- Классификация технологий параллельного программирования
- MPI: обзор возможностей

- Цели и классификация технологий параллельного программирования
- Характеристика MPI
- «Прожиточный минимум» MPI-функций
- Процедуры общего назначения
- Обмены «точка – точка»
- Коллективное взаимодействие процессов
- Работа с группами и коммуникаторами
- Виртуальные топологии
- Работа с разнотипными данными

Классификация технологий параллельного программирования (ТТП)

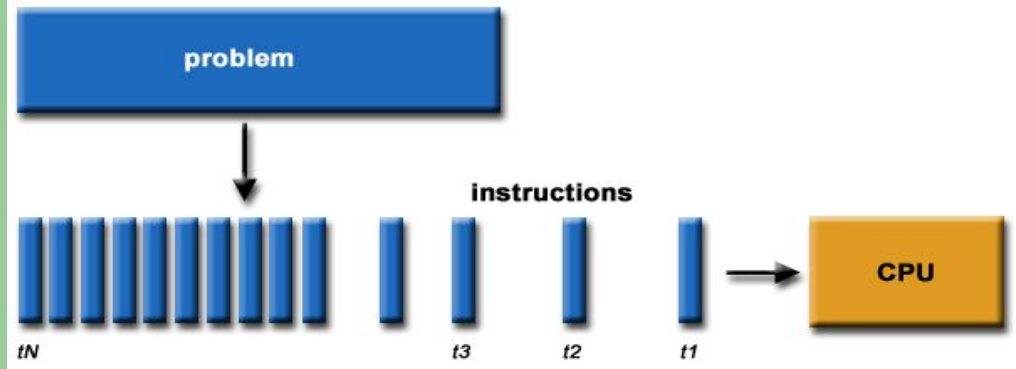
1. Технологии параллельного программирования (ТПП) обычно реализуются как расширения традиционных языков программирования, с добавлением элементов, поддерживающих параллельные вычисления

- набор функций для взаимодействия между параллельными процессами (MPI, PVM, LINDA и др.)
- Набор директив для организации параллельных областей (OpenMP, HPF, DVM и др.)

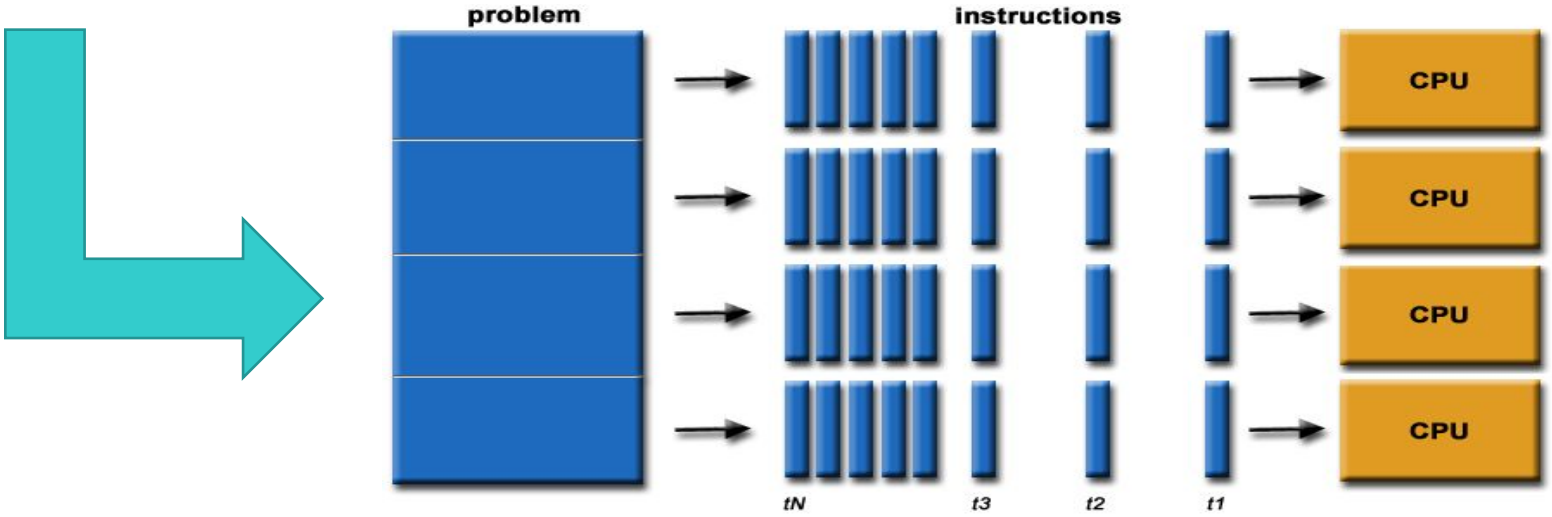
2. Существуют реализации в виде библиотек – параллельных аналогов традиционных проблемно-ориентированных библиотек (ScaLAPACK)

