Rješavanje problema u elektrostatici heterogenim metodama

Mijić, Nenad

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:949494

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-24**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Nenad Mijić

Rješavanje problema u elektrostatici heterogenim metodama

Diplomski rad

Zagreb, 2017.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA; SMJER NASTAVNIČKI

Nenad Mijić

Diplomski rad

Rješavanje problema u elektrostatici heterogenim metodama

Voditelj diplomskog rada: izv.prof.dr.sc. Davor Horvatić

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2017.

Prvenstveno neizmjerno se zahvaljujem svom mentoru, na ukazanom povjerenju i pruženoj prilici te na ogromnoj pomoći tijekom pisanja ovog diplomskog rada.
Zahvaljujem se i svojim roditeljima na podršci tijekom mog školovanja te svim prijateljima, kolegama i profesorima na svakom Vašem trenutku.

Sažetak

U ovom radu dan je kratak pregled osnova elektrostatike i računalnih heterogenih metoda. Razrađene su dvije platforme za rad na heterogenim sustavima, CUDA C i TensorFlow. Njih koristimo kako bi dobili ubrzanja izračuna u odnosu na homogene implementacije. CUDA C je detaljnije razrađen zbog kompleksnosti nužnih programskih paradigmi koje je potrebno usvojiti radi uspješne implementacije. Na kraju rada dan je pregled neuronskih mreža koje koristimo za izvrednjavanje potencijala na cijeloj domeni. Neuronske mreže trenirane su na diskretnom prostoru rješenja dobiveno metodom relaksacije.

Ključne riječi: elektrostatika, heterogene metode, CUDA C, TensorFlow, Keras, neuronske mreže

Solving problems in electrostatics with heterogeneous methods

Abstract

In this thesis a brief overview of the basis of electrostatics and computer heterogeneous methods is given. We discuss two platforms for implementing heterogeneous systems, CUDA C and TensorFlow, which are used to accelerate calculations in relation to homogeneous implementations. CUDA C is presented in more detail due to the complexity of the programming paradigms that need to be adopted for successful implementation. At the end of the thesis, we review the neural networks we use to compute potential across the entire domain. Neural networks are trained in the discreet space of the solution obtained by the relaxation method.

Keywords: electrostatic, heterogeneous computing, CUDA C, TensorFlow, Keras, neural networks

Sadržaj

1 Uvod **1**

2 Osnove elektrostatike **2**

2.1 Svojstva Laplaceove jednadžbe 7

2.1.1 Laplaceova jednadžba u jednoj dimenziji 7

2.1.2 Laplaceova jednadžba u dvije dimenzije 8

3 Računalne heterogene metode **9**

4 CUDA: platforma za heterogene sustave **12**

4.1 Struktura programa 12

4.2 Organizacija dretvi 14

4.3 Warp-ovi 17

4.4 Grananje 17

5 Analitičko i numeričko rješavanje Laplaceove jednadžbe **19**

5.1 Analitičko rješenje 19

5.2 Numeričko rješenje 22

5.2.1 Metoda konačnih razlika 22

5.2.2 Metoda relaksacije 26

6 TensorFlow **32**

7 Neuronske mreže **37**

7.1 Keras 39

8 Metodički dio **42**

8.1 Istraživački usmjerena nastava 42

8.2 Nastavna priprema: Napon i potencijal 43

9 Zaključak **50**

Literatura **51**

1 Uvod

Kompjutorske simulacije i izračuni postali su važna komponenta znanosti, a u fizici su se postavili kao značajan treći stup pored već tradicionalnih disciplina eksperimentalne i teorijske fizike. U čestičnoj i nuklearnoj fizici Monte Carlo simulacije su postale standardni alat za istraživanje jake nuklearne sile na kvantitativnoj razini, a slične pristupe imamo i u fizici kondezirane materije. Postoje još razni primjeri unutar geofizike, klimatskog modeliranja, bioinformatici i drugim područjima.

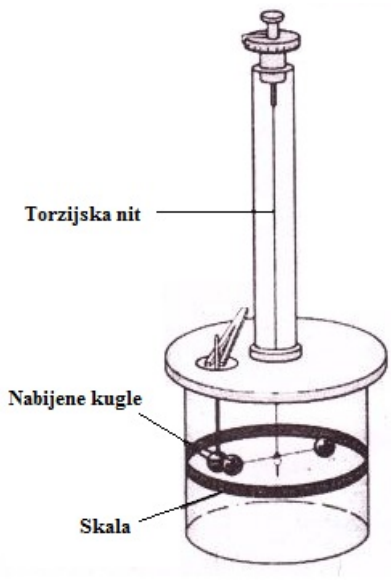
Navedene simulacije i izračuni su vrlo kompleksni te izvršavanje često zna potrajati i po nekoliko mjeseci. Zbog toga se pojavljuje interes, u znanosti i industriji vezanoj za znanost, za računalima s visokim performansama (high performance computing (HPC)). Po definiciji pojam se odnosi na korištenje više procesorskih jedinica ili računala kako bi se kompleksni programi izveli sa što većom efikasnošću (energetskom ili vremenskom). Pod HPC se ne smatra samo računalna arhitektura već uključuje hardverski sustav, programske alate i platforme te paradigme paralelnog izvršavanja programa. Tijekom zadnjeg desetljeća HPC se značajno razvio zbog pojave grafičko-procesorske heterogene arhitekture koja je dovela do nove paradigme u paralelnom izvršavanju programa.

U ovom diplomskom radu bit će izloženo rješavanje problema u elektrostatici heterogenim metodama. U drugom poglavlju dajemo kratki pregled osnova elektrostatike. U trećem uvodimo računalne heterogene metode. CUDA platformu obrađujemo u poglavlju četiri. Primjer rješavanja problema u elektrostatici dan je u poglavlju pet. U šestom poglavlju izložen je paket Tensorflow, a u sedmom poglavlju njegova upotreba preko biblioteke Keras za treniranje neuronskih mreža. Osmo poglavlje daje pregled metodičkog sata na kojem se uvodi pojam napona i potencijala.

2 Osnove elektrostatike

Elektrostatika je dio klasične elektrodinamike koja objašnjava fizikalne pojave vezane uz električne naboje u relativnom mirovanju. Osnovni zadatak teorije elektromagnetizma je izračunati silu koja djeluje na testni naboj koji se nalazi u električnom polju. Rješenje ovog problema moguće je naći pravilom superpozicije koje kaže da je međudjelovanje bilo koja dva naboja nezavisno od međudjelovanja s drugim nabojima. To znači da za izračun sile kojom nakupina od N naboja djeluje na testni naboj, prvo računamo silu između testnog naboja i proizvoljno odabranog prvog naboja zanemarujući za sada sve ostale naboje. Nakon toga računamo silu koju osjeća testni naboj od idućeg naboja, također zanemarujući sve ostale naboje, itd. Na kraju ukupna sila koju testni naboj osjeća je vektorski zbroj svih sila koje smo izračunali. Naizgled jednostavni zadatak u praktičnim problemima može postati izrazito kompleksan.

Silu kojom dva mirujuća naboja međudjeluju prvi je eksperimentalno izmjerio i postulirao Coulomb 1785. godine, mjereći silu između dva električki nabijena tijela pomoću torzijske vage. Unutar torzijske vage nalazile su se tri kuglice, od koje su dvije istih masa bile ovješene na torzijsku nit te su se mogle rotirati oko niti. Treća kuglica bila je statična. Pomoću kušalice dovodio je neku količinu naboja na statičnu kuglicu. Slobodna kuglica bi dotaknula statičnu i polovica naboja bi prešla na nju. Kada se sustav zaustavio na skali je očitao za koliko su se kuglice zarotirale te je iz te veličine izračunao silu kojom nabijene kuglice međudjeluju.

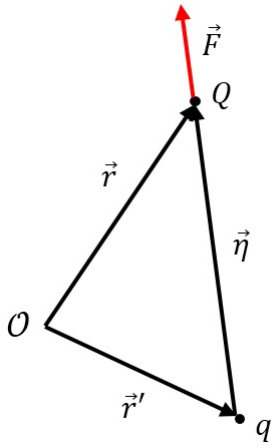


Slika 2.1: Postav Coulombovog eksperimenta.

Pomoću nenabijene kušalice, koja je bila iste veličine kao i statična kuglica, smanjio je naboj za pola iznosa na statičnoj kuglici te ponovo očitao za koliko su se kuglice zarotirale i izračunao silu. Došao je do zaključka da sila proporcionalno ovisi o količini naboja te obrnuto proporcionalno o kvadratu udaljenosti između naboja, te da je smjer sile na spojnici između naboja. Pokazao je da se dva istoimena naboja odbijaju, a raznoimena privlače. Danas međudjelovanje električnih naboja nosi ime upravo po Coulombu. Coulombov zakon glasi:

$$\vec{F}(\vec{r}) = \frac{1}{4\pi\varepsilon_0}\frac{q\,Q}{\eta^2}\vec{\eta}$$

Q je iznos probnog naboja, q je iznos naboja koji predstavlja izvor, a $\vec{\eta} = \vec{r} - \vec{r'}$ je vektor koji povezuje naboj q i Q. Vektor $\vec{r}$ povezuje ishodište i q, a vektor $\vec{r'}$ povezuje ishodište i Q. ε_0 je permitivnost vakuuma te iznosi $\varepsilon_0 = 8.85 \cdot 10^{-12}\,\mathrm{C^2N^{-1}m^{-2}}$.



Slika 2.2: Sila kojom naboj q djeluje na istoimeni naboj Q. Q je iznos probnog naboja, q je iznos naboja koji predstavlja izvor. Vektor $\vec{r}$ povezuje ishodište i Q, a vektor $\vec{r'}$ povezuje ishodište i q. $\vec{\eta} = \vec{r} - \vec{r'}$ je vektor koji povezuje naboj q i Q.

Kao što smo već naveli, ukupnu Coulombovu silu kojom naboji $q_1, q_2, \ldots, q_n$ djeluju na testni naboj Q možemo izračunati preko pravila superpozicije

$$\vec{F}(\vec{r}) = \vec{F}_1 + \vec{F}_2 + \ldots + \vec{F}_n = \frac{1}{4\pi\varepsilon_0}\left(\frac{q_1\,Q}{\eta_1^2}\hat{\eta}_1 + \frac{q_2\,Q}{\eta_2^2}\hat{\eta}_2 + \ldots + \frac{q_n\,Q}{\eta_n^2}\hat{\eta}_n\right)$$

$$\vec{F}(\vec{r}) = \frac{Q}{4\pi\varepsilon_0}\left(\frac{q_1}{\eta_1^2}\hat{\eta}_1 + \frac{q_2}{\eta_2^2}\hat{\eta}_2 + \ldots + \frac{q_n}{\eta_n^2}\hat{q}_n\right) = \frac{Q}{4\pi\varepsilon_0}\sum_{i=1}^{n}\frac{q_i}{\eta_i^2}\hat{\eta}_i$$

ili drugačije zapisano

$$\vec{F} = Q\vec{E}$$

gdje je

$$\vec{E}(\vec{r}) \equiv \frac{1}{4\pi\varepsilon_0}\sum_{i=1}^{n}\frac{q_i}{\eta_i^2}\hat{\eta}_i$$

električno polje koje stvaraju naboji. Električno polje je funkcija samo vektora $\vec{r}$ odnosno vektora koji povezuje proizvoljno odabrano ishodište i točku u kojoj računamo iznos električnog polja, ali ne ovisi o iznosu testnog naboja Q.

Kada neka konfiguracija sadrži izrazito puno naboja (npr. čak će naboj od 1 C sadržavati otprilike 10^{18} elementarnih naboja) sumu u izrazu za električno polje možemo zamijeniti integralom, a iznos naboja, gustoćom naboja. Coulombov zakon u tom slučaju možemo zapisati kao

$$\vec{F} = \frac{Q}{4\pi\varepsilon_0}\int_{\mathcal{V}}\frac{\rho(\vec{r'})\,d\tau'}{\eta^2}\hat{\eta},$$

a električno polje

$$\vec{E} = \frac{1}{4\pi\varepsilon_0} \int \frac{\rho(\vec{r'})\, d\tau'}{\eta^2}\, \hat{\eta} \tag{2.1}$$

Iako se možda ovakve jednadžbe na prvu čine jednostavnim, integrali ovog tipa su uglavnom teški za izračunati.

Helmholtzov teorem kaže da ako rotacija nekog vektorskog polja iščezava

$$\vec{\nabla} \times \vec{A} = 0$$

onda i linijski integral takvog vektorskog polja po proizvoljnom putu između dvije točke prostora ne ovisi o trajektoriji nego samo o početnim i konačnim točkama. To znači da zatvoreni linijski integral mora biti jednak nuli.

$$\oint \vec{A} \cdot \vec{dl} = 0$$

nadalje, možemo konstruirati skalarnu funkciju f čiji gradijent daje upravo početno vektorsko polje A

$$\vec{A} = \vec{\nabla} f.$$

Uvjet elektrostatike kaže da rotacija električnog polja iščezava u svim točkama prostora

$$\vec{\nabla} \times \vec{E} = \vec{0}.$$

To znači da su komponente takvog vektorskog polja međusobno zavisne

$$\frac{\partial E_x}{\partial y} = \frac{\partial E_y}{\partial x}, \quad \frac{\partial E_z}{\partial y} = \frac{\partial E_y}{\partial z}, \quad \frac{\partial E_x}{\partial z} = \frac{\partial E_z}{\partial x}.$$

Iz gore navedenoga imamo slobodu definirati električni potencijal

$$V(\vec{r}) \equiv -\int_{\mathcal{O}}^{\vec{r}} \vec{E} \cdot \vec{dl} \tag{2.2}$$

kao linijski integral električnog polja od proizvoljne točke $\mathcal{O}$ do točke u kojoj promatramo iznos potencijala $\vec{r}$. Odabir proizvoljne točke $\mathcal{O}$ mijenja iznos potencijala

$$V'(r) = -\int_{\mathcal{O}}^{r} \vec{E} \cdot \vec{dl} = -\int_{\mathcal{O}'}^{\mathcal{O}} \vec{E} \cdot \vec{dl} - \int_{\mathcal{O}}^{r} \vec{E} \cdot \vec{dl} = K + V(r)$$

za konstantu K u odnosu na potencijal iz (2.2), gdje je K iznos linijskog integrala od točke $\mathcal{O}'$ do $\mathcal{O}$. Bitno je da se razlika potencijala nije promijenila

$$V'(b) - V'(a) = V(b) - V(a).$$

To znači da će gradijent potencijala V' i V biti isti, odnosno da će električno polje koje izračunamo iz potencijala biti neovisno o izboru proizvoljne točke $\mathcal{O}$.

