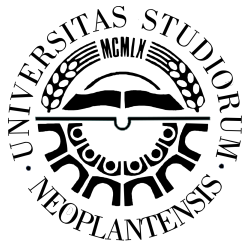
OpenMPI

Рачунарски системи високих перформанси

Горана Гојић Вељко Петровић

Факултет техничких наука
Универзитет у Новом Саду

Рачунарске вежбе, Зимски семестар 2020/2021.



Message Passing Interface (MPI)

Стандард који прописује комуникацију разменом порука на различитим паралелним архитектурама.

MPI 1.x (1994.)
MPI 2.x (1997.)
MPI 3.x (2012.)
MPI 4.x (2018.)

Подршка за C, (C++,) Fortran

Постоје различите имплементације MPI (комерцијалне и отвореног кода).

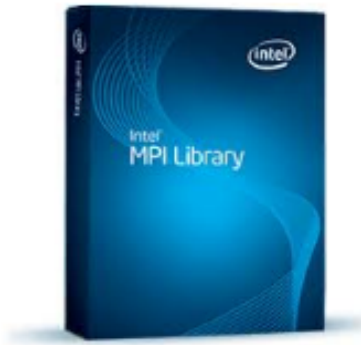
MPI имплементације

OPEN MPI

MVAPICH

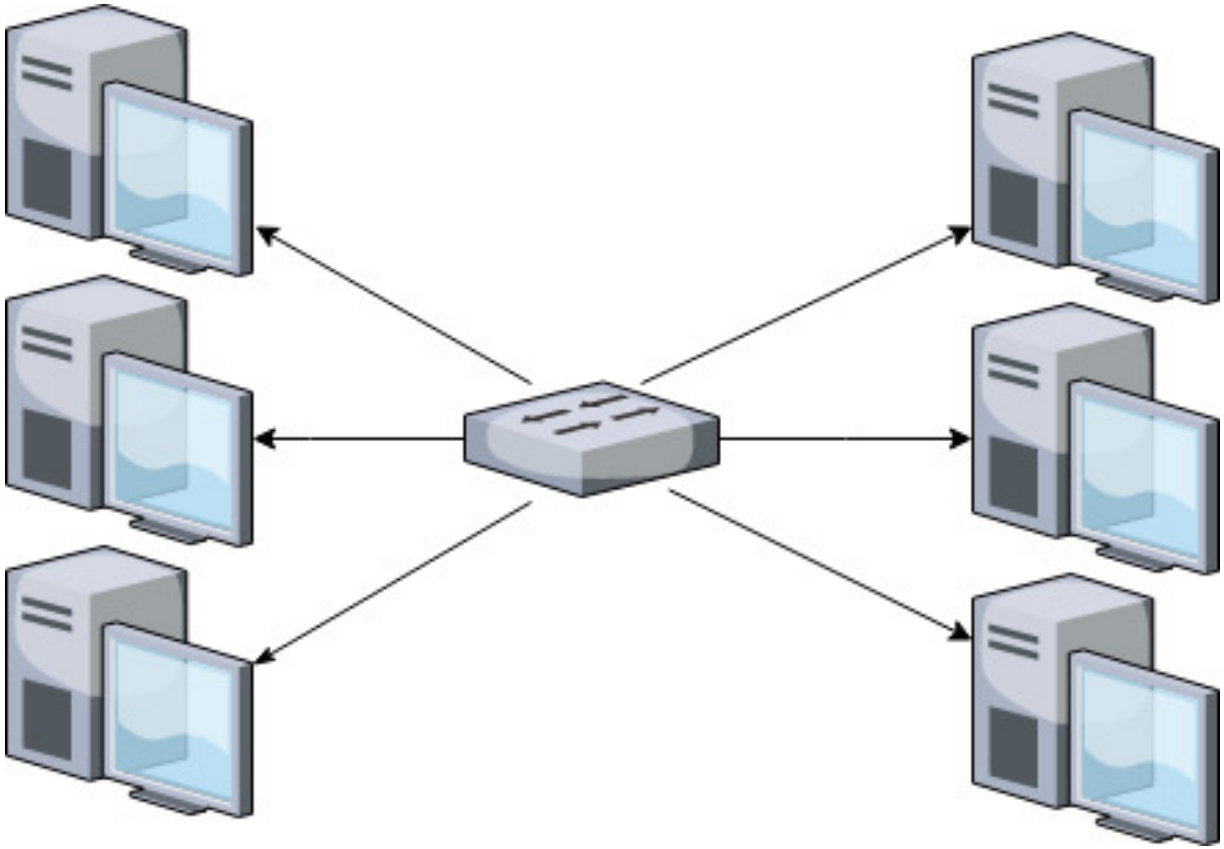
MPICH-V

MPI Implementation for Volatile resources



IBM Spectrum MPI

MPI циљна архитектура



Формат OpenMPI програма

```
#include <mpi.h>

int main(int argc, char *argv[]) {

    MPI_Init(&argc, &argv);

    // MPI code

    MPI_Finalize();

    return 0;
}
```

Компајлирање MPI програма

Позиционирати се у директоријум у којем се налази изворни код MPI програма и унети:

```
mpicc <izvorna_datoteka>
```

`mpicc` је омотачка скрипта за `gcc`, па се при компајлирању могу навести опције `gcc` компајлера.

