UNIVERZITET U BEOGRADU - ELEKTROTEHNIČKI FAKULTET

MULTIPROCESORKI SISTEMI (13S114MUPS, 13E114MUPS)

DOMAĆI ZADATAK 4 – CUDA

Izveštaj o urađenom domaćem zadatku

Predmetni saradnici:

doc. dr Marko Mišić

dipl. ing. Matija Dodović

Studenti:

Petar Marković 2020/0203

Lana Stamenković 2020/0206

Beograd, jun 2024.

SADRŽAJ

1. PROBLEM 1 – ARITMETIČKI BROJEVI

1.1. Tekst problema

Paralelizovati program koji vrši izračunavanje aritmetičkih brojeva. Pozitivan ceo broj je aritmetički ako je prosek njegovih pozitivnih delilaca takođe ceo broj. Program se nalazi u datoteci **aritmetic.c** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u **run** skripti.

Prilikom zadavanja izvršne konfiguracije jezgra, koristiti 1D rešetku (grid).

1.2. Delovi koje treba paralelizovati

1.2.1. Diskusija

U priloženom sekvencijalnom kodu za izračunavanje aritmetičkih brojeva uočili smo dve celine koje je potencijalno moguće paralelizovati. Jedna celina je petlja u **main** funkciji programa koja iterira kroz sve prirodne brojeve, proverava da li je dati broj aritmetički i staje tek kad je nađen zadati broj aritmetičkih brojeva. Druga celina u kodu koja bi mogla biti paralelizovana jesu **for** petlje u funkciji **divisor_count_and_sum()** koje traže proste činioce zadatog broja.

1.2.2. Zadatak 1 - rešenje

U poređenju sa implementacijama u prethodnim tehnologijama, CUDA implementacija datog problema je znatno konceptualno jednostavnija. Ideja je da se svakoj niti GPU-a dodeli ispitivanje jednog broja u intervalu od 1 do broja zadatog kao argument programa (dati broj predstavlja redni broj po aritmetičkog broja kog treba naći) pomnoženog sa faktorom 1,25. Ovaj faktor je izabran tako da se pomoću GPU-a uvek nađe više aritmetičkih brojeva od zadatog broja. Nakon kraja rada GPU-a u procesor se kopira bafer čiji elementi predstavljaju brojeve aritmetičkih brojeva nađenih u odgovarajućem bloku niti. Procesor agregira te sume u konačni broj aritmetičkih brojeva do trenutka kad je razlika izmedju nađenog broja i traženog broja aritmetičkih brojeva manja od veličine jednog bloka niti. Nakon toga se ispitivanje aritmetičkih brojeva vrši sekvencijalno pomoću procesora sve dok se ne nađe traženi broj.

1.3. Rezultati – Zadatak 1

U okviru ove sekcije su izloženi rezultati paralelizacije koda za traženje aritmetičkih brojeva pomoću 1D rešetke.

1.3.1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za sekvencijalno izvršavanje i izvršavanje nakon paralelizacije.

```
10001th arithmetic number is 12955
Number of composite arithmetic numbers <= 12955: 8459

real    0m0.008s
user    0m0.002s
sys     0m0.006s

100001th arithmetic number is 125589
Number of composite arithmetic numbers <= 125589: 88220

real    0m0.017s
user    0m0.017s
sys     0m0.000s

1000001th arithmetic number is 1228665
Number of composite arithmetic numbers <= 1228665: 905044

real    0m0.184s
user    0m0.183s
sys     0m0.001s

10000001th arithmetic number is 12088245
Number of composite arithmetic numbers <= 12088245: 9206548

real    0m4.199s
user    0m4.195s
sys     0m0.004s
```

