## 2.1. Общая организация библиотеки MPI

MPI-программа представляет собой набор независимых процессов, каждый из которых выполняет свою собственную программу, написанную на языке C или FORTRAN. Появились реализации MPI для C++. Процессы MPI-программы взаимодействуют друг с другом посредством вызова коммуникационных процедур. Как правило, каждый процесс выполняется в своем собственном адресном пространстве, однако допускается и режим разделения памяти.

MPI не специфицирует модель выполнения процесса – это может быть как последовательный процесс, так и многопотоковый. MPI не предоставляет никаких средств для распределения процессов по вычислительным узлам и для запуска их на исполнение. Эти функции возлагаются либо на операционную систему, либо на программиста. MPI не накладывает каких–либо ограничений на то, как процессы будут распределены по процессорам, в частности, возможен запуск MPI-программы с несколькими процессами на обычной однопроцессорной системе.

Для идентификации наборов процессов вводится понятие *группы,* объединяющей все или какую-то часть процессов. Каждая группа образует *область связи,* с которой связывается специальный объект – *коммуникатор* области связи. Процессы внутри группы нумеруются целым числом в диапазоне 0..groupsize–1. Все коммуникационные операции с некоторым коммуникатором будут выполняться только внутри области связи, описываемой этим коммуникатором. При инициализации MPI создается предопределенная область связи, содержащая все процессы MPI-программы, с которой связывается предопределенный коммуникатор MPI_COMM_WORLD. В большинстве случаев на каждом процессоре запускается один отдельный процесс, и тогда термины процесс и процессор становятся синонимами, а величина group size становится равной NPROCS – числу процессоров, выделенных задаче.

MPI достаточно объемная и сложная библиотека, состоящая примерно из 130 функций, в число которых входят:

- функции инициализации и закрытия MPI-процессов;
- функции, реализующие коммуникационные операции типа точка-точка;

- функции, реализующие коллективные операции;
- функции для работы с группами процессов и коммуникаторами;
- функции для работы со структурами данных;
- функции формирования топологии процессов.

В принципе, любая параллельная программа может быть написана с использованием всего 6 MPI-функций, а достаточно полную и удобную среду программирования составляет набор из 24 функций [35].

Каждая из MPI-функций характеризуется способом выполнения:

1. *Локальная функция* – выполняется внутри вызывающего процесса. Ее завершение не требует коммуникаций.

2. *Нелокальная функция* — для ее завершения требуется выполнение MPI-процедуры другим процессом.

3. *Глобальная функция* — процедуру должны выполнять все процессы группы. Несоблюдение этого условия может приводить к зависанию задачи.

4. *Блокирующая функция* — возврат управления из процедуры гарантирует возможность повторного использования параметров, участвующих в вызове. Никаких изменений в состоянии процесса, вызвавшего блокирующий запрос, до выхода из процедуры не может происходить.

5. *Неблокирующая функция* – возврат из процедуры происходит немедленно, без ожидания окончания операции и до того, как будет разрешено повторное использование параметров, участвующих в запросе. Завершение неблокирующих операций осуществляется специальными функциями.

В языке С все процедуры являются функциями, большинство из них возвращает код ошибки. При использовании имен подпрограмм и именованных констант необходимо строго соблюдать регистр символов. Массивы индексируются с 0. Логические переменные представляются типом int (true соответствует 1, а false – 0). Определение всех именованных констант, прототипов функций и определение типов выполняется подключением файла mpi.h. Введение собственных типов в MPI было продиктовано тем обстоятельством, что стандартные типы языков на разных платформах имеют различное представление. MPI допускает возможность запуска процессов параллельной программы на компьютерах различных

платформ, обеспечивая при этом автоматическое преобразование данных при пересылках.

