SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Izvedba algoritma za kompresiju medicinskih slikovnih podataka korištenjem grafičkog procesora s ciljem povećanja propusnosti i smanjenja utrošene energije

Marko Đurasević, Dino Šantl

Zagreb, lipanj 2013.

Ovaj rad je izrađen je na Zavodu za automatiku i računalno inženjerstvo Fakulteta elektrotehnike i računarstva pod vodstvom doc. dr. sc. Josipa Knezovića i predan je na natječaj za dodjelu Rektorove nagrade za najbolje studentske radove u akademskoj godini 2012./2013.

POPIS TABLICA

POPIS SLIKA

SADRŽAJ

1. Uvod

Današnji informacijski sustavi koji se koriste u svim granama ljudskih djelatnosti pa tako i u medicini odlikuju se iznimnom složenošću. Potrebe za računalnom moći i pohrani velikih količina podataka svakim je danom sve veća. Fizičke granice procesora polako se dosežu. Zbog toga sve je više višeprocesorskih sustava u upotrebi. Za potpuno iskorištavanje računalne moći potrebni su dobro izvedeni paralelni algoritmi. Osim računalne moći jedna od popularnijih disciplina u računarstvu danas je upravljanje velikim količinama podataka. Umjesto spremanja podataka u sirovom formatu, koriste se razne vrste kompresija kako bi se što je moguće racionalnije iskoristio prostor za pohranu podataka. Usporedno s porastom računskih mogućnosti i zahtjeva današnjih sustava, postoji i sve veći problem utrošene energije koja je potrebna za njihov rad. Cilj je uštedjeti što više energije [1]. Upravljanje energijom postaje sve važnije pitanje za informatičke sustave.

Bolnički informacijski sustavi su tipičan primjer složenog sustava. Jedna od glavnih karakteristika tog sustava je spremanje velikih količina podataka, najčešće vizualnih. Za osiguravanje kvalitetnog rada osoblja (korisnika sustava) potrebno je smanjiti nepotrebno vrijeme čekanja dohvata slika s udaljenih računala pomoću mreže. Jedan aspekt rješavanja navedenih problema je primjena kompresije podataka. Kompresijom se postiže ušteda prostora za pohranu podataka i veća propusnost podataka između računala. Važno svojstvo koje takav algoritam kompresije mora imati je kratko vrijeme izvođenja. Pojedine dijagnostičke metode generiraju veoma velike količine podataka. Tako primjerice studije magnetske rezonancije generiraju i do 4GB slikovnih podataka po pacijentu. Pretpostavimo da obavljanje jednog pregleda traje 30 minuta i da se uređaj koristi 8 sati dnevno (postoji tendencija da se oni koriste i više od tog vremena) dobivamo da se dnevno generira 64 GB slikovnih podataka dnevno, odnosno skoro 2 TB mjesečno.

U ovom radu predlaže se izvedba metode za kompresiju slika bez gubitaka *CBPC* (engl. *Contextual and Blending Predictive Coder*) predloženog u [2, 3, 4] za bolničke sustave. Radi se o metodi koja se zasniva na računski složenom predviđanju slikovnih

elemenata na kauzalnom kontekstu susjednih elemenata korištenjem metode "miješanja" statičkih funkcija predviđanja. Postupak "miješanja" sastoji se u iscrpnom pretraživanju lokalnog područja kako bi se našli "slični" slikovni elementi nad kojima se vrši adaptacija statičkog skupa funkcija predviđanja [3]. Ovaj dio metode odlikuje se velikom računskom složenošću, ali isto tako ima veliku količinu paralelizma među podatcima (slikovnim elementima). Zahtjevi koji su postavljeni pred izvedbu su što bolje vrijeme izvođenja, što veća kompresija podataka, što je moguće veća ušteda energije. Cilj rada je pokazati da izvedba zadovoljava postavljene zahtjeve.

Kao rješenje problema predložena je izvedba računski zahtjevnog koraka predviđanja korištenjem grafičkog procesora kao ubrzivača. Koncept korištenja grafičkih procesora kao ubrzivača u aplikacijama opće namjene je relativno novi pristup koji se pojavio razvojem računskih mogućnosti grafičkih procesora [5, 6, 7, 8]. U predloženoj izvedbi odabran je *CUDA* programski model i okruženje. *CUDA* je paralelna računalna platforma i programski model koji povećava performanse korištenjem računalne snage grafičkih procesora za opću namjenu [7]. Razlog izvedaba za grafičke procesore je u tome što sve više računalnih sustava koristi grafičke procesore kao osnovnu komponentu. Grafički procesori omogućuju podatkovni paralelizam pa je očekivano da se postigne što je moguće manje vrijeme izvođenja. Iako grafičke kartice troše više električne energije, ukoliko se postigne dovoljno veliko ubrzanje može se smanjiti i sama potrošnja električne energije u usporedbi s običnim procesorima. Nadolazeći grafički procesori podržavaju uz izvođenje grafički-specifičnih operacija što im je primarna uloga podržavaju i mogućnosti izvođenja aplikacija opće namjene korištenjem programskih modela kao što su *CUDA* i *OpenCL*. Tako je moguće obaviti kompresiju slikovnih dijagnostičkih podataka na računalu koje se nalazi pri samom dijagnostičkom uređaju te tako komprimirane podatke spremiti u centralni repozitorij podataka. Na taj način postiže se ušteda u količini prometa preko mrežne infrastrukture bolničkog informacijskog sustava te se smanjuju zahtjevi za propusnost. Ove prednosti naročito dolaze do izražaja u telemedicinskim primjenama gdje postoji potreba prijenosa ne samo statičkih slikovnih podataka nego i video, trodimenzionalnih i višedimenzionalnih podataka [9, 10].

2. Motivacija

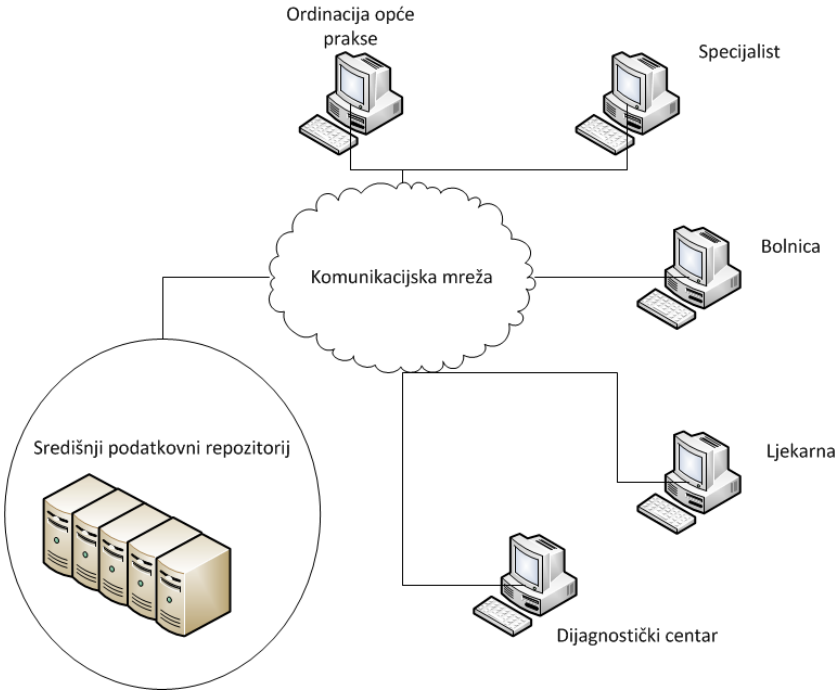
2.1. Bolnički informacijski sustavi

U današnje vrijeme sve se više ulaže u razvoj modernih informacijskih sustava koji se koriste u različitim sferama ljudske djelatnosti. Među njima se posebno ističu bolnički informacijski sustavi kojima se nastoji ostvariti jedinstven registar medicinskih podataka o pacijentima, brza i učinkovita razmjena medicinski bitnih podataka o pacijentima te ostale operacije koje imaju za cilj podizanje kvalitete cjelokupnog zdravstvenog sustava. Time se nastoji ostvariti da su svi podaci o pacijentima pohranjeni na jednom mjestu te su preko informacijskog sustava dostupni liječnicima na zahtjev. Podatci pacijenta više ne bi bili spremljeni u papirnatom obliku, već u obliku e-kartona, gdje bi bili pohranjeni svi podaci relevantni za pacijenta (primjerice povijest bolesti, recepti i slično). Osim toga, ovakav informacijski sustav treba omogućiti pohranu i brzu razmjenu dijagnostičkih podataka o pacijentima kako bi se rad cjelokupnog zdravstvenog sustava optimirao te se izbjegle nepotrebna ponavljanja dijagnostičkih postupaka koji su često iznimno skupi.

Konceptualni prikaz bolničkog informacijskog sustava prikazan je na slici 2.1. Središnji zdravstveni sustav predstavlja lokaciju na koju se spremaju svi podaci o pacijentima u obliku e-kartona, ali i njihovi radiološki nalazi. Svatko tko je unutar bolničkog sustava autoriziran za pristup podacima, može njima pristupiti putem medicinskog informacijskog sustava. Na taj način svi potrebni podaci o pacijentu su odmah dostupni liječnicima specijalistima. Na taj način provodi se centralizacija svih relevantnih medicinskih podataka o pacijentu na jedno jedinstveno mjesto.

Kako su medicinski slikovni podaci često velikih veličina od nekoliko stotina *MB*-a pa čak i nekoliko *GB*, treba se te podatke moći pohraniti, ali ih i razmjenjivati na što efikasniji način. U tu svrhu razvijen je *PACS* (engl. *Picture Archiving and Communication System*). *PACS* se bazira na medicinskim slikovnim podatcima, njihovom pohranjivanju, komunikacijskim tehnologijama, prikazu medicinskih slika i kliničkom radnom slijedu [11]. On omogućava efikasno arhiviranje medicinskih slikovnih poda-

taka, kao i njihov efikasan digitalni prijenos. Formati zapisivanja slika kao i njihovog prijenosa putem *PACS*-a se temelje na standardima kao što su *DICOM*[12] (engl. *Digital Imaging and Communications in Medicine*) i *NIfTI* [13] (engl. *Neuroimaging Informatics Technology Initiative*). *DICOM* predstavlja standard kojim se propisuju načini pohranjivanja, prenošenja i rukovanja medicinskim slikovnim podacima. U njemu su propisani korišteni podatkovni formati, kao i komunikacijski protokoli.



Slika 2.1: Primjer bolničkog informacijskog sustava

U Republici Hrvatskoj se također želi postići visoki stupanj informatizacije zdravstva. U tu svrhu koristi se Centralni zdravstveni informacijski sustav Hrvatske (CEZIH). CEZIH-om se nastoje povezati sve liječničke ordinacije, osiguravatelje te razne medicinske ustanove. Veliki iskorak prema daljnjoj informatizaciji je učinjen 2011. godine kada je uveden e-recept. Broj izdanih e-recepata godišnje iznosi otprilike 50 milijuna [14]. Iako je već veliki broj ordinacija i ljekarni povezan sa CEZIH-om, trenutačno još ne postoji međusobna informatička povezanost između bolnica kao ni jedinstven skup podataka koji se prate i šalju. Integracija bolničkih sustava i centralnog informatičkog sustava je tek u začetku i treba se vidjeti kada će ona biti ostvarena [14].

Nacionalna strategija razvoja zdravstva u Hrvatskoj stavlja također fokus i na razvoj telemedicine (engl. *telemedicine*) [14]. Jedan od elemenata telemedicine je i teleradiologija (engl. *teleradiology*), odnosno pohrana i razmjena različitih radioloških slika pacijenata poput *CT*-a i magnetske rezonancije [15]. Na taj način radiolozi bi mogli pružati svoje usluge pacijentima bez potrebe da budu fizički u njihovoj blizini.

Osim toga, teleradiologija bi omogućila i konzultaciju međusobno udaljenih liječnika vezanu uz interpretaciju raznih radioloških slika pacijenata [16].

2.2. Zeleno računarstvo

"Zeleno računarstvo" (engl. *Green Computing*) je pojam s kojim se danas sve više i više susrećemo. Pod tim pojmom podrazumijeva se istraživanje razvoja, korištenja računala, kao i zbrinjavanja računalnog otpada s minimalnim ili nikakvim utjecajem na okoliš [17]. "Zeleno računarstvo" je danas sve važniji koncept iz razloga što se proizvodi sve više računalnih komponenti i tako nastaje sve više računalnog otpada, kojeg je potrebno zbrinuti na prikladan način.

No "zeleno računarstvo" se ne odnosi samo da zbrinjavanje računalnog otpada. Sve se više koriste veliki računalni informacijski sustavi koji naravno troše veliku količinu električne energije [18, 19]. Upravo je jedan od ciljeva "zelenog računarstva" da se potrošnja takvih računalnih sustava što više minimizira. Naravno da smanjenje potrošnje ide i na ruku korisnicima takvih velikih informacijskih sustava, jer time smanjuju svoje troškove. Upravo iz navedenih razloga prilikom izgradnje takvih sustava, potrebno je voditi brigu o tome da se oni izgrade tako da se potrošnja što više smanji.

Na uštedu električne energije se također može utjecati i načinom na koji su aplikacije izvedene. Naime, tu postoji mogućnost ubrzanja različitih i veoma računski zahtjevnih algoritama, čime se može smanjiti ukupno vrijeme izvođenja, a time i potrošnja električne energije. Također se sve više koristi i mogućnost paralelizacije programa, tako da se njihovi pojedini dijelovi izvode istodobno, kako bi se smanjilo ukupno trajanje izvođenja, što naravno opet može utjecati na potrošnju električne energije. Grafički procesori kao specijalizirane jedinice odlikuju se visokom energetskom učinkovitošću za izvođenje operacija nad velikom količinom podataka korištenjem paralelizma među podatcima. Na taj način moguće je i opće aplikacije koje obrađuju velike količine podataka ubrzati ali i optimirati njihovu potrošnju korištenjem grafičkih procesora kao ubrzivača u programskim modelima kao što su *CUDA* i *OpenCL* [20]. Upravo je to i jedan od ciljeva ovog rada. Nastoji se iskoristiti mogućnost paralelizacije računski zahtjevnog algoritma kompresije slike na grafičkim procesorima kako bi se osiguralo što kraće vrijeme izvođenje istog i tako pokušala smanjiti ukupna potrošnja električne energije. U šestom poglavlju bit će prikazani podaci o količini energije koju je moguće uštedjeti korištenjem paralelizirane inačice algoritma nasuprot serijske inačice.

2.3. Ubrzavanje algoritama grafičkim procesorima

Obični računalni procesori se brzinom radnog takta sve više približavaju svojim fizičkim ograničenjima. Zbog tog razloga, ali i zbog činjenice da su današnji procesori najčešće višejezgreni, sve se više pažnje pridaje paralelizaciji algoritama, radi postizanja ubrzanja njihovog izvršavanja.

