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. Pavle Divović

Studenti:

Matija Dodović 2018/0072

Miloš Milošević 2018/0445

Beograd, maj 2022.

SADRŽAJ

1. PROBLEM 1 - SIMPLEX

1.1. Tekst problema

Paralelizovati program koji računa integral funkcije `F` na osnovu unutrašnjosti *simplex*-a (https://en.wikipedia.org/wiki/Simplex) u 20 dimenzija korišćenjem Monte Carlo metode. U izvornom kodu data je matrica eksponenata jednačine i ivica *simplex*-a. Ulazni parametar programa je broj iteracija aproksimacije. Prilikom paralelizacije nije dozvoljeno koristiti direktive za podelu posla (*worksharing* direktive), već je iteracije petlje koja se paralelizuje potrebno raspodeliti ručno. Obratiti pažnju na ispravno deklarisanje svih promenljivih prilikom paralelizacije. Program testirati sa parametrima koji su dati u datoteci `run`.

1.2. Delovi koje treba paralelizovati

1.2.1. Diskusija

While petlja koja prolazi kroz sve iteracije (lon(n) iteracija) invocira izračunavanje u kome se računa ne može da se paralelizuje zato što je rezultat jedne iteracije potreban za sledeću iteraciju. *simplex_sample* funkcija vrši izračunavanja koristeći neki od algoritama za generisanje radnom brojeva, koji u osnovi ima *seed*. Vrednost *seed*-a se menja za svako sledeće izračunavanje u zavisnosti od prethodne vrednosti, tako da ništa vezano za random izračunavanje ne može da se paralelizuje. Ono što može da se paralelizuje jeste dvostruka for petlja, u funkciji *simplex_unit_to_general*, koja na određeni način kopira vrednosti (uz dodatna izračunavanja) iz jedne matrice u drugu. Sve ostalo u *simplex_sample* funkciji, u proizvoljnoj dubini poziva funckija, što može da se paralelizuje što može da se paralelizuje daje lošije rezultate usled malog i jednostavnog posla koju bi svaka nit izvršavala, dok su režijski troškovi konstantni. Rezultati *simplex_sample* funckije se koriste za izračunavanje krajnjeg rezultata za svaku iteraciju.

Usled specifičnosti problema uočava se da krajnji rezultat programa, koji se u ovoj konstelaciji izračunava u svakoj iteraciji, i u svakom trenutku za određeni broj iteracija predstavlja konačno rešenje, može se izdvojiti. Izračunavanje koje je u osnovi algoritam za generisanje random brojeva, ostaje u petlji koja će sad da radi duplo manje iteracija. Nakon te petlje se pokreće finalni algoritam za random brojeve i pokreće se dvostruka petlja za generisanje rezultata. Samo restruktuiranje koda donosi poboljšanje.

1.2.2. Način paralelizacije

Paralelizacija se oslanja na samu činjenicu da GPU kreira veliki broj niti koji će odrađivati određeni unit-of-work. Na sam GPU se kopiraju nizovi potrebni za izračunavanje, da bi se nakon pokretanja i izvršavanja kompletnog programa predviđenog za GPU, rezultat prenese na CPU. Posao na GPU se sastoji iz ažurianja niza *phy*, na osnovu prenetih podataka. Na GPU se nalaze kompletni podaci potrebni za ažuriranje niza *phy*. Proces paralelizacije CUDA framework-om se sastoji u podeli celog posla na više unit-of-work-ova, koji je u ovom slučaju jedan element niza *phy*, određen sa phy[**point** * m + **dim**], gde je m dužina niza. Konkretan posao se dobija tako što se kreiraju niti u 2D rešetci:

(point, dim) = (blockIdx.x * blockDim.x + threadIdx.x, blockIdx.y * blockDim.y + threadIdx.y)

i svaka nit obrađuje 1 unit-of-work.

1.3. Rezultati

Logovi vremea i grafik ubrzanja su u nastavku.

