HIGH PERFORMANCE COMPUTING for SCIENCE & ENGINEERING (HPCSE) I

HS 2021

EXERCISE 06: MPI

Man Hin CHENG

Computational **S**cience and **E**ngineering **Lab**
ETH Zürich

10.12.2021

Outline

- Question 1 MPI Bug Hunt
- Question 2 MPI Reduction implementation
- Question 3 MPI non blocking

Compile and execute in Euler:

```
module load gcc/8.2.0 openmpi/4.0.2
mpirun -n N ./executable
```

Question 1: MPI Bug Hunt

- Find the bug(s) in the following MPI code snippets and find a way to fix the problem. Assume all the headers are imported correctly.

a)

```
const int N = 10000;
double* result = new double[N];
// do a very computationally expensive calculation
// ...

// write the result to a file
std::ofstream file("result.txt");

for(int i = 0; i <= N; ++i){
    file << result[i] << std::endl;
}

delete[] result;
```

b)

```
// only 2 ranks: 0, 1
double important_value;

// obtain the important value
// ...

// exchange the value
if(rank == 0)
    MPI_Send(&important_value, 1, MPI_DOUBLE, 1, 123, MPI_COMM_WORLD);
else
    MPI_Send(&important_value, 1, MPI_DOUBLE, 0, 123, MPI_COMM_WORLD);

MPI_Recv(
    &important_value, 1, MPI_INT, MPI_ANY_SOURCE,
    MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE
);

// do other work
```

Question 1: MPI Bug Hunt

- c) What is the output of the following program when run with 1 rank? What if there are 2 ranks? Will the program complete for any number of ranks?

```
1  MPI_Init(&argc, &argv);
2
3  int rank, size;
4  MPI_Comm_size(MPI_COMM_WORLD, &size);
5  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
6
7  int bval;
8  if (0 == rank)
9  {
10     bval = rank;
11     MPI_Bcast(&bval, 1, MPI_INT, 0, MPI_COMM_WORLD);
12 }
13 else
14 {
15     MPI_Status stat;
16     MPI_Recv(&bval, 1, MPI_INT, 0, rank, MPI_COMM_WORLD, &stat);
17 }
18
19 cout << "[" << rank << "] " << bval << endl;
20
21 MPI_Finalize();
22 return 0;
```

Question 2: Implement a distributed reduction

$$x_{tot} = \sum_{n=1}^{N} n = 1+2+3+\ldots+(N-1)+N$$

- a) Fill in the missing part in the Makefile in order to compile the code with MPI support

```
# TODO a): Set the compiler
MPICXX=
CXXFLAGS+=-std=c++11 -Wall -Wpedantic -O3
```

- b) Validate with exact solution

```
inline long exact(const long N){
    // TODO b): Implement the analytical solution.
    return 0;
}
```

