UNIVERZITET U BEOGRADU - ELEKTROTEHNIČKI FAKULTET

MULTIPROCESORKI SISTEMI (13S114MUPS, 13E114MUPS)

DOMAĆI ZADATAK 1 – MPI

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 koje 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. Rezultati *simplex_sample* funckije se koriste za izračunavanje krajnjeg rezultata za svaku iteraciju. Dvostruka for petlja koja to radi se može paralelizovati.

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 generisanje brojeva i pokreće se dvostruka petlja za generisanje rezultata. Ta petlja, prema ranijoj diskusiji može da se paralelizuje. Ova šandarmacija sa kodom u osnovnom smislu donosi poboljšanje, dok paralelizacije unose dodatna poboljšanja.

1.2.2. Način paralelizacije

Proces sa *rank*-om 0 će biti proces koji učitava podatke (prosleđene kao argumente main funkcije) i šalje ih svim ostalim procesima (*MPI_Bcast* direktiva). Takođe taj proces će otpočeti, i nakon završetka izračunavanja zaustaviti brojanje vremena (*MPI_Wtime* direktiva) pošto se konačni niz rezultata skuplja

baš u ovom procesu (*MPI_Gather* direktiva). Paralelizacija se vrši tako što svaki proces (uključujući i proces sa *rank*-om 0) vrši deo izračunavanja (ukupan posao određuje broj iteracija spoljašnje petlje, pomenute u odeljku 1.2.1). Podela posla se vrši tako da svaki proces dobije podjednak deo na obradu, a koji je to deo zavisi od njegovog ranka i ukupnog broja procesa.

1.3. Rezultati

Predstavljeni su rezultati za slučaj modifikovanog sekvencijalnog i modifikovanog paralelizovanog koda.

1.3.1. Logovi izvršavanja

Broj iteracija: 50000

```
NUM_THREADS=2
[modified_sequential] Elapsed (s): 0.631
[modifiec_omp] Elapsed (s): 0.4167
Test PASSED
speedup [modifiec_omp]: 1.514
```

```
NUM_THREADS=4
[modified_sequential] Elapsed (s): 0.631
[modifiec_omp] Elapsed (s): 0.326
Test PASSED
speedup [modifiec_omp]: 1.936
```

```
NUM_THREADS=8
[modified_sequential] Elapsed (s): 0.631
[modifiec_omp] Elapsed (s): 0.344
Test PASSED
speedup [modifiec_omp]: 1.834
```

Grupa listing 1. Poređenje rezultata za 50000 iteracija

Broj iteracija: 100000

```
NUM_THREADS=2
[modified_sequential] Elapsed (s): 1.339
[modifiec_omp] Elapsed (s): 0.845
Test PASSED
speedup [modifiec_omp]: 1.585
```

```
NUM_THREADS=4
[modified_sequential] Elapsed (s): 1.339
[modifiec_omp] Elapsed (s): 0.633
Test PASSED
speedup [modifiec_omp]: 2.115
```

```
NUM_THREADS=8
[modified_sequential] Elapsed (s): 1.339
[modifiec_omp] Elapsed (s): 0.62
Test PASSED
speedup [modifiec_omp]: 2.16
```

Grupa listing 2. Poređenje rezultata za 100000 iteracija

Broj iteracija: 1000000

```
NUM_THREADS=2
[modified_sequential] Elapsed (s): 10.884
[modifiec_omp] Elapsed (s): 6.823
Test PASSED
speedup [modifiec_omp]: 1.595
```

