Dokumentacija za projekat iz paralelnog programiranja

Student: Luka Petković SV16/2021

Tema: Detekcija ivica unutar slike

Sadržaj

Uvod i analiza problema

Digitalna obrada slike je proces manipulacije i analize slika pomoću računara. Uključuje primenu različitih tehnika i algoritama radi poboljšanja kvaliteta slike, izdvajanja ili izmena određenih karakteristika slike, kao i naprednih merenja i analiza slika.

Detekcija ivica unutar slika predstavlja osnovni korak u digitalnoj obradi slike. Cilj detekcije ivica je identifikacija prelaza intenziteta piksela koji označavaju promene u strukturi slike. Mesto na slici gde se intenzitet naglo menja je ivica, koja često ukazuje na granice između objekata ili regiona unutar slike.

Detekcija ivica ima značajnu ulogu u mnogim aplikacijama obrade slika, kao što su segmentacija objekata, prepoznavanje oblika i praćenje pokreta. Postoji veliki broj algoritama koji rešavaju ovaj problem, a u ovom projektu su korišćena dva algoritma: algoritam sa Prewitt operatorom, koji je jedan od popularnijih, i algoritam koji pretražuje okolinu svakog piksela.

Sva testiranja, analize i sam projekat su realizovani u programskom jeziku C++, uz upotrebu Intel-ove biblioteke Thread Building Blocks (TBB) za paralelizaciju algoritama i EasyBMP biblioteke za parsiranje slika.

Koncept rešenja

Na temelju digitalne obrade slike leži operacija dvodimenzionalne diskretne konvolucije. Jednostavno rečeno, ova operacija se može opisati kao množenje odgovarajućih elemenata dveju matrica i sumiranje rezultata. Prva matrica u konvolucijskoj operaciji je jezgro digitalnog filtra (kvadratna matrica neparnih dimenzija (3x3, 5x5, 7x7…)). Druga matrica je podmatrica slike koja sadrži podatke o vrednostima boje pojedinih tačaka (odnosno piksela) slike i ima iste dimenzije kao odabrani filter. Drugim rečima, cela operacija se može promatrati kao pomeranje matrice filtra preko svih podmatrica slike i izvođenje aritmetičkih operacija nad svakim parom koji se na taj način stvori.

Algoritam sa Prewitt operatorom

Filter koji je korišćen u ovom algoritmu je Prewitt operator. Računanje gradijenta intenziteta piksela se izvršava pomoću konvolucije. Gradijent predstavlja promenu intenziteta piksela u horizontalnom i vertikalnom smeru. Filtriranje slike koja se nalazi u matrici X, filtrom jezgra F (u ovom primeru dimenzija 2x2) se dobija pomoću sledeće formule, i njen rezultat se smešta u matricu Y.

$$Y[x,y] = \sum_{n=0}^{2} \sum_{m=0}^{2} X[x-1+m, y-1+n] * F[m,n]$$

Ulaznu sliku je potrebno filtrirati operatorima za horizontalne, odnosno vertikalne ivice, a rezultat (tačka izlazne slike) se računa pomoću zbira apsolutnih vrednosti horizontalne i vertikalne komponente. Gx – horizontalan operator, Gy – vertikalan operator

$$Gx = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} * A \qquad Gy = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} * A$$

$$|G| = |Gx| + |Gy|$$

Algoritam pretrage okoline svakog piksela

Drugi način za detekciju ivica unutar slike je algoritam pretrage okoline pojedinačnih piksela. Da bi algoritam radio, neophodno je raditi na crno-beloj slici bez izdvojenih ivica, koju ćemo dobiti postavljanjem vrednosti ulazne matrice na 0 ili 1 odsecanjem praga koji je u ovom slučaju T = 128 (0 ako je vrednost manja ili jednaka 128, inače 1) . Okolinu piksela čine njegovi susedni pikseli, odnosno pikseli koji okružuju posmatrani piksel sa svih strana. Pretragu okoline piksela vršimo na sledeći način:

- P(i,j) = 1, ako u okolini tačke postoji tačka sa vrednošću 1
- P(i,j) = 0, ako u okolini tačke ne postoji tačka sa vrednošću 1
- O(i,j) = 0, ako u okolini tačke postoji tačka sa vrednošću 0
- O(i,j) = 1, ako u okolini tačke ne postoji tačka sa vrednošću 0

Rezultujuća vrednost je zapravo razliku apsolutnih vrednosti P(i,j) I O(i,j) koju vraćamo u opseg 0 – 255 tako što vršimo odsecanje praga.

