Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Doktorska disertacija

ALGORITEM ZA UČINKOVIT IZRAČUN VERIGE ELEMENTARNIH MORFOLOŠKIH FILTROV NA CENTRALNI PROCESNI ENOTI

Maribor, marec 2021 Danijel Žlaus

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Doktorska disertacija

ALGORITEM ZA UČINKOVIT IZRAČUN VERIGE ELEMENTARNIH MORFOLOŠKIH FILTROV NA CENTRALNI PROCESNI ENOTI

Maribor, marec 2021

Danijel Žlaus

Mentor: izr. prof. dr. Domen Mongus

UDK: 004.925.8:519.254(043.3)

Avtor: Danijel Žlaus

Naslov: ALGORITEM ZA UČINKOVIT IZRAČUN VERIGE ELEMENTARNIH MORFOLOŠKIH FILTROV NA CENTRALNI PROCESNI ENOTI

UDK: 004.925.8:519.254(043.3)

Ključne besede: morfološki filtri, geodetski operatorji, veriga filtrov, obdelava slik, vzporedno procesiranje, podatkovno pretočna obdelava, vektorsko procesiranje, SIMD, centralna procesna enota

Število izvodov: 8

Obdelava besedila in slik: Danijel Žlaus

Jezikovni pregled: Darinka Verdonik

ZAHVALA

To delo je omogočil raziskovalni program št. P2-0041, katerega je sofinancirala Javna agencija za raziskovalno dejavnost Republike Slovenije iz državnega proračuna. Za svetovanje in izdatno strokovno pomoč tako ob in pred podiplomskim študijem se zahvaljujem mentorju izr. prof. dr. Domnu Mongusu in sodelavcem v laboratoriju GeMMA.

Za brezpogojno podporo in spodbujanje tekom študija se zahvaljujem svoji družini in jim posvečam to delo.

Algoritem za učinkovit izračun verige elementarnih morfoloških filtrov na centralni procesni enoti

Ključne besede:

morfološki filtri, geodetski operatorji, veriga filtrov, obdelava slik, vzporedno procesiranje, podatkovno pretočna obdelava, vektorsko procesiranje, SIMD, centralna procesna enota

Povzetek:

V doktorski disertaciji predstavimo nov algoritem za učinkovit izračun verige elementarnih filtrov na centralni procesni enoti. Verige filtrov so temeljni del tako imenovanih geodetskih operatorjev, ki uporabljajo postopek morfološke rekonstrukcije. Ta iterativno filtrira in omejuje vrednosti filtrirane slike. Dobljeno učinkovitost razvitega algoritma delimo na dva dela. Prvi del se zanaša na učinkovit enonitni izračun elementarnih morfoloških filtrov, kar dosežemo s procesiranjem na mestu, dekompozicijo strukturnega elementa in uporabo vektorskih registrov za pohitritev izračuna. Drugi del pa razvite filtre uporabi za učinkovit vzporeden in hkraten izračun več filtrov v podani verigi. Vzpo-

redni izračun je osnovan na emulaciji podatkovno pretokovnega procesiranja, kjer z analizo topologije predpomnilnika CPE in primernim pripenjanjem niti zagotovimo, da prenos podatkov med jedri CPE poteka preko predpomnilnika. Pokazali smo, da dosežemo največjo prepustnost procesiranja ob uporabi več neodvisnih vzporednih procesnih cevovodov, medtem ko najhitrejše odzivne čase dosežemo z uporabo enega cevovoda. Primerjava z algoritmi stanja tehnike in odprtokodnimi knjižnicami je pokazala, da je predlagani algoritem vedno dosegel boljše računske čase pri obdelavi verig filtrov. V primerjavi z iterativnim izračunom verige filtrov na splošnonamenskih grafičnih procesnih enotah se je predlagani algoritem prav tako izkazal za bistveno hitrejšega, tudi ko smo uporabili CPE nižjega cenovnega ranga.

Algorithm for efficient computation of elementary morphological filter chain on central processing unit

Key words:

morphological filters, geodesic operators, filter chain, image processing, parallel processing, dataflow processing, vector processing, SIMD, central processing unit

Abstract

In this doctoral dissertation we introduce a new algorithm for efficient calculation of filter chains on central processing unit. Filter chains are a fundamental part of the so-called geodesic operators, which use morphological reconstruction. The latter iteratively filters and bounds the values of the input image. The efficiency of the developed algorithm is compared from two points of view. The first relies on the efficient single-threaded calculation of elementary morphological filters, which is achieved by in-place processing, decomposition of

the structuring element and the use of vector registers to speed up calculations. The second part uses the developed filters to efficiently process multiple filters from a given filter chain in parallel. The parallel calculation is based on emulating dataflow processing, where analysis of CPU's cache topology and appropriate thread pinning ensures optimal data transfers between threads via the hierarchical cache. We have shown that the highest processing throughput is achieved using several independent processing pipelines in parallel, while the lowest processing latency is obtained with a single pipeline. A comparison with state-of-the-art algorithms and open source libraries showed that the proposed algorithm always achieves better computational times when processing filter chains. A comparison of computation times of filter chains using general-purpose graphics processing units showed that the proposed algorithm also significantly outperforms them, even when using a lower priced CPU.

Kazalo

Kratice

ALE	-	aritmetično-logična enota
AVX	-	ukazni nabor z 256-bitnimi vektorskimi registri, angl. Advanced Vector Extensions
CPE	-	centralno procesna enota
DD	-	algoritem Dokládal in Dokladalova
FPGA	-	angl. Field Programmable Gate Array
GK	-	algoritem Gil in Kimmel
GPGPU	-	splošnonamenska grafična procesna enota, angl. General Purpose Graphics Processing Unit
HGW	-	algoritem Herk, Gil in Werkman
IK	-	operator izvlečka kupol
L1, L2, L3	-	nivoji predpomnilnika CPE, angl. Level 1, Level 2 and Level 3
LiDAR	-	lasersko prebiranje razdalje, angl. Light Detection and Ranging
MAMBA	-	odprtokodna knjižnica za morfološko procesiranje slik, angl. MAthematical Morphology liBrAry
MIMD	-	angl. Multiple Instructions, Multiple Data streams
MISD	-	angl. Multiple Instructions, Single Data stream
MSPE	-	morfološka soprocesna enota
MMX	-	ukazni nabor s 64-bitnimi vektorskimi registri, angl. Matrix Math eXtension
OO	-	operator odstranjevanja objektov
OpenCV	-	splošnonamenska odprtokodna knjižnica za filtriranje slik, angl. Open source Computer Vision
PE	-	procesna enota
RK	-	operator ravnanja kupol

SIMD	-	angl. Single Instruction, Multiple Data streams
SIMT	-	angl. Single Instruction, Multiple Threads
SISD	-	angl. Single Instruction, Single Data stream
SMIL	-	angl. Simple Morphological Image Library
SMT	-	istočasna večnitnost, angl. simultaneous multithreading
SPL	-	operator selektivnega polnjenja robov
SSE	-	ukazni nabor s 128-bitnimi vektorskimi registri, angl. Streaming SIMD Extensions
TDR	-	operator transformacije domnevnih razdalj
TFLOPS	-	angl. Tera FLoating point Operations Per Second
UW	-	algoritem Urbach in Wilkinson
VO	-	funkcija velikosti objektov

Seznam oznak

ϵ_1^m/δ_1^m	-	elementarno geodetsko krčenje/širjenje
$\epsilon_{rek}^m/\delta_{rek}^m$	-	rekonstrukcija s krčenjem/širjenjem
ϵ_s/δ_s	-	morfološko krčenje/širjenje z w_s
f	-	vhodna slika
G_s	-	veriga filtrov dolžine s
γ_{rek}^s / φ_{rek}^s	-	morfološko odpiranje/zapiranje s postopkom rekonstrukcije
γ_s/φ_s	-	morfološko odpiranje/zapiranje z w_s
$\mathrm{IZF}_s^\gamma/\mathrm{IZF}_s^\varphi$	-	operator izmenično-zaporednih filtrov, ki prične z morfološkim odpiranjem/zapiranjem
K	-	korak premika
m	-	slika maske
$\mathcal{O}$	-	asimptotska časovna zahtevnost
P	-	prostor slike
p_i	-	i-ta urejena točka v sliki
V_r	-	velikost vektorskega registra
V_t	-	velikost podatkovnega tipa
w_1	-	strukturni element velikost 3×3
w_s	-	strukturni element velikost $(2s+1) \times (2s+1)$
X	-	širina slike
Y	-	višina slike

Poglavje 1

Uvod

1.1 Opredelitev problema

Morfološki filtri veljajo za pomembno orodje pri odkrivanju prostorskih struktur v različnih vrstah podatkovnih virov in tokov. Prostorske strukture so tipično vsebovane v sivinskih slikah, kjer vrednost posameznega piksla nosi informacijo o višini (oziroma globini) ali moči odboja elektromagnetnega sevanja v specifičnem pasu valovnih dolžin. Takšne slike običajno pridobimo s sistemi laserskega prebiranja LiDAR (angl. Light Detection and Ranging) [1, 2], sistemi brezpilotnih letalnikov [3, 4], sateliti za opazovanje Zemlje [5, 6], medicinskimi senzorji [7, 8] ali drugimi senzorskimi sistemi [9, 10, 11, 12]. V kontekstu njihove obdelave so še posebej pomembni tako imenovani geodetski operatorji matematične morfologije [13], ki so zmožni v celoti odstraniti izbrane robove (in z njimi določene povezane komponente), pri čemer ohranijo preostale komponente povsem nespremenjene. Njihov izračun temelji na iterativnem postopku tako imenovane morfološke rekonstrukcije, ki uporablja elementarno morfološko krčenje ali širjenje izbrane slike. Posledično pa izvedba tovrstnih operatorjev zahteva ovrednotenje dolgih verig zaporedno odvisnih

filtrov, kar navkljub uporabi preprostih elementarnih filtrov terja visoko računsko zahtevnost. Kljub temu pa so danes geodetski operatorji pogosto uporabljeni za analize pokritosti tal [14, 15, 16, 17], razpoznave objektov in terena [18, 19, 20, 21] ter zaznavo časovnih sprememb površja [22, 23, 24, 25].

V literaturi zasledimo več pristopov [26, 27, 28, 29], ki rešujejo problem učinkovitega izračuna poljubnega morfološkega krčenja in širjenja na centralni procesni enoti (CPE). Skupna lastnost vseh tehnik je optimizacija števila potrebnih primerjav za obdelavo posameznega piksla, kjer najsodobnejše metode dosegajo časovno zahtevnost $\mathcal{O}(1)$ na piksel [28, 29]. Nobena od teh tehnik pa ni uporabljena v realizaciji vzporednega izračuna več zaporedno odvisnih elementarnih filtrov na CPE. V ta namen zasledimo zgolj uporabo namenske strojne opreme s programirljivimi čipi FPGA (angl. Field Programmable Gate Array) [30, 31, 32], pri čemer ti pristopi izkoriščajo prostorski paralelizem znotraj sintetiziranega procesnega cevovoda. Nasprotno pa se algoritmi, ki se izvajajo na CPE, osredotočajo na pohitritev izračuna znanih metod, pri čemer pa za procesiranje posameznega filtra uporabijo razpoložljive večjedrne vire [33]. Nadaljnje pohitritve procesiranja posameznega filtra so tipično dosežene s prenosom izvajanja iz CPE na splošnonamensko grafično procesno enoto GPGPU (angl. General Purpose Graphics Processing Unit) [34, 35, 36], ki podpirajo masovno vzporeden izračun. Široko uporabljene odprtokodne knjižnice za obdelavo slik, kot sta to OpenCV [37] (angl. Open source Computer Vision) in SMIL [38] (angl. Simple Morphological Image Library), uporabljajo več implementacij razgrajenih elementarnih strukturnih elementov. Slednje jim omogoča, da v času izvajanja izberejo optimalno metodo glede na njihovo podano obliko. Vseeno pa tudi te knjižnice ne realizirajo vzporednega izračuna več filtrov, temveč se osredotočajo zgolj na pospešitev izračuna posameznega filtra z uporabo razpoložljivih večjedrnih virov. Zato so vsi algoritmi za CPE neučinkoviti pri izračunu geodetskih operaterjev, za katere so značilne dolge verige elementarnih filtrov. Posledično pa izdelava algoritma, zmožnega učinkovitega vzporednega izračuna verige elementarnih morfoloških filtrov, predstavlja pomemben izziv in je glavni cilj doktorske disertacije.

1.2 Cilji in hipoteze

Cilj doktorske disertacije je razvoj algoritma za učinkovit izračun verige elementarnih morfoloških filtrov na CPE. Predlagani algoritem razgradi vhodni morfološki filter na verigo zaporedno odvisnih elementarnih filtrov z uporabo dekompozicije. Sami elementarni filtri so prirejeni za učinkovito delovanje na CPE, saj so implementirani z ukaznim naborom AVX2, kjer upravljamo z 256-bitnimi vektorskimi registri. Tako z enim ukazom hkrati obdelamo od 4 (podatkovni tip *double*, ki obsega 64 bitov) do 32 pikslov (podatkovni tip *char*, ki obsega 8 bitov). Sama implementacija je dovolj splošna, da omogoča enostavno uporabo prihodnjih ukaznih naborov, ki razširijo vektorske registre na 512 bitov in 1024 bitov, kar dodatno poveča učinkovitost razvitega algoritma. Ker vse moderne CPE omogočajo vzporedno obdelavo, so elementarni filtri optimizirani za večjedrno delovanje, kjer si delovne podatke posredno izmenjujejo preko predpomnilnika CPE. Vseeno pa je pri tem potrebno upoštevati, da je predpomnilnik v modernih CPE heterogen, saj je hierarhičen (tipično deljen v nivoje L1, L2 in L3), sama topologija povezav med različnimi nivoji predpomnilnikov pa je odvisna od konkretne arhitekture CPE. Za učinkovito posredno izmenjavo podatkov med filtri algoritem v času zagona pregleda topologijo predpomnilnika CPE in prilagodi vzporedno obdelavo tako, da je optimizirana odzivnost ali prepustnost obdelave. Prilagoditev vzporedne obdelave je uporabniško nastavljiv parameter v času izvajanja. Razviti algoritem primerjamo na primeru verige elementarnih filtrov tudi s preostalimi algoritmi ter z implementacijo na GPGPU. Na podlagi opisanih ciljev zastavimo naslednjo tezo:

Z izkoriščanjem paralelizma ukaznega nivoja, vektorskega procesiranja in večjedrnega izvajanja je mogoče doseči večjo učinkovitost izvedbe dolgih verig elementarnih morfoloških filtrov v primerjavi z obstoječimi algoritmi.

Na osnovi teze postavimo sledeče, bolj specifične hipoteze:

Hipoteza 1. *Izračun verig elementarnih morfoloških filtrov po metodologiji, ki temelji na ukaznem naboru AVX2, je vsaj štirikrat hitrejši od trenutno poznanih metod, implementiranih na centralnih procesnih enotah.*

Hipoteza 2. *Izračun verig elementarnih morfoloških filtrov je možno učinkoviteje izvesti na centralni procesni enoti kot na splošnonamenskih grafičnih procesnih enotah z računsko zmogljivostjo do 10 TFLOPS.*

Hipoteza 3. *Zaradi zakasnitev, ki so posledica medjedrne komunikacije, dosežemo v času izvajanja optimalen izračun verige morfoloških filtrov z vzporedno obdelavo več verig filtrov in omejitvijo števila uporabljenih jeder na posamezno verigo.*

1.3 Izvirni prispevki

V disertaciji predstavimo algoritem za učinkovit izračun verig elementarnih morfoloških filtrov na CPE. Pri tem so v doktorski disertaciji predstavljeni naslednji izvirni znanstveni prispevki:

- razvoj računsko učinkovitega postopka za izračun elementarnih morfoloških filtrov, kjer so upoštevane značilnosti modernih CPE;
- vzporedna obdelava verige elementarnih filtrov na CPE s tehniko podatkovno pretokovnega procesiranja;
- odločitvena logika, ki v času izvajanja, s pomočjo pregleda topologije CPE, omogoča prilagoditev izvajalnih parametrov za dosego največje možne odzivnosti ali prepustnosti obdelave slik;

- analiza učinkovitosti izračuna verig elementarnih filtrov na CPE in GPGPU.

1.4 Struktura disertacije

Doktorska disertacija je sestavljena iz šestih poglavij. V drugem poglavju predstavimo teoretična izhodišča sivinskih morfoloških filtrov. V tretjem poglavju pregledamo sorodne metode, kjer se osredotočimo na ukazno pretokovne metode in njihovo izvedbo na CPE ter podatkovno pretokovne metode in njihovo izvedbo na FPGA, izvedene z namenom pohitritve izračuna morfoloških filtrov. V četrtem poglavju opišemo predlagani algoritem za učinkovit izračun verige elementarnih filtrov na CPE. Ta sestoji iz izračuna na mestu, izkoriščanja vektorskega procesiranja, vzporednega izračuna ter izkoriščanja topologije CPE za učinkovit prenos podatkov med nitmi. V petem poglavju predstavimo širšo analizo predlaganega algoritma na več arhitekturno različnih CPE. Pri tem se osredotočamo na ovrednotenje učinkovitosti enojedrnega izvajanja ter skaliranja ob uporabi večjedrnega izvajanja. Prav tako analiziramo tudi optimizacije glede na odzivnosti in prepustnosti posameznega testnega sistema. Nazadnje opravimo še primerjavo z najsodobnejšimi metodami, širše uporabljenimi programskimi knjižnicami in implementacijami na GPGPU. Disertacijo zaključimo s šestim poglavjem, kjer na podlagi rezultatov potrdimo ali ovržemo zastavljene hipoteze ter podpremo naše prispevke k znanosti.

Poglavje 2

Teoretična izhodišča

V tem poglavju podamo definicije morfoloških filtrov in operatorjev, ki predstavljajo osnovo izvedbe geodetskih operatorjev. Morfološki filtri izhajajo iz teorije matematične morfologije. Slednja opisuje sistematičen pristop k analizi prostorskih struktur v heterogenih podatkovnih virih neodvisno od njihove dimenzionalnosti. Tako lahko na primer na enak način definiramo operatorje nad enodimenzionalnimi signali, dvodimenzionalnimi slikami in višjimi dimenzionalnimi prostori (vokselski prostori ter multispektralne slike). Posledično lahko brez izgube splošnosti podanih definicij obravnavamo piksel kot elementarno prostorsko entiteto rasteriziranega prostora.

Morfološki operatorji so komplementarni konvencionalnim linearnim filtrom pri obdelavi slik. Izhajajo iz teorije množic in opredeljuje aritmetiko oblik [39, 40]. Njihovo matematično formulacijo sta v poznih šestdesetih podala Jean Serra in Georges Matheron [41], ki sta na osnovi unije in preseka opredelila tako imenovane binarne morfološke filtre. Uporabnost predstavljenih konceptov je vodila v številne razširitve, med katerimi je ena pomembnejših gotovo generalizacija binarnih konceptov v sivinsko matematično morfologijo [42]. Zaradi svoje diskretne zasnove slednja omogoča učinkovito analizo sivinskih slik.

Prostor slike $P = \{p_i\}$, kjer $P \subset \mathbb{Z}^2$ definira urejeno množico koordinat pikslov $p_i = \{x_i, y_i\}$, podanih s pari koordinat, dobljenih s kartezičnim produktom $P = \{0, 1, \ldots, X-1\} \times \{0, 1, \ldots, Y-1\}$, kjer X in Y določata širino in višino slike. Tako lahko sliko definiramo kot funkcijo $f : P \to \mathbb{R}$, ki koordinati p_i pripiše realno vrednost, ki predstavlja intenziteto piksla.

2.1 Osnovni morfološki operatorji

V tem podpoglavju podamo definicije temeljnih filtrov matematične morfologije, ki predstavljajo osnovne operande in omogočajo izpeljavo vseh naprednejših morfoloških operatorjev, transformacij in metod. Vsak morfološki filter za svojo izvedbo zahteva operanda, ki ju določata slika f in strukturni element w. Slednji definira doseg filtra in je lahko poljubne oblike, a je tipično prilagojen na obliko iskanih prostorskih struktur. Z namenom poenostavitve definicij se v tem delu omejimo na strukturne elemente kvadratne oblike in lihe velikosti. Slednje definiramo kot $w_s(p_i) = \{x_i - s, x_i - s + 1, \ldots, x_i + s\} \times \{y_i - s, y_i - s + 1, \ldots, y_i + s\}$, kjer s določa velikost strukturnega elementa in p_i njegovo izhodišče, pri čemer velja $w_s(p_i) \subseteq P$. Posledično je vloga strukturnega elementa podobna vlogi okna pri implementaciji tradicionalnih konvolucijskih filtrov s širšega področja obdelave signalov in slik.

Osnovna morfološka filtra imenujemo krčenje ter širjenje in sta dualna. Morfološko krčenje ϵ definiramo kot izbor najmanjše vrednosti v okolici w_s. Formalno [43]:

$$[\epsilon_s(f)](p_i) = \min_{q \in w_s(p_i)} f(q). \tag{2.1}$$

Nasprotno pa je morfološko širjenje δ definirano kot izbira največje vrednosti

v okolici w_s in je podano kot:

$$[\delta_s(f)](p_i) = \max_{q \in w_s(p_i)} f(q). \tag{2.2}$$

Dualnost operatorjev (angl. duality) definira enačba 2.3.

$$\epsilon_s(f) = -\delta_s(-f). \tag{2.3}$$

Za oba operatorja pokažemo tudi, da sta naraščajoča [43], kar izrazimo z:

$$\forall (f, f)' : f \leq f' \quad \Rightarrow \quad \begin{matrix} \epsilon_s(f) \leq \epsilon_s(f') \\ \delta_s(f) \leq \delta_s(f'), \end{matrix} \tag{2.4}$$

kjer sta f in f' sivinski sliki, pri čemer pravimo, da je slika f manjša ali enaka sliki f', če je vrednost slike f v vsakem pikslu manjša ali enaka vrednosti slike f'. Formalno to zapišemo kot:

$$f \leq f' \;\Rightarrow\; \forall p \in P : f(p) \leq f'(p). \tag{2.5}$$

Hkrati sta oba operatorja tudi distributivna, iz česar sledi:

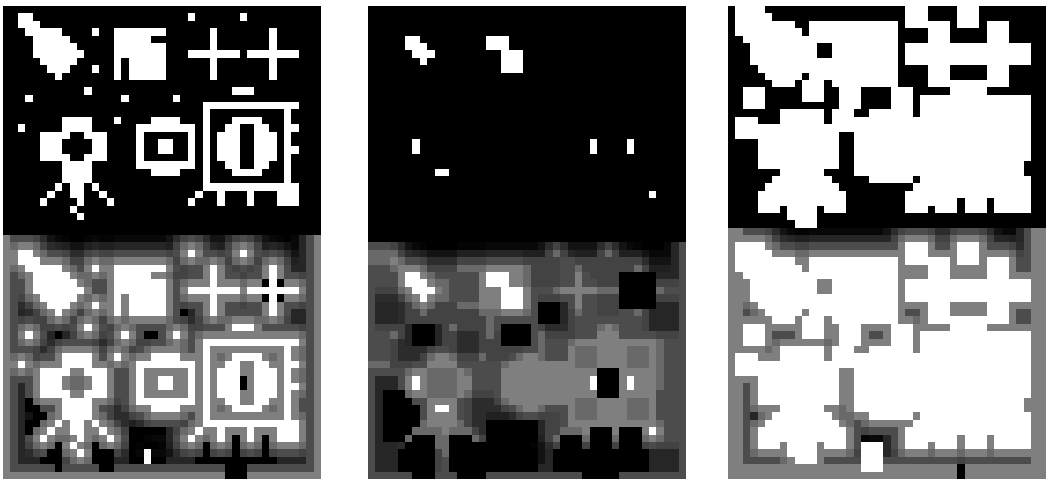
$$\min(\epsilon_s(f), \epsilon_s(f')) = \epsilon_s(\min(f, f')) \tag{2.6}$$

$$\max(\delta_s(f), \delta_s(f')) = \delta_s(\max(f, f')). \tag{2.7}$$

Pri tem za krčenje velja, da je nerazširljivo, medtem ko je širjenje razširljivo. Slednje zapišemo kot:

$$\epsilon_s(f) \leq f \leq \delta_s(f). \tag{2.8}$$

Primer krčenja in širjenja s strukturnim elementom w_1 prikazuje slika 2.1. Krčenje lahko razumemo kot odstranjevanje svetlih objektov, ki so manjši od uporabljenega strukturnega elementa. Nasprotno pa širjenje odstrani manjše temne objekte, ki so obdani s svetlejšim ospredjem. Pri tem oba operatorja



(a) (b) (c)

Slika 2.1: Filtriranje (a) vhodne slike, kjer (b) rezultat morfološkega krčenja ϵ_1 razširi ozadje, medtem ko (c) morfološko širjenje δ_1 razširi ospredje.

spremenita vrednosti vseh pikslov v sliki in posledično spremenita vsebovane prostorske strukture. Širjenje in krčenje zato pogosteje uporabljamo v paru, kjer lastnosti dualnosti, monotonosti, razširljivosti ter nerazširljivosti omogočajo povrnitev pomembnega dela prostorskih struktur v njihovo prvotno obliko. Tako definiranim operatorjem pravimo morfološko odpiranje γ in zapiranje φ.

Odpiranje je operator, ki omogoča odstranjevanje svetlih objektov tako, da najprej izvedemo krčenje in nato širjenje. V koraku krčenja odstranimo svetle objekte, v koraku širjenja pa povrnemo prvotne vrednosti preostalim strukturam. Odpiranje posledično ohrani večje prostorske strukture v njihovem izvornem območju intenzitet in odstrani manjše tako, da jih zlije s temnejšim ozadjem. Formalna definicija morfološkega odpiranja γ z oknom w_s je:

$$\gamma_s(f) = \delta_s(\epsilon_s(f)). \tag{2.9}$$

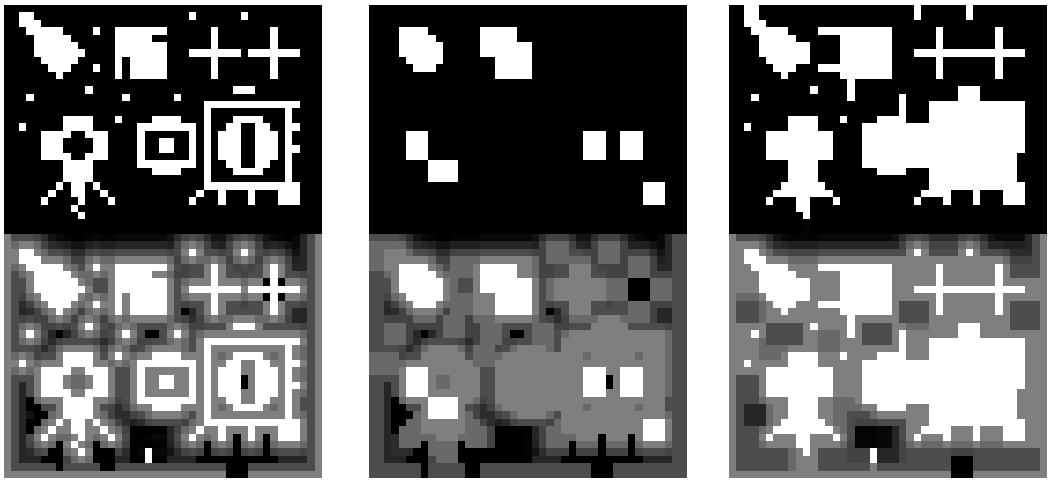
Nasprotno pa zapiranje odstrani temne objekte, manjše od strukturnega elementa w_s, in ohrani večje, tako da najprej izvede širjenje in nato krčenje.

Formalna definicija morfološkega zapiranja φ z oknom w_s je:

$$\varphi_s(f) = \epsilon_s(\delta_s(f)). \tag{2.10}$$

Podobno kot v primeru širjenja in krčenja lahko tudi za odpiranje in zapiranje pokažemo njuno dualnost, naraščanje in (ne)razširljivost [43]. Splošna definicija relacij med osnovnimi morfološkimi operatorji je tako:

$$\epsilon_s(f) \leq \gamma_s(f) \leq f \leq \varphi_s(f) \leq \delta_s(f). \tag{2.11}$$



(a) (b) (c)

Slika 2.2: Filtriranje (a) vhodne slike, kjer (b) morfološko odpiranje preostalemu ospredju povrne prvotne vrednosti, obratno pa (c) morfološko zapiranje povrne vrednosti preostalega ozadja.

Poleg navedenega sta odpiranje in zapiranje tudi idempotentna, kar formalno zapišemo kot:

$$\forall s_1, s_2 : s_1 \leq s_2 \quad \Rightarrow \quad \begin{array}{c} \gamma_{s_1}(\gamma_{s_2}(f)) = \gamma_{s_2}(f) \\ \varphi_{s_1}(\varphi_{s_2}(f)) = \varphi_{s_2}(f). \end{array} \tag{2.12}$$

Zaporedna izvedba istega operatorja tako ne spremeni rezultata, če ob vsaki

izvedbi uporabimo manjši ali enak strukturni element. Z velikostjo strukturnega elementa torej nadziramo velikost izločenih prostorskih struktur. Slednje prikazuje slika 2.2, kjer lahko vidimo, da odpiranje in zapiranje odstranita objekte in tiste dele objektov, ki so ožji od strukturnega elementa. Zaradi filtriranja na nivoju piksla, ki lahko delno spremeni večje objekte, temeljni filtri krčenja, širjenja, odpiranja in zapiranja ne omogočajo obravnavanja prostorskih struktur kot celote in posledično niso objektno usmerjeni (angl. object-based).

2.2 Geodetski operatorji

V tem podpoglavju podamo definicije temeljnih geodetskih filtrov, ki omogočajo izpeljavo objektno usmerjenih tehnik obdelave slik. Objekte na sivinskih slikah razumemo kot povezane komponente, ki so od okolice (ozadja) ločene glede na svoje spektralne lastnosti (na primer intenziteto ali teksturo). Objektno usmerjena analiza slik [44] omogoča obravnavanje tovrstnih komponent kot osnovnih enot, ki jih lahko glede na njihove lastnosti v celoti ohranimo ali odstranimo iz slike, ne da bi pri tem vplivali na ozadje ali ostale objekte. Tehniko matematične morfologije, ki omogoča tovrsten pristop, imenujemo povezani operatorji [13, 45]. Ti temeljijo na procesu tako imenovane geodetske rekonstrukcije. Geodetski operatorji razširijo število operandov, saj za svojo izvedbo zahtevajo označevalno sliko f, kontrolno sliko m in enotski strukturni element w_1. Za sliki f in m velja, da sta definirani nad istim prostorom P in se tako ujemata v ločljivosti. Kontrolna slika m pri tem podaja omejitev filtriranja in jo tudi imenujemo maska. Enotski strukturni element w_1 določa povezanost pikslov, ki je v vseh primerih znotraj tega doktorskega dela podana z 8-sosednostjo. Osnovna geodetska filtra imenujemo elementarno geodetsko krčenje in širjenje. Prvo izvede enotsko krčenje slike f s strukturnim elementom w_1, pri čemer maska m omejuje najmanjšo dovoljeno vrednost vsakega piksla p_i. Definicija elementarnega geodetskega krčenja za piksel p_i je tako

podana kot:

$$[\epsilon_1^m(f)](p_i) = \max([\epsilon_1(f)](p_i), m(p_i)). \tag{2.13}$$

Nasprotno pa elementarno geodetsko širjenje sliko f razširi s strukturnim elementom w_1, pri čemer maska m omejuje največjo dovoljeno vrednost v vsakem pikslu p_i. Formalno to zapišemo kot:

$$[\delta_1^m(f)](p_i) = \min([\delta_1(f)](p_i), m(p_i)). \tag{2.14}$$

Lastnost dualnosti obeh elementarnih geodetskih operatorjev izhaja iz enačbe 2.3 in jo lahko izpeljemo z:

$$\begin{aligned} \epsilon_1^m(f) &= \max(\epsilon_1(f), m) \\ &= -\min(-\epsilon_1(f), -m) \\ &= -\min(\delta_1(-f), -m) \\ &= -\delta_1^{-m}(-f), \end{aligned} \tag{2.15}$$

$$\begin{aligned} \delta_1^m(f) &= \min(\delta_1(f), m) \\ &= -\max(-\delta_1(f), -m) \\ &= -\max(\epsilon_1(-f), -m) \\ &= -\epsilon_1^{-m}(-f). \end{aligned} \tag{2.16}$$

Ker sta operatorja dualna, nadaljnje lastnosti pokažemo samo za elementarno geodetsko krčenje. Lastnost naraščanja je formalno definirana kot [43]:

$$\forall f, m, f', m' : \quad f \leq f' \leq m \leq m' \quad \Rightarrow \quad \epsilon_1^m(f) \leq \epsilon_1^{m'}(f'), \tag{2.17}$$

kjer sta f in f' označevalni sliki, m in m' pa maski. Nerazširljivost izpeljemo iz enačbe 2.8:

$$\epsilon_1(f) \leq f \quad \Rightarrow \quad max(\epsilon_1(f), m) \leq max(f, m) \quad \Rightarrow \quad \epsilon_1^m(f) \leq f, \tag{2.18}$$

kar pa velja zgolj, kadar $f \geq m$, saj drugače $max(f, m) \neq f$. Posledično nerazširljivost formalno izrazimo kot:

$$\forall f, m : f \geq m \quad \Rightarrow \quad \epsilon_1^m(f) \leq f. \tag{2.19}$$

Splošno je geodetsko krčenje velikosti s definirano kot veriga elementarnih krčenj:

$$\epsilon_s^m(f) = \underbrace{\epsilon_1^m(\epsilon_1^m(\ldots(\epsilon_1^m}_{s \text{ krat}}(f))\ldots)), \tag{2.20}$$

medtem ko je geodetsko širjenje velikosti s definirano kot veriga elementarnih širjenj:

$$\delta_s^m(f) = \underbrace{\delta_1^m(\delta_1^m(\ldots(\delta_1^m}_{s \text{ krat}}(f))\ldots)). \tag{2.21}$$

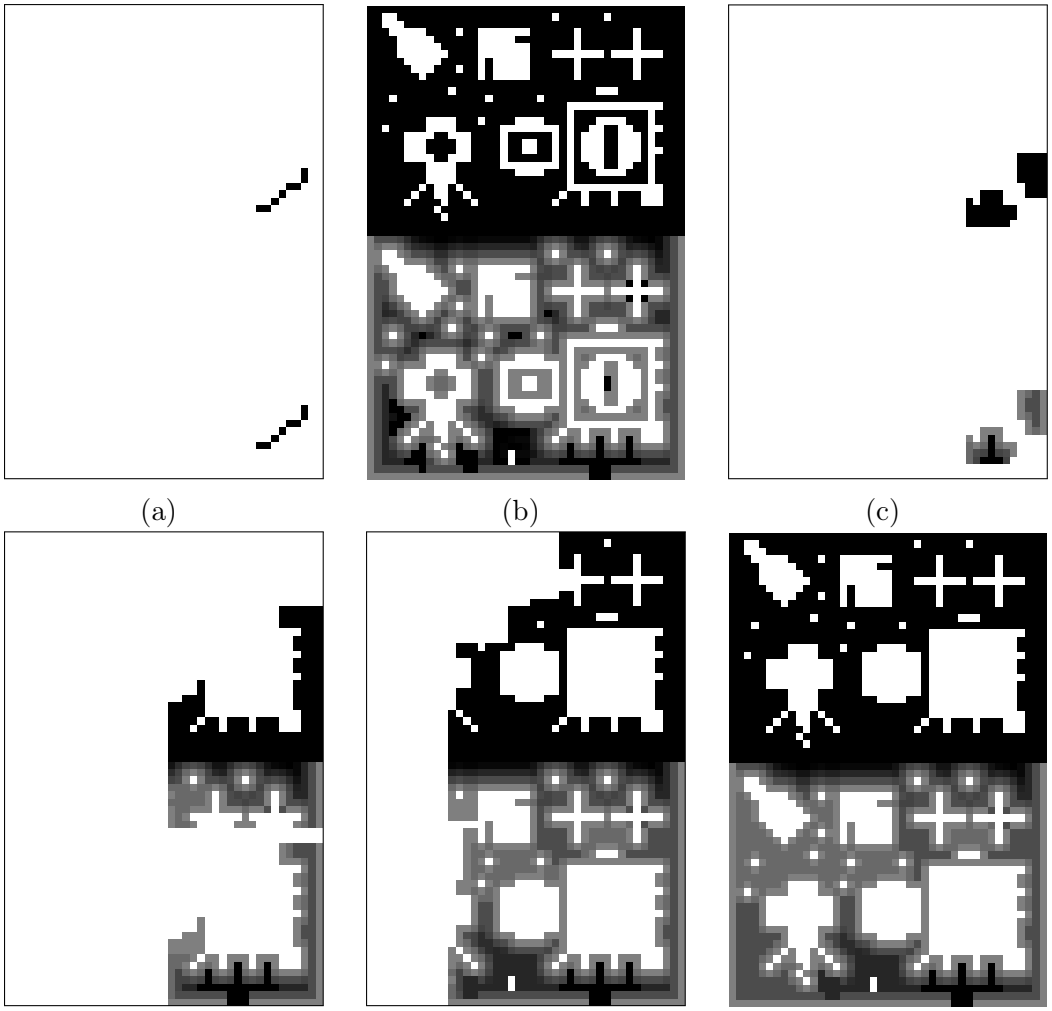
Zaradi nerazširljivosti elementarnega geodetskega krčenja (ki izhaja iz enačbe 2.18) in omejene najnižje vrednosti posameznih pikslov (ki izhaja iz njegove definiciji, podane z enačbo 2.13) ima splošno geodetsko krčenje svojo limito. Iterativni postopek izračuna limite imenujemo rekonstrukcija s krčenjem ϵ_{rek} in ga izrazimo kot:

$$\epsilon_{rek}^m(f) = \lim_{s \to \infty} \epsilon_s^m(f). \tag{2.22}$$

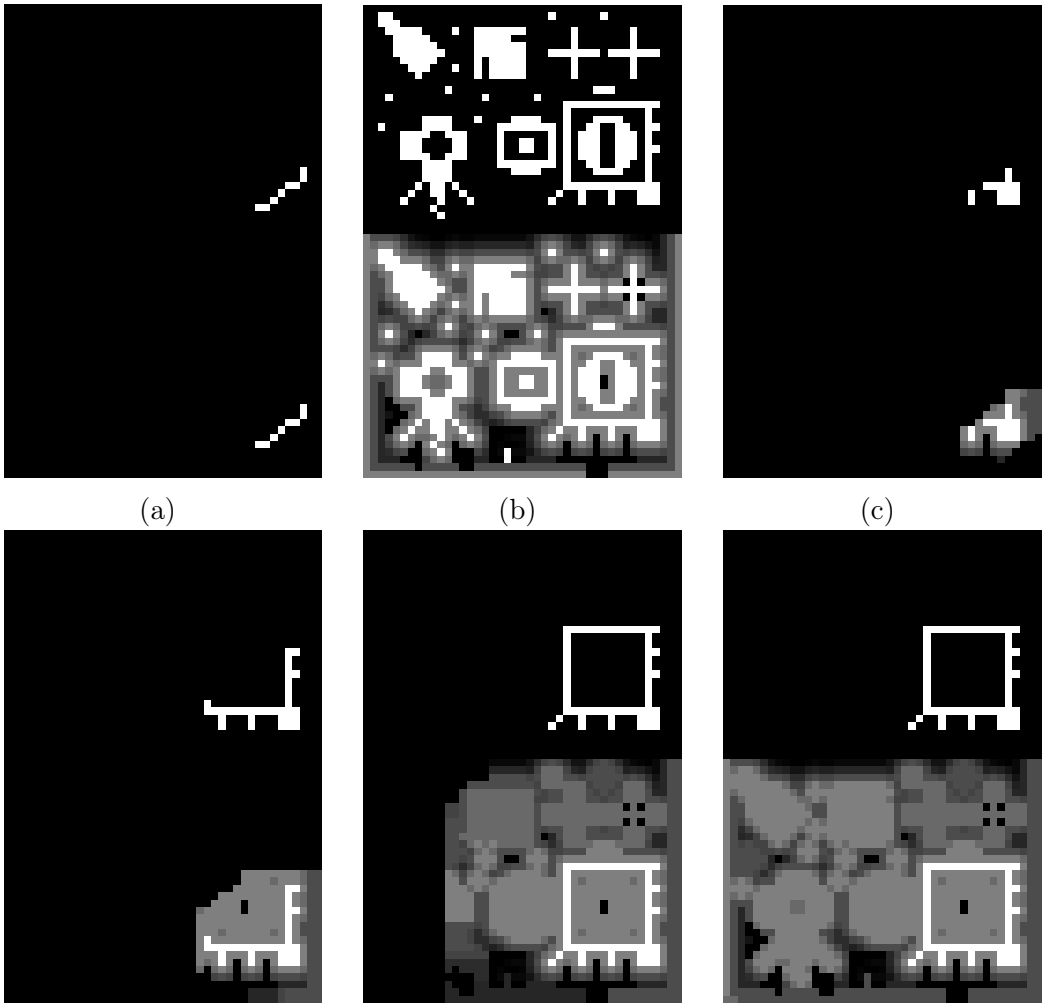
Zaradi dualnosti lahko tako definiramo tudi postopek rekonstrukcije s širjenjem δ_{rek}, ki ga definiramo kot:

$$\delta_{rek}^m(f) = \lim_{s \to \infty} -\epsilon_s^{-m}(-f) = \lim_{s \to \infty} \delta_s^m(f). \tag{2.23}$$

Primer temeljnih geodetskih filtrov prikazujeta sliki 2.3 in 2.4. Tako rekonstrukcija s krčenjem kot rekonstrukcija s širjenjem zahteva zgolj dva operanda, in sicer označevalno sliko f in masko m. Pri tem masko m običajno definira



Slika 2.3: Geodetsko krčenje nad (a) označevalno sliko z (b) masko, kjer (c, d in e) geodetska krčenja ϵ_2^m, ϵ_{12}^m in ϵ_{23}^m delno obnovijo označeno ozadje, medtem ko ga limita (f) krčenja z rekonstrukcijo ϵ_{rek} popolnoma obnovi.



