nabora naravnih transkripcijskih represorjev je realizacija poljubno velikih logičnih vezij še vedno problematična. V zadnjih letih postaja izvedljiva ideja o uporabi t.i. *sintetičnih transkripcijskih represorjih*. Umetni *cinkovi prsti* (angl. *zinc fingers*) predstavljajo primer takih represorjev. To so relativno stabilni proteini, z zelo visoko DNK vezavno afiniteto, njihova relativno enostavna oblika pa omogoča hiter pretok informacij v notranjosti celic. Hipoteze o možnosti razvoja poljubnih GRO z uporabo cinkovih prstov so v teku testiranja [21].

Pred fizično realizacijo v laboratoriju je potrebno delovanje načrtovanih GRO testirati *in silico*. Samo ob pravilnem delovanju vseh potrebnih testnih primerov se lahko lotimo *in vitro* implementacije načrtovanega GRO. Testiranje delovanja GRO poteka s simulacijo determinističnih in stohastičnih modelov, ki opisujejo delovanje načrtovanih omrežij.

V tem poglavju bomo na kratko predstavili poglavitne značilnosti stohastičnega modeliranja GRO in implementacije osnovnih konstruktov grafa pretoka podatkov, ki so bistveni za razvoj podatkovno pretokovnih računalniških sistemov, na osnovi gensko regulatornih omrežij.

5.1 Stohastično modeliranje dinamike bioloških sistemov

Vsako gensko regulatorno omrežje lahko formalno opišemo z množico *povezanih biokemijskih reakcij* (angl. *chemical reaction channels*) in *sistemom reakcij*. Vsaka reakcija sistema deluje z določeno hitrostjo, ki jo določajo njeni kinetični parametri. Večina numeričnih načinov modeliranja GRO uporablja sistem reakcij kot osnovno matematično postavko modela in hitrosti reakcij kot glavne parametre simulacij.

Deterministični modeli GRO predstavljajo natančno dinamiko sistema reakcij. Najbolj pogosto orodje za deterministično modeliranje GRO predstavljajo *sistemi navadnih diferencialnih enačb* (angl. *ordinary differential equations - ODEs*), ki opisujejo natančno kinetiko vseh upoštevanih biokemijskih reakcij. Njihovo numerično reševanje je relativno preprosto. Glavna slabost determinističnih modelov na osnovi sistemov diferencialnih enačb je zahtevna predstavitev nekaterih pojavov, ki lahko vsebinsko vplivajo na potek reakcij v GRO. To so na primer nedeterministično obnašanje proteinov v celici, neuspešen proces transkripcije mRNK molekule ali neuspešen proces translacije proteinov v ribosomih. Razlog za to je neupoštevanje prisotnosti določene količine šuma. Šum je glavni povzročitelj nedeterminističnega obnašanja bioloških procesov [27, 44].

Stohastični modeli omogočajo bolj natančno modeliranje dinamike bioloških sistemov zaradi stohastičnosti kemijskih procesov [44]. Stohastični modeli GRO pri modeliranju biokemijskih reakcij upoštevajo določeno stopnjo naključnosti, ki pripelje do stohastične kinetike[1] poteka koncentracij opazovanih kemijskih zvrsti. To kinetiko lahko opišemo s *kemijsko glavno enačbo* (angl. *chemical master equation - CME*) [44]. Njeno analitično reševanje nam omogoča natančno določanje verjetnosti posameznih reaktantov in produktov reakcij

[1] t.j. kinetika pri katerem se posameznemu reaktantu v sistemu dodeli določeno verjetnost pojavljanja

sistema skozi čas, ampak postane neobvladljiva že pri manjših sistemih [36, 44].

Ponavadi se za reševanje CME uporabljajo aproksimacijski pristopi, med katere spada tudi t.i. *stohastični simulacijski algoritem* - SSA (angl. *stochastic simulation algorithm* - SSA) [23], ki si ga bomo v nadaljevanju podrobneje ogledali.

5.1.1 Stohastični simulacijski algoritem

Predpostavimo, da se GRO nahaja v prostoru s konstantno prostornino Ω. Opazujemo M reakcij $\{R_1, ..., R_M\}$ s pripadajočimi reakcijskimi oziroma kinetičnimi parametri $\{c_1, ..., c_M\}$ in N kemijskimi zvrstmi $\{S_1, ..., S_N\}$. Pri tem $X_i(t)$ določa število molekul zvrsti S_i v sistemu ob času t. Vektor stanj $X(t) = \{X_1(t), ..., X_N(t)\}$ opisuje stanje sistema ob času t, $X(t_0) = x_0$ pa začetno stanje sistema. Reakcije R in kemijske zvrsti S povezuje *stoihiometrična matrika* ν_{ij} velikosti $M \times N$. Vektor, ki določa spremembe koncentracij opazovanih kemijskih zvrsti ob izvedbi reakcije j imenujemo *vektor prehajanja stanj*. Definiramo ga kot: $\nu_j = (\nu_{1j}, ..., \nu_{Nj})$. Izvedba reakcije R_j tako sistem pripelje v stanje $x + \nu_j$. Vsaki reakciji R_j pripada t.i. *funkcija nagnjenosti* (angl. *propensity function*), ki jo označimo z a_j. Vrednost $a_j(x)dt$ predstavlja verjetnost, da se pri $X(t) = x$, v naslednjem infinitezimalnem intervalu $[t, t+\mathrm{d}t)$ izvede reakcija R_j. Vrednost $a_j(x)$ računamo s pomočjo naslednjih enačb:

$$\begin{array}{lll} S_i \xrightarrow{c_j} ... & a_j(x) = c_j x_i & \\ S_i + S_{i'} \xrightarrow{c_j} ... & a_j(x) = c_j x_i x_{i'}, & i \neq i' \\ & a_j(x) = \frac{1}{2} c_j (x_i - 1) x_i, & i = i' \end{array}$$

Verjetnost izvedbe reakcije R_j pri stanju sistema $X(t) = x$ v časovnem intervalu $[t+\tau,\ t+\tau+d\tau)$ označujemo s $p(\tau, j|x, t)d\tau$. Verjetnost, da se ob danem stanju v tem intervalu ne izvede nobena reakcija označujemo s $P_0(\tau|x, t)$. Verjetnostna gostota naključne spremenljivke, ki določa 'čas τ do naslednje izvedbe reakcije j je določena z enačbo:

$$p(\tau, j|x, t) = p(\tau|x, t) \times p(j|\tau, x, t). \tag{5.1}$$

S pomočjo osnovnih zakonov verjetnostnega računa lahko zapišemo:

$$p(\tau, j|x, t)d\tau = P_0(\tau|x, t) \times a_j(x)d\tau. \tag{5.2}$$

$$P_0(\tau + d\tau|x,t) = P_0(\tau|x,t) \times \left[1 - \sum_{j'=1}^{M} a_{j'}(x)d\tau\right]. \tag{5.3}$$

Z enostavno algebrično preureditvijo enačbe 5.3 in upoštevanjem, da je $a_0(x) = \sum_{j'=1}^{M} a_{j'}(x)$, dobimo:

$$P_0(\tau + d\tau|x,t) - P_0(\tau|x,t) = -P_0(\tau|x,t)a_0(x)d\tau. \tag{5.4}$$

S predpostavko, da velja $d\tau \to 0$, pridemo do enostavne diferencialne enačbe[2], katere rešitev je:

$$P_0(\tau|x,t) = e^{-a_0(x)\tau}. \tag{5.5}$$

Če vstavimo enačbo 5.5 v enačbo 5.2, dobimo:

$$p(\tau, j|x,t) = a_j(x)e^{-a_0(x)\tau}. \tag{5.6}$$

Enačba 5.6 predstavlja osnovo za stohastično simuliranje dinamike sistema reakcij. Ker je $p(\tau, j|x,t)$ produkt gostot verjetnosti spremenljivk j in τ, lahko gostoto verjetnosti $p(j|\tau,x,t)$ (glej enačbo 5.6) definiramo kot uteženo verjetnost:

$$p(j|\tau,x,t) = \frac{a_j(x)}{a_0(x)}, \tag{5.7}$$

gostoto verjetnosti $p(\tau|x,t)$ pa kot:

$$p(\tau|x,t) = a_0(x)e^{-a_0(x)\tau}. \tag{5.8}$$

Za potrebe simulacije sistema reakcij na osnovi uporabe gostote verjetnosti iz enačb 5.7 in 5.8, potrebujemo metodo za računanje (aproksimativnih) vrednosti naključnih spremenljivk, v našem primeru j in τ. Z uporabo generatorja naključnih števil in gostote verjetnosti spremenljivk j in τ, lahko dobimo pripadajoče vrednosti spremenljivk na osnovi inverzne generacijske metode (angl. *inversion generating method*), ki predstavlja eno izmed temeljnih orodij *Monte Carlo* simulacije [22].

[2] z uporabo izreka inifinitezimalnega računa: $f'(x) = \lim_{\triangle x \to 0} \frac{f(x+\triangle x) - f(x)}{\triangle x}$

Čas τ v katerem se v sistemu ne pojavi nobena reakcija, lahko izračunamo na dva načina. S podano vrednostjo $P_0(\tau|x,t)$ lahko z uporabo enačbe 5.5 vrednost τ določimo neposredno. V primeru da vrednosti $P_0(\tau|x,t)$ ne poznamo, lahko v stohastični simulaciji uporabljamo inverzno generacijsko metodo. Na ta način določimo τ z uporabo enačbe 5.9:

$$\tau = \frac{1}{a_0(x)} ln\left(\frac{1}{r_1}\right). \tag{5.9}$$

Indeks naslednje reakcije, ki se bo pojavila v sistemu, lahko izračunamo z uporabo enačbe 5.10:

$$\min\left\{ j \mid \sum_{k=1}^{j} a_k(x) \geq r_2 a_0(x) \right\}. \tag{5.10}$$

Pri tem sta r_1 in r_2 naključni števili, ki sta enakomerno porazdeljeni na intervalu $(0,1)$. Enačbi 5.9 in 5.10 predstavljata t.i. *Monte Carlo* korak v stohastičnem simulacijskem algoritmu [22, 23, 44]:

1. Definiramo začetni čas simulacije $t = t_0$.
2. Definiramo začetno stanje sistema $x = x_0$.
3. Definiramo maksimalni čas simulacije t_{max}.
4. Izračunamo vse vrednosti $a_j(x)$ in njihovo vsoto $a_0(x)$.
5. Monte Carlo korak: izračunamo τ in j na osnovi enačb 5.9 in 5.10.
6. Povečamo časovni korak $t \leftarrow t+\tau$ in osvežimo stanje sistema $x \leftarrow x+\nu_j$.
7. Zabeležimo vrednosti (x,t) in se vrnemo na korak 4, če je $t < t_{max}$.

