

*Московский государственный университет имени М.В.Ломоносова
Факультет Вычислительной математики и кибернетики*

*СУПЕРКОМПЬЮТЕРЫ И
ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ
(лекция 12)*

А.С.Антонов

Вед. н.с. НИВЦ МГУ, к.ф.-м.н.

asa@parallel.ru

ВМК МГУ, 2024

МРІ

Группы и коммунікаторы

MPI

Необходимость групп и коммуникаторов:

- дать возможность некоторой группе процессов работать над своей независимой подзадачей;
- если только часть процессов должна обмениваться данными, бывает удобно завести для их взаимодействия отдельный коммуникатор;
- при создании библиотек подпрограмм нужно гарантировать, что пересылки данных в библиотечных модулях не пересекутся с пересылками в основной программе.

MPI

Группа – упорядоченное множество процессов. Каждому процессу в группе сопоставлено целое число – номер (ранг).

MPI_GROUP_EMPTY – пустая группа.

MPI_GROUP_NULL – значение, используемое для ошибочной группы.

Новую группу можно создать только из уже существующих групп или коммуникаторов.

MPI

```
int MPI_Comm_group(MPI_Comm  
comm, MPI_Group *group)
```

Получение группы **group**, соответствующей коммуникатору **comm**.

Поскольку изначально существует единственный нетривиальный коммуникатор **MPI_COMM_WORLD**, обычно в начале программы нужно получить соответствующую ему группу процессов.

MPI

```
int MPI_Group_intersection  
(MPI_Group group1, MPI_Group  
group2, MPI_Group *newgroup)
```

Создание группы **newgroup** из пересечения групп **group1** и **group2**. Полученная группа содержит все процессы группы **group1**, входящие также в группу **group2** и упорядоченные так же, как в первой группе.

MPI

```
int MPI_Group_union(MPI_Group  
group1, MPI_Group group2,  
MPI_Group *newgroup)
```

Создание группы **newgroup** из объединения групп **group1** и **group2**. Полученная группа содержит все процессы группы **group1** в прежнем порядке, за которыми следуют процессы группы **group2**, не вошедшие в группу **group1**, также в прежнем порядке.

MPI

```
int MPI_Group_difference  
(MPI_Group group1, MPI_Group  
group2, MPI_Group *newgroup)
```

Создание группы **newgroup** из разности групп **group1** и **group2**. Полученная группа содержит все элементы группы **group1**, не входящие в группу **group2** и упорядоченные, как в первой группе.

MPI

Пусть в группу **gr1** входят процессы **0, 1, 2, 4, 5**, а в группу **gr2** – процессы **0, 2, 3**.

```
MPI_Group_intersection(gr1, gr2, &newgr1);
```

```
MPI_Group_union(gr1, gr2, &newgr2);
```

```
MPI_Group_difference(gr1, gr2, &newgr3);
```

После ЭТИХ ВЫЗОВОВ:

newgr1: 0, 2;

newgr2: 0, 1, 2, 4, 5, 3;

newgr3: 1, 4, 5.

MPI

```
int MPI_Group_incl(MPI_Group  
group, int n, int *ranks,  
MPI_Group *newgroup)
```

Создание группы **newgroup** из **n** процессов группы **group** с рангами **ranks[0], ..., ranks[n-1]**, причём рангу **ranks[i]** в старой группе соответствует ранг **i** в новой группе. При **n=0** создается пустая группа **MPI_GROUP_EMPTY**.

MPI

```
int MPI_Group_excl(MPI_Group  
group, int n, int *ranks,  
MPI_Group *newgroup)
```

Создание группы **newgroup** из процессов группы **group**, исключая процессы с рангами **ranks[0],...,ranks[n-1]**, причём порядок процессов в новой группе соответствует порядку процессов в старой группе. При **n=0** создаётся группа, идентичная старой группе.

