acs
applied computer science

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA PRIMENJENE RAČUNARSKE NAUKE

Paralelno računarstvo

Računarske vežbe

Letnji semestar 2020/2021.
Studijski program: Informacioni inženjering

OpenMP

Primer 1 – Hello World!

```cpp
#include <iostream>

using namespace std;

int main() {

  #pragma omp parallel
  cout << "Hello World!" << endl;

  return 0;
}
```

Kompajliranje:

```
g++ -o hello_world -fopenmp hello_world.cpp
```

Izvršavanje: `./hello_world`

`parallel` konstrukcija

Opšta sintaksa:

`#pragma omp parallel` *`[klauzule]`*
`strukturirani blok`

- Strukturirani blok:
  - ima tačno jednu ulaznu tačku
  - ima tačno jednu izlaznu tačku (ne može sadržati `break`, `goto`, može `exit`)
- Klauzule: `num_threads(broj niti)`, ...

OpenMP biblioteka izvršavanja

- #include <omp.h>
- omp_set_num_threads(num_threads)
- omp_get_num_threads()
- omp_get_thread_num()
- omp_get_wtime()
- ...

OpenMP promenljive okruženja

- OMP_NUM_THREADS
  - `export OMP_NUM_THREADS=4; ./hello_world`
- Zadavanje broja niti:
  1) `OMP_NUM_THREADS`
  2) `omp_set_num_threads()` - prepisuje 1)
  3) `num_threads` – prepisuje 1) i 2)

Programsko okruženje

- Kompajler koji podržava OpenMP API
  - Lista OpenMP alata i kompajlera
  - GCC 7.5.0 puna implementacija OpenMP 4.5
    - Proveriti da li je instalirana `libgomp` biblioteka
  - GCC 4.8.4 puna implementacija OpenMP 3.0 (primeri prilagođeni ovoj verziji)
- Editor po izboru ili IDE

Zadatak 1

Napisati OpenMP C++ program koji zahteva kreiranje tima od 4 niti, gde svaka nit ispisuje svoj identifikator.

Zadatak 2

Modifikovati sekvencijalni C++ program za sumiranje elemenata niza tako da se račun podeli ravnomerno na dve niti. Pri implementaciji zadatka koristiti do sada uvedene OpenMP koncepte. Sekvencijalna implementacija zadatka je data u datoteci `suma_vektora.cpp`.

Kondicionalno prevođenje

- OpenMP program je moguće prevesti u **sekvencijalni** ili **paralelni** program
- Proverava se `_OPENMP` makro

```
#ifdef _OPENMP
// deo OpenMP koda
#endif
```

Zadatak 3

Modifikovati rešenje zadatka 2 tako da ga je moguće prevesti u sekvencijalni ili paralelni program.

Zadatak 4

Napisati C++ program za sumiranje elemenata niza tako da se račun podeli ravnomerno na N niti gde važi $1 \leq N \leq 16$. Smatrati da je dužina niza uvek veća ili jednaka maksimalnom zadatom broju niti. Pri implementaciji zadatka koristiti do sada uvedene OpenMP koncepte. Sekvencijalna implementacija zadatka je data u datoteci `suma_vektora.cpp`.

OpenMP
Konstrukcije za podelu posla

Konstrukcije za podelu posla

- `loop` **konstrukcija**
- `parallel loop` **konstrukcija**
- `single` **konstrukcija**

`loop` konstrukcija

Opšta sintaksa:

```
#pragma omp for [klauzule]
for petlje
```

- Iteracije petlje se automatski distribuiraju između niti
- Klauzule: `schedule`, `collapse`, `private`, `shared`, `firstprivate`, `lastprivate`, `reduction`, `nowait`, ...

Primer 2 - `loop` konstrukcija

```
1 /* ... */
2
3 int main() {
4
5     #pragma omp parallel
6     {
7         #pragma omp for
8         for (int i = 0; i < 8; i++) {
9             stringstream ss;
10            ss << "Nit " << omp_get_thread_num() << ", iteracija " << i << ".\n";
11            cout << ss.str();
12        }
13    }
14
15    return 0;
16 }
```

`parallel loop` konstrukcija

Opšta sintaksa:

`#pragma omp parallel for` *`[klauzule]`*
`for petlje`

- `parallel` i `for` konstrukcije se često spajaju kada se u paralelnom bloku nalazi samo for petlja
- Sve klauzule koje idu uz `for` i `parallel` direktive osim nowait

Primer 3 – parallel loop

```
1  /* ... */
2
3  int main() {
4
5      #pragma omp parallel for
6      for (int i = 0; i < 8; i++) {
7          stringstream ss;
8          ss << "Nit " << omp_get_thread_num() << ", iteracija " << i << ".\n";
9          cout << ss.str();
10     }
11
12     return 0;
13 }
```

`single` konstrukcija

Opšta sintaksa:

```
#pragma omp single [klauzule]
    strukturirani blok
```

- Samo jedna nit iz tima izvršava blok koda
- Klauzule: private, firstprivate, nowait, ...