Покретање:

```
mpiexec [-np <N>] <izvrsna_datoteka>
```

-np <N> - опција за задавање броја процеса који ће бити креирани при покретању програма.

Пример 1: Hello World!

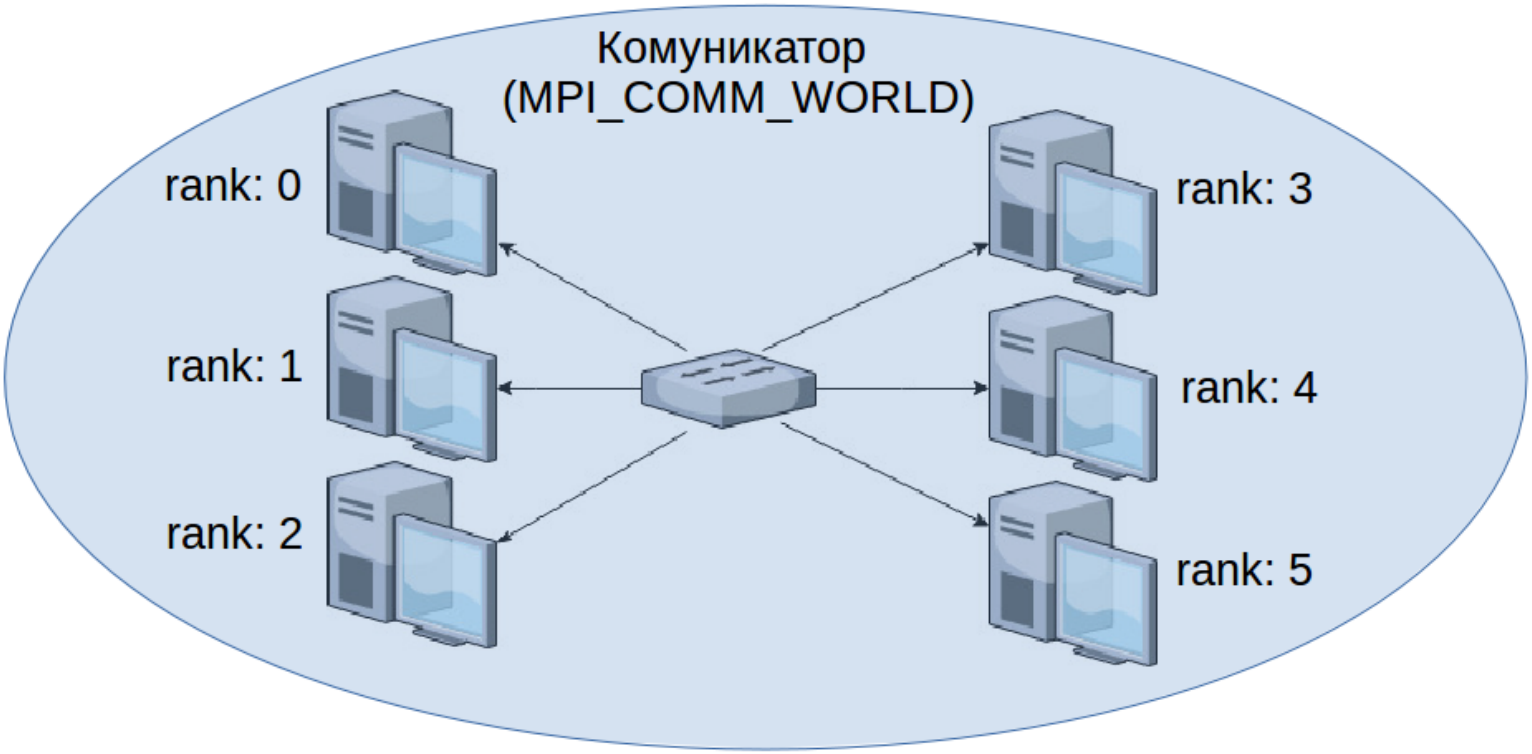
```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[]) {
    int size, rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    printf("Hello World iz %d/%d.\n", rank, size);

    MPI_Finalize();

    return 0;
}
```

MPI основни концепти

- Комуникатор (енг. *communicator*)
- Point to Point комуникација (енг. *Point-to-Point communication*)
- Колективна комуникација (енг. *Collective communication*)

Комуникатори

Логички гледано, **комуникатор** представља групу процеса унутар које сваки процес има свој **ранк** (односно идентификатор).



`MPI_COMM_SELF`, `MPI_COMM_NULL`

Комуникатори

Током извршавања MPI истовремено може да постоји више комуникатора.

`MPI_Comm` тип података.

