Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Bojana Kenda

Primerjava paralelizacij algoritmov za procesiranje slik v OpenCL

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Bojana Kenda, z vpisno številko **63110464**, sem avtorica diplomskega dela z naslovom:

Primerjava paralelizacij algoritmov za procesiranje slik v OpenCL

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 6. januarja 2015 Podpis avtorja:

Zahvaljujem se svojemu mentorju, izr. prof. dr. Patriciu Buliću za navdihujoča predavanja, ki so me privedla do dane tematike, za smernice v raziskovanju in za pomoč pri reševanju težav z viri, potrebnimi za izdelavo naloge.

Posebna zahvala velja tudi as. Roku Češnovarju za pomoč pri opredelitvi teme in tehnične nasvete od delu.

Staršem, za podporo v vseh letih študija.

Kazalo

Povzetek

V pričujoči nalogi smo raziskovali vpliv različnih nastavitev pri poganjanju programa za procesiranje slik z OpenCL ogrodjem. Vzeli smo tri konkretne metode iz programskega orodja GEGL, ki se tipično pojavljajo pri obdelavi slik, in primerjali njihovo hitrost izvajanja tako pri slikah različnih velikosti kot tudi na različni strojni opremi. Eno izmed metod smo poleg tega predelali za izrabo lokalnih pomnilnikov na OpenCL napravi in ugotavljali morebitne prednosti in slabosti takšne implementacije. V teoretičnem delu smo se v ta namen posvetili pregledovanju OpenCL arhitekture in možnostih, ki jih ponuja paralelno oziroma vzporedno računanje.

Ključne besede: OpenCL, paralelnost, delovne skupine, lokalni pomnilnik.

Abstract

The thesis explores the effects of running an image processing program with the OpenCL framework at different settings. Three methods, common in image processing, have been selected from the programming library GEGL and their execution times compared when using images of different sizes as well as on different hardware. In addition, one of the methods has been modified to use local memory on an OpenCL device, allowing us to identify the potential advantages and disadvantages of such an implementation. The theoretical part has therefore been dedicated to the overview of OpenCL architecture and the possibilities opened by parallel, or rather concurrent, computing.

Key words: OpenCL, parallelism, work groups, local memory.

Poglavje 1

Uvod

Namen dane diplomske naloge je raziskati odvisnost hitrosti procesiranja slik od načina implementacije programa z OpenCL ogrodjem. Natančneje, gre za primerjavo uporabe različnih nastavitev velikosti skupin niti, pri čemer vsaka nit rešuje določen problem na posamezni skupini pikslov. V primeru, da se to reševanje izvaja na procesorju grafične kartice, so najugodnejše nastavitve lahko drugačne od nastavitev, ki se izkažejo kot najprimernejše pri izvajanju na centralni procesorski enoti. V diplomski nalogi smo zato preiskali tudi morebitne razlike pri poganjanju jedrnega dela programa na eni in drugi arhitekturi.

Za razumevanje problema bo prvi del naloge namenjen krajši predstavitvi koncepta heterogenih in paralelnih sistemov z modeli sočasnega programiranja. V nadaljevanju bo sledil pregled značilnosti OpenCL kot programskega jezika in okolja za prenos izvajanja računskega dela na izbrano napravo, ki je v našem primeru bodisi CPE (centralna procesna enota) bodisi GPE (grafična procesna enota). Posebno pozornost bomo pri tem posvetili spominskim modelom in pomnilniški arhitekturi.

V drugem delu naloge bomo predstavili programske probleme, ki so predmet naše primerjane paralelizacije. Gre za trojico metod, ki so pogoste pri obdelavi slik, in sicer: odstranjevanje rdečih oči (ang. 'red-eye removal'), zameglitev s povprečjem skupnih najbližjih sosedov (ang. 'SNN mean') in Ga-

ussova zameglitev (ang. 'Gaussian blur'). Vse navedene programske metode so kot ščepci (ang. 'kernels') vzeti iz programske knjižnice GEGL (Generic Graphics Library), kjer so kot taki že prilagojeni OpenCL implementaciji.

Temu bo sledil osrednji del naloge, posvečen naši implementaciji programa v jeziku C, s katero smo beležili meritve izvajanja posameznih ščepcev ob različnih nastavitvah. Predstavili bomo opažanja in izpeljane ugotovitve.

V sklepu bodo med drugim predlagane tudi možnosti nadaljnih razširitev testiranja.

Poglavje 2

Paralelni sistemi in OpenCL

2.1 Vzporedno računanje

Pojem vzporednega računanja (ang. 'parallel computing') označuje posebno vrsto hkratnega programskega izvajanja. Potrebno ga je namreč ločiti od sicer sorodnega koncepta t.i. sočasnega računanja (ang. 'concurrent computing'), kjer je izvajalna "hkratnost"le iluzija, kot jo dojema uporabnik, oziroma je zaznavna navzven. Pri sočasnosti gre tako za izvajanje različnih tokov na način deljenja procesorskega časa. Računanje lahko poteka na enem samem procesorskem jedru in v primeru potrebe po hkratnem procesiranju večih ukazov ali podatkovnih tokov, procesor odredi vsakemu svojo časovno rezino izvajanja. Po drugi strani paralelizacija ali vzporednost v računanju pomeni dejansko hkratnost v izvajanju na večih procesorskih oziroma procesnih enotah. Drugače rečeno, sočasno računanje je pri vzporednosti podprto ne le programsko temveč tudi z ustrezno strojno opremo. Ker pa je za učinkovito porazdelitev dela med dostopne procesorske enote program potrebno prilagoditi konkretni strojni arhitekturi - tako z definiranjem tega, kaj se bo izvajalo vzporedno, kot tudi tega, kje in kako se bo izvajalo - paralelizacija kot rešitev obenem poudari problem računanja na heterogenih platformah.

Večina osebnih računalnikov dandanes predstavlja primer heterogene plat-

forme, saj združujejo večjedrno CPE, ki je sama po sebi sicer vzporedna strojna arhitektura, ne pa heterogeni sistem, in GPE. Razvoj slednje je že začetek stoletja zapustil okvire stroge grafične namembnosti, ko je doživela razcvet kot procesor za splošne namene (ang. 'general-purpose computing on graphics processing unit' ali 'GPGPU') zaradi vse boljše podpore za računanje s plavajočo vejico. Potencial za učinkovito izvajanje je s tem narasel, saj več specializiranih procesnih enot računanje kompleksnega problema opravi hitreje kot manj splošnih tipa CPE. Posamezni namenski procesor ima lahko pri reševanju določenega problema tako izkoriščenih več tranzistorjev, medtem ko pri splošnih procesorjih več tranzistorjev ob specifičnih nalogah ostaja neuporabljenih, saj niso vsi prilagojeni vsakršnim tipom operacij. Po drugi strani pa izvajanje določenih problemov oziroma delov programske naloge na določenih procesorjih zahteva več dela za programerja pri prilagajanju kode za konkretno strojno arihtekturo izbranega namenskega procesorja. V kolikor ne omejimo denimo dela namenjenega GPE zgolj na grafiko, mora program vključevati specifično definicjo tega, kje se bo v katerem delu izvajal. OpenCL, kot bomo videli v nadaljevanju, na tem mestu nastopi kot ogrodje, ki omogoča prenosljivost izvajanja v heterogenih sistemih.

Pri vzporednem računskem izvajanju je potrebno opredeliti še na kakšni ravni se paralelizacija izvaja in kaj je dejanski predmet paralelizacije. Paralelnost računanja se lahko izvaja med več računalniškimi sistemi ali znotraj enega samega. Delo je v prvem primeru porazdeljeno na primer med več delovnih postaj, v drugem pa med izvajanje na enotah grafične kartice ali CPE jedrih. Tema pričujoče naloge je paralelizacija slednjega tipa. Po drugi strani ločimo več vrst vzporednosti glede na objekt oziroma programski nivo, kjer nastopi paralelizirano delo. Najbolj osnovna delitev je na:

- **vzporednost na nivoju bitov** - boljši procesorji lahko obdelajo daljše bitne besede v enem ciklu;
- **vzporednost na nivoju ukazov** - izvaja se več ukazov hkrati, pogosto v cevododu, če niso med seboj podatkovno odvisni;
- **podatkovna vzporednost** - operacija se izvaja nad različnimi po-

datki;

- **vzporednost nalog** - niti oziroma izvajalni procesi z različnimi nalogami operirajo nad istimi ali različnimi podatki in so porazdeljeni med različne izvajalne enote.

OpenCL podpira le zadnja dva načina paralelizacije. Pri tem prevladuje podatkovna vzporednost, ki se "naravno prilega enotam izvajalnega modela OpenCL"[1][1, str. 27], za razliko od vzporednosti nalog, za katero sicer OpenCL podpira vrsto algoritmov[1, str. 27], a je uporabljana redko in predvsem s CPE napravami[2, str. 3]. Tudi v sklopu dane naloge smo raziskovali le podatkovno vzporednost.

Pri implementaciji vzporednosti si običajno prizadevamo pohitriti delovanje programa, zato smo tudi pri našem problemu iskali razmere, ki bi dosegle najhitrejše izvajanje. Več vzporednih tokov računanja namreč ne pomeni nujno hitrostnega izboljšanja zaradi narave izvajalega modela, kot bomo videli v nadaljevanju. Poleg tega ima vsaka optimizacija mejo svojih zmožnosti, kot je znano že iz Amdahlovega zakona[2], ki pojasnjuje, da je pohitritev omejena z odstotkom dela programa, ki ostaja sekvenčen. Pohitritev, dosežena s paralelnim izvajanjem, je v splošnem torej le del rešitve.

2.2 OpenCL

Strojna podpora vzporednemu računanju zahteva programsko opremo, ki omogoča paralelno izvajanje operacij. Za lažji in s tem širši razvoj le-te pa je potrebno okolje, ki omogoča prenosljivost izvajanja programa na eno ali drugo procesorsko napravo. NVIDIA je v ta namen leta 2007 razvila platformo in programski model CUDA, katere pomanjkljivost je bila neprenosljivost na grafične kartice tujih proizvajalcev. Apple, Inc. je leto kasneje

[1] "/.../ data parallelism is a natural fit to the OpenCL execution model items."

[2] $S(N) = \frac{1}{(1-P)+(P/N)}$, kjer je P paraleliziran del programa, N število procesorjev in S maksimalna pohitritev.

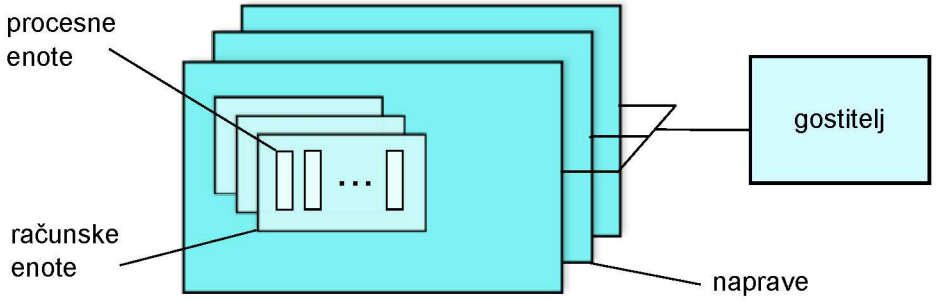
zasnoval OpenCL, Khronos Group pa definiral njegove specifikacije v smislu industrijskega standarda, ki je neodvisen od proizvajalca. Uporaben je tako za delo z GPE in CPE kot tudi za platforme, osnovane na DSP (ang. 'digital signal processor') in FPGA (ang. 'field-programmable gate array') vezjih. Proizvajalci, ki v svoje produkte vključujejo podporo za OpenCL, so med drugim: NVIDIA, AMD, Intel, QUALCOMM, MediaTek Inc., ARM, ARM Limited, Imagination Technologies, Apple, Inc., Altera Corporation, IBM Corporation in Samsung Electronics. Do časa pisanja te naloge so bile izdane štiri verzije: 1.0, 1.1, 1.2 in 2.0, pri čemer je bilo delo za nalogo narejeno na verziji 1.2.

Programski jezik, v katerem so pisani ščepci oziroma deli programa, ki se izvajajo vzporedno, je osnovan na ISO C99 standardu in je znan kot OpenCL programski jezik C (ang. 'OpenCL C'). Preostali del kode, ki predstavlja platformni OpenCL programski vmesnik in izvajalni OpenCL programski vmesnik, je običajno pisan v programskih jezikih C in C++, obstaja pa tudi podpora za druge jezike, kot so na primer Python, Ruby in C#. Kot platformni vmesnik pri tem nastopa definiranje izvajalnega konteksta. Na nekem sistemu je namreč lahko obenem več različnih platform, saj proizvajalci posameznih procesorskih naprav, na primer GPE in CPE, lahko definirajo različna OpenCL okolja. Izvajalni vmesnik pa po drugi strani vključuje funkcije, namenjene uporabljanju tega izvajalnega konteksta. V nadaljevanju si bomo ogledali podrobnejšo opredelitev značilnosti in strukture konteksta OpenCL aplikacije oziroma platformnega in izvajalnega modela.

2.2.1 Platformni in izvajalni model

Platformni model OpenCL zajema naslednje strukturne elemente:

- **gostitelj** (ang. 'host');
- **OpenCL naprave** (ang. 'OpenCL devices');
- **računske enote** (ang. 'compute units');
- **procesne enote** (ang. 'processing elements').



Slika 2.1: Shema OpenCL platformnega modela

Njihova hierarhična povezanost je prikazna na sliki 2.1. Gostitelj je naprava, na katerem se izvaja gostiteljska aplikacija (ang. 'host application'). Ta definira kontekst izvajanja in razporeja ščepce v izvajalno vrsto. Na gostitelja so nadalje povezane naprave, ki so posamezne procesorske arhitekture z OpenCL podporo, na primer GPE, CPE ali DSP, na katere gostitelj pošilja ščepce v izvajanje. Ti se nato izvajajo v sklopu računskih enot, pri čemer je procesna enota osnovni računski element. Grupiranje procesnih enot v računske enote je povezano z dostopi do ločenih oziroma skupnih pomnilniških struktur in sinhronizacijskimi zmožnostmi, kar bo podrobneje opredeljeno pri govoru o OpenCL pomnilniški arhitekturi.

Osnovna delitev programske strukture pri OpenCL aplikaciji je torej na:

- program, ki teče na gostitelju, in
- ščepce.