1.3.1. Logovi izvršavanja

Broj iteracija: 50000

```
[modified_sequential] Elapsed (s): 0.706
[modifiec_omp] Elapsed (s): 0.621
Test PASSED
speedup [modifiec_omp]: 1.1368
```

Grupa listing 1. Poređenje rezultata za 50000 iteracija

Broj iteracija: 100000

```
[modified_sequential] Elapsed (s): 1.418
[modifiec_omp] Elapsed (s): 1.227
Test PASSED
speedup [modifiec_omp]: 1.155
```

Grupa listing 2. Poređenje rezultata za 100000 iteracija

Broj iteracija: 1000000

```
[modified_sequential] Elapsed (s): 11.43
[modifiec_omp] Elapsed (s): 9.874
Test PASSED
speedup [modifiec_omp]: 1.1575
```

Grupa listing 3. Poređenje rezultata za 1000000 iteracija

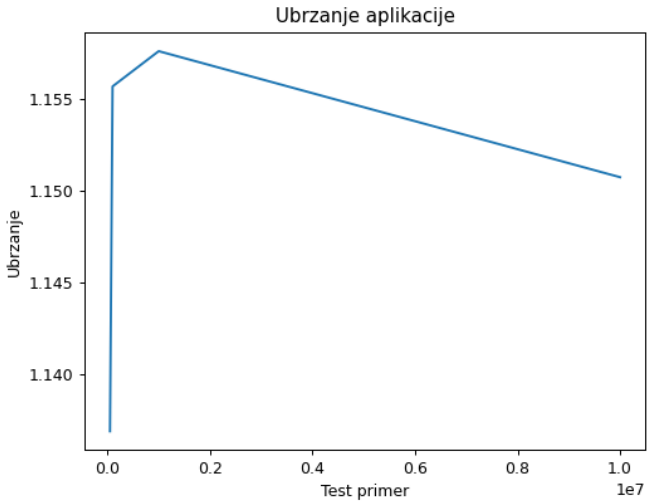
Broj iteracija: 10000000

```
[modified_sequential] Elapsed (s): 179.42
[modifiec_omp] Elapsed (s): 155.92
Test PASSED
speedup [modifiec_omp]: 1.1507
```

Grupa listing 4. Poređenje rezultata za 10000000 iteracija

1.3.2. Grafici ubrzanja

U nastavku je grafik ubrzanja za paralelizovani modifikovani kod.



Slika 1. Grafik zavisnosti ubrzanja sekvencijalnog koda u zavisnosti dimenzionalnosti problema

1.3.3. Diskusija dobijenih rezultata

Paralelizacija modifikovanog sekvencijalnog koda donosi ubrzanje. Podela posla među nitima, tako da svaka (od velikog broja) nit dobije po jedan element za ažuriranje donosi ubrzanje. Profajliranjem transakcija na GPU dobija se da je 50.32% vremena otišlo na računsko iračunavanje, dok je (31.43% GPU na CPU i 18.25% CPU na GPU) ostatak, približno isti, otišao na memorijski transfer. Dimenzionalnost problema je promenjiva u broju iteracija, kako se deo koda koji se ponavlja (zavisno od broja iteracija) paralelizuje, dobija se ubrzanje. Dimenzije nizova su konstantne (20 * current_iteration), tako da srazmerno broju iteracija će i transfer trajati duže. Problem nije pogodan za ovakvo masivno paralelno izračunavanje, zato što ne može da se paralelizuje finalno izračunavanje. Razlog tome je dimenzija tog rezultujućeg niza (40) koji je premali da bi se GPU invocirao da izvrši to.

2. PROBLEM 2 – GAME OF LIFE

2.1. Tekst problema

Paralelizovati program koji implementira simulaciju ćelijskog automata Game of Life. Simulacija je predstavljena dvodimenzionalnom matricom dimenzija w x h, a svaka ćelija c može uzeti vrednost 1 ukoliko predstavlja živu ćeliju, a 0 ukoliko je mrtva. Za svaku ćeliju se vrši izračunavanje vrednosti n koja predstavlja zbir živih ćelija u susedstvu posmatrane ćelije. Posmatra se osam suseda. Ćelije se rađaju i umiru prema pravilima iz sledeće tabele.