Također vrijedi

$$\vec{E} = -\vec{\nabla}V. \tag{2.3}$$

Minus u jednadžbi 2.3 stavljamo jer smo definirali da su pozitivni naboji izvor električnog polja. Obično je računanje potencijala jednostavnije nego računanje električnog polja (2.1). U takvim slučajevima imamo opciju da prvo izračunamo potencijal te nakon toga negativni gradijent potencijala kako bi dobili električno polje.

Kada se u prostoru nalazi više naboja primjenjujemo princip superpozicije

$$V(r) = \frac{1}{4\pi\varepsilon_0} \sum_{i=1}^{n} \frac{q_i}{\eta_i},$$

te u slučaju velikog broja naboja, sumu ponovo možemo zamijeniti integralom

$$V(r) = \frac{1}{4\pi\varepsilon_0} \int \frac{dq}{\eta},$$

$$V(r) = \frac{1}{4\pi\varepsilon_0} \int \frac{\rho(r')}{\eta} d\tau'. \tag{2.4}$$

Maxwellova prva jednadžba govori da su naboji izvori ili ponori električnog polja

$$\vec{\nabla} \cdot \vec{E} = -\frac{\rho}{\varepsilon_0}.$$

Korištenjem izraza (2.3)

$$\vec{\nabla} \cdot \vec{E} = \vec{\nabla} \cdot (-\vec{\nabla}V)$$

dobivamo Poissonovu jednadžbu

$$\nabla^2 V = -\frac{\rho}{\varepsilon_0} \tag{2.5}$$

Ako u prostoru koji promatramo nema naboja ($\rho = 0$) dobivamo Laplaceovu jednadžbu

$$\nabla^2 V = 0. \tag{2.6}$$

Uz zadane rubne uvjete rješenje ove parcijalne diferencijalne jednadžbe je dano jednadžbom (2.4).

2.1 Svojstva Laplaceove jednadžbe

2.1.1 Laplaceova jednadžba u jednoj dimenziji

Pogledajmo primjer jednodimenzionalne Laplaceove jednadžbe. Potencijal V ovisi o samo jednoj prostornoj varijabli x te Laplaceovu jednadžbu zapisujemo kao

$$\frac{\partial^2}{\partial x^2} V(x) = 0. \tag{2.7}$$

Rješenje takve jednadže je linearna funkcija

$$V(x) = k\, x + l.$$

Konstante k i l određujemo iz rubnih uvjeta zadanog problema.

Važna su slijedeća dva svojstva:

- $V(x)$ je srednja vrijednost od $V(x+a)$ i $V(x-a)$ odnosno

$$V(x) = \frac{1}{2}\left(V(x+a) + V(x-a)\right).$$

- druga derivacija $V(x)$, koja iznosi 0 za sve točke prostora gdje nema naboja, govori nam da je potencijal funkcija koja ne može imati lokalne ekstreme, odnosno nema lokalne minimume i maksimume. Ekstremalne vrijednosti mogu se nalaziti samo na rubovima prostora odnosno u rubnim točkama. Ovo svojstvo direktno proizlazi iz svojstva (1). U slučaju da nađemo lokalni ekstrem to bi značilo da potencijal u točki x tada nije srednja vrijednost susjednih točaka udaljenih za a.

2.1.2 Laplaceova jednadžba u dvije dimenzije

Ako potencijal ovisi o dvije prostorne varijable tada Laplaceova jednadžba u kartezijevom koordinatnom sustavu glasi

$$\frac{\partial^2 V(x,y)}{\partial x^2} + \frac{\partial^2 V(x,y)}{\partial y^2} = 0. \tag{2.8}$$

Ovakva jednadžba je parcijalna diferencijalna jednadžba drugog reda. Za nju vrijede slična pravila kao i za funkciju u jednoj dimenziji:

- potencijal V u točkama (x, y) je također srednja vrijednost svih točaka u okolini. Preciznije, ako izaberemo sve točke koje su za R udaljene od točke (x, y) tada je srednja vrijednost svih točaka na odabranoj kružnici (Γ) radijusa R potencijal u točki (x, y)
$$V(x,y) = \frac{1}{2R\pi} \oint_{\Gamma} V \; dl$$
- hesijan potencijala mora biti jednak nuli za sve točke prostora koje razmatramo. To također osigurava da je potencijal funkcija koja ne smije imati lokalne ekstreme, već se ekstremi nalaze samo na rubovima. Iz toga proizlazi da potencijal mora strogo biti glatka funkcija. Možemo zamisliti primjer iz stvarnog života tako da uzmemo željezni pravokutni okvir, čije su stranice različitih visina (to nam predstavlja rubne uvjete), te preko tog okvira napnemo gumenu membranu. Gumena membrana će se rastegnuti tako da minimizira svoju površinu bez ikakvih jama ili izbočina.

Od nastanka elektrostatike razvile su se različite metode rješavanjq problema. Rješenja su se dobivala korištenjem specijalnih funkcija, a najpoznatiji primjer je metoda multipolnog razvoja pomoću koje računamo električni potencijal kao sumu reda po multipolnim članovima. Prvi član predstavlja monopolni doprinos, drugi dipolni, treći kvadropolni itd. Iz tog pristupa možemo vidjeti da će svaka razdioba naboja, ako smo dovoljno daleko od izvora, moći u prvom redu biti aproksimiran točkastim izvorom. Razvijene su i različite numeričke metode, iako su računalno zahtjevne u većini slučajeva su jedine metode rješavanja važnih problema u industriji. Stoga se u ovom radu bavimo numeričkim metodama za rješavanje problema u elektrostatici na heterogenim sustavima koji omogućavaju izrazito ubrzanje izračuna.

3 Računalne heterogene metode

U početku razvoja računala su sadržavala samo centralne procesorske jedinice (CPU) dizajnirane za izvođenje općih programskih zadataka.U zadnjem desetljeću računala dobivaju dodatne procesorske elemente. Najčešći element je grafička procesorska jedinica (GPU) koja je bila dizajnirana za obavljanje specijaliziranih paralelnih grafičkih izračuna. Razvojem grafičkih procesorskih jedinica počinje njihova upotreba za matematički intenzivne izračune na velikim skupovima podataka zbog mogućnosti paralelnog izvršavanja računalnih zadataka.

Najčešće su diskretne CPU i GPU procesorske komponente povezane pomoću PCI-Express sabirnice unutar jednog računalnog čvora. U ovoj vrsti arhitekture GPU-ovi se nazivaju diskretnim uređajima. Prebacivanje s homogenih sustava u heterogene sustave predstavlja prekretnicu u povijesti računalstva visokih performansi. Homogeni sustavi koristi jedan ili više procesora iste arhitekture za izvršavanje programa. Heterogeni sustavi umjesto toga koriste više procesora različitih arhitektura za izvršavanje programa, dodjeljujući zadatke procesorima koji su prikladni za njihovo izvršavanje, što rezultira poboljšanjem performansi. Tako heterogeni sustavi ostvaruju bolje performanse. Iako heterogeni sustavi pružaju značajne prednosti u usporedbi s tradicionalnim računalnim sustavima visoke učinkovitosti, učinkovita uporaba takvih sustava trenutačno je ograničena kompleksnošću dizajna programa. Kada govorimo o heterogenim metodama mislimo na upotrebu heterogenih sustava za rješavanje numeričkih problema.

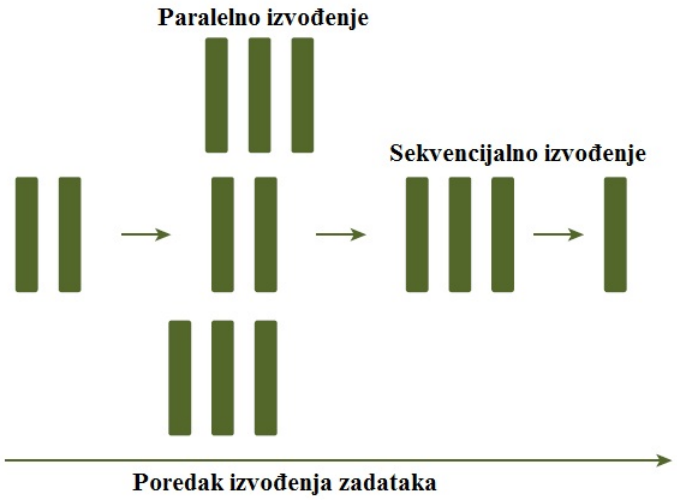
Danas se heterogeni sustavi sastoje od višejezgrenih CPU-a te nekoliko višejezgrenih GPU-ova. GPU trenutačno nije samostalni procesor nego služi kao koprocesor CPU-u. Stoga GPU-ovi moraju raditi zajedno s CPU-om. Zato za CPU koristimo izraz host, a GPU device.

Programi napisani za heterogene sustave sastoje se od dva dijela:

- host kod
- device kod

Host kod se izvršava na CPU-ovima, a device kod se pokreće na GPU-ovima. Program koji se izvršava na heterogenim sustavima inicijalizira CPU. CPU kod je odgovoran za upravljanje okruženjem, izvršavanje koda i prebacivanje podataka na GPU memoriju prije pokretanja device dijela koda.

Paralelno izvršavanje zadataka upravo dovodi do poboljšanja brzine izvođenja programa. Po definiciji paralelizacija je istovremeno izvođenje zadataka. Najčešće se neki složeniji problem podijeli u više manjih i jednostavnijih koji se zatim rješavaju istovremeno. Program se sastoji od dva najosnovnija dijela: instrukcija i podataka. Kada se računalni problem podijeli u manje dijelove, svaki dio zove se zadatak. U zadatku, pojedinačne instrukcije uzimaju ulazne podatke, primjenjuju funkciju na podatke te izbacuju rezultat. Ovisnost podataka nastaje kada neka instrukcija zahtjeva ulazne podatke koje su izlaz prijašnje instrukcije. Stoga možemo klasificirati zadatke kao ovisne, ukoliko zahtijevaju rezultate prijašnjih zadataka i neovisne koje ne ovise o drugim zadacima. Slika 3.1 prikazuje paralelno i sekvencijalno izvođenje zadataka i njihovu zavisnost.

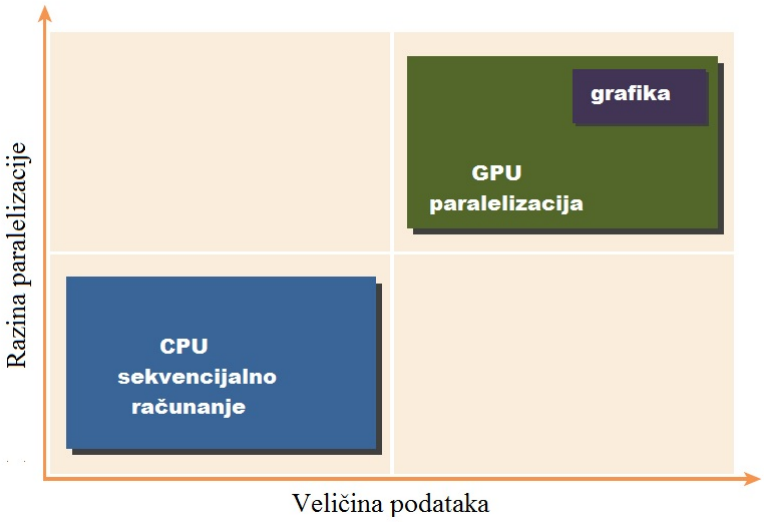


Slika 3.1: Prikazuje paralelno i sekvencijalno izvođenje zadataka te njihovu zavisnost. Preuzeto iz reference [2].

Računanje na GPU-u ne znači zamjenu računanja na CPU-u. Svaki pristup ima prednosti za određene vrste programa. CPU računanje je dobro za zadatke u kojima je velik broj različitih instrukcija dok je GPU računanje dobro za zadatke u kojima se lako može ostvariti paralelizacija obrade podataka. CPU je optimiziran za dinamička opterećenja karakterizirana kratkim sekvencama računalnih operacija i kada je nepredvidiv tijek izvođenja programa. GPU-ovi ciljaju na drugi kraj spektra, za programe u kojima dominiraju računalni zadatci s jednostavnom tijekom izvođenja. Kao što je prikazano na slici 3.2, postoje dvije dimenzije koje razlikuju opseg aplikacija za CPU i GPU:

- razina paralelizacije

- veličina podataka



Slika 3.2: Prikazuje dvije dimenzije. Preuzeto iz [2]

Ako problem ima malu veličinu podataka, sofisticiranu logiku i nisku razinu paralelizacije, CPU je tada bolji izbor zbog svoje sposobnosti da obrađuje složenu logiku. Ako problem umjesto toga obrađuje ogromnu količinu podataka i pokazuje masivnu paralelnost obrade podataka, GPU je pravi izbor jer ima veliki broj programabilnih jezgri. CPU + GPU heterogeni sustavi evoluirali su zbog toga što oba procesora imaju komplementarne atribute koji omogućuju programima da rade maksimalno učinkovito koristeći prednosti obje vrste procesora.

4 CUDA: platforma za heterogene sustave

CUDA je softverska platforma za paralelno programiranje opće namjene koju je razvila NVIDIA. Ona koristi NVIDIA GPU za rješavanje velikog broja kompleksnih numeričkih problema na učinkovit način. CUDA platforma je dizajnirana za rad s programskim jezicima kao što su C, C++ i FORTRAN. Ova pristupačnost olakšava korištenje GPU resursa, za razliku od prethodnih API-ja kao što su Direct3D i OpenGL, koji su zahtijevali napredne vještine u programiranju grafike.

