Izvještaj o performansama algoritma za diskretnu konvoluciju

1. Opis problema

Cilj ovog projekta je realizacija paralelizabilnog algoritma za obradu slike koristeći operaciju diskretne konvolucije. Algoritam treba da bude fleksibilan, omogućava korisnicima da specifikuju konvolucioni kernel pri pokretanju. Program takođe prihvata putanju do ulaznog kao i izlaznog fajla. Pored toga potrebno je analizirati i graficki predstaviti performanse algoritma korištenjem različitih nivoa kompajlerskih optimizacija i paralelizacije na višejezgrenom procesoru sa OpenMP bibliotekom. Takođe dodata je i GPGPU varijanta algoritma tako da se postigne krace vrijeme izvrsavanja.

2. Opis osnovnog algoritma

Osnovni algoritam za diskretnu konvoluciju primjenjuje dati konvolucioni kernel na svaku tačku slike. Algoritam prolazi kroz svaku tačku slike i računa novu vrijednost piksela koristeći vrijednost susjednih piksela i odgovarajućih vrijednosti iz konvolucionog krenela. Algoritam je implementiran u C++.

3. Opis optimizovanih varijanti algoritma

Optimizovane varijante algoritma koriste različite nivoe kompajlerskih optimizacija (O0, O1, O2, O3) koje omogućava GCC kompajler. Ove optimizacije uključuju:

- O0: Bez optimizacija, koristi se za debagovanje.
- O1: Osnovne optimizacije koje poboljšavaju performanse bez prevelikih promena u kodu,
- O2: Naprednije optimizacije koje uključuju uklanjanje mrtvog koda, unrolling petlji, itd.
- O3: Najnaprenije optimizacije koje uključuju sve iz O2 plus dodatne optimizacije za maksimalne performanse.
- Takođe postoji i varijanta algoritma koja koristi CUDA Toolkit koji se koristi za dobijanje bolji performasi , distribuiranjem izračunavanja na više multi-GPU konfiguracija.

4. Opis Načina Mjerenja i Hardvera

Mjerenja su izvršena na sledeći način:

- Veličine ulaza: Koristili smo slike različitih veličina ($N \in \{10^3, 10^4, 10^5, 10^6, 10^7\}$.)
- Zagrijavanje : Prije svakog mjerenja, algoritam je izvršen više puta bez mjerenja kako bi se izbjegle fluktuacije u rezultatima.
- Različiti slučajevi: Mjerenja su izvršena na različitim ulaznim podatcima kako BMP slikama tako i parametrima konvolucijskog kernela.

- Višestruko mjerenje: Za svaki ulaz, mjerenje je izvršeno 10 puta, a dokumentovana je prosječna vrijednost i varijansa.

5. Hardver

Hardver korišten za mjerenja:

- Procesor: Ryezen 7 7435HS sa 8 jezgara
- Memorija: 16GB RAM
- Operativni sistem: Windows 11 23H2 sa Linux distribucijom
- Linux distribucija: Ubuntu 24.04 LTS
- Grafička kartica: Nvidia RTX 4060 8GB

6. Grafički predstavljeni rezultati mjerenja

Za pokretanje svih mjerenja u okviru izvještaja koristi se skipta **full.sh** koja pokrece 3 mjerenja uz pomoćnu skriptu **script.sh** koja se koristi za pokretanje sve 3 vrste mjerenja pojedinačno (1 - Sharpen, 2 - Edge Detection, 3 - Slučajne vrijednosti). Kao argument komandne linije skript full.sh treba *prosljediti naziv ulaznog BMP fajla*. Skripta script.sh se moze koristiti nezavisno od full.sh, ako postoje ulazni fajlovi koji su generisani u odgovarajućem formatu, a u komandnoj liniji je potrebno specifikovati konvolucijski kernel. Izvršavanjem skripte čuvaju se rezultati koje proizvodi skripta script.sh za svako mjerenje i na kraj naziva je dodan redni broj mjerenja. U nastaku je opisan rad skripte script.sh.