Question 2: Implement a distributed reduction

$$x_{tot} = \sum_{n=1}^{N} n = 1 + 2 + 3 + \ldots + (N-1) + N$$

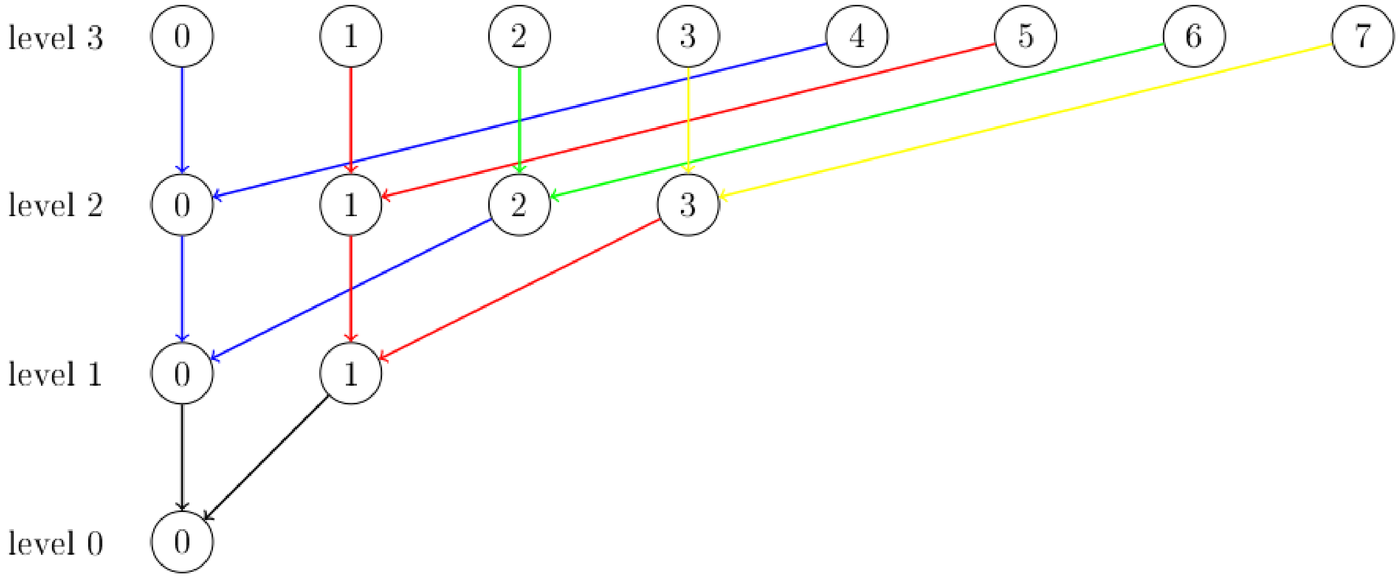
- c) Initialize and finalize MPI
- d) Distribute work with reasonable load balance

$$x_{rank} = N_{start} + (N_{start} + 1) + \ldots + N_{end}$$

```cpp
int main(int argc, char** argv){
    const long N = 1000000;

    // Initialize MPI
    int rank, size;
    // TODO c): Initialize MPI and obtain the rank and the number of processes (size)

    // ------------------------
    // Perform the local sum:
    // ------------------------
    long sum = 0;

    // Determine work load per rank
    long N_per_rank = N / size;

    // TODO d): Determine the range of the subsum that should be calculated by this rank.
    long N_start;
    long N_end;

    // N_start + (N_start+1) + ... + (N_start+N_per_rank-1)
    for(long i = N_start; i <= N_end; ++i){
        sum += i;
    }

    // ------------------------
    // Reduction
    // ------------------------
    reduce_mpi(rank, sum);
    //reduce_manual(rank, size, sum);

    // ------------------------
    // Print the result
    // ------------------------
    if(rank == 0){
        std::cout << std::left << std::setw(25) << "Final result (exact): " << exact(N) << std::endl;
        std::cout << std::left << std::setw(25) << "Final result (MPI): " << sum << std::endl;
    }
    // Finalize MPI
    // TODO c): Finalize MPI

    return 0;
}
```