3. ТПП могут быть ориентированы на широкий класс вычислительных архитектур, либо разрабатываются специально под определенную архитектуру (CUDA)

Классификация технологий параллельного программирования (ТПП)



Во всех случаях цель ТПП – организовать выполнение задачи пользователя в параллельном режиме:



Что такое MPI

- Message Passing Interface – одна из популярных технологий для организации параллельных вычислений.
- MPI – библиотека подпрограмм (Фортран) и функций (C\C++), предназначенная для поддержки работы параллельных процессов в терминах передачи сообщений.
- Каждая инструкция выполняется каждым процессом.
- Нет глобальных переменных! Каждый процесс имеет доступ к собственному адресному пространству.
- Для выполнения задачи создается группа MPI-процессов. Поведение всех процессов описывается одной и той же программой. Межпроцессные коммуникации программируются явно с помощью библиотеки MPI, которая и диктует стандарт программирования.
- Для организации обмена между процессами, одновременно выполняющими один код, необходим вызов процедур MPI.

Структура MPI

Главные составляющие MPI

- Коммуникатор (группа процессов плюс контекст взаимодействия)
- Типы передаваемых данных (всегда передаем данные одного типа, расположенные подряд, указываем начальную позицию массива передаваемых данных)
- Функции передачи сообщений (около 200)
- Виртуальные топологии

Функции разделяются на следующие группы:

- Функции общего назначения
- Функции для обменов между отдельными процессами
- Функции для коллективного взаимодействия процессов
- Функции для поддержки виртуальных топологий
- Функции для организации производных типов данных

Процедуры MPI повторение и дополнение пройденного на семинарах

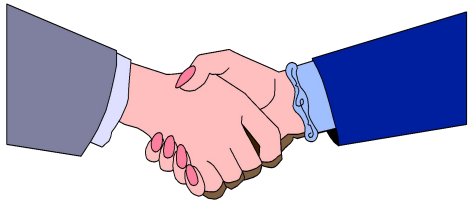
«ПРОЖИТОЧНЫЙ МИНИМУМ»

MPI_Init инициализация параллельной части программы
MPI_Finalize завершение параллельной части программы
MPI_Comm_size число процессов в группе
MPI_Comm_rank номер процесса
MPI_Send блокирующая передача данных
MPI_Recv блокирующий прием данных

Первые четыре функции – процедуры общего назначения, последние два – процедуры обмена между отдельными процессами.

Другие полезные процедуры общего назначения

MPI_Wtime: возвращает астрономическое время в сек.
MPI_Abort: завершение работы всей группы
MPI_Get_processor_name: имя узла, где запущен вызвавший процесс



Операции “Point-to-Point”: асинхронный обмен

MPI_Isend: Передача сообщения, аналогичная *MPI_Send*, однако возврат происходит сразу после инициализации процесса передачи без ожидания обработки всего сообщения, находящегося в *buf*. Это означает, что нельзя использовать данный буфер для других целей без получения информации о завершении данной посылки.

