



# Računarski sistemi visokih performansi

Petar Trifunović, Veljko Petrović

Računarske vežbe  
Zimski semestar 2023/24.



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



# ***OpenMP 2***



## Sadržaj

- Sinhronizacija zaključavanjem.
- Klauzule za rad sa podacima.
- Paralelizacija nepoznatog broja iteracija.
- *Task* konstrukcija.
- Konstrukcije za kreiranje i sinhronizaciju zadataka.

---

# *Lock* sinhronizacioni mehanizam



## Lock sinhronizacioni mehanizam

- Mehanizam sinhronizacije niskog nivoa.
- Implementira se funkcijama koje se pozivaju nad promenljivama tipa *omp\_lock\_t*.
- Spisak funkcija za implementaciju zaključavanja:
  - *void omp\_init\_lock(omp\_lock\_t\* lock)* — inicijalizuje *omp\_lock\_t* promenljivu
  - *void omp\_set\_lock(omp\_lock\_t\* lock)* — čeka dok prosleđena promenljiva ne bude slobodna i nakon toga je zaključava
  - *void omp\_unset\_lock(omp\_lock\_t\* lock)* — otključava prosleđenu promenljivu
  - *void omp\_destroy\_lock(omp\_lock\_t\* lock)* — uništava *omp\_lock\_t* promenljivu

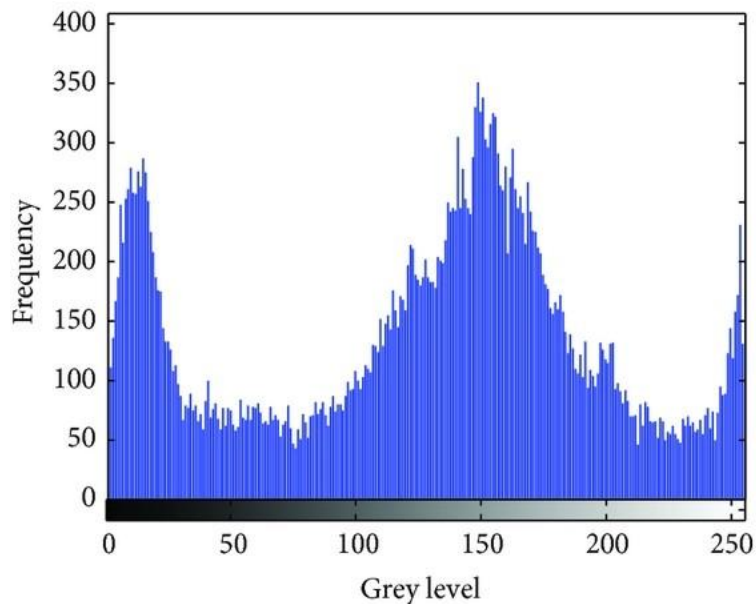


## ***Lock* sinhronizacioni mehanizam**

- *Critical* direktiva u pozadini koristi ovaj mehanizam.
- Zašto nam je onda ikada potreban ovaj mehanizam direktno?
- Ukoliko se obezbeđuje isključiv pristup jednoj promenljivoj, *critical* je dovoljno.
- Šta ako nam je potreban isključiv pristup pojedinačnim elementima niza?

OpenMP

## Histogram — primer *lock* mehanizma



```
int histogram[255];
```



## Histogram — primer *lock* mehanizma

```
#define NBUCKETS 255
```

broj mogućih različitih vrednosti u nizu  
(veličina histograma)

```
#define NVALS 1000
```

veličina niza



## OpenMP



## Histogram — primer *lock* mehanizma

```
int hist[NBUCKETS];
```

histogram

```
omp_lock_t hist_locks[NBUCKETS];
```

promenljive za zaključavanje

```
int arr[NVALS];
```

niz vrednosti



## Histogram — primer *lock* mehanizma

- Inicijalizacija histograma i promenljivih za zaključavanje.

```
#pragma omp parallel for
for (int i = 0; i < NBUCKETS; i++)
{
    hist[i] = 0;
    omp_init_lock(&hist_locks[i]);
}
```



