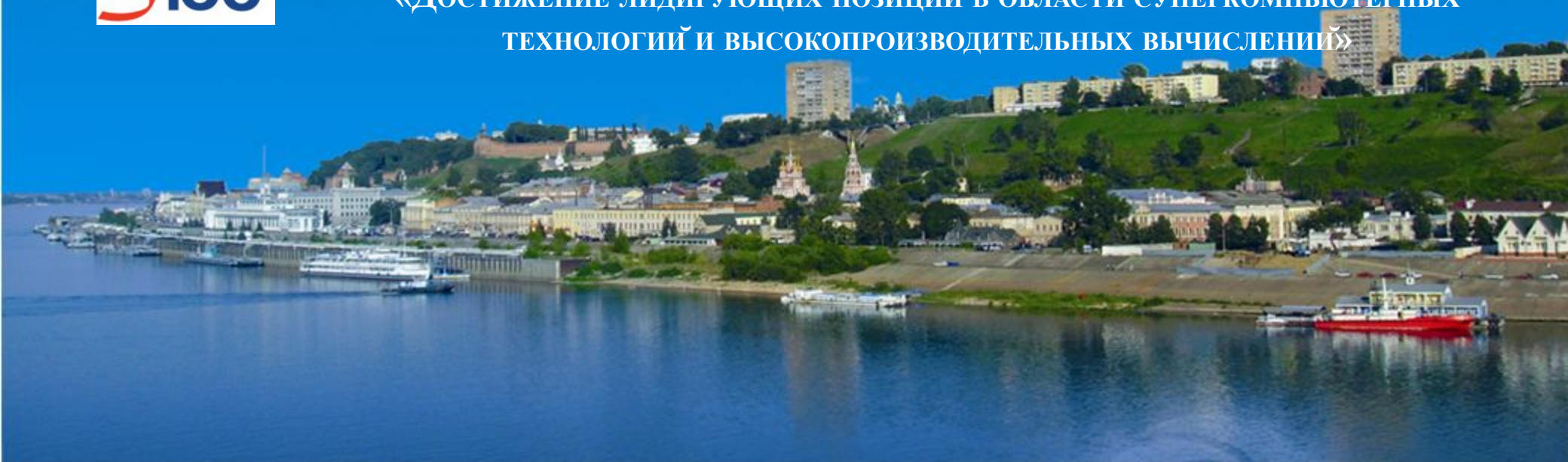



Нижегородский государственный университет имени Н.И. Лобачевского

Институт Информационных технологий, математики и механики

Проект повышения конкурентоспособности ведущих российских университетов среди ведущих мировых научно-образовательных центров

Стратегическая инициатива «Достижение лидирующих позиций в области суперкомпьютерных технологий и высокопроизводительных вычислений»



Нижегородский государственный университет
имени Н.И. Лобачевского
Институт Информационных технологий, математики
и механики

Параллельное программирование для многопроцессорных систем с распределенной памятью

03 Лекция
Производные типы данных, коммуникаторы и виртуальные топологии

Гергель В.П., Сысоев А.В.
Кафедра МОСТ

Содержание

- Производные типы данных MPI
  - Карта типа
  - Методы конструирования
  - Регистрация и удаление
  - Упаковка данных
- Группы процессов и коммуникаторы
  - Управление группами
  - Управление коммуникаторами
- Виртуальные топологии
  - Декартовы топологии (решетки)
  - Топологии графов

ПРОИЗВОДНЫЕ ТИПЫ ДАННЫХ MPI

Карта типа

Методы конструирования

Регистрация и удаление

Упаковка данных

Производные типы данных MPI…
Карта типа

- Во всех примерах далее полагается, что сообщения – непрерывные вектора, состоящие из элементов, тип которых предварительно определен в MPI
- Необходимые для передачи данные могут быть расположены далеко друг от друга и содержать значения различных типов
  - Данные могут быть переданы с помощью нескольких сообщений (этот метод не будет эффективен из-за накопления задержек)
  - Данные могут быть упакованы в формат непрерывного вектора (что требует дополнительных операций копирования данных)
  - Для описания расположения данных в памяти могут быть созданы производные типы данных MPI