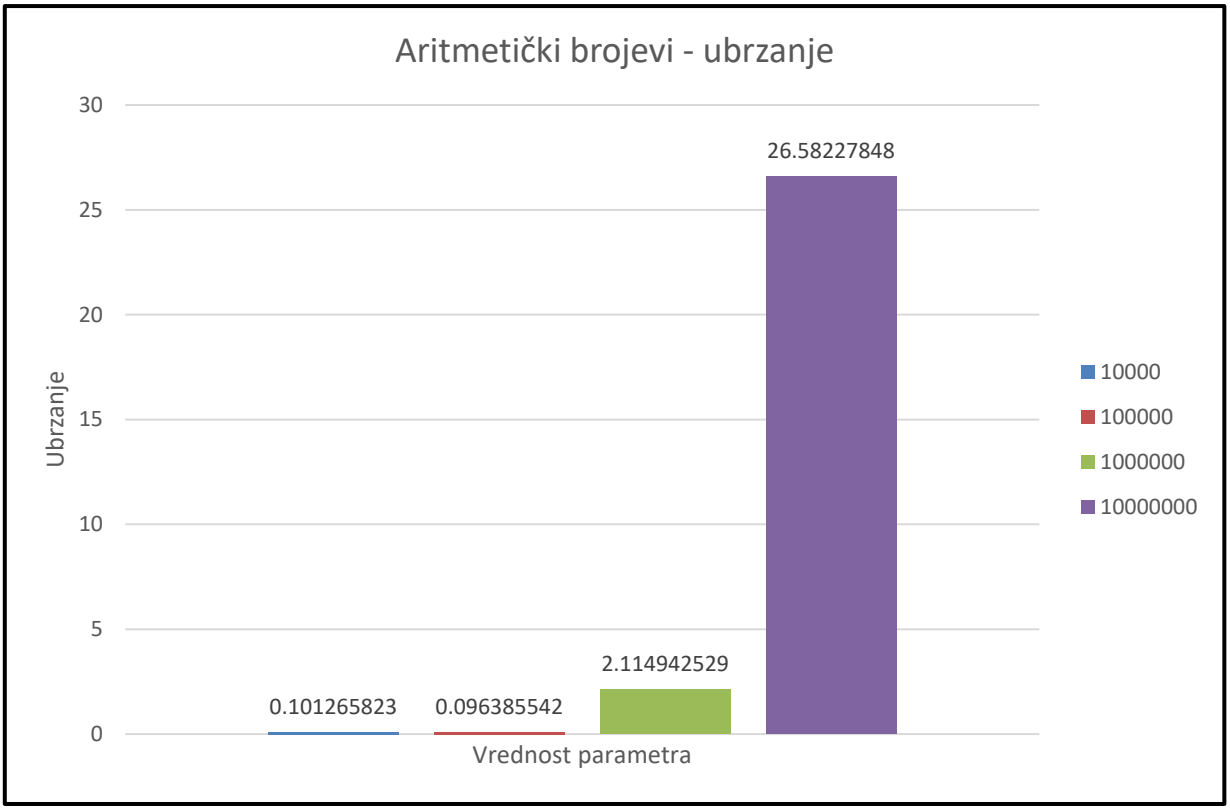
Listing 1. Sekvencijalno izvršavanje

```
10001th arithmetic number is 12955
Number of composite arithmetic numbers <= 12955: 8459

real    0m0.079s
user    0m0.008s
sys     0m0.051s

100001th arithmetic number is 125589
Number of composite arithmetic numbers <= 125589: 88220

real    0m0.083s
user    0m0.012s
sys     0m0.048s

1000001th arithmetic number is 1228665
Number of composite arithmetic numbers <= 1228665: 905044

real    0m0.087s
user    0m0.010s
sys     0m0.056s

10000001th arithmetic number is 12088245
Number of composite arithmetic numbers <= 12088245: 9206548

real    0m0.158s
user    0m0.091s
sys     0m0.050s
```

Listing 2. Izvršavanje nakon paralelizacije

1.3.2. Grafik ubrzanja

U okviru ove sekcije je dat grafik ubrzanja u odnosu na sekvencijalnu implementaciju.



Slika 1. Aritmetički brojevi - ubrzanje

1.3.3. Diskusija dobijenih rezultata

Ubrzanje je postignuto za veće dimenzije problema. Za najveći ulazni podatak, dobijeno je odlično ubrzanje. Postignut je dobar balans kod podele posla između GPU-a i CPU-a.

2. PROBLEM 2 – HALTONQMC

2.1. Tekst problema

2.1.1. Zadatak 2

Paralelizovati program koji vrši generisanje elemenata Halton Quasi Monte Carlo (QMC) sekvence. Program se nalazi u datoteci **halton.c** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u **run** skripti.