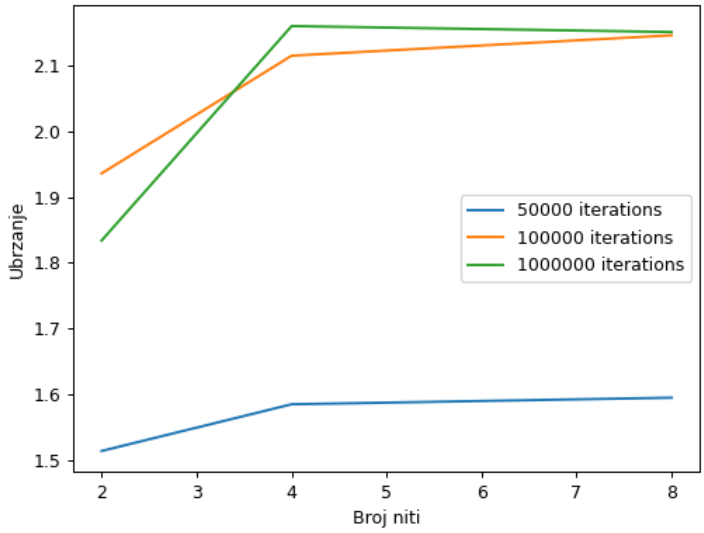
```
NUM_THREADS=4
[modified_sequential] Elapsed (s): 10.884
[modifiec_omp] Elapsed (s): 5.071
Test PASSED
speedup [modifiec_omp]: 2.146
```

```
NUM_THREADS=8
[modified_sequential] Elapsed (s): 10.884
[modifiec_omp] Elapsed (s): 5.059
Test PASSED
speedup [modifiec_omp]: 2.151
```

Grupa listing 3. Poređenje rezultata za 1000000 iteracija

1.3.2. Grafici ubrzanja

U nastavku je grafik ubrzanja za paralelizovani modifikovani kod.



Slika 1. Grafik zavisnosti ubrzanja sekvencijalnog koda u zavisnosti od broja niti i dimenzionalnosti problema

1.3.3. Diskusija dobijenih rezultata

Paralelizacija modifikovanog sekvencijalnog donosi ubrzanje. Povećanje broja niti poboljšava rezultate (smanjuje vremena izvršavanja), ubrzanje pri prelazu sa 2 na 4 niti je primetno, dok je povećanje neznatno u slučaju prelaza sa 4 na 8 niti. Ubrzanje se povećava sa povećanjem iteracija samog programa, što se tumači da MPI bolje radi pri većem opterećenju, zato što su *overhead*-i usled sinhronizacije i komunikacije.

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, najpre se kreira nova matrica na osnovu prethodnog stanja sistema, pri čemu se svako polje računa nezavisno od ostalih, pa ovo predstavlja pogodan deo koda za paralelizaciju.

2.2.2. Način paralelizacije

Spoljašnji ciklus, koji se kreće po redovima matrice je pogodan za podelu posla među različitim procesima. Podela se vrši statički (ukupan posao određuje broj iteracija) tako da svaki proces dobije podjednak deo za obradu, a koji je to deo zavisi od njegovog ranka i ukupnog broja procesa. Kako je podela posla takva da svaka nit dobija određeni broj redova za izračunavanje, ti redovi su iz intervala *[start, end)*. Da bi se izračunao red numerisan kao *start*, potrebno je da proces koji ga izračunava dobije prethodni red (u ovoj notaciji indeksiran kao *start – 1*), od procesa sa brojem *rank - 1*. Takođe poslednji red u skupu redova (u ovoj notaciji označen kao *end – 1*) se izračunava koristeći i red sa indeksom *end*,

koji je potrebno dobiti od procesa sa indeksom *rank + 1*. Potražnja za redovima iz neposredno susednih procesa proizilazi iz samog problema *game-of-life,* dok se samo prosleđivanje poruka vrši direktivama *MPI_Send* i *MPI_Recv*. Dodatno, red koji se prosleđuje je napravljen kao poseban (izvedeni) tip sekvencom direktiva *MPI_Datatype, MPI_Type_contiguous* i *MPI_Type_commit*.

