



# ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ

Занятие 1. Технология MPI



*Составитель: Герасимов А.С.*

# Учебный кластер МФТИ

head.vdi.mipt.ru

remote.vdi.mipt.ru:52960

ssh login@head.vdi.mipt.ru

- Узлы: 1 головной (head) и 7 вычислительных
- Узлы идентичны: 4 ядра, 15 ГБ ОЗУ
- Система очередей – Torque/PBS

## Пример PBS-задачи

job.sh

```
#!/bin/bash
```

```
#PBS -l walltime=00:10:00,nodes=7:ppn=1
```

```
#PBS -N job_name
```

```
#PBS -q batch
```

```
uname -n
```

## Запуск задачи

```
qsub job.sh
```

### Выход задачи:

- `<job_name>.o<ID>` – выход stdout
- `<job_name>.e<ID>` – выход stderr

### Ограничения:

- 5 заданий / пользователя
- 10 минут выполнения
- 1 ГБ памяти

## Просмотр текущих задач в очереди

qstat

```
[kolya@head mpi]$ qstat
```

Job id	Name	User	Time Use	S	Queue
-----	-----	-----	-----	-	-----
25.localhost	my_job	kolya	0	R	batch
26.localhost	my_job	kolya	0	R	batch
27.localhost	my_job	kolya	0	R	batch
28.localhost	my_job	kolya	0	R	batch
29.localhost	my_job	kolya	0	R	batch

## Удаление задачи

qdel <ID>

# MPI (Message Passing Interface)

- Библиотека функций, предназначенная для поддержки работы параллельных процессов.
- Базовый механизм связи между процессами – передача и приём сообщений.
- Ориентирован на системы с распределенной памятью
- Состав сообщений:
  - *отправитель — ранг (номер в группе) отправителя;*
  - *получатель — ранг получателя;*
  - *признак(и);*
  - *коммуникатор — код группы процессов.*
- Блокирующие / неблокирующие передачи

# Общие процедуры MPI. Инициализация

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



- MPI\_SUCCESS
- код ошибки



Аргументы функции main()

```
int MPI_Finalize (void)
```

# Общие процедуры MPI. Инициализация

## Основа программы

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

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

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

```
    ...
```

```
    MPI_Finalize();
```

```
}
```



## Общие процедуры MPI. Размер группы

```
int MPI_Comm_size  
    (MPI_Comm comm, int* size)
```



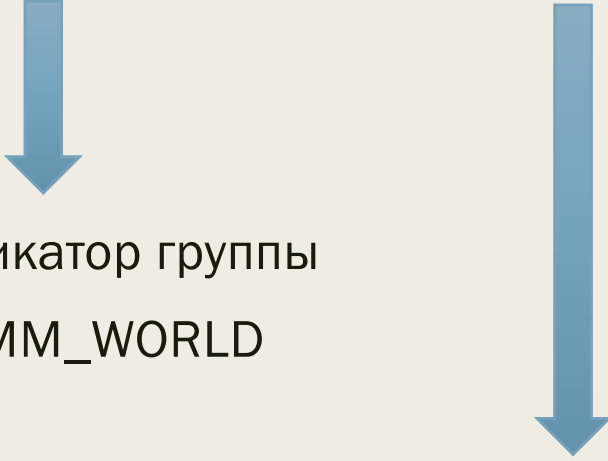
- Коммуникатор группы
- MPI\_COMM\_WORLD



(OUT) Размер группы

## Общие процедуры MPI. Ранг процесса

```
int MPI_Comm_rank  
    (MPI_Comm comm, int* rank)
```

- 
- Коммуникатор группы
  - MPI\_COMM\_WORLD

(OUT) Номер процесса в группе [0; size-1]

# Общие процедуры MPI. Подсчет времени

```
double MPI_Wtime (void)
```



Некоторое время в секундах

# Общие процедуры MPI. Оценка ускорения

## Закон Амдала

$$a \leq \frac{1}{(1 - p) + \frac{p}{n}}$$

$a$  – оценка ускорения

$p$  – распараллеливаемая часть программы (доля общего времени выполнения)

$n$  – количество процессов

### Компиляция программы

```
mpicc superhot.c -o hot
```

### Запуск программы

```
mpirun -np <thread_num> hot
```

job.sh

```
#!/bin/bash
```

```
#PBS -l walltime=00:01:00,nodes=1:ppn=3
```

```
#PBS -N my_job
```

```
#PBS -q batch
```

```
cd $PBS_O_WORKDIR
```

```
mpirun --hostfile $PBS_NODEFILE -np 3 ./hot
```