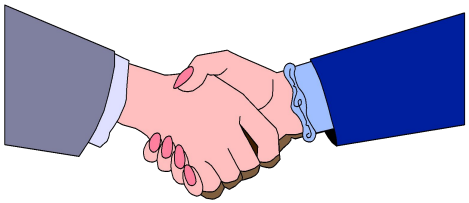
MPI_Irecv: Прием сообщения, аналогичный *MPI_Recv*, однако возврат происходит сразу после инициализации процесса приема без ожидания приема всего сообщения в *buf*.

Завершение процессов приема\передачи, когда можно использовать *buf,* можно определить с помощью специального параметра *request* и процедур **MPI_Wait и MPI_Test**.

Сообщение, отправленное **любой** из процедур *MPI_Send* и *MPI_Isend*, может быть принято **любой** из процедур *MPI_Recv* и *MPI_Irecv*.

MPI_Wait: Ожидание завершения асинхронных процедур *MPI_Isend* или *MPI_Irecv*, ассоциированных с идентификатором *request*.

MPI_Test: Проверка завершенности процедур *MPI_Isend* или *MPI_Irecv*. Параметр *flag*=1, если операция завершена, и 0 в противном случае.



Операции MPI “Point-to-Point”: Объединение запросов на взаимодействие

Несколько запросов на прием и/или передачу могут объединяться вместе для того, чтобы далее их можно было бы запустить одной командой для снижения накладных расходов на обработку обмена.

Способ приема сообщения никак не зависит от способа его посылки.

MPI_Send_init: Формирование запроса на выполнение пересылки данных. Все параметры такие же, как у *MPI_Isend*, однако пересылка не начинается до вызова *MPI_Startall*.

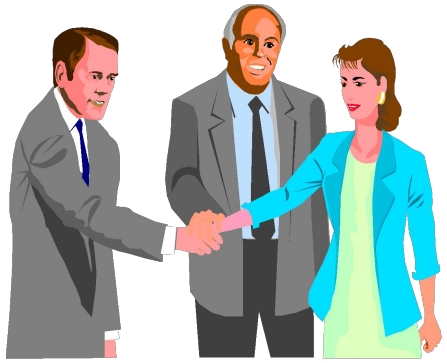
MPI_Recv_init: Формирование запроса на выполнение приема данных. Все параметры такие же, как у *MPI_Irecv*, однако реальный прием не начинается до вызова подпрограммы *MPI_Startall*.

MPI_Startall: Запуск всех отложенных взаимодействий, ассоциированных вызовами *MPI_Send_init* и *MPI_Recv_init*.

Все взаимодействия запускаются в режиме без блокировки, их завершение можно с помощью процедур *MPI_Wait* и *MPI_Test*.

Совмещенный обмен «point-to-point: **MPI_Sendrecv –** операция объединяет в едином запросе посылку и прием сообщений