Programsko rešenje

Za početak moramo odrediti parametre komandne linije kako bi kompajler znao nad kojim fajlovima se izvršava algoritam:

- input.bmp (ulazni fajl)
- outputSerialPrewitt.bmp (izlazni fajl za serijsku verziju algoritma sa Prewitt operatorom)
- outputParallelPrewitt.bmp (izlazni fajl za paralelnu verziju algoritma sa Prewitt operatorom)
- outputSerialEdge.bmp (izlazni fajl za serijsku verziju algoritma sa pretragom okoline)
- outputParallelEdge.bmp (izlazni fajl za paralelnu verziju algoritma sa pretragom okoline)

Konstante koje sam definisao u zaglavlju:

- FILTER_SIZE (dimenzije filtera)
- filterHor i filterVer (matrice [FILTER_SIZE*FILTER_SIZE] - vrednosti horizontalne i vertikalne komponente filtera)
- SURROUND_SIZE (veličina okruženja koje se posmatra, broj okoline piksela)
- CUTOFF (najmanje dimenzije matrice do kog se kreiraju novi paralelni zadaci)
- SKIP_PREWITT (broj piksela na okviru koji će se preskočiti kod primene Prewitt operatora)
- SKIP_SURROUND (broj piksela na okviru koji će se preskočiti kod primene pretrage okoline piksela)

Ove konstante možemo menjati po želji. Vreme merimo i za serijsku i za paralelnu implementaciju kod oba algoritma.

Implementacija algoritma sa Prewitt operatorom

Filter metoda

```
/**
* @brief Prewitt operator on input image submatrix around pixel
*
* @param inBuffer buffer of input image
* @param x horizontal coordinate of pixel
* @param y vertical coordinate of pixel
*
* @return scaled value of operator result
*/
int filter(int* inBuffer, int x, int y) {

	int save = SKIP_PREWITT - 1;
	int G = 0;
	int GX = 0;
	int GY = 0;
	int raw;
	for (int n = 0; n < FILTER_SIZE; n++) {
		for (int m = 0; m < FILTER_SIZE; m++) {
			raw = inBuffer[(x - save + m) + (y - save + n) * totalWidth];
			GX += raw * filterHor[m + n * FILTER_SIZE];
			GY += raw * filterVer[m + n * FILTER_SIZE];
		}
	}
	G = sqrt(GX * GX + GY * GY);
	return 255 * scale(G);
}
```

Funkcija filter_image kao parameter prima ulazni bafer i koordinate piksela. Kao što joj i ime kaže ona se koristi za filtriranje piksela. Kao povratnu vrednost funkcija nakon odsecanja praga vraća 0 ili 1 u zavisnosti od toga da li je piksel ivica ili nije. Za odsecanje praga se koristi jednostavna metoda scale koja vraća 0 ili 1 u zavisnosti od toga da li je uneta vrednost premašila prag ili ne.

Filter serial Prewitt

```
void filter_serial_prewitt(int* inBuffer, int* outBuffer, int width, int height, int col=0, int row=0)
{
	int lowerX, lowerY, upperX, upperY;

	if (col < SKIP_PREWITT)
		lowerX = SKIP_PREWITT;
	else
		lowerX = col;
	if (row < SKIP_PREWITT)
		lowerY = SKIP_PREWITT;
	else
		lowerY = row;
	if (col + width > totalWidth - (SKIP_PREWITT))
		upperX = totalWidth - (SKIP_PREWITT);
	else
		upperX = col + width;
	if (row + height > totalHeight - (SKIP_PREWITT))
		upperY = totalHeight - (SKIP_PREWITT);
	else
		upperY = row + height;


	for (int x = lowerX; x < upperX; x++) {
		for (int y = lowerY; y < upperY; y++) {
			outBuffer[x + y * totalWidth] = filter(inBuffer, x, y);
		}
	}
}
```

Prvo se kreiraju granice za pristup kako se ne bi izašlo van opsega matrice i pristupilo pogrešnoj memorijskoj lokaciji. Zatim se prolazi kroz cijelu matricu i filtriranje podmatrica se smešta u odgovarajuće elemente izlazne matrice.