### 2.2. Базовые функции библиотеки MPI

Любая прикладная MPI-программа должна начинаться с вызова функции инициализации MPI-функции MPI_Init. В результате выполнения этой функции создается группа процессов, в которую помещаются все процессы приложения, и создается область связи, описываемая предопределенным коммуникатором MPI_COMM_WORLD. Эта область связи объединяет все процессы-приложения. Процессы в группе упорядочены и пронумерованы от 0 до groupsize–1, где groupsize равно числу процессов в группе. Кроме этого, создается предопределенный коммуникатор MPI_COMM_SELF, описывающий свою область связи для каждого отдельного процесса.

Синтаксис *функции инициализации MPI_Init*:

int MPI_Init(int *argc, char ***argv)

*Функция завершения MPI-программ MP1Finalize*:

int MPI_Finalize(void)

*Функция определения числа процессов в области связи MPI_Comm_size*:

int MPI_Comm_size(MPI_Comm comm, int *size)

*Функция определения номера процесса MPI_Comm_rank*:

int MPI_Comm_rank(MPI_Comm comm, int *rank)

В минимальный набор следует включить также две функции передачи и приема сообщений.

*Функция передачи сообщения MPI_Send*:

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

*Функция приема сообщения MPI_Recv*:

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

Для определения временных параметров работы программ потребуются следующие две функции:

*Функция отсчета времени (таймер) MPI_ Wtime*:

double MPI_Wtime(void)

*Функция MPI_ Wtick*, имеющая точно такой же синтаксис, возвращает разрешение таймера (минимальное значение кванта времени).

### 2.3. Коммуникационные операции типа точка-точка

#### 2.3.1. Обзор коммуникационных операций типа точка-точка

К операциям этого типа относятся две представленные в предыдущем разделе коммуникационные процедуры. В коммуникационных операциях типа точка-точка всегда участвуют не более двух процессов: передающий и принимающий. В MPI имеется множество функций, реализующих такой тип обменов. Многообразие объясняется возможностью организации таких обменов множеством способов. Описанные в предыдущем разделе функции реализуют *стандартный режим с блокировкой.*

*Блокирующие функции* подразумевают выход из них только после полного окончания операции, т.е. вызывающий процесс блокируется, пока операция не будет завершена. Для функции посылки сообщения это означает, что все пересылаемые данные помещены в буфер (для разных реализаций MPI это может быть либо какой-то промежуточный системный буфер, либо непосредственно буфер получателя). Для функции приема сообщения блокируется выполнение других операций, пока все данные из буфера не будут помещены в адресное пространство принимающего процесса.

*Неблокирующие функции* подразумевают совмещение операций обмена с другими операциями, поэтому неблокирующие функции передачи и приема по сути дела являются функциями инициализации соответствующих операций. Для опроса завершенности операции (и завершения) вводятся дополнительные функции. Как для блокирующих, так и неблокирующих операций MPI поддерживает четыре режима выполнения. Эти режимы касаются только функций передачи данных, поэтому для блокирующих и неблокирующих

операций имеется по четыре функции посылки сообщения. В табл. 2.1 перечислены имена базовых коммуникационных функций типа точка-точка, имеющихся в библиотеке MPI.

Таблица 2.1.

Список коммуникационных функций типа точка-точка

| **Режимы** | **С блокировкой** | **Без блокировки** |
|---|---|---|
| Стандартная посылка | MPI Send | MPI Isend |
| Синхронная посылка | MPI Ssend | MPI Issend |
| Буферизованная | MPI Bsend | MPI Ibsend |
| Согласованная | MPI Rsend | MPI Irsend |
| Прием информации | MPI Recv | MPI Irecv |

Содержание выше представленной таблицы иллюстрирует принцип формирования имен функций. К именам базовых функций Send/Recv добавляются различные префиксы. Префикс S (synchronous) – означает синхронный режим передачи данных. Операция передачи данных заканчивается только тогда, когда заканчивается прием данных. Функция с префиксом S считается нелокальной. Префикс B (buffered) – означает буферизованный режим передачи данных. В адресном пространстве передающего процесса с помощью специальной функции создается буфер обмена, который используется в операциях обмена. Операция посылки заканчивается, когда данные помещены в этот буфер. Функция имеет локальный характер.