2.3. Rezultati

Predstavljeni su rezultati za različite test primere, za različit broj niti

2.3.1. Logovi izvršavanja

```
Parametri: w=30 h=30 num_iter=1000
```

```
NUM_THREADS=2
Execution time (non parallel): 0.053s
Execution time (parallel): 0.042
Test PASSED
speedup: 1.262
```

```
NUM_THREADS=4
Execution time (non parallel): 0.053s
Execution time (parallel): 0.03s
Test PASSED
speedup: 1.767
```

```
NUM_THREADS=8
Execution time (non parallel): 0.053s
Execution time (parallel): 0.07s
Test PASSED
speedup: 0.757
```

```
Parametri: w=500 h=500 num_iter=10
```

```
NUM_THREADS=2
Execution time (non parallel): 0.201s
Execution time (parallel): 0.104
Test PASSED
speedup: 1.933
```

```
NUM_THREADS=4
Execution time (non parallel): 0.201s
Execution time (parallel): 0.08s
Test PASSED
speedup: 2.513
```

```
NUM_THREADS=8
Execution time (non parallel): 0.201s
Execution time (parallel): 0.103s
Test PASSED
speedup: 1.951
```

```
Parametri: w=1000 h=1000 num_iter=100
```

```
NUM_THREADS=2
Execution time (non parallel): 5.952s
Execution time (parallel): 3.224s
Test PASSED
speedup: 1.846
```

```
NUM_THREADS=4
Execution time (non parallel): 5.952s
Execution time (parallel): 2.454s
Test PASSED
speedup: 2.425
```

```
NUM_THREADS=8
Execution time (non parallel): 5.952s
Execution time (parallel): 3.067s
Test PASSED
speedup: 1.941
```

```
Parametri: w=1000 h=1000 num_iter=1000
```

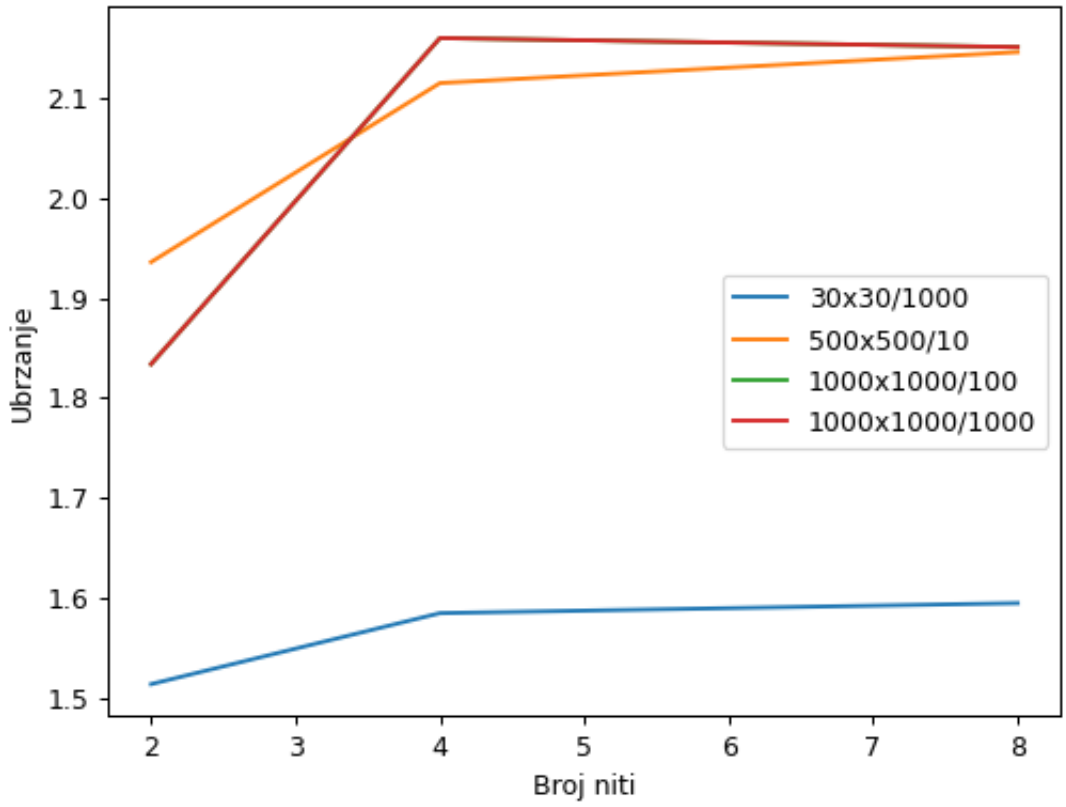
```
NUM_THREADS=2
Execution time (non parallel): 57.199s
Execution time (parallel): 32.017s
Test PASSED
speedup: 1.787
```

```
NUM_THREADS=4
Execution time (non parallel): 57.199s
Execution time (parallel): 23.069s
Test PASSED
speedup: 2.479
```

```
NUM_THREADS=8
Execution time (non parallel): 57.199s
Execution time (parallel): 31.201s
Test PASSED
speedup: 1.833
```