Question 2: Implement a distributed reduction

$$x_{tot} = \sum_{n=1}^{N} n = 1+2+3+\ldots+(N-1)+N$$



- e) Implement reduction manually and using MPI blocking collective

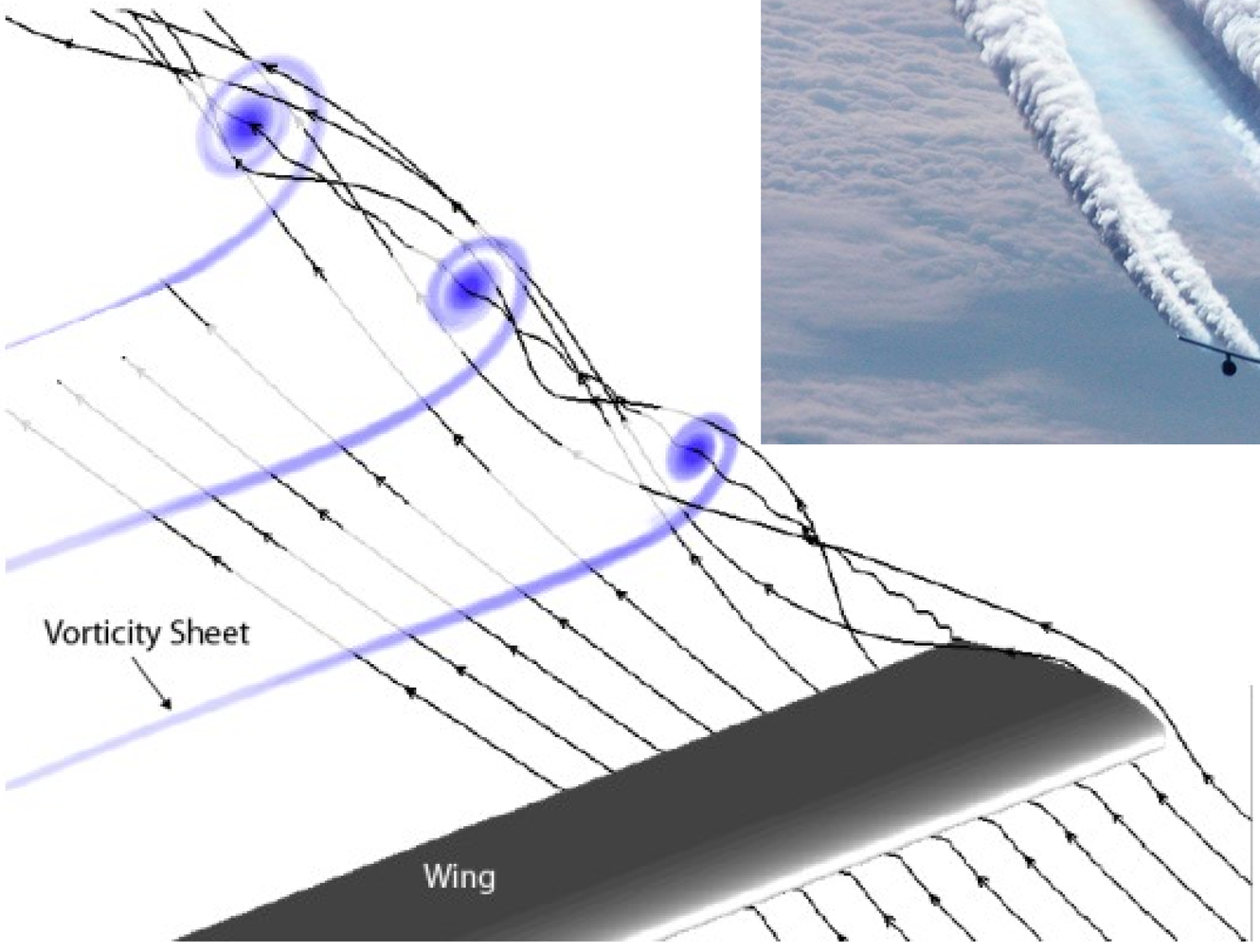
  (|ranks| = 2^n)

- f) Name 2 advantages with justification for reduction approach compare to naive approach

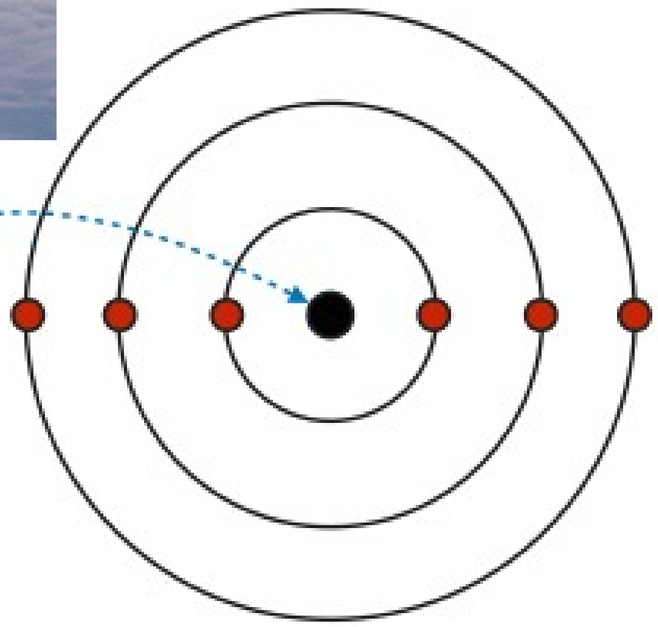
```
void reduce_mpi(const int rank, long& sum){
    // TODO e): Perform the reduction using blocking collectives.
}

// PRE: size is a power of 2 for simplicity
void reduce_manual(int rank, int size, long& sum){
    // TODO e): Implement a tree based reduction using blocking point-to-point communication.
}
```