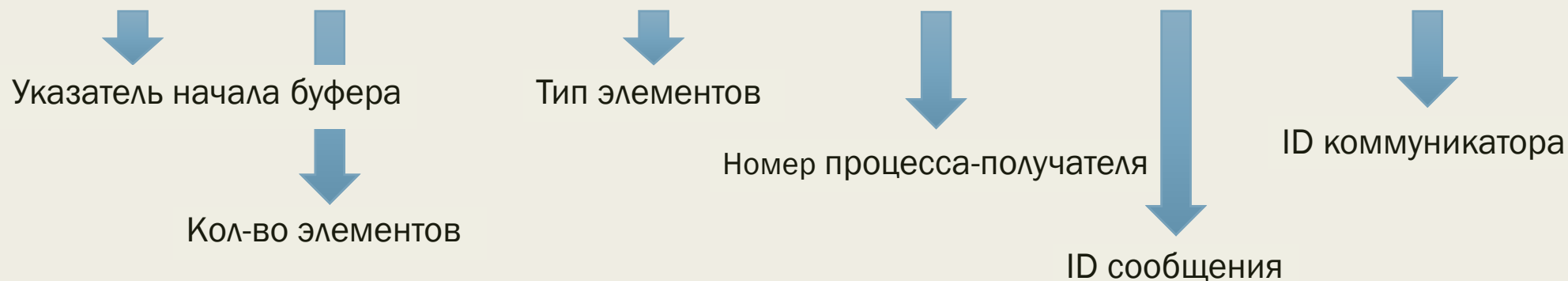
# Общие процедуры MPI. Задача 1

- Составить и запустить программу «Hello, world!»
- Вывести размер своего коммуникатора и своего процесса

# Общие процедуры MPI. Блокирующие передачи

**int MPI\_Send**

(void\* buf, int count, MPI\_Datatype datatype, int dest, int msgtag, MPI\_Comm comm)



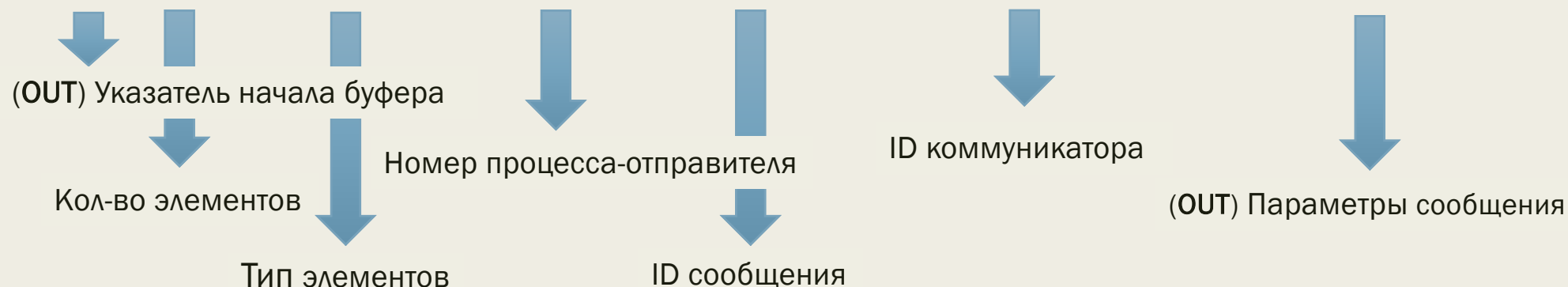
- Все элементы сообщения расположены подряд в буфере
- Значение *count* может быть нулем
- Тип элементов должен указываться с помощью констант типа
- Разрешается передавать сообщение самому себе
- Никаких гарантий передачи/приема



# Общие процедуры MPI. Блокирующие передачи

## int MPI\_Recv

(void\*, int, MPI\_Datatype, int source, int msgtag, MPI\_Comm comm, MPI\_Status \*status)



- Число элементов в сообщении не должно превосходить count
- Если нужно узнать точное число элементов в сообщении → MPI\_Probe
- Гарантия, что после возврата все элементы сообщения приняты и расположены в *buf*
- В качестве *source* можно указать predetermined константу MPI\_ANY\_SOURCE
- В качестве *msgtag* можно указать константу MPI\_ANY\_TAG
- Из двух приходящих сообщений выбирается то, что отправлено раньше

