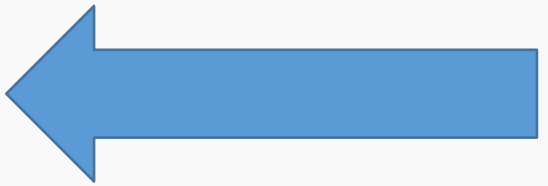
Типы данных MPI. Создание новых типов данных.

презентация подготовлена *Кузьмицким Владимиром*

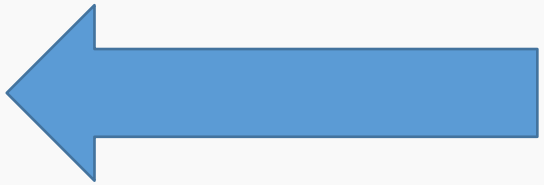
3 курс ФПМИ БГУ 2021

Функции для передачи данных

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```



```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```



Базовые типы

MPI datatype	C equivalent
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	char

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

int main(int argc, char **argv) {
    int size, rank, i;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int number;
    if (rank == 0) {
        number = 42;
        MPI_Send(&number, count: 1, MPI_INT, dest: 1, tag: 0, MPI_COMM_WORLD);
    } else {
        MPI_Recv(&number, count: 1, MPI_INT, source: 0, tag: 0, MPI_COMM_WORLD,
                 status: MPI_STATUS_IGNORE);
        printf( format: "Process %d received number %d from process 0\n", rank, number);
    }
    MPI_Finalize();
    return 0;
}
```

```
vkuzia@VKuziaPC:~/Desktop/BSU/AK/projects/test$ mpirun -np 2 ./example1 -fopenmp
Process 1 received number 42 from process 0
```

MPI_Status

- Sender’s rank
- Message Tag
- Message Length

```
MPI_Get_count(
    MPI_Status* status,
    MPI_Datatype datatype,
    int* count)
```

```
const int MAX_NUMBERS = 100;
int numbers[MAX_NUMBERS];
int number_amount;
if (rank == 0) {
    srand( seed: time( timer: NULL));
    number_amount = (rand() / (float) RAND_MAX) * MAX_NUMBERS;
    MPI_Send(numbers, number_amount, MPI_INT,  dest: 1,  tag: 0, MPI_COMM_WORLD);
    printf( format: "0 sent %d numbers to 1\n", number_amount);
} else if (rank == 1) {
    MPI_Status status;
    MPI_Recv(numbers, MAX_NUMBERS, MPI_INT,  source: 0,  tag: 0, MPI_COMM_WORLD,
             &status);
    MPI_Get_count(&status, MPI_INT, &number_amount);
    printf( format: "1 received %d numbers from 0. Message source = %d, "
           "tag = %d\n",
           number_amount, status.MPI_SOURCE, status.MPI_TAG);
}
```

```
vkuzia@VKuziaPC:~/Desktop/BSU/AK/projects/test$ mpirun -np 2 ./example2 -fopenmp
0 sent 74 numbers to 1
1 received 74 numbers from 0. Message source = 0, tag = 0
```

MPI_Probe

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

Пользовательские типы

Так делать не надо

```
#define msgTag 10
struct {
    int   i;
    float f[4];
    char  c[8];
} s;
MPI_Send(&s.i, count: 1, MPI_INT,   target_rank, msgTag,   MPI_COMM_WORLD );
MPI_Send( s.f, count: 4, MPI_FLOAT, target_rank, tag: msgTag+1, MPI_COMM_WORLD );
MPI_Send( s.c, count: 8, MPI_CHAR,  target_rank, tag: msgTag+2, MPI_COMM_WORLD );
```