Primer 4 – `single` konstrukcija

```
1  /* ... */
2
3  using namespace std;
4
5  int main() {
6
7      #pragma omp parallel
8      {
9          int tmp = 5;
10         #pragma omp single
11         tmp = 7;
12
13         stringstream ss;
14         ss << "Nit " << omp_get_thread_num() << ", tmp=" << tmp << ".\n";
15         cout << ss.str();
16     }
17     return 0;
18 }
```

OpenMP
Model podataka

Primer 5 – Proizvod brojeva

```cpp
#include <iostream>

using namespace std;

int main() {

    int proizvod = 1;

    #pragma omp parallel for
    for (int i = 1; i <= 5; i++)
        proizvod *= i;

    cout << "Proizvod prvih pet elemenata je " << proizvod;
    cout << ", a treba da bude 120." << endl;

    return 0;
}
```

Opseg promenljivih

- Deljene
  - Sve OpenMP niti imaju pristup istoj promenljivoj
  - **Trka do podataka**
- Privatne
  - Svaka OpenMP nit ima svoju kopiju promenljive

Opseg promenljivih

- Deljene
  - Deklarisane izvan paralelnog regiona
- Privatne
  - Deklarisane unutar paralelnog regiona
  - Brojačka promenljiva for petlje prvog nivoa
  - Promenljive deklarisane u funkciji pozvanoj iz paralelnog regiona

Klauzule za rad sa podacima 1

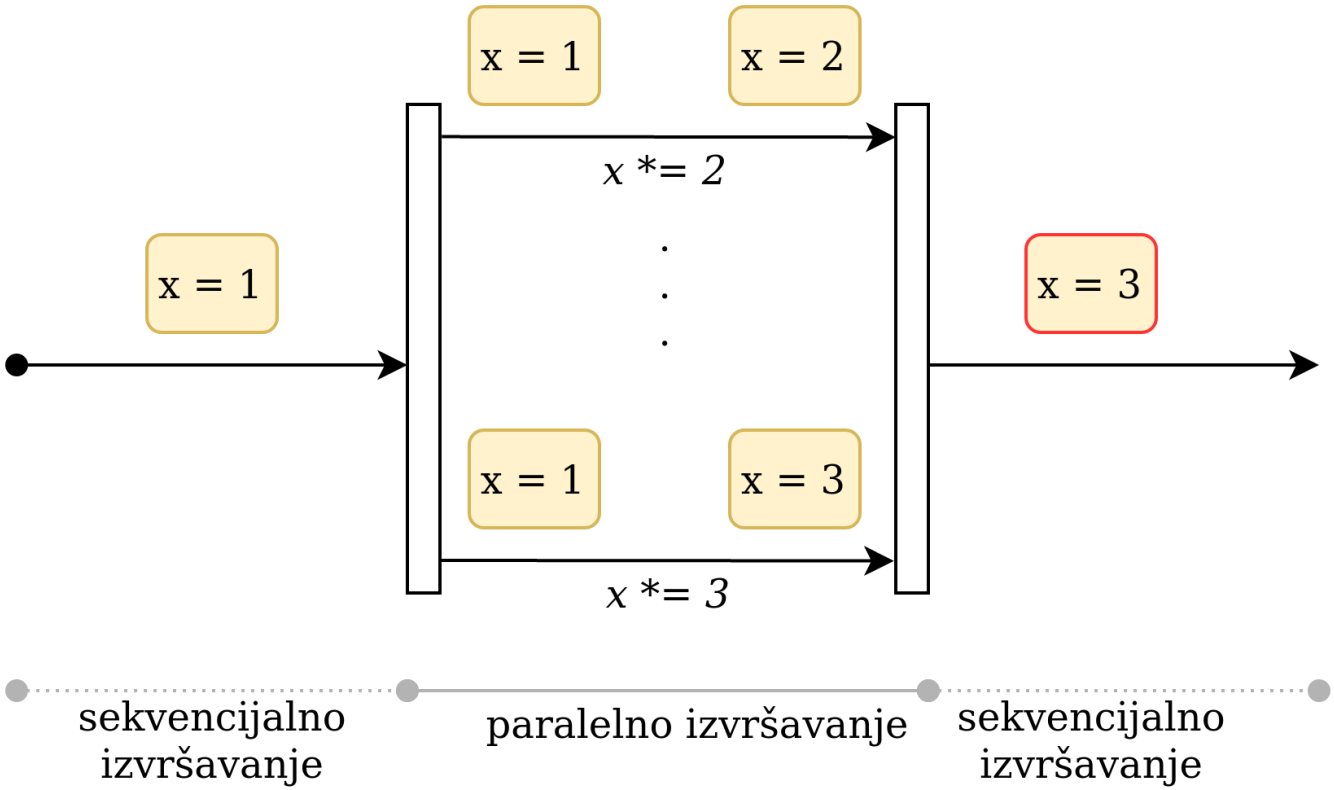
- `private(`*`lista promenljivih odvojenih zarezom`*`)`
- `shared(`*`lista promenljivih odvojenih zarezom`*`)`
- `default(shared|`**`none`**`)`