Izvajanje ščepcev je kot rečeno vezano na izvajalni kontekst, ki pa je pri OpenCL poleg ščepcev samih definiran še z naslednjimi strukturami in objekti:

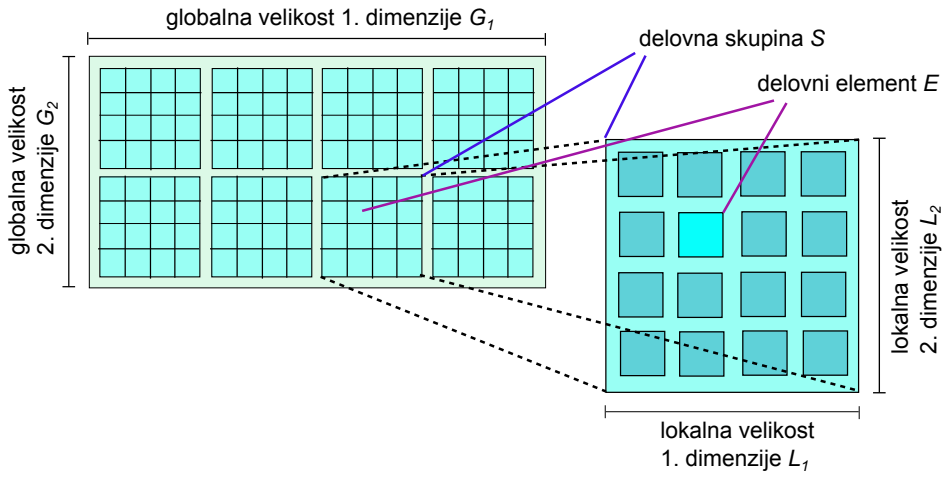
- naprave;
- programski objekti, ki implementirajo ščepce;
- pomnilniški objekti, vidni in dostopni za naprave.

Konkretna organizacija računskega dela, ki se izvaja na OpenCL napravi, je deloma odvisna od specifikacij naprave, deloma od narave ščepca ali morebitnih eksplicitnih definicij v ščepcu, preostanek pa je prepuščen programerju. V osnovi gre za delitev dela med t.i. delovne elemente (ang. 'work-items'), ki so sorodni konceptu niti v drugih sistemih (denimo pri CUDI). V primeru podatkovno paraleliziranega programa vsak delovni element izvaja kodo ščepca na svojem razdelku podatkovne celote. Definirani so s koordinatami večdimenzionalnega celoštevilsko indeksiranega prostora, ki ga sistem ustvari ob izvajanju, in so organizirani v t.i. delovne skupine (ang. 'work-groups'), konceptualno sorodne blokom niti (pri CUDI). Te v vsaki dimenziji enakomerno razdelijo globalni prostor, kar pomeni, da mora biti globalna velikost prostora v neki dimenziji vedno večkratnik števila delovnih elementov na skupino v tisti dimenziji.

Slika 2.2 prikazuje odnos med globalnimi in lokalnimi dimenzijami ter koncept grupiranja delovnih elementov v delovne skupine. Podan je primer dvodimenzionalnega prostora, pri čemer OpenCL podpira tudi eno- ali trodimenzionalni prostor. Globalna velikost prve dimenzije, označena kot $G1$, je v danem primeru enaka 16, saj vsebuje 4 delovne skupine, katerih lokalna velikost prve dimenzije je enaka 4. Globalna velikost druge dimenzije je temu ustrezno 8, saj vsebuje 2 delovni skupini z lokalno velikostjo iste dimenzije, ki je prav tako enaka 4. Če želimo definirati element E, potrebujemo njegov globalni identifikator (ang. 'global ID'), ki je vedno n-terica toliko celih števil, kot je definiranih prostorskih dimenzij. E je tako definiran z globalnim identifikatorjem $(g1, g2)$, ki je v našem primeru $(9, 5)$, do česar nas lahko pripelje tudi preračunavanje na podlagi identifikatorja delovne skupine $(s1, s2)$ in elementovega lokalnega identifikatorja $(l1, l2)$. Indeks $s1$ je pri skupini S v našem primeru enak 2, $s2$ pa 1, medtem ko se E znotraj skupine nahaja na lokalnem indeksu $(1, 1)$. Za globalni identifikator tako velja:

$$gx = sx * Lx + lx, \tag{2.1}$$

kjer je x element množice $\{1, 2, 3\}$ in predstavlja pripadnost eni od treh dimenzij.



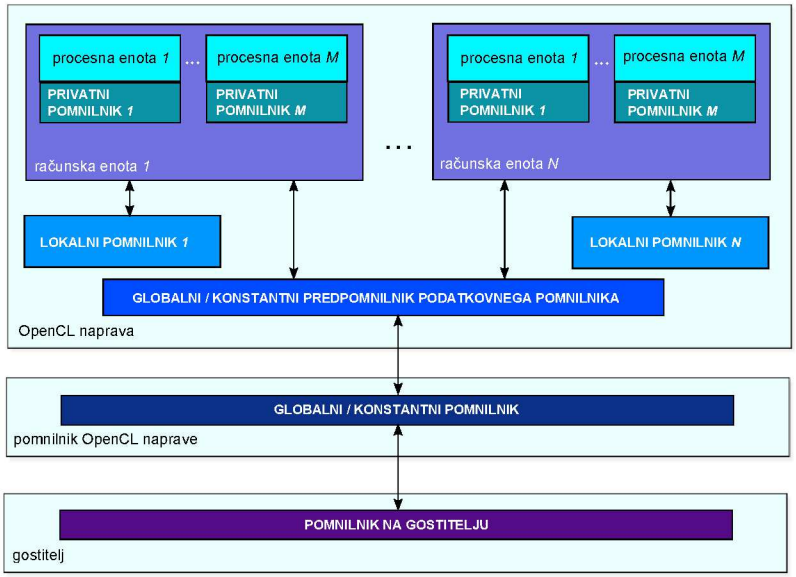
Slika 2.2: Organizacija delovnih niti in delovnih skupin

Prav tako velja že omenjeno razmerje med globalno in lokalno velikostjo dimenzij:

$$Lx = Gx/Sx, \tag{2.2}$$

kjer je x element množice $\{1, 2, 3\}$ in S število delovnih skupin v dani dimenziji.

Ideja grupiranja delovnih elementov v delovne skupine je neločljivo povezana s paralelizacijo dela. Vsi delovni elementi v neki skupini namreč svoje delo izvajajo sočasno. Ni pa nujno, da se vse delovne skupine izvajajo sočasno, pri čemer tudi vrstni red njihovega izvajanja ni definiran. Posledica tega je, da s sihronizacijskimi pregradami znotraj ščepcev lahko definiramo mesta, kjer se morajo vsi delovni elementi znotraj posamezne delovne skupine sinhronizirati. Sinhronizacija dela delovnih elementov iz različnih delovnih skupin po drugi strani ni podprta. Razlog je vezan tako na strukturo pomnilniških elementov pri OpenCL, ki omogočajo hitrejše dostope delovnih elementov do podatkov, kot tudi na dejansko implementacijo procesorjev na GPE, kjer si dve jedri znotraj enega procesorja delita skupni pomnilnik na istem čipu.



Slika 2.3: Pomnilniški model OpenCL in interakcija pomnilniških območij

2.2.2 Pomnilniška arhitektura

Pomnilniška območja pri OpenCL vključjejo:

- **pomnilnik gostitelja** (ang. 'host memory'), viden samo za gostitelja;
- **globalni pomnilnik** (ang. 'global memory'), do katerega lahko dostopajo vse delovne skupine;
- **pomnilnik s konstantami** (ang. 'constant memory'), ki se nahaja kot podobmočje znotraj globalnega pomnilniškega prostora in se ne spreminja ob izvajanju ščepca;
- **lokalni pomnilnik** (ang. 'local memory'), ki je lasten posamezni delovni skupini;
- **privatni pomnilnik** (ang. 'private memory'), ki je lasten posameznemu delovnemu elementu.

Slika 2.3 prikazuje arihtekturno shemo teh območij in interakcijo posameznih elementov pomnilniške hierarhije. Vsi podatki se pri zagonu aplikacije

nahajajo v pomnilniškem modelu gostitelja. V primeru, da do njih želi dostopati OpenCL naprava, ki ni CPE in s tem uporablja svoj ločen globalni pomnilnik, se morajo bodisi kopije podatkov tja prenesti, bodisi je potrebno preslikati (ang. 'map') dele pomnilniških objektov. Slednje omogoči gostitelju preslikavo objektov v svoj lastni naslovni prostor in s tem izvajanje bralnih ter pisalnih dostopov do njih.

Dana shema tudi pojasnjuje možnosti in omejitve sinhronizacij med delovnimi elementi. Delovni elementi iste delovne skupine se lahko sinhronizirajo znotraj izvajanja določenega ščepca, saj dostopajo do lokalnega pomnilnika skupine, ki ima lahko s programskimi pregradami (ang. 'work-group barriers') zagotovljene točke, kjer so vrednosti v pomnilniku konsistentne za vse elemente. Druga skupina bo podobno zagotavljala konsistentnost v svojem lokalnem pomnilniku in s tem sinhronizirala delo svojih delovnih elementov. Konsistentnost vrednosti med posameznimi delovnimi skupinami pa je lahko zagotovljena le ob začetku in koncu izvajanja ščepca, kjer je mogoč sinhroniziran dostop do globalnega pomnilnika.

Ob opredelitvi pomnilniške arhitekture je potrebno predstaviti tudi tipe OpenCL podatkovnega modela:

- **medpomnilniški objekti** (ang. 'buffer objects'), namenjeni kakršnimkoli podatkovnim strukturam;
- **slikovni objekti** (ang. 'image objects'), namenjeni zgolj delu s slikovnimi objekti.

Poglavitna razlika med obema je način shranjevanja in dostopanja do podatkov. Medpomnilniški objekti hranijo podatke v eni dimenziji, medtem ko slikovni objekti shranjujejo dvo- ali tridimenzionalne teksture oziroma slike. Po drugi strani slikovni objekti lahko hranijo le vnaprej definirane slikovne podatkovne tipe, medtem ko medpomnilniški objekti lahko prejmejo poljubno podatkovno strukturo. Dostop do elementov, hranjenih v medpomnilniškem objektu, je preko kazalcev, saj so elementi razporejeni v sekvenčnem vrstnem redu. Struktura elementov v slikovnem objektu pa uporabniku ni razvidna,

zato tudi dostop preko kazalcev ni možen in zanj skrbijo vgrajene funkcije znotraj ščepcev.

V pričujoči nalogi smo se osredotočili na shranjevanje podatkov v medpomnilniških objektih. Možnost nadaljnjih raziskav je zato v ponovitvi izvajanih meritev z uporabo slikovnih objektov in pregledu razlik, ki jih lahko s tem prinese drugačen način dostopanja do pomnilniških objektov.

Poglavje 3

Procesiranje slik

Procesiranje slik kot programski problem smo za nalogo izbrali zaradi pestrega nabora različnih algoritmov, ki jih lahko pri tem primerjamo. Od algoritma do algoritma je namreč odvisno, kako časovno zahtevna bo implementacija ščepcev, kakšne bodo potrebe pri dostopanju do pomnilnikov in posledično, kakšna bo najugodnejša razporeditev delovnih elementov v delovne skupine. Slike so poleg tega realen primer večjega nabora podatkov, ki jih mora bodisi CPE bodisi GPE pri povprečnem uporabniku urejevalnika slik pogosto in čim hitreje sprocesirati. Iskanje najoptimalnejših nastavitev za paralelizacijo tovrstnih metod je zato splošen problem, ki so ga nekateri proizvajalci urejevalnikov slik že načeli.

Za raziskovalni problem naloge smo vzeli tri metode iz programske knjižnice GEGL[5] (Generic Graphics Library), ki se kot odprtokodna programska oprema[1] uporablja pri aplikacijah za procesiranje slik, kot na primer pri GIMPu od verzije 2.6[6] dalje.

[1]GEGL je izdan pod licenco *GNU LGPL* ali *GNU Lesser General Public Licence*, ki dovoljuje redistribucijo in prosto modifikacijo kode, ne pa nujno vključevanje v lastniške produkte.

3.1 GEGL

Generic Graphics Library ali krajše GEGL je orodje za procesiranje slik na osnovi usmerjenih acikličnih grafov. Ena od ključnih prednosti knjižnice je podpora za neuničujoče urejanje (ang. 'non-destructive editing'), kjer se ne spreminja izvorna vsebina slike in zato ni izgub podatkov v primeru, da se želimo vrniti v predhodno stanje. V urejevalnem grafu tako vsako vozlišče predstavlja operacijo nad sliko, vsaka povezava pa predstavlja sliko. Gre za model izvajanja na zahtevo (ang. 'on-demand model'), kjer uporabnik lahko izvaja operacije na predogledu, GEGL pa šele ob zaključenem urejanju ponovi izbrane operacije nad dejansko sliko.

Optimizacija, ki jo GEGL doprinese h GIMPu, je med drugim podpora za procesiranje slik v formatih visoke natančnosti, kot je plavajoča vejica. Ker pa gre pri tem za časovno zahtevne računske operacije, so GEGL razširili z moduli za procesiranje v OpenCL, ki lahko teče na GPE v vzporednih tokovih. V času pisanja naloge je bilo na ta način podprtih že 39 filtrov[8] v obliki enega ali več ščepcev, ki se izvajajo v sklopu dane metode. OpenCL je v GIMP vključen z najnovejšo verzijo 2.8[2][7].

3.2 Paralelizirane metode

Izbrani filtri za dano nalogo vključujejo metodo za odstranjevanje rdečih oči, metodo za zameglitev s povprečjem skupnih najbližjih sosedov in metodo za Gaussovo zameglitev. Najkompleksnejša je metoda zameglitve s povprečjem skupnih najbližjih sosedov, sledi Gaussova zameglitev, ki implemetira sklop dveh ščepcev, izvajanjih v zaporedju, najenostavnejša pa je metoda za odstranjevanje rdečih oči.

[2]Od omenjenih 39 filtrov, dodanih v GEGLov GIT repozitorij ni nujno, da so vsi že dodani v tej verziji.

3.2.1 Odstranjevanje rdečih oči

Ščepec s funkcionalnostjo, ki na danem odseku slike odstrani rdeče oči, se nahaja v datoteki *red-eye-removal.cl*[9]. Na podanih vhodnih podatkih glede na uporabniško nastavljen prag zmanjša intenziteto rdečega kanala, v primeru, da je ta v previsokem razmerju glede na intenziteto zelenega in modrega kanala.

