

# Лекция 4

## Стандарт MPI

### Производные типы данных (derived data types)

**Курносов Михаил Георгиевич**

E-mail: [mkurnosov@gmail.com](mailto:mkurnosov@gmail.com)

WWW: [www.mkurnosov.net](http://www.mkurnosov.net)

Курс «Параллельные вычислительные технологии»

Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Осенний семестр, 2016

# Пользовательские типы данных (MPI Derived Data Types)

- Как передать структуру C/C++ в другой процесс?
- Как передать другому процессу столбец матрицы?
- (в C/C++ массивы хранятся в памяти строка за строкой – row-major order,
- в Fortran столбец за столбцом – column-major order)
- Как реализовать прием сообщений различных размеров
- (заголовок сообщения содержит его тип, размер)?

# Пользовательские типы данных (MPI Derived Data Types)

Как передать массив частиц другому процессу?

```
typedef struct {  
    double x;  
    double y;  
    double z;  
    double f;  
    int data[8];  
} particle_t;
```

```
int main(int argc, char **argv)  
{  
    int rank;  
    MPI_Init(&argc, &argv);  
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
  
    int nparticles = 1000;  
    particle_t *particles = malloc(sizeof(*particles) * nparticles);
```

# Пользовательские типы данных (MPI Derived Data Types)

```
/* Create data type for message of type particle_t */
MPI_Datatype types[5] = {MPI_DOUBLE, MPI_DOUBLE, MPI_DOUBLE, MPI_DOUBLE,
                        MPI_INT};

int blocklens[5] = {1, 1, 1, 1, 8};

MPI_Aint displs[0];
displs[0] = offsetof(particle_t, x);
displs[1] = offsetof(particle_t, y);
displs[2] = offsetof(particle_t, z);
displs[3] = offsetof(particle_t, f);
displs[4] = offsetof(particle_t, data);

MPI_Datatype parttype;
MPI_Type_create_struct(5, blocklens, displs,
                      types, &parttype);
MPI_Type_commit(&parttype);
```

```
                                // size = 64
struct {
    double x;    // offset 0
    double y;    // offset 8
    double z;    // offset 16
    double f;    // offset 24
    int data[8]; // offset 32
}
```

Компилятор выравнивает структуру

```
                                // size = 72
struct {
    double x;    // offset 0
    double y;    // offset 8
    double z;    // offset 16
    int type     // offset 24
    double f;    // offset 32
    int data[8]; // offset 40
}
```

# Пользовательские типы данных (MPI Derived Data Types)

```
/* Init particles */
if (rank == 0) {
    // Random positions in simulation box
    for (int i = 0; i < nparticles; i++) {
        particles[i].x = rand() % 10000;
        particles[i].y = rand() % 10000;
        particles[i].z = rand() % 10000;
        particles[i].f = 0.0;
    }
}
MPI_Bcast(particles, nparticles, parttype, 0, MPI_COMM_WORLD);

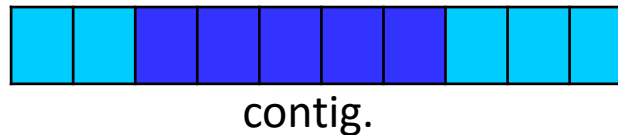
// code ...

MPI_Type_free(&parttype);
free(particles);
MPI_Finalize( );
return 0;
}
```

# MPI\_Type\_contiguous

```
int MPI_Type_contiguous(int count, MPI_Datatype oldtype,  
                        MPI_Datatype *newtype)
```