Primer:

```
#pragma omp parallel for default(none)
                         shared(i,j,k)
                         private(x,y)
```

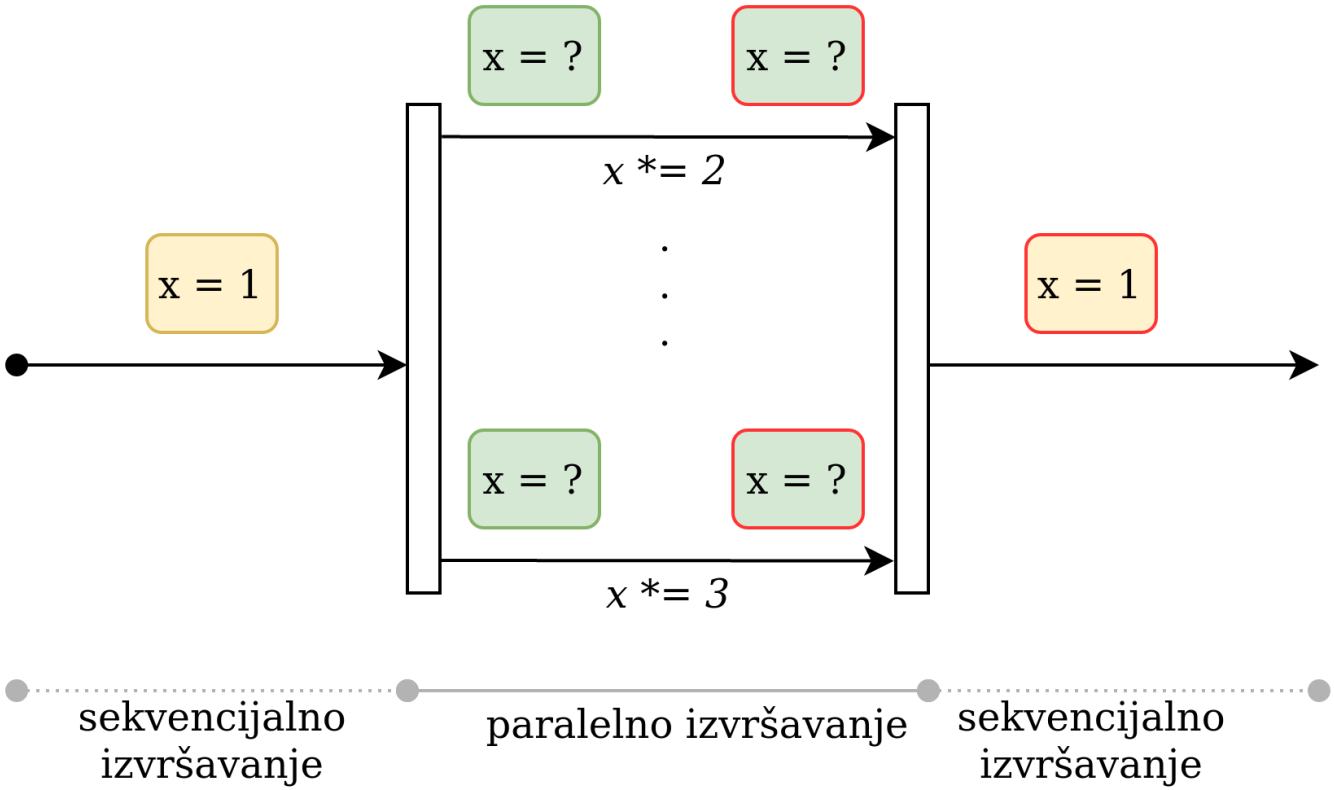
shared klauzula

int x = 1;
#pragma omp parallel for



`private` klauzula

int x = 1;