MPI

```
MPI_Comm_group(MPI_COMM_WORLD, &group);  
size1 = size/2;  
for(i=0; i<size1; i++) ranks[i] = i;  
MPI_Group_incl(group, size1, ranks, &group1);  
MPI_Group_excl(group, size1, ranks, &group2);
```

MPI

```
int MPI_Group_size(MPI_Group  
group, int *size)
```

```
int MPI_Group_rank(MPI_Group  
group, int *rank)
```

Определение количества процессов и номера процесса в группе. Если процесс не входит в группу, то возвращается значение **MPI_UNDEFINED**.

MPI

```
int MPI_Group_translate_ranks  
(MPI_Group group1, int n, int  
*ranks1, MPI_Group group2, int  
*ranks2)
```

В массиве **ranks2** возвращаются ранги в группе **group2** процессов с рангами **ranks1** в группе **group1**. Если процесс из группы **group1** не входит в группу **group2**, то для него возвращается значение **MPI_UNDEFINED**.

MPI

```
int MPI_Group_compare(MPI_Group  
group1, MPI_Group group2, int  
*result)
```

Сравнение групп **group1** и **group2**. Если совпадают, то возвращается значение **MPI_IDENT**. Если отличаются только рангами процессов, то возвращается значение **MPI_SIMILAR**. Иначе возвращается значение **MPI_UNEQL**.

MPI

```
int MPI_Group_free(MPI_Group  
*group)
```

Уничтожение группы **group**. Переменная **group** принимает значение **MPI_GROUP_NULL**.

MPI

Коммуникатор – контекст обмена группы. В операциях обмена используются только коммуникаторы!

MPI_COMM_WORLD – коммуникатор для всех процессов приложения.

MPI_COMM_NULL – значение, используемое для ошибочного коммуникатора.

MPI_COMM_SELF – коммуникатор, включающий только вызвавший процесс.

MPI

Коммуникатор даёт возможность независимых обменов. Каждой группе процессов может соответствовать несколько коммуникаторов, но каждый коммуникатор соответствует только одной группе.

Создание коммуникатора является коллективной операцией и требует операции межпроцессного обмена, поэтому такие процедуры могут оказаться весьма затратными по времени.

MPI

```
int MPI_Comm_dup(MPI_Comm comm,  
MPI_Comm *newcomm)
```

Создание нового коммуникатора **newcomm** с той же группой процессов и атрибутами, что и у коммуникатора **comm**.

MPI

```
int MPI_Comm_create(MPI_Comm  
comm, MPI_Group group, MPI_Comm  
*newcomm)
```

Создание нового коммуникатора **newcomm** из коммуникатора **comm** с группой процессов **group**. Вызов должен стоять во всех процессах коммуникатора **comm**. На процессах, не принадлежащих **group**, вернётся значение **MPI_COMM_NULL**.

MPI

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    int rank, size, i, rbuf;
    MPI_Group group, new_group;
    MPI_Comm new_comm;
    int ranks[128], new_rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_group(MPI_COMM_WORLD, &group);
```

MPI

```
for(i=0; i<size/2; i++) ranks[i] = i;
if(rank < size/2) MPI_Group_incl(group,
size/2, ranks, &new_group);
else MPI_Group_excl(group, size/2, ranks,
&new_group);
MPI_Comm_create(MPI_COMM_WORLD, new_group,
&new_comm);
MPI_Allreduce(&rank, &rbuf, 1, MPI_INT,
MPI_SUM, new_comm);
MPI_Group_rank(new_group, &new_rank);
printf("rank=%d newrank=%d rbuf=%d\n", rank,
new_rank, rbuf);
MPI_Finalize();
}
```

MPI

```
int MPI_Comm_split(MPI_Comm  
comm, int color, int key,  
MPI_Comm *newcomm)
```

Разбиение коммуникатора **comm** на несколько по числу значений параметра **color**. В одну подгруппу попадают процессы с одним значением **color**. Процессы с бóльшим значением **key** получают больший ранг.

MPI

Процессы, которые не должны войти в новые группы, указывают в качестве **color** константу **MPI_UNDEFINED**. Им в параметре **newcomm** вернется значение **MPI_COMM_NULL**.

