SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

CUDA implementacija algoritma za širenje epidemija u mrežama

Matija Šošić

voditelj: Doc. dr. sc Mile Šikić

Zagreb, travanj 2011.

Sadržaj

Uvod

Do određenih spoznaja mnogo je lakše doći promatranjem nego samom teorijskom razradom.

Prirodno je problem iz stvarnog svijega predstaviti odgovarajućim modelom u računalu. Gotovo nikad nije moguće uzeti u obzir sve čimbenike, ali prepoznavanjem onih bitnih mogu se dobiti vjerni rezultati iz kojih slijede valjani zaključci. Što se promatrana pojava manje apstrahira veća je točnost simulacije, ali je zahtjevnija za izvođenje. Brojne istraživačke grane, poput astrofizike ili biomolekularne znanosti koriste ovakav pristup. Zato znanost danas sve više ovisi o mogućnosti obrade vrlo velike količine podataka.

Kao odgovarajuće tehničko rješenje često se nameće arhitektura koja podržava istovremeno izvođenje velikog broja neovisnih zadataka. U prvom redu zbog prirode realnih problema, a također i zbog mnogo veće dostupnosti odgovarajućih uređaja u posljednje vrijeme.

Problem koji ću u ovom seminaru obraditi jest odabir i implementacija algoritma koji simulira širenje zaraze u populacij predstavljenoj kompleksnom mrežom [1, 2, 3]. U prvom poglavlju detaljno je opisan problem i pripadajući model. Slijedi opis i pseudokod klasičnog algoritma koji se izvodi na procesoru. Treće poglavlje je uvod u principe rada paralelne arhitekture na kojoj je implementiran poboljšani algoritam, što je nužno za daljnje praćenje i razumijevanje predstavljenih rješenja. Glavni dio rada, poglavlje „Paralelni CUDA algoritam“, iznosi implementaciju rješenja i predviđanje rezultata. Naposljetku, šesto poglavlje uspoređuje rezultate testiranja. Analizaju se prednosti i mane pojedinog algoritma i objašnjavaju odstupanja od početnih očekivanja.

1 SIR model i definicija problema

SIR model [4]

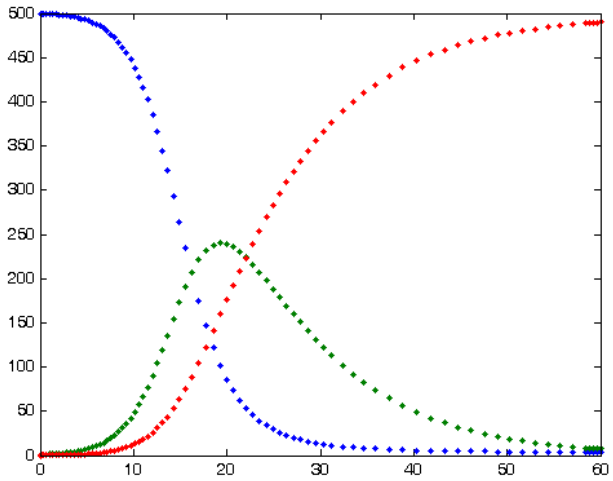
Pri opisivanju širenja zaraze u populaciji, različitost pripadnih jedinki mora biti svedena na nekoliko ključnih karakteristika. Primjerice, većina sezonskih prehlada ostavlja zaraženoj osobi trajni imunitet nakon prebolijevanja što čini smislenim podijeliti populaciju u tri nepreklapajuće skupine : podložni zarazi, trenutno zaraženi i oporavljeni, samim time i imuni.

Takav model pokazao se dobrim i često se koristi, a poznat je pod skraćenicom SIR prema nazivima navedenih skupina na engleskom jeziku : S – *susceptible*, I – *infectious* i R – *recovered*.

Napredak svake jedinke kroz faze bolesti je strogo određen što je prikazano na slici 1, iako jedinka ne mora nužno biti u posljednjem stanju po završetku simulacije. Oznake p i q nad prijelazima iz podložne u zaraženu i iz zaražene u oporavljenu skupinu su redom vjerojatnosti zaraze i oporavka jedinke.

Slika 1: Stanja i prijelazi SIR modela epidemije

SIR model je dinamičan, što je prikazano slikom 2. Svaka skupina, točnije broj njenih pripadnika, može se promatrati kao funkcija vremena pa se često koriste oznake S (t), I (t) i R (t). Tijekom epidemije, broj podložnih jedinki se ubrzano smanjuje na račun povećanja pripadnika ostale dvije skupine. Sve dok broj zaraženih I(t) ne poprimi vrijednost nula, simulacija se nastavlja.