2.2. Delovi koje treba paralelizovati

2.2.1. Diskusija

HaltonQMC je znatno lakši za paralelizaciju od problema izračunavanja aritmetičkih brojeva, pre svega zbog postojanja pravilnih pravougaonih petlji. Raspodela posla po iteracijama nije jednaka, međutim, u zavisnosti od ulaznih podataka, zahtevnije iteracije se mogu naći i na početku i na kraju petlje.

2.2.2. Zadatak 2 - rešenje

Radi postizanja što boljih performansi, prilikom izrade rešenja ovog zadatka težilo se upotrebi što manjeg broja naredbi za alociranje i kopiranje podataka između procesora i grafičkog procesora. Konačno rešenje ima samo jednu naredbu alokacije i jednu naredbu kopiranja. Na početku izvršavanja programa se na GPU-u alocira dovoljan prostor za sva ponavljanja generisanja Halton sekvence. Funkcija za generisanje halton sekvence se izmeni dodavanjem boolean promenljive koja utiče na to da li će na strani procesora biti alociran niz za rezultat, kao i da li će doći do kopiranja rezultata. Testiranje programa ne zahteva vraćanje rezultata svaki put prilikom generisanja Halton sekvence, zbog čega se mogu postići poprilična ubrzanja jer upravo alokacija i kopiranje rezultata predstavljaju usko grlo. Što se samog posla koji se izvršava na grafičkom procesoru tiče, jedna nit dobija jedan element Halton sekvence na izračunavanje.

2.3. Rezultati – Zadatak 2

U okviru ove sekcije su izloženi rezultati paralelizacije koda za izračunavanja elemenata Haltonove sekvence.

2.3.1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za sekvencijalno izvršavanje i izvršavanje nakon paralelizacije.

```
HALTON_TEST:

HALTON_SEQUENCE_TEST
  HALTON_SEQUENCE returns the elements I1 through I2
  of an M-dimensional Halton sequence.

  R:
  Col:          0             1             2             3             4
  Row
    0:          0.3125          0.5625          0.0625          0.875           0.375
...
    8:        0.434783        0.391304        0.347826        0.304348         0.26087
    9:        0.344828        0.310345        0.275862        0.241379        0.206897

  Normal end of execution.

  real    0m0.007s
  user    0m0.007s
  sys     0m0.000s

HALTON_TEST:

HALTON_SEQUENCE_TEST
  HALTON_SEQUENCE returns the elements I1 through I2
  of an M-dimensional Halton sequence.

  R:
  Col:          0             1             2             3             4
  Row
    0:          0.3125          0.5625          0.0625          0.875           0.375
...
   98:       0.0191205       0.0172084       0.0152964       0.0133843       0.0114723
   99:       0.0184843       0.0166359       0.0147874        0.012939       0.0110906
```

```
  Normal end of execution.

  real    0m0.025s
  user    0m0.021s
  sys     0m0.004s

HALTON_TEST:

HALTON_SEQUENCE_TEST
  HALTON_SEQUENCE returns the elements I1 through I2
  of an M-dimensional Halton sequence.

  R:
  Col:          0             1             2             3             4
  Row
    0:          0.3125          0.5625          0.0625          0.875           0.375
    ...
  998:       0.0012647      0.00113823      0.00101176     0.000885292     0.000758821
  999:      0.00126279      0.00113651      0.00101023      0.00088395     0.000757671

  Normal end of execution.

  real    0m2.483s
  user    0m2.479s
  sys     0m0.004s
```

