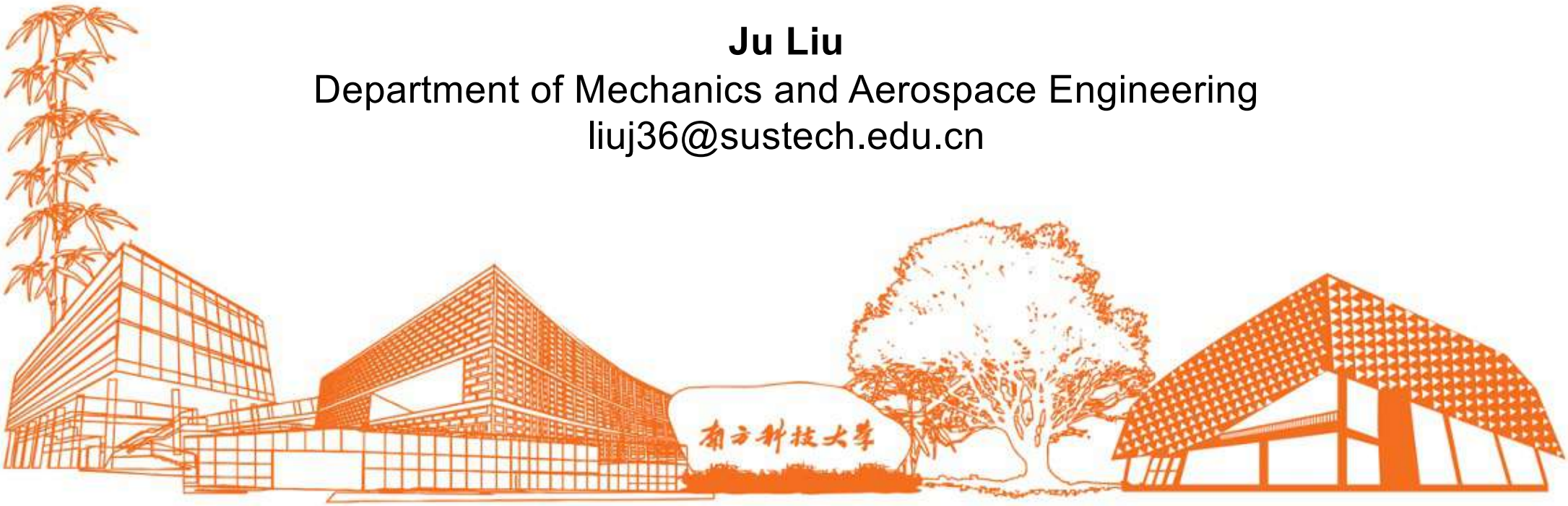
MAE 5032 High Performance Computing: Methods and Applications

