# Računarski sistemi visokih performansi

Nikola Vukić, Petar Trifunović, Veljko Petrović

Računarske vežbe
Zimski semestar 2024/25.

# *MPI 3*

# Sadržaj

- Šta su izvedeni tipovi podataka?
- Funkcije za kreiranje izvedenih tipova.
- Zadaci.

# Šta su izvedeni tipovi podataka?

# Izvedeni tipovi podataka

- Do sada smo videli da je moguće jednom porukom poslati i primiti više od jednog podatka nekog od *MPI* tipova.
- Ovo radimo postavljanjem parametara koji su u opisu funkcija najčešće nosili nazive oblika *count*, *sendcount*, *recvcount*.

# Izvedeni tipovi podataka

- Do sada smo videli da je moguće jednom porukom poslati i primiti više od jednog podatka nekog od *MPI* tipova.
- Ovo radimo postavljanjem parametara koji su u opisu funkcija najčešće nosili nazive oblika *count*, *sendcount*, *recvcount*.
- Šta ako nam je potrebno da u jednoj poruci pošaljemo više različitih tipova podataka? Ili ako je potrebno slanje grupe istih tipova podataka specifično raspoređenih u memoriji?

# Izvedeni tipovi podataka

- *MPI* omogućava definisanje takozvanih *izvedenih tipova podataka* kojima korisnik definiše izgled novog *MPI* podatka koji se sastoji od osnovnih, ili od drugih izvedenih.

# Funkcije za kreiranje izvedenih tipova

# *MPI_Type_create_struct*

```
int MPI_Type_create_struct(
    int count,
    int array_of_blocklengths[],
    const MPI_Aint array_of_displacements[],
    const MPI_Datatype array_of_types[],
    MPI_Datatype *newtype)
```

- Parametar *count* označava broj blokova koji čine izvedeni tip, a ujedno i dužinu naredna tri parametra funkcije.
- *i*-ti elementi naredna tri parametra znače sledeće: *i*-ti blok novog tipa udaljen je *array_of_displacements[i]* bajtova od početka novodefinisanog tipa i sadrži *array_of_blocklengths[i]* podataka tipa *array_of_types[i]*.

# *MPI_Aint*

- Pokazivači u *C*-u nisu nužno jednaki apsolutnoj adresi podatka (iako je ovo najčešće slučaj), a mogu se i različito definisati na mašinama sa različitim segmentiranjem adresnog prostora.
- Zato postoji *MPI_Aint* kao portabilan tip adrese, i pojedine funkcije koje rade sa tim tipom.

```
int MPI_Get_address(
    const void *location,
    MPI_Aint *address)
```

- Pretvara lokaciju na koju ukazuje *location* u *MPI* oblik adrese koja se smešta u promenljivu *address*.

```
MPI_Aint MPI_Aint_diff(
        MPI_Aint addr1,
        MPI_Aint addr2)
```

- Oduzima *addr2* od *addr1* i vraća novu adresu.
- Funkcija uvedena kako bi se sprečio problem oduzimanja pokazivača, jer je MPI_Aint označeni celobrojni tip, a pokazivači su po prirodi neoznačeni.

# *MPI_Type_create_struct* primer

```
…
struct person_t
{
   int age;
   double height;
   char name[10];
};
…
```

izgled strukture koja će poslužiti kao model za kreiranje novog *MPI* tipa

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Datatype person_type;
int lengths[3] = { 1, 1, 10 };
MPI_Datatype types[3] = { MPI_INT, MPI_DOUBLE, MPI_CHAR };

MPI_Aint displacements[3];
struct person_t dummy_person;
MPI_Aint base_address;
…
```


- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Datatype person_type;
int lengths[3] = { 1, 1, 10 };
MPI_Datatype types[3] = { MPI_INT, MPI_DOUBLE, MPI_CHAR };

MPI_Aint displacements[3];
struct person_t dummy_person;
MPI_Aint base_address;
…
```

u kombinaciji, ova dva niza znače da će novi tip podatka sadržati tri bloka, i to…

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Datatype person_type;
int lengths[3] = { 1, 1, 10 };
MPI_Datatype types[3] = { MPI_INT, MPI_DOUBLE, MPI_CHAR };

MPI_Aint displacements[3];
struct person_t dummy_person;
MPI_Aint base_address;
…
```

…prvi blok, od jednog *MPI_INT* podatka…

```
struct person_t {
   int age;
   double height;
   char name[10];
};
```

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Datatype person_type;
int lengths[3] = { 1, 1, 10 };
MPI_Datatype types[3] = { MPI_INT, MPI_DOUBLE, MPI_CHAR };

MPI_Aint displacements[3];
struct person_t dummy_person;
MPI_Aint base_address;
…
```