Функције за прављење комуникатора:

```
# креирање новог комуникатора дељењем постојећег
MPI_Comm_split(MPI_Comm comm, int color, int key,
        MPI_Comm* newcomm);

# нови комуникатор је копија постојећег
int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)

# и друге
```

Задатак 1: Комуникатори

Направити OpenMPI C програм који коришћењем функције `MPI_Comm_split` на основу подразумеваног прави два нова комуникатора. Процесе поделити у два комуникатора на основу парности ранка унутар `MPI_COMM_WORLD` комуникатора. При том релативни поредак процеса унутар комуникатора треба да буде исти као и у подразумеваном комуникатору. Сваки процес на стандардни излаз треба да испише свој ранк унутар `MPI_COMM_WORLD` и новоформираног комуникатора.

Пример исписа за један процес:

```
MPI_COMM_WORLD rank: 0/4 - ncomm rank: 0/2
```

Решење: датотека `communicators.c`, директоријум `resenja`.

MPI типови података

Зарад портабилности MPI стандард дефинише типове података.

MPI тип податка	**C тип податка**
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

[1] Комплетна листа MPI типова података по MPI2 стандарду.

Point to Point комуникација

Комуникација два процеса. Један процес **шаље** поруку, други процес **прима** поруку.

Функције за размену порука:

```
int MPI_Send(
  const void *buf,
  int count,
  MPI_Datatype datatype,
  int dest,
  int tag,
  MPI_Comm comm);
```

```
int MPI_Recv(
  void *buf,
  int count,
  MPI_Datatype datatype,
  int source,
  int tag,
  MPI_Comm comm,
  MPI_Status *status);
```

[1] OpenMPI 2.0 MPI_Recv docs
[2] OpenMPI 2.0 MPI_Send docs

Пример 2: Point to Point комуникација

```
// ...
if (rank == 0) {
    int message = 1;
    printf("Proces %d salje poruku procesu %d.\n", rank, 1);
    MPI_Send(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (rank == 1) {
    int message;
    printf("Proces %d treba da primi poruku od procesa %d.\n",
           rank, 0);
    MPI_Recv(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, NULL);
    printf("Proces %d primio poruku %d od procesa %d.\n",
           rank, message, 0);
}
// ...
```

Режими Point-to-Point комуникације

П2П комуникациони режими слања поруке:

- **Synchronous** (`MPI_Ssend`)- пошиљалац шаље захтев за слање поруке, након што прималац одговори порука се шаље (*handshake* протокол)
- **Buffered** (`MPI_Bsend`) - по иницирању слања порука се шаље у бафер одакле је прималац може преузети
- **Standard** (`MPI_Send`) - може имати карактеристике `buffered` или `synchronous` режима **(подразумевано)**
- **Ready** (`MPI_Rsend`) - претпоставља се да је процес који прима поруку већ чека на поруку у тренутку иницирања слања

За пријем поруке постоји само један режим и порука се сматра примљеном када је преузета и може даље да се користи.

Point-to-Point комуникација

Да би **процес2** примио поруку коју му шаље **процес**1 мора да важи:

- Да `comm` параметар оба процеса има исту вредност,
- Да параметар `dest` процеса 1 има исту вредност као и параметар `source` процеса 2 или да је вредност параметра процеса 2 постављена на `MPI_ANY_SOURCE`,
- Да параметар `tag` има исту вредност за оба процеса или да је вредност параметра `tag` `MPI_ANY_TAG`

Пример: `wrong_send_recv.c`, директоријум `primeri`.

Типови Point-to-Point комуникације

Слање и пријем поруке могу бити:

- **блокирајући** - Ако `MPI_Send` блокирајућа, контрола тока се неће вратити позиваоцу функције све док услов слања не буде испуњен. Након изласка из функције бафер поруке може бити безбедно преписан. Ако је `MPI_Recv` блокирајућа контрола се не враћа позиваоцу функције све док порука не буде преузета **(подразумевано)**.
- **неблокирајући** - Из `MPI_Send`, тј. `MPI_Recv` се излази након иницијације слања, тј. примања поруке. Када се појави потреба за коришћењем изворног односно одредишног бафера, потребно је претходно проверити да ли је податак послат тј. да ли је стигао.

```
MPI_{I}[S, B, R]Send(...), MPI_{I}Recv(...)
```

Задатак 2: Пинг понг

Направити OpenMPI програм имплементиран у C програмском језику који симулира играње пинг понга између два процеса. Лоптицу симулирати променљивом типа `int`. Увећати ову променљиву сваки пут када неки од процеса удари лоптицу, односно, пре него што неки од процеса пошаље променљиву другом процесу и исписати одговарајућу поруку.

Формат очекиваног исписа:

```
p0 sent ping_pong_count to p1 and incremented it to 1.
p1 received ping_pong_count 1 from p0.
p1 sent ping_pong_count to p0 and incremented it to 2.
```

Напомене:

- Претпоставка је да ће програм бити позван опцијом `-np 2` и од овога се не мора штитити.
- Не значи да програм није исправан уколико испис на екрану није у очекиваном редоследу.