Filter parallel Prewitt

```
void filter_parallel_prewitt(int* inBuffer, int* outBuffer, int width, int height, int col=0, int row=0)
{
	if (min(height, width) <= CUTOFF) {
		filter_serial_prewitt(inBuffer, outBuffer, width, height, col, row);
	}

	else {
		task_group t;
		int taskHeight = height / 2;
		int restHeight = height - taskHeight;
		int taskWidth = width / 2;
		int restWidth = width - taskWidth;
		t.run([&] {filter_parallel_prewitt(inBuffer, outBuffer, taskWidth, taskHeight, col, row); });
		t.run([&] {filter_parallel_prewitt(inBuffer, outBuffer, restWidth, taskHeight,
col+taskWidth, row); });
		t.run([&] {filter_parallel_prewitt(inBuffer, outBuffer, taskWidth, restHeight, col,
row+taskHeight); });
		t.run([&] {filter_parallel_prewitt(inBuffer, outBuffer, restWidth, restHeight,
col+taskWidth, row+taskHeight); });
		t.wait();
		}

}
```

Prolazak kroz matricu se može paralelizovati tako što se cela matrica podeli na manje podmatrice, nad kojima se vrši obrada. Svaka podmatrica može da se deli dalje do određene granice (CUTOFF) tako da se postiže veća paralelizacija. Za svaku podmatricu zadužen je jedan zadatak raspoređivača.

Implementacija algoritma sa pretragom okoline svakog piksela

Check surrounding

Funkcija **checkSurrounding** kao parametre prima ulazni bafer i koordinate piksela. Ona pretražuje okolinu piksela do određene veličine (SURROUND_SIZE) i vraća 0 ili 1.

```
int checkSurrounding(int* outBuffer, int x, int y) {
	int P = 0;
	int O = 1;
	int value;
	for (int i = 0; i < SURROUND_SIZE; i++) {
		for (int j = 0; j < SURROUND_SIZE; j++) {
			value = scale(outBuffer[(x - i + SKIP_SURROUND) + (y - j + SKIP_SURROUND)
* totalWidth]);
			if (value == 1) {
				P = 1;
			}
			else {
				O = 0;
			}
	}
}
return 255 * abs(P - O);
}
```

Serijska verzija funkcije sa pretragom okoline svakog piksela

Funkcija filter_serial_edge_detection radi na istom principu kao i funkcija filter_serial_prewitt, jer se takođe na početku podešavaju red i kolona, a jedina razlika je u tome što se umesto funkcije filter koristi nova funkcija za pretraživanje i filtriranje okoline checkSurrounding.

```
void filter_serial_edge_detection(int* inBuffer, int* outBuffer, int width, int height, int col=0, int
row=0)
{
	int lowerX, lowerY, upperX, upperY;
	if (col < SKIP_SURROUND)
		lowerX = SKIP_SURROUND;
	else
		lowerX = col;
	if (row < SKIP_SURROUND)
		lowerY = SKIP_SURROUND;
	else
		lowerY = row;
	if (col + width > totalWidth - (SKIP_SURROUND))
		upperX = totalWidth - (SKIP_SURROUND);
	else
		upperX = col + width;
	if (row + height > totalHeight - (SKIP_SURROUND))
		upperY = totalHeight - (SKIP_SURROUND);
	else
		upperY = row + height;

	for (int x = lowerX; x < upperX; x++) {
		for (int y = lowerY; y < upperY; y++) {
			outBuffer[x + y * totalWidth] = checkSurrounding(inBuffer, x, y);
		}
	}
}
```

Paralelna verzija funkcije sa pretragom okoline svakog piksela

Funkcija se paralelizuje na identičan način kao i funkcija koja koristi Prewitt operator. Sledi njen kod

```
void filter_parallel_edge_detection(int* inBuffer, int* outBuffer, int width, int height, int col = 0, int row = 0)
{
	if (min(height, width) <= CUTOFF) {
		filter_serial_edge_detection(inBuffer, outBuffer, width, height, col, row);
	}

	else {
		task_group t;
		int taskHeight = height / 2;
		int restHeight = height - taskHeight;
		int taskWidth = width / 2;
		int restWidth = width - taskWidth;
		t.run([&] {filter_parallel_edge_detection(inBuffer, outBuffer, taskWidth, taskHeight, col, row); });
		t.run([&] {filter_parallel_edge_detection(inBuffer, outBuffer, restWidth, taskHeight, col + taskWidth, row); });
		t.run([&] {filter_parallel_edge_detection(inBuffer, outBuffer, taskWidth, restHeight, col, row + taskHeight); });
		t.run([&] {filter_parallel_edge_detection(inBuffer, outBuffer, restWidth, restHeight, col + taskWidth, row + taskHeight); });
		t.wait();
}

}
```