А вот так уже можно

```
// On send
int buf_pos = 0;
char temp_buf[ sizeof(s) ];
MPI_Pack(&s.i,  incount: 1, MPI_INT, temp_buf, sizeof(temp_buf), &buf_pos, MPI_COMM_WORLD );
MPI_Pack(s.f,  incount: 4, MPI_FLOAT, temp_buf, sizeof(temp_buf), &buf_pos, MPI_COMM_WORLD );
MPI_Pack(s.c,  incount: 8, MPI_CHAR, temp_buf, sizeof(temp_buf), &buf_pos, MPI_COMM_WORLD );
MPI_Send(temp_buf, buf_pos, MPI_BYTE, target_rank, MSG_TAG, MPI_COMM_WORLD );

// On receive
int buf_pos2 = 0;
char temp_buf_2[ sizeof(s) ]; // IT'S A TRAP
MPI_Recv(temp_buf, sizeof(temp_buf), MPI_BYTE, source_rank, MSG_TAG,
         MPI_COMM_WORLD, &status );
MPI_Unpack(temp_buf_2, sizeof(temp_buf), &buf_pos2, &s.i,  outcount: 1, MPI_INT, MPI_COMM_WORLD);
MPI_Unpack(temp_buf_2, sizeof(temp_buf), &buf_pos2, s.f,  outcount: 4, MPI_FLOAT, MPI_COMM_WORLD);
MPI_Unpack(temp_buf_2, sizeof(temp_buf), &buf_pos2, s.c,  outcount: 8, MPI_CHAR, MPI_COMM_WORLD);
```

Обходим ловушку на приёмной стороне

```
int buf_size = 0;
void *buffer;
MPI_Pack_size( incount: 1, MPI_INT,   MPI_COMM_WORLD, &buf_size );
MPI_Pack_size( incount: 4, MPI_FLOAT, MPI_COMM_WORLD, &buf_size );
MPI_Pack_size( incount: 8, MPI_CHAR,  MPI_COMM_WORLD, &buf_size );
buffer = malloc(buf_size );
```

Пользовательские типы данных

Общие правила:

- пользовательские описатели типов создаются на базе созданных ранее пользовательских описателей, и на базе встроенных описателей
- встроенные описатели имеются для ВСЕХ стандартных типов Си: MPI_INT, MPI_CHAR, MPI_LONG, MPI_FLOAT, MPI_DOUBLE и так далее; тип MPI_BYTE служит для передачи двоичных данных
- после конструирования, но перед использованием описатель должен быть зарегистрирован функцией MPI_Type_commit;
- после использования описатель должен быть очищен функцией MPI_Type_free

```
int a[16];
MPI_Datatype intArray16;
MPI_Type_contiguous( count: 16, MPI_INT, &intArray16 );
MPI_Type_commit( &intArray16 );

MPI_Send( a, 16, MPI_INT, ... );
MPI_Send( a, 1, intArray16, ... );

MPI_Type_free( &intArray16 );
```

MPI Struct?

для однородных данных можно обойтись без них

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

struct Point {
    double x, y, z;
};

int main(int argc, char **argv) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Datatype dt_point;
    MPI_Type_contiguous( count: 3, MPI_DOUBLE, &dt_point);
    MPI_Type_commit(&dt_point);

    int points_count = 10;
    struct Point data[points_count];
    if (rank == 0) {
        for (int i = 0; i < points_count; ++i) {
            data[i].x = (double) i;
            data[i].y = (double) -i;
            data[i].z = (double) i * i;
        }
        MPI_Send(data, points_count, dt_point,  dest: 1,  tag: 0, MPI_COMM_WORLD);
    } else {
        MPI_Recv(data, points_count, dt_point,  source: 0,  tag: 0, MPI_COMM_WORLD,  status: MPI_STATUS_IGNORE);
        for (int i = 0; i < points_count; ++i) {
            printf( format: "Point #%d : (%lf, %lf, %lf)\n", i, data[i].x, data[i].y, data[i].z);
        }
    }
    MPI_Finalize();
}
```

Использование разнородной структуры