Префикс R (ready) представляет согласованный или подготовленный режим передачи данных. Операция передачи данных начинается только тогда, когда принимающий процессор выставил признак готовности приема данных, инициировав операцию приема. Функция с префиксом R является нелокальной. Префикс I (immediate) относится к неблокирующим операциям. Все функции передачи и приема сообщений могут использоваться в любой комбинации друг с другом. Функции передачи, находящиеся в одном столбце, имеют совершенно одинаковый синтаксис и отличаются только внутренней реализацией.

### 2.3.2.Блокирующие коммуникационные операции

Синтаксис базовых коммуникационных функций MPI_Send и MPI_Recv рассмотрен в подразделе 2.3.1, поэтому здесь приводится только семантика этих операций. В стандартном режиме выполнение операции обмена включает три этапа. Передающая сторона формирует

пакет сообщения, в который помимо передаваемой информации упаковываются адрес отправителя (source), адрес получателя (dest), идентификатор сообщения (tag) и коммуникатор (comm). Этот пакет передается отправителем в системный буфер, и на этом функция посылки сообщения заканчивается. Сообщение системными средствами передается адресату. Принимающий процессор извлекает сообщение из системного буфера, когда у него появится потребность в этих данных. Содержательная часть сообщения помещается в адресное пространство принимающего процесса (параметр buf), а служебная в параметр status. Поскольку операция выполняется в асинхронном режиме, адресная часть принятого сообщения состоит из трех полей:

– коммуникатора (comm), поскольку каждый процесс может одновременно входить в несколько областей связи;

– номера отправителя в этой области связи (source);

– идентификатора сообщения (tag), который используется для взаимной привязки конкретной пары операций посылки и приема сообщений.

Параметр count (количество принимаемых элементов сообщения) в процедуре приема сообщения должен быть не меньше, чем длина принимаемого сообщения. При этом реально будет приниматься столько элементов, сколько находится в буфере. Подобная реализация операции чтения связана с тем, что MPI допускает использование расширенных запросов:

– для идентификаторов сообщений (MPI_ANY_TAG читать сообщение с любым идентификатором);

– для адресов отправителя (MPI_ANY_SOURCE читать сообщение от любого отправителя).

Не допускается расширенных запросов для коммуникаторов. Расширенные запросы возможны только в операциях чтения. В этом отражается фундаментальное свойство механизма передачи сообщений: асимметрия операций передачи и приема сообщений, связанная с тем, что инициатива в организации обмена принадлежит передающей стороне. Таким образом, после чтения сообщения некоторые параметры могут оказаться неизвестными, а именно: число считанных элементов, идентификатор сообщения и адрес отправителя. Эту информацию можно получить с помощью параметра status. Переменные status должны быть явно объявлены в MPI-программе. В языке C status – это структура типа MPI_Status с тремя полями

MPI_SOURCE (процесс отправитель), MPI_TAG (идентификатор сообщения), MPI_ERROR (код ошибки). Количество считанных элементов в переменную status не заносится.

Далее перечислены основные функции этого класса:

MPI_Get_count

int MPI_Probe

MPI_Sendrecv

MPI_Sendrecv_replace

### 2.3.3. Неблокирующие коммуникационные операции

Использование неблокирующих коммуникационных операций повышает безопасность с точки зрения возникновения тупиковых ситуаций, а также может увеличить скорость работы программы за счет совмещения выполнения вычислительных и коммуникационных операций. Эти задачи решаются разделением коммуникационных операций на две стадии: инициирование операции и проверка завершения операции. Неблокирующие операции используют специальный скрытый (opaque) объект «запрос обмена» (request) для связи между функциями обмена и функциями опроса их завершения. Для прикладных программ доступ к этому объекту возможен только через вызовы MPI-функций. Если операция обмена завершена, подпрограмма проверки снимает «запрос обмена», устанавливая его в значение MPI REQUEST NULL. Снять запрос без ожидания завершения операции можно подпрограммой MPI_Request_free.