Slika 2.4: Geodetsko širjenje nad (a) označevalno sliko in (b) masko, kjer (c, d in e) geodetska širjenja δ_2^m, δ_8^m in δ_{23}^m delno obnovijo označene objekte in skupno ozadje, medtem ko jih limita (f) širjenja z rekonstrukcijo δ_{rek} popolnoma obnovi.

izvorna slika, medtem ko označevalna slika f predstavlja izbiro objektov zanimanja. Operator rekonstrukcije s širjenem lahko razumemo kot transformacijo oziroma rekonstrukcijo označevalnih pikslov slike f v objekte ospredja oziroma izdvajanje povezanih komponent, ki so prisotne v sliki m. Obratno pa lahko razumemo operator rekonstrukcije s krčenjem kot transformacijo temnih pikslov označevalne slike f v ozadje slike m. V kolikor pa slika f predstavlja rezultat filtriranja maske m, postopek rekonstrukcije iterativno obnovi tiste objekte, ki niso bili v celoti odstranjeni. Iz tega sledi, da sta operatorja objektno usmerjena, saj sta skupaj s smiselno uporabo dodatnih filtrov sposobna obravnavati prostorske strukture kot celote.

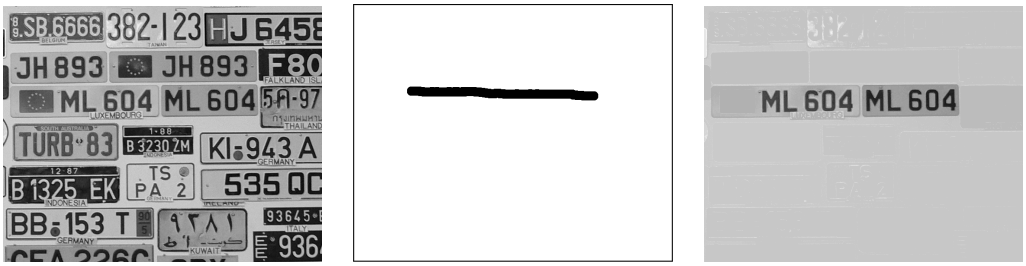
2.3 Izpeljani operatorji

V tem podpoglavju predstavimo nekatere pogosto uporabljene morfološke operatorje, ki izhajajo iz predstavljenih konceptov rekonstrukcije. Med te uvrščamo:

- **Selektivno polnjenje lukenj** (angl. hole filling) omogoča nadzorovano odstranjevanje temnih predelov slike, tako da jih zlijemo z njihovo svetlejšo okolico. Sliko f v tem primeru uporabimo kot masko, medtem ko rekonstrukcijo s krčenjem izvedemo nad označevalno sliko m. Slednjo lahko z uporabo poljubnih operatorjev izpeljemo iz vhodne slike f ali pa v ta namen uporabimo poljubno označevalno sliko, kot to prikazuje slika 2.5. Formalno operator SPL definiramo kot:

$$\mathrm{SPL}(f, m) = \epsilon^{f}_{rek}(m), \tag{2.24}$$

  Primer na sliki 2.5 prikazuje označevalno sliko, kjer črni piksli definirajo začetno regijo rekonstrukcije s krčenjem. Posledično se na vhodni sliki ohranijo samo tisti temni objekti, ki se prekrivajo z označevalno regijo,



(a) (b) (c)

Slika 2.5: Selektivno polnjenje lukenj na (a) vhodni sliki [46] z uporabo (b) označevalne slike, kjer (c) rezultat vsebuje označene temne objekte.

medtem ko se preostali zlijejo s svetlejšo okolico.

- **Odstranjevanje objektov** (angl. object removal) omogoča nadzorovano odstranjevanje objektov iz poljubne vhodne slike f, ki v tem primeru definira masko, medtem ko rekonstrukcijo s širjenjem izvedemo nad označevalno sliko m. Slednja označuje objekte, ki jih odstranimo. Formalno je odstranjevanje objektov OO podano kot cilindrična transformacija (angl. top-hat) širjenja z rekonstrukcijo:

$$\mathrm{OO}(f, m) = f - \delta^{f}_{rek}(m). \tag{2.25}$$

(a) (b) (c)

Slika 2.6: Odstranjevanje objektov na (a) vhodni sliki [46] z uporabo (b) označevalne slike, pri čemer (c) rezultat ne vsebuje označenih svetlih objektov.

Slika 2.6 prikazuje primer uporabe operatorja $\mathrm{OO}(f, m)$ nad vhodno sliko

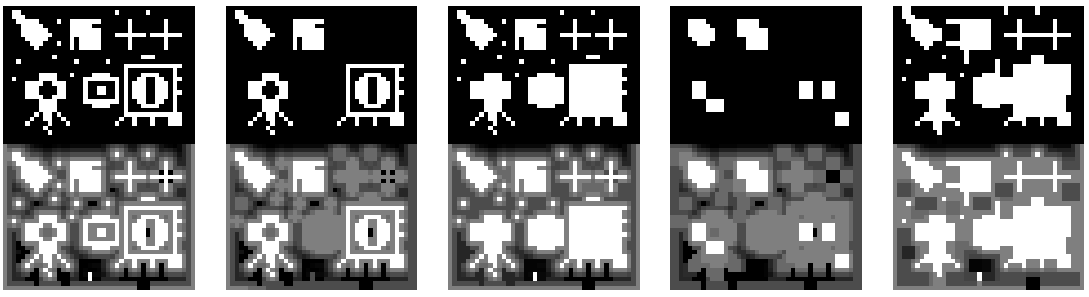
2.6a, ki odstrani objekte, označene s sliko 2.6b. Pri tem je potrebno poudariti, da so neoznačeni vgrajeni objekti na izhodni sliki 2.6c nižje intenzitete. To je posledica cilindrične transformacije, ki poleg objektov izloči tudi skupno ozadje vhodne in rekonstruirane slike. Vseeno pa se tako ohrani njihov kontrast.

- **Odpiranje in zapiranje z rekonstrukcijo** [47] v celoti odstranita povezane komponente, ki so ožje od okna w_s, medtem ko preostale ohranita povsem nespremenjene. Odpiranje z rekonstrukcijo γ_{rek} z oknom w_s izrazimo kot:

$$\gamma^s_{rek}(f) = \delta^f_{rek}(\epsilon_s(f)). \tag{2.26}$$

Medtem ko odpiranje z rekonstrukcijo najprej odstrani svetla območja z morfološkim krčenjem in nato izvede rekonstrukcijo s širjenjem, zapiranje z rekonstrukcijo najprej zlije vse temne regije s svetlejšo okolico z morfološkim širjenjem in zatem izvede rekonstrukcijo s krčenjem. Zapiranje z rekonstrukcijo φ_{rek} z oknom w_s tako definiramo kot:

$$\varphi^s_{rek}(f) = \epsilon^f_{rek}(\delta_s(f)). \tag{2.27}$$



(a) (b) (c) (d) (e)

Slika 2.7: Filtriranje (a) vhodne slike z (b) odpiranjem γ^1_{rek} in (c) zapiranjem φ^1_{rek} z rekonstrukcijo ter primerjava s tradicionalnim (d) morfološkim odpiranjem γ_1 in (e) zapiranjem φ_1.

Primerjavo med odpiranjem in zapiranjem z rekonstrukcijo ter tradicionalnim morfološkim odpiranjem in zapiranjem prikazuje primer na sliki 2.7. Odpiranje z rekonstrukcijo tako obnovi povezane komponente, ki jih morfološko krčenje ni v celoti odstranilo. Nasprotno pa zapiranje z rekonstrukcijo povrne obseg neodstranjenega ozadja ter tako prekine tiste povezane komponente, ki pred širjenjem niso bile združene.

- **Izmenično zaporedni filtri** (angl. alternating sequential filters) iz slike odstranijo tako svetle kot temne objekte. To izvedejo s postopnim povečevanjem okna med izmeničnim filtriranjem z odpiranjem in zapiranjem. Izmenično zaporedni filter IZF, ki najprej izvede morfološko odpiranje γ in odstrani objekte do velikosti s, formalno izrazimo kot:

$$\mathrm{IZF}_s^{\gamma}(f) = \varphi_s(\gamma_s(\varphi_{s-1}(\gamma_{s-1}(\dots\varphi_1(\gamma_1(f))\dots)))), \tag{2.28}$$

medtem ko IZF, ki najprej izvede morfološko zapiranje φ, formalno izrazimo kot:

$$\mathrm{IZF}_s^{\varphi}(f) = \gamma_s(\varphi_s(\gamma_{s-1}(\varphi_{s-1}(\dots\gamma_1(\varphi_1(f))\dots)))). \tag{2.29}$$

Primer operatorjev IZF_1^{γ} in IZF_1^{φ} prikazuje slika 2.8.

(a) (b) (c)

Slika 2.8: Filtriranje šuma iz (a) vhodne slike [48], kjer se filtriranje prične (b) z odpiranjem IZF_1^{γ} oziroma (c) z zapiranjem IZF_1^{φ}.

- **Ravnanje kupol** (angl. H-maxima) v sliki f zniža (oziroma odstrani) vse lokalne maksimume za vhodno vrednost $h \in \mathbb{R}^+$, tako da izvede rekonstrukcijo s širjenjem nad znižano sliko f. Operator ravnanja kupol RK definiramo kot:

$$\mathrm{RK}_h(f) = \delta^f_{rek}(f - h). \tag{2.30}$$

  RK_h tako odstrani vse povezane komponente, ki so največ za vrednost h višje od svoje okolice.

- **Izvleček kupol** (angl. dome extraction) [47] predstavlja cilindrično transformacijo na osnovi ravnanja kupol, ki omogoča identifikacijo lokalnih maksimumov slike f. Operator izvlečka kupol IK izpeljemo kot:

$$\mathrm{IK}_h(f) = f - \mathrm{RK}_h(f). \tag{2.31}$$

  Lokalni maksimumi so definirani kot množica pikslov p_i, za katere velja $f(p_i) - [\mathrm{RK}_h(f)](p_i) = h$. Slednje pa omogoča identifikacijo regijskih maksimumov (tako imenovanih platojev), ki so večji od enega piksla in jih zato ni mogoče identificirati zgolj z opazovanjem neposredne soseščine. Primer operatorjev ravnanja in izvlečka kupol prikazuje slika 2.9.

(a) (b) (c)

Slika 2.9: Ravnanje kupol na (a) vhodni sliki, kjer (b) odstranimo vse lokalne maksimume do višine 53, te pa (c) nato identificiramo z izvlečkom kupol.

- **Granulometrija** [49] (angl. granulometry) omogoča analizo velikosti objektov na sliki z beleženjem sprememb, ki jih povzroči vsak posamezen filter znotraj verige filtrov. Verigo filtrov G do velikosti s definiramo kot:

$$G_s = \{\Psi_0(f), \Psi_1(f), \ldots, \Psi_s(f)\}, \tag{2.32}$$

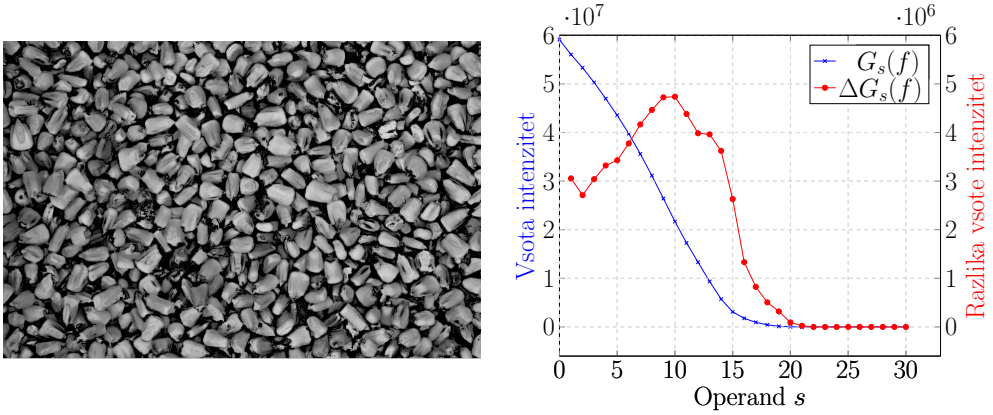
kjer Ψ označuje poljuben morfološki filter. Beleženje sprememb izvedemo s cilindrično transformacijo med posameznimi filtri, pripadajočo funkcijo sprememb pa imenujemo granulometrična funkcija ΔG (angl. pattern spectrum). Formalno definicijo granulometrične funkcije do velikosti s tako podamo kot:

$$\Delta G_s(f) = \{\Psi_0(f) - \Psi_1(f), \ldots, \Psi_{s-1}(f) - \Psi_s(f)\}. \tag{2.33}$$

Funkcija ΔG_s predstavlja dekompozicijo slike, medtem ko pripadajočo funkcijo velikosti objektov VO zapišemo kot:

$$\mathrm{VO}_s(f) = \sum_{p \in P} [\gamma_s(f)](p). \tag{2.34}$$

Funkcija VO_s iz vhodne slike z morfološkim odpiranjem najprej odstrani vse objekte, ožje od okna w_s, nato pa izmeri globalno intenziteto slike s seštevkom intenzitet vseh pikslov $p \in P$. Funkcija G tako beleži spremembe globalne intenzitete slike pri povečujočih se odpiranjih, medtem ko ΔG beleži spremembe med okni w_s in w_{s+1}, kot prikazuje slika 2.10. Posledično pa visoke vrednosti v funkciji ΔG nakazujejo, da vhodna slika f vsebuje objekte velikosti w_{s+1}.



(a) (b)

Slika 2.10: Granolumetrija nad (a) vhodno sliko, kjer (b) rezultat nakazuje na vsebnost objektov v velikosti med 19 in 21 piksli.

- **Transformacija domnevnih razdalj** [50] (angl. quasi-distance transformation) temelji na granulometrični funkciji in omogoča meritev razdalje med vsakim pikslom p_i in njemu najbližjim pikslom ozadja p_o [50]. Slednjega smatramo kot najbolj kontrasten piksel okolice. Razlike v intenziteti piksla za oceno kontrasta beležimo s cilindrično transformacijo povečujočih se krčenj, kjer velikost okna direktno poda razdaljo do piksla p_o. Meritev razdalje za piksel p_i do piksla p_o formalno definiramo kot:

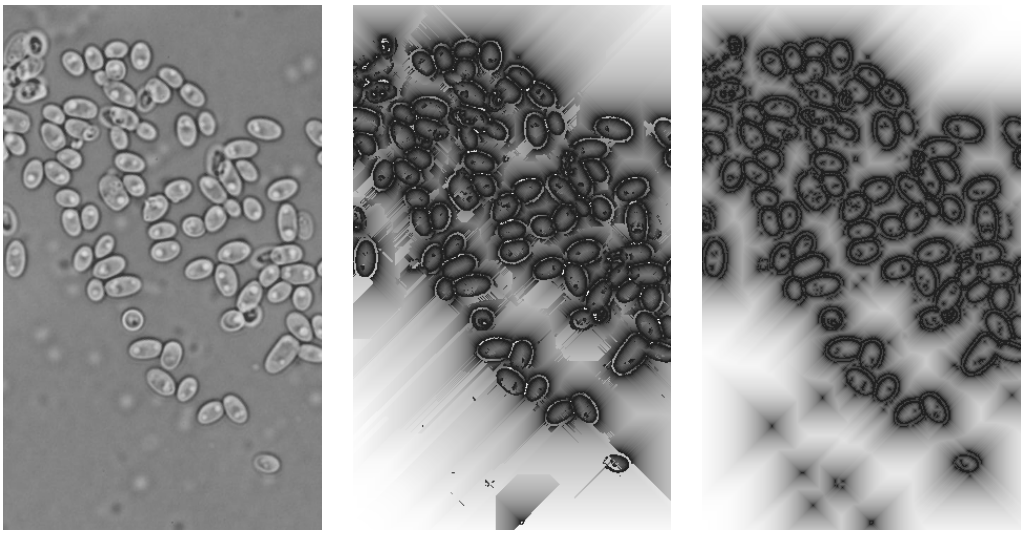
$$[d(f)](p_i) = \arg\max_s [\epsilon_s(f) - \epsilon_{s+1}(f)](p_i). \tag{2.35}$$

  Zaradi robnih pogojev, šuma v podatkih in morebitnih enakosti med spremembami, ki jih povzroči več filtrov, funkcija $d(f)$ ni zvezna (slika 2.11b). Zaradi celostne interpretacije je smiselno te nezveznosti odpraviti tako, da funkcijo $d(f)$ z enotskimi popravki prisilimo v 1-Lipschitzovo zveznost. Za piksel p_i to formalno izrazimo kot:

$$[\eta(f)](p_i) = \begin{cases} [\epsilon_1(f)](p_i)+1, & \text{če } [f-\epsilon_1(f)](p_i) > 1 \\ f(p_i), & \text{drugače,} \end{cases} \tag{2.36}$$

kar iterativno ponavljamo, dokler ne dosežemo stabilnega stanja oziroma limite. Rezultat, prikazan na sliki 2.11 c, imenujemo transformacija domnevnih razdalj $d_{L1}(f)$ z normo L_1. Formalno:

$$d_{L1}(f) = \lim_{s\to\infty} \underbrace{\eta(\dots(\eta}_{s \text{ krat}}(d(f)))\dots). \tag{2.37}$$



(a) (b) (c)

Slika 2.11: Transformacija domnevnih razdalj nad (a) vhodno sliko, kjer (b) nastale nezveznosti odpravimo in kot rezultat dobimo (c) L_1 normirano transformacijo razdalj.

2.4 Optimizacije

Podane definicije osnovnih in izpeljanih morfoloških operatorjev zahtevajo pogosta iskanja minimalnih in maksimalnih elementov znotraj relativno velikih množic števil in so zato računsko zahtevne. V tem podpoglavju zato podamo nekaj ključnih optimizacijskih tehnik, ki zmanjšajo število potrebnih primerjav na osnovi ponovne uporabe vmesnih rezultatov. Kot je pokazal Serra [51],

lahko osnovna filtra krčenja in širjenja s strukturnim elementom w_s izrazimo kot zaporedna filtriranja z enotskim strukturnim elementom w_1. To za morfološko krčenje ϵ formalno zapišemo kot:

$$\epsilon_s(f) = \underbrace{\epsilon_1(\epsilon_1(\dots\epsilon_1}_{s \text{ krat}}(f)\dots)). \tag{2.38}$$

Zaradi dualnosti to velja tudi za morfološko širjenje δ:

$$\delta_s(f) = \underbrace{\delta_1(\delta_1(\dots\delta_1}_{s \text{ krat}}(f)\dots)). \tag{2.39}$$

Časovna zahtevnost filtrov velikosti s se tako zmanjša z $\mathcal{O}((2s+1)^2-1) \simeq \mathcal{O}(s^2)$ na $\mathcal{O}(8s) \simeq \mathcal{O}(s)$. Zaporedno filtriranje pa ne deluje za morfološko odpiranje in zapiranje, saj sta ta operatorja idempotentna (enačba 2.12) in posledično $\gamma_s(f) \neq \gamma_{s-1}(\gamma_1(f))$. Vseeno pa lahko morfološko odpiranje γ izrazimo z verigo enotskih krčenj in širjenj:

$$\gamma_s(f) = \delta_s(\epsilon_s(f)) = \underbrace{\delta_1(\dots\delta_1}_{s \text{ krat}}(\underbrace{\epsilon_1(\dots\epsilon_1}_{s \text{ krat}}(f)\dots))\dots). \tag{2.40}$$

Podobno optimizacijo lahko izvedemo tudi v primeru morfološkega zapiranja φ:

$$\varphi_s(f) = \epsilon_s(\delta_s(f)) = \underbrace{\epsilon_1(\dots\epsilon_1}_{s \text{ krat}}(\underbrace{\delta_1(\dots\delta_1}_{s \text{ krat}}(f)\dots))\dots). \tag{2.41}$$

Bistveno zmanjšanje potrebnih računskih operacij dosežemo tudi z dekompozicijo elementarnega strukturnega elementa w_1 v enodimenzionalni vodoravni in navpični komponenti. Slednje dosežemo z:

$$w_1^x(p_i) = \{x_i - 1, x_i, x_i + 1\} \times \{y_i\}, \tag{2.42}$$

$$w_1^y(p_i) = \{x_i\} \times \{y_i - 1, y_i, y_i + 1\}. \tag{2.43}$$

Tradicionalno dvodimenzionalno krčenje ϵ_1 in širjenje δ_1 tako izrazimo kot kompozicijo enodimenzionalih krčenj in širjenj v poljubnem vrstnem redu:

$$\epsilon_1(f) = \epsilon_1^x(\epsilon_1^y(f)) = \epsilon_1^y(\epsilon_1^x(f)), \tag{2.44}$$

$$\delta_1(f) = \delta_1^x(\delta_1^y(f)) = \delta_1^y(\delta_1^x(f)). \tag{2.45}$$

Čeprav teoretična časovna zahtevnost za filter velikosti s v tem primeru ostaja nespremenjena, saj $\mathcal{O}(4s) \simeq \mathcal{O}(s)$, se število potrebnih primerjav za izračun vrednosti posameznega piksla ob uporabi okna w_1 zmanjša z 8 na 4, kar v praksi prepolovi izvajalne čase. Iz zgornjih definicij je očitno tudi, da je postopek dekompozicije enostavno razširljiv v poljubno število dimenzij.

Poleg teoretičnih optimizacij, ki izhajajo iz definicije operatorjev matematične morfologije, pa lahko učinkovitost postopkov izboljšamo tudi na implementacijskem nivoju. Pregled teh rešitev obravnavamo v naslednjem poglavju.

Poglavje 3

Analiza sorodnih raziskav

V tem poglavju podamo pregled stanja tehnike na področju učinkovitih pristopov k izračunu morfološkega krčenja in širjenja, ki sta osnova za implementacijo vseh podanih morfoloških in geodetskih operatorjev. Ker lahko filtriranja izvajamo na različnih računalniških sistemih, je učinkovitost izvajanja algoritmov odvisna tako od njihove primerne matematične zasnove kakor tudi prilagojene implementacije za posamezno tehnologijo. Flynn [52] je podal sledečo klasifikacijo računalniških arhitektur, ki temelji na številu ukaznih tokov **I** (angl. instruction streams) in podatkovnih tokov **D** (angl. data streams):

- **SISD** (angl. single instruction, single data stream) predstavlja tradicionalne enonitne sisteme. Ti so zmožni izvajati eno samo zaporedje ukazov, pri čemer procesna enota (PE) zaporedno obdeluje posamezne podatkovne elemente iz glavnega pomnilnika. V to kategorijo uvrščamo enojedrne CPE, kot je to na primer Intel Pentium 4. Kljub temu da slednji omogoča superskalarno procesiranje, kjer se na ukaznem nivoju lahko istočasno izvaja več neodvisnih ukazov, se sistem navzven obnaša kot tradicionalen SISD.
- **SIMD** (angl. single instruction, multiple data streams) je tipično realizi-

ran z razširitvijo ukaznega nabora računalniške arhitekture SISD, kot so to na primer nabor ukazov SSE (angl. Streaming SIMD Extensions) na arhitekturi x86, AVX (angl. Advanced Vector Extensions) na arhitekturi x86-64 in Neon na arhitekturi ARMv8. SIMD z enim ukazom istočasno izračuna več podatkovnih elementov, ki so hranjeni v tako imenovanih vektorskih registrih. V arhitekturi x86-64 je minimalna zagotovljena velikost vektorskega registra 128 bitov, pri čemer so MMX (angl. Matrix Math eXtension), SSE, AVX in AVX512 uvedli standarden ukazni nabor, ki upravlja s 64, 128, 256 in 512-bitnimi vektorskimi registri. Seveda pa je število elementov, ki jih lahko predstavimo na ta način, odvisno od velikosti uporabljenega podatkovnega tipa. Vektorskih arhitektur v tem delu ne obravnavamo, saj jih moderne CPE ne implementirajo.

- **MISD** (angl. multiple instructions, single data stream) nad vsakim vhodnim podatkom upravlja z več ukazi. Ti so izvedeni istočasno, tok podatkov med posameznimi ukazi pa je določen vnaprej. Zaradi istočasnega izvajanja pa implementacija vejitev zahteva hkraten izračun obeh vej, kar posledično omejuje stopnjo vzporednega procesiranja. Primer takšnega računalniškega sistema so cevovodne arhitekture, ki so realizirane na programirljivih čipih FPGA. To računsko arhitekturo zato najpogosteje zasledimo v specializiranih strojnih pospeševalnikih, uporabljenih v namenskih razširitvenih karticah [53, 54] in v obliki podenot znotraj CPE [55, 56]. Slednje omogoča bistveno razbremenitev CPE, saj lahko takšna namenska vezja opravljajo računsko zahtevne operacije, kot so na primer kodiranje in dekodiranje slik ter video in zvočnih posnetkov.
- **MIMD** (angl. multiple instructions, multiple data streams) zajema moderne heterogene CPE, ki podpirajo bodisi istočasno večnitno izvajanje (angl. simultaneous multithreading, SMT) znotraj enega jedra bodisi uporabo več jeder. V obeh primerih pa posamezna nit uporablja ukazni nabor tipa SISD, SIMD ali kombinacijo obeh.

Kljub relativni splošnosti podane delitve računalniških arhitektur modernih

GPGPU ne moremo umestimo v nobeno izmed opisanih kategorij. GPGPU namreč predstavlja posplošitev arhitekture SIMD na tako imenovano arhitekturo **SIMT** (angl. single instruction, multiple threads), ki omogoča več istočasnih izvedb posameznega niza ukazov [57]. Ker niti v tem primeru vodijo svoja lastna stanja, lahko vsaka izmed njih dostopa do lastnega pomnilniškega prostora, pri čemer pa vse vseeno izvajajo isti niz ukazov. Takšna posplošitev omogoča tudi uporabo vejitev, ki pa so lahko računsko zelo neučinkovite. Slednje namreč pogosto zahteva izvedbo ukazov v vseh možnih vejah. V času izvajanja se tako hkrati izvajajo samo tiste niti, ki sledijo specifični vejitvi, medtem ko se izvajanje preostalih niti začasno prekine. Čeprav je število procesnih enot in njihovo združevanje odvisno od uporabljenega GPGPU, danes tudi nizkocenovne izvedbe vsebujejo več kot tisoč jeder in tako omogočajo hkratno izvedbo velikega števila ukaznih tokov.

Z naraščajočimi potrebami po računski moči v zadnjih desetletjih se je bistveno povečala tudi heterogenost računalniških sistemov. Predstavljeno delitev računalniških arhitektur posledično danes dojemamo kot precej grobo. Čeprav moderne CPE umestimo v kategorijo MIMD, lahko implementacije algoritmov še vendar uporabljajo računske paradigme SISD, SIMD, MIMD ali poljubno kombinacijo le-teh. Uporaba SISD tako omogoča enostavno implementacijo izbranih algoritmov in nudi največjo fleksibilnost pri izvajanju. Seveda pa strogo sekvenčno izvajanje ukazov tako omejuje hitrost same izvedbe. Slednje lahko omilimo z uporabo ukazov SIMD, kjer vsak ukaz obdela več podatkovnih elementov in tako zmanjša skupno število potrebnih ukazov za izvedbo algoritma. Zaradi hkratne obdelave več elementov pa njihova uporaba omejuje splošnost vejitev in tipično zahteva, da so obdelani podatki v pomnilniku hranjeni zaporedno. Nasprotno implementacija algoritmov na arhitekturi MIMD predstavlja drugačne izzive, saj uporaba skupnega pomnilnika zahteva posebno skrb pri mednitni komunikaciji, ki lahko v nasprotnem primeru vodi v tekmovanje za podatki (angl. data race condition). Slednje se pojavi, ko več niti istočasno bere in piše v skupni pomnilnik in posledično povzroči, da posamezna nit prebere stare podatke ali pa prepiše spremembe drugih niti. Običajna

rešitev tega problema je uporaba binarnega semaforja (angl. mutex), s katerim omejimo istočasno izvajanje kritičnih sekcij na zgolj eno nit. Posledično algoritem postane delno sekvenčen in glede na Amdahlov zakon [58] omejuje največjo teoretično pohitritev, ki jo lahko dosežemo z večnitnim procesiranjem. Slednje je še posebej očitno pri izvedbi algoritmov na GPGPU, za katere je karakteristično veliko število jeder. Zaradi tega je še posebej pomembno, da uporabo GPGPU podpremo z razdrobljenim izračunom manjših in med seboj neodvisnih delnih izračunov. Samo s takšnim pristopom lahko ustvarimo tako imenovani popolnoma vzporedni izračun (angl. embarrassingly parallel), ki zahteva minimalno ali nič mednitne komunikacije.

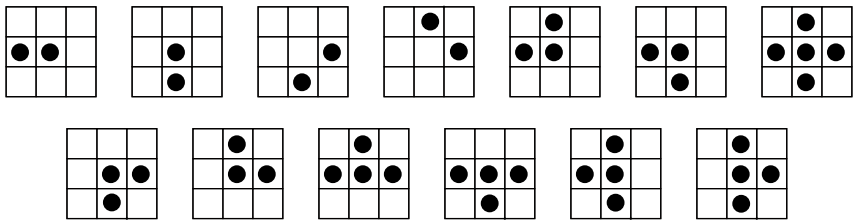
Skladno z opisanimi principi računalniških arhitektur v literaturi zasledimo tri ključne pristope, ki omogočajo zmanjšanje računske zahtevnosti operatorjev morfološkega krčenja in širjenja. Ti se osredotočajo na pohitritve izračuna:

- z razgradnjo strukturnega elementa v manjše enodimenzionalne strukturne elemente [27] ali v akorde [59] (angl. chords), s katerimi zmanjšamo količino potrebnih primerjav,
- s predobdelavo slike [28, 60], pri čemer nepomembne vrednosti zamenjamo s pomembnimi ter tako bistveno zmanjšamo število potrebnih primerjav na posamezen piksel, ter
- z uporabo učinkovitih podatkovnih struktur za sprotno filtriranje nepomembnih vrednosti brez predhodne obdelave [29].

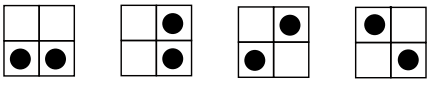
Čeprav vse naštete tehnike zmanjšajo število potrebnih primerjav za filtriranje vrednosti posameznega piksla, to dosežejo na bistveno drugačen način. Posledično so posamezne tehnike primernejše za realizacijo na specifičnih računalniških arhitekturah. V naslednjem podpoglavju zato podajamo pregled sorodnih del, ki omogočajo prilagoditev posameznih pristopov na učinkovito ukazno pretokovno izvajanje na CPE in GPGPU.

3.1 Metode ukazno pretokovnega procesiranja

V tem poglavju podamo pregled metod za učinkovit izračun krčenja in širjenja na CPE ter njihove prilagoditve za izvedbo na GPGPU. Prvi takšen pristop predstavlja optimalna razgradnja strukturnega elementa, ki sta jo z namenom učinkovite obdelave binarnih slik predstavila Zhuang in Haralick [61], Zhuang pa je ta pristop kasneje razširil za izvedbo sivinskih operatorjev [62]. Metoda razgradi poljuben strukturni element na množico manjših, pri čemer vsak operira nad zgolj dvema nepovezanima vhodnima piksloma. Na tej osnovi je Xu [27] demonstriral način, po katerem lahko poljuben 8-povezani konveksni strukturni element razgradimo v omejen nabor trinajstih 8-povezanih strukturnih elementov velikosti 3×3, ki jih prikazuje slika 3.1. Normand [63] je teh trinajst strukturnih elementov posplošil na nabor zgolj štirih, ki so prikazani na sliki 3.2. Čeprav je bila tudi ta metoda v svoji zasnovi omejena na obdelavo binarnih slik, je Déforges [30] uspel predstavljeno metodologijo posplošiti na sivinske operatorje.