V programu, s katerim smo testirali čas izvajanja ščepca, se nahaja v manjši meri predelana različica izvorne kode, v psevdokodi podana pod izsekom 3.1.

Izsek 3.1: Ščepec za odstranjevanje rdečih oči

```
RED_FACTOR = 0.5133333f
GREEN_FACTOR = 1
BLUE_FACTOR = 0.1933333f

funkcija cl_red_eye_removal(srcBuf, dstBuf, threshold)
   vhodni_piksel = srcBuf[globalni_ID]
   nova_rdeca = vhodni_piksel.rdeca_komponenta * RED_FACTOR
   nova_zelena = vhodni_piksel.zelena_komponenta *
      GREEN_FACTOR
   nova_modra = vhodni_piksel.modra_komponenta *
      BLUE_FACTOR
   prag = (threshold - 0.4) * 2

   IF nova_rdeca >= (nova_zelena - prag) in nova_rdeca >=
      (nova_modra - prag):
      rdeci_kanal = (nova_zelena + nova_modra) / (2 *
         RED_FACTOR)
      IF rdeci_kanal > 1:
         rdeci_kanal = 1
      vhodni_piksel.rdeca_komponenta = rdeci_kanal

   dstBuf[globalni_ID] = vhodni_piksel
```

3.2.2 Zameglitev s povprečjem skupnih najbližjih sosedov

Funkcionalnost zameglitve s povprečjem skupnih najbližjih sosedov je zajeta v ščepcu *snn_mean*[10] pod datoteko *snn-mean.cl*. Algoritem, ki ga uporablja, je povzet po metodi Jarvisa in Patricka[12], ki sta prva predlagala razvrščanje v skupine na podlagi iskanja skupnih najbližjih sosedov. Po njuni definiciji sta si točki v večdimenzionalnem atributnem vektorskem prostoru podobni, če imata skupen seznam k najbližjih sosedov, v katerem sta tudi sami vključeni[12, str. 1026]. Dana OpenCL implementacija uporablja ta pristop s prejetjem dveh parametrov, ki določata obseg iskanja najbližjih sosedov. Pri prvem gre za polmer, $radius$, obsega pikslov okoli točke, določene z globalnim identifikatorjem, znotraj katerega algoritem išče podobne sosede; drugi, $pairs$, pa predstavlja število sosedov, ki jih primerja ob vsaki iteraciji odločanja, kateri piksel sodi med bližnje sosede glede na barvno sorodnost, pri čemer so to štirje v primeru, da je parameter enak 2 in dva v primeru, da je parameter enak 1. Slednji bo dosegel manj intenzivno zameglitev, saj bodo sosedi, na podlagi katerih se naračuna povprečje kot nova vrednost za izbrani piksel, manj razpršeni, oziroma izbrani iz polovičnega nabora sosedov na obsegu danega radija.

Testiranje smo izvajali pri konstantnih vrednostih parametrov $pairs$ ter $radius$, in sicer smo za $pairs$ vzeli zahtevnejšo različico, vrednost 2, za $radius$ pa vrednost 5.

Glede na časovno kompleksnost algoritma, ki za vsak piksel, z vsakim delovnim elementom, pregleduje okoliških $(2 * radius)^2$ pikslov, oziroma pri $radius = 5$ še $100 - 1$ dodatnih pikslov, smo za primerjavo dodali še izpeljani ščepec *snn_mean_local*, ki izkorišča rabo lokalnega pomnilnika. Za dostopnost vseh potrebnih pikslov iz lokalnega pomnilnika prvi del kode v ščepcu poskrbi za prenos zahtevanih podatkov v lokalni pomnilnik. Gre za

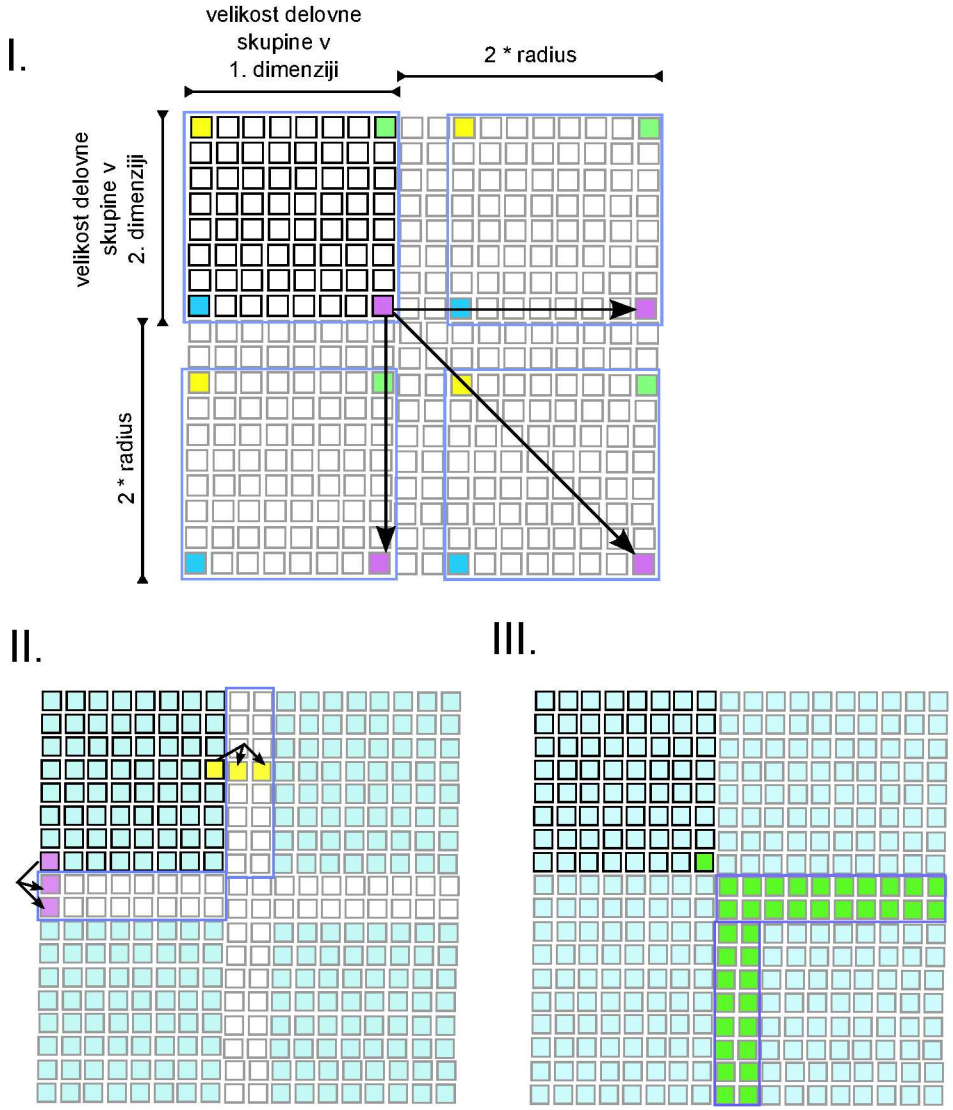
izsek slike v velikosti $(Lx + 2 * radius) * (Ly + 2 * radius)$, kjer sta Lx in Ly velikosti delovne skupine po x in y osi oziroma 1. in 2. dimenziji. Vsak izmed delovnih elementov, katerih lokalni identifikator v neki dimenziji kaže na pozicijo znotraj delovne skupine, ki je od spodnjega ali desnega roba oddaljena manj ali enako od dvakratne dolžine *radiusa*, bo za svoje računanje potreboval piksle izven obsega delovne skupine. Zato poskrbi še za prenos enega do treh dodatnih pikslov v lokalni pomnilnik, odvisno od tega, v koliko dimenzijah s svojimi potrebami prekorači obseg delovne skupine. Pomanjkljivost tega algoritma je v pasu zahtevanih pikslov, ki na ta način ostanejo neprenešeni v primeru, da je velikost delovne skupine manjša od $2 * radius$, kolikor dodatnih pikslov potrebuje posamezen element v vsako od smeri in s tem ne morejo biti prenešeni niti če vsak delovni element prenese poleg podatkov o svoji poziciji še podatke za tri dodatne piksle. Delovni elementi ob spodnjem in desnem robu zato poskrbijo za prenos preostalih podatkov, pri čemer je element v skrajno desnem spodnjem kotu najbolj obremenjen. Razporeditve skupin, ki bodo vsebovale manj elementov v posamezni dimenziji kot je dvakratna velikost polmera zameglitve, bodo zato bistveno manj optimalne. Algoritem bi bilo z boljšim razporejanjem manjkajočih podatkov med delovne elemente mogoče še bistveno izboljšati, kar je zagotovo ena od možnosti prihodnjih razširitev. Relevantni del predelane kode za dano metodo je v obliki psevdokode prikazan pod izsekom 3.2, prenos podatkov za piksle izven meja delovne skupine pa je ponazorjeno pod sliko 3.1.

Izsek 3.2: Ščepec za zameglitev s povprečjem skupnih najbližjih sosedov z izrabo lokalnega pomnilnika

```
funkcija snn_mean_local(srcBuf, srcWidth, srcHeight,
   dstBuf, radius, pairs, tempBuf)
   tempBuf[lokalni_ID] = srcBuf[globalni_ID]

   Po potrebi prepis dodatnih vrednosti v tempBuf.

   Sinhronizacijska pregrada.
```



Slika 3.1: Prepis dodatnih vrednosti v lokalni medpomnilnik OpenCL naprave.

```
centralni_piksel = tempBuf[(radius + lokalni_ID.x) +
   (radius + lokalni_ID.y) * velikost_medpomnilnika.x]
sesteti, stevec <- inicializacija vsote in stevca
   vrednosti najblizjih sosedov

IF pairs == 2:
   Za vsak i = -radius ... 0:
      Za vsak j = -radius ... 0:
         izbrani_piksel = centralni_piksel
         max_razlika = 1000

         lokalni_x = [lokalni_ID.x + i + radius,
            lokalni_ID.x - i + radius]
         lokalni_y = [lokalni_ID.y + j + radius,
            lokalni_ID.y - j + radius]
         Za vsako kombinacijo (x, y) vrednosti iz
            lokalni_x in lokalni_y:
            temp_piksel = tempBuf[lokalni_x[x] +
               lokalni_y[y] * velikost_medpomnilnika.x]
            razlika =
               izracunaj_barvno_razliko(temp_piksel,
               centralni_piksel)
            IF razlika < max_razlika:
               max_razlika = razlika
               izbrani_piksel = temp_piksel

         sesteti += izbrani_piksel
         stevec += 1

   dstBuf[globalni_ID] = sesteti / stevec
   return
```

3.2.3 Gaussova zameglitev

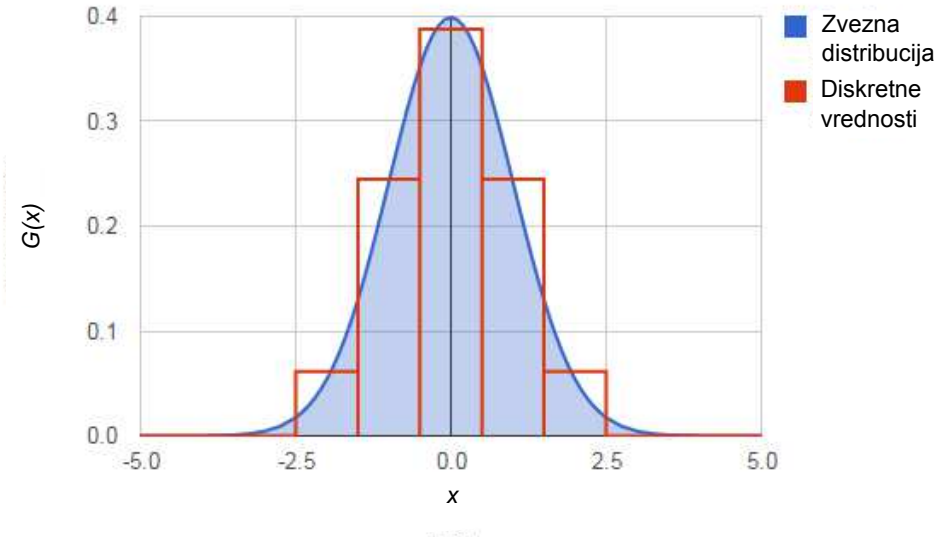
Gaussovo zameglitev smo izvajali s sklopom dveh ščepcev v datoteki *gaussian-blur.cl*[11]. Prvi, *fir_ver_blur*, se praviloma izvaja nad izvornimi podatki in nad sliko naredi vertikalno zameglitev, medtem ko se drugi, *fir_hor_blur*, izvede nad predelano sliko oziroma nad podatki, ki jih izračuna prvi ščepec, ter doda horizontalno zameglitev. Oba sta odvisna od konvolucijske matrike, ki je podana kot parameter, in vrednosti parametra $yoff$ pri prvem, oziroma $xoff$ pri drugem, ki predstavlja odmik po osi x oziroma y.

V naši implementaciji smo zaradi poenostavitve pri poganjanju testnih izvedb sklopa ščepcev obema podali enake vrednosti parametrov: za odmik po osi x in odmik po osi y smo vzeli vrednost 0, za izvorne podatke, nad katerimi posamezni ščepec računa, pa smo prav tako pri obeh uporabili podatke izvorne slike. Končni rezultat slike zato ni realen, so pa realni časi izvajanja, ki v danem primeru upoštevajo seštevek časa prvega in drugega ščepca.

Konvolucijska matrika je pri kombinaciji dveh enodimenzionalnih ščepcev prav tako enodimenzionalna. Vrednosti posameznih elementov so vzeti iz vnosa spletnega dnevnika *The Devil in the Details* pod naslovom *Gaussian Kernel Calculator*[13], kjer so bili izračunani na podlagi Gaussove funkcije v enojni dimenziji:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}, \tag{3.1}$$

pri $\sigma = 1.0$ in dolžini matrike 5. Vrednosti zvezne Gaussove krivulje so bile nato diskretizirane na srednje vrednosti kot prikazano na sliki 3.2.



