Kazalo

Slike

1 Motivacija

Rezanje šivov (angl. seam carving) je tehnika, s katero spreminjamo velikost slike z namenom, da ohranimo njeno vsebino brez popačenj. Potreba po dinamičnem prilagajanju velikosti slik je nastala zaradi številnih medijev različnih velikosti, ki že podpirajo dinamične spremembe v postavitvi strani in besedila (npr. HTML), ne pa slik.

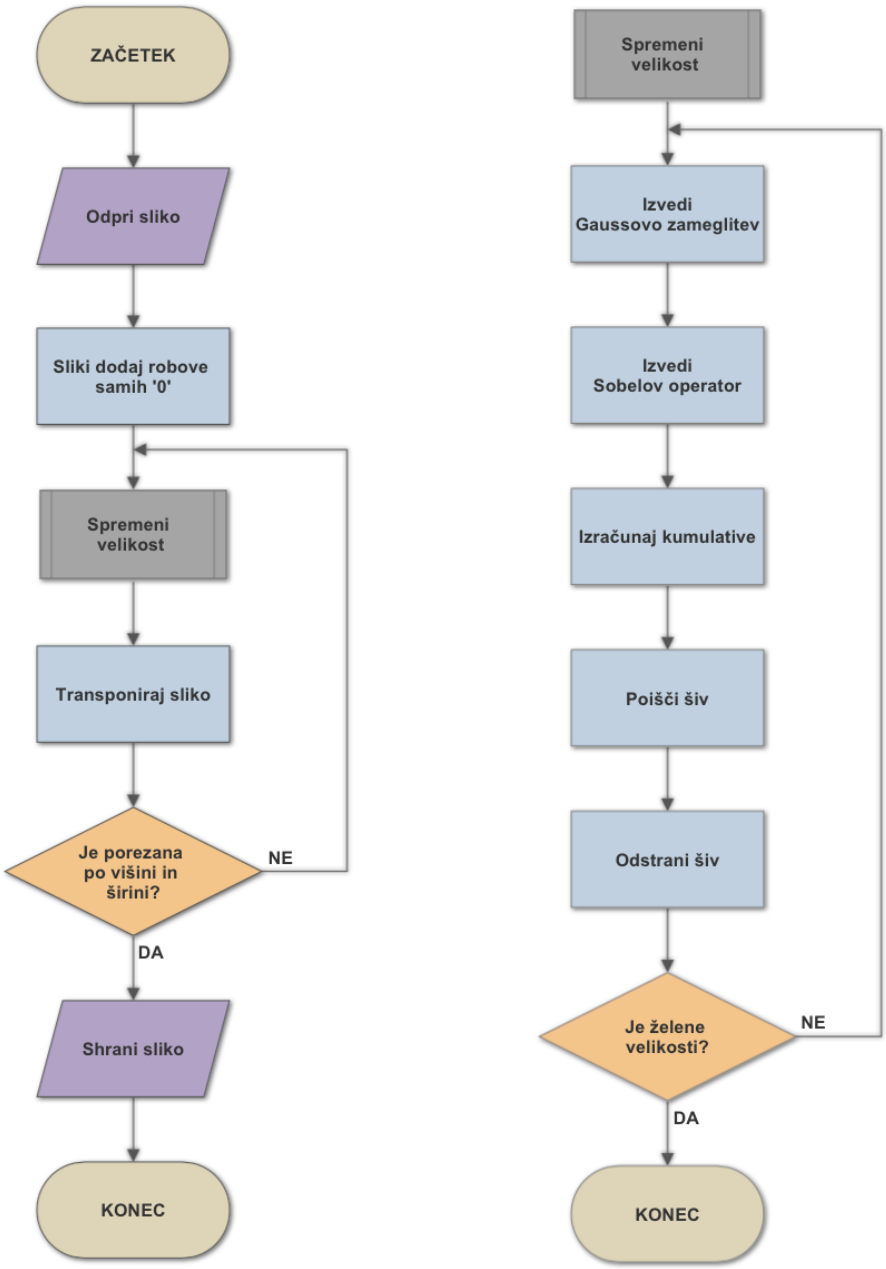
Algoritem poišče najmanj pomembne poti (angl. seams) na sliki, ki jih odstrani, s čimer se zmanjša velikost slike oziroma jih doda, s čimer se velikost slike poveča. Nadgradnja algoritma omogoča, da definiramo območja, ki jih dodatno utežimo. S tem lahko ohranimo dele slike nedotaknjene ali pa odstranimo večje objekte.

Za implementacijo algoritma smo se odločili, ker je idealen za paralelizacijo. Tehnike, ki jih spoznavamo pri predmetu Porazdeljeni sistemi, bomo implementirali na izbranem problemu. Serijski algoritem je pri obdelavi slik večje resolucije zelo počasen, zato bo služil kot odlična referenca o izboljšavah.

2 Serijski algoritem

2.1 Implementacija

Serijski algoritem je implementiran v jeziku C++. Pri implementaciji je bila uporabljena knjižnjica OpenCV [6]. OpenCV je odprtokodna knjižnica, ki vsebuje algoritme, ki se pogosto uporabljajo pri računalniškem vidu. Kljub temu da knjižnica omogoča mnogo naprednih algoritmov, je uporabljena le za branje in prikazovanje slik. Vhod programa je slika ter želena širina in višina slike. Izhod programa je črno-bela slika želenih dimenzij.



Slika 1: Diagram poteka serijskega algoritma za rezanje šivov

Algoritem glavne zanke

Algoritem glavne zanke je sestavljen iz naslednjih korakov:

1. Branje slike v pomnilnik.
2. Razširjanje slike z dodatnimi praznimi robovi. Dodatni robovi so sliki dodani zato, da poenostavijo delovanje algoritma na robovih slike.
3. Rezanje slike po njeni širini. Postopek rezanja je bolj podrobno opisan v nadaljevanju.

4. Transponiranje slike. Slika je transponirana zato, da je postopek rezanja lahko implementiran samo po eni dimenziji.

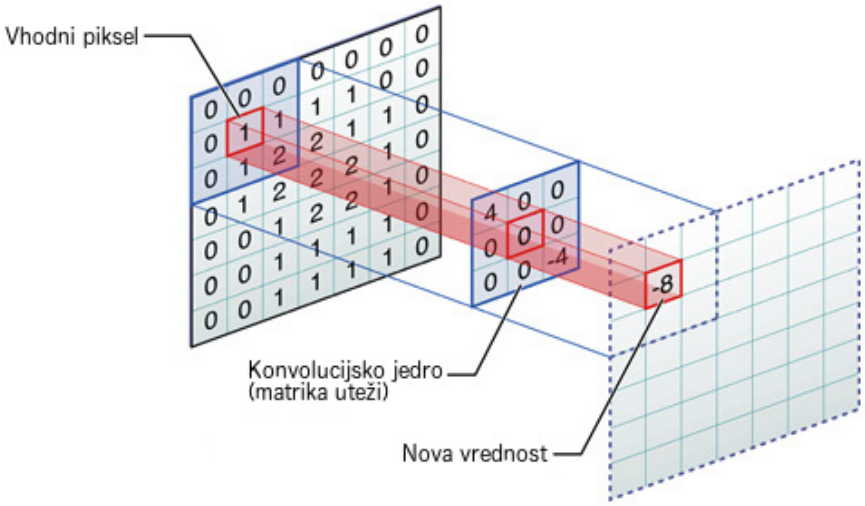
5. Če je rezanje slike končano po obeh dimenzijah, nadaljujemo z naslednjim korakom, v nasprotem primeru pa se vrnemo v korak 3.

6. Prikaz slike

Postopek rezanja slike

Postopek rezanja slike je sestavljen iz naslednjih korakov:

1. Sliki je dodan Gaussian-ov filter [2] za megljenje slike (angl. Gaussian blur). Filter je uporabljen zato, da iz slike odstranimo šum in posledično izboljšamo rezultat filtra za detekcijo robov (Slika 2). Izračunan je s pomočjo uteženega povprečja sosednjih točk.



Slika 2: Prikaz postopka računanja vrednosti točke s pomočjo konvolucijskega jedra

$$gauss = \begin{bmatrix} 0,087133 & 0,120917 & 0,087133 \\ 0,120917 & 0,167799 & 0,120917 \\ 0,087133 & 0,120917 & 0,087133 \end{bmatrix}$$

Slika 3: Matrika 3x3 Gaussovega jedra za megljenje slike z $\sigma = 1,2$

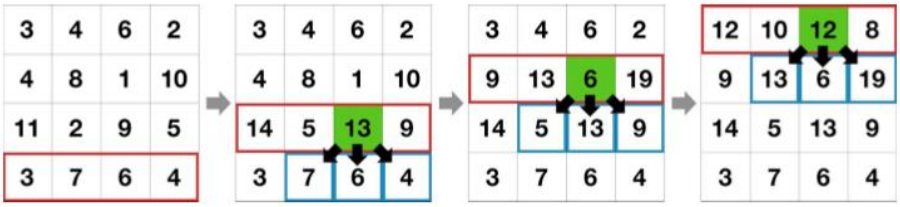
2. Sliki je dodan Sobel-ov filter za detekcijo robov. Postopek izračuna je enak kot je prikazan na sliki (Slika 2), le da v tem primeru izračunamo dve matriki G_x in G_y, rezultat pa je koren vsote kvadratov G_x in G_y.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A \quad \text{in} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

$$G = \sqrt{G_x^2 + G_y^2}$$

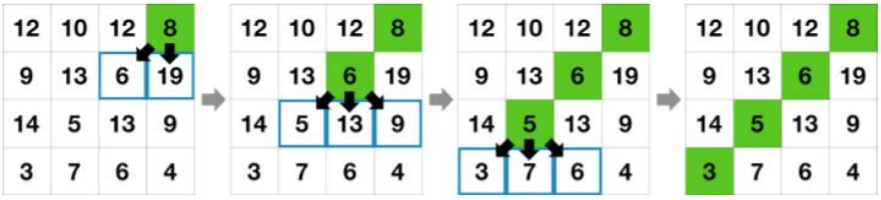
Slika 4: Matriki Sobelovega operatorja za izračun prehodov (robov) na sliki

3. Pri odkrivanje šivov je uporabljen pristop dinamičnega programiranja. Najprej se izračunajo kumulative intenzitet. Računanje se začne na spodnjem robu slike. Vsaki točki priredimo kumulativo svoje itenzitete in ji prištejemo najmanjšo itenziteto izmed njenih spodnjih treh sosednjih točk. Računanje se nadaljuje v zgornji vrstici, dokler ne pridemo do zgornjega roba slike.



