SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

SEMINARSKI RAD

Napredne računalne arhitekture

SOBEL FILTER

Ivan Akrapović

Jozo Krstanović

Dominik Mišadin

Split, lipanj 2025.

SADRŽAJ

1. UVOD

U današnje vrijeme obrada slike ima široku primjenu u različitim područjima, poput medicinske dijagnostike, autonomnih vozila, nadzornih sustava, umjetne inteligencije i mnogih drugih. S obzirom na količinu podataka koja se obrađuje u realnom vremenu, performanse sustava postaju od ključne važnosti. Tradicionalno, obrada slike se izvodila na procesorima (CPU), no kako su zahtjevi za brzinom i paralelizmom rasli, sve se češće koristi grafički procesor (GPU) za paralelnu obradu.

Jedna od osnovnih operacija u obradi slike je detekcija rubova, kojom se ističu promjene u intenzitetu piksela. Detekcija rubova koristi se za segmentaciju objekata, pronalaženje oblika i izdvajanje značajki. U ovom seminaru fokusirat ćemo se na implementaciju Sobel filtera na GPU-u korištenjem CUDA tehnologije. Usporedit ćemo izvedbu CPU i GPU verzije algoritma, analizirati dobivene rezultate i prikazati slike rezultata.

2. CUDA

CUDA (Compute Unified Device Architecture) je razvojna platforma i API koji je razvila tvrtka NVIDIA za paralelno programiranje na njihovim grafičkim karticama. CUDA omogućuje programerima da pišu programe u C/C++ jeziku i izvršavaju ih na GPU-u, čime se postižu značajna ubrzanja kod zadataka koji se mogu paralelizirati.

2.1. Osnovni pojmovi

- __global__: označava funkciju (tzv. "kernel") koja se izvršava na GPU-u, ali se poziva s CPU-a.
- threadIdx, blockIdx, blockDim, gridDim: ugrađene CUDA varijable koje identificiraju položaj niti (eng. threads) unutar bloka i bloka unutar mreže (eng. grid).
- dim3: CUDA tip koji definira dimenzije mreže i blokova u 1D, 2D ili 3D.

2.2. Struktura izvršavanja

Kada se kernel poziva s CPU-a, koristi se sintaksa:

```
myKernel<<<numBlocks, numThreads>>>(arguments);
```

Svaka nit izvršava identičan kod, ali na različitim dijelovima podataka. Na taj način se postiže visoka razina paralelizma, idealna za obradu slike gdje se nad svakim pikselom može izvršiti ista operacija neovisno od ostalih.

U nastavku ćemo objasniti Sobel filter te prikazati implementaciju u C++/CUDA jeziku.

3. SOBEL FILTER

Sobel filter je algoritam za detekciju rubova u slici, temeljen na izračunu gradijenta intenziteta piksela. Koristi se za pronalaženje područja slike gdje dolazi do nagle promjene intenziteta, što često odgovara rubovima objekata. Na slici 3.1 prikazana je usporedba originalne slike [2] i što se dobije kao rezultat provedbe Sobel filtra.

Slika 3.1. Originalna slika (lijevo) i detektirani rubovi (desno)

Filter koristi dvije 3x3 matrice (kernel), jednu za detekciju promjena u horizontalnom smjeru (G_X), a drugu u vertikalnom smjeru (G_Y):