U posljednjih nekoliko godina sve se više razvija nova metoda paralelizacija algoritama, a to je izvođenje algoritama putem grafičkih kartica. Grafičke kartice su danas prisutne u gotovo svakom računalu i njihove performanse svakodnevno rastu, dok im se cijena polako smanjuje. Grafičke kartice su prije svega veoma zanimljive jer sadrže veliki broj procesorskih jezgri. Iako su te jezgre po svom radnom taktu dosta sporije od običnih procesora, njihova velika količina nam potencijalno omogućuje obradu velikih količina nezavisnih podataka paralelno, odnosno istovremeno. Zbog toga se veliki napori ulažu u izgradnju i poboljšanje tehnologija koje će omogućiti izvođenje općenitih algoritama na grafičkim karticama (engl. *GPGPU - General-Purpose Computing on Graphics Processing Units*). Trenutno dvije najpopularnije tehnologije koje se koriste u tu svrhu su *OpenCL* [8] i *CUDA* [7]. Osim ovih dviju navedenih tehnologija, Microsoft je nedavno izdao i svoju tehnologiju za (engl. *GPGPU*) pod nazivom *C++ AMP* [21], koja se nije još uspjela afirmirati, ali dokazuje kako je ovo područje još u razvoju i kako će se s vremenom pojavljivati sve nove i sve bolje tehnologije.

Bez obzira što se radi o dosta novom području, već su se pojavile brojne izvedbe različitih algoritama izvedenih na grafičkim karticama kako bi se mogle isprobati mogućnosti paralelizacije i ubrzanja algoritama na njima. Tako su se primjerice već implementirali algoritmi za kompresiju i kodiranje u *JPEG 2000* standardu [22] kao i izvedba fraktalne kompresije za medicinske slike [23]. U oba ova primjera pokazano je kako postoji veliki potencijal za primjenu *GPGPU*-a za paralelizaciju i ubrzanje mnogih algoritama. Sve je popularnije primjena ovih tehnologija i za medicinske svrhe. Tako su grafičke kartice iskorištene za vizualizaciju i simulaciju medicinskih ultrazvukova na temelju *CT* slikovnih podataka [24], gdje su njihovim korištenjem postignute bolje performanse naspram klasičnih računalnih procesora. Proučavanje mogućnosti ubrzanja algoritama putem grafičkih procesora je jedna od glavnih motivacija i ovog rada. Kako grafičke kartice pružaju mogućnost obrade relativno velike količine podataka istodobno, razumno je algoritme koji se obavljaju na velikom količinom podataka, pokušati paralelizirati i ubrzati korištenjem grafičkih procesora.

3. Algoritam za kompresiju CBPC

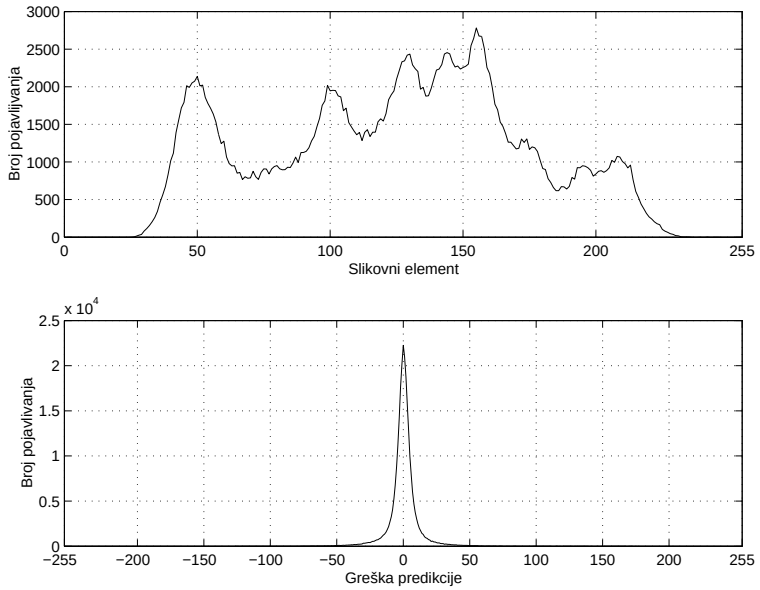
U ovom poglavlju opisan je algoritam za kompresiju slika bez gubitaka predložen u [2, 3]. Dodatno, koristit će se adaptivna metoda predviđanja. Pri kompresiji medicinskih slika dosta često je potrebno osigurati da se iz kodirane slike može dobiti slika koja je istovjetna originalnoj slici dobivenoj na dijagnostičkom uređaju. Zbog toga se u takvim slučajevima koristi kompresija bez gubitaka. Stupanj kompresije ostvariv na ovaj način je znatno manji od stupnja kompresije s gubitcima. Međutim, savršena rekonstrukcija podataka ne dozvoljava korištenje metoda kompresije s gubitcima. U primjenama gdje se ipak žele ostvariti veći stupnjevi kompresije moguće je pojedine dijelove slike komprimirati bez gubitaka (dijagnostički bitne) a pojedine dijelove s gubitcima (dijagnostički nebitni dijelovi kao što su pozadina). Algoritam *CBPC* spada u skupinu algoritama bez gubitaka a postoji mogućnost primjene u gore spomenutom hibridnom sustavu. Posljedice gubitka informacija na medicinskim slikama mogu dovesti do smrti pacijenta, zbog toga kao jedini ispravan i očit izbor je kompresija bez gubitaka. Biti će objašnjen samo postupak kodiranja. Za dekodiranje koristi se isti algoritam samo se koraci obavljaju obrnutim redoslijedom.

3.1. Kompresija slika

Kompresija slika danas zauzima posebno područje u teoriji kompresije. Razlog tome je specifičnost slike kao informacije. Pokazalo se da pristup kodiranju slika metodama koje se koriste za kodiranje teksta (klasični entropijski koderi ili metode rječnika) nije dobar, to je vidljivo na histogramima na slici 3.1. Kada bi se koristila frekvencije pojavljivanja vrijednosti slikovnih elemenata tada bi izračunate vjerojatnosti pojavljivanja vrijednosti slikovnih elemenata bile ujednačene što ne dovodi do dobre kompresije. Cilj je dobiti što manju entropiju (izraz 3.1), a ujednačavanje vjerojatnosti dovodi do maksimalne entropije, što je suprotno željenom.

$$H(X) = \sum_{i=1}^{N} p_i \cdot log(p_i) \tag{3.1}$$

Slikovni dvodimenzionalni podatci odlikuju se velikim stupnjem koreliranosti susjednih slikovnih elemenata. Skup susjednih slikovnih elementa koji su po vrijednosti slični tekućem, nepoznatom elementu, s aspekta entropijskog kodiranja sačinjava njegov kontekst. Ovisnost elementa slike o kontekstu u kojem se nalazi želi se iskoristi za kompresiju. Pri korištenju metoda predviđanja cilj je pronaći model koji će predvidjeti slikovni element na temelju slikovnih elemenata koji se nalaze u njegovoj blizini. Ako je model dobar tada će matematičko očekivanje pogreške biti 0. Uz to, histogram greške predviđanja (slika 3.1) imat će šiljasti oblik što je idealno za kodiranje klasičnim entropijskim koderima.



Slika 3.1: Prikaz histograma prilikom korištenja frekvencije pojavljivanja slikovnih elemenata i prilikom korištenja greške

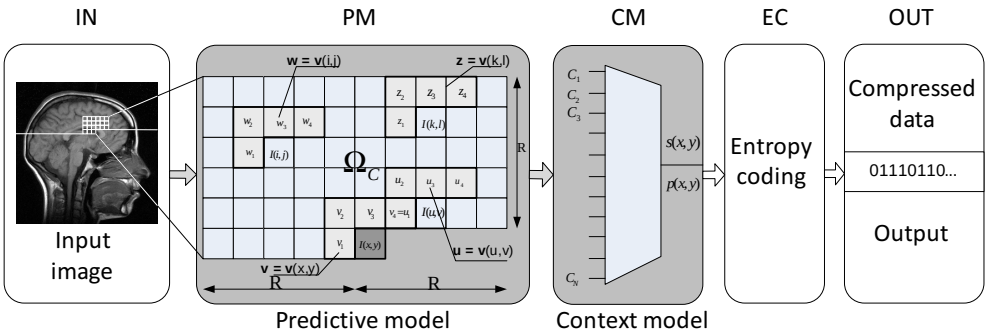
Slika se može predstaviti matricom $I(i, j)$, gdje pojedini element matrice predstavlja vrijednost slikovnog elementa i vrijedi $0 < i \leqslant H, 0 < j \leqslant W$. Gdje su H i W visina odnosno širina slike. Algoritam za kompresiju koristi korak predviđanja slikovnih elemenata s ciljem uklanjanja statističke zalihosti prije samog entropijskog kodiranja. Slikovni elementi obrađuju se pojedinačno redoslijedom prikaza po recima a zatim po stupcima. Za svaki slikovni element izvršava se korak predviđanja koji na izlazu daje grešku predviđanja. U ovom radu korištene su crno-bijele slike (engl. *grayscale*). Na takvim slikama minimalna vrijednost slikovnog elementa je 0, a maksimalna 255. Algoritam se može jednostavno proširiti i na slike u boji s tri komponente pri čemu

se svaka od komponenti obrađuje zasebno. Takva izvedba je ponovno prikladna za korištenje grafičkog procesora.

3.2. Koraci algoritma CBPC

Korišteni algoritam može se opisati (slika 3.4) u nekoliko glavna koraka [4]:

1. **Ulaz** - Učitavanje slikovnih podataka: učitavanje slike ili niza slika iz pojedine studije,
2. **PM - Predikcijsko modeliranje**: predviđanje trenutnog slikovnog elementa i računanje pogreške,
3. **CM - Kontekstualno modeliranje**: klasifikacija pogreške i generiranje različitih histograma za svaki razred pogreške,
4. **EC - Entropijsko kodiranje**: korištenje klasičnog načina kodiranja pogreške predikcije.
5. **Izlaz** - Spremanje komprimiranih podataka.

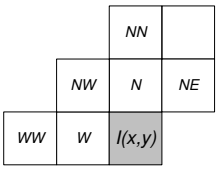


Slika 3.2: Prikaz koraka *CBPCB* algoritma

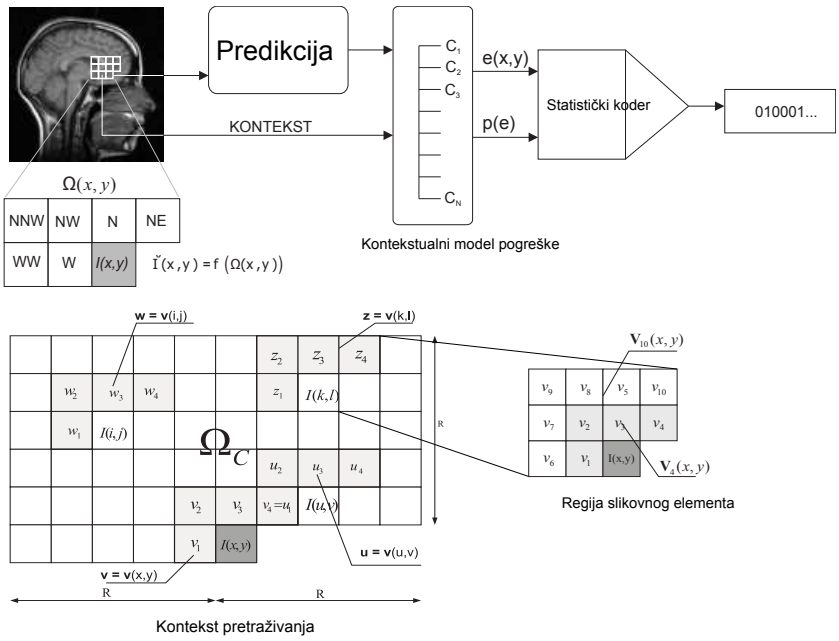
3.2.1. Predikcijsko modeliranje

Predviđanje se temelji na vrijednostima slikovnih elemenata koji se nalaze u njegovom susjedstvu. Nadalje te elemente zvat ćemo regijama ili kontekstom slikovnog elementa. Potrebno je definirati regiju slikovng elementa. Ona može biti odabrana proizvoljno. Algoritam koji se opisuje koristi slijedni unazadni adaptivni model. To znači da se slika obrađuje po redcima. Kad završi obrada jednog retka kreće se na

sljedeći. Ovakav način obrade idealan je za tokovni programski model. Izravna posljedica je ograničenje na definiranje regije slikovnog elementa. Slikovni element u svojoj regiji može imati bilo koje elemente koji su obrađeni prije njega. U konkretnom algoritmu koristi se regija prema slici 3.3, gdje su oznakama *NN, NW, N NE, WW, W* označeni slikovni elementi koji se koriste kao regija za slikovni element na poziciji (x, y).



Slika 3.3: Regija za slikovni element na poziciji (x, y)



Slika 3.4: Blok shema rada algoritma

Predviđanje se općenito računa kao:

$$\hat{I}(x, y) = \sum_{I(i,j) \in \Omega} a_{i,j} \cdot I(i, j) \tag{3.2}$$

Pri tome dobivena greška predviđanja je:

$$e(x, y) = I(x, y) - \hat{I}(x, y) \tag{3.3}$$

U izrazu 3.2 koeficijenti $a_{i,j}$ predstavljaju parametre linearnog modela predviđa-

nja. S Ω označeni su svi slikovni elementi koji se nalaze u regiji. Statistička svojstva slikovnih elemenata bitno se mijenjaju od područja do područja. Ako se uzme predikcijska funkcija koja dobro predviđa u nekoj regiji, ona najčešće ne predviđa dobro na drugim regijama. Zato postoje različite predikcijske funkcije koje su specijalizirane za određene regije. Algoritam pretpostavlja da u različitim regijama na slici dominiraju različita svojstva ali pritom ne isključuje druga svojstva koja mogu biti prisutna. Zbog toga se posvećuje velika pažnja adaptaciji predikcijske funkcije kako bi se za različite regije mogle odabrati najbolje predikcijske funkcije, za razliku od nekih popularnih metoda kompresije slika bez gubitaka (*CALIC, JPEG-LS* i sl.) gdje se koriste statičke predikcijske funkcije [25, 26, 27].

Kvaliteta predviđanja ovisi o odabiru parametara $a_{i,j}$. Da bi predviđanje bilo uspješno za određene regije na slici postavljaju se sljedeća ograničenja na parametre $a_{i,j}$:

– **Glatka područja** - vrijednosti slikovnih elemenata su približno jednaki.

$$\sum a_{i,j} = 1 \tag{3.4}$$

Izraz 3.4 mora vrijediti jer ako su elementi u regiji slični, znači da i element za koji se predviđa vrijednost mora biti sličan kao i ostali u regiji. Većina slikovnih elemenata ima upravo regiju s tim svojstvom.

– **Područja s izraženom količinom šuma** - u ovom području je nemoguće predviđati slikovne elemente. U takvom slučaju potrebno je raditi što manju pogrešku. Pokazuje se da je za to potrebno minimizirati izraz:

$$\sum |a_{i,j}| \tag{3.5}$$

Za minimiziranje tog izraza potrebno kao predviđanje dati srednju vrijednost elemenata iz pripadajuće regije.