Vrednost C	Vrednost N	Nova vrednost C	Komentar
1	0, 1	0	Usamljena ćelija umire
1	4, 5, 6, 7, 8	0	Ćelija umire usled prenaseljenosti
1	2,3	1	Ćelija živi
0	3	1	Rađa se nova ćelija
0	0, 1, 2, 4, 6, 7, 8	0	Nema promene stanja

Može se smatrati da su ćelije van opsega posmatrane matrice mrtve.

2.2. Delovi koje treba paralelizovati

2.2.1. Diskusija

Program se sastoji od *iter* uzastopnih poziva *evolve* funkcije u kojoj se vrši obrada matrice koja predstavlja stanje sistema. Unutar *evolve* funkcije svako polje matrice se računa nezavisno od ostalih, pa ovo predstavlja pogodan deo koda za paralelizaciju. Mogućnost za ovakav vid izračunavanja donosi činjenica da se stanje sistema računa u novu matricu, a da se onda ta matrica prepisuje u stanje sistema za sledeću iteraciju.

2.2.2. Način paralelizacije

CUDA framework omogućuje formiranje velikog broja niti koje će izvršavati određeni unit-of-work. Formiranjem 2D grida se omogućuje da se, ovaj matrični, problem podeli na jedinični element matrice i da se svakoj niti dodeli

(x, y) = (blockIdx.x * blockDim.x + threadIdx.x, blockIdx.y * blockDim.y + threadIdx.y)

element matrice za izračunavanje. 3x3 matrica sa centrom u (x, y) se prolazi i ažurira se stanje za sledeću iteraciju. Sve niti je potrebno da se sinhronizuju nakon ovog izračunavanja, ne bi li sledeća

iteracija krenula, a da se novo izračunato stanje proglasi za tekuće. Nakon kompletne obrade na GPU, izvršena je sinhronizacija (zbog nepostojanja implicitne sinhronizacije) primitivom *cudaDeviceSynchronize()*. U kodu koji odgovara CPU-u se sada nova matrica proglašava za trenutnu, zarad izračunavanja u sledećoj iteraciji.

2.3. Rezultati

Predstavljeni su rezultati za različite test primere.

2.3.1. Logovi izvršavanja

```
Test run variables: Rows=30 Columns=30 Iterations=1000
Evolve function execution time(sequential): 0.058ms
Evolve function execution time(parallel): 0.012ms
Evolve function execution time speedup: 4.833
Total simulation execution time(sequential): 0.055s
Total simulation execution time(parallel): 0.010s
Total execution time speedup: 5.500
Test PASSED
```

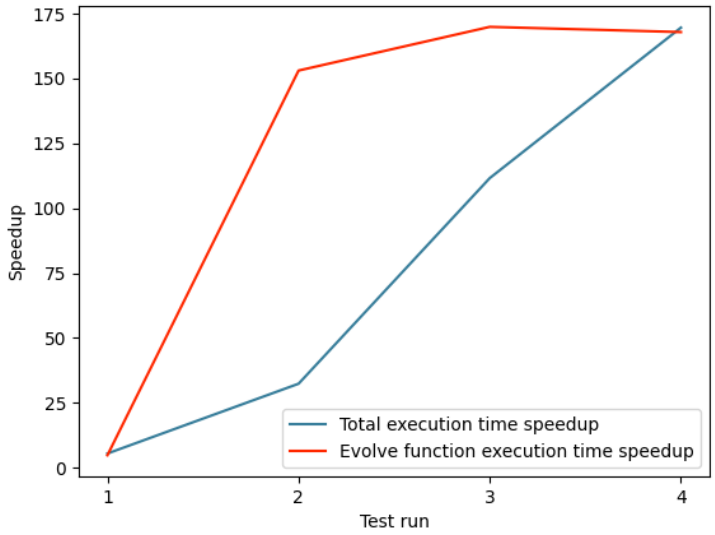
```
Test run variables: Rows=500 Columns=500 Iterations=10
Evolve function execution time(sequential): 14.389ms
Evolve function execution time(parallel): 0.094ms
Evolve function execution time speedup: 153.074
Total simulation execution time(sequential): 0.162s
Total simulation execution time(parallel): 0.005s
Total execution time speedup: 32.400
Test PASSED
```

```
Test run variables: Rows=1000 Columns=1000 Iterations=100
Evolve function execution time(sequential): 58.449ms
Evolve function execution time(parallel): 0.344ms
Evolve function execution time speedup: 169.910
Total simulation execution time(sequential): 5.916s
Total simulation execution time(parallel): 0.053s
Total execution time speedup: 111.623
Test PASSED
```

```
Test run variables: Rows=1000 Columns=1000 Iterations=1000
Evolve function execution time(sequential): 57.422ms
Evolve function execution time(parallel): 0.342ms
Evolve function execution time speedup: 167.901
Total simulation execution time(sequential): 58.006s
Total simulation execution time(parallel): 0.342s
Total execution time speedup: 169.608
Test PASSED
```

