Introduction to MPI

Derived Data Types

Steve Crouch

Learning Objectives

- Explain what a derived data type is and how they can help
- Define and explain the challenges associated with non-contiguous memory
- Describe the limitations of using **`malloc()`** in terms of memory contiguity
- Use vectors to send column slices of an array to another rank

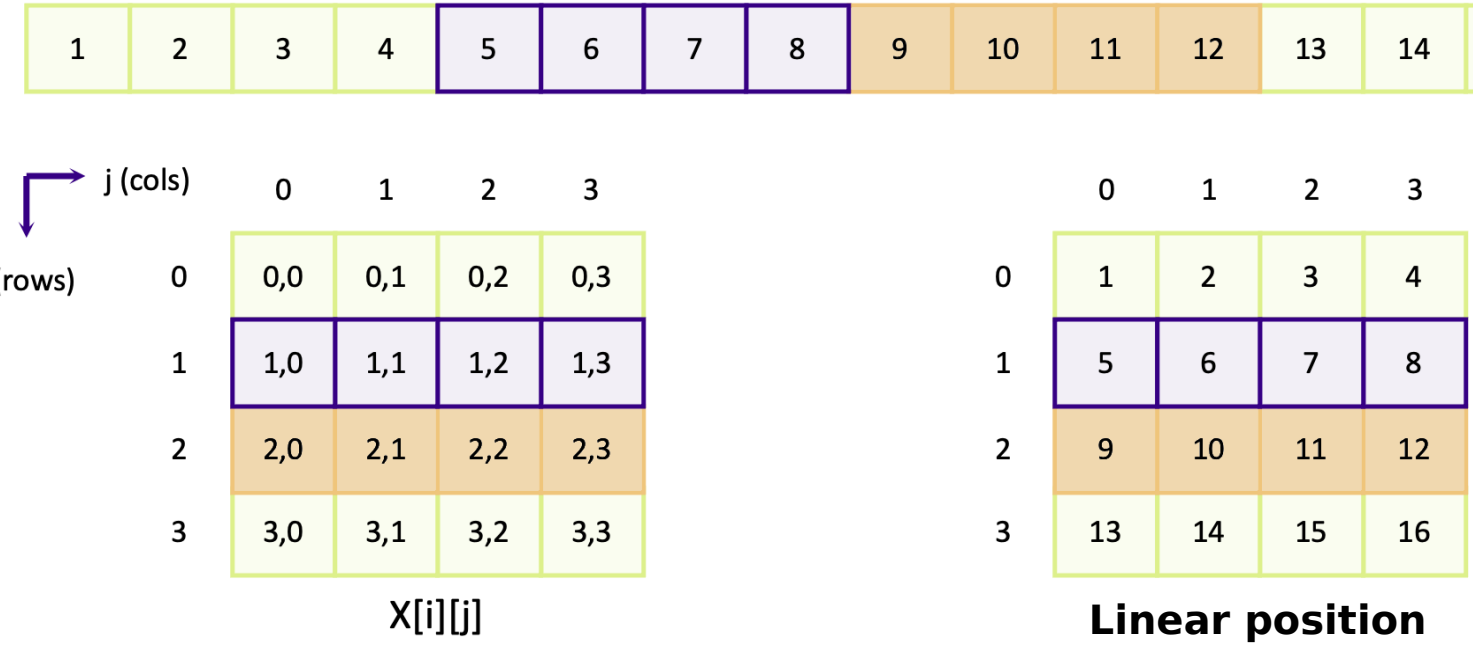
Going Beyond Simple Data

- In practice, we'll need to communicate complex data
  - e.g. n-dimensional arrays, custom structures, etc.
  - Almost all scientific computing problems require n-dimensional thinking
- MPI provides *derived data types*
  - An interface to specify how to translate complex data into efficiently transmissible data
  - Use to e.g. send slices/vectors as subset of an array