Rezultati mjerenja su predstavljeni u grafiku performase_comparison.png koji prikazuje performanse algoritma pre i posle optimizacije i paralelizacije i GPGPU varijantu algoritma. Takođe generisana je slika performance_comparison_with_variance koja ima dodatak prikaza varijanse u svakoj tacki. Generisan je i fajl all_results.txt koji sadrzi tekstulano sve rezultate svih pokrenutnih mjerenja. Takođe izlazi iz svih varijanti algoritma se čuvaju u folderima output, te gp_output za GPGPU varijantu algoritma. U svakoj iteraciji mjer Za pokretanje i prikaz rezultata mjerenja potrebno je pokrenuti bash skriptu **script.sh** koja će generisati prethodno opisane rezultate mjerenja. Takođe skripta će pokrenuti program za validaciju koji će ispisati na konzoli odgovarajuću poruku ako su fajlovi validni a ako algoritam nije proizveo validan rezultat prikazaće se na konozli o kom fajlu se radi tj. o kojoj kombinaciji ulaznih parametara se radi.

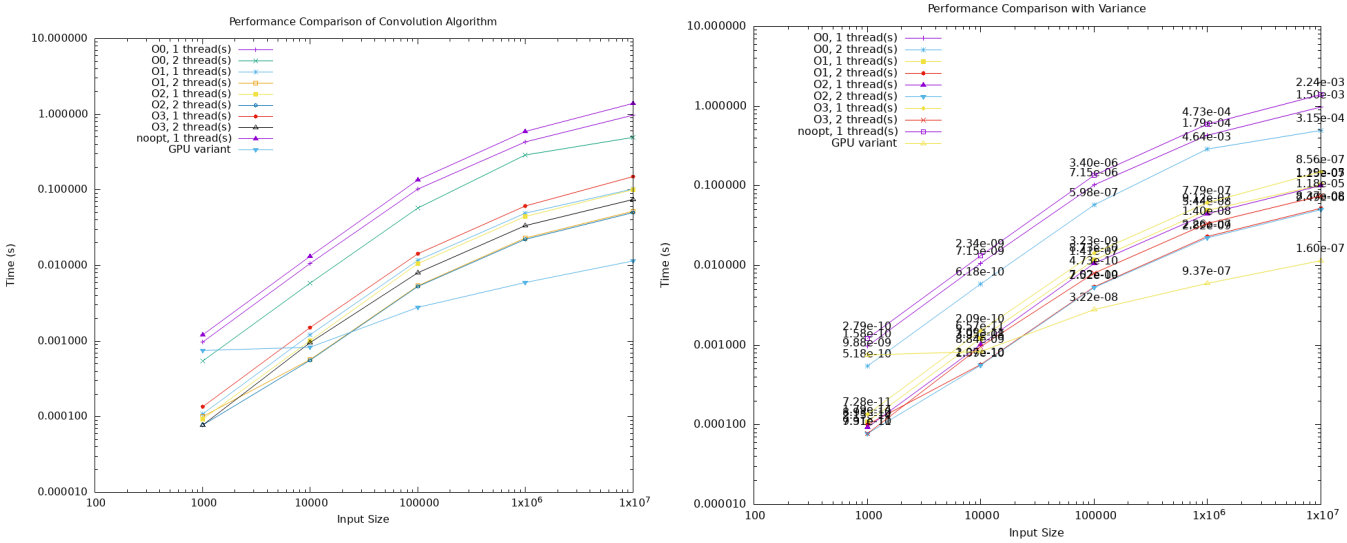
Ukoliko je skripta full.sh uspjesno izvršena za sve varijante mjerenja onda bi u trenutnom direktorijumu trebali imati:

- 3 grafika (**performance_comparison_i**, i - broj mjerenja),
- 3 grafika (**performance_comparison_with_variance_i**, i - broj mjerenja),
- 3 tekstualna fajla (**all_results_i**.txt, i - broj mjerenja) .