CUDA C je proširenje standardnog ANSI C s nekoliko jezičnih ekstenzija kako bi se omogućilo heterogeno programiranje, ali i API za upravljanje grafičkim procesorima, njenom memorijom i ostalim zadatcima koje se izvršavaju na njoj. CUDA je također skalabilan programski model koji omogućuje programima transparentno skaliranje paralelizacije na GPU s različitim brojem jezgri.

NVIDIA CUDA `nvcc` kompajler odvaja device dio od host dijela koda tijekom kompajliranja. Host kod je standardni C kod i dalje je kompajliran s C kompajlerom. Device kod napisan je korištenjem CUDA C proširenog s ključnim riječima za označavanje funkcija koje se izvode na GPU, zvane kerneli. Device kod kompajlira se pomoću `nvcc` kompajlera.

4.1 Struktura programa

Programi napisani u CUDA C u većini će slučajeva imati slijedeću strukturu, odnosno imati će slijedećih 5 koraka:

- alokacija memorije na GPU
- prebacivanje podatka s računalne memorije na memoriju u grafičkoj kartici
- pokretanje kernela na GPU koji će obraditi podatke
- prebacivanje podataka s GPU memorije na računalnu memoriju
- oslobađanje memorije na GPU

Funkcija kojom alociramo memoriju na GPU-u je `cudaMalloc`

```
cudaMalloc ( void** devPtr, size_t size )
```

Ova funkcija alocira linearni raspon memorije sa zadanom veličinom izraženom u bytu. Alocirana memorija je vraćena u pokazivač `devPtr`.

Funkcija kojom možemo prebaciti podatke s računalne memorije u memoriju na grafičkoj kartici je `cudaMemcpy`

```
cudaMemcpy ( void* dst, const void* src, size_t count,
cudaMemcpyKind kind )
```

Ova funkcija kopira bytove s početne memorije, na koju pokazuje pokazivač `src`, na odredišnu, na koju pokazuje pokazivač `dst`. Varijabla `count` je veličina prebačene memorije također izražena u bytu. Smjer kopiranja određuje varijabla `kind` koja može biti slijedećeg tipa:

- `cudaMemcpyHostToHost` - prebacivanje iz računalne u računalnu memoriju
- `cudaMemcpyHostToDevice` - prebacivanje iz računalne u grafičku memoriju
- `cudaMemcpyDeviceToHost` - prebacivanje iz grafičke memorije u računalnu
- `cudaMemcpyDeviceToDevice` - prebacivanje iz grafičke memorije u grafičku

Ova funkcija je sinkrona što znači da će host kod blokirati daljnje izvršavanje instrukcija sve dok ova funkcija ne završi, odnosno dok se transfer ne izvrši.

Kernel je kao što smo već naveli funkcija koja će se izvršavati na GPU. Kernel definiramo s `__global__`, a ta funkcija može zaprimiti pokazivače na memoriju na grafičkoj kartici. Slijedeći primjer je kernel koji samo ispisuje poruku

```
__global__ void helloFromGPU ()
{
printf("Hello World from GPU!\n");
}
```

Kernel pokrećemo unutar `main` dijela programa naredbom

```
helloFromGPU<<<gridDim, blockDim>>>();
```

gdje su `gridDim` i `blockDim` varijable kojima organiziramo dretve, te ćemo ih u slijedećem potpoglavlju detaljno objasniti.

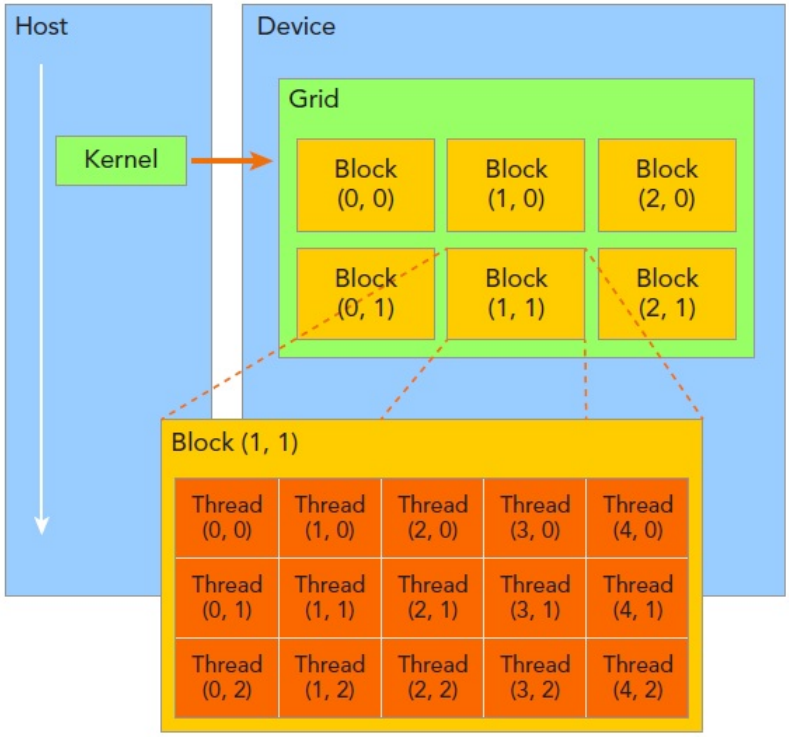
```
int main()
{
printf("Hello World from CPU!\n");
helloFromGPU <<<1, 4>>>();
}
```

Kada kompajliramo i pokrenemo program, poruka se ispisuje s GPU-a 4 puta jer smo pokrenuli kernel sa 4 dretve.

```
Hello World from CPU!
Hello World from GPU!
Hello World from GPU!
Hello World from GPU!
Hello World from GPU!
```

4.2 Organizacija dretvi

Kada se kernel pokrene izvršavanje se premješta na GPU na kojem se generiraju veliki broj dretvi i svaka dretva izvodi instrukcije koje su napisane u kernelu. Znanje o organiziranju dretvi je ključan dio CUDA programiranja. CUDA izlaže apstrakciju hijerarhije dretvi kako bi omogućila programerima organizaciju istih. Postoje dvije razine hijerarhije dretvi koje se dijeli na blokove i gridove dretvi.



Slika 4.1: Pokretanje kernela na hostu i dvodimenzionalna organizacija dretvi tog kernela. Preuzeto iz refence [2].

Sve dretve koje pokrene jedan kernel se nazivaju grid. Sve dretve unutar grida dijele zajedničku globalnu memoriju na samoj grafičkoj kartici. Takav grid je sastavljen od blokova dretvi koje čine grupe dretvi. Blok je zasebna jedinica zato što dretve iz različitih blokova ne mogu komunicirati. Dretve unutar istog bloka mogu komunicirati ili preko sinkronizacije svih dretvi unutar bloka ili preko zajedničke memorije bloka (block-local shared memory). Sinkronizacija dretvi znači da će aktivne dretve unutar bloka izvršiti sve instrukcije prije instrukcije sinkronizacije i čekati ostale dretve da dođu do iste instrukcije. Kada su sve dretve došle do instrukcije sinkronizacije tek tada dalje nastavljaju izvršavati instrukcije.

CUDA je organizirala gridove i blokove u 3 dimenzije. Slika 4.1 pokazuje dvodimenzionalnu organizaciju dretvi. Dimenzije gridova i blokova su određene pomoću dvije CUDA varijable:

- `blockDim` - broj dretvi u bloku
- `gridDim` - broj blokova u gridu

Ove varijable su tipa `dim3`, cjelobrojne varijable s 3 komponente. Svaka komponenta ovakve varijable može se dohvatiti preko x, y i z polja, na primjer

```
gridDim.x  blockDim.x
gridDim.y  blockDim.y
gridDim.z  blockDim.z
```

Dretve se oslanjaju na sljedeće dvije jedinstvene koordinate kako bi se razlikovale jedna od druge:

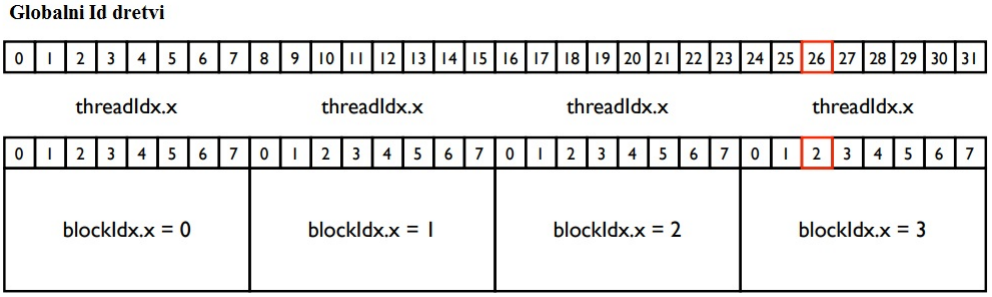
- `blockIdx` - indeks bloka unutar grida
- `threadIdx` - indeks dretve unutar bloka dretve

CUDA varijable unaprijed su inicijalizirane i njima se može pristupiti unutar kernela. Kada se izvršava kernel, koordinate varijable `blockIdx` i `threadIdx` su dodijeljene svakoj dretvi. Na temelju tih koordinata moguće je dodijeliti dijelove podataka na različite dretve, koje su također trodimenzionalne i svakoj komponenti se opet na sličan način može pristupiti unutar kernela.

```
blockIdx.x  threadIdx.x
blockIdx.y  threadIdx.y
blockIdx.z  threadIdx.z
```

Slika 4.2 prikazuje primjer dretvi u jednoj dimenziji gdje je `gridDim.x=4`, `blockDim.x=8`. Ovakvu konfiguraciju dretvi pokrećemo s naredbom `kernel<<<4,8>>>();` Kao što je prikazano `threadIdx.x` unutar svakog bloka se ponavlja, a da bi ostvarili jedinstvenu identifikaciju dretvi, unutar kernela možemo koristiti slijedeći kod

```
gloablId = blockIdx.x * blockDim.x + threadIdx.x
```



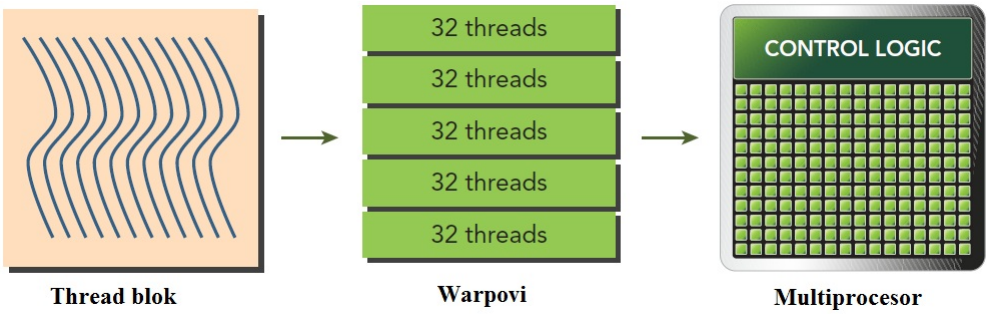
Slika 4.2: Organizacija dretvi u jednoj dimenziji te korištenje CUDA varijabli za identifikaciju pojedine dretve.

Slična je stvar s dretvama koje organiziramo u dvije dimenzije, a želimo recimo pristupiti svakom dijelu matrice `A`.

```
int col = blockIdx.x*blockDim.x+threadIdx.x;
int row = blockIdx.y*blockDim.y+threadIdx.y;
int index = col + row * N;
A[index] = ...
```

4.3 Warp-ovi

Kada se pokreće kernel razmišljamo o tome da se sve dretve aktiviraju u isto vrijeme te da obavljaju zadane naredbe, ali s hardverske strane to nije moguće što znači da se sve dretve ne aktiviraju u isto vrijeme. Oni se aktiviraju u grupama od 32 uzastopne dretve te se takva jedinica naziva warp. Kada se pokreće grid blokova oni se distribuiraju na multiprocesore od kojih su NVIDIA grafički procesori napravljeni. Jednom kada se blok dodijeli na jedan multiprocesor on se dalje dijeli u warp-ove.



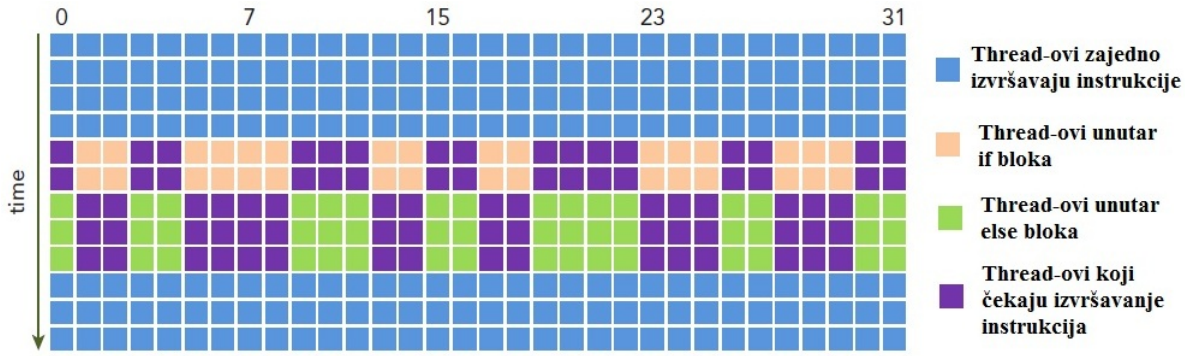
Slika 4.3: Veza između logičkog i hardverskog pogleda na dretve. Slika preuzeta iz reference [2].

Multiprocesor uvijek alocira diskretan broj warp-ova za dodijeljen blok. U warp ne mogu ući dretve iz različitih blokova. Ako veličina bloka nije cjelobrojni višekratnik broja warp-ova, u zadnjem warp-u će ostati dretve koje biti neaktivne, odnosno dretve koje neće izvoditi ikakve instrukcije, a pritom zauzimaju hardversko mjesto. To dovodi do sporije izvedbe čitavog programa.