Slika 2: Dinamika jednostavnog SIR modela

Postoje različite inačice SIR modela. Vjerojatnosti p i q također mogu biti funkcije vremena, kao i ukupni kardinalitet populacije. Primjer je rođenje novih jedinki čime se povećava brojnost skupine podložnih. Cilj rada je pokazati prednosti paralelne implementacije

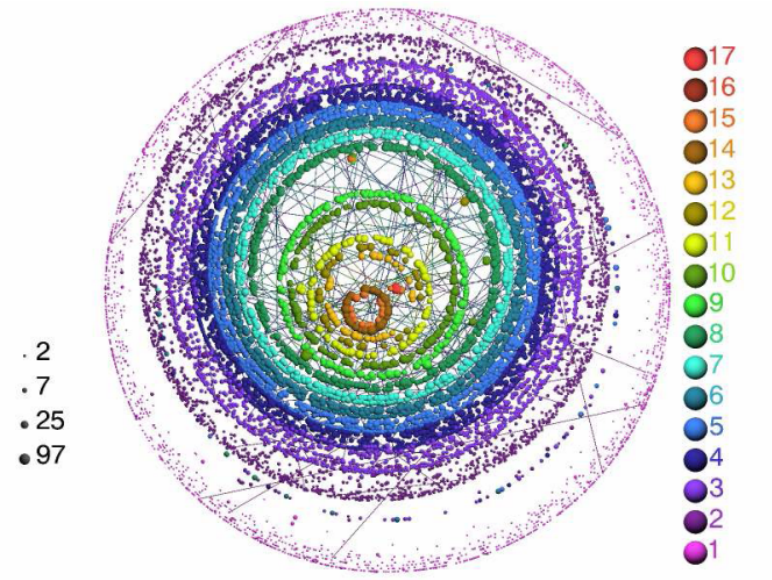
simulacije nad sekvencijalnom, a ne proučavanje konkretne zaraze stoga je odabran najjednostavniji SIR model.

Definicija problema

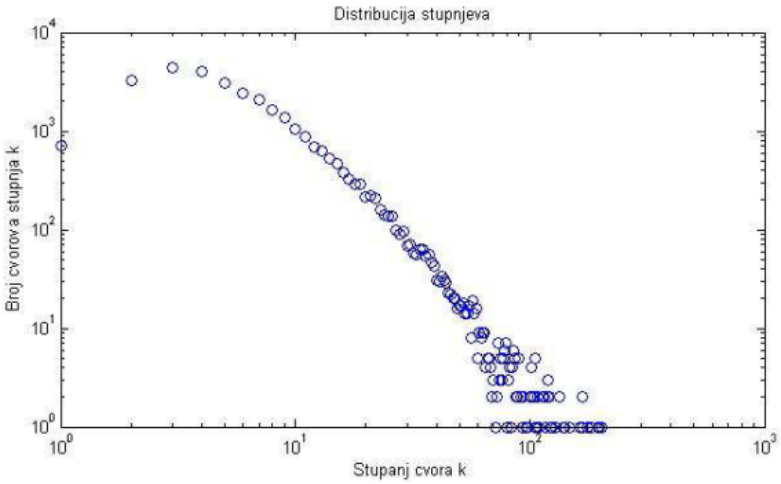
Modeli epidemije jako su ovisni o načinu na koji je populacija opisana. Dobar model koji se i ovdje koristi jest mreža. Mreža je bestežinski neusmjereni graf *G*, definiran skupom pripadajućih čvorova *V(G)* i bridova *E(G)*. Svaki čvor predstavlja jednu jedinku, a brid kontakt preko kojeg se zaraza može proširiti.

Početni uvjeti simulacije su mreža opisana grafom, čvor izvor zaraze i ranije navedeni parametri p i q. Oni se zadaju prije početka simulacije realnim vrijednostima između 0 i 1 uključivo s desna. Graf je također nepromjenjiv tijekom simulacije – broj čvorova, veza i raspored se ne mijenjaju, jedino se mijenja pripadnost jedinke određenoj skupini. Završetkom simulacije smatra se ulazak populacije u mirno stanje koje nastupa kada je I (t)= 0, skup zaraženih jedinki prazan. Odabir čvora koji će biti izvor zaraze može imati velik utjecaj na daljnje širenje, no radi jednostavnosti i lakše usporedbe rezultata u svakoj testnoj mreži odabire se uvijek isti čvor.

Primjer realne mreže na kojoj će se testirati implementacije je javno dostupna mreža kolaboracije znanstvenika „Condensed matter collaborations 2003“ [5]. Čvorovi predstavljaju znanstvenike koji su objavljivali radove na zajedničkoj lokaciji između 1995. i 1999. godine. Slike 3. i 4. jasno prikazuju jednu od najvažnijih karakteristika realnih mreža, distribuciju stupnjeva čvorova. Takva vrlo neravnomjerna distribucija imat će velik utjecaj na rezultate, što je detaljnije objašnjeno u petom poglavlju.



Slika 3: Prikaz mreže znanstvenika, svaki čvor ovisno o stupnju ima drugačiju boju i veličinu [6]



Slika 4: Distribucija stupnjeva čvorova mreže znanstvenika [6]

2 Osnovni algoritam – CPU implementacija

U ovom poglavlju predstavljeno je programsko rješenje problema koje se izvodi na procesoru računala. Za potrebe testiranja konkretno je implementirano u programskom jeziku C++, ali sam algoritam je vrlo općenit pa ga je bez većih izmjena moguće ostvariti u većini drugih viših programskih jezika. Rješenje koji slijedi sigurno je moguće dodatno optimizirati, ali za potrebe ovog rada pokazalo se dovoljno dobrim za usporedbu s implementacijom u četvrtom poglavlju.