## Histogram — primer *lock* mehanizma

- Isključiv pristup elementima histograma pri ažuriranju.

```
#pragma omp parallel for
for (int i = 0; i < NVALS; i++)
{
    int val = arr[i];

    omp_set_lock(&hist_locks[val]);
    hist[val]++;
    omp_unset_lock(&hist_locks[val]);
}
```

## OpenMP



## Histogram — primer *lock* mehanizma

- Isključiv pristup elementima histograma pri ažuriranju.

```
#pragma omp parallel for
for (int i = 0; i < NVALS; i++)
{
    int val = arr[i];
```

```
    omp_set_lock(&hist_locks[val]);
    hist[val]++;
    omp_unset_lock(&hist_locks[val]);
```

```
}
```

← kritična sekcija




## Histogram — primer *lock* mehanizma

- Isključiv pristup elementima histograma pri ažuriranju.

```
#pragma omp parallel for
for (int i = 0; i < NVALS; i++)
{
    int val = arr[i];
    omp_set_lock(&hist_locks[val]);
    hist[val]++;
    omp_unset_lock(&hist_locks[val]);
}
```

ako više niti ima istu vrednosti *val*  
promenljive, samo jedna će  
zaključati element histograma i  
proći do sledeće naredbe






## Histogram — primer *lock* mehanizma

- Isključiv pristup elementima histograma pri ažuriranju.

```
#pragma omp parallel for
for (int i = 0; i < NVALS; i++)
{
    int val = arr[i];
    omp_set_lock(&hist_locks[val]);
    hist[val]++;
    omp_unset_lock(&hist_locks[val]);
}
```

ostale niti će čekati da  
se vrednost otključa






## Histogram — primer *lock* mehanizma

- Isključiv pristup elementima histograma pri ažuriranju.

```
#pragma omp parallel for
for (int i = 0; i < NVALS; i++)
{
    int val = arr[i];

    omp_set_lock(&hist_locks[val]);
    hist[val]++;
    omp_unset_lock(&hist_locks[val]);
}
```

nakon što nit otključa promenljivu,  
neka druga nit će moći da uđe u  
kritičnu sekciju





## Histogram — primer *lock* mehanizma

- Uništavanje promenljivih za zaključavanje.

```
#pragma omp parallel for
for (int i = 0; i < NBUCKETS; i++)
{
    omp_destroy_lock(&hist_locks[i]);
}
```



---

# Klauzule za rad sa podacima



## Klauzule za rad sa podacima — *shared*

- Eksplicitno navodi koje promenljive su deljene.
- U C/C++ implementaciji *OpenMP* standarda, ovo je podrazumevano ponašanje za promenljive deklarisanе pre ulaska u blok *omp* konstrukcije.
- Može biti deo mnogih konstrukcija, ali za nas je najvažnije da se može dodati uz *parallel* konstrukciju.



## Klauzule za rad sa podacima — *private*

- Navodi koje promenljive deklarisanе van bloka *omp* konstrukcije su privatne unutar bloka.
- Inicijalna vrednost navedenih promenljivih je nedefinisana unutar bloka, makar bila i eksplicitno postavljena pre njega.
- Po završetku bloka konstrukcije, privatne promenljive nestaju, a promenljiva ima vrednost koju je imala pre ulaska u blok.
- Može stajati uz većinu već viđenih konstrukcija (*parallel, loop, single, sections*).