Slika 5: Prikaz postopka računanja kumulativ

4. Iz izračunanih kumulativ se odkrije šiv z najmanjšo itenziteto. V zgornjem robu slike se poišče točka z najmanjšo kumulativo. Tam se šiv začne. Šiv se nadaljuje v eni izmed spodnjih treh sosednjih točk. Izbere se točka, ki ima najmanjšo kumulativo. Iskanje šiva se tako nadaljuje do spodnjega roba. Postopek iskanja kumulativ je prikazan na sliki (Slika 5).



Slika 6: Prikaz postopka iskanja šiva

5. Odkriti šiv se iz slike izbriše. Točka šiva je prepisana z njeno desno sosednjo točko, ta pa ponovno z njeno sosednjo točko. Prepisovanje se nadaljuje do desnega roba in nato za vsako točko šiva. Na koncu se širina slike zmanjša za eno točko. Postopek odkrivanja šivov je prikazan na sliki (Slika 6).
6. Če je dosežena želena dimenzija, se postopek konča. V nasprotnem primeru se celoten postopek rezanja ponovi.

Diagram poteka algoritma je prikazan na sliki (Slika 1).

3 Algoritem pThread

3.1 Implementacija

Serijski algoritem je paraleliziran z uporabo knjižnice pThread. Knjižnica je implementacija programskega vmesnika POSIX threads, ki je nastal zaradi potrebe po enotnem vmesniku za

delo z nitmi [9].

Algoritem glavne zanke je ostal isti kot pri serijskem algoritmu. Transponiranje slike ni paralelizirano zato, ker se v celotnem programu slika transponira le dvakrat, kar je malo v primerjavi z delom rezanja slike.

Postopek rezanja slike

Na začetku rezanja slike se niti inicializirajo. Vsaka nit dobi množnico parametrov, ki vključujejo identifikator niti, meta-podatke o sliki, število šivov, ki jih mora odstraniti, kazalec na originalno sliko in kazalca na pomožna pomnilnika. Pomožna pomnilnika in pomnilnik, ki vsebuje originalno sliko je za vse niti enak. Preko njih poteka komunikacija med njimi. Za sinhronizacijo niti je uporabljena prepreka iz knjižnice pThread, ob kateri se niti sinhronizirajo pred nadaljevanjem. Naloga je bila razvita v okoljih Linux in Unix (Mac OS X). Slednji nima podprtih preprek ob knjižnici pThread. Težavo smo rešili tako, da smo k programu dodali programsko kodo preprek [10]. Po inicializaciji niti vstopijo v zanko, ki šteje število porezanih šivov. V zanki se izvajajo naslednji koraki:

1. Vsaka nit glede na svoj identifikator izračuna dve območji delovanja, ki ji pripadata. Prvo je območje, ki niti pripada, če sliko razdelimo na horizontalne pasove, drugo pa območje, ki niti pripada, če sliko razdelimo na vertikalne pasove.

2. Vsaka nit opravi Gaussian-ovo konvolucijo na svojem prvem območju. Ko konča na prepreki počaka druge niti.

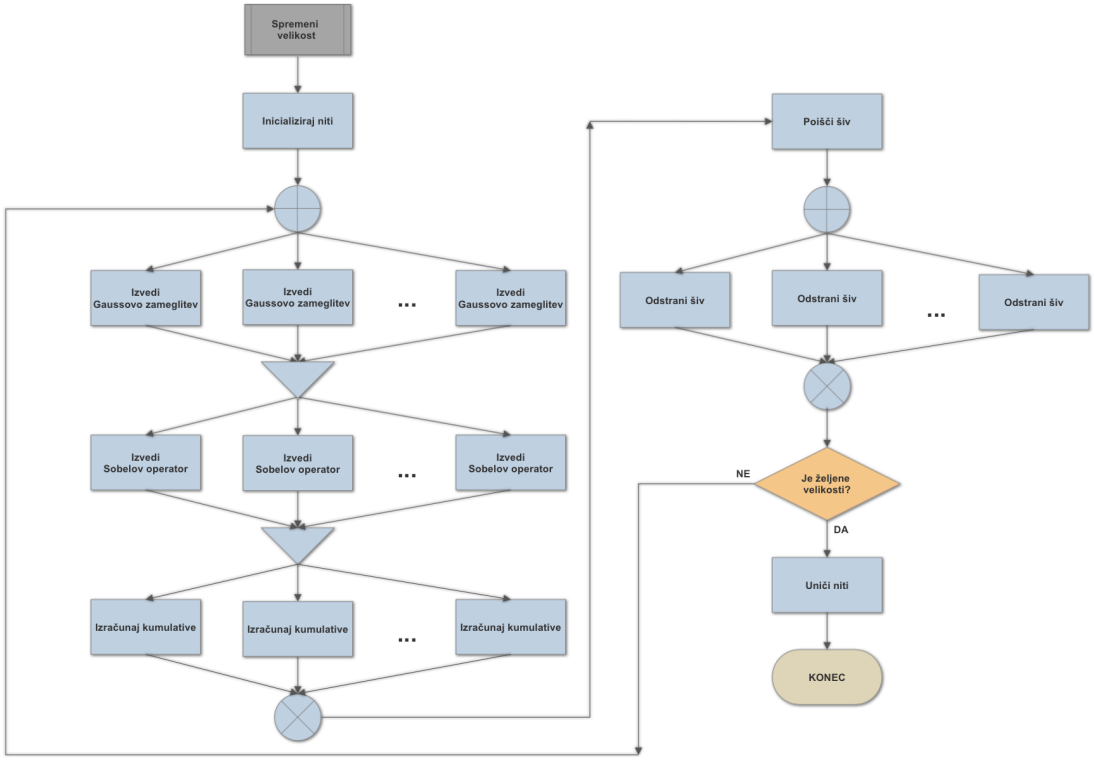
3. Vsaka nit opravi Sobel-ovo konvolucijo na svojem prvem območju. Za tem se niti sinhronizirajo na prepreki.

4. Vsaka nit začne na svojem območju računati kumulative. Ker za to potrebujemo izračunane podatke prejšnje vrstice, se morajo niti sinhronizirati na koncu računanja vsake vrstice.

5. Nit z identifikatorjem 0 poišče šiv glede na izračunane kumulative. Ker je algoritem iskanja šiva težavnosti $O(m+n)$, kjer je m višina slike in n širina slike, algoritem ni paraleliziran. Ostale niti na prepreki počakajo prvo nit.

6. Vsaka nit na svojem prvem območju izbriše šiv, ki mu pripada. Na koncu se niti ponovno sinhronizirajo. Na tem mesu zanka preveri, ali so vsi šivi porezani. Če niso, se izvajanje nadaljuje v koraku (Korak 1). V nasprotnem primeru niti končajo svoje delovanje. Glavna nit nato izvede transformacijo slike in postopek rezanja slike ponovi še po drugi dimenziji.

Postopek rezanja slike je prikazan na sliki (Slika 7).



Slika 7: Diagram poteka funkcije za rezanje šivov, ki je prilagojena, da se algoritem izvaja paralelno.

4 Algoritem OpenMP

Za implementacijo algoritma je uporabljena knjižnica OpenMP [7]. OpenMP je aplikacijski programski vmesnik, ki podpira paralelizacijo z deljenim pomnilnikom na številnih napravah. Sestavlja ga skupek direktiv prevajalniku, knjižničnih rutin in globalnih spremenljivk, ki vplivajo na način izvajanja programa.

Kot osnovo za implementacijo OpenMP algoritma smo uporabili serijski algoritem (Poglavje 2). Na začetku smo dodali klic, ki nastavi število niti, ki jih bo OpenMP uporabljal. Dodali smo tri direktive pragma, s katerimi upravljamo paralelno izvajanje programa. V Gauss-ovi in Sobel-ovi konvoluciji smo direktivi dodali pred zankama, ki iterirata po vrsticah in stolpcih. V pragmi smo dodali parameter, ki pove prevajalniku, naj zanki združi. Z uporabo direktive pragma smo paralelizirali tudi zanko, ki pri brisanju šivov iterira po vrsticah.

Zaradi slabih rezultatov testiranja časov, ki ga je algoritem pThread porabil za računanje kumulativ smo se odločili, da tega pri implementaciji OpenMP algoritma ne bomo paralelizirali.

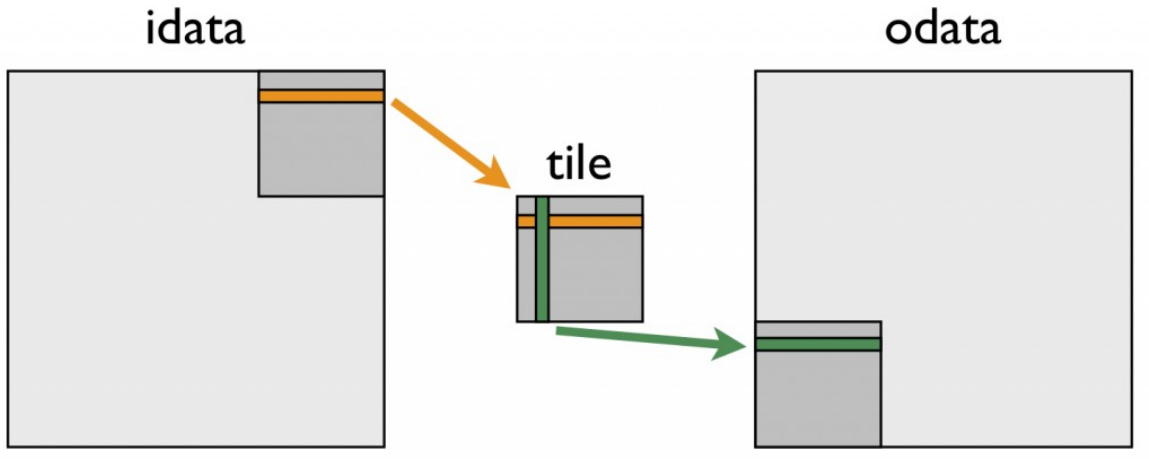
5 Algoritem OpenCL

5.1 Implementacija

Za implementacijo algoritma je uporabljena knjižnica OpenCL [5]. OpenCL (*Open Computing Language*) je odprt standard za razvoj programske opreme, ki teče na raznoliki strojni opremi.

V našem primeru algoritem izkorišča grafične procesne enote (GPE oziroma angl. GPU).

Kljub temu, da je algoritem nekoliko prilagojen, da kar se da dobro izkorišča arhitekturo GPE, sva ostala zvesta konceptu prikazanem na sliki (Slika 7). Dodana vrednost te implementacije algoritma je porazdeljeno transponiranje slike. Ideja je povzeta z objave na blogu Nvidia-je, ki je bila objavljena leta 2013 s strani Marka Harrisa [1]. Algoritem deluje na nivoju poravnanega dostopa do globalnega pomnilnika in transpozicije v lokalnem pomnilniku, saj je cena dostopov manjša, kot pri neporavnanem dostopu do glavnega pomnilnika. Zapis v glavni pomnilnik pa je ponovno poravnan (Slika 8). V literaturi se takšna tehnika imenuje *angl. "tiled transpose"*, saj se transponirajo manjši kosi oziroma ploščice slike.