…drugi blok, od jednog *MPI_DOUBLE* podatka, i…

```
struct person_t {
   int age;
   double height;
   char name[10];
};
```

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Datatype person_type;
int lengths[3] = { 1, 1, 10 };
MPI_Datatype types[3] = { MPI_INT, MPI_DOUBLE, MPI_CHAR };

MPI_Aint displacements[3];
struct person_t dummy_person;
MPI_Aint base_address;
…
```

…treći blok, od deset *MPI_CHAR* podataka

```
struct person_t {
   int age;
   double height;
   char name[10];
};
```

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Datatype person_type;
int lengths[3] = { 1, 1, 10 };
MPI_Datatype types[3] = { MPI_INT, MPI_DOUBLE, MPI_CHAR };

MPI_Aint displacements[3];
struct person_t dummy_person;
MPI_Aint base_address;
…
```

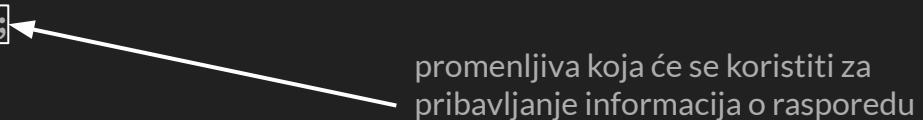
niz koji će sadržati udaljenosti svakog bloka od početka podatka novog tipa

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Datatype person_type;
int lengths[3] = { 1, 1, 10 };
MPI_Datatype types[3] = { MPI_INT, MPI_DOUBLE, MPI_CHAR };

MPI_Aint displacements[3];
struct person_t dummy_person;
MPI_Aint base_address;
…
```



promenljiva koja će se koristiti za pribavljanje informacija o rasporedu podataka unutar strukture

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

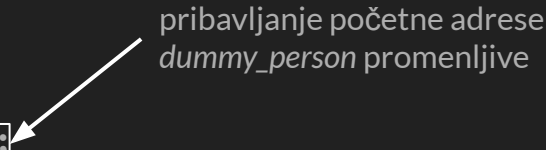
```
…
MPI_Datatype person_type;
int lengths[3] = { 1, 1, 10 };
MPI_Datatype types[3] = { MPI_INT, MPI_DOUBLE, MPI_CHAR };

MPI_Aint displacements[3];
struct person_t dummy_person;
MPI_Aint base_address;
…
```

promenljiva koja će čuvati početnu adresu promenljive *dummy_person*

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

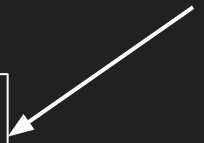



```
…
MPI_Get_address(&dummy_person, &base_address);
MPI_Get_address(&dummy_person.age, &displacements[0]);
MPI_Get_address(&dummy_person.height, &displacements[1]);
MPI_Get_address(&dummy_person.name[0], &displacements[2]);
displacements[0] = MPI_Aint_diff(displacements[0], base_address);
displacements[1] = MPI_Aint_diff(displacements[1], base_address);
displacements[2] = MPI_Aint_diff(displacements[2], base_address);
…
```

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

pribavljanje adresa pojedinačnih podataka iz strukture *dummy_person*



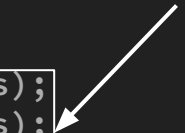
```
…
MPI_Get_address(&dummy_person, &base_address);
MPI_Get_address(&dummy_person.age, &displacements[0]);
MPI_Get_address(&dummy_person.height, &displacements[1]);
MPI_Get_address(&dummy_person.name[0], &displacements[2]);
displacements[0] = MPI_Aint_diff(displacements[0], base_address);
displacements[1] = MPI_Aint_diff(displacements[1], base_address);
displacements[2] = MPI_Aint_diff(displacements[2], base_address);
…
```

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Get_address(&dummy_person, &base_address);
MPI_Get_address(&dummy_person.age, &displacements[0]);
MPI_Get_address(&dummy_person.height, &displacements[1]);
MPI_Get_address(&dummy_person.name[0], &displacements[2]);
displacements[0] = MPI_Aint_diff(displacements[0], base_address);
displacements[1] = MPI_Aint_diff(displacements[1], base_address);
displacements[2] = MPI_Aint_diff(displacements[2], base_address);
…
```

računanje udaljenosti svakog posebnog polja iz strukture koju predstavlja *dummy_person* promenljiva



- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

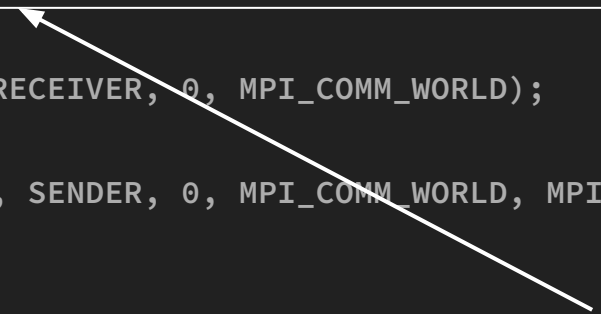
```
…
MPI_Get_address(&dummy_person, &base_address);
MPI_Get_address(&dummy_person.age, &displacements[0]);
MPI_Get_address(&dummy_person.height, &displacements[1]);
MPI_Get_address(&dummy_person.name[0], &displacements[2]);
displacements[0] = MPI_Aint_diff(displacements[0], base_address);
displacements[1] = MPI_Aint_diff(displacements[1], base_address);
displacements[2] = MPI_Aint_diff(displacements[2], base_address);
…
```

kombinacija ovih stvari daje ispravne *displacement*-e, uključujući *padding*, veličine podataka, i sve stvari specifične za jezik i arhitekturu koje utiču na način pakovanja strukture u memoriji

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Type_create_struct(3, lengths, displacements, types, &person_type);
MPI_Type_commit(&person_type);
…
MPI_Send(&buffer, 1, person_type, RECEIVER, 0, MPI_COMM_WORLD);
…
MPI_Recv(&received, 1, person_type, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
…
```



- *primeri/p13_mpi_struct.c*

na kraju preostaje da se pozove funkcija za kreiranje novog tipa, i da se novi tip potvrdi (*commit*-uje)...

# *MPI_Type_create_struct* primer

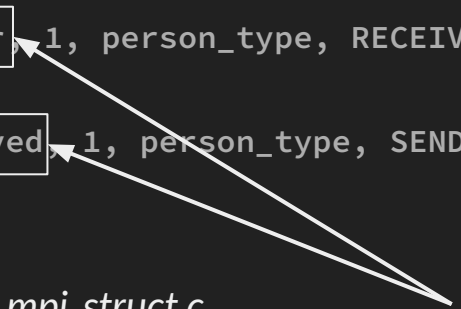
```
…
MPI_Type_create_struct(3, lengths, displacements, types, &person_type);
MPI_Type_commit(&person_type);
…
MPI_Send(&buffer, 1, person_type, RECEIVER, 0, MPI_COMM_WORLD);
…
MPI_Recv(&received, 1, person_type, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
…
```

…nakon čega se može koristiti u funkcijama za razmenu poruka

- *primeri/p13_mpi_struct.c*

# *MPI_Type_create_struct* primer

```
…
MPI_Type_create_struct(3, lengths, displacements, types, &person_type);
MPI_Type_commit(&person_type);
…
MPI_Send(&buffer, 1, person_type, RECEIVER, 0, MPI_COMM_WORLD);
…
MPI_Recv(&received, 1, person_type, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
…
```



podaci koji se zapravo razmenjuju su promenljive tipa strukture *person_t*, odnosno strukture na osnovu koje je kreirana *MPI* struktura *person_type* (isto kao što se za razmenu promenljivih tipa *int* koristi *MPI_INT* kao *datatype* parametar)

- *primeri/p13_mpi_struct.c*

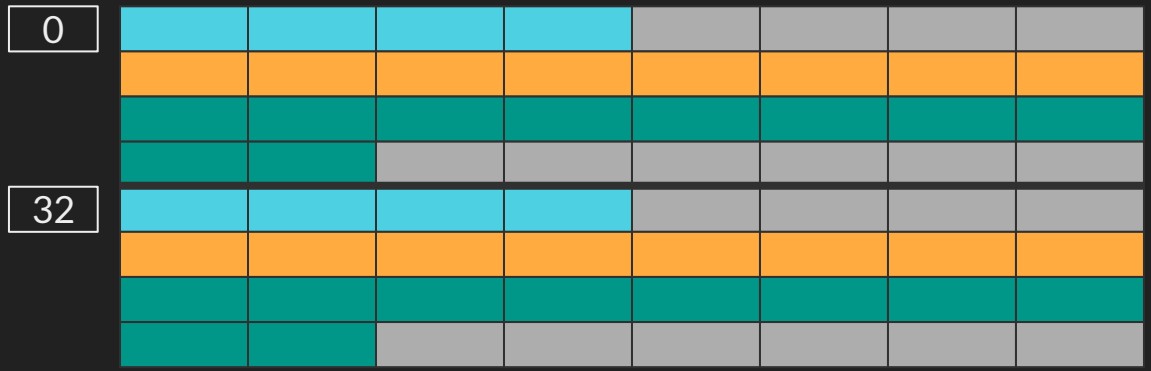
# *MPI_Type_create_struct* primer

izgled strukture u memoriji (*C*-ovsko pakovanje na x64 arhitekturi, *gcc* kompajler)


