Univerzitet u Beogradu - Elektrotehnički fakultet
Multiprocesorski sistemi (13S114MUPS, 13E114MUPS)

Domaći zadatak 4 - CUDA

Izveštaj o urađenom domaćem zadatku

Predmetni saradnici:
doc. dr Marko Mišić
dipl. ing. Pavle Divović

Studenti:
Iva Potkonjak 2019/0301
Lazar Premović 2019/0091

Beograd, januar 2023.

Sadržaj

1 PROBLEM 1 - PRIME

1.1 Tekst problema

Paralelizovati program koji vrši određivanje ukupnog broja prostih brojeva u zadatom opsegu. Prilikom zadavanja izvršne konfiguracije jezgra, koristiti 1D rešetku (grid). Obratiti pažnju na efikasnost paralelizacije i potrebu za redukcijom. Program se nalazi u datoteci **`prime.c`** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u datoteci **`run`**.

1.2 Delovi koje treba paralelizovati

1.2.1 Diskusija

Delovi koda, koje je moguće paralelizovati, predstavljaju dve for petlje, od kojih je jedna ugnježdena i while petlja. Paralelizovana je spoljna for petlja, koja iterira kroz brojeve u zadatom opsegu. Unutrašnja for petlja nije paralelizovana, a njen zadatak je da proveri da li je trenutno posmatrani broj iz opsega deljiv nekim od brojeva manjih od njega. Razlog za ovo je što unutrašnja petlja nije regilarnih dimenzija. While petlja se nalazi u delu koda koji nije eksplicitno deo algoritma, već pripada delu koji se odnosi na testiranje rada programa, zbog ovoga nije rađena paralelizacija.

1.2.2 Način paralelizacije

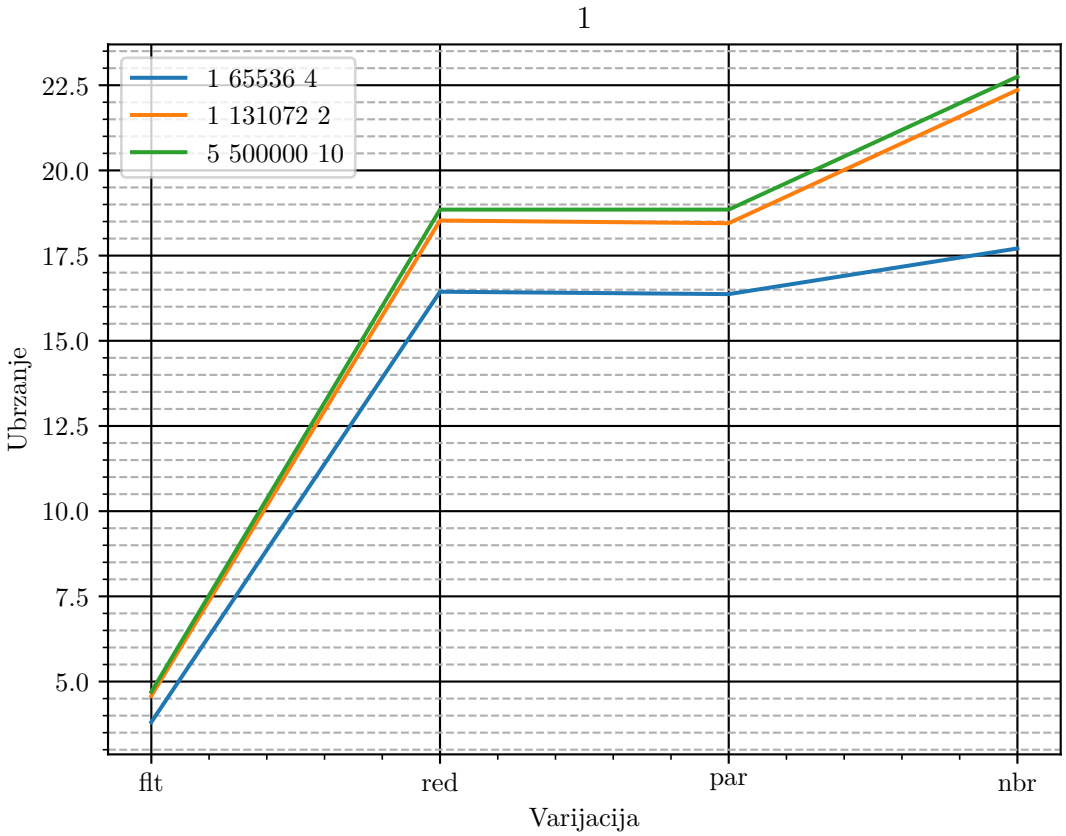
Spoljna petlja je zamenjena 1D rešetkom, gde svaka nit uzima po jedan broj iz intervala i proverava da li je prost. Brojevi se ciklično uzimaju od strane niti, ovo je urađeno kako bi niti koje obrađuju parne brojeve upali u iste blokove. Svaka nit prema indeksu pristupa jednom elementu iz niza u deljenoj memoriji i tamo čuva informaciju o tome da li je njen broj prost ili ne. Na kraju se radi redukcija ovog niza prema optimalnom algoritmu sa vežbi, konačan režultat će se naći na nultom indeksu na kraju redukcije i biće atomično dodat u globalnu memoriju.
Kako bi osigurali da niti koje obrađuju parne brojeve jesu u istim blokovima, broj blokova će uvek biti stepen broja dva veći od jedan, dok je broj niti po bloku stepen broja dva tako da bi se poboljšalo iskorišćenje niti.