– **Područje rubova i tekstura** - vizualno najbitniji dio slike [28]. Za predviđanje je potrebna adaptivna metoda.

Da bi predviđanje bilo uspješno potrebno je odrediti svojstva regije i prema tome iskoristiti predikcijsku funkciju. U ovom radu koristi se funkcija predikcije s miješanjem. Funkcija predikcije s miješanjem kombinira različite predikcijske funkcije kako bi se u određenoj regiji koristila sva svojstva. Greška pojedine predikcijske funkcije funkcije određuje koja su svojstva dominantnija u regiji, tj. ako je greška manja predikcijska funkcija ima veću važnost u konačnom predviđanju. U nastavku slijedi detaljan

opis funckije predikcije s predviđanjem prema definiciji u [2].

3.2.2. Funkcija predviđanja s miješanjem (engl. *blending*)

Regija nekog slikovnog elementa može se prikazati u vektorskom obliku:

$$\mathbf{v_{i,j}} = (NN, NW, N, NE, WW, W) \tag{3.6}$$

Vektorski uzorak $v_{i,j}$ sačinjen je od 6 najbližih susjednih elemenata označenih kao:

- N sjeverni element: $I(x, y-1)$
- W zapadni element: $I(x-1, y)$
- NN element iznad sjevernog elementa: $I(x, y-2)$
- WW element lijevo od zapadnog elementa: $I(x-2, y)$
- NW element iznad zapadnog elementa: $I(x-1, y-1)$
- NE element desno od sjevernog elementa: $I(x+1, y-1)$

Prethodno opisano označivanje koristi se često u publikacijama koje obrađuju tematiku kompresije slika bez gubitaka metodama predviđanja.

Definira se područje Ω_C koje se naziva prozor pretraživanja. Prozor pretraživanja čine slikovni elementi koji su obrađeni i nalaze se u prozoru visine R i širine $2R$ (slika 3.4).

U prvom koraku funkcije predviđanja traže se takvi slikovni elementi čija je regija najmanje udaljena od regije trenutnog slikovnog elementa. Kako je regija predstavljena vektorom uzima se M minimalnih euklidskih udaljenosti prema izrazu:

$$D(x, y) = \|\mathbf{v_{i,j}} - \mathbf{v_{x,y}}\| \tag{3.7}$$

Pri čemu je regija za trenutni slikovni element (i,j), a (x, y) su elementi u prozoru pretraživanja ($(x, y) \in \Omega_C$). Za skup od M slikovnih elemenata čije su regije po udaljenosti od trenutnog elementa najmanje određen je kontekst za miješanje statičkih funkcija predviđanja Ω_B. Za utvrđivanje udaljenosti slikovnih elemenata iz prozora pretraživanja koristi se euklidska udaljenost, ali može se primijeniti i neka druga. Definira se skup od N predikcijskih funkcija $\{f_1, f_2, ..., f_N\}$. Za svaku od funkcija izračunava se penalitet G_k. Penalitet se definira prema tome koliko loše predikcijska funkcija predviđa slikovne elemente u dobivenom kontekstu za miješanje (izraz 3.8).

$$G_k = \frac{1}{M} \sum_{I(x,y)\in\Omega_B} (\hat{I}_k(x,y)) - I(x,y))^2 \tag{3.8}$$

Gdje je $\hat{I}_k$ predviđanje funkcije f_k za element u čeliji. Konačna funkcija predviđanja za tekući slikovni element računa se za trenutni element (i,j) računa se prema izrazu:

$$\hat{I}(i,j) = \frac{\sum_{k=1}^{N} \frac{1}{G_k} \cdot \hat{I}_k(i,j)}{\sum_{k=1}^{N} \frac{1}{G_k}} \tag{3.9}$$

Izraz 3.9 naziva se funkcija predikcije s miješanjem. Iz izraza se može vidjeti da svaka predikcijska funkcija f_k utječe na ukupan rezultat prema tome koliko griješi. Ako je greška manja predikcijska funkcija ima veći utjecaj. Ako se dogodi da penalitet G_k jednak 0, tada se druge predikcijske funkcije ne uzimaju u obzir, nego je izraz za predviđanje slikovnog elementa jednak predikcijskoj funkciji f_k.

U tablici 3.1 prikazane su konkretne funkcije f_k koje se koriste i njihove definicije.

Tablica 3.1: Funkcije predviđanja

Funkcija predviđanja
$f_N = I(x, y-1)$
$f_W = I(x-1, y)$
$f_{NW} = I(x-1, y-1)$
$f_{NE} = I(x+1, y-1)$
$f_{GW} = 2 \cdot I(x, y-1) - I(x, y-2)$
$f_{GN} = 2 \cdot I(x-1, y) - I(x-2, y)$
$f_{PL} = I(x, y-1) + I(x-1, y) - I(x-1, y-1)$

Parametri koji se koriste u prema gore opisanom algoritmu su:

- Veličina regije pretraživanja - R
- Veličina vektorskog uzorka (konteksta) - M
- Skup predikcijskih funkcija - f_k i dr.

Dodatno, kao simbol za kodiranje ne prosljeđuje se vrijednost pogreške, već se ona preslikava Potrebno je preslikati pogrešku tako da se održi šiljasta distribucija greške predviđanja. Moguće je i pronaći takvu funkciju preslikavanja da distribucija greške predviđanja bude pogodnija. Razlog je postizanje bolje kompresije klasičnim entropijskim kodiranjem [2].

3.2.3. Kontekstualno modeliranje greške predviđanja

Za ostvarivanje kompresije dovoljan bi bio i prvi korak. U prvom koraku kompresija je ostvarena na temelju svojstva slike, tj. redundantnih dijelova. Ako se pogreška prikaže u obliku slike, tako da na mjestu (x,y) nije vrijednost slikovnog elementa već pogreška, mogu se uočiti preostale strukturne zalihosti [4]. Ovaj problem rješava se kontekstnim modeliranjem greške predviđanja u kojem se određuje kontekst u kojem se greška predviđanja pojavljuje. Za određivanje konteksta koristi se trenutni slikovni element i greška predviđanja neposredno prije obrađenog slikovnog elementa. Zbog podatkovnog paralelizma greška neposredno prije obrađenog slikovnog elementa se ne koristi u radu kako je predloženo u [2].

Koristi se diskriminanta energije greške [26] koja je definirana:

$$\Delta = d_h + d_v + 2 * |e_w| \tag{3.10}$$

Gdje su d_h i d_v procijene gradijenta u horizontalnom, odnosno u vertikalnom smjeru i računaju se kao:

$$d_h = 2 \cdot |W - WW| + |N - NW| \tag{3.11}$$

$$d_v = 2 \cdot |W - NW| + |N - NN| \tag{3.12}$$

Gdje velika slova predstavljaju konkretne vrijednosti slikovnih elemenata u regiji. Oznaka e_w predstavlja pogrešku pretdhodno obrađenog slikovnog elementa.

Kao rezultat ovog postupka dobije se informacija koja govori koji histogram se koristi za kodiranje trenutnog slikovnog elementa. Ta informacija se zove kvantizirana energija. Kvantizirana energija se klasificira u 8 različithi vrijednosti kako je to predloženo na osnovu eksperimentalnih rezultata u [26]. Ovih 8 vrijednosti predstavlja kontekst za kodiranje koji se prosljeđuje entropijskom koderu. Svaki kontekst u sebi sadrži distribuciju vjerojatnosti pojavljivanja greške predviđanja na osnovu koje se u konačnici generira kodna riječ u entropijskom kodiranju.

3.2.4. Entropijsko kodiranje

Entropijsko kodiranje koristi grešku predviđanja i njezin kontekst. Na temelju distribucije greške predviđanja određuju se kôdovi. Kao entropijski koder u ovome radu koristi se aritmetičko kodiranje. Mogu se koristi i drugi klasični načini entropijskog

kodiranja. Kao drugi primjer klasičnog načina kodiranja je *Huffmanovo* kodiranje. Problem *Huffmanovog* kodiranja je u tome što su kodne riječi cjelobrojne duljine, što nije idealno za šiljaste distribucije (slika 3.1).

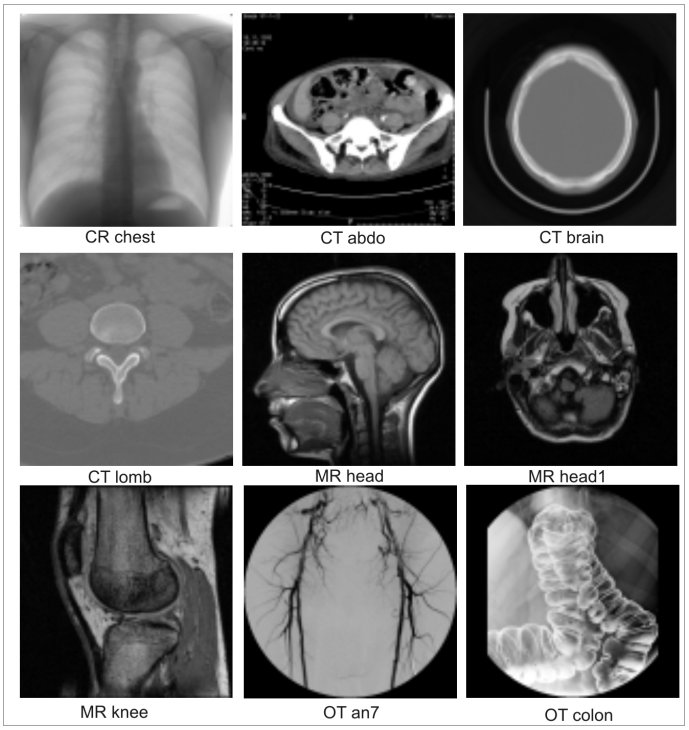
Kao izlaz ovog koraka dobije se tok binarnih podataka koji predstavljaju komprimirani niz slikovnih podataka iz ulaza.

3.2.5. Učinkovitost kompresije

Tablica 3.2 prikazuje ostvarene stupnjeve kompresije algoritma CBPC te daje usporedbu sa nekoliko popularnih postojećih metoda kompresije slika bez gubitaka. U zagradama su prikazane vrijednosti potrebnog broja bitova po slikovnom elementu, odn. simbolu. Skup ispitnih slika sastoji se od 9 medicinskih dijagnostičkih slika dobivenih različitim metodama, a prikazan je na slici **??**. Ovaj skup slika se često koristi u publikacijama za usporedbu ostvarenih stupnjeva kompresije. Algoritam CBPC ostvaruje najbolji stupanj kompresije (najmanji potreban prosječan broj bitova po slikovnom elementu). Međutim, potrebno je napomenuti da se poboljšanja u stupnju kompresije ostvaruju pomoću veće računske složenosti algoritma u odnosu na uspoređene algoritme. Na primjer, norma za kompresiju slika bez gubitaka JPEG–LS koristi jako jednostavnu nelinarnu funkciju predviđanja [27].

Tablica 3.2: Usporedba algoritama za kompresiju

Slika	CALIC	JPEG-LS	JPEG 2000	CBPC
CR chest	3.40 (2.35)	3.35 (2.39)	3.17(2.52)	3.52 (2.27)
CT abdo	3.52 (2.27)	4.23 (1.89)	3.09 (2.59)	4.17 (1.92)
CT brain	6.45 (1.24)	6.20 (1.29)	5.63 (1.42)	7.27 (1.10)
CT lomb	3.62 (2.21)	3.42 (2.34)	3.36 (2.38)	3.69 (2.17)
MR head	1.87 (4.28)	1.80 (4.44)	1.79 (4.47)	1.91 (4.19)
MR head1	1.80 (4.44)	1.73 (4.62)	1.70 (4.71)	1.85 (4.32)
MR knee	1.61 (4.97)	1.57 (5.10)	1.57 (5.10)	1.63 (4.91)
OT an7	2.16 (3.70)	2.18 (3.67)	2.02 (3.96)	2.21 (3.62)
OT colon	2.49 (3.21)	2.50 (3.2)	2.32 (3.45)	2.58 (3.10)
Geom. Mean	2.72 (2.94)	2.71 (2.95)	2.52 (3.17)	2.86 (2.80)



Slika 3.5: Medicinske slike korištene za ispitivanje performansi algoritama

4. Programski model za grafičke procesore *CUDA*

Cilj *CUDA* arhitekture je omogućiti korištenje grafičke kartice za opću namjenu. Za korištenje *CUDA*-e potrebno je koristiti grafičku karticu koja je podržana takvom arhitekturom što nužno znači da grafička kartica mora biti proizvedena od tvrtke *NVIDIA*. Za programiranje grafičkih procesora uz *CUDA* okruženje popularno je i *OpenCL* okruženje. *OpenCL* nešto je općenitije okruženje jer osim grafičkih kartica podržava i razne druge paralelne arhitekture. Za razliku od *CUDA* okruženja u kojemu ipak treba voditi računa o arhitekturi grafičke kartice, *OpenCL* okruženje je na višem apstraktnom nivou. Pozitivna strana je lakše programiranje i prenosivost programa, a negativna strana je što se ne može direktno utjecati na sklopovsku podršku koju grafičke kartice pružaju [29]. U nastavku rada koristi se *CUDA* okruženje.

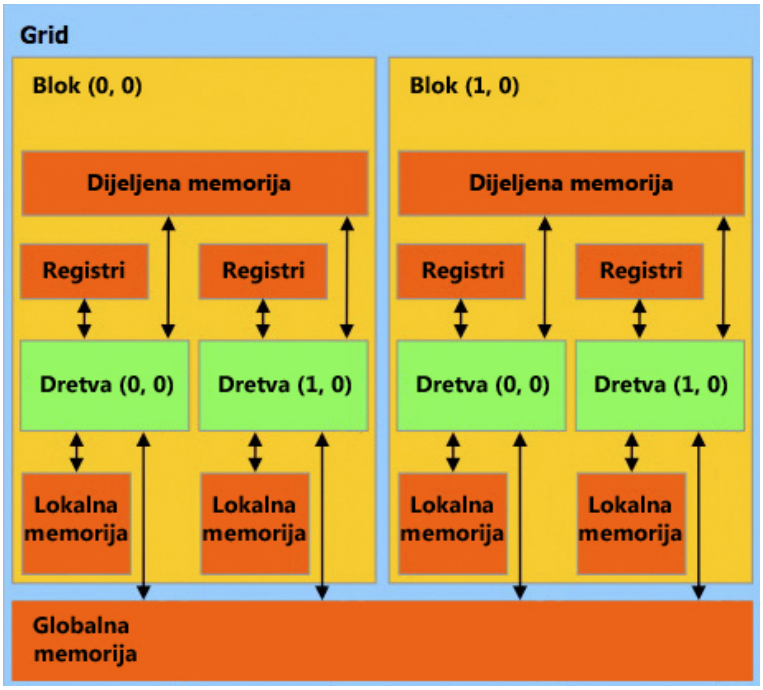
Za programiranje u *CUDA* okruženju potrebno je koristiti neki od omogućenih programskih jezika. U ovom radu korišten je *C/C++* programski jezik a njegovo proširenje za *CUDA* okruženje naziva se *CUDA C*. U nastavku rada pretpostavlja se njegovo korištenje.

