MPI

Message Passing Interface

Способы организации передачи сообщений

```
Process S:
    P = 10;
s1: send P to R;
s2: P = 20;

Process R:
    Q = 0;
r1: receive Q from S;
r2: T = Q + 1;
```

- *Синхронный* (*synchronous*) – завершение send/receive после окончания передачи (*T*==11)
- *Блокирующий* (*blocking*) – send завершается после копирования сообщения в промежуточный буфер, receive ждет окончания приема (T==11)
- *Неблокирующий* (*nonblocking*) – send /receive только инициируют начало операции передачи (*T*==1,11,21)

MPI - это

- Стандарт

  MPI 1.0 1995 год

  MPI 2.0 1998 год.

- Библиотека

  C, Fortran.

  #include <mpi.h>

- MPI Forum

Возможности MPI-1.2 и MPI-2.0

MPI-1.2

- Коммуникация точка-точка
- Коллективная коммуникация
- Коммуникаторы
- Топологии процессов
- Пользовательские типы данных
- Интерфейс профилирования

MPI-2.0

- Динамическое создание процессов
- Параллельный ввод-вывод

Основы MPI

1. **Операции передачи сообщений**
   MPI включает 129 функций. Минимально-необходимый набор – 6 функций.

2. **Тип данных**
   Необходимо указывать тип передаваемых данных. В MPI имеется набор стандартных типов (MPI_INT, MPI_DOUBLE,..), возможным является конструирование новых типов данных.

3. **Коммуникатор**

Понятие коммуникатора

Коммуникатор – контекст обмена группы.

- **MPI_COMM_WORLD** – коммуникатор для всех процессов приложения.
- **MPI_COMM_NULL** – значение, используемое для ошибочного коммуникатора.
- **MPI_COMM_SELF** – коммуникатор, включающий только вызвавший процесс.

Понятие коммуникатора (2)

- На одном компьютере может быть запущено несколько копий (*процессов*) исходной программы.
- Все процессы могут быть распределены на группы (*коммуникаторы*).
- В составе коммуникатора процесс идентифицируется уникальным номером (рангом).
- Каждый процесс может одновременно входить в разные коммуникаторы.
- Два основных атрибута процесса: коммуникатор (группа) и номер процесса в коммуникаторе (группе).
- Если группа содержит n процессов, то номер любого процесса в данной группе лежит в пределах от 0 до n – 1.

Понятие сообщения

- *Сообщение* — набор данных некоторого типа.
- Атрибуты сообщения: номер процесса-отправителя, номер процесса-получателя, идентификатор сообщения и др.
- Идентификатор сообщения - целое неотрицательное число в диапазоне от 0 до 32767.
- Для работы с атрибутами сообщений введена структура **MPI_Status**.

Сборка MPI-приложения.

- Сборка MPI-приложения осуществляется с помощью специальной утилиты. В случае Си – **mpicc**. Пример:

```
mpicc –o mpihello mpihello.c
```

- Запуск MPI-приложения осуществляется с помощью команды **mpirun.**

```
mpirun –np 4 mpihello
```

Функции MPI

В состав библиотеки входит 129 функций.

MPI_Init	Инициализация MPI
MPI_Comm_size	Определение числа процессов
MPI_Comm_rank	Определение процессом собственного номера
MPI_Send	Посылка сообщения
MPI_Recv	Получение сообщения
MPI_Finalize	Завершение программы MPI

Функция MPI_Init()

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

Вход	argc, argv:	аргументы функции main()
Результат:		MPI_SUCCESS или код ошибки

Назначение: Инициализация MPI.

Функция должна быть вызвана каждым процессом MPI до использования им любой другой функции MPI.

В каждом выполнении программы может выполняться только один вызов MPI_Init()

Функция MPI_Finalize()

int MPI_Finalize()

Назначение: Завершение программы MPI.

Предусловие: Все незаконченные коммуникации должны быть завершены.
Функция должна быть вызвана каждым процессом MPI до завершения работы процесса.
После вызова MPI_Finalize() не может быть вызовов других функций MPI.

Структура программы

```
#include “mpi.h”
main(int argc, char *argv[]){
...
  MPI_Init(&argc, &argv);
  /* место, где допустим вызов MPI-функций
  */
  MPI_Finalize();
...
}
```

Функция MPI_Comm_size()

int MPI_Comm_size(MPI_Comm comm, int *size)

Вход comm: Коммуникатор
Выход size: Число процессов в коммуникаторе

Назначение: Определение числа процессов в коммуникаторе.