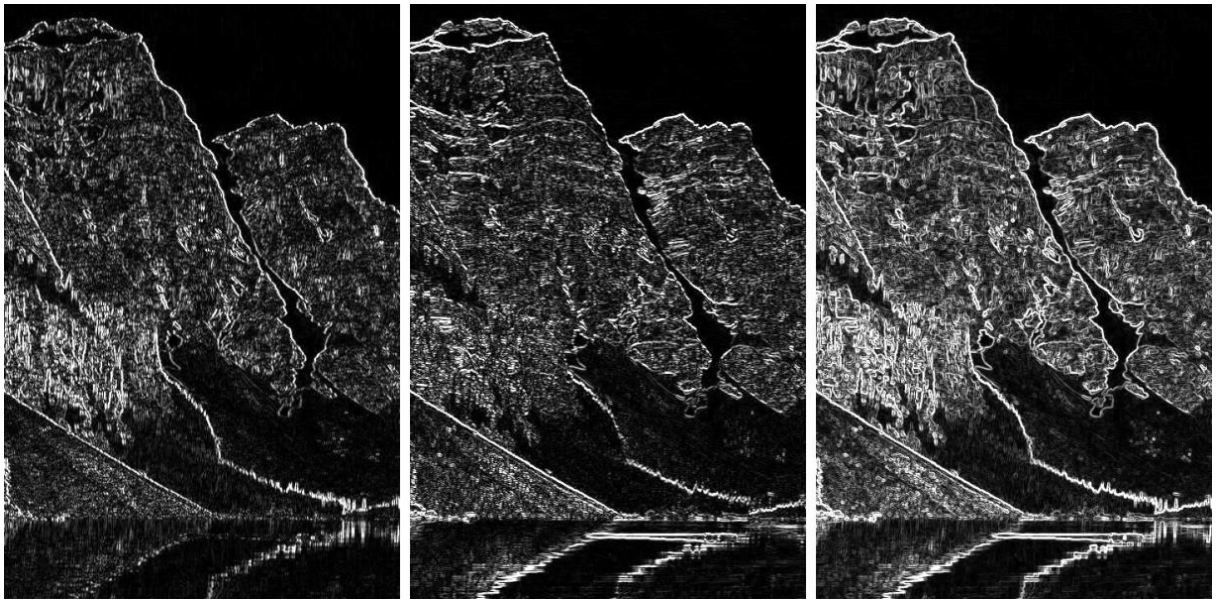
$$G_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_Y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Za svaki piksel u slici, izračunava se horizontalni i vertikalni gradijent, a zatim se ukupna magnituda izračuna formulom:

$$magnitude = \sqrt{g_x^2 + g_y^2}$$

Magnituda označava jačinu ruba u tom pikselu. Rezultat filtra je nova slika u sivim tonovima koja naglašava rubove objekata. Ovaj pristup je osjetljiv na šum, ali daje jasne rezultate u

detekciji pravaca i struktura. Izgled zasebnih prolaza po X i Y-osima mogu se vidjeti na slici 3.2.



Slika 3.2. Usporedba prolaza X-osi (lijevo) i Y-osi (sredina) te prolaz sa spojenom magnitudom (desno)

U sljedećem poglavlju prikazat ćemo kako se Sobel filter implementira u C++ korištenjem CUDA paralelizacije.

4. IMPLEMENTACIJA

4.1. Učitavanje slike i priprema podataka

Programski kod dostupan na GitHubu [1], započinje učitavanjem slike pomoću OpenCV biblioteke:

```
Mat image = imread("large_image.jpg", IMREAD_COLOR);
if (image.empty()) {
    cerr << "Greska: Slika nije ucitana!" << endl;
    return -1;
}

int width = image.cols;
int height = image.rows;
```

Ovdje se slika učitava u RGB formatu, a zatim se dohvaćaju dimenzije slike. Učitani podaci koriste se i u CPU i GPU implementaciji.

4.2. Konverzija slike u sivu nijansu pomoću CUDA kernela

Prije primjene Sobelovog filtra, potrebno je sliku pretvoriti u crno-bijelu. Za to se koristi CUDA kernel koji paralelno obrađuje svaki piksel:

```
__global__ void rgb2gray(const uchar3* input, unsigned char* gray,
                         int width, int height)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x >= width || y >= height) return;

    int idx = y * width + x;
    uchar3 pixel = input[idx];
    gray[idx] = static_cast<unsigned char>(0.299f * pixel.x + 0.587f *
                                           pixel.y + 0.114f * pixel.z);
}
```

Ovdje je uchar3 struktura koja sadrži tri komponente piksela (RGB). Kernel koristi standardnu formulu za pretvorbu boje u sivu nijansu:

```
0.299f * pixel.x + 0.587f * pixel.y + 0.114f * pixel.z
```