Задатак 3: Пинг понг - секвенцијални испис

Модификовати основни пинг понг задатак тако да се испис на стандардни излаз одвија у редоследу у којем процеси ударају пинг понг лоптицу. Програм покренути са три процеса - процеси 0 и 1 играју пинг понг, док трећи процес исписује поруке на стандардни излаз. Сваки пут када неки од процеса играча удари лоптицу, он процесу штампачу шаље поруку коју треба исписати на стандардни излаз. Поруке за испис процесу штампачу стижу у произвољном редоследу, али он треба да их испише у редоследу који одговара секвенцијалном извршавању програма. Све поруке су исте дужине. Пинг понг се игра до 9.

Пример извршавања:

```
p0 sent ping_pong_count to p1 and incremented it to 1.
p1 sent ping_pong_count to p0 and incremented it to 2.
p0 sent ping_pong_count to p1 and incremented it to 3.
```

Задатак 4: Прстен

Написати OpenMPI C програм који прослеђује жетон између процеса по принципу прстена. Жетон је представљен бројем -1 и поседује га процес ранга 0. Сви процеси осим последњег шаљу жетон процесу са рангом за један већи од свог. Последњи процес (процес са највећим рангом у комуникатору) жетон прослеђује назад процесу ранга 0. Након што процес ранга 0 прими жетон, програм се завршава. Исписати поруку на стандардни излаз сваки пут када неки од процеса прими жетон.

Формат очекиваног исписа:

```
Process 1 received token -1 from process 0
Process 2 received token -1 from process 1
Process 3 received token -1 from process 2
Process 0 received token -1 from process 3
```

П2П: Неблокирајућа комуникација

`MPI_Isend` (и други режими), `MPI_Irecv`

Процес иницира слање или примање поруке, али **не чека на завршетак** операције.

```
int MPI_Test(
  MPI_Request *request,
  int *flag,
  MPI_Status *status)
```

Тестира стање захтева и поставља променљиву `flag` на `true` уколико је захтев извршен, односно на `false` уколико није.

```
int MPI_Wait(
  MPI_Request *request,
  MPI_Status *status)
```

Тестира да ли је захтев извршен и завршава се када се захтев изврши. Функција је блокирајућа.

[1] MPI_Test docs
[1] MPI_Wait docs

Пример 3: send_recv_nonblocking.c

```
if (rank == 0) {
    MPI_Request send_request;
    char *message = "Zdravo!";

    MPI_Issend(message, 8, MPI_CHAR, 1, 0, MPI_COMM_WORLD,
               &send_request);
    printf("Proces %d inicirao slanje poruke.\n", rank);

    printf("Proces radi nesto drugo dok se poruka salje.");

    int flag = 0;
    MPI_Test(&send_request, &flag, MPI_STATUS_IGNORE);
    if (flag != 0) /* poslato */ else /* nije */
} else /* ... */
```

Пример 3: send_recv_nonblocking.c

```
if (rank == 0) /* ... */
} else {
    MPI_Request receive_request;
    char message[8];

    MPI_Irecv(message, 8, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
              &receive_request);
    printf("Proces %d inicirao primanje poruke.\n", rank);

    printf("Proces radi nesto drugo dok se poruka prima...");

    MPI_Wait(&receive_request, MPI_STATUS_IGNORE);
    printf("Proces %d primio poruku: \"%s\"\n", rank, message);
}
```

Задатак 5: Пинг понг - неблокирајући секвенцијални испис

Модификовати задатак 4 тако да слање порука процесу штампачу буде неблокирајуће. При том се обезбедити да програм ради коректно, односно да се не деси да порука која није послата буде преписана новом поруком пре него се стара пошаље.

Решење: датотека `ping_pong_printf_async.c`

П2П: динамичка комуникација

Некада поруке које процеси размењују нису фиксне дужине. Тада је прво потребно очитати дужину поруке, алоцирати бафер за поруку, па тек онда започети њено примање.

```
MPI_Probe(
  int source,
  int tag,
  MPI_Comm comm,
  MPI_Status* status)
```

Симулира примање поруке. Попуњава `status` поље.

```
MPI_Get_count(
  const MPI_Status *status,
  MPI_Datatype datatype,
  int *count)
```

Очитава број примљених података типа `datatype` на основу `status` поља.

[1] MPI_Probe docs
[1] MPI_Get_count docs

Пример 4: dynamic_communication.c

```
if (rank == 0) {
    int size = rand() % 10 + 1;
    char *message = (char *) calloc(size + 1, sizeof(char));

    for (int i = 0; i < size; i++) {
        message[i] = 'a';
    }

    MPI_Send(message, size + 1, MPI_CHAR, 1, 0,
             MPI_COMM_WORLD);
    free(message);
} else {
    /* ... */
}
```

Пример 4: dynamic_communication.c

```
if (rank == 0) {
    /* ... */
} else {
    MPI_Status status; int size;
    MPI_Probe(0, 0, MPI_COMM_WORLD, &status);
    MPI_Get_count(&status, MPI_CHAR, &size);

    char *message = (char *) malloc(size * sizeof(char));
    MPI_Recv(message, size, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);

    printf("Primljena poruka: %s.\n", message);
}
```

Задатак 6: Пинг понг - порука варијабилне дужине

Модификовати задатак 4 тако да се процесу принтеру шаљу поруке променљиве дужине. Користити функције `MPI_Probe` и `MPI_Get_count`. Процеси могу да изврше максимално 999 размена лоптицом.