Slika 3.1: Minimalni nabor strukturnih elementov velikosti 3×3, s katerimi je mogoče tvoriti poljuben 8-povezan konveksen strukturni element.

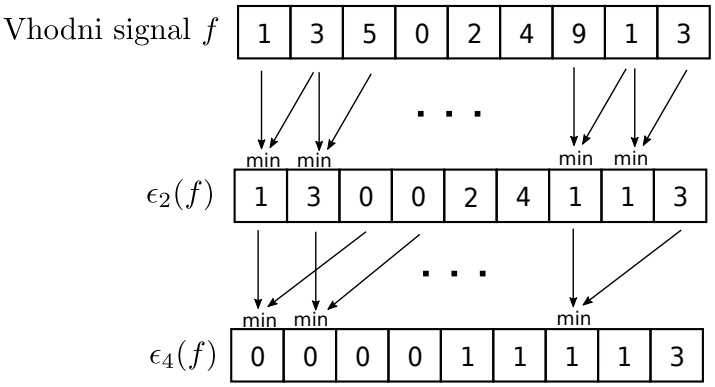


Slika 3.2: Nabor strukturnih elementov velikosti 2×2, ki posplošijo Xujev nabor strukturnih elementov s slike 3.1.

Takšen pristop razgradnje strukturnih elementov zmanjša število potrebnih primerjav in stopnjo naključnega dostopa do glavnega pomnilnika. Slednje

tako namreč zamenjajo pretežno zaporedni dostopi, kar bistveno izboljša izkoriščenost predpomnilnika CPE. Vseeno pa je izvedba velikih strukturnih elementov časovno potratna, saj zahteva večje število iteracij. Posledično je v teh primerih še posebej pomembna učinkovitost implementacije manjših strukturnih elementov. V odprtokodnih knjižnicah OpenCV, SMIL, Fulguro [64] in MAMBA [65] (angl. MAthematical Morphology liBrAry) zasledimo optimizirane implementacije različno oblikovanih manjših strukturnih elementov, ki za pohitritev uporabljajo impliciten SIMD in večnitno procesiranje.

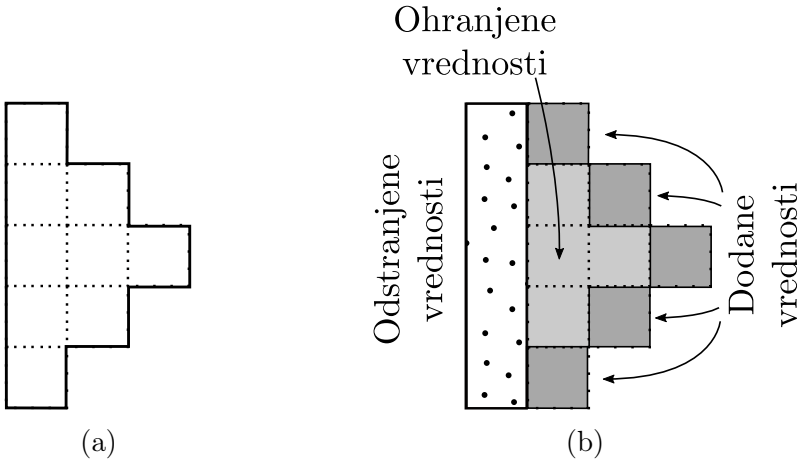
Nasprotno pa je Pecht [26] predstavil postopek, ki omogoča izboljšanje računske zahtevnosti izvedbe operatorjev z velikimi strukturnimi elementi, ki temelji na eksponentni razgradnji enodimenzionalnega strukturnega elementa. Z uporabo delnih rezultatov takšna dekompozicija zniža časovno zahtevnost direktnega izračuna vrednosti piksla z $\mathcal{O}(s-1)$ na $\mathcal{O}(\log_2(s))$. Zaradi eksponentne dekompozicije pa je metoda omejena na strukturne elemente velikosti potence 2. Dvodimenzionalni strukturni elementi postanejo omejeni na pravokotne strukturne elemente s stranicami potence 2, saj so realizirani s kompozicijo. Primer krčenja ϵ_4^x in uporabe delnih rezultatov je podan na sliki 3.3.



Slika 3.3: Eksponentna razgradnja strukturnega elementa w_4, kjer z vsako iteracijo realiziramo naslednjo potenco velikosti okna.

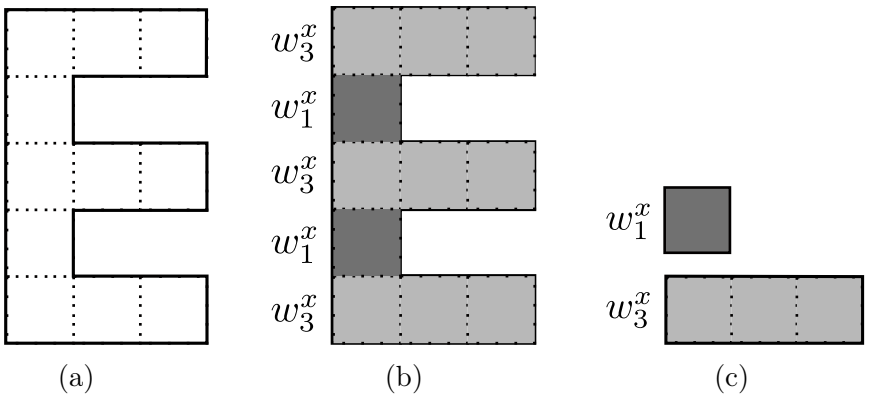
Za učinkovit izračun morfoloških filtrov s poljubnim strukturnim elementom je Van Droogenbroeck s sodelavci [66] predlagal uporabo histograma. Zasnova algoritma izhaja iz dejstva, da premik strukturnega elementa na sosednji pik-

sel ohrani večino vrednosti, ki jih zahteva izračun pripadajoče vrednosti. Histogram pri tem hrani vse vrednosti pikslov, ki se nahajajo znotraj preseka položajev strukturnega elementa. Kot prikazuje slika 3.4, se v vsaki iteraciji tako nadomestijo zgolj vrednosti pikslov izven preseka prejšnjih položajev strukturnega elementa z vrednostmi znotraj preseka trenutnih položajev. Ker so premiki strukturnega elementa omejeni na sosednje piksle, je tako potrebno vsako drugo vrstico obravnavati v nasprotni smeri. Takšen pomnilniški dostop pa se v praksi izkaže za neučinkovitega, kar predstavlja ključno omejitev metode. Zaradi uporabe histograma je metoda prav tako omejena na podatkovne tipe z manjšo zalogo vrednosti, kot sta to na primer `char` in `short`.



Slika 3.4: Z uporabo histograma in (a) poljubnega strukturnega elementa lahko v premiku na naslednji piksel (b) odstranimo (oziroma dodamo) samo tiste piksle, ki jih strukturni element več ne pokriva (oziroma jih na novo pokriva).

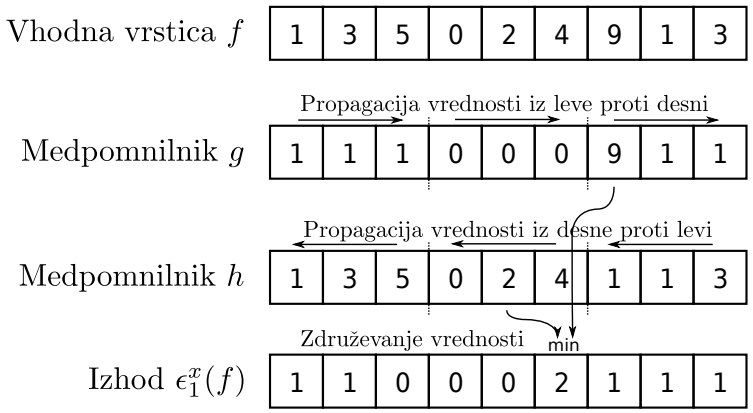
Urbach in Wilkinson [59] sta ti slabosti odpravila z metodo UW, ki je uvedla uporabo akordov. Akordi so manjši enodimenzionalni strukturni elementi, ki skupaj sestavljajo večji dvodimenzionalen strukturni element. Primer takšne dekompozicije prikazuje slika 3.5. V času izvajanja je vhodna slika obdelana zgolj z omejenim naborom unikatnih akordov, iz tako pridobljenih delnih rezultatov pa je mogoče ovrednotiti končne vrednosti posameznih pikslov. Moreaud in Itthirad [67] sta metodo prilagodila tudi za učinkovit izračun na GPGPU.



Slika 3.5: (a) Poljuben strukturni element, v katerem so (b) identificirani vsi akordi in za filtriranje slike so uporabljeni samo (c) unikatni akordi.

Časovna zahtevnost izračuna filtrirane vrednosti posameznega piksla je v do sedaj navedenih primerih neposredno odvisna od velikosti uporabljenega strukturnega elementa, ki je lahko poljubne oblike. Vseeno pa v praksi večinoma uporabljamo omejeno množico simetričnih strukturnih elementov, ki nam zagotavljajo neodvisnost rezultata od orientacije objektov na sliki. Takšna omejitev strukturnega elementa omogoča njegovo dekompozicijo na enodimenzionalne strukturne elemente. V kolikor ti sledijo črti, dobljeni z Brensenhamovim algoritmom [68], lahko imajo tudi poljubno orientacijo [69]. Kompozicija tako pridobljenih delnih rezultatov pa podaja dovolj širok nabor oblik večdimenzionalnih strukturnih elementov za praktično uporabo. Tovrstne operatorje lahko izvedemo s hitrostjo $\mathcal{O}(1)$ na piksel, saj časovna zahtevnost ni odvisna od velikosti uporabljenega strukturnega elementa. Prvi in hkrati najbolj poznan pristop k izračunu morfološkega širjena in krčenja s poljubno velikim enodimenzionalnim strukturnim elementom v konstantnem času predstavlja metoda HGW, ki so jo hkrati predstavili Herk [28] ter Gil in Werkman [60]. Pri tem sta za izračun uporabljena dva medpomnilnika, g in h, ki sta kopiji vhodne vrstice in sta razdeljena v segmente velikosti $2s + 1$. V primeru krčenja so minimalne vrednosti vsakega segmenta v medpomnilniku g (oziroma h) propagirane od leve proti desni (oziroma desne proti levi). Končna vrednost vsakega piksla je nato določena z unikatno kombinacijo dveh vrednosti iz g in h. Primer tvor-

jenja medpomnilnikov in končnega rezultata je podan na sliki 3.6. Metoda je bila izvedena tako z uporabo SIMD [70], kakor tudi z GPGPU [71, 72]. Gil in Kimmel [73] sta predstavila metodo GK. Ta izboljša fazo propagiranja metode HGW, tako da s povečanjem kompleksnosti algoritma zmanjša povprečno število potrebnih primerjav na piksel. Nadaljnje izboljšave pri implementaciji velikih strukturnih elementov pa sta predstavila Yuan in Mikhail [74], ki sta ponovno identificirala in odpravila odvečne primerjave ob propagiranju medpomnilnikov g in h. Čeprav je metoda teoretično bolj učinkovita od metode GK, se v praksi algoritem ne izkaže za učinkovitega, kadar so primerjave med vrednostmi pikslov strojno pospešene.



Slika 3.6: Primer krčenja ϵ_1^x z metodo HGW, kjer sta najprej obdelana medpomnilnika g in h, nato pa uporabljena za enolično določitev končnih izhodnih vrednosti.

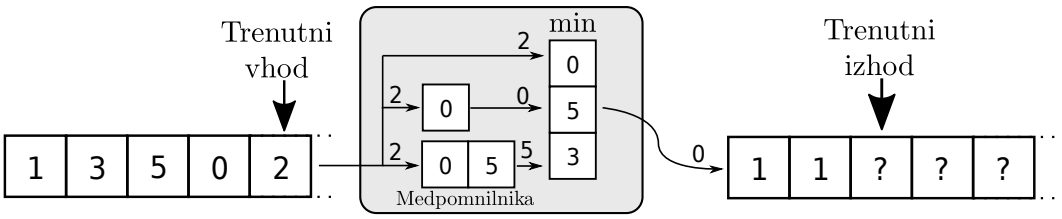
Vseeno pa tako metoda HGW kakor tudi vse njene kasnejše izboljšave pri obdelavi večdimenzionalnih podatkov zahtevajo več iteracij. To slabost sta odpravila Dokládal in Dokladalova [29] z metodo DD, ki za izračun uporablja vrsto [75] (angl. queue). Tako kot HGW tudi ta pristop v osnovi omogoča obdelavo zgolj enodimenzionalnih podatkov, pri čemer pa večdimenzionalne podatke obdelujemo s kompozicijo delnih rezultatov. Vrsta je uporabljena za pomnjenje položaja in vrednosti tako imenovanih pomembnih vrednosti, ki predstavljajo potencialen rezultat operatorja v nadaljnjih iteracijah. V primeru krčenja je vrsta v vsaki iteraciji posodobljena na sledeči način:

1. Z začetka vrste so odstranjene vse vrednosti, ki imajo višjo ali enako vrednost kot naslednji piksel.
2. Vrednost naslednjega piksla in njegov položaj sta dodana v začetek vrste.
3. V kolikor je relativen položaj vrednosti na koncu vrste večji od velikosti strukturnega elementa, je vrednost odstranjena.
4. V kolikor smo prebrali vsaj toliko vrednosti, kot je velik strukturni element, vrednost na koncu vrste predstavlja izhodni rezultat.

Avtorji so kompozicijo dvodimenzionalnega strukturnega elementa realizirali tako, da izhod obdelave posamezne vrstice predstavlja neposreden vhod v vrsto, ki opravlja obdelavo stolpca. Izhod obdelave stolpca nato določa končno vrednost piksla. Višje dimenzije so realizirane po enakem principu neposredne kompozicije. Metoda tako za poljubno velik strukturni element potrebuje le en zaporeden prehod slike, pri čemer vrednost vsakega piksla preberemo le enkrat. Zaradi uporabe podatkovne strukture realizacija metode na CPE zahteva relativno veliko število ukazov. Posledično pa se njena enonitna implementacija izkaže za slabšo od metode HGW [29]. To slabost je odpravil Karas [76], ki je metodo prilagodil za optimalen izračun enodimenzionalih operatorjev na GPGPU, pri čemer je dosegel tudi do 50-kratne pohitritve v primerjavi s CPE.

3.2 Metode podatkovno pretokovnega procesiranja

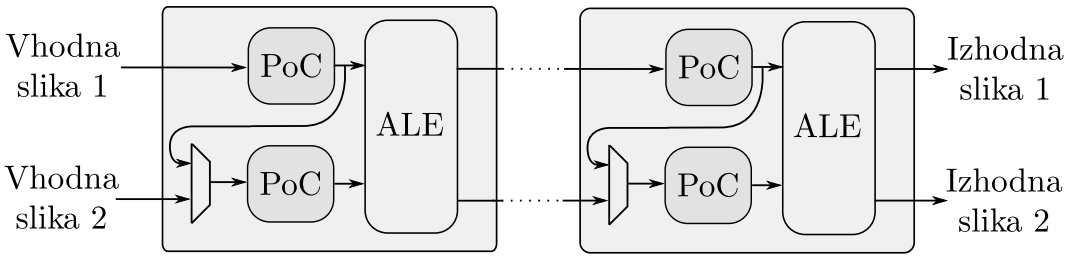
V tem podpoglavju podamo pregled metod, ki uporabljajo namensko strojno opremo za filtriranje z morfološkimi filtri. Čeprav lahko v literaturi zasledimo tudi uporabo specializiranih čipov ASIC (angl. application-specific integrated circuit) [77, 78], se osredotočimo zgolj na metode, ki uporabljajo programirljive čipe FPGA. V primerjavi z ASIC je njihova največja prednost relativno



Slika 3.7: Primer krčenja ϵ_1^x, kjer so uporabljeni medpomnilniki za dostop do starejših vrednosti toka. Zamik na izhodu je posledica velikosti strukturnega elementa.

nizka cena in zmožnost ponovnega programiranja čipa, kar omogoča hiter razvoj specializiranih arhitektur [79]. Ker v nasprotju s CPE in GPGPU čipi FPGA običajno ne dostopajo neposredno do zunanjega pomnilnika, je tovrsten način procesiranja v grobem deljen v dve fazi. Prva faza opravi inicializacijo sintetizirane arhitekture, ki jo je mogoče prilagajati z uporabniško nastavljivimi parametri. V drugi fazi pa se izvaja procesiranje, kjer čip obdela tok podatkov, kot je to na primer slika. V tej fazi prejme čip en vhodni piksel vsak urin cikel in po zasičenosti cevovoda na izhodu poda filtrirane vrednosti pikslov v istem zaporedju, kot so bili poslani vanj. Posledično je naključen dostop v zaporednem toku precej omejen, saj zahteva uporabo medpomnilnikov za začasno hranjenje vrednosti v toku. Primer slednjega prikazuje slika 3.7. Za realizacijo filtriranja z uporabo FGPA so tako najbolj pogosto uporabljene tehnike razgradnje, ki ne zahtevajo velike stopnje naključnega dostopa znotraj toka, in tehnike, ki popolnoma odstranijo naključen dostop do toka z uporabo učinkovitih podatkovnih struktur. Slednje so še posebej pomembne, saj zaradi prostorske paralelnosti njihova relativno visoka kompleksnost implementacije ne predstavlja težave iz vidika računske učinkovitosti.

V splošnem so bile tehnike razgradnje velikih strukturnih elementov v manjše (običajno velikosti 3×3) razvite ravno za uporabo na namenski strojni opremi [27]. Glavni razlog za to se skriva v omejenih kapacitetah pomnilnika in programirljivih logičnih vrat, ki preprečujejo neposreden izračun velikih strukturnih elementov [81, 82]. Nasprotno pa med izvedbo filtrov z majhnimi

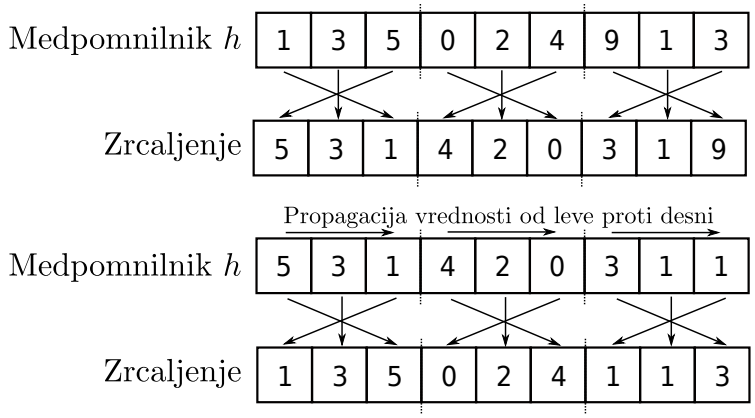


Slika 3.8: Clientijev [80] morfološki cevovod, kjer vsaka podenota PoC podpira krčenja ali širjenja s poljubnim strukturnim elementom velikosti 3×3.

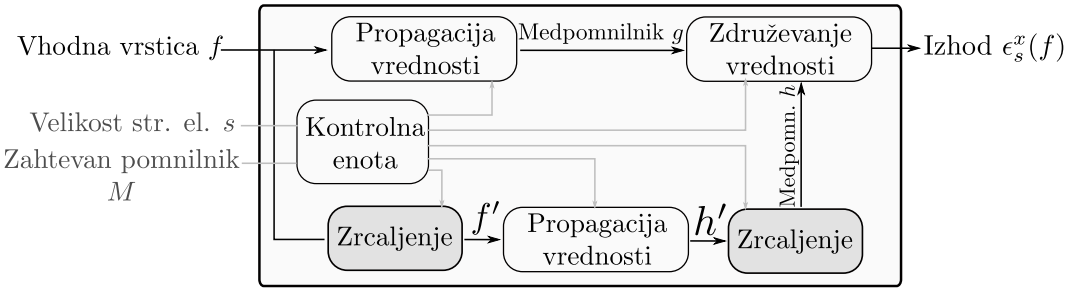
strukturnimi elementi vsak izmed njih hrani le eno do dve vrstici slike znotraj cevovoda. Tako je Clienti [80] predlagal programirljiv morfološki cevovod, kjer je vsaka podenota PoC zmožna opraviti krčenja in širjenja s poljubnimi strukturnimi elementi velikosti 3×3. Slika 3.8 prikazuje primer arhitekture takšnega cevovoda, kjer vsak filter nastopa kot samostojna enota, medtem ko aritmetično-logična enota (ALE) podpira zgolj osnovne relacijske operacije, seštevanje, odštevanje in izbor minimalne ali maksimalne vrednosti. Takšen pristop omogoča tudi izračun elementarnega geodetskega krčenja in širjenja in posledično izračun geodetskih operatorjev. Ob omejitvi na 8-povezane konveksne strukturne elemente se lahko omejimo na samo 13 strukturnih elementov velikosti 3×3, kot je demonstriral Xu [27]. S ciljem še kompaktnejše implementacije operatorjev na čipih FPGA je Déforges [30] predlagal posplošitev na le štiri strukturne elemente. Oba nabora strukturnih elementov sta prikazana na slikah 3.1 in 3.2.

Na predstavljeni osnovi je Client razvil arhitekturo, ki je zmožna izračunati enodimenzionalen HGW [83] na FPGA. Osnovno ozko grlo tega pristopa predstavlja medpomnilnik h, saj propagacija vrednosti pikslov v segmentih poteka v nasprotni smeri, kot so ti podani v podatkovnem toku. To je zaobšel z zrcaljenjem vrednosti segmentov pred in po propagaciji, ki jo je tako lahko opravil od leve proti desni. Primer predstavljenega postopka je podan na sliki 3.9, medtem ko predlagano arhitekturo vidimo na sliki 3.10.

Bartovskỳ s sodelavci [84] je predlagal alternativno rešitev v obliki morfološke



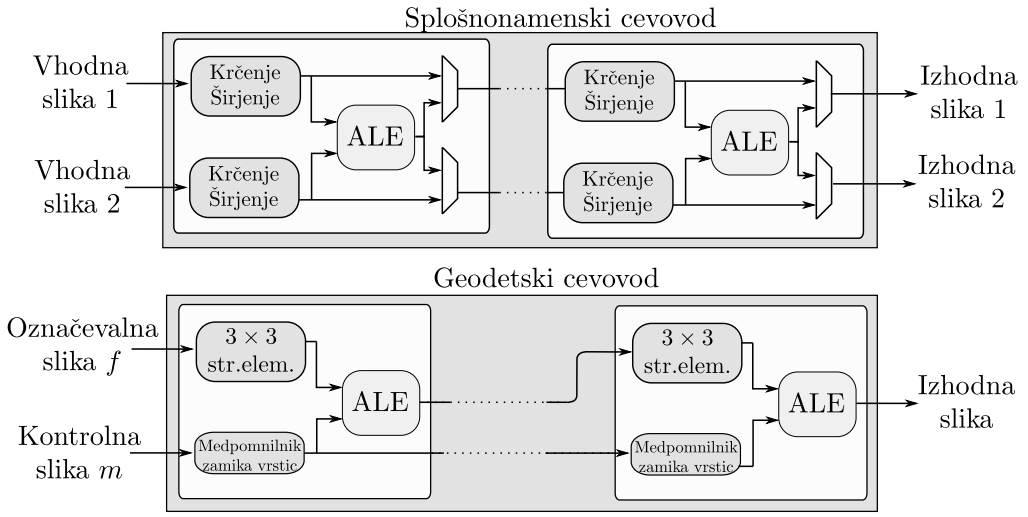
Slika 3.9: Zrcaljenje segmentov medpomnilnika h omogoči implementacijo algoritma HGW tudi na FPGA [83], saj propagacija vrednosti nato poteka od leve proti desni.



Slika 3.10: Pregled predlagane arhitekture HGW na FPGA [83].

soprocesne enote MSPE (angl. morphological co-processing unit for embedded devices), ki jo sestavljata dva procesna cevovoda (glej sliko 3.11). Pri tem prvi cevovod obravnava splošnonamensko filtriranje z velikimi strukturnimi elementi in je izveden z metodo DD [29]. Predlagana arhitektura omogoča uporabniško nastavljive ALE in posledično implementacijo poljubnih filtrov, ki pa so zaradi omejenih virov uporabljenega čipa FPGA omejeni na največjo velikost strukturnega elementa 31×31. Takšen splošnonamenski cevovod pa je vseeno neučinkovit za izvedbo iterativnega postopka rekonstrukcije, ki uporablja strukturne elemente le velikosti 3×3. Optimizacijo iz tega vidika so avtorji dosegli z namenskim geodetskim cevovodom, ki opravi neposreden izračun krčenja oziroma širjenja velikosti 3×3. Njegova sinteza tako porabi bi-

stveno manj programirljivih logičnih vezij, posledično pa je dolžina izvedenega geodetskega cevovoda daljša od splošnonamenskega. To je računsko ugodno, saj postopek rekonstrukcije zahteva iterativen izračun dolgih verig filtrov.



Slika 3.11: Pregled predlagane arhitekture MSPE na FPGA [84].

Vseeno pa je iz predstavljenega stanja tehnike razvidno, da poznane metode, namenjene za izvedbo na ukazno pretočnih arhitekturah, ne obravnavajo optimizacij izračuna zaporedno odvisnih elementarnih filtrov, saj se osredotočajo predvsem na učinkovit izračun posameznih filtrov. Implementacije v odprtokodnih knjižnicah, kot sta SMIL in OpenCV, sicer omogočajo učinkovito filtriranje z enotskimi okni različnih oblik in tvorbo filtrov z večjimi strukturnimi elementi na osnovi kompozicije delnih rezultatov. Vseeno pa te knjižnice realizirajo naiven vzporeden izračun, pri čemer razdelijo vhodno sliko na primerno število neprekrivajočih se delov, ki jih obdelamo vzporedno. Takšno naivno vzporedno procesiranje pa ob večjem številu niti hitro izčrpa pasovno širino glavnega pomnilnika, kar vodi v slabše računske čase. Nasprotno implementacije na podatkovno pretokovnih procesnih enotah izkoriščajo prostorski paralelizem. Ta omogoča filtriranje vhodnih slik z več filtri v enem samem prehodu. Število filtrov, kompleksnost uporabljene metode in podprti strukturni

elementi pa so odvisni od implementirane arhitekture in omejenih kapacitet uporabljenega čipa FPGA.

Tako danes ne poznamo metode učinkovitega izračuna verige elementarnih filtrov, ki bi bila prilagojena za ukazno pretokovne enote. Prav nasprotno, takšne metode so realizirane na podatkovno pretokovnih enotah, ki omogočajo vzpostavitev poljubne računske arhitekture. V naslednjem poglavju zato predstavimo algoritem za učinkovit izračun verige elementarnih morfoloških filtrov na centralni procesni enoti.

Poglavje 4

Algoritem za učinkovit izračun verige morfoloških filtrov

V tem poglavju predstavimo algoritem, ki omogoča učinkovit izračun geodetskih operatorjev. Ker izračun zahteva dolge verige zaporedno odvisnih elementarnih filtrov ϵ_1^m in δ_1^m, učinkovitost metode izhaja iz njene zmožnosti lokalizacije procesiranih podatkov in ponovne uporabe informacij. V zasnovi algoritma upoštevamo implicitne mehanizme modernih CPE, ki povečajo računsko moč, na primer z napovedovanjem vejitev (angl. branch predictor) ter vnaprejšnjim branjem pomnilnika (angl. hardware based memory prefetching). Za elementarne filtre tako uporabimo neposreden izračun brez uporabe podatkovnih struktur. Slednje pri manjših oknih bistveno povečajo implementacijsko kompleksnost metode, kar vodi v povečano število ukazov in vejitev. Izračun filtrov pri tem zahteva, da ga priredimo na striktno zaporeden dostop do slike, kar omogoča trivialno vnaprejšnje branje. Algoritem prav tako minimizira potrebno pasovno širino glavnega pomnilnika, ne glede na uporabljeno število jeder. Slednje namreč pogosto predstavlja ozko grlo sorodnih metod, ki sicer izkazujejo teoretično nizko časovno zahtevnost (rezultati primerjav so podani v podpoglavju 5.4). Pomembne pohitritve dosežemo tudi z uporabo

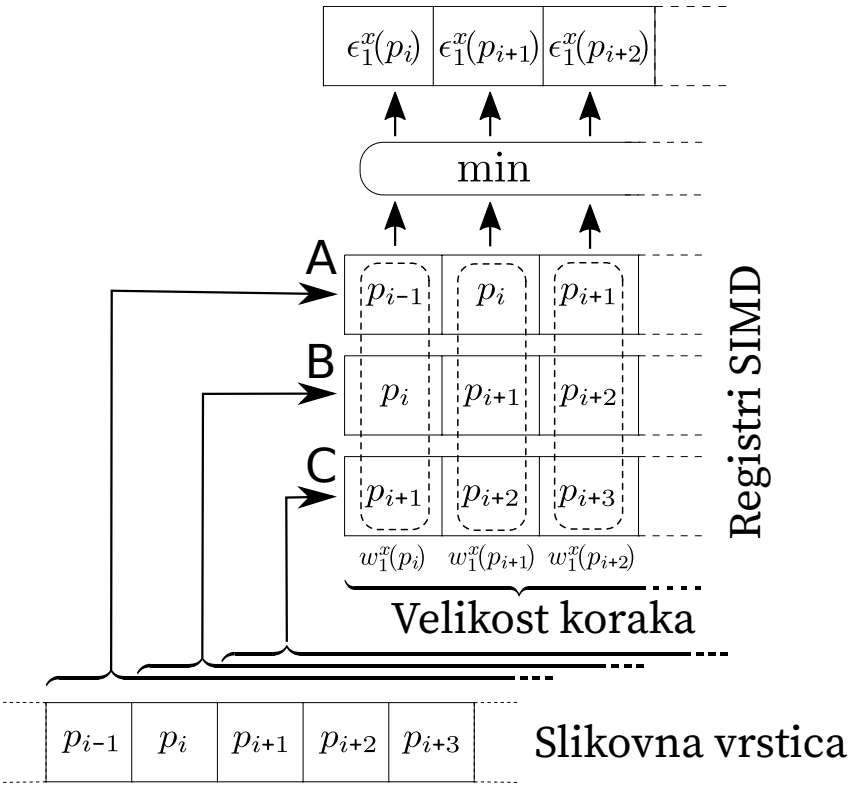
ukazov SIMD ter z večjedrnim izračunom. Predlagani algoritem predstavimo v naslednjih korakih:

1. **Implementacija elementarnih filtrov z ukaznim naborom AVX2** za pospešitev izračuna enotskega krčenja.
2. **Procesiranje na mestu**, s čimer zmanjšamo potreben pomnilniški prostor in posledično potrebno pasovno širino.
3. **Pretočno procesiranje**, kjer filtriramo celotno sliko f s striktno zaporednim prehodom skozi vse piksle.
4. **Večjedrni izračun** verige morfoloških filtrov.
5. **Analiza topologije CPE za optimizacijo odzivnosti in prepustnosti**, kjer s pripenjanjem niti optimiziramo pretok podatkov na CPE.
6. **Razširitev na rekonstrukcijske filtre**, kjer verigo filtrov tvorimo v času zagona, saj ni znana vnaprej.
7. **Razgradnja morfoloških filtrov na verigo elementarnih filtrov**, kadar obravnavamo bolj kompleksne filtre.

4.1 Implementacija elementarnih filtrov z ukaznim naborom AVX2

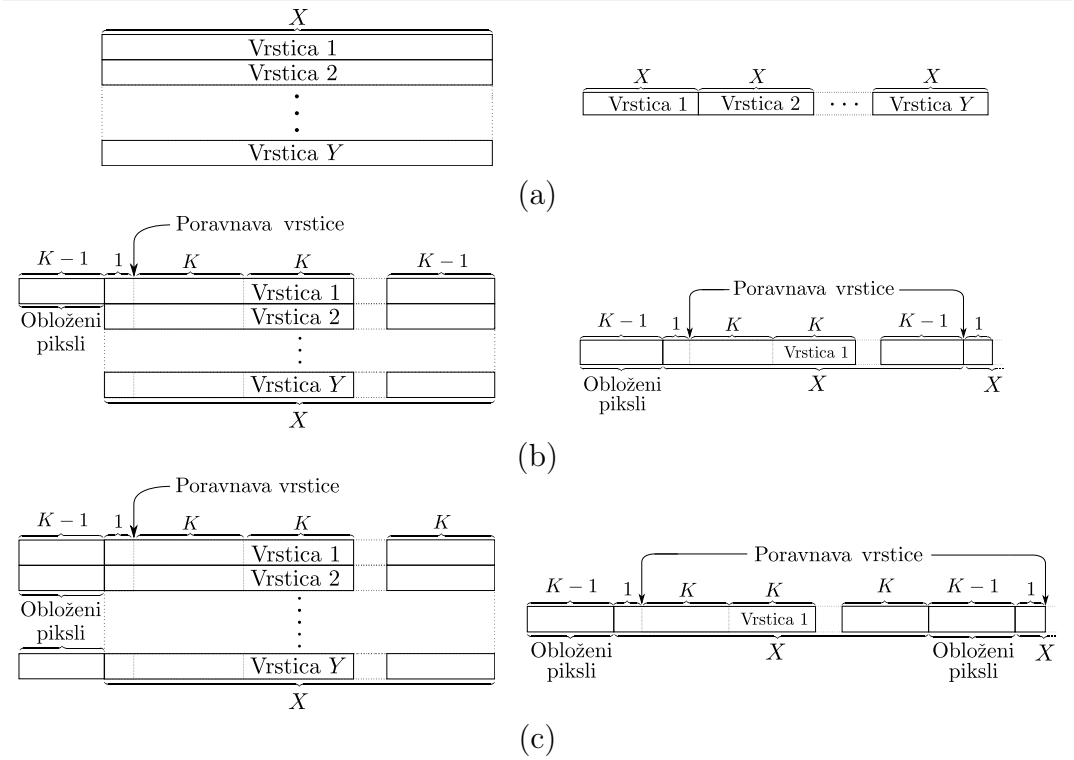
Direkten izračun ϵ_1 ločimo na filtra ϵ_1^x in ϵ_1^y. Izračun slednjih lahko enostavno pohitrimo z ukazi SIMD. Število istočasnih izračunov K je odvisno od velikosti podatkovnega tipa V_t, ki ga uporabimo za predstavitev posameznega piksla, ter velikosti uporabljenih registrov SIMD V_r. Oboje je v arhitekturi x86-64 zakodirano v samem ukazu zbirnika, kar pomeni, da vsaka sprememba podatkovnega tipa ali velikosti registra zahteva ločeno oziroma namensko implementacijo. Zaradi preprostosti pa se v opisu metode omejimo zgolj na elementarno

krčenje, ukazni nabor AVX2, ki omogoča uporabo registrov z $V_r = 256$ biti, ter na podatkovni tip `double`, ki obsega $V_t = 8$ zlogov. Korak premika skozi piksle slike oziroma število pikslov na register je v tem primeru $K = V_t/V_r = 4$. Izračun $[\epsilon_1^x(f)](p_i)$ opravimo s tremi vektorskimi registri A, B in C. Pri tem v registru A vzdržujemo vrednosti iz intervala $[p_{i\text{-}1}, p_{i+2}]$, medtem ko registra B in C hranita vrednosti na intervalih $[p_i, p_{i+3}]$ in $[p_{i+1}, p_{i+4}]$. Izračun minimalne vrednosti med registri tako omogoča istočasni izračun rezultata filtriranja $\epsilon_1^x(f)$ na intervalu $[p_i, p_{i+3}]$, kot je prikazano na sliki 4.1. Na podoben način lahko izvedemo tudi izračun vrednosti $\epsilon_1^y(f)$, pri čemer pa pomike med stolpci zamenjamo s premiki med vrsticami. Tudi v tem primeru istočasno izračunamo vrednosti štirih elementov in tako efektivno znižamo število potrebnih primerjav na piksel za faktor 4 oziroma na 25 %. Vsi dostopi do glavnega pomnilnika so posredni, saj so podatki najprej preneseni v medpomnilnik CPE, nalaganje v registre pa je realizirano preko najvišjega medpomnilnika L1. Dostop do slednjega je razdeljen na podatkovne linije, ki danes obsegajo 64 zlogov oziroma 512 bitov. Poljubno branje in zapisovanje znotraj iste podatkovne linije je prav tako hitro, medtem ko je operacija, ki posega v dve sosednji liniji, do dvakrat počasnejša. Zato v predlagani metodi uporabimo poravnavo vrstic slike za zagotavljanje učinkovitega dostopa do pomnilnika. Hkrati dosežemo rešitev robnih primerov na začetku in koncu vsake slikovne vrstice, kjer obseg filtra sega čez rob slike. Tradicionalni pristop, pri katerem obložimo rob slike z dodatnimi piksli, v našem primeru namreč ni primeren, saj zahteva določanje obsega vrednosti, ki zagotavljajo nespremenjen rezultat filtriranja. Takšna vrednost ne obstaja za morfološke filtre. Če namreč filtriramo s krčenjem, bi obložene piksle morali nastaviti na maksimalno možno vrednost. Nasprotno pa bi morali biti v primeru širjenja nastavljeni na minimalno možno vrednost, ki jo uporabljen podatkovni tip lahko zasede. Ker obojega ni možno zagotoviti hkrati, je potrebno prvi in zadnji element vsake vrstice izračunati s skalarnim procesiranjem. Posledično pa za optimalen vektorski izračun sliko pomnilniško poravnamo na drugi piksel vsake vrstice, kot prikazuje slika 4.2, kjer za faktor poravnave uporabimo velikost vektorskega registra V_r.