2.3.2. Grafici ubrzanja

U nastavku je grafik ubrzanja za paralelizovani modifikovani kod.



Slika 1. Grafik zavisnosti ubrzanja sekvencijalnog koda u zavisnosti od broja niti i dimenzionalnosti problema

2.3.3. Diskusija dobijenih rezultata

Paralelizacija sekvencijalnog donosi ubrzanje. Povećanje broja niti poboljšava rezultate (smanjuje vremena izvršavanja), ubrzanje pri prelazu sa 2 na 4 niti je primetno, dok je povećanje neznatno u slučaju prelaza sa 4 na 8 niti (može čak i da degradira, ali je približno konstantno). Ubrzanje se povećava sa povećanjem dimenzionalnosti samog problema. Dimenzionalnost se ogleda ili u povećanju veličine matrice, ili u povećanju broja iteracija. Paralelizacija pri većim dimenzijama matrice daje bolje rezultate, čemu je uzrok činjenica da su penali za komunikaciju približno isti, dok je količina posla koji svaka nit treba da odradi veća, pa se dobijaju veći rezultati. Primećuje se da na ubrzanje ne utiče broj iteracija (slučaj 1000x1000 dimenzija matrice, zelena i crvena boja na grafiku) gde su ubrzanja gotovo ista za sve varijante broja niti. Izvodi se zaključak da paralelizam ima smisla

usled veće dimenzije same matrice, dok povećanje broja iteracija ne utiče na samo ubrzanje (režijski troškovi su isti).

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 na regione koji su dodeljeni različitim procesima za obradu. Taj region predstavlja redove matrice iz opsega [*start(rank)*, *end(rank)*), gde svaki proces sa njegovim identifikatorom *rank* računa nove vrednosti za polja iz njemu dodeljenog regiona. Kako bi se smanjili režijski troškovi koji nastaju usled medjuprocesorske komunikacije *MASTER* proces inicijalno šalje ostalim procesima samo njihov respektivni region matrice, a ne celu matricu. Posledica toga je nedostatak 2 reda, *start(rank) – 1* i *end(rank)*, koji su potrebni za izračunavanje, jer na svaki rezultat utiču susedni redovi. Ovaj problem se rešava medjuprocesnom komunikacijom tako što svaki proces šalje prethodnom (*rank - 1*) njegov prvi red (*start(rank)*), od prethodnika primi njegov poslednji (*end(rank - 1) - 1*), šalje sledećem (*rank + 1*) njegov polednji red (*end(rank) - 1*), od sledbenika primi njegov prvi red (*start(rank + 1)*), i onda nastavi sa izračunavanjem. Izuzetak od ovog šablona su prvi i poslednji proces koji nemaju prethodnika i sledbenika respektivno pa oni

obavljaju samo njima odgovarajuću polovinu opisane komunkacije. Ova komunikacija se obavlja u svakoj iteraciji kako bi procesi imali korektne vrednosti za računanje vrednosti sledeće iteracije.

