OpenMPI — део 2

Рачунарски системи високих перформанси

Петар Трифуновић Вељко Петровић

Факултет техничких наука
Универзитет у Новом Саду

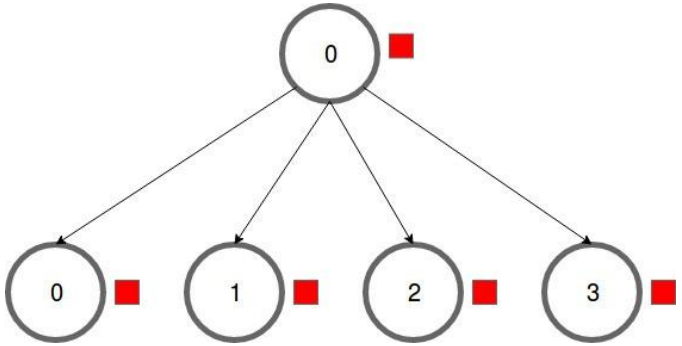
Рачунарске вежбе, Зимски семестар 2022/2023.

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. Аргументи функције су:
  - показивач на податке који се шаљу,
  - број података који се шаље,
  - тип података који се шаље (аргумент је типа *MPI_Datatype*),
  - `root` процес (ранк процеса који емитује податке осталима) и
  - комуникатор унутар кога се врши емитовање података
- Исписати поруку након што `root` процес пошаље податке, као и након што сваки од процеса прими податке. Обезбедити да и `root` процес сам себи пошаље поруку.
- **Формат могућег исписа:**

```
Proces 2 prima podatak 100 od root procesa
Proces 3 prima podatak 100 od root procesa
Proces 0 poslao poruku svima
Proces 0 prima podatak 100 od root procesa
Proces 1 prima podatak 100 od root procesa
```

- **Решење:** датотека `07_bcast.c`, директоријум `resenja`.

Задатак 7: Broadcast — појашњење решења

- `root` процес мора да прими поруку одмах након што је пошаље самом себи:
  - да би се обезбедило да не дође до закочења извршења, у случају блокирајућег, синхроног слања
  - да не би било никакве могућности да се програм оконча пре него што `root` прими своју поруку, у случају неблокирајућих или баферованих слања

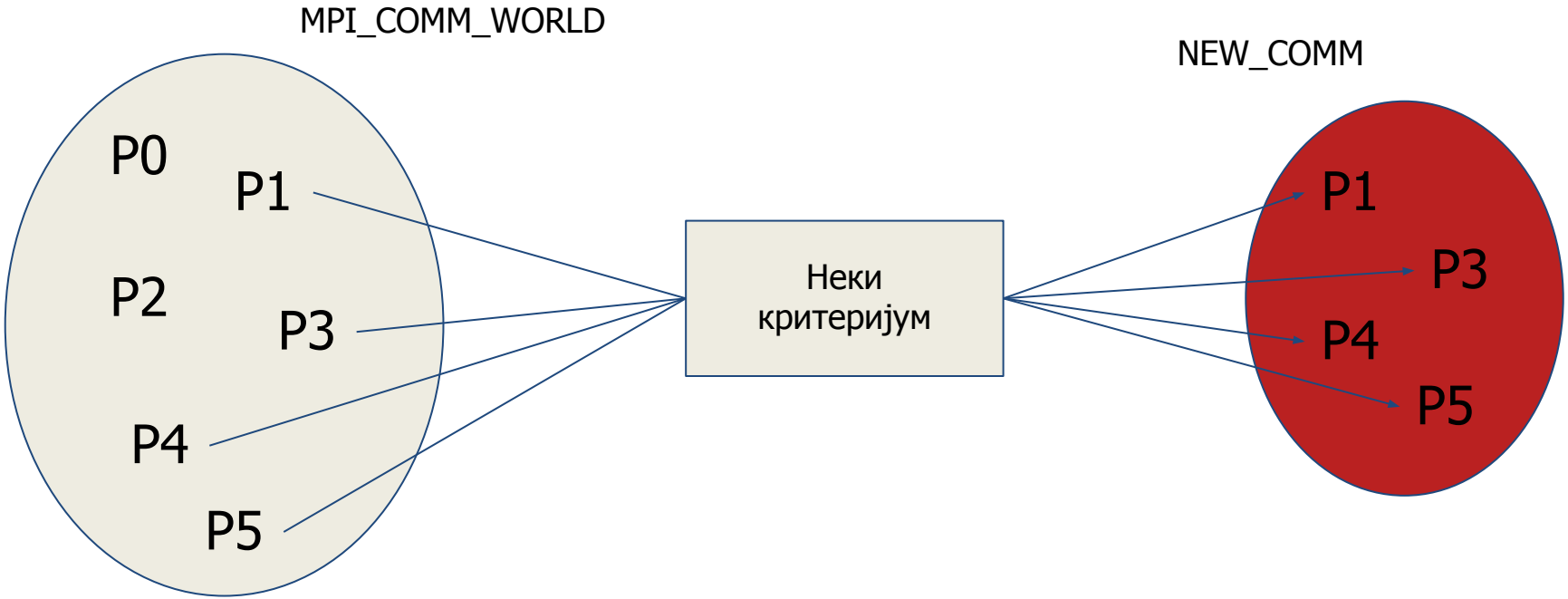
Колективна комуникација над подскупом комуникатора

