MPI

Message Passing Interface

Параллельная программа - множество одновременно выполняемых **процессов.**

Каждый процесс порождается на основе одного и того же программного кода (fork).

Количество процессов определяется в момент запуска программы.

Все процессы последовательно пронумерованы от 0 до **p-1** (ранг процесса), где **p** есть общее количество процессов.

MPI

- Подключение библиотеки:

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

- Начало работы MPI:

```
int MPI_Init ( int *agrc,
char ***argv );
```

- Завершение работы:

```
MPI_Finalize();
```

```
#include "mpi.h"

int main ( int argc, char *argv[] )
{
        <программный код без MPI функций>
    MPI_Init ( &agrc, &argv );
        <программный код с MPI функциями>
    MPI_Finalize();
        <программный код без MPI функций>
    return 0;
}
```

Функции **MPI_Init** и **MPI_Finalize** являются обязательными и должны быть выполнены (только один раз) каждым процессом параллельной программы.

MPI

Компиляция программы:

mpicc -o <исполняемый файл> <исходный файл>.c

Запуск:

mpirun -n <число процессов>

<исполняемый файл> [аргументы]

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

int main(int argc, char* argv[])
{
    int ProcRank;
    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    printf("Hello from process %d\n", ProcRank);

    MPI_Finalize();
    return 0;
}

/// mpicc -o hello main.c
/// mpirun -n 4 hello
```

MPI

- Определение количества процессов

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

- Определения ранга процесса

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

```
#include "mpi.h"

int main ( int argc, char *argv[] )
{
    int ProcNum, ProcRank;
        <программный код без использования MPI функций>
    MPI_Init ( &agrc, &argv );
    MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank);
        <программный код с использованием MPI функций>
    MPI_Finalize();
        <программный код без использования MPI функций>
    return 0;
}
```

MPI

Коммуникатор - служебный объект, объединяющий в своем составе группу процессов и контекст, используемый при передачи данных.

Все процессы входят в состав создаваемого по умолчанию коммуникатора с идентификатором MPI_COMM_WORLD.

MPI_Comm comm = MPI_COMM_WORLD;

MPI

Операции передачи сообщений:

- Парные (point-to-point) – операции между двумя процессами

  MPI_Send, MPI_Recv…

- Коллективные (collective) – коммуникационные действия для **одновременного** взаимодействия нескольких процессов

  MPI_Bcast, MPI_Reduce…

MPI

Базовые типы

MPI_Datatype	C Datatype
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	Double
MPI_FLOAT	Float
MPI_INT	Int
MPI_LONG	Long
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_SHORT	short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

Производные типы

- Функции создания типа
  MPI_Type_contiguous
  MPI_Type_vector
  MPI_Type_hvector
  MPI_Type_indexed
  MPI_Type_hindexed
  MPI_Type_struct

- Функция регистрации типа
  `int` **`MPI_Type_commit`**
  `(MPI_Datatype *datatype);`

MPI

Передача сообщений:

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

Приём сообщений:

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

MPI_ANY_SOURCE

MPI_ANY_TAG

MPI_COMM_WORLD

buf – адрес буфера с данными отправляемого сообщения.
count – количество элементов данных в сообщении.
type - тип элементов данных пересылаемого сообщения.
dest - ранг процесса, которому отправляется сообщение.
source - ранг процесса, от которого должен быть выполнен прием сообщения.
tag - значение-тег, используемое для идентификации сообщений.
comm - коммуникатор, в рамках которого выполняется передача данных.
status – указатель на структуру данных с информацией о результате выполнения операции.

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

int main(int argc, char* argv[])
{
    int ProcNum, ProcRank, RecvRank;
    MPI_Status Status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    if ( ProcRank == 0 )
    {
        /// Действия, выполняемые только процессом с рангом 0
        printf ("\n Hello from process %d", ProcRank);
        for ( int i = 1; i < ProcNum; i++ )
        {
            MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
            MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
            printf("\n Hello from process %d", RecvRank);
        }
    } else {
        /// Сообщение, отправляемое всеми процессами,
        /// кроме процесса с рангом 0
        MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
    }

    MPI_Finalize();
    return 0;
}
```

MPI

Описание всех функций:
https://www.open-mpi.org/doc/current/

Подробное описание MPI на русском:
http://rsusu1.rnd.runnet.ru/tutor/method/m2/content.html