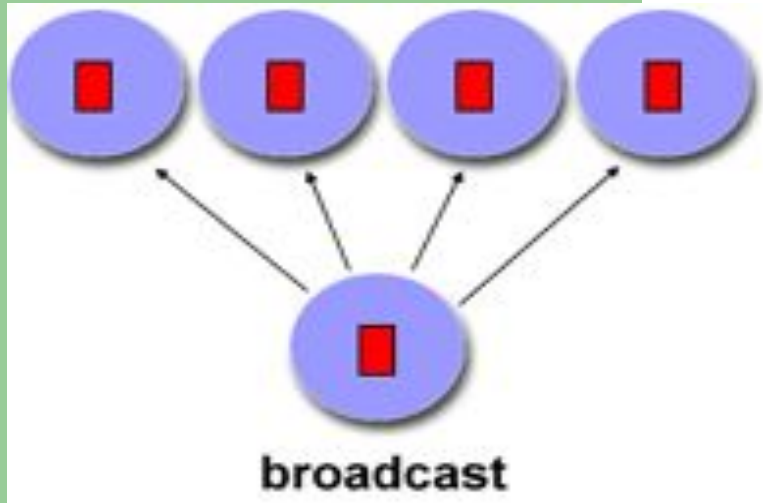
Коллективные операции MPI



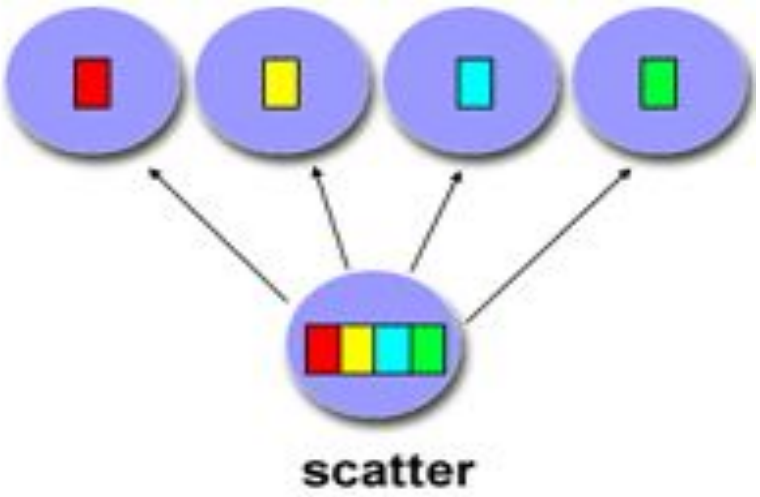
В операциях коллективного взаимодействия процессов участвуют все процессы коммуникатора. Соответствующая процедура должна быть вызвана каждым процессом, быть может, со своим набором параметров.

Возврат из процедуры коллективного взаимодействия может произойти, когда участие процесса в данной операции уже закончено. Возврат означает, что разрешен свободный доступ к буферу приема\посылки, но не означает, что операция завершена другими процессами.

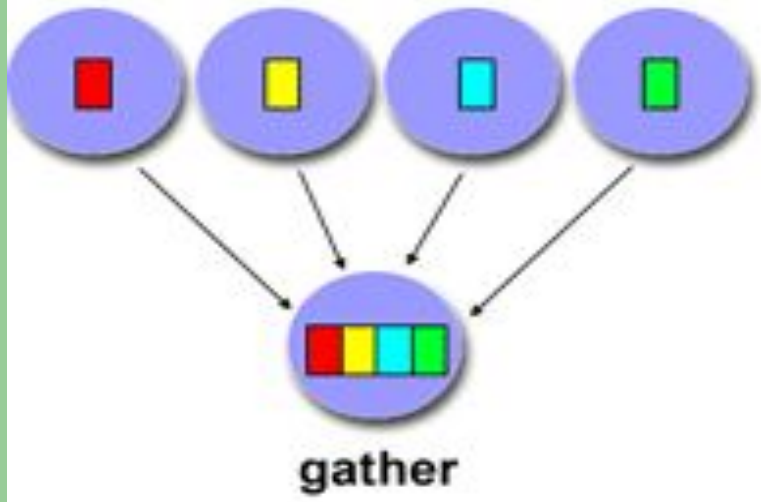
- **MPI_Barrier(comm) – барьерная синхронизация**
- **MPI_Bcast – рассылка данных всем процессам**
- **MPI_Gather – сборка данных**
- **MPI_Scatter – рассылка сегментов массива всем процессам**
- **MPI_Reduce, MPI_AllReduce – глобальные операции**