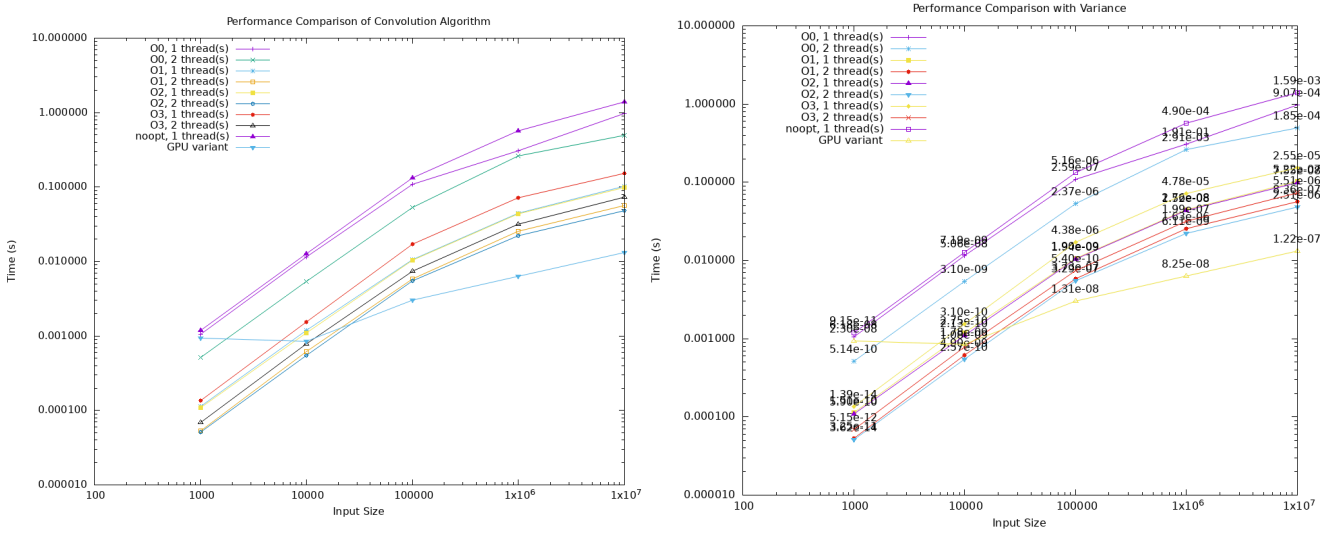
Informacije o **validnosti** rezultata svih mjerenja su ispisana na konzoli.

7. Zaključak

Na slici ispod su prikazani rezultati pokretanja algoritma sa različitim ulazima i vrstama optimizacije kao i brojem jezgara. Za ovaj rezultat korišten je konvolucioni kernel koji proizvodi efekat Sharpen, odnosno izoštravanje slike.



Nakon toga algoritam je pokrenut sa istim ulaznim parametrima, ali je ovaj put promjenjen konvolucioni kernel kako bi otkrili da li doći do promjene performansi. U ovom slučaju kernel koji je korišten da je efekat Edge Detection odnosno detekciju krajeva. Rezultati su prikazani na slici ispod.