Varijable blockIdx, threadIdx i blockDim koriste se za izračun globalnog indeksa piksela u slici. Svaka nit obrađuje jedan piksel. Ako je izračunati indeks izvan granica slike, kernel se odmah prekida. Ova funkcija se poziva iz glavnog programa pomoću:

```
dim3 block(16, 16);
dim3 grid((width + 15) / 16, (height + 15) / 16);
rgb2gray <<<grid, block>>> (d_input, d_gray, width, height);
cudaDeviceSynchronize();
```

Blokovi i mreža dimenzionirani su tako da pokrivaju cijelu sliku, a svaka nit izvršava operaciju nad jednim pikselom. Funkcija cudaDeviceSynchronize() osigurava da su sve niti završile prije nego se nastavi s programom.

4.3. Sobelov filter na CPU-u

Za usporedbu s GPU implementacijom, Sobelov filter je implementiran i na CPU-u korištenjem dvostruke petlje. U nastavku se nalazi funkcija koja ručno računa gradijente i magnitudu:

```
void sobelFilterCPU(const cv::Mat& gray, cv::Mat& output) {
    int width = gray.cols;
    int height = gray.rows;

    output = cv::Mat::zeros(height, width, CV_8U);

    for (int y = 1; y < height - 1; ++y) {
        for (int x = 1; x < width - 1; ++x) {
            int gx = - gray.at<uchar>(y - 1, x - 1)
                     - 2 * gray.at<uchar>(y, x - 1)
                     - gray.at<uchar>(y + 1, x - 1)
                     + gray.at<uchar>(y - 1, x + 1)
                     + 2 * gray.at<uchar>(y, x + 1)
                     + gray.at<uchar>(y + 1, x + 1);

            int gy = - gray.at<uchar>(y - 1, x - 1)
                     - 2 * gray.at<uchar>(y - 1, x)
                     - gray.at<uchar>(y - 1, x + 1)
                     + gray.at<uchar>(y + 1, x - 1)
                     + 2 * gray.at<uchar>(y + 1, x)
                     + gray.at<uchar>(y + 1, x + 1);

            int magnitude = sqrtf((float)(gx * gx + gy * gy));

            output.at<uchar>(y, x) = magnitude > 255 ? 255 : magnitude;
        }
    }
}
```

4.4. Sobelov filter na GPU-u

Implementacija na GPU-u koristi CUDA kernel za paralelno računanje gradijenata. Svaka nit obrađuje jedan piksel slike:

```
__global__ void sobelFilter(const unsigned char* gray,
                            unsigned char* output, int width, int height)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x <= 0 || x >= width - 1 || y <= 0 || y >= height - 1) return;

    int idx = y * width + x;

    int gx = - gray[(y - 1) * width + (x - 1)]
             - 2 * gray[y * width + (x - 1)]
             - gray[(y + 1) * width + (x - 1)]
             + gray[(y - 1) * width + (x + 1)]
             + 2 * gray[y * width + (x + 1)]
             + gray[(y + 1) * width + (x + 1)];

    int gy = - gray[(y - 1) * width + (x - 1)]
             - 2 * gray[(y - 1) * width + x]
             - gray[(y - 1) * width + (x + 1)]
             + gray[(y + 1) * width + (x - 1)]
             + 2 * gray[(y + 1) * width + x]
             + gray[(y + 1) * width + (x + 1)];

    int magnitude = sqrtf((float)(gx * gx + gy * gy));

    output[idx] = magnitude > 255 ? 255 : magnitude;
}
```

Kernel koristi istu logiku kao i CPU verzija, ali se izvodi paralelno za svaki piksel. Ograničenja `if (x <= 0 || x >= width - 1...)` sprječavaju pristup izvan granica slike. Svaka nit koristi indeks `idx` da pročita i napiše podatke iz globalne memorije.

Kernel se pokreće s istim grid i block dimenzijama kao kod konverzije u sivu nijansu:

```
rgb2gray<<<grid, block>>>(d_input, d_gray, width, height);
cudaDeviceSynchronize();

sobelFilter<<<grid, block>>>(d_gray, d_output, width, height);
cudaDeviceSynchronize();
```

Poziv funkcije i sinkronizacija osiguravaju da se svi rezultati završe prije povratka na CPU.