Listing1. Sekvencijalno izvršavanje

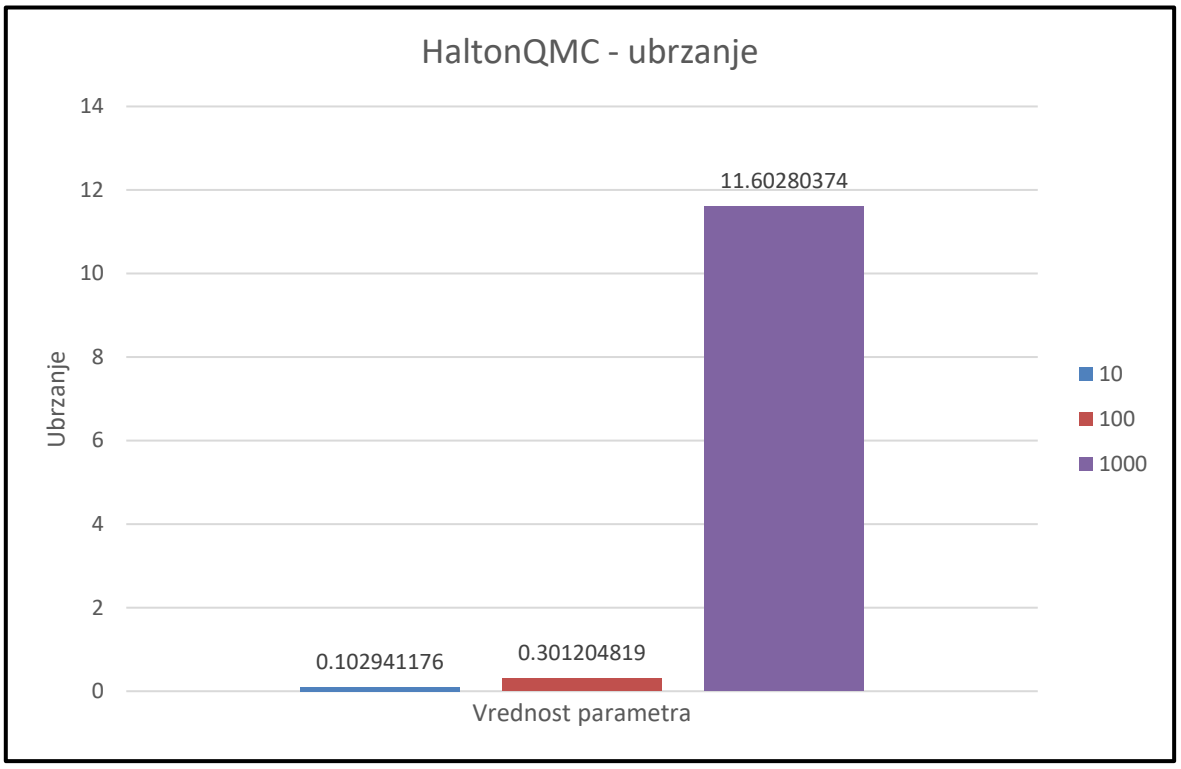
```
HALTON_TEST:

HALTON_SEQUENCE_TEST
  HALTON_SEQUENCE returns the elements I1 through I2
  of an M-dimensional Halton sequence.

  R:
  Col:          0             1             2             3             4
  Row
    0:          0.3125          0.5625          0.0625          0.875           0.375
...
    8:        0.434783        0.391304        0.347826        0.304348         0.26087
    9:        0.344828        0.310345        0.275862        0.241379        0.206897

  Normal end of execution.
```

```
  real    0m0.068s
  user    0m0.009s
  sys     0m0.052s

HALTON_TEST:

HALTON_SEQUENCE_TEST
  HALTON_SEQUENCE returns the elements I1 through I2
  of an M-dimensional Halton sequence.

  R:
  Col:          0             1             2             3             4
  Row
    0:          0.3125          0.5625          0.0625          0.875           0.375
...
   98:       0.0191205       0.0172084       0.0152964       0.0133843       0.0114723
   99:       0.0184843       0.0166359       0.0147874        0.012939       0.0110906

  Normal end of execution.

  real    0m0.083s
  user    0m0.025s
  sys     0m0.056s

HALTON_TEST:

HALTON_SEQUENCE_TEST
  HALTON_SEQUENCE returns the elements I1 through I2
  of an M-dimensional Halton sequence.

  R:
  Col:          0             1             2             3             4
  Row
    0:          0.3125          0.5625          0.0625          0.875           0.375
    ...
  998:       0.0012647      0.00113823      0.00101176     0.000885292     0.000758821
  999:      0.00126279      0.00113651      0.00101023      0.00088395     0.000757671

  Normal end of execution.

  real    0m0.214s
  user    0m0.161s
  sys     0m0.048s
```

Listing2. Izvršavanje nakon paralelizacije

2.3.2. Grafik ubrzanja

U okviru ove sekcije je dat grafik ubrzanja u odnosu na sekvencijalnu implementaciju.