## Общие процедуры MPI. Предопределенные константы

Константа MPI	Тип в C
<i>MPI_CHAR</i>	signed char
<i>MPI_SHORT</i>	signed int
<i>MPI_INT</i>	signed int
<i>MPI_LONG</i>	signed long int
<i>MPI_UNSIGNED_CHAR</i>	unsigned char
<i>MPI_UNSIGNED_SHORT</i>	unsigned int
<i>MPI_UNSIGNED</i>	unsigned int
<i>MPI_UNSIGNED_LONG</i>	unsigned long int
<i>MPI_FLOAT</i>	float
<i>MPI_DOUBLE</i>	double
<i>MPI_LONG_DOUBLE</i>	long doubl

# Общие процедуры MPI. Предопределенные константы

## MPI\_Status - атрибуты сообщений

MPI\_Source (номер процесса-отправителя)  
MPI\_Tag (ID сообщения)  
MPI\_Error (код ошибки)

## Константы-пустышки

MPI\_COMM\_NULL  
MPI\_DATATYPE\_NULL  
MPI\_REQUEST\_NULL

## Код успешного завершения процедуры

MPI\_SUCCESS

# Общие процедуры MPI. Асинхронные передачи

```
int MPI_Isend
```

```
(void*, int, MPI_Datatype, int, int, MPI_Comm, MPI_Request *request)
```



(OUT) ID операции

- Возврат происходит сразу после инициализации
- Нельзя повторно использовать буфер для других целей без получения информации о завершении посылки
- Окончание процесса передачи можно определить с помощью *request* и процедур MPI\_Wait и MPI\_Test
- Сообщение, отправленное любой из процедур MPI\_Send и MPI\_Isend, может быть принято любой из процедур MPI\_Recv и MPI\_Irecv

# Общие процедуры MPI. Асинхронные передачи

```
int MPI_Irecv
```

```
(void*, int, MPI_Datatype, int, int, MPI_Comm, MPI_Request *request)
```



(OUT) Указатель начала буфера



(OUT) ID операции

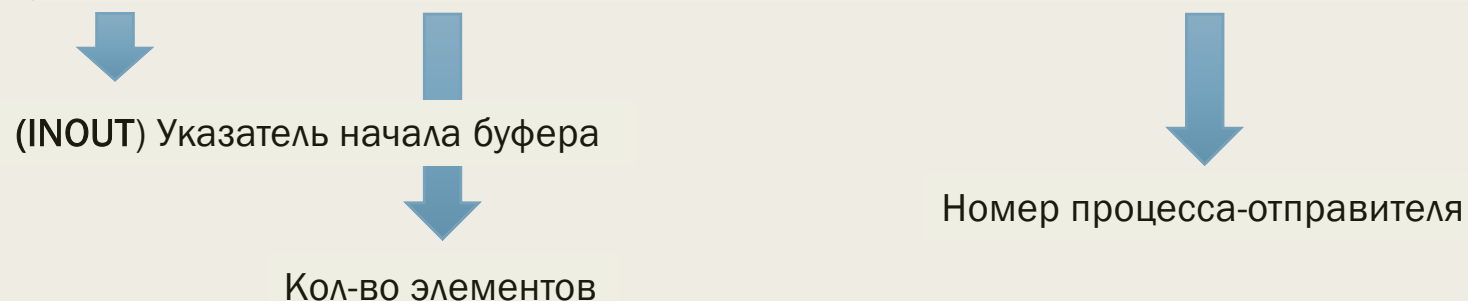
- Возврат происходит сразу после инициализации
- Окончание процесса можно определить с помощью *request* и процедур MPI\_Wait и MPI\_Test
- Сообщение, отправленное любой из процедур MPI\_Send и MPI\_Isend, может быть принято любой из процедур MPI\_Recv и MPI\_Irecv

## Общие процедуры MPI. Задача 2

- Составить параллельную программу, суммирующую все натуральные числа от 1 до  $N$
- Каждый процесс получает свой диапазон чисел для суммирования
- $N$  задается аргументом запуска
- Вести подсчет времени выполнения всей программы + времени подсчета на каждом процессе

## Общие процедуры MPI. Коллективные взаимодействия

```
int MPI_Bcast  
(void *, int count, MPI_Datatype, int source, MPI_Comm)
```



- Рассылка сообщения происходит от *source* всем процессам, включая рассылающий процесс
- При возврате из процедуры содержимое *buf* процесса *source* будет скопировано в локальный буфер процесса
- Значения параметров *count*, *datatype* и *source* должны быть одинаковыми у всех процессов.