Graf .el formata se učitava iz tekstualne datoteke i sprema u memoriju u obliku liste susjedstva (eng. *adjacency list*). Odabire se jedan čvor koji će biti izvor zaraze i njegov indeks se zapisuje u red (eng. *queue*) za obradu. U posebnom nizu prati se stanje u kojem se svaki čvor nalazi – podložan ili imun. Do informacije o pripadnosti čvora skupini oporavljenih dolazi se pomoću dva uvjeta – ako je čvor imun i nije u redu za obradu tada predstavlja oporavljenu jedinku.

Algoritam 1 CPU_zaraza (Graf *G*(*V;E*), Čvor izvor *S, p, q*)

```
1: Stvori listu susjedsta iz tekstualne datoteke s opisom grafa
2: Stvori niz Imuni veličine jednake broju čvorova u grafu
3: Stvori prazni red obrade Zarazeni
4: Postavi sve vrijednosti niza Imuni na 0
5: Imuni[S] = 1, stavi S u red Zarazeni
6: Dok red Zarazeni nije prazan radi
7:      uzmi cvor iz Zarazeni i spremi u cvor_trenutni
8:      Za sve susjede od cvor_trenutni radi
9:           Ako Imuni[susjed] == 0 i vjerojatnost(p) == 1
10:               stavi susjeda u red Zarazeni
11:               Imuni[susjed] = 1
12:          kraj Ako
13:     kraj Za
14:     Ako vjerojatnost(q) == 0        //pokusaj ozdraviti
15:          Stavi cvor_trenutni  u red Zarazeni
16: kraj Dok
```

Nakon što pokuša zaraziti sve svoje neposredne susjede, jedinka pokuša ozdraviti. Ukoliko ne uspije, opet ulazi u red za obradu.

Funkcija *vjerojatnost(parametar)* vraća vrijednost istina ili neistina što označava uspješnost zaraze ili ozdravljenja o ovisno ulazu. Generira pseudoslučajan broj koji se skalira na vrijednost između 0 i 1 i ukoliko je tako dobiveni broj strogo manji od ulaznog parametra funkcija vraća logičku vrijednost istine.

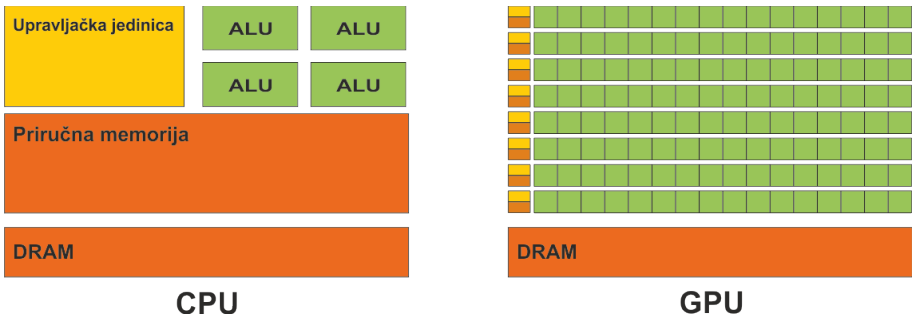
Očiti nedostatak ovakvog rješenja jest što sve zaražene jedinke moraju čekati dok se samo jedna obrađuje iako bi mogle neovisno o njoj širiti zarazu. Takvo ograničenje proizlazi iz arhitekture na kojoj se program izvodi i zato ćemo ubrzanje pokušati postići upravo primjenom arhitekture prikladnije ovakvom problemu.

3 Uvod u CUDA tehnologiju

Compute Unified Device Architecture – što je to?

CUDA [7] jest sklopovska arhitektura prilagođena za paralelno izvršavanje velikog broj neovisnih zadataka. Prati je odgovarajuće programsko sučelje vrlo slično višem programskom jeziku C. Predstavljena je 2007. godine od tvrtke Nvidia i od tada bilježi stalni porast broja korisnika, pogotovo u akademskim i znanstvenim krugovima. Koristi se za simulacije i proračune gdje može postići ubrzanje do čak 400 puta u odnosu na klasična rješenja. CUDA je prva omogućila korištenje grafičke kartice u općenite svrhe bez dodatnog prevođenja problema u kontekst prikazivanja slike. Do njezine pojave paralelno programiranje bilo je moguće samo na posebno izgrađenim i vrlo skupim uređajima, a danas se gotovo svako novije računalo s grafičkom karticom može koristiti u istu svrhu.