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 Threads=2
Total simulation execution time(sequential): 0.088s
Total simulation execution time(parallel): 0.057s
Total execution time speedup: 1.544
Test PASSED
```

```
Test run variables: Rows=256 Columns=256 Iterations=8192 Threads=2
Total simulation execution time(sequential): 5.816s
Total simulation execution time(parallel): 3.224s
Total execution time speedup: 1.804
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=4096 Threads=2
Total simulation execution time(sequential): 48.186s
Total simulation execution time(parallel): 26.982s
Total execution time speedup: 1.786
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=8192 Threads=2
Total simulation execution time(sequential): 97.097s
Total simulation execution time(parallel): 57.926s
Total execution time speedup: 1.676
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=16384 Threads=2
```

```
Total simulation execution time(sequential): 194.524s
Total simulation execution time(parallel): 112.326s
Total execution time speedup: 1.732
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=32768 Threads=2
Total simulation execution time(sequential): 385.537s
Total simulation execution time(parallel): 212.797s
Total execution time speedup: 1.812
Test PASSED
```

```
Test run variables: Rows=32 Columns=32 Iterations=8192 Threads=4
Total simulation execution time(sequential): 0.088s
Total simulation execution time(parallel): 0.065s
Total execution time speedup: 1.354
Test PASSED
```

```
Test run variables: Rows=256 Columns=256 Iterations=8192 Threads=4
Total simulation execution time(sequential): 5.816s
Total simulation execution time(parallel): 2.652s
Total execution time speedup: 2.193
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=4096 Threads=4
Total simulation execution time(sequential): 48.186s
Total simulation execution time(parallel): 21.300s
Total execution time speedup: 2.262
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=8192 Threads=4
Total simulation execution time(sequential): 97.097s
```

```
Total simulation execution time(parallel): 41.760s
Total execution time speedup: 2.325
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=16384 Threads=4
Total simulation execution time(sequential): 194.524s
Total simulation execution time(parallel): 82.476s
Total execution time speedup: 2.359
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=32768 Threads=4
Total simulation execution time(sequential): 385.537s
Total simulation execution time(parallel): 165.900s
Total execution time speedup: 2.324
Test PASSED
```

```
Test run variables: Rows=32 Columns=32 Iterations=8192 Threads=8
Total simulation execution time(sequential): 0.088s
Total simulation execution time(parallel): 0.121s
Total execution time speedup: 0.727
Test PASSED
```