4.4 Grananje

Grananje je jedan od osnovnih konstrukta u programskim jezicima visoke razine. CUDA podržava tradicionalan stil grananja kao u C jeziku. CPU ima ugrađen hardver

za predikciju grananja radi upravljanja kompleksnim grananjem. GPU su razmjerno jednostavni bez ugrađenog hardvera za kompleksno grananje, stoga od tuda i dolazi uvjet da dretve unutar istog warp-a moraju izvoditi iste instrukcije tijekom jednog ciklusa. To može dovesti do problema ako jedna dretva unutar warp-a krene na drugi put unutar grananja. Na primjer uzmemo jednostavan primjer if-else grananja. Pretpostavimo da polovica dretvi unutar warp-a zadovoljava uvjet, a druga polovica ne. Tada polovica warp-a prvo izvodi instrukcije unutar if bloka te nakon toga ostatak warp-a izvršava else blok. Dretve unutar warp-a koje izvode različite instrukcije naziva se warp divergencija. Warp divergencija uzrokuje značajan pad performansi računanja. Ako warp divergira tada on mora serijski izvršiti svako grananje. Slika 4.4 pokazuje divergenciju warp-a tijekom if-else grananja. Kako bi se ostvarile najbolje performanse preporuka je da se izbjegava grananje unutar warp-a.



Slika 4.4: Warp divergencija kod if-else grananja. Slika preuzeta iz reference [2].

5 Analitičko i numeričko rješavanje Laplaceove jednadžbe

Za primjer problema na koji ćemo primijeniti računalne heterogene metode odabrali smo dvije beskonačno dugačke uzemljene metalne ploče u $y = 0$ i $y = a$ povezane s dvije metalne ploče u $x = \pm b$ koje su na konstantnom potencijalu V_0. Između svake ploče je postavljen tanki izolator.

Zadatak je riješiti Laplaceovu jednadžbu u dvije dimenzije

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0,$$

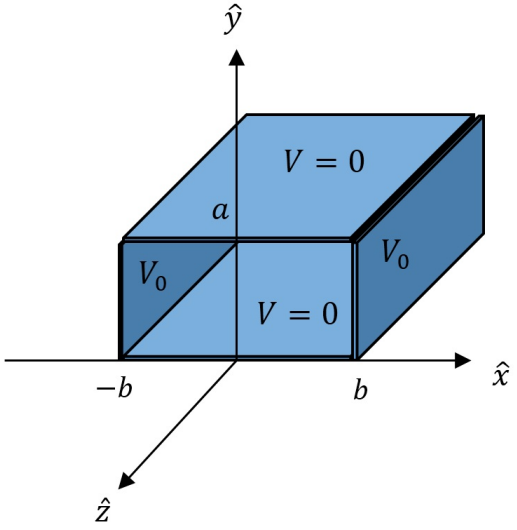
uz rubne uvjete

(i) $V(x, y = 0) = 0$

(ii) $V(x, y = a) = 0$

(iii) $V(x = b, y) = V_0$

(iv) $V(x = -b, y) = V_0$



Slika 5.1: Rubni uvjeti i prikaz geometrije problema.

5.1 Analitičko rješenje

Problem ćemo riješiti analitički, metodom separacije varijabli. Pretpostavimo rješenje u obliku produkta dviju funkcija X i Y koje svaka ovisi o samo jednoj varijabli $X(x)$,

$Y(y)$

$$V(x,y) = X(x) \cdot Y(y). \tag{5.1}$$

te kada izraz uvrstimo u jednadžbu (2.8) dobivamo slijedeći izraz

$$\frac{\partial^2}{\partial x^2}(X(x)Y(y) + \frac{\partial^2}{\partial y^2}(X(x)Y(y) = 0$$

$$Y(y)\frac{d^2X(x)}{dx^2} + X(x)\frac{d^2Y(y)}{dy^2} = 0.$$

Radi jednostavnosti ne pišemo ovisnost o varijablama

$$\frac{1}{X}\frac{d^2X}{dx^2} + \frac{1}{Y}\frac{d^2Y}{dy^2} = 0.$$

U ovoj jednadžbi prvi dio ovisi samo o x varijabli, a drugi samo o y. Odnosno imamo jednadžbu koju možemo zapisati u obliku

$$f(x) + g(y) = 0$$

Jedino rješenje su konstantne funkcije f i g. $f(x) = C_1$, a $g(y) = C_2$, uz uvjet da je $C_1 + C_2 = 0$. Jedna konstanta mora biti pozitivna, a druga negativna ili su obje nula. Uzmimo da je $C_1 = k^2$, a $C_2 = -k^2$

$$\frac{d^2X}{dx^2} = k^2\ X$$

$$\frac{d^2Y}{dy^2} = -k^2\ Y$$

te dobivamo dvije obične parcijalne jednadžbe koje možemo lakše riješiti nego početnu jednadžbu. Rješenja funkcija $X(x)$ i $Y(y)$ su

$$X(x) = A\ e^{kx} + B\ e^{-kx}$$
$$Y(y) = C\ \sin(ky) + D\ \cos(ky)$$

tako da je potencijal onda

$$V(x,y) = \left(A\ e^{kx} + B\ e^{-kx}\right)(C\ \sin(ky) + D\ \cos(ky))$$

Ovo je zadovoljavajuće riješenje Laplaceove jednadžbe koje smo dobili metodom separacije varijabli. Preostaje još jedino naći rješenje koje zadovoljava sve rubne uvjete.

Pošto su rubni uvjeti simetrični s obzirom na x tako da je $V(-x,y) = V(x,y)$ slijedi da $A = B$

$$V(x,y) = A\left(e^{kx} + e^{-kx}\right)\left(C\ \sin(ky) + D\ \cos(ky)\right).$$

Koristeći izraz

$$e^{kx} + e^{-kx} = 2\ \cosh(kx)$$

te apsorpcijom konstante $2A$ u konstante C i D dobivamo

$$V(x,y) = \cosh(kx)\ \left(C\ \sin(ky) + D\ \cos(ky)\right).$$

Rubni uvjeti (i) i (ii) daju $D = 0$ te $k = n\pi/a$, što rezultira rješenjem

$$V_n(x,y) = C_n\ \cosh\left(\frac{n\pi\ x}{a}\right)\ \sin\left(\frac{n\pi\ y}{a}\right).$$

Preostaje još konstruirati opću linearnu kombinaciju rješenja

$$V(x,y) = \sum_{n=1}^{\infty} C_n\ \cosh\left(\frac{n\pi\ x}{a}\right)\ \sin\left(\frac{n\pi\ y}{a}\right)$$

te odrediti koeficijente C_n da zadovolje (iii) rubni uvjet

$$V(x=b,y) = \sum_{n=1}^{\infty} C_n\ \cosh\left(\frac{n\pi\ x}{a}\right)\ \sin\left(\frac{n\pi\ y}{a}\right) = V_0.$$

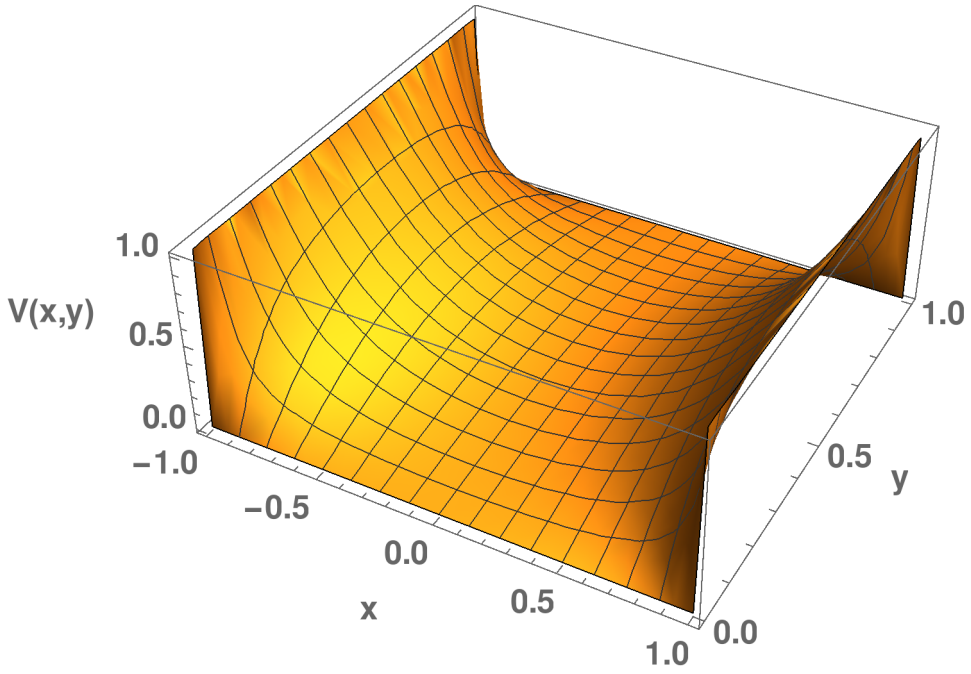
Slijedi:

$$C_n = \begin{cases} 0 & \text{za parne } n \\ \frac{4V_0}{n\pi} & \text{za neparne } n \end{cases}$$

Konačno rješenje potencijala za dani problem je

$$V(x,y) = \frac{4V_0}{\pi} \sum_{n=1,3,5\cdots} \frac{1}{n}\ \frac{\cosh(n\pi x/a)}{\cosh(n\pi b/a)} \sin(n\pi y/a). \tag{5.2}$$

Slika 5.2 pokazuje potencijal u prostoru za odabrane $V_0 = 1$, $b = \pm 1$ te $y = 1$.



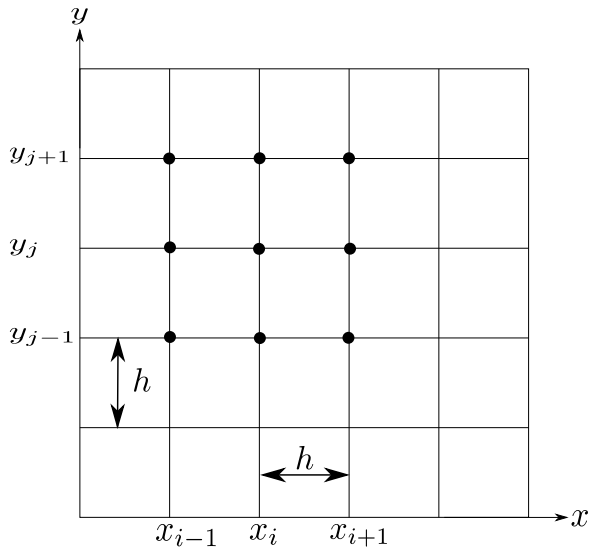
Slika 5.2: Analitičko rješenje potencijala.

5.2 Numeričko rješenje

5.2.1 Metoda konačnih razlika

Metoda konačnih razlika (MKR) je jedna od najjednostavnijih i najlakših numeričkih aproksimacija kojima rješavamo diferencijalne jednadžbe. Nju ju koristio čak i L. Euler u 18. stoljeću za probleme u jednoj dimenziji, veću upotrebu vidimo tek oko 1950. godine zbog razvoja računalnih arhitektura.

Kako bi izračunali numeričko rješenje jednadžbe 2.8 diskretiziramo prostornu i vremensku domenu te aproksimaciju rješenja računamo u tim točkama. U elektrostatici potencijal se ne mijenja u vremenu stoga ne moramo računati rješenje u vremenskoj domeni. Prostornu domenu podijelili smo na ekvidistantne koordinate (rešetka) koje su međusobno udaljene za $h = \frac{X}{N} = \frac{Y}{N}$, gdje su X i Y duljine po x i y osi domene.



Slika 5.3: Ekvidistantna diskretizacija domene u kojoj aproksimiramo rješenje. Indeksi diskretne x koordinate su označeni s i, a y koordinate s j.

Glavni koncept MKR je aproksimacija parcijalne derivacije neprekinute funkcije ϕ u svakoj točki diskretne domene pomoću okolnih točaka koristeći razvoj funkcije u Taylorov red. Ako razvijamo funkciju u točki x_0

$$\phi(x_0 + \Delta x) = \sum_{n=0}^{N} \frac{\phi^{(n)}(x_0)}{n!}(x_0 + \Delta x - x_0)^n = \sum_{n=0}^{N} \frac{\phi^{(n)}(x_0)}{n!}\Delta x^n \tag{5.3}$$

Prvih nekoliko članova tog razvoja su

$$\phi(x_0 + \Delta x) = \phi(x_0) + \Delta x \frac{d\phi}{dx}(x_0) + \frac{\Delta x^2}{2!}\frac{d^2\phi}{dx^2}(x_0) + \frac{\Delta x^3}{3!}\frac{d^3\phi}{dx^3}(x_0) + \mathcal{O}(\Delta x^4) \tag{5.4}$$

gdje je $\mathcal{O}(\Delta x^4)$ ostatak kojem najveći doprinos ima član sa Δx^4. Zbog tog ostatka upravo i govorimo o aproksimaciji. Gornji izraz nam govori da ako poznajemo vrijednost funkcije i njene derivacije u točki x_0 tada možemo aproksimirati vrijednost funkcije ϕ u točki $x + \Delta x$.