Производные типы данных MPI…
Карта типа

- Производные типы данных MPI – это описание множества значений предопределенных в MPI типов, не обязательно непрерывно расположенных в памяти
- Тип устанавливается в MPI с помощью **карты типа** в виде последовательных описаний значений, включенных в тип, каждое отдельное значение описывается указанием на тип и смещением от начального адреса

TypeMap = $\{(type_0, disp_0), (type_1, disp_1), \dots, (type_{n-1}, disp_{n-1})\}$

- Часть карты типа, которая содержит только типы значений называется **сигнатура типа**

TypeSignature = $\{type_0, type_1, \dots, type_{n-1}\}$

Производные типы данных MPI…
Карта типа

Пример

- Пусть сообщение содержит следующие переменные

```
double a; /* address 24 */
double b; /* address 40 */
int    n; /* address 48 */
```

- Предположим, мы знаем адреса переменных a, b, n в памяти
- Тогда производный тип для описания данных должен иметь карту следующего вида

```
{
 (MPI_DOUBLE, 0),
 (MPI_DOUBLE, 16),
 (MPI_INT, 24)
}
```

Производные типы данных MPI…
Карта типа

- Следующие концепции используются в MPI для производных типов данных
  - **Нижняя граница** типа
$$lb\ (TypeMap) = \min_j (disp_j)$$
  - **Верхняя граница** типа
$$ub\ (TypeMap) = \max_j (disp_j + sizeof\ (type_j)) + \Delta$$
  - **Протяженность** типа (размер памяти в байтах, который должен быть выделен для элемента производного типа)
$$extent\ (TypeMap) = ub\ (TypeMap) - lb\ (TypeMap)$$
  - **Размер** типа данных – число байт, требуемое для размещения одного значения типа данных
  - Разница между значениями протяженности и размера в дополнительной памяти, необходимой для выравнивания адреса

Производные типы данных MPI…
Карта типа

- MPI предоставляет следующие функции для получения значений протяженности и размера типа

```
int MPI_Type_extent(MPI_Datatype type, MPI_Aint *extent);
int MPI_Type_size(MPI_Datatype type, MPI_Aint *size);
```

- Нижние и верхние границы типов могут быть определены при помощи следующих функций

```
int MPI_Type_lb(MPI_Datatype type, MPI_Aint *disp);
int MPI_Type_ub(MPI_Datatype type, MPI_Aint *disp);
```

- При конструировании производных типов используется функция получения адреса переменной

```
int MPI_Address(void *location, MPI_Aint *address);
```

Производные типы данных MPI… Методы конструирования

- **Непрерывный** метод позволяет определять непрерывный набор элементов некоторого типа как новый производный тип
- **Векторный** метод предоставляет возможность создавать новый производный тип как набор элементов некоторого доступного типа. Между элементами могут быть регулярные интервалы памяти. Размер интервалов определяется числом элементов исходного типа, в случае метода **h-вектора** данный размер должен быть установлен в байтах
- **Индексный** метод отличается от векторного метода тем, что интервалы между элементами нерегулярны
- **Структурный** метод предоставляет наиболее общее описание производного типа, непосредственно используя карту типа создаваемого типа данных

Производные типы данных MPI… Векторный метод

- При векторном способе производный тип создается как набор блоков из элементов исходного типа, при этом между блоками могут иметься регулярные промежутки по памяти

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

- **count** – число блоков
- **blocklen** – число элементов в каждом блоке
- **stride** – число элементов между началами соседних блоков
- **oldtype** – исходный тип данных
- **newtype** – создаваемый тип данных

Производные типы данных MPI… Векторный метод

- Если интервалы между блоками задаются в байтах, то сконструировать производный тип данных можно с использованием следующей функции

```
int MPI_Type_hvector(int count, int blocklen,
  MPI_Aint stride, MPI_Datatype oldtype,
  MPI_Datatype *newtype);
```

Производные типы данных MPI… Векторный метод

- Производные типы данных для описания подмассивов многомерных массивов могут быть созданы функцией

```
int MPI_Type_create_subarray(int ndims, int sizes[],
  int subsizes[], int starts[], int order,
  MPI_Datatype oldtype, MPI_Datatype *newtype);
```

