

Е. М. Карчевский

## КРАТКОЕ ВВЕДЕНИЕ В МРІ

Презентация выступления на семинаре «Суперкомпьютерное  
моделрование» (рук. профессор Н. Б. Плещинский)

Казань, 7 октября 2013 г.

## MPI

- Message Passing Interface (среда передачи сообщений)

## ЛИТЕРАТУРА

- 1) Воеводин В.В. Математические модели и методы в параллельных процессах. Москва, 1986.
- 2) Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. Санкт–Петербург, 2002.
- 3) Малышкин В.Э., Корнеев В.Д. Параллельное программирование мультикомпьютеров. Новосибирск, 2006.

- 4) Корнеев В.Д. Параллельное программирование в МРІ. Новосибирск, 2002.
- 5) Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте МРІ. Минск, 2002.
- 6) Гришагин В.А., Свистунов А.Н. Параллельное программирование на основе МРІ. Нижний Новгород, 2005.
- 7) Антонов С.А. Параллельное программирование с использованием технологии МРІ. Изд-во МГУ, 2004.

- .
- 8) Керниган Б., Ритчи Д. Язык программирования С. Изд-во «Вильямс», 2009.
  - 9) Дейтел Х.М., Дейтел Дж.П. Как программировать на С++. Изд-во «Бином», 2008.

# ФУНКЦИИ ПЕРЕДАЧИ СООБЩЕНИЙ (НЕ МРІ)

•  
send(address, length, destination, tag)

- address — адрес начала буфера с посылаемыми данными
- length — длина посылаемых данных в байтах
- destination — номер процесса, которому посылается сообщение
- tag — номер типа сообщения

•

recv(address, maxlen, source, tag, datlen)

- `address` — адрес начала буфера с принимаемыми данными
- `maxlen` — длина буфера с принимаемыми данными в байтах
- `source` — номер процесса, от которого пришло сообщение
- `tag` — получаем сообщения только определенного типа
- `datlen` — число принятых байтов



•

## Недостатки (address, length)

- Сообщение может не быть непрерывным (напр., строка матрицы, хранящейся в столбцах)
- Неоднородные вычислительные комплексы (разные форматы чисел в разных машинах)

•

Буфер сообщения в MPI (address, count, datatype)

count элементов данных

типа datatype,

начинающихся с address

(A, 10, MPI\_REAL) — вектор A из 10 вещественных чисел

## Базовые функции MPI

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

# Первая программа

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
    int rank, numprocs, i, message;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("\n Hello from process %3d", rank);
    MPI_Finalize();
    return 0;
}
```

•  
  
#include <stdio.h>

- Включение файла `stdio.h` стандартной библиотеки ввода вывода

•  
  
#include "mpi.h"

- Включение файла библиотеки mpi.h
- Кавычки в "mpi.h" указывают, что поиск файла mpi.h начинается с текущего каталога

•  
int main(int argc, char \*argv[])

- Программа начинает выполняться с начала функции main
- int — целое число
- char — символ (1 байт)
- argc — количество аргументов командной строки (от argument count)
- argv[] — указатель на массив символьных строк, содержащих сами аргументы (от argument vector)
- [] — размер массива явно не указан



•  
  
int rank, numprocs, i, message;

- Определение целых переменных

•  
  
MPI\_Init(&argc, &argv);

- Устанавливает среду (environment) MPI
  - argc — количество аргументов командной строки процесса
  - argv — вектор указателей на эти аргументы
- & — операция взятия адреса
- \* — операция ссылки по указателю

•  
MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);

MPI\_COMM\_WORLD — предопределенный коммуникатор, определяющий единый контекст и начальную группу всех процессов параллельной программы

MPI\_Comm\_size возвращает в numprocs число запущенных для данной программы процессов

•

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

- Процессы каждой группы пронумерованы целыми числами, начиная с 0, которые называются рангами (rank)
- Каждый процесс определяет свой номер в группе, связанной с данным коммуникатором, с помощью MPI\_Comm\_rank

•  
printf("\n Hello from process %3d", rank);

- Каждый процесс печатает свой номер (ранг)

"\n — переход на новую строку

%3d — вывести аргумент как десятичное целое число в поле шириной не менее трех символов

•

MPI\_Finalize();

- Ликвидация среды MPI

•

Что произойдет в результате выполнения программы?

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
    int rank, numprocs, i, message;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("\n Hello from process %3d", rank);
    MPI_Finalize();
    return 0;
}
```



•

## Возможный вариант ответа

Hello from process 2

Hello from process 0

Hello from process 1

Hello from process 3

•

Система с выделенным нулевым процессом, который должен принимать сообщения от всех остальных процессов.

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

```
#include "mpi.h"
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    int rank, n, i, message;
```

```
    MPI_Status status;
```

```
    MPI_Init(&argc, &argv);
```

```
    MPI_Comm_size(MPI_COMM_WORLD, &n);
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

•  
MPI\_Status status;

Определяет переменную `status` типа `MPI_Status`

`MPI_Status` — структура, содержащая три поля:

`MPI_SOURCE` — источник полученного сообщения,

`MPI_TAG` — тип полученного сообщения,

`MPI_ERROR` — код ошибки полученного сообщения.