```
MPI_Comm_split(MPI_COMM_WORLD, rank%3, rank,  
new_comm)
```


MPI

```
int MPI_Comm_compare(MPI_Comm  
comm1, MPI_Comm comm2, int  
*result)
```

Сравнение **comm1** и **comm2**. Если совпадают, то в **result** возвращается **MPI_IDENT**.

Если соответствуют одинаковым группам, а отличаются контекстом, то

MPI_CONGRUENT. Если соответствуют группам с одними процессами, но разной нумерацией, то возвращается

MPI_SIMILAR. Иначе - **MPI_UNEQUAL**.

MPI

```
int MPI_Comm_free(MPI_Comm comm)
```

Удаление коммуникатора **comm**. Переменной **comm** присваивается значение **MPI_COMM_NULL**.

MPI

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    int rank, size, rank1;
    MPI_Comm comm_revs;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_split(MPI_COMM_WORLD, 1, size-rank,
&comm_revs);
    MPI_Comm_rank(comm_revs, &rank1);
    printf("rank = %d rank1 = %d\n", rank,
rank1);
    MPI_Comm_free(&comm_revs);
    MPI_Finalize();
}
```

MPI

Виртуальные топологии

MPI

Топология – механизм сопоставления процессам альтернативной схемы адресации. В MPI топологии виртуальны, не связаны с физической топологией сети.

Два типа топологий:

- *декартова* (прямоугольная решётка произвольной размерности)
- *топология графа*.

MPI

Декартовы топологии

MPI

```
int MPI_Cart_create(MPI_Comm  
comm, int ndims, int *dims, int  
*periods, int reorder, MPI_Comm  
*comm_cart)
```

Создание коммуникатора **comm_cart** с декартовой топологией. **ndims** – размерность декартовой решётки, **dims** – число элементов в каждом измерении.

MPI

periods – массив из **ndims** элементов, определяющий, является ли решётка периодической вдоль каждого измерения.

reorder – при значении **1** системе разрешено менять порядок нумерации процессов.

Процедура должна быть вызвана всеми процессами коммуникатора.

Некоторым процессам может вернуться значение **MPI_COMM_NULL**.

MPI

```
dims[0] = 4;  
dims[1] = 3;  
dims[2] = 2;  
periods[0] = periods[1] = periods[2] = 1;  
MPI_Cart_create(MPI_COMM_WORLD, 3, dims,  
periods, 1, comm_cart);
```

MPI

```
int MPI_Dims_create(int nnodes,  
int ndims, int *dims)
```

Определение размеров **dims** для каждой из **ndims** размерностей при создании декартовой топологии для **nnodes** процессов. **dims[i]** рассчитывается, если перед вызовом функции оно равно 0, иначе остается без изменений.

MPI

Размеры по разным размерностям устанавливаются так, чтобы быть возможно близкими друг к другу. Перед вызовом функции значение **nnodes** должно быть кратно произведению ненулевых значений массива **dims**. Выходные значения массива **dims**, переопределённые процедурой, будут упорядочены в порядке убывания.

MPI

dims перед вызовом	вызов процедуры	dims после вызова
(0, 0, 0)	MPI_Dims_create(6, 3, dims)	(3, 2, 1)
(0, 0, 0)	MPI_Dims_create(7, 3, dims)	(7, 1, 1)
(0, 3, 0)	MPI_Dims_create(6, 3, dims)	(2, 3, 1)
(0, 3, 0)	MPI_Dims_create(7, 3, dims)	ошибка

MPI

```
int MPI_Cart_coords(MPI_Comm  
comm, int rank, int maxdims, int  
*coords)
```

Определение декартовых координат процесса по его рангу. Координаты возвращаются в массиве **coords**. Отсчёт координат по каждому измерению начинается с 0.

