NHR FAU

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Practical parallel algorithms with MPI Tutorial

Erlangen National High Performance Computing Center

Department of Computer Science

FAU Erlangen-Nürnberg

WS 25/26

Solution assignment 4: Poisson solver

- This is one solution. There are other good (or better) solutions!
- Exchange ghost layer at start of Poisson solver iteration loop:

```
MPI_Request requests[4] = { MPI_REQUEST_NULL,…, MPI_REQUEST_NULL };

if (solver->rank + 1 < solver->size) { /* exchange ghost cells with top neighbor */
   int top = solver->rank + 1;
   double* src = solver->p + (solver->jmaxLocal) * (solver->imax+2) + 1;
   double* dst = solver->p + (solver->jmaxLocal+1) * (solver->imax+2) + 1;
   MPI_Isend(src, solver->imax, MPI_DOUBLE, top, 1, MPI_COMM_WORLD, &requests[0]);
   MPI_Irecv(dst, solver->imax, MPI_DOUBLE, top, 2, MPI_COMM_WORLD, &requests[1]);
}

if (solver->rank > 0) { /* exchange ghost cells with bottom neighbor */
   int bottom = solver->rank - 1;
   double* src = solver->p + (solver->imax+2) + 1;
   double* dst = solver->p + 1;
   MPI_Isend(src,  solver->imax, MPI_DOUBLE, bottom, 2, MPI_COMM_WORLD, &requests[2]);