MPI_COMM_WORLD – предопределённый стандартный коммуникатор. Его группа процессов включает все процессы параллельного приложения

Функция MPI_Comm_rank()

int MPI_Comm_rank(MPI_Comm comm, int *rank)

Вход comm: Коммуникатор
Выход size: Номер процесса, запустившего функцию

Назначение: Определение процессом своего номера.

Единственный способ различения процессов.
Номер процесса – уникальный идентификатор внутри группы процессов коммуникатора.

Функция MPI_Send()

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

Вход	buf:	Начальный адрес буфера посылки сообщения
Вход	count:	Число передаваемых элементов в сообщении
Вход	dtype:	Тип передаваемых элементов
Вход	dest:	Номер процесса-получателя
Вход	tag:	Идентификатор сообщения
Вход	comm:	Коммуникатор

Назначение: Посылка сообщения.

Блокирующая операция

Функция MPI_Recv()

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

Выход	buf:	Начальный адрес буфера процесса-получателя
Вход	count:	Число элементов в принимаемом сообщении
Вход	dtype:	Тип элементов сообщения
Вход	source:	Номер процесса-отправителя
Вход	tag:	Идентификатор сообщения
Вход	comm:	Коммуникатор
Выход	status:	Статусная информация

Назначение: Посылка сообщения.
Блокирующая операция
Сообщение должно быть меньше чем размер буфера.

Типы данных MPI

MPI	C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	unsigned 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	1 байт
MPI_PACKED	Упакованные данные

Функция MPI_Wtime()

double MPI_Wtime()

Назначение: Оценка времени выполнения

- Функция MPI_Wtime() дает время в секундах в виде числа двойной точности.
- Выдает число секунд между двумя ближайшими последовательными “тиками” часов

Режимы передачи сообщений…

- **Синхронный (synchronous)** – send не возвращает управление, пока не начат receive
- **Буферизуемый (buffered) -** send возвращает управление после копирования сообщения в буфер передачи (работа с буфером MPI_Buffer_attatch, MPI_Buffer_detatch)
- **Стандартный (standard)** – режим по умолчанию (синхронный или буферизуемый)
- **По готовности (Ready)** – может применяться для передачи данных при гарантированности, что операция приема уже активна (иначе send выдаст сообщение об ошибке)

Режимы передачи сообщений…

- Для операции передачи возможны 8 различных вариантов
  - 4 режима передачи
  - 2 способа блокировки
- Для операции приема возможны 2 различных варианта
  - 1 режим передачи (standard)
  - 2 способа блокировки

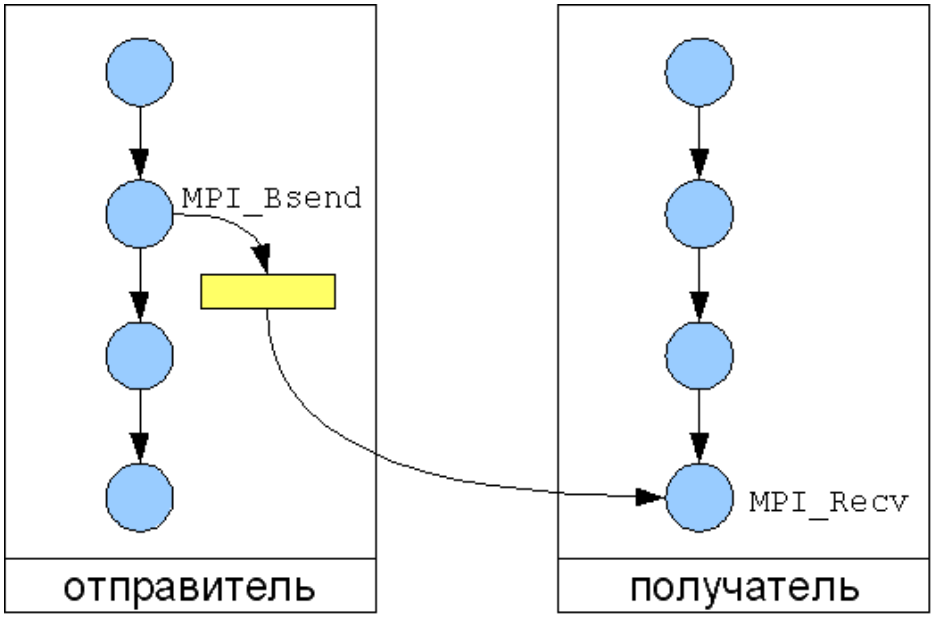
Режимы передачи сообщений…

Режим передачи	Блокирующий	Не блокирующий
Standard	MPI_Send	MPI_Isend
Synchronous	MPI_Ssend	MPI_Issend
Buffered	MPI_Bsend	MPI_Ibsend
Ready	MPI_Rsend	MPI_Irsend

Режим приема	Блокирующий	Не блокирующий
Standard	MPI_Recv	MPI_Irecv

Буферизованная пересылка

- Процесс-**отправитель** выделяет буфер и регистрирует его в системе.
- Функция MPI_Bsend помещает данные выделенный буфер, .