MPI

```
int MPI_Cart_rank(MPI_Comm comm,  
int *coords, int *rank)
```

Определение ранга процесса по его декартовым координатам. Для периодических решёток координаты вне допустимых интервалов пересчитываются, для неперiodических – ошибочны.

MPI

```
int MPI_Cart_sub(MPI_Comm comm,  
int *dims, MPI_Comm *newcomm)
```

Расщепление коммуникатора **comm** на подгруппы, соответствующие декартовым подрешёткам меньшей размерности. **i**-ый элемент массива **dims** равен **1**, если **i**-ое измерение должно остаться в подрешетке.

MPI

```
dims[0] = 1;  
dims[1] = 0;  
dims[2] = 1;  
MPI_Cart_sub(comm, dims, &newcomm);
```


MPI

```
int MPI_Cartdim_get(MPI_Comm  
comm, int *ndims)
```

Определение размерности декартовой топологии коммуникатора **comm**.

MPI

```
int MPI_Cart_get(MPI_Comm comm,  
int maxdims, int *dims, int  
*periods, int *coords)
```

Получение информации о декартовой топологии коммуникатора **comm** и координатах в ней вызвавшего процесса.

MPI

```
int MPI_Cart_shift(MPI_Comm  
comm, int direction, int disp,  
int *source, int *dest)
```

Получение номеров посылающего (**source**) и принимающего (**dest**) процессов в декартовой топологии коммуникатора **comm** для осуществления сдвига вдоль измерения **direction** на величину **disp**.

MPI

Для периодических измерений осуществляется циклический сдвиг, для непериодических – линейный сдвиг.

Для **n**-мерной декартовой решётки значение **direction** должно быть в пределах от **0** до **n-1**.

Значения **source** и **dest** можно далее использовать, например, для обмена функцией **MPI_Sendrecv**.

MPI

```
periods[0]=1;
periods[1]=1;
MPI_Cart_create(MPI_COMM_WORLD, 2, dims,
periods, 1, &comm);
MPI_Comm_rank(comm, &rank);
MPI_Cart_coords(comm, rank, 2, coords);
shift = 2;
MPI_Cart_shift(comm, 0, shift, &source,
&dest);
MPI_Sendrecv_replace(a, 1, MPI_FLOAT, dest, 0,
source, 0, comm, &status);
```

MPI

Топологии графа

MPI

```
int MPI_Graph_create(MPI_Comm  
comm, int nnodes, int *index,  
int *edges, int reorder,  
MPI_Comm *comm_graph)
```

Создание топологии графа **comm_graph**.
nnodes – число вершин графа,
index[i-1] содержит суммарное
количество соседей для первых **i** вершин.

MPI

edges содержит упорядоченный список номеров процессов-соседей.

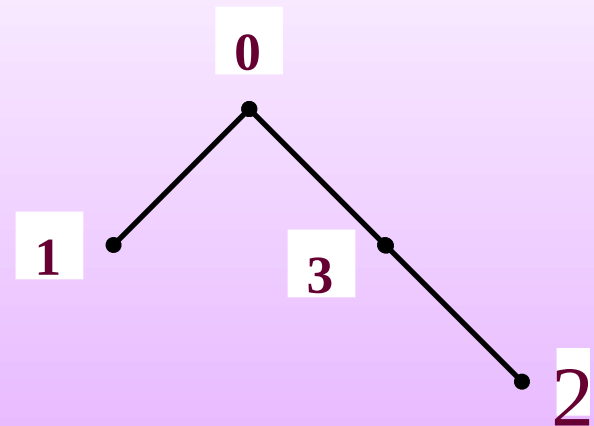
reorder – при значении **1** системе разрешено менять порядок нумерации процессов.

Процедура должна быть вызвана всеми процессами коммуникатора.

Некоторым процессам может вернуться значение **MPI_COMM_NULL**.