- **ndims** – размерность массива
- **sizes** – число элементов в каждой размерности исходного массива
- **subsizes** – число элементов в каждой размерности создаваемого подмассива
- **starts** – индексы начальных элементов в каждой размерности подмассива
- **order** – порядок хранения для подмассива, а также для исходного массива (**MPI_ORDER_C**, **MPI_ORDER_FORTRAN**)
- **oldtype** – тип данных элементов исходного массива
- **newtype** – создаваемый тип данных для описания подмассива

Производные типы данных MPI… Индексный метод

- При индексном способе производный тип создается как набор блоков разного размера из элементов исходного типа, при этом между блоками могут иметься разные промежутки по памяти

```
int MPI_Type_indexed(int count, int blocklens[],
  int offsets[], MPI_Datatype oldtype,
  MPI_Datatype *newtype);
```

- **count** – число блоков
- **blocklen** – число элементов в каждой блоке
- **offsets** – смещение каждого блока от начала типа (в количестве элементов исходного типа)
- **oldtype** – исходный тип данных
- **newtype** – создаваемый тип данных

Производные типы данных MPI… Индексный метод

- Если смещения блоков задаются в байтах, то сконструировать производный тип данных можно с использованием следующей функции

```
int MPI_Type_indexed(int count, int blocklens[],
  MPI_Aint offsets[], MPI_Datatype oldtype,
  MPI_Datatype *newtype);
```

Производные типы данных MPI…
Индексный метод

Пример

- Конструирование типа для описания верхнетреугольной матрицы размера *n x n*

```
int *blocklens, *offsets;
MPI_Datatype UTMatrixType;

// выделение памяти для блоков и смещений
for (i = 0, i < n; i++)
{
  blocklens[i] = n - i;
  offsets[i]   = i * n + i;
}
MPI_Type_indexed(n, blocklens, offsets, MPI_DOUBLE,
  &UTMatrixType);
```

Производные типы данных MPI… Структурный метод

- Самый общий метод конструирования производного типа данных при явном задании соответствующей карты типа

```
int MPI_Type_struct(int count, int blocklens[],
  int offsets[], MPI_Datatype oldtypes[],
  MPI_Datatype *newtype);
```

- **count** – число блоков
- **blocklen** – число элементов в каждом блоке
- **offsets** – смещение каждого блока от начала типа (в количестве элементов исходного типа)
- **oldtype** – исходный тип данных для каждого блока отдельно
- **newtype** – создаваемый тип данных

Производные типы данных MPI… Регистрация и удаление

- Созданный тип данных перед использованием должен быть зарегистрирован с помощью следующей функции

```
int MPI_Type_commit(MPI_Datatype *type);
```

- После прекращения использования производный тип должен быть аннулирован с помощью следующей функции

```
int MPI_Type_free(MPI_Datatype *type);
```