4.5. Izvođenje i spremanje rezultata

Za analizu performansi implementacije koristi se standardna C++ biblioteka chrono, koja omogućuje mjerenje vremena izvođenja pojedinih dijelova koda:

```
Mat gray_cpu, sobel_cpu;
auto t1 = chrono::high_resolution_clock::now();
cvtColor(image, gray_cpu, COLOR_BGR2GRAY);
sobelFilterCPU(gray_cpu, sobel_cpu);
auto t2 = chrono::high_resolution_clock::now();
cout << "CPU vrijeme: "
     << chrono::duration_cast<chrono::microseconds>(t2 - t1).count()
     << " µs" << endl;
```

Za GPU verziju, vrijeme se mjeri oko poziva CUDA kernela:

```
auto t3 = chrono::high_resolution_clock::now();
rgb2gray<<<grid, block>>>(d_input, d_gray, width, height);
cudaDeviceSynchronize();

sobelFilter<<<grid, block>>>(d_gray, d_output, width, height);
cudaDeviceSynchronize();
auto t4 = chrono::high_resolution_clock::now();
cout << "GPU vrijeme: "
     << chrono::duration_cast<chrono::microseconds>(t4 - t3).count()
     << " µs" << endl;
```

Prije izvođenja kernela, podaci slike moraju biti prebačeni u memoriju GPU-a:

```
cudaMalloc(&d_input, numPixels * sizeof(uchar3));
cudaMalloc(&d_gray, numPixels * sizeof(unsigned char));
cudaMalloc(&d_output, numPixels * sizeof(unsigned char));

cudaMemcpy(d_input, image.ptr<uchar3>(), numPixels * sizeof(uchar3),
           cudaMemcpyHostToDevice);
```

Nakon izvršavanja GPU kernela, rezultati se vraćaju natrag u glavnu memoriju:

```
Mat result_gpu(height, width, CV_8U);
cudaMemcpy(result_gpu.ptr(), d_output, numPixels * sizeof(unsigned char),
           cudaMemcpyDeviceToHost);
```

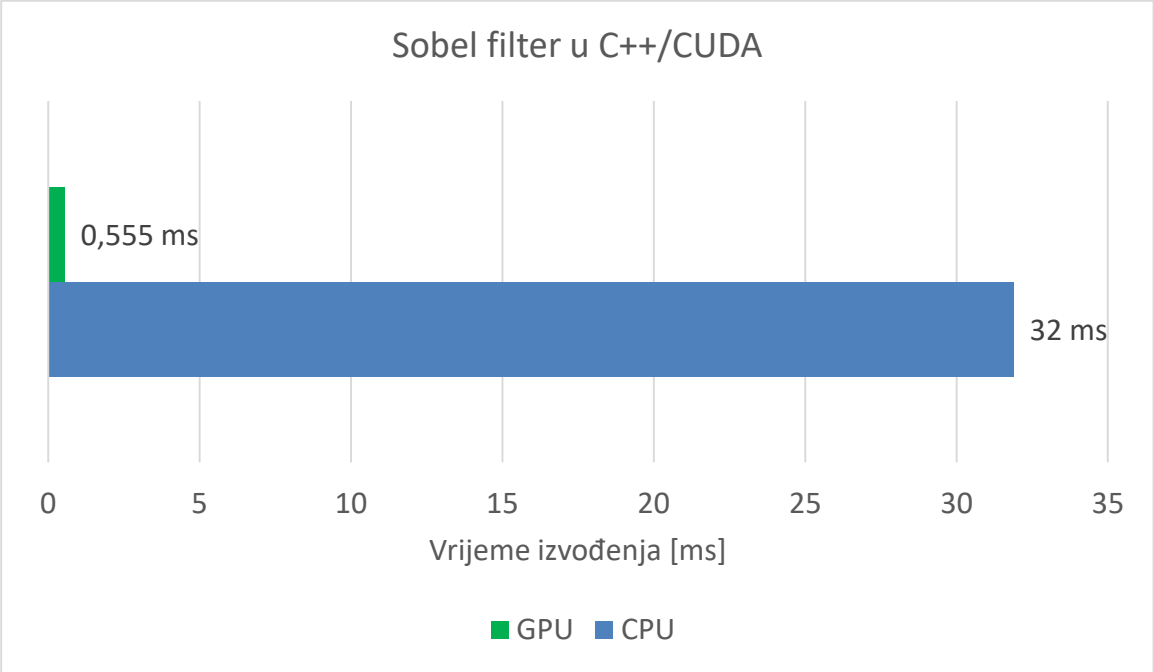
Za kraj, rezultati obrade pohranjuju se u datoteke koristeći OpenCV funkciju imwrite:

```
imwrite("sobel_cpu.jpg", sobel_cpu);
imwrite("sobel_gpu.jpg", result_gpu);
```