Буферизованная пересылка

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

- Завершается после копирования данных из буфера buf в буфер для отсылаемых сообщений, выделенный программой.
- Если места в буфере недостаточно, то возвращается ошибка.

Функции работы с буфером обмена

```
int MPI_Buffer_attach( buffer, size )
void *buffer; /* in */
int  size;    /* in */
```

buffer - адрес начала буфера
size - размер буфера в байтах

```
int MPI_Buffer_detach( bufferptr, size )
void *bufferptr; /* out */
int  *size;      /* out */
```

*bufferptr - адрес высвобожденного буфера
*size - размер высвобожденного пространства

функция **MPI_Buffer_detach** блокирует процесс до тех пор, пока все данные не отправлены из буфера

Вычисление размера буфера

```
int MPI_Pack_size(int incount, MPI_Datatype
datatype, MPI_Comm comm, int *size)
```

Вычисляет размер памяти для хранения одного сообщения.

`MPI_BSEND_OVERHEAD` – дополнительный объем для хранения служебной информации (организация списка сообщений).

Размер буфера для хранения n одинаковых сообщений вычисляется по формуле:

```
n x (размер_одного_сообщения + MPI_BSEND_OVERHEAD)
```

Порядок организации буферизованных пересылок

- Вычислить необходимый объем буфера (MPI_Pack_size).
- Выделить память под буфер (malloc).
- Зарегистрировать буфер в системе (MPI_Buffer_attach).
- Выполнить пересылки.
- Отменить регистрацию буфера (MPI_Buffer_dettach)
- Освободить память, выделенную под буфер (free).

Особенности работы с буфером

•Буфер всегда один.

• Для изменения размера буфера сначала следует отменить регистрацию, затем увеличить размер буфера и снова его зарегистрировать.

•Освобождать буфер следует только после того, как отменена регистрация.

Пример буферизованной пересылки

```
MPI_Pack_size(1, MPI_INT, MPI_COMM_WORLD,&msize)
blen = M * (msize + MPI_BSEND_OVERHEAD);
buf = (int*) malloc(blen);
MPI_Buffer_attach(buf, blen);
for(i = 0; i < M; i ++) {
  n = i;
  MPI_Bsend(&n, 1, MPI_INT, 1, i, MPI_COMM_WORLD);
}
MPI_Buffer_detach(&abuf, &ablen);
free(abuf);
```

Коллективные операции…

- **Синхронизация вычислений** (согласование времени нахождения исполняемых процессов в необходимых точках параллельного метода решения задачи)

int MPI_Barrier(MPI_Comm comm);

! Вызов MPI_Barrier должен осуществляться в каждом процессе

```
Process S:
  ...
  MPI_Barrier(comm);
  ...
```

```
Process Q:
  ...
  MPI_Barrier(comm);
  ...
```

Коллективные операции…

- **Рассылка сообщений всем процессам**

  int MPI_Bcast(&Buf, Count, Type, Root, &Comm);

  Root – ранг процесса, источника рассылки

  ! Вызов MPI_Bcast должен осуществляться в каждом процессе

```
Process S:
 …
 MPI_Bcast(&n,1,MPI_INT,0, &Comm);
 …
Process Q:
 …
 MPI_Bcast(&n,1,MPI_INT,0, &Comm);
 …
```

Коллективные операции…

- **Прием сообщения от всех процессов с выполнением стандартной операции обработки**

```
int MPI_Reduce(&Sbuf, &Rbuf, Count, DataType,
Operation, Root, &Comm);
```

  - Sbuf – буфер отправляемого сообщения,
  - Rbuf – буфер для приема сообщения,
  - Root – ранг процесса для приема сообщения,
  - Operation – операция обработки (MPI_SUM,…)