Основные функции этого типа:

MPI_Isend

MPI_Irecv

MPI_Iprobe

Имеется два типа функций завершения неблокирующих операций (ожидание завершения и проверки завершения):

1. Операции семейства WAIT блокируют работу процесса до полного завершения операции.

2. Операции семейства TEST возвращают значения TRUE или FALSE в зависимости от того, завершилась операция или нет. Они не блокируют работу процесса и полезны для предварительного определения факта завершения операции.

В MPI имеется набор подпрограмм для одновременной проверки на завершение нескольких операций. Их перечень приведен в табл. 2.2.

Кроме того, MPI позволяет для неблокирующих операций формировать целые пакеты запросов на коммуникационные операции MPI_Send_init и MPI_Recv_init, которые запускаются функциями MPI_Start или MPI_Startall. Проверка на завершение выполнения производится обычными средствами с помощью функций семейства WAIT и TEST.

Таблица 2.2.

Функции коллективного завершения неблокирующих операций

| **Выполняемая проверка** | **Функции ожидания (блокирующие)** | **Функции проверки (неблокирующие)** |
|---|---|---|
| Завершились все операции | MPI_Waitall | MPI_Testall |
| Завершилась по крайней мере одна операция | MPI_Waitany | MPI_Testany |
| Завершилась одна из списка проверяемых | MPI_Waitsome | MPI_Testsome |

## 2.4. Коллективные операции

### 2.4.1. Обзор коллективных операций

Набор операций типа точка-точка является достаточным для программирования любых алгоритмов, однако MPI вряд ли бы завоевал такую популярность, если бы ограничивался только этим набором коммуникационных операций. Одной из наиболее привлекательных сторон MPI является наличие широкого набора коллективных операций, которые берут на себя выполнение наиболее часто встречающихся при программировании действий. Например, часто возникает потребность разослать некоторую переменную или массив из одного процессора всем остальным. Каждый программист может написать такую процедуру с использованием операций Send/Recv, однако гораздо удобнее воспользоваться коллективной операцией MPI_Bcast. Причем гарантировано, что эта операция будет выполняться гораздо эффективнее, поскольку MPI-функция реализована с использованием внутренних возможностей коммуникационной среды. Главное отличие коллективных операций от операций типа точка-точка состоит в том, что в них всегда участвуют все процессы, связанные с некоторым коммуникатором. Несоблюдение

этого правила приводит либо к аварийному завершению задачи, либо к еще более неприятному зависанию задачи.

Набор коллективных операций включает:

- Синхронизацию всех процессов с помощью барьеров(MPI_Barrier).
- Коллективные коммуникационные операции, в число которых входят:
  - рассылка информации от одного процесса всем остальным членам некоторой области связи (MPI_Bcast);
  - сборка (gather) распределенного по процессам массива в один массив с сохранением его в адресном пространстве выделенного (root) процесса (MPI_Gather, MPI_Gatherv);
  - сборка (gather) распределенного массива в один массив с рассылкой его всем процессам некоторой области связи (MPI_Allgather, MPI_Allgatherv);
  - разбиение массива и рассылка его фрагментов (scatter) всем процессам области связи (MPI_Scatter, MPI_Scatterv);
  - совмещенная операция Scatter/Gather (All-to-All), каждый процесс делит данные из своего буфера передачи и разбрасывает фрагменты всем остальным процессам, одновременно собирая фрагменты, посланные другими процессами, в свой буфер приема (MPI_Alltoall, MPI_Alltoallv).
- Глобальные вычислительные операции (sum, min, max и др.) над данными, расположенными в адресных пространствах различных процессов:
  - с сохранением результата в адресном пространстве одного процесса (MPI_Reduce);
  - с рассылкой результата всем процессам (MPI_Allreduce);
  - совмещенная операция Reduce/ Scatter (MPI_Reduce_scatter);
  - префиксная редукция (MPI_Scan).