Slika 4.1: Okno $w_1^x(p_i)$ poda pomnilniški naslov, iz katerega preberemo vrednosti registrov. Izračun minimalne vrednosti med registri nam, kadar je $K = 4$, poda istočasen izračun vrednosti $\epsilon_1^x(f)$ za piksle na intervalu $[p_i, p_{i+3}]$.

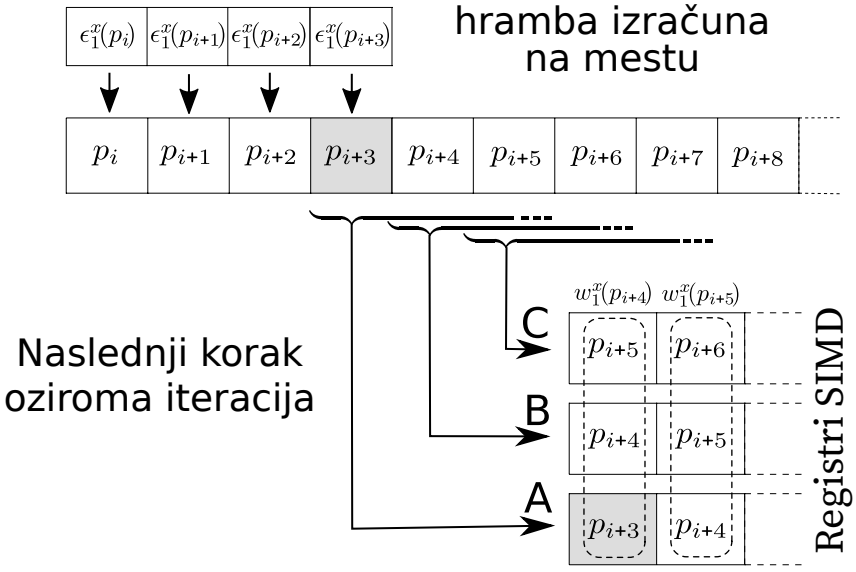
Optimalno pomnilniško učinkovitost tako dosežemo, kadar je širina slike enaka $X = K * n$, saj nam med vrstice slike ni treba dodajati obloženih pikslov. Nasprotno pa najslabši primer predstavljajo širine slike $X = K * n + 1$, saj moramo vsako novo vrstico začeti s $K - 1$ obloženimi piksli. V splošnem torej vsako vrstico razširimo na $X_{aligned} = ceil(X/K) * K$ pikslov oziroma na večkratnik velikosti vektorskega registra V_r. Čeprav začetek in konec vsake vrstice izračunamo skalarno, se večina izračunov še vedno izvede z uporabo SIMD. Posledično pa je, neglede na uporabljen podatkovni tip piksla, bistveno zvišana potreba po pomnilniški pasovni širini, saj vedno operiramo z V_r biti. V naslednjem podpoglavju predstavimo način, po katerem lahko to ublažimo z uporabo izračuna na mestu.



Slika 4.2: (a) Vhodna slika ter njena linearna pomnilniška predstavitev, kjer lahko glede na širino X opravimo (b) optimalno poravnavo slike z minimalno dodatno porabo pomnilnika, drugače je potrebno začetku vsake vrstice dodati obložene piksle, kjer jih (c) v najslabšem primeru potrebujemo $K - 1$.

4.2 Procesiranje na mestu

Izračuna $\epsilon_1^x(f)$ ni možno hraniti neposredno na mestu, saj bi tako prepisali vrednosti, ki jih potrebujejo sosednja okna. Kot prikazuje slika 4.3, moramo po uporabi okna w_1^x v okolici piksla p_i ohraniti prvotno vrednost piksla p_{i+K-1}, saj slednjo zahteva register A ob uporabi okna w_1^x v okolici piksla p_{i+K}. Zato naprej pripravimo register A z novimi vrednostmi, ki vsebuje prekrivajočo vrednost, ter šele zatem shranimo izračunane minimalne vrednosti na mestu. Ker uporabljamo okno w_1^x, pa lahko registre B in C naložimo kasneje, saj



Slika 4.3: Preden shranimo vrednosti izračuna na mestu, moramo prekrivajočo, tj. sivo vrednost ohraniti oziroma naložiti v register, saj je potrebna za naslednji korak izračuna.

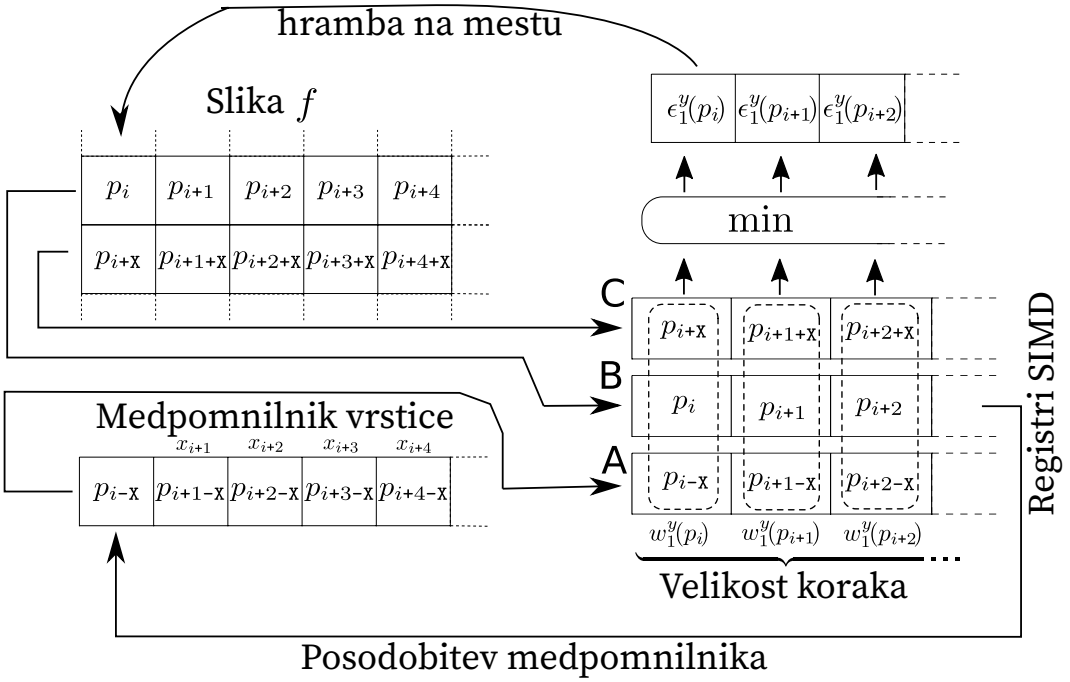
se nadaljnji piksli ne spremenijo. Posledično tako za izračun še vedno uporabimo zgolj tri registre. Psevdokoda predstavljenega postopka je podana v algoritmu 1, kjer funkcija $Naloži_K(f, idx)$ s slike f naloži K pikslov iz zamikov $[idx, idx + K - 1]$. Nasprotno pa $Shrani_K(f, idx)$ iz registra shrani K pikslov na podano pomnilniško lokacijo. Vseeno pa je za vzpostavitev pretokovnega procesiranja izračun $\epsilon_1^y(f)$ potrebno izvesti v istem zaporedju kot $\epsilon_1^x(f)$. Zato potrebujemo dodatni medpomnilnik c, ki je velik X pikslov in predstavlja eno vrstico slike, $c : \mathbb{Z} \to \mathbb{R}$. Medpomnilnik potrebujemo za začasno ohranitev vrednosti trenutne vrstice, saj izračun opravimo na mestu, medtem ko izvorne vrednosti potrebujemo za izračun naslednje vrstice. Zato v tem primeru register A prebere vrednosti prepisane vrstice iz medpomnilnika c na intervalu $[x_i, x_{i+3}]$, medtem ko lahko registra B in C prebereta vrednosti iz slike na intervalih $[p_i, p_{i+3}]$ in $[p_{i+\mathtt{X}}, p_{i+\mathtt{X}+3}]$. Izračun minimalne vrednosti registrov nam poda operator $\epsilon_1^y(f)$, ki ga zapišemo na mestu v interval $[p_i, p_{i+3}]$. Za izračun naslednje vrstice pa vrednosti registra B shranimo v medpomnilnik, kot prikazuje slika 4.4 in psevdokoda algoritma 2. Postopek, po katerem združimo $\epsilon_1^x(f)$

Data: slika f, vrstica y_i, širina slike X, velikost koraka K

function `krčenje_vrstice_na_mestu`(f, y, X, K)**:**

```
    // Inicializacija registra A
    j ← X * y
    A ← Naloži_K (f, j − 1)
    for idx ← j to j + X − 1 s korakom K do
        // Izračun ε_1^x
        B ← Naloži_K (f, idx)
        C ← Naloži_K (f, idx+1)
        B ← min A, B, C
        // Priprava na naslednjo iteracijo
        A ← Naloži_K (f, idx + L − 1)
        // Hramba rezultata na mestu
        Shrani_K(f, idx) ← B
    end
```

Algoritem 1: Izračun na mestu filtra $\epsilon_1^x(f)$ za vrstico y.



Slika 4.4: Izračun na mestu filtra $\epsilon_1^y(f)$, kjer uporabimo medpomnilnik vrstic. Vrednosti nazadnje prepisane vrstice si začasno shranimo, saj jih potrebujemo za izračun vrednosti naslednje vrstice.

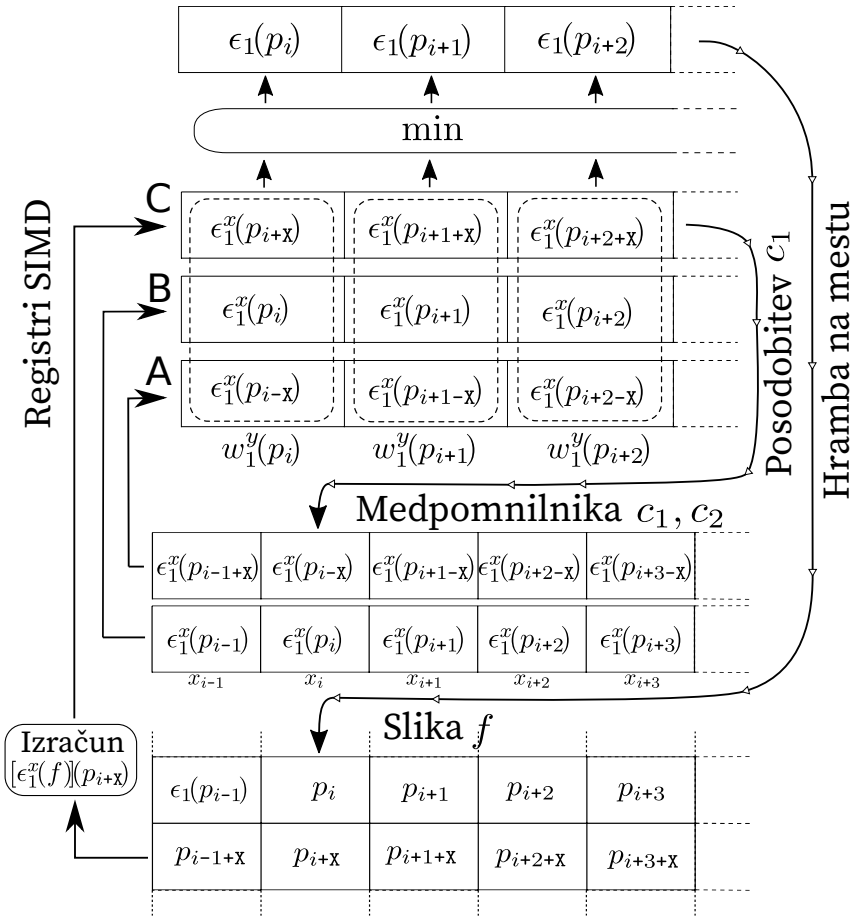
in $\epsilon_1^y(f)$ in tako ovrednotimo operator $\epsilon_1(f)$ v enem linearnem prehodu slike, podrobneje predstavimo v naslednjem podpoglavju.

```
   Data: slika f, širina slike X, višina slike Y, velikost koraka K
 1 function krčenje_stolpcev_na_mestu(f, X, Y, K):
       // Inicializacija medpomnilnika
 2     c ← {max ℝ, ..., max ℝ}
 3     for idx_v ← 0 to Y − 1 do
 4         for idx_s ← 0 to X − 1 s korakom K do
 5             idx ← idx_v * X + idx_s
               // Izračun ε_1^y na mestu
 6             A ← Naloži_K(c, idx_s)
 7             B ← Naloži_K(f, idx)
 8             C ← Naloži_K(f, idx + X)
 9             A ← min A, B, C
10             Shrani_K(f, idx) ← A
               // Posodobi medpomnilnik
11             Shrani_K(c, idx_s) ← B
12         end
13     end
```

Algoritem 2: Izračun na mestu filtra $\epsilon_1^y(f)$ z linearnim dostopom do pikslov skozi celotno sliko f.

4.3 Pretočno procesiranje

V predlaganem postopku izvedemo pretočno procesiranje $\epsilon_1(f)$ z uvedbo dveh medpomnilnikov za vrstici $c_1, c_2 : \mathbb{Z} \to \mathbb{R}$, pri čemer sta oba velika X pikslov. Medpomnilnika uporabljamo za začasno pomnjenje izračuna $\epsilon_1^x(f)$ prejšnjih dveh vrstic, ki ju prepišemo po izračunu $\epsilon_1(f)$. Registra A in B tako vsebujeta vrednosti na intervalu $[x_i, x_{i+3}]$ iz medpomnilnikov c_1 in c_2. Manjkajoči $\epsilon_1^x(f)$ je ovrednoten iz vrednosti slike na intervalu $[p_{i+\mathtt{X}}, p_{i+\mathtt{X}+3}]$, katerega vrednosti hranimo v registru C. Minimalna vrednost registrov A, B in C nam torej ovrednoti $\epsilon_1(f)$ na intervalu $[p_i, p_{i+3}]$. Ocenjene vrednosti zapišemo na mestu. Za nadaljnje izračune register C zapišemo v medpomnilnik c_1 na intervalu



Slika 4.5: Izračun na mestu filtra $[\epsilon_1(f)](p_i)$, kjer za pretočno procesiranje potrebujemo dva medpomnilnika, ki hranita izračune filtra $\epsilon_1^x(f)$ za prejšnji dve vrstici.

$[x_i, x_{i+3}]$. Po končani obdelavi posamezne vrste slike se vsebini medpomnilnika c_1 in c_2 zamenjata. Celoten postopek prikazuje slika 4.5. Za elementarno geodetsko krčenje $\epsilon_1^m(f)$ vrednost minimalnega izračuna omejimo z ustreznimi vrednostmi iz slike maske m, kot je podano v algoritmu 3. V naslednjem podpoglavju predstavimo uporabo pretočnega filtra v verigi filtrov ter vzporeden izračun verige na več jedrih.

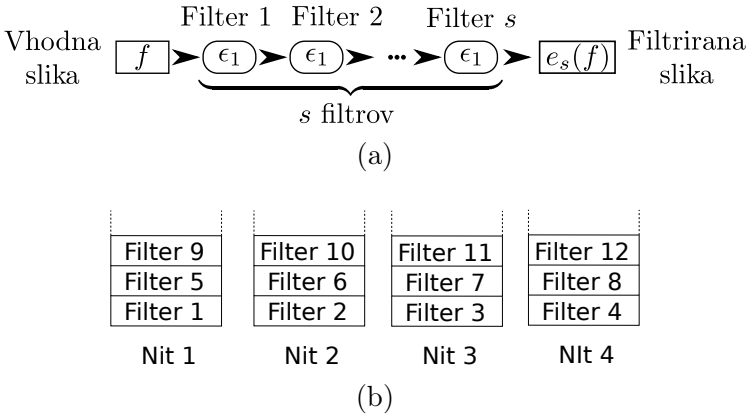
Data: slika f, slika maske m, širina sike X, višina slike Y, velikost koraka K

```
1  function geodetsko_krčenje(f, m, X, Y, K):
       // Inicializacija medpomnilnikov
2      c_1 ← {max ℝ, ..., max ℝ}
3      c_2 ← {[ε_1^x(f)](p_0), ..., [ε_1^x(f)](p_{X-1})}
4      for idx_v ← 0 to Y − 1 do
5          for idx_s ← 0 to X − 1 s korakom K do
6              idx ← idx_v * X + idx_s
               // Izračun ε_1
7              A ← Naloži_K(c_1, idx_s)
8              B ← Naloži_K(c_2, idx_s)
9              C ← Naloži_K(ε_1^x(f), idx + X)
10             A ← min A, B, C
               // Omeji ε_1 z m
11             B ← Naloži_K(m, idx)
12             Shrani_K(f, idx) ← max A, B
               // Posodobi medpomnilnik c_1
13             Shrani_K(c_1, idx_s) ← C
14         end
15         Zamenjaj(c_1, c_2)
16     end
```

Algoritem 3: Izračun na mestu filtra $\epsilon_1^m(f)$ z linearnim dostopom do pikslov slike f.

4.4 Večjedrni izračun morfoloških filtrov

V predlagani metodi za izvedbo vzporednega izračuna verige filtrov na večjedrni CPE vzpostavimo procesni cevovod. Slednji je sestavljen iz podrejenih niti, pri čemer vsaka izmed njih obravnava svojo vrsto opravil (angl. task queue). Niti so umeščene v zbirko oziroma bazen (angl. thread pool), kar omogoča njihovo centralno upravljanje. Zbirka niti se v fazi inicializacije algoritma napolni s T podrejenimi nitmi, kjer T predstavlja število jeder na izvajalnem računskem sistemu. V kolikor CPE omogoča SMT, se vsako fizično jedro predstavi kot več logičnih jeder (dve logični jedri na sistemih x86-64). Vsaki niti se dodeli lastna vrsta opravil, pri čimer pa se izvajanje niti začasno ustavi, kadar se njena vrsta izprazni. Tako je med zagonom algoritma vsaka nit odgovorna samo za svoja opravila oziroma za izračun njej dodeljenih filtrov. Kadar ustvarimo zaporedje filtrov oziroma njihovo verigo, jih zaporedno umestimo v vrsto opravil posameznih niti. Zaporedno umestitev opravimo s sprehodom skozi verigo, pri čemer zaporedni filter j iz verige filtrov umestimo v vrsto opravil, ki pripada niti $t = 1 + (j - 1) \bmod T$. Primer na sliki 4.6 prikazuje umestitev verige filtrov v zbirko $T = 4$ niti. Ker se hkrati opravlja izračun



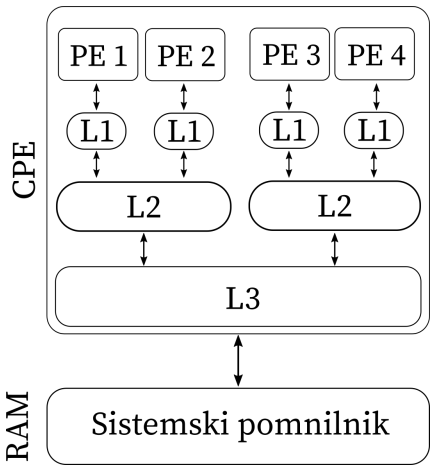
Slika 4.6: Primer (a) verige filtrov, sestavljene iz s elementarnih filtrov, ki jo (b) uvrstimo v štiri ($T = 4$) vrste opravil.

T filtrov, je potrebno zagotoviti, da so niti med seboj primerno sinhronizirane.

V ta namen si vsak filter j vodi števec obdelanih vrstic z atomarnim števcem r_j. Slednjega povečamo po končanem procesiranju vrstice slike, kar pomeni, da ena vrstica slike predstavlja najmanjšo enoto sinhronizacije med dvema filtroma. Zaporednost izračuna filtrov in preprečitev mednitnega tekmovanja za podatki (angl. data race condition) dosežemo tako, da filtru j omejimo dostop do vrednosti pikslov iz zamika $idx = idx_v * X + idx_s$, šele ko filter $j - 1$ to vrstico v celoti obdela, torej $idx_v < r_{j-1}$. Ta omejitev sinhronizira celotno verigo filtrov, njena realizacija pa je odvisna od implementacije. V našem primeru smo uporabili ključavnico v zanki (angl. spinlock) z atomarnimi števci (glej vrstice 6 do 8 v algoritmu 4), saj pričakujemo, da bo vrstica idx_v hitro na voljo in posledično ne želimo prekiniti izvajanja niti. Slednje pa bi bilo potrebno izvesti, v kolikor bi uporabili kakšen drug sinhronizacijski mehanizem (na primer pogojne spremenljivke), pri katerem jedru operacijskega sistema oddamo izvajalni čas, da lahko izvede niti drugih procesov. Čeprav sinhroniziramo samo števec r_{j-1}, nam semantika pridobi – objavi (angl. acquire-release semantics) zagotavlja, da so sinhronizirane tudi preostale spremembe, ki so se zgodile pred spremembo oziroma povečanjem števca [85]. S poznavanjem topologije CPE lahko predstavljeno sinhronizacijo tudi bistveno pohitrimo.

4.5 Analiza topologije CPE za optimizacijo odzivnosti in prepustnosti

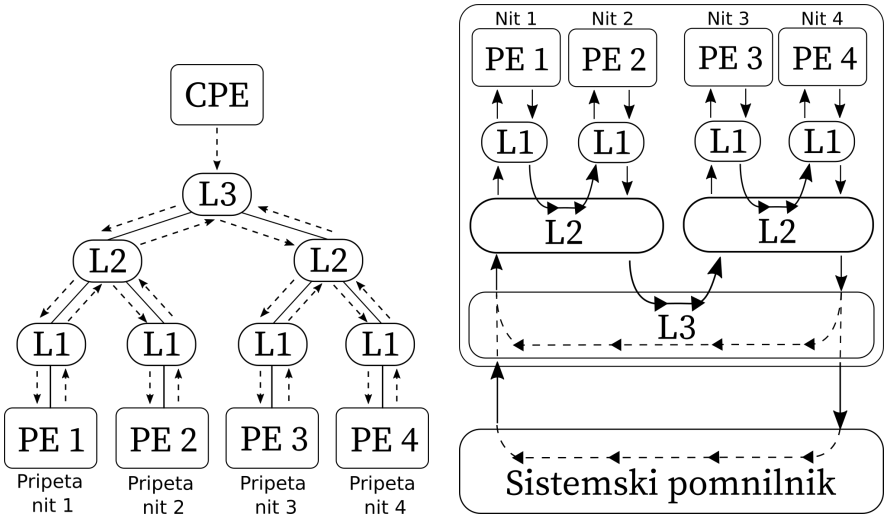
Ker prenos podatkov med različnimi procesnimi enotami poteka na nižjih nivojih predpomnilnika, ki si ga delijo jedra, je poznavanje topologije izvajalnega CPE pomemben dejavnik za pohitritev sinhronizacije. Primer na sliki 4.7 prikazuje sodobno arhitekturo CPE, ki uporablja tronivojski predpomnilnik. Vsako fizično jedro ima izključno last nad predpomnilnikom L1, medtem ko sta nivoja L2 in L3 deljena med več jedri. Velikost ter število nivojev predpomnilnika je seveda odvisno od konkretne arhitekture, vseeno pa v splošnem velja, da so višji nivoji manjši in hitrejši kot nižji nivoji. Poznavanje topolo-



Slika 4.7: Primer štirijedrnega CPE in povezave procesnih enot s predpomnilnikom ter sistemskim pomnilnikom.

gije predpomnilnika lahko izkoristimo pri procesiranju verige filtrov. Rezultat filtra $j-1$ namreč ni hranjen neposredno v sistemski pomnilnik, ampak se najprej začasno in posredno hrani v predpomnilnik L1. Ker ta rezultat predstavlja neposreden vhod v filter j, ga je smiselno ohraniti kar se da visoko v predpomnilniški hierarhiji in posledično zagotoviti njegov hiter prenos. V ta namen niti pripnemo na procesne enote tako, da so si vedno sosednje glede na povezavo med medpomnilniki. Optimalno pripenjanje niti opravimo v fazi inicializacije algoritma in je ne spreminjamo. Pripenjanje izvedemo z iskanjem v globino, kjer preiščemo drevesno strukturo sistemske topologije, ki jo pridobimo s knjižnico *hwloc* [86]. Ko med preiskovanjem topologije naletimo na novo procesno enoto (angl. processing unit), pripnemo naslednjo nit. Tako optimalno pripnemo vseh T ustvarjenih niti, s čimer zagotovimo, da bodo podatkovni prenosi vedno izvedeni v najvišjem možnem nivoju predpomnilnika med dvema sosednjima enotama ter tako v celoti zaobidemo problem ozkega grla sistemskega pomnilnika. Primer pripenjanja in toka podatkov je prikazan na sliki 4.8, kjer prenos podatkov med pripetima nitma na PE1 in PE2 (PE3 in PE4) poteka preko nivoja L2, medtem ko prenos med PE2 in PE3 poteka preko nivoja L3. V kolikor je slednji dovolj velik, da lahko hrani celotno sliko, tudi prenos podatkov med PE4 in PE1 poteka znotraj njega. Zgolj kadar to

ni mogoče, pa se za prenos podatkov uporabi tudi sistemski pomnilnik.



Slika 4.8: (a) Iskanje v globino nad dobljeno topologijo sistema ter zaporedno pripenjanje niti na procesne enote, kjer zagotovimo (b) optimalen pretok podatkov preko predpomnilnika v času izračuna.

Predstavljeni postopek omogoča optimalno izvedbo tradicionalnih morfoloških filtrov. Za izvedbo filtrov, ki uporabljajo postopek rekonstrukcije, pa potrebujemo dodatne mehanizme, ki verigo filtrov podaljšujejo v samem času izvajanja. Slednje predstavimo v nadaljevanju.

4.6 Razširitev na rekonstrukcijske filtre

V tem poglavju se osredotočamo na izvedbo verig morfoloških filtrov, katerih dolžina ni poznana vnaprej. Takšen primer je postopek morfološke rekonstrukcije (glej enačbi 2.22 in 2.23), ki poteka iterativno do konvergence rezultata. Potrebna dolžina je odvisna tako od velikosti slik kakor tudi od same vsebine slik f in m. Zato je potrebno dolžino verige določiti med samim izračunom. V

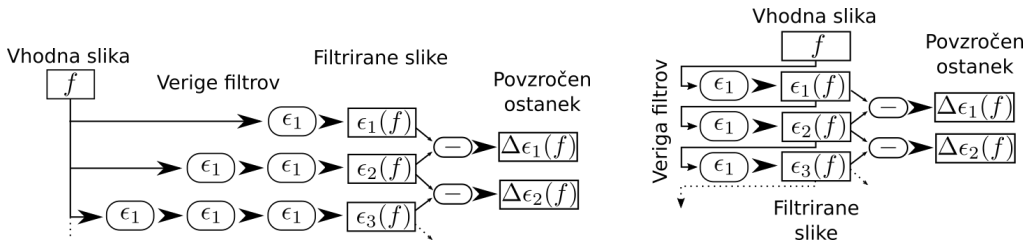
ta namen razširimo filtriranje z elementarnim geodetskim krčenjem ϵ_1^m, tako da med filtriranjem beležimo spremembe v intenzitetah pikslov slike f (glej vrstice 19 do 21 v algoritmu 4). Ob vsaki zaznani spremembi vrednosti vsaj enega piksla filter ϵ_1^m ponovno dodamo v vrsto opravil izvajane niti (glej vrstice 27 do 29 v algoritmu 4), v nasprotnem primeru pa izvajanje prekinemo. Za postopek rekonstrukcije tako potrebujemo samo T filtrov oziroma enega za vsako vrsto opravil niti. To povzroči $T - 1$ odvečnih filtriranj, saj mora vsaka izmed niti zaznati konvergenco filtriranja. Odvečna filtriranja predstavljajo samo majhen del izvajalnega časa, saj lahko pričakujemo bistveno večjo velikost tvorjenje verige od $T - 1$. Pri tem predstavljeni pristop omogoča optimalno izvedbo verig filtrov, sestavljenih iz izključno zaporednih filtrov, pri čemer lahko pomembne optimizacije izvedemo tudi na nivoju njihovih verig. Slednje predstavimo v nadaljevanju.

4.7 Razgradnja morfoloških filtrov na verigo elementarnih filtrov

Neposredna preslikava formalnih definicij morfoloških operatorjev v verigo filtrov ni vedno optimalna, saj nekatere izmed njih vsebujejo odvečne izračune in podvojene izvedbe verig elementarnih filtrov. Primer takšnega izračuna je transformacija domnevnih razdalj, kjer za vsak piksel izračunamo razdaljo do najbolj kontrastnega piksla okolice. Slednje izvedemo z uporabo cilindrične transformacije in iterativnega krčenja (glej enačbo 2.35). Kot prikazuje primer na sliki 4.9a, formalna definicija transformacije zahteva izvedbo morfološkega krčenja nad vhodno sliko v vsaki iteraciji, kar vodi v presežek potrebnih izračunov. Nasprotno pa lahko verige preuredimo tako, da rezultat prejšnje verige uporabimo za nadaljnje filtriranje pri naslednji verigi. Kot prikazuje primer na sliki 4.9b, tako dobimo eno samo verigo filtrov. Zato da ohranimo obdelavo na mestu in se izognemo nepotrebnim vmesnim slikam, filter ϵ_1 razširimo s cilindrično transformacijo. V tem primeru je to bistvenega pomena, saj za

```
Data: slika f, slika maske m, širina sike X, višina slike Y, velikost
      koraka K, zaporedni indeks filtra j, število niti T
1  function pretočno_geo_krčenje(f, m, X, Y, L, j):
2      c_1 ← {max ℝ, ..., max ℝ}
3      c_2 ← {[ε_1^x(f)](p_0), ..., [ε_1^x(f)](p_X-1)}
4      sprememba ← false
5      for idx_v ← 0 to Y − 1 do
6          while idx_v ≥ r_j-1 do
7              Počakaj
8          end
9          for idx_s ← 0 to X − 1 s korakom L do
10             idx ← idx_v * X + idx_s
               // Izračun ε_1
11             A ← Naloži_K(c_1, idx_s)
12             B ← Naloži_K(c_2, idx_s)
13             C ← Naloži_K(ε_1^x(f), idx + X)
14             A ← min A, B, C
               // Posodobi medpomnilnik c_1
15             Shrani_K(c_1, idx_s) ← C
               // Omeji ε_1 z m
16             B ← Naloži_K(m, idx)
17             A ← max A, B
               // Primerjaj s starimi vrednostmi slike f
18             B ← Naloži_K(f, idx)
19             if A ≠ B then
                   // Shrani spremenjene piksle
20                 Shrani_K(f, idx) ← A
                   // Potrebno dodatno filtriranje
21                 sprememba ← true
22             end
23         end
           // Povečaj števec vrstic in zamenjaj medpomnilnika
24         r_j ← r_j + 1
25         Zamenjaj(c_1, c_2)
26     end
       // Ob spremembi slike f dodaj e_m^1 v vrsto opravil
27     if sprememba then
28         Vstavi(pretočno_geo_krčenje(f, m, X, Y, L, j+T))
29     end
```

Algoritem 4: Izračun filtra ϵ_1^m z zaznavo konvergence, ki jo uporabimo za postopek rekonstrukcije.



(a) (b)

Slika 4.9: (a) Neposreden izračun povečujočega se krčenja, kjer za vsak filter uporabimo novo verigo filtrov, ter (b) ponovna uporaba prejšnjega filtriranja, s čimer zmanjšamo število potrebnih izračunov ter ohranimo procesiranje na mestu.

vsak piksel iščemo velikost okna, ki povzroči največjo spremembo intenzitete. Velikost okna hkrati definira razdaljo do njemu najbolj kontrastnega piksla, medtem ko najdene vrednosti hranimo v dodatni sliki $d(f)$. Posledično moramo vzdrževati tudi vrednosti največjih kontrastov, ki jih hranimo v sliki največjih odzivov $r(f)$. Postopek izvedbe zahteva največ $max\,(X, Y)$ dolgo verigo filtrov, saj je potrebno izvesti krčenje vseh velikosti oken, ki jih lahko vsebuje slika. Ker v izračunu vsakega filtra spreminjamo vrednosti treh slik, to znatno poveča zahtevano pasovno širino sinhronizacije. Slednjo ublažimo tako, da sliki $d(f)$ in $r(f)$ spremenimo samo, kadar je to nujno potrebno, in posledično zmanjšamo število pikslov, ki jih je potrebno sinhronizirati. Spremembe $d(f)$ so neposredno odvisne od sprememb $r(f)$, za katere pa pričakujemo, da so redke. Zato pred vsakim zapisom vrednosti v slikah $d(f)$ in $r(f)$ najprej testiramo, ali je sploh prišlo do spremembe vrednosti (glej vrstice 15 do 26 v algoritmu 5). V kolikor ni sprememb, slik ne posodobimo in tako zmanjšamo potrebno pasovno širino. Slednje pa odtehta tudi uporabo vejitev, ki sicer zmanjšajo efektivnost procesiranja (glej rezultate v tabelah 5.6, 5.9 in 5.13). Preostanek izračuna $d_{L1}(f)$ zahteva postopek rekonstrukcije, ki ga realiziramo s primernimi spremembami filtra ϵ_1^m (glej enačbi 2.36 in 2.37). Obsežna testiranja učinkovitosti predstavljene metodologije podrobneje predstavimo v naslednjem poglavju.

```
Data: slika f, slika največje razlike r, slika razdalj d, višina slike Y,
      velikost koraka K, zaporedni indeks filtra j
1  function krčenje_TDR(f, r, d, X, Y, L, j):
2      c_1 ← {max ℝ, ..., max ℝ}
3      c_2 ← {[ε_1^x(f)](p_0), ..., [ε_1^x(f)](p_{X-1})}
4      for idx_v ← 0 to Y − 1 do
5          while idx_v ≥ r_{j−1} do
6              Počakaj
7          end
8          for idx_s ← 0 to X − 1 s korakom L do
9              idx ← idx_v * X + idx_s
               // Izračun ε_1
10             A ← Naloži_K(c_1, idx_s)
11             B ← Naloži_K(c_2, idx_s)
12             C ← Naloži_K(ε_1^x(f), idx + X)
13             A ← min A, B, C
               // Posodobi medpomnilnik
14             Shrani_K(c_1, idx_s) ← C
               // Naloži prejšnji rezultat ε_1
15             B ← Naloži_K(f, idx)
16             if A ≠ B then
                   // Posodobi sliko f z rezultatom ε_1
17                 Shrani_K(f, idx) ← A
                   // Izračun novega ostanka
18                 B ← B − A
                   // Posodobi r(f) in d(f) samo ob višjem ostanku
19                 A ← Naloži_K(r(f), idx)
20                 maska ← B > A
21                 if maska ≠ 0 then
22                     PogojnoShrani_L(r(f), idx, maska) ← B
23                     B ← {j, ..., j}
24                     PogojnoShrani_L(d(f), idx, maska) ← B
25                 end
26             end
27         end
28         r_j ← r_j + 1
29         Zamenjaj(c_1, c_2)
30     end
```

Algoritem 5: Razširitev filtra ϵ_1, ki z uporabo slike $r(f)$ izračuna nezvezno funkcijo $d(f)$.

Poglavje 5

Rezultati in analiza

Testiranje smo opravili na treh različnih testnih sistemih z arhitekturo x86-64, ki so se bistveno razlikovali glede na topologijo predpomnilnika ter povezave med procesnimi enotami, pri čemer pa so vsi podpirali ukazni nabor AVX2 in implementirali fizične registre z 256 biti. Prvi testni sistem T1 je bil sestavljen iz štirih jeder s tronivojskim predpomnilnikom in ni podpiral večnitnega izvajanja na nivoju posameznega jedra. Rezultati prvega sistema so tako predstavljali referenčne vrednosti, ki smo jih uporabili za oceno pohitritev, doseženih na bistveno zmogljivejših enotah glede na podane specifikacije. Drugi testni sistem T2 je uporabljal heterogen CPE, ki je bil sestavljen iz štirih manjših procesnih podenot. Te so vsebovale tri fizična oziroma šest logičnih jeder in 16 MiB deljenega predpomnilnika L3. Testi na tem sistemu so tako podali po-

Tabela 5.1: Uporabljeni testni sistemi

Testni sistem	CPE			Število CPE	Velikost predpomnilnika CPE		
	Proizvajalec	Model	Frekvenca		L1D	L2	L3
T1	Intel	Core i5-6600k	3,4 GHz	1	128 KiB	1 MiB	6 MiB
T2	AMD	Ryzen 3900X	3,8 GHz	1	384 KiB	6 MiB	64 MiB
T3	Intel	Xeon E5-2683 v4	2,10 GHz	2	512 KiB	4 MiB	40 MiB

Tabela 5.2: Podrobnosti testnih CPE ter delitev predpomnilnika glede na število logičnih procesnih enot.

