# 2 Термины и соглашения в MPI

Этот раздел описывает обозначения и соглашения используемые в данном документе, некоторые выбранные варианты и их целесообразность.

## 2.1 Замечания в документе

> Целесообразность. В этом формате выделены объяснения причин выбора конкретной спецификации интерфейса. Некоторые читатели могут пропустить эти замечания, в то время как читатели интересующиеся проектированием интерфейса могут захотеть прочитать их внимательнее. (Конец замечания.)
>
> Совет пользователям. В этом формате выделены материалы адресованные пользователям и иллюстрирующие способы использования описанных функций стандарта. Некоторые читатели могут пропустить эти секции, в то время как читатели интересующиеся программированием в MPI могут захотеть прочитать его внимательнее. (Конец совета пользователям.)
>
> Совет разработчикам. В этом формате выделены материалы, которые являются главным образом комментариями для разработчиков. Некоторые читатели могут пропустить эти секции, в то время как читатели интересующиеся разработкой MPI могут захотеть прочитать их внимательнее. (Конец совета разработчикам.)

## 2.2 Спецификация процедуры

Процедуры MPI определяются используя независимую систему обозначений. Аргументы вызовов процедур обозначаются как IN, OUT или INOUT. Их предназначение следующее:

- вызов использующий, но не изменяющий аргумент обозначается IN,
- вызов, который может изменять аргумент обозначается OUT,
- вызов использующий и изменяющий аргумент обозначается INOUT.

Существует специальный случай, если аргумент управляет "непрозрачным" объектом (этот термин определен в секции 2.4.1), и объект изменяется вызываемой процедурой, тогда аргумент помечается как OUT. Он помечается так даже если хэндлер  не изменяется - мы используем OUT атрибут для обозначения того, что ссылки на хэндлер изменяются.

Определения MPI стараются избегать насколько возможно, использование INOUT аргумента, потому что такое использование может привести к ошибкам, особенно для скалярных аргументов.

В общем случае для функций MPI аргумент, который используется как IN в одних процессах используется как OUT в других процессах. Такие аргументы синтаксически - INOUT аргументы и также обозначаются, хотя семантически они не используются в одном вызове для ввода и вывода одновременно.

Другая часто встречающаяся ситуация, когда значение аргумента требуется только подмножеству процессов. Если аргумент не имеет значения на данном процессе, тогда может передаваться произвольное значение.

Если не определено иначе, аргумент типа OUT или типа INOUT не может быть смешан с любым другим аргументом переданным MPI процедуре. Пример смешивания аргументов в Си появляется ниже. Если мы определяем процедуру приведённым ниже способом,

```
void copyIntBuffer(int *pin, int *puot, int len )
{ int i;
  for (i = 0; i < len; ++i) *puot++ = *pin++;
```

```
}
```

тогда её вызов в следующем кодовом фрагменте имеет смешанные аргументы.

```
int a[10];
copyIntBuffer(a, a+3, 7);
```

Хотя язык Си позволяет это, такое использование MPI процедур запрещается, если не определено иначе. Заметим, что Фортран запрещает смешивание аргументов.

Все функции MPI вначале определяются в независимом виде. Ниже приведены функции для версии ANSI C и те же функции для Fortran 77.

## 2.3 Семантические термины

При обсуждении MPI процедур использовались следующие семантические термины. Первые два обычно применяются в операциях обменов.

Не блокирующий если процедура может возвратить управление до завершения операции, то есть прежде, чем пользователю позволяется повторно использовать ресурсы (такие как буфера) указанные в вызове.

Блокирующий Если возвращение из процедуры указывает, что пользователь может повторно использовать ресурсы указанные в вызове.

Локальный Если завершение процедуры зависит только от локально выполняющихся процессов. Такие операции не требуют обменов с другими процессами пользователя.

Не локальный Если завершение процедуры может потребовать выполнения некоторых MPI процедур на других процессах. Такие операции могут потребовать обменов с другими процессами пользователя.

Групповой Если все процессы в группе процессов должны исполнять процедуру.