   MPI_Irecv(dst,  solver->imax, MPI_DOUBLE, bottom, 1, MPI_COMM_WORLD, &requests[3]);
}
MPI_Waitall(4, requests, MPI_STATUSES_IGNORE);
```

Solution assignment 4: Poisson solver

- Collect result from all processes:

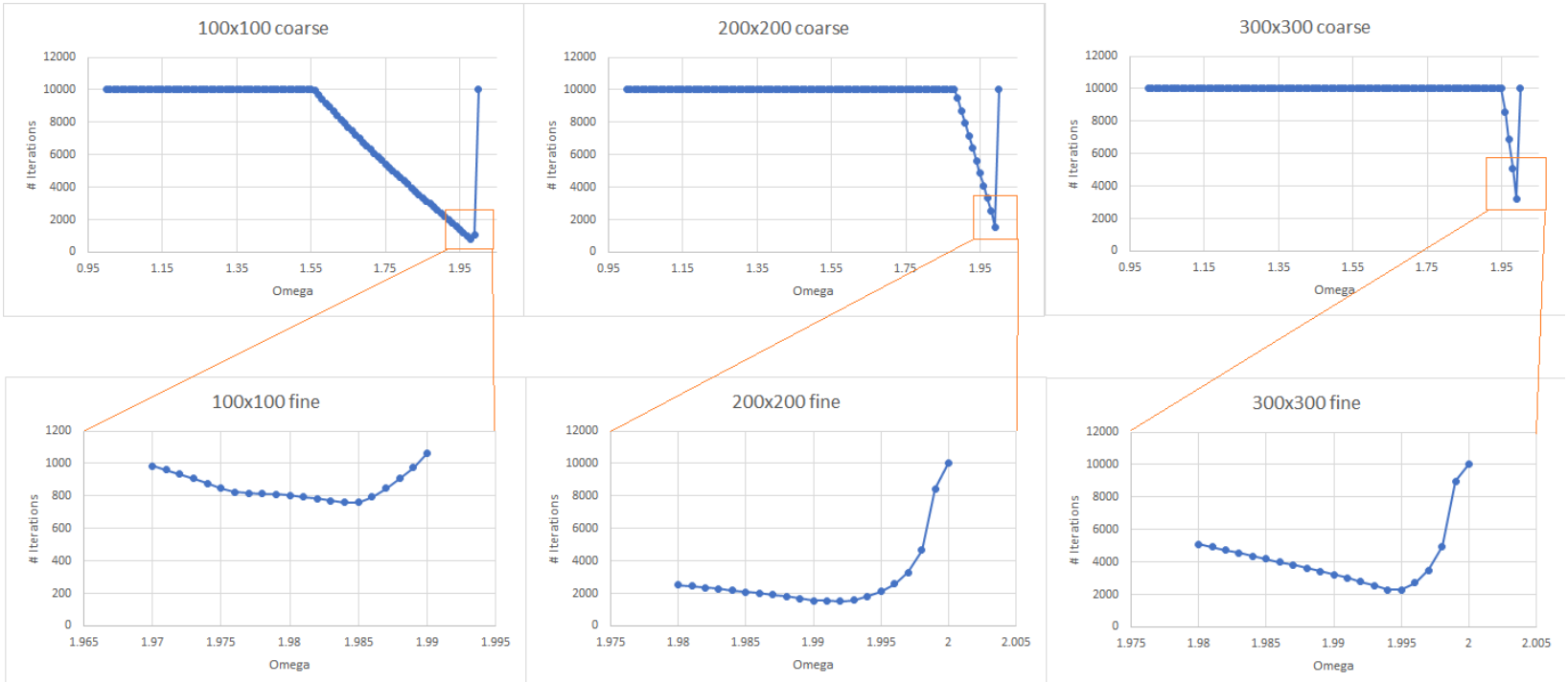
```
double* Pall = NULL; int *rcvCounts, *displs;

if ( solver->rank == 0 ) {
  Pall = allocate(64, (solver->imax+2) * (solver->jmax+2) * sizeof(double));
  rcvCounts = (int*) malloc(solver->size * sizeof(int));
  displs = (int*) malloc(solver->size * sizeof(int));
  rcvCounts[0] = solver->jmaxLocal * (solver->imax+2);
  displs[0] = 0; int cursor = rcvCounts[0];

  for ( int i=1; i < solver->size; i++ ) {
     rcvCounts[i] = sizeOfRank(i, solver->size, solver->jmax) * (solver->imax+2);
     displs[i] = cursor;
     cursor += rcvCounts[i];
  }
}
int cnt = solver->jmaxLocal*(solver->imax+2);
double* sendbuffer = solver->p + (solver->imax+2);
MPI_Gatherv(sendbuffer, cnt, MPI_DOUBLE, Pall,
   rcvCounts, displs, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Solution assignment 4: Poisson solver

- Relation of Omega and the domain size:



Solution assignment 4: Red Black SOR solver

- RBSOR with if condition:

```
#run for the desired number of iterations
repeat number of iterations times
exchange_ghost_cells();
# red half iteration
for(i=1 ; i < imaxLocal+1 ; ++i) {
          for(j=1 ; j < jmaxLocal+1 ; ++j) {
                    if ( (i+j) mod 2 == 0) {
                              P[i,j]=(P[i-1,j] + P[i+1,j] + P[i,j-1] + P[i;j+1] + RHS[i,j] )/4.0
                    }
          }
}
exchange_ghost_cells();
# black half iteration
for(i=1 ; i < imaxLocal+1 ; ++i) {
          for(j=1 ; j < jmaxLocal+1 ; ++j) {
                    if ( (i+j) mod 2 == 1) {
                              P[i,j]=(P[i-1,j] + P[i+1,j] + P[i,j-1] + P[i,j+1] + RHS[i,j] )/4.0
                    }
          }
}
end repeat
```

Solution assignment 4: Red Black SOR solver

- RBSOR without if condition:

```
#run for the desired number of iterations
repeat number of iterations times
isw = 1;

for (pass = 0; pass < 2; pass++)
{
            jsw = isw;
            exchange_ghost_cells();
            # red half iteration if isw = 1 and black half iteration if isw = 2
            for(i=1 ; i < imaxLocal+1 ; ++i)