Slika 8: Potek transpozicije slike s pomočjo lokalnega deljenega pomnilnika. Slika se poravnano prebere z glavnega pomnilnika (oranžna barva) in prenese v lokalen pomnilnik. Nato se iz lokalnega pomnilnika slika prebira neporavnano zaradi cenejših dostopov in se zopet poravnano zapiše v glaven pomnilnik (zelena barva).

Koraki algoritma

1. V prvem koraku se inicializira okolje in izbere napravo (GPE) na kateri se bo izvajalo računanje. Nato se za izbrano napravo pripravi kontekst in ukazna vrsta. Iz posebne datoteke *gpu.cl* se naloži programska koda, ki bo tekla na napravi in se zanjo ustrezno prevede.

2. V drugem koraku se na GPE prenese slika, ki jo želimo obrezati. Hkrati se rezervira prostor tudi za vmesne izračune. GPE je s tem pripravljena za računanje.

3. Nato se izvede glajenje slike s pomočjo Gaussovega jedra. Niti hkrati delujejo na celi sliki (1 nit na 1 točki), saj je operacija neodvisna. Naslednji korak se izvede, ko zaključijo vse niti oziroma se zaključijo vsi ščepci (globalna sinhronizacija na CPE). Končno poročilo: - kazalo - uvod - implementacije in testi - skupna primerjava vseh arhitektur - zaključek (povzetek) - + nekje teoretična analiza (stroški komunikacije pri MPI, raztegljivost, ...)

4. V četrtem koraku se izvede Sobelov operator za iskanje robov na sliki. Tudi v tem koraku delujejo niti hkrati na celotni sliki (1 nit na 1 točki), saj je tudi ta operacija neodvisna.

Naslednji korak se izvede, ko zaključijo vse niti oziroma se zaključijo vsi ščepci (globalna sinhronizacija na CPE).

5. Računanje komulativ se izvaja po vrsticah. Niti hkrati poračunavajo na celotni vrstici in se sinhronizirajo na CPE. CPE torej za vsako vrstico zažene svoj ščepec, da poračuna kumulative na celotni sliki.

6. Izračunane kumulative se prenesejo nazaj v pomnilnik na gostitelju, kjer se s serijsko implementacijo algoritma poišče šiv z najmanjšo intenziteto.

7. Lokacija šiva se prenese v glavni pomnilnik GPE, s pomočjo katerih ga niti po vrsticah odstranijo.

8. Algoritem se vrne na korak (Korak 3), dokler slika ni željene velikosti. Slika se nato transponira in v enakih korakih poreže še po drugi osi. Za konec se slika transponira še enkrat, da se vrne v začetno stanje.

6 MPI algoritem

Za implementacijo algoritma smo uporabili standard MPI, ki definira model za paralelno programiranje, predvsem to kako prenašati podatke iz naslovnega prostora enega procesa v naslovni prostor nekega drugega procesa [4].

MPI poskrbi za pravilno ustvarjanje procesov, algoritem pa je sestavljen iz naslednjih korakov:

1. Prvi proces prebere sliko in jo razpošlje drugim procesom. Slika je razdeljena po vrsticah, število vrstic pa je določeno tako, da vsak proces (vključno s prvim) dobi enako število vrstic. Poleg tega proces dodatno prejme še eno vrstico nad svojim delom slike in eno pod njim. Ta dva robova sta potrebna za pravilno računanje v nadaljnjih korakih.

2. Vsak proces nad svojim delom slike opravi Gaussian-ovo konvolucijo.

3. Robovi se med procesi sinhronizirajo. To pomeni, da vsak proces pošlje procesu, ki obdeluje naslednji del slike, svojo spodnjo vrstico. Podobno vsak proces pošlje procesu, ki obdeluje prejšnji del slike svojo zgornjo vrstico. Ob zaključku te komunikacije imajo vsi procesi pravilne robove.

4. Vsak proces nad svojim delom slike opravi Sobel-ovo konvolucijo.

5. Proces, ki obdeluje najbolj spodnji del slike, začne izračun kumulativ. Ko konča delo na svojem delu slike, pošlje najbolj zgornjo vrstico procesu, ki obdeluje prejšnji del slike. Ta proces vrstico prejme in opravi delo na svojem delu slike, nato pa ponovno odpošlje zgornjo vrstico. Postopek se ponavlja, dokler izračun kumulativ ne konča še zadnji proces. Ta postopek računanja kumulativ je po svoji naravi serijski, kljub temu da se izvaja na več procesih, saj mora vsak proces vedno počakati proces, ki obdeluje del slike pod njim,

da konča svoje delo in odpošlje zgornjo vrstico. Alternativa temu postopku bi bila ta, da bi se celotna slika zbrala na prvem procesu, ki bi izračun kumulativ prav tako opravil serijsko, poiskal šiv in ga razposlal procesom (ti bi nato nadaljevali z brisanjem šiva). Prednost izbranega postopka je v tem, da ni potrebno opraviti pošiljanja celotne slike, ampak zadostuje pošiljanje toliko vrstic, kot je število procesov.

6. Iskanje šiva poteka v obratni smeri iskanja kumulativ. Proces, ki obdeluje najbolj zgornji del slike začne z iskanjem šiva. Ko z iskanjem šiva konča, pošlje indeks stolpca zadnje vrstice šiva procesu, ki obdeluje naslednji del slike. Ta z iskanjem šiva nadaljuje na svojem delu slike. Postpotek se ponavlja, dokler ni odkrit celoten šiv.

7. Vsak proces izbriše šiv na svojem delu slike.

8. Robovi se med procesi sinhronizirajo z uporabo postopka, opisanega v koraku 3. S tem je postopek rezanja enega šiva končan. Če želena širina slike še ni dosežena, algoritem postopek ponovi s korakom 2.

9. Ko je rezanje slike po eni dimenziji končano, vsi procesi pošljejo svoj del slike prvemu procesu. Ta sliko združi in jo transponira. Nato odreže odstranjene vrstice in transponirano sliko ponovno razpošlje vsem procesom. Celoten algoritem se nato ponovi, da je slika obrezana tudi po drugi dimenziji.

7 Rezultati

Algoritem smo testirali na eni sliki (Slika 9), ki je podana v različnih dimenzijah. Najmanjša je velikosti 500x254, največja pa 6000x3049 točk.

Slika 9: Slika uporabljena za testiranje algoritma, ki je podana v petih različnih dimenzijah.

Slika (Slika 10) prikazuje sliko v različnih delih algoritma. Zgoraj levo je prikaz slike po dodanem Gaussian-ovem in Sobel-ovem filtru. Zgoraj desno je prikaz izračunanih kumulativ. Spodaj levo je prikaz najdenega šiva (na desni strani slike, označen z belo barvo). Spodaj desno je končni rezultat algoritma.

Slika 10: Testna slika v različnih fazah algoritma.

Testiranje smo opravili na prenosniku ASUS N550JV, ki vsebuje 4-jedrni procesor Intel Core i7-4700HQ in 16GB pomnilnika.

Teoretična zahtevnost algoritma je $O(m \times n \times N)$, kjer je m širina, n višina slike in N

število odstranjenih šivov. Pri vsaki sliki smo z algoritem izvedli z enako vrednostjo parametra N ($N = 400$). Za vsako sliko smo izvedli 10 ponovitev ter izračunali povprečni čas izvajanja in standardno napako. Rezultati za serijski algoritem se nahajajo v tabeli (Tabela 1). Paralelni algoritem smo testirali z uporabo 1, 2, 4 in 8 niti. Rezultati implementacije paralelne implementacije s pomočjo knjižnice pThread in knjižnice OpenMP se nahajajo v tabeli (Tabela 2). Pri meritvah paralelnih algoritmov smo izračunali tudi pohitritev po formuli $S = t_s/t_p$, kjer t_s predstavlja čas izvajanja serijskega algoritma, t_p pa čas izvajanja paralelnega algoritma, ter učinkovitost po formuli $E = S/p$, kjer p predstavlja število niti, ki jih je paralelni program uporabljal.

N (širina x višina)	Čas [s]	SE [s]
127000 (500 x 254)	0,334	0,007
508000 (1000 x 508)	1,718	0,005
1143000 (1500 x 762)	4,260	0,031
4572000 (3000 x 1524)	18,347	0,012
18294000 (6000 x 3049)	76,286	0,076

Tabela 1: Čas izvajanja in standardna napaka meritve (SE) serijskega algoritma v odvisnosti od velikosti slike *waterfall.jpg*.

Št. niti	N (širina x višina)	pThread				OpenMP			
		Čas [s]	SE [s]	S	E	Čas [s]	SE [s]	S	E
1	127000 (500 x 254)	0,346	0,002	0,934	0,934	0,608	0,002	0,358	0,358
	508000 (1000 x 508)	1,800	0,003	0,986	0,986	3,180	0,004	0,540	0,540
	1143000 (1500 x 762)	4,365	0,009	0,985	0,985	7,861	0,007	0,542	0,542
	4572000 (3000 x 1524)	18,586	0,040	0,997	0,997	34,144	0,020	0,537	0,537
	18294000 (6000 x 3049)	77,111	0,139	1,006	1,006	144,158	0,140	0,529	0,529
2	127000 (500 x 254)	0,461	0,017	0,701	0,350	0,339	0,004	0,985	0,492
	508000 (1000 x 508)	1,528	0,011	1,161	0,580	1,702	0,002	1,009	0,504
	1143000 (1500 x 762)	3,341	0,053	1,287	0,643	4,225	0,010	1,008	0,504
	4572000 (3000 x 1524)	12,540	0,116	1,479	0,739	18,711	0,072	0,980	0,490
	18294000 (6000 x 3049)	49,716	0,449	1,560	0,780	84,416	1,075	0,903	0,451
4	127000 (500 x 254)	0,609	0,007	0,531	0,133	0,216	0,010	1,546	0,386
	508000 (1000 x 508)	2,014	0,021	0,881	0,220	1,302	0,111	1,319	0,330
	1143000 (1500 x 762)	4,169	0,034	1,032	0,258	3,168	0,189	1,345	0,336
	4572000 (3000 x 1524)	14,442	0,161	1,283	0,320	15,082	0,394	1,216	0,304
	18294000 (6000 x 3049)	54,715	0,810	1,417	0,354	60,614	0,558	1,258	0,314
8	127000 (500 x 254)	0,923	0,007	0,350	0,044	0,222	0,008	1,504	0,188
	508000 (1000 x 508)	2,490	0,010	0,712	0,089	0,974	0,013	1,763	0,220
	1143000 (1500 x 762)	4,887	0,231	0,880	0,110	2,713	0,010	1,570	0,196
	4572000 (3000 x 1524)	14,789	0,274	1,253	0,157	12,651	0,412	1,450	0,181
	18294000 (6000 x 3049)	52,093	0,610	1,489	0,186	55,328	0,353	1,378	0,172

Tabela 2: Čas izvajanja, standardna napaka meritve (SE), pohitritev (S) in učinkovitost (E) pThread in OpenMP algoritmov v odvisnosti od velikosti slike *waterfall.jpg* in števila niti.