Все коммуникационные подпрограммы, за исключением MPI_Bcast, представлены в двух вариантах:

- простой вариант, когда все части передаваемого сообщения имеют одинаковую длину и занимают смежные области в адресном пространстве процессов;
- «векторный» вариант, который предоставляет более широкие возможности по организации коллективных коммуникаций, снимая ограничения, присущие простому варианту, как в части длин блоков, так и в части размещения данных в адресном пространстве процессов. Векторные варианты отличаются дополнительным символом «v» в конце имени функции.

Отличительные особенности коллективных операций:

- Коллективные коммуникации не взаимодействуют с коммуникациями типа точка-точка.
- Коллективные коммуникации выполняются в режиме с блокировкой. Возврат из подпрограммы в каждом процессе происходит тогда, когда его участие в коллективной операции завершилось, однако это не означает, что другие процессы завершили операцию.
- Количество получаемых данных должно быть равно количеству посланных данных.
- Типы элементов посылаемых и получаемых сообщений должны совпадать.
- Сообщения не имеют идентификаторов.

*Функция синхронизации процессов MPI_Barrier* блокирует работу вызвавшего ее процесса до тех пор, пока все другие процессы группы также не вызовут эту функцию. Завершение работы этой функции возможно только всеми процессами одновременно (все процессы «преодолевают барьер» одновременно).

### 2.4.2. Функции сбора блоков данных от всех процессов группы

Семейство функций сбора блоков данных от всех процессов группы состоит из четырех подпрограмм: MPI_Gather, MPI_Allgather, MPI_Gatherv, MPI_Allgatherv. Каждая из указанных подпрограмм расширяет функциональные возможности предыдущих.

### 2.4.3. Функции распределения блоков данных по всем процессам группы

Семейство функций распределения блоков данных по всем процессам группы состоит из двух подпрограмм: MPI_Scatter и MPI_Scatterv.

### 2.4.4. Совмещенные коллективные операции

*Функция MPI_Alltoall* совмещает в себе операции Scatter и Gather и является расширением операции Allgather, когда каждый процесс посылает различные данные разным получателям. Процесс i посылает j-ый блок своего буфера sendbuf процессу j, который помещает его в i-ый блок своего буфера recvbuf. Количество посланных данных должно быть равно количеству полученных данных для каждой пары процессов. *Функция MPI_Alltoallv* реализует векторный вариант операции Alltoall, допускающий передачу и прием блоков различной длины с более гибким размещением передаваемых и принимаемых данных.

### 2.4.5. Глобальные вычислительные операции над распределенными данными

В параллельном программировании математические операции над блоками данных, распределенных по процессорам, называют *глобальными операциями редукции.* В общем случае *операцией редукции* называется операция, аргументом которой является вектор, а результатом – скалярная величина, полученная применением некоторой математической операции ко всем компонентам вектора. В частности, если компоненты вектора расположены в адресных пространствах процессов, выполняющихся на различных процессорах, то в этом случае отличают *глобальную (параллельную) редукцию.* Например, пусть в адресном пространстве всех процессов некоторой группы процессов имеются копии переменной var (необязательно имеющие одно и то же значение), тогда применение к ней операции вычисления глобальной суммы или, другими словами, операции редукции SUM возвратит одно значение, которое будет содержать сумму всех локальных значений этой переменной. В MPI глобальные операции редукции представлены в нескольких вариантах:

- с сохранением результата в адресном пространстве одного процесса (MPI_Reduce);

- с сохранением результата в адресном пространстве всех процессов (MPI_Allreduce);
- префиксная операция редукции, которая в качестве результата операции возвращает вектор. I-я компонента этого вектора является результатом редукции первых i компонент распределенного вектора (MPI_Scan);
- совмещенная операция Reduce/Scatter (MPI_Reduce_scatter).