Kod MKR krećemo od pretpostavke da su nam vrijednost funkcije ϕ poznate u svim točkama rešetke, a zanima nas koliki su iznosi derivacija. Iz jednadžbe (5.3) uzmemo samo prva dva člana razvoja, umjesto Δx uvrstimo h

$$\phi(x_0 + h) = \phi(x_0) + h\frac{d\phi}{dx}(x_0) + \mathcal{O}(h^2) \tag{5.5}$$

te izvučemo izraz za prvu derivaciju

$$\frac{d\phi}{dx}(x_0) = \frac{\phi(x_0+h)-\phi(x_0)}{h} - \frac{\mathcal{O}(h^2)}{h}.$$

Uvodimo drugačiju notaciju za derivaciju radi ljepšeg zapisa

$$\left(\frac{d\phi}{dx}\right)\Big|_{x_0} = \frac{\phi(x_0+h)-\phi(x_0)}{h} - \mathcal{O}(h)$$

te ako odbacimo ostatak dobivamo diferencijsku jednadžbu

$$\left(\frac{d\phi}{dx}\right)\Big|_{x_0} = \frac{\phi(x_0+h)-\phi(x_0)}{h}$$

koja se naziva diferencija unaprijed jer za računanje derivacije u točki x_0 je potrebno znati iznos funkcije u slijedećoj točki. Ako pomoću Taylorov razvoja (5.3) želimo dobiti iznos funkcije u točki $(x_0 - h)$ dobivamo slijedeći izraz

$$\phi(x_0-h) = \phi(x_0) - \left(\frac{d\phi}{dx}\right)\Big|_{x_0} h + \mathcal{O}(h^2). \tag{5.6}$$

Zanemarimo ostatak te kada izrazimo derivaciju dobivamo diferencijsku jednadžbu

$$\left(\frac{d\phi}{dx}\right)\Big|_{x_0} = \frac{\phi(x_0)-\phi(x_0-h)}{h}$$

koju nazivamo diferencija unatrag jer za računanje derivacije u točki x_0 potrebno je znati vrijednost funkcije u prijašnjoj točki. Ako od (5.5) oduzmemo (5.6) dobivamo

$$\phi(x_0+h)-\phi(x_0-h) = \phi(x_0)-\phi(x_0)+\left(\frac{d\phi}{dx}\right)\Big|_{x_0} h + \left(\frac{d\phi}{dx}\right)\Big|_{x_0} h = 2\left(\frac{d\phi}{dx}\right)\Big|_{x_0} h$$

te iz gornjeg izraza izrazimo derivaciju

$$\left(\frac{d\phi}{dx}\right)\Big|_{x_0} = \frac{\phi(x_0+h)-\phi(x_0-h)}{2h}$$

dobivamo diferencijsku jednadžbu koju nazivamo centralnom diferencijom.

Zbrajanjem jednadžbi (5.6) i (5.5) dobivamo izraz iz kojeg možemo izraziti dife-

rencijsku jednadžbu drugog reda, odnosno možemo aproksimirati drugu derivaciju

$$\phi(x_0 - \Delta x) + \phi(x_0 + \Delta x) = \phi(x_0) - h \left(\frac{d\phi}{dx}\right)\Big|_{x_0} + \frac{h^2}{2} \left(\frac{d^2\phi}{dx^2}\right)\Big|_{x_0} + \phi(x_0) + h \left(\frac{d\phi}{dx}\right)\Big|_{x_0} + \frac{h^2}{2} \left(\frac{d^2\phi}{dx^2}\right)\Big|_{x_0} + \mathcal{O}(h^3)$$

$$\left(\frac{d^2\phi}{dx^2}\right)\Big|_{x_0} = \frac{\phi(x_0 - \Delta x) - 2\phi(x_0) + \phi(x_0 + \Delta x)}{h^2} \tag{5.7}$$

Iz gornje jednadžbe vidimo da je za aproksimaciju druge derivaciju funkcije u točki x_0 potrebno znati iznose funkcije u istoj točki te u susjednim točkama.

Sada možemo jednadžbu (2.8) raspisati pomoću metode konačnih razlika. Za izvod smo gledali funkciju ϕ koja je ovisila o jednoj varijabli x radi lakšeg zapisa. Dalje razmatramo funkciju dvije varijable x i y. Druge derivacije zamijenimo s izrazima za numeričke derivacije (5.7) s time kada radimo parcijalne derivacije po x tada y držimo konstantnim

$$\frac{\phi(x_0 - h, y_0) - 2\phi(x_0, y_0) + \phi(x_0 + h, y_0)}{h^2} + \frac{\phi(x_0, y_0 - h) - 2\phi(x_0, y_0) + \phi(x_0, y_0 + h)}{h^2} = 0 \tag{5.8}$$

Iz gornje jednadžbe izrazimo iznos funkcije u točki x_0, y_0

$$\phi(x_0, y_0) = \frac{1}{4} \left(\phi(x_0 - h, y_0) + \phi(x_0 + h, y_0) + \phi(x_0, y_0 - h) + \phi(x_0, y_0 + h)\right) \tag{5.9}$$

te vidimo da će iznos funkcije u toj točki biti srednja vrijednost funkcija u okolnim točkama.

Laplaceova jednadžba mora biti zadovoljena u svim točkama prostora. Možemo generalizirati izraz (5.9) tako da koristimo notaciju i, j koju smo uveli

$$\phi_{i,j} = \frac{1}{4} \left(\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1}\right) \tag{5.10}$$

Izvrednjujući taj izraz za sve sukcesivne točke prostora bez rubnih, počevši od točke $(2, 2)$ dobivamo sustav (N-2)(N-2) jednadžbi

$$
\begin{array}{lllll}
\phi_{2,2} = & 0.25\phi_{1,2}+ & 0.25\phi_{3,2}+ & 0.25\phi_{2,1}+ & 0.25\phi_{2,3} \\
\phi_{3,2} = & 0.25\phi_{2,2}+ & 0.25\phi_{4,2}+ & 0.25\phi_{3,1}+ & 0.25\phi_{3,3} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\phi_{N-1,2} = & 0.25\phi_{N-2,2}+ & 0.25\phi_{N,2}+ & 0.25\phi_{N-1,1}+ & 0.25\phi_{N-1,3} \\
\phi_{2,3} = & 0.25\phi_{1,3}+ & 0.25\phi_{3,3}+ & 0.25\phi_{2,2}+ & 0.25\phi_{2,4} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\phi_{N-1,N-1} = & 0.25\phi_{N-2,N-1}+ & 0.25\phi_{N,N-1}+ & 0.25\phi_{N-1,N-2}+ & 0.25\phi_{N-1,N}
\end{array}
$$

Taj sustav se može zapisati u obliku

$$A\ \phi = F$$

te ga je moguće rješavati s Gaussovom eliminacijskom metodom. Što je veći prostor odnosno broj točaka u prostoru to i sustav jednadžbi raste i vrijeme potrebno za izračun se povećava.

5.2.2 Metoda relaksacije

U praktičnoj primjeni matrica A je često velika matrica što čini direktno računanje vremenski zahtjevnim. Efikasnija metoda pogodna za ovaj izračun je relaksacijska (iteracijska) metoda. Postave se rubni uvjeti, proizvoljni početni iznosi funkcije u unutrašnjosti prostora te se izračuna novi iznos preko relacije (5.10) koja je bolja aproksimacija funkcije. Ovaj proces se nastavlja dok se ne postigne zadana predefinirana tolerancija.

Uvodimo indeks iteracije m i te dobivamo Jacobijevu jednadžbu

$$\phi_{i,j}^{m+1} = \frac{1}{4}\left(\phi_{i-1,j}^{m} + \phi_{i+1,j}^{m} + \phi_{i,j-1}^{m} + \phi_{i,j+1}^{m}\right) \tag{5.11}$$

Za svaku točku unutar domene (i, j), iznos funkcije ϕ za slijedeću iteraciju (m+1) računamo preko (5.11). Kada je iteracija izvršena za sve točke, gledamo koliko se rješenja u prethodnoj i izračunatoj iteraciji razlikuju $|\phi^{m+1} - \phi^{m}|$. Ako je razlika manja od zadane tolerancije (ε)

$$|\phi^{m+1} - \phi^{m}| \leq \varepsilon$$

tada je rješenje problema (2.8) numerička funkcija ϕ^{m+1}.

Pomoću C koda testirali smo brzinu izvršavanja iteracijske metode. Samo vrijeme izvršavanja while petlje gledamo kao vrijeme izvršavanja. Toleranciju smo postavili na $\varepsilon = 10^{-7}$ te pomoću `counter` varijable brojimo koliko je iteracija izvršeno.

Inicijaliziramo matricu V, za 10^2 točaka s kojom ćemo računati potencijal te inicijaliziramo ostale varijable te postavimo toleranciju

```
int n=10,m=10;
double *V;
V = (double*) malloc(n*m*sizeof(double));
int counter=0;
int i,j;
double temp, tolerance=1e-7, err=10, temp_err;
```

Zatim postavimo zadane rubne uvjete iz problema

```
for(i=0; i<n; i++)
{
    V[i]=1;
    V[i+(m-1)*m]=1;
    V[0+i*m]=0;
    V[(m-1)+i*m]=0;
}
```

Unutar `while` petlje računamo relaksirano rješenje te maksimalnu grešku između dvije iteracije sve dok se tolerancija ne zadovolji. Na kraju pogledajmo koliko je potrebno iteracija da se taj uvjet zadovolji.

```
clock_t begin = clock();
while(err > tolerance){
        for(i=1; i<(n-1); i++){
            err=0;
            for(j=1; j<(m-1); j++){
                temp=0.25*(V[i-1][j]+V[i+1][j]+V[i][j-1]+V[i][j+1]);
```

```
                temp_err=fabs(temp-V[i][j]);
                V[i][j]=temp;
                if(temp_err > err) err = temp_err;
                }
    }
counter++;
}
clock_t end = clock();
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
```

Istu metodu smo napravili pomoću računalne heterogene metode, odnosno napravili smo CUDA C program koji se vrti na GPU. Unutar već gore napisanog koda smo inicijalizirali pokazivače na memoriju grafičke kartice te prebacili početnu matricu `V` u matrice `dev_V` i `dev_b` na grafičkoj kartici.

```
double* dev_V;
double* dev_b;
cudaMalloc((void **) &dev_V, sizeof(V[0][0]) *n*m);
cudaMalloc((void **) &dev_b, sizeof(V[0][0]) *n*m);
cudaMemcpy(dev_V, V, sizeof(double)*n*m, cudaMemcpyHostToDevice);
cudaMemcpy(dev_b, V, sizeof(double)*n*m, cudaMemcpyHostToDevice);
```

Konstruirali smo kernel `conv` koja prima dva pokazivača odnosno matrice. U njemu računamo prvo indekse dretvi stupaca `col` i redova `row` te zatim indekse elementa matrice zato jer je matrica zapisana u memoriju kao jednodimenzionalni array. Odnosno u CUDA C se ne može elementu matrice pristupiti preko dva indeksa npr. `V[x][y]`. Na ovakav način svaka dretva računa jedan element matrice. Da naglasim, nikakve optimizacije kernela niti optimizacije memorijske obrade nismo u ovom radu primjenjivali.

```
__global__ void conv( double* a, double* b)
{
        int col = blockIdx.x*blockDim.x+threadIdx.x+1;
        int row = blockIdx.y*blockDim.y+threadIdx.y+1;
```

```
        int n=(gridDim.x*blockDim.x+2);
        int index = col + row * n;
        b[index]=0.25*(a[(col+1) + row * n] + a[(col-1) + row * n]
        + a[col + (row+1) * n] + a[col + (row-1) * n]);
}
```

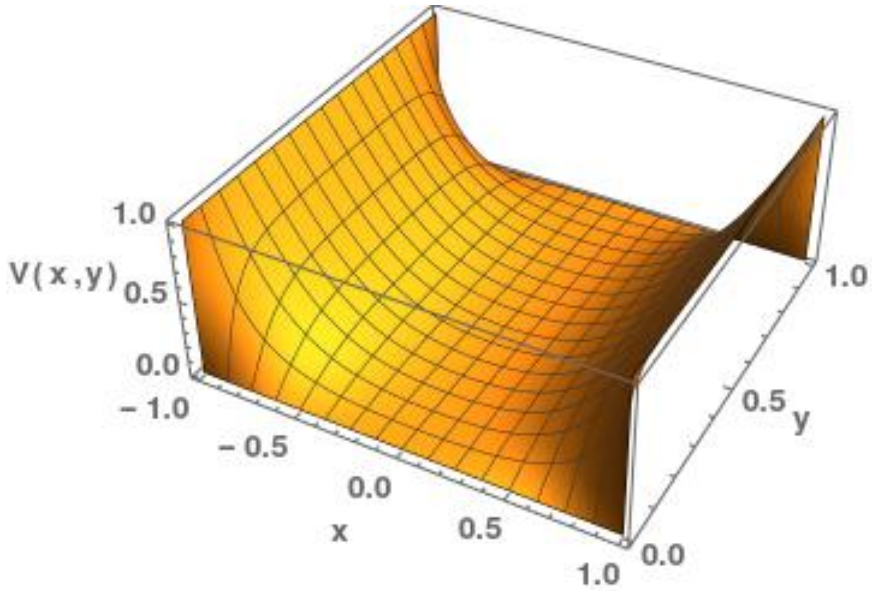
U host kodu `while` petlju zamijenili smo `for` petljom u kojoj smo pokretali kernele. `For` petlja se iterirala isti broj puta kao i `while` petlja u C kodu. Prvim kernelom smo napravili prvu iteraciju te je kernel izračunao potencijal i spremio u matricu `dev_b`, a zatim smo ponovno pokrenuli kernel i napravili drugu iteraciju relaksacijske metode i rezultate spremili u matricu `dev_V`. Pomoću naredbe `cudaDeviceSynchronize();` i gore opisanog pokretanja kernela, osigurali smo da se dretve ne preklapaju. Efektivno smo napravili dvostruko više iteracija relaksacijske metode. Vrijeme izvršavanja koda gledali smo kao vrijeme izvršavanja ove `for` petlje.

```
for(i=0; i<iteration; i++)
{
        conv<<<grid, block>>>( dev_V, dev_b);
        cudaDeviceSynchronize();
        conv<<<grid, block>>>( dev_b, dev_V);
        cudaDeviceSynchronize();
}
```

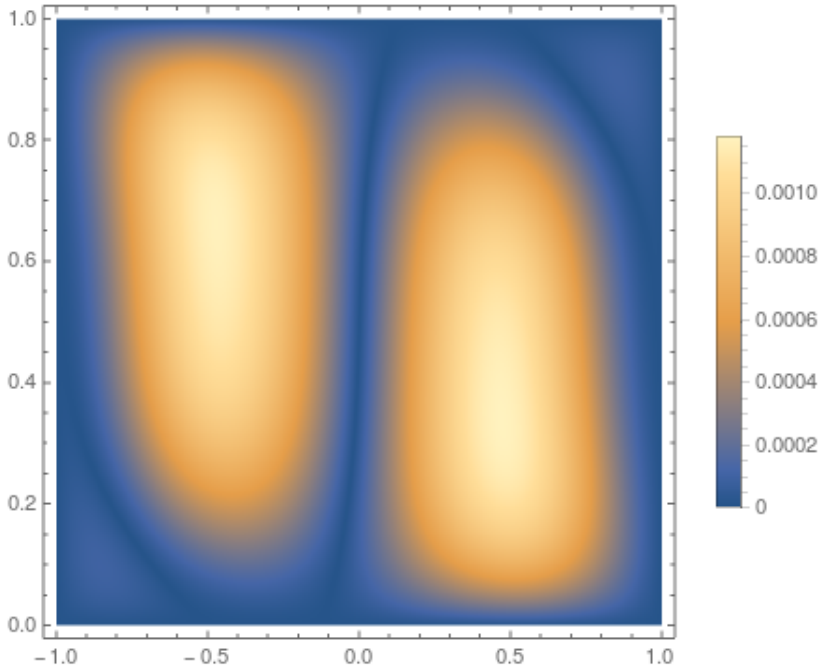
Kernele smo pokretali u slijedećoj konfiguraciji dretvi. Dvodimenzionalni blokovi dretvi su bili veličine 32x32 što je bio maksimum koji grafička kartica hardverski dopušta. Veličinu grida smo izračunali tako da dobijemo broj dretvi koliko nam je bilo potrebno da računamo samo po elementima unutar matrice, a da ne uzimamo i rubne elemente matrica odnosno rubne uvjete koji se ne smiju mijenjati.

```
dim3 block(32,32); //
dim3 grid((n-2)/block.x, (m-2)/block.y);
```

Rezultate ove metode, izračunate na CPU-u te na heterogenom sustavu, vizualno se ne razlikuju prikaz možemo vidjeti na slici 5.4. Na slici 5.5 se vidi koliko rezultati izračunati na CPU i na GPU odstupaju jedan od drugog.



