Notes on Internode MPI implementation in `moment_kinetics`

Michael Hardman @ Tokamak Energy

November 22, 2022

Domain decomposition for 2D MPI in `Julia`

In `moment_kinetics` (`https://github.com/mabarnes/moment_kinetics`) we plan to implement internode MPI using the `MPI.jl` package Byrne et al. [2021]. To develop the right methods of the implementation we have developed 1D and 2D tests scripts `run_MPI_test.jl` and `run_MPI_test2D.jl`, respectively. These may be found in the branch `https://github.com/mabarnes/moment_kinetics/tree/radial-vperp-standard-DKE-Julia-1.7.2-mpi`.

The very basic MPI command creates a communicator and assigns a unique identifier to each process, e.g.,

```
MPI.Init()
comm = MPI.COMM_WORLD
nrank = MPI.Comm_size(comm) # number of ranks
irank = MPI.Comm_rank(comm) # rank of this process
```

where `nrank` and `irank` are the number of processes and the identifier, respectively. The integer `irank` varies from 0 to `nrank-1`. The task now is to organise calculations on each of these processes, and transfer data between them, to allow for larger and faster simulations.

Our problem to to parallelise a spectral element (x, y) grid. We imagine that we can split up the grid into distinct regions, which are stored on separate processes. See figure 1 for a diagram. We must couple data in these different regions together in the calculation, and so we must have the tools to transfer data between these regions. This is facilitated by various standard MPI commands. To make our job easier using those commands, it is conventional to split the 'communicator' that contains all the `nrank` processes into useful subsets. This is carried out with the following command

```
    MPI.Comm_split(comm,color,key)
# comm -> communicator to be split
# color -> label of group of processes
# key -> label of process in group
```

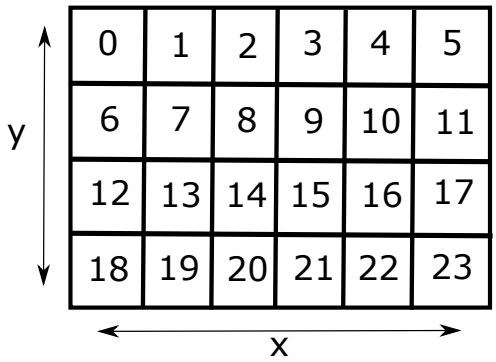

Figure 1: An (x, y) domain split up into `nrank`=24 separate regions, with `irank` indicated for each region.

To specify the `color` and `key` variables, we identity 'blocks' in the domain that can be usefully aggregated together. In figures 2 and 3 we illustrate how we define 'blocks' of processes: a block is a domain that must be connected to carry out a physical operation such as differentiation. The (x, y) grid naturally requires a row and column index to

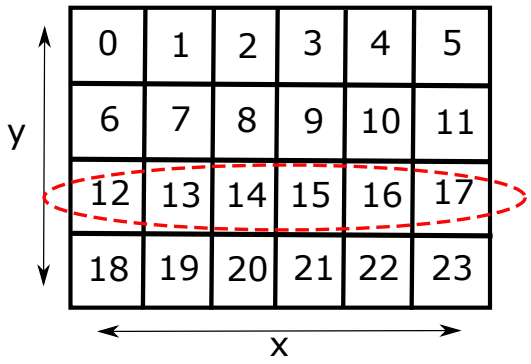

Figure 2: The (x, y) domain with a 'block' of x points highlighted in red. Here, we define a 'block' to be a domain in x at a given range of y. In this diagram, there are 4 x blocks.

label the processes that belong in x or y 'blocks'. To define these indices, we require to know the number of 'blocks' in each dimension. This is known once we specify the number of elements local to a process, and the number global to each domain. We define the following variables:

```
y_nblocks = floor(Int,x_nelement_global/x_nelement_local)
x_nblocks = floor(Int,y_nelement_global/y_nelement_local)
```

which then allow use to define row and column indices –

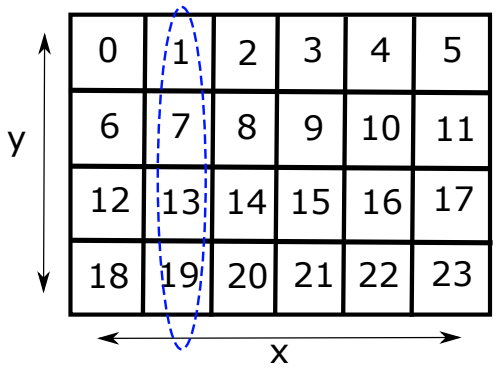

Figure 3: The (x, y) domain with a 'block' of y points highlighted in blue. Here, we define a 'block' to be a domain in y with a given range of y. In this diagram, there are 6 yy blocks.

```
y_nrank_per_block = floor(Int,nrank/y_nblocks)
# column index
y_iblock = mod(irank,y_nblocks) # irank - > y_iblock
# row index
y_irank_sub = floor(Int,irank/y_nblocks) # irank -> y_irank_sub
# to get the irank use:
# irank = y_iblock + x_nrank_per_block * y_irank_sub

x_nrank_per_block = floor(Int,nrank/x_nblocks)
# row index
x_iblock = y_irank_sub # irank - > x_iblock
# column index
x_irank_sub = y_iblock # irank -> x_irank_sub
# to get the irank use:
# irank = x_iblock * x_nrank_per_block + x_irank_sub
```

Once these indices are defined, we have the `color` and `key` variables to define the appropriate communicators for operators along the x and y domains.

```
y_comm_sub = MPI.Comm_split(comm,y_iblock,y_irank_sub)
x_comm_sub = MPI.Comm_split(comm,x_iblock,x_irank_sub)
```

Note that we can easily switch the definitions of x and y to reverse the parallelisation. The is acheived by instead defining the row and column indices in the following way.

```
# column index
y_iblock = floor(Int,irank/y_nblocks) # irank -> y_iblock
# row index
```

```
y_irank_sub = mod(irank,y_nblocks) # irank - > y_irank_sub

x_nrank_per_block = floor(Int,nrank/x_nblocks)
# row index
x_iblock =y_iblock # irank - > x_iblock
# column index
x_irank_sub =  y_irank_sub  # irank -> x_irank_sub
```

Bibliography

Simon Byrne, Lucas C. Wilcox, and Valentin Churavy. Mpi.jl: Julia bindings for the message passing interface. *Proceedings of the JuliaCon Conferences*, 1(1):68, 2021. doi: 10.21105/jcon.00068. URL https://doi.org/10.21105/jcon.00068.