```
Test run variables: Rows=256 Columns=256 Iterations=8192 Threads=8
Total simulation execution time(sequential): 5.816s
Total simulation execution time(parallel): 3.128s
Total execution time speedup: 1.859
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=4096 Threads=8
Total simulation execution time(sequential): 48.186s
Total simulation execution time(parallel): 23.830s
```

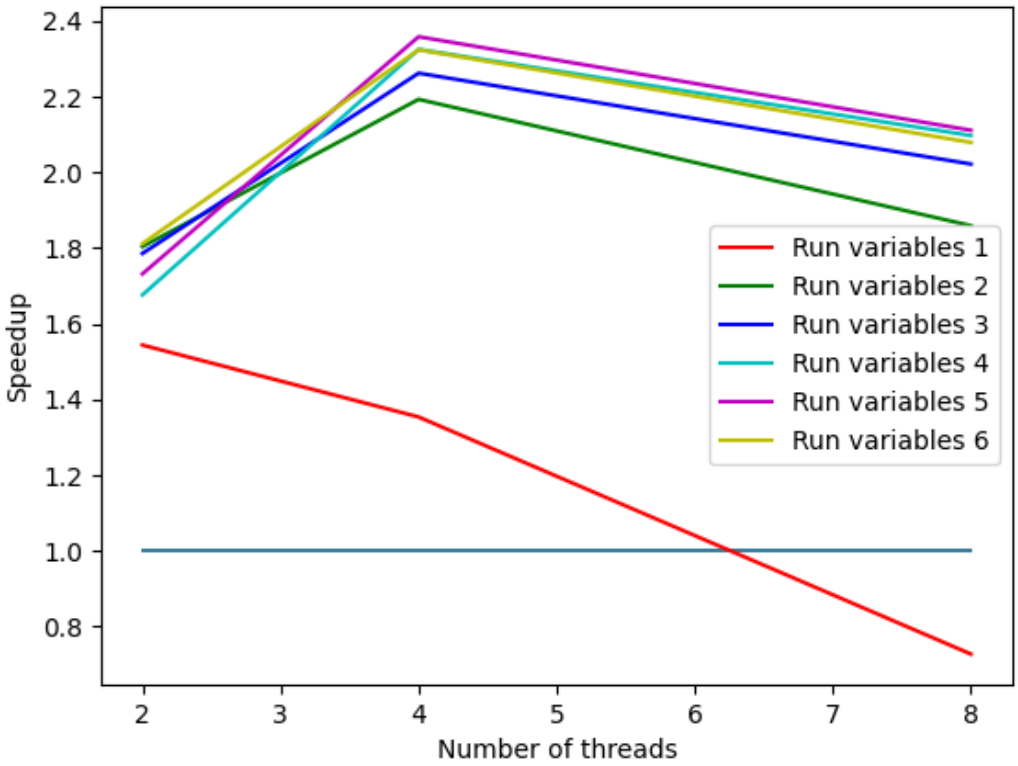
```
Total execution time speedup: 2.022
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=8192 Threads=8
Total simulation execution time(sequential): 97.097s
Total simulation execution time(parallel): 46.294s
Total execution time speedup: 2.097
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=16384 Threads=8
Total simulation execution time(sequential): 194.524s
Total simulation execution time(parallel): 92.133s
Total execution time speedup: 2.111
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=32768 Threads=8
Total simulation execution time(sequential): 385.537s
Total simulation execution time(parallel): 185.428s
Total execution time speedup: 2.079
Test PASSED
```

3.3.2. Grafici ubrzanja



Grafik ubrzanja izvršavanja simulacije u zavisnosti od broja niti za razilčite test primere.

3.3.3. Diskusija dobijenih rezultata

Kao što je i očekivano, paralelizacija dovodi do poboljšanja performansi u opštem slučaju. Sa grafika se mogu uočiti par pravilnosti sa par izuzetaka koje ćemo kasnije razmotriti. Paralelizacija donosi poboljšanje preformansi i povećanjem broja procesa ubrzanje raste dok ne dostigne maksimum korišćenjem 4 procesa, nakon čega ubrzanje polako opada dodatnim povećanjem broja procesa, prouzrokovano povećanjem režijskih troškova koji počnu da prevazilaze dobitke usled paralelizacije. Jedan izuzetak od ovog pravila je prvi test primer kod kog se može uočiti ubrzanje koje konstantno opada sa porastom broja procesa, do te mere da 8 procesa u paraleli duže obavlja posao nego 1 proces koji izvršava sekvencijlni kod. Takođe ubrzanje prilikom paralelizacije je univerzalno manje od ubrzanja preostalih premera, nezavisno od broja procesa. Ovo je posledica specifičnosti primera 1, koji je jedini u ovom skupu kod koga se obrađuje matrica relativno manjih dimenzija 32x32 (u poredjenju sa 256x256 i 1024x1024 koje su prisutne u sledećim primerima) pa medjuprocesna komunikacija i režijski troškovi konzumiraju više vremena nego sama obrada dela matrice koji se šalje na obradu.

4. PROBLEM 4 – HOTSPOT

4.1. 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].

4.2. Delovi koje treba paralelizovati

4.2.1. Diskusija

Identična kao u zadatku 3.

4.2.2. Način paralelizacije

Paralalelizacija se realizuje korišćenjem *worker-manager* modela gde *MASTER* proces šalje delove matrice na obradu preostalim procesima, prikuplja dobijene rezultate i ponavlja proces sve dok ima još poslova da budu raspoređeni, tj. postoji još redova za obradu koji nisu dodeljeni nekom procesu. Svakom procesu kao deo posla se šalje $N + 2$ redova pri čemu će proces obraditi N redova, a dodatna dva reda koja su mu prosledjena su neophodni za izračunavanje rezultata.

4.3. Rezultati

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

4.3.1. Logovi izvršavanja

