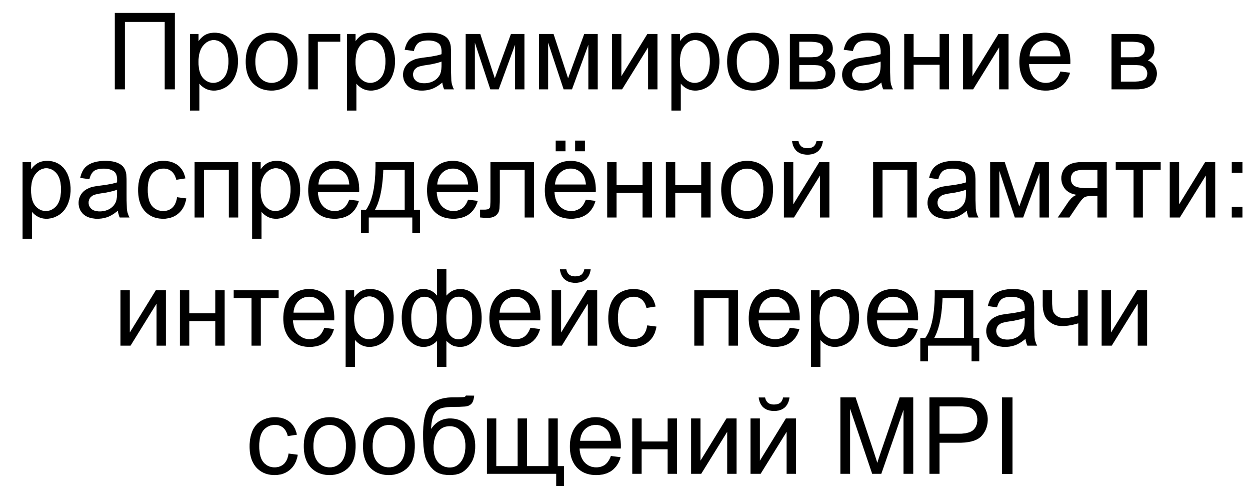


Востокин Сергей Владимирович

План

- Общая характеристика MPI
- Базовые функции, простейшая MPI-программа
- Обзор коммуникационных операций типа точка-точка
- Блокирующие коммуникационные операции точка-точка
- Неблокирующие коммуникационные операции точка-точка

Общая характеристика MPI

История разработки, документация

- **Message Passing Interface Forum**
  - ***MPI-3.1*** 04.06.2015 последняя версия
  - ***MPI-3.0*** 21.09.2012 предпоследняя версия
  - ***MPI-1.1*** 12.10.1998 первая официальная версия
  - Supercomputing 1994 (Ноябрь 1994)
- http://www.mpi-forum.org/ сайт с документацией стандарта

Назначение

- Стандартизированная
- Переносимая
- Высокопроизводительная
- Библиотека функций, использующих передачу сообщений
- Для языков программирования Fortran, C

Модель программирования

- SPMD-модель для распределённой памяти
- Некоторые элементы разделяемой памяти введены в версии MPI-2
- Синхронизация/коммуникация процессов реализуется посредством операций точка-точка и групповых операций

Концепции модели

- Коммуникаторы
- Двухсторонние коммуникации (типа точка-точка)
- Коллективные коммуникации
- Производные типы данных
- Односторонние коммуникации (MPI-2)
- Динамические процессы (MPI-2)
- Параллельный ввод-вывод (MPI-2)

Группы функций MPI

- 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

Всего в базовой версии MPI-1 — около 130 функций

Способ выполнения функций

- Локальная функция
- Нелокальная функция
- Глобальная функция
- Блокирующая функция
- Неблокирующая функция

Привязка к языку реализации (1/2)

- Таблица соответствия типов языка С и MPI

Тип MPI	**Тип языка С**
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	

Привязка к языку реализации (2/2)

