universidade de aveiro

Computação Paralela
Módulo MPI
2021/2022

Rui Costa

Email: americo.costa@ua.pt

Derived Datatypes

universidade de aveiro 

- MPI supports a wide range of functions creation of custom user-defined datatypes.
- These are typically used to transfer pieces of data that are stored/saved non-contiguously in the local memory, avoiding the need for repeated invocations of send/receive instructions, with advantages for simplicity of code writing and especially for communications efficiency.
- MPI derived datatypes may include different elementary datatypes (`int`, `float`, etc) with arbitrary spacings between them.

Derived Datatypes

In many kinds situations it is useful (and recommendable) to take advantage of these derived datatypes. Consider, as an example relevant to our programs, the communication of a column of a 2D array:
In memory the matrix is stored in row-major order, and elements to be sent and received are scattered with regular spacings of `ncols` between them. To deal with this inconvenience we may:

- Send the elements one by one using a cycle, at the cost of a lot of overhead in communications;
- Copy the elements to a contiguous buffer, and make a regular send, with still some overhead and extra coding work;
- Define a vector-like datatype, including skips of `ncols-1` spaces, and access the desired elements directly in a single operation.

Derived Datatypes

universidade de aveiro 

Suppose process 1 has a matrix `a[nrows][ncols]` of `double float` numbers and we want it to send the second column to process 0, who will store it as the last column of its own matrix `b[nrows][ncols]`. Note that `ncols` may be different in processes 0 and 1.

Each process defines a new vector-like datatype with

```
MPI_Type_vector(nrows, 1, ncols, MPI_DOUBLE, &column);
```

followed by

```
MPI_Type_commit(&column);
```

Derived Datatypes

universidade de aveiro 

To send and receive the column we use any standard communication function and provide the new datatype as an argument. For example, process 1 could call a send as,

```
MPI_Send(a[0][1], 1, column, 0, tag, comm)
```

While process 0 could receive the message with a

```
MPI_Recv(b[0][ncols-1], 1, column, 1, tag, comm,
MPI_STATUS_IGNORE)
```

Parallel reading and writing files

universidade de aveiro 

- MPI contains the notion of *file view*, which provides an easy way of multiple processes to read and write separate parts of the same file.
- Each process has its own *file view* that defines which parts (contiguous or non-contiguous) of the file are visible to the process. A read or write function can only read or write the data that is visible to the process, and skips all other data.
- The *file views* are specified using basic and derived datatypes.
- For example, in the cartesian decomposition of our programs we want each process to write a sub-matrix that is a part of the global matrix to be stored in the file.

Parallel reading and writing files

universidade de aveiro 

Each process defines a datatype that allows access only to its own sub-matrix in the global matrix:

```
MPI_Type_create_subarray(2, gsizes, lsizes,
start_ind, MPI_ORDER_C,MPI_DOUBLE, &filetype);
```

not forgetting to commit

```
MPI_Type_commit(&filetype);
```

- The fist argument is the number of dimensions of the array, we use 2.
- `gsizes` are the sizes of the global array in each dimension.
- `lsizes` are the sizes of the local array in each dimension.
- `start_ind` is the position of the first element of the subarray on the global array.

Parallel reading and writing files

universidade de aveiro 

In our programs the local matrices have extra ghost rows and/or columns that we do not wish to be written by the process, because they belong to neighboring processes and should be written by those neighbors.

This issue can also be fixed with the same function, defining a sub-array datatype that includes all the points of the local matrix that belong to the process, and excludes the ghost points:

```
MPI_Type_create_subarray(2, memsizes, lsizes,
start_ind, MPI_ORDER_C, MPI_DOUBLE, &memtype);
```

Parallel reading and writing files

universidade de aveiro 

At last, we are ready to create and open a file for storing the global matrix

```
MPI_File fp;

MPI_File_open(comm, ”matrix.bin", MPI_MODE_CREATE
| MPI_MODE_WRONLY, MPI_INFO_NULL, &fp);
```

And to change the *file view* of each process using the derived `filetype`

```
MPI_File_set_view(fp, 0, MPI_DOUBLE, filetype,
"native", MPI_INFO_NULL);
```

(The second argument is an offset to the beginning of the file. We use `0` since we created `filetype` with `MPI_Type_create_subarray`).

Parallel reading and writing files

universidade de aveiro 

Now that each process ‘sees’ only its own part of the file, we can use the MPI’s convenient collective I/O operations for each process to write the desired data in the desired part of the file:

```
MPI_File_write_all(fp, myVnew, 1, memtype,
MPI_STATUS_IGNORE);
```

Notice that to this function only the local memory subarray `memtype` is passed explicitly. The subarray `filetype` was already used to define the *file view* when we used `MPI_File_set_view()`, and is held by `fp`.

Lets not forget to close the file:

```
MPI_File_close(&fp);
```

Bibliography

universidade de aveiro 

Using MPI: portable parallel programming with the message-passing interface, 3rd edition, William Gropp, Ewing Lusk, and Anthony Skjellum, MIT press (2014).

Using Advanced MPI: Modern Features of the Message-Passing Interface. William Gropp, Torsten Hoefler, Rajeev Thakur, and Ewing Lusk. MIT Press, Cambridge, MA, (2014).