# Общие процедуры MPI. Коллективные взаимодействия

## MPI\_Scatter

(void\* send\_buf, int send\_count, MPI\_Datatype,



Буфер отправки



Кол-во отправляемых элементов  
в пакете



Тип отправляемых элементов

void\* recv\_buf, int recv\_count, MPI\_Datatype, int root, MPI\_Comm)



(OUT) Буфер приема



Кол-во принимаемых элементов  
в пакете



Тип принимаемых элементов

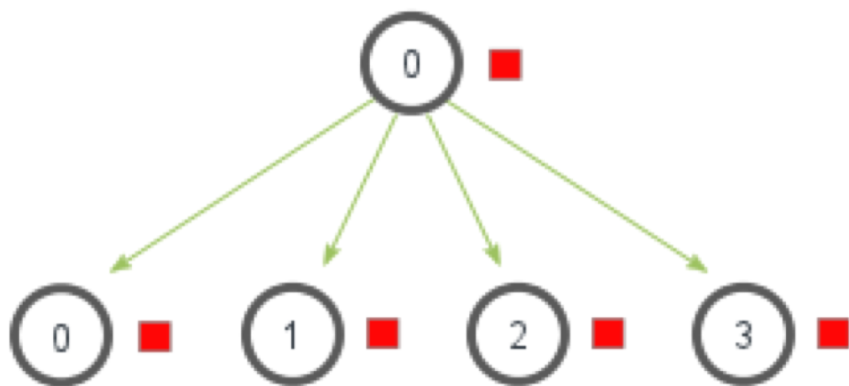


Номер рассылающего процесса

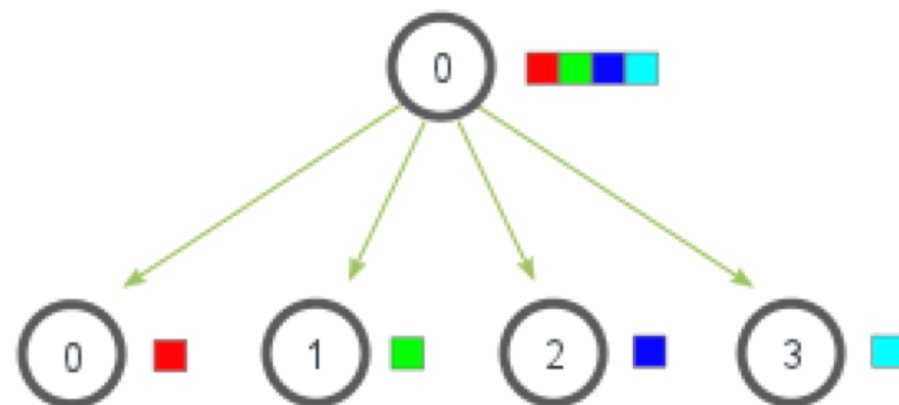


# Общие процедуры MPI. Коллективные взаимодействия

MPI\_Bcast



MPI\_Scatter



# Общие процедуры MPI. Коллективные взаимодействия

## MPI\_Gather

(void\* send\_buf, int send\_count, MPI\_Datatype,



Буфер отправки



Кол-во отправляемых элементов  
в пакете



Тип отправляемых элементов

void\* recv\_buf, int recv\_count, MPI\_Datatype, int dest, MPI\_Comm)