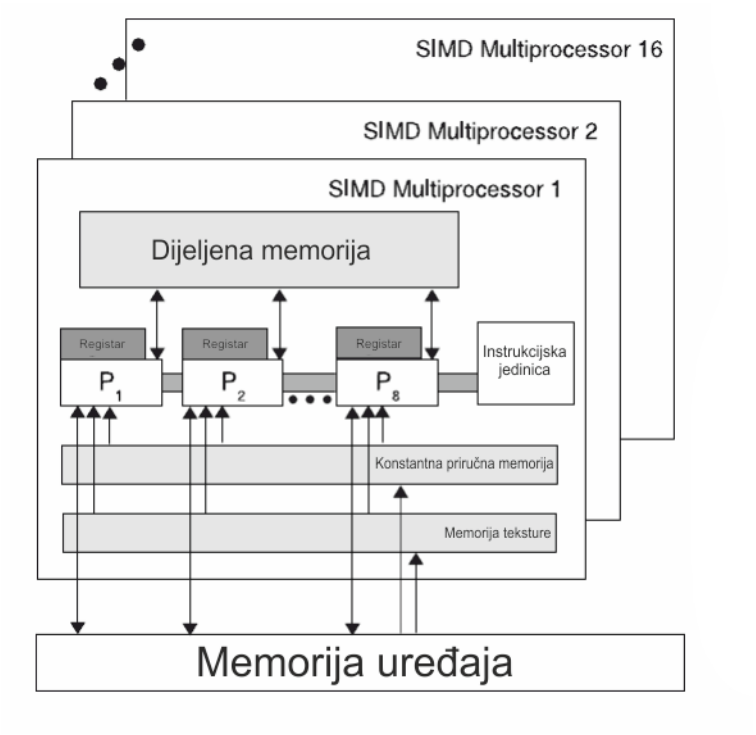
Osnovna razlika prema klasičnom procesoru koji je prilagođen vrlo brzom izvršavanju pojedine dretve jest sporije izvršavanje velikog broja dretvi u isto vrijeme. Zato je više tranzistora namijenjeno obradi podataka nego priručnoj memoriji i upravljačkoj jedinici, što shematski prikazuje slika 5.



Slika 5: Usporedba arhitekture procesora i CUDA grafičke kartice [7]

CUDA – sklopovski model

Za primjer ću uzeti grafičku karticu 8800 GTX. Apstraktno gledajući kartica i dalje slijedi tok obrađivanja slike u četiri koraka. Ipak, na sklopovskoj razini ne sastoji se od četiri zasebne jedinice. Svih 128 procesora su iste vrste i jednake brzine pristupa memoriji, što čini 8800 GTX velikim paralelnim procesorom. Kartica sadrži 16 multiprocesora, od kojih se svaki sastoji od 8 procesora. Svaki multiprocesor ima vlastitu dijeljenu memoriju koju svi pripadajući procesori dijele. Također sadrži skup 32 – bitnih registara te konstatnu i memoriju teksture (eng. *constant i memory cache*). Na svaki takt svih 8 procesora izvršava istu naredbu na drugačijem skupu podataka, zbog čega ih se naziva SIMD (eng. *Single instruction multiple data*) procesorima. Komunikacija među multiprocesorima može se odvijati samo preko globalne memorije uređaja koja je dostupna svima.



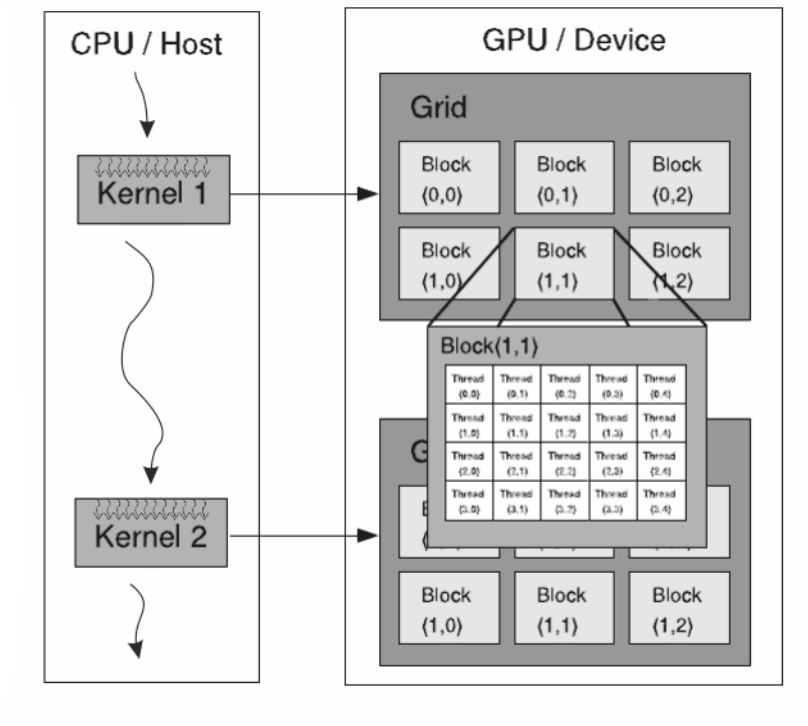
Slika 6: Sklopovski model CUDA uređaja [7]

CUDA – programski model

CUDA programsko sučelje omogućuje upotrebu opisane paralelne arhitekture za opću namjenu, a ostvareno je kao skup funkcija koje proširuju standardnu C sintaksu. Kompilator stvara izvršni kod za CUDA uređaj, a CPU ga prepoznaje kao ko-procesor. Ne postoje memorijska ograničenja kao u starijim GPGPU modelima, već je u potpunosti na raspolaganju programu. Jedino brzina pristupa ovisi o tipu memorije koji se koristi.