            {
                        for(j=jsw ; j < jmaxLocal+1 ; j+=2)
                        {
                                    P[i,j]=(P[i-1,j] + P[i+1,j] + P[i,j-1] + P[i,j+1] + RHS[i,j] )/4.0
                        }
                        jsw = 3 - jsw;
            }
            isw = 3 - isw;
}
end repeat
```

Normal GS SOR solver vs RedBlack GS SOR solver

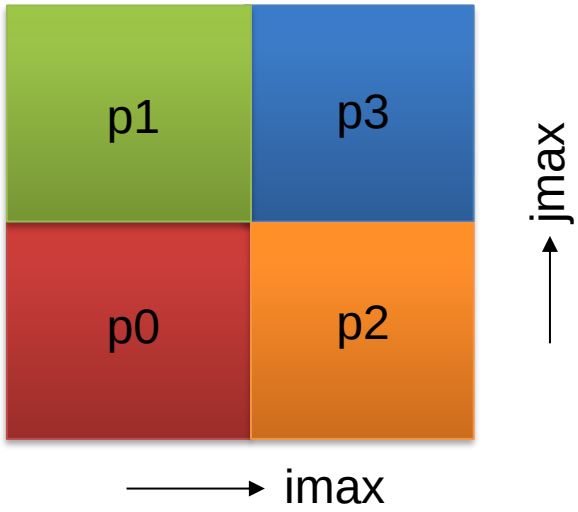
- RB GS solver converges where normal Gauss Seidel SOR diverges.
- Example of 200 (imax) x 200 (jmax):
  - Normal GS SOR:
    - **$ mpirun -n 25 ./exe-ICC poisson.par**
    - Solver took 10000 **iterations** to reach 6988478020553852587077688109354651958813429262637464799638698104834501689801215619359954161332160324089917992790457524944896 0.000000 **residual** using **omega**=1.800000
    - Solver took 10000 **iterations** to reach 0.726767 **residual** using **omega**=1.400000
  - RB SOR:
    - **$ mpirun -n 25 ./exe-ICC poisson.par**
    - Solver took 10000 **iterations** to reach 0.001025 **residual** using **omega**=1.800000

Normal GS SOR diverges with omega=1.8, but converges with omega=1.4