Ispitivanje algoritama

Informacije o računaru

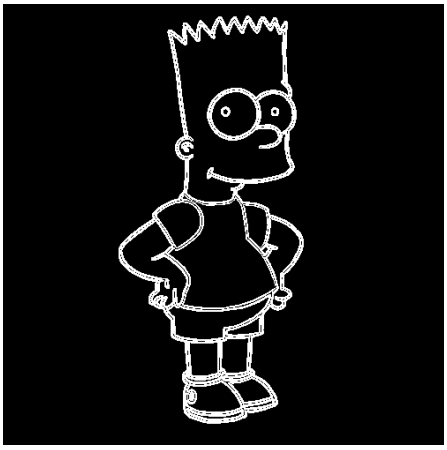
Sva testiranja, merenja i računanja vremena, kao i celokupan kod, vršio sam na laptopu sa **AMD** procesorom **Ryzen 7 5700U** koji sadrži 8 jezgara, i baznu frekvenciju od 1.8Ghz.

Za operativni sistem korišćen je Windows 11

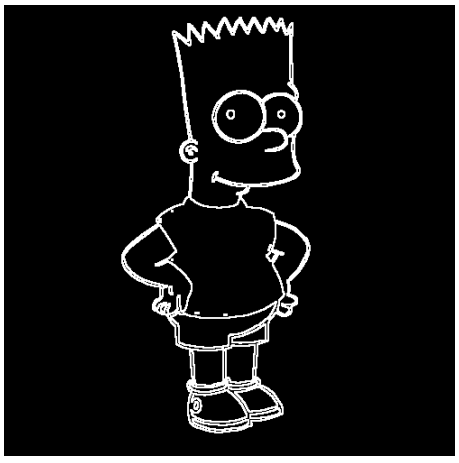
Testiranje

Prikazaćemo rezultate za 4 različite slike od jednostavnijih ka složenijim

Bart Simpson



Bart Simpson (Prewitt)



Bart Simpson (pretraživanje okoline)

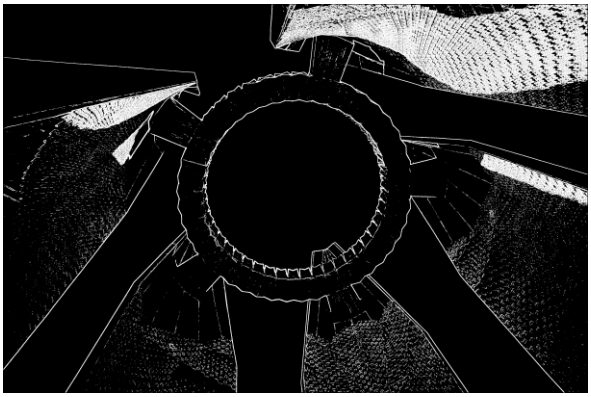
Veličina slike: 500x500px

Veličina filtera: 3x3

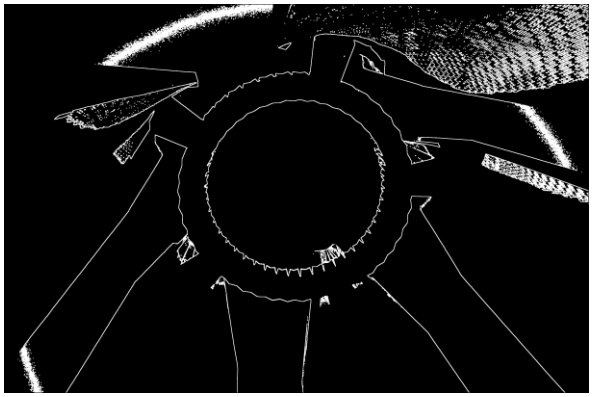
Veličina okoline: 3

Slika je relativno mala, i svi detalji na njoj su jasno definisani

Architecture



Architecture (Prewitt)



Architecture (pretraživanje okoline)