Slika 1. HaltonQMC - ubrzanje

2.3.3. Diskusija dobijenih rezultata

Kao i u prethodnom zadatku, ubrzanje se postiže za najveće ulazne podatke. Bitno je napomenuti da korišćenje CUDA okruženja sa sobom povlači izvesne vremenske troškove koji su konstantni. Sekvencijalna implementacija ovog programa se izvršava za veoma kratko vreme koje je uvek manje od vremena inicijalizacije CUDA okruženja zbog čega nije moguće postići ubrzanje za ove dimenzije problema.

3. PROBLEM 3 – N BODY PROBLEM

3.1. Tekst problema

3.1.1. Zadatak 3

Paralelizovati program koji se bavi problemom n tela (n-body problem). Sva tela imaju jediničnu masu, trokomponentni vektor položaja (x, y, z) i trokomponentni vektor brzine (vx, vy, vz). Simulaciju n tela se odvija u iteracijama, pri čemu se u svakoj iteraciji izračunava sila kojom sva tela deluju na sva ostala, a zatim se brzine i koordinate tela ažuriraju prema II Njutnovom zakonu. Brzine i položaji su slučajno generisani na početku simulacije. Zbog same prirode numeričke simulacije uveden je parametar SOFTENING, koji predstavlja korektivni faktor prilikom izračunavanja rastojanja između čestica (kako je gravitaciona sila obrnuto proporcionalna rastojanju između čestica, za nulta rastojanja i rastojanja bliska nuli, izračunata gravitaciona sila postaje izuzetno velika – teži beskonačnosti).

Program se nalazi u datoteci direktorijumu **nbodymini** u arhivi koja je priložena uz ovaj dokument. Program koji treba paralelizovati nalazi se u datoteci **nbody.c**. Pored samog izračunavanja, program čuva rezultate svake iteracije u zasebnim datotekama (za svako telo se čuvaju pozicije i brzine), dok kod **show_nbody.py** kreira gif same simulacije.

Skripta run pokreće simulaciju za različite parametre, i nakon toga, za određene simulacije poziva python kod koji kreira gifove.

3.2. Delovi koje treba paralelizovati

3.2.1. Diskusija

N Body Problem u svakoj iteraciji glavne petlje u main funkciji ažurira brzine tela na osnovu njihovog položaja u odnosu na druga tela. Zatim, položaji i brzine se sačuvaju u .csv fajl, nakon čega se ažuriraju položaji tela za narednu iteraciju. Zbog zavisnosti po podacima između iteracija koje je potrebno zabeležiti, nije moguće raspodeliti iteracije simulacije po nitima. Dodatan problem su upisi u fajlove u svakoj iteraciji simulacije, koje nije moguće paralelizovati. Sa druge strane, moguće je raspodeliti ažuriranje brzine i položaja tela između niti, tako da svaka nit dobije svoju grupu tela sa kojom radi.

3.2.2. Zadatak 3 – rešenje

Ovaj problem se pokazao znatno težim za paralelizaciju pomoću CUDA tehnologije. Izabrano je rešenje gde je jedino paralelizovana petlja u kojoj se vrši izračunavanje sila koje deluju na tela. Podaci o telima se čuvaju u globalnoj memoriji GPU-a, a svaka nit dobije na izračunavanje sile pojedinačnog tela.

3.3. Rezultati – Zadatak 3

U okviru ove sekcije su izloženi rezultati paralelizacije koda za izračunavanja problema n tela.

3.3.1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za sekvencijalno izvršavanje i izvršavanje nakon paralelizacije.

```
Running ./nbody with 30 particles, 100 iterations, and saving to simulation_1 folder

real    0m0.012s
user    0m0.007s
sys     0m0.005s


Running ./nbody with 30 particles, 1000 iterations, and saving to simulation_2 folder

real    0m0.065s
user    0m0.053s
sys     0m0.013s


Running ./nbody with 3000 particles, 100 iterations, and saving to simulation_3 folder

real    0m2.403s
user    0m2.383s
sys     0m0.020s


Running ./nbody with 3000 particles, 1000 iterations, and saving to simulation_4 folder

real    0m24.402s
user    0m24.185s
sys     0m0.160s
```