7.1 Algoritem pThread

Rezultate algoritma pThread smo prikazali tudi grafično. Slika (Slika 11) prikazuje čas izvajanja serijskega in paralelnega algoritma pThread pri različnem številu uporabljenih niti. Slika (Slika 12) prikazuje pohitritev algoritma implementiranega s knjižnico pThread.

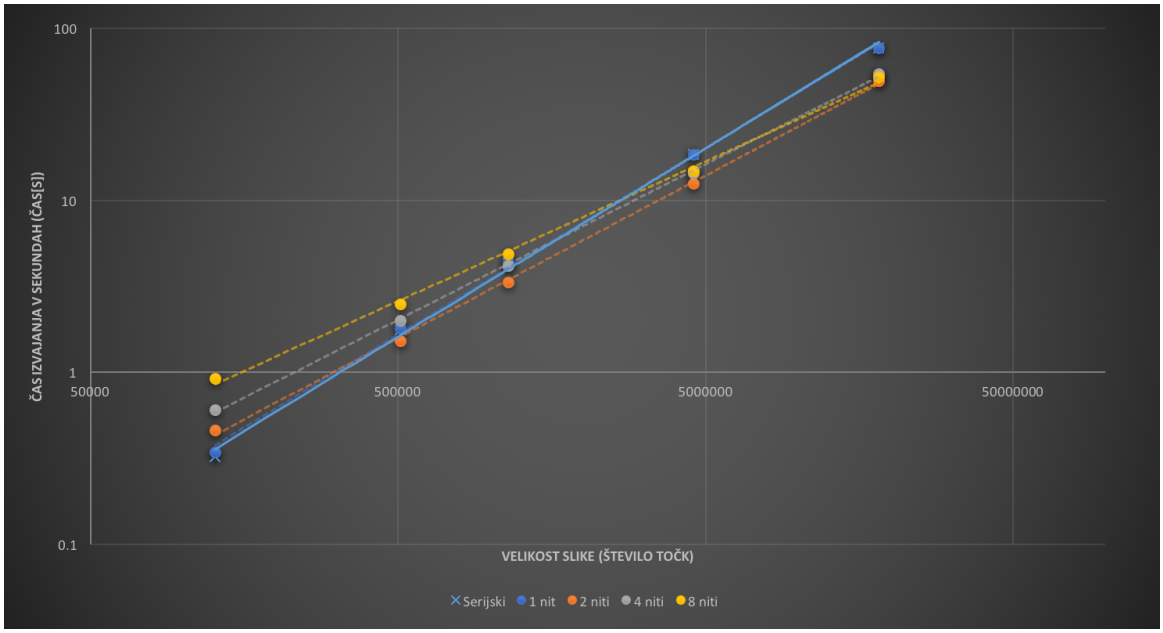
Iz grafa (Slika 11) je razvidno, da se serijski algoritem obnaša linearno glede na velikost slike. Rezultati so skladni s teoretično zahtevnostjo algoritma.

Razvidno je tudi, da je za manjše velikosti slik serijski algoritem hitrejši od algoritma pThread, saj je delo, ki ga paralelni program porabi za upravljanje niti večje od dela, ki ga prihrani s paralelizacijo. Paralelni program se z uporabo ene niti obnaša skoraj enako kot serijski.

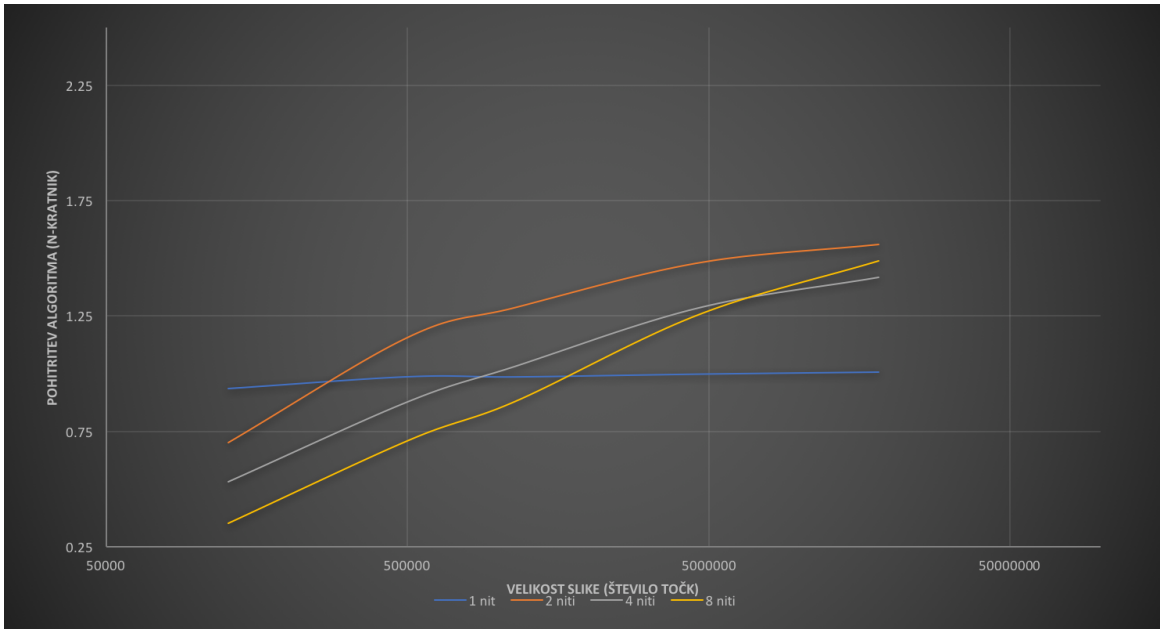
Na grafu pohitritev algoritma pThread (Slika 12 vidimo, da največjo pohitritev (približno 1,5) dosežemo že z uporabo dveh niti. Uporaba več niti algoritem upočasni. Sklepamo, da se to zgodi zaradi dodatnega časa, ki ga niti porabijo za sinhronizacijo ter pribitka na račun zgrešitev

v predpomnilniku.

Naslednji korak izboljšave programa bi bil ta, da združimo serijski in paralelni algoritem. Za manjše velikosti problema bi tako uporabili serijski algoritem, ko pa velikost problema preseže mejo, kjer je serijski algoritem manj učinkovit od paralelnega, pa bi uporabili paralelni algoritem.



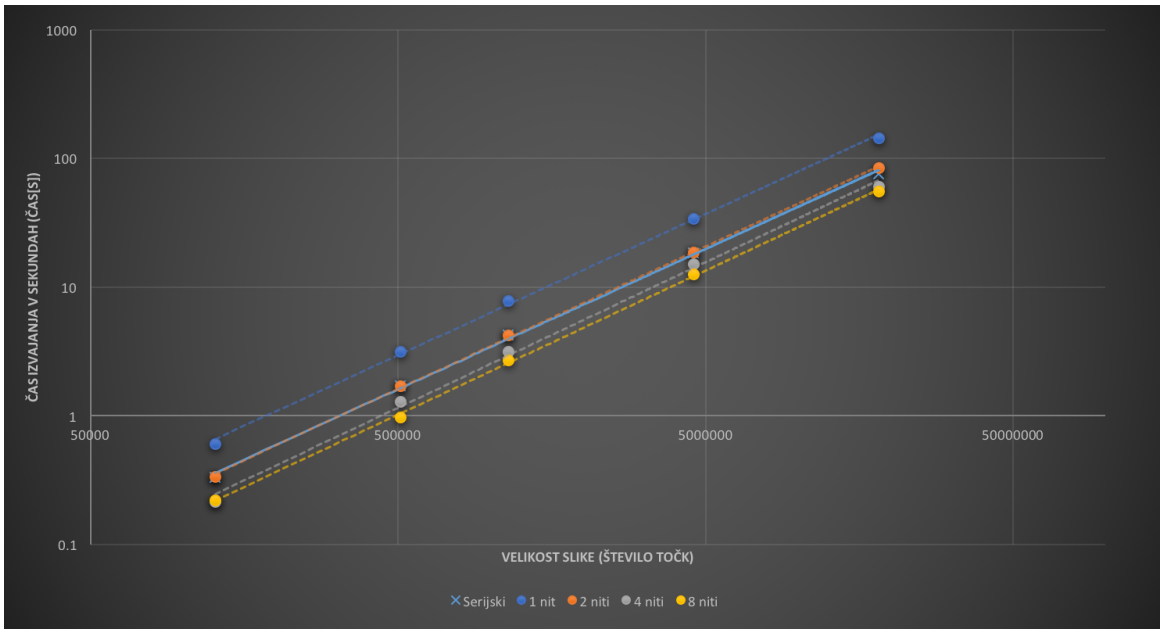
Slika 11: Čas[s] izvajanja algoritma v odvisnosti od velikosti slike pri enakem številu odstranjenih šivov. Polna (svetlo modra) črta prikazuje čas serijskega algoritma, prekinjene črte označujejo paralelno implementacijo. Podatki so prikazani na logaritemski osi.



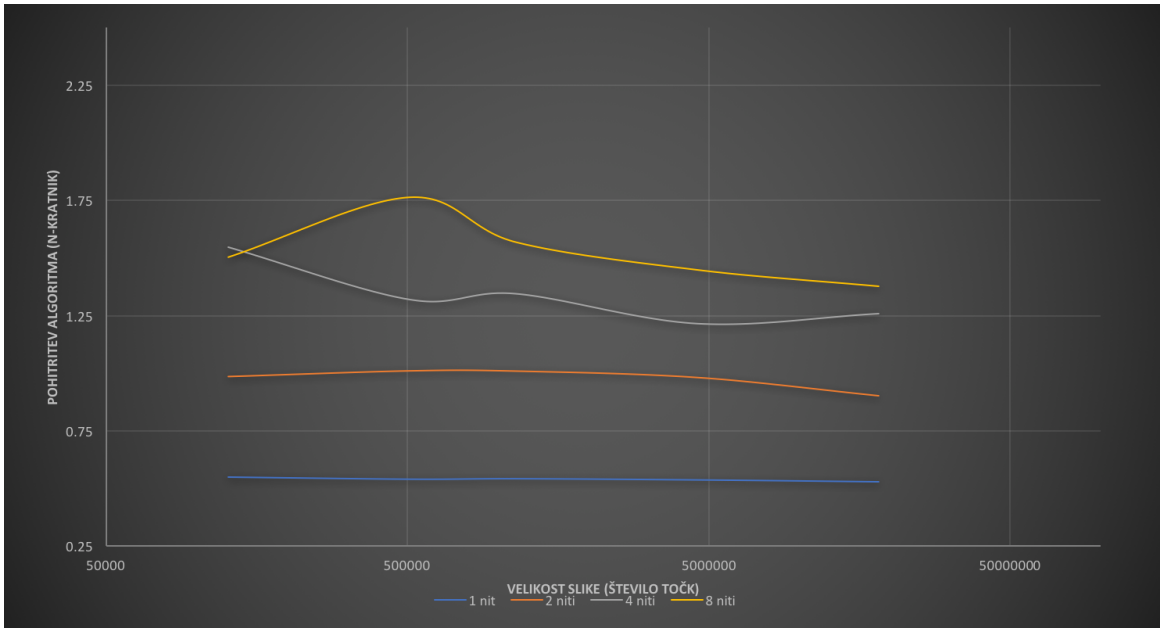
Slika 12: Pohitritev paralelnega algoritma pri različnem številu uporabljenih niti. Podatki so prikazani na logaritemski osi.

