Arhitektura računara

Daniel Majstorović 1176/21

**Konvolucija slike**

Konvolucija slike je matematička operacija koja se koristi u obradi slika kako bi se izvršila transformacija slike pomoću filtera, poznatog kao kernel. Ova operacija podrazumeva prelaženje kernela preko slike, gde se elementi kernela množe sa odgovarajućim pikselima slike, a zatim se ti proizvodi sabiraju kako bi se dobio novi piksel u izlaznoj slici. Konvolucija omogućava izdvajanje određenih karakteristika iz slike, poput detekcije ivica, zamućivanja, izoštravanja ili uklanjanja šuma. Ova tehnika je široko primenjena u različitim oblastima obrade slika, kao što su prepoznavanje oblika i unapređenje vizuelnog kvaliteta.

**Osnovni algoritam za konvoluciju slike implementiran u funkciji performConvolution radi na sledeći način:**

1. **Priprema dimenzija kernela**: Prvi korak je izračunavanje polovine dimenzija kernela. Ove vrednosti su neophodne za pravilno pozicioniranje kernela prilikom prolaska kroz sliku.
2. **Konverzija ulazne slike**: Ulazna slika inputImage se konvertuje u tip CV\_64FC3, što omogućava rad sa slikom sa trokanalnim pikselima (RGB) sa 64-bitnom preciznošću za svaku komponentu boje. Ova konverzija omogućava preciznije računanje tokom operacije konvolucije.
3. **Proširenje ulazne slike**: Kreira se proširena verzija slike, nazvana expandedImage, koja ima dodatne margine u odnosu na originalnu sliku. Ove margine su potrebne da bi se omogućilo centriranje kernela na pikselima koji se nalaze uz ivice slike. Pikseli margina su inicijalizovani na vrednost nula.
4. **Popunjavanje proširene slike**: U sledećem koraku, originalna slika se kopira u centar proširene slike, dok margine ostaju na nultim vrednostima. Na ovaj način se omogućava da se kernel može pomerati po čitavoj slici, uključujući i ivice.
5. **Kreiranje rezultujuće slike**: Definiše se rezultujuća slika resultImage sa istim dimenzijama kao ulazna slika. Ova slika se inicijalizuje sa pikselima postavljenim na nultu vrednost.
6. **Primena kernela na svaki piksel**: Algoritam zatim prolazi kroz proširenu sliku, pomerajući centar kernela preko svakog piksela originalnog sadržaja slike. Za svaki piksel u rezultujućoj slici, računa se nova vrednost tako što se vrednosti piksela u prozoru veličine kernela pomnože sa odgovarajućim elementima kernela, a zatim se ti proizvodi sabiraju. To se radi za svaku boju (R, G, B) posebno.
7. **Ažuriranje rezultujuće slike**: Vrednosti crvene, zelene i plave boje (R, G, B) koje su dobijene primenom kernela po formuli, smeštaju se u odgovarajući piksel u resultImage.
8. **Konverzija tipa rezultujuće slike**: Kada je proces konvolucije završen, rezultujuća slika se konvertuje nazad u tip CV\_8UC3, što predstavlja sliku sa 8-bitnim vrednostima za svaku od tri komponente boje.
9. **Vraćanje rezultata**: Na kraju, funkcija vraća rezultujuću sliku nakon što je konvolucija primenjena na čitavoj slici.

**Optimizovane varijante algoritma**

Omogućavaju značajno ubrzanje izvršavanja osnovnog algoritma konvolucije slike, a time i postizanje boljih performansi. Optimizacija je postignuta korišćenjem dve ključne tehnike: paralelizacija putem OpenMP-a i SIMD programiranje korišćenjem intrinzičnih funkcija za rad sa AVX registrima.

**Paralelizacija pomoću OpenMP-a**

Paralelizacija je omogućena kroz #pragma omp direktive u kodu, koristeći postojeću podršku za OpenMP. Cilj ove optimizacije je da se osnovni algoritam prilagodi tako da koristi više jezgara procesora, čime se postiže brže izvršavanje. Direktive koje su korišćene uključuju:

* #pragma omp parallel for schedule(static, 2): Ova direktiva omogućava statičko raspoređivanje niti tako da svaka nit obrađuje dva uzastopna reda slike. Time se obezbeđuje ravnomerno raspoređivanje opterećenja među nitima i optimizuje se efikasnost izvršavanja.
* #pragma omp parallel for reduction(+:R, G, B): Ova direktiva koristi reduction klauzulu za sabiranje delimičnih rezultata komponenti R, G i B tokom izračunavanja novih vrednosti piksela unutar petlji. Paralelizovanjem unutrašnje petlje, više niti istovremeno računa vrednosti piksela, a reduction klauzula osigurava ispravno sabiranje rezultata bez trke za podacima. Na taj način se paralelizuje deo algoritma koji je računski najintenzivniji.

**SIMD optimizacija korišćenjem intrinzičnih funkcija**

Pored paralelizacije, postignuto je dodatno ubrzanje korišćenjem SIMD (Single Instruction, Multiple Data) instrukcija kroz AVX registre i intrinzične funkcije. SIMD omogućava istovremenu obradu više podataka u jednom ciklusu instrukcija, čime se smanjuje vreme potrebno za primenu konvolucije na svaki piksel.