Решење: датотека `ping_pong_printf_variablelen.c`

Колективна комуникација

(енг. *collective communication*)

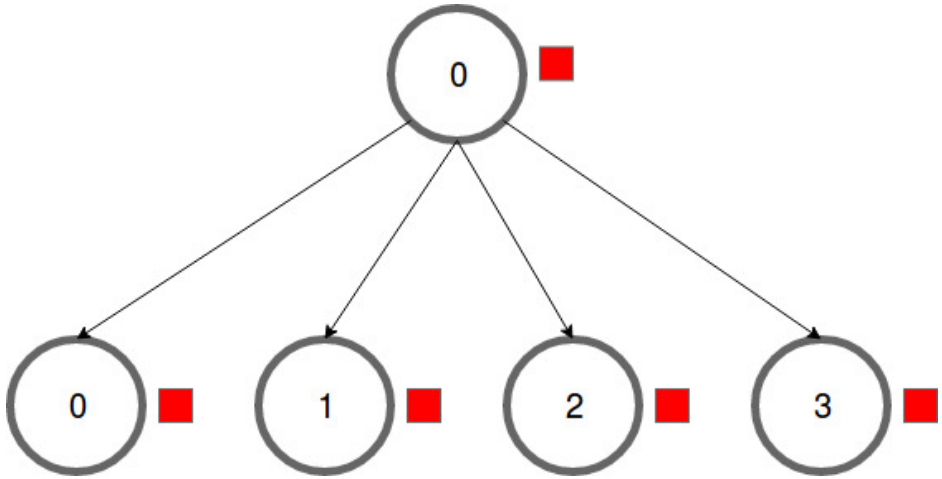
Комуникација **свих** процеса унутар једног комуникатора.

Врсте колективне комуникације:

- **Broadcast**
- **Scatter**
- **Gather**
- **AllGather**
- **Reduction**
- **AllReduction**
- ...

Колективна комуникација: Broadcast

```
int MPI_Bcast(
  void *buffer,
  int count,
  MPI_Datatype datatype,
  int root,
  MPI_Comm comm);
```



Процес чији је ранк једнак вредности `root` параметра шаље поруку свим осталим процесима из комуникатора, укључујући и себе.

[1] MPI_Bcast docs

Пример 5: bcast.c

```c
int main(int argc, char *argv[]) {
    int rank, root = 0;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int token;
    if (rank == root) token = 123;
    MPI_Bcast(&token, 1, MPI_INT, root, MPI_COMM_WORLD);
    printf("Proces %d primio token %d.\n", rank, token);

    MPI_Finalize();

    return 0;
}
```

Задатак 7: Broadcast

Написати OpenMPI C имплементацију `MPI_Bcast` функције коришћењем `MPI_Send` и `MPI_Recv` функција. Ранк процеса који емитује вредност жетона се уноси као аргумент позива програма. Вредност жетона који се емитује је -1. Након што `root` процес (процес који емитује вредност жетона) пошаље жетон исписати поруку о томе. Након што сваки од преосталих процеса прими жетон, исписати поруку и вредност примљеног жетона.

Формат очекиваног исписа:

```
Proces 0 poslao zeton -1.
Proces 1 primio zeton -1.
Proces 2 primio zeton -1.
Proces 3 primio zeton -1.
```

Задатак 8: Колективна комуникација над подскупом комуникатора

У колективној комуникацији учествују сви процеси унутар комуникатора. Међутим, при прешавању комплекснијих проблема може се појавити потреба да се неки податак пошаље само делу процеса комуникатора. Установили смо да коришћење метода колективне комуникације може бити ефикасније у односу на појединачно позивање `MPI_Send` и `MPI_Recv` за сваки од процеса у комуникатору којима треба проследити податак.

Како бисте податак послали само делу процеса у неком комуникатору коришћењем колективне комуникације?

