OpenMPI cheat sheet **2**
1. “Opšte” funkcije 2
2. Point-to-Point komunikacija 3
    2.1. Blokirajuća komunikacija 3
    2.2. Neblokirajuća komunikacija 4
3. Kolektivna komunikacija 5
4. Rad sa grupama i komunikatorima 7
5. Važni tipovi i konstante 9

OpenMPI cheat sheet

1. “Opšte” funkcije

```
/**
 * Inicijalizacija MPI programa.
 * Ako se main-u proslede (int argc, char* argv[]), onda se poziva kao
 * MPI_Init(&argc, &argv).
*/
int MPI_Init(int *argc, char ***argv)


/**
 * Okončava MPI program. Svi procesi moraju pozvati ovu funkciju.
*/
int MPI_Finalize()


/**
 * Pribavlja broj procesa u komunikatoru comm, i smešta ga u
 * promenljivu size.
*/
int MPI_Comm_size(MPI_Comm comm, int *size)


/**
 * Pribavlja rank pozivajućeg procesa u komunikatoru comm, i smešta
 * ga u promenljivu rank.
*/
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

2. *Point-to-Point* komunikacija

2.1. Blokirajuća komunikacija

```
/**
 * Blokirajuće slanje.
 * ===============================
 * >>>> Parametri <<<<:
 ** buf - memorijska adresa podataka za slanje
 ** count, datatype - šalje se count podataka tipa datatype
 ** dest, comm - šalje se procesu sa rankom dest u komunikatoru comm
 ** tag - vrednost tag-a mora biti ista u pozivu ove funkcije i
 **   pozivu funkcije za prijem poruke
 * ===============================
 * >>>> Pod-varijante <<<<:
 ** MPI_Ssend - sinhrono slanje; funkcija blokira izvršenje dok se
 **   ne pojavi MPI_Recv sa kojim se može upariti.
 ** MPI_Bsend - baferovano slanje; funkcija blokira izvršenje dok
 **   se poruka ne smesti u privremeni bafer.
 ** MPI_Send će imati karakteristike sinhronog ili baferovanog slanja,
 ** u zavinosti od okolnosti slanja.
 * ===============================
*/
int MPI_Send(const void *buf, int count, MPI_Datatype datatype,
     int dest, int tag, MPI_Comm comm)
```

```
/**
 * Blokirajuć prijem.
 * ===============================
 * >>>> Parametri <<<<:
 ** buf - memorijska adresa na koju se smeštaju podaci
 ** count, datatype - prima se najviše count podataka tipa datatype
 ** source, comm - poruka se prima od procesa sa rankom source u
 **   komunikatoru comm
 ** tag - mora biti isti kao i u pozivu funkcije za slanje
 ** status - sadrži informacije o primljenoj poruci; proslediti
 **   MPI_STATUS_IGNORE ako ovaj parametar nije od značaja
 * ===============================
*/
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,
     int tag, MPI_Comm comm, MPI_Status *status)
```

2.2. Neblokirajuća komunikacija

```
/**
 * Neblokirajuće slanje.
 * ==============================
 * >>>> Parametri <<<<:
 ** osim request, svi su isti kao u blokirajućoj varijanti
 ** request - promenljiva koja će se koristiti za ispitivanje statusa
 **   slanja (videti MPI_Wait i MPI_Test)
 * ==============================
 * >>>> Pod-varijante <<<<:
 ** MPI_Issend, MPI_Ibsend, sa istim značenjem kao i blokirajuće
 * ==============================
*/
int MPI_Isend(const void *buf, int count, MPI_Datatype datatype,
     int dest, int tag, MPI_Comm comm, MPI_Request *request)
```

```
/**
 * Neblokirajuć prijem.
 * ==============================
 * >>>> Parametri <<<<:
 ** isti kao u blokirajućoj varijanti, samo request ide umesto status
 ** request - promenljiva koja će se koristiti za ispitivanje statusa
 **   prijema (videti MPI_Wait i MPI_Test)
 * ==============================
*/
int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source,
     int tag, MPI_Comm comm, MPI_Request *request)
```

