MPI
Message Passing Interface

http://www.mpi-forum.org/

Что это?

MPI – интерфейс передачи сообщений

Это библиотека функций, предназначенная для поддержки работы параллельных процессов в терминах передачи сообщений

Кому он нужен, этот MPI?

Этот стандарт предназначен для пользователей, которые хотят писать переносимые программы для передачи сообщений на языках C/C++ и Fortran.

К ним относятся прикладные программисты, разработчики программного обеспечения для параллельных машин и создатели исполнительных сред, реализующих MPI.

Почему MPI?

- единый стандарт

единственная технология, реализации которой существуют для всех HPC платформ

Почему MPI?

- переносимость

Можно использовать один и тот же исходный код и быть уверенным, что он будет работать одинаково на всех платформах, для которых существует поддержка MPI

Почему MPI?

- производительность

Поставщики оборудования могут иметь свои собственные реализации (или расширения), «заточенные» под их аппаратное обеспечение.

Почему MPI?

- функциональность

Несколько сотен различных функций на все случаи жизни

Почему MPI?

- доступность

Существует множество реализаций, как свободных, так и закрытых.

OpenMPI, mpich, mvapich и многие другие.

Версии стандарта

MPI-1.0 — июнь 1994
MPI-1.1 — 12 июня 1995
MPI-1.2 — 18 июля 1997
MPI-2.0 — 18 июля 1997
MPI-1.3 — 30 мая 2008
MPI-2.1 — 23 июня 2008
MPI-2.2 — 4 сентября 2009
MPI-3.0 — 21 сентября 2012

Цели

- API

- Эффективный обмен данными: избегать копирования память-память, совмещение операций обмена с вычислениями и разгрузка коммуникационного сопроцессора

- Возможность реализаций, позволяющих работу в гетерогенной среде

- Привязки к C, C++, Fortran-77 и Fortran-95

Цели

- Обеспечить надежный коммуникационный интерфейс: с неисправностями должен справляться не пользователь, а нижележащие слои коммуникационной подсистемы
- Определить интерфейс, который может быть имплементирован на множестве различных платформ, без значительных изменений в аппаратной и системной частях
- Семантика интерфейса должна быть независима от языка программирования
- Потоковая безопасность (thread safety)

Что входит в стандарт?

- Парные обмены (p-t-p communications)
- Типы данных
- Коллективные операции
- Группы процессов
- Коммуникационные контексты
- Топологии процессов
- Управление и запросы к вычислительной среде (Environmental Management and inquiry)
- Привязки к языкам Fortran, C и C++
- Интерфейс профилирования

Что входит в стандарт? 2.0

- Возможность запуска новых процессов во время выполнения MPI- программы;
- Односторонние двухточечные обмены;
- Параллельные операции ввода-вывода;
- Модифицированные привязки к языкам;
- Новые предопределённые типы данных;
- Расширенные возможности коллективных обменов;
- Внешние интерфейсы;
- Поддержка многопоточности и другие.

Передача сообщений: терминология

Терминология

Процесс

Программы MPI состоят из автономных процессов, выполняющих собственный код, написанный в стиле MIMD.

Процессы взаимодействуют через вызовы коммуникационных примитивов MPI.

Обычно каждый процесс выполняется в его собственном адресном пространстве

Терминология

Номер процесса

целое неотрицательное число, являющееся уникальным атрибутом каждого процесса в группе.

Терминология

Атрибуты сообщения

номер процесса-отправителя, номер процесса-получателя, идентификатор сообщения и коммуникатор.

Для них заведена структура MPI_Status, содержащая три поля:
MPI_Source (номер процесса отправителя),
MPI_Tag (идентификатор сообщения),
MPI_Error (код ошибки); могут быть и добавочные поля

Терминология

Идентификатор сообщения (msgtag)

атрибут сообщения, являющийся целым неотрицательным числом, лежащим в диапазоне от 0 до 32767

Терминология

Процессы объединяются в **группы**.

Внутри группы все процессы перенумерованы. С каждой группой ассоциирован свой **коммуникатор**.

При осуществлении пересылки необходимо указать идентификатор группы, внутри которой производится эта пересылка.