Производные типы данных MPI…
Упаковка данных

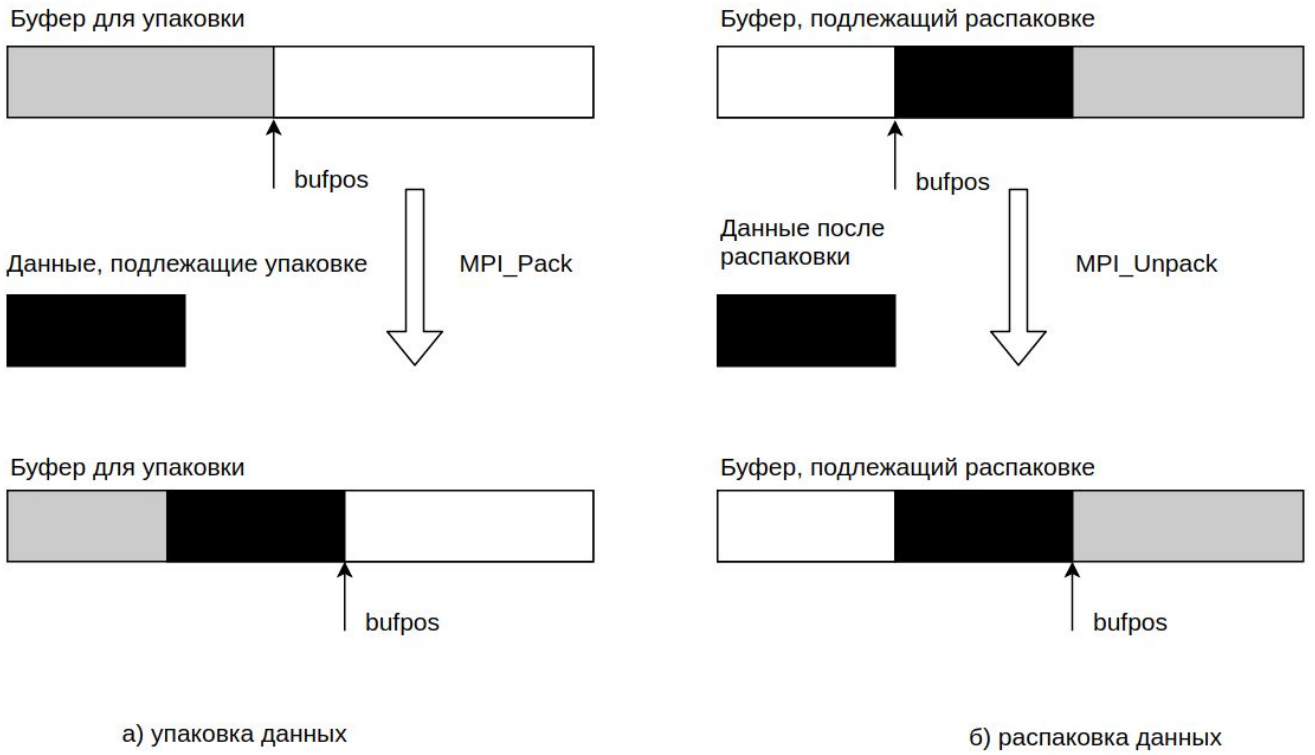
- Явный метод упаковки и распаковки сообщений, которые содержат значения разных типов и расположены в разных местах памяти

```
int MPI_Pack(void *data, int count, MPI_Datatype type,
  void *buf, int bufsize, int *bufpos, MPI_Comm comm);
```

- **data** – буфер памяти с элементами для упаковки
- **count** – число элементов в буфере
- **type** – тип данных упаковываемых элементов
- **buf** – буфер памяти для упаковки
- **buflen** – размер буфера в байтах
- **bufpos** – позиция начала буферизации (в байтах от начала буфера)
- **comm** – коммуникатор для упакованного сообщения

Производные типы данных MPI…
Упаковка данных

- Схема упаковки и распаковки данных



а) упаковка данных

б) распаковка данных

Производные типы данных MPI…
Упаковка данных

- Чтобы определить размер буфера, необходимый для упаковки, можно использовать следующую функцию

```
MPI_Pack_size(int count, MPI_Datatype type, MPI_Comm comm,
  int *size);
```

- Чтобы отправить упакованные данные, подготовленный буфер должен использоваться в функции **MPI_Send()** с типом **MPI_PACKED**,
- После приема сообщения с типом **MPI_PACKED**, данные могут быть распакованы с помощью следующей функции

```
int MPI_Unpack(void *buf, int bufsize, int *bufpos,
  void *data, int count, MPI_Datatype type, MPI_Comm comm);
```

Производные типы данных MPI…
Упаковка данных

- Функция **`MPI_Pack()`** вызывается последовательно для упаковки всех необходимых данных. Таким образом, если сообщение представляет собой набор переменных **`a`**, **`b`** и **`n`**,

```
double a; /* адрес 24 */
double b; /* адрес 40 */
int    n; /* адрес 48 */
```

чтобы упаковать данные, необходимо выполнить следующие операции

```
bufpos = 0;
MPI_Pack(a,1,MPI_DOUBLE,buf,buflen,&bufpos,comm);
MPI_Pack(b,1,MPI_DOUBLE,buf,buflen,&bufpos,comm);
MPI_Pack(n,1,MPI_INT,buf,buflen,&bufpos,comm);
```