Ova implementacija omogućuje jednostavnu usporedbu kvalitete i brzine između CPU i GPU verzije te jasno pokazuje prednosti paralelne obrade slike korištenjem CUDA-e.

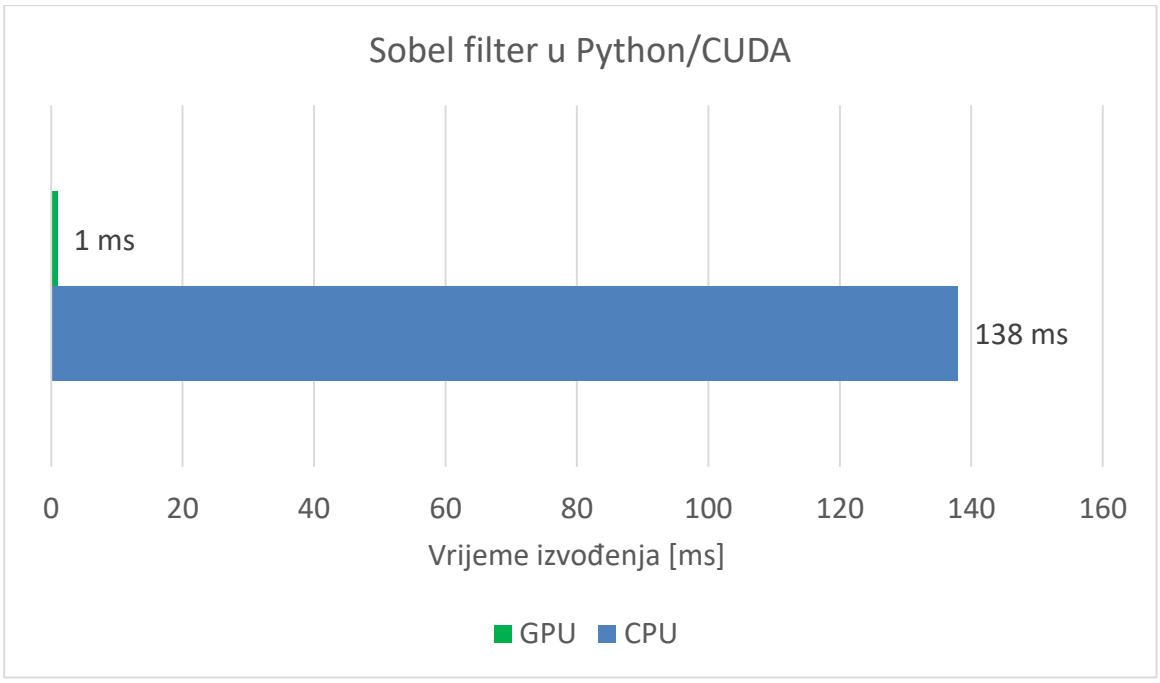
5. REZULTATI

Rezultati izvođenja Sobel filtera na slici rezolucije 3000x2000 pokazali su značajnu prednost GPU-a u odnosu na CPU. Na procesoru Intel Core i9-10850K, koji ima 10 fizičkih jezgri i 20 threadova te radni takt do 5.2 GHz, obrada je trajala približno 32 milisekunde. S druge strane, grafička kartica NVIDIA GeForce RTX 3070, temeljena na Ampere arhitekturi, sadrži 5888 CUDA jezgri i 8 GB GDDR6 memorije, a isti zadatak izvršila je za svega 0.555 milisekundi. Ovi rezultati jasno pokazuju kako paralelizacija koju omogućuje GPU donosi višestruko ubrzanje u obradi slike, što je posebno izraženo kod operacija poput konvolucije, koje se lako distribuiraju po velikom broju jezgri. Takva razlika u performansama naglašava prednosti korištenja GPU-a za računalni vid i slične zadatke koji zahtijevaju intenzivne numeričke izračune nad velikim količinama podataka, što je vidljivo na slici 5.1.



Slika 5.1. Usporedba izvođenja Sobel filtra na CPU i GPU u C++ programskom jeziku

U Python verziji programa pokrenutoj u Jupyter notebooku, obrada slike dimenzija 3000x2000 pomoću Sobel filtera na CPU-u trajala je 138 milisekundi, dok je na GPU-u trajala svega 1 milisekundu. Ova razlika vidljiva je na slici 5.2 te još više ističe prednosti GPU-a u Python okruženju, gdje paralelizacija putem CuPy biblioteke omogućuje višestruko ubrzanje u odnosu na standardnu obradu pomoću NumPy-a i CPU-a.



Slika 5.2. Usporedba izvođenja Sobel filtra na CPU i GPU u Python programskom jeziku

6. ZAKLJUČAK

U ovom radu implementiran je Sobelov filter za detekciju rubova na slici korištenjem CPU i GPU tehnologije, pri čemu je GPU implementacija ostvarena pomoću CUDA programiranja, dok je paralelno razvijena i Python verzija koristeći odgovarajuću GPU-akceleriranu biblioteku. Analiza rezultata pokazala je značajna ubrzanja prilikom korištenja GPU-a zahvaljujući mogućnosti paralelne obrade velikog broja piksela istovremeno.