```
if (rank == 0){
    printf("\n Hello from process %3d", rank);
    for (i=1; i<n; i++){
        MPI_Recv(&message, 1, MPI_INT, MPI_ANY_SOURCE,
        MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("\n Hello from process %3d", message);
    }
} else
    MPI_Send(&rank,1,MPI_INT,0,0,MPI_COMM_WORLD);
MPI_Finalize();
return 0;
}
```

MPI\_Send(&rank,1,MPI\_INT,0,0,MPI\_COMM\_WORLD);

&rank,1,MPI\_INT — процесс посылает сообщение, состоящее из одного целого числа, находящегося по адресу &rank

0 — номер процесса получателя в группе, определяемой коммуникатором MPI\_COMM\_WORLD

0 — тип сообщения (тег, м. б. любым целым числом от 0 до 32767)

MPI\_Recv(&message, 1, MPI\_INT, MPI\_ANY\_SOURCE,  
MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

&message,1,MPI\_INT — процесс помещает по адресу &message  
принимаемое сообщение, состоящее из одного целого числа

MPI\_ANY\_SOURCE — процесс принимает сообщение от любого  
процесса из группы коммутатора MPI\_COMM\_WORLD

MPI\_ANY\_TAG — процесс принимает сообщение любого типа

status — источник, тип и код ошибки полученного сообщения

•

Что произойдет в результате выполнения программы?



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

```
#include "mpi.h"
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    int rank, n, i, message;
```

```
    MPI_Status status;
```

```
    MPI_Init(&argc, &argv);
```

```
    MPI_Comm_size(MPI_COMM_WORLD, &n);
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
if (rank == 0){
    printf("\n Hello from process %3d", rank);
    for (i=1; i<n; i++){
        MPI_Recv(&message, 1, MPI_INT, MPI_ANY_SOURCE,
        MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("\n Hello from process %3d", message);
    }
} else
    MPI_Send(&rank,1,MPI_INT,0,0,MPI_COMM_WORLD);
MPI_Finalize();
return 0;
}
```

•

## Возможный вариант ответа

Hello from process 0

Hello from process 2

Hello from process 1

Hello from process 3

•

Последовательный прием данных от процессов в порядке возрастания их рангов

## Заменим

```
for (i=1; i<n; i++){  
    MPI_Recv(&message, 1, MPI_INT, MPI_ANY_SOURCE,  
    MPI_ANY_TAG, MPI_COMM_WORLD, &status);  
    printf("\n Hello from process %3d", message);  
}
```

## На

```
for (i=1; i<n; i++){  
    MPI_Recv(&message, 1, MPI_INT, i,  
    MPI_ANY_TAG, MPI_COMM_WORLD, &status);  
    printf("\n Hello from process %3d", message);  
}
```

.

## ОТВЕТ

Hello from process 0

Hello from process 1

Hello from process 2

Hello from process 3

•

Проблема: отправка сообщения происходит с блокировкой, т.е.

управление не возвращается программе до тех пор, пока сообщение не будет принято.

•

Неблокирующий обмен: MPI\_Isend, MPI\_Irecv



•

Передача данных от одного процесса всем и от всех процессов одному. Скалярное произведение векторов.

```
#include "mpi.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    double x[100], y[100];
    double res, p_res = 0.0;
    MPI_Status status;
    int n, myid, numprocs, i, N;
    N=100;
```

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
if (0 == myid)
    read_vectors(x,y);
MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(y, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

•  
`MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);`

- Рассылка данных из буфера `x`, содержащего `N` элементов типа `MPI_DOUBLE` с процесса, имеющего номер `0`, всем процессам, входящим в коммуникатор `MPI_COMM_WORLD`
- На нулевом процессе вызов функции `MPI_Bcast` обеспечивает отправку данных, а на остальных — их приемку
- Данные отправляются из массива `x` и принимаются в ту же область памяти, только на другом процессе

```
for (i = myid *N/numprocs; i < (myid + 1)*N/numprocs; i++)
    p_res = p_res + x[i] * y[i];
MPI_Reduce(&p_res, &res, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
if (0 == myid)
    printf("\n Inner product = %10.2f", res);
MPI_Finalize();
return 0;
}
```

MPI\_Reduce(&p\_res, &res, 1, MPI\_DOUBLE, MPI\_SUM, 0,  
MPI\_COMM\_WORLD);

&p\_res, 1, MPI\_DOUBLE — буфер вывода с данными, которые  
посылаются каждым процессом коммутатора

MPI\_COMM\_WORLD

&res, 1, MPI\_DOUBLE — буфер ввода с данными, которые при-  
нимает только процесс с номером 0

MPI\_SUM — операция редукции, применяемая к посылаемым  
данным (всего 12 операций, напр., MPI\_MAX, MPI\_MIN)