Slika 3.2: Diskretizacija Gaussove krivulje za vrednosti konvolucijske matrike, povzeto po spletnem dnevniku *The Devil in the Details*[13]

Poglavje 4

Implementacija in rezultati meritev

Pri pregledu implementacije testiranja bo predstavljena struktura programa v jeziku C, s katerim smo zaganjali OpenCL ščepce in preverjali rezultate s prikazovanjem procesiranih slik, na kratko pa bo opisana tudi logika spisanih skript v jeziku Python za pregled in vizualizacijo rezultatov. V okviru rezultatov izvajanih meritev bo namreč grafično predstavljena odvisnost izvajalnega časa posameznega ščepca od različnih kombinacij velikosti delovnih skupin. Podrobnejše informacije o najoptimalnejših nastavitvah bodo razvidne tudi iz podanih tabel.

4.1 Program za izvajanje meritev

Pred začetkom priprave OpenCL procesa in izvajanja ščepcev na izbranih OpenCL napravah je bila potrebna priprava podatkov. Za testiranje smo kot predmet procesnih metod vzeli dve sliki v PNG formatu; eno velikosti 512×512 pikslov, drugo v velikosti 2048×1024 pikslov. Ker so vsi ščepci za svoje parametre zahtevali podatke tipa float4[1], smo morali podatke iz slike prebrati v obliko, kjer informacijo posameznega piksla predstavlja četverica

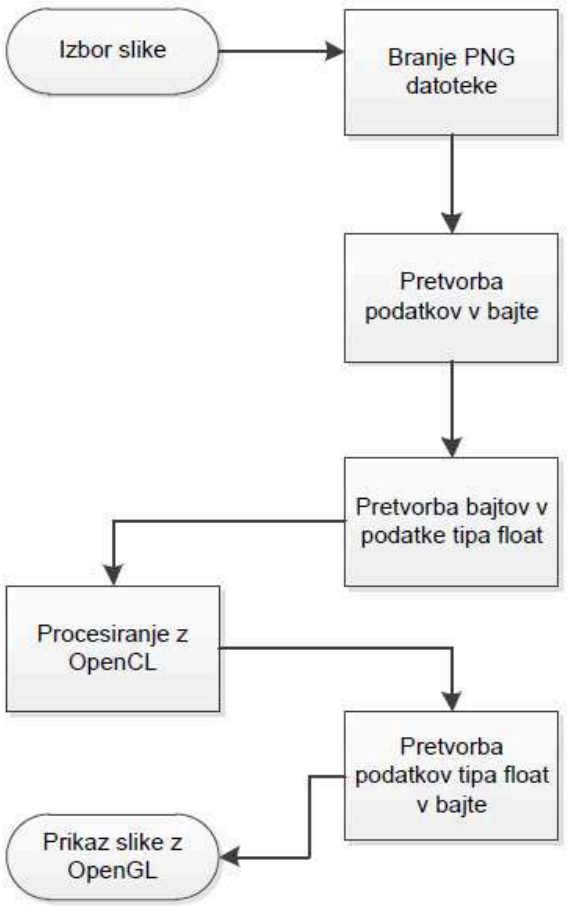
[1] *float4* v OpenCL spada med vektorske komponente[4, str. 132-135].

zapisov v plavajoči vejici. Pri tem smo za branje slike v RGBA zapis z bajti uporabili knjižnico libpng verzije 1.5.17[16]. V pomoč pri uporabi knjižnice za obdelavo podatkov v programu v C nam je bil primer G. Cottenceauja[14] in njegova predelana različica Y. Niwe[15], oba prosto dostopna pod pogoji licence X11. Po nadaljnji pretvorbi bajtov v podatke tipa float je OpenCL proces lahko sliko obdelal glede na izbrani ščepec in vrnil rezultat v obliki zapisa s plavajočo vejico.

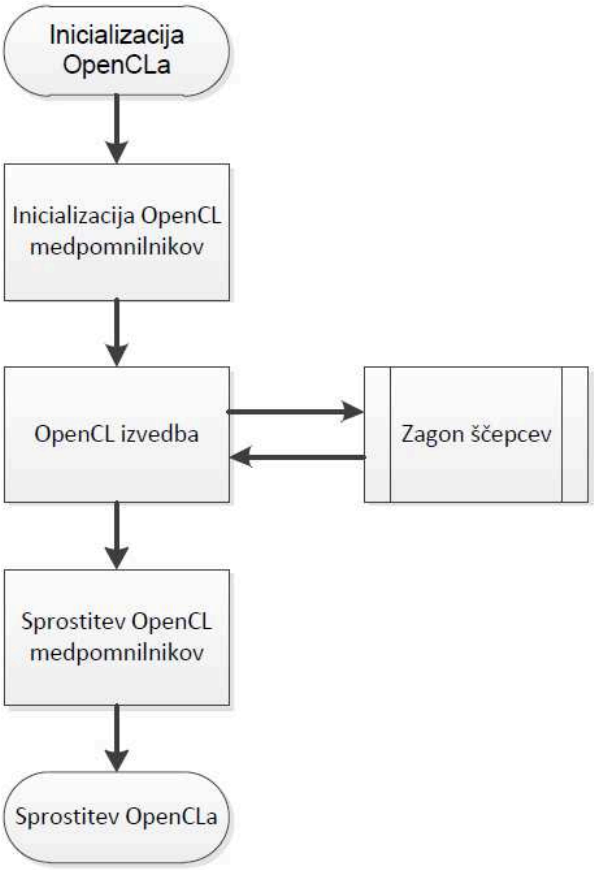
Zadnja stopnja v programu je bil izris obdelane slike na zaslon za preverbo pravilnosti delovanja danega ščepca. V ta namen smo posegli po knjižnici OpenGL[17], kjer prikaz teksture oziroma slike ponovno zahteva podatke v obliki bajtov. Podatke tipa float smo zato ponovno pretvorili v bajte in izročili OpenGL procesu. Metode in funkcionalnosti OpenGL izrisovanja tekstur so bile povzete po prostodostopnih primerih na spletu[19][18].

Okvirna shema zunanjega poteka delovanja programa je prikazana na sliki 4.1, medtem ko slika 4.2 prikazuje podrobnejše sosledje procesov v okviru OpenCL dela programa. Najprej je potrebna inicializacija OpenCL konteksta s `clCreateContextFromType()`, ki se ustvari glede na izbrano zastavico tipa naprave, `CL_DEVICE_TYPE_CPU` ali `CL_DEVICE_TYPE_GPU`. Na podlagi konteksta se nato določi identifikator naprave s `clGetDeviceIDs()`, preko obeh pa nadalje ukazna vrsta s `clCreateCommandQueue()`. V sklopu OpenCL inicializacije se ustvari še program na osnovi datoteke z izbranim ščepcem, ki jo podamo kot niz funkciji `clCreateProgramWithSource()`, medtem ko klic `clBuildProgram()` dani program zgradi. Ker se pri tem v programu lahko nahaja ne le več funkcij, temveč tudi več ščepcev, je nazadnje potrebno definirati tudi ime ščepca s `clCreateKernel()`, ki naj gre v izvajanje ob zagonu OpenCL programa na napravi.

Sledi inicializacija OpenCL medpomnilnikov. S funkcijo `clCreateBuffer()` se za vsak zahtevan parameter izbranega ščepca, ki ni enostavni tip, ustvari medpomnilniški objekt. Zastavici `CL_MEM_READ_ONLY` in `CL_MEM_WRITE_ONLY` pri tem določata, ali bo objekt namenjen shranjevanju rezultatov ali hranjenju izvornih podatkov. V primeru, da se bo ščepec izvajal



Slika 4.1: Diagram poteka programa za izvajanje meritev



Slika 4.2: Diagram poteka OpenCL procesa znotraj programa za izvajanje meritev

na CPE, dani izmed njih dodamo še `CL_MEM_USE_HOST_PTR`, kar omogoča uporabo kazalca na del pomnilnika, ki je že rezerviran v pomnilniškem prostoru gostitelja. Kadar želimo za izvajanje ščepca uporabljati GPE, je za pomnilniške objekte, namenjene branju, potrebno zagotoviti še kopiranje podatkov iz ponilnškega prostora gostitelja v pomnilniški prostor naprave s funkcijo `clEnqueueWriteBuffer()`.

Pri OpenCL izvedbi gre na prvem mestu za nastavljanje posameznih argumentov, ki so zahtevani pri izbranem ščepcu s `clSetKernelArg()`. Za tem nastopi zagon ščepca s funkcijo `clEnqueueNDRangeKernel()`, ki ukaz za izvajanje ščepca postavi v definirano ukazno vrsto. Kot sugerira ime funkcije, se ob izvajanju ustvari N-dimenzionalni celoštevilski prostor, zato je med parametri, ki jih funkcija prejme, podana tudi informacija o tem, v koliko dimenzijah naj se delovni elementi razporedijo, kakšna je globalna velikost in kakšne naj bodo lokalne dimenzije posameznih delovnih skupin. Funkciji `clWaitForEvents()` in `clFinish()` počakata na zaključek ukaza za izvajanje ščepca in do takrat blokirata ukazno vrsto.

Sprostitev OpenCL medpomnilnikov poskrbi za kopiranje podatkov iz pomnilniškega prostora naprave v pomnilniški prostor gostitelja, kjer je to potrebno in kadar gre za izvajanje na GPE, s funkcijo `clEnqueueReadBuffer()`. Sledi sprostitev OpenCL pomnilniških objektov s `clReleaseMemObject()`. Nazadnje sprostitev OpenCL sprosti ščepec oziroma zmanjša števec referenc na ščepec s funkcijo `clReleaseKernel()`, sproti program oziroma zmanjša števec referenc nanj s funkcijo `clReleaseProgram()`, sprosti ukazno vrsto oziroma zmanjša števec referenc nanjo s funkcijo `clReleaseCommandQueue()`, in sprosti kontekst oziroma zmanjša števec referenc nanj s funkcijo `clReleaseContext()`.

Za potrebe izvajanja meritev pri različnih nastavitvah, ki vključujejo tako različne vhodne slike in ščepce, kot tudi različne dimenzije delovnih skupin in različno število zaporednih izvajanj, smo parametrizirali vse zahtevane možnosti in kombinacije nastavitev. V funkciji `main()`, iz katere kličemo procesiranje z OpenCL, kot ga prikazuje diagram na sliki 4.2, definiramo

naslednje spremenljivke:

- `deviceType`, ki določa, ali želimo poganjati ščepec na CPE ali na GPE;
- `kernelFile`, ki določa datoteko z metodo, ki jo želimo testirati (*snn-mean.cl*, *red-eye-removal.cl*, ali *gaussian-blur.cl*);
- `image`, ki določa pot do slike, ki jo želimo vzeti kot predmet procesiranja;
- `dimension3`, ki določa velikost delovnih skupin v tretji dimenziji za dano serijo meritev;
- `defaultDim`, ki določa, ali želimo izvajati meritve s privzetimi vrednostmi velikosti delovnih skupin;
- `demo`, ki določa, ali želimo procesirati sliko brez izvajanja meritev in le za prikaz delovanja ščepcev;
- `iterations`, ki določa število iteracij izvajanja ščepca pri posamezni kombinaciji velikosti dimenzij;
- `localSnn`, ki v primeru, da imamo pod `kernelFile` izbrano datoteko *snn-mean.cl*, določa, ali želimo izvajati ščepec, ki implementira uporabo lokalnega pomnilnika;
- `wDims`, ki prav tako le v primeru, da imamo pod `kernelFile` izbrano datoteko *snn-mean.cl* in `localSnn` nastavljen na vrednost 1, določa seznam z velikostjo prve in druge dimenzije delovnih skupin.

Glede na podane parametre se nato tekom izvajanja programa nastavijo ustrezni medpomnilniški objekti, ustrezni ščepci in njihovi argumenti, večnivojska `for`-zanka pa poskrbi za izvajanje v izbranih kombinacijah velikosti dimenzij delovnih skupin. Ker se nastavljanje argumentov ščepcev izvaja izven zanke, ki skrbi ne le za večkratne iteracije pri istih nastavitvah, temveč tudi pri različnih velikostih prve in druge dimenzije, smo metodo za Gaussovo zameglitev pri merjenju poenostavili na izvajanje z istimi vhodnimi argumenti in le podvojili število iteracij pri posamezni kombinaciji dimenzij, pri čemer se je z vsako liho iteracijo zagnal prvi ščepec, z vsako sodo itera-

cijo pa drugi ščepec iz datoteke *gaussian-blur.cl*. Podoben problem je nastopil pri izvajanju ščepca *snn_mean_local*, ki kot argument `__local float4 *temp_buf` zahteva medpomnlniški objekt v lokalnem pomnilniku, katerega velikost je odvisna od velikosti delovne skupine. Ker gre za parameter, ki ga je potrebno definirati že pred zanko, ki v drugih primerih skrbi za sprehajanje preko različnih kombinacij velikost delovnih skupin, smo dani ščepec testirali za vsako kombinacijo ločeno, pri tem pa posegli po rabi parametra `wDims`.

Da bi pri merjenju časa, potrebnega za izvajanje posamezne metode na OpenCL napravi, dosegli čim večjo natančnost, smo za vsako nastavitev izvedli 1000 meritev in čase v milisekundah zapisali v ločeno datoteko. Pri časovnem beleženju začetka in konca izvajanja smo uporabili OpenCL funkcijo `clGetEventProfilingInfo()` z zastavicama `CL_PROFILING_COMMAND_START` in `CL_PROFILING_COMMAND_END`, ki zabeležita trenutni časovni števec naprave ob trenutku, ko se določen ukaz začne oziroma konča izvajati na napravi.

4.2 Skripte za obdelavo rezultatov

Za interpretacijo pridobljenih meritev sta bili pripravljeni dve skripti, spisani v programskem jeziku Python.

Prva za podano metodo procesiranja slik, podano OpenCL napravo in podano vhodno sliko, iz datotek, v katerih so zabeleženi časi izvajanja 1000 meritev, za posamezno nastavitev izračuna mediano seznama časov in poišče nastavitev, pri kateri ima ta časovna količina najmanjšo vrednost. Rezultat vizualzira s pomočjo knjižnice matplotlib[20] v obliki grafa, ki prikazuje čas izvajanja v odvisnosti od velikosti dimenzij delovnih skupin. Za večjo preglednost skripta ustvari in shrani na disk ločen graf za vsako velikost 3. dimenzije, tudi če tretje dimenzije ni bilo implementirane in je takrat ta obravnavana z vrednostjo 0. Iz posameznega grafa je s tem razvidno gibanje časa trajanja izvajanja pri posameznih kombinacijah 1. in 2. dimenzije za

neko metodo in dano OpenCL napravo pri izbrani procesirani sliki. Ker so razlike med sosednjim vrednostmi ponekod malenkostne, graf vključuje tudi posebej označeno točko, kjer je čas izvajanja najkrajši.