Simulacijo je smiselno ustaviti, ko sistem doseže ravnovesno stanje.

Algoritem SSA je računsko zelo potraten, saj so vrednosti spremenljivke τ v večini primerov zelo majhne. Največja prednost SSA je eksaktno posnemanje sistema reakcij, saj njegove trajektorije niso zgolj aproksimacije, temveč natančni posnetki sistema [36, 44]. Glavni cilj simulacije je ugotoviti ali se določen sistem reakcij (kot je recimo GRO) ustali v določenem ravnovesnem

stanju ali odraža oscilatorno delovanje glede na podane začetne parametre. Pri danih omejitvah iščemo najboljše začetne koncentracije reaktantov S_i, pri katerih je delovanje sistema v skladu s predvidenim logičnim delovanjem.

5.1.2 Zakasnjeni stohastični simulacijski algoritem

V splošnem so reakcije, ki zadevajo GRO zelo kompleksne, zato na njih gledamo kot na posebne biokemijske procese. Biokemijske kompleksne procese kot sta transkripcija in translacija[3] opisuje zahtevna dinamika, ki jo še danes v celoti ne poznamo in je zato eden izmed glavnih predmetov študija na področju molekularne biologije. Predstavljanje transkripcije ali translacije z enostavnimi linearnimi kemijskimi reakcijami je zato zgolj približek, ki ga infinitezimalni račun lahko delno kompenzira. Natančno matematično opisovanje teh dveh procesov predstavlja danes pravi izziv za vse tiste, ki se ukvarjajo z modeliranjem gensko regulatornih omrežij.

Proces transkripcije v sesalskih celicah poteka znotraj jedra. Rezultat procesa, t.j. molekula mRNK, ki vsebuje zapis kodirajočega proteina, potuje skozi jedrno membrano v citoplazmi. Nato se v ribosomih prepiše (s pomočjo encima tRNK) v ciljni protein v procesu translacije. Med procesom transkripcije in translacije je prisotna zakasnitev, zaradi potrebnega prehajanja molekule mRNK iz membrane jedra v citoplazmo. Kinetika kemijskih reakcij, ki se uporablja v sistemu reakcij (iz razdelka 5.1.1) te latence ne vključuje in zato opisano dinamiko lahko (v prvem približku) samo aproksimativno modeliramo in simuliramo z algoritmom iz razdelka 5.1.1.

Opisano pomanjkljivost lahko v modelu iz razdelka 5.1.1 premostimo s t.i. *zakasnjenim stohastičnim simulacijskim algoritmom* (angl. *delayed stochastic simulation algorithm* - '*delayed SSA*') [41, 43]. To je posebna varianta algoritma SSA, pri katerem določeno stopnjo nelinearnosti dosežemo z dodajanjem zakasnjenih reakcijah ob določenih časovnih intervalih. Z uporabo posebne čakalne vrste lahko opisano zakasnitev reakcij učinkovito implementiramo. Simulacija lahko tako omogoča zelo natančno oponašanje dinamike transkripcije in translacije, poleg tega pa vsebuje vse prednosti in slabosti SSA iz razdelka 5.1.1. Algoritem lahko opišemo z naslednjimi petimi koraki:

[3] ta dva procesa sta ključna za delovanje vseh GRO

1. Definiramo začetni čas (angl. start time) $t \leftarrow 0$ in končni čas simulacije (angl. stop time) t_{stop}. Preberemo začetno število molekul in reakcij ter sestavimo prazno čakalno vrsto L.

2. Izvedemo Monte Carlo korak algoritma SSA s podanimi (novimi) vhodnimi podatki, da lahko dobimo indeks naslednje reakcije R_j, ki se vrši v času t_1 (oz. $t + \tau$).

3. Če je: $t_1 + t < t_{min}$ (najmanjši čas v L), postavimo $t \leftarrow t + t_1$. Posodobimo število molekul sistema z izvajanjem reakcije R_j in v primeru pojava zakasnitve v L dodamo pripadajoče produkte.

4. Če je: $t_1 + t \geq t_{min}$, postavimo $t \leftarrow t_{min}$. Posodobimo število molekul sistema s sprostitvijo prvega elementa v L.

5. Če je: $t < t_{stop}$ se vrnemo na točko 2.

Opisani algoritem je učinkovito implementiran v prosto dostopnem orodju *SGN Sim* (angl. *stochastic genetic networks simulator*) [42]. Na tem mestu je potrebno omeniti, da sta simulacijska algoritma SSA in zakasnjeni SSA samo dva primera možnih načinov modeliranja in simuliranja sistema reakcij. Metoda *τ-skokov* (angl. *tau-leaping*) [44, 36] predstavlja zelo zanimivo varianto algoritma SSA, saj je njena računska zahtevnost manjša v primerjavi s SSA. Poleg tega omogoča implementacijo čakalne vrste, tako kot pri zakasnjeni SSA. V primeru zelo velikih sistemov reakcij lahko implementirani SSA algoritem odpove zaradi prevelike računske zahtevnosti. Zanimiva rešitev tega problema je predlagana v [7], pri katerem za simulacijo modela izkoriščamo modularnost *p-jezika* (angl. *p-calculus*).

5.2 Modeliranje osnovnih konstruktov podatkovno pretokovnega procesiranja

V tem razdelku bomo predstavili simulacije delovanja nekaterih poglavitnih komponent grafa pretoka podatkov implementiranih v gensko regulatornih omrežjih.

5.2.1 Izhodišča pri postavitvi modelov

Realizacija *in vitro* določenega sintetičnega GRO, zahteva predčasno analizo in podrobno načrtovanje vseh potrebnih gradnikov in sestavnih delov: od vrste uporabljenih gostiteljev (evkariontske ali prokariontske celice) do vrste vektorjev oziroma plazmidov, na katere želimo vstaviti gene in vezavna mesta odgovorna za regulacijo. Poleg tega je iz množice obstoječih cinkovih prstov nujno izbrati tiste, ki zadovoljujejo največ pogojev načrtovanja. Realizacija določenega GRO zahteva natančno načrtovanje z metodološkim pristopom. Za potrebe modeliranja zadošča načrtovanje hipotetičnega modela, tako kot je prikazano na sliki 5.2. Model sestavljajo različni DNK segmenti, regulirani s sintetičnimi represorji oziroma aktivatorji (v tem primeru cinkovi prsti). Na tem mestu je potrebno opozoriti, da bomo v naših modelih uporabljali samo represorke DNK segmente. Od tu naprej bomo z besedo *DNK segment* označevali model iz slike 3.3 z enim ali dvema DNK vezavnima mestoma. V vseh predstavljenih modelih tega poglavja bomo predpostavili uporabo promotorja T7. To je promotor, ki ima izvor v genomu virusnega faga T7. Promotor T7 (v modelih je označeno kot $pT7$) ima izjemno afiniteto vezave za pripadajočo RNK polimerazo ($RNAP_{T7}$). Z njegovo uporabo lahko dosežemo zelo visoko stopnjo ekspresije. Za T7 RNK polimerazo smo predpostavili konstitutivno ekspresijo. Uporabljene cinkove prste smo označili s kratico ZNF_i.

Na DNK segment iz slike 3.3 delujejo določene osnovne reakcije, ki so v vseh DNK segmentih znotraj GRO približno enake (tu lahko opazimo veliko modularnost, ki je posebej obravnavana v [7]). Določen DNK segment, ki vsebuje samo eno DNK vezavno mesto, na katero se lahko veže represor, opišemo s sistemom reakcij prikazanih v tabeli 5.1.

Reakcije v tabeli 5.1 opisujejo delovanje logične negacije (NOT vrata), katere shemo delovanja prikazuje slika 3.3. Če privzamemo, da je ZNF_r (protein X) vhodni in ZNF_{out} (protein Y) izhodni signal vezja, lahko ugotovimo, da se ob prisotnosti visoke koncentracije proteina ZNF_r (X) v sistemu izhodni protein ne pojavi (zaradi inhibiranega promotorja $pT7znf'$). V primeru, da je v sistemu prisotna zelo nizka koncentracija represorja ZNF_r, se bo (zaradi ekspresije aktiviranega promotorja - $pT7znf^*$) koncentracija izhodnega proteina ZNF_{out} (Y) počasi povečala. Podobno delovanje poteka v DNK segmentu, ki vsebuje dve DNK vezavni mesti za dva različna represorja. Reakcije takega sistema so prikazane v tabeli 5.2. V tem primeru sistem deluje v skladu z NOR vrati. Posamezni represor (ZNF_{r_1} ali ZNF_{r_2}) predstavlja vhodni signal vezja.

	reakcija	**opis**
1:	$pT7znf + RNAP_{T7} \xrightarrow{k_{on}} pT7znf^*$	vezava T7 RNA polimeraze na prom.
2:	$pT7znf^* \xrightarrow{k_{off}} RNAP_{T7} + pT7znf$	disociacija T7 RNA polimeraze iz prom.
3:	$pT7znf + ZNF_r \xrightarrow{k_{bnd}} pT7znf^r$	vezava represorja ZNF_r ob prom.
4:	$pT7znf^r \xrightarrow{k_{unbnd}} ZNF_r + pT7znf$	disociacija represorja ZNF_r iz prom.
5:	$pT7znf^r \xrightarrow{k_{degb}} pT7znf$	razgradnja represorja ZNF_r iz prom.
6:	$pT7znf^* \xrightarrow{k_{trs}} pT7znf + RNAP_{T7} + mRNK$	transkripcija mRNK
7:	$mRNK \xrightarrow{k_{trl}} ZNF_{out} + mRNK$	translacija proteina ZNF_{out}
8:	$mRNK \xrightarrow{k_{degm}} \emptyset$	degradacija $mRNK$
9:	$ZNF_{out} \xrightarrow{k_{degp}} \emptyset$	degradacija proteina ZNF_{out}

Tabela 5.1: Tabela reakciji enostavnega DNK segmenta v GRO (NOT vrata).