Testni sistem	Število PE	SMT	Delitev predpomnilnika					
			L1D	#PE	L2	#PE	L3	#PE
T1	4	Ena nit	32 KiB	1	256 KiB	1	6 MiB	4
T2	6+6+6+6	Dve niti	32 KiB	2	512 KiB	2	16 MiB	6
T3	32	Dve niti	32 KiB	2	256 KiB	2	40 MiB	32

Tabela 5.3: Podrobnosti glavnega pomnilnika testnih sistemov.

Testni sistem	Število pomnilniških modulov	Tip in velikost	Prenos podatkov na modul	Število uporabljenih pomnilniških kanalov	Skupni prenos na CPE
T1	2	16GB DDR4	19200 MB/s	2	38 GB/s
T2	4	32GB DDR4	21333 MB/s	2	43 GB/s
T3	2	32GB DDR4	19200 MB/s	2	38 GB/s

membne informacije o učinkovitosti skaliranja na večje število jeder, medtem ko je dostop do glavnega pomnilnika ostal enoten oziroma enakomeren. Nasprotno pa je bil tretji sistem T3 namenjen testiranju neenakomernega dostopa do glavnega pomnilnika (NUMA, angl. Non-Uniform Memory Access). Sistem je bil namreč sestavljen iz dveh ločenih CPE, pri čemer je vsaka vsebovala 16 fizičnih oziroma 32 logičnih jeder. Sinhronizacija med dvema CPE je potekala neposredno, preko medsebojne povezave QPI (angl. QuickPath Interconnect). Ta testni sistem je tako služil za analizo skaliranja, ko ima računski sistem več medsebojno povezanih CPE. Podroben pregled vsakega testnega sistema, topologije predpomnilnika in glavnega pomnilnika za posamezne CPE je podan v tabelah 5.1, 5.2 in 5.3. Podporne tehnike za zviševanje računske zmogljivosti, na primer kratkoročno povečanje frekvence in SMT (angl. simultaneous multithreading), smo omogočili na vseh podprtih testnih sistemih.

Implementacijo vseh filtrov smo izvedli v programskem jeziku C++ (revizija C++17) z uporabo funkcijske predloge (angl. function template) in knjižnice std::experimental::simd [87, 88]. Slednja nam je omogočila splošno

implementacijo posameznih filtrov, ne glede na uporabljen podatkovni tip ali velikost registra. Ker je bila velikost koraka in pomnilniškega prostora za medpomnilnike filtrov ter celotne slike odvisna od uporabljenega podatkovnega tipa, smo testiranje opravili nad podatkovni tipi `unsigned char` (1 zlog), `unsigned short` (2 zloga), `float` (4 zlogi) in `double` (8 zlogi). Podatkovnih tipov `unsigned int` (4 zlogi) in `unsigned long long` (8 zlogi) pa nismo testirali, saj so računski časi enaki tem, ki smo jih dosegli z uporabo `float` in `double`. Računski časi merijo samo čas filtriranja oziroma izvajalni čas metode; zato tudi ne zajemajo inicializacije programa ali predpriprave podatkov. Pri vektorskih registrih smo se omejili na 256-bitne registre in ukazni nabor AVX2, saj nas zanima le najvišja možna učinkovitost na testiranih sistemih. Učinkovitost metode na sorodnih sistemih lahko pri tem določimo analitično, saj bi uporaba 128-bitnih registrov podvojila število potrebnih ukazov in tako izvajalne čase kvečjemu podvojila, medtem ko bi uporaba 512-bitnih registrov prepolovila število izvedenih ukazov in posledično kvečjemu dosegla dvakratno pohitritev.

Število zahtevanih izračunov in izvajalni časi filtrov, ki temeljijo na postopkih rekonstrukcije, so odvisni od same vsebine slike kakor tudi od morebitnih vhodnih parametrov filtra. Referenčne meritve smo izvedli nad verigami elementarnih filtrov ϵ_1 s konstantno dolžino. V kolikor ni podano drugače, smo pri testiranjih uporabili sliko *Sand* iz referenčne baze SIPI [48], pri čemer so bili vsi testi opravljeni nad slikami velikosti 1024×1024 pikslov. Posamezne vrstice slike so bile pri tem vedno pomnilniško poravnane, kot je to opisano v poglavju 4.1. Pri izvedbi testov, kjer katera od omejitev ni držala, pa specifike izvedbe podrobneje opredelimo med opisom testa samega.

Za vsa prevajanja kode smo uporabili prevajalnik Clang (različica 10.0) z nivojem optimizacije `-O3`, ukaznim naborom `-mavx2` in prilagoditvijo ustvarjenega zbirnika `-mtune=haswell`. Z uporabo diagnostičnega orodja ThreadSanitizer `-fsanitize=thread` pa smo v ločenem zagonu potrdili, da v predstavljeni metodi ni prišlo do mednitnega tekmovanja za podatki.

Za potrditve ali ovržbe zastavljenih hipotez smo testiranja osredotočili na naslednje vidike:

- Ovrednotenje računskih časov predstavljene metode z uporabo tradicionalnega filtra ϵ_1 v več možnih sestavah, kot na primer uporaba različnega števila niti, dolžine verige filtrov ter različno velikih podatkovnih tipov.
- Ovrednotenje vpliva širine in višine slike na računske čase. Pri tem smo izhajali iz dejstva, da širina slike predstavlja najmanjšo enoto sinhronizacije in direktno vpliva na velikost medpomnilnika v filtrih, medtem ko višina slike določi maksimalno število potrebnih mednitnih sinhronizacij med dvema zaporednima filtroma. Najmanjša testirana dimenzija je 128 pikslov, saj metoda potrebuje vsaj 128 pikslov v vrstici kadar uporabimo podatkovni tip `char`, medtem ko za testni sistem 3 potrebujemo vsaj 128 vrstic, da v celoti zasedemo vseh 64 niti.
- Ovrednotenje vpliva medjedrne komunikacije na izračun verige filtrov, kjer lahko dosežemo boljše računske čase, v kolikor opravimo vzporeden izračun več verig filtrov, pri čemer za vsako verigo omejimo število uporabljenih procesnih enot.
- Ovrednotenje računskih časov izpeljanih operatorjev z različnimi podatkovni tipi in sorodnimi metodami.
- Ovrednotenje računskih časov verige elementarnih filtrov in izpeljanih operatorjev na GPGPU.

V naslednjem poglavju najprej predstavimo učinkovitost predstavljene metode, ko se omejimo na eno jedro.

5.1 Enojedrni testi

Predlagano metodo smo najprej ovrednotili v vseh testnih pogojih z uporabo ene niti. Tako smo iz testov izločili vse preostale neodvisne spremenljivke, kot sta to na primer pasovna širina glavnega pomnilnika in vpliv mednitne sinhronizacije. Pridobljene meritve izvajalnih časov različno dolgih verig elementarnih filtrov ob uporabi različnih podatkovnih tipov na opisanih testnih sistemih prikazuje tabela 5.4. Čeprav so razlike med tesnimi sistemi pričakovane tudi ob uporabi enakih podatkovnih tipov, meritve ni mogoče pojasniti zgolj na osnovi razlik v frekvenci CPE. Glavni razlog za to se skriva v različnih stopnjah paralelizma na nivoju ukaznega nabora, ki so odvisne od same arhitekture CPE. Tako se je na primer sistem T2 izkazal za skoraj 1,5-krat hitrejšega od sistema T1, čeprav je bila razlika v frekvenci največ 200 MHz oziroma zgolj 5 %. Da se izognemo tovrstnim vplivom smo znotraj vsakega testnega sistema z empiričnim pristopom raje primerjali upočasnitev, ki jo je povzročila uporaba večjega podatkovnega tipa. Pričakovane upočasnitve in doseženi faktorji so podani v tabeli 5.5. Faktorji upočasnitve so se skoraj popolnoma ujemali s pričakovanimi upočasnitvami v vseh primerih, razen v primeru sistema T1. V tem primeru je namreč opazno večje odstopanje ob uporabi podatkovnega tipa `double`, ki ga je povzročila prevelika velikost vhodne slike. Slednja je bila namreč prevelika (8 MiB), da bi jo bilo mogoče v celoti hraniti v predpomnilniku L3 s kapaciteto 6 MiB. Posledično je vsak filter ponovno prebral in zapisal celotno sliko v glavni pomnilnik, kar je drastično poslabšalo hitrost izvedbe glede na razpoložljivo pasovno širino pomnilnika (glej tabelo 5.3). Glede na podane podatke, odsotnost ozkih grl in primerjavo s sistemom 1 smo v nadaljnjih testih za sistem 2 pričakovali faktor pohitritve $\approx 1{,}4$ in faktor upočasnitve $\approx 1{,}6$ za sistem 3.

Za oceno vpliva dimenzije slike na izvajalne čase smo v naslednjem testu omejili dolžino verige na 512 filtrov ϵ_1 in pri tem izvedli teste nad tremi slikami, ki so bile različnih dimenzij. Kot referenčno sliko smo uporabili kvadratno sliko

Tabela 5.4: Enonitni računski časi verig filtrov fiksnih dolžin na vseh testnih sistemih in podatkovni tipi. Vsi podani časi so v milisekundah.

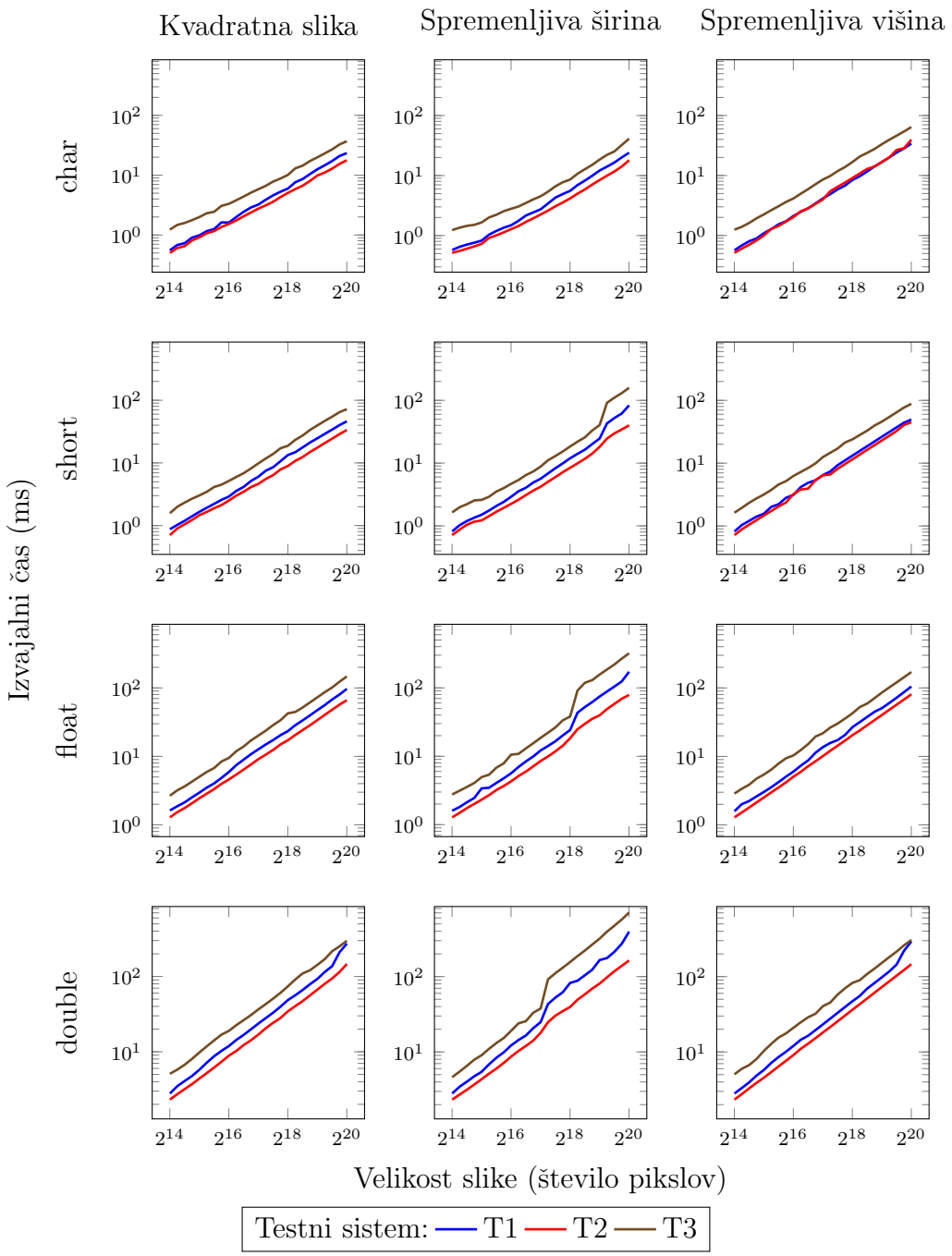
Dolžina verige	char			short			float			double		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
32	1,5	1,1	2,6	2,9	2,1	4,7	6,2	4,2	9,5	18,0	9,1	18,6
64	3,0	2,2	5,3	5,8	4,2	9,1	12,2	8,3	18,9	35,1	18,0	37,7
96	4,5	3,3	7,6	8,7	6,3	13,7	18,3	12,4	28,2	51,9	26,9	56,1
128	6,0	4,4	10,6	11,6	8,4	18,1	24,3	16,5	37,4	68,6	35,5	75,2
160	7,4	5,5	12,5	14,5	10,5	23,0	30,4	20,7	46,8	85,2	44,4	93,9
192	8,9	6,7	15,2	17,4	12,6	27,2	36,4	24,8	55,9	101,7	53,3	111,3
224	10,3	7,8	18,0	20,3	14,7	32,1	42,5	28,9	65,2	118,5	62,1	129,7
256	11,8	8,9	20,5	23,2	16,8	36,4	48,5	33,0	74,7	134,5	71,2	147,2
288	13,3	10,0	22,9	26,1	18,9	41,0	54,5	37,1	83,8	151,6	79,9	165,9
320	14,8	11,2	25,0	29,0	20,9	44,5	60,6	41,3	93,3	168,2	88,8	185,1
352	16,2	12,3	27,4	31,9	23,0	49,6	66,6	45,5	102,7	184,7	97,6	203,0
384	17,6	13,4	30,0	34,8	25,1	54,2	72,6	49,6	111,7	201,5	106,4	221,4
416	19,1	14,5	32,0	37,6	27,2	58,3	78,7	53,7	120,4	217,7	115,3	240,7
448	20,5	15,7	34,7	40,5	29,3	62,7	84,7	57,8	130,9	234,0	124,1	259,8
480	22,0	16,7	37,0	43,4	31,4	68,0	90,7	62,0	141,2	250,7	133,0	278,4
512	23,5	17,8	39,7	46,3	33,5	72,0	96,8	66,1	149,4	266,9	141,8	296,2

Tabela 5.5: Upočasnitve izvajanja znotraj vsakega testnega sistema v primerjavi s podatkovnim tipom `char` in tradicionalnim filtrom ϵ_1.

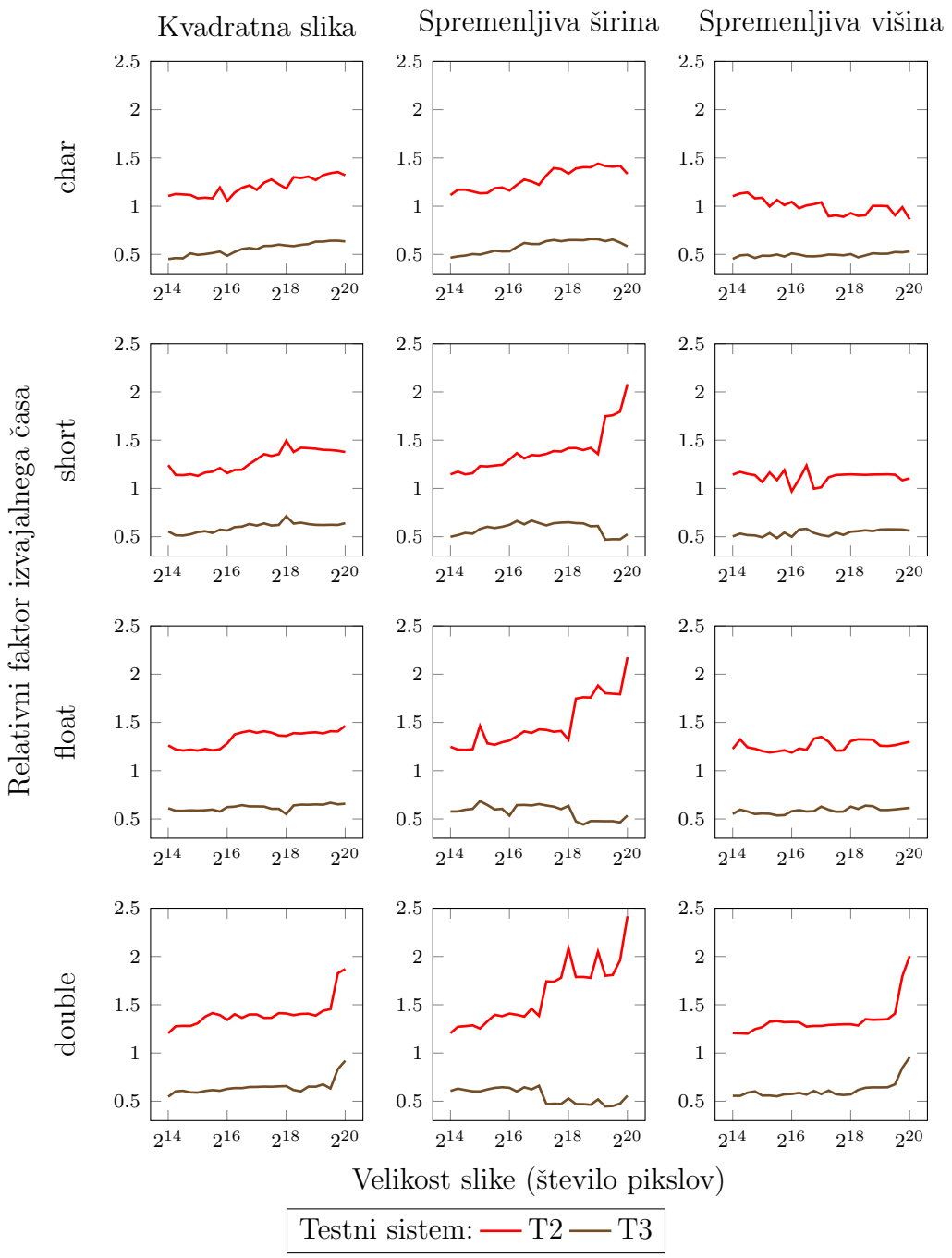
	Podatkovni tip			
	char	short	float	double
Pričakovana upočasnitev	1	0,5	0,25	0,125
Sistem T1	1	0,506	0,242	0,088
Sistem T2	1	0,535	0,271	0,126
Sistem T3	1	0,546	0,263	0,133

dimenzij $X \times Y$, kjer $X = Y$. Višina druge slike je bila konstantna $Y = 128$, širino pa smo povečevali za faktor $\sqrt[4]{2}$, dokler nismo dosegli 1 megapiksla, torej $X = [128, \ldots, 1024^2/128]$. Nasprotno je bila tretja slika široka $X = 128$, pri čemer pa smo povečevali njeno višino $Y = [128, \ldots, 1024^2/128]$. Pridobljene meritve izvajalnih časov in relativne pohitritve glede na sistem 1 so podane na slikah 5.1 in 5.2. V meritvah smo zasledili dve anomaliji, kjer je prišlo do nenadno slabših izvajalnih časov ob povečevanju širine slike v primeru sistemov T1 in T3. To je bila posledica medpomnilnikov c_1 in c_2 iz filtra ϵ_1, saj pri večjih širinah presežeta kapaciteto predpomnilnika L1D. Slednje je namreč povzročilo, da so bile njune vrednosti vedno prebrane iz nivoja L2. Ker so imeli nižji nivoji manj pasovne širine, je to povzročilo ozko grlo pri samem branju vrednosti in do dvakratno poslabšanje izvajalnih časov v primerjavi s kvadratno sliko. Ker je imel sistem T2 hitrejši predpomnilnik L2, je bil vpliv ozkega grla ublažen in posledično je bila dosežena do dvakratna pohitritev v primerjavi s sistemom T1. Nasprotno je sistem T3 imel počasnejši predpomnilnik L2, kar je vodilo v dvakratno upočasnitev.

Ob omejitvi širine slike sistem T2 ni dosegel pričakovanega faktorja pohitritve $\approx 1{,}4$ s podatkovnima tipoma `char` in `short`, ampak je ta bila bližje faktorju pohitritve ≈ 1. V teh dveh primerih so bile posamezne vrstice hitro izračunane, saj je vsaka iteracija zanke obdelala 32 in 16 pikslov. Ker je bila širina slike omejena na $X = 128$, je tako bil omejen paralelizem na nivoju ukaznega nabora. S povečanjem podatkovnega tipa se je primerno povečalo tudi število potrebnih iteracij za izračun posamezne vrstice. Slednje pa je vodilo do pričakovane pohitritve za faktor $\approx 1{,}4$. Kadar je velikost slike presegla 6 MiB, so bile dodatne pohitritve v primerjavi s sistemom T1 pričakovane.



Slika 5.1: Enonitni izvajalni časi pri povečujoči se sliki različnih dimenzij, kjer rezultati kvadratne slike služijo kot kontrolne meritve, slika s spremenljivo širino meri vpliv povečane uporabe predpomnilnika in slika s spremenljivo višino meri vpliv povečanja mednitne sinhronizacije.



Slika 5.2: Relativni izvajalni časi sistemov 2 in 3 pri povečujoči se sliki različnih dimenzij glede na sistem T1.

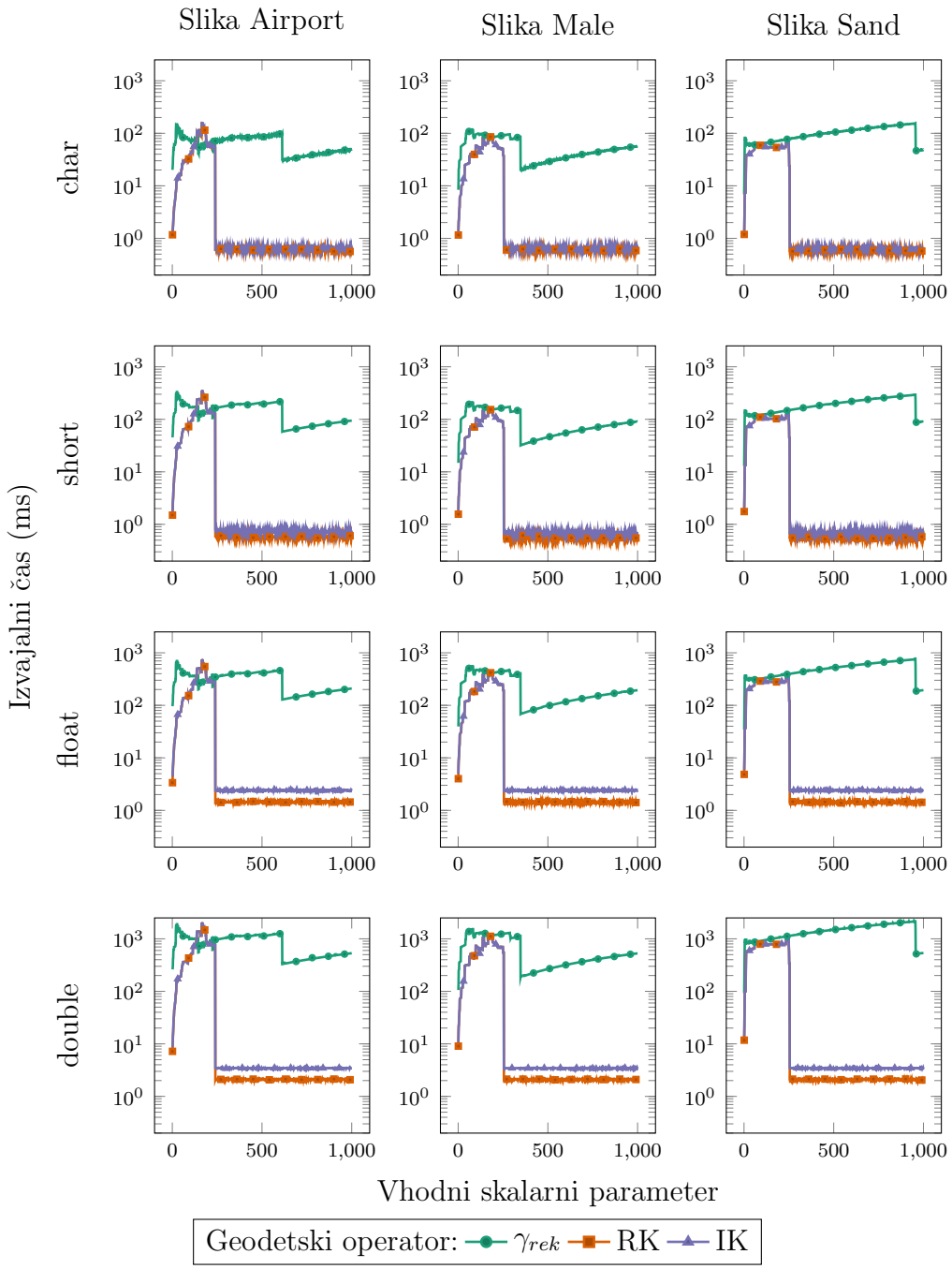
Tabela 5.6: Enonitni izvajalni časi (v milisekundah) izpeljanih filtrov pri različnih vhodnih slikah, testnih sistemih in podatkovnih tipih.

Operator	Slika	Povprečna dolžina verige	char			short			float			double		
			T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
SPL	Airport	1706	109,1	75,4	163,5	198,1	136,1	306,7	548,6	269,6	629,3	1546,1	861,9	1429,9
	Male	852	56,4	39,2	86,1	101,3	72,7	169,1	303,0	138,9	327,4	728,8	433,9	734,9
	Sand	747	53,0	38,3	78,2	93,8	66,7	145,5	260,0	128,4	287,8	710,6	374,5	649,8
OO	Airport	1463	96,6	69,4	145,0	178,7	123,8	271,2	479,2	239,6	549,2	1326,1	747,9	1234,7
	Male	1648	103,8	72,6	164,4	191,7	137,1	302,6	571,2	255,9	612,1	1394,0	799,0	1391,7
	Sand	957	70,2	53,0	104,6	129,4	90,6	189,4	341,5	173,9	382,6	916,9	504,8	845,1
γ_{rek}	Airport	1291	80,9	52,3	118,0	184,7	99,9	223,8	390,7	204,1	475,7	1073,9	599,0	1098,6
	Male	1330	86,0	56,6	126,6	155,6	111,0	241,2	413,3	223,8	489,1	1145,2	664,9	1097,0
	Sand	1813	102,9	71,6	159,5	196,6	137,6	316,3	513,2	284,4	662,6	1462,0	808,6	1495,1
RK	Airport	733	54,0	36,0	75,6	122,1	68,9	145,0	256,4	135,8	296,4	690,8	394,2	699,0
	Male	647	47,3	33,1	72,3	85,0	63,5	135,9	223,6	122,9	260,2	602,6	357,0	573,5
	Sand	794	51,2	35,7	83,2	96,4	69,0	162,8	260,7	138,9	325,2	729,9	413,2	705,4
IK	Airport	734	54,0	36,1	78,0	122,2	68,9	141,3	257,4	136,4	298,5	691,7	395,0	697,2
	Male	648	47,4	33,1	71,8	85,0	63,5	133,2	224,4	122,9	261,4	603,8	357,3	579,6
	Sand	795	51,3	35,8	83,9	96,5	68,9	157,2	261,7	138,5	329,3	731,0	414,0	708,6
TDR	Airport	1070	78,8	49,4	117,5	133,1	87,5	226,2	293,5	177,6	381,8	697,1	397,2	724,0
	Male	1064	73,1	47,4	112,7	125,0	82,4	207,6	260,0	163,5	359,0	809,4	364,2	691,7
	Sand	1054	72,9	49,1	117,3	125,8	83,1	208,1	276,7	160,7	344,3	672,0	387,8	701,6

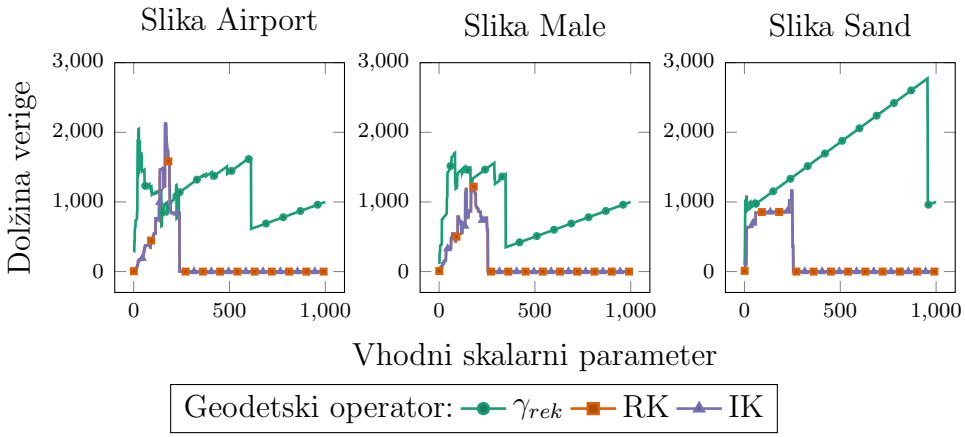
(a) (b) (c)

Slika 5.3: Uporabljene testne slike (a) *Airport*, (b) *Male* in (c) *Sand* za meritve izvajalnih časov geodetskih operatorjev.

Nazadnje smo preverili še izvajalne čase izpeljanih geodetskih operatorjev, kjer smo testirali nad slikami *Airport*, *Male* in *Sand*, vidnimi na sliki 5.3. Rezultati tega testa so podani v tabeli 5.6. Pri filtrih SPL in OO smo kot masko uporabili robne piksle vhodne slike. Filtrom s skalarnim vhodnim parametrom smo izvajalne čase in dolžine verig povprečili na intervalu, ko vhodna slika ni bila enobarvna kot posledica prevelikega parametra. Na primer povprečenje operatorja $\mathrm{IK}_h(f)$ smo omejili na vhodne parametre $h = [1, ..., max(f)]$, saj so preostale vrednosti h popolnoma odstranile vsebino slike. Podrobnejše meritve izvajalnih časov glede na vhodni parameter in uporabljen podatkovni tip vidimo na sliki 5.4. Dolžine tvorjenih verig pri posameznem vhodnem parametru vidimo na sliki 5.5 in so enake za vse podatkovne tipe. Od vseh testiranih filtrov se je odpiranje z rekonstrukcijo izkazalo kot najbolj zahtevno, saj vhodni parameter predstavlja dolžino verige tradicionalnih krčenj ϵ_1, ki mu sledi širjenje z rekonstrukcijo. Ker prvi del verige ne preverja monotonosti, nam to prestavlja najkrajšo možno verigo filtra, kadar slika vsebuje zgolj piksle konstantne intezitete. Takšen primer nastopi, ko sliko *Male* filtriramo z $\epsilon_{s=348}$, kjer dobimo enobarvno sliko, in posledično je tvorjena veriga dolga $s + 1$ filtrov. Iz testov so bili izvzeti izmenično-zaporedni filtri, saj so ti sestavljeni iz tradicionalnih filtrov ϵ_1 in δ_1. Posledično so njihovi izvajalni časi enaki časom tradicionalnih filtrov, predstavljenih v tabeli 5.4.



Slika 5.4: Enonitni izvajalni časi izpeljanih geodetskih operatorjev ob povečujočem vhodnem parametru na sistemu T1.



Slika 5.5: Dolžina tvorjene verige filtrov ob povečujočem se vhodnem parametru.

Iz podanih meritev smo nato izluščili vpliv števila vhodnih in izhodnih slik na izvajalne čase. Filtri, ki temeljijo na postopku rekonstrukcije, so namreč potrebovali dve vhodni sliki, kjer pa je bila slika maske m uporabljena samo za branje. Granolumetrični filtri, kot je to na primer transformacija domnevnih razdalj, potrebujejo celo tri vhodne slike. Ker je treba v obeh primerih hkrati uporabljati več kot eno sliko, so bili izvajalni časi slabši v primerjavi z enako dolgo verigo filtrov ϵ_1, saj je bila kapaciteta predpomnilnika L1D hitreje presežena. Posledično so bili podatki prebrani ali iz nižjega nivoja predpomnilnika ali iz glavnega pomnilnika. Relativne upočasnitve glede na uporabljen podatkovni tip so podane v tabeli 5.7. Pri tem je očitno, da so rekonstrukcijski operatorji bistveno bolj zahtevni od ostalih, čeprav so delovali z eno sliko manj. Prav tako lahko vidimo vpliv večjega predpomnilnika L3 v sistemu T3, kjer sta lahko bili obe sliki hranjeni v predpomnilniku pri vseh podatkovnih tipih. Zaradi manjšega predpomnilnika L3 je sistem T2 lahko hranil obe sliki samo do podatkovnega tipa `float`, testi sistem T1 pa je bil omejen na podatkovna tipa `short` in `char`. Za natančnejšo analizo zahtevnosti posamičnega filtra smo primerjali povprečen čas za izračun obeh enotnih filtrov glede na tradicionalni filter ϵ_1. Povprečni časi in relativne upočasnitve proti filtru ϵ_1 so podani v tabeli 5.8. Po pričakovanjih se je z večanjem podatkovnega tipa rekonstrukcij-

skim filtrom povečeval faktor upočasnitve. Nasprotno in proti pričakovanjem pa so se pri granolumetričnih filtrih faktorji upočasnitve izboljševali z večanjem podatkovnega tipa. To je posledica pogojnega branja in posodabljanja slik $r(f)$ in $d(f)$ (glej vrstice 16–26 v algoritmu 5). V teh primerih se je namreč večina filtriranj opravljala samo nad vhodno sliko f, medtem ko sta se slika razlik $r(f)$ in slika razdalj $d(f)$ redko brali in posodabljali, kar je sprostilo pritisk na predpomnilnik. Nasprotno so rekonstrukcijski filtri vedno brali vrednosti pikslov iz dveh slik (glej vrstice 16–18 v algoritmu 4), kar je povečalo zahtevo po predpomnilniku in so se izvajalni časi poslabšali. To smo potrdili s števci dogodkov branja in pisanja slik med filtriranjem slike *Airplane* z operatorjema SPL in TDR, kjer so vrednosti števcev podane v tabeli 5.9. Operator SPL je uporabil algoritem 4, kjer smo najprej šteli branje slik f in m v vrsticah 13 in 16, medtem ko smo pogojno posodobitev slike f šteli v vrstici 20. Operator TDR je uporabil algoritem 5, kjer smo najprej šteli branje slike f v vrsticah 12 in 15, temu je sledilo pogojno štetje posodobitve slike f in branja slike $r(f)$ v vrsticah 17 in 19 ter nazadnje še štetje vgnezdene pogojne posodobitve slik $r(f)$ in $s(f)$ v vrsticah 22 in 24. Pri obeh operatorjih je bilo posodabljanje vrednosti slik redko, kjer je pogostost posodabljanja padala z večanjem podatkovnega tipa. Kadar je bila posodobitev sprožena zaradi spremembe samo enega piksla, je bilo namreč potrebno dodatno obdelati 31 nepomembnih pikslov v primeru podatkovnega tipa `char`. Nasprotno pa smo v primeru podatkovnega tipa `double` morali dodatno obdelati le 3 nepomembne piksle, preostalih 28 pa ne. Tako bi v najslabšem primeru, kjer bi se vedno posodabljal samo en piksel, pričakovali, da bi se pogostost posodabljanja med podatkovnima tipoma `char` in `double` zmanjšala za faktor 8. Iz podanih meritev je razviden faktor ≈ 4, kar nakazuje, da sta bila v povprečju vedno obdelana vsaj 2 piksla na posodobitev.

V naslednjem podpoglavju predstavimo rezultate večjedrnega procesiranja na vseh testnih sistemih.

Tabela 5.7: Upočasnitve izvajanja izpeljanih geodetskih operatorjev znotraj vsakega testnega sistema glede na podatkovni tip `char`.

	Podatkovni tip			
	char	short	float	double
Pričakovana upočasnitev rekonstrukcijskih filtrov	1,000	0,500	0,250	0,125
Sistem T1	1,000	0,521	0,202	0,075
Sistem T2	1,000	0,534	0,271	0,091
Sistem T3	1,000	0,530	0,261	0,116
Pričakovana upočasnitev granolumetričnih filtrov	1,000	0,500	0,250	0,125
Sistem T1	1,000	0,585	0,271	0,103
Sistem T2	1,000	0,577	0,291	0,127
Sistem T3	1,000	0,541	0,320	0,164

Tabela 5.8: Povprečni izvajalni časi posameznega filtra, podani v mikrosekundah, ter relativna upočasnitev v primerjavi s tradicionalnim filtrom ϵ_1.