Druga skripta za natančnejši pregled razlik med najboljšimi nastavitvami in rezultati privzetih nastavitev ustvari tabelo ključnih vrednosti za podano metodo procesiranja slik in dano OpenCL napravo. Tudi pri tej skripti se poišče mediana časov izvajanja pri posamezni nastavitvi in izbere najugodnejšo nastavitev glede na najmanjšo vrednost te mediane, le da za razliko od skripte za izris grafov, ta poišče najugodnejšo nastavitev izmed celote možnih nastavitev dimenzij. Poleg tega izvede enako iskanje pri obeh testiranih slikah, med rezultate pa vključi še mediano časov izvajanja pri privzetih nastavitvah za dano metodo na dani napravi. Rezultat je tabela, spisana v LaTex sintaksi in shranjena v tekstovno datoteko, od koder jo je mogoče enostavno vključiti v dano nalogo.

4.3 Rezultati meritev

Meritve smo ločeno izvajali na grafični enoti HD Graphics 4000 (v nadaljevanju GPE), ki vključje 16 izvajalnih enot, in procesorski enoti Intel® Core™ i5-3210M CPU @ 2.50GHz (v nadaljevanju CPE), ki vključuje 2 procesorski jedri.

S funkcijo `clGetDeviceInfo()` smo za obe napravi naredili poizvedbo po specifikah glede dovoljenih velikosti delovnih entitet. Z zastavico `CL_DEVICE_MAX_WORK_ITEM_SIZES`, ki nam pove maksimalno globalno število delovnih elementov v vsaki dimenziji, smo za GPE pri vseh metodah dobili rezultate za 1., 2. in 3. dimenzijo v vrednostih: 512, 512, 512; medtem ko CPE v 1. dimenziji dovoli največ 1024 elementov, v 2. in 3. pa le 1. Podobno uporaba zastavice `CL_DEVICE_MAX_WORK_GROUP_SIZE` poda informacijo o tem, kolikšno je največje številno delovnih elementov znotraj posamezne delovne skupine, ki se izvaja v podatkovnem paralelizmu z drugimi delovnimi skupinami. Pri GPE je to največ 512, CPE pa v vsako delovno skupino

velikost slike (št. pikslov)	najboljši čas (ms)	dimenzije z najboljšim časom	čas pri privzetih nastavitvah (ms)
512×512	0.872	64, 1	0.873
2048×1024	6.646	64, 1	6.675

Tabela 4.1: **Odstranjevanje rdečih oči**: primerjava najugodnejših nastavitev velikosti delovnih skupin (dimenzij) s časom izvajanja pri privzetih nastavitvah na CPE

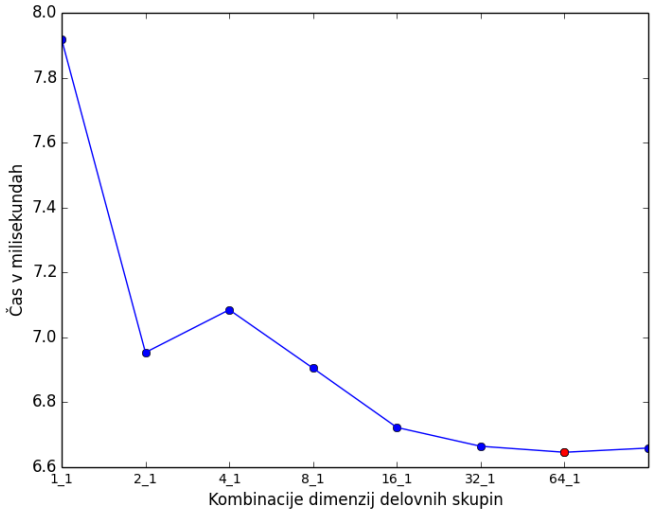
sprejme največ 1024 elementov.

4.3.1 Odstranjevanje rdečih oči

Za vsakega od ščepcev smo izvedli poizvedbo o priporočeni velikosti delovnih skupin, oziroma večkratniku priporočene velikosti s funkcijo `clGetKernelWorkGroupInfo()` in zastavico `CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE`. Na GPE je bila pri ščepcu *cl_red_eye_removal* vrnjena vrednost 16, na CPE pa 1. Ob zastavici `CL_KERNEL_WORK_GROUP_SIZE` smo dodatno izvedeli največjo velikost delovne skupine pri določeni napravi in danem ščepcu. Za CPE je bila ta velikost 128, pri GPE pa se je kot največje število delovnih elementov v skupini izkazala vrednost 512.

Tabela 4.1 prikazuje razlike med časom izvajanja pri privzetih nastavitvah velikosti delovnih skupin in najkrajšim časom izvajanja, ki smo ga dosegli s testiranjem različnih kombinacij velikosti delovnih skupin na CPE. Pri tem smo lahko glede na ugotovljene omejitve velikosti skupin kombinirali le različne velikosti 1. dimenzije z velikostjo 1 v 2. dimenziji. Poleg tega je bila 1. dimenzija lahko le v velikosti potenc števila 2 in manjša ali enaka številu 128.

Slika 4.3 prikazuje odvisnost časa izvajanja ščepca od velikosti 1. dimenzije delovnih skupin na CPE za večjo od testiranih slik. Rezultati pri več kot polovico manjši sliki so prikazani na grafu pod sliko 4.4. V trendu gibanja

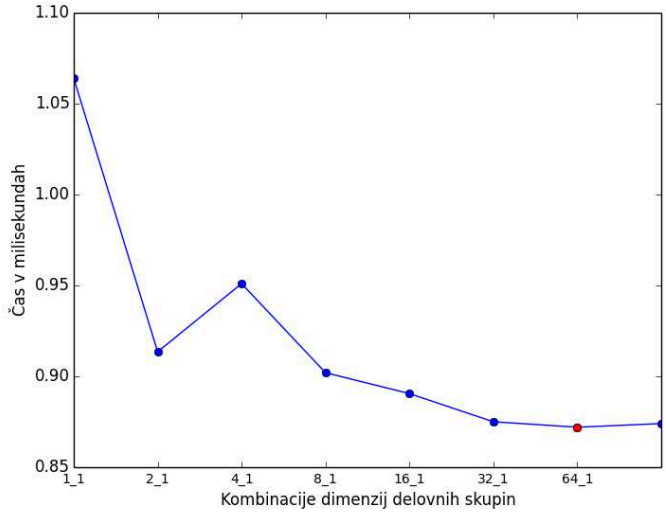


Slika 4.3: Čas izvajanja metode za odstranjevanje rdečih oči na CPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 2048×1024.

časa izvajanja pri eni in drugi ni razvidnih večjih razlik, zanimiv pa je razkorak med časom pri velikostih skupin 2 in 4, saj se čas izvajanja krajša pri večanju števila delovnih skupin do minimuma, ki ga doseže pri velikost 64, z izjemo povečanja potrebnega časa izvajanja pri velikosti 4.

Tabela 4.2 prikazuje razlike med časom izvajanja pri privzetih nastavitvah velikosti delovnih skupin in najkrajšim časom izvajanja, ki smo ga dosegli s testiranjem različnih kombinacij velikosti delovnih skupin še na GPE. Tu smo lahko kombinirali velikosti vseh treh dimenzij, pri čemer se je izkazala raba 3. dimenzije pri dani implementaciji ščepca manj učinkovita. Za boljšo izrabo 3. dimenzije bi bilo potrebno ščepec predelati, kar ostaja kot možnost izboljšave pri nadaljnjih testiranjih.

Glede na ugotovljeno maksimalno število delovnih elementov v skupini velikosti skupin po dimenzijah v skupnem produktu niso smele presegati 512. Ker smo kot globalno velikost 3. dimenzije podali vrednost 4, ki je dolžina zaporednja števil v plavajoči vejici za določanje enega piksla, je bila tudi lokalna velikost delovne skupine v 3. dimenziji največ 4. Slika 4.5 prikazuje



Slika 4.4: Čas izvajanja metode za odstranjevanje rdečih oči na CPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 512×512.

velikost slike (št. pikslov)	najboljši čas (ms)	dimenzije z najboljšim časom	čas pri privzetih nastavitvah (ms)
512×512	0.442	128, 1	0.519
2048×1024	4.397	16, 1	4.423

Tabela 4.2: **Odstranjevanje rdečih oči**: primerjava najugodnejših nastavitev velikosti delovnih skupin (dimenzij) s časom izvajanja pri privzetih nastavitvah na GPE

trend gibanja časa izvajanja glede na kombinacijo velikosti dimenzij[2] v delovnih skupinah pri implementaciji brez vključevanja 3. dimenzije za večjo od testiranih slik, slika 4.6 prikazuje enako razmerje[3] za manjšo od testiranih slik, slike 4.7[4], 4.8[5] in 4.9[6] pa za primerjavo še razmerja med časom in kombinacijami velikosti skupin pri vključitvi 3. dimenzije.

Pri primerjavi rezultatov časov med kombinacijami velikosti skupin, kjer je 3. dimenzija velikosti 1; skupin, kjer je 3. dimenzija velikosti 2 in skupin, kjer je 3. dimenzija veliksoti 4, opazimo, da velja, kot pri različnih velikostih 1. dimenzije, da se časi izvajanja do neke meje izboljšujejo pri povečevanju števila delovnih elementov na dimenzijo. Pri 3. dimenziji z velikostjo 4 so rezultati v povprečju boljši od rezultatov pri 3. dimenziji z velikostjo 2, medtem ko so rezultati pri 3. dimenziji z velikostjo 1 najslabši.

---

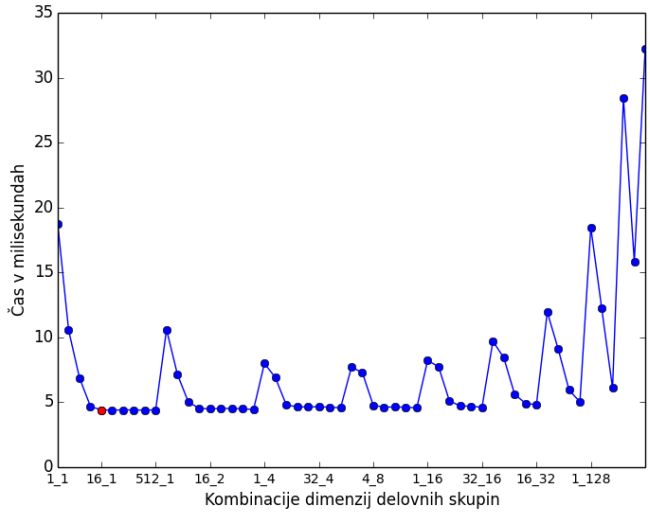
[2]Kombinacije, kot si sledijo na ordinatni osi: 1_1, 2_1, 4_1, 8_1, 16_1, 32_1, 64_1, 128_1, 256_1, 512_1, 1_2, 2_2, 4_2, 8_2, 16_2, 32_2, 64_2, 128_2, 256_2, 1_4, 2_4, 4_4, 8_4, 16_4, 32_4, 64_4, 128_4, 1_8, 2_8, 4_8, 8_8, 16_8, 32_8, 64_8, 1_16, 2_16, 4_16, 8_16, 16_16, 32_16, 1_32, 2_32, 4_32, 8_32, 16_32, 1_64, 2_64, 4_64, 8_64, 1_128, 2_128, 4_128, 1_256, 2_256, 1_512.

[3]Kombinacije, kot si sledijo na ordinatni osi: 1_1, 2_1, 4_1, 8_1, 16_1, 32_1, 64_1, 128_1, 256_1, 512_1, 1_2, 2_2, 4_2, 8_2, 16_2, 32_2, 64_2, 128_2, 256_2, 1_4, 2_4, 4_4, 8_4, 16_4, 32_4, 64_4, 128_4, 1_8, 2_8, 4_8, 8_8, 16_8, 32_8, 64_8, 1_16, 2_16, 4_16, 8_16, 16_16, 32_16, 1_32, 2_32, 4_32, 8_32, 16_32, 1_64, 2_64, 4_64, 8_64, 1_128, 2_128, 4_128, 1_256, 2_256, 1_512.

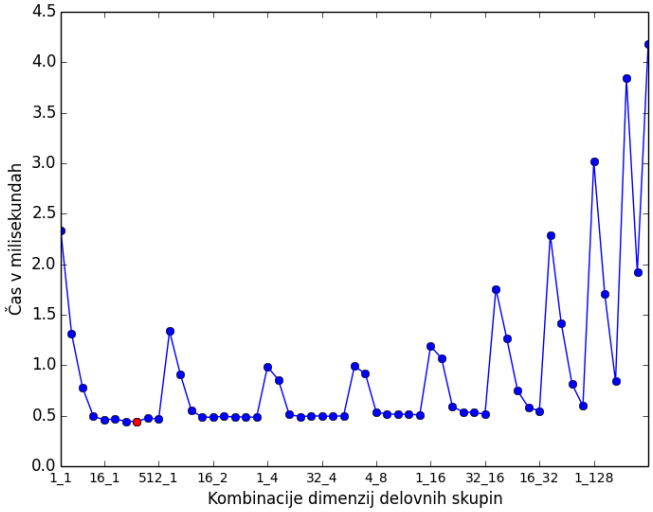
[4]Kombinacije, kot si sledijo na ordinatni osi: 1_1_1, 2_1_1, 4_1_1, 8_1_1, 16_1_1, 32_1_1, 64_1_1, 128_1_1, 256_1_1, 512_1_1, 1_2_1, 2_2_1, 4_2_1, 8_2_1, 16_2_1, 32_2_1, 64_2_1, 128_2_1, 256_2_1, 1_4_1, 2_4_1, 4_4_1, 8_4_1, 16_4_1, 32_4_1, 64_4_1, 128_4_1, 1_8_1, 2_8_1, 4_8_1, 8_8_1, 16_8_1, 32_8_1, 64_8_1, 1_16_1, 2_16_1, 4_16_1, 8_16_1, 16_16_1, 32_16_1, 1_32_1, 2_32_1, 4_32_1, 8_32_1, 16_32_1, 1_64_1, 2_64_1, 4_64_1, 8_64_1, 1_128_1, 2_128_1, 4_128_1, 1_256_1, 2_256_1, 1_512_1.