Ekspresija gena se ob prisotnosti določene koncentracije vsaj enega izmed teh dveh represorjev, inhibira. Posledično se izhod ZNF_{out} ne pojavi v celici. Do sinteze izhodnega proteina ZNF_{out} lahko pride samo takrat, ko v sistemu ni prisotnih nobenih represorjev (ZNF_{r_1}, ZNF_{r_2}).

Shema modela na sliki 5.2 omogoča direktno ekstrapolacijo enačb sistema reakcij v obliki, ki je prikazana v tabeli 5.1 in 5.2. Če zelimo izvesti deterministično ali stohastično simulacijo s pomočjo reakcij iz tabel 5.1 ali 5.2, je potrebno zbrati natančne kinetične parametre posameznih reakcij. V splošnem za kinetične parametre uporabljamo kar kinetične konstante posameznih reakcij, ki nam omogočajo natančno opisovanje dinamike sistema reakcij.

Pogosto pri simulacijah potrebujemo kinetične konstante, za katere vrednosti še niso določene. To predstavlja enega izmed glavnih problemov pri uporabi kinetike kot postavke za modeliranje sistema reakcij. Pogosto je možno parametre izmeriti posredno s posebnimi laboratorijskimi metodami, na primer s *površinsko plazmonsko resonanco* (angl. *surface plasmon resonance* - SPR). Kljub temu je natančne vrednosti določenih kinetičnih konstant težko ugotoviti. To sta na primer hitrosti transkripcije in translacije. Pri tem si pomagamo z drugačnimi poznanimi vrednostmi. V reakciji transkripcije je pomembna informacija o hitrosti RNK polimeraze v odpiranju vijačnice DNK in v sestavljanju mRNK segmentov. V primeru T7 RNA polimeraze lahko ta hitrost doseže 200 b.p./sekundo (baznih parov na sekundo) [35]. Hitrost transkripcije je odvisna od mnogih faktorjev, ki jo lahko zavirajo, upočasnijo ali pospešujejo. Poleg tega lahko v sesalskih celicah izmerimo zelo različne

	reakcija	**opis**
1:	$pT7znf + RNAP_{T7} \xrightarrow{k_{on}} pT7znf^*$	vezava T7 RNA polimeraze na promotor
2:	$pT7znf^* \xrightarrow{k_{off}} RNAP_{T7} + pT7znf$	disociacija T7 RNA polimeraze iz promotorja
3:	$pT7znf + ZNF_{r_1} \xrightarrow{k_{bnd}} pT7znf^{r_1}$	vezava represorja ZNF_{r_1} na promotorju
4:	$pT7znf^{r_1} \xrightarrow{k_{unbnd}} ZNF_{r_1} + pT7znf$	disociacija represorja ZNF_{r_1} iz promotorja
5:	$pT7znf^{r_1} \xrightarrow{k_{degb}} pT7znf$	razgradnja represorja ZNF_{r_1}
6:	$pT7znf + ZNF_{r_2} \xrightarrow{k_{bnd}} pT7znf^{r_2}$	vezava represorja ZNF_{r_2} na promotorju
7:	$pT7znf^{r_2} \xrightarrow{k_{unbnd}} ZNF_{r_2} + pT7znf$	disociacija represorja ZNF_{r_2} iz promotorja
8:	$pT7znf^{r_2} \xrightarrow{k_{degb}} pT7znf$	razgradnja represorja ZNF_{r_2}
9:	$pT7znf^{r_1} + ZNF_{r_2} \xrightarrow{k_{bnd}} pT7znf^{rr}$	vezava represorja ZNF_{r_2} na inhibiranem promotorju
10:	$pT7znf^{rr} \xrightarrow{k_{unbnd}} ZNF_{r_2} + pT7znf^{r_1}$	disociacija represorja ZNF_{r_2} iz inhibiranega promotorja
11:	$pT7znf^{r_2} + ZNF_{r_1} \xrightarrow{k_{bnd}} pT7znf^{rr}$	vezava represorja ZNF_{r_1} na inhibiranem promotorju
12:	$pT7znf^{rr} \xrightarrow{k_{unbnd}} ZNF_{r_1} + pT7znf^{r_2}$	disociacija represorja ZNF_{r_1} iz inhibiranega promotorja
13:	$pT7znf^{rr} \xrightarrow{k_{degb}} pT7znf^{r_2}$	razgradnja represorja ZNF_{r_1}
14:	$pT7znf^{rr} \xrightarrow{k_{degb}} pT7znf^{r_1}$	razgradnja represorja ZNF_{r_2}
15:	$pT7znf^* \xrightarrow{k_{trs}} pT7znf + RNAP_{T7} + mRNK$	transkripcija mRNK proteina ZNF
16:	$mRNK \xrightarrow{k_{trl}} ZNF_{out} + mRNK$	translacija proteina ZNF
17:	$mRNK \xrightarrow{k_{degm}} \emptyset$	degradacija $mRNK$
18:	$ZNF_{out} \xrightarrow{k_{degp}} \emptyset$	degradacija proteina ZNF

Tabela 5.2: Tabela reakciji DNK segmenta z dvema vezavnima mestoma. Reakcije opisujejo notranje delovanje *NOR* vrat. Pri tem sta proteina ZNF_{r_1} in ZNF_{r_2}, vhodna signala, protein ZNF_{out} pa izhod vezja.

parameter	**vrednost**	**referenca**
k_{on}	56 $\mu M^{-1}s^{-1}$	[34]
k_{off}	0.2 s^{-1}	[34]
k_{bnd}	0.031 $\mu M^{-1}s^{-1}$	[55]
k_{unbnd}	0.0002 s^{-1}	[55]
k_{degb}	k_{degp}	[32]
k_{trs}	200 b.p. s^{-1}	[35]
k_{trl}	15 a.k. s^{-1}	[31]
k_{degm}	$\frac{ln\ 2}{t_{\frac{1}{2}}}\ s^{-1}$, $t_{\frac{1}{2}} = 2$ min	[17]
k_{degp}	$\frac{ln\ 2}{t_{\frac{1}{2}}}\ s^{-1}$, $t_{\frac{1}{2}} = 10$ min	[17]

Tabela 5.3: Tabela kinetičnih parametrov osnovnih reakciji iz tabele 5.1.

povprečne hitrosti v primerjavi z bakterijskimi celicami. Za potrebe simulacije je povsem zadovoljivo, če upoštevamo samo povprečno hitrost transkripcije [35]. Pri dani povprečni hitrosti 200 baznih parov na sekundo lahko aproksimativno določimo povprečno kinetično konstanto transkripcije z enačbo 5.11 [31]:

$$k_{trs} = \frac{povprečna\ \ hitrost\ \ RNK\ \ polimeraze\ \ [b.p./s]}{dolžina\ \ gena\ \ [b.p.]}. \tag{5.11}$$

Podobno velja za proces translacije. Za translacijo je težko neposredno določiti vrednost kinetične konstante k_{trl} (enačba 7 v tabeli 5.1). V večini primerov lahko konstanto ugotovimo neposredno s pomočjo hitrosti sinteze proteinov v ribosomih. Nekateri eksperimenti so izmerili hitrost: 15 a.k./s (aminokislin na sekundo) [31]. Posledično lahko izračunamo kinetično konstanto reakcije translacije po enačbi 5.12:

$$k_{trl} = \frac{povprečna\ \ hitrost\ \ sinteze\ \ proteinov\ \ v\ \ ribosomih\ \ [a.k./s]}{dolžina\ \ sintetiziranega\ \ proteina\ \ [a.k.]}. \tag{5.12}$$

Z uporabo vrednosti iz tabele 5.3 in iz podatka o dolžini kodirajočega gena[4] lahko določimo približno vrednost kinetičnih konstant reakcij transkripcije in translacije na podlagi enačb 5.11 in 5.12:

$$k_{trs} = \frac{200\ \ b.p./s}{350\ \ b.p.} = 0.571\ \ s^{-1}, \tag{5.13}$$

[4]v povprečju so geni cinkovih prstov kodirani, z zaporedjem 350 baznih parov [51]

$$k_{trl} = \frac{15\ a.k./s}{116\ a.k.} = 0.129\ s^{-1}. \tag{5.14}$$

Zanimiva lastnost, ki jo imajo sintetični GRO, je podobnost kinetike vseh uporabljenih represorjev oziroma cinkovih prstov. Iz tega sledi, da lahko uporabimo iste kinetične parametre za vse cinkove prste (represorje), ki sodelujejo v GRO. To omogoča gradnjo stabilnih in predvsem zelo robustnih modelov, kakršnih so prikazanih na slikah 5.2, 5.5, 5.7 in 5.9.