Kao i u drugim paralelnim okruženjima tako je i za *CUDA* okruženje potrebno poznavati apstraktnu arhitekturu. Apstrakcija arhitekture omogućava lakšu prenosivnost aplikacija između različitih sklopovskih arhitektura grafičkih kartica, odnosno različitih generacija. Na slici 4.1 prikazana je *CUDA* arhitektura. Ovako prikazana arhitektura može se razložiti na dva modela - programski i memorijski model.

4.1. Programski model

Programski model definira na koji se način paralelno izvode instrukcije. *CUDA* programski model definira tri razine izvođenja. Najviša razina je *grid*[1]. Pri pozivu funk-

[1] Koristi se engleski naziv zbog sličnosti engleskih i hrvatskih riječi koje se koriste u tom kontekstu: *thread* i *dretva* te *block* i *blok*, kako bi se *grid* mogao prevesti s *mreža* to dovodi do određene nejasnoće



Slika 4.1: Prikaz *CUDA* arhitekture

cije (engl. *kernel*) na grafičkoj kartici pokreće se točno jedan *grid*. *Grid* se sastoji od blokova. Paralelne instrukcije se izvode u dretvama (engl. *thread*). Dretve se nalaze unutar blokova. Ograničenja za broj dretvi i blokova definirana su za svaku grafičku karticu posebno. Dretve u istim blokovima mogu komunicirati preko dijeljene memorije, dok dretve u različitim blokovima za komunikaciju trebaju koristiti globalnu memoriju. Pristup globalnoj memoriji je sporiji od pristupa dijeljenoj memoriji. Stoga prilikom izvedbe programa treba voditi računa da se podatci koji se dijele među dretvama nalaze u dijeljenoj memoriji.

Za razliku od klasičnog poziva funkcije koja se izvodi na centralnoj procesnoj jedinici (*CPU*), za poziv funkcije na grafičkoj karitici potrebno je koristiti nešto drugačju sintaksu koja je definirana u programskom modelu *CUDA*. Osim parametara funkcije potrebno je navesti koliko blokova i koliko dretvi će biti pokrenuto. Primjer je poziva funkcije addVector u programskom kôdu 4.1. Broj potrebnih blokova prenosi se kao predložak *blockno* u pozivu, a broj potrebnih dretvi kao predložak *threadno* u pozivu. Argumenti *vectorA*, *vectorB* i *res* predstavljaju ulazne vektore koje je potrebno obraditi te rezultatni vektor preko kojeg se vraća rezultat.

Programski kôd 4.1: Poziv funkcije na grafičkoj kartici

```
1    addVector<<<blockno, threadno>>>(vectorA, vectorB, res);
```

prilikom pregledavanja literature na engleskom jeziku.

4.2. Memorijski model

Memorijski model definira vrste memorija koje se mogu koristit prilikom programiranja. Vrste memorija se razlikuju u dosegu pristupa i vremenu pristupa kako je pikazano na slici 4.1.

4.2.1. Registri

Registri se mogu koristiti unutar samo jedne dretve. Svaki kooprocesor ima vlastite registre. Sve operacije koje se obavljaju, kao operande koriste vrijednosti koje su zapisane u registu, što znači da je prije obavljanja operacije potreban dohvat podataka u registar. Registri nisu vidljivi u programskom modelu. Oni se koriste implicitno što znači da nije moguće unaprijed odrediti koji podaci će koristiti registre. *CUDA* programski model nastoji sve lokalne varijable čuvati u registrima. Razlog je što su registri memorija s najmanjim vremenom pristupa podacima.

4.2.2. Lokalna memorija

Lokalna memorija se koristi unutar jedne dretve i nju koriste sve varijable koje su definirane unutar iste dretve. Fizički su podaci spremljeni na *DRAM* memoriji koja se nalazi na grafičkoj kartici (globalna memorija). Lokalni podatci za koje više nema mjesta u registrima spremaju se u lokalnu memoriju. Ako se za lokane varijable ne koriste registri, vrijeme pristupa je isto kao i za globalnu memoriju. U novjim verzijama *CUDA* okruženja ovaj nedostatak nastoji se popraviti tako što se lokalna memorija fizički odvaja od globalne memorije pa postaje nešto brža [7].

4.2.3. Dijeljena memorija

Dijeljena memorija je zajednička svim dretvama unutar bloka. Vrijeme pristupa je duže od pristupa registrima ali kraće od pristupa globalnoj memoriji. Dijeljena memorija omogućuje komunikaciju između dretvi u istom bloku. Osim same komunikacije dijeljena memorija omogućuje sinkronizaciju između dretvi. Za korištenje dijeljene memorije pri deklaraciji varijable potrebno je staviti oznaku "*__shared__*".

4.2.4. Globalna memorija

Globalna memorija je namjenjena za razmjenu podataka između grafičke kartice i centralne procesne jedinice. Ova memorija je najsporija memorija u memorijskom mo-

delu. Vrijeme pristupa je i za nekoliko redova veličina veće od dijeljene memorije [7]. Dobra strana globalne memorije je što je ona veća od svih ostalih. Pristup globalnoj memoriji može se ostvariti prosljeđivanjem pokazivača ili se može deklarirati varijabla s ključnom riječi "__*device*__".

4.2.5. CPU memorija

Memorija centralne procesorske jedinice nije vezana za grafičku karticu. Ona se pojavljuje u *CUDA* modelu kako bi se pokazala veza između grafičke kartice i ostatka računalnog sustava kada je potrebno prenijeti podatke iz radne meorije (CPU memorije) u memoriju grafičkog podsustava i obratno. Izmjena između memorija centralne procesorske jedinice i grafičke kartice obavlja se prije obrade podataka na grafičkoj kartici i nakon obrade podataka na grafičkoj kartici kako bi se rezultati mogli dalje obrađivati na centralnoj procesorskoj jedinici.

4.3. Detalji CUDA okruženja važni za izvedbu

4.3.1. Razmjena memorije

Prije pokretanja računskih operacija na grafičkom procesoru potrebno je prenijeti potrebne podatke iz radne memorije. Postoji više načina na koje se podaci mogu proslijediti grafičkoj kartici. Najčešće se upotrebljava sinkrono kopiranje podataka. Prije pokretanja funkcija na grafičkoj kartici potrebno je alocirati globalnu memoriju te u nju s posebnom naredbom kopirati podatke. U programskom kôdu 4.2 prikazan je primjer postupka. Deklarira se pokazivač u memoriji procesora. Pozivom funkcije *cudaMalloc* alocira se blok u globalnoj memoriji grafičkog sustava u kojeg će se prenijeti podaci iz radne memorije. Pri tome prvi argument predstavlja pokazivač na početak bloka memorije, dok drugi argument predstavlja veličinu prostora kojeg je potrebno zauzeti. Nakon toga je u alociranu memoriju na grafičkom sustavu potrebno kopirati podatke iz radne memorije. Za to se koristi funkcija *cudaMemcpy*. Njezin prvi argument predstavlja pokazivač na memorijsku lokaciju u koju će se kopirati podaci, drugi argument lokaciju bloka iz kojeg će se kopirati podaci, treći argument veličinu bloka podataka koji će se kopirati, dok četvrti parametar predstavlja zastavicu koja označuje kopiraju li se podaci iz globalne memorije grafičkog sustava u radnu memoriju ili obrnuto. Kad je to obavljeno, funkcija na grafičkoj kartici može biti pokrenuta. Nakon završetka funkcije potrebno je dobivene izlaze kopirati u memoriju procesora. Koristi se ista

funkcija kao i za obrnuti postupak samo što se navede drugi parametar. Konačno je naredbom *cudaFree* potrebno osloboditi zauzetu globalnu memoriju na grafičkim sustavima. Funkcija prima jedan argument, i to pokazivač na početak bloka podataka kojeg je potrebno osloboditi.

Programski kôd 4.2: Razmjena podataka između memorija

```
1  int *pictureCuda; //pokazivac na globalnu memoriju graficke
      kartice
2  cudaMalloc((void **) &pictureCuda,
      sizeof(int)*width*height); //alokacija globalne memorije
3  cudaMemcpy(pictureCuda, pixels, sizeof(int)*width*height,
      cudaMemcpyHostToDevice); //kopiranje iz memorije
      procesora u globalnu memoriju graficke kartice
4
5  ... //pozivi funkcija koje obavljaju racunske operacije nad
      primljenim podatcima
6
7  cudaMemcpy(newPixels, pictureCuda, sizeof(int)*width*height,
      cudaMemcpyDeviceToHost); //kopiranje iz globalne memorije
      graficke kartice u memoriju procesora
8  cudaFree(pictureCuda); //oslobadanje globalne memorije na
      grafickoj kartici
```

Osim sinkronog posoji i asinkroni prijenos podataka. Za postizanje takve komunikacije potrebno je korisititi malo drugačije funkcije. Umjesto standardnih funkcija za alokaciju memorije na memoriji centralne procesne jedinice koristi se funkcija iz *CUDA* okruženja: *cudaHostAlloc*. Operacijski sustav jamči da memorija rezervirana na ovakav način neće biti straničena. U svakom trenutku fizička adresa je očuvana. Zbog toga se za kopiranje memorije koristi *DMA* (engl. *Direct memory access*). Posljedica je veoma brza razmjena memorije centralne procesne jedinice i grafičke kartice. Funkcija koja kopira memoriju je *cudaMemcpyAsync*. Samo ako je memorija na centralnoj procesnoj jedinici zauzeta s funkcijom *cudaHostAlloc* može se koristiti asinkrona funkcija za kopiranje. Pojam asinkrone funkcije odnosi se na trenutak kopiranja memorije. Pri sinkronom načinu kopiranja završetak funkcije jamči da je prijenos podataka obavljen. Asinkrone funkcije samo daju oznaku da je potrebno kopirati podatke, ali kada će to biti nije određeno. Asinkrone funkcije mogu se iskoristiti za ostvarivanje programskog cijevovoda (engl. *pipeline*). Okruženje *CUDA* nudi *Stream* tehniku poziva funkcija na grafičkoj kartici. Ideja je da se u isto vrijeme izvodi funkcija na grafičkoj kartici i da se kopira memorija na grafičku karticu koja će biti korištena prilikom poziva slijedeće funkcije. Za *Stream* tehniku potrebno je koristiti asinkrone funkcije.

4.3.2. Sinkronizacija

Kada više dretvi djeluje nad istom memorijom, za postizanje determinizma potrebna je sinkronizacija. U *CUDA* okruženju postoji više mehanizama sinkronizacija [30]. Najčešće se koristi sinkronizacija između dretvi koje se nalaze u istom bloku *__syncthreads()*. Za sinkronizaciju svih dretvni koristi se funkcija *__threadfence()* i ona je znatno sporija jer koristi globalnu memoriju. Ova metode umeću barijeru u sve dretve na mjestu poziva pa će dretve nastaviti izvođenej tek nakon što sve dostignu do navedene barijere. Za sinkronizaciju funkcije koje izvodi procesor i dretvi na grafičkim karticama koristi se funkcija *cudaThreadSynchronize()*. Nakon što funkcija koja se izvodi na procesoru pozove navedenu funkciju, procesor čeka grafičku karticu da obavi posao koji ima.

4.3.3. Optimizacija pristupa memoriji

Pravilnim korištenjem *CUDA* okruženja može se dobiti logički ispravna aplikacija. Problem je što se ne jamči ubrzanje. Definirani programski model i memorijski model ne omogućuju direktno upravljanje optimizacijama. Za ubrzanje aplikacije nekad je dovoljno postići samo logičku ispravnost te se ubrzanje očituje u paralelnoj arhitekturi. Ali ponekad se dogodi da je paralelizirana aplikacija sporija od serijske. Problem je najčešće u memoriji i nedovoljnom poznavanju mehanizama koji se kriju iza upravljanja memorijom.

Za *CUDA* okruženje biti će objašnjen mehanizam koja može ubrzati aplikaciju a pomoću njega biti će objašnjeni neki rezultati na kraju ovoga rada.

Programski gledano dretve u blokovima se izvršavaju paralelno, ali na sklopovlju (grafičkoj karitici) to nije tako. Svaki blok se stastoji od skupine *warp*-ova koji se mogu fizički paralelno izvoditi. Jedan *warp* sastoji se od 32 dretve. Često je bitnji pojam *half-warp* koji označava nakupinu od 16 dretvi.

Spajanje dohvata globalne memorije (engl. *Coalescing*)

Poželjno svojstvo algoritama koji su napisani za programski model *CUDA* jest da dretve dohvaćaju globanu memoriju prema svojem identifikacijskom broju i to tako da dretva i dohvaća podatak na lokaciji $k \cdot i$ a dretva $i + 1$ na lokaciji $k \cdot (i + 1)$ i tako redom, gdje je k prirodni broj. Ovakav pristup globalnoj memoriji u sklopovlju grafičkog procesora spaja se u jedan dohvat, što znatno skraćuje vrijeme pristupa globalnoj memoriji, a samim time i ubrzava rad cjelokupnog programa. Za različite verzije ar-

hitekture globalni pristup je poboljšavan, pa tako u novjim verzijama nije potrebno da dretve pristupaju prema svojoj identifikaciji već je dovoljno da razlika adresa memorijskih lokacija ne bude veća od 16, kako bi dohvat podatka bio lokaliziran u *half-warp*-ovima.

5. Izvedba algoritma CBPC korištenjem programskog modela *CUDA*

U ovom poglavlju bit će detaljno opisana izvedba prethodno opisanog algoritma za kompresiju slika bez gubitaka.

Kao što je navedeno u jednom od prijašnih poglavlja algoritam se izvodi u ovim koracima (slika 3.2).

1. **Ulaz** - Učitavanje slikovnih podataka: učitavanje slike ili niza slika iz pojedine studije
2. **PM - Predikcijsko modeliranje**: predviđanje trenutnog slikovnog elementa i računanje pogreške
3. **CM - Kontekstualno modeliranje**: klasifikacija pogreške i generiranje različitih distribucija za svaki razred pogreške
4. **EC - Entropijsko kodiranje**: korištenje klasičnog načina kodiranja pogreške predikcije.
5. **Izlaz** - Spremanje komprimiranih podataka

Tablica 5.1: Prikaz postotaka vremena izvođenja ovisno o koraku algoritma

Korak algoritma	Vrijeme izvođenja (%)
Ulaz	6.71
PM	84.69
CM	1.84
EC	5.05
Izlaz	1.71

Tablica **??** prikazuje koliko je koji korak algoritma zahtjevan. Najzahtjevniji dio je predikcijsko modeliranje. Zbog toga se taj korak odabire za paralelizaciju.

5.1. Ciljevi izvedbe

Glavni cilj ove izvedbe algortima za kompresiju je ubrzanje središnjeg dijela algoritma, odnosno dijela algoritma koji se odnosi na predviđanje slikovnih elemenata. Razlog tome je što taj korak algoritma najzahtjevniji (tablica **??**). Ubrzanje se ostvaruje paralelizacijom na grafičkom procesoru. Razlog odabira grafičkog procesora kao sklopovlja na kojem se algoritam izvršava je u tome što je predviđanje slikovnih elemenata moguće ostvariti podatkovnom paralelizacijom, a rad grafičkih procesora upravo se zasniva na podatkovnom paralelizmu, *SIMD*. (engl. *Single Instruction Multiple Data*).