7.2 Algoritem OpenMP

Zaradi slabšega obnašanja algoritma pThread pri večjem številu uporabljenih niti smo se odločili, da pred implementacijo OpenMP algoritma izmerimo čase, ki jih algoritem porabi v posameznih delih algoritma. Izmerili smo čase, ki jih algoritem porabi za Gauss-ovo konvolucijo, Sobel-ovo konvolucijo, računanje kumulativ in brisanje šiva. Čas, ki ga algoritem porabi za iskanje šiva, nismo izmerili, saj ta ni paraleliziran. Meritve smo opravili tako, da smo na isti sliki šiv odstranili 200-krat, nato pa rezultat povprečili in izračunali standardno napako meritve. Meritve smo za vsak algoritem opravili na dveh velikostih slike. Rezultati so prikazani v tabeli (Tabela 5). Po implementaciji OpenMP algoritma, smo tabeli dodali tudi meritve, ki smo jih izmerili z njim.



Slika 13: Čas[s] izvajanja algoritma v odvisnosti od velikosti slike pri enakem številu odstranjenih šivov. Polna (svetlo modra) črta prikazuje čas serijskega algoritma, prekinjene črte označujejo paralelno implementacijo (OpenMP). Podatki so prikazani na logaritemski osi.



Slika 14: Pohitritev paralelnega algoritma (OpenMP implementacija) pri različnem številu uporabljenih niti.

Iz rezultatov je razvidno, da za paralelno računanje kumulativ porabimo več časa kot pri serijskem računanju kumulativ. Sklepamo, da se to zgodi zaradi časa, ki ga niti porabijo za sinhronizacijo na koncu vsake vrstice. Zaradi te ugotovitve smo se odločili, da računanje kumulativ pri algoritmu OpenMP ne paraleliziramo.

Rezultate algoritma OpenMP smo prav tako prikazali grafično. Slika (Slika 13) prikazuje čas izvajanja serijskega in paralelnega algoritma OpenMP pri različnem številu uporabljenih niti. Slika (Slika 14) prikazuje pohitritev algoritma implementiranega s knjižnico OpenMP.

Iz grafa časov izvajanja algoritma (Slika 13) je razvidno, da je algoritem OpenMP hitrejši od serijskega algoritma pri uporabi štirih ali osmih niti.

Iz grafa pohitritev (Slika 14) je razvidno, da je OpenMP algoritem hitrejši od serijskega že pri majhnih slikah. Pohitritve so za določeno število niti približno konstantne. Največjo pohitritev smo dosegli z osmimi nitmi (približno 1.75).

7.3 Algoritem OpenCL

Testiranje smo opravili na GPE NVIDIA GeForce GT 750M. Kartica je zasnovana na NVIDIA® Kepler arhitekturi [3] in omogoča do 6,3 kratno pohitritev v primerjavi z Intel HD Graphics 4000. Podpira OpenCL 1.2 standard.

Meritve so prikazane v tabeli (Tabela 3). Iz meritev je razvidno, da je algoritem najučinkovitejši pri delovni skupini velikosti 128. GPE omogoča največ 1024 niti znotraj ene delovne skupine. Najvišja pohitritev je okoli 2,8 krat, kar je precejšna izboljšava napram serijskemu algoritmu predvsem pri velikih slikah. Pohitritve so prikazane tudi grafično na sliki (Slika 15).

Algoritem je sicer hitrejši, vendar pa smo mnenja, da bi se dalo še bolje. Hrošč, ki se skriva v programski kodi in ga nismo uspeli odpraviti, vpliva na hitrost izvajanja. Iz neznanega razloga

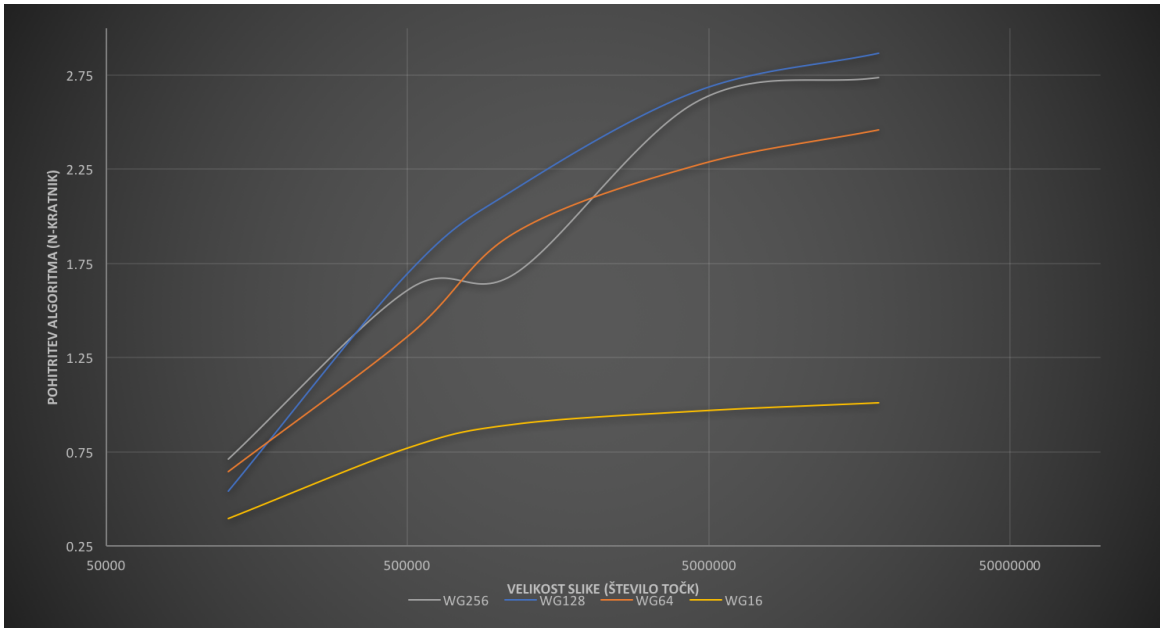
čas pri zaporednih meritvah raste, če med testi ne počakamo dovolj časa. Zaradi tega nam nekaterih delov algoritma ni uspelo povsem prilagoditi GPE.

Eno ozko grlo algoritma je prenos slike na gostitelja in iskanje šiva. Pri velikih slikah se v tej točki izgubi precej časa. Če bi se iskanje šiva izvajalo na GPE, sklepamo, da bi bil algoritem hitrejši.

Kljub temu so časi mnogo boljši, kot v prejšnjih implementacijah.

Delovna skupina	N (širina x višina)	OpenCL	
		Čas [s]	S
16	127000 (500 x 254)	0,839	0,398
	508000 (1000 x 508)	2,215	0,775
	1143000 (1500 x 762)	4,741	0,898
	4572000 (3000 x 1524)	18,955	0,967
	18294000 (6000 x 3049)	75,411	1,011
32	127000 (500 x 254)	0,553	0,603
	508000 (1000 x 508)	1,510	1,137
	1143000 (1500 x 762)	3,076	1,384
	4572000 (3000 x 1524)	12,036	1,524
	18294000 (6000 x 3049)	47,337	1,611
64	127000 (500 x 254)	0,517	0,646
	508000 (1000 x 508)	1,251	1,373
	1143000 (1500 x 762)	2,221	1,918
	4572000 (3000 x 1524)	8,070	2,273
	18294000 (6000 x 3049)	30,994	2,461
128	127000 (500 x 254)	0,615	0,543
	508000 (1000 x 508)	1,004	1,711
	1143000 (1500 x 762)	1,983	2,148
	4572000 (3000 x 1524)	6,880	2,666
	18294000 (6000 x 3049)	26,600	2,867
256	127000 (500 x 254)	0,470	0,710
	508000 (1000 x 508)	1,062	1,617
	1143000 (1500 x 762)	1,980	2,151
	4572000 (3000 x 1524)	7,044	2,604
	18294000 (6000 x 3049)	27,873	2,736

Tabela 3: Čas izvajanja in pohitritev (S) OpenCL algoritma v odvisnosti od velikosti slike *waterfall.jpg* in števila niti.



Slika 15: Pohitritev algoritma implementiranega na GPE pri različnih velikostih delovnih skupin.

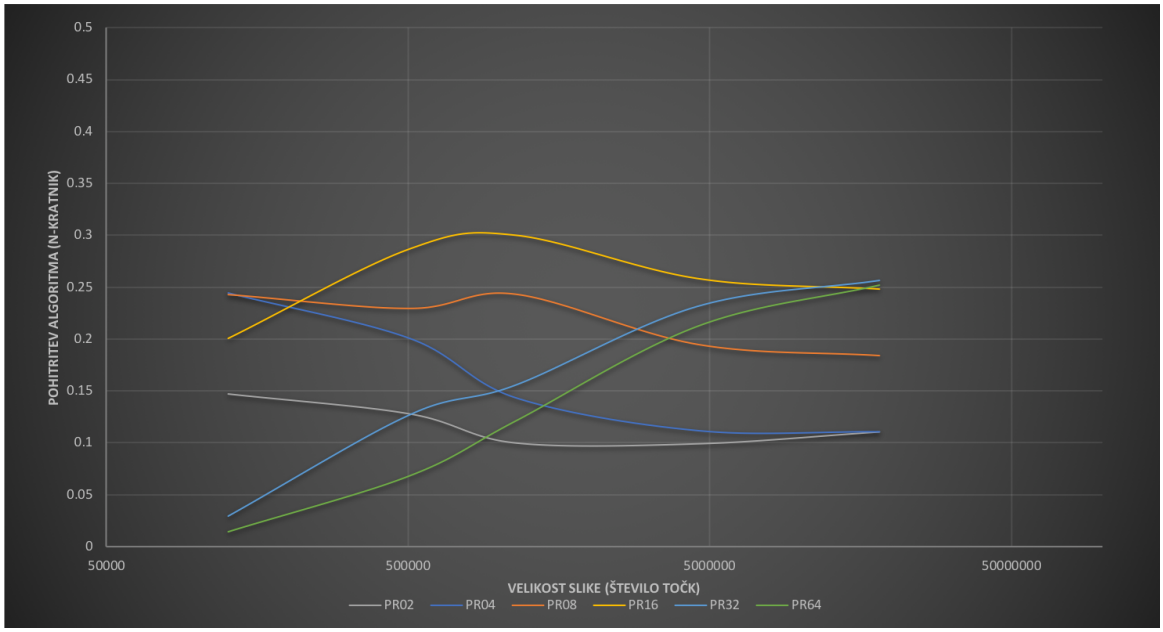
7.4 Algoritem MPI

Algoritem MPI smo testirali na dveh sistemih. Prvi sistem je bilo lokalno okolje prenosnika, opisanega na začetku poglavja. Testiranje smo opravili z različnim številom procesov. Rezultati testiranja so prikazani v tabeli (Tabela 6). Drug sistem, na katerem smo testirali algoritem, je gruča SLING. SLING je konzorcij za razvoj omrežja grid in upravljanje razpršenih računskih infrastruktur v Sloveniji [8]. Testiranje na gruči smo prav tako opravili z različnim številom procesov, vendar smo tu uporabili tudi do 64 procesov. Rezultati so prikazani v tabeli (Tabela 4). Vse meritve smo ponovili desetkrat in izračunali standardno napako ter pohitritev.