1.3 Rezultati

1.3.1 Logovi izvršavanja

```
Running task: dz4z1
Building variant: prime_v0seq
Building variant: prime_v1flt
Building variant: prime_v2red
Building variant: prime_v3par
Building variant: prime_v4nbr
Running variant: prime_v0seq
Running test case: 1 of 3 (1 65536 4) N=1 3 times
    AVG Time: 0.2522
Running test case: 2 of 3 (1 131072 2) N=1 3 times
    AVG Time: 0.9548
Running test case: 3 of 3 (5 500000 10) N=1 3 times
    AVG Time: 12.2478
Running variant: prime_v1flt
Running test case: 1 of 3 (1 65536 4) N=1 3 times
    AVG Time: 0.0662 AVG Speedup: 3.81 ALL Tests PASSED
Running test case: 2 of 3 (1 131072 2) N=1 3 times
    AVG Time: 0.2085 AVG Speedup: 4.58 ALL Tests PASSED
Running test case: 3 of 3 (5 500000 10) N=1 3 times
    AVG Time: 2.6036 AVG Speedup: 4.70 ALL Tests PASSED
Running variant: prime_v2red
Running test case: 1 of 3 (1 65536 4) N=1 3 times
    AVG Time: 0.0153 AVG Speedup: 16.44 ALL Tests PASSED
Running test case: 2 of 3 (1 131072 2) N=1 3 times
    AVG Time: 0.0515 AVG Speedup: 18.53 ALL Tests PASSED
Running test case: 3 of 3 (5 500000 10) N=1 3 times
    AVG Time: 0.6497 AVG Speedup: 18.85 ALL Tests PASSED
Running variant: prime_v3par
Running test case: 1 of 3 (1 65536 4) N=1 3 times
    AVG Time: 0.0154 AVG Speedup: 16.37 ALL Tests PASSED
Running test case: 2 of 3 (1 131072 2) N=1 3 times
    AVG Time: 0.0517 AVG Speedup: 18.45 ALL Tests PASSED
Running test case: 3 of 3 (5 500000 10) N=1 3 times
    AVG Time: 0.6499 AVG Speedup: 18.85 ALL Tests PASSED
Running variant: prime_v4nbr
Running test case: 1 of 3 (1 65536 4) N=1 3 times
    AVG Time: 0.0142 AVG Speedup: 17.71 ALL Tests PASSED
Running test case: 2 of 3 (1 131072 2) N=1 3 times
    AVG Time: 0.0427 AVG Speedup: 22.36 ALL Tests PASSED
Running test case: 3 of 3 (5 500000 10) N=1 3 times
    AVG Time: 0.5383 AVG Speedup: 22.75 ALL Tests PASSED
Total time taken: 2.6456
```