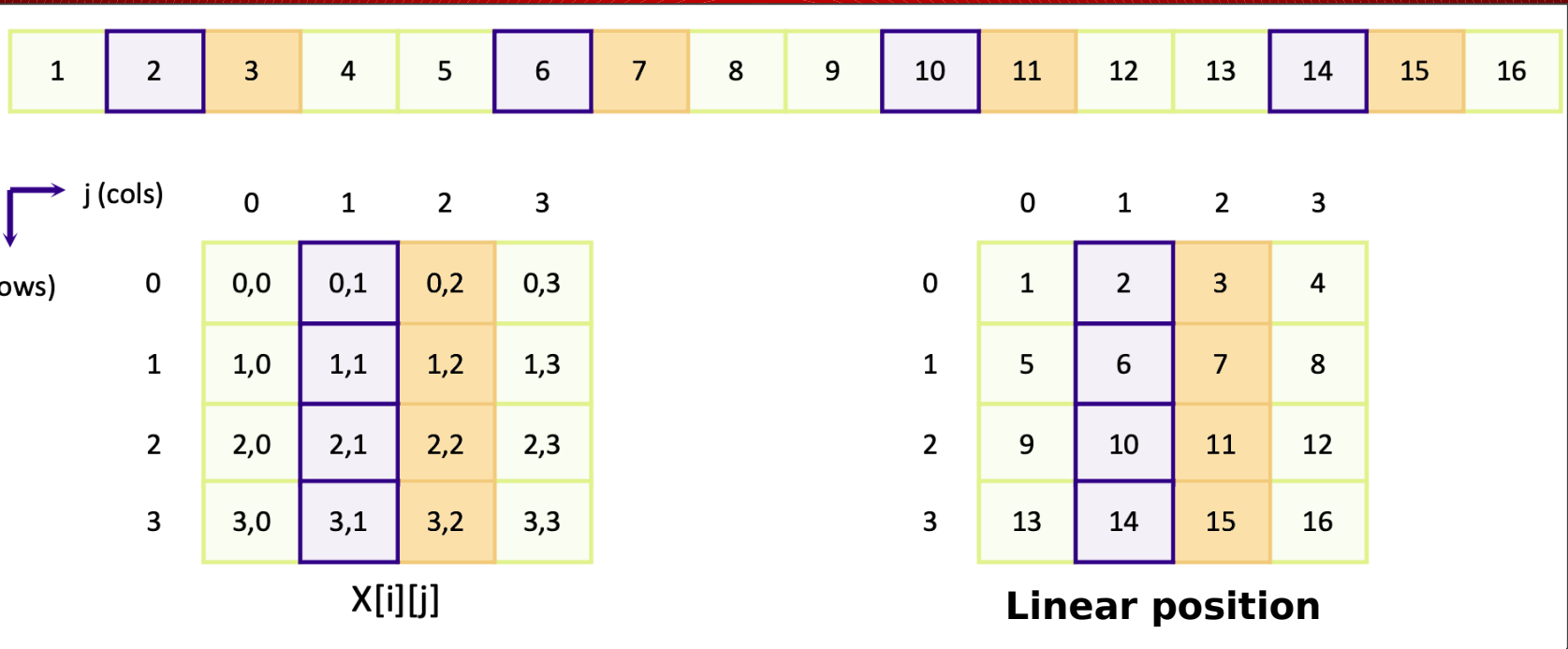
The Importance of Memory Contiguity

- Contiguous: multiple adjacent things without anything in-between
  - In context of MPI, array elements are contiguous when previous and next elements are stored in adjacent memory locations
- However, computer memory space is *linear*
  - Compiler and OS decide how to map/store elements in linear space
- Two ways to do this
  - **Row-major (C)** - elements in each column of a row are contiguous, e.g. `x[i][j]` preceded by `x[i][j-1]` and followed by `x[i][j+1]`
  - **Column-major (Fortran)** - the opposite, e.g. `x(i,j)` preceded by `x(i-1,j)` and followed by `x(i+1,j)`

Contiguity: Row-major Example



Contiguity: Row-major Example II



What About `malloc()`?