```
struct person_t {
    int age;
    double height;
    char name[10];
};
```

# *MPI_Type_create_struct* primer

u nizu ovakvih struktura, svaka naredna bi počela na adresi deljivoj sa veličinom najšireg tipa prisutnog u strukturi (u ovom slučaju *double*, odnosno 8 bajtova), tako da bi ovako izgledala dva uzastopna elementa tog niza (npr. po imenu *arr*) :




```
struct person_t {
    int age;
    double height;
    char name[10];
};
```

# *MPI_Type_create_struct*

- *MPI_Type_create_struct* je najgeneralnija funkcija za kreiranje proizvoljnog tipa.
- Omogućava pakovanje potpuno raznorodnih podataka na proizvoljne načine.
- Ostale funkcije kreiraju nove podatke po nekom šablonu.

# *MPI_Type_create_struct*

- Obratiti pažnju na vrednosti *displacement*-a pri kreiranju ovakvih struktura.
- Pokrenuti primer *primeri/p14_mpi_struct_padding.c* i videti na kojoj udaljenosti (*displacement*-u) od početka strukture se nalaze pojedina njena polja.
- U 45. liniji, zameniti mesta *double* i *char* poljima strukture, i videti kako ta promena utiče na *displacement*-e.

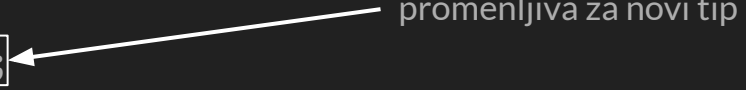
# *MPI_Type_contiguous*

```
int MPI_Type_contiguous(
    int count,
    MPI_Datatype oldtype
    MPI_Datatype *newtype)
```

- Funkcija koja kreira novi tip replicirajući tip *oldtype* na *count* uzastopnih lokacija.
- Svaka naredna replika starog tipa smešta se na lokaciju odmah nakon lokacije koju je zauzela prethodna (što uključuje *padding* i poravnanje).

# *MPI_Type_contiguous* primer

```
…
MPI_Datatype triple_int_type;
if (my_rank == SENDER)
{
    MPI_Type_contiguous(3, MPI_INT, &triple_int_type);
    MPI_Type_commit(&triple_int_type);
}
…
```




- *primeri/p15_mpi_contiguous.c*

# *MPI_Type_contiguous* primer

```
…
MPI_Datatype triple_int_type;
if (my_rank == SENDER)
{
    MPI_Type_contiguous(3, MPI_INT, &triple_int_type);
    MPI_Type_commit(&triple_int_type);
}
…
```

dovoljno je da novi tip poznaje samo proces koji će ga koristiti

- *primeri/p15_mpi_contiguous.c*

# *MPI_Type_contiguous* primer

```
…
MPI_Datatype triple_int_type;
if (my_rank == SENDER)
{
    MPI_Type_contiguous(3, MPI_INT, &triple_int_type);
    MPI_Type_commit(&triple_int_type);
}
…
```

kreira se novi tip koji se sastoji od tri uzastopna *MPI_INT* podatka, i potvrđuje se (*commit*-uje)

- *primeri/p15_mpi_contiguous.c*

# *MPI_Type_contiguous* primer

izgled strukture u memoriji (*C*-ovsko pakovanje na x64 arhitekturi, *gcc* kompajler)


**prvi *MPI_INT***
**drugi *MPI_INT***
**treći *MPI_INT***

- *primeri/p15_mpi_contiguous.c*

# *MPI_Type_contiguous* primer

izgled strukture u memoriji (C-ovsko pakovanje na x64 arhitekturi, *gcc* kompajler)


naredni podatak bi mogao da počne odavde, jer je *int* poravnat na 4 bajta

**prvi *MPI_INT***
**drugi *MPI_INT***
**treći *MPI_INT***

- *primeri/p15_mpi_contiguous.c*

# *MPI_Type_contiguous* primer

proces koji poznaje novi tip,
koristiće ga za slanje

```
…
switch(my_rank)
{
    case SENDER:
    {
        int buffer_sent[3] = {123, 456, 789};
        MPI_Send(&buffer_sent, 1, triple_int_type, RECEIVER, 0, MPI_COMM_WORLD);
        break;
    }
    case RECEIVER:
    {
        int received[3];
        MPI_Recv(&received, 3, MPI_INT, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        break;
    }
}
…
```

- *primeri/p15_mpi_contiguous.c*

# *MPI_Type_contiguous* primer

proces koji ne poznaje novi tip, koristiće za prijem odgovarajuć broj drugog tipa

```
…
switch(my_rank)
{
    case SENDER:
    {
        int buffer_sent[3] = {123, 456, 789};
        MPI_Send(&buffer_sent, 1, triple_int_type, RECEIVER, 0, MPI_COMM_WORLD);
        break;
    }
    case RECEIVER:
    {
        int received[3];
        MPI_Recv(&received, 3, MPI_INT, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        break;
    }
}
…
```

- *primeri/p15_mpi_contiguous.c*

# *MPI_Type_vector*