## 2.4 Типы данных

### 2.4.1 Скрытые объекты

MPI управляет системной памятью, которая используется для буферизации сообщений и сохранения внутренних представлений различных объектов MPI таких как группы, коммуникаторы, типы данных и так далее. Эта память недоступна пользователю на прямую, и объекты хранятся там скрыто: их размер и содержимое не видимы для пользователя. Доступ к скрытым объектам происходит через хэндлеры, которые существуют в пользовательском пространстве. Процедуры MPI, которые работают со скрытыми объектами получают хэндлеры для доступа к этим объектам. Кроме использования в вызовах MPI для доступа к скрытым объектам хэндлеры могут участвовать в присваивании и сравнении.

В Фортране все хэндлеры имеют тип INTEGER. В Си для каждой категории объектов определяется свой тип хэндлеров. Это будут типы, которые поддерживают операторы присваивания и сравнения.

В Фортране хэндлер может быть индексом в системной таблице скрытых объектов; в Си он может быть таким же индексом или указателем на объект. Существуют и более причудливые возможности.

Скрытые объекты выделяются и освобождаются специфичными для каждого типа объектов вызовами. Они перечислены в секции где, описаны объекты. Вызов принимает хэндлер как аргумент соответствующего типа. В выделяющем вызове он является аргументом

типа OUT, в котором возвращается допустимый хэндлер объекта. В освобождающих вызовах он является аргументом типа INOUT, который возвращается с недопустимым значением "неверный хэндлер". MPI предусматривает константу "неверный хэндлер" для каждого типа объекта. Сравнение этих констант используется для определения правильности хэндлера.

Освобождающий вызов присваивает хэндлеру недопустимое значение и помечает объект как освобожденный. Объект становится недоступным для пользователя после такого вызова. Однако MPI не обязательно освобождает объект немедленно. Любая ожидающая (во время высвобождения) операция, которая включает этот объект завершится нормально; объект будет освобожден после этого.

Скрытые объекты и их хэндлеры важны только для процессов, в которых они были созданы и не могут быть переданы другому процессу.

MPI предусматривает некоторые предопределенные скрытые объекты и предопределенные статические хэндлеры для этих объектов. Такие объекты не могут быть уничтожены.

Целесообразность. Такое построение скрывает внутреннее представление используемое для структур данных MPI, таким образом допуская похожие вызовы в Си и Фортране. Это также позволяет избегать конфликтов с соглашениями типов в этих языках, и позволяет просто внедрять будущие функциональные расширения. Механизм скрытых объектов используется здесь приблизительно следуя за POSIX стандартом связей Fortran'а.

Явное разделение хэндлеров в пользовательской области и объектов в системной области, позволяет производить исправление памяти и освобождающие вызовы в соответствующих местах в программе пользователя. Если скрытые объекты были бы в области пользователя, он должен быть очень осторожен, чтобы не выйти за пределы области видимости перед любым ожидаемым действием требующим формирования объекта. Указанный проект позволяет объекту быть помеченным как освобожденный, программа пользователя может тогда выходить за пределы области видимости, и объект все еще сохраняется пока любые ожидаемые действия не завершаться.

Требование, что хэндлеры поддерживают присваивание/сравнение сделано, так как такие действия общие. Это ограничивает область возможных применений. Альтернативой было бы позволить хэндлеру быть произвольным, скрытым типом. Это вынудило бы ввести подпрограммы делающие присваивание и сравнение, добавляя сложность, и был поэтому исключен. (Конец замечания.)

Совет пользователям. Пользователь может случайно создать повисшую ссылку, присваивая хэндлеру значение другого хэндлера, а затем высвободить объект ассоциирующийся с этим хэндлером. Обратно, если хэндлер освобождается до того как освободится ассоциированный с ним объект, то объект становится недоступным (это может произойти, например, если хэндлер локальная переменная внутри подпрограммы, и подпрограмма завершается до освобождения объекта ассоциированного с хэндлером). На пользователях лежит ответственность избегать добавления или уничтожения ссылок на скрытые объекты, за исключением вызовов, которые выделяют или освобождают такие объекты. (Конец совета пользователям.)