Listing 1. Zadatak 3– Sekvencijalno izvršavanje

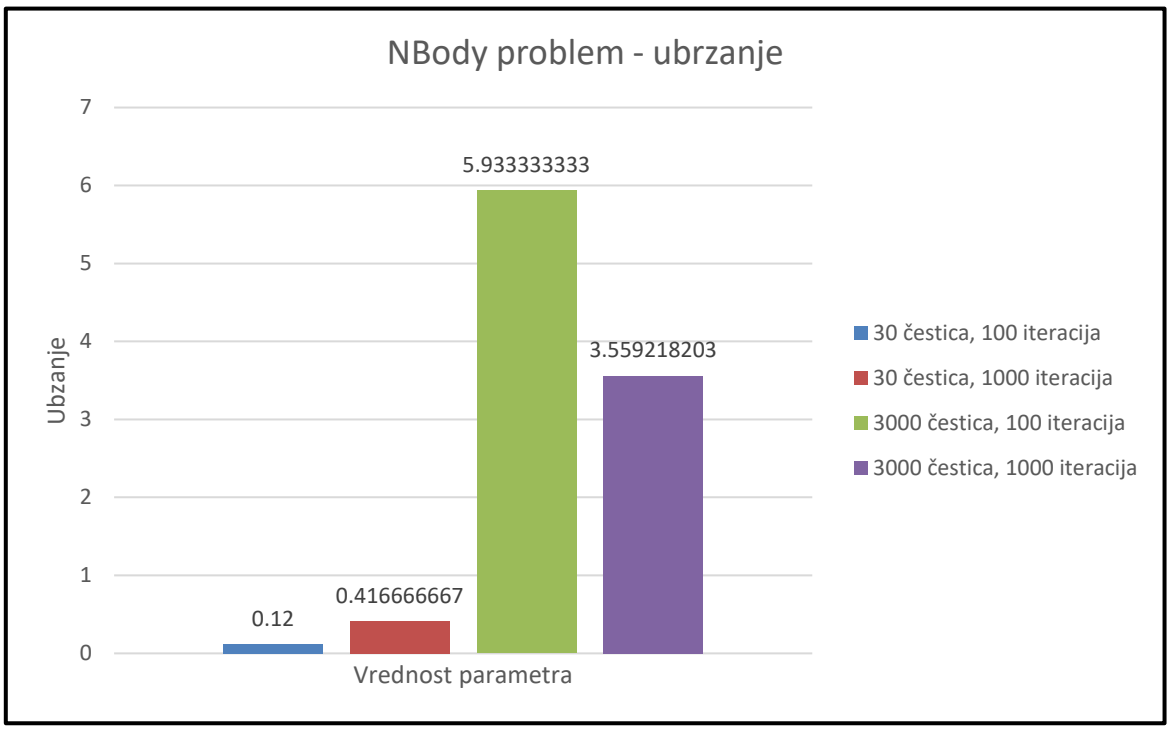
```
Running ./nbody with 30 particles, 100 iterations, and saving to simulation_1 folder

real    0m0.100s
user    0m0.021s
sys     0m0.058s


Running ./nbody with 30 particles, 1000 iterations, and saving to simulation_2 folder
```

```
real    0m0.156s
user    0m0.052s
sys     0m0.078s


Running ./nbody with 3000 particles, 100 iterations, and saving to simulation_3 folder

real    0m0.405s
user    0m0.310s
sys     0m0.076s


Running ./nbody with 3000 particles, 1000 iterations, and saving to simulation_4 folder

real    0m6.856s
user    0m3.281s
sys     0m0.212s
```

Listing 2. Zadatak 3 – Izvršavanje nakon paralelizacije

3.3.2. Grafik ubrzanja

U okviru ove sekcije je dat grafik ubrzanja u odnosu na sekvencijalnu implementaciju



.

Slika 1. Nbody problem - ubrzanje

3.3.3. Diskusija dobijenih rezultata

Ubrzanje je ponovo postignuto za veće ulazne podatke. Treba napomenuti kao značajno usko grlo činjenicu da se podaci u svakoj iteraciji simulacije kopiraju dva puta, jednom sa grafičkog procesora na glavni procesor, kako bi se sačuvali podaci, i drugi put, u suprotnom smeru, kako bi se vratili ažurni podaci za dalje izračunavanje. Zbog velikog broja ovih poziva, dolazi do značajnog pada u performansama sa većim povećanjem iteracija simulacije.