```
int MPI_Type_vector(
    int count,
    int blocklength,
    int stride,
    MPI_Datatype oldtype,
    MPI_Datatype *newtype)
```

- Funkcija koja kreira novi tip od *count* blokova, od kojih svaki sadrži *blocklength* podataka tipa *oldtype*.
- Udaljenost svakog bloka **od početka prethodnog** izražena je kao proizvod parametra *stride* i veličine tipa *oldtype* (uključujući i eventualan *padding* i poravnanje).

# *MPI_Type_vector* primer

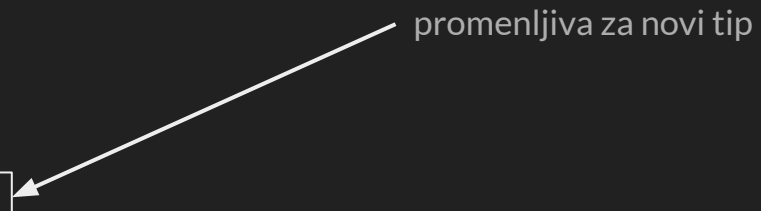
```
…
case SENDER:
{
    MPI_Datatype column_type;
    MPI_Type_vector(3, 1, 3, MPI_INT, &column_type);
    MPI_Type_commit(&column_type);
…
```

dovoljno je da novi tip poznaje samo proces koji će ga koristiti

- *primeri/p16_mpi_vector.c*

# *MPI_Type_vector* primer

```
…
case SENDER:
{
   MPI_Datatype column_type;
   MPI_Type_vector(3, 1, 3, MPI_INT, &column_type);
   MPI_Type_commit(&column_type);
…
```




- *primeri/p16_mpi_vector.c*

# *MPI_Type_vector* primer

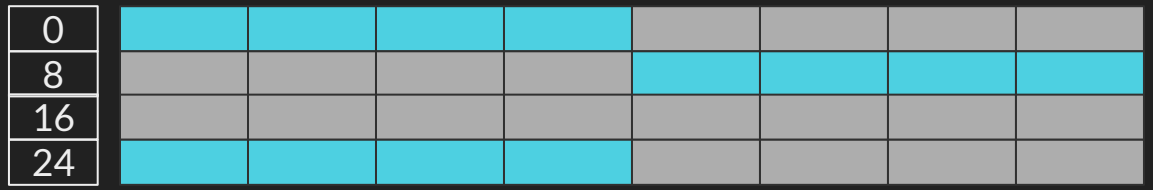
```
…
case SENDER:
{
   MPI_Datatype column_type;
   MPI_Type_vector(3, 1, 3, MPI_INT, &column_type);
   MPI_Type_commit(&column_type);
…
```

kreira se novi tip koji se sastoji od tri bloka od po jednog *MPI_INT* podatka, gde je razmak između početaka uzastopnih blokova po tri *MPI_INT* podatka, i potvrđuje se (*commit*-uje)

- *primeri/p16_mpi_vector.c*

# *MPI_Type_vector* primer

izgled strukture u memoriji (*C*-ovsko pakovanje na x64 arhitekturi, *gcc* kompajler); tri podatka, sa po jednim ***MPI_INT*** podatkom i razmakom od po tri *MPI_INT*-a između početaka uzastopnih elemenata




```
MPI_Type_vector(3, 1, 3, MPI_INT, &column_type)
```

- *primeri/p16_mpi_vector.c*

# *MPI_Type_vector* primer

proces koji poznaje novi tip, koristiće ga za slanje

```
…
case SENDER:
{
   …
   int buffer[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
   MPI_Send(&buffer[0][1], 1, column_type, RECEIVER, 0, MPI_COMM_WORLD);
   break;
}
case RECEIVER:
{
   int received[3];
   MPI_Recv(&received, 3, MPI_INT, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   break;
}
…
```

- *primeri/p16_mpi_vector.c*

# *MPI_Type_vector* primer

slanje može krenuti od bilo kog validnog pokazivača,
ne mora od prvog elementa niza

```
…
case SENDER:
{
   …
   int buffer[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
   MPI_Send(&buffer[0][1], 1, column_type, RECEIVER, 0, MPI_COMM_WORLD);
   break;
}
case RECEIVER:
{
   int received[3];
   MPI_Recv(&received, 3, MPI_INT, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   break;
}
…
```

- *primeri/p16_mpi_vector.c*

# *MPI_Type_vector* primer

proces koji ne poznaje novi tip, koristiće za prijem odgovarajuć broj drugog tipa

