Лекция 6
Стандарт MPI
Производные типы данных (derived data types)

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Параллельные вычислительные технологии»
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)
Осенний семестр

Пользовательские типы данных (MPI Derived Data Types)

- Как передать структуру C/C++ в другой процесс?
- Как передать другом 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)?

Размещение сообщений в памяти

Непрерывный участок памяти (contiguous)

Описание:
- адрес буфера (buf)
- размер буфера в байтах (size/count)

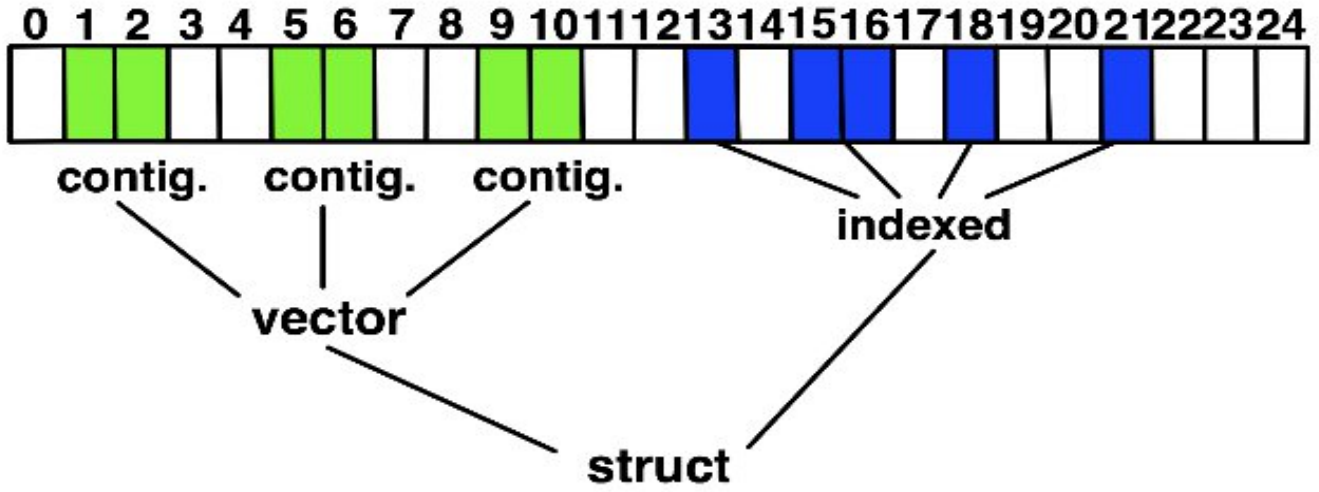
Пять блоков с регулярным шагом 1 (stride 1, non-contiguous)

Описание:
- адрес буфера (buf)
- количество блоков (blocks)
- размер блока в байтах (block size)
- шаг — расстояние между блокам в байтах (stride)

Пять блоков с произвольным размещением

Описание:
- адрес буфера (buf)
- количество блоков (blocks)
- массив размеров размер блоков (blocksize[])
- массив смещений блоков относительно buf (offsets[], displacements[])

Пользовательские типы данных (MPI Derived Data Types)



- **Type Signature** — список базовых типов данных (MPI_INT, MPI_DOUBLE, ...), составляющих производный тип
- **Type Map** — список элементов производного типа данных (элементарный тип, смещение)
- **extent** — размер типа, разность между смещением последнего и первого байта типа данных

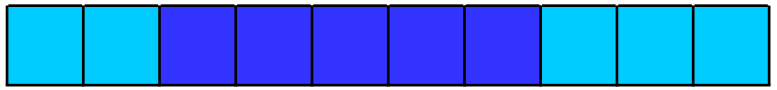
TypeMap = {(type[0], disp[0]), ..., (type[n-1] , disp[n-1])}

lb = min(disp[i]), ub = max(disp[i] + sizeof(type[i]) + eps), extent = ub(TypeMap) - lb(TypeMap)
eps = max(align[i]) — выравнивание на границу наибольшего блока

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



contig.