#pragma omp parallel for private(x)

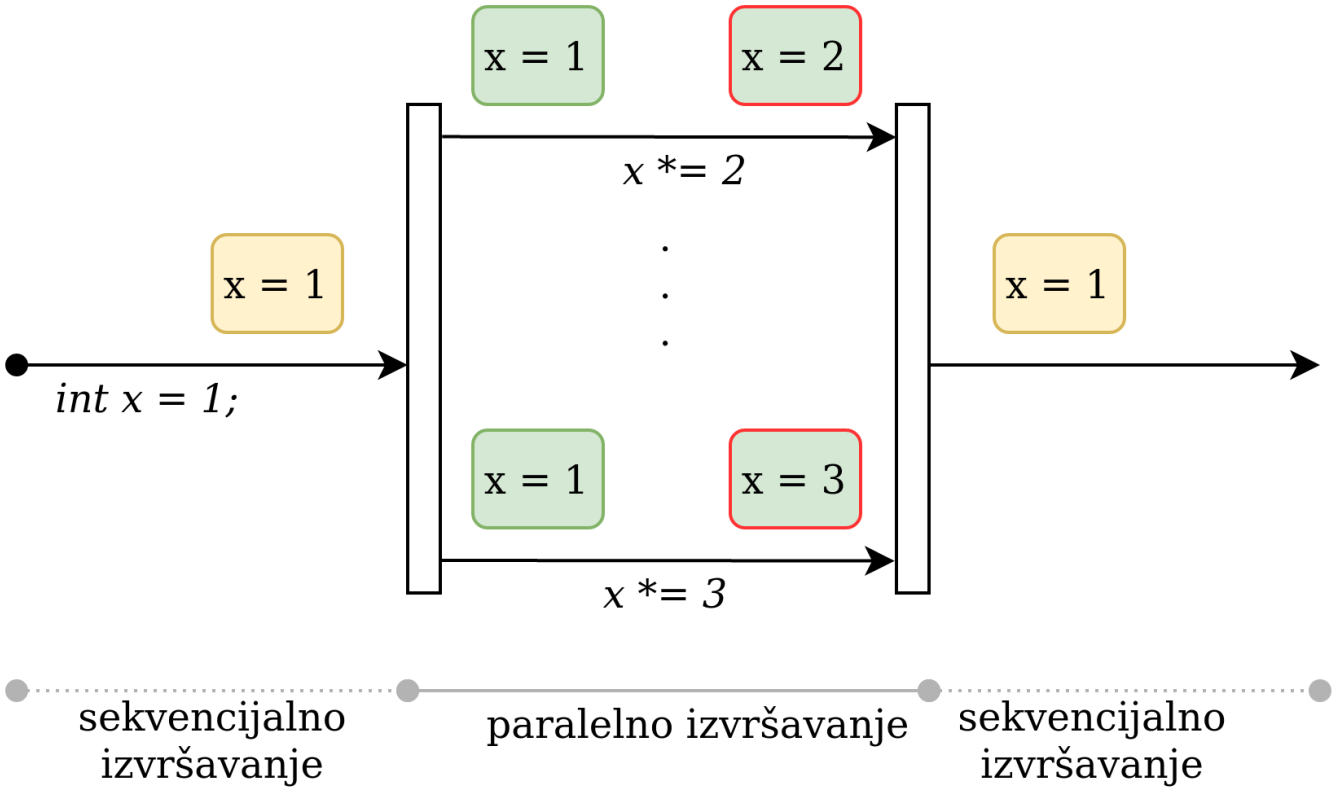


Klauzule za rad sa podacima 2

- `firstprivate(`*`lista promenljih odvojenih zarezom`*`)`
- `lastprivate(`*`lista promenljivih odvojenih zarezom`*`)`
- `reduction(`*`operator:redukciona promenljiva`*`)`