```
Test run variables: Rows=32 Columns=32 Iterations=8192 Threads=2
Total simulation execution time(sequential): 0.088s
Total simulation execution time(parallel): 0.209s
Total execution time speedup: 0.421
Test PASSED
```

```
Test run variables: Rows=256 Columns=256 Iterations=8192 Threads=2
Total simulation execution time(sequential): 5.816s
Total simulation execution time(parallel): 7.559s
Total execution time speedup: 0.769
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=4096 Threads=2
Total simulation execution time(sequential): 48.186s
Total simulation execution time(parallel): 61.755s
Total execution time speedup: 0.780
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=8192 Threads=2
Total simulation execution time(sequential): 97.097s
Total simulation execution time(parallel): 123.316s
Total execution time speedup: 0.787
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=16384 Threads=2
Total simulation execution time(sequential): 194.524s
Total simulation execution time(parallel): 247.777s
Total execution time speedup: 0.785
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=32768 Threads=2
Total simulation execution time(sequential): 385.537s
Total simulation execution time(parallel): 494.838s
Total execution time speedup: 0.779
Test PASSED
```

```
Test run variables: Rows=32 Columns=32 Iterations=8192 Threads=4
Total simulation execution time(sequential): 0.088s
Total simulation execution time(parallel): 0.123s
Total execution time speedup: 0.715
Test PASSED
```

```
Test run variables: Rows=256 Columns=256 Iterations=8192 Threads=4
Total simulation execution time(sequential): 5.816s
Total simulation execution time(parallel): 3.328s
Total execution time speedup: 1.748
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=4096 Threads=4
Total simulation execution time(sequential): 48.186s
Total simulation execution time(parallel): 28.789s
Total execution time speedup: 1.674
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=8192 Threads=4
Total simulation execution time(sequential): 97.097s
Total simulation execution time(parallel): 57.093s
Total execution time speedup: 1.701
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=16384 Threads=4
Total simulation execution time(sequential): 194.524s
Total simulation execution time(parallel): 111.429s
Total execution time speedup: 1.746
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=32768 Threads=4
Total simulation execution time(sequential): 385.537s
Total simulation execution time(parallel): 223.105s
Total execution time speedup: 1.728
Test PASSED
```

```
Test run variables: Rows=32 Columns=32 Iterations=8192 Threads=8
Total simulation execution time(sequential): 0.088s
Total simulation execution time(parallel): 0.270s
Total execution time speedup: 0.326
Test PASSED
```