- We normally use **`malloc()`** to allocate memory for arrays
  - Obtain a contiguous array of elements
- e.g.

```
int num_rows = 3, num_cols = 5;

// Each pointer is the start of a row
float **matrix = malloc(num_rows * sizeof(float*));
for (int i = 0; i < num_rows; ++i) {
   // Allocate memory to store column elements for row i
   matrix[i] = malloc(num_cols * sizeof(float));
}

for (int i = 0; i < num_rows; ++i) {
   for (int j = 0; i < num_cols; ++j) {
      // Indexing done as matrix[rows][cols]
      matrix[i][j] = 3.14159;
   }
}
```

Problem:

- No guarantee subsequent memory is contiguous
- Assigns whatever memory is free
  - e.g. not always contiguous!

Some workarounds exist

- e.g. just use 1D arrays and map elements
- Use **`calloc()`**

Sending Data from our 2D Matrix

- For a row-major array, we can send elements of a single row (for a 4 x 4 matrix) easily...

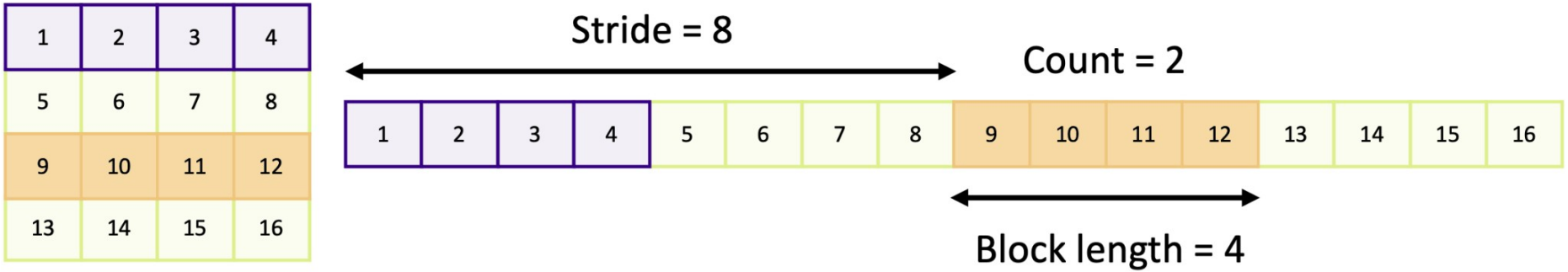
```
MPI_Send(&matrix[1][0], 4, MPI_INT ...);
```

- Can't do the same for a matrix column!
  - Elements vertically down the column aren't contiguous

Using Vectors to Send Array "Slices"

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

Number of "blocks" which make up the vector
Number of contiguous elements in a block
Number of elements between start of each block
Data type of original elements of the vector
Newly created data type to represent the vector



"Committing" and "Freeing" a Vector

- Before we can use a vector, we need to *commit* it
  - This allocates internal resources to handle this datatype
  - e.g. data structures, memory buffers, bookkeeping data

```
int MPI_Type_commit(
   MPI_Datatype *datatype
);
```

- Once we're finished with a vector, we need to *free* it

```
int MPI_Type_free(
   MPI_Datatype *datatype
);
```

Using Vectors

```
MPI_Datatype rows_type;
const int count = 2;
const int blocklength = 4;
const int stride = 8;

MPI_Type_vector(count, blocklength, stride,
                MPI_INT, &rows_type);
MPI_Type_commit(&rows_type);

int matrix[4][4] = {
   { 1,  2,  3,  4},
   { 5,  6,  7,  8},
   { 9, 10, 11, 12},
   {13, 14, 15, 16} };

if (my_rank == 0) {
    MPI_Send(&matrix[1][0], 1, rows_type, 1, 0,
             MPI_COMM_WORLD);
} else {
    const int num_elements = count * blocklength;
    int recv_buffer[num_elements];
    MPI_Recv(recv_buffer, num_elements, MPI_INT, 0, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
MPI_Type_free(&rows_type);
```

Both ranks

- Define data to subset
- Create vector, commit it
- Create 4x4 matrix
  - Want to send rows 2 and 4

Rank 0

- Send subset using vector

Rank 1

- Create receive buffer
- Receive data

Both ranks

- Free vector

Acknowledgements

Material developed for DiRAC in collaboration with

www.software.ac.uk