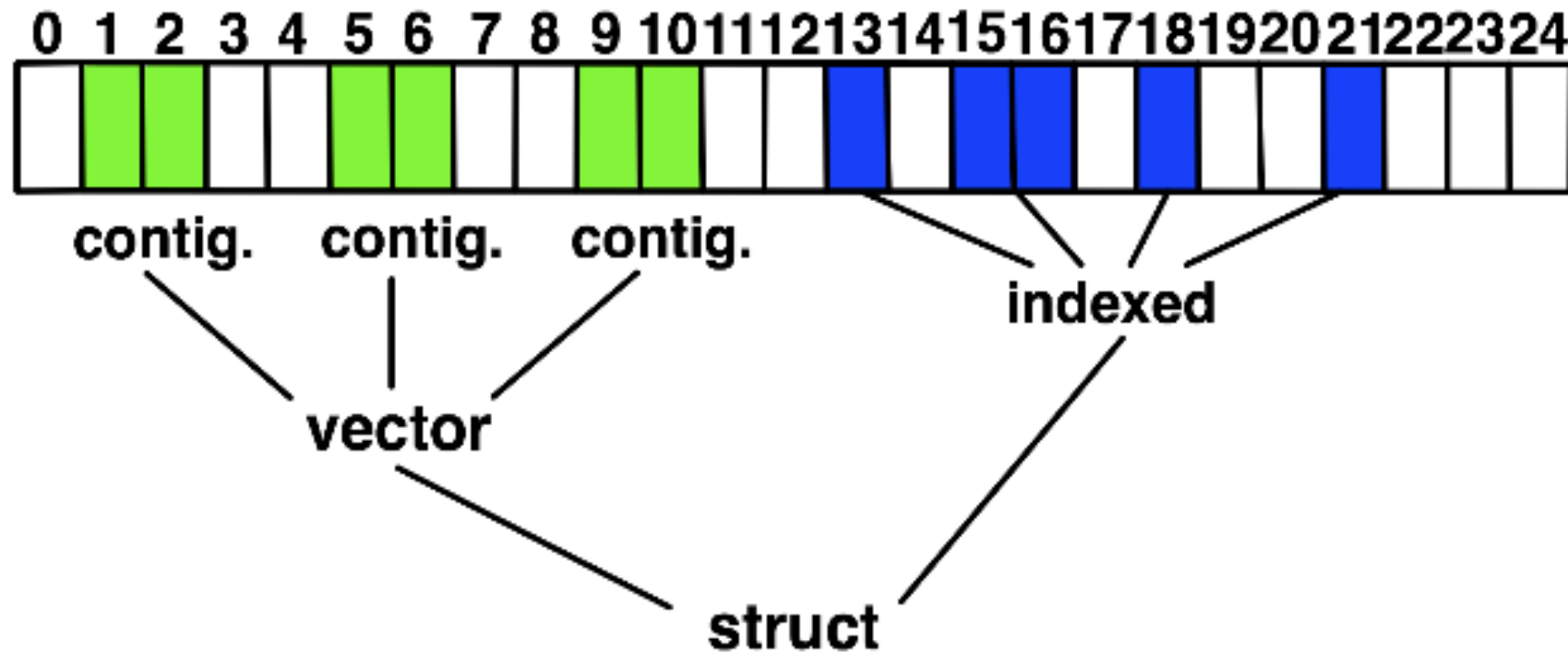
- Непрерывная последовательность в памяти (массив) элементов типа oldtype
- Одномерный массив



## Пример

- oldtype = {(double, 0), (char, 8)}  
(1 double и 1 char, 0 и 8 — смещения)
- MPI\_Type\_contiguous(3, MPI\_Datatype oldtype, &newtype)
- newtype = {(double, 0), (char, 8), (double, 16), (char, 24), (double, 32), (char, 40)}

# Пользовательские типы данных (MPI Derived Data Types)



# Пользовательские типы данных (MPI Derived Data Types)

```
int MPI_Type_vector(int count, int blocklength, int stride,  
                    MPI_Datatype oldtype, MPI_Datatype *newtype)  
  
int MPI_Type_indexed(int count, const int array_of_blocklengths[],  
                     const int array_of_displacements[],  
                     MPI_Datatype oldtype, MPI_Datatype *newtype)  
  
int MPI_Type_create_struct(int count,  
                           const int array_of_blocklengths[],  
                           const MPI_Aint array_of_displacements[],  
                           const MPI_Datatype array_of_types[],  
                           MPI_Datatype *newtype)  
  
int MPI_Type_create_subarray(int ndims, const int array_of_sizes[],  
                             const int array_of_subsizes[],  
                             const int array_of_starts[],  
                             int order, MPI_Datatype oldtype,  
                             MPI_Datatype *newtype)
```

...