```
struct Person {
    int age;
    double height;
    char name[10];
};

struct Person DEFAULT_PERSON = {
        .age: 19, .height: 1.78, .name: "Kuzia"
};
```

```
MPI_Datatype person_type;
int lengths[3] = {1, 1, 10};
MPI_Aint displacements[3] = {offsetof( TYPE: struct Person, age),
                             offsetof( TYPE: struct Person, height),
                             offsetof( TYPE: struct Person, name)};

MPI_Datatype types[3] = {MPI_INT, MPI_DOUBLE, MPI_CHAR};
MPI_Type_create_struct( count: 3, lengths, displacements, types, &person_type);
MPI_Type_commit(&person_type);
```

```c
if (rank == 0) {
    struct Person buffer = DEFAULT_PERSON;
    printf( format: "MPI process %d sends person:\n\t- age = %d\n\t- height = %f\n\t- name = %s\n", rank, buffer.age,
           buffer.height, buffer.name);
    MPI_Send(&buffer,  count: 1, person_type,  dest: 1,  tag: 0, MPI_COMM_WORLD);
} else {
    struct Person received;
    MPI_Recv(&received,  count: 1, person_type,  source: 0,  tag: 0, MPI_COMM_WORLD,  status: MPI_STATUS_IGNORE);
    printf( format: "MPI process %d received person:\n\t- age = %d\n\t- height = %f\n\t- name = %s\n", rank, received.age,
           received.height, received.name);
}
MPI_Type_free(&person_type);
MPI_Finalize();
```

```
vkuzia@VKuziaPC:~/Desktop/BSU/AK/projects/test$ mpirun -np 2 ./example4
MPI process 0 sends person:
        - age = 19
        - height = 1.780000
        - name = Kuzia
MPI process 1 received person:
        - age = 19
        - height = 1.780000
        - name = Kuzia
```

MPI_Type_vector

```
MPI_Type_vector(
  int count,        /* количество элементов в новом типе */
  int blocklength, /* количество ячеек базового типа в одном элементе */
  int stride,       /* расстояние между НАЧАЛАМИ эл-тов, в числе ячеек */
  MPI_Datatype oldtype,  /* описатель базового типа, т.е. типа ячейки */
  MPI_Datatype &newtype  /* ссылка на новый описатель */
);
```

MPI_Type_indexed

Немного контроля

MPI_Type_extent и **MPI_Type_size** : важные информационные функции. Их характеристики удобно представить в виде таблицы:

<table>
<tr><th>Вид данных</th><th>sizeof</th><th>MPI_Type_extent</th><th>MPI_type_size</th></tr>
<tr><td>стандартный тип</td><td colspan="3">равносильны</td></tr>
<tr><td>массив</td><td colspan="3">равносильны</td></tr>
<tr><td>структура</td><td colspan="2">равносильны</td><td>sizeof(поле1)+sizeof(поле2)+...</td></tr>
<tr><td>Описатель типа MPI с перекрытиями и разрывами</td><td>не определена</td><td>адрес последней ячейки данных - адрес первой ячейки данных + sizeof(последней ячейки данных)</td><td>sizeof(первой ячейки данных) + sizeof(второй ячейки данных) + ...</td></tr>
</table>

Можно сказать, что MPI_Type_extent сообщает, сколько места переменная типа занимает при хранении в памяти, а MPI_Type_size - какой МИНИМАЛЬНЫЙ размер она будет иметь при передаче (ужатая за счет неиспользуемого пространства).

Полезные ссылки

- Введение в MPI

  https://www.opennet.ru/docs/RUS/MPI_intro/
- MPI_Send & MPI_Recv + Elementary datatypes

  https://mpitutorial.com/tutorials/mpi-send-and-receive/
- MPI_Probe & MPI_Status

  https://mpitutorial.com/tutorials/dynamic-receiving-with-mpi-probe-and-mpi-status/

- Примеры из презентации

  https://github.com/VKuzia/mpi-examples

Спасибо за внимание :)