Slika 5.4: Potencijal u prostoru za dani problem koji smo izračunali na računalnom procesoru.



Slika 5.5: Apsolutna razlika potencijala koje smo izračunali na računalnom i grafičkom procesoru.

U tablici 5.1 su prikazani rezultati numeričkog računa. n je veličina matrice, `CPU` je vrijeme u sekundama potrebno da se program završi na računalnom procesoru, `GPU` je vrijeme u sekundama potrebno da se program završi na grafičkom procesoru. U stupcu iteracija prikazan je broj potrebnih iteracija da relaksacijska metoda pronađe zadovoljavajuće rješenje. Za numeričke izračune u ovom radu koristili smo Intel Core i7-4790 procesor i NVidia GeForce GTX 980 grafičku karticu.

n	CPU /s	GPU / s	iteracija	ubrzanje
34	0,0063	0,048	969	0,13
322	13	1,54	21097	8,44
642	40	0,63	15531	63,49
6402	6590	158	15516	41,71

Tablica 5.1: Vremena izvršavanja programa za zadanu toleranciju $\varepsilon = 10^{-7}$

6 TensorFlow

TensorFlow je softverska biblioteka otvorenog koda koja je prvobitno razvijena s ciljem lakše implementacije algoritama za strojno učenje. Razvio ju je tim Google Brain za internu upotrebu, a napisana je u pythonu, C++ i CUDA C programskim jezicima te je od 2015. godine javno dostupna. Riječ je o simboličkoj matematičkoj biblioteci, koja je prvobitna razvijena za algoritme strojnog učenja kao što su duboke neuronske mreže, ali preko te biblioteke se lako mogu iskoristiti heterogeni sustavi ako su dostupni. Upravo zbog toga smo u ovom radu i koristili Tensorflow. Obrađenu relaksacijsku metodu implementirali smo u python programskom jeziku pomoću TensorFlow biblioteke te usporedili brzinu izvršavanja programa na CPU i heterogenom sustavu.

TensorFlow biblioteka ima neuobičajenu praksu programiranja jer se sve u njoj temelji na izradi takozvanih računalnih grafova. Čvorovi u grafu predstavljaju tenzore koji mogu biti konstante (`tf.constant`), ulazni podatci (`tf.placeholder`), varijable (`tf.variable`) te tenzorski izrazi. U slijedećem primjeru pozivamo TensorFlow biblioteku te kreiramo konstantu `x` i pridjeljujemo joj vrijednost 35. Nakon toga kreiramo varijablu `y` definiranu jednadžbom $x + 5$. Na kraju ispisujemo vrijednost varijable `y`.

```
import tensorflow as tf

x = tf.constant(35)
y= tf.Variable(x + 5)

print(y)
```

Kada pokrenemo gornji kod on ispisuje

```
<tf.Variable 'y:0' shape=() dtype=int32_ref>
```

Razlika u odnosu na python je upravo u tome što smo gornjim kodom zadali naredbe grafu što će raditi, ali nismo još pokrenuli izvođenje samog grafa. Print naredbom samo ispisali kreirani objekt odnosno tenzor. Kako bi izvrijednili graf potrebno je pokrenuti izvedbeni kontekst `session` te pozvati metodu `run` pokrenutog izvedbenog

konteksta. Dodatno TensorFlow varijable (`tf.variable`) nisu inicijalizirane te ih je potrebno inicijalizirati. To smo napravili u slijedećem kodu

```
model =  tf.global_variables_initializer()

with tf.Session() as session:
  session.run(model)
  print(session.run(y))
```

Tek sada dobivamo računski rezultat.

Do sada smo koristili samo konstante i varijable, no kao što smo naveli postoji još dodatna osnovna struktura `placeholder`. Ona je zapravo tenzorska varijabla kojoj ćemo tek kasnije prilikom izvršavanja grafa dodjeliti neku vrijednost. U TensorFlow terminologiji, mi unosimo podatke u graf preko tih placeholdera. Pozivamo metodu `run` stavljajući prvi argument listu čvorova koje želimo izračunati, u ovom slučaju čvor `y`. Drugi argument su vrijednosti ulaznih podataka koje će biti iskorištene tijekom izračuna. `y` nakon poziva metode `run` referencira listu `[ 2.  4.  6.]`.

```
import tensorflow as tf

x = tf.placeholder("float", None)
y = x * 2

with tf.Session() as session:
    result = session.run(y, feed_dict={x: [1, 2, 3]})
    print(result)
```

Ulazni podatci mogu biti zadani ili numpy objektima ili ugrađenim tipovima python programskog jezika te je prijenos podataka iz numpy u Tensorflow na taj način omogućen.

Tenzorskim varijablama ne možemo jednostavno pridružiti vrijednost tijekom pisanja koda nego moramo koristiti naredbu `tf.assign` odnosno tenzorsku operaciju. U slijedećem primjeru tenzoru `Ut` dodjeljujemo novu vrijednost

```
op = tf.assign(Ut, np.array([1,2]))
```

Operaciju `op` tijekom izvođenja koda moramo izvršiti pomoću `run(op)`. Postoji još i interaktivni izvedbeni kontekst. Jedina razlika između običnog i interaktivnog izvedbenog konteksta je u tome da se interaktivni kontekst prilikom pozivanja nameće kao zadani izvedbeni kontekst te se operacije mogu pokretati s `op.eval()`.

Sada ćemo razmotriti implementaciju relaksacijske metode u TensorFlowu. Inicijaliziramo veličinu početne matrice varijablom `m` te postavljamo rubne uvjete.

```
m = 642
U=np.zeros((m, m),dtype=np.float64)
U[0]=1
U[m-2]=1
```

Definiramo varijablu `Ut` kojoj pridružujemo matricu `U`. Koristimo standardni 2D konvolucijski kernel kako implementirali jednadžbu 5.11. Na taj način smo dobili usrednjavanje člana početne matrice sa susjedima

$$\begin{bmatrix} 0 & 0.25 & 0 \\ 0.25 & 0 & 0.25 \\ 0 & 0.25 & 0 \end{bmatrix}$$

```
Ut  = tf.Variable(U)

def make_kernel(a):
        """Transform a 2D array into a convolution kernel"""
        a = np.asarray(a)
        a = a.reshape(list(a.shape) + [1,1])
        return tf.constant(a, dtype="float64")

def simple_conv(x, k):
    """A simplified 2D convolution operation"""
    x = tf.expand_dims(tf.expand_dims(x, 0), -1)
    y = tf.nn.depthwise_conv2d(x, k, [1, 1, 1, 1], padding='SAME')
```

```
    return y[0, :, :, 0]

def laplace(x):
    """Compute the 2D laplacian of an array"""
    laplace_k = make_kernel([[0., 0.25, 0.],
                             [0.25, 0., 0.25],
                             [0., 0.25, 0.]])
    return simple_conv(x, laplace_k)
```

Definiramo operacije za računanje iteracija u relaksacijskoj metodi `assign_op` te operacije `assign_op2-assign_op5` za postavljanje rubnih uvjeta nakon svake iteracije. To činimo upravo zato jer prva operacija djeluje preko čitave matrice te prebrisuje i rubne uvjete.

```
npc=np.ones((m), dtype=np.float64)
assign_op = tf.assign(Ut, laplace(Ut))
assign_op2 = tf.assign(Ut[0], npc)
assign_op3 = tf.assign(Ut[m-1], npc)
assign_op4 = tf.assign(Ut[1:(m-1),0], np.zeros((m-2), dtype=np.float64))
assign_op5 = tf.assign(Ut[1:(m-1),m-1], np.zeros((m-2), dtype=np.float64))
```

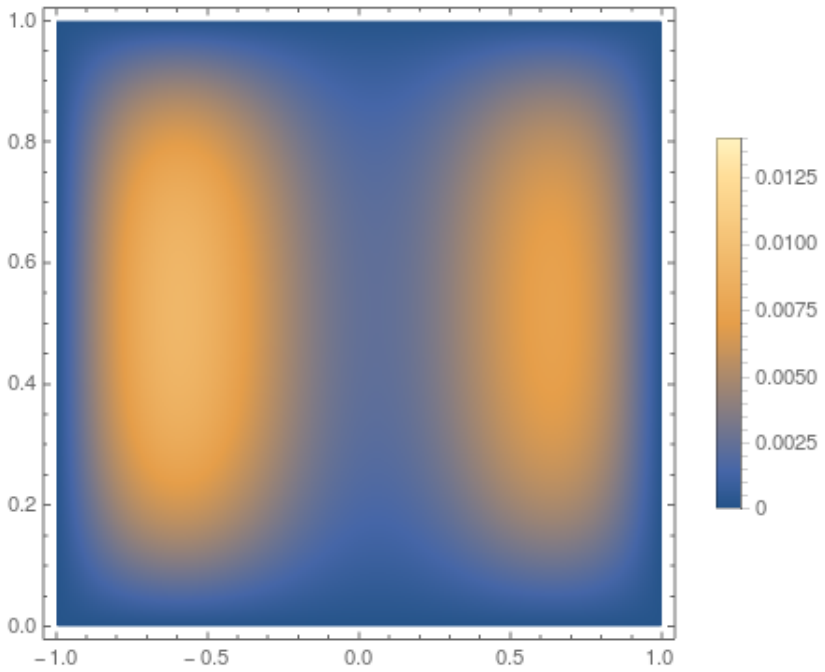
Na kraju inicijaliziramo varijable, pokrećemo izvedbeni kontekst te pokrećemo svaku operaciju unutar `foor` petlje isti broj puta kao i u prijašnjem poglavlju. Kod se izvršava na CPU ako inicijaliziramo sve varijable i operacije unutar funkcije

```
with tf.device('/cpu:0'):
```

ili na GPU-u funkcijom

```
with tf.device('/gpu:0'):
```

Rezultati se također vizualno ne razlikuju stoga stavljamo sliku 6.1 koja prikazuje apsolutnu razliku između izračuna u CUDA C i TensorFlow implementaciji na GPU.



Slika 6.1: Apsolutna razlika izračunatih potencijala u CUDA C implementaciji i TensorFlow biblioteci.

Prednost korištenja TensorFlowa za računanje na heterogenim sustavima je lak prelazak s pythona na ovu biblioteku zbog slične sintakse te mogućnost korištenja numpy biblioteke u oba slučaja. Dodatno nije potrebno nikakvo znanje o hardveru i arhitekturi grafičkih kartica, te posebnosti programiranja na njima kako bi se iskoristile njihove mogućnosti. Kao što vidimo lakši pristup i nije uvijek najbolji te su ubrzanja drastično manja.

n	CPU /s	GPU / s	iteracija	ubrzanje
34	1,13	1,03	969	1,10
322	46,1	37,8	21097	1,22
642	111	51	15531	2,18
6402	16260	6900	15516	2,35

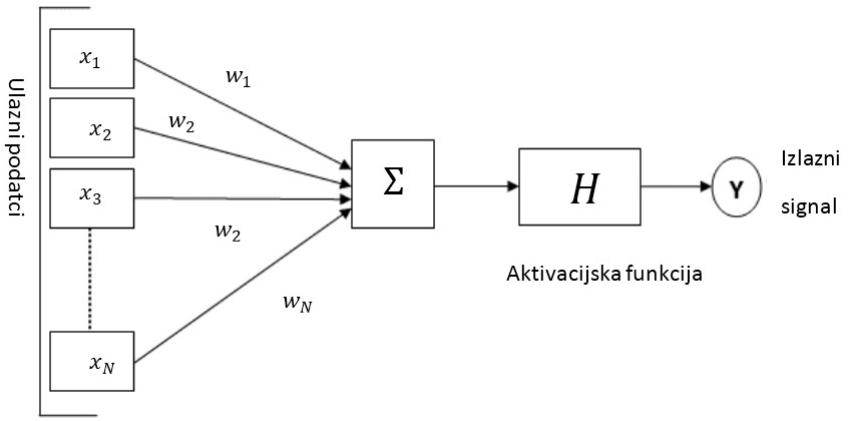
Tablica 6.1: Vremena izvršavanja programa u TensorFlow biblioteci za CPU i GPU implementaciju.

7 Neuronske mreže

Numeričke metode koje smo do sada iznijeli u ovom radu, ali i druge metode, poput metode konačnih elemenata i volumena se oslanjaju na diskretizaciju prostora i aproksimacije rješenja na diskretnom prostoru. Takve metode imaju ograničenja zbog ograničene diferencijabilnosti i upitne mogućnosti interpolacije. Kako bi se izbjegli ovi problemi danas se koriste neuronske mreže za rješavanje diferencijalnih jednadžbi pa tako i parcijalnih diferencijalnih jednadžbi. U ovom poglavlju dat ćemo kratki pregled osnova neuronskih mreža te našu implementaciju neuronske mreže u Keras paketu. Neuronsku mrežu iskoristit ćemo za interpolaciju rješenja problema na diskretni prostor s većim brojem točaka.