Совет разработчикам. Имеется в виду что семантика скрытых объектов такая, что каждый скрытый объект отделен от другого; каждый вызов для выделения такого объекта копирует всю информацию необходимую для объекта. Реализации должны избегать чрезмерного копирования заменяя его ссылками. Например, предпочтительнее, чтобы полученный тип данных мог содержать ссылки на свои компоненты чем их копии; предпочтительнее, чтобы вызов MPI_COMM_GROUP мог вернуть ссылки на группы ассоциирующиеся с коммуникатором, чем копии этих групп. В этом

случае реализация должна сохранять счетчик ссылок, выделять и освобождать объекты так, чтобы видимый эффект был такой как будто объекты копировались. (Конец совета разработчикам.)

### 2.4.2 Массив аргументов

Вызов MPI может нуждаться в аргументе, который является массивом скрытых объектов, или массивом хэндлеров. Массив хэндлеров - регулярное множество с элементами, которые являются хэндлерами объектов одного типа в последовательно расположенных в массиве. Всякий раз, когда используется такое множество, требуется дополнительный аргумент len, чтобы указать число допустимых элементов (если это количество не может быть получено иначе). Допустимые элементы располагаются в начале массива; len указывает сколько их имеется, и не обязан быть полным размером массива. Тот же самый подход применяется для других массивов аргументов.

### 2.4.3 Состояние

MPI процедуры используют в различных местах аргументы с типами состояния. Все выражения такого типа данных идентифицированы именами, и нет никаких операций для их определения. Например, стандартная подпрограмма MPI_ERRHANDLER_SET имеет аргумент типа состояние со значениями MPI_ERRORSFATAL, MPI_ERRORSRETURN, и т.д.

### 2.4.4 Поименованные константы

MPI процедуры иногда используют специальные значения основного аргумента типа; например tag - целочисленный аргумент операций двухточечных обменов, может иметь специальное значение, MPI_ANY_TAG. Такие аргументы будут иметь диапазон регулярных значений, который является надлежащим подмножеством диапазона значений соответствующего основного типа; специальные значения (типа MPI_ANY_TAG) будут вне регулярного диапазона. Диапазон регулярных выражений можно определить используя функции запроса окружения. (Глава 7).

### 2.4.5 Выбор

Функции MPI иногда используют аргументы с типами данных choice (или объединение). Особые вызовы такой процедуры могут передавать ссылкой фактические параметры различных типов. Механизм для обеспечения таких аргументов будет отличаться от языка к языку.

### 2.4.6 Адреса

Некоторые MPI процедуры используют аргументы адреса, которые представляют абсолютный адрес в вызываемой программе. Тип данных такого аргумента целочисленный размера необходимого, чтобы содержать любой допустимый адрес.

## 2.5 Привязка к языку

Эта часть определяет правила для привязки MPI к языку в общем и для Фортрана77 и ANSI C в частности. Здесь определены различные представления объектов, а также условные обозначения используемые для определения этого стандарта. Фактические вызывающие последовательности определены в другом месте.

Ожидается что любые реализации Fortran90 и C++ используют привязки к Fortran77 и ANSI C, соответственно. Хотя мы полагаем, что MPI предварительно определяет другие привязки к Fortran90 и C++, текущие привязки разработаны, чтобы поощрить, а не отбить охоту, от экспериментирования с лучшими привязками, которые могли бы быть приняты позже.

Так слово PARAMETER - ключевое слово в языке Fortran, мы используем слово "аргумент", чтобы обозначить аргументы подпрограммы. Они обычно упомянуты как параметры в Си, однако, мы ожидаем что Си программисты будут понимать слово "аргументы"

(которое не имеет никакого определенного значения в Си), таким образом позволяя нам избежать ненужного беспорядка для программистов Фортрана.

Имеются несколько важных спорных вопросов привязки к языку не обращенные к этому стандарту. Этот стандарт не обсуждает возможности передачи сообщений между языками. Ожидается, что большое количество реализаций будут иметь такие особенности, и что такие особенности - признак качества реализации.

### 2.5.1 Проблемы привязки к Фортрану77

Все имена MPI имеют префикс MPI_, и все символы в названиях заглавные.

Все Фортрановские подпрограммы MPI имеют код возврата в последнем аргументе. Некоторые операции MPI являются функциями, которые не имеют аргумента кода возврата. Значение кода возврата при успешном завершении MPI_SUCCESS; коды ошибок только потенциально независимы от реализации; см. Главу 7.