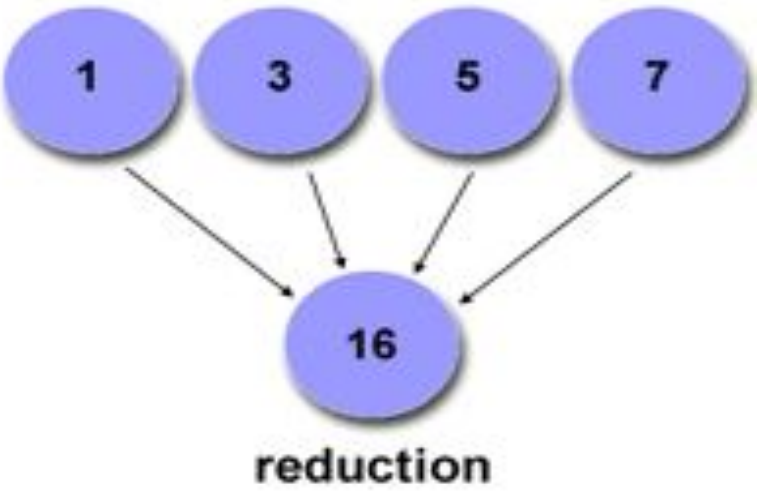
broadcast



scatter



gather



reduction

Группы и коммуникаторы

В MPI существуют широкие возможности для операций над группами процессов и коммуникаторами. Это бывает необходимо в случаях:

Во-первых, чтобы дать возможность некоторой группе процессов работать над своей независимой подзадачей.

Во-вторых, если особенность алгоритма такова, что только часть процессов должна обмениваться данными, бывает удобно завести для их взаимодействия отдельный коммуникатор.

В-третьих, при создании библиотек подпрограмм нужно гарантировать, что пересылки данных в библиотечных модулях не пересекутся с пересылками в основной программе. Решение этих задач можно обеспечить в полном объеме только при помощи создания нового независимого коммуникатора.

Группа – упорядоченное множество процессов. Каждому процессу в группе сопоставлено целое число (***ранг***). Базовая группа связана с коммуникатором **MPI_COMM_WORLD**, в нее входят все процессы приложения.

Операции с группами процессов

Новые группы можно создавать как на основе уже существующих групп, так и на основе коммуникаторов

MPI_Comm_group(COMM, GROUP). Получение группы GROUP, соответствующей коммуникатору COMM.

MPI_Group_incl(GROUP, N, RANKS, NEWGROUP). Создание группы NEWGROUP из N процессов прежней группы GROUP с рангами RANKS

MPI_Group_intersection: Создание группы из пересечения двух групп.

MPI_Group_union: Создание группы путем объединения двух групп.

MPI_Group_difference: Создание новой группы как разности двух групп.

MPI_Group_size(GROUP, SIZE). Определение количества SIZE процессов в группе GROUP.

MPI_Group_rank(GROUP, RANK). Определение номера процесса RANK в группе GROUP

MPI_Group_free(GROUP).

Операции с коммуникаторами

Коммуникатор предоставляет возможность независимых обменов данными в отдельной группе. Каждой группе процессов может соответствовать несколько коммуникаторов, но каждый в любой момент времени однозначно соответствует только одной группе. Следующие коммуникаторы создаются сразу после вызова процедуры **MPI_Init**:

MPI_COMM_WORLD – коммуникатор, объединяющий все процессы;

MPI_COMM_NULL – ошибочный коммуникатор;

MPI_COMM_SELF – коммуникатор включает только вызвавший процесс.

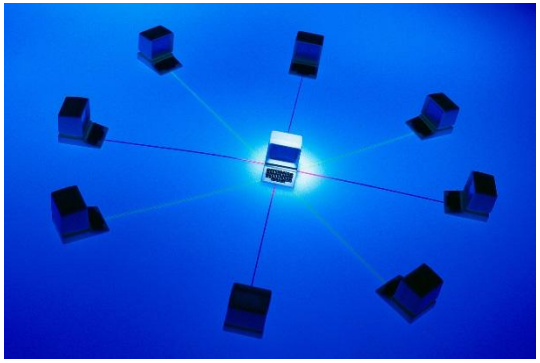
MPI_Comm_dup(COMM, NEWCOMM). ***Создание*** нового коммуникатора NEWCOMM с той же группой процессов и атрибутами, что и у коммуникатора COMM.

MPI_Comm_create. ***Создание*** нового коммуникатора из имеющегося коммуникатора для группы процессов, которая является подмножеством группы, связанной с существующим коммуникатором.

MPI_Comm_split. ***Разбиение*** коммуникатора на несколько новых.

MPI_Comm_free(COMM). ***Удаление*** коммуникатора COMM.