Описание синтаксиса данных функций приведено в Приложении 1.

### 2.5. Производные типы данных и передача упакованных данных

Рассмотренные ранее коммуникационные операции позволяют посылать или получать последовательность элементов одного типа, занимающих смежные области памяти. При разработке параллельных программ иногда возникает потребность передавать данные разных типов (например, структуры) или данные, расположенные в несмежных областях памяти (части массивов, не образующих непрерывную последовательность элементов). MPI предоставляет два механизма эффективной пересылки данных в упомянутых выше случаях:

- путем создания производных типов для использования в коммуникационных операциях вместо предопределенных типов MPI;
- пересылку упакованных данных (процесс-отправитель упаковывает пересылаемые данные перед их отправкой, а процесс-получатель распаковывает их после получения).

В большинстве случаев оба эти механизма позволяют добиться желаемого результата, но в конкретных случаях более эффективным может оказаться либо один, либо другой подход.

#### 2.5.1. Производные типы данных

Производные типы MPI не являются в полном смысле типами данных, как это понимается в языках программирования. Они не могут использоваться ни в каких других операциях, кроме коммуникационных. Производные типы MPI следует понимать как описатели расположения в памяти элементов базовых типов. Производный тип MPI представляет собой скрытый (opaque) объект, который специфицирует две последовательности: последовательность

базовых типов и последовательность смещений. Последовательность таких пар определяется как отображение (карта) типа:

Туретар = $\{(type_0, disp_0), \ldots, (type_{n-1}, disp_{n-1})$

Значения смещений не обязательно должны быть неотрицательными, различными и упорядоченными по возрастанию. Отображение типа вместе с базовым адресом начала расположения данных buf определяет коммуникационный буфер обмена. Этот буфер будет содержать n элементов, а i–й элемент будет иметь адрес buf+desp и иметь базовый тип type. Стандартные типы MPI имеют предопределенные отображения типов. Например, MPI_INT имеет отображение {(int,0)}.

Использование производного типа в функциях обмена сообщениями можно рассматривать как трафарет, наложенный на область памяти, которая содержит передаваемое или принятое сообщение.

Стандартный сценарий определения и использования производных типов включает следующие шаги:

- Производный тип строится из предопределенных типов MPI и ранее определенных производных типов с помощью специальных функций–конструкторов
MPI_Type_contiguous, MPI_Type_vector, MPI_Type_hvector, MPI_Type_indexed, MPI_Type_hindexed, MPI_Type_struct.
- Новый производный тип регистрируется вызовом функции MPI_Type_commit. Только после регистрации новый производный тип можно использовать в коммуникационных подпрограммах и при конструировании других типов. Предопределенные типы MPI считаются зарегистрированными.
- Когда производный тип становится ненужным, он уничтожается функцией MPI_Type_free.

Любой тип данных в MPI имеет две характеристики: протяженность и размер, выраженные в байтах:

- Протяженность типа определяет, сколько байт переменная данного типа занимает в памяти. Эта величина может быть вычислена как адрес последней ячейки данных – адрес первой ячейки данных + длина последней ячейки данных (опрашивается подпрограммой MPI_Type_extent).
- Размер типа определяет количество реально передаваемых байт в коммуникационных операциях. Эта величина равна сумме длин

всех базовых элементов определяемого типа (опрашивается подпрограммой MPI_Type_size).

Для простых типов протяженность и размер совпадают.

### 2.5.2. Передача упакованных данных

Для посылки элементов разного типа из нескольких областей памяти их следует предварительно запаковать в один массив, последовательно обращаясь к функции упаковки MPI_Pack. При первом вызове функции упаковки параметр position, как правило, устанавливается в 0, чтобы упакованное представление размещалось с начала буфера. Для непрерывного заполнения буфера необходимо в каждом последующем вызове использовать значение параметра position, полученное из предыдущего вызова.