Производные типы данных MPI…
Упаковка данных

- Для распаковки данных необходимо выполнить следующее

```
bufpos = 0;
MPI_Unpack(buf,buflen,&bufpos,a,1,MPI_DOUBLE,comm);
MPI_Unpack(buf,buflen,&bufpos,b,1,MPI_DOUBLE,comm);
MPI_Unpack(buf,buflen,&bufpos,n,1,MPI_INT,comm);
```

Производные типы данных MPI…
Упаковка данных

- Данный подход требует дополнительных операций по упаковке и распаковке данных
- Может быть оправдан, если размеры сообщений относительно малы и сообщения упаковываются / распаковываются достаточно редко
- Упаковка и распаковка могут оказаться полезными при использовании буферизованной передачи данных (MPI_Bsend)

ГРУППЫ ПРОЦЕССОВ И КОММУНИКАТОРЫ

Управление группами

Управление коммуникаторами

Группы процессов и коммуникаторы… Управление группами

- Процессы объединены в группы. Группы могут содержать все процессы параллельной программы или только часть доступных процессов. Один и тот же процесс может принадлежать нескольким группам
- Группы процессов формируются для создания на их основе коммуникаторов
- Группы процессов могут быть определены на основе только доступных групп. Группа, связанная с некоторым коммуникатором, может быть выделена с помощью следующей функции

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

Группы процессов и коммуникаторы… Управление группами

- Новые группы могут быть созданы на основе существующих групп
- Можно создать новую группу **newgroup** на основе группы **oldgroup**, которая включает в себя **n** процессов
  - Ранги процессов, **включаемых** в **newgroup**, перечисляются в массиве **ranks**

```
int MPI_Group_incl(MPI_Group oldgroup, int n, int ranks[],
  MPI_Group *newgroup);
```

  - Ранги процессов, **не включаемых** в **newgroup**, перечисляются в массиве **ranks**

```
int MPI_Group_excl(MPI_Group oldgroup, int n, int ranks[],
  MPI_Group *newgroup);
```

Группы процессов и коммуникаторы… Управление группами

- Новые группы также могут быть созданы с помощью операций:
  - Создание новой группы **newgroup** путем **объединения** групп **group1** и **group2**

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

  - Создание новой группы **newgroup** из **общих** процессов групп **group1** и **group2**

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

  - Создание новой группы **newgroup** из **разности group1** и **group2**

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

Группы процессов и коммуникаторы… Управление группами

- Следующие MPI функции предоставляют получение информации о группе процессов
  - Получение числа процессов в группе

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

  - Получение ранга текущего процесса в группе

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

- После прекращения использования группа должна быть удалена

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

Группы процессов и коммуникаторы… Управление коммуникаторами

- **Коммуникатор** в MPI – специальный объект управления, который объединяет в своем составе группу процессов и ряд дополнительных параметров (контекст), которые используются в операциях передачи данных
- В этом подразделе обсуждается управление **интракоммуникаторами**, которые используются для работы с данными в рамках группы процессов

Группы процессов и коммуникаторы… Управление коммуникаторами

- Для создания нового коммуникатора используются методы
  - Дублирование существующего коммуникатора

```
int MPI_Comm_dup(MPI_Comm oldcom, MPI_comm *newcomm);
```

  - Создание нового коммуникатора из подмножества процессов существующего коммуникатора

```
int MPI_comm_create(MPI_Comm oldcom, MPI_Group group,
  MPI_Comm *newcomm);
```

- Операция создания коммуникаторов коллективная и должна выполняться всеми процессами исходного коммуникатора
- После прекращения использования коммуникатор должен быть удален

```
int MPI_Comm_free(MPI_Comm *comm);
```

Группы процессов и коммуникаторы… Управление коммуникаторами

- Следующая функция предоставляет быстрый и удобный метод одновременного создания нескольких коммуникаторов

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

- **oldcomm** – исходный коммуникатор,
- **color** – процессы с одинаковым значением color попадут в один коммуникатор
- **key** – порядок ранжирования процессов в создаваемом коммуникаторе
- **newcomm** – создаваемый коммуникатор

- Функция **MPI_Comm_split()** должна быть вызвана в каждом процессе коммуникатора **oldcomm**