# Выбор типа данных

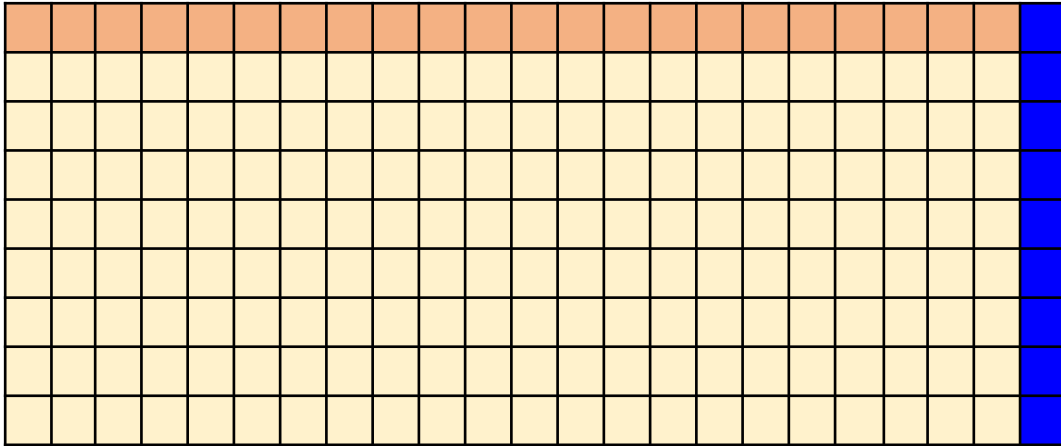
1. Предопределенные типы  
(MPI\_DOUBLE, MPI\_INT, MPI\_CHAR, ...)
2. contig
3. vector
4. index\_block
5. index
6. struct

*Slower*



# Передача строк и столбцов матрицы

```
double *grid = malloc(ny * nx * sizeof(double))
```



```
// Top and bottom borders type
```

```
MPI_Datatype row;
```

```
MPI_Type_contiguous(nx, MPI_DOUBLE, &row);
```

```
MPI_Type_commit(&row);
```

```
// Column type
```

```
MPI_Datatype col;
```

```
MPI_Type_vector(ny, 1, nx, MPI_DOUBLE, &col);
```

```
MPI_Type_commit(&col);
```

```
MPI_Recv(&grid[0], 1, row, rank, 0, comm, MPI_STATUS_IGNORE); // recv top row
```

```
MPI_Send(&grid[nx - 1], 1, col, rank, 0, comm); // send right column
```

```
MPI_Type_free(&row);
```

```
MPI_Type_free(&col);
```

## Упаковка данных (MPI\_Pack)

```
int main(int argc, char **argv)
{
    int rank, packsize, position;
    int a;
    double b;
    uint8_t packbuf[100];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0) {
        a = 15;
        b = 3.14;
```

Как передать int a и double b  
одним сообщением ?

## Упаковка данных (MPI\_Pack)

```
    packsize = 0; /* Pack data into the buffer */
    MPI_Pack(&a, 1, MPI_INT, packbuf, 100, &packsize, MPI_COMM_WORLD);
    MPI_Pack(&b, 1, MPI_DOUBLE, packbuf, 100, &packsize, MPI_COMM_WORLD);
}

MPI_Bcast(&packsize, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(packbuf, packsize, MPI_PACKED, 0, MPI_COMM_WORLD);
if (rank != 0) {
    position = 0; /* Unpack data */
    MPI_Unpack(packbuf, packsize, &position, &a, 1, MPI_INT, MPI_COMM_WORLD);
    MPI_Unpack(packbuf, packsize, &position, &b, 1, MPI_DOUBLE,
               MPI_COMM_WORLD);
}
printf("Process %d unpacked %d and %lf\n", rank, a, b);
MPI_Finalize( );
return 0;
}
```

# Обработка изображения (contrast)



height

width

- 1) Вычисляется среднее квадратичное значений всех пикселей (RMS)