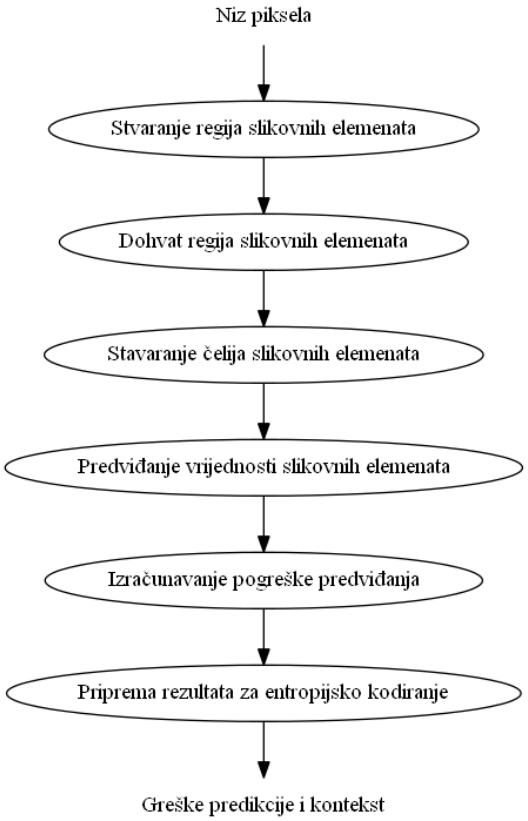
Prije izvedbe algoritma postavljeni su zahtjevi koji su morali biti ostvareni. Bitan zahtjev postavljen pri izvedbi je prenosivost izvornog programa prema novim generacijama grafičkih procesora koji podržavaju programski model *CUDA*. Kako bi se održala jednostavnost korišten je tokovni programski model. Tokovni programski model omogućuje brzi uvid u komponente algoritma te pojednostavnjuje prenošenje algoritma na druga okruženja i arhitekture. Tokovni model prikazuje se pomoću usmjerenog grafa gdje su čvorovi aktori, odnosno računski posao, a veze komunikacija između aktora. Paralelizacija algoritama na grafičkim procesorima često zahtjeva optimiranje do najmanjih detalja. Takvim programiranjem dobiva se veliko ubrzanje ali složenost i razumljivost programskog kôda pada. Tokovni programski model predstavlja kompromis između ta dva zahtjeva.

Kako je glavna ideja kompresija podataka, izvedba za grafičke procesore ne smije narušiti stupanj kompresije. Problem koji bi mogao narušavati supanj kompresije je sama arhitektura grafičke kartice, te je pojedine operacije potrebno modificirati kako bi algoritam mogao biti izvodljiv.

Glavni zahtjev u izvedbi je ostvarenje ubrzanja u odnosu na serijsku izvedbu algoritma. Na taj način bi se omogućila njegova upotreba u stvarnim aplikacijama. Dodatno, upotrebom grafičkih procesora kao ubrzivača omogućeno je komprimiranje podataka na računalima koji se nalaze uz dijagnostičke uređaje. To znači da se preko mrežne infrastrukture informacijskog sustava prenose manje količina podataka a samim time ostvaruju se uštede u zahtjevanoj propusnosti. Cilj koj mora biti ostvaren a da izvedba ima dodanu vrijednost je ušteda električne energije. Skup svih navedenih ciljeva čini izvedbu pogodnu za bolničke sustave ali i za druge slične sustave.

5.2. Arhitektura modela

Izvedba algoritma sastoji se od dva glavna dijela: paraleliziranog i neparaleliziranog. Kako je navedeno, u neparalelizirani dio spadaju elementi izvedbe koji se brinu za učitavanje slike, spremanje slike te entropijsko kodiranje (aritmetički koder). Neparalelizirani (serijski) dio ostvaren je korištenjem programskog jezika C++, i taj dio se neće detaljnije razmatrati u ovom radu. Paralelizirani dio izvedbe, kao što je već i prije napomenuto, odnosi se na predviđanje vrijednosti slikovnih elemenata. U nastavku razmotriti će se aritektura sustava (algoritma). Tokovni programski model koji je korišten za izvedbu prikazan je na slici 5.1. Prikazan je dio algoritma koji se paralelizira. Na slici je prikazan trivijalan tokovni model (nema grananja) i zbog toga je pogodan za izvođenje na grafičkim procesorima. Čvorovi u grafu prikazuju operacije koje se moraju obaviti nad svakim slikovnim elementaom. Važno je da se koraci obavljaju pravilnim redoslijedom. Podatkovna paralelizacija izvedena je tako što se prikazani tokovni model obavlja paralelno na različitim grafičkim koprocesorima i nad nezavisnim slikovnim elementima.



Slika 5.1: Tokovni model

Ulaz u tokovni model je vrijednost slikovnog elementa koji se nalazi na slici. Nad

pojedinim slikovnim elementom obaviti će se operacije zadanim redoslijedom. Unutar tokovnog modela osim vrijednosti slikovnog elementa koriste se i druge strukture podataka (regija slikovnih elemenata, kontekst za mješanje predikcijskih funkcija). Kao izlaz iz tokovnog modela dobije se greška predviđanja, njezin kontekst i simbol koji je potrebno kodirati entropijskim koderom (entropijski koder koristi sve izlazne podatke).

Programski model *CUDA* okruženja zahtjeva organizaciju programa u blokovima i dretvama. Granulacija korištena pri izvedbi je na razini slikovnog elementa, što znači da se slikovni elementi pripajaju pojedinim dretvama te svaka dretva brine o pojedinom slikovnom elementu. Ideja je da se svaka dretva brine o pojedinom slikovnom elementu. Nakon što se slika učita u globalnu memoriju grafičke kartice, ona ostaje tamo do kraja izvođenja algoritma. Kako je broj blokova i dretvi ograničen (ovisno o veličini globalne memoriji i generaciji grafičke kartice), slika se ne obrađuje paralelno u potpunosti. Izvedba je izvedena tako da se dimenzije podslike koja se potpuno paralelno obrađuje može dinamički mijenjati. Razlog tome je ostvarivanje optimizacije prema broju dretvi i blokova što je opisano u sljedećem podpoglavlju.

Poziv modula koji izvršava korak predviđanja na grafičkoj kartici prvo pokreće potrebne inicijalizacijske radnje kako bi sve bilo pripremljeno za grafičku karticu. Nakon toga kreće paralelna obrada podslika. Programski kôd 5.1 prikazuje kako je ostvarena funkcija za izvođenje grafičkom procesoru korištenjem navedenog tokovnog programskog modela.

Programski kôd 5.1: Glavna funkcija koja se izvodi na grafičkom procesoru

```
1  PixelRegionProducer(widthOffset, heightOffset, pixels,
       symbols, width, height, templsize, radius, cellsize, cpr);
2
3   __threadfence();
4
5   PopulateCells(widthOffset, heightOffset, pixels, symbols,
       width, height, templsize, radius, cellsize, cpr);
6
7   PredictorPenalty(widthOffset, heightOffset, width, height,
       cpr);
8
9   ComputePredictionPenaltiesAndContext(widthOffset,
       heightOffset, width, height, cellsize, cpr, symbols);
```

U trećem retku vidljivo je korištenje globalne sinkronizacije. Ona je potrebna zbog toga što funkcija koja stvara regije slikovnih elemenata (*PixelRegionProducer*) koristi podatke iz slikovnih elemenata za koje dretva nije zadužena. Kako se radi o nedestruktivnim operacijama čitanja podataka iz drugih dretvi, ali da bi se osigurala konzistent-

nost u slijedećm koraku potrebna je sinkronizacija. Između ostalih funkcija ona nije potrebna.

5.3. Optimizacija

Nakon što je algoritam predikcije paraleliziran i tako postignuto već određeno ubrzanje, postoje još neki dodatni faktori koje je potrebno razmotriti prilikom izvedbe. Kako je u trećem poglavlju navedeno, prilikom poziva funckije koja se izvodi na grafičkim procesorima potrebno je specificirati broj dretvi i broj blokova za izvođenje te funkcije. Iako se to čini kao jednostavan problem, jer je intuitivno da će se veće ubrzanje postići ukoliko se pokrene što više dretvi na što više blokova, empirijski rezultati su pokazali da to ne mora nužno biti tako. Stoga se kao problem prilikom izvedbe javlja i optimizacija broja blokova i dretvi s kojima se pokreće funkcija na grafičkim procesorima. Ako se pokaže da je optimizacija broja blokova teži k relativno malom broju, potrebno je probati da li korištenje sinkronizacije unutar jednog bloka (koja je brža zbog korištenja dijeljene memorije) daje bolje rezultate. Najčešće se to ne dogodi jer broj blokova nadoknadi vrijeme sinkronizacije. Ovdje se ova optimizacija spominje kao jednostavan način poboljšanja vremena izvođenja. U šestom poglavlju biti će objašnjeno za koje se vrijednosti tih parametara dobivaju najbolja vremena izvršavanja.

To nije jedina mogućnost optimizacije. Ova izvedba se uvelike temelji na tokovnom programskom modelu, koji je prilagođen za *SIMD* model. Naravno da se izvedba može još više prilagoditi za specifičnosti *CUDA* modela i tako bi se moglo ostvariti dodatno ubrzanje (primjerice smanjivanjem pristupa globalnoj memoriji korištenjem optimizacije pristupa). Time bi se nažalost izgubilo na jasnoći, jednostavnosti i modularnosti pogramskog kôda te bi se smanjila mogućnost za održivost programskog kôda u budućnosti. Upravo zbog tih razloga trenutno se nije krenulo na daljnje prilagođavanje izvedbe u *CUDA* modelu.

Programski kôd 5.2: Glavna funkcija koja se izvodi na grafičkom procesoru

```
1
2   for(int j = 0 ; j < height ; j += stepY)
3     for(int i = 0 ; i < width ; i += stepX)
4     {
5       runAlgorithm<<<stepY,stepX>>>(i, j, pictureCuda,
            cudaSymbols, width, height, templsize, radius,
            cellsize, pixelRegionCuda);
6     }
```

Programski kôd 5.2 pokazuje kako se poziva funckija na grafičkoj kartici. Koriste se parametri *stepY* i *stepX* za određivanje visine i širine podslike koja se u potpunosti paralelno obrađuje. Naziv funkcije koja se izvršava na grafičkoj kartici je *runAlgorithm*. Sintaksa poziva pokazuje da se radi o funkciji za grafičku karticu. Unutar trostrukih šiljastih zagrada navodi se broj blokova i dretvi koji će biti pokrenuti za paralelno izvođenje. Algoritam je dizajniran tako da svaki blok predstavlja jedan redak slike, a dretva određeni slikovni element u tome redu.

6. Rezultati

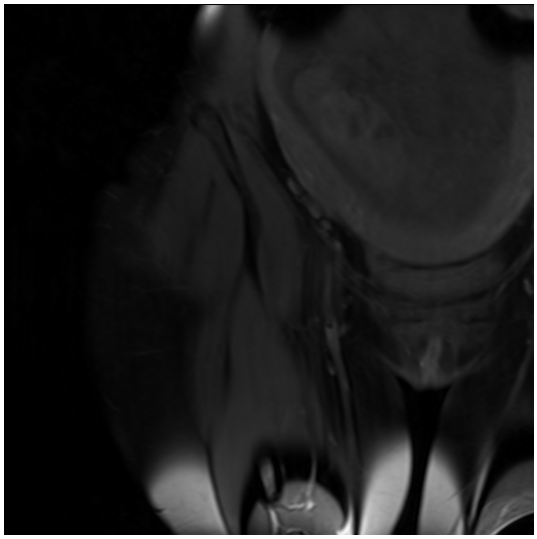
6.1. Ispitivanje

Izvođenje promijenjenog programa *CBPC* ispitano je na 5 različitih studija medicinskih slikovnih podataka. Tablica 6.1 prikazuje osnovna svojstva korištenih studija. Svi slikovni podaci su pohranjeni u formatu *PGM* i imaju veličinu od 512x512 piksela (izvedba naravno ne ovisi o veličini slike). Primjer jedne medicinske slike prikazan je na slici 6.1. Za svaku od studija izmjereno je trajanje izvođenja algoritma kao i potrošnja električne energije potrebna za komprimiranje pojedine studije. Osim navedene izvedbe, nad ovim podacima provedena su testiranja i sa serijskom izvedbom algoritma, kao i izvedbom u kojoj je predikcijski model izveden u programskom jeziku *StreamIt* [31] i paraleliziran za izvođenje na višejezgrenom procesoru [32], radi mogućnosti njihove međusobne usporedbe.

Tablica 6.1: Ispitne studije

Naziv studije	Broj slika	Veličina studije [MB]
mr-breast-dm	2534	486,43
mr-breast	2241	559,53
mr-pelvis	383	95,76
ct-mix	3003	750,79
ot	1373	343,27

Osim navedenih ispitivanja i međusobne usporedbe o potrošnji i ubrzanju, provedena su još neka dodatna ispitivanja nad ovim izvedbama. Tako je primjerice ispitano kako veličina slike utječe na vrijeme potrebno za njeno komprimiranje za svaku od ovih izvedaba. Nadalje, provedena je usporedba brzine izvršavanja za dvije različite generacije grafičkih kartica te je prikazano kako se napredak grafičkih kartica odražava na postignuto ubrzanje. U konačnici je provedeno ispitivanje i o tome kako brzina izvođenja na grafičkim karticama ovisi o tome s kolikim je brojem blokova i dretvi pokrenuto izvršavanje algoritma.



Slika 6.1: Primjer medicinske slike

U tablici 6.2 mogu se vidjeti informacije o komponentama kao i verzijama programskih alata koji su korišteni tijekom ispitivanja gore navedenih izvedbi algoritma za kompresiju.

Tablica 6.2: Konfiguracija ispitnog računala

Operacijski sustav	Ubuntu 11.04
Procesor	Intel Core2 Quad Q9400 @ 2.66GHz (4 jezgre)
Radna memorija	8 GB
Grafička kartica	nVidia GeForce GTX 550 Ti
Alternativna grafička kartica	nVidia GeForce 9800 GT
Verzija *GCC* prevoditelja	4.6.1
Verzija *CUDA* prevoditelja	4.2

U idućim poglavljima biti će detaljnije uspoređeni rezultati koji su dobiveni po pojedinim kategorijama.