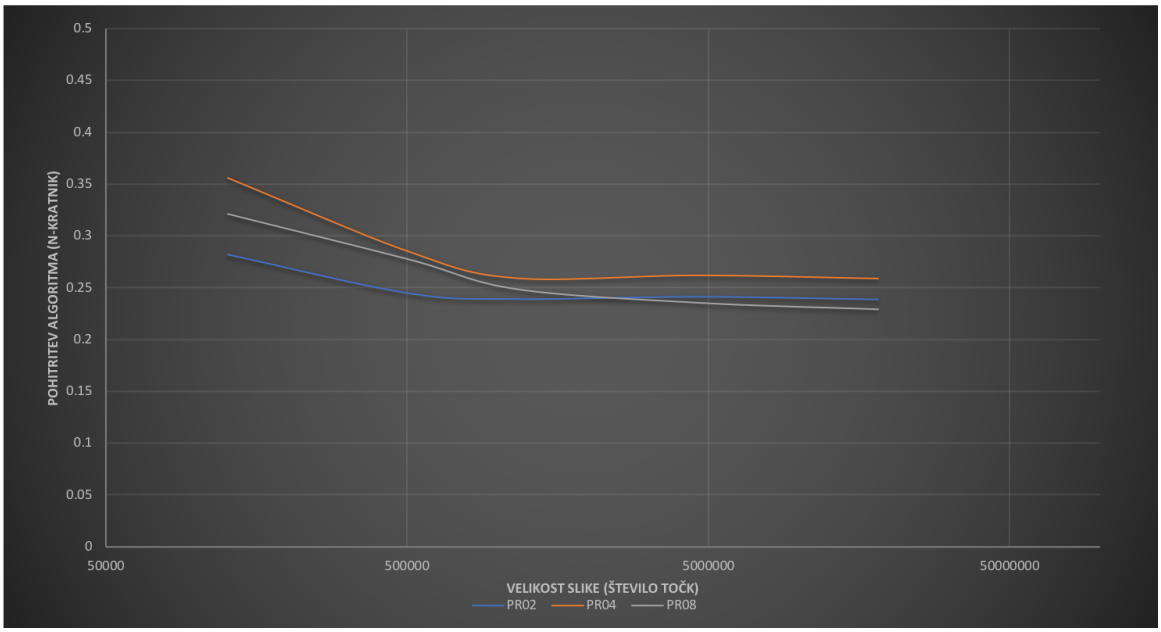
Rezultate smo prikazali tudi grafično. Pohitritev algoritma MPI v odvisnosti od velikosti slike, izmerjena na gruči SLING, je prikazana na sliki (Slika 16). Pohitritev algoritma MPI v odvisnosti od velikosti slike, izmerjena na lokalnem prenosniku, je prikazana na sliki (Slika 17).

Število procesov	N (širina x višina)	MPI	
		Čas [s]	S
2	127000 (500 x 254)	2,267	0,147
	508000 (1000 x 508)	13,430	0,127
	1143000 (1500 x 762)	42,734	0,099
	4572000 (3000 x 1524)	185,491	0,098
	18294000 (6000 x 3049)	690,668	0,110
4	127000 (500 x 254)	1,367	0,244
	508000 (1000 x 508)	8,569	0,200
	1143000 (1500 x 762)	29,752	0,143
	4572000 (3000 x 1524)	164,303	0,112
	18294000 (6000 x 3049)	690,074	0,111
8	127000 (500 x 254)	1,374	0,243
	508000 (1000 x 508)	7,476	0,229
	1143000 (1500 x 762)	17,502	0,243
	4572000 (3000 x 1524)	93,990	0,195
	18294000 (6000 x 3049)	413,576	0,184
16	127000 (500 x 254)	1,665	0,200
	508000 (1000 x 508)	5,984	0,287
	1143000 (1500 x 762)	14,191	0,300
	4572000 (3000 x 1524)	71,013	0,258
	18294000 (6000 x 3049)	307,385	0,248
32	127000 (500 x 254)	11,298	0,029
	508000 (1000 x 508)	13,493	0,127
	1143000 (1500 x 762)	27,287	0,156
	4572000 (3000 x 1524)	79,109	0,231
	18294000 (6000 x 3049)	297,002	0,256
64	127000 (500 x 254)	23,661	0,014
	508000 (1000 x 508)	25,203	0,068
	1143000 (1500 x 762)	35,151	0,121
	4572000 (3000 x 1524)	86,381	0,212
	18294000 (6000 x 3049)	302,708	0,252

Tabela 4: Čas izvajanja, pohitritev (S) in učinkovitost (E) MPI algoritma v odvisnosti od velikosti slike *waterfall.jpg* in števila procesov. Meritve so bile izvedene na gruči (SLING).



Slika 16: Pohitritev algoritma implementiranega z MPI pri različnem številu procesov. Meritve so bile izvedene gruči (SLING).



Slika 17: Pohitritev algoritma implementiranega z MPI pri različnem številu procesov. Meritve so bile izvedene lokalno.

7.4.1 Analiza algoritma MPI

Algoritem MPI smo podrobno analizirali. Ocena časovne zahtevnosti serijskega algoritma je prikazana v enačbi (Enačba 2).

Zaradi poenostavitve enačb smo predpostavili, da ima slika enaki dolžini stranic (Enačba 1).

$$m * n \approx n^2 \tag{1}$$

$$t_s(n) = C * n^2 + n \tag{2}$$

Časovno zahtevnost paralelnega algoritma smo izračunali tako, da smo najprej ocenili časovno zahtevnost posameznih delov paralelnega algoritma. Časovne zahtevnosti delov paralelnega algoritma so naslednje:

1. Gaussianova konvolucija: $\frac{n^2}{p} + \beta * n * p$
2. Sobelova konvolucija: $\frac{n^2}{p} + \beta * n * p$
3. Izračun energij: $n^2 + \beta * n * p$
4. Iskanje šiva: $n + \beta * n * p$
5. Brisanje šiva: $\frac{n^2}{p} + \beta * n * p$

Iz teh izračunov smo ocenili časovno zahtevnost paralelnega algoritma, ki je prikazana v enačbi (Enačba 3).

$$t_p(n, p) = C * \frac{n^2}{p} + \beta * n * p \tag{3}$$

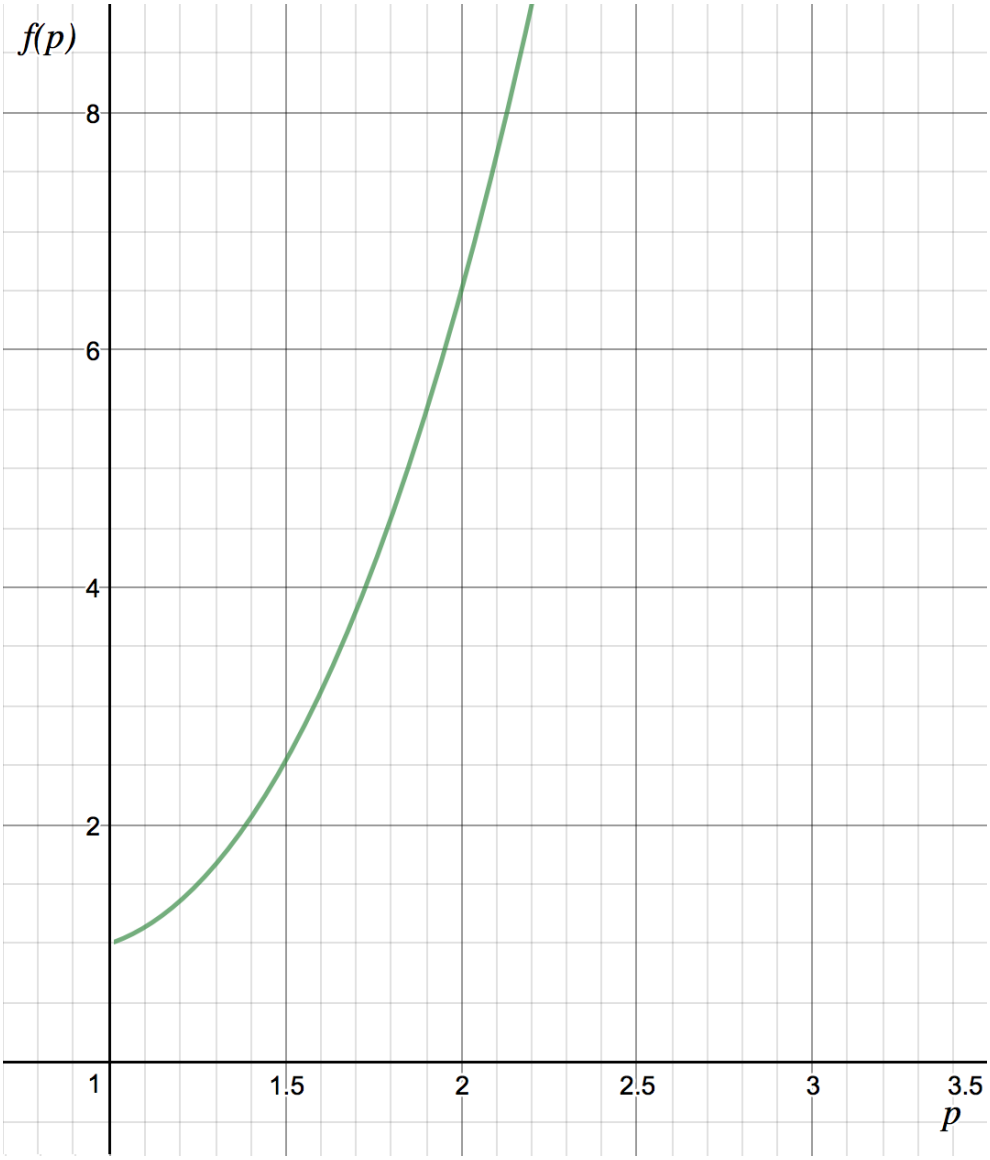
Ocenili smo prostorsko zahtevnost serijskega algoritma. Prikazana je v enačbi (Enačba 4). Izračunali smo tudi razteglјivostno funkcijo paralelnega algoritma. Prikazana je v enačbi (Enačba 5). Graf razteglјivostne funkcije je prikazan na sliki (Slika 18).

$$M(n) = C * n^2 \tag{4}$$

$$\frac{M(g(p))}{p} = C_1 * p^3 - C_2 * p + \frac{1}{p} \tag{5}$$

Iz enačbe za razteglјivost lahko razberemo, da z majhnim večanjem števila procesov, moramo močno povečati velikost slike, da ohranjamo primerno razmerje med velikostjo problema in številom procesov. Ker funkcija narašča z eksponentno hitrostjo, s 3 stopnjo, mora biti slika velikosti približno 400.000 (624x624) točk že pri štirih procesih. Za osem procesov pa mora slika presegati velikost 25.000.000 (5000x5000) točk.