5.2.2 Stohastična simulacija modelov

V razdelku 2.1.1 smo spoznali osnovne komponente grafa pretoka podatkov, med temi tudi kontrolne strukture (oziroma operatorje) *merge* in *switch*. Če želimo implementirati osnovne kontrolne strukture PP procesiranja v sintetičnih GRO, je potrebno njihovo delovanje testirati s pomočjo simulacijskih modelov. V nadaljevanju bomo modelirali sledeče logične strukture:

1. operator združevanja (*merge*)
2. preklopni operator (*switch*)
3. logični operator XOR
4. eno-bitni polni seštevalnik

Modeliranje se prične z minimizacijo logične funkcije, ki jo želimo realizirati. Naslednji korak zahteva zbiranje potrebnih shem in modelov GRO komponent, DNK segmentov in logičnih vrat, ki jih potrebujemo za realizacijo predvidene logične funkcije. V našem primeru potrebujemo množico DNK segmentov cinkovih prstov z njihovimi enojnimi ali dvojnimi vezavnimi domenami (odvisno od zahteve logične funkcije)[5]. V naslednjem koraku preuredimo DNK segmente in po potrebi dodamo vezavne domene tistim segmentom, ki ne izpolnjujejo zahteve načrtovanja. Rezultat tega postopka je model, ki je podoben tistemu na sliki 5.2. Iz postavljenega modela DNK segmentov, je potrebno ekstrapolirati sistem reakcij, podobno kot v tabelah 5.1 in 5.2, za vse DNK

[5]DNK segment z dvojnim vezavnim mestom implementira NOR vrata, medtem ko DNK segment z enojnim DNK vezavnim mestom implementira negacijo

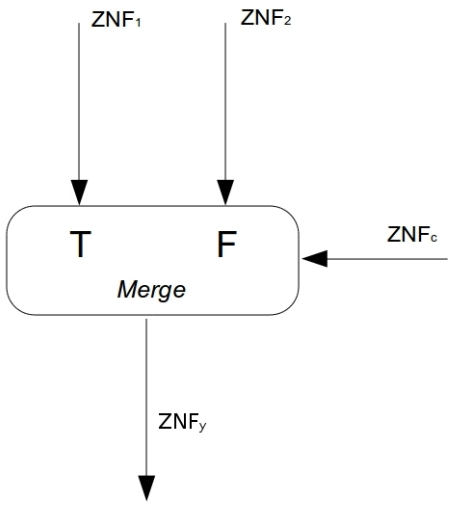
segmente v modelu. V naslednjem koraku je potrebno pridobiti pripadajoče kinetične konstante. Glede na to kar smo povedali v prejšnjem razdelku lahko kinetiko močno poenostavimo, če se odločimo za uporabo podobnih vezavnih proteinov v vseh DNK segmentih sistema. Po pridobljenih kinetičnih parametrih lahko izvedemo stohastično simulacijo. V razdelku 5.1 smo spoznali dva zelo uspešna algoritma za stohastično simuliranje. Odločili smo se za realizacijo oziroma modeliranje teh komponent v okolju sesalskih celic, saj je zaradi obširnih modelov (recimo eno-bitni polni seštevalnik ali *merge* operator) implementacija omenjenih operatorjev v prokariontskih celicah rizična oz. težko izvedljiva. Modeliranje GRO v sesalskih celicah zahteva implementacijo zakasnjenih reakcij translacije. To zahtevo lahko implicitno izpolnimo, če se odločimo za uporabo zakasnjenega stohastičnega simulacijskega algoritma. Za simulacijo smo zato uporabili orodje *SGN Sim*, ki smo ga omenili v razdelku 5.1. Podrobnosti o načinu programiranja ne bomo obravnavali. Bralec lahko dobi vse specifikacije in primere uporabe v dodatku članka [42].

V nobenem izmed simuliranih modelov, nismo predpostavili kooperativnost vezave represorjev na pripadajoča vezavna mesta. Čeprav je ta predpostavka nujno potrebna za pravilno modeliranje nekaterih GRO na osnovi naravnih represorjev [8] je v primeru uporabe cinkovih prstov (kot umetnih represorjev) odvečna.

5.2.3 Operator *merge*

Prvi operator grafa pretoka podatkov, ki ga želimo realizirati v GRO je operator združevanja ali *merge*. Shema operatorja v GPP prikazuje slika 5.1.

Delovanje tega operatorja je enostavno: če je logična vrednost kontrolnega signala ZNF_c enaka "1", se bo na izhodu ZNF_y pojavila logična vrednost vhodnega signala ZNF_1. Če je logična vrednost signala ZNF_c enaka "0", bo na izhod konvergirala logična vrednost vhodnega signala ZNF_2. Če razmišljamo o koncentracijah vhodnih in izhodnih proteinov, potem si lahko delovanje predstavljamo na naslednji način: če je koncentracija kontrolnega proteina ZNF_c dovolj visoka, bo koncentracija izhodnega proteina ZNF_y dosegla približno enako vrednost kot koncentracija vhodnega proteina ZNF_1; če pa koncentracija kontrolnega proteina ZNF_c dovolj nizka, bo koncentracija izhodnega proteina ZNF_y dosegla približno enako vrednost kot koncentracija vhodnega proteina ZNF_2.



Slika 5.1: Kontrolni operator *merge*.

ZNF_1	ZNF_2	ZNF_c	ZNF_y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

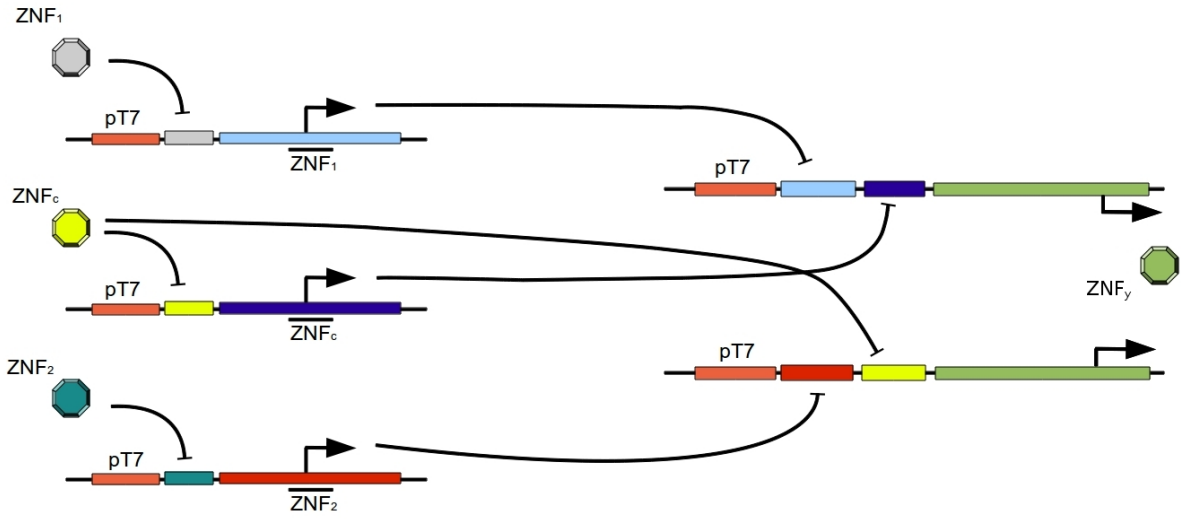
Tabela 5.4: Pravilnostna tabela operatorja *merge*.

Operator *merge* tako ni nič drugega kot enostavni 2/1 multiplekser (MUX 2/1). Pravilnostna tabela logične funkcije operatorja *merge* prikazuje tabela 5.4. Logično enačbo operatorja lahko izenačimo z enačbo 2/1 multiplekserja:

$$ZNF_y \;=\; ZNF_1\, ZNF_c \;\vee\; ZNF_2\, \overline{ZNF_c} \tag{5.15}$$

Če želimo v GRO implementirati logično funkcijo iz enačbe 5.15, potrebujemo torej dvoje AND vrat in en negator. OR vrat ne potrebujemo, saj lahko

dosežemo isti učinek z uporabo istega izhodnega proteina na več DNK segmentih hkrati. V razdelku 5.2.1 smo spoznali, da DNK segmenti, ki jih imamo na razpolago (z enim ali dvema represorskima vezavnima mestoma), omogočajo implicitno implementacijo NOT in NOR vrat. Za realizacijo AND vrat imamo torej na voljo logične konstrukte NOR in NOT. AND vrata dobimo, če vhode NOR vrat negiramo. Če vhoda DNK segmenta, katero dinamiko opisuje tabela 5.2 (to sta represorja ZNF_{r_1} in ZNF_{r_2}), povežemo z izhodi dveh DNK segmentov, katere reakcije opisuje tabela 5.1, potem dobljeno omrežje DNK segmentov predstavlja logično funkcijo AND. Opisani postopek je implementiran tudi na sliki 5.2, ki si jo bomo bolj podrobno ogledali. V tem primeru je GRO sestavljen iz pet DNK segmentov[6].

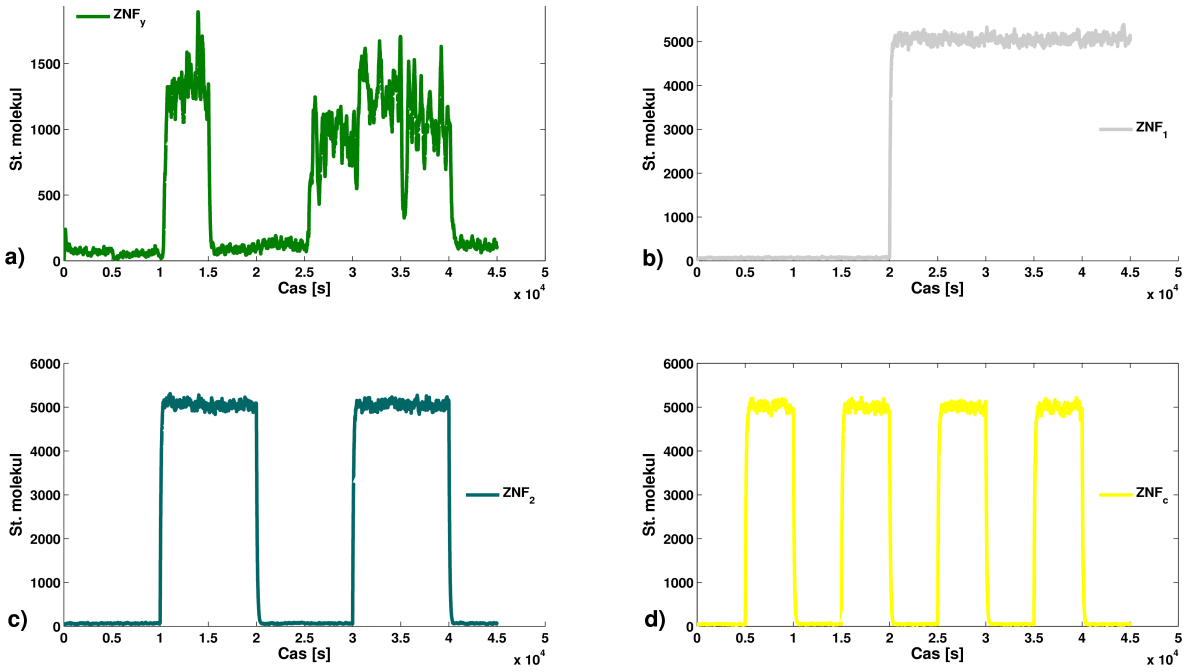


Slika 5.2: Gensko regulatorno omrežje, ki implementira logično funkcijo kontrolnega operatorja *merge* (enačba 5.15). Zaradi preglednosti na sliki niso prikazani konstitutivni geni, ki sodelujejo v regulaciji omrežja.