```
…
case SENDER:
{
   …
   int buffer[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
   MPI_Send(&buffer[0][1], 1, column_type, RECEIVER, 0, MPI_COMM_WORLD);
   break;
}
case RECEIVER:
{
   int received[3];
   MPI_Recv(&received, 3, MPI_INT, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   break;
}
…
```

- *primeri/p16_mpi_vector.c*

# *MPI_Type_indexed*

```
int MPI_Type_indexed(
    int count,
    const int array_of_blocklengths[],
    const int array_of_displacements[],
    MPI_Datatype oldtype,
    MPI_Datatype *newtype)
```

- Funkcija koja kreira novi tip od *count* blokova, gde je *i*-ti blok veličine *array_of_blocklengths[i].*
- Udaljenost *i*-tog bloka **od početka novog tipa** računa se kao proizvod parametra *array_of_displacements[i]* i veličine tipa *oldtype* (uključujući i eventualan *padding* i poravnanje).

# *MPI_Type_indexed* primer

```
…
case SENDER:
{
        MPI_Datatype triangle_type;
        int lengths[3] = { 1, 2, 3 };
        int displacements[3] = { 0, 3, 6 };
        MPI_Type_indexed(3, lengths, displacements, MPI_INT, &triangle_type);
        MPI_Type_commit(&triangle_type);
…
```

dovoljno je da novi tip poznaje samo proces koji će ga koristiti

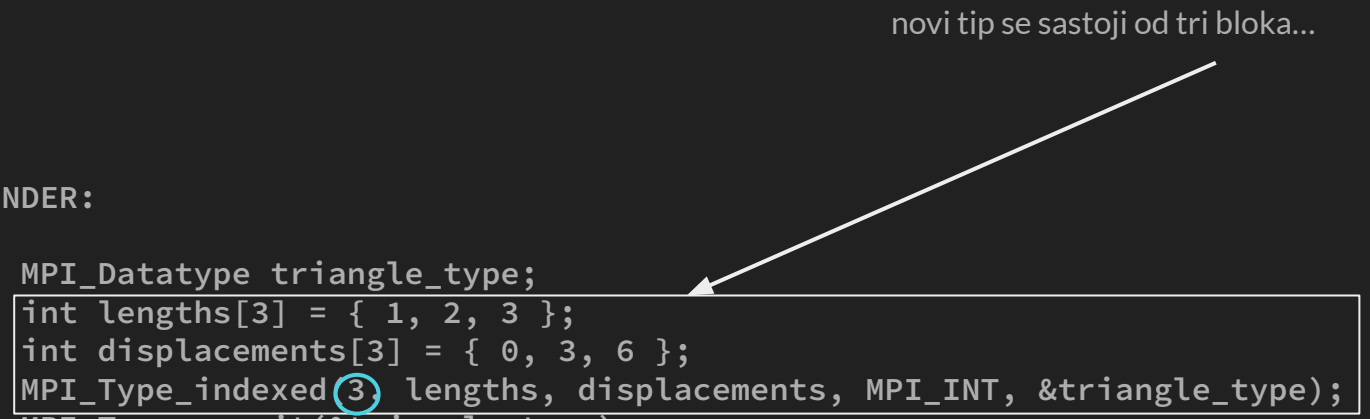
- *primeri/p17_mpi_indexed.c*

# *MPI_Type_indexed* primer