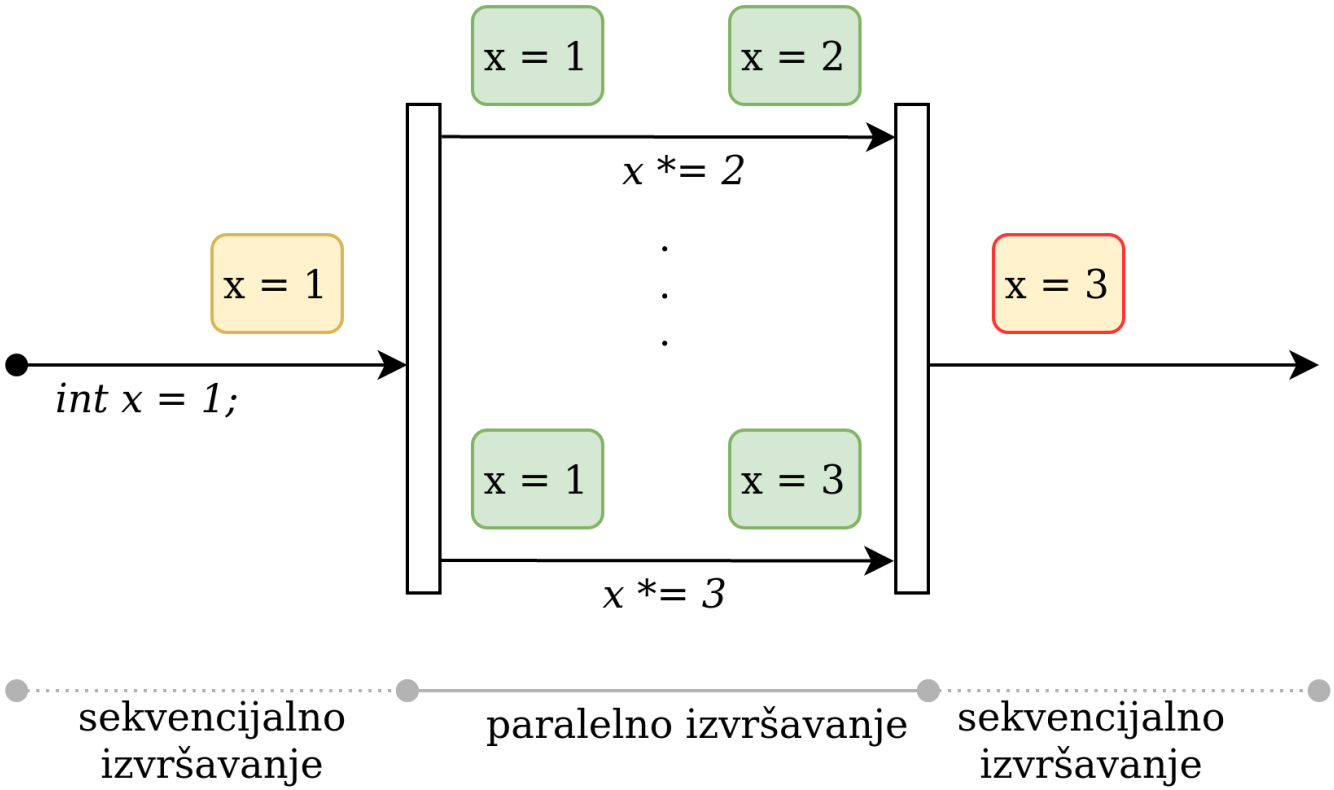
firstprivate klauzula

int x = 1;
#pragma omp parallel for firstprivate(x)