Kot ugotovitev lahko povemo, da algoritem ni mogoče neskončno pohitriti oziroma ne moremo učinkovito večati števila procesov, če močno ne povečamo števila podatkov, ki jih mora obdelati. Pričakovati je, da bi v praksi algoritem lahko tekel pri devetih procesih, če bi morali obdelati sliko, ki je večja od 70MP (do 100.000.000 točk), kar znaša okoli 32MB.



Slika 18: Graf raztegljivostne funkcije v odvisnosti od števila procesov

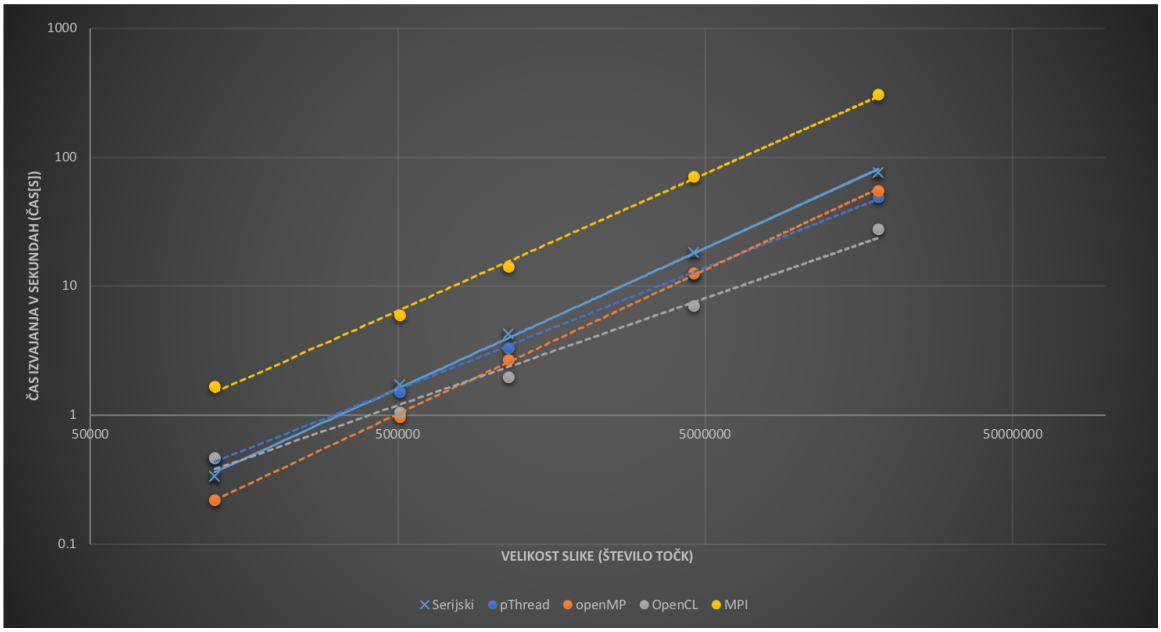
7.5 Primerjava metod

Primerjali smo implementaciji paralelnih algoritmov pThread in OpenMP, MPI ter algoritma OpenCL za GPE. Za primerjavo smo izbrali rezulate algoritma pThread pri uporabi dveh niti, rezultate algoritma OpenMP pri uporabi osmih niti, MPI pri uporabi 16 procesov in rezultate algoritma OpenCL pri delovni skupini 256 nitk. Te rezultate smo izbrali zato, ker so se algoritmi takrat obnašali najboljše. Rezultati časov izvajanja so prikazani na sliki (Slika 19), rezultati

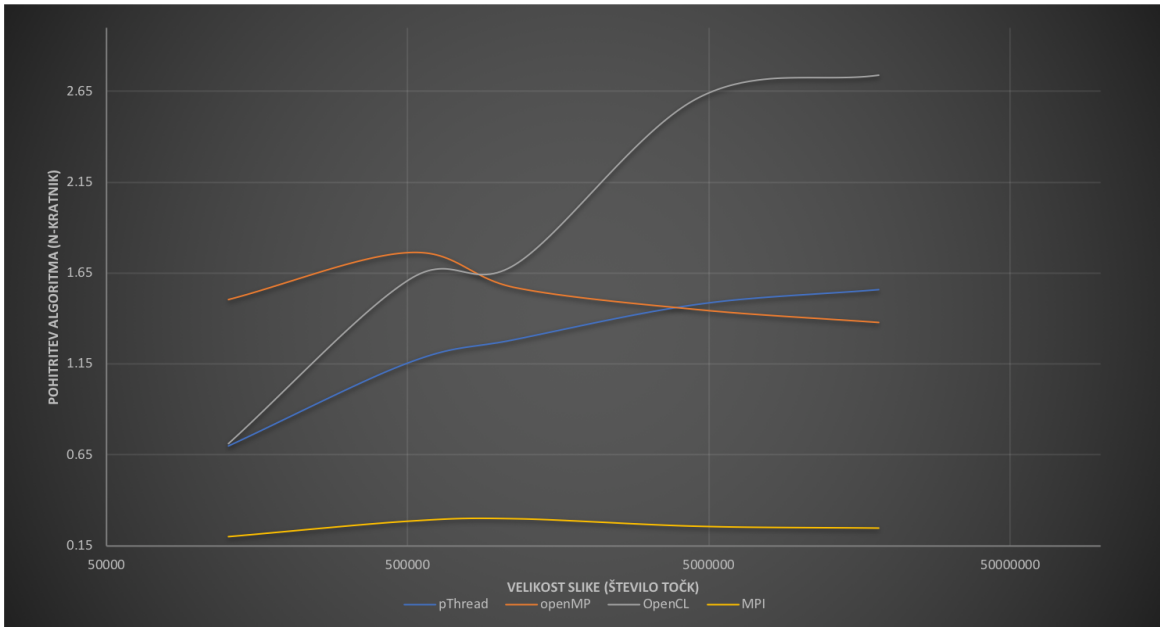
pohitritev pa na sliki (Slika 20).

Iz grafov je razvidno, da je algoritem OpenMP boljši pri majših slikah, pri zelo velikih slikah pa algoritem OpenCL deluje najhitreje. Pohitritev je za enkrat večja od pThread algoritma pred tem. Daleč najslabše se je odrezal MPI, ki je dosegel pohitritev nižjo od 1. To pomeni, da je slabši tudi od serijskega algoritma. Rezultat je pričakovan, saj algoritem ni primeren za izvajanje na gruči, ker vsebuje mnogo manjših sinhronizacij. Vsaka sinhronizacija zahteva vzpostavitev komunikacije med procesi, kar pa je draga operacija.

Kje tiči vzrok za takšno razliko v pohitritvi pri večjih slikah na GPU je razvidno na sliki (Slika 22), ki prikazuje vmesne meritve algoritma. Rezultati so normalizirani glede na serijski algoritem. Opaziti je mogoče, da je bila pohitrena velika večina algoritma, še najbolj računanje konvolucije s Sobelom, najmanj pa računanje kumulativ. Kljub pohitritvam, nastaja kar nekaj izgub pri branju in pisanju vmesnih rezultatov med pomnilnikoma na GPE in gostitelju. Rezultati so zato nekaj slabši, saj bi bilo potrebno optimizirati še ta del algoritma.



Slika 19: Primerjava časa[s] izvajanja algoritma med različnimi implementacijami.



Slika 20: Primerjava pohitritev algoritma med različnimi implementacijami.

8 Zaključek

Skozi semester smo pri predmetu *Porazdeljeni Sistemi* spoznavali različne pristope k porazdeljeni implementaciji algoritmov. Najprej smo implementirali serijski algoritem, ki je bil zasnova za kasnejšo porazdeljeno implementacijo. Algoritem smo nato predelali v pThread, OpenMP, OpenCL in MPI implementacijo.

V končni primerjavi različnih implementacij, se je najbolje odrezal algoritem z OpenCL implementacijo za GPE pri slikah večjih od 1.000.000 točk. Pri manjših slikah pa se je izkazalo, da OpenMP najbolje opravi svoje delo. Ostalo je še kar nekaj prostora za izboljšave, še posebej pri algoritmu OpenCL in algoritnmu MPI. Pri OpenCL so se pokazale manjše težave, ki onesposobijo grafično kartico in sistem postane ne odziven. Možnih vzrokov je več, a sklepamo, da je težava v gonilniku, ki je bil nameščen na OS Linux.

Če bi nalogo opravnjala še enkrat, bi algoritem že v serijskem delu zasnovala tako, da ga bi bilo lažje paralelizirati, saj zdaj precej bolje poznava arhitekturne podrobnosti in dobre prakse, ki se pojavijo pri uporabi posamezne tehnike paralelizacije.

Literatura

[1] An Efficient Matrix Transpose in CUDA C/C++. Dosegljivo: https://devblogs.nvidia.com/parallelforall/efficient-matrix-transpose-cuda-cc/. [Dostopano: 2. 1. 2018].

[2] Computer Vision, Lecture 2: Image filtering [e-prosojnice]. Dosegljivo: http://www.cs.cornell.edu/courses/cs6670/2011sp/lectures/lec02_filter.pdf. [Dostopano: 1. 11. 2017].

[3] Kepler - the world's fastest, most efficient hpc architecture. Dosegljivo: http://www.nvidia.com/object/nvidia-kepler.html. [Dostopano: 2. 1. 2018].

[4] MPI: A Message-Passing Interface Standard. Dosegljivo: http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf. [Dostopano: 19. 1. 2018].