[5]Kombinacije, kot si sledijo na ordinatni osi: 1_1_2, 2_1_2, 4_1_2, 8_1_2, 16_1_2, 32_1_2, 64_1_2, 128_1_2, 256_1_2, 1_2_2, 2_2_2, 4_2_2, 8_2_2, 16_2_2, 32_2_2, 64_2_2, 128_2_2, 1_4_2, 2_4_2, 4_4_2, 8_4_2, 16_4_2, 32_4_2, 64_4_2, 1_8_2, 2_8_2, 4_8_2, 8_8_2, 16_8_2, 32_8_2, 1_16_2, 2_16_2, 4_16_2, 8_16_2, 16_16_2, 1_32_2, 2_32_2, 4_32_2, 8_32_2, 1_64_2, 2_64_2, 4_64_2, 1_128_2, 2_128_2, 1_256_2.

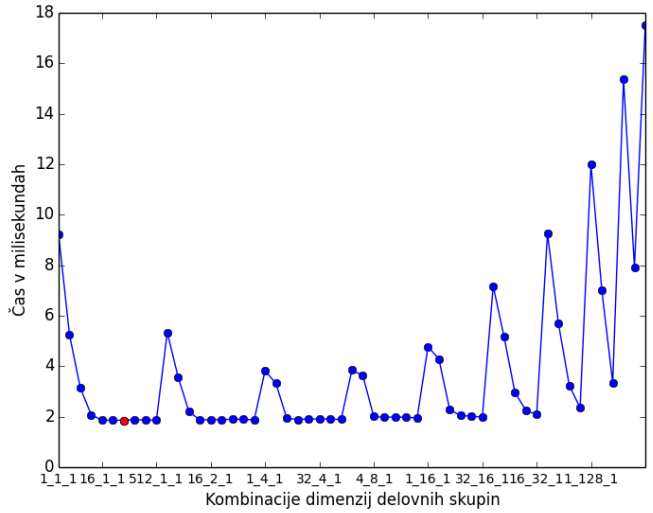
[6]Kombinacije, kot si sledijo na ordinatni osi: 1_1_4, 2_1_4, 4_1_4, 8_1_4, 16_1_4, 32_1_4, 64_1_4, 128_1_4, 1_2_4, 2_2_4, 4_2_4, 8_2_4, 16_2_4, 32_2_4, 64_2_4, 1_4_4, 2_4_4, 4_4_4, 8_4_4, 16_4_4, 32_4_4, 1_8_4, 2_8_4, 4_8_4, 8_8_4, 16_8_4, 1_16_4, 2_16_4, 4_16_4, 8_16_4, 1_32_4, 2_32_4, 4_32_4, 1_64_4, 2_64_4, 1_128_4.



Slika 4.5: Čas izvajanja metode za odstranjevanje rdečih oči na GPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 2048×1024.



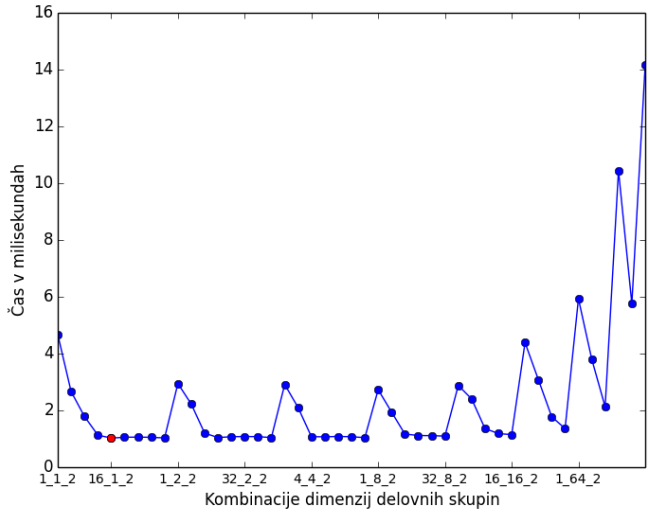
Slika 4.6: Čas izvajanja metode za odstranjevanje rdečih oči na GPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 512×512.



Slika 4.7: Čas izvajanja metode za odstranjevanje rdečih oči na GPE pri nastavitvah delovnih skupin s 3. dimenzijo velikosti 1 na sliki velikosti 512×512.

Podobno je opaziti, da se tako pri vključitvi 3. dimenzije, kot pri kombinacijh z zgolj 1. in 2. dimenzijo, časi izboljšujejo do točke, kjer je velikost 1. dimenzije 16, nakar se ustalijo na približno istem nivoju, z manjšimi izboljšavami ali poslabšavami. Rezultati zato tu niso konsistentni; najboljši čas na manjši sliki je bil izmerjen pri kombinaciji, kjer je velikost prve dimenzije 128, na večji pa pri velikosti 1. dimenzije 16. Podobne diskrepance so vidne tudi pri kombinacijah z vključitvijo 3. dimenzije.

Za razliko od opaženega trenda izboljšav pri večjih velikostih 1. in 3. dimenzije pa pri 2. dimenziji velja, da so rezultati skoraj brez izjem boljši pri velikosti 1 in se z večanjem števila delovnih elementov v tej dimenziji le poslabšujejo.

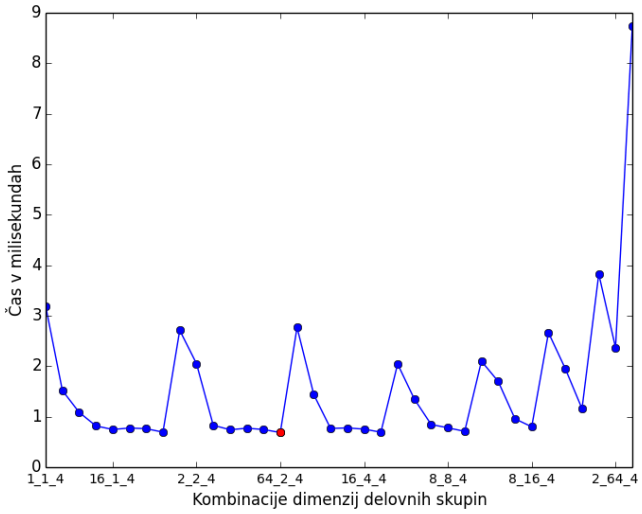


Slika 4.8: Čas izvajanja metode za odstranjevanje rdečih oči na GPE pri nastavitvah delovnih skupin s 3. dimenzijo velikosti 2 na sliki velikosti 512×512.

4.3.2 Zameglitev s povprečjem skupnih najbližjih sosedov

Izvorni ščepec pri zameglitvi s povprečjem skupnih najbližjih sosedov lahko sprejme največ 128 delovnih elementov na skupino pri CPE in največ 512 elementov na skupino pri GPE. Priporočena velikost pri CPE je 1 in pri GPE 16, kar je enako kot pri metodi za odstranjevanje rdečih oči.

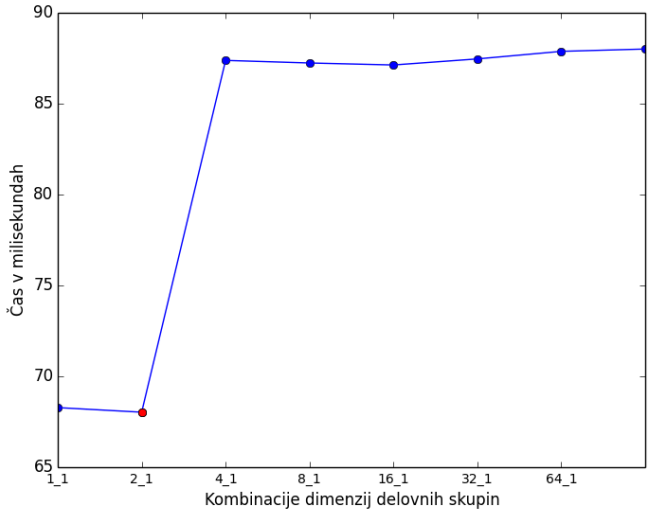
Pri testiranju ni bilo večjih razlik med rezultati izvajanj na manjši in večji sliki. Kot opredeljeno v tabeli 4.3, kjer so primerjani rezultati izvajanj na CPE, se je kot najboljša nastavitev za obe sliki izkazala kombinacija 2×1, kjer je 2 velikost 1. dimenzije in 1 velikost 2. dimenzije. Slika 4.10 prikazuje odvisnost časa izvajanja od velikosti 1. dimenzije kot edine, ki jo je bilo pri dani napravi mogoče spreminjati. Opazen je silovit skok med hitrostjo izvajanja pri velikostih 1 in 2 ter kombinacijami, pri kateih je velikost 1. dimenzije večja ali enaka 4.



Slika 4.9: Čas izvajanja metode za odstranjevanje rdečih oči na GPE pri nastavitvah delovnih skupin s 3. dimenzijo velikosti 4 na sliki velikosti 512×512.

velikost slike (št. pikslov)	najboljši čas (ms)	dimenzije z najboljšim časom	čas pri privzetih nastavitvah (ms)
512×512	68.0265	2, 1	86.3395
2048×1024	526.069	2, 1	684.6555

Tabela 4.3: **Zameglitev s povprečjem skupnih najbližjih sosedov**: primerjava najugodnejših nastavitev velikosti delovnih skupin (dimenzij) s časom izvajanja pri privzetih nastavitvah na CPE



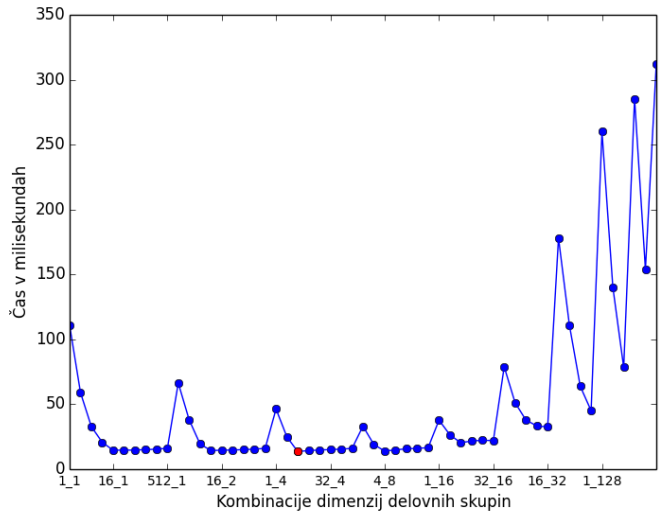
Slika 4.10: Čas izvajanja metode za zameglitev s povprečjem skupnih najbližjih sosedov na CPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 512×512.

Pri izvajanju na GPE se je kot najugodnejša nastavitev pri obeh slikah izkazala kombinacija 4×4, implementacija brez vključitve 3. dimenzije pa ponovno v splošnem hitrejša. V tabeli 4.4 so predstavljene razlike med časi ob privzetih nastavitvah in najugodnejših časih za obe sliki. Slika 4.11[7] prikazuje odvisnost časov od kombinacij 1. in 2. dimenzije, kjer so opazni lokalni minimumi časov trajanja pri kombinacijah z velikostjo 1. dimenzije večjo ali enako 4.

[7]Kombinacije, kot si sledijo na ordinatni osi: 1_1, 2_1, 4_1, 8_1, 16_1, 32_1, 64_1, 128_1, 256_1, 512_1, 1_2, 2_2, 4_2, 8_2, 16_2, 32_2, 64_2, 128_2, 256_2, 1_4, 2_4, 4_4, 8_4, 16_4, 32_4, 64_4, 128_4, 1_8, 2_8, 4_8, 8_8, 16_8, 32_8, 64_8, 1_16, 2_16, 4_16, 8_16, 16_16, 32_16, 1_32, 2_32, 4_32, 8_32, 16_32, 1_64, 2_64, 4_64, 8_64, 1_128, 2_128, 4_128, 1_256, 2_256, 1_512.

velikost slike (št. pikslov)	najboljši čas (ms)	dimenzije z najboljšim časom	čas pri privzetih nastavitvah (ms)
512×512	13.568	4, 4	16.1345
2048×1024	109.695	4, 4	135.8525

Tabela 4.4: **Zameglitev s povprečjem skupnih najbližjih sosedov**: primerjava najugodnejših nastavitev velikosti delovnih skupin (dimenzij) s časom izvajanja pri privzetih nastavitvah na GPE



Slika 4.11: Čas izvajanja metode za zameglitev s povprečjem skupnih najbližjih sosedov na GPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 512×512.

V primerjavo implementacijam brez vključitve 3. dimenzije slike 4.12[8], 4.13[9] in 4.14[10] prikazujejo odvisnosti časov izvajanja od kombinacij 1. in 2. dimenzije, kjer je velikost 3. dimenzije 1, 2 in 4. Za razliko od predhodne metode pri tej ni večjih časovnih odstopanj, ki bi jih povzročalo povečevanje velikosti skupine v 3. dimenziji. Najboljši čas je pri vseh treh 4 do 4,2-krat večji od najboljšega časa pri kombinaciji brez 3. dimenzije.

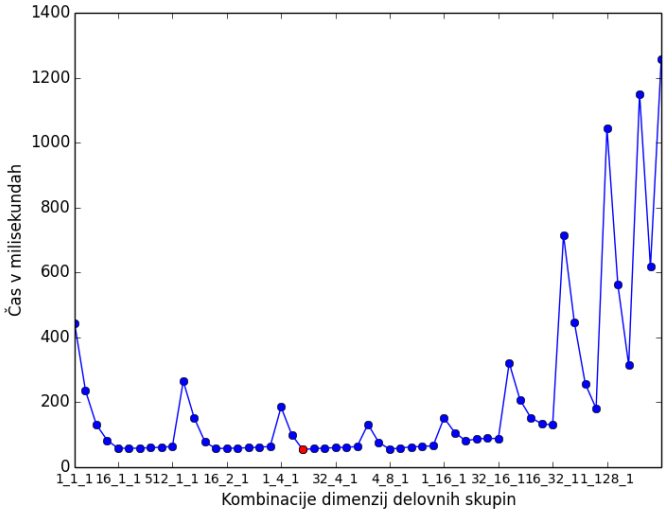
Predelava ščepca za zameglitev s povprečjem skupnih najbližjih sosedov za implementacijo lokalnega pomnilnika naprave se je pri ustreznih nastavitvah izkazala za optimalnejšo, ob manj ugodnih nastavitvah pa za časovno mnogo bolj potratno. Tabela 4.5 podrobneje predstavlja razmerja med najboljšimi nastavitvami in časi pri implementaciji brez in z lokalnim pomnilnikom, tabela 4.6 pa predstavlja tudi razliko med najslabšimi nastavitvami in časi pri obeh različicah.