Na testnoj slici visoke rezolucije (3000x2000 piksela), CPU implementacija izvršila je obradu u vremenu od približno 32 milisekunde u C++ verziji te oko 138 milisekundi u Pythonu. S druge strane, GPU implementacija postigla je vrijeme obrade od svega 0.555 milisekundi u C++ i 1 milisekundu u Python okruženju. To znači da je GPU pristup bio između 80 i 130 puta brži od CPU pristupa, ovisno o implementaciji. Ova razlika posebno dolazi do izražaja kod slika većih dimenzija, gdje količina podataka i broj potrebnih izračuna eksponencijalno raste.

Unatoč velikim razlikama u brzini, kvaliteta rezultata između CPU i GPU verzije bila je vizualno jednaka, što potvrđuje ispravnost implementacije i pokazuje da optimizacija performansi ne mora značiti kompromis u kvaliteti. Korištenjem CUDA tehnologije moguće je obraditi milijune piksela gotovo u stvarnom vremenu, što je izuzetno korisno u područjima kao što su video analitika, autonomna vožnja, medicinska dijagnostika i sustavi nadzora.

Ovaj projekt može se dodatno unaprijediti kroz:

- Implementaciju dodatnih algoritama za detekciju rubova (npr. Canny, Laplace)
- Optimizaciju pristupa memoriji na GPU-u
- Korištenje dijeljene memorije radi poboljšanja performansi
- Paralelnu obradu više koraka u jednom GPU kernelu

Zaključno, rad pokazuje kako pravilno iskorištavanje paralelizma koje pruža GPU i CUDA može dovesti do višestrukog povećanja brzine obrade slike, zadržavajući pritom točnost i pouzdanost rezultata.

LITERATURA

[1] Mišadin, D.; Akrapović, I.; Krstanović, J.: „CUDA Sobel filter“, 2025., https://github.com/dmisadin/cuda-sobel-filter

[2] Francisco, R.: „Moraine Lake at 6AM“, s Interneta, zadnji put pristupljeno: 11.6.2025., https://unsplash.com/photos/landscape-photography-of-snowy-mountains-gdQnsMbhkUs