Соглашения по названиям процедур

C: MPI_Class_action_subset
(MPI_Action_subset)

C++: MPI::Class::Action_subset
(MPI::Action_subset)

Fortran: MPI_CLASS_ACTION_SUBSET
(MPI_ACTION_SUBSET)

Длина идентификаторов
ограничена 30 символами

Спецификация процедур

Возвращает 0 в случае успеха.

Принимает 3 типа аргументов:

IN : вызов использует, но не изменяет аргумент

OUT: вызов может изменять аргумент

INOUT: вызов использует и изменяет аргумент

Соответствия типов данных

MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Hello, World!

Исходный код

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char* argv[])
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello, world, I am %d of %d\n",
rank, size);
    MPI_Finalize();

    return 0;
}
```

Компиляция и запуск

```
# mpicc hello.c -o hello

# mpirun -np 5 --host 127.0.0.1,localhost hello
Hello, world, I am 4 of 5
Hello, world, I am 1 of 5
Hello, world, I am 0 of 5
Hello, world, I am 3 of 5
Hello, world, I am 2 of 5
```

Инициализация и завершение

Инициализация

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

Инициализация параллельной части программы.
Все другие процедуры MPI могут быть вызваны только после вызова MPI_Init.

Завершение

```
void Finalize()
```

Завершение параллельной части приложения. Все последующие обращения к любым процедурам MPI, в том числе к MPI_INIT, запрещены.

Парная коммуникация

Блокирующая передача

```
MPI_SEND(buf, count, datatype, dest, tag, comm)

IN buf начальный адрес буфера посылки сообщения
(альтернатива)
IN count число элементов в буфере посылки
(неотрицательное целое)
IN datatype тип данных каждого элемента в
буфере посылки (дескриптор)
IN dest номер процесса-получателя (целое)
IN tag тэг сообщения (целое)
IN comm коммуникатор (дескриптор)
```

Блокирующая передача

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

MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM,
IERROR)
     <type> BUF(*)
     INTEGER COUNT, DATATYPE, DEST, TAG, COMM,
IERROR

void MPI::Comm::Send (const void* buf,
     int count, const MPI::Datatype& datatype,
     int dest, int tag) const
```

Блокирующий прием

MPI_RECV (buf, count, datatype, source, tag, comm, status)

OUT buf начальный адрес буфера процесса-получателя (альтернатива)
IN count число элементов в принимаемом сообщении (целое)
IN datatype тип данных каждого элемента сообщения (дескриптор)
IN source номер процесса-отправителя (целое)
IN tag тэг сообщения (целое)
IN comm коммуникатор (дескриптор)
OUT status статус (параметры) принятого сообщения (статус)

Блокирующий прием

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

MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG,
COMM, STATUS, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM,
    STATUS(MPI_STATUS_SIZE), IERROR

void MPI::Comm::Recv (void* buf, int count,
     const MPI::Datatype& datatype,
     int source, int tag,
     MPI::Status& status) const
```

Блокирующий обмен: пример

```
char message[100];
int rank, size, i, tag, node;
MPI_Status status;
. . . . .
if (rank == 0)
   for( i=1; i<size; i++){
       MPI_Recv( message, 100, MPI_CHAR, i,
           tag, MPI_COMM_WORLD, &status);
       printf ("node:%d  %s\n", rank, message);
   }
else
   MPI_Send( message, sizeof(message),
       MPI_CHAR, 0, tag, MPI_COMM_WORLD);
```

Коммуникационные режимы

- Стандартный (блокирующий)
- Буферизированный
- Синхронный
- Режим по готовности

Режимы передачи

Коммуникационный режим отмечается одной префиксной буквой:

```
MPI_[I][R|S|B]Send
```

I — обозначает наблокирующую операцию
R – передача по готовности
S – синхронный
B – буферизированный

Режимы передачи

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

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

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

Стандартный

Нежелание разрешать в стандартном режиме буферизацию проистекает от стремления сделать программы переносимыми — программа не должна зависеть в стандартном режиме от системной буферизации.

Передача может быть завершена до окончания приема. Посылка в стандартном режиме является нелокальной операцией: она может зависеть от условий приема.

Буферизированный

Может стартовать вне зависимости от того, инициирован ли соответствующий прием.

Однако, в отличие от стандартной посылки, эта операция является локальной и ее завершение не зависит от обстоятельств приема.

Зависит от размера буфера.

Добавление/удаление буфера для приема сообщений