```
Test run variables: Rows=256 Columns=256 Iterations=8192 Threads=8
Total simulation execution time(sequential): 5.816s
Total simulation execution time(parallel): 3.575s
Total execution time speedup: 1.627
Test PASSED
```

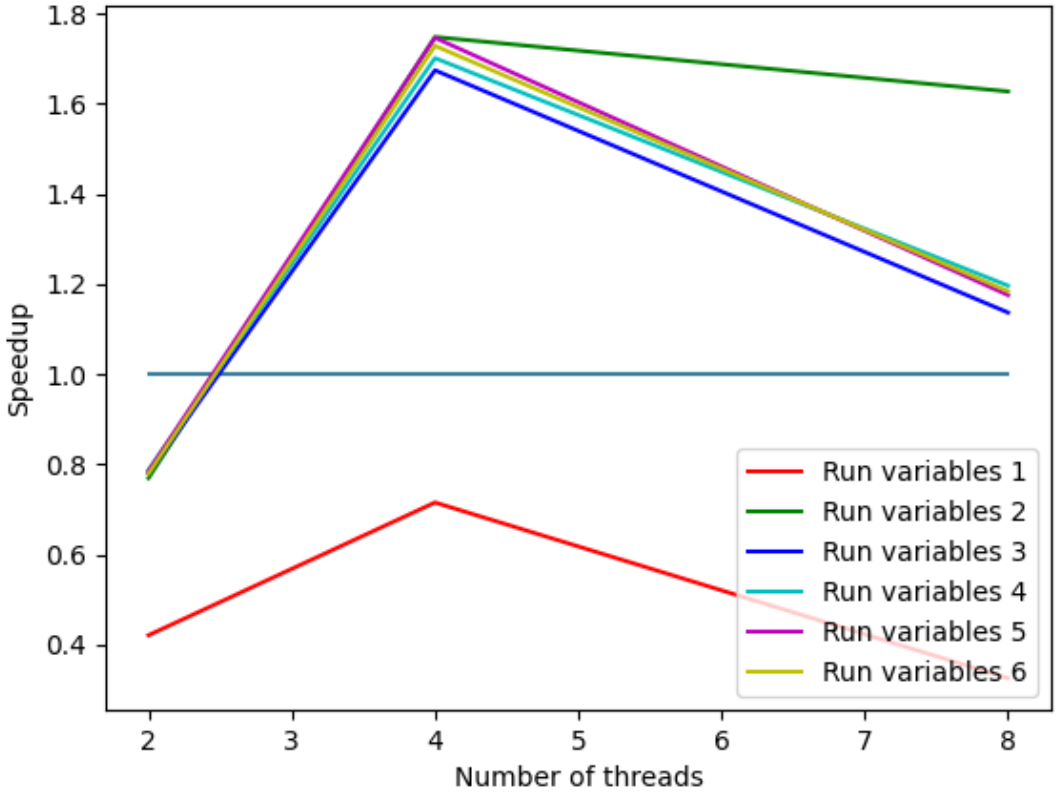
```
Test run variables: Rows=1024 Columns=1024 Iterations=4096 Threads=8
Total simulation execution time(sequential): 48.186s
Total simulation execution time(parallel): 42.395s
Total execution time speedup: 1.137
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=8192 Threads=8
Total simulation execution time(sequential): 97.097s
Total simulation execution time(parallel): 81.171s
Total execution time speedup: 1.196
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=16384 Threads=8
Total simulation execution time(sequential): 194.524s
Total simulation execution time(parallel): 165.518s
Total execution time speedup: 1.175
Test PASSED
```

```
Test run variables: Rows=1024 Columns=1024 Iterations=32768 Threads=8
Total simulation execution time(sequential): 385.537s
Total simulation execution time(parallel): 325.686s
Total execution time speedup: 1.184
Test PASSED
```

4.3.2. Grafici ubrzanja



Grafik ubrzanja izvršavanja simulacije u zavisnosti od broja niti za razilčite test primere.

4.3.3. Diskusija dobijenih rezultata

Sa grafika ubrzanja možemo uočti pravilnosti veoma slične onima iz zadatka 3. Ubrzanje raste porastom broja procesa, dostiže maksimum za broj 4, i onda opada pri daljem porastu broja procesa. Test primer 1 opet pokazuje lošije rezultate od preostalih procesa, ali za razliku od prethodnog zadatka ovde i ovaj primer prati isti šablon porasta pa zatim opadanja ubrzanja. Glavna razlika u odnosu na zadatak 3 predstavlja univerzalno lošije performanse paralelizacije. Ovo je očekivana posledica jer je medjuprocesna komunikacija i sinhronizacija medju procesima mnogo prisutnija i intenzivnija pa se javljaju dodatni režijski troškovi. Ti režijski troškovi veoma utiču na performanse, na šta ukazuje i činjenica da za 2 niti svi test primeri daju lošije performanse nego sekvencijalni kod, a test primer 1 pokazuje lošije performanse od sekvencijalnog koda nezavisno od broja niti. Gore pomenuti broj redova N koji se šalje na obradu procesima utiče na performanse, i testiranjem je uočeno da se dobijaju najbolje performanse za N = 2. Prikazani su rezultati upravo za taj *best-case* slučaj, dok ostali nisu prikazani radi preglednosti i konciznosti izveštaja.