Question 3: Roll-up a vortex line

- Objective: simulate the evolution of a vorticity sheet



$\Gamma_\nu > 0$



Question 3: Roll-up a vortex line

- Objective: simulate the evolution of a vorticity sheet
- Basic fluid mechanics
  - Vorticity ω: measure the “spinning motion” of a continuum near a point
  - Circulation Γ: measure the amount of vorticity contained in a volume

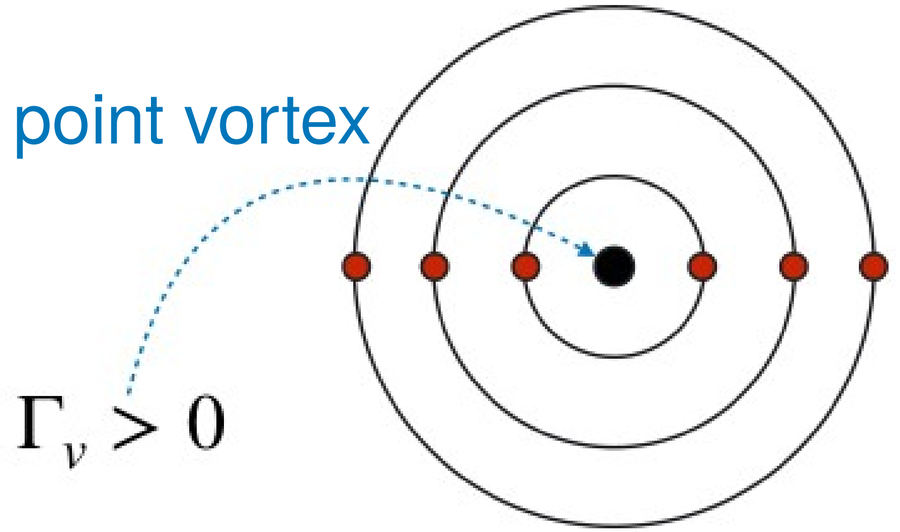
$$\Gamma_v = \int_V d\vec{x}\,\omega(\vec{x})$$

- Example: point vortex located at $\mathbf{x}_v$:

  $\mathbf{u} = (u_x, u_y)$ is the velocity field caused by the point vortex

$$u_x(x,y,t) = \frac{\Gamma_v}{2\pi}\frac{-[y-y_v]}{[x-x_v]^2+[y-y_v]^2}$$

$$u_y(x,y,t) = \frac{\Gamma_v}{2\pi}\frac{[x-x_v]}{[x-x_v]^2+[y-y_v]^2}$$

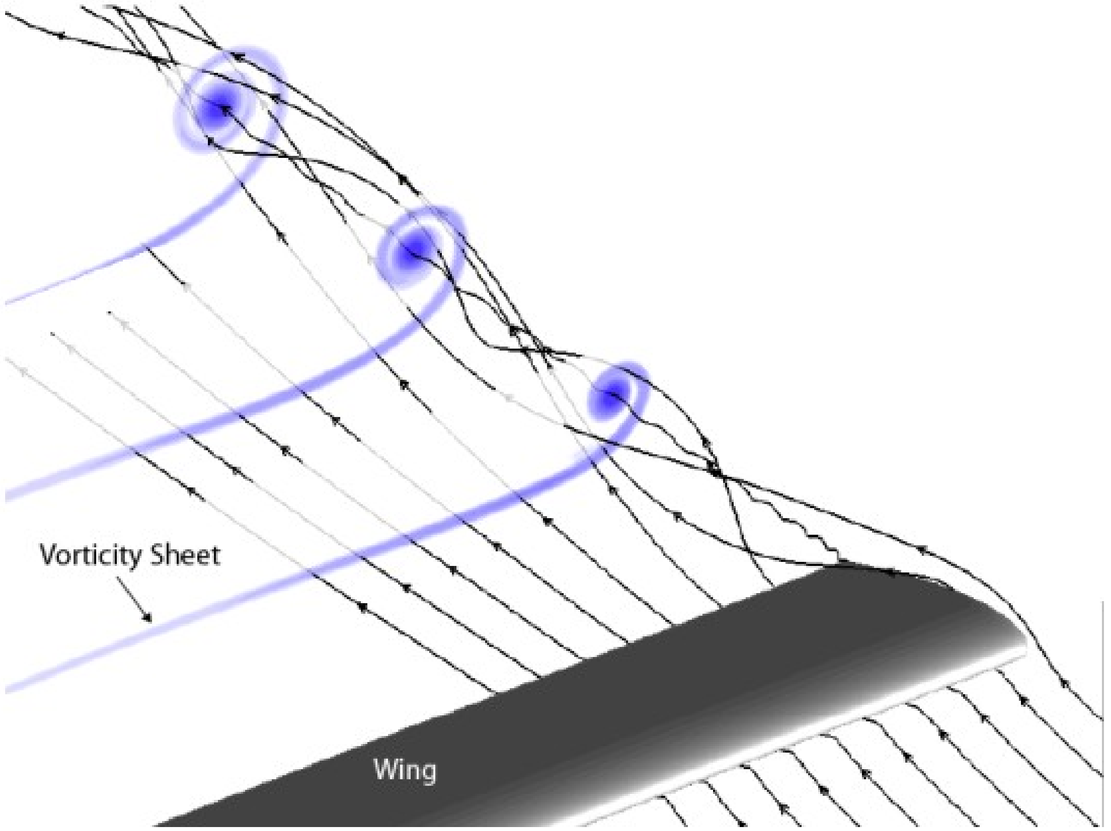


Question 3: Roll-up a vortex line

- Discretize a flow field with particles
  - A sufficient spatial distribution of particles
  - Each carrying a varying amount of vorticity

$$u_x(x_j, y_j, t) = \sum_{i=0}^{N} \frac{\Gamma_i}{2\pi} \frac{-[y_j - y_i(t)]}{[x_j - x_i(t)]^2 + [y_j - y_i(t)]^2}$$

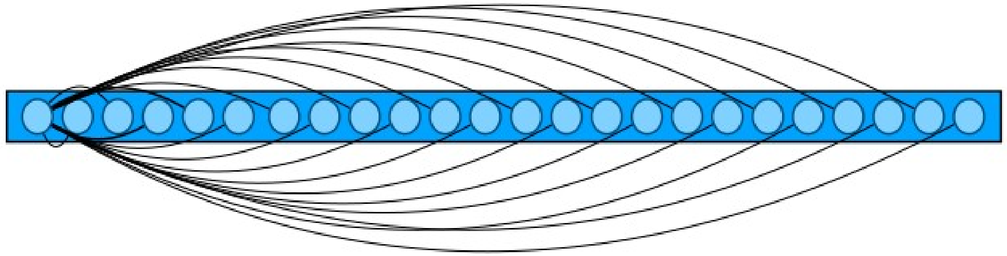
$$u_y(x_j, y_j, t) = \sum_{i=0}^{N} \frac{\Gamma_i}{2\pi} \frac{[x_j - x_i(t)]}{[x_j - x_i(t)]^2 + [y_j - y_i(t)]^2}$$