MPI

Процесс	Соседи
0	1, 3
1	0
2	3
3	0, 2



nnodes=4

index=2, 3, 4, 6

edges=1, 3, 0, 3, 0, 2

```
MPI_Graph_create(MPI_COMM_WORLD, nnodes, index,  
edges, 1, &comm_graph);
```

MPI

```
int MPI_Graph_neighbors_count  
(MPI_Comm comm, int rank, int  
*nneighbors)
```

Определение количества непосредственных соседей данной вершины графа.

MPI

```
int MPI_Graph_neighbors(MPI_Comm  
comm, int rank, int max, int  
*neighbors)
```

Определение непосредственных соседей
данной вершины графа.

MPI

```
int MPI_Graphdims_get(MPI_Comm  
comm, int *nnodes, int *nedges)
```

Определение числа вершин и числа дуг
графа.

MPI

```
int MPI_Graph_get(MPI_Comm comm,  
int maxindex, int maxedges, int  
*index, int *edges)
```

Определение информации о топологии графа в том виде, как она задается при создании топологии.

MPI

```
int MPI_Graph_map(MPI_Comm comm,  
int nnodes, int *index, int  
*edges, int *newrank)
```

Процедура вычисляет «оптимальное» относительно данной графовой топологии расположение процессов на процессорах (если поддерживается реализацией). В параметре **newrank** возвращается новый ранг процесса. Если вызывающий процесс не включен в графовую топологию, то возвращается значение **MPI_UNDEFINED**.

MPI

```
#include <stdio.h>
#include "mpi.h"
#define MAXPROC 128
#define MAXEDGES 512
int main(int argc, char **argv)
{
    int rank, rank1, i, size;
    int a, b;
    MPI_Status status;
    MPI_Comm comm_graph;
    int index[MAXPROC], edges[MAXEDGES];
    int num, neighbors[MAXPROC];
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    for(i = 0; i<size; i++) index[i] = size+i-1;
```

MPI

```
for(i = 0; i<size-1; i++){
    edges[i] = i+1;
    edges[size+i-1] = 0;
}
MPI_Graph_create(MPI_COMM_WORLD, size, index, edges, 1,
    &comm_graph);
MPI_Graph_neighbors_count(comm_graph, rank, &num);
MPI_Graph_neighbors(comm_graph, rank, num, neighbors);
for(i = 0; i<num; i++){
    MPI_Sendrecv(&rank, 1, MPI_INT, neighbors[i], 1,
    &rank1, 1, MPI_INT, neighbors[i], 1, comm_graph,
    &status);
    printf("process %d communicate with process %d\n",
    rank, rank1);
}
MPI_Finalize();
}
```


MPI

```
int  
MPI_Dist_graph_create_adjacent(M  
PI_Comm comm, int indegree, int  
sources[], int sourceweights[],  
int outdegree, int  
destinations[], int  
destweights[], MPI_Info info,  
int reorder, MPI_Comm  
*comm_dist_graph)
```

Создание коммуникатора
comm_dist_graph с топологией
распределённого графа.

MPI

Вызывающий процесс не указывает структуру всего графа, а только своих непосредственных соседей (те процессы, с которыми он будет обмениваться данными). Параметр **indegree** задаёт число процессов, от которых вызывающий процесс будет получать данные, в массиве **sources** задаются номера таких процессов. Параметр **outdegree** задаёт число процессов, которым вызывающий процесс будет посылать данные, в массиве **destinations** задаются номера таких процессов.

MPI

В массивах **sourceweights** и **destweights** задаются «веса» (неотрицательные целые числа) соответствующих дуг графа, которые могут использоваться для лучшей маршрутизации сообщений (использование зависит от реализации). Если один и тот же процесс входит в оба списка, то соответствующие веса должны совпадать. Вместо массива весов можно указать предопределённое значение **MPI_UNWEIGHTED**, означающее, что все дуги будут иметь одинаковые веса.