```
MPI_BUFFER_ATTACH(buffer, size)
IN buffer ссылка на буфер (choice)
IN size размер буфера в байтах(non-negative integer)

int MPI_Buffer_attach(void* buffer, int size)

MPI_BUFFER_DETACH(buffer_addr, size)
OUT buffer_addr  ссылка на буфер(choice)
OUT size  размер буфера в байтах(non-negative
integer)

int MPI_Buffer_detach(void* buffer_addr, int* size)
```

Синхронный

Может стартовать вне зависимости от того, был ли начат соответствующий прием.

Однако, посылка будет завершена успешно, только если соответствующая операция приема стартовала. Следовательно, завершение синхронной передачи не только указывает, что буфер отправителя может быть повторно использован, но также и отмечает, что получатель достиг определенной точки в своей работе.

По готовности

Может быть запущена только тогда, когда прием уже инициирован. В противном случае операция является ошибочной и результат будет неопределенным.

Завершение операции посылки не зависит от состояния приема и в основном указывает, что буфер посылки может быть повторно использован.

Неблокирующий обмен

Дополнительный префикс I (immediate - непосредственный) указывает, что вызов неблокирующий (MPI_Isend).

Префикс B, S, или R используются соответственно для буферизованного, синхронного режима или режима готовности (MPI_Ibsend, MPI_Issend, MPI_Irsend).

Неблокирующий режим: передача

```
MPI_ISEND(buf, count, datatype, dest, tag, comm,
request)
```

IN buf начальный адрес буфера посылки (альтернатива)
IN count число элементов в буфере посылки (целое)
IN datatype тип каждого элемента в буфере посылки (дескриптор)
IN dest номер процесса-получателя (целое)
IN tag тэг сообщения (целое)
IN comm коммуникатор (дескриптор)
OUT request запрос обмена (дескриптор)

Неблокирующий режим: передача

```
int MPI_Isend(void* buf, int count,
     MPI_Datatype datatype, int dest, int tag,
     MPI_Comm comm, MPI_Request *request)

MPI_ISEND (BUF, COUNT, DATATYPE, DEST, TAG, COMM,
REQUEST, IERROR)
<type> BUF(*)

MPI::Request Comm::Isend(const void* buf, int
     count, const MPI::Datatype& datatype,
     int dest, int tag) const
```

Неблокирующий режим: прием

```
MPI_IRECV(buf, count, datatype, source, tag,
comm, request)
```

IN buf начальный адрес буфера приема (альтернатива)
IN count число элементов в буфере приема(целое)
IN datatype тип каждого элемента в буфере посылки (дескриптор)
IN source номер процесса-отправителя (целое)
IN tag тэг сообщения (целое)
IN comm коммуникатор (дескриптор)
OUT request запрос обмена (дескриптор)

Неблокирующий режим: прием

```
int MPI_Irecv(void* buf, int count,
     MPI_Datatype datatype, int source,
     int tag, MPI_Comm comm,
     MPI_Request *request)

MPI_IRECV(BUF, COUNT, DATATYPE, SOURCE, TAG,
COMM, REQUEST, IERROR)

<type> BUF(*)

INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM,
REQUEST, IERROR

MPI::Request MPI::Comm::Irecv(void* buf,
     int count, const MPI::Datatype& datatype,
     int source, int tag) const
```

Неблокирующий режим

Эти вызовы создают объект коммуникационного запроса и связывают его с дескриптором запроса (аргумент request). Запрос может быть использован позже, чтобы узнать статус обмена или чтобы ждать его завершения.

Отправитель не должен обращаться к любой части буфера посылки после того, как вызвана операция неблокируемой передачи, пока посылка не завершится.

Неблокирующий режим: завершение передачи

Чтобы завершить неблокирующий обмен, используются функции MPI_WAIT и MPI_TEST.

Завершение операции посылки указывает, что отправитель теперь может изменять содержимое ячеек буфера посылки (операция посылки сама не меняет содержание буфера).

Операция завершения не извещает, что сообщение было получено, но дает сведения, что оно было буферизовано коммуникационной подсистемой.

MPI_WAIT