Izhodna signala $\overline{ZNF_1}$ in $\overline{ZNF_c}$ predstavljata negacijo signalov ZNF_1 in ZNF_c. Negirana signala delujeta kot represorja in torej kot vhodna signala v prvi DNK

[6]Segment, na katerem poteka konstitutivno izražanje T7 RNK polimeraze smo zaradi preglednosti iz slike izpustili. Izražanje T7 RNK polimeraze običajno poteka pod različnim promotorjem. Zelo pogosto se uporablja promotor CMV. Ta v nasprotju s T7 promotorjem izhaja iz virusa *Cytomegalovirus*. DNK segment določenega konstitutivnega proteina ne vsebuje nobenih vezavnih mest za represorje kot v primeru slike 3.3.

segment. Ta DNK segment vsebuje dve vezavni represorski mesti za inhibicijo ekspresije gena, ki kodira informacijo za sintezo izhodnega signala t.j. ZNF_y. Koncentracija ZNF_y v omrežju se regulira po logiki AND vrat. ZNF_y se močno izraža, kadar sta koncentracij vhodnih proteinov ZNF_1 in ZNF_c tolikšni, da se zaradi tega ekspresija proteinov $\overline{ZNF_1}$ in $\overline{ZNF_c}$ popolnoma utiša. V primeru, da je koncentracija vsaj enega izmed vhodnih proteinov ničelna (oziroma zelo nizka), se v pripadajočem DNK segmentu ekspresija poveča, kar se posledično kaže v utišanju ekspresije ZNF_y. Podobno delovanje imajo AND vrata z vhodnimi proteini ZNF_2 in ZNF_c in z istim izhodnim proteinom ZNF_y. Z uporabo NOR in NOT vrat kot v zgornjem primeru lahko teoretično realiziramo poljubno logično funkcijo[7].



Slika 5.3: Potek stohastične simulacije *merge* operatorja. Slika a) prikazuje odziv gensko regulatornega omrežja iz slike 5.2 na vhodne testne signale ZNF_1, ZNF_2 in ZNF_c, ki so prikazani na slikah b), c) in d).

Delovanje GRO iz slike 5.2 prikazuje graf a) na sliki 5.3. Če interpretiramo

[7]pravzaprav lahko to dosežemo tudi z izključno uporabo NOR vrat, saj ta predstavlja funkcijski poln nabor

nizke koncentracije posameznih proteinov kot logično vrednost "0", visoke koncentracije posameznih proteinov pa kot logično vrednost "1", potem lahko takoj ugotovimo, da graf na sliki 5.3 predstavlja pravilni potek pravilnostne tabele 5.4.

Iz modela na sliki 5.2 lahko ekstrapoliramo enačbe reakcij sistema v obliki tabel 5.1 in 5.2. Orodje *SGN Sim* omogoča njihov neposreden vnos skupaj s kinetičnimi parametri. Če želimo preveriti delovanje modela pri vseh možnih kombinacijah vhodnih signalov, je potrebno v simulacijo vključiti vse možne testne primere vhodnih signalov. To lahko v orodju *SGN Sim* implementiramo z ukazom *queue*, ki ob točno določenem času simulacije, spremeni vrednosti vhodnih testnih kombinacij. V našem primeru je potrebno testirati vse možne kombinacije spremenljivk ZNF_1, ZNF_2 in ZNF_c. Za potrebe testiranja logične funkcije operatorja *merge* je teh kombinacij 2^3. Običajno jih simuliramo tako, da sledimo kombinacijam, ki so zapisane v pravilnostni tabeli.
SGN Sim uporablja za izvajanje simulacij sistema reakcij določenega GRO modela zakasnjeni SSA algoritem. Orodje omogoča zelo učinkovito upravljanje z zakasnjenimi reakcijami, kot je recimo translacija. Posameznim reakcijam lahko poleg običajnih vrednosti kinetičnih konstant, dodelimo posebno porazdelitveno funkcijo ali konstanto s katero povemo, kdaj naj se v simulacij pojavi določen produkt v okolju reakcij. V našem primeru opisana lastnost omogoča učinkovito simuliranje reakcij transkripcije in translacije. Z uporabo teh načel lahko simuliramo poljubno število reakcij, v našem primeru tudi več kot 50 reakcij, ki jih predstavlja model na sliki 5.2. Simulacijo lahko izvedemo po postavljanju simulacijskih parametrov kot je časovni korak (v našem primeru smo uporabljali časovni korak dolžine ene sekunde) ali dolžina simulacije (število časovnih korakov). V primeru uporabe modelov reakcij iz tabel 5.1 in 5.2 ter uporabe kinetičnih konstant oziroma parametrov iz tabele 5.3, dobimo rezultat[8] prikazan na sliki 5.3. Glavna slabost orodja *SGN Sim* je

[8]pri stohastičnih simulacijah je pogosto zahtevano, da se prikažejo vhodne in izhodne koncentracije v številu molekul. V našem primeru imamo v tabeli 5.3 na voljo dva podatka, ki sta podana z molarnostjo. Če želimo parameter 56 $\mu M^{-1}s^{-1}$ spremeniti v format "$molekul$"$^{-1}s^{-1}$ potrebujemo podatek o volumnu okolja Ω, v katerem se vršijo reakcije. Pretvorba poteka na naslednji način:

$$k\left[\text{“}molekul\text{“}^{-1}s^{-1}\right] = \frac{k \quad \left[M^{-1}s^{-1}\right]}{N_A\,[mol^{-1}] \quad \times \quad \Omega\,[dm^3]}$$

kjer: M = mol/dm^3 (molarnost), N_A = Avogadrovo število ($6.022 \times 10^{23} mol^{-1}$), Ω = volumen celice ($2 \times 10^{-15} dm^3$) [36]

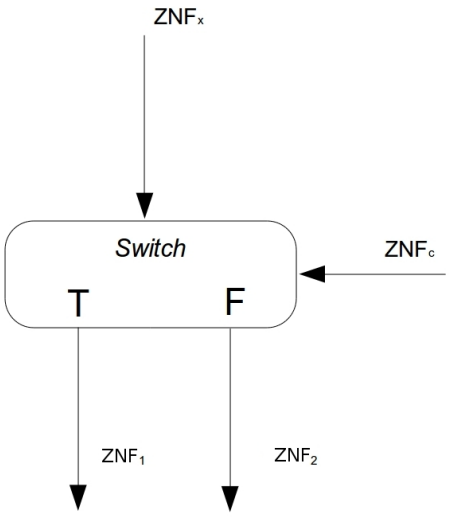
nepreglednost kode pri velikem številu reakcij.

V primeru, da bi želeli v modelu 5.2 uporabiti reporter (namesto cinkovega prsta ZNF_y) na primer zeleni fluorescentni protein - GFP (angl. *green fluorescent protein*), bi morali simulacijo ponovno izvesti, saj ima molekula GFP povsem drugačne kinetične konstante. GFP se po stabilnosti (ali po razpolovni dobi) razlikuje od cinkovih prstov in tudi pri konstantah razgradnje.

Operator *merge* igra v grafu pretoka podatkov pomembno vlogo saj z njim omogočamo izvajanje določenega dela grafa na račun drugega glede na logično vrednost preverjenega pogoja. Skupaj z njegovim komplementarnim operatorjem (*switch*) omogoča implementacijo pogojnih stavkov in torej tudi zank.

5.2.4 Operator *switch*

Drugi operator grafa pretoka podatkov, ki ga želimo realizirati v GRO je preklopni operator (*switch*). Shemo operatorja prikazuje slika 5.4.



Slika 5.4: Kontrolni operator GPP, tipa *switch*.

Delovanje operatorja *switch* je preprosto: če je logična vrednost signala ZNF_c enaka "0", se logična vrednost vhodnega signala ZNF_x prepiše na izhodni signal ZNF_1. V primeru, da je logična vrednost signala ZNF_c enaka "1",

se na izhodni signal ZNF_2 prepiše logična vrednost vhodnega signala ZNF_x. Podobno kot pri operatorju *merge* lahko zapišemo pravilnostno tabelo (tabela 5.5) s pripadajočimi logičnimi funkcijami (enačbi 5.16 in 5.17).

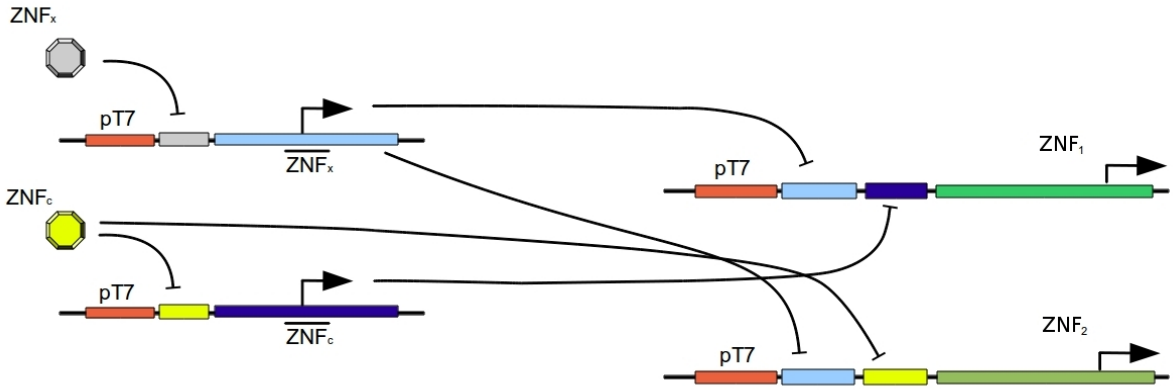
ZNF_x	ZNF_c	ZNF_1	ZNF_2
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0

Tabela 5.5: Pravilnostna tabela operatorja *switch*.