Упакованный буфер пересылается любыми коммуникационными операциями с указанием типа MPI_PACKED и коммуникатора comm, который использовался при обращениях к функции MPI_Pack.

Полученное упакованное сообщение распаковывается в различные массивы или переменные. Это реализуется последовательными вызовами функции распаковки MPI_Unpack с указанием числа элементов, которое следует извлечь при каждом вызове, и с передачей значения position, возвращенного предыдущим вызовом. При первом вызове функции параметр position следует установить в 0. В общем случае, при первом обращении должно быть установлено то значение параметра position, которое было использовано при первом обращении к функции упаковки данных. Очевидно, что для правильной распаковки данных очередность извлечения данных должна быть той же самой, как и при упаковке. Функция MPI_Pack_size помогает определить размер буфера, необходимый для упаковки некоторого количества данных типа datatype.

## 2.6. Сравнение версий библиотеки MPI-1.1 и MPI-2

В функциональности MPI-1.1 есть пробелы, которые устранены в MPI-2. Спецификация на MPI-2 включает в себя следующие нововведения:

- Развит механизм взаимодействия между приложениями. Обеспечена поддержка механизма «клиент–сервер». В результате подобных расширений можно решать на MPI-2 не только расчетные

математические задачи, но и задачи системы массового обслуживания (обслуживания базы данных и прочее).

- Обеспечено динамическое порождение ветвей. Для программирования расчетных задач это не нужно, однако такая возможность однозначно необходима для решения задач систем массового обслуживания.
- Создан архитектурно-независимый интерфейс для работы с файлами. Это имеет особенное значение, если диск находится на одной ЭВМ, а ветвь, которая должна с ним работать – на другой. В отсутствие такого интерфейса пересылку данных необходимо было организовывать вручную, либо полагаться на сетевые возможности операционной системы. По сравнению и с тем, и с другим, MPI-2 гарантирует лучший баланс между универсальностью и быстродействием.
- Сделан шаг в сторону SMP-архитектуры. Теперь разделяемая память может быть не только каналом связи между ветвями, но и местом совместного хранения данных. Для этого ветви делегируют в MPI-2 так называемые буфера-«окна». Интерфейс выполнен так, чтобы в принципе его можно было реализовать и через передачу сообщений в системах, не относящихся к классу SMP. MPI-2 автоматически поддерживает идентичность содержимого всех «окон» с одинаковым идентификатором. Данный механизм назван термином «One-sided communications» («односторонние коммуникации»), так как ветви-получателю не требуется явно вызывать функцию приема для получения новой информации. Передача данных в ветви-отправители осуществляется с применением механизма «Remote memory access» («удаленный доступ к памяти», RMA).
- Рекомендовано использование функций MPI-2 вместо ряда функции стандарта MPI–1.1.
- Расширены многие коллективные процедуры MPI-1.1 для интеркоммуникаторов, предложены дополнительные процедуры для создания интеркоммуникаторов и разработаны две новые коллективные процедуры: обобщенное `all-to-all` и эксклюзивное сканирование.

Следует отметить что программа, написанная с использованием MPI-1.1, является программой соответствующей MPI-2. В связи с этим исследования новых направлений в развитии программного обеспечения для высокопроизводительных вычислительных систем

целесообразно начать с применения проверенной на практике библиотеки коммуникационных функций MPI-1.1.

### 2.7. Выводы

1. Библиотека функций MPI не накладывает ограничений на распределение работ по процессорам.
2. Отладка программ с включенными функциями библиотеки MPI может проводиться в однопроцессорной системе.
3. Минимально необходимый набор функций распараллеливания для любой программы составляют 6 функций библиотеки MPI.
4. Все функции передачи и приема сообщений могут использоваться в любой комбинации друг с другом.
5. Библиотека функций MPI предоставляет множество возможностей для различных реализаций одного и того же вычислительного процесса.