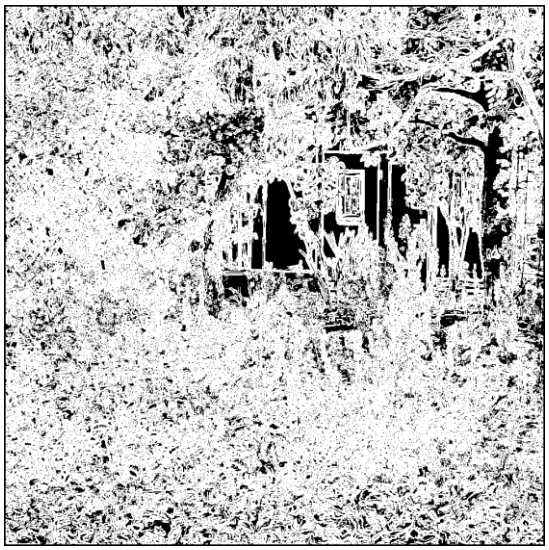
Veličina slike: 3888x2592px

Veličina filtera: 5x5

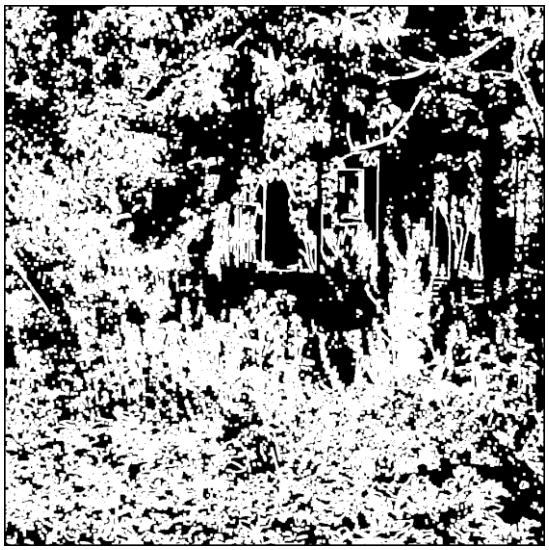
Veličina okoline: 5

Slika je mnogo veća, i sadrži puno detalja, tako da je i vreme izvršavanja bilo duže

Garden



Garden(Prewitt)



Garden (pretraživanje okoline)

Veličina slike: 1000x1000px

Veličina filtera: 5x5

Veličina okoline: 5

Slika je malo manja od prethodne, ali sadrži još više detalja. Ipak, vreme izvršavanja je relativno malo

Dog

Dog (Prewitt)

Dog (pretraživanje okoline)

Veličina slike: 1200x1600px

Veličina filtera: 5x5

Veličina okoline: 5

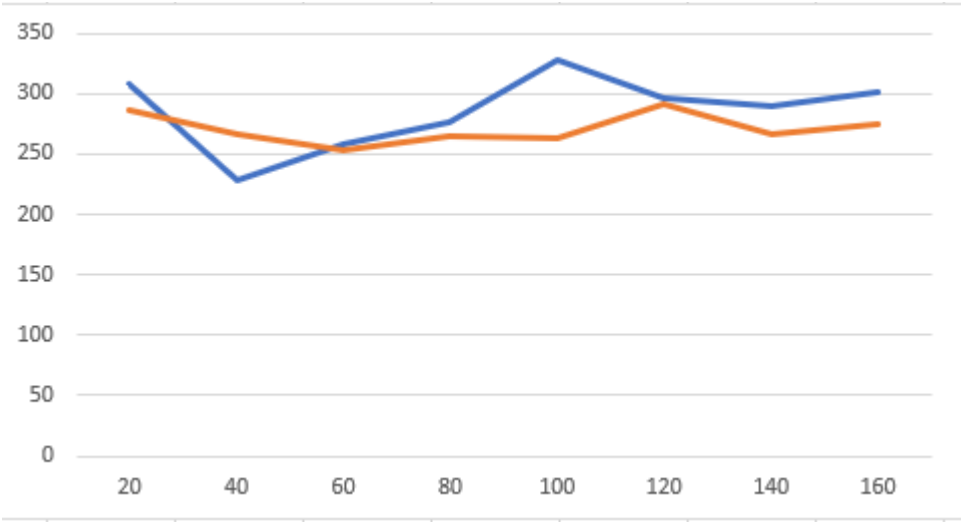
Na slici nisu toliko jasno izražene ivice. Možemo primetiti da je Prewitt algoritam pokazao bolje rezultate