Группы процессов и коммуникаторы… Управление коммуникаторами

- Выполнение функции **`MPI_Comm_split()`** приводит к разделению процессов на непересекающиеся группы
- Каждая новая группа формируется из процессов, имеющих одинаковое значение параметра **`color`**
- На основе созданных групп создается набор коммуникаторов
- Порядок перечисления для рангов процессов выбирается таким образом, чтобы он соответствовал порядку значений key (процесс с большим значением key будет иметь более высокий ранг)

ВИРТУАЛЬНЫЕ ТОПОЛОГИИ

Декартовы топологии (решетки)

Топологии графов

Виртуальные топологии...

- Топология компьютерной системы – структура узлов и линий связи в сети. Топология может быть представлена в виде графа, где вершины – процессоры (процессы), а дуги – каналы связи
- Парные операции передачи данных могут выполняться для любых процессов одного и того же коммуникатора. В коллективных операциях участвуют все процессы коммуникатора. Т.о. **логическая топология** линий связи в параллельной программе представляет собой **полный граф**
- MPI предоставляет возможность организовать логическое представление любой необходимой виртуальной топологии

Виртуальные топологии...
Декартовы топологии (решетки)

- Декартовы топологии предполагают представление множества процессов в виде прямоугольной решетки и использование декартовой системы координат для адресации процессов
- Для создания декартовой топологии используется функция

```
int MPI_Cart_create(MPI_Comm oldcomm, int ndims, int *dims,
  int *periods, int reorder, MPI_Comm *cartcomm);
```

- **oldcomm** – исходный коммуникатор
- **ndims** – размерность декартовой решетки
- **dims** – массив длины **ndims**, определяющий количество процессов в каждой размерности решетки
- **periods** – массив длины **ndims**, определяющий, является ли решетка периодической по каждому измерению
- **reorder** – может ли порядок рангов процессов быть изменен
- **cartcomm** – коммуникатор создается с декартовой топологией

Виртуальные топологии...
Декартовы топологии (решетки)

- Чтобы определить декартовы координаты процесса в соответствии с его рангом, можно использовать функцию

```
int MPI_Card_coords(MPI_Comm comm, int rank, int ndims,
  int *coords);
```

- **comm** – коммуникатор с топологией решетки
- **rank** – ранг процесса, для которого определяются декартовы координаты
- **ndims** – размерность декартовой решетки
- **coords** – декартовы координаты процесса, вычисленные функцией

Виртуальные топологии...
Декартовы топологии (решетки)

- Определение ранга процесса по его декартовым координатам обеспечивается с помощью функции

```
int MPI_Cart_rank(MPI_Comm comm, int coords[], int *rank);
```

- **comm** – коммуникатор с топологией решетки
- **coords** – декартовы координаты процесса
- **rank** – ранг процесса, вычисляемый функцией

Виртуальные топологии...
Декартовы топологии (решетки)

- Процедура разделения решетки на подрешетки меньшей размерности обеспечивается функцией

```
int MPI_Card_sub(MPI_Comm comm, int subdims[],
  MPI_Comm *newcomm);
```

- **comm** – исходный коммуникатор с топологией сетки
- **subdims** – массив для выбора размерностей для подрешеток (0 или 1)
- **newcomm** – создаваемый коммуникатор

- Функция **MPI_Cart_sub()** создает коммуникаторы для каждой комбинации координат фиксированных размерностей исходной решетки

Виртуальные топологии...
Декартовы топологии (решетки)

- Дополнительная функция **`MPI_Cart_shift()`** предоставляет поддержку передач со сдвигом по размерности решетки
  - **Циклический сдвиг** на k позиций вдоль размерности решетки. Данные процесса i передаются процессу (i + k) mod n, где n – размер измерения, по которому выполняется сдвиг
  - **Линейный сдвиг** на k позиций вдоль размерности решетки. Данные от процессора i передаются процессору i + k (если последний доступен)
- Функция **`MPI_Cart_shift()`** определяет только ранги процессов, которые должны обменяться данными в ходе операции сдвига. Обмен данных может выполняться, например, с помощью функции **`MPI_Sendrecv()`**