Хэндлеры представляются в Fortran'е как INTEGER.

Массивы аргументов индексируются с единицы.

Если явно не указано, привязки MPI F77 не противоречат стандарту ANSI Fortran77. Имеются несколько мест где этот стандарт отклоняется от стандарта ANSI Fortran77. Эти исключения не противоречат общей практике принятой пользователями Фортрана. В частности:

- MPI идентификаторы ограничены тридцатью, а не шестью, значимыми символами.

- Идентификаторы MPI могут содержать подчеркивание после первого символа.

- Подпрограмма MPI с аргументом типа choice может быть вызвана с различными типами аргументов. Пример приведён ниже. Это нарушает букву стандарта Fortran'a, но такое нарушение - общая практика. Альтернативой будет иметь отдельную версию MPI_SEND для каждого типа данных.

  Пример вызова стандартной процедуры с несовпадающими формальными и фактическими параметрами.

```
double precision a
integer b
...
call MPI_send(a,...)
call MPI_send(b,...)
```

- Хотя это не требуется, предполагается, что MPI константы описываются в файле названном mpif.h. В системах не поддерживающих включаемые файлы разработка должна определять значение этих констант.

- Поставщики поддержали обеспечение объявления типов в файле mpif.h для Фортрановских систем, которые поддерживают типы определенные пользователем. В нем должен быть определен, если возможно, тип MPI_ADDRES который должен быть целым такого размера чтобы мог содержать адрес в окружающей среде выполнения. В системах где определение типов не поддерживается, пользователь должен использовать целое правильного типа для представления адресов (то есть, INTEGER*4 на 32 разрядной машине, INTEGER*8 на 64 разрядной машине, и т.д.).

### 2.5.2 Проблемы привязки к Си

Мы используем формат описания ANSI C. Все имена в MPI имеют префикс MPI_, в определенных константах все буквы заглавные, а

определенные типы и функции имеют одну заглавную букву после префикса.

Определения констант, прототипов функций и определения типов должны содержаться в файле mpi.h.

Почти все Си функции возвращают код ошибки. Код успешного завершения должен быть MPI_SUCCESS, но код неудачного завершения зависит от разработки. Некоторые Си функции не возвращают значения, таким образом они могут выполнятся как макросы.

Предусматривается объявление типов для хэндлеров каждой категории скрытых объектов. Используется либо указатель либо целое.

Элементы массива индексируются с нуля.

Логические флаги - это целые со значением 1 и 0 обозначающие "true" и "false" соответственно.

Chioce аргументы это указатели типа void*.

Аргументы адреса являются определенным типом MPI MPI_Aint. Он определен как тип int размера необходимого для содержания любого действительного адреса на конкретной архитектуре.

## 2.6 Процессы

Программа MPI состоит из автономных процессов, выполняющих свой собственный код в MIMD стиле. Коды выполняемые каждым процессом не обязаны быть идентичны. Обычно, каждый процесс выполняется в собственном адресном пространстве, хотя возможны реализации MPI программ в разделяемой памяти. Этот документ определяет поведение параллельной программы предполагая, что для связи используются только MPI вызовы. Взаимодействие программы MPI с другими возможными средствами связи (например, разделенная память) не определено.

MPI не определяет модель выполнения для каждого процесса. Процесс может быть последовательным, или многопотоковым, с потоками возможно выполняющимися одновременно. Была предпринята попытка сделать MPI "потоко-защищенным", избегая использования полного режима. Хотелось добиться такого взаимодействия MPI с потоками, чтобы все параллельные потоки позволяли выполнять MPI вызовы и MPI подпрограммы можно было вызывать дважды; блокирующий MPI вызов блокирует только вызвавший поток, позволяя продолжение работы другого потока.