## Klauzula *private* — primer

```
int x = 5;
#pragma omp parallel num_threads(4) private(x)
{
    // vrednost x je ovde nedefinisana
    printf("Nit %d: %d\n", omp_get_thread_num(), x);
}
printf("Vrednost nakon paralelnog bloka: %d\n", x);
```

- Primer izvršavanja:

Nit 0: 32710

Nit 3: 0

Nit 2: 0

Nit 1: 0

Vrednost nakon paralelnog bloka: 5



## Klauzule za rad sa podacima — *firstprivate* i *lastprivate*

- Proširenja u odnosu na *private* klauzulu.
- Klauzula *firstprivate* obezbeđuje da inicijalna vrednost privatnih promenljivih bude jednaka vrednosti koju je promenljiva imala pre ulaska u blok konstrukcije; može stajati uz *parallel*, *loop*, *sections*, *single*.
- Klauzula *lastprivate* obezbeđuje da se, po izlasku iz bloka, sačuva vrednost iz logički poslednje iteracije petlje (ako se navede uz *loop* konstrukciju), ili vrednost iz sekcije koja se poslednja javlja u kodu (ako se navede uz *sections*).
- Ovo su jedine dve klauzule u kojima se jedna promenljiva može istovremeno naći u jednoj istoj konstrukciji.



## Klauzula *firstprivate* — primer

```
int x = 5;
#pragma omp parallel num_threads(4) firstprivate(x)
{
    // x će zadržati vrednost od pre paralelnog regiona
    printf("Nit %d: %d\n", omp_get_thread_num(), x);
}
```

- Primer izvršavanja:

Nit 3: 5

Nit 0: 5

Nit 1: 5

Nit 2: 5



## Klauzula *lastprivate* — primer

```
int x = 5;
#pragma omp parallel for num_threads(4) lastprivate(x)
for (int i = 0; i < 4; i++)
{
    x = i;
}
printf("x = %d\n", x);
```

- Primer izvršavanja:

x = 3



## Kombinacija *firstprivate* i *lastprivate* — primer

```
int x = 5;
#pragma omp parallel for num_threads(4) firstprivate(x) lastprivate(x)
for (int i = 0; i < 4; i++)
{
    x += i;
}
printf("x = %d\n", x);
```

- Primer izvršavanja:

x = 8

- Bez *firstprivate*, rezultat bi bio nedefinisan.





## Zadatak 3

- Datoteka *zadaci/03\_omp\_mandelbrot.c* u direktorijumu *zadaci* sadrži paralelno *OpenMP* rešenje za određivanje Mandelbrotovog skupa.
- Postoje problemi u rešenju vezani za:
  - štetno preplitanje, i
  - korišćenje klauzula za rad sa podacima.
- Potrebno je pronaći probleme i ukloniti ih.
- Ispravljeno rešenje može se naći u datoteci *resenja/03\_omp\_mandelbrot.c*.

---

# Paralelizacija nepoznatog broja iteracija



## Paralelizacija nepoznatog broja iteracija

- *OpenMP* je zamišljen za paralelizaciju petlji kod kojih se unapred zna broj iteracija.
- Problem:
  - Kako paralelizovati *while* petlju?
  - Kako paralelizovati rekurziju?



## Paralelizacija nepoznatog broja iteracija

- *OpenMP* je zamišljen za paralelizaciju petlji kod kojih se unapred zna broj iteracija.
- Problem:
  - Kako paralelizovati *while* petlju?
  - Kako paralelizovati rekurziju?
- Rešenja:
  - Transformisati problem u *for* petlju ako je moguće.
  - Upotrebiti *task* konstrukciju (uvedena u *OpenMP* 3.0).



## Paralelizacija nepoznatog broja iteracija — primer

- Potrebno je paralelizovati sledeći kôd za obradu čvorova liste:

```
while (p != NULL)
{
    processwork(p);
    p = p->next;
}
```



## Paralelizacija nepoznatog broja iteracija — primer

- Prebrojati elemente u listi:

```
int nelems = 0;
while (p != NULL)
{
    nelems++;
    p = p->next;
}
```



## Paralelizacija nepoznatog broja iteracija — primer

- Sačuvati sve pokazivače:

```
p = head;
for (int i = 0; i < nelems; i++)
{
    ptrs[i] = p;
    p = p->next;
}
```



## Paralelizacija nepoznatog broja iteracija — primer

- Paralelizovati obradu:

```
#pragma omp parallel for
for (int i = 0; i < nelems; i++)
{
    processwork(p[i]);
}
```