RB GS SOR converges with omega=1.8,

Assignment 5: Basic CFD solver

- Domain decomposition:

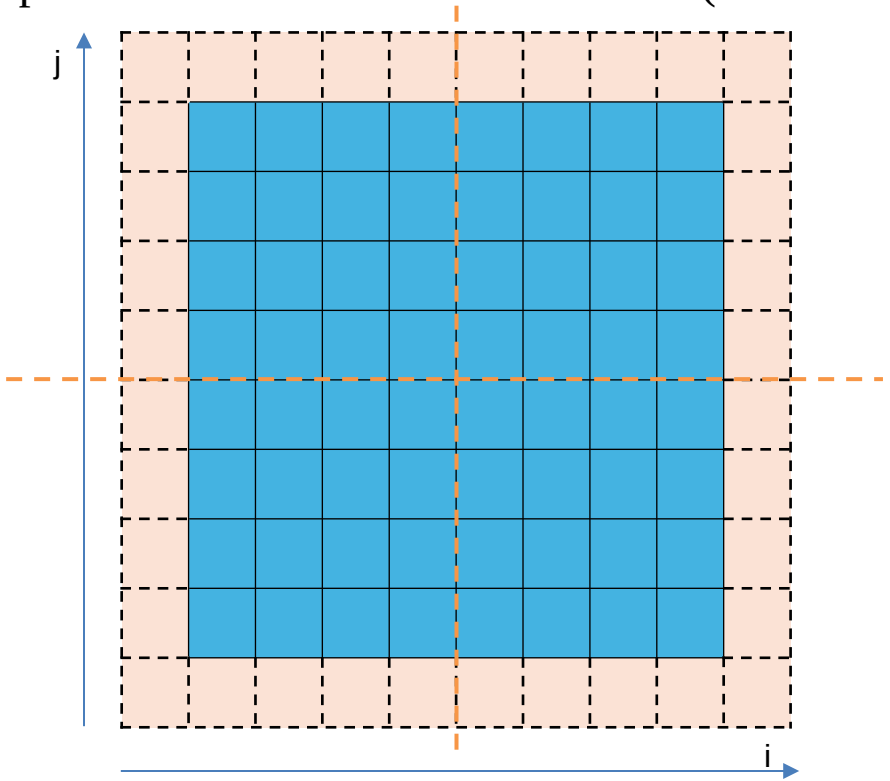


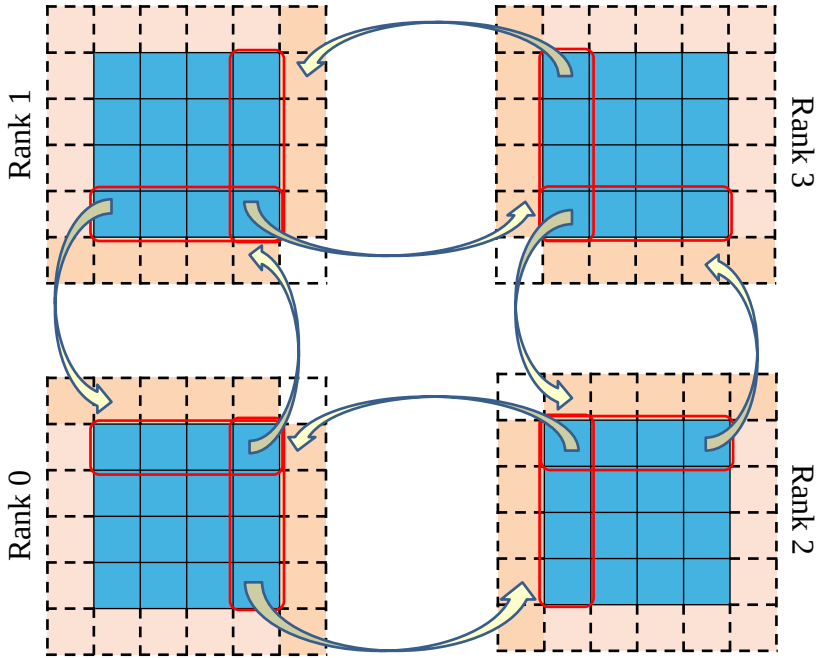
Basic strategy

- How should the arrays be distributed across processes?
- Which data/information needs to be communicated?
- How to structure the MPI code: Separate or merged into the sequential code?
- Plan treatment of staggered grid

2D Domain decomposition with ghost cell exchange

Example : Mesh size : 8x8 (10x10 with ghost cells) with 2D domain decomposition by 4 processes. Local domain size : 4x4 (6x6 with ghost cells)

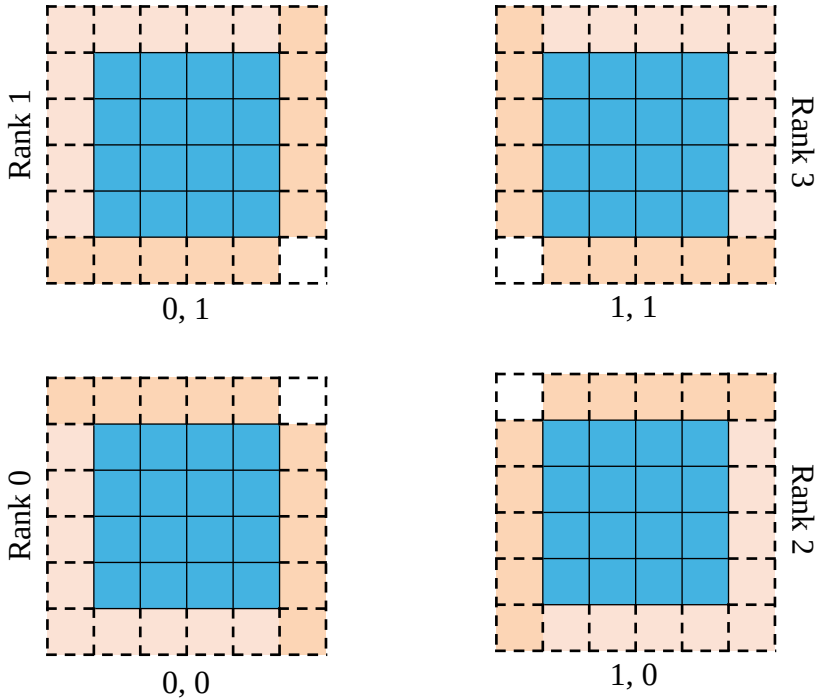




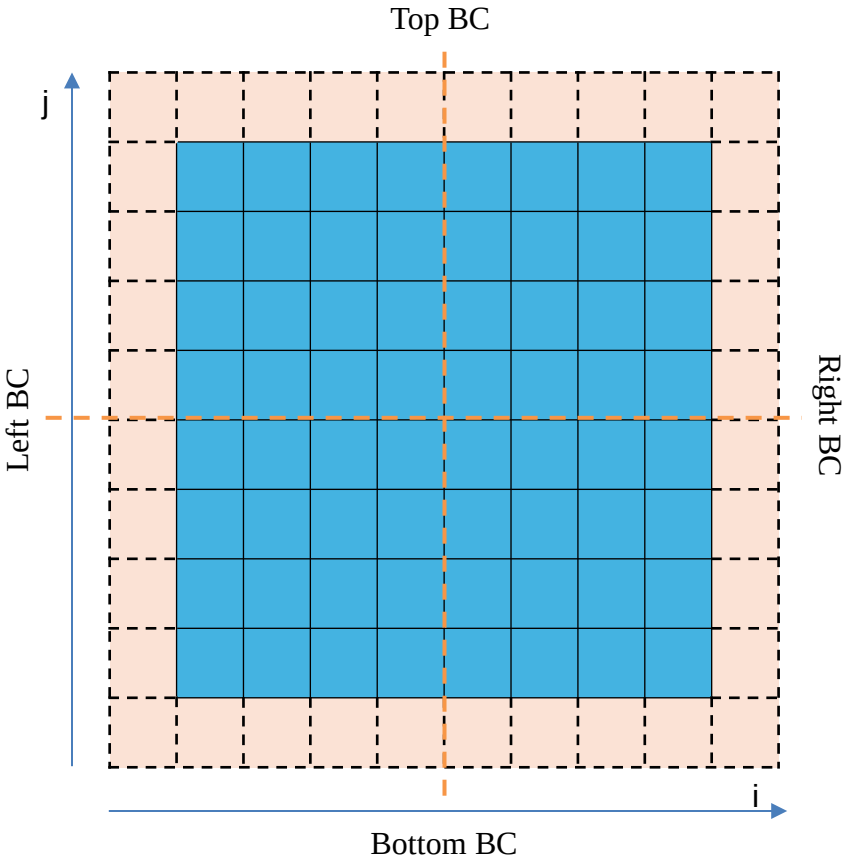
Exchange for **U**, **V** and **P** arrays

Acquire cartesian coordinates

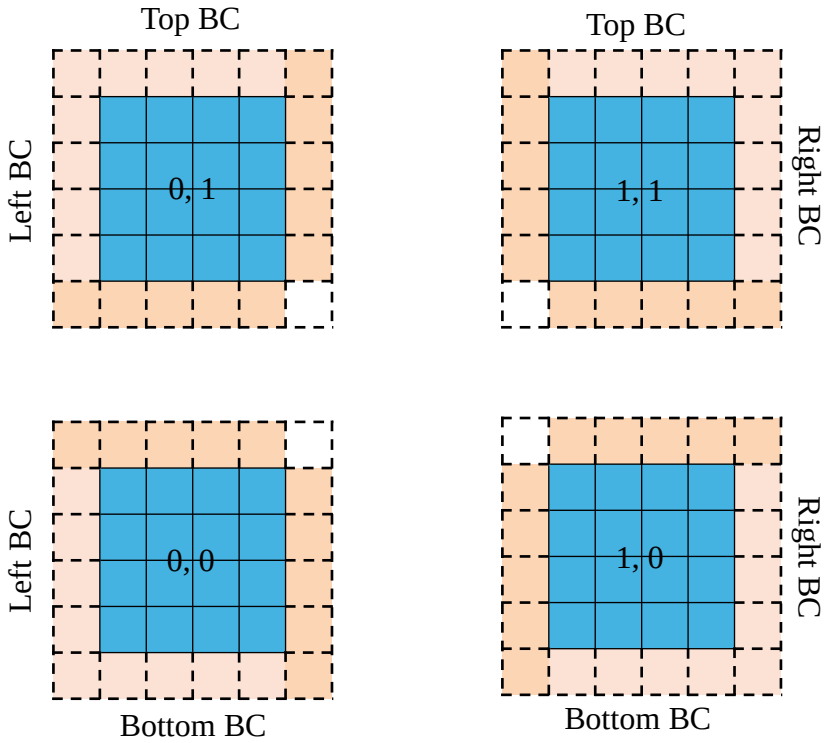
- Use virtual topologies to create a new communicator with dimensions and cartesian coordinates for the ranks.
- You also will store the neighbours directly using MPI_Cart_shift(). neighbours[IDIM] and neighbour[JDIM] will contain neighbours in I and J dimension.