2.3.2. Grafici ubrzanja

U nastavku je grafik ubrzanja celokupnog vremena izvršavanja kao i ubrzanja izvršavanja funkcije *evolve* za različite test primere.



2.3.3. Diskusija dobijenih rezultata

Paralelizacija koda donosi značajna ubrzanja, kako vremena totalnog izvršavanja tako i vremena izvršavanja funkcije *evolve*. Povećanje dimenzija matrice koja se obrađuje donosi značajno povećanje ubrzanja prilikom izvršavanja funkcije, sve dok postoji dovoljan broj slobodnih niti da vrši obradu. Vreme totalnog izvršavanja se takodje smanjuje povećanjem dimenzija problema kao i brojem iteracija.

3. PROBLEM 3 – HOTSPOT

3.1. Tekst problema

Paralelizovati program koji rešava problem promene temperature na čipu procesora u dvodimenzionalnom prostoru kroz vreme, ako su poznati početna temperatura i granični uslovi. Simulacija rešava seriju diferencijalnih jednačina nad pravilnom mrežom tačaka kojom se aproksimira površina procesora. Svaka tačka u mreži predstavlja prosečnu temperaturu za odgovarajuću površinu na čipu. Mreža tačaka je predstavljena odgovarajućom matricom koja opisuje trenutne temperature. Analizirati dati kod i obratiti pažnju na način izračunavanja temperatura. Verifikaciju paralelizovanog rešenja vršiti nad dobijenim temperaturama u poslednjem stanju sistema. Način pokretanja programa se nalazi u datoteci run. [1, N].

3.2. Delovi koje treba paralelizovati

3.2.1. Diskusija

Glavni deo programa predstavlja petlja koja se sastoji od *num_iterations* iteracija u kojima se izvršava *single_iteration* funkcija koja vrši obradu nad marticom veličine *row*col* pri čemu se svako polje matrice računa nezavisno. Ta nezavisna obrada predstavlja priliku za paralelizaciju koda. Vrednost jednog izvršavanja funkcije se koristi u narednoj iteraciji, pa nije moguće paralelizovati obradu po iteracijama već samo u okviru jedne iteracije.

3.2.2. Način paralelizacije

Kod je paralelizovan podelom matrice, pri čemu se svaki element matrice dodeljuje jednoj niti 2D grida na izračunavanje. Pozicija u matrici koju nit izračuava se računa na osnovu broja blockIdx i threadIdx po formuli (r, c) = (blockIdx.x * blockDim.x + threadIdx.x, blockIdx.y * blockDim.y + threadIdx.y). Nakon izvršene obrade odgovarajućeg polja, nova vrednost se upisuje u novu matricu. Nakon kompletne obrade na GPU, izvršena je sinhronizacija (zbog nepostojanja implicitne sinhronizacije) primitivom *cudaDeviceSynchronize()*. U kodu koji odgovara CPU-u se sada nova matrica proglašava za trenutnu, zarad izračunavanja u sledećoj iteraciji.