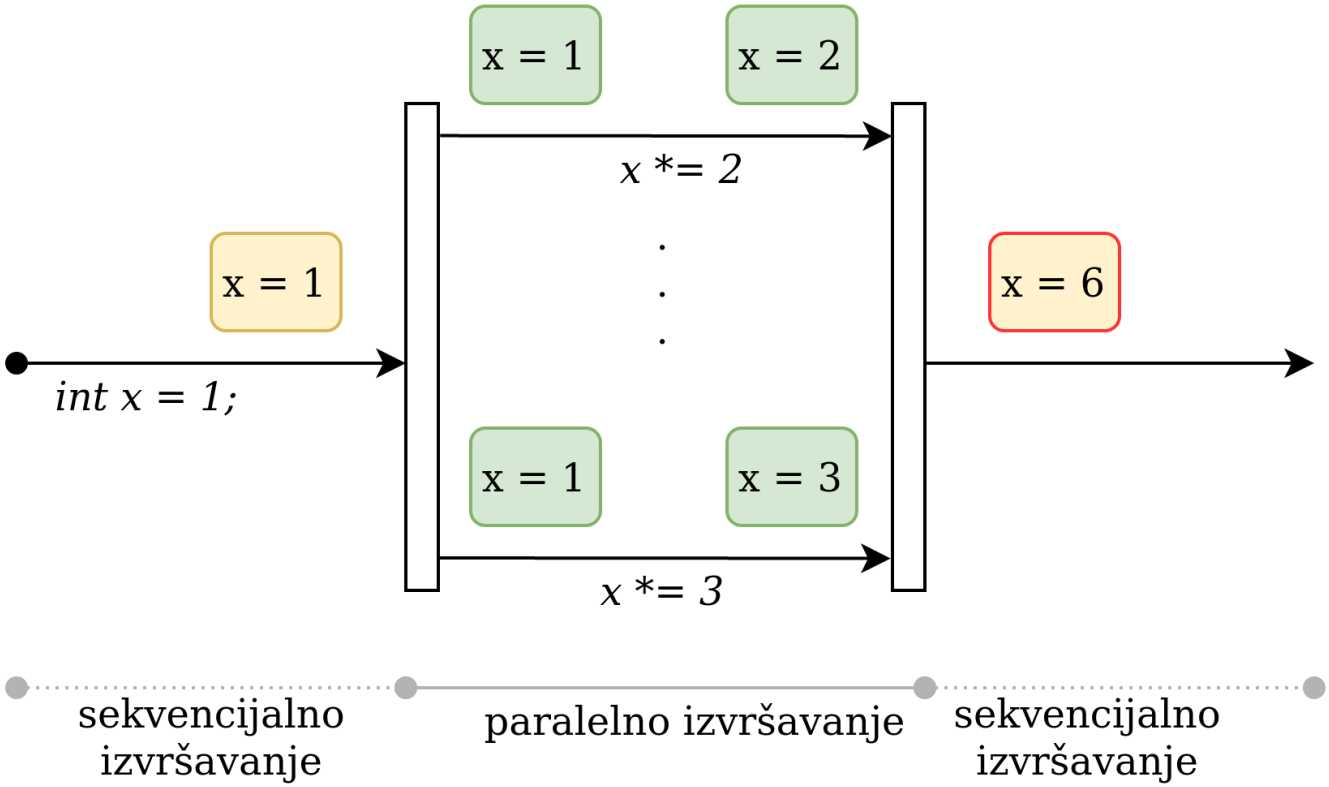
`lastprivate` klauzula

int x = 1;
#pragma omp parallel for firstprivate(x) ***lastprivate(x)***



reduction klauzula

```
int x = 1;
#pragma omp parallel for reduction(*:x)
```



`reduction` klauzula

Redukcioni operatori (inicijalizacione vrednosti privatnih kopija)	
+ (0)	| (0)
* (1)	^ (0)
- (0)	&& (1)
& (~0)	|| (0)
Max (najmanji broj za zadati tip redukcione promenljive)	
Min (najveća vrednost za zadati tip redukcione promenljive)	

- Moguće je defininisati proizvoljnu redukciju

Zadatak 5

Ispraviti rešenje dato u primeru 5 koristeći odgovarajuće OpenMP mehanizmne za rad sa podacima. Izvorni kod rešenja je dat u datoteci `proizvod_pet.cpp`.

Zadatak 6 – Mandelbrotov set

U direktorijumu `zadaci` je data OpenMP paralelizovana implementacija algoritma za računanje Mandelbrotovog seta (datoteka `mandelbrot.cpp`). Nakon paralelizacije, implementacija dalje netačan rezultat usled štetnog preplitanja nastalog greškama u radu s promenljivim. Pronaći i ispraviti greške.

Dodatni materijali

- OpenMP
- Tim Mattson – Introduction to OpenMP
  - Objašnjenje zadatka 6 - 12 Module 7
  - Objašnjenje rešenja zadatka 6 – 13 Discussion 5
- Tim Mattson – A “Hands-on” Introduction to OpenMP