Grafici i rezultati

Za početak, posmatraćemo vreme izvršavanja **paralelnih** algoritama u zavisnosti od vrednosti **CUTOFF** konstante. Koristićemo sliku dimenzija 3888x2592 sa veličinom filtera i okruženja – 5. Rezultate prikazujemo na tabeli (u sekundama)

CUTOFF	20	40	60	80	100	120	140	160
Prewitt	0.3081	0.2287	0.2585	0.2760	0.3282	0.2975	0.2905	0.3028
Edge	0.2877	0.2671	0.2544	0.2652	0.2632	0.2911	0.2678	0.2742

Možemo primetiti da nema prevelikih razlika u brzini izvršavanja, i da je algoritam približno najbrži za vrednosti cutoff-a koji su između 40 i 60. Stoga ćemo koristiti vrednost 50 za CUTOFF u narednim analizama. Prikazujemo vrednosti na grafiku (u milisekundama).



Nakon toga, posmatramo vrednosti vremena izvršavanja paralelnih algoritama u zavisnoti od veličine filtera kao i okruženja. Dobijena su sledeća merenja

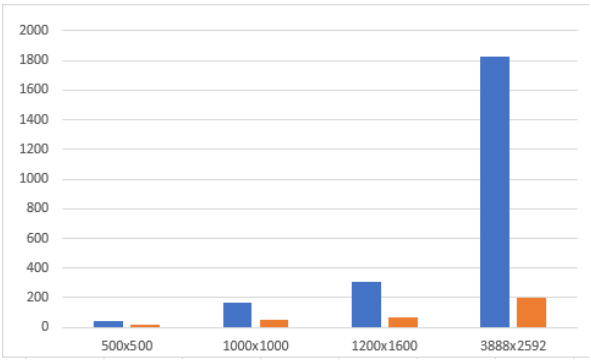
Filter/okruženje	3	5	7
Prewitt	0.1839	0.3128	0.4053
Edge	0.1439	0.2774	0.4121

Lako se primećuje da algoritmi najbrže rade za najmanju veličinu filtera i okruženja

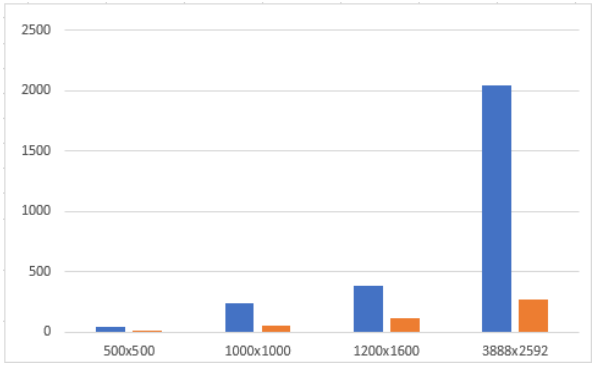
Za kraj, analiziramo vreme izvršavanja sva 4 algoritma sa istim veličinama filtera i okruženja, kao i cutoff-ova ali u zavisnosti od dimenzija slike. Stanje izgleda ovako

Dimenzije slike	500x500	1000x1000	1200x1600	3888x2592
Prewitt serijski	0.0478	0.1640	0.3066	1.8286
Prewitt paralelni	0.0183	0.0511	0.0701	0.1998
Edge serijski	0.0443	0.2414	0.3805	2.0479
Edge paralelni	0.0124	0.0596	0.1216	0.2754

Kao što smo i očekivali, vreme izvršenja se povećava sa povećanjem dimenzija slike. Naravno, izvršenja paralelnih algoritama su dosta manja od vremena izvršenja serijskih, pogotovo kod većih slika, što je i bio krajnji cilj.



Prewitt algoritam



Edge algoritam

Analiza rezultata

Iz svih navedenih rezultata možemo zaključiti da vreme izvršavanja programa kao i ubrzanje kod paralelizacije znatno zavisi od dimenzija slike, vrednosti parametra CUTOFF, veličine filtera, kao i veličine okruženja. Vremena nikada neće biti potpuno ista, čak i kad više puta pokrenemo isti program, ali te razlike su neprimetne. Uspeli smo da dostignemo ubrzanje od 3 do 4 puta (u zavisnosti od parametara), što smo i očekivali, tako da se za paralelizaciju može reći da je uspešna.