! Вызов MPI_Reduce должен осуществляться в каждом процессе

Коллективные операции…

Операции функции MPI_Reduce

- **MPI_MAX** и **MPI_MIN** ищут поэлементные максимум и минимум;
- **MPI_SUM** вычисляет сумму векторов;
- **MPI_PROD** вычисляет поэлементное произведение векторов;
- **MPI_LAND**, **MPI_BAND**, **MPI_LOR**, **MPI_BOR**, **MPI_LXOR**, **MPI_BXOR** - логические и двоичные операции И, ИЛИ, исключающее ИЛИ;
- **MPI_MAXLOC**, **MPI_MINLOC** - поиск индексированного минимума/максимума

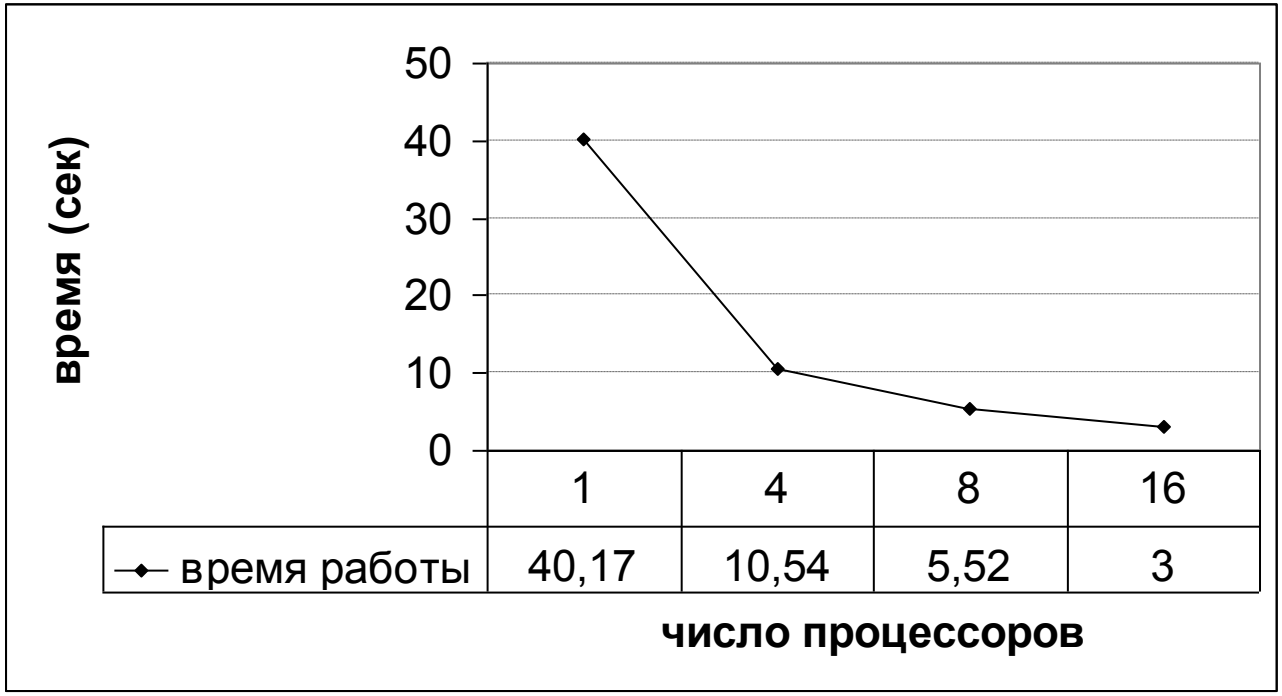
Вычисление числа π с помощью MPI

```
#include <stdio.h>
#include <math.h>
// Подключение заголовочного файла MPI
#include "mpi.h"

double f(double a)
{
    return (4.0 / (1.0+ a*a));
}

 int main(int argc, char **argv) {

int done = 0, n, myid, numprocs, i;
double PI25DT = 3.14159265358979323846264643;
double mypi, pi, h, sum, x;
 double startwtime = 0.0, endwtime;
int namelen;
char processor_name[MPI_MAX_PROCESSOR_NAME];
```

```
// Инициализация подсистемы MPI
MPI_Init(&argc, &argv);
// Получить размер коммуникатора MPI_COMM_WORLD
// (общее число процессов в рамках задачи)
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
// Получить номер текущего процесса в рамках
 // коммуникатора MPI_COMM_WORLD
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
MPI_Get_processor_name(processor_name,&namelen);
// Вывод номера потока в общем пуле
fprintf(stdout, "Process %d of %d is on %s\n",
myid,numprocs,processor_name);
fflush(stdout);

while(!done){
 // количество интервалов
    if(myid==0) {
         fprintf(stdout, "Enter the number of intervals: (0 quits) ");
        fflush(stdout);
        if(scanf("%d",&n) != 1)
                { fprintf(stdout, "No number entered; quitting\n");
                 n = 0; }
         startwtime = MPI_Wtime();
         }
```

```
// Рассылка количества интервалов всем процессам (в том числе и себе)
      MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
      if(n==0) done = 1;
      else {
          h = 1.0 / (double) n;
          sum = 0.0;
          // Обсчитывание точки, закрепленной за процессом
          for(i = myid + 1 ; (i <= n) ; i += numprocs){
        x = h * ((double)i - 0.5);
                 sum += f(x);}
          mypi = h * sum;
           // Сброс результатов со всех процессов и сложение
     MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
           // Если это главный процесс, вывод полученного результата
          if(myid==0)
          {  printf("PI is%.16f, Error is %.16f\n",pi,fabs(pi-PI25DT));
              endwtime = MPI_Wtime();
              printf("wall clock time = %f\n", endwtime-startwtime);
              fflush(stdout);
  }}}
  // Освобождение подсистемы MPI
  MPI_Finalize();
return 0; }
```