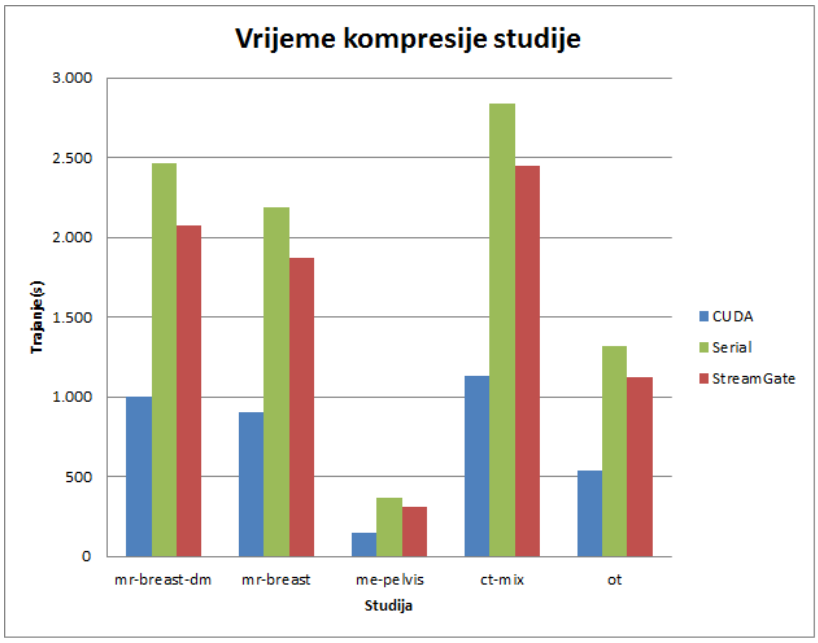
6.2. Vrijeme izvođenja

6.2.1. Vrijeme izvođenja kompresije nad studijama

Jedan od važnijih kriterija ocjene pojedine izvedbe je brzina izvođenja komprimiranja nad čitavim studijima. Kako je u tablici 6.1 pokazano u prethodnom poglavlju, veličina pojedinih studija može biti i do nekoliko stotina MB i sadržavati nekoliko tisuća slika. Dakle radi se o velikoj količini podataka koju je potrebno obraditi. Zbog toga je

vrlo bitno u kojem se vremenskom intervalu može obaviti komprimiranje jedne velike količine podataka.

Na slici 6.9 prikazani su dobiveni rezultati za vrijeme izvođenja svih gore navedenih izvedbi za 5 pripremljenih studija. Sa *StreamGate* označena je izvedba koja koristi programski jezik *StreamIt* za paralelizaciju predikcijskog modela na višejezgrenom procesoru - u našem slučaju na 4 jezgre procesora računalnog sustava koji je korišten u ispitivanjima.



Slika 6.2: Rezultati trajanja izvođenja algoritma na pojedinim izvedbama u minutama

Kao što je iz grafičkog prikaza rezultata vidljivo, izvedba algoritma na grafičkim koprocesorima (*CUDA*) pokazuje veoma dobra ubrzanja naspram serijske izvedbe, ali i paralelne izvedbe, koja se izvodi na jezgrama procesora opće namjene bez korištenja grafičkog koprocesora, bazirane na tokovnom podatkovnom modelu (*StreamGate*). U izvedbi korištenjem grafičkih procesora postignuto je srednje ubrzanje od otprilike 2.5 puta u odnosu na početnu, serijsku izvedbu. Iako ovo ubrzanje na prvi pogled ne djeluje pretjerano veliko, mora se ipak uzeti u obzir činjenica da je ova izvedba izvedena prateći tokovni programski model i kako nisu obavljene dodatne prilagodbe algoritma ili optimizacije koda koje bi najvjerojatnije rezultirale dodatnim ubrzanjem ove izvedbe, a što je predviđeno za buduća istraživanja. Svakako treba uzeti u obzir i ograničenja u paralelnoj učinkovitosti cjelokupnog algoritma. Iako je skalabilnost razmotrena u sklopu ovog rada u smislu poboljšanja koje donose nove generacije grafičkih procesora, detaljnija analiza skalabilnosti samoga algoritma je svakako potrebna prije nego što se krene u dodatne pokušaje paralelizacije.

U tablici 6.3 prikazana su vremena potrebna za kompresiju svake pojedine studije izražene u minutama. Za provedene ispitne studije postignuta je prilično velika ušteda u vremenu potrebnom za obradu. Tako za studiju mr-breast-dm može vidjeti da je serijskoj izvedbi potrebno otprilike četrdeset minuta, dok je izvedbi na grafičkim procesorima potrebno svega sedamnaest minuta.

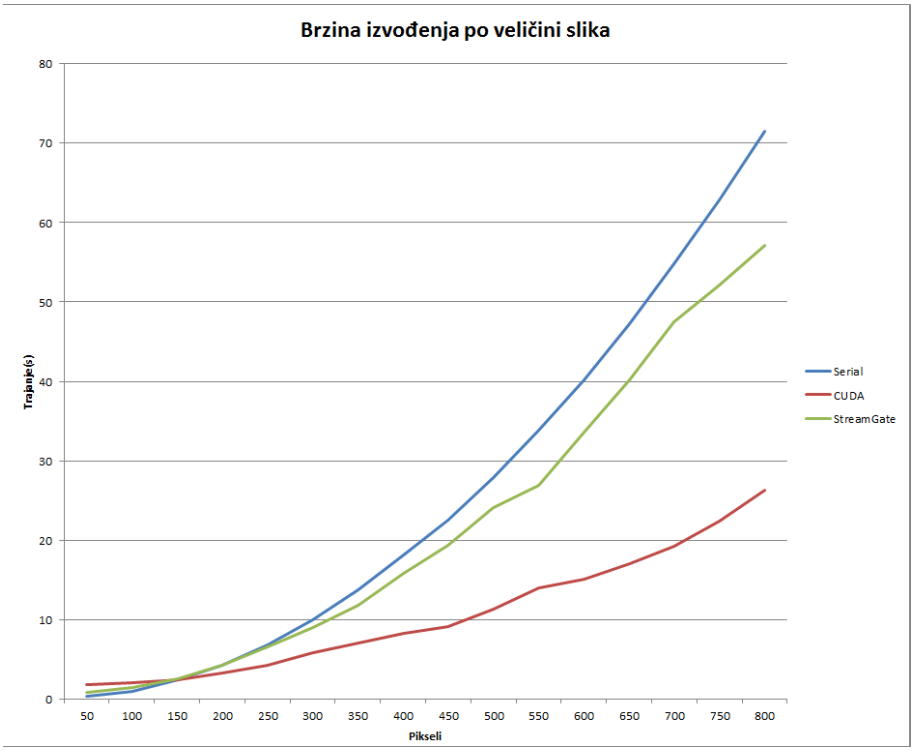
Tablica 6.3: Brzina kompresije po studijama

Naziv studije	CUDA izvedba	StreamGate izvedba	Serijska izvedbe
mr-breast-dm	16.7	34.6	41.02
mr-breast	15.02	31.23	36.51
mr-pelvis	2.45	5.16	6.13
ct-mix	18.85	40.78	47.33
ot	8.93	18.73	21.97

Također se iz grafičkog prikaza može vidjeti da *StreamGate* izvedba ne postiže neke puno bolje rezultate od serijske izvedbe. Razlog tome je da je serijska izvedba visoko optimirana. Prilikom prevođenja *StreamIt* programa, generira se program u programskom jeziku *C* koji je gotovo u potpunosti orijentiran na paralelno izvođenje. Naravno da takav generirani *C* program nije optimiziran kao što je serijska izvedba i zbog toga se uvelike gubi na brzini. Osim toga *StreamGate* izvedba očekuje veliku količinu podataka na ulazu kako bi se prekrila dodatna vremena potrebna za kreiranje sinkronizaciju i terminiranje dretvi na centralnoj procesorskoj jedinici. Ovaj nedostatak dolazi do izražaja upravo kod obrade slika manjih dimenzija, kao što su slike u ovim studijima. Bolja strategija za *StreamGate* izvedbu bi bila da se učita niz slika te da se odjendom pokrene njihova obrada. Bez obzira na sve navedeno, postignuto ubrzanje nije uopće zanemarivo te bi se sigurno moglo postići i još veće ubrzanje kada bi se ta izvedba dodatno prilagodila.

6.2.2. Ovisnost vremena izvođenja o veličini slike

Grafovi na slici 6.3 prikazuju ovisnost vremena izvođenja algoritma o veličini slike. Mjerenje je izvršeno za sve tri izvedbe. Izvedba na grafičkoj kartici postiže najbolje rezultate. Zanimljivo je što za male slike (manje od *150x150*) serijska izvedba najbolja. Razlog zbog kojeg izvođenje na grafičkoj kartici traje najduže je u tome što vrijeme potrebno za kopiranje radne memorije procesora u globalnu memoriju grafičke kartice oduzima više vremena od same izvedbe algoritma. Serijska izvedba sve podatke ima u radnoj memoriji i nije potrebno dodatno kopiranje. *StreamGate* verzija za male slike



Slika 6.3: Rezultat trajanja izvođenja u ovisnosti o veličine slike

ima slično vrijeme izvođenja kao i serijska izvedba. Kako i *StreamGate* verzija ne koristi dodatno kopiranje može se zaključiti da je zaista problem u izvedbi za grafičku karticu kopiranje memorije. U praksi se najčešće ne pojavljuju ovako male slike pa pretdhodno razmatranje u praksi nije toliko važno. Dodatno, uz manje dimenzije slike, postoji mogućnost da se grafičkom procesoru pošalje slijed slika kako bi se optimirao prijenos podataka odnosno povećao na zadovoljavajuću veličinu. To je moguće ostvariti s obzirom na činjenicu da se medicinske studije sastoje od velikog broja slika. Ali općenito je važno uočiti problem koji posjeduju grafičke izvedbe - vrijeme kopiranja memorije može biti puno veće od vremena izvođenja samog algoritma. Za veće slike vrijeme kopiranja memorije prekriva se vremenom izvođenja algoritma.

6.2.3. Ovisnost vremena izvođenja o broju dretvi i blokova

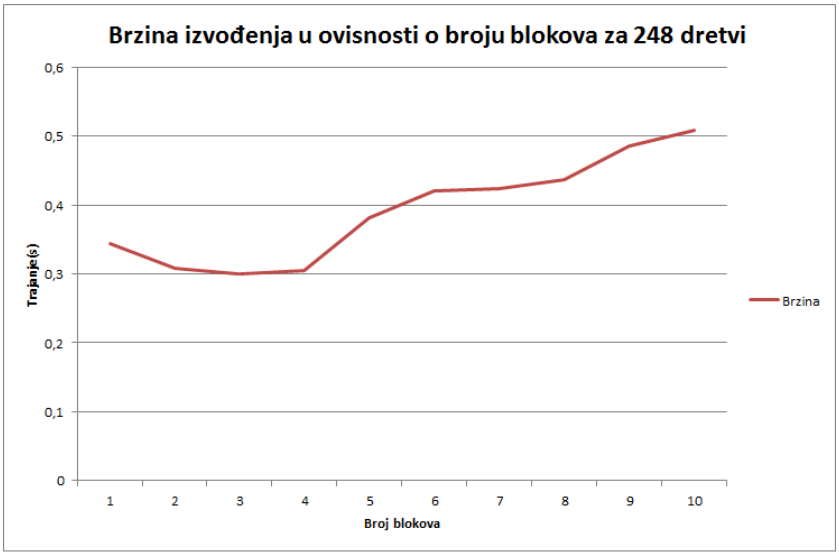
Jedna od bitnih stvari koje je potrebno napraviti prilikom izrade programa koji će se obavljati grafičkim karticama u programskom modelu *CUDA*, jest odrediti broj blokova i dretvi za optimalno izvršavanje programa. To je bitan korak jer o odabiru ovih parametara uvelike ovise performanse izvođenja. Odabirom krivih vrijednosti može se dobiti vrijeme izvođenja koje je nekoliko puta dulje od optimalnog vremena izvođenja. Stoga veliku pažnju potrebno posvetiti traženju optimalnih vrijednosti za broj blokova

i broj dretvi, kako bi se postigle bolje performanse.

Nažalost traženje optimalnih parametara se svodi na isprobavanje svih smislenih kombinacija za vrijednosti broja blokova i dretvi te usporedbom dobivenih vremena za pojedine ispitne podatka. Dodatni problem predstavlja i činjenica da parametri koji su optimalni za jednu grafičku karticu ne moraju biti i optimalni za druge grafičke kartice. Time se dovodi u pitanje prenosivost ovakve izvedbe, jer bi se za svaku grafičku karticu trebalo ponovo odrediti broj blokova i dretvi. Na svu sreću pokazalo se da za isprobane grafičke kartice ti parametri nisu previše odstupali jedni od drugih, a vremena izvođenja između njih nisu bitno različita. Primjerice za grafičku karticu GTX 550 (nova generacija) dobiveni su optimalne veličine od 3 bloka i 248 dretvi, dok su za karticu GT 9800 (stara generacija *CUDA* kartica) dobivene veličine iznosile 1 blok i 256 dretvi. Vrijeme izvođenja paraleliziranog dijela algoritma, na jednoj medicinskoj slici, za parametre od 3 bloka i 248 dretvi na grafičkoj kartici GTX iznosilo je 0.300024 dok bi za parametre od 1 bloka i 256 dretvi iznosilo 0.354034. Dakle vidi se da razlika postoji, ali ona nije toliko velika da bi značajnije utjecala na vrijeme izvođenja. U daljnjem radu, ako bi se uspio prikupiti veći broj takvih ispitivanja za različite modele grafičkih kartica, mogla bi se odrediti heuristička metoda koja bi na temelju svojstava grafičke kartice mogla procijeniti broj blokova za pokretanje programa s ciljem dobivanja što boljih performansi. Ovdje je bitno napomenuti da "optimalan" raspored računskih jedinica za izvođenje na grafičkom procesoru (broj blokova i dretvi) uvelike ovisi i o rasporedu podataka kojima pristupaju dretve odnosno o dekompoziciji podataka koju je programer načinio prilikom izvedbe programa. To znači da je dosta teško automatizirati ovaj korak određivanja konfiguracije pa on ovisi o iskustvu programera u korištenju programskog modela.

Na slici 6.4 prikazan je graf koji pokazuje kako se vrijeme izvođenja ponaša za promjenu broja blokova, dok se broj dretvi drži konstantnim (u ovom slučaju broj dretvi je postavljen na 248, što predstavlja optimum za ispitanu grafičku karticu). Ovi rezultati dobiveni su izvođenjem na grafičkoj kartici GTX 550. Kako se iz slike može vidjeti brzina izvođenja se prvo smanjuje dok ne dosegne optimum za veličinu od tri bloka, a nakon toga počinje sve brže rasti te dolazi do degradacije performansi. Očito je da se nakon određenog broja blokova (u ovom slučaju tri) počinje događati zagušenje te da su dretve sve više blokirane i zbog toga dolazi do sve većeg opadanja performansi.