$$\text{RMS} = \sqrt{\text{sum}(L[i][j] * L[i][j]) / (H * W)}$$

- 2) Каждый пиксель  $L[i,j]$  преобразуется

$$L[i,j] = 2 * L[i, j] - \text{RMS}$$

$$\text{if } (L[i,j] < 0) \text{ then } L[i,j] = 0$$

$$\text{if } (L[i,j] > 255) \text{ then } L[i,j] = 255$$

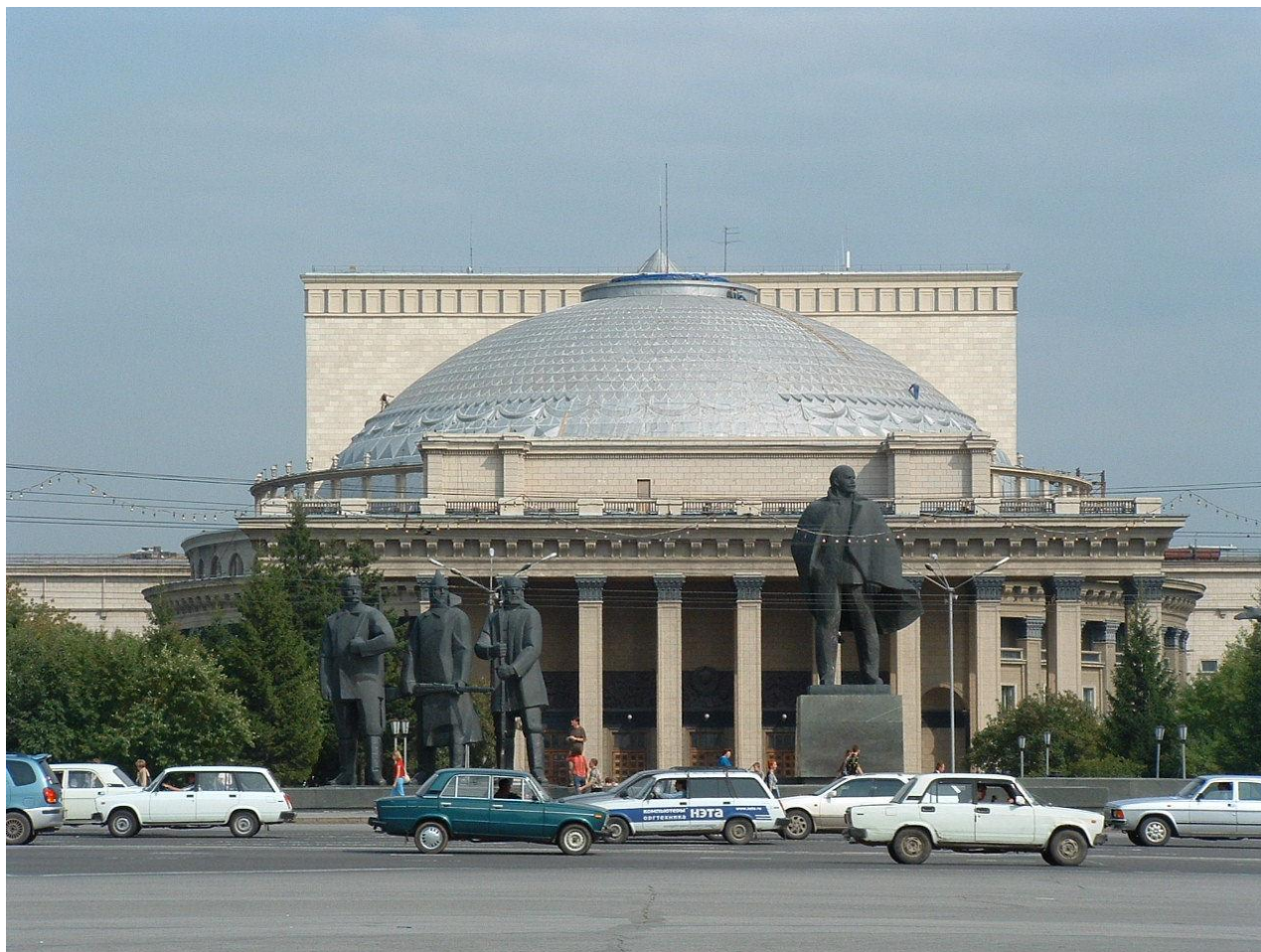




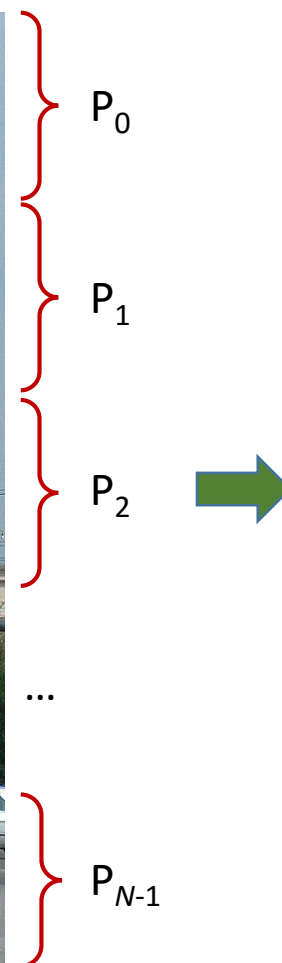
# Обработка изображения (contrast)

```
npixels = width * height;  
npixels_per_process = npixels / commsize;
```