MPI

Параметр **info** может использоваться для указания способа интерпретации заданных весов. Значения зависят от реализации, например: минимизировать количество дуг или минимизировать сумму весов. Если не требуется, можно использовать значение **MPI_INFO_NULL**. Параметр **reorder** при значении **1** означает, что разрешено менять порядок нумерации процессов для оптимизации их распределения по процессорам; система может использовать значения заданных весов дуг.

MPI

```
int  
MPI_Dist_graph_create(MPI_Comm  
comm, int n, int sources[], int  
degrees[], int destinations[],  
int weights[], MPI_Info info,  
int reorder, MPI_Comm  
*comm_dist_graph)
```

Создание коммуникатора **comm_dist_graph** с топологией распределённого графа. Вызывающий процесс задаёт произвольную часть графа.

MPI

n задаёт число описываемых данным процессом начальных вершин дуг, сам список начальных вершин – в массиве **sources**. Для каждой начальной вершины в массиве **degrees** указывается количество задаваемых дуг. В массиве **destinations** в соответствующем порядке задаются конечные вершины дуг. Сначала все конечные вершины для первой начальной вершины, потом – для второй и т.д. В массиве **weights** - веса для всех дуг.

MPI

int

MPI_Dist_graph_neighbors_count(MPI_Comm comm, int *indegree, int *outdegree, int *weighted)

Определяется количество **indegree** входящих и **outdegree** исходящих дуг в топологии распределённого графа. В параметре **weighted** возвращается значение **0**, если при создании топологии было задано значение **MPI_UNWEIGHTED**, иначе — значение **1**.

MPI

```
int  
MPI_Dist_graph_neighbors(MPI_Comm  
comm, int maxindegree, int  
sources[], int sourceweights[],  
int maxoutdegree, int  
destinations[], int  
destweights[])
```

В параметре **sources** определяются начальные вершины дуг, входящих в данную, в **destinations** – конечные вершины дуг, исходящих из данной.

MPI

В массивах **sourceweights** и **destweights** определяются соответствующие веса дуг, если они были заданы при создании топологии. Параметры **maxindegree** и **maxoutdegree** задают размеры определяемых массивов.

MPI

```
int MPI_Topo_test(MPI_Comm comm,  
int *type)
```

Определение типа топологии, связанной с коммуникатором **comm**.

- **MPI_GRAPH** для графа;
- **MPI_DISTGRAPH** для распределённого графа;
- **MPI_CART** для декартовой топологии;
- **MPI_UNDEFINED** — нет связанной топологии.

MPI

Литература

1. MPI: A Message-Passing Interface Standard. Version 4.1. Message Passing Interface Forum. November 3, 2023. <https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf>
2. MPI – the complete reference, second edition / Ed. by M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra. – The MIT Press, 1998.
3. Антонов А.С. Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. Предисл.: В.А.Садовничий. - М.: Издательство Московского университета, 2012.-344 с.- (Серия "Суперкомпьютерное образование").
4. Антонов А.С. Введение в параллельные вычисления (методическое пособие). – М.: Изд-во Физического факультета МГУ, 2002.
5. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: Изд-во МГУ, 2004.

MPI

Литература

6. Букатов А.А., Дацюк В.Н., Жегуло А.И. Программирование многопроцессорных вычислительных систем. – Ростов-на-Дону: Издательство ООО «ЦВВР», 2003.
7. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб: БХВ-Петербург, 2002.
8. Корнеев В.Д. Параллельное программирование в MPI. – Новосибирск: Изд-во СО РАН, 2000.
9. Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем. – СПб: БХВ-Петербург, 2002.
10. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI: Пособие. – Минск: БГУ, 2002.

*Московский государственный университет имени М.В.Ломоносова
Факультет Вычислительной математики и кибернетики*

*СУПЕРКОМПЬЮТЕРЫ И
ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ
(лекция 12)*

А.С.Антонов

Вед. н.с. НИВЦ МГУ, к.ф.-м.н.

asa@parallel.ru

ВМК МГУ, 2024