```
/**
 * Neblokirajuće ispitivanje statusa neblokirajućeg slanja/prijema.
 * ==============================
 * >>>> Parametri <<<<:
 ** request - ista promenljiva koja je bila prosleđena neblokirajućem
 **   slanju ili prijemu kao poslednji parametar
 ** flag - 0 ako slanje/prijem na koje se odnosi request nije
 **   okončano, različito od 0 ako jeste
 ** status - čuva informacije o poslatoj/primljenoj poruci; proslediti
 **   MPI_STATUS_IGNORE ako ovaj parametar nije od značaja
 * ==============================
*/
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
```

```
/**
 * Blokirajuće ispitivanje statusa neblokirajućeg slanja/prijema.
 * Funkcija blokira izvršavanje sve dok se slanje/prijem na koji se
 * odnosi parametar request ne okonča.
 * ================================
 * >>>> Parametri <<<<:
 ** request - ista promenljiva koja je bila prosleđena neblokirajućem
 **   slanju ili prijemu kao poslednji parametar
 ** status - čuva informacije o poslatoj/primljenoj poruci; proslediti
 **   MPI_STATUS_IGNORE ako ovaj parametar nije od značaja
 * ================================
*/
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

3. Kolektivna komunikacija

```
/**
 * Slanje poruke svim procesima unutar komunikatora.
 * ================================
 * >>>> Parametri <<<<:
 ** buffer - memorijska lokacija podataka koje šalje proces koji
 **   sadrži podatke, a na koju će se poruka u svim procesima smestiti
 **   nakon prijema
 ** count, datatype - šalje se/prima se count podataka tipa datatype
 ** root - proces koji sadrži podatke, odnosno iz koga će se podaci
 **   emitovati svim procesima
 ** comm - komunikator unutar kog se vrši kolektivno emitovanje
 * ================================
*/
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
     int root, MPI_Comm comm)

/**
 * Obrnuto od Gather. Poruka se deli na onoliko jednakih segmenata
 * koliko ima procesa u komunikatoru, i svakom od njih (uključujući i
 * onoga ko šalje) se šalje po jedan segment. Segmenti se raspoređuju
 * po ranku, u rastućem redosledu (prvi segment ide ranku 0, sledeći
 * ranku 1…).
 * ================================
 * >>>> Parametri <<<<:
 ** sendbuf - memorijska lokacija podataka koji se šalju
 ** sendcount, sendtype - svakom procesu se šalje sendcount podataka
 **   tipa sendtype (sendcount ne označava kolika je ukupna veličina
 **   svih poslatih podataka, već kolika je veličina jednog poslatog
```

```
 **  segmenta)
 ** recvbuf - memorijska lokacija na koju se primaju podaci
 ** recvcount, recvtype - svaki proces prima po recvcount podataka
 **  tipa recvtype
 ** root - proces koji šalje podatke svim procesima
 ** comm - komunikator unutar kog se vrši kolektivna komunikacija
 * ==============================
*/
int MPI_Scatter(const void *sendbuf, int sendcount,
     MPI_Datatype sendtype, void *recvbuf, int recvcount,
     MPI_Datatype recvtype, int root, MPI_Comm comm)


/**
 * Obrnuto od Scatter. Svaki proces u komunikatoru (uključujući i
 * root) šalje root procesu po jedan segment, a root nadovezuje
 * segmente jedan na drugi i to po ranku procesa koji šalju (prvo ide
 * segment procesa 0, pa procesa 1…).
 * ==============================
 * >>>> Parametri <<<<:
 ** sendbuf - memorijska lokacija podataka koji se šalju
 ** sendcount, sendtype - svaki proces šalje po sendcount podataka
 **  tipa sendtype
 ** recvbuf - memorijska lokacija na koju se primaju podaci
 ** recvcount, recvtype - root prima recvcount podataka tipa recvtype
 **  od svakog procesa (recvcount je veličina jednog segmenta, a ne
 **  konačna veličina spojenih segmenata)
 ** root - proces koji prima segmente od svih procesa i nadovezuje ih
 ** comm - komunikator unutar kog se vrši kolektivna komunikacija
 * ==============================
*/
int MPI_Gather(const void *sendbuf, int sendcount,
     MPI_Datatype sendtype, void *recvbuf, int recvcount,
     MPI_Datatype recvtype, int root, MPI_Comm comm)
```