CUDA model moguće je shvatiti kao skup dretvi koje se paralelno izvršavaju. Skup dretvi koje rade istovremeno na jednom multiprocesoru naziva se *warp* i njegova veličina ovisi o uređaju. Programer odlučuje koliki broj dretvi želi pokrenuti na multiprocesoru i ukoliko je on veći od veličine *warpa* izvršavaju se serijski u skupinama. Skup dretvi kojem je u jednom trenutku dodijeljen neki multiprocesor naziva se *block*. Jedan multiprocesor može biti zadužen za više njih i tada se opet serijski izmjenjuju. Skup svih blokova stvorenih u jednom pozivu programa na uređaju zove se *grid* (slika 7). Svakoj dretvi i njezinom *blocku* dodijeljen je pripadajući ID kojemu svaka dretva može pristupiti tijekom rada i tako odrediti skup podataka na kojem radi. Skup instrukcija koje svaka dretva izvršava naziva se *kernel*. Različiti multiprocesori ne mogu izvršavati istodobno različite *kernele*, iako je to moguće simulirati uvjetnim naredbama.

Uz ovakav model, CUDA uređaj se može gledati kao paralelni SIMD procesor ograničen jedino veličinom memorije uređaja. 8800 GTX ima 768 MB memorije, a novije kartice i više što čini mogućim spremanje vrlo velikih grafova.



Slika 7: CUDA programski model [7]

Kao što prikazuje ilustracija 7, *grid* se logički sastoji od dvodimenzionalne skupine *blockova*, dok svaki *block* sadrži uređaj pripadajućih dretvi u najviše tri dimenzije. Pojedina dretva dobiva ID ovisno o položaju unutar *blocka*, a *block* u odnosu na *grid.* Korištenjem oba ID-a, dretva saznaje svoj globalni položaj i zna s kojim skupom podataka treba raditi.

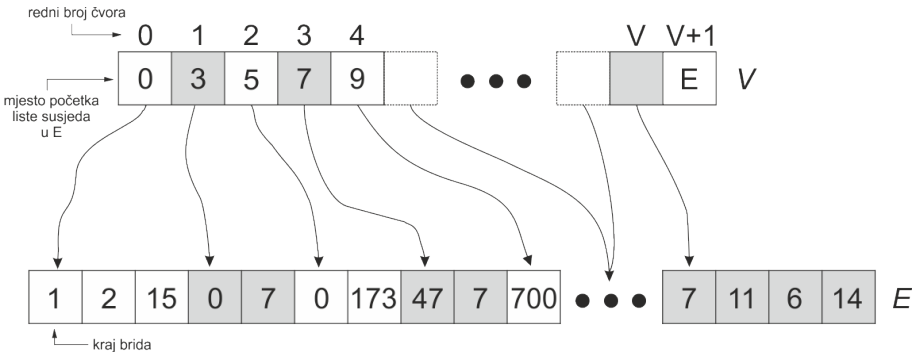
Tipični CUDA program sastoji se od više faza koje se izvršavaju ili na procesoru ili na grafičkoj kartici. Procesorski kod odrađuje sekvencijalni dio programa dok se dijelovi bogati paralelizmom prenose na CUDA uređaj pozivanjem *kernela.*

4 Ubrzani algoritam – CUDA implementacija

Reprezentacija grafa

Od kritične je važnosti na koji će način graf biti spremljen u memoriju grafičke kartice. Tradicionalna metoda poput matrice susjedstva osigurava konstantno vrijeme pristupa, *O(1),* pri određivanju postojanja veze između dva čvora ali zauzima više memorije nego što je potrebno. Lista susjedstva optimalno koristi memorijski prostor ali je provjera postojanja veze složenija. S obzirom na nezanemarivo memorijsko ograničenje za prikaz grafa koristi se lista susjedstva, ali u ponešto kompaktnijem obliku. Takav način zapisa grafa osmislili su Harish i Narayanan [8].

Umjesto posebnog niza za skup susjeda svakog čvora, svi nizovi susjeda spremljeni su u jedan niz. Čvorovi grafa predstavljeni su nizom *V*, gdje je za svaki čvor zapisano na kojem mjestu u nizu E počinje njegova lista susjeda. Niz *E* je time lista bridova u kojoj je svaki brid zapisan dva puta.



Slika 8: Reprezentacija grafa na CUDA uređaju [8]

Niz čvorova *V* sadrži jedan element za svaki čvor s koji pokazuje na listu susjeda u nizu bridova *E* i još jedan element koji pamti veličinu niza *E*. Pomoću njega moguće je svaki puta jednako obići susjede čvora bez obzira na njegov redni broj, od *V[redni broj]* do *V[redni broj+1]*, što bi inače bio problem za posljednji čvor u *V*.