height



width



1D decomposition



# Обработка изображения (contrast)

```
int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);
    int rank, commsize;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &commsize);

    int width, height, npixels, npixels_per_process;
    uint8_t *pixels = NULL;
    if (rank == 0) {
        width = 15360; // 15360 x 8640: 16K Digital Cinema (UHDTV) ~ 127 MiB
        height = 8640;
        npixels = width * height;
        pixels = xmalloc(sizeof(*pixels) * npixels);
        for (int i = 0; i < npixels; i++)
            pixels[i] = rand() % 255;
    }

    MPI_Bcast(&npixels, 1, MPI_INT, 0, MPI_COMM_WORLD); // Send size of image
    npixels_per_process = npixels / commsize;
    uint8_t *rbuf = xmalloc(sizeof(*rbuf) * npixels_per_process);
    // Send a part of image to each process
    MPI_Scatter(pixels, npixels_per_process, MPI_UINT8_T, rbuf, npixels_per_process, MPI_UINT8_T,
               0, MPI_COMM_WORLD);
}
```

## Обработка изображения (contrast)

```
int sum_local = 0;
for (int i = 0; i < npixels_per_process; i++)
    sum_local += rbuf[i] * rbuf[i];

/* Calculate global sum of the squares */
int sum = 0;
// MPI_Reduce(&sum_local, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Allreduce(&sum_local, &sum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

double rms;
// if (rank == 0)
rms = sqrt((double)sum / (double)npixels);

//MPI_Bcast(&rms, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```



# Обработка изображения (contrast)

```
/* Contrast operation on subimage */
for (int i = 0; i < npixels_per_process; i++) {
    int pixel = 2 * rbuf[i] - rms;
    if (pixel < 0)
        rbuf[i] = 0;
    else if (pixel > 255)
        rbuf[i] = 255;
    else
        rbuf[i] = pixel;
}
MPI_Gather(rbuf, npixels_per_process, MPI_UINT8_T, pixels,
           npixels_per_process, MPI_UINT8_T, 0, MPI_COMM_WORLD);
if (rank == 0)
    // Save image...

free(rbuf);
if (rank == 0)
    free(pixels);
MPI_Finalize();
}
```

# Обработка изображения (contrast)

**Равномерно ли пиксели изображения распределяются по процессам?**

$\text{npixels\_per\_process} = \text{npixels} / \text{commsize};$

## **Загрузка исходного изображения**

- а) Корневой процесс загружает изображение и рассылает части остальным (изображение может не поместиться в память одного узла)
- б) Каждый процесс загружает часть изображения (MPI I/O, NFS, Lustre)

## **Обработка изображения**

Вычисление RMS: MPI\_Reduce + MPI\_Bcast или MPI\_Allreduce

## **Сохранение изображения**

Сборка Gather — изображение формируется в памяти процесс 0

Сборка Send/Recv — части изображения последовательно сохраняются в файл

# Домашнее чтение

Pavan Balaji, William Gropp, Torsten Hoefler, Rajeev Thakur. **Advanced MPI Programming**  
// Tutorial at SC14, November 2014, <http://www.mcs.anl.gov/~thakur/sc14-mpi-tutorial/>

Torsten Hoefler. **Advanced MPI 2.2 and 3.0 Tutorial** //  
[http://hlor.inf.ethz.ch/teaching/mpi\\_tutorials/cscs12/hoefler\\_tutorial\\_advanced-mpi-2.2-and-mpi-3.0\\_cscs.pdf](http://hlor.inf.ethz.ch/teaching/mpi_tutorials/cscs12/hoefler_tutorial_advanced-mpi-2.2-and-mpi-3.0_cscs.pdf)