Пример

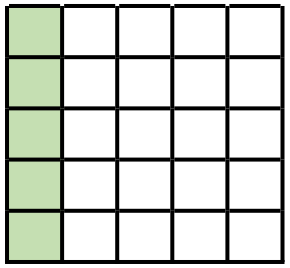
- `oldtype map = {(double, 0), (char, 8)}`
  `(1 double и 1 char, 0 и 8 – смещения, extent = 16)`

- `MPI_Type_contiguous(3, MPI_Datatype oldtype, &newtype)`

- `newtype = {(double, 0), (char, 8), (double, 16), (char, 24), (double, 32), (char, 40)}`

Передача матрицы с транспонированием

```
int N = 5, m[5 * 5], mm[5 * 5];

MPI_Aint lb, extent;
MPI_Datatype col, colresized, mat;
MPI_Type_vector(N, 1, N, MPI_INT, &col);
MPI_Type_get_extent(col, &lb, &extent);          // extent = 84

MPI_Type_create_resized(col, 0, sizeof(*m), &colresized);
MPI_Type_get_extent(colresized, &lb, &extent); // extent = 4

MPI_Type_contiguous(N, colresized, &mat);
MPI_Type_get_extent(mat, &lb, &extent);          // extent = 20
MPI_Type_get_true_extent(mat, &lb, &extent);    // extent = 100
MPI_Type_commit(&mat);
```



```
TypeMap(col) = ((int,0), (int,20), ..., (int,80)), extent=84
extent(colresized) = 4

TypeMap(mat) = ((colresized,0), (colresized,4), ..., (colresized,16)), extent=20
TrueExtent(mat) = 100
```

Передача матрицы с транспонированием

```
int m[5 * 5], mm[5 * 5];

// ...

if (rank == 0) {
    MPI_Send(m, N * N, MPI_INT, 1, 0, MPI_COMM_WORLD);

} else if (rank == 1) {
    MPI_Recv(mm, 1, mat, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}

MPI_Type_free(&colresized);
MPI_Type_free(&col);
MPI_Type_free(&mat);
```

Пользовательские типы данных (MPI Derived Data Types)

```
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);
```

Как передать массив частиц другому процессу?

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;
}
```

Пользовательские типы данных (MPI Derived Data Types)

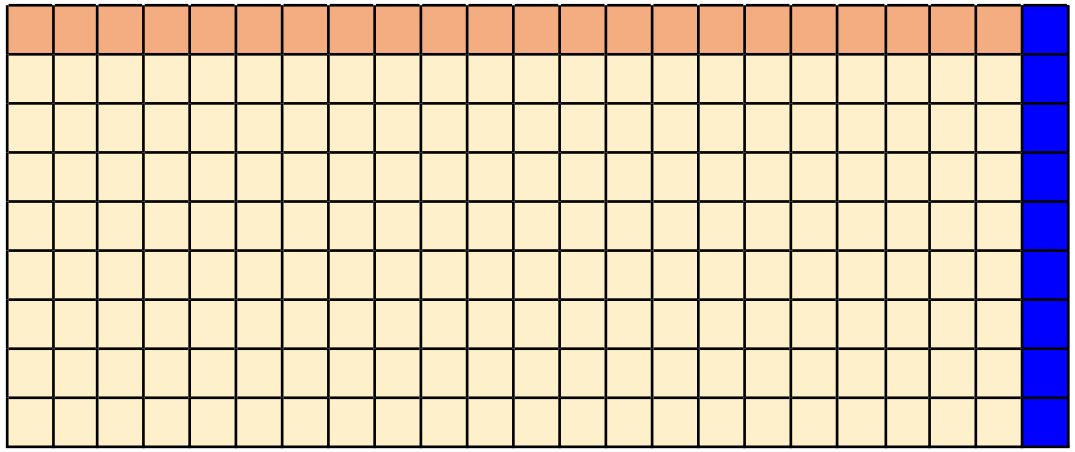
```
int MPI_Type_vector(int count, int blocklength, int stride,
                    MPI_Datatype oldtype, MPI_Datatype *newtype)

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 конструктора производного типа данных

1. Предопределенные типы (MPI_DOUBLE, MPI_INT, MPI_CHAR, ...)
2. contig
3. vector
4. index_block
5. index
6. struct

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;
}
```

Литература

- 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://htor.inf.ethz.ch/teaching/mpi_tutorials/cscs12/hoefler_tutorial_advanced-mpi-2.2-and-mpi-3.0_cscs.pdf