Kod koristi AVX intrinzične funkcije na sledeći način:

* **Inicijalizacija AVX registara**: Registar \_\_m256d se koristi za smeštanje više vrednosti R, G, B komponenata piksela, dok se kernel vrednosti dupliraju u AVX registre radi simultanog množenja.
* **Množenje i sabiranje vrednosti**: Za svaki piksel u slici, vrednosti RGB komponenti piksela i odgovarajuće vrednosti kernela učitavaju se u AVX registre koristeći funkcije poput \_mm256\_set\_pd i \_mm256\_set1\_pd. Zatim se ove vrednosti međusobno množe funkcijom \_mm256\_mul\_pd, a rezultati se sabiraju pomoću \_mm256\_add\_pd. Ovaj proces omogućava paralelno množenje i sabiranje više komponenti u jednom ciklusu.
* **Ekstrakcija rezultata**: Nakon završetka izračunavanja, rezultati se izvlače iz AVX registara i vraćaju u regularne promenljive kako bi se sačuvali u rezultujuću sliku.

**Kombinovana SIMD i paralelizacija**

* Najnaprednija verzija algoritma kombinuje SIMD optimizaciju sa OpenMP paralelizacijom, omogućavajući obradu piksela u AVX registrima uz paralelnu obradu različitih delova slike na više jezgra procesora. Ova verzija koristi prednosti i paralelizacije spoljašnjih petlji i SIMD operacija za množenje i sabiranje unutar petlji. Kombinacija ovih tehnika predstavlja najbržu varijantu algoritma, značajno povećavajući performanse u poređenju sa osnovnom implementacijom.

**Specifikacija hardvera**

|  |  |
| --- | --- |
| Processor | Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 1498 Mhz, 4 Core(s), 8 Logical Processor(s) |
| Installed Physical Memory (RAM) | 16.0 GB |
| OS Name | Microsoft Windows 11 Pro |

**Način mjerenja**

Način mjerenja performansi algoritma zasnovan je na tačnom vremenskom praćenju, koristeći funkciju omp\_get\_wtime() iz biblioteke <omp.h>, koja mjeri proteklo stvarno vrijeme izvršavanja programa. Testiranje performansi obuhvata proces zagrijavanja algoritma, nakon čega slijedi višestruko izvršavanje i mjerenje vremena.  
Algoritam je najprije "zagrijan" tri puta kako bi se eliminisao uticaj faktora kao što su inicijalna učitavanja podataka i optimizacije sistema. Nakon zagrijavanja, algoritam se izvršava tri puta, pri čemu se mjeri vrijeme za svaku iteraciju.  
Na osnovu dobijenih rezultata iz ovih mjerenja, izračunava se srednje vrijeme izvršavanja i varijansa kako bi se dobila statistička tačnost i stabilnost algoritma. Srednje vrijeme daje prosjek trajanja algoritma, dok varijansa pokazuje odstupanje između pojedinačnih mjerenja, što pomaže u procjeni konzistentnosti performansi.

**Rezultati:**

Sekvencijalno/Paralelno – Optimizacija

Sekvencijalno – Optimizacija odnosi se na implementaciju konvolucije koja se izvršava sekvencijalno, ali koristi određenu optimizaciju.   
Paralelno – Optimizacija podrazumijeva implementaciju konvolucije koja koristi paralelno procesiranje zajedno sa nekom od optimizacija.   
Ako optimizacija nije posebno navedena, riječ je o osnovnoj implementaciji konvolucije bez ikakvih dodatnih optimizacija.

**Srednje vrijednosti vremena izvršavanja svih verzija algoritma:**



Grafički prikaz tabelarnih vrijednosti:

**Varijanse vremena izvršavanja svih verzija algoritma:**



Grafički prikaz tabelarnih vrijednosti:

**Zaključak:**

Rezultati mjerenja pokazuju da optimizacije poput O1, O2 i OX imaju pozitivan uticaj na smanjenje vremena izvršavanja kod sekvencijalnih implementacija konvolucije. Ove kompajlerske optimizacije omogućavaju bolje iskorišćavanje resursa procesora i efikasnije izvršavanje instrukcija, što rezultira kraćim vremenima obrade. Međutim, zanimljivo je da kod paralelnih implementacija, ove iste optimizacije dovode do povećanja vremena izvršavanja, što ukazuje na to da njihova primjena u paralelnom okruženju nije uvek optimalna i može dovesti do komplikacija u raspoređivanju zadataka i sinhronizaciji.

S druge strane, ručna optimizacija korištenjem SIMD instrukcija pokazala se kao najefikasniji pristup. Bez obzira da li je riječ o sekvencijalnoj ili paralelnoj implementaciji, ovaj pristup dosledno daje najbolje rezultate. Sekvencijalna implementacija sa ručnom optimizacijom ostvaruje značajno ubrzanje u poređenju sa standardnim sekvencijalnim metodama, dok paralelna implementacija dodatno maksimizira performanse iskorišćavanjem višejezgarnih procesora u kombinaciji sa SIMD instrukcijama.

Zaključno, ručna optimizacija predstavlja najefikasniji pristup za obradu konvolucije slike, omogućavajući maksimalno ubrzanje u svim scenarijima, kako sekvencijalnim, tako i paralelnim.