CUDA algoritam

Osnovna ideja je pridijeliti svakom čvoru dretvu koja pokušava zaraziti sve susjede. Razlika u odnosu na CPU rješenje jest paralelizacija obrađivanja čvorova. Dok su prije uzimani iz reda obrade jedan po jedan, sada sve jedinke odjednom pokušavaju zaraziti svoje susjede. Ipak, i dalje svaki čvor prolazi serijski kroz listu susjeda pošto mu je dodijeljena samo jedna dretva. Algoritam koristi metodu globalne sinkronizacije, što znači pozivanje zasebnog *kernela* za svaki stupanj obrade svih čvorova. Globalna sinkronizacija također je nužna za provjeru uvjeta završetka simulacije.

Algoritam 2 CPU_zaraza_2 (Graf *G*(*V;E*), Čvor izvor *S, p, q, ima_zarazenih*)

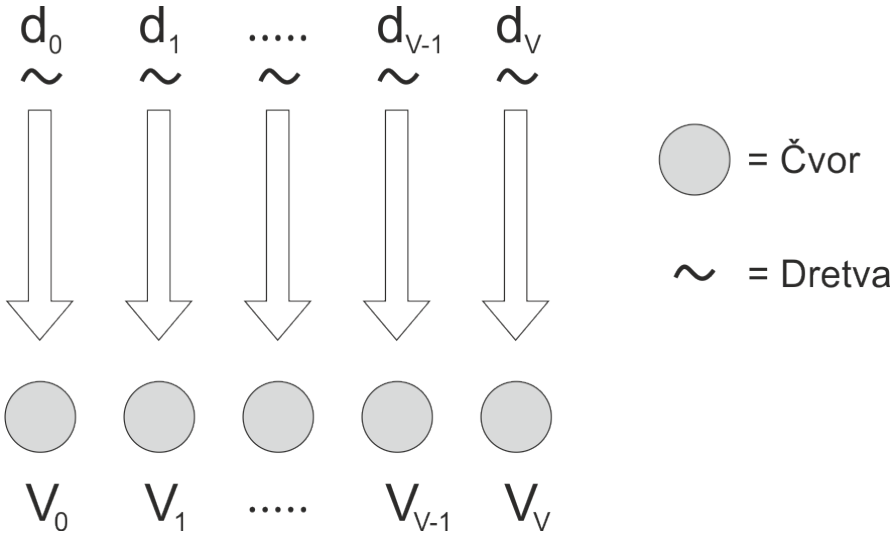
```
1: Stvori niz čvorova V i niz bridova E iz zapisa grafa G(V,E)
2: Stvori nizove Imuni i Zarazeni veličine jednake broju čvorova u grafu
3: Postavi sve vrijednosti niza Imuni i Zarazeni na 0
4: Imuni[S] = 1, Zarazeni[S] = 1
5: Stvori zastavicu ima_zarazenih
6: ima_zarazenih = 1
7: Dok ima_zarazenih radi
8:      ima_zarazenih = 0
9:      Za svaki čvor radi
10:             pozovi CUDA_1_kernel( V, E, Zarazeni, Imuni, p, q, &ima_zarazenih )
11: kraj Dok
```

Algoritam 2 izvodi se na procesoru sve do poziva potprograma CUDA_1_kernel kada se izvršavanje prenosi na CUDA uređaj. Po njegovom završetku opet postaje aktivan CPU dio programa koji provjerava treba li ponovno pozvati spomenuti potprogram. Time je ostvarena globalna sinkronizacija. Prije svakog stupnja obrade zastavica *ima_zarazenih* postavlja se na vrijednost 0.

Algoritam 3 CUDA_zaraza_1 (Graf *G*(*V;E*), Izvor *S, p, q, ima_zarazenih*)

```
1: cvor_index = izracunajDretvaID
2: Ako Zarazeni[cvor_index] == 1
3:      Za svakog susjeda radi
4:              Ako Imuni[susjed] == 0 i vjerojatnost(p) == 1
5:                      Imuni[susjed] == 1
6:                      Zarazeni[susjed] == 1
7:              kraj Ako
8:      kraj Za
9:      Ako vjerojatnost(q) == 1        //ako uspije ozdraviti
10:             Imuni[cvor_index] = 0
11:             Zarazeni[cvor_index] = 0
12:     Inače ima_zarazenih = 1
13: kraj Ako
```

Ako barem jedan od zaraženih čvorova ne uspije ozdraviti, *ima_zarazenih* poprimit će vrijednost jedan. Ovakav način provjere kraja puno je učinkovitiji od provjeravanja jesu li svi elementi niza *Zarazeni* jednaki nuli.



Slika 9: Vizualizacija pridjeljivanja jedne dretve jednom čvoru

Očekivanja rezultata

Za očekivati je da će grafovi čiji su čvorovi većeg stupnja bolje iskoristiti opisano paralelno rješenje. Najgori slučaj predstavlja graf oblika „lanca“ na kojemu je nemoguće ostvariti paralelnost ovakvim pristupom. U svakom stupnju zaraze može se zaraziti najviše jedna nova jedinka dok sve ostale moraju čekati da ih zarazi prethodeći susjed. Time se problem svodi na svoje sekvencijalno izdanje za što je procesor puno sposobniji.