Opazimo, da je pri rabi lokalnega pomnilnika najugodnejša nastavitev z dvema dimenzijama v velikosti 16×16, medtem ko je najslabši rezultat dosežen pri najmanjših velikostih obeh dimenzij. Razlog za počasnejše procesiranje pri manjših skupinah je v potrebi po kopiranju večjih količin dodatnih podatkov v lokalni pomnilnik, saj vsak delovni element potrebuje podatke $(2 * radius)^2 - 1$ pikslov poleg sebi istoležečega. Pri večjih skupinah

[8]Kombinacije, kot si sledijo na ordinatni osi: 1_1_1, 2_1_1, 4_1_1, 8_1_1, 16_1_1, 32_1_1, 64_1_1, 128_1_1, 256_1_1, 512_1_1, 1_2_1, 2_2_1, 4_2_1, 8_2_1, 16_2_1, 32_2_1, 64_2_1, 128_2_1, 256_2_1, 1_4_1, 2_4_1, 4_4_1, 8_4_1, 16_4_1, 32_4_1, 64_4_1, 128_4_1, 1_8_1, 2_8_1, 4_8_1, 8_8_1, 16_8_1, 32_8_1, 64_8_1, 1_16_1, 2_16_1, 4_16_1, 8_16_1, 16_16_1, 32_16_1, 1_32_1, 2_32_1, 4_32_1, 8_32_1, 16_32_1, 1_64_1, 2_64_1, 4_64_1, 8_64_1, 1_128_1, 2_128_1, 4_128_1, 1_256_1, 2_256_1, 1_512_1.

[9]Kombinacije, kot si sledijo na ordinatni osi: 1_1_2, 2_1_2, 4_1_2, 8_1_2, 16_1_2, 32_1_2, 64_1_2, 128_1_2, 256_1_2, 1_2_2, 2_2_2, 4_2_2, 8_2_2, 16_2_2, 32_2_2, 64_2_2, 128_2_2, 1_4_2, 2_4_2, 4_4_2, 8_4_2, 16_4_2, 32_4_2, 64_4_2, 1_8_2, 2_8_2, 4_8_2, 8_8_2, 16_8_2, 32_8_2, 1_16_2, 2_16_2, 4_16_2, 8_16_2, 16_16_2, 1_32_2, 2_32_2, 4_32_2, 8_32_2, 1_64_2, 2_64_2, 4_64_2, 1_128_2, 2_128_2, 1_256_2.

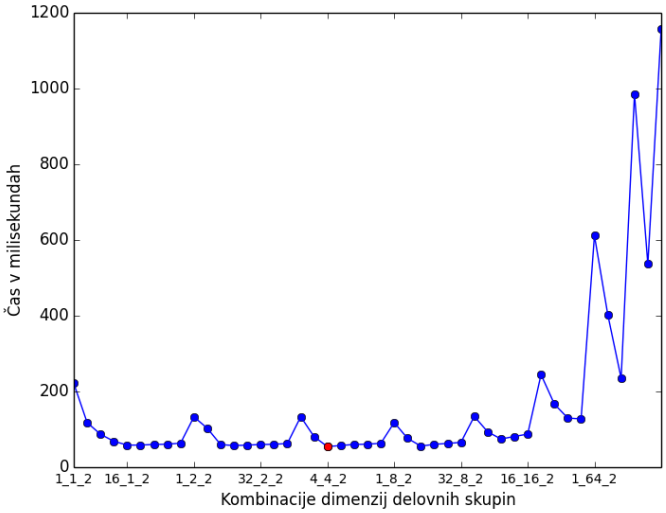
[10]Kombinacije, kot si sledijo na ordinatni osi: 1_1_4, 2_1_4, 4_1_4, 8_1_4, 16_1_4, 32_1_4, 64_1_4, 128_1_4, 1_2_4, 2_2_4, 4_2_4, 8_2_4, 16_2_4, 32_2_4, 64_2_4, 1_4_4, 2_4_4, 4_4_4, 8_4_4, 16_4_4, 32_4_4, 1_8_4, 2_8_4, 4_8_4, 8_8_4, 16_8_4, 1_16_4, 2_16_4, 4_16_4, 8_16_4, 1_32_4, 2_32_4, 4_32_4, 1_64_4, 2_64_4, 1_128_4.



Slika 4.12: Čas izvajanja metode za zameglitev s povprečjem skupnih najbližjih sosedov na GPE pri nastavitvah delovnih skupin s 3. dimenzijo velikosti 1 na sliki velikosti 512×512.

velikost slike (št. pikslov)	najboljši čas (ms) brez LP	najboljše dimenzije brez LP	najboljši čas (ms) z LP	najboljše dimenzije z LP
512×512	13.568	4, 4	8.1645	16, 16
2048×1024	109.695	4, 4	64.9035	16, 16

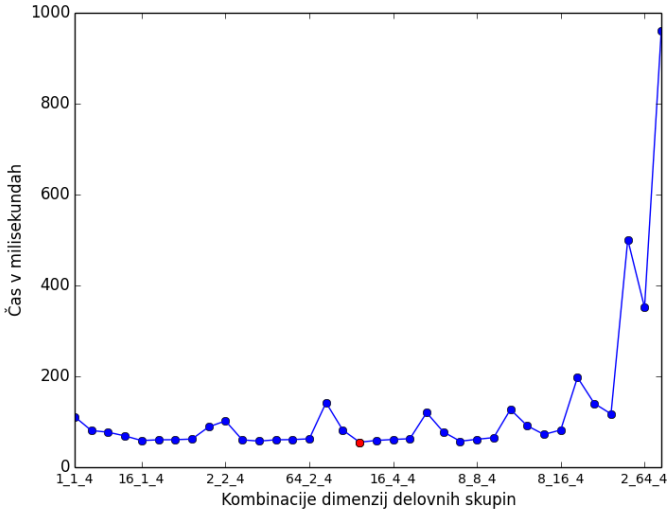
Tabela 4.5: **Zameglitev s povprečjem skupnih najbližjih sosedov**: primerjava najugodnejših nastavitev velikosti delovnih skupin (dimenzij) in časov izvajanja pri implementaciji brez ter z lokalnim pomnilnikom (krajšano LP) na GPE



Slika 4.13: Čas izvajanja metode za zameglitev s povprečjem skupnih najbližjih sosedov na GPE pri nastavitvah delovnih skupin s 3. dimenzijo velikosti 2 na sliki velikosti 512×512.

velikost slike (št. pikslov)	najslabši čas (ms) brez LP	najslabše dimenzije (v 2D prostoru) brez LP	najslabši čas (ms) z LP	najslabše dimenzije (v 2D prostoru) z LP
512×512	312.214	1, 512	574.376	1, 1
2048×1024	1853.769	1, 512	2313.986	2, 1

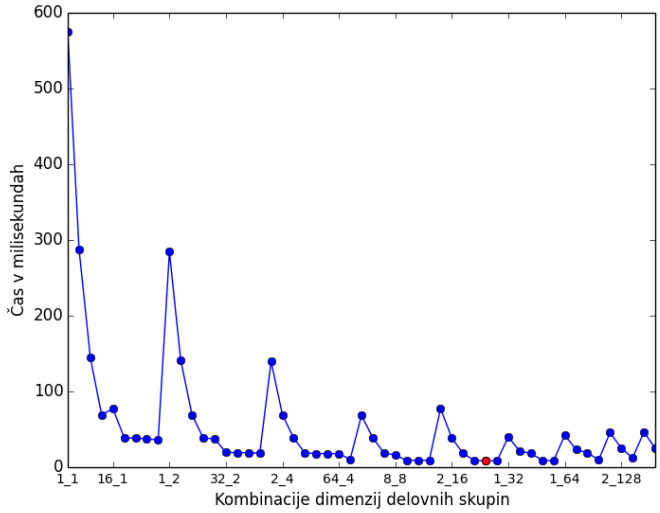
Tabela 4.6: **Zameglitev s povprečjem skupnih najbližjih sosedov**: primerjava najslabših nastavitev velikosti delovnih skupin (dimenzij) ter časov izvajanja pri implementaciji brez 3. dimenzije in brez ter z lokalnim pomnilnikom (krajšano LP) na GPE



Slika 4.14: Čas izvajanja metode za zameglitev s povprečjem skupnih najbližjih sosedov na GPE pri nastavitvah delovnih skupin s 3. dimenzijo velikosti 4 na sliki velikosti 512×512.

za večji del teh podatkov poskrbijo sosednji delovni elementi, medtem ko pri manjših skupinah neugodno razmerje med elementi, ki prenašajo samo podatke sebi istoležečih pikslov, in elementi, ki poleg tega prenašajo še več dodatnih pikslov, povzroča, da vsak delovni element opravi znatno več dela kot bi ga zgolj z dostopanjem do globalnega pomnilnika. Na sliki 4.15[11] je tako opazna porast v trajanju izvajanja pri kombinacijah z majhnimi velikostmi dimenzij in krajšanje časa izvajanja pri večjih delovnih skupinah, kar je ravno nasprotno situaciji pri implementaciji metode brez rabe lokalnega pomnilnika. Ker smo metode v vseh primerih izvajali pri `radius` vrednosti 5, je kombinacija 16×16 prva, pri kateri obstajajo delovni elementi, ki jim ni potrebno prenašati dodatnih podatkov. S tem pride hitrost dostopanja

[11]Kombinacije, kot si sledijo na ordinatni osi: 1_1, 2_1, 4_1, 8_1, 16_1, 32_1, 64_1, 128_1, 256_1, 1_2, 2_2, 4_2, 8_2, 16_2, 32_2, 64_2, 128_2, 256_2, 1_4, 2_4, 4_4, 8_4, 16_4, 32_4, 64_4, 128_4, 1_8, 2_8, 4_8, 8_8, 16_8, 32_8, 64_8, 1_16, 2_16, 4_16, 8_16, 16_16, 32_16, 1_32, 2_32, 4_32, 8_32, 16_32, 1_64, 2_64, 4_64, 8_64, 1_128, 2_128, 4_128, 1_256, 2_256.



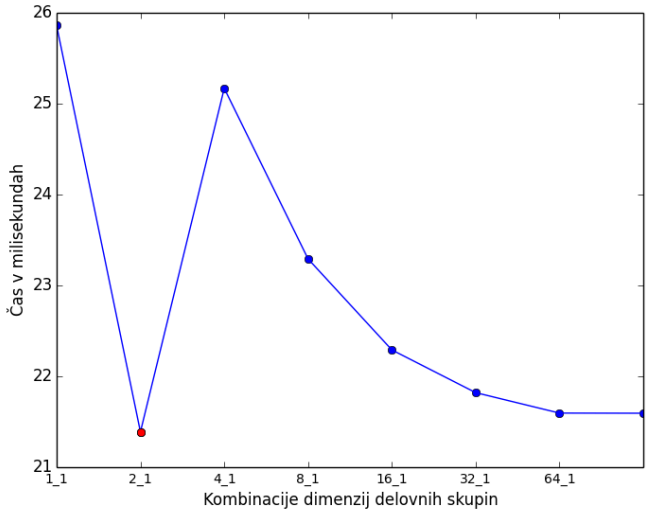
Slika 4.15: Čas izvajanja metode za zameglitev s povprečjem skupnih najbližjih sosedov z uporabo lokalnega pomnilnika na GPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 512×512.

do lokalnega pomnilnika toliko bolj do izraza in omogoča kar 40-odstotno izboljšavo v času izvajanja.

4.3.3 Gaussova zameglitev

Pri obeh ščepcih metode za Gaussovo zameglitev je enako kot pri prejšnjih dveh metodah priporočena velikost delovne skupine, ki jo vrne CPE v vlogi OpenCL naprave, 1, medtem ko je pri GPE v vrednosti 16. Največja velikost, ki jo ščepca še sprejmeta za delovno skupino, je pri CPE 128 in pri GPE 512.

Pri izvajanju na CPE je, podobno kot pri metodi za odstranjevanje rdečih oči, tudi pri Gaussovi zameglitvi najslabši čas dosežen pri kombinaciji 1. in 2. dimenzije z obema v velikosti 1, nakar se čas sunkovito izboljša pri kombinaciji s 1. dimenzijo velikosti 2, ponovno narase pri vrednosti 1. dimenzije 4 in nato počasi pada. Sliki 4.16 in 4.17 prikazujeta povečevanje in padanje časa izvajanja pri različnih kombinacijah na CPE za večjo in manjšo od

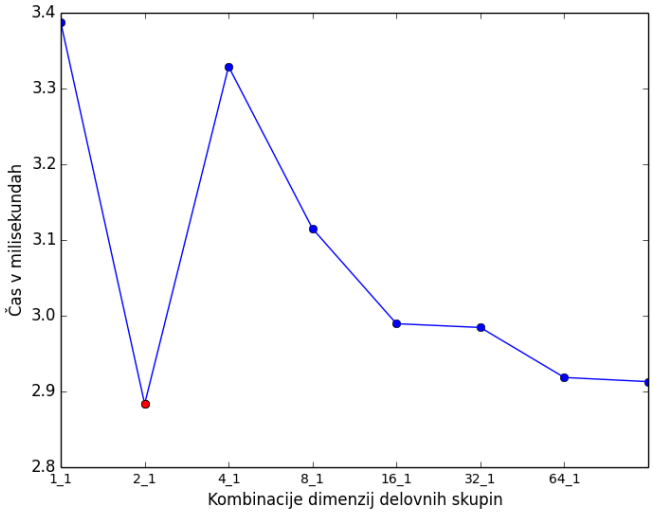


Slika 4.16: Čas izvajanja metode za Gaussovo zameglitev na CPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 2048×1024.

testiranih slik, pri čemer med obema ni zaznati večjih odstopanj. Izjema je manj regularno izboljšanje časa pri prehodu med velikostima 1. dimenzije 16 in 32.

Kot razvidno iz obeh slik, za razliko od metode za odstranjevanje rdečih oči najkrajši čas izvajanja ni dosežen pri večjih velikostih 1. dimenzije, temveč že pri velikosti 2. Tabela 4.7 podrobneje predstavi vrednosti najkrajših časov izvajanja ob primerjavi z rezultati pri privzetih nastavitvah.

Pri izvajanju metode na GPE so rezultati manj predvidljivi. Kot je razvidno iz tabele 4.8, najboljši čas pri manjši sliki ni izmerjen pri enakih nastavitvah kot pri večji sliki. Podobno kot pri metodi za odstranjevanje rdečih oči je tudi tu povprečje časov izvajanja boljše pri kombinacijah brez 3. dimenzije, le da rezultati z vključitvijo 3. dimenzije v povprečju niso toliko slabši.