- У колективној комуникацији учествују сви процеси унутар комуникатора.
- При решавању 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 коришћењем колективне комуникације?**

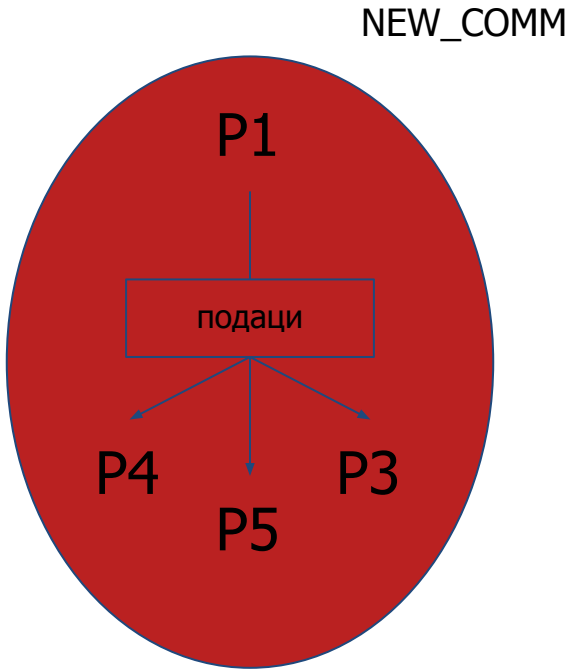
Колективна комуникација над подскупом комуникатора

- У колективној комуникацији учествују сви процеси унутар комуникатора.
- 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 коришћењем колективне комуникације?**
- **Одговор:** Направити нови комуникатор за процесе којима треба послати податке и користити колективну комуникацију на нивоу новонаправљеног комуникатора.

Колективна комуникација над подскупом комуникатора



Колективна комуникација над подскупом комуникатора

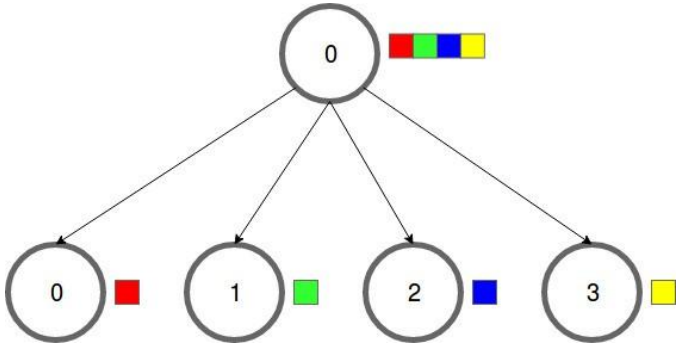


Колективна комуникација над подскупом комуникатора

- За креирање произвољног комуникатора, потребно је прво издвојити одговарајуће процесе у **групу**, па на основу ње креирати нови комуникатор.
- Погледати пример `06_comm_subset.c`.