1.3.2 Grafici ubrzanja



Slika 1: Grafik zavisnosti ubrzanja od varijacije za različite test primere

1.3.3 Diskusija dobijenih rezultata

Najbolje performanse upravo daje ciklična podela po nitima, koja je rađena prema formuli `threadIdx.x * gridDim.x + blockIdx.x`.
Takođe rađeno je poređenje redukcije izmedju blokova na GPU i na CPU i dobijene su praktično iste vrednosti ubrzanja.

2 PROBLEM 2 - FEYMAN

2.1 Tekst problema

Paralelizovati program koji vrši izračunavanje 3D Poasonove jednačine korišćenjem Feyman-Kac algoritma. Algoritam stohastički računa rešenje parcijalne diferencijalne jednačine krenuvši N puta iz različitih tačaka domena. Tačke se kreću po nasumičnim putanjama i prilikom izlaska iz granica domena kretanje se zaustavlja računajući dužinu puta do izlaska. Proces se ponavlja za svih N tačaka i konačno aproksimira rešenje jednačine. Program se nalazi u datoteci **`feyman.c`** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u datoteci **`run`**.

2.2 Delovi koje treba paralelizovati

2.2.1 Diskusija

Deo programa, koji je moguće paralelizovati predstavljaju tri for petlje. Takođe, postoji i četvrta ugnježdena for petlja, koja je ovog puta paralelizovana.

2.2.2 Način paralelizacije

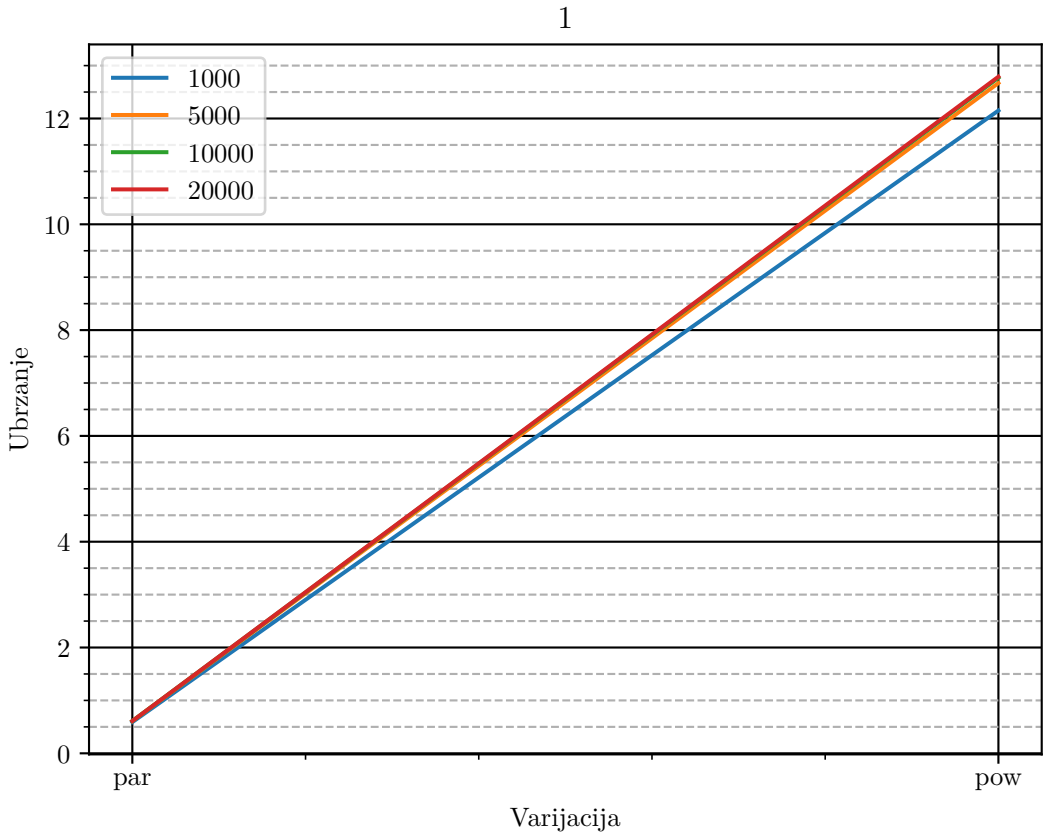
Prve tri petlje menja 3D rešetka, gde svaki blok uzima i obrađuje po jednu iteraciju ove tri petlje, a niti iz bloka obrađuju po jednu iteraciju četvrte petlje nezavisno. Naredba continue je zamenjena if-om u koji se ulazi samo ako bi se preskočila naredba continue u sekvencijalnom kodu. Što se tiče promenljive seed ona je na generisana za svaki nit u odnosu na id bloka u kom se nalazi i njen id unutar bloka. Takođe, za promenljivu wt potrebno je alocirati niz u deljenoj memoriji, gde ce svaka nit upisivati svoju vrednost wt u odgovarajući element ovog niza. Niz će na kraju biti redukovan optimalnim algoritmom i konačan rezultat će bit upisa atomično u globalnu memoriju od strane jedne niti.