Виртуальные топологии...
Декартовы топологии (решетки)

- Функция **`MPI_Cart_shift()`** обеспечивает получение рангов процессов, которые должны обмениваться данными с процессом, вызвавшем **`MPI_Cart_shift()`**

```
int MPI_Card_shift(MPI_Comm comm, int dir, int disp,
  int *source, int *dst);
```

- **`comm`** – коммуникатор с топологией решетки
- **`dir`** – номер размерности, по которому выполняется сдвиг
- **`disp`** – значение сдвига (<0 – сдвиг к началу измерения)
- **`source`** – ранг процесса, из которого должны быть получены данные
- **`dst`** – ранг процесса, которому должны быть отправлены данные

Виртуальные топологии...
Топологии графов

- Для создания коммуникатора с топологией графа в MPI предусмотрена следующая функция

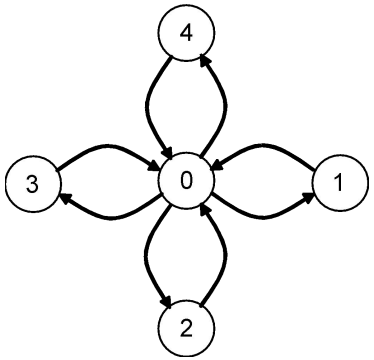
```
int MPI_Graph_create(MPI_Comm oldcomm, int nnodes,
  int index[], int edges[], int reorder,
  MPI_Comm *graphcomm);
```

- **oldcomm** – исходный коммуникатор
- **nnodes** – число вершин графа
- **index** – число дуг, идущих от каждой вершины
- **edges** – последовательный список дуг
- **reorder** – флаг для указания, можно ли переупорядочить ранги процессов
- **graphcomm** – создаваемый коммуникатор с топологией графа

Виртуальные топологии...
Топологии графов

Пример

- Число процессов равно 5, порядок вершин графа – (4,1,1,1,1), а матрица инцидентов выглядит следующим образом



Процессы	Линии связи
0	1,2,3,4
1	0
2	0
3	0
4	0

- Чтобы создать топологию в виде указанного графа, необходимо выполнить следующий код

```
int index[] = { 4,1,1,1,1 };
int edges[] = { 1,2,3,4,0,0,0,0 };
MPI_Comm StarComm;
MPI_Graph_create(MPI_COMM_WORLD,5,index,edges,1,&StarComm);
```

Виртуальные топологии...
Топологии графов

- Число соседних процессов, которые содержат исходящие дуги от текущего процесса, которое может быть получено с помощью функции

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

- Получение рангов соседних вершин обеспечивается следующей функцией

```
int MPI_Graph_neighbors(MPI_Comm comm, int rank,
  int nneighbors, int *neighbors);
```

где **nneighbors** размер массива **neighbors**

Резюме

- Разобрано использование производных типов данных MPI
- Рассмотрено управление группами и коммуникаторами
- Рассмотрены виртуальные топологии

Упражнения

- Разработайте программу для каждого метода построения производного типа данных, доступного в MPI
- Разработайте программу, используя функции упаковки и распаковки. Проведите эксперименты и сравните результаты с вариантами с использованием производных типов данных
- Разработайте производные типы данных для строк, столбцов и диагоналей матриц
- Разработайте программу для декартовой топологии
- Разработайте программу для топологии графа
- Разработайте подпрограммы для создания набора дополнительных виртуальных топологий (звезда, дерево и т.д.)

Ссылки

1. Интернет-ресурс, описывающий стандарт MPI: http://www.mpiforum.org
2. Одна из самых широко используемых реализаций стандарта, библиотека MPICH, размещена по адресу http://www.mpich.org
3. Quinn, M.J. (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.
4. Pacheco, P. (1996). Parallel Programming with MPI. - Morgan Kaufmann.
5. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J. (1996). MPI: The Complete Reference. – MIT Press, Boston, 1996.
6. Group, W., Lusk, E., Skjellum, A. (1999). Using MPI – 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation). – MIT Press.
7. Group, W., Lusk, E., Thakur, R. (1999). Using MPI-2: Advanced Features of the Message Passing Interface (Scientific and Engineering Computation). – MIT Press.