```
…
case SENDER:
{
        MPI_Datatype triangle_type;
        int lengths[3] = { 1, 2, 3 };
        int displacements[3] = { 0, 3, 6 };
        MPI_Type_indexed(3, lengths, displacements, MPI_INT, &triangle_type);
        MPI_Type_commit(&triangle_type);
…
```


- *primeri/p17_mpi_indexed.c*

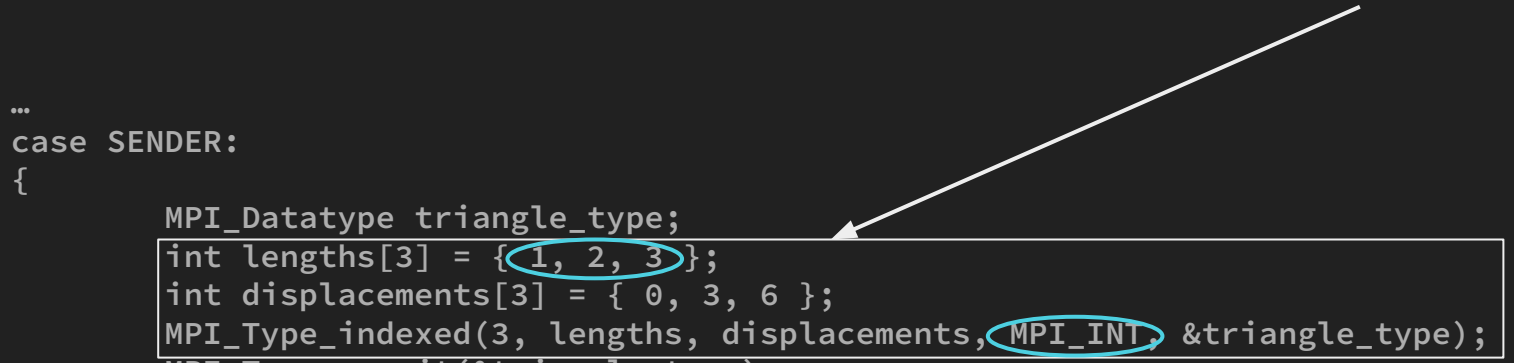
# *MPI_Type_indexed* primer




- *primeri/p17_mpi_indexed.c*

# *MPI_Type_indexed* primer

…koji su redom veličine 1, 2 i 3 *MPI_INT*-a…

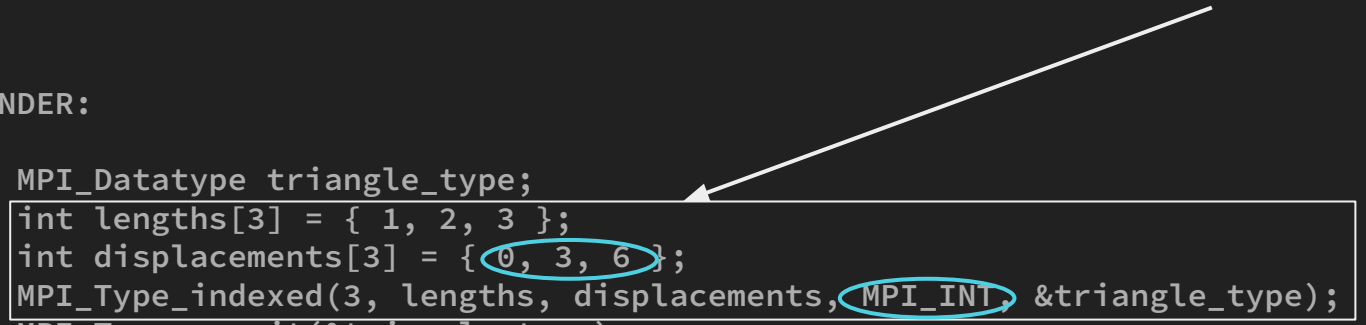


```
…
case SENDER:
{
        MPI_Datatype triangle_type;
        int lengths[3] = { 1, 2, 3 };
        int displacements[3] = { 0, 3, 6 };
        MPI_Type_indexed(3, lengths, displacements, MPI_INT, &triangle_type);
        MPI_Type_commit(&triangle_type);
…
```

- *primeri/p17_mpi_indexed.c*

# *MPI_Type_indexed* primer

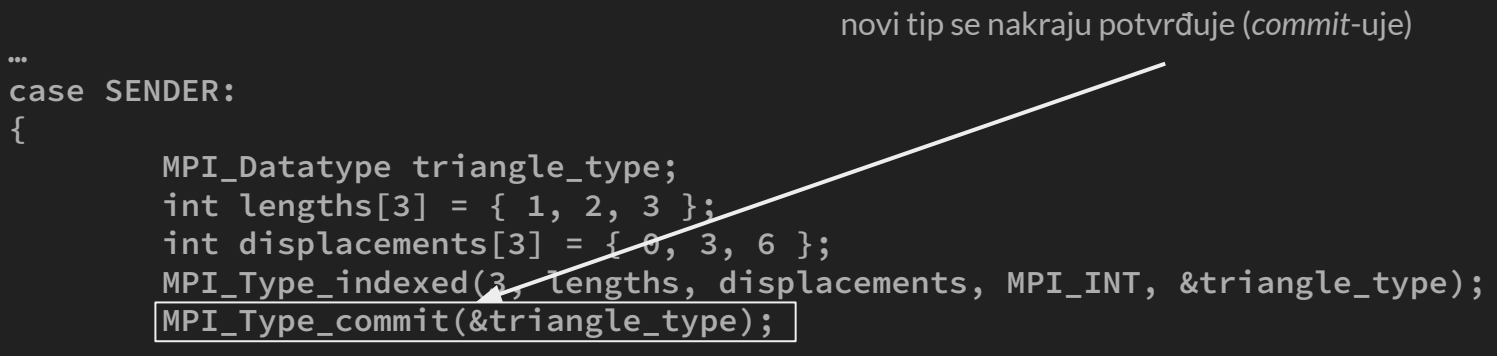
…i udaljeni su, redom, po 0, 3 i 6 veličina *MPI_INT*-a od početka podatka novog tipa