Slika 6.5 s druge strane prikazuje kako promjena broja dretvi utječe na performanse izvođenje ako se blokovi drže konstantnima. Ovo ispitivanje je također provedeno na grafičkoj kartici nVidia GeForce 550 GTX. Prikazani su rezultati za tri odabrane veličine bloka (1, 3 i 7) a za svaki od tih blokova prikazana su vremena izvođenja za

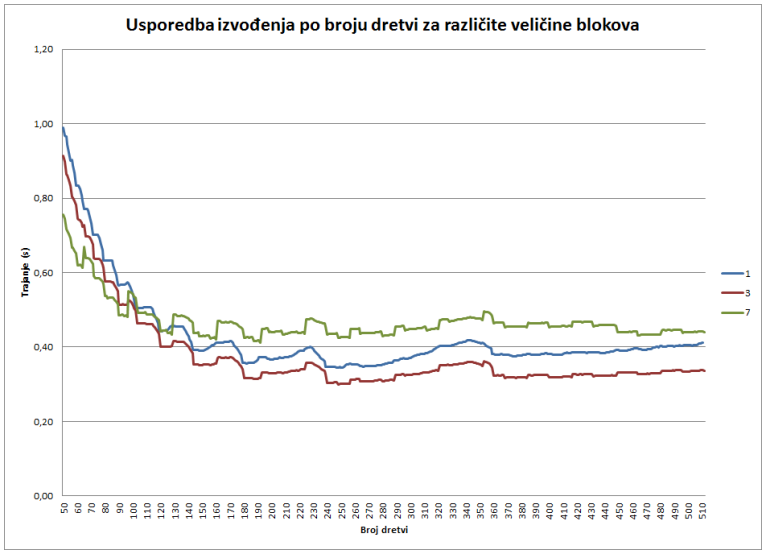


Slika 6.4: Rezultat trajanja izvođenja s obzirom na broj korištenih blokova

broj dretvi od 50 do 512. Manji broj dretvi od 50 po bloku nije poželjno uzimati jer vrijeme izvođenja veoma brzo raste. Razlog tome je što se izvedba algoritma počinje sve više ponašati kao serijska izvedba, odnosno obrađuju se manje količine slikovni elemenata u isto vrijeme. Tako bi se primjerice za veličinu od jednog bloka i jedne dretve ova izvedba ponašala u potpunosti serijski, samo što bi njena izvedba trajala mnogo dulje od trajanje stvarne serijske izvedbe bez pomoći grafičkog koprocesora. Na grafu se može vidjeti kako se za mali broj dretvi postižu najlošija vremena. Također je vidljivo da za veći broj blokova ta vremena jesu nešto manja, što je i logično s obzirom da će se pokrenuti više dretvi zbog veće količine blokova. Kako se broj dretvi povećava, tako se i vrijeme izvođenja počinje smanjivati. No to smanjivanje traje samo do određenog broja dretvi (za ove primjere otprilike 200-250) te se nakon toga vrijeme izvođenja počinje ponovo povećavati. To je iz razloga što se stvori preveliki broj dretvi, a kao što je opisano u poglavlju 4.3.3. dretve se samo prividno izvršavaju paralelno, ali je sklopovski ograničeno koliko se dretvi unutar jednog bloka u isto vrijeme može izvršavati. Vidljivo je da za veličinu od tri bloka i za broj dretvi veći od otprilike 100, postižu se najbolje performanse što se brzine izvedbe tiče. Najlošije performanse se s druge strane dobivaju za broj od sedam blokova.

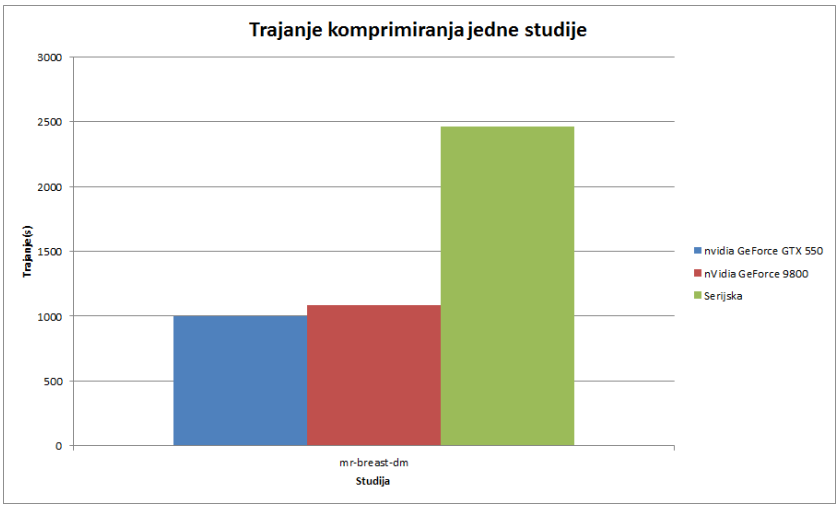
6.2.4. Usporedba različitih grafičkih procesora

Na slici 6.6 prikazana su vremena izvođenja algoritma za pojedinu generaciju grafičke kartice. Za relativnu usporedbu u grafu je prikazano i vrijeme izvođenja za serijsku izvedbu. Vidljivo je da vrijeme izvođenja na novijoj generaciji grafičkih kartica nije



Slika 6.5: Rezultat trajanja izvođenja s obzirom na broj korištenih dretvi

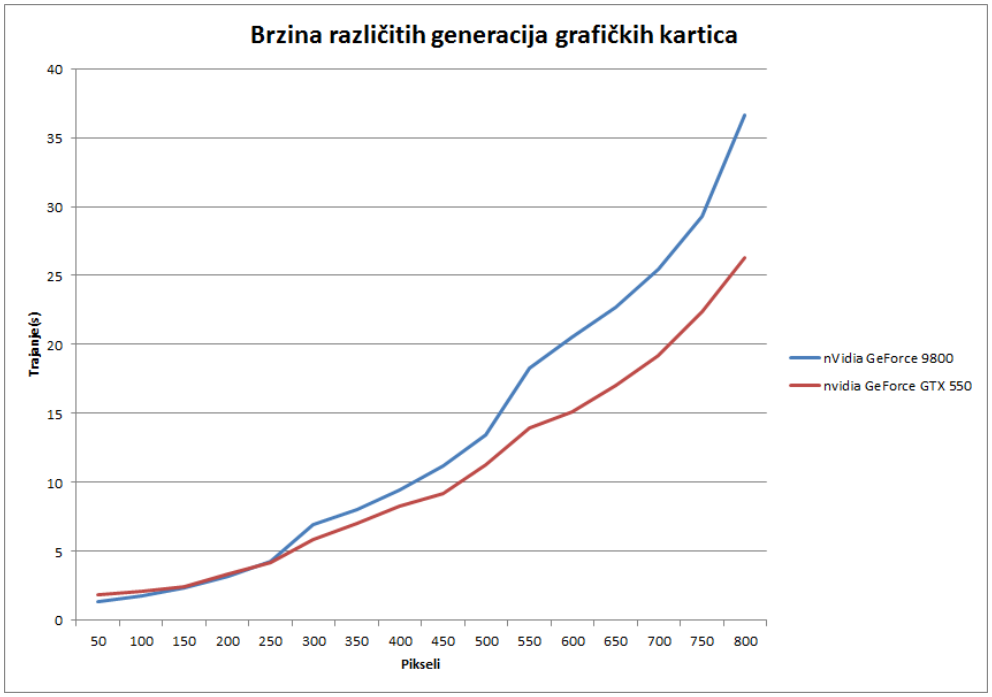
pretjerano bolje ako se uzima relativna mjera s obzirom na serijsku izvedbu. Izvedba algoritma za grafičku karticu trebala bi biti skalabilna. Načelno to i je jer postiže nešto bolje rezultate. Razlog zašto rezultati nisu još bolji je činjenica da je za dizajniranje algoritma korišteno načelo jednostavnosti i modularnosti. Zbog toga se izgubilo na skalabilnosti. Iako u praksi to znači da novije generacije grafičkih kartica neće jako poboljšati vremena izvođenja, izvedba u postojeći sustav biti će lakša. U budućem radu nastojati će se optimizirati postojeći programski kôd čime će se postići bolja skalabilnost.



Slika 6.6: Rezultati trajanja izvođenja u ovisnosti generaciji grafičkih kartica

Na slici 6.7 prikazana je ovisnost vremena o veličini slike za pojedine generacije grafičkih kartica za studiju mr-breast-dm. Vidljivo je da novija generacija grafičkih

kartica postiže nešto bolje rezultate, što je i očekivano. Na malim slikama nema velike razlike u vremenima izvođenja, ali u praksi ta činjenica nije važna jer se takve slike veoma rijetko komprimiraju.



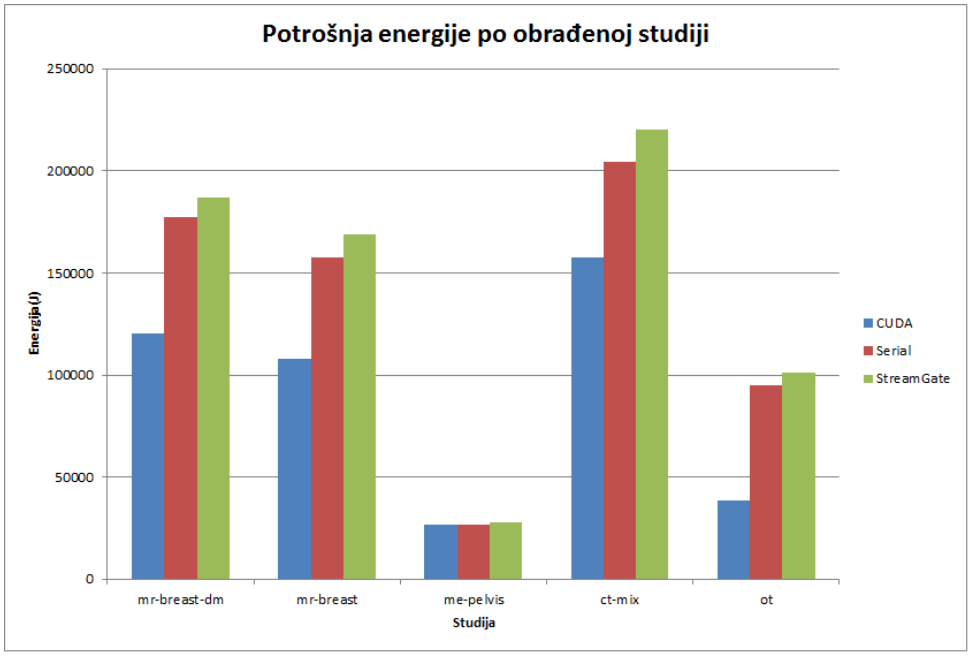
Slika 6.7: Rezultati trajanja izvođenja u ovisnosti generaciji grafičkih kartica i veličini slike

6.3. Potrošnja energije

U današnje vrijeme nastoji se sve više smanjiti potrošnju električne energije od strane različitih računalnih sustava. Kako se za paralelne izvedbe algoritama koristi više računalnih resursa, postavlja se logično pitanje, na koji način se to odražava na razinu potrošnje električne energije tijekom postupka komprimiranja. Naravno da ako pojedina izvedba troši dvostruko više električne energije od neke druge izvedbe, mada bila bolja po brzini izvođenja, upitna je njena iskoristivost u realnim informacijskim sustavima. Upravo zbog takvih razloga mjerena je i potrošnja pojedinih izvedbi koja je dobivena tijekom komprimiranja medicinskih studija.

Na slici 6.8 prikazana je potrošnja električne energije koja je dobivena tijekom izvođenja programa nad ispitnim studijama. Iz rezultata je moguće zaključiti da za manje studije nema velike razlike između potrošnje među različitim izvedbama. No, za veće studije pokazuje se da izvedba na grafičkim procesorima postiže puno manju potrošnju od ostale dvije izvedbe. Iako grafičke kartice troše više snage od procesora,

i samim time povisuju potrošnju, zbog značajno manjeg vremena izvođenja, konačan utrošak energije je manji. S druge strane je vidljivo da je potrošnja *StreamGate* izvedbe nešto veća od serijske izvedbe. Razlog tome jest da su prilikom izvođenja *StreamGate* iskorištene sve procesorske jezgre, dok je kod serijske iskorištena samo jedna, pa je zbog toga i potrošnja veća. Kako vrijeme izvođenja *StreamGate* izvedbe nije puno manje od serijske izvedbe, veća prosječna potrošnja uzrokuje da ukupna potrošnja bude također nešto veća od serijske izvedbe.



Slika 6.8: Potrošena energija prilikom komprimiranja pojedine studije

Za mjerenje potrošnje korišten je utični mjerač energije tip UP-M1. Naravno da se na ovaj način potrošnja ne može odrediti s apsolutnom točnošću i da će postojati određene greške prilikom mjerenja, ali predstavlja dovoljno dobru metodu mjerenja uz pomoć koje se može odrediti okvirna potrošnja pojedinih izvedbi algoritama. Iz prikazanih rezultata može se zaključiti kako korištenjem grafičkih procesora možemo, osim vremena izvođenja, smanjiti i ukupnu utrošenu energiju za kompresiju podataka u odnosu na serijsku izvedbu algoritma koja ne koristi grafičku karticu.

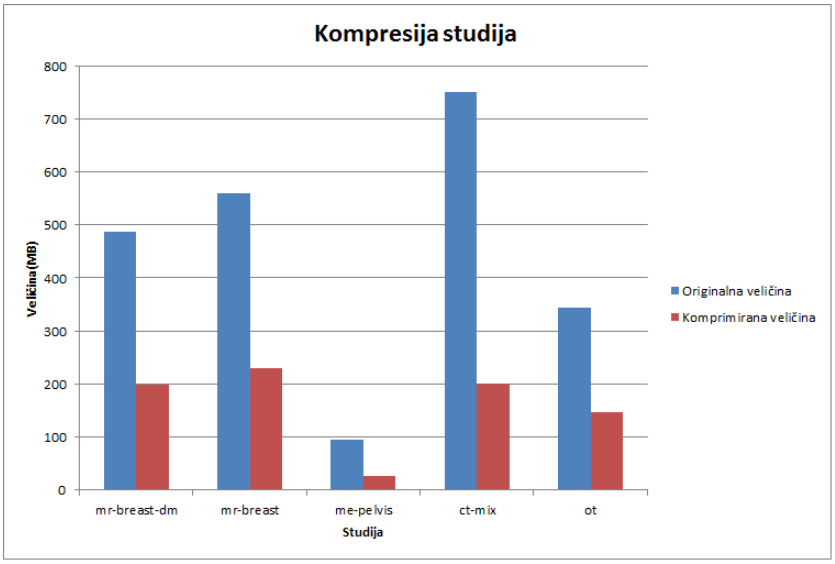
6.4. Kompresija

Kako se ipak radi o algoritmu za komprimiranje podataka, potrebno je ocijeniti koji je stupanj kompresije dobiven upotrebom ovog algoritma. U tablici 6.4 su prikazani

Tablica 6.4: Rezultati kompresije nad ispitnim studijama

Naziv studije	Veličina (MB)	Komp. veličina (MB)	Stupanj kompresije
mr-breast-dm	486.43	198.00	2.4567
mr-breast	559.53	229.47	2.4384
mr-pelvis	95.76	27.47	3.4860
ct-mix	750.79	201.54	3.7253
ot	343.27	147.86	2.3216

rezultati dobiveni kompresijom svake od testnih studija. Može se vidjeti kako je potrebni prostor za spremanje komprimiranih studija redovito više nego dvostruko manji. Naravno kompresija će uvelike ovisiti i o svojstvima medicinskih slika. Primjerice, kompresija će biti puno bolja ako na slikama postoje veća područja crne boje. Upravo takvo svojstvo nerijetko i imaju medicinske slike, što svakako pogoduje tome da se za njih postiže bolja kompresija.