Također su mogući neočekivani rezultati kad se testiranja vrše na relativno malim mrežama. Unatoč povoljnoj distribuciji stupnjeva čvorova CPU opet može postići kraće vrijeme izvršavanja. Uzrok tome je nezanemarivost uzastopnog pozivanja *kernela* u odnosu na posao koji treba obaviti. Ako je graf dovoljno velik i odgovarajuće distribucije stupnjeva, očekuju se bolje performanse CUDA algoritma nasuprot onome koji se izvršava na procesoru.

5 Rezultati

Ovo poglavlje predstavlja rezultate testiranja algoritama opisanih u drugom i četvrtom poglavlju. Oba algoritma testirana su na istom skupu podataka što omogućuje međusobnu usporedbu i donošenje zaključka.

Metoda testiranja

Aspekt rezultata testiranja koji se koristi za usporedbu CUDA i sekvencijalnog algoritma jest vrijeme izvršavanja programa. Skup podataka na kojima je testiranje provedeno sastoji se od realnih grafova sljedećih karakteristika:

	Broj čvorova	Broj bridova	Prosječni stupanj čvora
El. mreža SAD-a [9]	4941	6594	2.7
Cond-mat-el-2003	27519	116181	8.4
YouTube	1133999	2864514	5.1

Tablica 1. Mreže korištene u testiranju

Navedeni brojevi uz svaku mrežu zapravo opisuju njezinu najveću povezanu komponentu koja je prethodno izolirana iz početnog zapisa mreže.

Pri testiranju moguće je mijenjati tri parametra : vjerojatnosti zaraze i ozdravljenja p i q te graf. Kao što je već objašnjeno p i q zadaju se pozitivnim realnim brojevima manjim od jedan dok graf može biti jedan od tri navedena. Jedinstveni izbor ova tri parametra dalje u tekstu nazivat će se testna konfiguracija.

Važno je napomenuti da procesor i grafička kartica također izvršavaju i druge zadatke tijekom testiranja : GPU ispisuje sliku na zaslon, a CPU pokreće operacijski sustav. Zbog takvih „vanjskih utjecaja“, koji mogu smanjiti količinu dostupne globalne memorije CUDA na čak 80 %, moguće je dobiti različite rezultate prilikom testiranja u naizgled jednakim uvjetima. Zato je svaka testna konfiguracija pokrenuta više puta i iz dobivenih vremena izvršavanja na kraju izveden prosjek koji je proglašen konačnim rezultatom.

Uvjeti testiranja

Sva testiranja provedena su na osobnom računalu (PC) sa 8 GB radne memorije i Intel Core 2 Quad Q9550 @ 2.83 GHz procesorom na 64 – bitnom Windows 7 operacijskom sustavu. Svi testovi CUDA programa izvršeni su na MSI NVIDIA GeForce 570GTX grafičkoj kartici s 1280 MB memorije. CUDA algoritam napisan je u CUDA razvojnom okruženju, inačica 3.2 u integraciji s Visual Studio 2008, a sekvencijalni program u okruženju Dev – C ++.

q = 1, p = 1	Broj čvorova	Broj bridova	CPU vrijeme**(ms)**	CUDA 10X vrijeme**(ms)**	CUDA vrijeme**(ms)**
El. mreža SAD-a	5K	6.5K	**< 10**	**60**	**<10**
Cond-mat-el-2003	27.5K	116K	**40**	**40**	**<10**
YouTube	1.1M	2.8M	**1370**	**350**	**40**

Tablica 2.

q = 0.5, p = 0.5	Broj čvorova	Broj bridova	CPU vrijeme**(ms)**	CUDA 10X vrijeme**(ms)**	CUDA vrijeme**(ms)**
El. mreža SAD-a	5K	6.5K	**< 10**	**120**	**15**
Cond-mat-el-2003	27.5K	116K	**80**	**70**	**10**
YouTube	1.1M	2.8M	**2400**	**840**	**80**

Tablica 3.

q = 0.01, p = 1	Broj čvorova	Broj bridova	CPU vrijeme**(ms)**	CUDA 10X vrijeme**(ms)**	CUDA vrijeme**(ms)**
El. mreža SAD-a	5K	6.5K	**300**	**2010**	**200**
Cond-mat-el-2003	27.5K	116K	**4400**	**2860**	**290**
YouTube	1.1M	2.8M	**128 000**	**45400**	**4540**

Tablica 4.

	Učitavanje grafa **(ms)**	Postavljanje random strukture **(ms)**	Ukupni trošak **(s)**
El. mreža SAD-a	780	90	**0.87**
Cond-mat-el-2003	850	90	**0.94**
YouTube	2900	7670	**10.57**

Tablica 5. Vremenski trošak prije prve simulacije

Analiza rezultata

Tablice 2, 3, i 4 sadrže rezultate provedenih testova za tri različita odabira parametara p i q. U prvoj tablici zaraza i ozdravljenje su sigurni događaji, što je postignuto postavljanjem p i q na vrijednost jedan. Posljedica je da svaki čvor nakon što odmah iz prvog pokušaja zarazi sve susjede sigurno ozdravi i postane imun. Ovime se simulacija svodi na najjednostavniji slučaj, „bojanje“ ili „poplavu“ po čvorovima, a primjer korištenja je određivanje najveće povezane komponente nekog grafa. U prvoj, manjoj mreži procesor očekivano postiže bolji rezultat dok CUDA algoritam postiže tim veće ubrzanje što je povoljniji omjer „težine“ posla i uzastopnog pozivanja *kernela.*

Iduća konfiguracija, $p = 0.5$ i $q = 0.5$ predstavlja slučaj prosječne složenosti. Ipak, u najmanjoj mreži i dalje prevladava trošak pozivanja CUDA uređaja što je još više izraženo u odnosu na prethodni slučaj gdje svaki čvor ozdravi iz prvog pokušaja.