---

# *Task* konstrukcija



## Task konstrukcija

- Nezavisna jedinica posla.
- Čine je:
  - kôd koji zadatak izvršava,
  - podaci (privatne i deljene promenljive), i
  - *Internal Controll Variables* (ICV) (indikator da li zadatak može da bude dodeljen različitim nitima, vrsta raspoređivanja, broj niti u paralelnom regionu...).

```
#pragma omp task [klauzule]
```

Blok koda



## ***Task* konstrukcija — kombinacija sa *single***

- *task* i *single* konstrukcije se često koriste zajedno.
- Jedna nit pravi zadatke koji se smeštaju u red zadataka odakle su dostupni svim nitima.
- Niti, po internom mehanizmu raspoređivanja, uzimaju zadatke iz reda i izvršavaju ih.
- Moguće je da jedna nit izvrši više zadataka, ali će svaki zadatak biti izvršen samo jednom.

## OpenMP



## *Task* konstrukcija — kombinacija sa *single*

```
#pragma omp parallel
{
    #pragma omp single
    {
        #pragma omp task
        foo();

        #pragma omp task
        bar();
    }
}
```

- Bez *single*, svaka nit bi napravila po dva zadatka.



## ***Task* konstrukcija — razlika u odnosu na *sections***

- Sama *task* konstrukcija nema implicitnu sinhronizaciju (u našem primeru, sinhronizaciju uvodi *single*).
- *Sections* konstrukcija je pogodna kada se unapred zna broj potrebnih sekcija.
- *Task* konstrukcija je pogodna kada je potreban dinamički broj zadataka.



## Zadatak 4 — modifikacija liste

- Data je sekvencijalna implementacija liste (datoteka *zadaci/04\_omp\_linked\_list.c*) u kojoj svaki element sadrži po jedan Fibonačijev broj dobijen funkcijom *processwork*.
- Napraviti paralelni *OpenMP* program korišćenjem *task* konstrukcije.
- Primer rešenja: datoteka *resenja/04\_omp\_linked\_list.c*.



## ***Task* konstrukcija — sinhronizacija**

- *taskwait* — sinhronizuje samo zadatke istog nivoa

```
#pragma omp taskwait
```

- *taskgroup* — sinhronizuje i podzadatke

```
#pragma omp taskgroup  
Blok koda
```

## Task konstrukcija — *taskwait* sinhronizacija

```
#pragma omp parallel
#pragma omp single
{
    #pragma omp task
    {
        printf("Task 1\n");

        #pragma omp task
        {
            printf("Task 2\n");
        }
    }

    #pragma omp taskwait
    #pragma omp task
    {
        printf("Task 3\n");
    }
}
```

- Zadatak 3 će se sigurno izvršiti nakon zadatka 1, jer *taskwait* obezbeđuje sinhronizaciju između njih.
- Zadatak 2 će se **možda** izvršiti pre zadatka 3, ali za to ne postoji garancija, jer *taskwait* ne sinhronizuje zadatke u podnivou.



## Task konstrukcija — *taskgroup* sinhronizacija

```
#pragma omp parallel
#pragma omp single
{
    #pragma omp taskgroup
    {
        #pragma omp task
        {
            printf("Task 1\n");

            #pragma omp task
            {
                sleep(1);
                printf("Task 2\n");
            }
        }

        #pragma omp task
        {
            printf("Task 3\n");
        }
    }
}
```

- Svi zadaci iz *taskgroup* grupe moraju okončati svoje izvršavanje pre nego što se izvrši naredni zadatak.
- Ovime se garantuje da će se i zadatak 1 i zadatak 2 izvršiti pre zadatka 3.



## Zadatak 5 — Fibonačijev niz

- U datoteci *zadaci/05\_omp\_fibonacci.c* data je sekvencijalna implementacija rekurzivnog algoritma koji računa  $n$ -ti element Fibonačijevog niza.
- Paralelizovati implementaciju korišćenjem *task* konstrukcija i sinhronizacija.
- Uporediti vreme izvršavanja paralelne i sekvencijalne implementacije.
- Primer rešenja: datoteka *resenja/05\_omp\_fibonacci.c*.