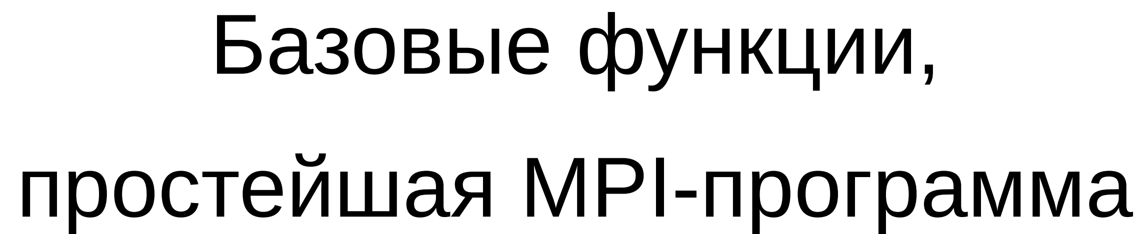
- Соглашение о именовании функций и констант

  префикс ***MPI***_ первая буква – в верхнем регистре

  MPI_Класс_действие_подмножество

  MPI_Класс_действие

- Для некоторых действий введены стандартные наименования: ***create, get, set, delete, is***
- Имена констант MPI – в верхнем регистре
- #include <mpi.h>


Базовые функции,
простейшая MPI-программа

Инициализация / завершение

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

  каждому процессу при инициализации передаются аргументы функции main, полученные из командной строки

- ***int MPI_Finalize(void);***

  закрывает все MPI-процессы и ликвидирует все области связи

Области связи по умолчанию

- ***MPI_COMM_WORLD***

  эта область связи объединяет все процессы-приложения

- ***MPI_COMM_SELF***

  область связи для каждого отдельного процесса

Параметры области связи

- ***int MPI_Comm_size(MPI_Comm comm, int *size);***

  IN comm - коммуникатор; OUT size - число процессов в области связи коммуникатора comm

- ***int MPI_Comm_rank(MPI_Comm comm, int *rank);***

  IN comm - коммуникатор; OUT rank - номер процесса, вызвавшего функцию

Передача сообщения: MPI_Send

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

  IN buf - адрес начала расположения пересылаемых данных; IN count - число пересылаемых элементов; IN datatype - тип посылаемых элементов; IN dest - номер процесса-получателя в группе, связанной с коммуникатором comm; IN tag - идентификатор сообщения; IN comm - коммуникатор области связи

Приём сообщения: MPI_Recv

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

  OUT buf - адрес начала расположения принимаемого сообщения; IN count - максимальное число принимаемых элементов; IN datatype - тип элементов принимаемого сообщения; IN source - номер процесса-отправителя; IN tag - идентификатор сообщения; IN comm - коммуникатор области связи; OUT status - атрибуты принятого сообщения

«Джокеры» для приёма сообщений

- ***MPI_ANY_SOURCE***

  приём сообщений от любого процесса в заданной области связи

- ***MPI_ANY_TAG***

  приём сообщения с любым тэгом

Измерение времени

- ***double MPI_Wtime(void);***

  возвращает астрономическое время в секундах, прошедшее с некоторого момента в прошлом

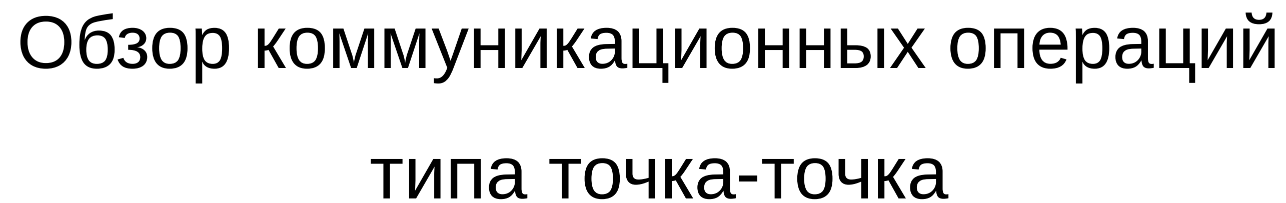
- ***double MPI_Wtick(void);***

  возвращает разрешение таймера (минимальное значение кванта времени)