Neuronske mreže imaju dobra svojstva zbog čega ih se sve više koristi za numeričke izračune. Rješenja koja takve mreže daju su diferencijabilna te su u obliku zatvorene analitičke forme. Također teorem o univerzalnoj aproksimaciji (Cybenko teorem) kaže da umjetne neuronske mreže mogu aproksimirati bilo koju neprekidnu funkciju na $\mathbb{R}^n$. Neuronske mreže se mogu implementirati na heterogenim sustavima.

Umjetna neuronska mreža (artificial neural network) je mreža umjetnih neurona koji su međusobno povezani. Takva mreža je inspirirana biološkim neuralnim mrežama. Svaka veza (sinapsa) među neuronima šalje signal od jednog neurona do drugog. Neuron koji prima signal izvrši operaciju na tom signalu te ga šalje dalje. Najčešće u najjednostavnijim neuronskim mrežama signal je realni broj te izlazni signal svakog neurona se računa preko nelinearne funkcije kao suma ulaznih signala pomnoženih s nekim koeficijentima te na konačni rezultat djeluje aktivacijska funkcija. Pravilo po kojem mreža uči modificira koeficijente pojedinog neurona. Nadzirano učenje se odvija tako da se početni koeficijenti nasumično odaberu te se izlazni signal uspoređuje s referentnim vrijednostima (set podataka za učenje) kako bi se izračunala predefinirana funkcija greške (slično kao u metodi najmanjih kvadrata). Cilj učenja je minimizirati upravo tu funkciju kroz procese nalaženja najboljih koeficijenata.



Slika 7.1: Neuronsku mrežu koja ima samo jedan sloj te samo jedan neuron.

Slika (7.1) prikazuje neuronsku mrežu koja ima samo jedan neuron te se takva mreža naziva perceptron. Ona prima $\vec{x} = (x_1, \cdots, x_N)$ ulaznih podataka te njen izlazni signal je

$$Y = H(\sum_{i=1}^{N} w_i \, x_i)$$

gdje je H step funkcija. Ovakva mreža je jedna od prvih koja je razvijena (1950. godine) te ne pruža puno mogućnosti, ali i danas ju često koristimo za npr. binarnu klasifikaciju.

Češće se koriste višeslojne neuronske mreže s N ulaznih podataka. Razmotrimo primjer mreže sa jednim skrivenim slojem koji ima M neurona koji koriste sigmodialnu aktivacijsku funkciju. Izlaz takve mreže je

$$Y = \sum_{i=1}^{M} v_i \, \sigma(z_i)$$

gdje je

$$z_i = \sum_{j=1}^{N} w_{ij} \, x_j + u_i$$

i-ti izlaz iz skrivenih neurona. w_{ij} su koeficijenti s j-tog ulaznog člana na i-ti skriveni neuron, v_i su koeficijenti s i-tog skrivenog neurona na izlaz mreže. u_i su konstantni doprinosi (bias) skrivenih neurona te $\sigma(z)$ je sigmoidalna aktivacijska funkcija. Često korišten oblik aktivacijske funkcije je:

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

7.1 Keras

Keras je biblioteka otvorenog koda napisana u python programskom jeziku. Glavna namjena joj je strojno učenje kao i kod TensorFlow biblioteke, ali sa puno jednostavnijim funkcijama za izradu neuronskih mreža. U Kerasu generiramo neuronsku mrežu te ju treniramo na rješenjima koje smo dobili relaksacijskom metodom. Dvodimenzionalni prostor smo diskretizirali s 10^2 točaka.

Keras pozivamo s kodom

```
from keras.models import Sequential
from keras.layers import Dense
import keras.backend as K
```

kreiramo neuronsku mrežu koju smo nazvali `model`. Sastoji se od ulaznih članova koji svaki prima dva podatka (x i y koordinate), četiri skrivena sloja te izlaznog sloja. Izlazni neuroni procesiraju signal s linearnom aktivacijskom funkcijom dok svi ostali neuroni koriste `relu` aktivacijsku funkciju koja je definirana kao

$$f(x) = \max(0, x)$$

```
model = Sequential()
model.add(Dense(100, input_dim=2, activation='relu'))
model.add(Dense(80, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='linear'))
```

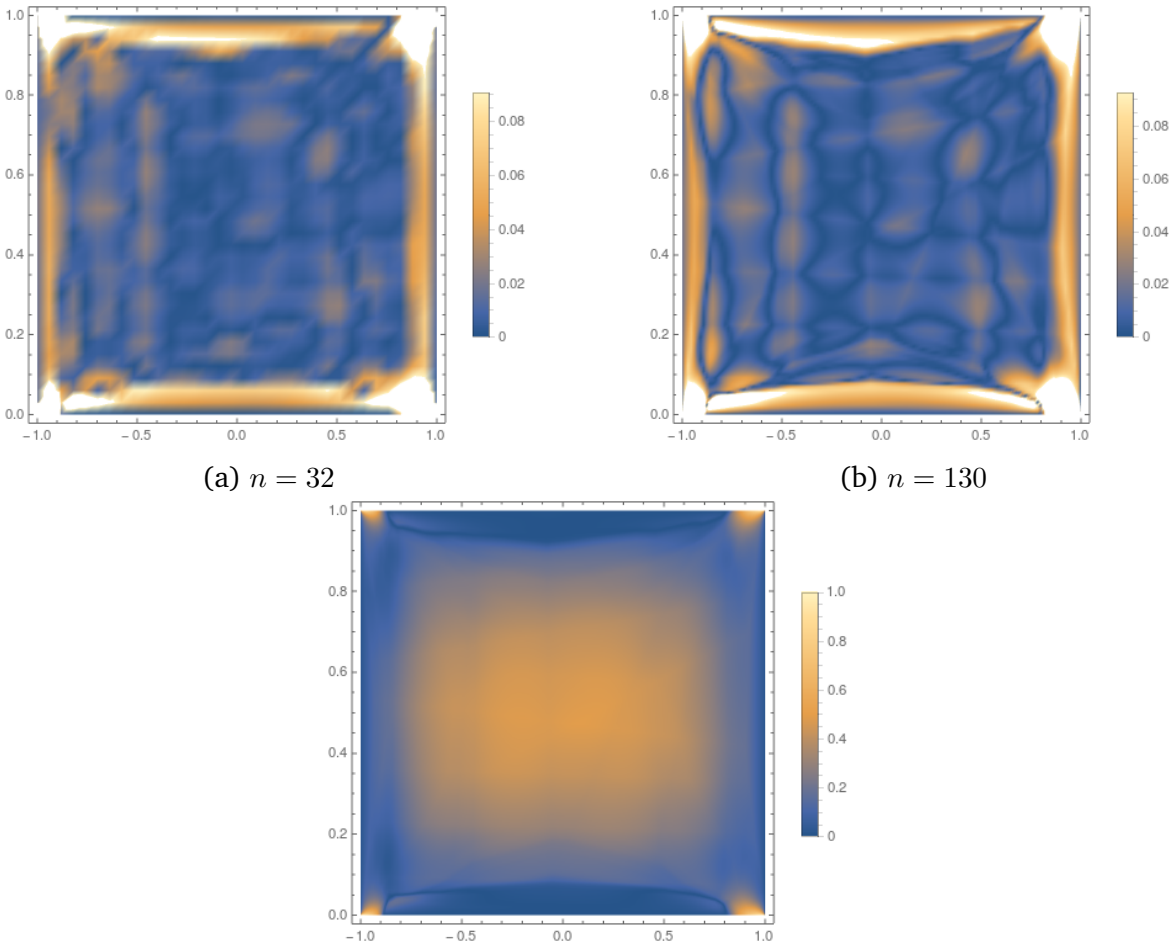
Slijedećim komadom koda kompajliramo mrežu. Za funkciju greške smo uzeli srednju kvadratnu grešku, pomoću opcije `optimizer` namještamo algoritam za učenje. Metodom pokušaja i pogrešaka smo odabrali metodu stohastičke optimizacije (`adam`) te `relu` aktivacijske funkcije jer su dale najbolje rezultate.

```
model.compile(loss= "mean_squared_error", optimizer='adam')
```

Proces učenja pokrećemo s naredbom `model.fit`. Namjestili smo da mreža tisuću puta prolazi kroz iteraciju učenja. `model.predict` generira izlazni signal mreže.

```
model.fit(np.array(X_discrete), np.array(Y_discrete), epochs=1000, batch_size=1)
predictions = model.predict(np.array(X_discrete))
```

Usporedili smo rezultate dobivene neuronskom mrežom s rezultatima koje smo dobili relaksacijskom metodom u CUDA C implementaciji te promotrili odstupanja rezultata. Slike (7.2) prikazuju apsolutna odstupanja između rješenja interpoliranog iz istrenirane neuronske mreže i rješenje koje smo ranije dobili. Prostor smo diskretizirali na tri načina s 32^2, 130^2 i 642^2 točaka.



(a) $n = 32$

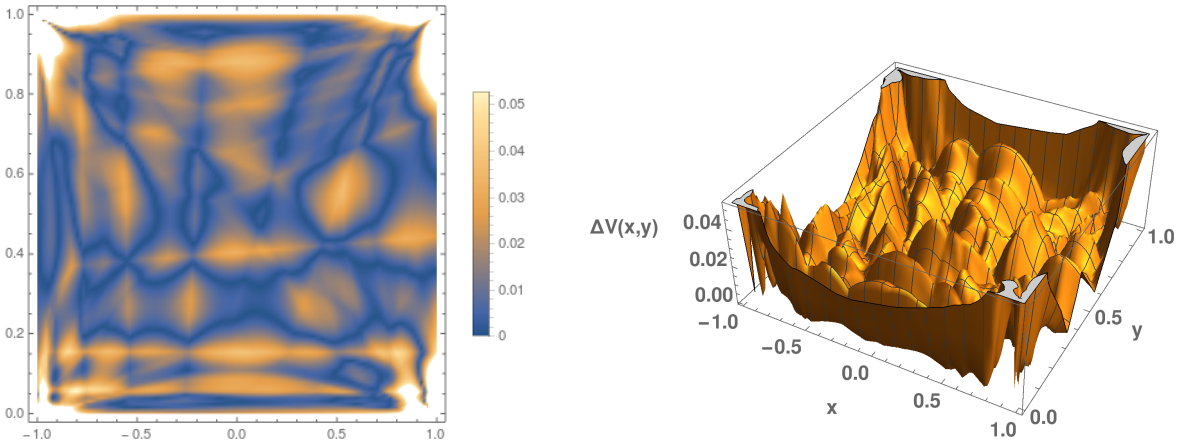
(b) $n = 130$

(c) $n = 642$

Slika 7.2: Apsolutna razliku između rješenja interpoliranog iz istrenirane neuronske mreže i rješenje koje smo dobili u CUDA C implementaciji za različite veličine diskretnog prostora.

Isti postupak ponovili na prostoru od 100^2 točaka. Takvu neuronsku mrežu smo

izvrijednili na prostoru od 130^2 točaka te izračunali apsolutnu razliku između takvog rješenja i rešenja iz relaksacijske metode (slika 7.3).



(a) (b)

Slika 7.3: Apsolutna razlika između rješenja interpoliranog iz istrenirane neuronske mreže na prostoru 100^2 od točaka te interpoliranog na prostor od 130^2 od točaka i rješenja koje smo dobili u CUDA C implementaciji.

Uočavamo da s istreniranim neuronskim mrežama na malom broju točaka diskretnog prostora možemo dobiti rješenje na gušćoj diskretnoj mreži domene. Takva interpolirana rješenja ne odstupaju značajno te uočavamo da su najveća odstupanja na rubovima prostora. Što je veća razlika između broja točaka prostora na kojoj treniramo mrežu i na kojoj interpoliramo rješenje, to je i odstupanje značajnije.

8 Metodički dio

8.1 Istraživački usmjerena nastava

Obrazovanje, u užem smislu, je proces stjecanja znanja, vještina i kompetencija koje koristimo u daljnjem životu. Suvremena društva svoje stupnjeve razvoja često gledaju i kroz formu obrazovanja pa tako napredna društva vrlo često imaju i dobro organizirano obrazovanje. Tako i obrazovanje podliježe promjenama u svrhu napretka i poboljšanja društva. U Republici Hrvatskoj uočavamo sve jaču potrebu za promjenama u obliku cjelovite reforme obrazovanja. PISA testovi kao međunarodni testovi služe za procjenu znanja i vještina petnaestogodišnjih učenika, ali i neposredno za evaluaciju obrazovnog sustava. Testiraju se područja matematike, prirodoslovlja i čitalačke pismenosti. Hrvatska je u takvim testiranjima redovito ispod prosjeka zemalja koje također vrše ta testiranja te u zadnjih nekoliko testiranja nastavlja padati. Mogući zaključak je da je trenutni obrazovni sustav zastario u odnosu na sustave drugih zemalja.

Trenutno se nastava velikim dijelom temelji na tradicionalnom predavačkom tipu nastave gdje profesori drže nastavu te se naglasak stavlja na reprodukciju činjenica, a ne na razumijevanju novog znanja i povezivanju s već postojećim znanjem. Istraživanja su pokazala da je nastava fizike zahtjeva od učenika visoki intelektualni angažman što nije moguće postići tradicionalnim pristupom. Nastava fizike se počela poboljšavati pokušavajući uvesti interaktivnu i istraživačku usmjerenu nastavu. Takva nastava ima dva aspekta kako joj i sam naziv govori, istraživački i interaktivni pristup.

Istraživački pristup se temelji na osnaživanju učeničkog znanstvenog propitivanja i razmišljanja potičući ih da sami traže odgovore na znanstvena pitanja. Jedan od veoma važnih ciljeva koje želimo ostvariti je da učenici mogu sami postavljati, testirati i evaluirati znanstvene hipoteze te sukladno njima osmišljavati i provoditi pokuse te na tim temeljima donositi zaključke. Također želimo da samostalno zapisuju svoja predviđanja, bilježe svoja opažanja te donose zaključke.