[5] The open standard for parallel programming of heterogeneous systems. Dosegljivo: https://www.khronos.org/opencl/. [Dostopano: 2. 1. 2018].

[6] OpenCV, odprtokodna knjižnica za računalniški vid. Dosegljivo: https://opencv.org/. [Dostopano: 1. 11. 2017].

[7] OpenMP. Dosegljivo: http://www.openmp.org/specifications/. [Dostopano: 7. 12. 2017].

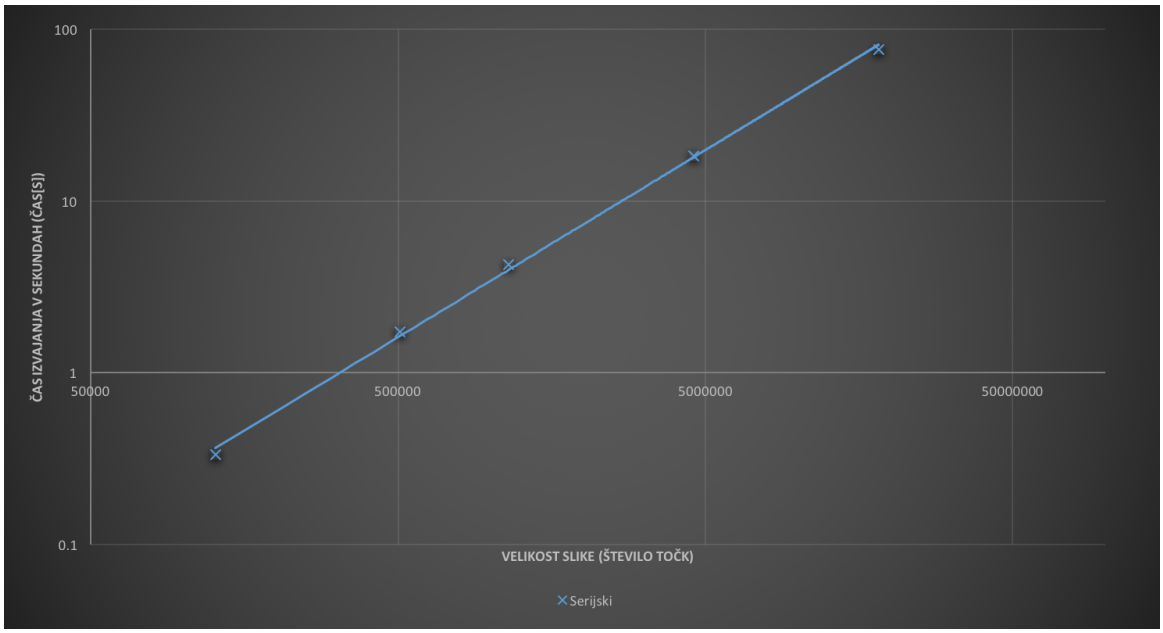
[8] SLING - Slovenska iniciativa za nacionalni grid. Dosegljivo: http://www.sling.si/sling/. [Dostopano: 19. 1. 2018].

[9] Blaise Barney. POSIX Threads Programming. Dosegljivo: https://computing.llnl.gov/tutorials/pthreads/. [Dostopano: 26. 11. 2017].

[10] Ethan Spitz. Using pthread barrier on Mac OS X. Dosegljivo: http://blog.albertarmea.com/post/47089939939/using-pthreadbarrier-on-mac-os-x, 2012–2017.

Priloge

A Čas izvajanja serijskega algoritma



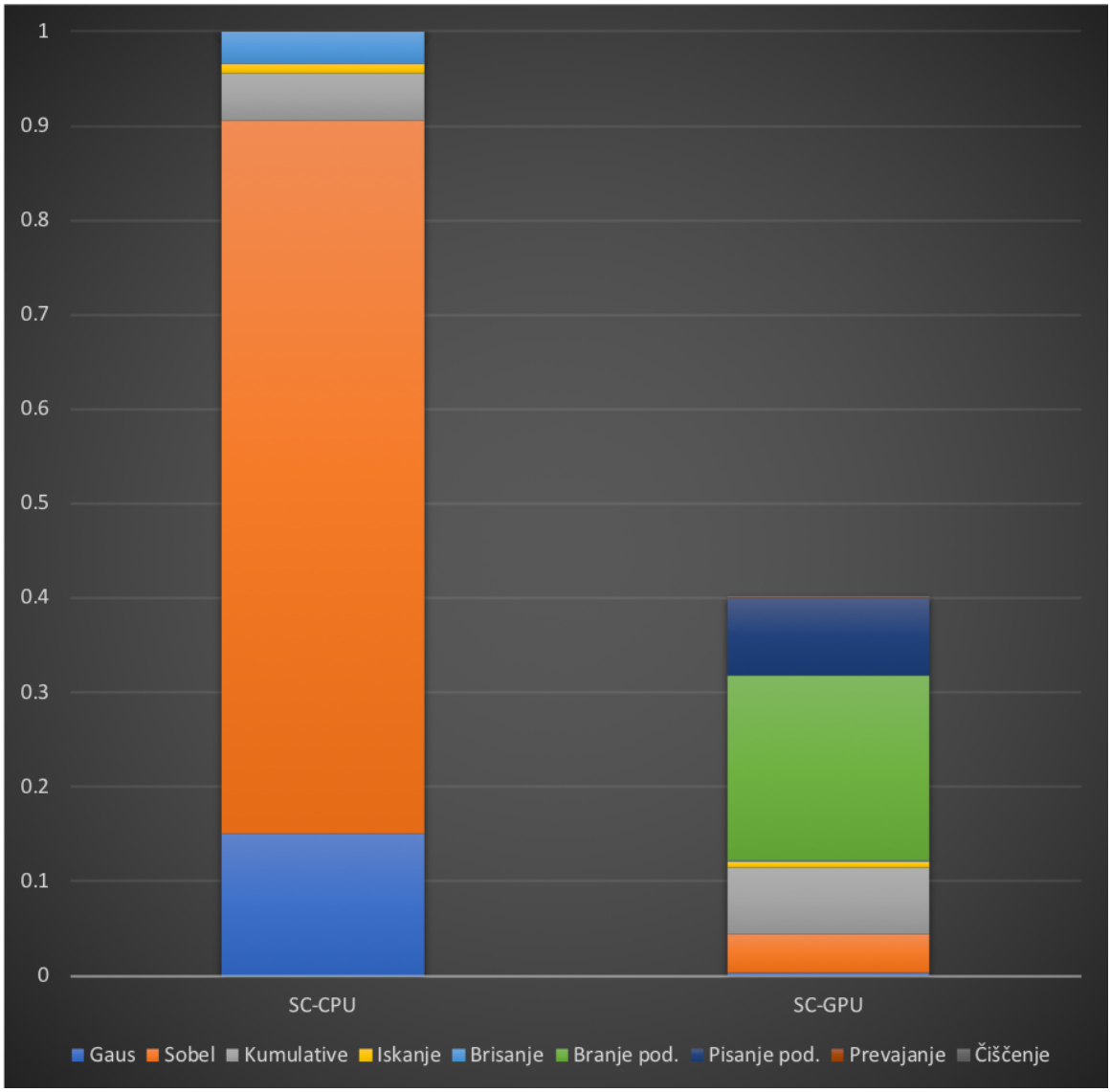
Slika 21: Čas[s] izvajanja serijskega algoritma v odvisnosti od velikosti slike.

B Lokalno merjenje časa izvajanja algoritma

Algoritem	N (širina x višina)	Gauss		Sobel		Kumulative		Brisanje šiva	
		Čas [s]	SE [s]	Čas [s]	SE [s]	Čas [s]	SE [s]	Čas [s]	SE [s]
Serijski	1143000 (1500 x 762)	0,00241	0,00003	0,00973	0,00003	0,00064	0,00001	0,00034	0,00001
	4572000 (3000 x 1524)	0,00764	0,00001	0,03820	0,00001	0,00249	0,00001	0,00174	0,00001
pThread, 2 niti	1143000 (1500 x 762)	0,00108	0,00001	0,00513	0,00005	0,00206	0,00001	0,00023	0,00001
	4572000 (3000 x 1524)	0,00428	0,00004	0,02014	0,00014	0,00482	0,00002	0,00155	0,00001
pThread, 4 niti	1143000 (1500 x 762)	0,00121	0,00003	0,00571	0,00010	0,00319	0,00001	0,00025	0,00001
	4572000 (3000 x 1524)	0,00436	0,00008	0,01822	0,00053	0,00675	0,00003	0,00154	0,00001
OpenMP, 2 niti	1143000 (1500 x 762)	0,00362	0,00004	0,00810	0,00007	0,00070	0,00001	0,00063	0,00001
	4572000 (3000 x 1524)	0,01392	0,00002	0,03115	0,00003	0,00253	0,00001	0,00235	0,00001
OpenMP, 4 niti	1143000 (1500 x 762)	0,00247	0,00004	0,00540	0,00009	0,00081	0,00001	0,00045	0,00001
	4572000 (3000 x 1524)	0,00815	0,00011	0,01788	0,00021	0,00287	0,00004	0,00169	0,00001
GPU, 32 snip	1143000 (1500 x 762)	0,0001707	/	0,0020890	/	0,00351	/	0,0000784	/

Tabela 5: Čas, ki ga posamezen algoritem porabi v različnih delih algoritma in standardna napaka meritve (SE).

C GPE podrobne meritve



Slika 22: Podrobne meritve izvajanja agloritma na GPE.

D Lokalen čas izvajanja in pohitritve MPI algoritma

Število procesov	N (širina x višina)	MPI		
		Čas $[s]$	S	E
2	127000 (500 x 254)	1,184	0,281	0,140
	508000 (1000 x 508)	7,023	0,244	0,122
	1143000 (1500 x 762)	17,815	0,239	0,119
	4572000 (3000 x 1524)	76,040	0,241	0,120
	18294000 (6000 x 3049)	319,546	0,238	0,119
4	127000 (500 x 254)	0,939	0,355	0,089
	508000 (1000 x 508)	6,039	0,284	0,071
	1143000 (1500 x 762)	16,442	0,259	0,065
	4572000 (3000 x 1524)	70,112	0,261	0,065
	18294000 (6000 x 3049)	294,974	0,258	0,064
8	127000 (500 x 254)	1,040	0,320	0,040
	508000 (1000 x 508)	6,203	0,276	0,034
	1143000 (1500 x 762)	17,138	0,248	0,031
	4572000 (3000 x 1524)	77,988	0,235	0,029
	18294000 (6000 x 3049)	333,365	0,228	0,028

Tabela 6: Čas izvajanja, pohitritev (S) in učinkovitost (E) MPI algoritma v odvisnosti od velikosti slike *waterfall.jpg* in števila procesov. Meritve so bile izvedene lokalno.