Naposljetku, odabir $p = 1$ i $q = 0.01$ osigurava najviše posla za obaviti. Svi čvorovi će biti zaraženi, ali vrlo mala vjerojatnost oporavka znatno odgađa kraj simulacije. Posebno je zanimljiv rezultat postignut na najvećoj mreži YouTube gdje CUDA algoritam završi za manje od pet sekundi nasuprot dvije minute izvršavanja programa na procesoru.

Vidljivo je da za svaku kombinaciju p i q ubrzanje raste s porastom broja čvorova u grafu, što je i sukladno očekivanjima. Jedno od najvećih ograničenja za postizanje boljih rezultata je nekorištenje dijeljene memorije u pojedinom multiprocesoru. Rješenjem tog problema ubrzanje bi sigurno bilo višestruko veće od ovdje postignutog.

Prije prvog pokretanja algoritma za simulaciju širenja zaraze potrebno je učitati graf iz datoteke u radnu memoriju i inicijalizirati polje slučajnih brojeva. Tablica 5. sadrži podatke za svaku mrežu.

Zaključak

CUDA arhitektura pokazala se kao prikladan izbor za implementaciju algoritma za širenje zaraze u kompleksnim mrežama. Čak i koje je jednostavno za implementirati postiže ubrzanje do 40 puta. Tako postaje moguće u kraćem vremenu pokrenuti veći broj testova i samim time brže doći do traženog zaključka.

Dosizanjem granica brzina klasičnih procesora, što zbog fizikalnih ograničenja samih materijala, što zbog neisplativosti ulaganja u daljnji razvoj, postalo je isplativije promijeniti cijeli sustav. Tu ideju vrlo uspješno utjelovila je upravo CUDA tehnologija, pogotovo jer je postalo moguće baviti se paralelnim programiranjem na običnom stolnom računalu opremljenom odgovarajućom karticom. Pretpostavlja da se trenutno u svijetu postoji više od sto milijuna tako opremljenih uređaja, što stvara veliko tržište i isplativost razvijanja CUDA pogonjenih aplikacija. Do danas se stvorila velika zajednica razvijatelja programa, a CUDA zbog svoje jednostavnosti postaje i sredstvo učenja paralelnog programiranja na pojedinim fakultetima.

Sve u svemu, CUDA se pokazala vrlo intuitivnim načinom primjene paralelnog programiranja što je čini pogodnom kako za edukativne, tako i komercijalne svrhe. Otkrivanje granica CUDA pogonjenog paralelnog programiranja tek slijedi, a na nama je da ih dosegnemo i pomaknemo još više.

Literatura

[1] R. Albert i A.-L. Barabási, Rev. Mod. Phys **74**, 47 (2002).

[2] M. E. J. Newman, SIAM Rev. **45**. 167 (2003).

[3] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes, Rev. Mod. Phys. **80**, 1275 (2008)

[4] W. O. Kermack i A. G. McKendrick, Proc. Roy. Soc. Lond. **A 115**, 700 (1927).

[5] M. E. J. Newman, Proc. Natl. Acad. Sci USA **98**, 404 (2001).

[6] Alen Lančić, Nino Antulov-Fantulin *: Širenje epidemija na kompleksnim mrežama - teorijski modeli i računalne simulacije* (2009).

[7] Nvidia: *Nvidia CUDA programming guide* (2010).

[8] Harish, P. i Narayanan, P. J.: *Accelerating large graph algorithms on the GPU using CUDA* (2007).

[9] D. J. Watts i S. H. Strogatz, Nature **393**, 440 (1998).

[10] David B.Kirk i Wen-mei W.Hwu: *Programming massively parallel processors* (2010).

[11] J. M. Kemp: *Parallel graph searching algorithms using CUDA*

Sažetak

Seminar opisuje implementaciju algoritma za širenje epidemije u mreži na CUDA arhitekturi i njegove prednosti prema klasičnom sekvencijalnom rješenju.

Definira se SIR model njegova inačica korištena u ovom radu. Predstavljana je osnovna implementacija algoritma na procesoru te su navedeni nedostaci takvog rješenja. Slijedi kratak uvod u način rada CUDA – e na programskom i sklopovskom modelu. Detaljno se razrađuje CUDA implementacija s naglaskom na razlike prema opisanom CPU algoritmu. Predstavljeni su rezultati dobiveni testiranjem na različitim realnim mrežama te se prepoznaju i analiziraju uzroci koji su do njih doveli.