---

# Dodatni zadaci



## Zadatak 6 — Traženje korena funkcije

- U direktorijumu *zadaci/06\_omp\_bisection*, datoteka *main.c*, data je sekvencijalna implementacija metode za određivanje korena funkcije nad zadatim intervalom metodom bisekcije.
- Ispratiti uputstvo za pokretanje iz datoteke *README.md*.
- Implementirati paralelno rešenje.
- Uporediti vreme izvršavanja paralelne i sekvencijalne implementacije.
- Primer rešenja: direktorijum *resenja/06\_omp\_bisection*, datoteka *main.c*.



## Zadatak 7 — Genetski algoritam

- U direktorijumu *zadaci/07\_omp\_genetic\_algorithm*, datoteka *main.c*, data je sekvencijalna implementacija jednostavnog genetskog algoritma.
- Ispratiti uputstvo za pokretanje iz datoteke *README.md*.
- Nakon izvršenja programa biće ispisano koji deo algoritma troši koji procenat ukupnog vremena izvršenja.
- Obratiti pažnju koji delovi algoritma su vremenski najzahtevniji, razmotriti mogućnosti paralelizacije, i paralelizovati delove gde za to postoji prostor.
- Obratiti pažnju kako promena broja iteracija i jedinki utiče na rezultat
- Primer rešenja: direktorijum *resenja/07\_omp\_genetic\_algorithm*, datoteka *main.c*.
- Zakomentarisati neke od uvedenih paralelizacija i obratiti pažnju na njihov uticaj na vreme izvršavanja.



## Zadatak 8 — Množenje matrica

- U direktorijumu *zadaci/08\_omp\_matrix\_mul*, datoteka *main.c*, dat je kostur zadatka za množenje matrica.
- Ispratiti uputstvo za pokretanje iz datoteke *README.md*.
- Implementirati sekvencijalno rešenje (funkcija *matrix\_multiply*).
- Implementirati paralelno rešenje (funkcija *matrix\_multiply\_openmp*).
- Uporediti vreme izvršavanja paralelne i sekvencijalne implementacije.
- Primer rešenja: direktorijum *resenja/08\_omp\_matrix\_mul*, datoteka *main.c*.
- Videti uputstvo za generisanje matrica i proveru rešenja na **sledećem slajdu**.



## Zadatak 8 — Množenje matrica

- Kao pomoć za generisanje matrica i proveru rešenja treba koristiti *python* skripte za rad sa *hdf5* formatom podataka.
- Skripte se nalaze na *acs*-u, u repozitorijumu predmeta, na putanji *vezbe/hdf5utils*.
- Obe skripte sadrže uputstvo za upotrebu u vidu komentara.
- Za rad sa ovim skriptama najbolje je:
  - kreirati i aktivirati [python virtuelno okruženje](#),
  - instalirati *h5py* biblioteku komandom *pip install h5py*.



## Zadatak 9 — Transponovanje matrice

- U direktorijumu *zadaci/09\_omp\_matrix\_transp*, datoteka *main.c*, data je sekvencijalna implementacija transponovanja matrice (funkcija *transpose*).
- Implementirati paralelno transponovanje matrica (funkcija *transpose\_openmp*).
- Da li bi bilo moguće transponovati matricu na isti način, ukoliko se podaci ne prepisuju u novu matricu, već se samo menja stara (**mat[i, j]** postaje **mat[j, i]**)?
- Videti uputstvo za generisanje matrica i proveru rešenja na **sledećem slajdu**.





## Zadatak 9 — Transponovanje matrice

- Za generisanje matrica može se iskoristiti skripta kao i za zadatak sa množenjem matrica.
- Za proveru rešenja koristiti skriptu *hdf5\_matrix\_transpose\_comparator.py*, koja sadrži uputstvo za korišćenje u vidu komentara.