$$ZNF_1 = ZNF_x \, ZNF_c \tag{5.16}$$

$$ZNF_2 = ZNF_x \, \overline{ZNF_c} \tag{5.17}$$

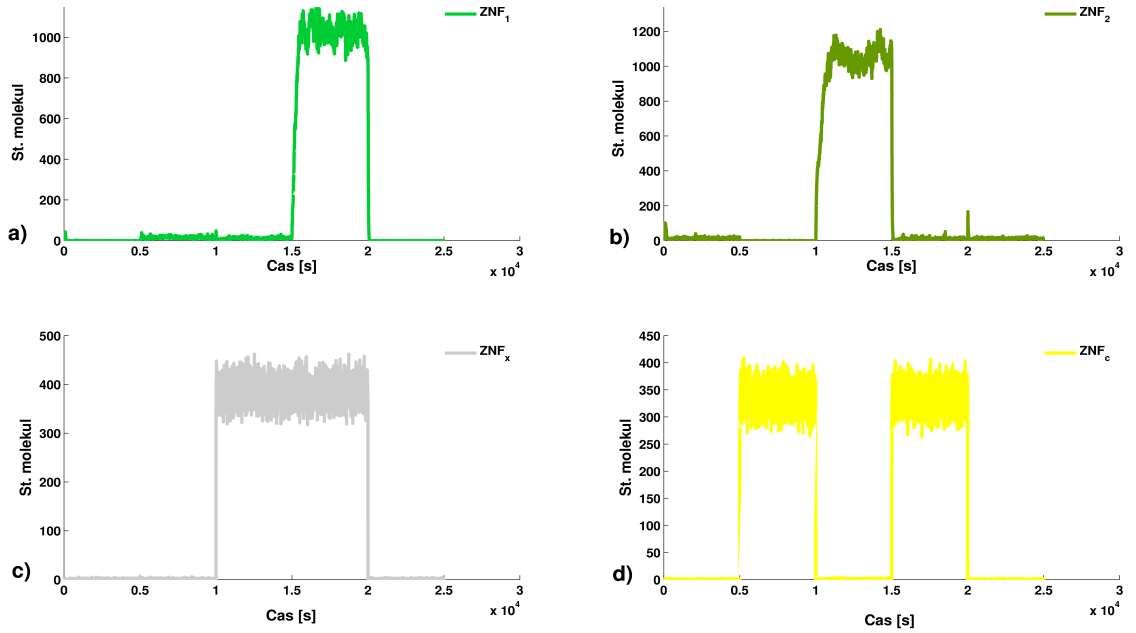
Iz enačb 5.16 in 5.17 vidimo, da tudi v tem primeru potrebujemo AND vrata za realizacijo logične funkcije. Zato tudi tokrat uporabimo dva DNK segmenta z dvema vezavnima mestoma za vhodne cinkove prste. Gensko regulatorno omrežje, ki implementira enačbi 5.16 in 5.17 prikazuje slika 5.5.



Slika 5.5: Gensko regulatorno omrežje kontrolnega operatorja *switch*.

Iz slike 5.5 lahko, podobno kot pri operatorju *merge*, ekstrapoliramo enačbe reakcij in simuliramo njihovo delovanje z orodjem *SGN Sim*. V tem primeru

je realizacija enostavnejša, saj za model iz slike 5.5 potrebujemo en DNK segment manj v primerjavi z modelom na sliki 5.2.



Slika 5.6: Potek stohastične simulacije delovanja operatorja *switch*. Na slikah a) in b) sta prikazana časovna odziva modela iz slike 5.5 oziroma spremembe v koncentracij izhodnih proteinov ZNF_1 in ZNF_2. Sliki c) in d) pa prikazujeta vhodna testna signala, ki smo ju uporabljali za ugotavljanje časovnih odzivov sistema iz slike 5.5. Iz slik a) in b) lahko razberemo, da model deluje po pravilnostni tabeli 5.5.

Na sliki 5.6 so prikazani rezultati simulacij. Simulacije smo izvajali pri istih pogojih, kot pri operatorju *merge*. Model iz slike 5.5 pravilno deluje za širok spekter uporabljenih parametrov, kar potrjuje robustnost modelov na osnovi kinetike sintetičnih cinkovih prstov.

5.2.5 Operator XOR

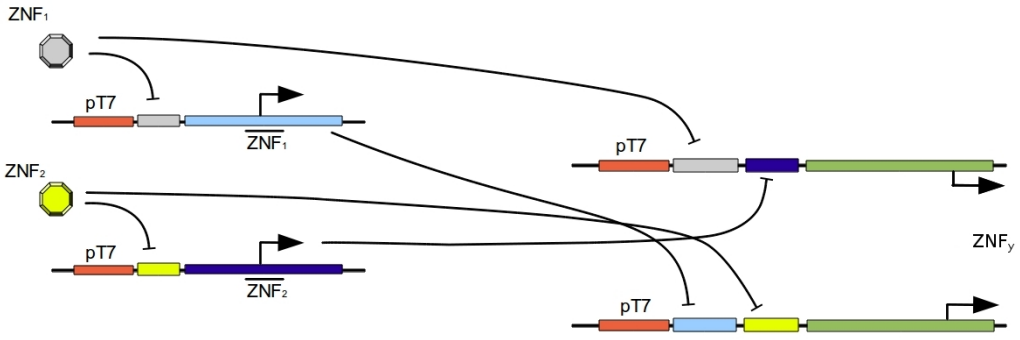
XOR predstavlja enega izmed temeljnih logičnih vezij moderne elektronike. Za razvoj računalniškega sistema na osnovi PP procesiranja in GRO, je realizacija tega operatorja skoraj obvezna. XOR je temeljni operator za operacije seštevanja in odštevanja binarnih števil, zato mu je na tem mestu potrebno posvetiti posebno pozornost.

ZNF_1	ZNF_2	ZNF_y
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 5.6: Pravilnostna tabela operatorja *XOR*.

$$ZNF_y = ZNF_1 \veebar ZNF_2 \tag{5.18}$$

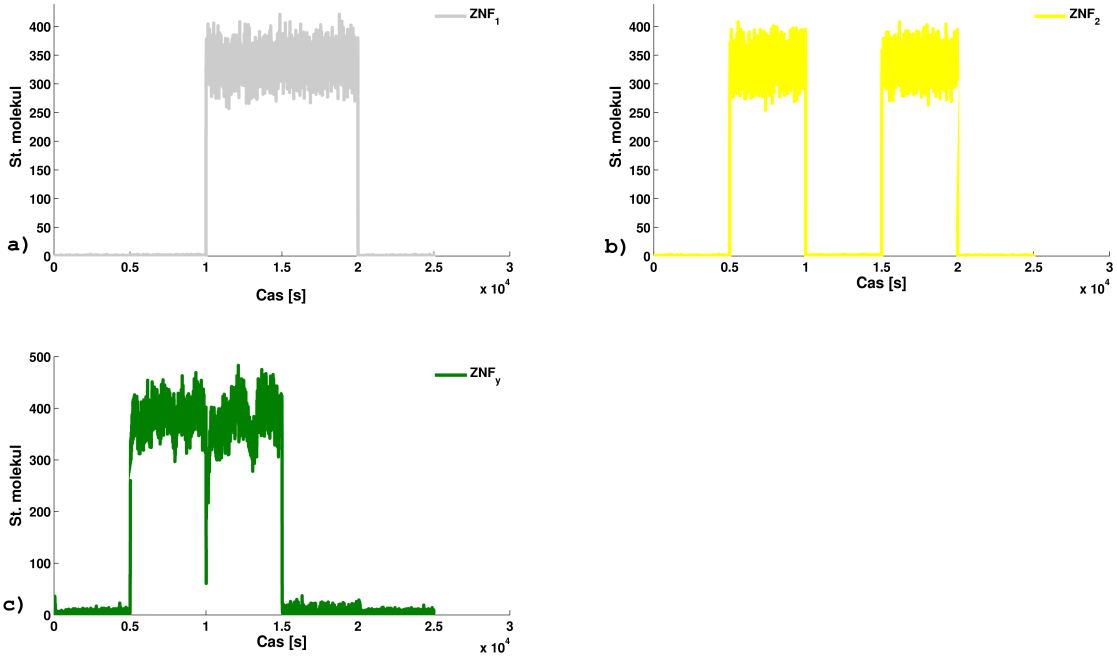
Delovanje XOR vrat prikazuje pravilnostna tabela 5.6 in logična enačba 5.18. Realizacija XOR vrat v gensko regulatornih omrežij je posebno tema, ki je vzbudila veliko zanimanje raziskovalcev v zadnjem desetletju. Bralec si lahko o modelih možne implementacije XOR operatorja v GRO prebere v [16]. V modelu smo se odločili za uporabo izključno sintetičnih transkripcijskih represorjev (cinkovi prsti). Podobno kot pri operatorjih *merge* in *switch* lahko logično enačbo operatorja implementiramo v model z uporabo NOR in NOT vrat. Gensko regulatorno omrežje, ki oponaša logično delovanje XOR operatorja prikazuje slika 5.7.



Slika 5.7: Gensko regulatorno omrežje logičnega operatorja XOR.

Tudi v tem primeru smo za potrebe simulacije uporabljali vhodne testne signale kot v primeru *switch* operatorja. Rezultat simulacije modela iz slike 5.7,

z uporabo podobne kinetike iz tabele 5.3 v orodju *SGN Sim*, prikazuje slika 5.8.



Slika 5.8: Potek stohastične simulacije XOR operatorja iz modela 5.7. Slika c) prikazuje časovni odziv pri simulacij s testnima vhodnima signaloma, ki jih prikazujeta sliki a) in b). Simulacije kažejo veliko robustnost modela ne glede na spremembe, ki jih lahko opravimo v vhodnih (začetnih) parametrih.

5.2.6 Eno-bitni polni seštevalnik

Kot primer uporabe osnovnih logičnih vezij v bolj kompleksnih logičnih strukturah, smo se odločili za primer eno-bitnega polnega seštevalnika. Logični enačbi 5.19 in 5.20 prikazujeta minimizacijo eno-bitnega polnega seštevalnika z uporabo XOR vrat. Logične funkcije so povzete po pravilnostni tabeli 5.7.