Задатак 9: Колективна комуникација над подскупом комуникатора

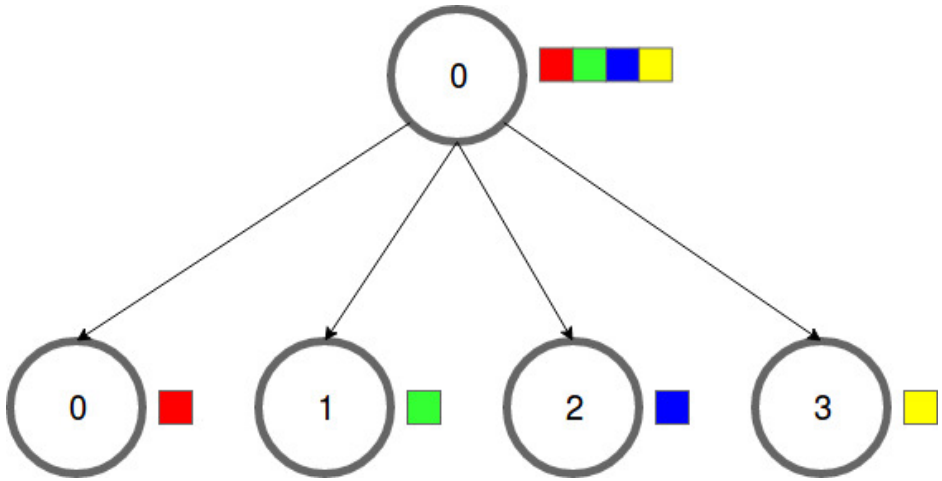
У колективној комуникацији учествују сви процеси унутар комуникатора. Међутим, при прешавању комплекснијих проблема може се појавити потреба да се неки податак пошаље само делу процеса комуникатора. Установили смо да коришћење метода колективне комуникације може бити ефикасније у односу на појединачно позивање `MPI_Send` и `MPI_Recv` за сваки од процеса у комуникатору којима треба проследити податак.

Како бисте податак послали само делу процеса у неком комуникатору коришћењем колективне комуникације?

Одговор: Направити нови комуникатор за процесе којима треба послати података и користити колективну комуникацију на нивоу новонаправљеног комуникатора.

Колективна комуникација: Scatter

```
int MPI_Scatter(
    const void *sendbuf,
    int sendcount,
    MPI_Datatype sendtype,
    void *recvbuf,
    int recvcount,
    MPI_Datatype recvtype,
    int root,
    MPI_Comm comm);
```



Коренски процес шаље делове података процесима из комуникатора. Сваки процес добија "парче" податка исте величине.

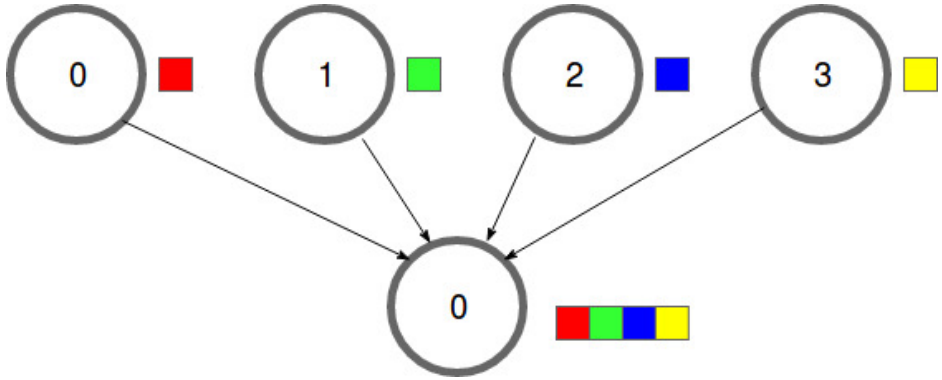
[1] MPI_Scatter docs

Пример 6: scatter.c

```
int main(int argc, char *argv[]) {
    /* ... */

    int *data = NULL, *partial_data = NULL;
    int piecelen = datalen / size;
    if (rank == root) {
        /* inicijalizacija data niza */
    }
    partial_data = (int *) malloc(sizeof(int) * piecelen);

    MPI_Scatter(data, piecelen, MPI_INT, partial_data,
                piecelen, MPI_INT, root, MPI_COMM_WORLD);

    /* ... */
    return 0;
}
```

Колективна комуникација: Gather

```
int MPI_Gather(
    const void *sendbuf,
    int sendcount,
    MPI_Datatype sendtype,
    void *recvbuf,
    int recvcount,
    MPI_Datatype recvtype,
    int root,
    MPI_Comm comm);
```



Коренски процес прима податке од процеса из комуникатора. Сваки процес шаље "парче" података исте величине.

[1] MPI_Gather docs

Пример 7: gather.c

```
int main(int argc, char *argv[]) {
    int *data = NULL, *partial_data = NULL;
    int piecelen = datalen / size;
    partial_data = (int *) malloc(sizeof(int) * piecelen);
    /* ... */

    if (rank == root)
        data = (int *) malloc(sizeof(int) * datalen);

    MPI_Gather(partial_data, piecelen, MPI_INT, data,
               piecelen, MPI_INT, root, MPI_COMM_WORLD);

    /* ... */

    return 0;
}
```