Povprečni izvajalni časi (μs) posameznega filtra.

	T1	T2	T3	T1	T2	T3
Tip filtra		char			short	
Tradicionalni	45,9	34,9	77,3	90,5	65,4	140,6
Rekonstrukcijski	65,9	45,7	99,7	126,1	85,3	188,2
Granolumetrični	70,5	45,8	109,0	120,4	79,4	201,3
		float			double	
Tradicionalni	189,1	129,1	292,0	522,3	277,0	579,3
Rekonstrukcijski	328,3	168,0	382,9	888,3	502,9	862,7
Granolumetrični	260,4	157,4	340,4	683,3	360,5	664,1

Upočasnitev v primerjavi s tradicionalnim filtrom ϵ_1.

	T1	T2	T3	T1	T2	T3
Relativna upočasnitev		char			short	
Rekonstrukcijski	0,70	0,76	0,78	0,72	0,77	0,75
Granolumetrični	0,65	0,76	0,71	0,75	0,82	0,70
		float			double	
Rekonstrukcijski	0,58	0,77	0,76	0,59	0,55	0,67
Granolumetrični	0,73	0,82	0,86	0,76	0,77	0,87

Tabela 5.9: Števci dogodkov ter pogostost pisanja in branja slik med filtriranjem slike *Airplane* z operatorjema SPL in TDR.

Selektivno polnjenje robov – SPL

Podatkovni tip	Dogodek	Števec	Pogostost
char	Branje: f,m	54.134.091	100,00 %
	Pisanje: f	1.624.009	3,00 %
short	Branje: f,m	110.014.443	100,00 %
	Pisanje: f	2.007.704	1,82 %
float	Branje: f,m	221.775.147	100,00 %
	Pisanje: f	2.491.584	1,12 %
double	Branje: f,m	445.296.555	100,00 %
	Pisanje: f	3.141.496	0,71 %

Transformacija domnevnih razdalj – TDR

Podatkovni tip	Dogodek	Števec	Pogostost
char	Branje: f	32.474.112	100,00 %
	Branje: r(f), Pisanje: f	3.198.599	9,85 %
	Pisanje: r(f), d(f)	518.733	1,60 %
short	Branje: f	65.995.776	100,00 %
	Branje: r(f), Pisanje: f	4.223.840	6,40 %
	Pisanje: r(f), d(f)	659.550	1,00 %
float	Branje: f	133.039.104	100,00 %
	Branje: r(f), Pisanje: f	5.562.386	4,18 %
	Pisanje: r(f), d(f)	849.624	0,64 %
double	Branje: f	267.125.760	100,00 %
	Branje: r(f), Pisanje: f	7.562.925	2,83 %
	Pisanje: r(f), d(f)	1.121.401	0,42 %

5.2 Večjedrni testi

Tabela 5.10: Izvajalni časi (v milisekundah) procesiranja dolge verige 512 ϵ_1 filtrov ob povečujočem se številu niti ter uporabi različnih podatkovnih tipov.

Število niti	char			short			float			double		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
1	23,5	17,8	39,7	46,3	33,5	72,0	96,8	66,1	149,4	266,9	141,8	296,2
2	13,1	16,9	28,3	24,6	31,9	56,8	56,2	65,6	121,8	195,1	146,7	395,3
4	8,1	8,7	15,3	12,9	16,0	28,2	28,6	32,6	62,5	103,2	73,7	201,4
6	-	6,1	13,0	-	10,9	20,5	-	22,4	46,3	-	50,6	143,1
8	-	7,1	12,3	-	14,0	15,6	-	28,1	34,9	-	56,3	112,2
12	-	4,8	10,5	-	9,4	13,9	-	18,7	24,7	-	37,7	76,4
16	-	4,4	8,0	-	8,5	13,7	-	17,0	18,2	-	34,0	58,5
18	-	3,9	7,3	-	7,7	12,3	-	15,2	16,8	-	30,7	51,7
24	-	3,5	5,7	-	6,9	10,2	-	13,5	13,4	-	26,8	39,4
32	-	-	4,3	-	-	8,2	-	-	12,1	-	-	30,6
34	-	-	4,2	-	-	7,6	-	-	11,7	-	-	30,8
64	-	-	2,6	-	-	4,6	-	-	7,8	-	-	17,4

Pohitritve, ki jih predlagana metoda doseže s hkratno izvedbo nad več jedri, smo ovrednotili z uporabo $1, 2, 4, 6, 8, 12, 16, 18, 24, 32, 34$ in 64 niti. Zaradi manjšega števila jeder na sistemih T1 in T2 smo v teh primerih teste izvedli zgolj z največ 4 in 24 nitmi. Iz enonitnih testov je razvidno, da so bili izvajalni časi linearno odvisni od dolžine verige, zato smo testiranje najprej izvedli s konstantno verigo dolžine 512 filtrov ϵ_1. Tako smo zagotovili, da je bilo na vseh testnih sistemih opravljenih več iteracij skozi procesni cevovod ne glede na število uporabljenih niti. Pridobljene meritve izvajalnih časov elementarnih filtrov ob uporabi različnih podatkovnih tipov in števila niti vidimo v tabelah 5.10 in 5.11. Slednja predstavlja relativne pohitritve v primerjavi z enonitnimi izvajalnimi časi.

Pri tem je očitno, da sistem T1 ni dosegel linearnega skaliranja v primeru podatkovnih tipov `char` in `double`. Pri slednjem je bilo to tudi pričakovano, saj so zaradi premajhnega predpomnilnika L1D in L3 nelinearno odvisnost po-

kazali že enojedrni testi. Ob uporabi podatkovnega tipa `char` pa je obdelava vrstic potekala tako hitro, da je zakasnitev predpomnilnika omejevala medjedrni prenos podatkov med nitmi (glej podpoglavje 5.3).

Nasprotno je sistem T2 izkazoval popolno skaliranje do šestih niti oziroma treh jeder. Izjema so primeri z uporabo dveh niti, kjer so bili izvajalni časi enaki tistim z eno nitjo. Slednje je posledica uporabe SMT, saj sta se obe niti hkrati izvajali na istem fizičnem jedru. Že ena nit je v celoti izkoristila pasovno širino jedra za prenos podatkov med medpomnilnikom L1D in vektorskimi registri. Posledično uporaba druge niti ni izboljšala računskih časov. Tako je uporaba šestih niti oziroma treh jeder, omogočala trikratno pohitritev. Pri uporabi več kot šestih niti je učinkovitost skaliranja pričela padati, ob uporabi osmih niti pa se je čas izračuna celo poslabšal. V tem primeru smo namreč za izračun uporabili več podenot, pri čemer pa je nezadostna pasovna širina med podenotami bistveno omejila učinkovitost skaliranja predlagane metode. Vseeno lahko opazimo, da je uporaba več celotnih podenot (12, 18 in 24 niti oziroma 2, 3 in 4 podenot) vedno omogočila izboljšanje računskih časov.

Podobno je tudi sistem T3 omogočal uporabo SMT z izvajanjem dveh niti na enem fizičnem jedru. Pri tem pa zaradi doseganja nižje stopnje paralelizma na nivoju ukaznega nabora ena nit ni v celoti izkoristila jedra pri prenosu podatkov med medpomnilnikom L1D in vektorskimi registri. Posledično je uporaba dveh niti na tem sistemu vodila do izboljšanja računskih časov v vseh primerih, razen v primeru podatkovnega tipa `double`, kjer je bil deljeni predpomnilnik L1D premajhen za hrambo celotnih medpomnilnikov filtrov. Učinkovitost uporabe dveh niti je z večanjem podatkovnega tipa padala, saj se je tako povečalo tudi število iteracij notranje zanke (glej vrstico 8 v algoritmu 3). To je povečalo stopnjo paralelizma na nivoju ukaznega nabora in posledično je ena nit bolje izkoristila celotno jedro CPE. Vseeno pa je ob uporabi 6 ali več niti s `char`, 8 ali več niti s `short` in 18 ali več niti s `float` učinkovitost skaliranja pričela upadati glede na število uporabljenih jeder. Ne glede na podatkovni tip se je z večanjem niti namreč začel povečevati tudi vpliv zamika

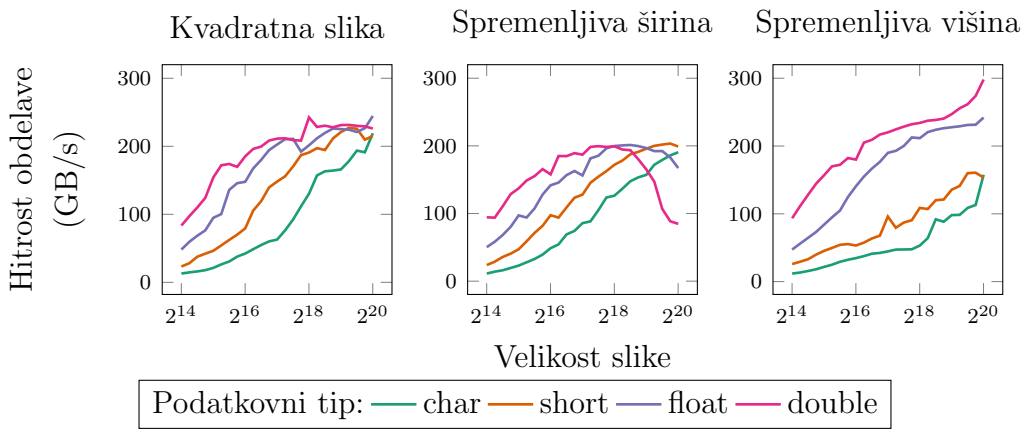
predpomnilnika L3. Slednji ni bil zmožen streči podatkov vsem nitim hkrati in posledično je bil ob povečevanju niti čedalje bolj zasičen. Uporaba večjih podatkovnih tipov je podaljšala čas za medjedrno sinhronizacijo, zato je tam prišlo do zasičenja šele pri večjem številu niti. Povezava med monolitnima CPE pa se je izkazala za dovolj hitro, da se hitrost procesiranja ni poslabšala, tudi ko smo uporabili samo eno jedro na drugem CPE (34 niti). Uporaba dveh monolitnih CPE (64 niti) pa vseeno ni nikoli doprinesla dvakratne pohitritve.

Tabela 5.11: Relativna pospešitev večnitnih izvajalnih časov v primerjavi z enonitnimi izvajalnimi časi.

Število niti	char			short			float			double		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
2	1,8	1,1	1,4	1,9	1,1	1,3	1,7	1,0	1,2	1,4	1,0	0,7
4	2,9	2,0	2,6	3,6	2,1	2,6	3,4	2,0	2,4	2,6	1,9	1,5
6	-	2,9	3,1	-	3,1	3,5	-	3,0	3,2	-	2,8	2,1
8	-	2,5	3,2	-	2,4	4,6	-	2,4	4,3	-	2,5	2,6
12	-	3,7	3,8	-	3,6	5,2	-	3,5	6,0	-	3,8	3,9
16	-	4,0	5,0	-	3,9	5,3	-	3,9	8,2	-	4,2	5,1
18	-	4,6	5,4	-	4,4	5,9	-	4,3	8,9	-	4,6	5,7
24	-	5,1	7,0	-	4,9	7,1	-	4,9	11,1	-	5,3	7,5
32	-	-	9,2	-	-	8,8	-	-	12,3	-	-	9,7
34	-	-	9,5	-	-	9,5	-	-	12,8	-	-	9,6
64	-	-	15,3	-	-	15,7	-	-	19,2	-	-	17,0

Da smo preverili učinkovitost večnitnih računskih časov pri različnih geometrijah vhodne slike in geodetskih operatorjev, so bila nadaljnja testiranja opravljena le pri maksimalnem številu niti, ki jih je podpiral posamezen testni sistem, medtem ko so preostale omejitve ostale nespremenjene. Tako smo v primeru sistema T1 uporabili 4 niti, sistem T2 smo ovrednotili s 24 nitmi in sistem T3 ob uporabi 64 niti. Meritve vpliva dimenzije slike na večnitne izvajalne čase so podane na sliki 5.7. Anomalije, ki smo jih zaznali pri enonitnih testih ob povečevanju širine slike v primeru sistemov T1 in T3 niso bile opazne. Prenos podatkov med jedri je namreč potekal preko predpomnil-

nika L3, ki je počasnejši od predpomnilnika L2. Zato so anomalije presežene kapacitete predpomnilnika L1D ostale zakrite. Kljub temu pa se je ponovno pojavilo nenadno poslabšanje izvajalnih časov, ko je bila presežena kapaciteta predpomnilnika L3 s podatkovnim tipom `double`. Nasprotno se računski časi ob povečevanju višine slike niso bistveno razlikovali od časov, doseženih z referenčno kvadratno sliko. Slednje nakazuje, da zamik predpomnilnika L3 ni bistveno vplival na računske čase. Nasprotno pa je zamik med različnimi podenotami na sistemu T2 bistveno vplival na učinkovitost izvedbe testnih verig filtrov. To je očitno iz meritev izvedbe nad referenčno kvadratno sliko ter razširjenimi slikami. V teh primerih so se namreč računski časi nenadno izboljšali, ko je bila slika dovolj široka oziroma ko je ena vrstica slike zavzemala vsaj ≈ 500 zlogov pomnilnika. Vpliv zamika predpomnilnika vidimo tudi na povečujoči se višini slike, kjer se je čas izračuna izboljševal do podatkovnega tipa `double`. Pri slednjem in pri podatkovnem tipu `float` so bili računski časi neodvisni od dimenzij vhodne slike. Na sistemu T3 je bil vpliv zamika predpomnilnika poudarjen zaradi večjega števila niti. Ob povečevanju širine slike so bili namreč doseženi konstantni časi izračuna, dokler ni slikovna vrstica presegla velikosti ≈ 1000 zlogov. V primerjavi z referenčno kvadratno slike je bilo zato pričakovano, da je povečevanje višine slike dosegalo slabše izvajalne čase, saj je bila širina slike omejena na 128 pikslov. Izjema je bila uporaba podatkovnega tipa `double`, kjer je velikost predpomnilnika L1D omejevala hitrost izračuna verige filtrov nad referenčno kvadratno sliko, kar pa se ni zgodilo v primeru omejene širine slike. Hitrost procesiranja na sistemu T3 je podana na sliki 5.6, kjer je viden vpliv prekoračitve predpomnilnika L1D pri večanju širine slike. Ko je vrstica slike presegla pomnilniško velikost ≈ 11 kilozlogov, je pričela učinkovitost procesiranja strmo padati, saj je vsaka nit morala prebrati čedalje več podatkov iz predpomnilnika L2. Ker do prekoračitev ni prišlo pri kvadratni sliki ali ob povečevanju višine slike, je bilo naraščanje hitrosti procesiranja z večanjem vhodne slike pričakovano. Za vse tri testne sisteme pa velja, da nižja frekvenca CPE ne razloži nižje stopnje skaliranja ob uporabi večjega števila niti. Iz rezultatov lahko zato sklepamo, da je na sistemih T1 in

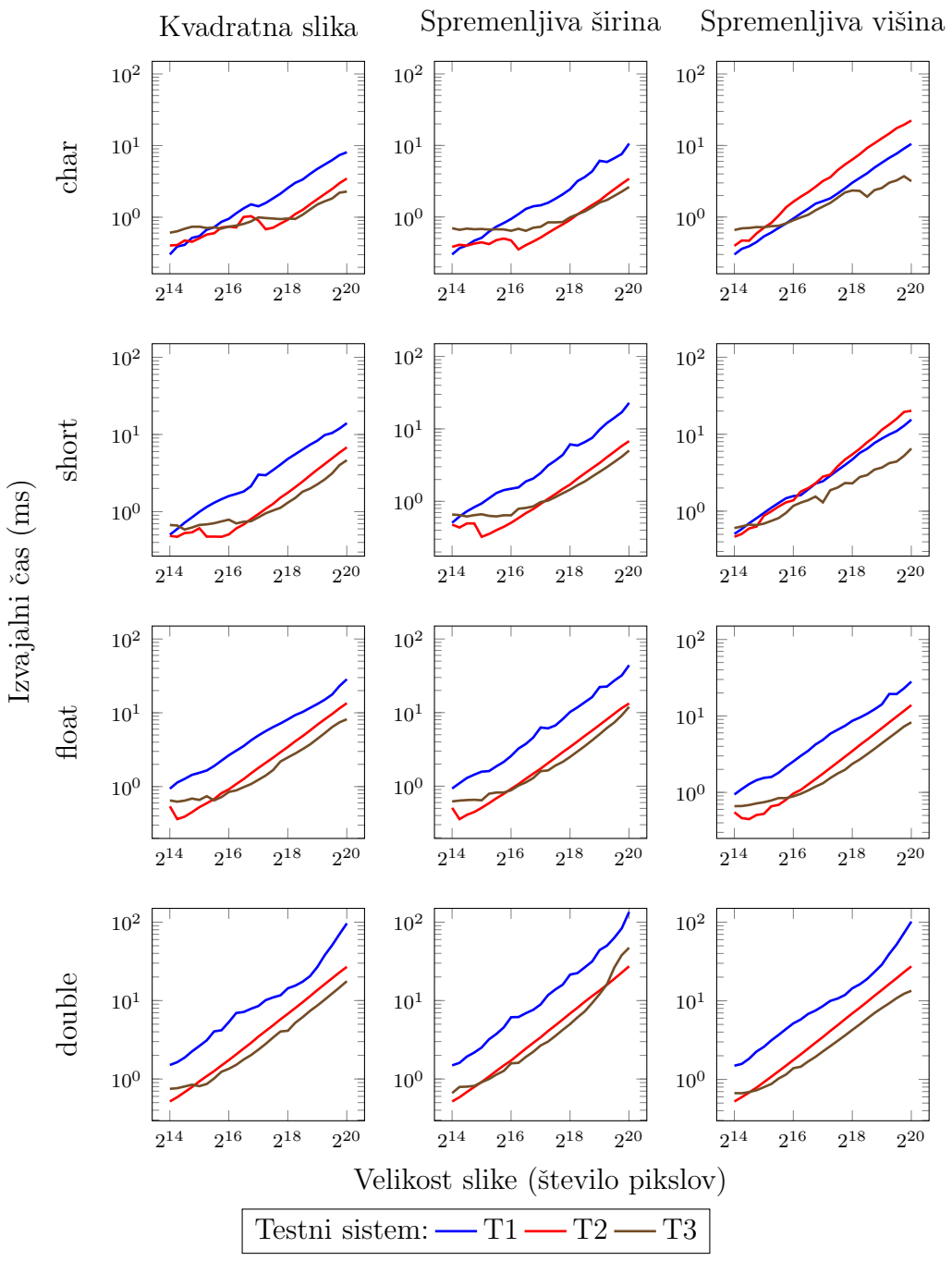


Slika 5.6: Hitrost obdelave vhodne slike na sistemu T3 s 64 nitmi in verigo 512 filtrov ϵ_1.

T3 z monolitnim CPE največje ozko grlo predstavljal zamik predpomnilnika L3. Nasprotno pa je bilo ozko grlo na sistemu T2 nezadostna pasovna širina med podenotami heterogenega CPE.

Večnitni izvajalni časi izpeljanih geodetskih operatorjev so podani v tabeli 5.12. Kot pričakovano, se je sistem T1 izkazal za najpočasnejšega, saj je bil omejen s številom niti in velikostjo predpomnilnika L3. Vseeno pa je razvidno iz tabele 5.13, da so bili izvajalni časi posamičnega filtra podobni enonitnim izvajalnim časom. Izračun tako ni bil bistveno omejen zaradi zamika predpomnilnika pri prenosu podatkov.

Sistema T2 in T3 sta bila glede na izvajalne čase podobna, čeprav je bil sistem T3 skoraj dvakrat zmogljivejši. Oba sistema sta bila bistveno omejena zaradi zamika in prepustnosti predpomnilnika. Izračun posameznega filtra je bil od 2,5- do 5-krat počasnejši kot v primeru enonitnega izračuna (glej tabelo 5.8). Presenetljivo pa je bil na sistemu T2 večnitni izračun verige z rekonstrukcijskimi ali granolumetričnimi filtri vedno hitrejši kot izračun verige s tradicionalnim filtrom ϵ_1. To je bila posledica pogojnega posodabljanja slik, saj vsebine pomnilnika ni bilo potrebno posodabljati v primerih, ko je vsebina slike ob izvedbi filtra ostala nespremenjena. Posledično je CPE opravil



Slika 5.7: Večnitni izvajalni časi pri povečujoči se sliki, kjer so bile uporabljene 4 niti za sistem T1, 24 niti za sistem T2 in 64 niti za sistem T3.

bistveno manj sinhronizacije med jedri. Pogostost posodabljanja pri rekonstrukcijskih in granolumetričnih filtrih je bila enaka kot v primeru enonitnega izračuna (glej tabelo 5.9). Zaradi pogojnega posodabljanja sta tudi sistema T1 in T3 v primerjavi z enonitnimi rezultati dosegla manjše faktorje upočasnitve rekonstrukcijskega in granolumetričnega filtra. Oba sistema nista dosegla boljših izvajalnih časov kot tradicionalni filter ϵ_1.

Omejitve arhitekture določenih CPE, kot so na primer premajhen predpomnilnik L3 za celotno pomnjenje vhodnih slik ali premajhen predpomnilnik L1D za hranjenje medpomnilnikov filtrov, so bile prisotne tako v enonitnih kot večnitnih testih in jih ni bilo možno odpraviti. Iz večnitnih meritev je očitno, da predlagana metoda deluje s poljubnim številom niti, kjer pa je bilo skaliranje na večje število niti omejeno ali z nezadostno pasovno širino predpomnilnika (oziroma povezav med predpomnilniki podenot) ali z zamikom predpomnilnika. Kljub omejitvam je uporaba enega procesnega cevovoda in največjega števila niti na vsakem testnem sistemu dosegla najboljšo odzivnost. Težavo slabega skaliranja naslovimo v naslednjem podpoglavju, kjer smo na vsakem testnem sistemu z vzporedno uporabo več ločenih procesnih cevovodov preverili največjo možno prepustnost.

Tabela 5.12: Večnitni izvajalni časi (v milisekundah) izpeljanih filtrov pri različnih vhodnih slikah, testnih sistemih in podatkovnih tipih, kjer smo za izračun uporabili 4 niti na sistemu T1, 24 niti ob uporabi sistema T2 ter 64 niti za sistem T3.

Operator	Slika	Povprečna dolžina verige	char			short			float			double		
			T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
SPL	Airport	1706	28,8	7,3	7,6	53,1	13,2	12,1	175,8	28,2	31,7	502,7	77,9	59,5
	Male	852	14,9	4,0	4,1	27,7	7,0	7,0	90,2	14,8	16,9	251,1	40,8	32,3
	Sand	747	14,4	3,7	3,9	25,6	6,6	6,7	81,9	13,8	13,9	223,7	36,9	28,0
OO	Airport	1463	25,7	6,7	6,7	47,9	11,9	11,3	153,1	31,6	28,7	433,8	73,7	54,5
	Male	1648	27,4	7,9	7,7	52,4	13,8	13,0	171,9	34,5	32,0	483,4	81,0	60,7
	Sand	957	19,2	5,0	5,1	35,5	8,8	9,1	107,3	24,7	18,7	288,8	53,0	37,7
γ_{rek}	Airport	1291	21,3	7,2	5,8	38,2	12,3	10,3	118,2	25,3	23,7	342,8	62,5	44,9
	Male	1330	24,5	7,0	6,0	42,6	12,0	10,4	140,4	25,1	24,5	296,6	61,5	47,5
	Sand	1813	35,3	9,7	8,2	58,4	17,4	16,2	150,2	36,0	29,2	386,2	91,0	59,7
RK	Airport	733	14,1	4,0	3,5	24,8	6,5	6,1	78,6	14,0	14,9	215,4	35,6	27,5
	Male	647	13,9	3,6	3,4	23,1	5,9	5,6	75,4	12,6	13,2	154,7	31,0	25,1
	Sand	794	17,0	4,3	4,1	28,1	6,9	7,6	75,9	15,1	15,2	188,8	40,8	28,6
IK	Airport	734	14,3	4,3	4,0	25,1	6,7	6,9	79,9	20,9	16,0	216,8	42,1	29,6
	Male	648	13,9	3,7	3,8	23,2	6,1	6,3	76,7	19,5	14,3	156,5	37,4	27,5
	Sand	795	17,0	4,6	4,6	28,2	7,2	8,3	77,4	21,9	17,6	190,7	46,8	33,1
TDR	Airport	1070	21,7	6,6	6,3	36,3	10,9	11,5	79,0	19,7	20,9	203,2	40,1	42,9
	Male	1064	20,2	5,6	5,9	35,0	9,6	10,9	68,6	18,2	19,5	175,6	36,0	43,1
	Sand	1054	20,5	5,6	5,9	34,5	9,9	10,9	79,9	18,3	21,1	199,6	37,2	42,0

Tabela 5.13: Izvajalni časi posameznega filtra v mikrosekundah ob večnitnem izvajanju, njihova relativna upočasnitev v primerjavi s tradicionalnim filtrom ϵ_1 in v primerjavi z enonitnim izračunom.

Povprečni izvajalni časi (μs) posameznega filtra.

	T1	T2	T3	T1	T2	T3
Tip filtra	char			short		
Tradicionalni	63,3	164,1	325,0	100,8	323,4	575,0
Rekonstrukcijski	74,7	123,3	310,9	132,2	211,4	542,2
Granolumetrični	78,3	134,0	363,4	132,7	228,9	668,5
	float			double		
Tradicionalni	223,4	632,8	975,0	806,3	1.256,3	2.175,0
Rekonstrukcijski	409,2	502,0	1.229,9	1.118,3	1.206,1	2.361,5
Granolumetrični	285,4	423,1	1.234,6	725,7	852,9	2.569,6

Upočasnitev v primerjavi s tradicionalnim filtrom ϵ_1.

	T1	T2	T3	T1	T2	T3
Relativna upočasnitev	char			short		
Rekonstrukcijski	0,84	1,33	1,04	0,76	1,53	1,06
Granolumetrični	0,81	1,21	0,89	0,76	1,42	0,87
	float			double		
Rekonstrukcijski	0,55	1,26	0,79	0,72	1,04	0,92
Granolumetrični	0,78	1,50	0,79	1,11	1,47	0,85

Upočasnitev glede na enonitni izračun (podan v tabeli 5.8).

	T1	T2	T3	T1	T2	T3
Tip filtra	char			short		
Tradicionalni	0,73	0,21	0,24	0,90	0,20	0,24
Rekonstrukcijski	0,88	0,37	0,32	0,95	0,40	0,35
Granolumetrični	0,90	0,34	0,30	0,91	0,35	0,30
	float			double		
Tradicionalni	0,85	0,20	0,30	0,65	0,22	0,27
Rekonstrukcijski	0,80	0,33	0,31	0,79	0,42	0,37
Granolumetrični	0,91	0,37	0,28	0,94	0,42	0,26

5.3 Optimizacija odzivnosti in prepustnosti

Iz rezultatov večnitnih testov je razvidno, da uporaba večjega števila niti v procesnem cevovodu lahko vodi v slabše faktorje skaliranja oziroma slabšo računsko učinkovitost testnih sistemov. Zato se osredotočimo na določitev optimalnega števila vzporednih cevovodov, da dosežemo največjo računsko učinkovitost posameznega testnega sistema. V ta namen smo na vsakem testnem sistemu in za vsak podatkovni tip izvedli testiranja nad 192 verigami filtrov, sestavljenih iz 512 operatorjev ϵ_1. Ker smo uporabili več verig filtrov, je vsaka veriga procesirala neodvisno oziroma ločeno vhodno sliko *Sand*. Meritve smo najprej izvedli z uporabo ene niti na procesni cevovod, število uporabljenih niti pa smo nato povečevali do omejitve uporabljenega testnega sistema. Za popoln izkoristek vseh jeder oziroma niti testnega sistema smo uporabili primerno število neodvisnih cevovodov, ki so se izvajali vzporedno. Kadar smo imeli več kot en cevovod, pa smo 192 verig filtrov enakomerno razdelili med njih. Tako je bilo zagotovljeno, da je bilo vedno opravljeno enako število operacij, neodvisno od uporabljenega števila niti na cevovod, podatkovnega tipa ali testnega sistema. Meritve izvajalnih časov, ločene na posamezne testne sisteme, so podane v tabeli 5.14. Pri tem še enkrat omenimo, da so bili testi opravljeni z različnimi nabori niti in posledično različnim številom vzporednih cevovodov.

Najkrajše računske čase in največje prepustnosti smo dosegli z uporabo več vzporednih cevovodov. Pri tem so se optimalni rezultati posameznih testnih sistemov bistveno razlikovali. Tako je bila na primer pri sistemu T1 s podatkovnim tipom `char` optimalna uporaba štirih procesnih cevovodov, kjer je vsak uporabljal samo eno nit. Pri podatkovnem tipu `short` se je to zmanjšalo na uporabo dveh cevovodov, kjer je vsak uporabljal dve niti. V tem primeru je uporaba štirih cevovodov z eno nitjo povzročila, da je bil presežen predpomnilnik L3, saj je bila vhodna slika velika 2 MiB. Slednje je bil razlog za nižje prepustnosti tudi ob uporabi podatkovnih tipov `float` in `double`, kjer je bila

največja prepustnost dosežena z uporabo enega samega cevovoda. Sistem T2 je dosegel podobne računske čase, ko je bilo uporabljenih 1, 2 ali 6 niti na cevovod. Izjema je bila v primeru podatkovnega tipa `float` in 24 vzporednih cevovodov, saj je bil na vseh podenotah presežen 16 MiB velik predpomnilnik L3. Slednje se je izkazalo tudi v primeru podatkovnega tipa `double` s 24 in 12 vzporednimi procesnimi cevovodi. V primerih, ko je bilo uporabljenih 4 ali 8 niti na cevovod, pa so bili računski časi bistveno daljši, saj je procesiranje potekalo na dveh podenotah. To je bilo pričakovano, saj smo že pri večnitnih meritvah zaznali ozko grlo ob uporabi več podenot. Najzmogljivejši sistem T3 pa je dosegel najkrajše izvajalne čase z uporabo 2/4, 4/8, 8/16 in 8/16 niti na cevovod pri podatkovnih tipih `char`, `short`, `float` in `double`. Po pričakovanjih so se računski časi bistveno poslabšali, ko je skupna velikost slik presegla 40 MiB velik predpomnilnik L3, saj je bil izračun omejen s pasovno širino glavnega pomnilnika. V vseh primerih smo iz podanih meritev lahko določili vpliv števila niti na cevovod in števila vzporednih cevovodov na odzivnost in prepustnost procesiranja. Optimalno prepustnost smo lahko ovrednotili glede na povprečen čas procesiranja verige filtrov. Rezultati so podani v tabeli 5.15.

V primerjavi z večnitnimi meritvami je na sistemu T1 prišlo do izboljšanja računskih časov samo v primeru podatkovnih tipov `char` in `double`, medtem ko so računski časi pri preostalih dveh podatkovnih tipih ostali pretežno nespremenjeni. Ker je testni sistem imel nizko število jeder, nismo pričakovali bistvenih razlik. Nasprotno pa je sistem T2 dosegel bistveno izboljšavo računskih časov, ki so se prepolovili pri vseh podatkovnih tipih. Posledično se je izboljšala tudi relativna pospešitev v primerjavi z enonitnimi izvajalnimi časi, kjer je uporaba podatkovnih tipov `char`, `short`, `float` in `double` v teh primerih dosegla faktorje pohitritev $\approx 11{,}1$; $\approx 11{,}1$; $\approx 10{,}5$ in $\approx 9{,}8$. Ker so bila obremenjena vsa jedra, pa je posledično CPE deloval pri nižji frekvenci kot v primeru uporabe ene same niti. Zato smo sklepali, da smo v primeru podatkovnih tipov `char` in `short` dosegli popolno linearno pohitritev glede na število uporabljenih jeder. Pri podatkovnem tipu `float` in `double` tega ne moremo trditi, saj sta faktorja pričela upadati. Razlog za to je bil premajhen predpo-

Tabela 5.14: Meritve izvajalnih časov izračuna 192 verig filtrov, kjer je vsaka veriga sestavljena iz 512 filtrov ϵ_1. Vsi podani časi so v sekundah, najboljši rezultati so poudarjeni s krepko pisavo.

Število niti na cevovod	Št. vzporednih cevovodov	Podatkovni tip			
		char	short	float	double
1	4	**1,35**	4,81	20,42	47,79
2	2	1,42	**2,65**	10,12	27,96
4	1	1,87	2,66	**5,77**	**16,22**

(a) Sistem T1

Število niti na cevovod	Št. vzporednih cevovodov	Podatkovni tip			
		char	short	float	double
1	24	**0,31**	0,61	5,30	52,77
2	12	**0,31**	**0,58**	**1,22**	6,05
4	6	0,65	1,32	2,63	5,18
6	4	0,35	**0,59**	**1,21**	**2,77**
8	3	0,98	1,99	3,97	7,91
12	2	0,66	1,32	2,65	5,30
24	1	0,66	1,37	2,72	5,46

(b) Sistem T2

Število niti na cevovod	Št. vzporednih cevovodov	Podatkovni tip			
		char	short	float	double
1	64	0,38	11,44	23,34	28,64
2	32	**0,22**	0,72	11,49	25,91
4	16	**0,22**	**0,40**	1,09	12,87
8	8	0,31	**0,42**	**0,94**	**3,73**
16	4	0,37	0,63	**0,94**	**3,87**
32	2	0,38	0,77	1,15	4,11
64	1	0,40	0,81	1,43	6,85

(c) Sistem T3

mnilnik L1D, saj sta se zaradi uporabe SMT sedaj na enem jedru izvajali dve niti. Posledično ni bilo vanj možno istočasno hraniti celotnih medpomnilnikov obeh filtrov, zaradi česar je bil uporabljen počasnejši predpomnilnik L2. Ker je podatkovni tip `double` zahteval dvojno količino prostora, pa je bila uporaba predpomnilnika L2 povečana, kar je dodatno zmanjšalo faktor pohitritve. Slednje se je primerilo tudi na sistemu T3, kjer je uporaba podatkovnih tipov `float` in `double` dosegla faktorje pohitritev $\approx 30{,}6$ in $\approx 15{,}2$ v primerjavi z enonitnimi rezultati. S podatkovnim tipom `char` in `short` pa je bil dosežen faktor pohitritve $\approx 34{,}8$ in $\approx 34{,}1$. Slednja faktorja sta presegla število uporabljenih jeder, čeprav je bila frekvenca CPE nižja zaradi polne obremenitve testnega sistema. Višje faktorje lahko pojasnimo z ugotovitvami iz večnitnih meritev, kjer smo potrebovali dve niti za zasičenje prenosa med predpomnilnikom L1D in vektorskimi registri. Nasprotno pa smo odzivnost procesiranja določili glede na povprečen čas izračuna verige 512 filtrov ϵ_1 znotraj posameznega cevovoda. Povprečni časi so podani v tabeli 5.16. Meritve so potrdile, kar so nakazovale večnitne meritve. Tako je bila neglede na podatkovni tip najboljša odzivnost procesiranja verige filtrov dosežena pri uporabi samo enega procesnega cevovoda, pri čemer pa je ta uporabil največje podprto število niti glede na posamezen testni sistem. Kadar je bila presežena velikost predpomnilnika L3, so se izvajalni časi posameznega cevovoda bistveno povečali, saj je bil izračun omejen s pasovno širino glavnega pomnilnika. Slednja je bila na testnih sistemih približno desetkrat manjša od hitrosti predpomnilnika L3, kar se odraža tudi v povečanih časih izračuna. Meritve predlagane metode v naslednjem podpoglavju postavimo v širši kontekst s primerjavo računskih časov sorodnih metod.

Tabela 5.15: Povprečni izvajalni časi izračuna ene verige filtrov izmed uporabljenih 192 verig filtrov. Vsi podani časi so v milisekundah, najboljši rezultati so poudarjeni s krepko pisavo.

Število niti na cevovod	Št. vzporednih cevovodov	Podatkovni tip			
		char	short	float	double
1	4	**7,0**	25,1	106,4	248,9
2	2	7,4	**13,8**	52,7	145,7
4	1	9,8	13,9	**30,0**	**84,5**

(a) Sistem T1

Število niti na cevovod	Št. vzporednih cevovodov	Podatkovni tip			
		char	short	float	double
1	24	**1,6**	3,2	27,6	274,8
2	12	**1,6**	**3,0**	**6,4**	31,5
4	6	3,4	6,9	13,7	27,0
6	4	1,8	**3,1**	**6,3**	**14,4**
8	3	5,1	10,4	20,7	41,2
12	2	3,4	6,9	13,8	27,6
24	1	3,4	7,1	14,2	28,4

(b) Sistem T2

Število niti na cevovod	Št. vzporednih cevovodov	Podatkovni tip			
		char	short	float	double
1	64	2,0	59,6	121,5	149,2
2	32	**1,1**	3,7	59,8	134,9
4	16	**1,1**	**2,1**	5,7	67,0
8	8	1,6	**2,2**	**4,9**	**19,4**
16	4	1,9	3,3	**4,9**	**20,1**
32	2	2,0	4,0	6,0	21,4
64	1	2,1	4,2	7,5	35,7

(c) Sistem T3

Tabela 5.16: Povprečni izvajalni časi enega cevovoda ob procesiranju verige 512 filtrov ϵ_1. Podani izvajalni časi so v milisekundah, najboljši rezultati so poudarjeni s krepko pisavo.