Interaktivni pristup se temelji na korištenju interkativnih metoda u nastavi u svrhu ostvarivanja bolje komunikacije između profesora i učenika. Neke od takvih metoda su:

- razredna rasprava
- konceptualna pitanja s karticama
- interaktivno izvođenje pokusa
- računalne interaktivne simulacije
- rješavanje zadataka u grupama

Na taj način se smanjuje uloga profesora jer on više ne iznosi same činjenice, već se aktivnost prebacuje na učenike kako ne bi bili pasivni promatrači nastave. Istraživački usmjeren sat se temelji na pokusima te profesori navode učenike i usmjeravaju ih postavljajući im pitanja.

Nastavni sat se sastoji od uvodnog, središnjeg te završnog dijela. U uvodnom dijelu nam je bitno ostvariti interakciju s učenicima te razrednom raspravom prikupiti povratne informacije o postojećem znanju. Nadalje u uvodnom dijelu bitno je prezentirati atraktivni uvodni problem kako bi povećali interes učenika za nastavnu cjelinu koja će se obrađivati. Takvi problemi su najčešće povezani sa svakodnevnim iskustvom učenika te bi ih bilo idealno potkrijepiti demonstracijskim pokusom. Središnji dio sata koristimo za konstrukciju fizikalnog modela kojeg istražujemo te želimo matematički opisati izgrađeni model. U ovom dijelu sata se često koristimo interaktivnim metodama, pogotovo razrednom raspravom, kako bi navodili i usmjeravali učenike pri njihovom zaključivanju. U završnom dijelu sata želimo primijeniti i procijeniti novo stečeno znanje kako bi sat imao zaokruženu i cjelovitu priču. Ovaj dio sada je veoma bitan i za profesora i za učenike. Učenicima omogućava da procjene koliko su tijekom sata uspjeli razumjeti i savladati novo gradivo, ali i kako bi učenici mogli odgovoriti na pitanja koje je značenje toga što smo učili ili čemu mi to može poslužiti. Nastavniku daje priliku da dobije povratnu informaciju o razini razumijevanja kod učenika.

8.2 Nastavna priprema: Napon i potencijal

U ovom podpoglavlju prikazat ćemo nastavnu pripremu za drugi razred gimnazije u kojoj razrađujemo kako bi trebao izgledati sat na temu napon i potencijal. Ova tema je jedna od konceptualno najtežih tema za učenike za razumjeti i primijeniti.

Iako bi trebalo koristiti pokuse u nastavi ova tema to ne dopušta, zato naglasak stavljamo na primjere koji su učenici već obradili te se stalno pozivamo na staro znanje. Kako bi što bolje razumjeli napon i potencijal koristimo usporedbu sa primjerom tijela koje podižemo u homogenom gravitacijskom polju. Na kraju sata pomoću par konceptualnih pitanja provjeravamo učeničkoj razumijevanje obrađene teme.

Obrazovni ishodi (očekivana učenička postignuća)

Učenici će:

- opisati električnu potencijalnu energiju
- pisati pretvorbe energiju u sustavu nabijene čestice i izvora električnog polja
- objasniti napon
- objasniti potencijal

Odgojni ishodi (koje će vrijednosti učenici usvajati tijekom sata)

Učenici će:

- usvojiti radne navike i pozitivni stav prema radu
- poštivati tuđe mišljenje
- izražavati vlastito mišljenje
- učiti komunicirati

Vrsta nastave: interaktivna istraživački usmjerena nastava

Nastavne metode:

- metoda razgovora - razredna rasprava
- metoda pisanja /crtanja

Oblici rada:

- frontalni
- individualni

Literatura:

- Fizika 2, Udžbenik za 2. razred gimnazije, Rudolf Krsnik, Školska knjiga, Zagreb
- Fizika 2, Udžbenik za 4. razred gimnazije, Vladimir Paar, Školska knjiga, Zagreb

TIJEK NASTAVNOG SAT

Uvodni dio: otvaranje problema, prikupljanje ideja, upoznavanje pojave

UVODNI PROBLEM: U kinematici smo razmatrali probleme na dva načina. Sjećate li se koji su to načini bili?

- očekujem da će učenici odgovoriti kako smo probleme razmatrali preko Newtonovih zakona i preko energija.
- uvodnim pitanjem želim učenike podsjetiti na naučeno gradivo koje ćemo iskoristiti kako bi povukli analogiju između kinematike tijela i gibanja naboja.

Što mislite čime ćemo se danas baviti?

- očekujem da će učenici odgovoriti kako ćemo danas razmatrati energije sustava nabijenih čestica i izvora električnog polja

Koje energije smo do sada promatrali?

- očekujem da će reći da ima kinetičku, elastičnu potencijalnu, gravitacijsku potencijalnu...

Što mislite koje bi od gore navedenih energija mogli razmatrati kod nabijenih čestica?

- očekujem odgovore kinetičku i nekakvu potencijalnu
- potpitanjima ih navesti da zaključe da ako se čestica nalazi u električnom polju tada sustav može imati električnu potencijalnu energiju

Pišem naslov na ploču.

- Električna potencijalna energija

Središnji dio: konstrukcija modela - fizikalni i matematički opis pojave

Idemo prvo pogledati kako se mijenja gravitacijska potencijalna energija pri gibanju tijela u gravitacijskom polju. Sjećate li se kakvo je bilo gravitacijsko polje?

- očekujem da će učenici odgovoriti kako je polje bilo homogeno
- ukoliko se ne sjete potpitanjima ću ih navesti na odgovor

Pretpostavite da stalnom brzinom podižete kutiju mase m na visinu h. Koliko ste rad izvršili na sustav kutija Zemlja?

$$W = m\ g\ h$$

Za koliko se tada promijenila gravitacijska potencijalna energija sustava?

$$\Delta E_p = W$$

ISTRAŽIVAČKO PITANJE: Kako se mijenja električna potencijalna energija pri gibanju naboja u električnom polju?

Ako uzmemo dva naboja, neka pozitivni miruje i neka nam on glumi Zemlju kao u prošlom primjeru. Drugi naboj koji je negativan želim pomaknuti za udaljenost d dok se nalazi u električnom polju prvog naboja. Je li nam ovaj primjer sličan prvom primjeru?

- očekujem kako će učenici reći da ovaj primjer nije sličan prvom primjeru zato jer nemam homogeno električno polje

Gledajući prijašnji primjer kakvo električno polje treba biti u sustavu?

- očekujem da učenici odgovoriti da treba biti homogeno
- ako ne znaju dati odgovor podsjećam ih na gravitacijsko polje koje su ponovili
- učenicima objašnjavam da homogeno električno polje možemo napraviti pomoću suprotno nabijenih ploča
- skiciram na ploči pritom ih pitam za smjer i iznos električnog polja

Pretpostavite sada da stalnom brzinom podižemo česticu pozitivnog naboja q za visinu d u homogenom električnom polju E. Koliki ste rad izvršili na sustav čestica nabijene ploče kada podignete česticu od točke A do točke B?

- tražim od učenika da svatko u bilježnicu izračuna iznos

Što mislite za koliko se promijenila električna potencijalna energija sustava?

$$\Delta E_p = W = E\ q\ d$$

Ako pustite česticu da se giba od točke B do točke A, kako se ona giba što se događa s električnom potencijalnom energijom i kinetičkom energijom sustava? Za koliko se sada promijenila električna potencijalna energija sustava?

- tražim od učenika da zapišu odgovore u bilježnicu
- očekujem da će odgovoriti kako se potencijalna energija promijenila za negativni iznos rada

Ako promotrimo izraz za promjenu električne potencijalne energije ΔE_p koje veličine u tom izrazu predstavljaju svojstva čestice, a koja svojstva prostora?

- očekujem kako će učenici odgovoriti da naboj q je svojstvo čestice, a električno polje E i udaljenost između točaka d svojstvo prostora.

Ako želimo konstruirati fizikalnu veličinu koja opisuje promjenu električne potencijalne energije pri gibanju čestice kroz električno polje, ali takvu da ne ovisi o svojstvima čestica što moramo napraviti izrazu za promjenu potencijalne energije?

- očekujem kako će učenici odgovoriti kako trebamo ΔE_p podijeliti s nabojem q
- $U_{AB} = \frac{\Delta E_p}{q}$

Ima li smisla govoriti o naponu u jednoj točki polja?

- očekujem kako će učenici odgovoriti da nema smisla jer će razlika potencijalne energije biti jednaka iznosu rada

Razmotrimo slijedeći problem. Nacrtajte graf ovisnosti električne sile koja djeluje na probni naboj o udaljenosti od naboja.

- želim učenike navesti da zaključe kako u oba slučaja postoji sličnost. Da električno polje trne na velikim udaljenostima.

A koliki bi rad morao izvršiti kako bi probni naboj iz velike udaljenosti doveo u neku točku bliže naboju?

- očekujem kako će učenici reći da je površina ispod grafa jednaka izvršenom radu

Što ako bi baš željeli govoriti o naponu u jednoj točki polja što bi morali napraviti?

- očekujem da će odgovoriti kako i dalje moramo gledati napon između dviju točaka samo što sada uzimamo jednu referentnu točku koja je jako udaljena i iznos električnog polja je zanemariv (kažemo da je ta točka u beskonačnosti)

Možemo definirati novu veličinu električni potencijal.

- prozivam nekoliko učenika da objasne što njima električni potencijal predstavlja

Što mislite koliku vrijednost možemo odabrati za točku u beskonačnosti?

- očekujem da će učenici odgovoriti da uzimamo da je $V_\infty = 0$ V

Završni dio : primjena modela – korištenje novostečenog znanja u novim situacijama, provjera ostvarenosti obrazovnih ishoda

Konceptualni zadatci: Protumačite iz definicije napona kako će se i za koliko promijeniti električna potencijalna energija sustava čestice i izvora polja, naboja 1 C, u sljedećim slučajevima:

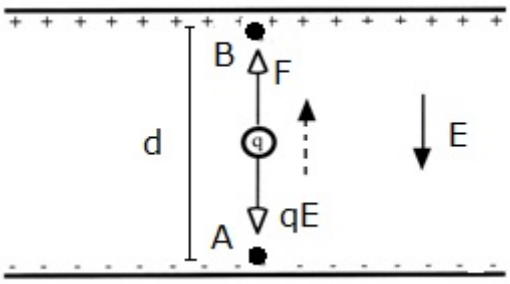
1. pozitivno nabijena čestica prolazi kroz napon $U_{AB} = 1\text{V}$
2. negativno nabijena čestica prolazi kroz napon $U_{AB} = 1\text{V}$
3. pozitivno nabijena čestica prolazi kroz napon $U_{AB} = -1\text{V}$
4. negativno nabijena čestica prolazi kroz napon $U_{AB} = -1\text{V}$

Električni potencijal u točki A električnog polja iznosi 15 V. Koliko bi se promijenila električna potencijalna energija pozitivno nabijene čestice (+ip) naboja 1 C, kad bismo je prenijeli iz beskonačnosti u točku A? Koliki bismo rad nad sustavom morali za to izvršiti?

Električni potencijal u točki B električnog polja iznosi 22 V. Kolika bi bila promjena električne potencijalne energije pozitivno nabijene čestice (+ip) naboja 1 C pri njenom prenošenju iz beskonačnosti u točku B?

Kolika bi bila promjena električne potencijalne energije iste čestice (+ip) pri njenom prenošenju iz točke A u točku B?

Vratimo se ponovno na primjer podizanja pozitivno nabijene čestice u homogenom električnom polju između nabijenih ploča.



1. Koja od točaka A i B ima viši potencijal? Obrazložite.
2. Kad se čestica giba u smjeru polja, giba li se prema višem ili prema nižem potencijalu?
3. Kad se čestica giba suprotno od smjera polja, giba li se prema višem ili prema nižem potencijalu?
4. Elektron i proton unesemo u homogeno električno polje i pustimo. Hoće li se oni spontano gibati prema području višeg ili nižeg potencijala? Obrazložite.
5. Želimo ubrzati elektron i proton u električnom polju. Za koju ćemo česticu upotrijebiti pozitivan, a za koju negativan napon? Čime je određeno je li napon između dviju točaka pozitivan ili negativan?

9 Zaključak

U ovom radu je izloženo korištenje heterogenih računalnih sustava radi ubrzavanja numeričkih metoda za izračune u elektrostatici. Koristili smo CUDA C te TensorFlow platforme. U CUDA C prikazali smo rad s kernelima te ostvarili značajna ubrzanja koristeći grafički procesor, u odnosu na implementaciju koja se izvršavala samo na glavnom računalnom procesoru. Ubrzanje je bilo izraženije za veći broj točaka diskretnog prostora korištenog u metodi relaksacije. Najveće ubrzanje (63x) smo ostvarili za prostor od 642^2 točaka. Upotrebom TensorFlow paketa također smo dobili ubrzanja izračuna no ona su bila značajno manja, te je najveće ubrzanje na grafičkom procesoru bilo dvostruko u odnosu na računalni procesor.

U zadnjem dijelu rada iznijeli smo osnove neuronskih mreža te smo jednu konfiguracija mreža istrenirali na rješenju u diskretnom prostoru s malim brojem točaka, koje smo dobili relaksacijskom metodom. Zatim smo iskoristili analitička svojstva neuronske mreže da bi ju izvrijednili u prostoru s većim brojem točaka te pokazali da je takvo rješenje i dalje zadovoljavajuće.

Bibliography

[1] Griffiths, D.J. Introduction to Electrodynamics. 3rd ed. Prentice Hall, 1999.

[2] Cheng, J.; Grossman, M.; McKercher, T. Professional CUDA C Programming. 1st ed. Wrox Press Ltd., 2014.

[3] Lagaris, I.E.; Likasand, A.; Fotiadis, D.I.; Artifcial Neural Networks for Solving Ordinary and Partial Diferential Equations // IEEE Transactions on Neural Networks. Vol. 9 , 5 (1998), str. 987-1000.

[4] TensorFlow.org, `https://www.tensorflow.org`, 1.12.2017.

[5] CUDA Toolkit Documentation v9.0.176, `https://docs.nvidia.com/cuda/`, 1.12.2017.

[6] Keras: The Python Deep Learning library, `https://keras.io`, 1.12.2017.