Колективна комуникација: Scatter

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



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

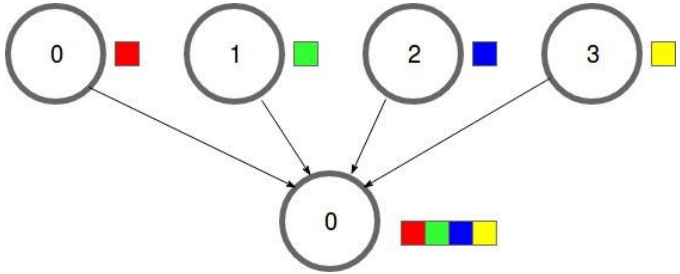
[1] MPI_Scatter docs

Пример 7: 07_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" податка исте величине.

[1] MPI_Gather docs

Пример 8: 08_gather.c

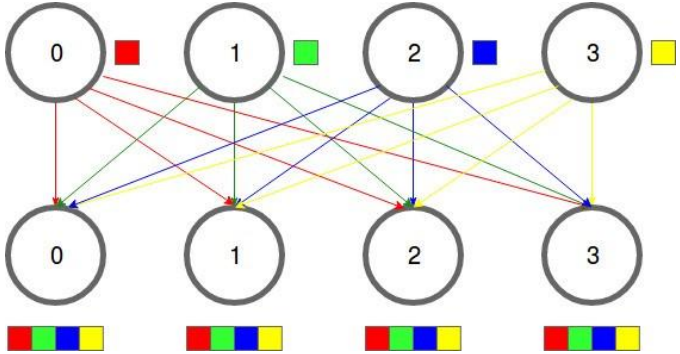
```
int main(int argc, char *argv[]) {

    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 8: 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.
  - Разделити изгенерисани низ на једнаке делове између свих процеса.
  - Сваки процес треба да израчуна своју парцијалну средњу вредност.
  - Након што су све парцијалне вредности срачунате, пребацују се назад коренском процесу који од парцијалних рачуна укупну средњу вредност.
  - Као аргумент командне линије проследити број жељених елемената по процесу.
- **Решење:** датотека `08_avg.c`, директоријум `resenja`.

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.

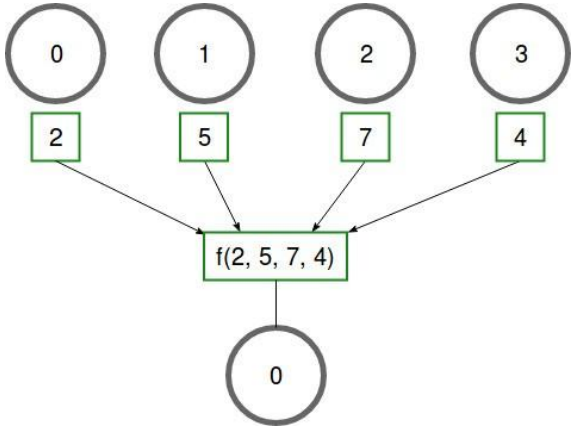
[1] MPI_Allgather docs

Пример 9: 09_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 10: 10_reduction.c

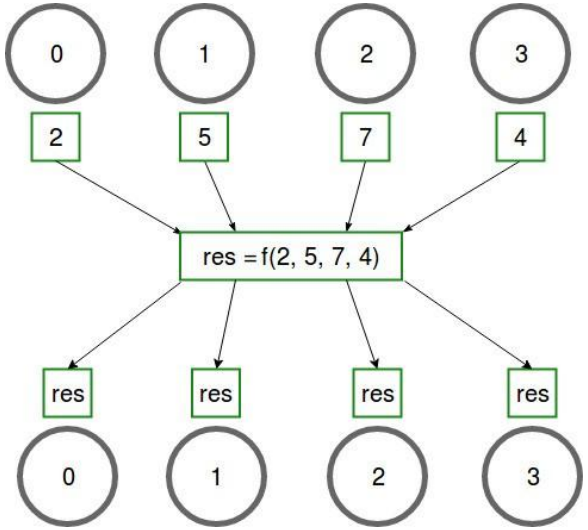
```c
int main(int argc, char *argv[]) {
     /*...*/

    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 9: 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 11: 11_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 10: 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. За рад са подацима у hdf5 формату, погледати костур наредног задатка, као и за детаље о компајлирању и покретању.
- 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 11: Множење матрица - **домаћи**

- Написати OpenMPI C програм за множење две квадратне матрице. Обе улазне матрице садрже бројеве у једнострукој прецизности. 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