3.3. Rezultati

Kod je testiran nad skupom različitih ulaznih parametara i broja niti. Dobijeni reziltati se nalaze u nastavku.

3.3.1. Logovi izvršavanja

```
Test run variables: Rows=32 Columns=32 Iterations=8192
Total simulation execution time(sequential): 0.091s
Total simulation execution time(parallel): 0.071s
Total execution time speedup: 1.282
Test PASSED
```

```
Test run variables: Rows=256 Columns=256 Iterations=8192
Total simulation execution time(sequential): 5.898s
Total simulation execution time(parallel): 0.367s
Total execution time speedup: 16.071
Test PASSED
```

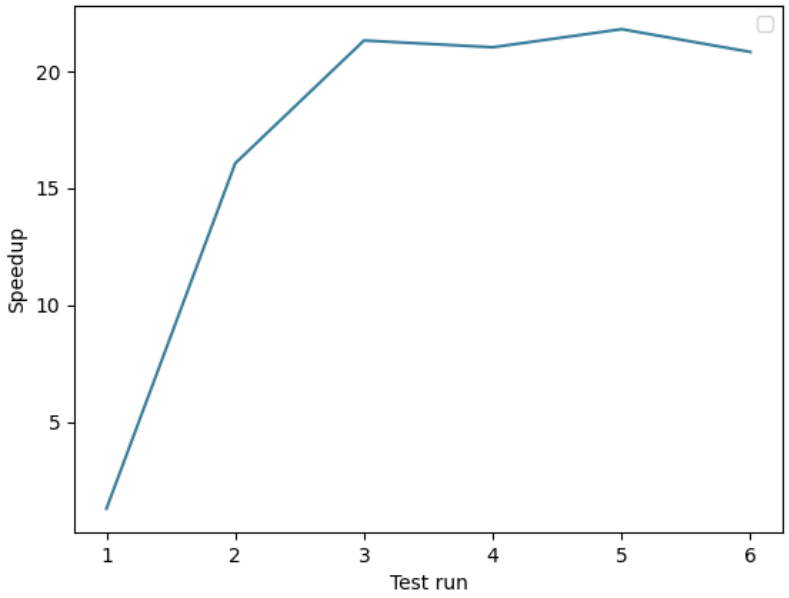
```
Test run variables: Rows=1024 Columns=1024 Iterations=4096
Total simulation execution time(sequential): 48.892s
Total simulation execution time(parallel): 2.292s
Total execution time speedup: 21.332
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=8192
Total simulation execution time(sequential): 96.385s
Total simulation execution time(parallel): 4.581s
Total execution time speedup: 21.040
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=16384
Total simulation execution time(sequential): 199.675s
Total simulation execution time(parallel): 9.153s
Total execution time speedup: 21.815
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=32768
Total simulation execution time(sequential): 381.165s
Total simulation execution time(parallel): 18.291s
Total execution time speedup: 20.839
Test PASSED
```

3.3.2. Grafik ubrzanja



Grafik ubrzanja izvršavanja simulacije za razilčite test primere.

3.3.3. Diskusija dobijenih rezultata

Sa grafika se može uočiti da se paralelizacijom postižu bolje performanse, pri čemu se može uočiti da dimenzije problema utiču na performanse više nego broj iteracija koje se izvršavaju. Manje dimenzije problema ne dovode do značajnijih poboljšanja performansi jer najveći deo vremena se provede u režijskim troškovima prepisivanja matrice sa host-a na device, a veliki deo dostupnih niti ostane neiskorišćeno. Kada se dimenzije povećaju sa 32*32 na 256*256 može se uočiti veliki skok u poboljšanju performansi, kao i prelazak na dimenzije 1024*1024, jer poboljšanje dobijeno paralelizacijom nadmašuje vreme izgubljeno dodatnim režijskim troškovima.