Število niti na cevovod	Št. vzporednih cevovodov	Podatkovni tip			
		char	short	float	double
1	4	28,0	100,2	425,5	995,7
2	2	14,8	27,6	105,5	291,3
4	1	**9,8**	**13,9**	**30,0**	**84,5**

(a) Sistem T1

Število niti na cevovod	Št. vzporednih cevovodov	Podatkovni tip			
		char	short	float	double
1	24	38,63	76,45	662,56	6.595,72
2	12	19,22	36,19	76,41	378,00
4	6	20,43	41,13	82,26	161,89
6	4	7,29	12,36	25,25	57,68
8	3	15,36	31,10	62,08	123,61
12	2	6,87	13,75	27,59	55,23
24	1	**3,44**	**7,13**	**14,15**	**28,42**

(b) Sistem T2

Število niti na cevovod	Št. vzporednih cevovodov	Podatkovni tip			
		char	short	float	double
1	64	125,1	3.811,9	7.778,9	9.548,1
2	32	36,4	119,4	1.914,7	4.317,9
4	16	18,2	33,7	91,0	1.072,5
8	8	13,1	17,3	39,1	155,3
16	4	7,7	13,2	19,5	80,5
32	2	4,0	8,0	12,0	42,8
64	1	**2,1**	**4,2**	**7,5**	**35,7**

(c) Sistem T3

5.4 Primerjava s sorodnimi metodami

V tem podpoglavju se osredotočamo na primerjavo učinkovitosti predlaganega pristopa s sorodnimi metodami. Ker slednje običajno temeljijo na uporabi paradigme SISD, so njihove implementacije v odprtokodnih knjižnicah omejene na eno samo nit, skalarno procesiranje, uporabo podatkovnega tipa `char` ali kombinacijo le-teh. Pri izvedbi testov smo sledili tem omejitvam, vsa morebitna odstopanja pa so v nadaljevanju podrobno opisana. Primerjave smo pri tem izvajali nad sliko *Sand*. Ker je pri tem očitno, da vsebina slike neposredno vpliva na dolžino verige filtrov (glej sliko 5.5), čas izvedbe geodetskih operatorjev pa je od slednje linearno odvisen, lahko iz navedenih rezultatov ocenimo tudi izvajalne čase na poljubnih slikah, v kolikor poznamo dolžino tvorjene verige.

V prvi analizi smo se osredotočili na primerjavo predstavljene metode z obstoječimi implementacijami v popularnih odprtokodnih knjižnicah za obdelavo slik, kot sta knjižnici OpenCV (različica 4.4) in SMIL (različica 0.10). Ker OpenCV ne implementira večnitnega procesiranja pri izvedbi morfološkega filtriranja, smo se pri izvedbi testiranj omejili na rezultate, dosežene z eno nitjo. Vseeno pa knjižnica omogoča izvedbo morfološkega krčenja in širjenja z uporabo vektorskega procesiranja z naborom AVX2. Ker knjižnica nudi tudi neposredno izvedbo filtra s poljubno velikostjo, smo teste opravili pri različnih velikostih filtra in temu ustrezni dolžini verige. Tako smo v primeru knjižnice omogočili optimalen izračun filtra, medtem ko je bila predlagana metoda vedno omejena samo na izračun verige filtrov.

Slika 5.8 prikazuje primerjavo izvajalnih časov, doseženih s predlagano metodo in izvedbo morfoloških operatorjev v knjižnici OpenCV. Iz meritev je razvidno, da je predlagana metoda primerljiva ali učinkovitejša od implementacije OpenCV na vseh testnih sistemih. Konkretneje, knjižnica OpenCV je dosegla primerljive rezultate s predlagano metodo zgolj pri obdelavi podatkov-

nih tipov `char` in `short` ter izvedbi filtra z velikim strukturnim elementom, medtem ko je izvedba verige filtrov, kot jo zahtevajo geodetski operatorji, bila bistveno počasnejša v vseh primerih. S podatkovnim tipom `float` so bili izvajalni časi implementacije OpenCV opazno višji na sistemu T2, z `double` pa na vseh testnih sistemih. Nasprotno je bila izvedba verig filtrov vedno počasnejša od predlagane metode, kjer se je sam klic funkcije krčenja izkazal za časovno neučinkovitega. Čeprav je imela funkcija krčenja možnost navedbe števila iteracij, tega nismo uporabili, saj je potrebno v postopku rekonstrukcije vsako iteracijo omejiti s sliko maske. Nasprotno pa knjižnica SMIL ni imela teh težav in je prav tako podpirala tako večnitno kakor tudi vektorsko procesiranje z uporabo ukaznega nabora AVX2. Izvajalni časi filtrov z velikim strukturnim elementom so tako primerljivi z izvajalnimi časi njihove dekompozicije in iterativnega izračuna verige filtrov. Zato na vseh testnih sistemih poročamo zgolj večnitne rezultate izračuna verige filtrov, podani na sliki 5.9, medtem ko so v tabeli 5.17 podani faktorji pohitritve predlagane metode. Iz tabele pohitritev je razvidno, da je predlagana metoda vedno presegla dosežene hitrosti knjižnice vsaj za faktor $\approx$ 2-krat. Do slabše pospešitve na sistemih T2 in T3 je prišlo zaradi uporabe SMT, saj so se v primeru predlagane metode jedra dodeljevala zaporedno. To je ob večnitnem izvajanju povzročilo izvajanje dveh niti na posameznem jedru, pri čemer pa je na primer na sistemu T2 že ena nit popolnoma izkoristila jedro CPE (glej podpoglavje 5.2) saj je izračun minimalne vrednosti neposredno odvisen od hitrosti nalaganja vrednosti iz predpomnilnika L1D. Knjižnica SMIL pa je, nasprotno, uporabljala samo eno nit na fizično jedro, dokler ni število niti preraslo števila jeder testnega sistema. Posledično je predlagana metoda na sistemu T2 dosegla dodaten dvakratni faktor pohitritve z 12 na 24 niti, saj so bila šele tedaj uporabljena vsa jedra CPE za izračun. Naivni vzporedni izračun posameznega filtra, ki ga implementira knjižnica SMIL, je na sistemu T3 vodil v drastično upočasnitev pri uporabi med 32 in 64 nitmi. To smo pripisali pretežno uporabi dveh fizično ločenih CPE. Predlagana metoda namreč uspe minimizirati prenos podatkov med enotami, medtem ko knjižnica SMIL ni zmožna izvesti smiselne razpore-

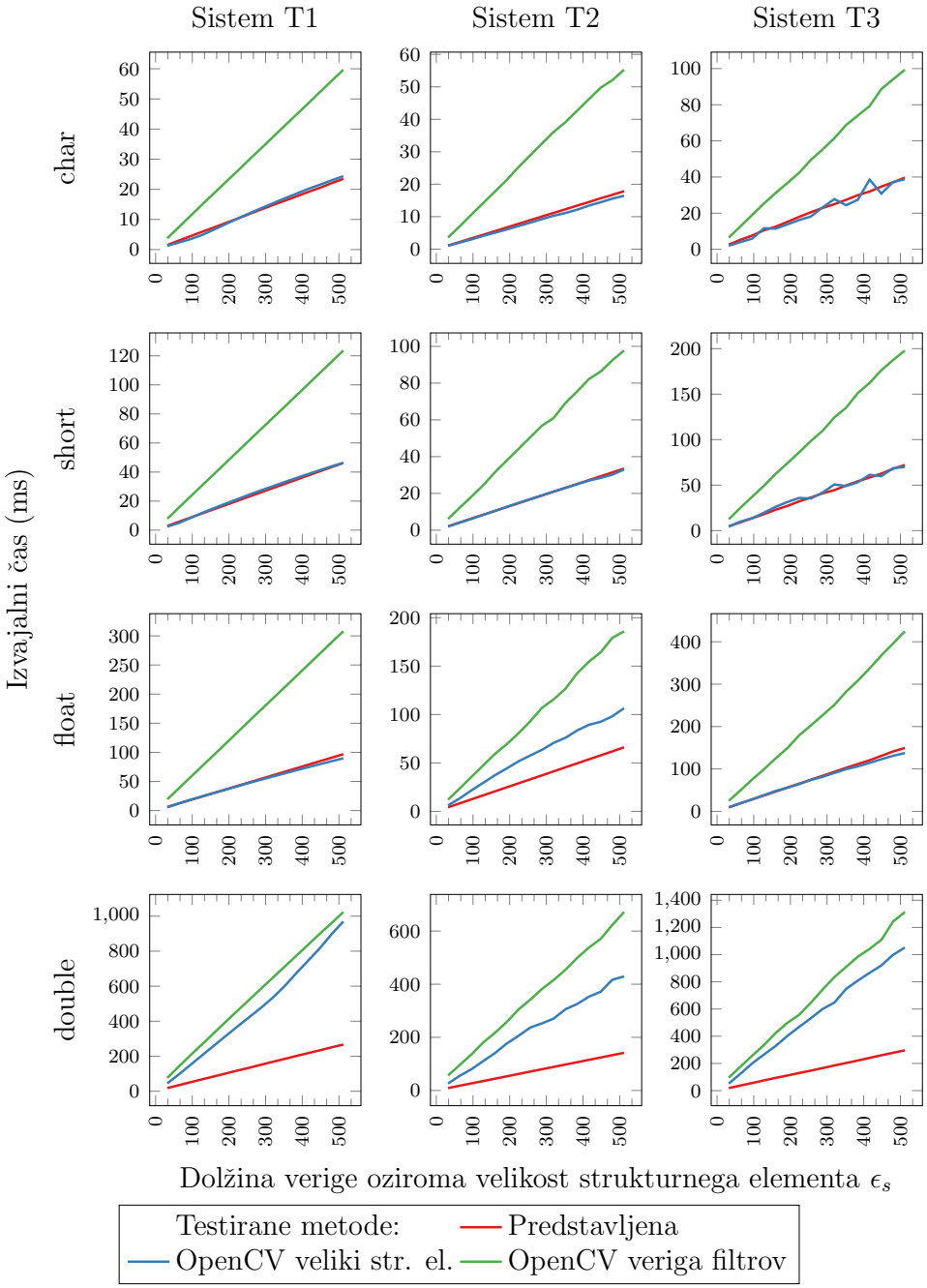
ditve niti. Slednje pa zahteva bistveno več komunikacije med parom CPE in je tako podaljšalo izvajalne čase. To dejstvo potrjujejo tudi rezultati sistema T3 na sliki 5.9, ki kažejo, da je implementacija knjižnice SMIL dosegla boljše izvajalne čase z uporabo 1, 2 ali 4 niti kakor z uporabo 64 niti pri obdelavi vseh podatkovnih tipov. Podobno poslabšanje smo zasledili tudi na sistemu T2, kjer je knjižnica SMIL z uporabo vseh 24 niti dosegala slabše izvajalne čase kot z uporabo zgolj 6 niti.

Tabela 5.17: Faktor pohitritve predlagane metode v primerjavi s knjižnico SMIL pri izvedbi verig 512 filtrov ϵ_1.

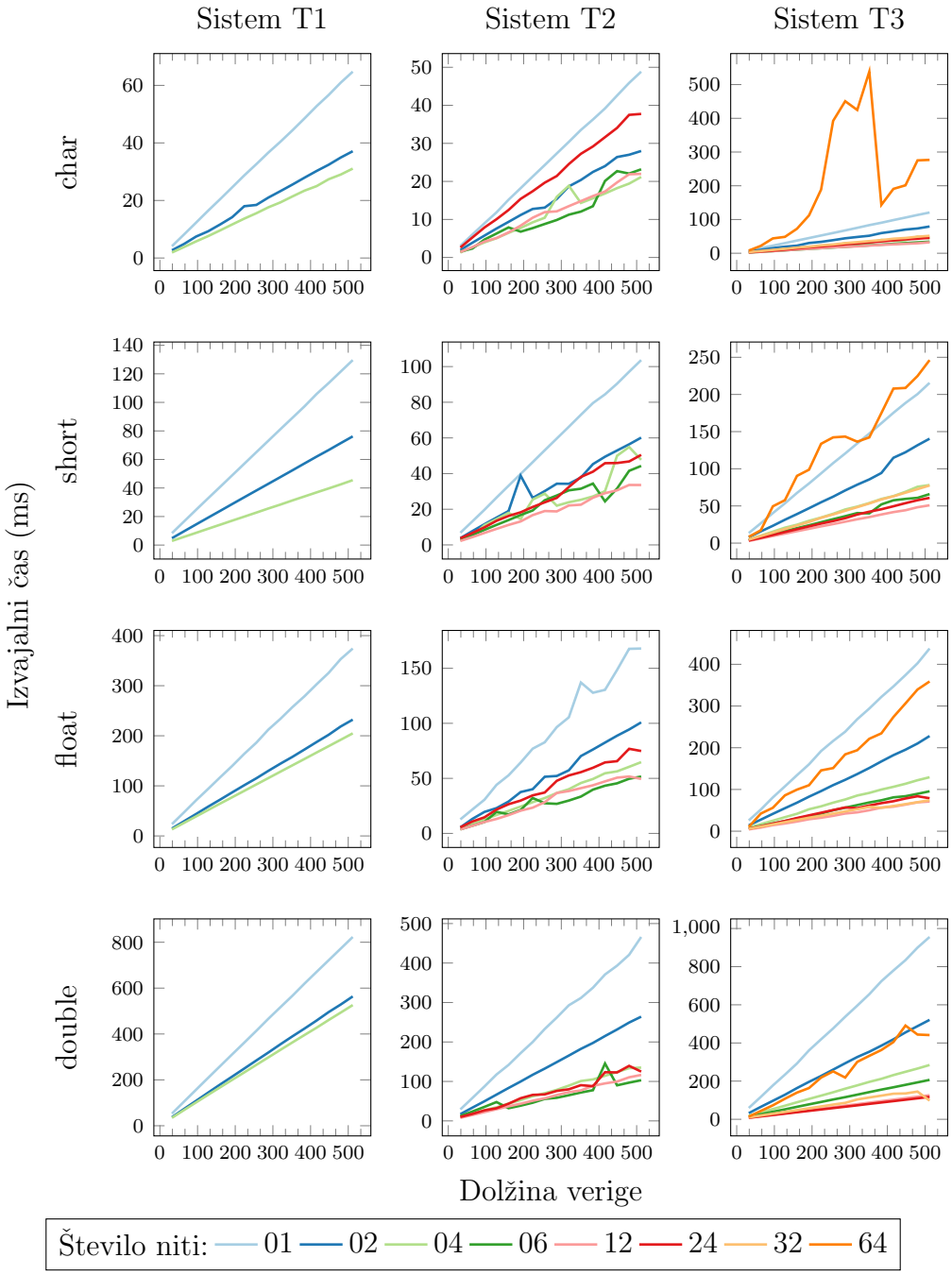
Število niti	char			short			float			double		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
1	2.8	2.7	3.0	2.8	3.1	3.0	3.9	2.5	2.9	3.1	3.3	3.2
2	2.8	1.7	2.8	3.1	1.9	2.5	4.1	1.5	1.9	2.9	1.8	1.3
4	3.8	2.4	2.9	3.5	3.0	2.8	7.2	2.0	2.1	5.1	1.8	1.4
6	-	3.8	2.6	-	3.1	3.2	-	2.3	2.1	-	2.0	1.4
8	-	3.0	2.9	-	1.8	3.8	-	2.0	2.3	-	2.6	1.5
16	-	5.9	4.8	-	4.0	3.8	-	3.0	3.5	-	2.8	1.8
18	-	7.5	5.3	-	4.9	4.3	-	3.8	4.2	-	3.9	2.2
24	-	10.8	8.1	-	7.3	6.0	-	5.5	5.9	-	4.7	3.0
32	-	-	12.0	-	-	9.5	-	-	6.1	-	-	3.2
64	-	-	106.4	-	-	53.5	-	-	46.0	-	-	25.4

Nadaljnjo analizo učinkovitosti predlagane metode smo izvedli s primerjavo izvajalnih časov metode UW [59], pri čemer je bila uporabljena njena odprtokodna implementacija, prenesena s spletnega portala GitHub[1]. Čeprav je njena implementacija omejena na podatkovni tip `char`, podpira tako vektorsko procesiranje z ukazi AVX2 kakor tudi večnitno izvajanje z uporabo OpenMP. Na enak način kot pri knjižnici OpenCV, smo primerjali računske čase velikega strukturnega elementa in iterativnega izračuna verige filtrov. Rezultati obeh so podani na sliki 5.10, faktorji pohitritve predlagane metode pa v tabeli 5.18. Tudi v tem primeru je predlagana metoda bistveno presegla računske

[1]Izvorna koda `UW`: `https://github.com/ReneBrals/uw-morpho`.



Slika 5.8: Primerjava enonitnih izvajalnih časov predlagane metode s knjižnico OpenCV, kjer je bil testiran tako direktni kot iterativni izračun.



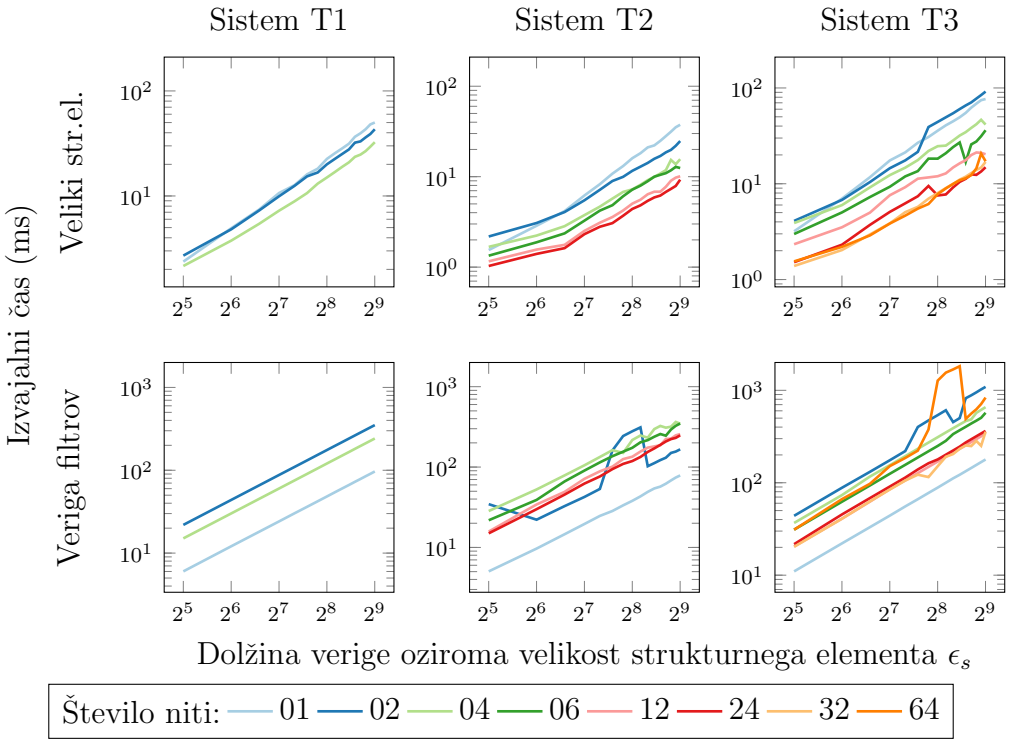
Slika 5.9: Izvajalni časi knjižnice SMIL glede na uporabljen podatkovni tip, testni sistem, dolžino verige ter število niti.

čase metode UW pri izračunu verige filtrov. Proti pričakovanjem pa se je obnesla boljše tudi v primerjavi z neposrednim izračunom velikega strukturnega elementa. Ti rezultati so še posebej presenetljivi zaradi dejstva, da metoda UW bolje izkorišča večnitno procesiranje. Slednje je razvidno iz linearnega skrajševanja izvajalnih časov glede na število uporabljenih niti v primerjavi z rezultati knjižnice SMIL.

Tabela 5.18: Faktor pohitritve predlagane metode v primerjavi z metodo UW ob izvedbi filtra z velikim strukturnim elementom velikosti 1025×1025 in verigi filtrov dolžine 512.

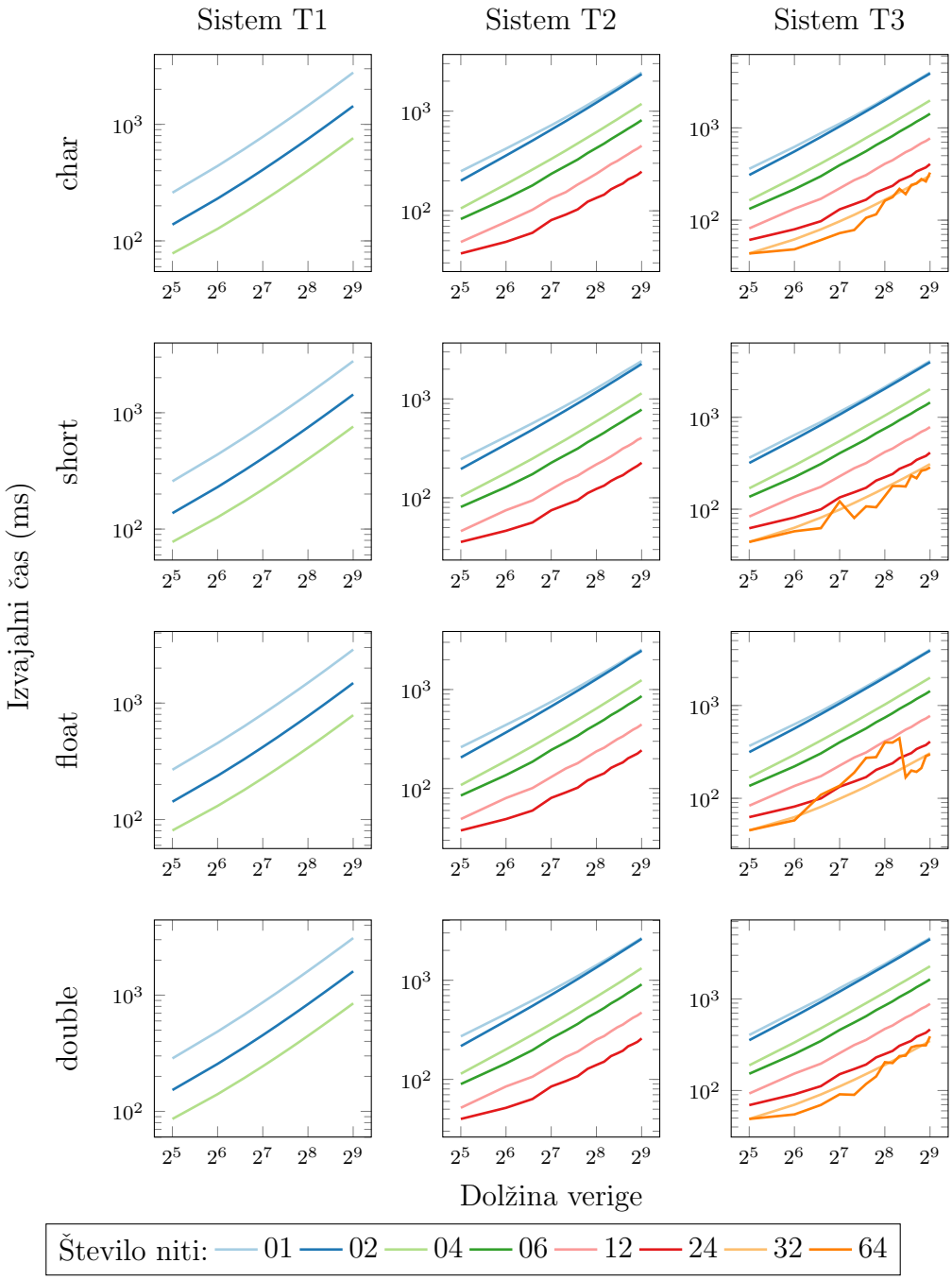
Število niti	Veliki str. el.			Veriga filtrov		
	T1	T2	T3	T1	T2	T3
1	2,1	2,1	1,9	4,1	4,4	4,5
2	3,3	1,5	3,2	26,8	9,8	38,5
4	4,0	1,8	2,7	29,8	39,9	43,0
6	-	2,0	2,8	-	57,1	43,9
8	-	1,8	3,0	-	44,8	32,6
12	-	2,1	1,9	-	53,6	33,0
16	-	2,0	2,4	-	52,5	46,3
18	-	2,4	2,6	-	64,9	46,7
24	-	2,6	2,6	-	70,9	63,8
32	-	-	3,9	-	-	83,0
34	-	-	3,7	-	-	88,7
64	-	-	6,6	-	-	320,8

Zadnjo primerjavo smo izvedli z lastno implementacijo metode DD [29], pri kateri nismo uporabili ukazov SIMD, saj njihova preslikava na uporabo vrste ni bila očitna. Implementirani filtri so bili integrirani v predstavljeno ogrodje in tako smo lahko v primeru izračuna verige filtrov uporabili vzporedno procesiranje, kjer smo z metodo DD hkrati obdelovali več filtrov. Posledično je bil direkten izračun velikega strukturnega elementa omejen na zgolj eno nit. Ker časovna zahtevnost metode ni odvisna od velikosti strukturnega elementa, ta omejitev ne vpliva na izmerjene rezultate. Doseženi večnitni izvajalni časi

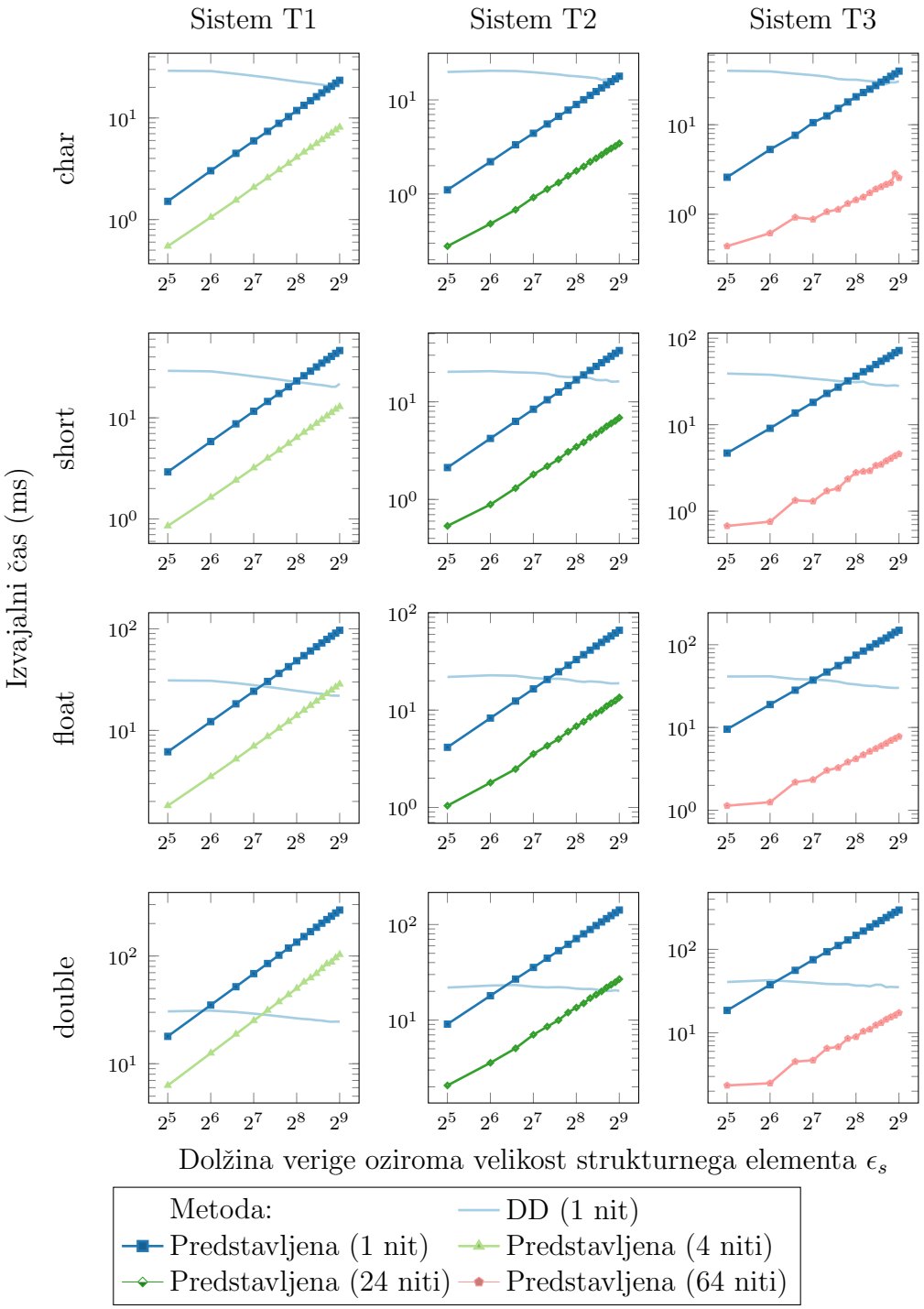


Slika 5.10: Izvajalni časi metode UW v primeru procesiranja verige filtrov in neposrednega izračuna velikega strukturnega elementa.

metode DD so podani na sliki 5.11, medtem ko slika 5.12 podaja primerjavo v učinkovitosti procesiranja velikega strukturnega elementa med predlagano metodo in metodo DD. Kot je razvidno iz podanih izvajalnih časov, je metoda DD dosegla dobro skaliranje ob vzporednem procesiranju verige filtrov v vseh konfiguracijah. Edina izjema je bil sistem T3, kjer med uporabo 32 in 64 niti ni bilo bistvene razlike. Rezultati so prav tako pokazali neodvisnost izvajalnih časov metode DD od uporabljenega podatkovnega tipa. Slednje lahko pripišemo skalarnemu procesiranju, kjer je hitrost obdelave enaka neglede na podatkovni tip. Pri izračunu velikega strukturnega elementa so bili doseženi časi, kot pričakovano, neodvisni od velikosti strukturnega elementa. Trend padanja računskega časa z večanjem okna je posledica najdenega minimuma, ki skoraj nikoli ni odstranjen iz vrste. Presenetljivo pa je bilo dejstvo, da je bila enonitna izvedba predlagane metode v primeru podatkovnega tipa `char` skoraj vedno hitrejša kot metoda DD. Iz doseženih enonitnih faktorjev pohitritve v tabeli 5.19 smo zato pričakovali boljše računske čase samo do okna velikosti $\approx 251 \times 251$ oziroma verige dolžine 125 filtrov ϵ_1. Vseeno pa rezultati v tabeli 5.20 kažejo, da je bila predstavljena metoda hitrejša do verige dolžine 448 filtrov ϵ_1 oziroma okna velikosti 897×897. Seveda ti faktorji padajo z večanjem podatkovnega tipa zaradi omejene velikosti vektorskih registrov. Tudi v tem primeru je očitno, da je predlagana metoda bistveno učinkovitejša in tako primernejša za izvedbo postopka geodetske rekonstrukcije od sorodnih metod, implementiranih na CPE. V naslednjem podpoglavju pa predlagano metodo primerjamo še z iterativno rešitvijo, implementirano na GPGPU.



Slika 5.11: Večnitni izvajalni časi metode DD ob povečujoči se dolžini verige pri različnih podatkovnih tipih.



Slika 5.12: Primerjava enonitnih in večnitnih izvajalnih časov predlagane metode z metodo DD, kjer slednja izvede neposreden izračun velikega strukturnega elementa.

Tabela 5.19: Doseženi faktorji pohitritve predlagane metode v primerjavi z metodo DD ob izračunu verige 512 ϵ_1 filtrov.

Število niti	char			short			float			double		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
1	118,3	137,0	100,1	60,0	72,1	56,8	29,8	38,3	26,8	11,6	18,8	15,6
2	109,6	139,3	137,5	58,3	70,9	69,9	26,5	37,6	32,0	8,2	17,8	11,4
4	94,0	136,1	129,5	58,9	71,2	71,7	27,5	38,1	31,9	8,2	17,9	11,3
6	-	133,3	109,6	-	71,6	70,7	-	38,3	30,9	-	18,0	11,5
8	-	87,6	90,0	-	43,3	72,1	-	23,0	31,9	-	12,2	11,3
12	-	93,7	73,3	-	43,0	56,4	-	23,7	31,3	-	12,5	11,6
16	-	80,8	72,6	-	38,1	43,1	-	20,3	32,1	-	10,8	11,4
18	-	81,3	72,2	-	38,0	43,6	-	20,7	31,5	-	10,9	11,7
24	-	70,5	71,1	-	32,8	40,4	-	18,0	30,4	-	9,7	11,8
32	-	-	70,7	-	-	37,7	-	-	25,3	-	-	11,4
34	-	-	71,0	-	-	39,9	-	-	25,8	-	-	11,1
64	-	-	126,2	-	-	62,0	-	-	38,1	-	-	22,4

Tabela 5.20: Enonitni faktorji pohitritve predlagane metode v primerjavi z metodo DD, kjer slednja opravi neposreden izračun velikega strukturnega elementa.

Dolžina verige	char			short			float			double		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
32	19,3	17,9	15,5	10,0	9,6	8,3	5,0	5,3	4,3	1,7	2,4	2,2
64	9,6	9,2	7,5	4,9	4,9	4,2	2,5	2,8	2,2	0,9	1,3	1,1
96	6,1	6,1	4,9	3,1	3,2	2,6	1,6	1,8	1,4	0,6	0,9	0,7
128	4,4	4,4	3,4	2,2	2,4	1,9	1,1	1,3	1,0	0,4	0,6	0,5
160	3,4	3,4	2,7	1,7	1,8	1,4	0,9	1,0	0,8	0,3	0,5	0,4
192	2,7	2,8	2,1	1,4	1,4	1,2	0,7	0,8	0,6	0,3	0,4	0,3
224	2,3	2,3	1,8	1,1	1,2	1,0	0,6	0,7	0,5	0,2	0,4	0,3
256	1,9	2,0	1,6	1,0	1,1	0,9	0,5	0,6	0,4	0,2	0,3	0,3
288	1,7	1,7	1,4	0,9	0,9	0,8	0,4	0,5	0,4	0,2	0,3	0,2
320	1,5	1,5	1,2	0,8	0,8	0,7	0,4	0,5	0,3	0,2	0,2	0,2
352	1,3	1,4	1,1	0,7	0,7	0,6	0,3	0,4	0,3	0,1	0,2	0,2
384	1,2	1,2	1,0	0,6	0,7	0,5	0,3	0,4	0,3	0,1	0,2	0,2
416	1,1	1,1	0,9	0,6	0,6	0,5	0,3	0,4	0,3	0,1	0,2	0,1
448	1,0	1,0	0,9	0,5	0,5	0,5	0,3	0,3	0,2	0,1	0,2	0,1
480	0,9	1,0	0,8	0,5	0,5	0,4	0,2	0,3	0,2	0,1	0,2	0,1
512	0,9	0,9	0,8	0,5	0,5	0,4	0,2	0,3	0,2	0,1	0,1	0,1

5.5 Primerjava z implementacijo na GPGPU

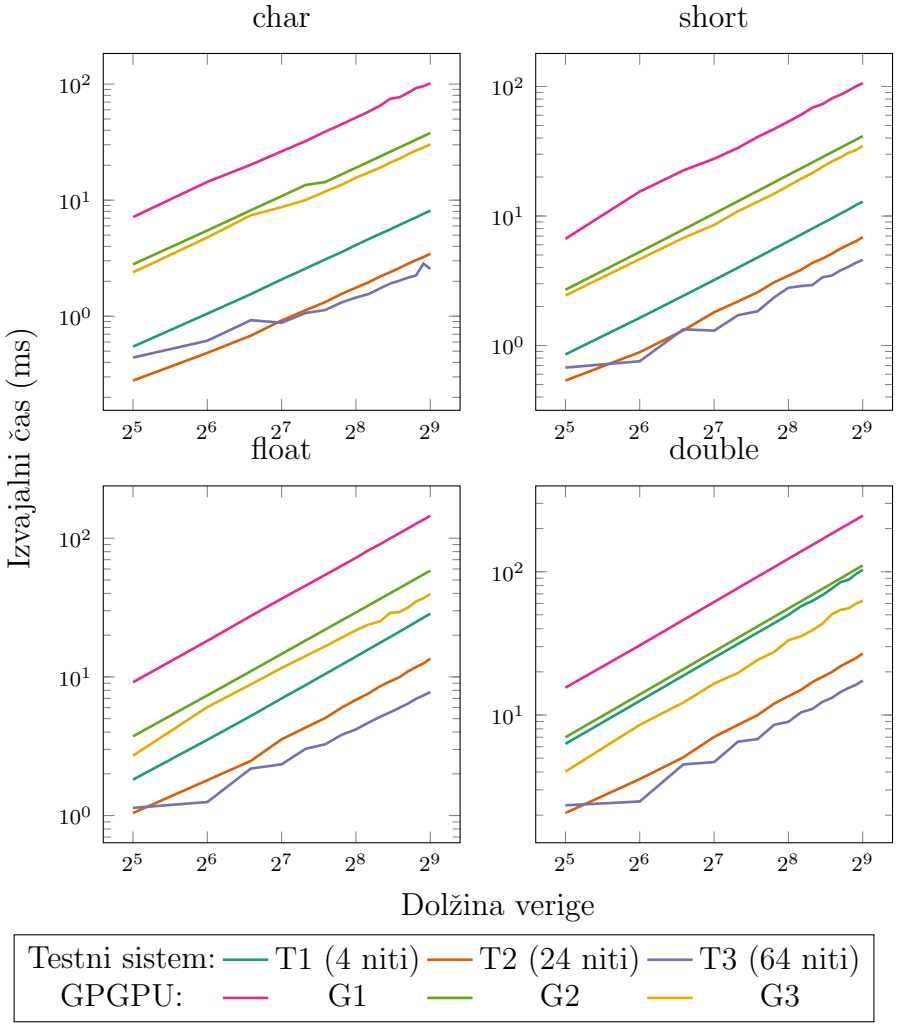
Tabela 5.21: Specifikacije uporabljenih splošnonamenskih grafičnih procesnih enot.