Результаты вычислительного эксперимента



Данные получены на MVS-15000BM.

Проверка завершения операций…

- **Блокирующая проверка**

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

- при завершении MPI_Wait
  - для MPI_Isend сообщение переслано или скопировано в буфер,
  - для MPI_Irecv сообщение скопировано в буфер процесса
  - request == MPI_REQUST_NULL

Проверка завершения операций

- **Неблокирующая проверка**

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

при завершении MPI_Test для незавершенной операции пересылки flag == 0

УПАКОВКА СООБЩЕНИЙ

- дает возможность пересылать разнородные данные в одном сообщении;
- отделяет операцию формирования сообщения от операции пересылки;
- способствует развитию библиотек на базе MPI;

MPI_PACK

```
int MPI_Pack(void* inbuf, int incount, MPI_Datatype
datatype, void *outbuf, int outcount, int *position,,
MPI_Comm comm)
```

inbuf – буфер с данными для запаковки;
incount – число элементов для запаковки;
datatype – тип элементов данных;
outbuf – буфер сообщения;
outsize – размер буфера сообщения;
position – позиция в буфере сообщения, с которой заполнять буфер (изменяется);
comm – коммуникатор, по которому сообщение будет посылаться;

MPI_UNPACK

```
int MPI_Unpack(void* inbuf, int insize, int *position, void
*outbuf, int outcount, MPI_Datatype datatype, MPI_Comm
comm)
```

inbuf – буфер сообщения;
incount – число элементов данных для распаковки
position – позиция, с которой распаковывать данные (изменяется);
outbuf – буфер для распаковки;
outcount – число элементов для распаковки;
datatype – тип элементов данных;
comm – коммуникатор, по которому сообщение будет посылаться;

ОПРЕДЕЛЕНИЕ РАЗМЕРА СООБЩЕНИЯ

```
int MPI_Pack_size(int incount, MPI_Datatype
datatype, MPI_Comm comm, int *size)
```

incount – число элементов данных в сообщении;

datatype – тип элементов данных;

comm – коммуникатор;

size – размер сообщения;

ПРИМЕР

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

#define N 3

main(int argc, char* argv[])
{
  int r;
  int i;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &r);
```

ПРИМЕР (2)

```
if(r == 0){
    int sz;
    int pos = 0;
    int a = 1;
    void* buf;

    MPI_Pack_size(N, MPI_INT, MPI_COMM_WORLD, &sz);
    buf = (void*) malloc(sz);
    for(i = 0; i < N; i ++) {
      MPI_Pack(&a, 1, MPI_INT, buf, sz, &pos, MPI_COMM_WORLD);
      a ++;
    }

    MPI_Send(buf, pos, MPI_PACKED, 1, 0, MPI_COMM_WORLD);
  }
```

ПРИМЕР (3)

```
else {
    MPI_Status st;
    int A[N];

    MPI_Recv(A, N, MPI_INT, 0, 0, MPI_COMM_WORLD, &st);
    for(i = 0; i < N; i ++)
      printf("%d ", A[i]);

  }

  MPI_Finalize();
}
```

Литература

- MPI home page : www.mcs.anl.gov/mpi
- "MPI: The Complete Reference" by Snir, Otto, Huss-Lederman, Walker, and Dongarra, MIT Press (also in Postscript and html)
- "Using MPI" by Gropp, Lusk and Skjellum, MIT Press