```
MPI_WAIT(request, status)
INOUT request  request (handle)
OUT status  status object (Status)

int MPI_Wait(MPI_Request *request, MPI_Status
*status)

MPI_WAIT(REQUEST, STATUS, IERROR)
INTEGER REQUEST, STATUS(MPI_STATUS_SIZE),
IERROR

void MPI::Request::Wait(MPI::Status& status)
```

MPI_TEST

```
MPI_TEST(request, flag, status)
INOUT request  request (handle)
OUT flag  true если операция завершена(logical)
OUT status  status object (Status)

int MPI_Test(MPI_Request *request, int *flag,
MPI_Status *status)

MPI_TEST(REQUEST, FLAG, STATUS, IERROR)
LOGICAL FLAG
INTEGER REQUEST, STATUS(MPI_STATUS_SIZE),
IERROR

bool MPI::Request::Test(MPI::Status& status)
```

Неблокирующий режим: пример

```
if( rank == 0){
   MPI_Isend( buf, 100, MPI_CHAR, 1, 99,
      MPI_COMM_WORLD, &req);

// здесь могли бы быть ваши вычисления

   MPI_Wait( &req, &status);
  }
 else{
   MPI_Irecv( buf, 100, MPI_CHAR, 0, 99,
      MPI_COMM_WORLD, &req);

// здесь могли бы быть ваши вычисления

   MPI_Wait( &req, &status);
  }
```

Проба и отмена

Операции MPI_PROBE и MPI_IPROBE позволяют проверить входные сообщения без их реального приема. Пользователь затем может решить, как ему принимать эти сообщения, основываясь на информации, возвращенной при пробе либо отменить ждущие сообщения.

Проба и отмена

int MPI_Probe(int source, int tag, MPI_Comm comm, MPI_Status *status)

int MPI_Iprobe(int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status)

int MPI_Cancel(MPI_Request *request)

Совмещенный прием/передача

Операция send-receive комбинирует в одном обращении посылку сообщения одному получателю и прием сообщения от другого отправителя.

Эта операция весьма полезна для выполнения сдвига по цепи процессов.

Совмещенный прием/передача

```
MPI_SENDRECV(sendbuf, sendcount, sendtype,
 dest, sendtag,recvbuf, recvcount, recvtype,
 source, recvtag, comm, status)

IN   sendbuf   адрес буфера отправителя
IN   sendcount    число элементов в буфере
IN   sendtype  тип элементов в буфере
IN   dest    номер процесса-получателя
IN   sendtag   тэг процесса-отправителя
OUT  recvbuf   адрес приемного буфера
IN   recvcount    число элементов в буфере
IN   recvtype  тип элементов в буфере
IN   source номер процесса-отправителя
IN   recvtag   тэг процесса-получателя
IN   comm    коммуникатор
OUT  status  статус
```

Совмещенный прием/передача

```
MPI_SENDRECV_REPLACE(buf, count, datatype,
 dest, sendtag, source, recvtag, comm, status)

INOUT  buf      адрес буфера отправителя и получателя
IN     count    число элементов в буфере отправителя
и получателя
IN     datatype тип элементов в буфере отправителя
и получателя
IN     dest     номер процесса-получателя
IN     sendtag    тэг процесса-отправителя
IN     source   номер процесса-отправителя
IN     recvtag    тэг процесса-получателя
IN     comm     коммуникатор (дескриптор)
OUT    status   статус (статус)
```

NULL процессы

Во многих случаях удобно описать «фиктивного» отправителя или получателя для коммуникаций. Это упрощает код, который необходим для работы с границами, например, в случае нециклического сдвига, выполненного по вызову send-receive.

Вместо номера процесса может быть использовано специальное значение MPI_PROC_NULL