2.3 Rezultati

2.3.1 Logovi izvršavanja

```
Running task: dz4z2
Building variant: feyman_v0seq
Building variant: feyman_v1par
Building variant: feyman_v2pow
Running variant: feyman_v0seq
Running test case: 1 of 4 (1000) N=1 3 times
    AVG Time: 3.0034
Running test case: 2 of 4 (5000) N=1 3 times
    AVG Time: 15.0578
Running test case: 3 of 4 (10000) N=1 3 times
    AVG Time: 30.1115
Running test case: 4 of 4 (20000) N=1 3 times
    AVG Time: 60.1816
Running variant: feyman_v1par
Running test case: 1 of 4 (1000) N=1 3 times
    AVG Time: 5.0807 AVG Speedup: 0.59 ALL Tests PASSED
Running test case: 2 of 4 (5000) N=1 3 times
    AVG Time: 24.7715 AVG Speedup: 0.61 ALL Tests PASSED
Running test case: 3 of 4 (10000) N=1 3 times
    AVG Time: 49.1987 AVG Speedup: 0.61 ALL Tests PASSED
Running test case: 4 of 4 (20000) N=1 3 times
    AVG Time: 98.3862 AVG Speedup: 0.61 ALL Tests PASSED
Running variant: feyman_v2pow
Running test case: 1 of 4 (1000) N=1 3 times
    AVG Time: 0.2472 AVG Speedup: 12.15 ALL Tests PASSED
Running test case: 2 of 4 (5000) N=1 3 times
    AVG Time: 1.1883 AVG Speedup: 12.67 ALL Tests PASSED
Running test case: 3 of 4 (10000) N=1 3 times
    AVG Time: 2.3577 AVG Speedup: 12.77 ALL Tests PASSED
Running test case: 4 of 4 (20000) N=1 3 times
    AVG Time: 4.7067 AVG Speedup: 12.79 ALL Tests PASSED
Total time taken: 1.7244
```

2.3.2 Grafici ubrzanja



Slika 2: Grafik zavisnosti ubrzanja od varijacije za različite test primere

2.3.3 Diskusija dobijenih rezultata

Na grafiku vidimo dve varijante paralelnog rešenja, gde je u prvoj korišćena pow funkcija za stepenovanje brojeva, dok je u drugoj ovo zamenjeno sa običnim množenjem jer su svi stepeni u programu zapravo kvadrati broja. Možemo videti da ova zamena daje značajno ubrzanje programa.