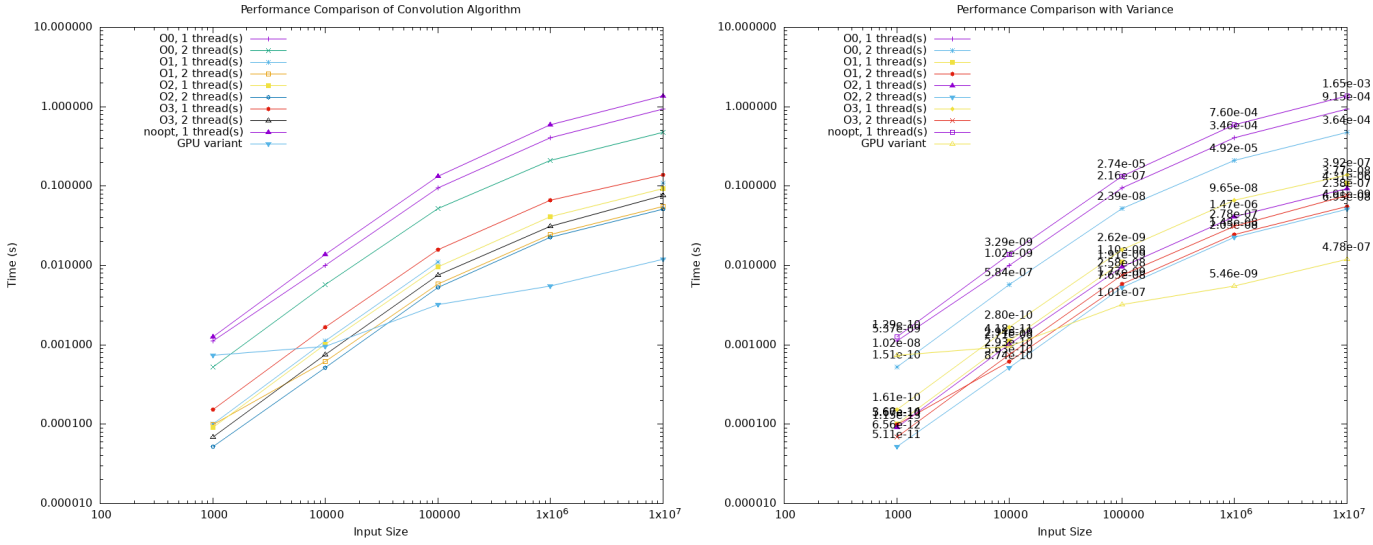
Rezultati nam pokazuju da nije došlo do značajnih promjena u performansama. Nizi nivoi oprimizacije su i dalje imali vrlo slične rezultate kao u prethodnom mjerenju. Jedina

razlika je što su u ovom slučaju performanse algoritma sa O1, 2 jezgra, krenule da opadaju sa povećanjem veličine ulaza. Ali je vazno napomenuti da je razlika i dalje mala.

Uopsteno vidimo da se najveći skok u performansama ostvaruje pri pokretanju sa 2 jezgra i ovo vazi za svaki nivo optimizacije. Postoje i slučajevi u kojima je nizi nivo optimizacije sa 2 jezgra imao bolje performanse nego slučaj sa optimizacijom O2 i 2 jezgra imao znatno bolje performanse nego slučaj sa optimizacijom O3 i 1 jezgrom. Najbolje performanse su postignute sa O2 optimizacijom i paralelizacijom na 2 jezgra za oba kernela. Vazno je napomenuti da je O3 nivo optimizacije imo losije performanse od O2 pa čak i O1 nivoa optimizacije za oba kernela kada se algoritam pokretao za 2 jezgra, dok je imao bolje performanse kada se algoritam pokretao za 1 jezgro.

Pogledajmo i varijanse koje su označene na drugom grafu. Sa slike vidimo da su varijanse jako male, pogotovo za manje ulaze. Sa slike takođe vidimo da je GPGPU varijanta algoritma ostvarila bolje performanse od bilo koje druge varijante algoritma ali za dovoljno velike veličine ulaza. Za ulaz veličine reda 1000 performanse su gotovo iste kao u baznom slučaju, a ulaz veličine reda 10000 performanse su gotovo identične varijanti sa O3 optimizacijom i 2 thread-a. Dok je za sve veće ulaze GPGPU varijnata algoritma ima ubjedljivo najbolje performanse i na nju najmanje utiče povećanje veličine na ulazu. Vazno je napomenuti da se za sve velicine ulaza dobijaju odgovarajuci rezultati, i da se u svakom pokretanju algoritma dobija ispravan rezultat što nam se ispisuje u konzoli.

Pokrenut je algoritam sa slučajnim parametrima na ulazu i rezultati su bili gotovo identični i nemaju uticaj na rad algoritma. Prikaz tih rezultata je naveden na slici ispod:



Optimizacija i paraleilizacija su omogućile ubrzanje algoritma, smanjujući vrijeme izvršavanja za velike veličine ulaza. Sve varijante algoritma proizvodile su tačne rezultate, potvrđujući ispravnost implementacije.