  - The evolution of the flow (assuming no dissipation) can be described by advecting the particles:

$$\frac{d}{dt}\vec{x}_i = \vec{u}(x, y, t)$$

Question 3: Roll-up a vortex line



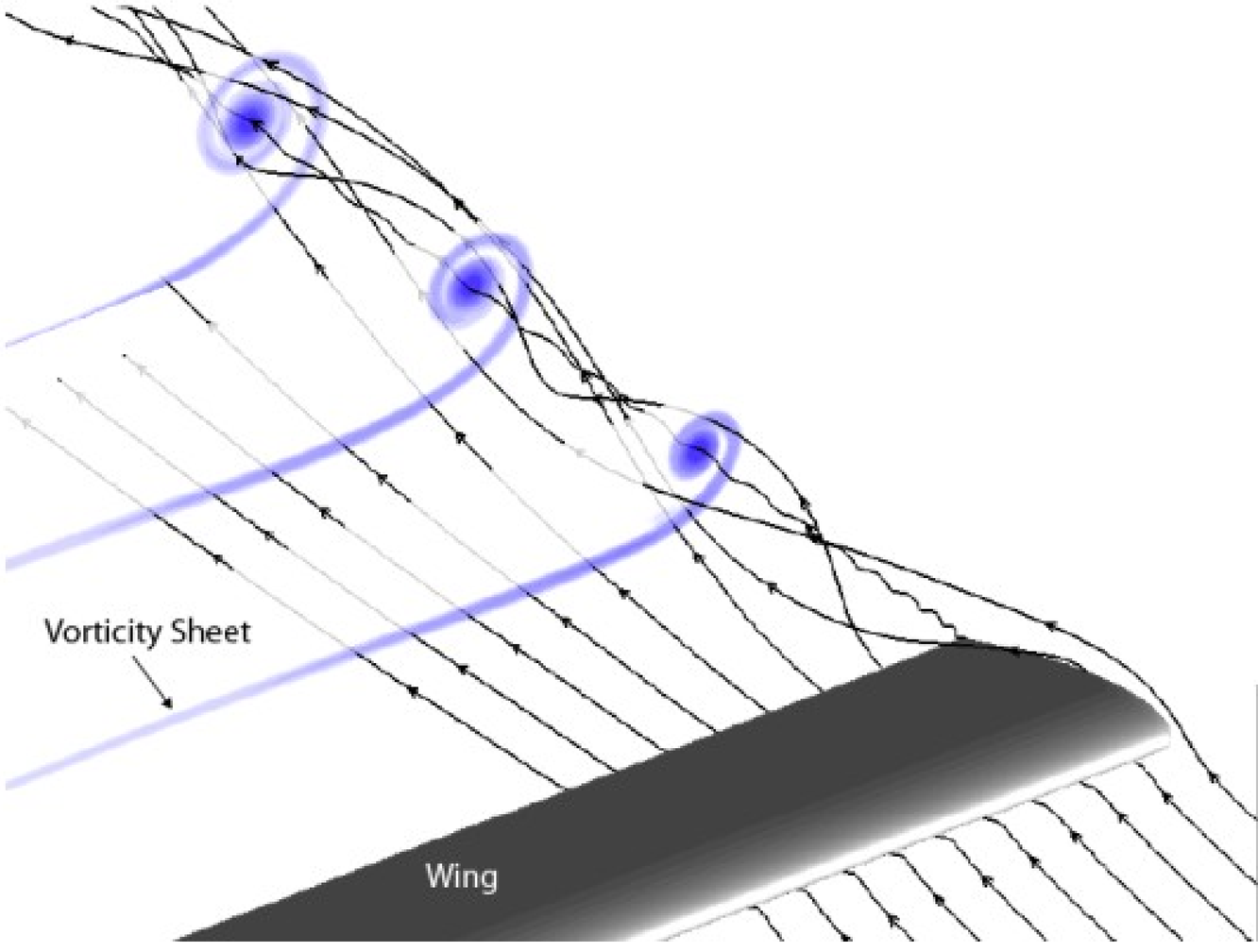
- N particles are initialized:
  - Uniformly arranged along the x-axis from -0.5 to 0.5

$$(x_i, y_i) = \left(-\frac{1}{2} + i\frac{1}{N}, 0 \right)$$

  - With zero initial velocity
  - With circulation given by:

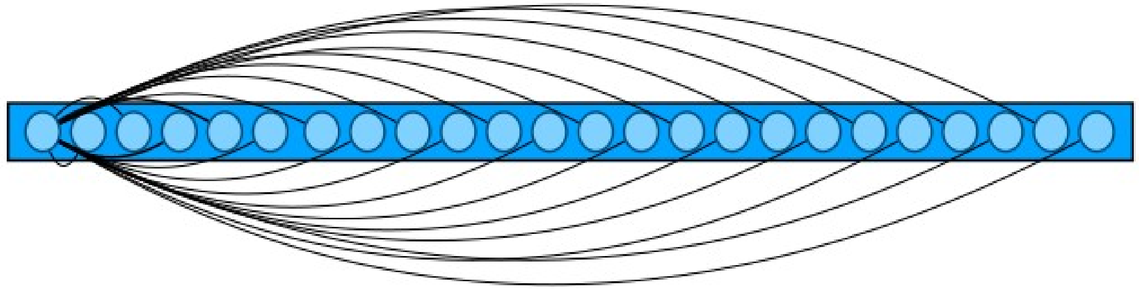
$$\Gamma_i = V_i \frac{4x}{\sqrt{1-4x^2}} = \frac{1}{N_i} \frac{4x}{\sqrt{1-4x^2}}$$

V_i (particle's volume)



Question 3: Roll-up a vortex line

- a) Implement serial version for computeVelocities
  - N^2 particle-based velocity solver
  - Each particle interacts with all the others

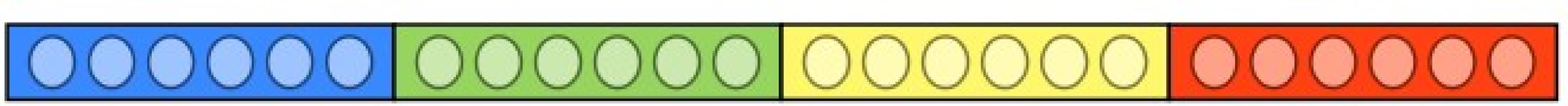


$$u_x(x_j, y_j, t) = \sum_{i=0}^{N} \frac{\Gamma_i}{2\pi} \frac{-[y_j - y_i(t)]}{[x_j - x_i(t)]^2 + [y_j - y_i(t)]^2}$$

$$u_y(x_j, y_j, t) = \sum_{i=0}^{N} \frac{\Gamma_i}{2\pi} \frac{[x_j - x_i(t)]}{[x_j - x_i(t)]^2 + [y_j - y_i(t)]^2}$$