3 PROBLEM 3 - MOLDYN

3.1 Tekst problema

Paralelizovati jednostavan program koji se bavi molekularnom dinamikom. Kod predstavlja simulaciju molekularne dinamike argonovog atoma u ograničenom prozoru (prostoru) sa periodičnim graničnim uslovima. Atomi se inicijalno nalaze raspoređeni u pravilnu mrežu, a zatim se tokom simulacije dešavaju interakcije između njih. U svakom koraku simulacije u glavnoj petlji se dešava sledeće:

- Čestice (atomi) se pomeraju zavisno od njihovih brzina i brzine se parcijalno ažuriraju u pozivu funkcije `domove`.
- Sile koje se primenjuju na nove pozicije čestica se izračunavaju; takođe, akumuliraju se prosečna kinetička energija (*virial*) i potencijalna energija u pozivu funkcije `forces`.
- Sile se skaliraju, završava ažuriranje brzine i izračunavanje kinetičke energije u pozivu funkcije `mkekin`.
- Prosečna brzina čestice se računa i skaliraju temperature u pozivu funkcije `velavg`.
- Pune potencijalne i prosečne kinetičke energije (*virial*) se računaju i ispisuju u funkciji `prnout`.

Program se nalazi u direktorijumu `MolDyn` u arhivi koja je priložena uz ovaj dokument. Program se sastoji od više datoteka, od kojih su od interesa datoteke `main.c` i `forces.c`, jer se u njima provodi najviše vremena. Analizirati dati kod i obratiti pažnju na redukcione promenljive unutar datoteke `forces.c`.

3.2 Delovi koje treba paralelizovati

3.2.1 Diskusija

U ovom zadatku za paralelizaciju su posmatrane samo funkcije main i forces. U main funkciji primećena je jedna for petlja, koja se ne može paralelizovati zbog zavisnosti po podacima. U forces funkciji primećene su dve ugnježdene for petlje. Kombinacija dve petlje nije moguća, pa je doneta oodluka da se paralelizuje samo spoljna petlja. Izabrano je da funkcija forces predstavlja kernel.