Виртуальные топологии



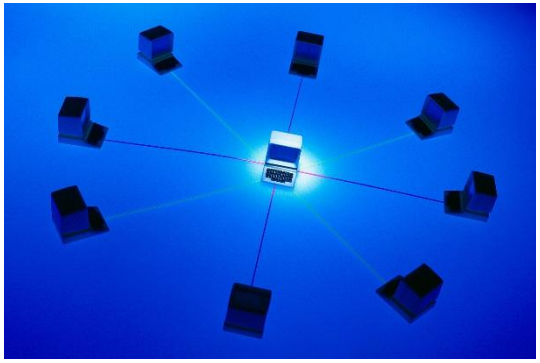
Топология – механизм сопоставления процессам некоторого коммуникатора альтернативной схемы адресации.

- Топология используется программистом для более удобного обозначения процессов, и таким образом, приближения параллельной программы к структуре математического алгоритма.
- Топология может использоваться системой для оптимизации распределения процессов по физическим процессорам используемого параллельного компьютера при помощи изменения порядка нумерации процессов внутри коммуникатора.

- **Декартова топология** (прямоугольная решетка произвольной размерности)
- **Топология графа.**

MPI_Topo_test. Процедура определения типа топологии.

Виртуальные топологии: декатрова топология



Декартова топология

MPI_Cart_create: Создание коммуникатора, обладающего декартовой топологией, из процессов существующего коммуникатора с заданной размерностью получаемой декартовой решетки

Каждому процессу ставится в соответствие набор индексов - декартовых координат в соответствии с размерностью задаваемой топологии. Если топология трехмерная – каждому процессу соответствует набор (I,j,k), определяющий его место в виртуальной решетке.

Некоторые возможности операций с декартовой топологией.

- Определение декартовых координат процесса по его рангу в коммуникаторе.
- Определение ранга процесса в коммуникаторе по его декартовым координатам.
- Расщепление коммуникатора, с которым связана декартова топология, на подгруппы (декартовым подрешеткам меньшей размерности).

Виртуальные топологии: топология графа

MPI_Graph_create. Создание на основе существующего коммуникатора нового коммуникатора с топологией графа с заданным числом вершин.

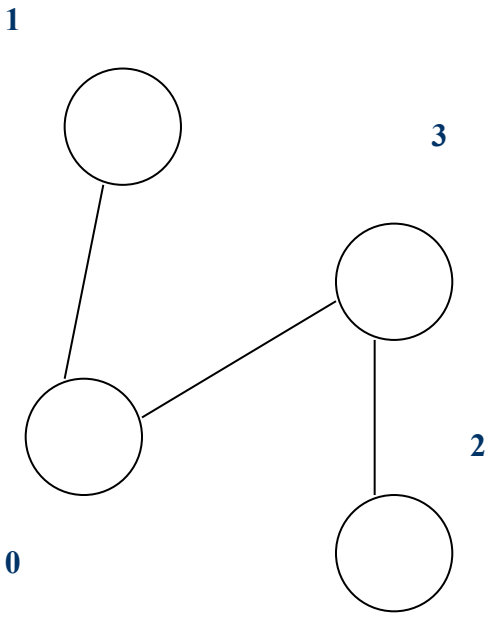
INDEX содержит суммарное количество соседей для первых I вершин.

EDGES содержит упорядоченный список номеров процессов-соседей всех вершин.

Граф определяется количеством вершин и списком их соседей

Некоторые операции с графовой топологией.

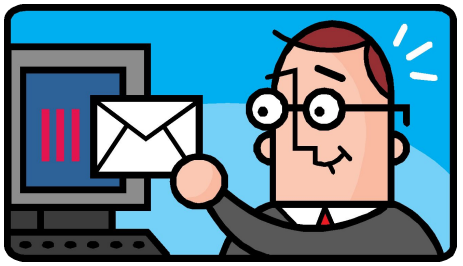
- Определение числа вершин-соседей процесса с заданным рангом.
- Определение рангов вершин-соседей процесса с заданным рангом.
- Определение числа вершин и числа ребер данной графовой топологии.