```
…
case SENDER:
{
        MPI_Datatype triangle_type;
        int lengths[3] = { 1, 2, 3 };
        int displacements[3] = { 0, 3, 6 };
        MPI_Type_indexed(3, lengths, displacements, MPI_INT, &triangle_type);
        MPI_Type_commit(&triangle_type);
…
```

- *primeri/p17_mpi_indexed.c*

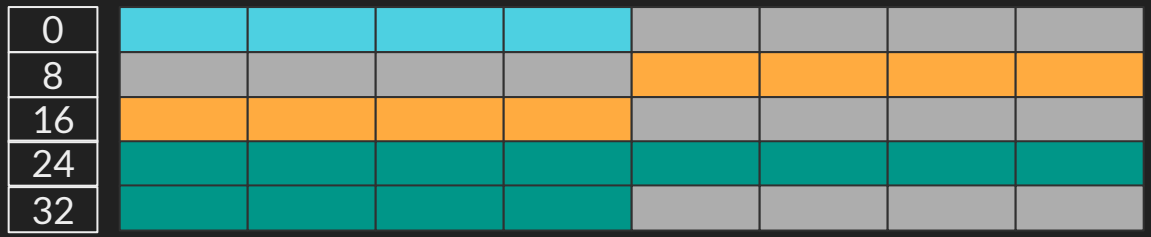
# *MPI_Type_indexed* primer




- *primeri/p17_mpi_indexed.c*

# *MPI_Type_indexed* primer

izgled strukture u memoriji (*C*-ovsko pakovanje na x64 arhitekturi, *gcc* kompajler); tri podatka, sa po jednim ***MPI_INT*** podatkom i razmakom od po tri *MPI_INT*-a između početaka uzastopnih elemenata




```
int lengths[3] = { 1, 2, 3 };
int displacements[3] = { 0, 3, 6 };
MPI_Type_indexed(3, lengths, displacements, MPI_INT, &triangle_type);
```

- *primeri/p17_mpi_indexed.c*

# *MPI_Type_indexed* primer

proces koji poznaje novi tip, koristiće ga za slanje

```
…
case SENDER:
{
   …
   int buffer[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
   MPI_Send(buffer, 1, triangle_type, RECEIVER, 0, MPI_COMM_WORLD);
   break;
   }
case RECEIVER:
{
   int received[6];
   MPI_Recv(&received, 6, MPI_INT, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   break;
}
…
```

- *primeri/p17_mpi_indexed.c*

# *MPI_Type_indexed* primer

proces koji ne poznaje novi tip, koristiće za prijem odgovarajuć broj drugog tipa

```
…
case SENDER:
{
   …
   int buffer[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
   MPI_Send(buffer, 1, triangle_type, RECEIVER, 0, MPI_COMM_WORLD);
   break;
   }
case RECEIVER:
{
   int received[6];
   MPI_Recv(&received, 6, MPI_INT, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   break;
}
…
```

- *primeri/p17_mpi_indexed.c*

# Ostale funkcije za rad sa izvedenim tipovima

- Pogledati ostale funkcije za rad sa izvedenim tipovima podataka u zvaničnoj dokumentaciji *OpenMPI*-a (https://docs.open-mpi.org/en/v5.0.x/man-openmpi/man3/index.html) .
- Zanimljiv nastavak već obrađenih funkcija mogle bi biti funkcije *MPI_Type_create_subarray* i *MPI_Type_create_resized*.
- *MPI_Type_create_resized* je naročito korisna kada se koriste funkcije *MPI_Gather i MPI_Scatter* za kolektivnu komunikaciju, ili kada se šalje/prima više od jednog izvedenog podatka.

# Zadaci

# Zadatak 9

- Napisati program u *C*-u koji korišćenjem *OpenMPI*-a šalje podatke iznad glavne dijagonale matrice *NxN* iz procesa 0.
- Podatke treba da primi proces 1 i da ih smesti ispod glavne dijagonale matrice iste veličine.
- Za slanje i prijem iskoristiti samo po jednu *MPI_Send* i *MPI_Recv* funkciju, koje će odjednom poslati podatke sa odgovarajućeg mesta i smestiti ih po prijemu na odgovarajuće mesto.
- Za slanje i prijem kreirati odgovarajuće *MPI* tipove.
- Napisati program tako da radi za proizvoljno *N*.

# Zadatak 10

- Napisati program u *C*-u koji korišćenjem *OpenMPI*-a šalje kolonu po kolonu matrice *NxM* iz procesa 0.
- Proces 1 prihvata jednu po jednu kolonu i smešta ih redom u vrste matrice *MxN*.
- Za slanje i prijem svake kolone/vrste iskoristiti po jedan par *MPI_Send* i *MPI_Recv* kojima će se odjednom poslati čitava kolona i prihvatiti čitava vrsta.
- Za slanje i prijem kreirati odgovarajuće *MPI* tipove.
- Napisati program tako da radi za proizvoljne veličine matrice.