(OUT) Буфер приема



Кол-во принимаемых элементов  
в пакете



Тип принимаемых элементов



Номер принимающего процесса

## Общие процедуры MPI. Коллективные взаимодействия

### MPI\_Gather

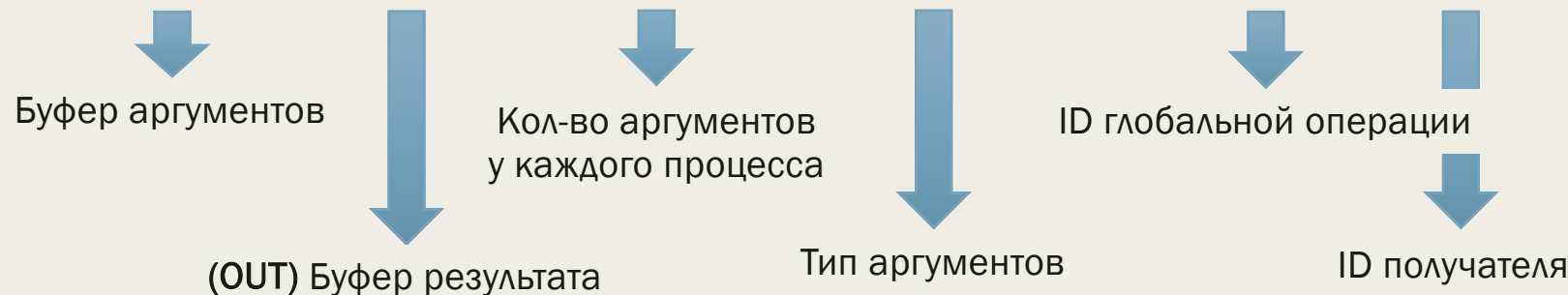
```
(void* send_buf, int send_count, MPI_Datatype, void*  
recv_buf, int recv_count, MPI_Datatype, int dest, MPI_Comm)
```

- Собирающий процесс сохраняет данные в *recv\_buf*, располагая их в порядке возрастания номеров процессов
- Параметр *recv\_buf* имеет значение только на собирающем процессе и на остальных игнорируется
- Значения параметров *count*, *datatype* и *dest* должны быть одинаковыми у всех процессов

# Общие процедуры MPI. Коллективные взаимодействия

## int MPI\_Reduce

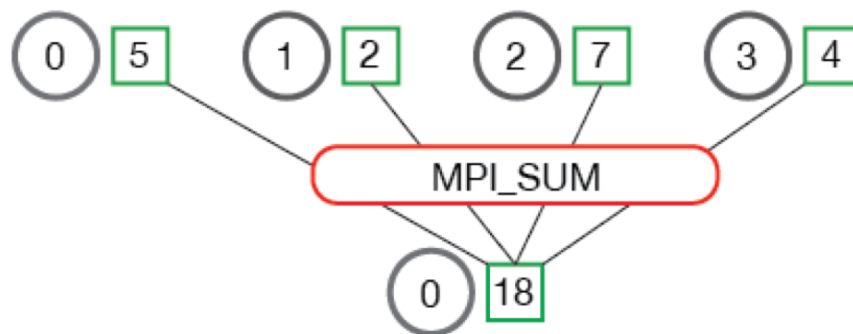
(void \*sbuf, void \*rbuf, int count, MPI\_Datatype, MPI\_Op op, int root, MPI\_Comm)



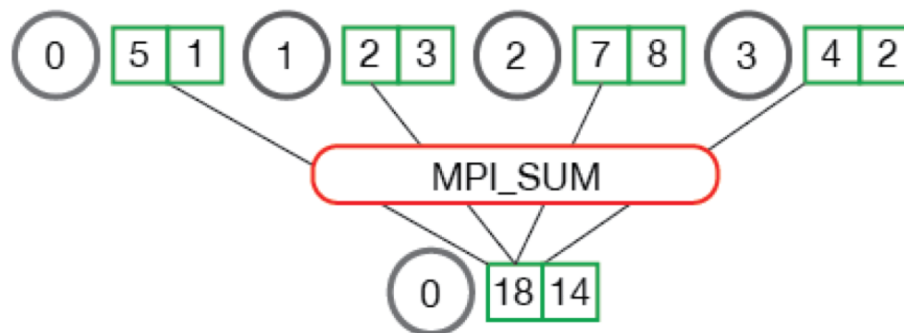
- Выполнение *count* глобальных операций *op* с возвратом результата в буфер *rbuf* процесса *root*
- Операция выполняется независимо над соответствующими аргументами всех процессов.
- Значения *count* и *datatype* у всех процессов должны быть одинаковыми
- Из соображений эффективности реализации предполагается, что операция *op* обладает свойствами ассоциативности и коммутативности

# Общие процедуры MPI. Коллективные взаимодействия

MPI\_Reduce



MPI\_Reduce



## Общие процедуры MPI. Задача 2\*

- Составить параллельную программу, суммирующую все натуральные числа от 1 до N с использованием операций коллективного взаимодействия
- N задается аргументом запуска
- Вести подсчет времени выполнения всей программы + времени подсчета на каждом процессе