Простейшая программа

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

int main(int argc,char *argv[]){
	int myid, numprocs;
	MPI_Init (&argc, &argv) ;
	MPI_Comm_size (MPI_COMM_WORLD, &numprocs) ;
	MPI_Comm_rank (MPI_COMM_WORLD, &myid) ;
	fprintf (stdout, "Process %d of %d\n", myid, numprocs);
	MPI_Finalize() ;
	return 0;
}
```


Обзор коммуникационных операций
типа точка-точка

Функции типа точка-точка

Режимы выполнения	**С блокировкой**	**Без блокировки**
Стандартная посылка	MPI_Send	MPI_Isend
Синхронная посылка	MPI_Ssend	MPI_Issend
Буферизованная посылка	MPI_Bsend	MPI_Ibsend
Согласованная посылка	MPI_Rsend	MPI_Irsend
Прием информации	MPI_Recv	MPI_Irecv

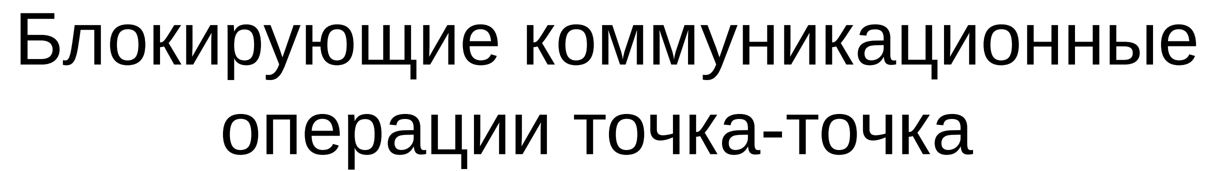
Пояснения к префиксам функций

- ***S (synchronous)*** – синхронный режим передачи данных. Операция передачи данных заканчивается только тогда, когда заканчивается прием данных. Функция нелокальная
- ***B (buffered)*** – буферизованный режим передачи данных. В адресном пространстве передающего процесса с помощью специальной функции создается буфер обмена. Операция посылки заканчивается, когда данные помещены в этот буфер. Функция локальная
- ***R (ready)*** – согласованный или подготовленный режим передачи данных. Операция передачи данных начинается только тогда, когда принимающий процессор инициировал операцию приема. Функция нелокальная
- ***I (immediate)*** – относится к неблокирующим операциям

Почему так много функций?

Так как в реальных приложениях необходимо проверять

- Получено сообщение или только отправлено и находится на доставке
- Можно ли повторно использовать буфер после возврата управления из функции
- Можно ли выполнять пересылку в нескольких функциях одновременно и выполнять вычисления по алгоритму
- Не возникает ли переполнение приёмного буфера, так как получатель не успевает вовремя извлекать сообщения


Блокирующие коммуникационные
операции точка-точка

Порядок выполнения обмена в стандартном режиме

- 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 процесс 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

Проверка наличия сообщения

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

IN source - номер процесса-отправителя; IN tag - идентификатор сообщения; IN comm - коммуникатор; OUT status - атрибуты опрошенного сообщения

Определения числа фактически полученных элементов сообщения

- ***int MPI_Get_count (MPI_Status *status, MPI_Datatype datatype, int *count);***

  IN status - атрибуты принятого сообщения; IN datatype - тип элементов принятого сообщения; OUT count - число полученных элементов

Взаимный обмен данными между процессами (1/2)

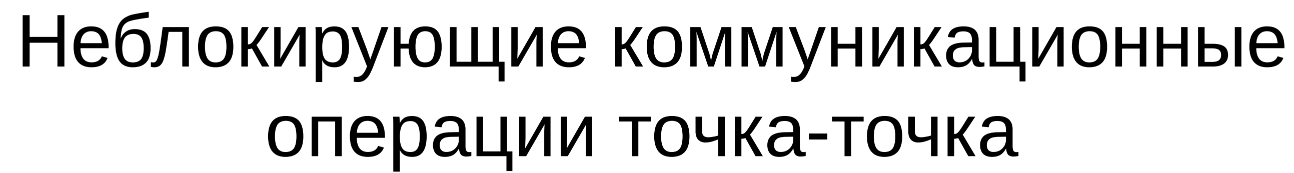
- ***int MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int source, MPI_Datatypea recvtag, MPI_Comm comm, MPI_Status *status);***

  IN sendbuf - адрес начала расположения посылаемого сообщения; IN sendcount - число посылаемых элементов; IN sendtype - тип посылаемых элементов; IN dest - номер процесса-получателя; IN sendtag - идентификатор посылаемого сообщения; OUT recvbuf - адрес начала расположения принимаемого сообщения; IN recvcount - максимальное число принимаемых элементов; IN recvtype - тип элементов принимаемого сообщения; IN source - номер процесса-отправителя; IN recvtag - идентификатор принимаемого сообщения; IN comm - коммуникатор области связи; OUT status - атрибуты принятого сообщения

Взаимный обмен данными между процессами (2/2)

- ***MPI_Sendrecv_replace(void* buf, int count, MPI_Datatype datatype,int dest, int sendtag, int source, int recvtag,MPI_Comm comm, MPI_Status *status);***

  INOUT buf - адрес начала расположения посылаемого и принимаемого сообщения; IN count - число передаваемых элементов; IN datatype - тип передаваемых элементов; IN dest - номер процесса-получателя; IN sendtag - идентификатор посылаемого сообщения; IN source - номер процесса-отправителя; IN recvtag - идентификатор принимаемого сообщения; IN comm - коммуникатор области связи; OUT status - атрибуты принятого сообщения


Неблокирующие коммуникационные
операции точка-точка

Передача сообщения без блокировки MPI_Isend

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

  IN buf - адрес начала расположения передаваемых данных; IN count - число посылаемых элементов; IN datatype - тип посылаемых элементов; IN dest - номер процесса-получателя; IN tag - идентификатор сообщения; IN comm - коммуникатор; OUT request - "запрос обмена"

Прием сообщения без блокировки MPI_Irecv

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

  OUT buf - адрес для принимаемых данных; IN count - максимальное число принимаемых элементов; IN datatype - тип элементов принимаемого сообщения; IN source - номер процесса-отправителя; IN tag - идентификатор сообщения; IN comm - коммуникатор; OUT request - "запрос обмена"

Чтение параметров полученного сообщения MPI_Iprobe

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

  IN source - номер процесса-отправителя; IN tag - идентификатор сообщения; IN comm - коммуникатор; OUT flag - признак завершенности операции; OUT status - атрибуты опрошенного сообщения

Ожидание завершения неблокирующей операции MPI_Wait

- ***int MPI_Wait(MPI_Request *request, MPI_Status *status);***

  INOUT request - "запрос обмена"; OUT status - атрибуты сообщения

Снятие / перывание запроса без ожидания завершения неблокирующей операции

- ***int MPI_Request_free(MPI_Request *request);***
- ***int MPI_Cancel(MPI_Request *request);***

INOUT request - "запрос обмена"

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

Также имеются функции пакетных запросов на коммуникационные операции, инициализируемые ***MPI_Send_init*** и ***MPI_Recv_init;*** которые запускаются функциями ***MPI_Start*** или ***MPI_Startall***

Шаблон MPI-программы (1/4)