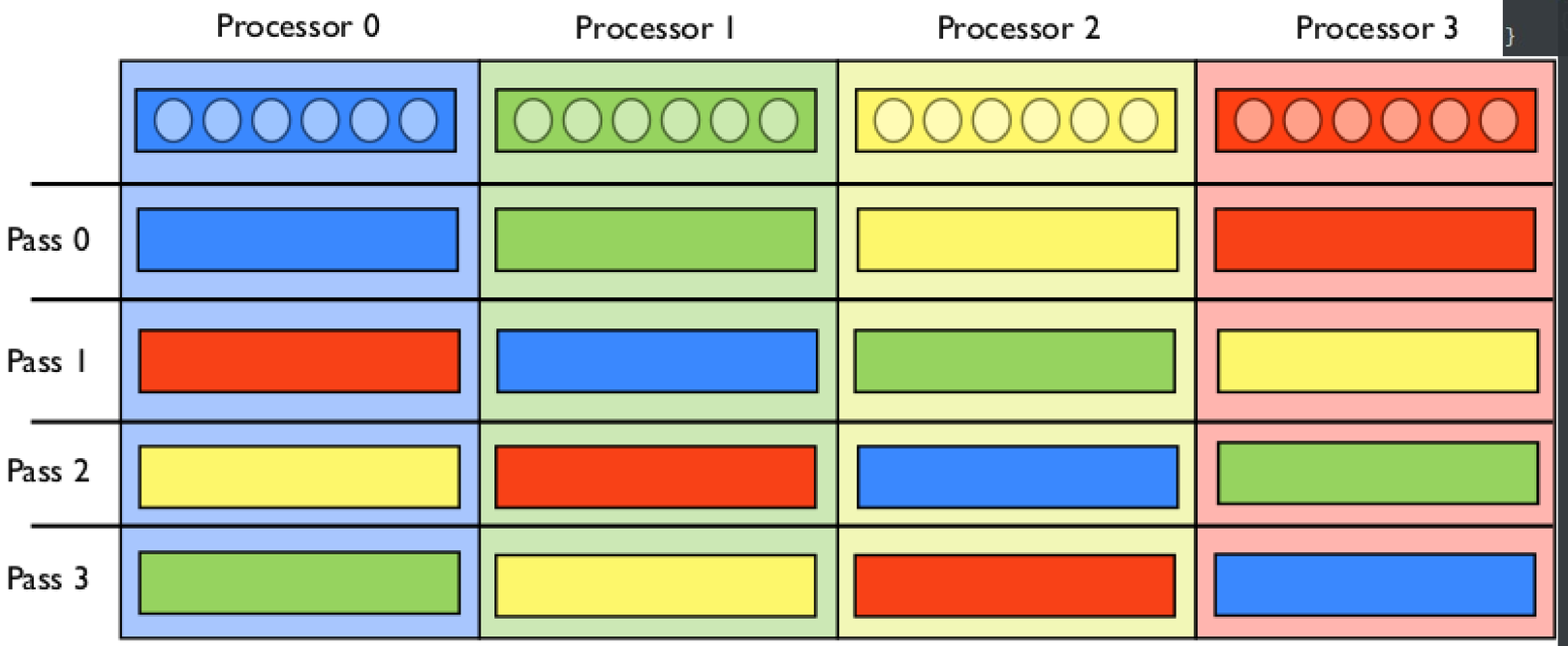
```
static void computeVelocities(double epsSq, const std::vector<double>& x,
                              const std::vector<double>& y,
                              const std::vector<double>& gamma,
                              std::vector<double>& u, std::vector<double>& v)
{
    // TODO compute the interactions between the particles and write the result
    // in the velocity vectors u and v
}
```

- b) Parallelize the code with MPI
  - Each processes need to exchange information to compute velocity



Question 3: Roll-up a vortex line

- b) Parallelize the code with MPI
  - Each processes need to exchange information to compute velocity
  - Processes need to exchange information to compute the velocity
  - Send simulation snapshots to rank 0 for output




```cpp
static void computeVelocities(MPI_Comm comm, double epsSq,
                              const std::vector<double>& x,
                              const std::vector<double>& y,
                              const std::vector<double>& gamma,
                              std::vector<double>& u, std::vector<double>& v)
{
    // TODO: perform multi pass to compute interactions and update the local
    // velocities.
}
```

```cpp
static void dumpToCsv(MPI_Comm comm, int step, const std::vector<double>& x,
                      const std::vector<double>& y,
                      const std::vector<double>& gamma)
{
    int rank, nranks;
    MPI_Comm_rank(comm, &rank);
    MPI_Comm_size(comm, &nranks);

    std::vector<double> xAll, yAll, gammaAll;

    // TODO Gather the data on rank zero before dumping to the csv files.

    if (rank == 0)
        dumpToCsv(step, xAll, yAll, gammaAll);
}
```

```cpp
    if (argc != 2) {
        fprintf(stderr, "usage: %s <total number of particles>\n", argv[0]);
        exit(1);
    }

    MPI_Comm comm = MPI_COMM_WORLD;
    int rank, nranks;
    MPI_Comm_rank(comm, &rank);
    MPI_Comm_size(comm, &nranks);

    const int nglobal = std::atoi(argv[1]);

    if (nglobal % nranks != 0) {
        fprintf(stderr,
                "expected n to be a multiple of the number of ranks.\n");
        exit(1);
    }

    // TODO initialize the data for each rank.

    const int n = nglobal;      // TODO
    const double extents = 1.0; // TODO

    const double startX = -0.5; // TODO
    const double endX = startX + extents;
```

Question 3: Roll-up a vortex line

- c) Parallelize the code with MPI for overlap communication
  - Use non blocking MPI operation to exchange information while computing the velocities of previous data