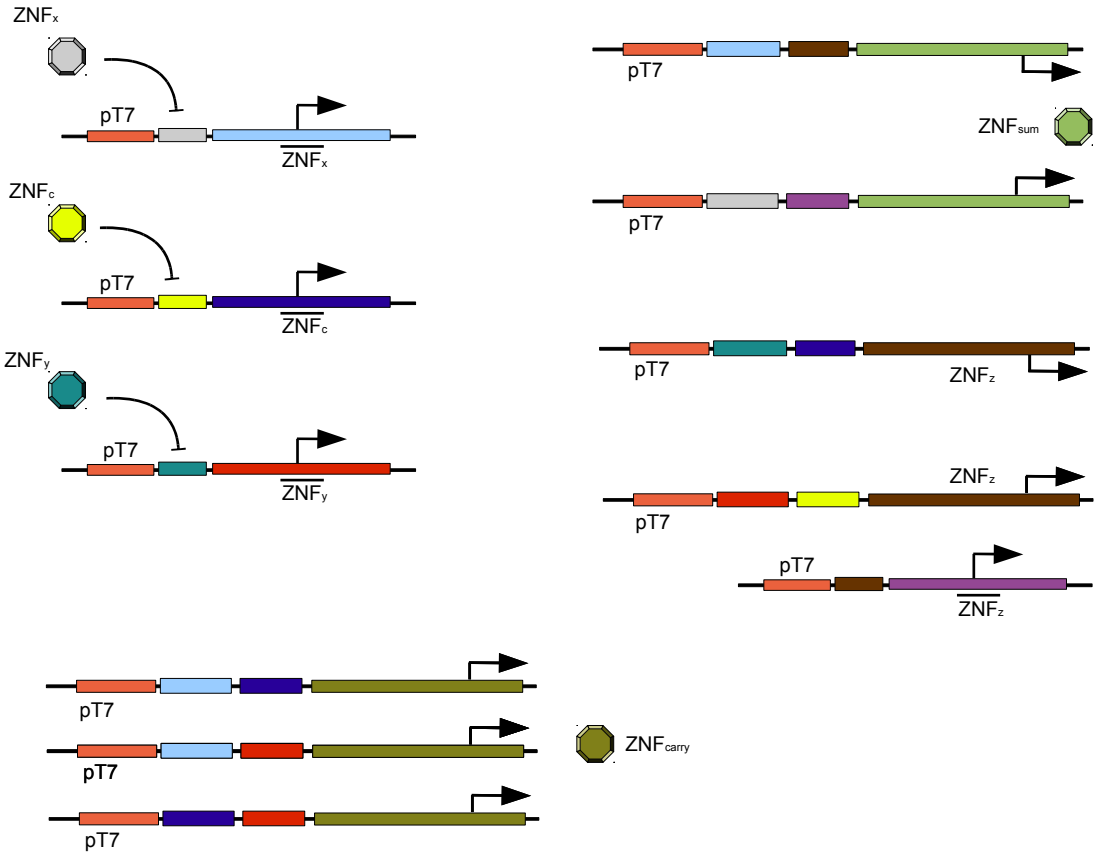
$$ZNF_{sum} = ZNF_x\,\overline{(ZNF_y \veebar ZNF_c)} \vee \overline{ZNF_x}\,(ZNF_y \veebar ZNF_c) \tag{5.19}$$

$$ZNF_{carry} = ZNF_x\,ZNF_y \vee ZNF_x\,ZNF_c \vee ZNF_y\,ZNF_c \tag{5.20}$$

Ena izmed možnih implementaciji eno-bitnega polnega seštevalnika z gensko regulatornimi omrežiji je prikazana na sliki 5.9. Z orodjem *SGN Sim* smo

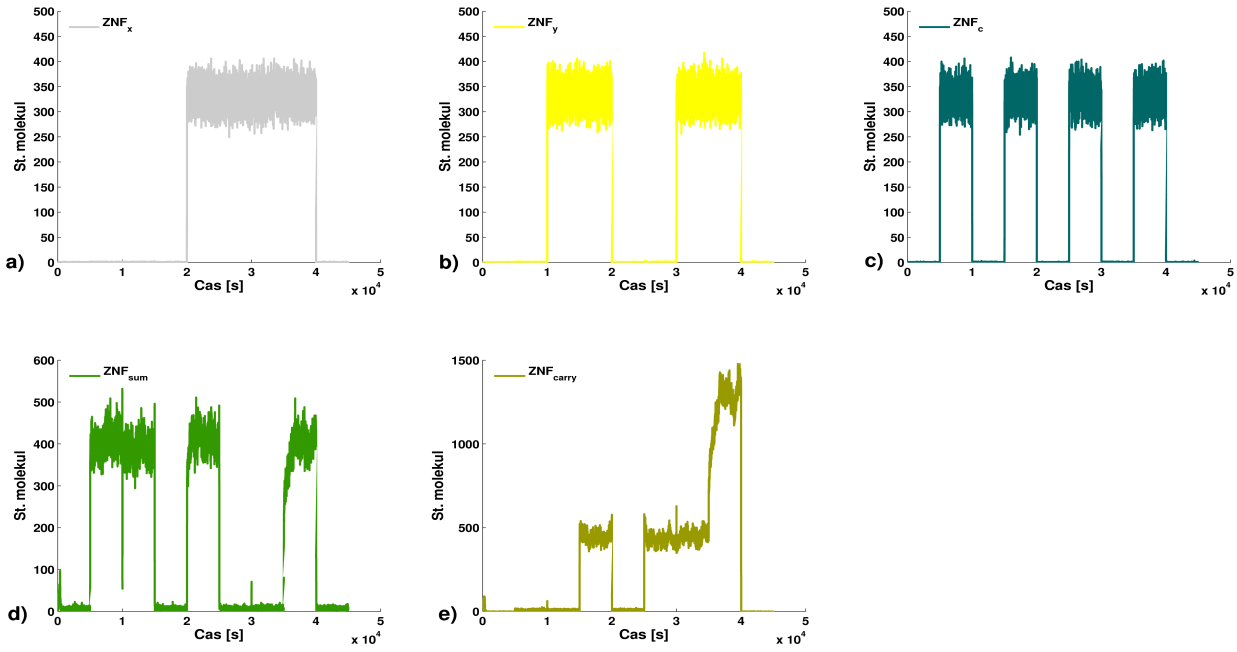
ZNF_x	ZNF_y	ZNF_c	ZNF_{sum}	ZNF_{carry}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabela 5.7: Pravilnostna tabela eno-bitnega polnega seštevalnika.



Slika 5.9: Gensko regulatorno omrežje eno-bitnega polnega seštevalnika.

simulirali delovanje s testiranjem vseh možnih testnih kombinacij vhodnih signalov. Rezultati simulacije konvergirajo v grafih na sliki 5.10.



Slika 5.10: Potek stohastične simulacije modela eno-bitnega polnega seštevalnika iz slike 5.9. Sliki d) in e) predstavljata odziva seštevalnika pri vzbujanju s testnimi signali iz grafov a), b) in c). Zanimiv efekt se pojavi na sliki e). Ker se za implemntacijo OR vrat uporabljajo trije vzporedni DNK segmenti, lahko pride do ekspresije proteinov vseh treh DNK segmentov naenkrat. To lahko povzroči skoraj tri-kratno povečane koncentracije izhodnih proteinov v GRO.

Poglavje 6

Zaključek

V tej diplomski nalogi smo pregledali glavne lastnosti modelov PP arhitektur. Izpostavili smo glavne prednosti, ki jih ima PP procesiranje napram klasičnemu von Neummanovemu modelu računanja. Povedali smo, da je velika večina realiziranih prototipov podatkovno pretokovnih računalnikov v primerjavi z enakovrednimi von Neummanovi računalniki nekoliko počasnejša, predvsem pa veliko dražja. Izvor te slabosti smo identificirali v zahtevni realizaciji nekaterih ključnih komponent PP arhitektur z moderno mikroelektronsko tehnologijo, na kateri temelji realizacija današnjih računalnikov. V tretjem poglavju smo spoznali osnovne lastnosti gensko regulatornih omrežij in možnosti realizacije logičnih struktur na biološki osnovi. Razvoj procesne platforme na nivoju GRO smo hipotetično nakazali v četrtem poglavju s predstavitvijo in uporabo nekaterih analogij GRO in PP procesiranja. V petem poglavju smo ob predpostavki uporabe sintetičnih DNK vezavnih proteinov kot ključnih gradnikov za realizacijo logičnih vezij v GRO, predstavili implementacijo nekaterih osnovnih komponent oziroma operatorjev, ki jih uporabljamo v PP procesiranju. Povedali smo tudi, da lahko z uporabo zakasnjenega stohastičnega simulacijskega algoritma učinkovito modeliramo GRO. Ta algoritem smo zato tudi uporabili za simulacijo vzpostavljenih modelov.

Uporaba sintetičnih DNK vezavnih proteinov je ključnega pomena za snovanje *digitalnih GRO* oziroma digitalne procesne platforme na osnovi GRO. Zamisel potrebuje še veliko eksperimentalnih dokazov, saj so za sintetične DNK vezavne proteine (cinkovi prsti) fizične implementacije še v fazi razvoja [21].

V diplomski nalogi smo predlagali načrtovanje relativno enostavnih sintetičnih GRO, v katerih deluje naenkrat največ deset sintetičnih DNK vezavnih pro-

teinov. Pomembno za razvoj PP procesiranja na osnovi GRO je testiranje bolj kompleksnih struktur kot so povezani moduli iz četrtega poglavja. Za načrtovanje tako obširnih struktur bi morali razširiti delovanje GRO na večcelične modele kar zahteva uporabo učinkovite medcelične komunikacije. Za trenutno stanje tehnologije to predstavlja zahteven problem. Zaenkrat se moramo omejiti na relativno majhne strukture, ki so zmožne procesiranja informacij. Po drugi strani je lahko zamisel o PP procesiranju prvi korak do konstrukcije univerzalnega računskega stroja na podlagi GRO v bioloških sistemih.

Menim, da je za nadaljnje delo potreben razvoj formalnega jezika za uporabo PP procesiranja v sklopu GRO. Formalni jezik, ki bi temeljil na modelu delovanja GRO, bi omogočil lažje načrtovanje in strukturiranje robustnejših modelov.