Процесс	Соседи
0	1, 3
1	0
2	3
3	0, 2

INDEX=2, 3, 4, 6
EDGES=1, 3, 0, 3, 0, 2

Пересылка разнотипных данных



Под *сообщением* в MPI понимается массив *однотипных* данных, расположенных в последовательных ячейках памяти.

Стандартно передаются расположенные подряд данные одного типа.

НО: часто в программах требуются пересылки более сложных объектов данных, состоящих из *разнотипных* элементов и\или расположенных не в последовательных ячейках памяти.

В этом случае можно либо (1) посылать данные небольшими порциями расположенных подряд элементов одного типа, либо (2) использовать копирование данных перед отсылкой в некоторый промежуточный буфер.

Оба варианта являются достаточно неудобными и требуют дополнительных затрат как времени, так и оперативной памяти.

Поэтому пересылки разнотипных данных в MPI предусмотрены два специальных способа:

- *Производные типы данных*;
- *Упаковка данных*.

Пересылка разнотипных данных: Производные типы данных

Производные типы данных создаются во время выполнения программы с помощью процедур-конструкторов на основе существующих к моменту вызова конструктора типов данных.

Создание типа данных состоит из двух этапов:

- Конструирование типа.
- Регистрация типа.

После регистрации производный тип данных можно использовать наряду с предопределенными типами в операциях пересылки, в том числе и в коллективных операциях. После завершения работы с производным типом данных его рекомендуется аннулировать. При этом все произведенные на его основе новые типы данных остаются и могут использоваться дальше.

Производный тип данных характеризуется последовательностью базовых типов данных и набором целочисленных значений смещения элементов типа относительно начала буфера обмена. Смещения могут быть как положительными, так и отрицательными, не обязаны различаться, не требуется их упорядоченность.

Пересылка разнотипных данных: Производные типы данных

Таким образом, последовательность элементов данных в производном типе может отличаться от последовательности исходного типа, а один элемент данных может встречаться в конструируемом типе многократно.

MPI_Type_contiguous(COUNT, TYPE, NEWTYPE). ***Создание*** нового типа данных NEWTYPE, состоящего из COUNT последовательно расположенных элементов базового типа TYPE. Фактически новый тип данных представляет массив данных базового типа как отдельный объект.

MPI_Type_vector(COUNT, BLOCKLEN, STRIDE, TYPE, NEWTYPE). ***Создание*** типа NEWTYPE, состоящего из COUNT блоков по BLOCKLEN элементов базового типа TYPE. Следующий блок начинается через STRIDE элементов базового типа после начала предыдущего блока.

MPI_Type_struct – создание структурного типа данных.

MPI_Type_commit – регистрация созданного производного типа данных.

MPI_Type_size – определение размера типа данных в байтах.

MPI_Type_free – аннулирование производного типа данных.

Пересылка разнотипных данных: Упаковка данных

Для пересылок разнородных данных типов можно использовать **операции упаковки и распаковки данных.** Разнородные или расположенные не в последовательных ячейках памяти данные помещаются в один непрерывный буфер, который и пересылается, далее полученное сообщение снова распределяется по нужным ячейкам памяти.

MPI_Pack(INBUF,INCOUNT,TYPE,OUTBUF,OUTSIZE,POSITION,COMM).
Упаковка INCOUNT элементов типа TYPE из массива INBUF в массив OUTBUF со сдвигом POSITION байт от начала массива. После выполнения процедуры параметр POSITION увеличивается на число байт, равное размеру записи. Параметр COMM указывает на коммуникатор, в котором в дальнейшем будет пересылаться сообщение. Для пересылки упакованных данных используется тип данных **MPI_PACKED**.

MPI_Unpack(INBUF,INSIZE,POSITION,OUTBUF,OUTCOUNT,TYPE,COMM).
Распаковка OUTCOUNT элементов TYPE из массива INBUF со сдвигом POSITION байт от начала массива в массив OUTBUF. Массив INBUF имеет размер не менее INSIZE байт.

MPI_Pack_size – определение необходимого для упаковки объема памяти