Testni GPGPU	Proizvajalec	Model	Delovna frekvenca	Število jeder	Pasovna širina pomnilnika	Teoretična rač. zmogljivost
G1	Nvidia	GTX 750 Ti	1 GHz	640	86 GB/s	1.4 TFLOPS
G2	Nvidia	GTX 1650 Super	1.5 GHz	1280	192 GB/s	4.4 TFLOPS
G3	Nvidia	Titan X Pascal	1.4 GHz	3584	480 GB/s	11 TFLOPS

Za zagotovitev reprezentativnosti predstavljenih rezultatov smo meritve izvedli nad grafičnimi procesnimi enotami (GPGPU) nizkega (G1), srednjega (G2) in visokega (G3) cenovnega razreda. Specifikacije posamezne GPGPU so podane v tabeli 5.21. Za izvedbo primerjalnih testov smo s tehnologijo CUDA implementirali morfološko širjenje in krčenje, kjer smo ϵ_1 realizirali s kompozicijo filtrov ϵ_1^x in ϵ_1^y. Implementacija je bila izpeljana iz vzorca kode, ki opravlja konvolucijo [89]. Preostale osnovne morfološke filtre smo nato razširili v elementarno geodetsko krčenje, širjenje in transformacijo domnevnih razdalj. Izračune na GPGPU smo izvedli iterativno, saj implementacija pretokovnega procesiranja na njih ni očitna. Ker se je procesiranje na GPGPU izvajalo z uporabo vseh jeder, smo primerjavo s CPE omejili zgolj na rezultate, dosežene z največjim možnim številom niti glede na posamezen testni sistem, torej 4, 24 in 64 niti za sisteme T1, T2 in T3. Kot testno sliko smo tudi v tem primeru uporabili sliko *Sand*. Izvajalni časi verige filtrov, izmerjeni na GPGPU in na testnih sistemih, so podani na sliki 5.13, relativne pohitritve izvedbe na v primerjavi z GPGPU pa so podane v tabeli 5.22, pri čemer referenčno vrednost predstavljajo izvajalni časi G1. Iz meritev izvajalnih časov je očitno, da se je sistem T1 izkazal za učinkovitejšega od vseh testiranih GPGPU v vseh primerih, razen v primeru podatkovnega tipa `double`, kjer je boljše rezultate dosegla najzmogljivejša GPGPU G3. Splošno nizke hitrosti izračuna na GPGPU lahko interpretiramo kot posledico uporabe majhnega strukturnega elementa in strogo zaporednega izračuna. Računski časi pri uporabi podat-

kovnih tipov `char` in `short` so bili pretežno nespremenjeni, kar je bilo tudi pričakovano. Nasprotno pa so se računski časi povečali v povprečju za $\approx$ 1,5- in $\approx$ 2,5-krat v primeru uporabe podatkovnih tipov `float` in `double`. Slednje nakazuje na prisotnost ozkega grla v prenosu podatkov do jeder GPGPU.

Na tej osnovi smo analizirali še izvajalne čase geodetskih operatorjev, podane v tabeli 5.23. Pri tem smo ponovno primerjali relativne pohitritve izvajalnih časov glede na referenčni GPGPU, to je G1, faktorji pohitritve pa so podani v tabeli 5.24. Po pričakovanjih je predlagana metoda tudi v večini teh primerov dosegla boljše izvajalne čase kot pa uporaba GPGPU. Povprečni faktorji pohitritve grafičnih kartic G2, G3 in sistema T1 so ostali podobni pohitritvam, doseženim pri izračunu verige filtrov s tradicionalnimi morfološkimi filtri. Nasprotno pa smo bistvene pohitritve dosegli na sistemih T2 in T3, kjer smo zabeležili faktorje $\approx$ 1,8- in $\approx$ 1,3-krat. To lahko pripišemo vplivu pogojnega posodabljanja slik, ki ga implementira predstavljena metoda. Skozi teste se je namreč pokazalo, da je bilo posodabljanje slik med izvedbo geodetskih operatorjev relativno redko (glej tabelo 5.9), pri čemer pa je implementacija na GPGPU zahtevala posodabljanje izhodne slike. To tudi potrjujejo višji faktorji pohitritve pri transformaciji domnevnih razdalj, kjer je predlagana metoda tipično posodabljala samo eno sliko, medtem ko je implementacija na GPGPU zahtevala posodabljanje vseh treh slik.



Slika 5.13: Primerjava izvajalnih časov GPGPU in lastne metode ob povečujoči se dolžini verige filtrov ter z različnimi podatkovnimi tipi.

Tabela 5.22: Doseženi faktorji pohitritve izračuna verige filtrov glede na izvajalne čase grafične kartice G1.

Dolžina verige	char			short			float			double		
	G1	G2	G3	G1	G2	G3	G1	G2	G3	G1	G2	G3
32	1,0	2,6	3,0	1,0	2,5	2,7	1,0	2,5	3,4	1,0	2,2	3,9
64	1,0	2,6	3,0	1,0	2,9	3,3	1,0	2,5	3,0	1,0	2,2	3,6
96	1,0	2,5	2,7	1,0	2,9	3,3	1,0	2,5	3,1	1,0	2,2	3,8
128	1,0	2,4	3,0	1,0	2,7	3,2	1,0	2,5	3,2	1,0	2,2	3,7
160	1,0	2,4	3,2	1,0	2,6	3,1	1,0	2,5	3,2	1,0	2,2	3,9
192	1,0	2,7	3,3	1,0	2,6	3,2	1,0	2,5	3,3	1,0	2,2	3,8
224	1,0	2,7	3,3	1,0	2,6	3,2	1,0	2,5	3,3	1,0	2,2	3,9
256	1,0	2,7	3,3	1,0	2,6	3,1	1,0	2,5	3,3	1,0	2,2	3,7
288	1,0	2,7	3,3	1,0	2,6	3,1	1,0	2,5	3,4	1,0	2,2	3,9
320	1,0	2,7	3,4	1,0	2,6	3,2	1,0	2,5	3,6	1,0	2,2	3,9
352	1,0	2,9	3,5	1,0	2,6	3,1	1,0	2,5	3,4	1,0	2,2	3,9
384	1,0	2,7	3,4	1,0	2,6	3,1	1,0	2,5	3,7	1,0	2,2	3,7
416	1,0	2,7	3,4	1,0	2,6	3,0	1,0	2,5	3,7	1,0	2,2	3,7
448	1,0	2,8	3,4	1,0	2,6	3,0	1,0	2,5	3,6	1,0	2,2	3,9
480	1,0	2,7	3,4	1,0	2,6	3,1	1,0	2,5	3,7	1,0	2,2	3,9
512	1,0	2,7	3,4	1,0	2,6	3,1	1,0	2,5	3,7	1,0	2,2	3,9

(a) Relativne pohitritve grafičnih kartic G1, G2 in G3.

Dolžina verige	char			short			float			double		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
32	13,1	25,6	16,2	7,8	12,4	9,9	5,0	8,8	8,1	2,5	4,3	6,6
64	13,7	29,8	23,4	9,4	17,4	20,4	5,2	10,1	14,5	2,4	6,0	12,3
96	13,0	29,7	21,9	9,3	17,2	16,9	5,3	11,1	12,6	2,4	6,6	10,2
128	12,7	28,7	30,0	8,6	15,3	21,3	5,2	10,3	15,6	2,4	7,2	13,1
160	12,5	28,7	30,3	8,4	15,3	19,6	5,2	10,6	15,0	2,5	7,7	11,8
192	12,6	29,4	34,4	8,5	15,9	22,2	5,2	10,8	16,7	2,5	7,7	13,6
224	12,6	28,9	34,2	8,4	15,3	20,0	5,2	10,5	16,6	2,5	8,0	12,6
256	12,6	29,2	35,6	8,4	15,5	19,2	5,2	10,6	17,3	2,5	8,2	13,7
288	12,5	29,5	37,3	8,4	15,8	21,0	5,2	10,8	17,5	2,4	8,2	13,3
320	12,7	29,6	37,5	8,6	15,8	23,4	5,1	10,6	17,6	2,5	8,3	14,0
352	13,4	31,2	39,1	8,3	15,6	21,7	5,1	10,7	17,9	2,5	8,5	13,7
384	12,6	29,6	38,0	8,4	15,8	23,4	5,1	10,9	18,2	2,4	8,4	14,0
416	12,7	29,7	39,2	8,3	15,4	22,7	5,1	10,7	18,3	2,4	8,5	13,8
448	13,0	30,4	41,2	8,3	15,5	22,9	5,1	10,8	18,3	2,5	8,6	13,9
480	12,6	29,7	33,7	8,2	15,7	23,0	5,1	10,9	18,5	2,4	8,6	14,2
512	12,6	29,4	39,8	8,2	15,4	23,1	5,1	10,8	18,7	2,4	9,2	14,2

(b) Relativne pohitritve sistemov T1 (4 niti), T2 (24 niti) in T3 (64 niti).

Tabela 5.23: Izvajalni časi (v milisekundah) geodetskih operatorjev na grafičnih karticah in testnih sistemih.

Operator	char			short			float			double		
	G1	G2	G3	G1	G2	G3	G1	G2	G3	G1	G2	G3
SPL	200	74	59,3	225	84,1	69,9	328,7	131,6	82,1	563,6	252,8	140
OO	155,9	57,9	45,7	169,8	65,7	54,9	254,8	102,8	61,8	440,5	197,4	113,6
γ_{rek}	378,2	137,2	115,4	427,1	153,3	130,6	571,5	232,2	159,4	1056,1	445,3	259,1
RK	176	61,9	49,5	194,3	69,9	58,8	287,2	109,5	71,5	469,1	210,5	122,8
IK	177,8	62,1	51,4	195,5	70	60,8	271,8	109,5	73,7	469,5	210,6	122,6
TDR	322,6	112	84,1	397,4	145,4	112	593,4	250,5	137,6	1096	488	248,8
	char			short			float			double		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
SPL	14,4	3,7	3,9	25,6	6,6	6,7	81,9	13,8	13,9	223,7	36,9	28
OO	19,2	5	5,1	35,5	8,8	9,1	107,3	24,7	18,7	288,8	53	37,7
γ_{rek}	35,3	9,7	8,2	58,4	17,4	16,2	150,2	36	29,2	386,2	91	59,7
RK	17	4,3	4,1	28,1	6,9	7,6	75,9	15,1	15,2	188,8	40,8	28,6
IK	17	4,6	4,6	28,2	7,2	8,3	77,4	21,9	17,6	190,7	46,8	33,1
TDR	20,5	5,6	5,9	34,5	9,9	10,9	79,9	18,3	21,1	199,6	37,2	42

Tabela 5.24: Faktorji pohitritve izračuna geodetskih operatorjev glede na izvajalne čase grafične kartice G1.

Operator	char			short			float			double		
	G1	G2	G3	G1	G2	G3	G1	G2	G3	G1	G2	G3
SPL	1	2,7	3,4	1	2,7	3,2	1	2,5	4	1	2,2	4
OO	1	2,7	3,4	1	2,6	3,1	1	2,5	4,1	1	2,2	3,9
γ_{rek}	1	2,8	3,3	1	2,8	3,3	1	2,5	3,6	1	2,4	4,1
RK	1	2,8	3,6	1	2,8	3,3	1	2,6	4	1	2,2	3,8
IK	1	2,9	3,5	1	2,8	3,2	1	2,5	3,7	1	2,2	3,8
TDR	1	2,9	3,8	1	2,7	3,5	1	2,4	4,3	1	2,2	4,4
	char			short			float			double		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
SPL	13,9	54,1	51,3	8,8	34,1	33,6	4	23,8	23,6	2,5	15,3	20,1
OO	8,1	31,2	30,6	4,8	19,3	18,7	2,4	10,3	13,6	1,5	8,3	11,7
γ_{rek}	10,7	39	46,1	7,3	24,5	26,4	3,8	15,9	19,6	2,7	11,6	17,7
RK	10,4	40,9	42,9	6,9	28,2	25,6	3,8	19	18,9	2,5	11,5	16,4
IK	10,5	38,7	38,7	6,9	27,2	23,6	3,5	12,4	15,4	2,5	10	14,2
TDR	15,7	57,6	54,7	11,5	40,1	36,5	7,4	32,4	28,1	5,5	29,5	26,1

Poglavje 6

Zaključek

V doktorski disertacijo opišemo problematiko učinkovitega izračuna geodetskih filtrov, ki temeljijo na postopku morfološke rekonstrukcije. Slednji je v svoji zasnovi iterativni postopek filtriranja, ki ga izvajamo z morfološkim krčenjem ali širjenjem s strukturnim elementom velikosti 3×3. Izračun geodetskih filtrov tako predstavlja izračun dolge verige zaporedno odvisnih elementarnih filtrov. V pregledu stanja tehnike zato predstavimo znane metode učinkovitega izračuna morfološkega krčenja in širjenja. Te se osredotočajo na zmanjšanje števila potrebnih primerjav na osnovi razgradnje strukturnega elementa, ki ga dosežejo z dekompozicijo večdimenzionalnega strukturnega elementa na manjše večdimenzionalne strukturne elemente ali pa na enodimenzionalne strukturne elemente poljubne velikosti. Ne glede na velikost uporabljenega strukturnega elementa lahko slednjega izračunamo v konstantnem času na piksel. Vseeno pa je pri tem očitno, da je učinkovitost sorodnih rešitev odvisna od računalniške arhitekture, na kateri jo implementiramo. Tako na ukazno pretokovnih računalniških arhitekturah, kot sta CPE in GPGPU, zasledimo predvsem implementacije metod, ki lahko učinkovito izračunajo filtre s poljubno velikimi strukturnimi elementi. V odprtokodnih knjižnicah običajno zasledimo optimizirane implementacije morfološkega krčenja in širjenja z manjšimi struktur-

nimi elementi, ki realizirajo vzporeden izračun posameznega filtra. Nasprotno pa podatkovno pretokovne arhitekture, tipično realizirane na programirljivih čipih FPGA, opravijo hkraten izračun več filtrov z uporabo prostorskega paralelizma. Ker čipi FPGA vsebujejo končno število programirljivih logičnih elementov, so implementacije filtrov omejene v njihovi kompleksnosti. V novejših arhitekturah zato pogosto zasledimo hibridne pristope, ki vsebujejo več procesnih cevovodov, tako za učinkovit izračun filtrov z velikimi kot z majhnimi strukturnimi elementi. To jim omogoča tudi učinkovit izračun geodetskih filtrov, saj z enim pretokom slike skozi procesni cevovod opravijo večje število filtriranj. Metode, ki bi realizirala vzporeden izračun več zaporedno odvisnih filtrov, pa na ukazno pretokovnih arhitekturah ne poznamo, saj ta zahteva strogo iterativen izračun. Ta se izkaže za neučinkovitega pri izračunu dolgih verig filtrov.

V okviru doktorske naloge podrobneje predstavimo teoretična izhodišča matematične morfologije in izpeljanih geodetskih operatorjev, ki jih sestavljajo dolge verige filtrov. Na tej osnovi smo zasnovali učinkovit pristop za njihov izračun, ki emulira podatkovno pretokovni način procesiranja na ukazno pretokovni arhitekturi. To omogoča enostaven vzporeden in hkraten izračun več filtrov, pri čemer je stopnja vzporednega izračuna omejena zgolj s številom jeder uporabljenega CPE. Filtri morfološkega krčenja in širjenja s strukturnim elementom velikosti 3×3 so implementirani z vektorskim procesiranjem, ki uporablja ukazni nabor AVX2. Vseeno pa hkratna uporaba vektorskega procesiranja in emulacije podatkovno pretokovnega procesiranja zahteva veliko količino pasovne širine med jedri. Slednjo smo zato lokalizirali znotraj predpomnilnika CPE, ki na osnovi pregleda topologije predpomnilnika smiselno pripne niti na posamezna jedra. Tako minimiziramo pot medjedrnega prenosa podatkov skozi predpomnilnik in posledično omogočamo višjo možno hitrost izvedbe.

6.1 Potrditev zastavljenih hipotez

Cilj doktorske disertacije je bil razvoj algoritma za učinkovit izračun verig elementarnih filtrov na centralni procesni enoti. V uvodu tega dela smo zato podali naslednjo tezo:

Z izkoriščanjem paralelizma ukaznega nivoja, vektorskega procesiranja in večjedrnega izvajanja je mogoče doseči večjo učinkovitost izvedbe dolgih verig elementarnih morfoloških filtrov v primerjavi z obstoječimi algoritmi.

Na osnovi te teze smo postavili sledeče hipoteze:

Hipoteza 1. *Izračun verig elementarnih morfoloških filtrov po metodologiji, ki temelji na ukaznem naboru AVX2, je vsaj štirikrat hitrejši od trenutno poznanih metod, implementiranih na centralnih procesnih enotah.*

Hipotezo 1 smo potrdili na podlagi rezultatov primerjave izvajalnih časov predlagane metode s trenutnim stanjem tehnike, predstavljenim v podpoglavju 5.4. V primerjavi s knjižnico OpenCV dosežemo do $\approx$ 4,4-kratno pohitritev izračuna verig elementarnih morfoloških filtrov (glej sliko 5.8), medtem ko v primerjavi s knjižnico SMIL dosežene pohitritve prerastejo tudi faktor 100 (glej tabelo 5.17). Podobne pohitritve smo dosegli tudi v primerjavi z metodo DD (glej tabelo 5.19). V primerjavi z metodo UW pa so pohitritve prerasle celo faktor 300 (glej tabelo 5.18).

Hipoteza 2. *Izračun verig elementarnih morfoloških filtrov je možno učinkoviteje izvesti na centralni procesni enoti kot na splošnonamenskih grafičnih procesnih enotah z računsko zmogljivostjo do 10 TFLOPS.*

Kot izhaja iz tabel 5.22b in 5.23, je najšibkejši sistem T1 s samo štirimi

jedri skoraj vedno dosegel višje faktorje pohitritve kot GPGPU G1, G2 in G3. Sistema T2 in T3 pa sta v vseh primerih dosegla višje faktorje pohitritve, tudi v primerjavi z 11 TFLOPS grafično kartico G3, kar potrjuje hipotezo 2.

Hipoteza 3. *Zaradi zakasnitev, ki so posledica medjedrne komunikacije, dosežemo v času izvajanja optimalen izračun verige morfoloških filtrov z vzporedno obdelavo več verig filtrov in omejitvijo števila uporabljenih jeder na posamezno verigo.*

Rezultati optimizacije odzivnosti in prepustnosti, predstavljeni v podpoglavju 5.3, potrjujejo hipotezo 3. Iz rezultatov, podanih v tabeli 5.15, izhaja, da najvišjo prepustnost dosežemo, ko uporabimo več vzporednih procesnih cevovodov in omejimo število niti na vsak cevovod. Optimalnega števila cevovodov ne moremo določiti vnaprej, saj je to odvisno od specifične arhitekture CPE, kar zahteva prilagoditev v času zagona algoritma. Nasprotno pa največjo odzivnost vedno dosežemo z uporabo enega procesnega cevovoda, ki uporablja vsa jedra CPE (glej tabelo 5.16).

V doktorski disertaciji smo tako potrdili vse postavljene hipoteze in posledično tudi zastavljeno tezo.

6.2 Izvirni prispevki k znanosti

Doktorska disertacija predstavi algoritem za učinkovit izračun verig elementarnih morfoloških filtrov na CPE. Predstavljena metoda preseže učinkovitost implementacij stanja tehnike tako na CPE kakor tudi njihov iterativni izračun na GPGPU. Izvirne prispevke doktorske disertacije k znanosti povzamemo z naslednjimi točkami:

- **računsko učinkovit postopek za elementarne morfološke filtre**, kjer so upoštevane značilnosti modernih CPE, kot sta na primer večnivojski predpomnilnik in vektorsko procesiranje,
- **vzporedna obdelava verige elementarnih filtrov na CPE s tehniko podatkovno pretokovnega procesiranja**, kjer s pregledom topologije predpomnilnika poljubnega CPE zagotovimo optimalen prenos podatkov med jedri,
- **prilagoditev izvajalnih parametrov** za dosego največje možne odzivnosti ali prepustnosti obdelave slik,
- **analiza učinkovitosti izračuna verig elementarnih filtrov**, kjer se predstavljeni algoritem izkaže kot najboljši, in
- **objava izvirnega znanstvenega članka** [90].

Literatura

[1] J Bryan Blair, David L Rabine, and Michelle A Hofton. The laser vegetation imaging sensor: a medium-altitude, digitisation-only, airborne laser altimeter for mapping vegetation and topography. *ISPRS Journal of Photogrammetry and Remote Sensing*, 54(2-3):115–122, 1999.

[2] Stephen E Reutebuch, Hans E Andersen, and Robert J McGaughey. Light detection and ranging (LiDAR): an emerging tool for multiple resource inventory. *Journal of forestry*, 103(6):286–292, 2005.

[3] Michele Calì and Rita Ambu. Advanced 3D photogrammetric surface reconstruction of extensive objects by UAV camera image acquisition. *Sensors*, 18(9):2815, 2018.

[4] Markus Schartel, Ralf Burr, Winfried Mayer, et al. UAV-based ground penetrating synthetic aperture radar. In *2018 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, pages 1–4. IEEE, 2018.

[5] Randy Showstack. Sentinel satellites initiate new era in Earth observation. *Eos, Transactions American Geophysical Union*, 95(26):239–240, 2014.

[6] Alan S Belward and Jon O Skøien. Who launched what, when and why; trends in global land-cover observation capacity from civilian Earth observation satellites. *ISPRS Journal of Photogrammetry and Remote Sensing*, 103:115–128, 2015.

[7] Scott A Huettel, Allen W Song, Gregory McCarthy, et al. *Functional magnetic resonance imaging*, volume 1. Sinauer Associates Sunderland, MA, 2004.

[8] Stelios Halkiotis, Taxiarchis Botsis, and Maria Rangoussi. Automatic detection of clustered microcalcifications in digital mammograms using mathematical morphology and neural networks. *Signal Processing*, 87(7):1559–1568, 2007.

[9] Emmanuel Brun, Abdelâali Hassaïne, Bernard Besserer, and Etienne Decencière. Restoration of variable area soundtracks. In *2007 IEEE International Conference on Image Processing*, volume 4, pages IV–13. IEEE, 2007.

[10] Dieu S Ly, Serge Beucher, and Michel Bilodeau. Real-time contrast medium detection in x-ray images by mathematical morphology operators. *Journal of Electronic Imaging*, 24(6):061202, 2015.

[11] Marco A Garduño-Ramón, Ivan R Terol-Villalobos, Roque A Osornio-Rios, and Luis A Morales-Hernandez. A new method for inpainting of depth maps from time-of-flight sensors based on a modified closing by reconstruction algorithm. *Journal of Visual Communication and Image Representation*, 47:36–47, 2017.

[12] Jia Shi, Yuye Wang, Degang Xu, et al. Terahertz imaging based on morphological reconstruction. *IEEE Journal of Selected Topics in Quantum Electronics*, 23(4):1–7, 2017.

[13] Philippe Salembier and Jean Serra. Flat zones filtering, connected operators, and filters by reconstruction. *IEEE Transactions on Image Processing*, 4(8):1153–1160, 1995.

[14] Martino Pesaresi and Jón A Benediktsson. A new approach for the morphological segmentation of high-resolution satellite imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 39(2):309–320, 2001.

[15] Jón A Benediktsson, Jón A za Palmason, and Johannes R Sveinsson. Classification of hyperspectral data from urban areas based on extended morphological profiles. *IEEE Transactions on Geoscience and Remote Sensing*, 43(3):480–491, 2005.

[16] Keqi Zhang. Identification of gaps in mangrove forests with airborne LiDAR. *Remote Sensing of Environment*, 112(5):2309–2325, 2008.

[17] Hüseyin G Akçay and Selim Aksoy. Automatic detection of geospatial objects using multiple hierarchical segmentations. *IEEE Transactions on Geoscience and Remote Sensing*, 46(7):2097–2111, 2008.

[18] Sébastien Lefèvre, Jonathan Weber, and David Sheeren. Automatic building extraction in VHR images using advanced morphological operators. In *2007 Urban Remote Sensing Joint Event*, pages 1–5. IEEE, 2007.

[19] Domen Mongus, Niko Lukač, and Borut Žalik. Ground and building extraction from LiDAR data based on differential morphological profiles and locally fitted surfaces. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93:145–156, 2014.

[20] Domen Mongus and Borut Žalik. Computationally efficient method for the generation of a digital terrain model from airborne LiDAR data using connected operators. *IEEE journal of selected topics in applied Earth observations and remote sensing*, 7(1):340–351, 2013.

[21] Yong Li, Bin Yong, Peter van Oosterom, et al. Airborne LiDAR data filtering based on geodesic transformations of mathematical morphology. *Remote Sensing*, 9(11):1104, 2017.

[22] Agnus Vincent, Ronse Christian, and Heitz Fabrice. Spatio-temporal segmentation using 3D morphological tools. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 3, pages 877–880. IEEE, 2000.

[23] Pascal Le Quéré, Patrick Maupin, Robert Desjardins, et al. Change detection from remotely sensed multi-temporal images using morphological operators. In *IEEE International Geoscience and Remote Sensing Symposium Proceedings. Remote Sensing-A Scientific Vision for Sustainable Development*, volume 1, pages 252–254. IEEE, 1997.

[24] Monidipa Das and Soumya K Ghosh. Modeling spatio-temporal change pattern using mathematical morphology. In *Proceedings of the 3rd IKDD Conference on Data Science, 2016*, pages 1–10, 2016.

[25] Monidipa Das and Soumya K Ghosh. Spatio-temporal pattern analysis for regional climate change using mathematical morphology. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 2, 2015.

[26] Josef Pecht. Speeding-up successive minkowski operations with bit-plane computers. *Pattern Recognition Letters*, 3(2):113–117, 1985.

[27] Jianning Xu. Decomposition of convex polygonal morphological structuring elements into neighborhood subsets. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):153–162, 1991.

[28] Marcel van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recognition Letters*, 13(7):517–521, 1992.

[29] Petr Dokládal and Eva Dokladalova. Computationally efficient, one-pass algorithm for morphological filters. *Journal of Visual Communication and Image Representation*, 22(5):411–420, 2011.

[30] Olivier Déforges, Nicolas Normand, and Marie Babel. Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture. *Journal of Real-Time Image Processing*, 8(2):143–152, 2013.

[31] Jan Bartovskỳ, Petr Dokládal, Eva Dokládalová, and Vjačeslav Georgiev. Parallel implementation of sequential morphological filters. *Journal of Real-Time Image Processing*, 9(2):315–327, 2014.

[32] Jan Bartovskỳ, Petr Dokládal, Eva Dokladalova, et al. Real-time implementation of morphological filters with polygonal structuring elements. *Journal of Real-Time Image Processing*, 10(1):175–187, 2015.

[33] David Valencia and Antonio Plaza. Efficient implementation of morphological opening and closing by reconstruction on multi-core parallel systems. In *Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing*, pages 1–4. IEEE, 2009.

[34] Mugdha A Rane. Fast morphological image processing on GPU using CUDA. *Pune: Department of Computer Engineering and Information Technology, College of Engineering*, 2013.

[35] Jun M Koay, Yoong C Chang, Shahirina M Tahir, and Sankaraiah Sreeramula. Parallel implementation of morphological operations on binary images using CUDA. In *Advances in Machine Learning and Signal Processing*, pages 163–173. Springer, 2016.

[36] Shuo Li, Hui Wang, Qiuhe Ma, and Xuan Zha. A morphological LiDAR points cloud filtering method based on GPGPU. In *GISTAM*, pages 80–84, 2016.

[37] Gary Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[38] Matthieu Faessel and Michel Bilodeau. SMIL: Simple Morphological Image Library. In *Séminaire Performance et Généricité, LRDE*, Villejuif, France, March 2013.

[39] Henk JAM. Heijmans and Christian Ronse. The algebraic basis of mathematical morphology. *Computer Vision, Graphics, and Image Processing*, 50(3):245–295, 1990.

[40] Laurent Najman and Hugues Talbot. *Mathematical morphology: from theory to applications*. John Wiley & Sons, 2013.

[41] Jean Serra. *Image analysis and mathematical morphology*. Academic Press, Inc., 1983.

[42] Stanley R Sternberg. Grayscale morphology. *Computer vision, graphics, and image processing*, 35(3):333–355, 1986.

[43] Pierre Soille. *Morphological image analysis: principles and applications*. Springer Science & Business Media, 2013.

[44] Thomas Blaschke. Object based image analysis for remote sensing. *ISPRS journal of photogrammetry and remote sensing*, 65(1):2–16, 2010.

[45] Philippe Salembier and Michael HF Wilkinson. Connected operators. *IEEE Signal Processing Magazine*, 26(6):136–157, 2009.

[46] Smithson Museum - Wall of Licence Plates, zadnji dostop Oktober 2020. https://ccsearch-dev.creativecommons.org/photos/2d1bf53f-d298-4835-af2d-5eb132cb49ba.

[47] Luc Vincent. Morphological grayscale reconstruction in image analysis: applications and efficient algorithms. *IEEE Transactions on Image Processing*, 2(2):176–201, 1993.

[48] USC-SIPI database, zadnji dostop Februar 2020. http://sipi.usc.edu/database/database.php?volume=misc.

[49] Georges Matheron. Random sets and integral geometry. 1975.

[50] Serge Beucher. Numerical residues. *Image and Vision Computing*, 25(4):405–415, 2007.

[51] Jean Serra. Introduction to mathematical morphology. *Computer vision, graphics, and image processing*, 35(3):283–305, 1986.

[52] Michael J Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 100(9):948–960, 1972.

[53] Shizuka Ishikawa, Asuka Tanaka, and Toshiaki Miyazaki. Hardware accelerator for BLAST. In *IEEE 6th International Symposium on Embedded Multicore SoCs*, pages 16–22. IEEE, 2012.

[54] Jasmin Ajanovic. PCI Express*(PCIe*) 3.0 accelerator features. *Intel Corporation*, 2008.

[55] Bulent Abali, Bart Blaner, John Reilly, et al. Data Compression Accelerator on IBM POWER9 and z15 Processors: Industrial Product. In *ACM/IEEE 47th Annual International Symposium on Computer Architecture*, pages 1–14. IEEE, 2020.

[56] Brad Smith. ARM and Intel battle over the mobile chip's future. *Computer*, 41(5):15–18, 2008.

[57] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, 2008.

[58] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Spring Joint Computer Conference*, pages 483–485, 1967.

[59] Erik R Urbach and Michael HF Wilkinson. Efficient 2-D grayscale morphological transformations with arbitrary flat structuring elements. *IEEE Transactions on Image Processing*, 17(1):1–8, 2008.

[60] Joseph Gil and Michael Werman. Computing 2-D min, median, and max filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):504–507, 1993.

[61] Xinhua Zhuang and Robert M Haralick. Morphological structuring element decomposition. *Computer Vision, Graphics, and Image Processing*, 35(3):370–382, 1986.

[62] Xinhua Zhuang. Decomposition of morphological structuring elements. *Journal of Mathematical Imaging and Vision*, 4(1):5–18, 1994.

[63] Nicolas Normand. Convex structuring element decomposition for single scan binary mathematical morphology. In *International Conference on Discrete Geometry for Computer Imagery*, pages 154–163. Springer, 2003.

[64] Christophe Clienti. Fulguro image processing library. *Source Forge*, 2008.

[65] Beucher Nicolas. MAMBA documentation. *Web documents (available at http://www.mamba-image.org)*, 2009.

[66] Marc van Droogenbroeck and Hugues Talbot. Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recognition Letters*, 17(14):1451–1460, 1996.

[67] Maxime Moreaud and Frederic Itthirad. Fast algorithm for dilation and erosion using arbitrary flat structuring element: Improvement of Urbach and Wilkinson's algorithm to GPU computing. In *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*, pages 289–294. IEEE, 2014.

[68] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[69] Pierre Soille, Edmond J Breen, and Ronald Jones. Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):562–567, 1996.

[70] Michael J Cree. Vectorised SIMD implementations of morphology algorithms. In *International Conference on Image and Vision Computing New Zealand*, pages 1–6. IEEE, 2015.

[71] Luke Domanski, Pascal Vallotton, and Dadong Wang. Parallel van Herk/Gil-Werman image morphology on GPUs using CUDA. In *GTC 2009 Conference posters*, 2009.

[72] Matthew J Thurley and Victor Danell. Fast morphological image processing open-source extensions for GPU processing with CUDA. *IEEE Journal of Selected Topics in Signal Processing*, 6(7):849–855, 2012.

[73] Joseph Yossi Gil and Ron Kimmel. Efficient dilation, erosion, opening, and closing algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(12):1606–1617, 2002.

[74] Hao Yuan and Mikhail J Atallah. Running max/min filters using 1+O(1) comparisons per sample. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(12):2544–2548, 2011.

[75] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[76] Pavel Karas, Vincent Morard, Jan Bartovskỳ, Thierry Grandpierre, Eva Dokládalová, Petr Matula, and Petr Dokládal. GPU implementation of linear morphological openings with arbitrary angle. *Journal of Real-Time Image Processing*, 10(1):27–41, 2015.

[77] Jean C Klein and Jean Serra. The texture analyser. *Journal of microscopy*, 95(2):349–356, 1972.

[78] Jean C Klein and René Peyrard. PIMM1, an image processing ASIC based on mathematical morphology. In *Proceedings of Second Annual IEEE ASIC Seminar and Exhibit*, pages 7–1. IEEE, 1989.

[79] Stephen M Steve Trimberger. Three ages of FPGAs: a retrospective on the first thirty years of FPGA technology. *IEEE Solid-State Circuits Magazine*, 10(2):16–29, 2018.

[80] Christophe Clienti, Serge Beucher, and Michel Bilodeau. A system on chip dedicated to pipeline neighborhood processing for mathematical morphology. In *2008 16th European Signal Processing Conference*, pages 1–5. IEEE, 2008.

[81] Frank YC Shih and Owen R Mitchell. Decomposition of gray-scale morphological structuring elements. *Pattern Recognition*, 24(3):195–203, 1991.

[82] Hochong Park and Jisang Yoo. Structuring element decomposition for efficient implementation of morphological filters. *IEE Proceedings-Vision, Image and Signal Processing*, 148(1):31–35, 2001.

[83] Christophe Clienti, Michel Bilodeau, and Serge Beucher. An efficient hardware architecture without line memories for morphological image processing. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 147–156. Springer, 2008.

[84] Jan Bartovskỳ, Petr Dokládal, Matthieu Faessel, et al. Morphological co-processing unit for embedded devices. *Journal of Real-Time Image Processing*, 15(4):775–786, 2018.

[85] Manuel Pöter and Jesper L Träff. Memory Models for C/C++ Programmers. *arXiv preprint arXiv:1803.04432*, 2018.

[86] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, et al. hwloc: A generic framework for managing hardware affinities in HPC applications. In *Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 180–186. IEEE, 2010.

[87] Matthias Kretz. *Extending C++ for explicit data-parallel programming via SIMD vector types.* PhD thesis, Goethe University Frankfurt am Main, 2015.

[88] Jared Hoberock. Working Draft, C++ extensions for parallelism version 2 . Technical Report 3, WG21 - The C++ Standards Committee - ISOCPP, 2019. Paper N4808.

[89] Victor Podlozhnyuk. Image convolution with CUDA. *NVIDIA Corporation white paper, June*, 2007.

[90] Danijel Žlaus and Domen Mongus. Efficient method for parallel computation of geodesic transformation on CPU. *IEEE Transactions on Parallel and Distributed Systems*, 31(4):935–947, 2019.