4. Rad sa grupama i komunikatorima

```
/**
 * Deli postojeći komunikator na jedan ili više različitih. Svi
 * procesi unutar comm komunikatora moraju pozvati ovu funkciju.
 * ==============================
 * >>>> Parametri <<<<:
 ** comm - komunikator koji se deli
 ** color - svi procesi koji pozovu funkciju sa istom vrednošću ovog
 **   parametra, naći će se u istom rezultujućem komunikatoru
 ** key - rangiranje u novom komunikatoru vrši se na osnovu ovog
 **   parametra; ko ima najmanji key biće rank 0, sledeći rank 1 i tako
 **   dalje; key ne mora biti isti kao novi rank (ako proces prosledi
 **   50 kao key, ali svi procesi sa istim color proslede veći key,
 **   proces koji prosledi 50 imaće rank 0 u novom komunikatoru).
 ** newcomm - promenljiva koja ukazuje na novi komunikator
 * ==============================
*/
int MPI_Comm_split(MPI_Comm comm, int color, int key,
     MPI_Comm *newcomm)


/**
 * Kreira grupu group i smešta u nju sve procese iz komunikatora comm.
*/
int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)


/**
 * Kreira novi komunikator newcomm od komunikatora comm, vadeći iz
 * njega sve procese koji se nalaze u grupi group (parametar tag
 * postaviti na 0).
*/
int MPI_Comm_create_group(MPI_Comm comm, MPI_Group group, int tag,
     MPI_Comm *newcomm)


/**
 * Kreira novu grupu newgroup vadeći iz grupe group n procesa čiji su
 * rankovi navedeni u nizu ranks. Rankovi u novoj grupi određeni su
 * redosledom navođenja procesa u nizu ranks.
*/
int MPI_Group_incl(MPI_Group group, int n, const int ranks[],
    MPI_Group *newgroup)
```

```
/**
 * Nova grupa newgroup sadržaće procese koji ostanu kada se iz grupe
 * group izbaci n procesa čiji su rankovi navedeni u nizu ranks.
 * Rankovi u novoj grupi određeni su rankovima iz stare grupe group.
*/
int MPI_Group_excl(MPI_Group group, int n, const int ranks[],
    MPI_Group *newgroup)


/**
 * Nova grupa newgroup sadržaće procese koji se nalaze i u group1 i u
 * group2. Rankovi su određeni na osnovu rankova iz grupe group1.
*/
int MPI_Group_intersection(MPI_Group group1, MPI_Group group2,
     MPI_Group *newgroup)


/**
 * Nova grupa newgroup sadržaće sve procese iz grupa group1 i group2,
 * bez duplikata. Rankovi se redom dodeljuju prvo procesima iz group1,
 * pa zatim procesima iz group2 kojih nema u group1.
*/
int MPI_Group_union(MPI_Group group1, MPI_Group group2,
     MPI_Group *newgroup)


/**
 * Nova grupa newgroup sadržaće sve procese iz grupe group1, koji se
 * ne nalaze u group2. Rankovi su određeni rankovima iz group1.
*/
int MPI_Group_difference(MPI_Group group1, MPI_Group group2,
    MPI_Group *newgroup)


/**
 * Oslobađanje grupe.
*/
int MPI_Group_free(MPI_Group *group)


/**
 * Oslobađanje komunikatora.
*/
int MPI_Comm_free(MPI_Comm *comm)
```

5. Važni tipovi i konstante

MPI_SUCCESS - povratna vrednost svih funkcija, ukoliko prođu uspešno

MPI_Comm, MPI_Group - tipovi za komunikatore i grupe
MPI_COMM_WORLD - svetski komunikator, svi pokrenuti procesi mu pripadaju
MPI_COMM_SELF - svaki proces ima svoj *self* komunikator i jedini je u njemu
MPI_COMM_NULL - vrednost promenljive komunikatora kome proces ne pripada