Slike

Tabele

Literatura

[1] Arvind, V. Kathail, "A Multiple Processor Dataflow Machine that Supports Generalized Procedures", *Computation Structures Group Memo 205, MIT*, Cambridge, MA, June 1981

[2] Arvind, D.E. Culler, "Why Dataflow Architectures", *Computation Structures Group Memo 229-1, MIT*, Cambridge, MA, Sept. 1983

[3] Arvind, D.E. Culler, "Dataflow Architectures", *Annual review of computer science*, Palo Alto, CA, 1986, vol. 1

[4] Arvind, D.E. Culler, K.Ekanadham, "The Price of Asynchronous Parallelism: An Analysis od Dataflow Architectures", *Computation Structures Group Memo 278, MIT*, Cambridge, MA, June 1988

[5] Arvind, R.S. Nikhil, "Executing a Program on the MIT Tagged-Token Dataflow Architecture", *Computation Structures Group Memo 278, MIT*, Cambridge, MA, June 1988

[6] Arvind, "Well Behaved Dataflow Graphs", 2006, MIT, Lectures slides from Multithreaded Parallelism, Dostopno na: http://csg.csail.mit.edu/6.827/handouts

[7] R. Blossey, L. Cardelli, A. Phillips, "Compositionality, stochasticity, and cooperativity in dynamic models of gene regulation", *HFSP Journal*, published online 14 November 2007. Dostopno na: http://hfspj.aip.org/

[8] H.S. Booth, C.J. Burden, M.Hegland, L. Santoso, "Stochastic Model of Gene Regulation Using the Chemical Master Equation", *Mathematical Modeling of Biological Systems*, 2007, Vol. 1/A, str. 71-81

[9] D.E. Culler, G.M. Papadopoulos, "The Explicit Token Store", *Journal of Parallel and Distributed Computing*, 1990, vol. 10/4, str. 289-308

[10] B. Cvetković, "Analiza možnosti realizacije primitivnih računalniških struktur na osnovi DNK gradnikov", *Diplomska naloga*, Ljubljana, 2008

[11] J.W. Dale, M. von Schantz, "From Genes to Genome - Concepts and Application of DNA Technology", *John Wiley & Sons, Ltd.*, 2007

[12] J.B. Dennis, "The Varieties of Data Flow Computers", *Computation Structures Group Memo 183-1, MIT*, Cambridge, MA, August 1979

[13] J.B. Dennis, "Data Flow Computer Architecture", *Computation Structures Group Memo 174*, March 1979, MIT, Laboratory for Computer Science, NSF research. Dostopno na: http://csg.csail.mit.edu/pubs/publications.html

[14] J.B. Dennis, W.B. Ackerman, "VAL – A Value-Oriented Algorithmic Language: Preliminary Reference Manual", *Technical Report 218*, Laboratory for Computer Science, MIT, Cambridge, MA, June, 1979

[15] J.B. Dennis, W.Y. Lim, W.B. Ackerman, "The MIT Data Flow Engineering Model", *Computation Structures Group Memo 222*, November 1982, MIT, Laboratory for Computer Science, NSF research. Dostopno na: http://csg.csail.mit.edu/pubs/publications.html

[16] Davidson-Missouri Western team, "DNA Encoded XOR Gates", *International Genetically Engineered Machine Competition - iGEM*, 2008, MIT, Cambridge, MA, dostopno na: http://2008.igem.org

[17] M.B. Elowitz, S. Leibler, "A synthetic oscillatory network of transcriptional regulators", *Nature*, 2000, Vol. 403, str. 335-338

[18] R.A. Finkel,, "Advanced Programming Language Design", *Addison-Wesley Computer Science*, Dec. 1995, pogl. 6, str. 169-183

[19] J. Fischer, T.A. Henzinger, "Executable cell biology", *Nature Biotechnology*, Vol.25/11, str. 1239-1249

[20] A. Formella, W. Massonne, W.J. Paul, "Cost effectiveness of data flow machines and vector processors", *Lectures Notes in Computer Science*, 1993, Vol. 678, str. 48-65

[21] R. Gaber, "Uporaba sintetičnih DNK vezavnih proteinov za procesiranje informacij in usmerjanje procesov v bioloških sistemih" *Predlog doktorske disertacije* , 2011, v pripravi

[22] D. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions", *Journal of Computational Physics*, 1976, Vol.22/4, str. 403-434

[23] D. Gillespie, "Exact stochastic simulation of coupled chemical reactions", *Journal of Physical Chemistry*, 1977, Vol.81, str. 2340-2361

[24] J.R. Gurd, C.C. Kirkham, I. Watson, "The Manchester Prototype Data-Flow Computer", *Commun. ACM*, Jan. 1985, Vol. 28, str. 34-52

[25] J. Hicks, D. Chiou, B.S. Ang, Arvind, "Performance Studies of the Monsoon Dataflow Processor", *Journal of Parallel and Distributed Computing*, July 1993, vol. 18/3, str. 273-300

[26] P.R. Jelenkovic, "Systems Biology: Design Principles of Biological Circuits", prosojnice iz predavanj, dostopno na: http://www.ee.columbia.edu/~predrag/E6010_Course_Introduction.pdf

[27] M. Kaern, W.J. Blake, J.J. Collins, "The Engineering of Gene Regulatory Networks", *Annu. Rev. Biomedical Engineering*, 2003, Vol. 5, str. 179-206

[28] D. Kodek, "Arhitektura in organizacija računalniških sistemov", *Bi-Tim*, Ljubljana 2008

[29] B. Lee, A.R. Hurson, "Issues in Dataflow Computing", *Advances in Computers*, 1993, Vol.37, str. 285-333

[30] J. Li, C. Attila, L. Wang, T.K. Wood, J.J. Valdes, W.E. Bentley, "Quorum Sensing in *Escherichia coli* Is signaled by AI-2/LsrR: Effects on small RNA and Biofilm Architecture", *Journal of Bacteriology*, August 2007, Vol. 189/16, str. 6011-6020

[31] T. Lipniacki, P. Paszek, "Mathematical model of NP-κB regulatory model"Dostopno na: mbi.osu.edu/2003/ws1materials/lipniacki.ppt

[32] A. Loinger, O. Biham, "Stochastic simulations of the repressilator circuit", *Physical Review* 2007, Vol. 76, 051917, str. 1-9

[33] M.A. Marchisio, J. Stelling, "Computational design of synthetic gene circuits with composable parts", *Bioinformatics*, 2008, Vol. 26, str. 1903-1910

[34] C.T. Martin, A.Újvarí, "Thermodynamic and Kinetic Measurements of Promoter Binding by T7 RNA Polymerase", *Biochemistry*, 1996, Vol.35, str. 14574-14582

[35] W.T. McAllister, "Transcription by T7 RNA polymerase", *Nucleic Acids and Molecular Biology*, 1997, Vol. 11, str. 15-25

[36] M. Moškon, "Modeli in metrike dinamike preklopa v enostavnih bioloških sistemih za potrebe računalniških struktur prihodnosti", 2011, Doktorska disertacija (v pripravi)

[37] J.J. Mulawka, T. Janczak, A. Malinowski, R. Nowak, "DNA Computing - Promise for Information Processing", 2000

[38] R.S. Nikhil, "ID Language Reference Manual", *Computation Structures Group Memo 284-2, MIT*, Cambridge, MA, July, 1991

[39] J.L. Peterson, "Petri Net Theory and the Modeling of Systems", *Prentice-Hall*, 1981

[40] L. Qian, E. Winfree, "Scaling Up Digital Circuit Computation with DNA Strand Displacement Cascades", *Science*, June 2011, Vol. 332/6034, str. 1196-1201

[41] A.S. Ribeiro, "A Model of Genetic Networks with Delayed Stochastic Dynamics" in "Analysis of Microarray Data: Network based Approaches", M. Dehmer, F. Emmert-Streib (Editors), *Wiley*, 2007, str. 167-199

[42] , A.S. Ribeiro, J.Lloyd-Price, "SGN Sim, a Stochastic Genetic Networks Simulator", *Bioinformatics: Application Note*, 2007, Vol. 23/6, str. 777-779

[43] M.R. Roussel, R. Zhu, "Validation of an algorithm for delay stochastic simulation of transcription and translation in prokaryotic gene expression", *Phys. Biol.*, December 2006, Vol. 3/4, str. 274-84

[44] H. El Samad, M. Khammash, L. Petzold, D. Gillispie, "Stochastic modelling of gene regulatory networks", *International Journal of Robust and Nonlinear Control*, 2005, Vol. 15, str. 691-711

[45] T. Sera, "Zinc-finger-based artificial transcription factors and their applications", *Drug Delivery Reviews*, 2009, Vol. 61/7-8, str. 513-526

[46] J. Šilc, B. Robič, "Osnovna načela DF sistemov", *Informatica*, 1985, vol. 2, str. 10-15

[47] J. Šilc, B. Robič, "On choosing a plan for the execution of data flow program graph", *Informatica*, 1986, vol. 3, str. 11-17

[48] J. Šilc, "Sinhronizirana podatkovno pretokovna računalniška arhitektura", *Informatica*, 1992, vol. 2, str. 59-72

[49] J. Šilc, B. Robič, T. Ungerer, "Asynchrony in parallel computing: From dataflow to multithreading", *Parallel and Distributed Computing Practices*, 1998, vol. 1, str. 3-30

[50] J. Šilc, B. Robič, T. Ungerer, "Processor Architecture: From Dataflow to Superscalar and Beyond", *Springer-Verlag*, New York, Berlin, 1999

[51] Team Slovenia, "DNA Coding Beyond Triplets", *International Genetically Engineered Machine Competition - iGEM*, 2010, MIT, Cambridge, MA, dostopno na: http://2010.igem.org/Team:Slovenia

[52] TMS320C66x High-Performance Multicore DSP Documentation, dostopno na spletni strani: http://www.ti.com/

[53] G. Tkačik, A.M. Walczak, "Information transmission in genetic regulatory networks: a review", *Journal of Physics: Condensed Matter*, 2011, Vol.23

[54] P. Wasiewicz, A. Malinowski, R. Nowak, J.J. Mulawka, P. Borsuk, P. Węgleński, A. Plucienniczak, "DNA computing: implementation of data flow logical operations", *Future Generation Computer Systems*, 2001, Vol. 17, str. 361-378

[55] Wei-Ping Yang, H. Wu, C.F. Barbas III, "Surface plasmon resonance based kinetic studies of zinc finger-DNA interactions", *Journal of Immunological Methods*, 1995, Vol. 183, str. 175-182