3.2.2 Način paralelizacije

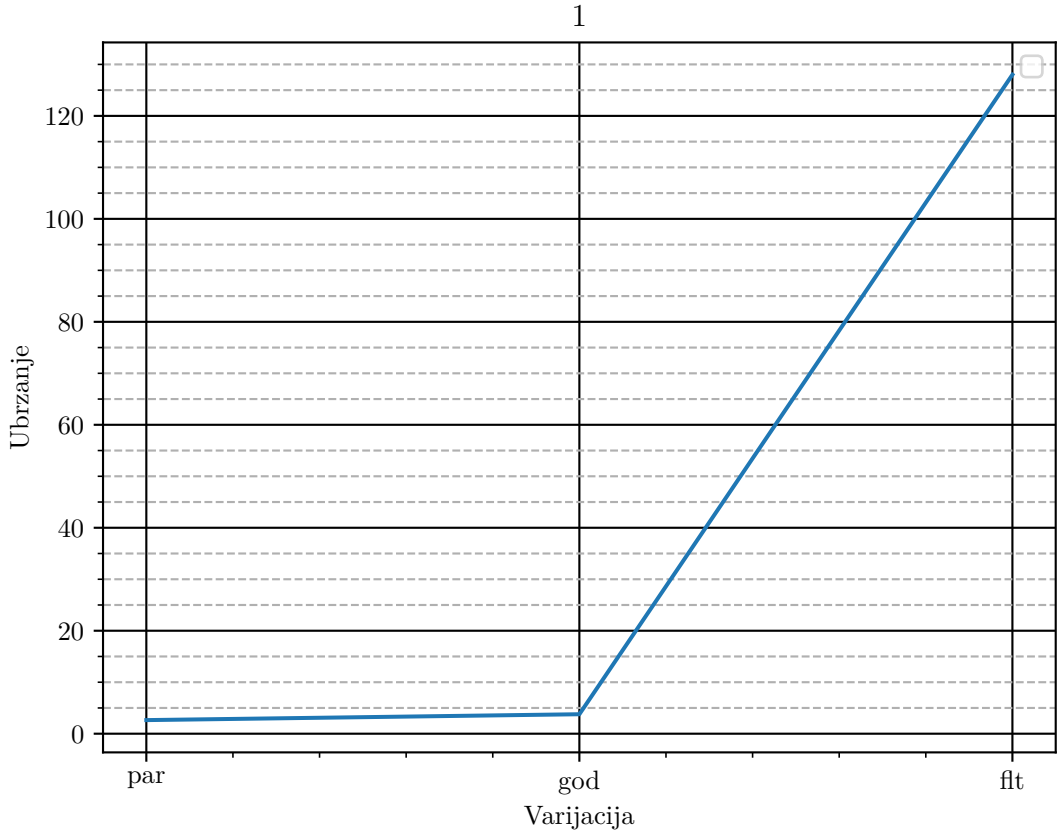
For petlja je zamenjena 1D rešetkom, gde jedna nit uzima jednu iteraciju prethodne for petlje. Za korektonst izvršavanja potrebno je obratiti pažnju na niz f i promenljive vir i epot. Pri modifikaciji elemenata niza f je korišćena funkcija atomicAdd. Za promenljive vir i epot su alocirani nizovi u deljenoj memoriji, gde svaka svaka nit čuva izračunate vrednosti vir i epot u odgovarajućem elementu ovih nizova. Na kraju se radi redukcija ovih nizova i jedna nit prepisuje konačni rezultat u globalnu memoriju.

3.3 Rezultati

3.3.1 Logovi izvršavanja

```
Running task: dz4z3
Building variant: md_v0seq
Building variant: md_v1par
Building variant: md_v2god
Building variant: md_v3flt
Running variant: md_v0seq
Running test case: 1 of 1 () N=1 3 times
    AVG Time: 5.8454
Running variant: md_v1par
Running test case: 1 of 1 () N=1 3 times
    AVG Time: 2.2163 AVG Speedup: 2.64 ALL Tests PASSED
Running variant: md_v2god
Running test case: 1 of 1 () N=1 3 times
    AVG Time: 1.5472 AVG Speedup: 3.78 ALL Tests PASSED
Running variant: md_v3flt
Running test case: 1 of 1 () N=1 3 times
    AVG Time: 0.0457 AVG Speedup: 128.00 ALL Tests PASSED
Total time taken: 2.5011
```

3.3.2 Grafici ubrzanja



Slika 3: Grafik zavisnosti ubrzanja od varijacije

3.3.3 Diskusija dobijenih rezultata

Glavno ograničenje posmatranog problema je veličina niza f, koji je potreban svakom od blokova, a nije moguće alocirati potrebnu količinu deljene memorije za njega. Iz ovog razloga pokušana su dva pristupa. Prvi pristup je da svaka nit alocira f lokalno i na kraju obrade ažurira globalno f, a drugi jeste da se koriste atomične operacije nad nizom f. Drugi pristup u velikom broju slučajeva rezultuje u manjem broju pristupa globalnoj memoriji, što je i pokazano na grafiku u vidu većeg ubrzanja.
U trećoj varijanti za neke promenljive je tip sa double promenjen na float, što je dalo značajno ubrzanje, što je posledica arhitekture grafičkog procesora.