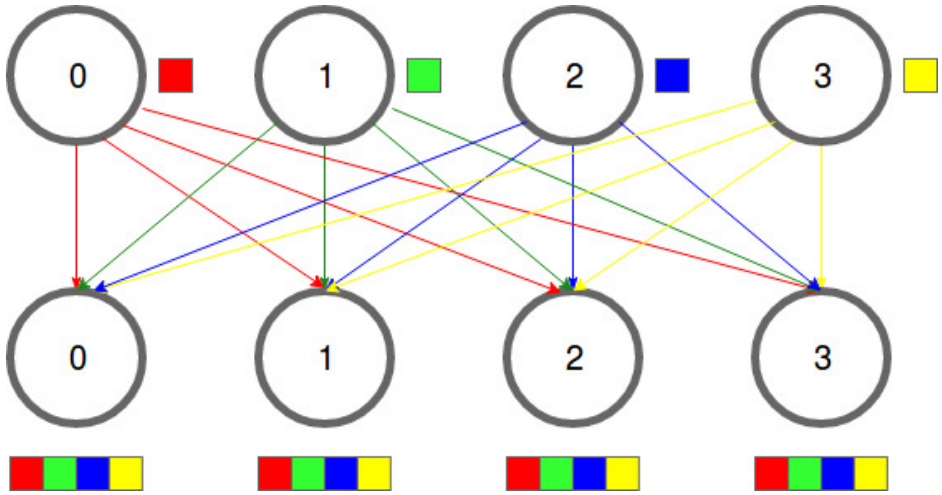
Задатак 10: Рачунање средње вредности

Направити OpenMPI C програм који рачуна средњу вредност елемената низа у више процеса коришћењем функција `MPI_Scatter` и `MPI_Gather`. Програм написати тако да:

- Коренски процес иницијализује низ дужине n насумично изгенерисаним целим бројевима. Дужина низа мора бити дељива бројем покренутих процеса.
- Разделити изгенерисани низ на једнаке делове између свих процеса.
- Сваки процес треба да израчуна суму елемената низа који су му прослеђени.
- Након што су све парцијалне суме срачунате, пребацују се назад коренском процесу који од парцијалних сума прави коначну суму коју дели укупним бројем елемената и исписује средњу вредност низа.

Колективна комуникација: AllGather

```
int MPI_Allgather(
    const void *sendbuf,
    int  sendcount,
    MPI_Datatype sendtype,
    void *recvbuf,
    int recvcount,
    MPI_Datatype recvtype,
    MPI_Comm comm);
```



Ефективно позив `MPI_Gather` праћен `MPI_Bcast` позивом.

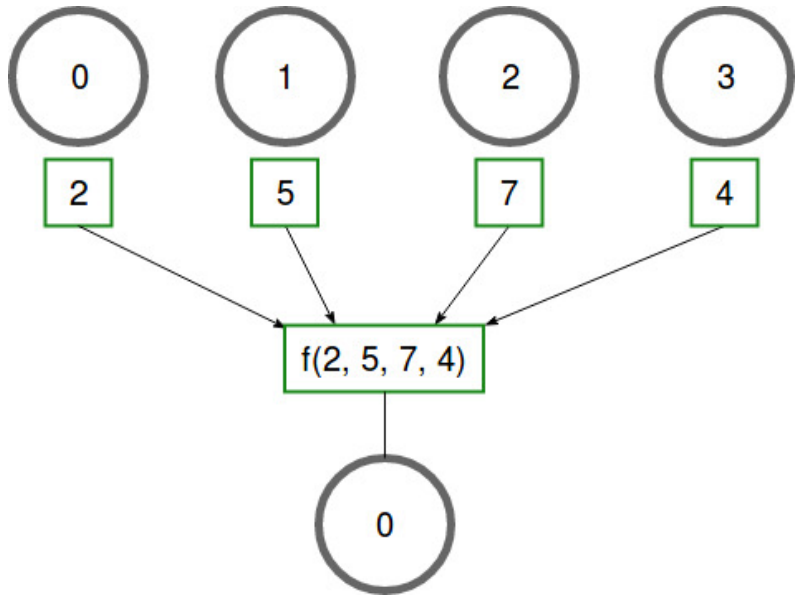
[1] MPI_Allgather docs

Пример 8: allgather.c

```c
int main(int argc, char *argv[]) {

    /* ... */

    int *data = (int *) malloc(sizeof(int) * size);

    int token = rank;
    MPI_Allgather(&token, 1, MPI_INT, data,
                  1, MPI_INT, MPI_COMM_WORLD);

    free(data);

    /* ... */

    return 0;
}
```

Колективна комуникација: Reduce

```
int MPI_Reduce(
  const void *sendbuf,
  void *recvbuf,
  int count,
  MPI_Datatype datatype,
  MPI_Op op,
  int root,
  MPI_Comm comm);
```



Процеси унутар комуникатора шаљу податке коренском процесу који врши редукцију над подацима задатом функцијом (нпр. `MPI_SUM` `MPI_AND`...)