MPI не предусматривает механизма, чтобы определить начальное распределение процессов для MPI вычислений и их закреплению на физические процессоры. Ожидается что поставщики будут обеспечивать механизмы, чтобы делать это или во время загрузки или во время выполнения. Такие механизмы будут позволять определение начального числа требуемых процессов, код выполняется каждым начальным процессом, и процессы распределяются на процессоры. Также, текущее предложение не предусматривает динамическое создание или уничтожение процессов во время выполнения программы (общее число процессов фиксировано), хотя намечена согласованность с такими расширениями. Для идентификации процесса используется его ранк относительно группы, определяемой коммуникатором. Ранк может принимать любое целое из промежутка 0..groupsize-1. (где groupsize - размер группы, ранки всех процессов группы различны)

## 2.7 Обработка ошибок

MPI обеспечивает пользователя надежной передачей сообщений. Посланное сообщение всегда будет получено правильно, и пользователь не должен проверять наличие ошибки передачи, тайм-аут или другие условия ошибки. Другими словами, MPI не предусматривает механизмов поведения в системе связи с отказами. Если MPI разработка построена на ненадежном основном механизме, тогда работой разработчика MPI подсистемы является изолировать пользователя от этой ненадежности, или отображать отказы как невосстанавливаемые ошибки. Аналогично, MPI не обеспечивает никаких механизмов для обработки отказов. Возможные варианты обработки ошибок описаны в части 7.2 и могут использоваться для ограничения сферы невосстанавливаемых ошибок или разработки восстановления после ошибок на уровне приложений.

Конечно, MPI программы могут все еще быть ошибочными. Программная ошибка может происходить, когда MPI процедура вызвана с неправильным аргументом (несуществующее место назначения в операции передачи, слишком маленький буфер в операции приема, и т.д.) Этот тип ошибки происходил бы в любой разработке. Кроме того, ошибка ресурса может происходить когда программа превышает количество доступных ресурсов системы (число ожидаемых сообщений, системных буферов, и т.д.). Возникновение этого типа ошибки зависит от количества доступных ресурсов в системе и используемого механизма их распределения; что может отличаться от системы к системе. Разработка высокого качества должна предусматривать широкие пределы на важные ресурсы, чтобы облегчить решение этой проблемы.

Почти все MPI вызовы возвращают код, который указывает на успешное завершение операции. Где возможно вызовы MPI возвращают код ошибки, если во время вызова произошла ошибка. По умолчанию ошибка обнаруживается во время исполнения MPI библиотеки вызывая окончание параллельных вычислений. Однако MPI предусматривает для пользователей механизм для замены этого умолчания и обработки восстанавливаемых ошибок. Пользователь может определить, что ошибка не фатальна и самостоятельно обработать код ошибки возвращаемый MPI. Также пользователь может предусмотреть свои собственные подпрограммы обработки ошибок, которые будут выполняться, когда вызов MPI завершиться ненормально. Возможности управления обработкой ошибок описаны в части 7.2.

Несколько факторов ограничивает способность MPI вызовов возвращаться со значимым кодом ошибки, когда она (ошибка) происходит. MPI может не быть способным определить некоторые ошибки; другие ошибки могут быть очень дорого стоящими для их определения в нормальном режиме выполнения; наконец некоторые ошибки могут быть "катастрофичными" и мешать MPI вернуть управление вызывающему в непротиворечивом состоянии.

Возникает другой тонкий вопрос, из-за того что по природе связь асинхронная: MPI вызовы могут инициализировать операции, которые продолжают асинхронно выполняться после завершения вызова. Так операция может закончиться с успешным кодом завершения, однако позднее явиться причиной ошибки вызывающей особую ситуацию. Если это последовательный вызов относящийся к той же самой операции (например, вызов который проверяет что асинхронная передача завершилась), тогда код ошибки ассоциированный с этим вызовом можно использовать для определения природы этой ошибки. В некоторых случаях ошибка может произойти после всех вызовов, которые относятся к завершенной операции, тогда не будет использовано никакого значения для определения природы ошибки (например, ошибка в передаче в режиме готовности). Такие ошибки должны обрабатываться как фатальные, так как информацию невозможно вернуть для того, чтобы пользователь воспользовался ей.

Этот документ не определяет состояния вычислений после ошибочного MPI вызова. Желательно такое поведение, чтобы возвращался подходящий код ошибки, и влияние ошибки локализовалось в большинстве возможных случаев. Например, очень желательно чтобы ошибочный вызов приема не вызывал записей в любую часть памяти получателя, вне области определенной для приема сообщений.