Slika 6.9: Rezultati kompresije nad pojedinim studijama

Iz prikazanih rezultata na slici **??** vidljivo je da se komprimiranjem medicisnkih slikovnih podataka korištenjem ovog algoritma može uštedjeti više od polovice njihove veličine. No nije to jedina prednost uporabe komprimiranih podataka u bolničkim informacijskim sustavima. U takvim sustavima se veoma često javlja potreba za dohvaćanjem podataka iz centralnog sustava. Među te podatke često spadaju i velike količine slikovnih medicinskih podataka. Logično je zaključiti da se komprimiranjem podataka može postići njihov puno brži dohvat i tako smanjiti i opterećenost mreže koja se za to koristi.

6.5. Propusnost

Problem koji javlja zbog prijenosa velike količine podataka preko mreže direktno je rješen kompresijom podataka. Rezultati za propusnost direktno slijede rezultate za kompresiju. Može se napraviti analiza prema tablici 6.4. Pretpostavlja se da se unutar bolničkog sustava koristi lokalna mreža. Propusnost koja mreža nudi obično je 10 *Mbps*. To je samo teorijski maksimum jer zbog stanja u mreži može biti i puno manja. Tablica 6.5 prikazuje uštedu vremena za dohvaćanje različitih vrsta medicinskih slika (nazivi prema studijama). Neka se za svaku vrstu dohvaća 5 *GB* podataka odjednom (veličina originalnih slika). Rezultati su grubo izračunati, te je njihov cilj samo da prikažu da značajna razlika u vremenu prenošenja podataka postoji.

Tablica 6.5: Ušteda vremena korištenjem kompresije

Naziv studije	Dohvat studije (min)	Dohvat kompr. studije (min)	Razlika (min)
mr-breast-dm	68	27	41
mr-breast	68	28	40
mr-pelvis	68	20	48
ct-mix	68	18	50
ot	68	29	39

Isto tako, razmatramo li propusnost predložene izvedbe u odnosu na serijsku izvedbu algoritma, možemo za navedene testne studije prikazati rezultate u tablici 6.6. Prikazane su mogućnosti propusnosti svih izvedbi algoritma CBPC obzirom na veličinu ulaznih podataka koju je moguće obraditi u jedinici vremena. Očekivano, izvedba pomoću grafičkog procesora kao ubrzivača ostvaruje najveću propusnost u odnosu na ostale dvije izvedbe, što ju čini prihvatljivom za realne primjene u današnjim i budućim računalima koji će imati grafičke podsustave koji su kompatibilni sa nekim od programskih modela za operacije opće namjene kao što su CUDA ili OpenCl.

Tablica 6.6: Okvirna propusnost izvedbe u kB/s

Naziv studije	Veličina (MB)	CUDA	SG	Serial
mr-breast-dm	486,43	≈ 485	≈ 234	≈ 197
mr-breast	559,53	≈ 621	≈ 298	≈ 255
mr-pelvis	95,76	≈ 651	≈ 308	≈ 260
ct-mix	750,79	≈ 663	≈ 306	≈ 264
ot	343,27	≈ 640	≈ 305	≈ 260

7. Zaključak

U sklopu ovog rada izveden je algoritam za kompresiju slikovnih medicinskih podataka bez gubitaka, korištenjem grafičkih procesora. Ciljevi ove izvedbe su svakako postizanje što većeg stupnja kompresije, uz što manje trajanje izvođenja algoritma i potrošnju električne energije. Izrađena izvedba je uspoređena sa serijskom izvedbom algoritma, kao i s izvedbom pomoću programskog jezika *StreamIt*, koja navedeni algoritam paralelizira za izvođenje na višejezgrenom procesoru. Sve tri izvedbe su ispitane nad 5 reprezentativnih studija medicinskih slikovnih podataka te su uspoređeni dobiveni rezultati brzine izvođenja i potrošnje električne energije. Izrađena izvedba je pokazala bolje rezultate u vremenu izvođenja (otprilike 2.5 puta bolje od serijske izvedbe) i potrošnji električne energije (otprilike 1.5 puta manje potrošnja od serijske izvedbe) od ostale dvije izvedbe. Potrebno je također uzeti u obzir da je vrijeme izvedbe moguće još dodatno poboljšati posebnim prilagodbama nad algoritmom i dodatnim optimizacijama programskog koda. Dobiveni stupnjevi kompresije su pokazali kako se korištenjem komprimiranih podataka dobivaju značajne uštede u potrebnom podatkovnom prostoru za pohranu medicinskih slikovnih podataka, ali i propusnosti prilikom prijenosa tih podataka. Nadalje, brzina izvođenja ove izvedbe je ispitana na dvije generacije grafičkih kartica za jednu studiju te su uspoređena njihova međusobna vremena izvođenja. Pokazano je kako se sa korištenjem grafičke kartice novije generacije postiže nešto bolja vremena izvođenja, no njihova razlika nije odviše velika.

Cilj ove izvedbe algoritma za kompresiju medicinskih slikovnih elemenata bez gubitaka jest da se ona integrira u postojeće, ali i buduće moderne medicinske sustave za pohranu slikovnih podataka (*PACS*) i na taj način poboljšaju performanse tih sustava u smislu brzine prijenosa podataka, propusnosti i energetske učinkovitosti.. Kako je u medicinskim sustavima potrebno pohraniti velike količine podataka, ali također te podatke i razmjenjivati putem mreže, javlja se potreba za dobrim kompresijskim algoritmima.

Navedena izvedba sa svojim prethodno opisanim svojstvima i prednostima predstavlja sasvim dobar izbor za integraciju u buduće medicinske sustave koji će se razvi-

jati u Hrvatskoj. Dodatno, predloženi pristup korištenja grafičkih procesora za računski zahtjevne operacije može se primijeniti i u drugim aspektima medicinskih slikovnih aplikacija, kao što su pred-dijagnostička analiza slike, segmentacija objekata u računalom potpomognutoj dijagnostici i sl. Sve ove aplikacije danas je moguće učinkovito izvesti na paralelnim računalnim sustavima kao što su grafički procesori.

LITERATURA

[1] T. R. Soomro and M. Sarwar, "Green Computing: From Current to Future Trends," *World Academy of Science, Engineering and Technology*, 2012.

[2] J. Knezović, M. Kovač, and H. Mlinarić, "Classification and blending prediction for lossless image compression," in *Electronic Proceedings 13th IEEE Mediterranean Electrotechnical Conference*, (Malaga, Spain), 2006.

[3] J. Knezović, M. Kovač, I. Klapan, H. Mlinarić, v. Vranješ, J. Lukinović, and M. Rakić, "Application of novel lossless compression of medical images using prediction and contextual error modeling," *Collegium antropologicum*, vol. 31, no. 4, pp. 1143–1150, 2007.

[4] M. a. M. Knezović, Josip; Kovač, "Lossless Predictive Algorithm for Medical Image Compression," *Medical Image Understanding and Analysis Conference*, 2007.

[5] "General-purpose computation on graphics hardware," May 2012.

[6] Owens, D. John, Luebke, David, Govindaraju, Naga, Harris, Mark, Kruger, Jens, Lefohn, E. Aaron, Purcell, and J. Timothy, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, pp. 80–113, March.

[7] "Cuda homepage."

[8] "Opencl homepage."

[9] K. Ivica, "COMPUTER ASSISTED SURGERY AND TELESURGERY IN OTORHINOLARYNGOLOGY,"

[10] P. K. V. A. R. R. S. V. S. D. B. J. Klapan I, Simicić Lj, "Realtime transfer of live video images in parallel with three-dimensional modelling of the surgical field in computer-assisted telesurgery,"

[11] H. Huang, “Short history of PACS. Part I: USA,” *European Journal of Radiology*, 2011.

[12] “Dicom homepage,” Travanj 2013.

[13] “Nifti homepage,” Travanj 2013.

[14] *Nacionalna strategija razvoja zdravstva*. 2011.

[15] E. R. F.H. Barneveld Binkhuysen, “Teleradiology: Evolution and concepts,” *European Journal of Radiology*, 2010.

[16] H. P. Peeter Ross, Ruth Sepper, “Cross-border teleradiology—Experience from two international teleradiology projects,” *European Journal of Radiology*, 2009.

[17] S. Murugesan, “Harnessing Green IT: Principles and Practices,” *IT Professional*, 2008.

[18] N. A. Yan Zhang, “Green Data Centers,” *Handbook of Green Information and Communication Systems*, 2013.

[19] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen, “GreenCloud: a new architecture for green data center,” *ICAC-INDST ’09 Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, 2009.

[20] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, “Gpus and the future of parallel computing,” *IEEE Micro*, vol. 31, pp. 7–17, Sept. 2011.

[21] “Amp homepage,” Travanj 2013.

[22] M. Ciznickia, M. Kierzynkaa, P. Koptaa, K. Kurowskia, and P. Gepnerb, “Benchmarking data and compute intensive applications on modern cpu and gpu architectures,”

[23] J. T. Pedersen, “Parallel fractal compression for medical imaging,”

[24] N. N. Oliver Kuttera, Ramtin Shamsb, “Visualization and gpu-accelerated simulation of medical ultrasound from ct images,”

[25] X. Wu, N. Memon, K. Sayood, “A Context–Based, Adaptive, Lossless/nearly–lossless Coding Scheme for Continuous–Tone Images.” ISO Working Document ISO/IEC/ SC29/WG1/N256, 1995.

[26] X. Wu, “Lossless Compression of Continuous-tone Images via Context Selection, Quantization, and Modeling,” *IEEE Transactions On Image Processing*, vol. 6(5), pp. 656–664, 1997.

[27] M.J. Weinberger, G. Seroussi, G. Sapiro, “The LOCO–I Lossless Image Compression Algorithm: Principles and Standardization into JPEG–LS,” *HP Laboratories Technical Report*, vol. HPL-98-198, 1998.

[28] T. Seemann and P. Tischer, “Genralized locally adaptive DPCM,” *Proc. IEEE Data Compression Conference (DCC’97)*, pp. 473–488, March 1997.

[29] B. Gaster, L. Howes, D. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL*. Morgan Kaufmann, Elsevier Science & Technology Books, 2012.

[30] “Cuda c programming guide,” Travanj 2013.

[31] W. Thies, *Language and Compiler Support for Stream Programs*. Phd thesis, Massachusetts Institute Of Technology, Cambrige, MA, 2009.

[32] J. Knezović, M. Žagar, and H. Mlinarić, “Software system for efficient lossless compression of medical images,” *Collegium antropologicum, prihvaćen za objavljivanje*.

Izvedba algoritma za kompresiju medicinskih slikovnih podataka korištenjem grafičkog procesora s ciljem povećanja propusnosti i smanjenja utrošene energije

Sažetak

U današnje vrijeme složenih informacijskih sustava postoji potreba za paralelnim programima koji se izvode na višeprocesorskim računalnim sustavima. Veoma popularan način ubrzavanja algoritama ostvaruje se pomoću grafičkih procesora (GPGPU, engl. General Purpose Computing on Graphics Processing Unit). Prednost korištenja grafičkih procesora je u tome što se često nalaze u računalnom sustavu kao osnovna komponenta. U današnjim bolničkim informacijskim sustavima postoji velik broj različitih dijagnostičkih uređaja koji generiraju iznimne količine slikovnih podataka (MR, CT, RTG). Dobiveni podaci pohranjuju se u elektroničkom obliku u podatkovne centre u kojima je potrebno optimirati kapacitet i potrošnju električne energije. Dodatno, dijagnostički podaci se razmjenjuju elektroničkim putem između klijentskih dijagnostičkih uređaja i centralnog sustava. Zbog toga je potrebna kompresija kako bi se smanjili zahtjevi za podatkovnim prostorom te ubrzao prijenos podataka.

U ovom radu opisana je izvedba algoritma za kompresiju medicinskih slika prilagođena za izvođenje na grafičkim procesorima. Ciljevi izvedbe nisu samo smanjenje zahtjeva u pohrani medicinskih slikovnih podataka i poboljšanje vremena izvođenja nego i potrošnje električne energije te mogućnost primjene u postojećim i budućim bolničkim informacijskim sustavima. Algoritam za kompresiju zasniva se na računski zahtjevnom modelu predviđanja slikovnih elemenata te kontekstualnom kodiranju dobivene greške predviđanja. Navedeni model odlikuje se iznimnom količinom podatkovnog paralelizma pa je zbog toga ovaj korak izveden na grafičkom procesoru računalnog sustava. Izvedba je uspoređena sa serijskom izvedbom istog algoritma i s izvedbom koja je prilagođena za izvođenje na višejezgrenim procesorima korištenjem tokovnog programskog jezika. Napravljena je usporedba vremena izvođenja algoritma i potrošnje električne energije. Pomoću predložene izvedbe postignuto je ubrzanje te ušteda u utrošenoj energiji što je eksperimentalno utvrđeno na studiji medicinskih dijagnostičkih slika. **Ključne riječi:** kompresija medicinskih slikovnih podataka,

metode kompresije bez gubitaka, paralelizacija računskh zahtjevnih aplikacija, programski model za programiranje grafičkih procesora, "zeleno" računarstvo, tokovni programski model

Implementation of Medical Visual Data Compression Algorithm on GPU for Increased Throughput and Reduced Energy Consumptionl

Abstract

In today's complex information systems there is a growing need for parallel programs for multiprocessor environments. A very popular way to improve the execution time of algorithms is to use graphics processing units (GPGPU). GPUs are common components in computer systems, which is a great advantage for their usage in parallelisation. There are many different diagnostical devices which are used in today's hospital information systems (MR, CT, RTG). The data is stored in electronical format in data centers where there is a need for optimizing the storage costs, throughput and consumption of electrical energy. Additionally, the diagnostic data is transmitted through network between client diagnostic devices and the central system. Because of these reasons there is a need for compressing the data to lower the needs of data storage and data transmission.

In this paper an algorithm for compressing medical images on GPUs is described. The objectives of this implementation are not only to reduce the demands of storing medical pictures and to improve the execution time, but to reduce the consumption of electrical energy and to provide a possibility for integration with existing and future hospital information systems. The algorithm for compression is based on a computationally complex prediction model of pixels and a contextual coding of the given prediction errors. The above model excels in great amount of data parallelism because of which this step is then implemented on GPUs. The above implementation is compared with a serial implementation of the same algorithm and with an implementation in a stream programming model which is adjusted for a multiprocessor environment. A comparative analysis between the above mentioned implementations based on execution time and electrical energy consumption has been made. Experimental results show that a decrease in execution speed and savings in consumption of electrical energy has been achieved through the use of the proposed implementation.

Keywords: Medical Visual Data Compression, Lossless Coding Parallelization, Data-Intensive Computing, Green Computing, General Purpose GPU Programming, Streaming Programming Model