- coord[IDIM] and coord[JDIM] will contain cartesian coordinates in I and J dimension, assigned by the MPI_Cart_get().
- You will be using cartesian coordinates to determine which boundary does the rank have (TOP/BOTTOM/LEFT/RIGHT).



Boundary conditions

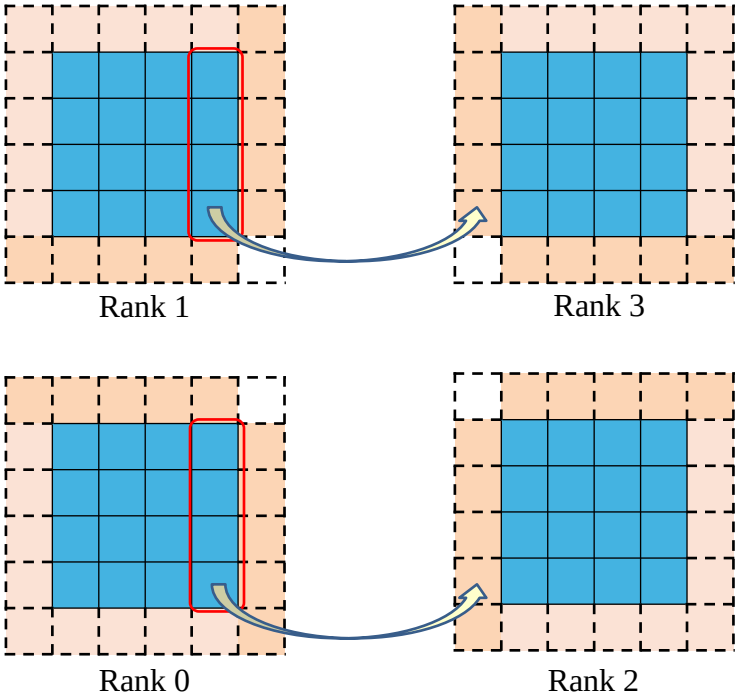


Using isBoundary() and coord[], you will be able to check boundaries for the ranks

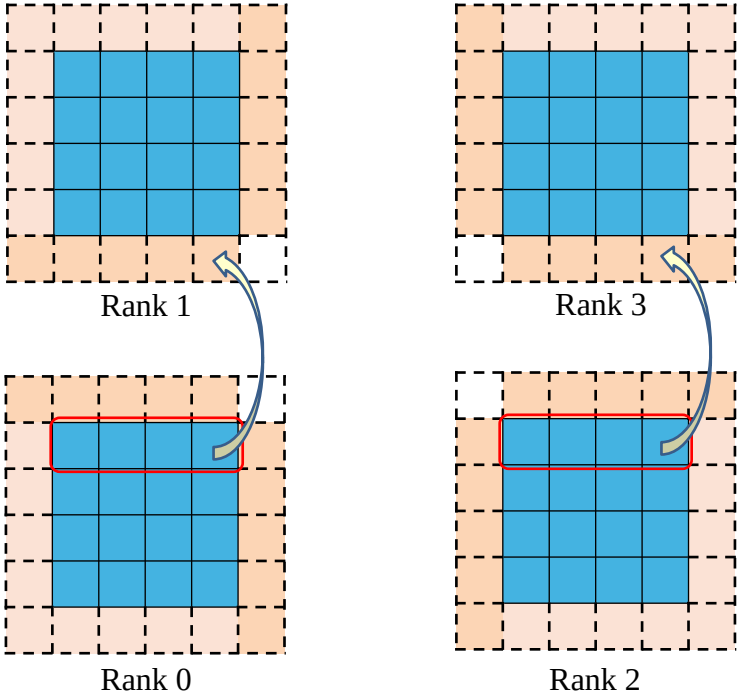


Shift function

For the array **F**, you just have to do ghost cell exchange from left neighbour to the right neighbour.

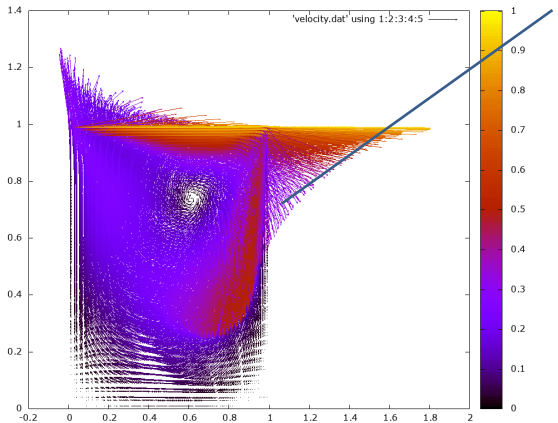


For the array **G**, you just have to do ghost cell exchange from bottom neighbour to the top neighbour.



Test case lid driven cavity

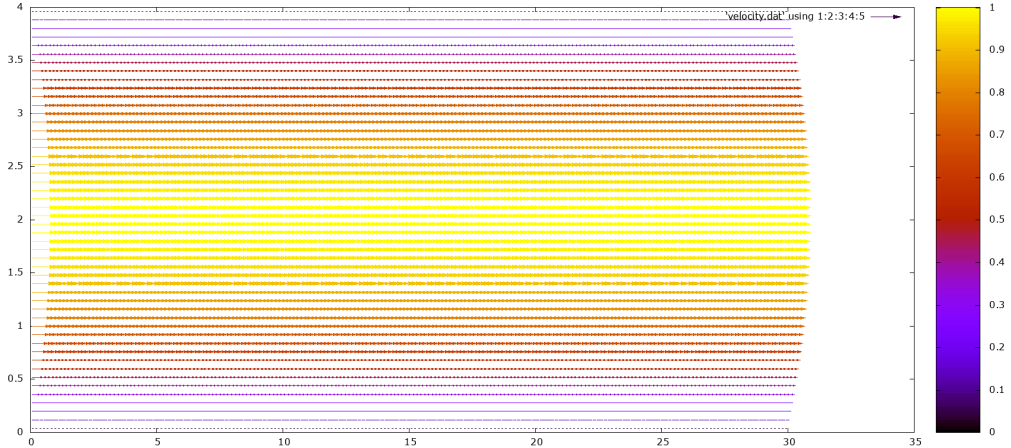
- Parameters you can play with:
  - Reynolds number: **`re`** **`100.0`** possible range: 10-1000
  - Grid resolution: **`imax, jmax`** You may try between 40x40 to 200x200
  - End time: **`te`** **`10.0`** Try between 4s and 20 s
  - If you encounter instabilities with the solver you may want to play with **`omg`** and **`gamma`** parameters.



Feel free to improve the gnuplot script or use random vector placement

Test case laminar channel flow

- Parameters you can play with:
  - End time: `te 10.0` Try between 10s and 100s
  - If you encounter instabilities with the solver you may want to play with `omg` parameters.
  - For validation compare the time step size with the sequential solver



Hints and help

- During the start up of the application, check from the information if the neighbours and the boundaries of the rank are correct.
- This will help you check the correctness of your virtual topology implementation.
- See if you are not iterating over the local domain size of
  - (imaxLocal + 2) * (jmaxLocal + 2)
- You may see randomness if you access an array over local domain size.
- Use printExchange() and printShift() methods to test your exchange() and shift() implementation.
- Check if you have correctly implemented the boundary conditions according to the applicable boundaries for the respective rank.
- Adapt the setSpecialBoundaryCondition() method according to the ranks.