```
/*
   "Hello World" MPI Test Program
*/
#include <mpi.h>
#include <stdio.h>
#include <string.h>

#define BUFSIZE 128
#define TAG 0

int main(int argc, char *argv[])
{
    char idstr[32];
    char buff[BUFSIZE];
    int numprocs;
    int myid;
    int i;
    MPI_Status stat;
```

Шаблон MPI-программы (2/4)

```
/* MPI programs start with MPI_Init; all 'N' processes exist thereafter */
MPI_Init(&argc,&argv);
/* find out how big the SPMD world is */
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
/* and this processes' rank is */
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
```

Шаблон MPI-программы (3/4)

```
/* At this point, all programs are running equivalently, the rank
   distinguishes the roles of the programs in the SPMD model, with
   rank 0 often used specially... */
if(myid == 0){
    printf("%d: We have %d processors\n", myid, numprocs);
    for(i=1;i<numprocs;i++){
        sprintf(buff, "Hello %d! ", i);
        MPI_Send(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD);
    }
    for(i=1;i<numprocs;i++){
        MPI_Recv(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD, &stat);
        printf("%d: %s\n", myid, buff);
    }
}
```

Шаблон MPI-программы (4/4)

```
    else{
        /* receive from rank 0: */
        MPI_Recv(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD, &stat);
        sprintf(idstr, "Processor %d ", myid);
        strncat(buff, idstr, BUFSIZE-1);
        strncat(buff, "reporting for duty\n", BUFSIZE-1);
        /* send to rank 0: */
        MPI_Send(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD);
    }
    /* MPI programs end with MPI Finalize; this is a weak synchronization point */
    MPI_Finalize();
    return 0;
}
```