[1] MPI_Reduce docs

Пример 9: reduce.c

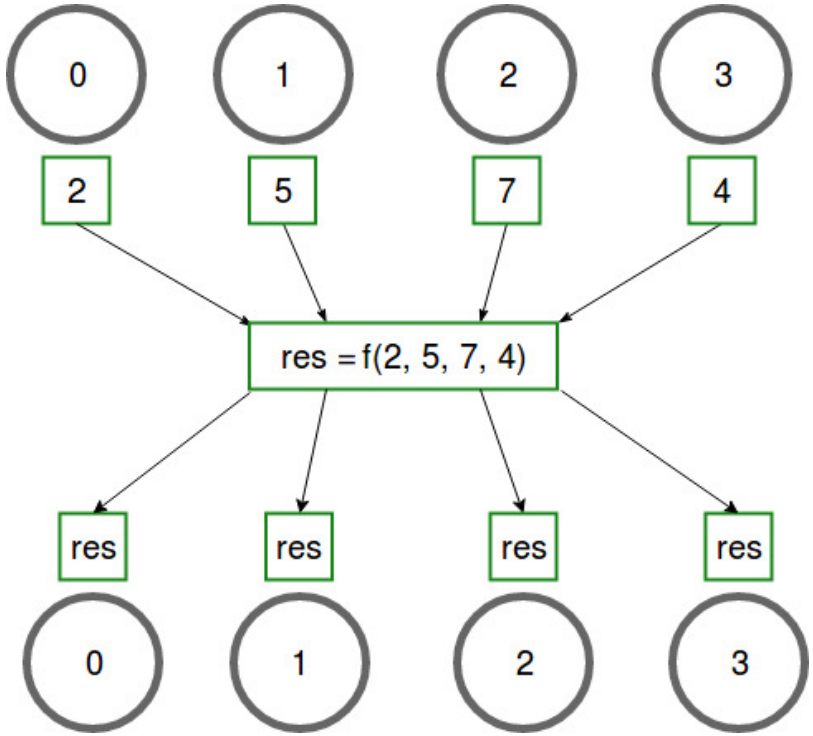
```
int main(int argc, char *argv[]) {
    /* ... */

   int token = rank, result;
    MPI_Reduce(&token, &result, 1, MPI_INT,
               MPI_SUM, root, MPI_COMM_WORLD);

    printf("Proces %d: result = %d.\n", rank, result);

    MPI_Finalize();

    return 0;
}
```

Задатак 11: Рачунање средње вредности 2

Модификовати задатак који рачуна средњу вредност елемената низа тако да се у одговарајућем кораку користи функција `MPI_Reduce`.

Колективна комуникација: AllReduce

```
int MPI_Allreduce(
  const void *sendbuf,
  void *recvbuf,
  int count,
  MPI_Datatype datatype,
  MPI_Op op,
  MPI_Comm comm);
```



Ефективно позив `MPI_Reduce` праћен `MPI_Bcast` позивом.

[1] MPI_Allreduce docs

Пример 10: allreduce.c

```c
int main(int argc, char *argv[]) {
    /* ... */

    int token = rank, result;
    MPI_Allreduce(&token, &result, 1, MPI_INT,
                  MPI_SUM, MPI_COMM_WORLD);

    printf("Proces %d: result = %d.\n", rank, result);

    /* ... */

    return 0;
}
```

Задатак 12: Множење матрице и вектора - **домаћи**

Написати OpenMPI C програм за множење квадратне матрице и вектора. Улазна матрица и вектор садрже разломљене бројеве у једнострукој прецизности и подразумева се да ће димензије матрице и вектора бити одговарајуће. Улазни подаци за задатак су изгенерисани и дати у h5 формату. Такође, дат је костур решења са примерима како учитати матрицу/вектор из h5 датотеке и исписати их на стандардни излаз (директоријум `MatrixVectorMultiplication`). За детаље око компајлирања и покретања решења погледати `README.md` датотеку.

Имплементирати:

- Секвенцијални алгоритам за множење матрице и вектора. Мерити време извршавања и исписати га на стандардни излаз.
- OpenMPI C алгоритам за множење матрице и вектора. Мерити време извршавања и исписати га на стандардни излаз. Оставити могућност да се на стандардни излаз испише и резултат рачунања.

Задатак 13: Множење матрица - **домаћи**

Написати OpenMPI C програм за множење две квадратне матрице. Обе улазне матрице садрже бројеве у једнострукој прецизности. Улазни подаци за задатак су изгенерисани и дати у h5 формату. Такође, дат је костурешења са примерима како учитати матрицу из h5 датотеке и исписати их на стандардни излаз (директоријум MatrixMultiplication). За детаље око компајлирања и покретања решења, погледати README.md датотеку.

Имплементирати:

- Секвенцијални алгоритам за множење две квадратне матрице. Мерити време извршавања и исписати га на стандардни излаз.
- OpenMPI C алгоритам за множење две квадратне матрице. Мерити време извршавања и исписати га на стандардни излаз. Оставити могућност исписа резултата. У симулацијама на једном рачунару OpenMPI решење не мора бити брже од секвенцијалног.

Корисни материјали

- MPI стандард документација
- OpenMPI документација
- Peter S. Pacheco "Parallel Programming with MPI"
- Victor Eijkhout "Parallel Computing" (бесплатна онлајн верзија књиге)
- MPI туторијал