Разработки могут выходить за рамки этого документа в поддерживании полезных способов MPI вызовов, определение которых здесь является ошибкой. Например, MPI определяет точный тип соответствующих правил между соответствующими операциями передачи и приема: будет ошибкой послать вещественную переменную и получать целое. Разработки должны выходить вне этих типов соответствующих правил, и предусматривать автоматическую конверсию типов в такой ситуации. Было бы полезным формировать предупреждающие сообщения для такого несогласующегося поведения.

## 2.8 Вопросы разработки

Имеется ряд областей, где MPI реализация может взаимодействовать с операционной средой и системой. Пока MPI не

предписывает обеспечения какого-либо сервиса (типа ввода - вывода или обработки сигнала), так как это налагает сильные ограничение на поведение если тот сервис доступен. Это - важный пункт в достижении мобильности между платформами, которые обеспечивают тот же самый набор услуг.

### 2.8.1 Независимость основных runtime подпрограмм

MPI программы требуют, чтобы библиотека стандартных подпрограмм, которая является частью основного окружения языка (таких как date и write в Фортране, а также printf и malloc в ANSI C) выполнялись после MPI_INIT и до MPI_FINALIZE работали независимо, и чтобы их завершение не зависело от действий других процессов в MPI программе.

Обратите внимание, что это никоим образом не предотвращает создание библиотеки стандартных подпрограмм, которая обеспечивает параллельный сервис чьи операции коллективны. Однако следующая программа как ожидается завершена в ANSI C окружении не считаясь с размером MPI_COMM_WORLD (допуская, что ввод/вывод доступен в узлах выполнения).

```
int rank;
MPI_Init ( argc, argv );
MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
if (rank == 0) printf( "Starting program\n" );
MPI_Finalize();
```

Соответствующая программа для Fortran77 также ожидается к завершению.

Пример того что не требуется особенный порядок действий этих подпрограмм, когда вызываются несколькими задачами. Например MPI не налагает никаких требований или рекомендаций для вывода из следующих программ (снова надеющихся что ввод/вывод доступен в узлах выполнения).

```
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
printf( "Output from task rank %d\n", rank);
```

Кроме того, запросы, которые терпят неудачу из-за истощения ресурса или другой ошибки не рассматриваются здесь нарушением требований (однако, требуется, чтобы они закончились, не обязательно успешно).

### 2.8.2 Взаимодействие с сигналами POSIX

MPI не определяет взаимодействия процессов с сигналами в окружении UNIX или с другими событиями, которые не имеют отношения к обменам MPI. Эти сигналы не значимы с точки зрения MPI, и разработчики должны попытаться построить MPI так, чтобы сигналы были прозрачны: вызов MPI приостановленный сигналом должен возобновиться и завершиться после обработки сигнала. В общем, состояние вычислений, которые видимы и значимы с точки зрения MPI должно находиться под воздействием только вызов MPI.

Намерение MPI быть потоковой и сохраняющей сигнал имеет ряд тонких эффектов. Например, в UNIX системах, захватывающий сигнал такой как SIGALRM (сигнал тревоги) не должен заставлять MPI подпрограмму вести себя не так, как она себя ведет в отсутствие этого сигнала. Конечно вызов сигнала результат MPI вызовов или изменений в окружении, в котором MPI процедура работает (например, использование всего пространства памяти), MPI процедура должна подходяще вести себя для этой ситуации (в частности, в этом случае поведение должно быть такое же как при многопотоковом MPI выполнении).

Второй эффект такой, что обработка сигналов, которую выполняют MPI вызовы не должна мешать операциям MPI. Например, MPI прием для любого типа, который происходит внутри обработки сигнала не должен быть причиной ошибочного поведения MPI процедуры. Заметим, что

допускается реализация запрещающая использования MPI вызовов внутри обработчика сигналов, и не требующая регистрировать такое использование.

Очень желательно чтобы MPI не использовал SIGALRM, SIGFPE или SIGIO. Разработка требует ясный список всех сигналов, которые использует реализация MPI; подходящее место для этой информации это Unix "man" на MPI.