Slika 4.17: Čas izvajanja metode za Gaussovo zameglitev na CPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 512×512.

velikost slike (št. pikslov)	najboljši čas (ms)	dimenzije z najboljšim časom	čas pri privzetih nastavitvah (ms)
512×512	2.8835	2, 1	2.89
2048×1024	21.3885	2, 1	21.5675

Tabela 4.7: **Gaussova zameglitev**: primerjava najugodnejših nastavitev velikosti delovnih skupin (dimenzij) s časom izvajanja pri privzetih nastavitvah na CPE

velikost slike (št. pikslov)	najboljši čas (ms)	dimenzije z najboljšim časom	čas pri privzetih nastavitvah (ms)
512×512	2.33	16, 1	2.372
2048×1024	18.5995	16, 2	19.21

Tabela 4.8: **Gaussova zameglitev**: primerjava najugodnejših nastavitev velikosti delovnih skupin (dimenzij) s časom izvajanja pri privzetih nastavitvah na GPE

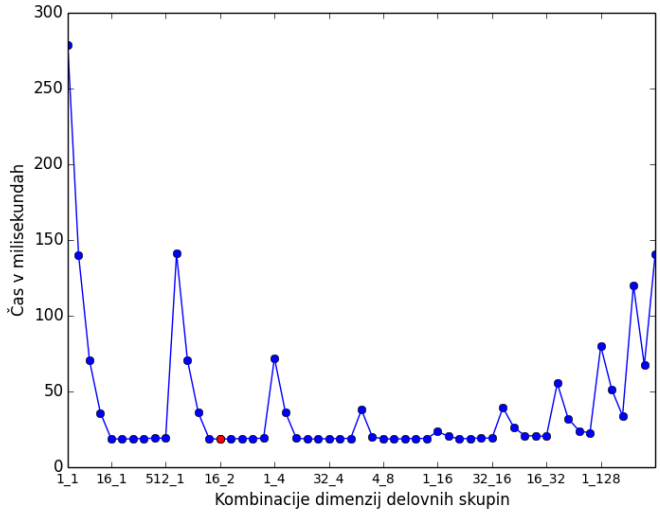
Ob primerjavi slik 4.18[12] in 4.19[13], ki prikazujeta razmerja med časom in velikostmi skupin brez 3. dimenzije za večjo in manjšo sliko na eni strani in sklopa slik 4.20[14] na drugi strani, kjer je bila v kombinacijah vpletena še 3. dimenzija, je povprečni čas izvajanja vidno slabši le, kjer je velikost 3. dimenzije enaka 2.

Zanimivo je, da so pri testih na manjši od slik najboljši rezultati izmerjeni pri manjših velikostih 2. dimenzije, bodisi pri vrednosti 1, bodisi pri vrednosti 2 in le izjemoma pri vrednosti 4. Po drugi strani so pri večji sliki najboljši rezultati za posamezno vrednost 3. dimenzije pri velikostih 2. dimenzije 4 ali 8 in le izjemoma 2.

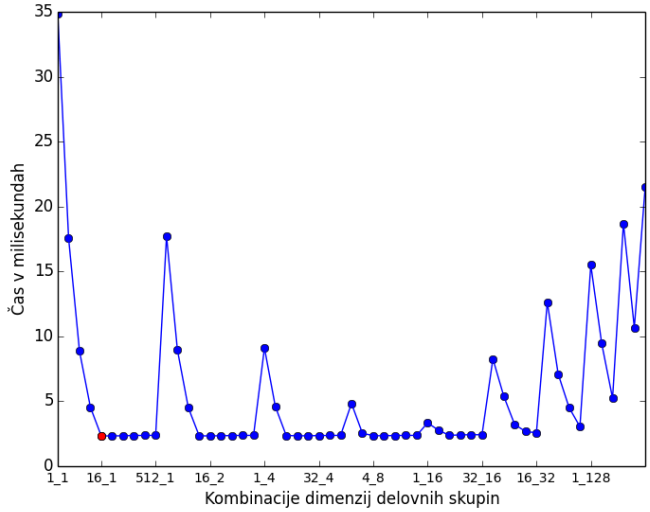
[12]Kombinacije, kot si sledijo na ordinatni osi: 1_1, 2_1, 4_1, 8_1, 16_1, 32_1, 64_1, 128_1, 256_1, 512_1, 1_2, 2_2, 4_2, 8_2, 16_2, 32_2, 64_2, 128_2, 256_2, 1_4, 2_4, 4_4, 8_4, 16_4, 32_4, 64_4, 128_4, 1_8, 2_8, 4_8, 8_8, 16_8, 32_8, 64_8, 1_16, 2_16, 4_16, 8_16, 16_16, 32_16, 1_32, 2_32, 4_32, 8_32, 16_32, 1_64, 2_64, 4_64, 8_64, 1_128, 2_128, 4_128, 1_256, 2_256, 1_512.

[13]Kombinacije, kot si sledijo na ordinatni osi: 1_1, 2_1, 4_1, 8_1, 16_1, 32_1, 64_1, 128_1, 256_1, 512_1, 1_2, 2_2, 4_2, 8_2, 16_2, 32_2, 64_2, 128_2, 256_2, 1_4, 2_4, 4_4, 8_4, 16_4, 32_4, 64_4, 128_4, 1_8, 2_8, 4_8, 8_8, 16_8, 32_8, 64_8, 1_16, 2_16, 4_16, 8_16, 16_16, 32_16, 1_32, 2_32, 4_32, 8_32, 16_32, 1_64, 2_64, 4_64, 8_64, 1_128, 2_128, 4_128, 1_256, 2_256, 1_512.

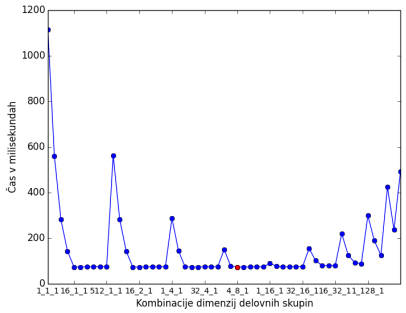
[14]Kombinacije na ordinatnih oseh so v enakem vrstnem redu za posamezno velikost tretje dimenzije kot pri grafih za metodo odstranjevanja rdečih oči.



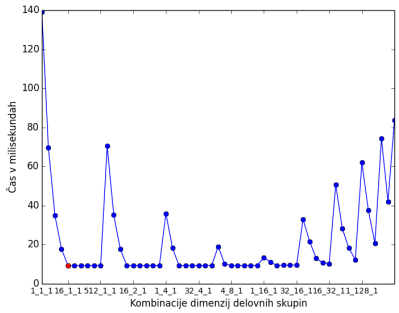
Slika 4.18: Čas izvajanja metode za Gaussovo zameglitev na GPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 2048×1024.



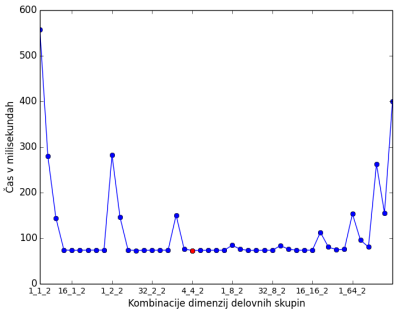
Slika 4.19: Čas izvajanja metode za Gaussovo zameglitev na GPE pri nastavitvah s 1. in 2. dimenzijo delovnih skupin na sliki velikosti 512×512.



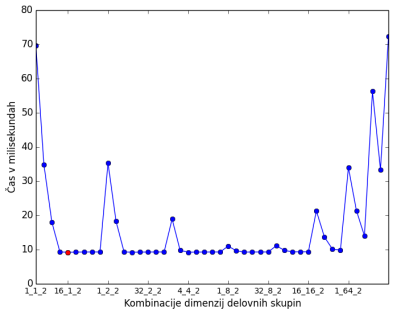
(a) Nastavitve s 3. dimenzijo velikosti 1 na sliki velikosti 2048×1024.



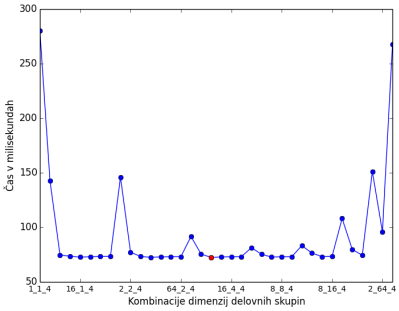
(b) Nastavitve s 3. dimenzijo velikosti 1 na sliki velikosti 512×512.



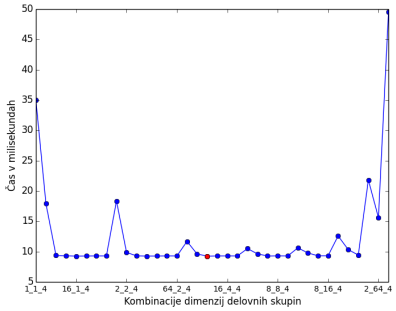
(c) Nastavitve s 3. dimenzijo velikosti 2 na sliki velikosti 2048×1024.



(d) Nastavitve s 3. dimenzijo velikosti 2 na sliki velikosti 512×512.



(e) Nastavitve s 3. dimenzijo velikosti 4 na sliki velikosti 2048×1024.



(f) Nastavitve s 3. dimenzijo velikosti 4 na sliki velikosti 512×512.

Slika 4.20: Čas izvajanja metode za Gaussovo zameglitev na GPE pri vključitvi 3. dimenzije.

Poglavje 5

Sklepne ugotovitve

Testiranje posameznih ščepcev pri različnih kombinacijah velikosti dimenzij delovnih skupin je pokazalo na pomembnost pravilno izbranih dimenzij. Privzete nastavitve so se izkazale za malenkostno počasnejše, razen v primeru izvanjanja ščepca za zameglitev s povprečjem skupnih najbližjih sosedov z lokalnim pomnilnikom, kjer je bila naša implementacija skoraj dvakrat hitrejša. V primeru, da bi izbrali kot polmer zameglitve pri dani metodi večje število, bi bili razultati lahko manj optimalni. Polmer v vrednosti 100 bi denimo zahteval dimenzije skupine v velikosti 256×256 za dosego podobne pokritosti in optimiziranega prenosa podatkov v lokalni pomnilnik, kar pa presega zmožnosti naše naprave in prostor lokalnega pomnilnika[1]. Najboljša rešitev v tem in temu podobnih primerih bi bilo prilagojeno izvajanje glede na vhodne parametre. Če pri dani metodi dvakratna vrednost polmera presega 16 ali 32, ali je polmer manjši ali enak 2, pri čemer dostopov do istih podatkov še ni dovolj, da bi se obrestovalo prenašanje v lokalni pomnilnik, naj se izvaja ščepec brez rabe lokalnega pomnilnika, sicer pa ščepec z lokalnim pomnilnikom. Testiranje učinkovitosti tovrstnega pristopa je lahko predmet nadaljnjih raziskav.

[1] S funkcijo `clGetDeviceInfo()` ob uporabi zastavice `CL_DEVICE_LOCAL_MEM_SIZE` smo za testirano GPE napravo zabeležili velikost lokalnega pomnilnika v vrednosti 65536 bajtov.

Razlike v dimenzijah procesiranih slik se niso pokazale kot ključne pri vprašanju boljši ali slabših nastavitev. V nekaterih primerih se je pri večji sliki sicer kot boljša kombinacija dimenzij izkazala drugačna nastavitev kot pri manjši sliki, vendar ne z dovolj signifikantno razliko, da je ne bi bilo mogoče zanemariti.

Literatura

[1] A. Munshi, B. R. Gasterm, T. G. Mattson, J. Fung, D. Ginsburg. *OpenCL Programming Guide*. Addison-Wesley, 2012.

[2] J. Tompson, K. Schlachter. *An Introduction to the OpenCL Programming Model*. Person Education, 2012.

[3] N. Trevett. *OpenCL Introduction*. Khronos Group, 2013. Dostopno na: https://www.khronos.org/assets/uploads/developers/library/overview/opencl_overview.pdf

[4] A. Munshi. *The OpenCL Specification*. Khronos OpenCL Working Group, 2012. Dostopno na: https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf

[5] GEGL. Dostopno na: http://http://www.gegl.org

[6] GIMP 2.6 Release. Dostopno na: http://www.gimp.org/release-notes/gimp-2.6.html

[7] *GIMP 2.8 RC 1 includes OpenCL acceleration - khronos.org news*. Khronos Group, 2012. Dostopno na: https://www.khronos.org/news/permalink/gimp-2.8-rc-1-includes-opencl-acceleration

[8] GNOME GIT Repository. Dostopno na: https://git.gnome.org/browse/gegl/tree/opencl

[9] C. Zubieta, J. Vesely. red-eye-removal.cl. Dostopno na: https://git.gnome.org/browse/gegl/tree/opencl/red-eye-removal.cl

[10] V. Oliveira, T. Mazars. snn-mean.cl. Dostopno na: https://git.gnome.org/browse/gegl/tree/opencl/snn-mean.cl

[11] V. Oliveira. gaussian-blur.cl. Dostopno na: https://git.gnome.org/browse/gegl/tree/opencl/gaussian-blur.cl

[12] R. A. Jarvis, E. A. Patrick. "Clustering Using a Similarity Measure Based on Shared Near Neighbors", *IEEE Transactions on Computers*, zv. C-22, št. 11, str. 1025-1034, 1973. Dostopno na: http://davide.eynard.it/teaching/2011_PAMI/papers/01672233.pdf

[13] "Gaussian Kernel Calculator", *The Devil in the Details*, 2014. Dostopno na: http://dev.theomader.com/gaussian-kernel-calculator/

[14] G. Cottenceau. "A simple libpng example program", 2002-2010. Dostopno na: http://zarb.org/ gc/html/libpng.html

[15] Y. Niwa. libpng_test.c Dostopno na: https://gist.github.com/niw/5963798

[16] libpng Dostopno na: http://www.libpng.org/pub/png/libpng.html

[17] OpenGL. Dostopno na: https://www.opengl.org/

[18] N. Stewart. GLUT Shapes Demo, 2003. Dostopno na: http://lpaste.net/6429

[19] "An Introduction on OpenGL with 2D Graphics - OpenGL Tutorial", 2012. Dostopno na: https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_Introduction.html

[20] matplotlib: python plotting - Matplotlib 1.4.2 documentation Dostopno na:
http://matplotlib.org/