Lab 8: MPI for n-body code

Ju Liu
Department of Mechanics and Aerospace Engineering
liuj36@sustech.edu.cn

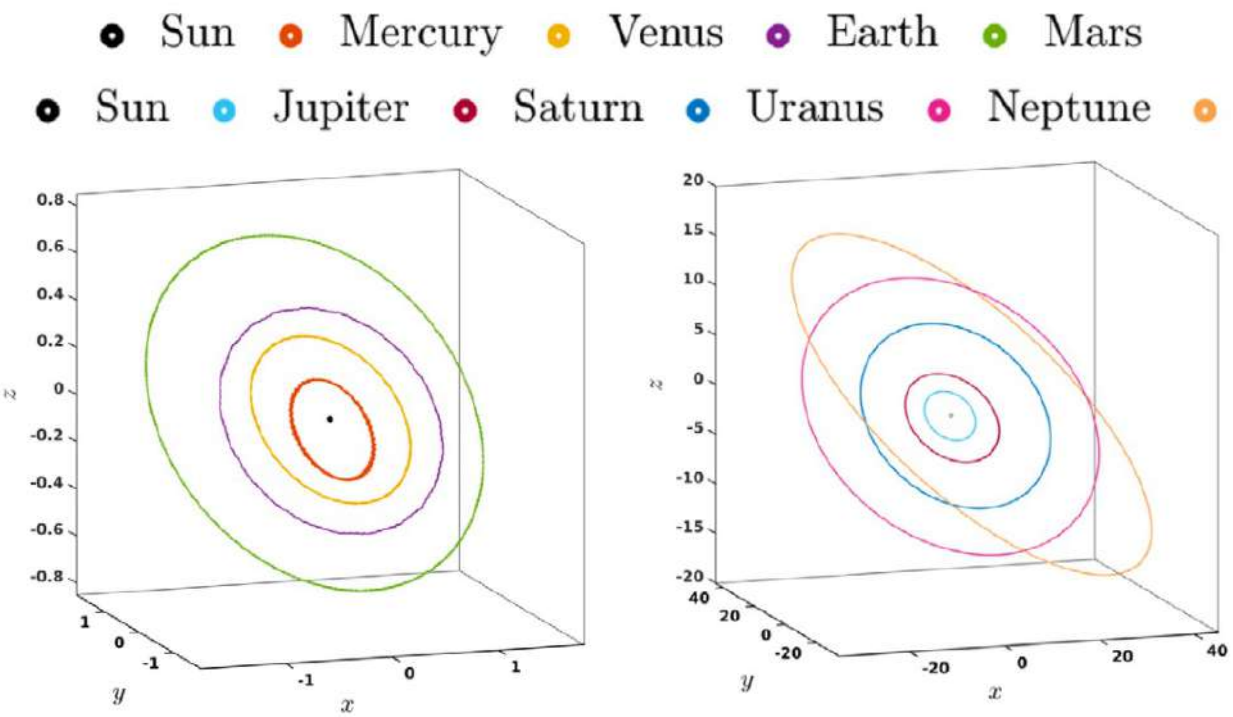


Objective

- understand the background of the n-body problem
- design a parallelism for the computation
- implement with MPI routines

Physical background

- The n-body problem describes the individual motion of a group of celestial objects interacting with each other gravitationally.



The gravitational force felt on mass of particle i by a single particle j:

$$\boldsymbol{F}_{ij} = \frac{G m_i m_j}{\|\boldsymbol{q}_i - \boldsymbol{q}_j\|^3} (\boldsymbol{q}_i - \boldsymbol{q}_j)$$

n-body equation of motion:

$$m_i \frac{d^2 \boldsymbol{q}_i}{dt^2} = \sum_{j=1 j \neq i}^{n} \boldsymbol{F}_{ij}$$

Serial code

- The n-body problem describes the individual motion of a group of celestial objects interacting with each other gravitationally.

We have a struct that contains all necessary info for a particle:

```
typedef struct {
  double x, y;
  double vx, vy;
  double mass;
} Particle;
```

The gravitational force felt on mass of particle i by a single particle j:

$$\boldsymbol{F}_{ij} = \frac{G m_i m_j}{\|\boldsymbol{q}_i - \boldsymbol{q}_j\|^3} (\boldsymbol{q}_i - \boldsymbol{q}_j)$$

n-body equation of motion:

$$m_i \frac{d^2 \boldsymbol{q}_i}{dt^2} = \sum_{j=1 j \neq i}^{n} \boldsymbol{F}_{ij}$$

Serial code

- The n-body problem describes the individual motion of a group of celestial objects interacting with each other gravitationally.

```
void compute_forces(Particle *p, double *fx, double *fy)
{
  for (int i = 0; i < N; ++i) {
    fx[i] = 0.0;
    fy[i] = 0.0;
    for (int j = 0; j < N; ++j) {
      if (i != j) {
        double dx = p[j].x - p[i].x;
        double dy = p[j].y - p[i].y;
        double dist_sqr = dx * dx + dy * dy + SOFTENING;
        double dist = sqrt(dist_sqr);
        double force = G * p[i].mass * p[j].mass / dist_sqr;
        fx[i] += force * dx / dist;
        fy[i] += force * dy / dist;
      }
    }
  }
}
```

The gravitational force felt on mass of particle i by a single particle j:

$$\boldsymbol{F}_{ij} = \frac{G m_i m_j}{\|\boldsymbol{q}_i - \boldsymbol{q}_j\|^3} (\boldsymbol{q}_i - \boldsymbol{q}_j)$$

n-body equation of motion:

$$m_i \frac{d^2 \boldsymbol{q}_i}{dt^2} = \sum_{j=1 j \neq i}^{n} \boldsymbol{F}_{ij}$$

Serial code

- The n-body problem describes the individual motion of a group of celestial objects interacting with each other gravitationally.

```
void update_particles(Particle *p, double *fx, double *fy)
{
  for (int i = 0; i < N; ++i) {
    p[i].vx += fx[i] / p[i].mass * DT;
    p[i].vy += fy[i] / p[i].mass * DT;
    p[i].x += p[i].vx * DT;
    p[i].y += p[i].vy * DT;
  }
}
```

The gravitational force felt on mass of particle i by a single particle j:

$$\boldsymbol{F}_{ij} = \frac{G m_i m_j}{\|\boldsymbol{q}_j - \boldsymbol{q}_i\|^3} (\boldsymbol{q}_j - \boldsymbol{q}_i)$$

n-body equation of motion:

$$m_i \frac{d^2 \boldsymbol{q}_i}{dt^2} = \sum_{j=1 j \neq i}^{n} \boldsymbol{F}_{ij}$$

Parallel design

- Observation: it is the force calculation that takes most of the computing time as it involves two for-loops for all particles.

- Idea: we may assign each processor a subset of the whole particles.

```
int base = N / size;
int extra = N % size;
int local_n = base + (rank < extra ? 1 : 0);
int start_idx = rank * base + (rank < extra ? rank : extra);

int *counts = (int *)malloc(size * sizeof(int));
int *displs = (int *)malloc(size * sizeof(int));
int offset = 0;
for (int i = 0; i < size; ++i) {
  counts[i] = base + (i < extra ? 1 : 0);
  displs[i] = offset * sizeof(Particle);
  offset += counts[i];
}
```

how much data each process sends

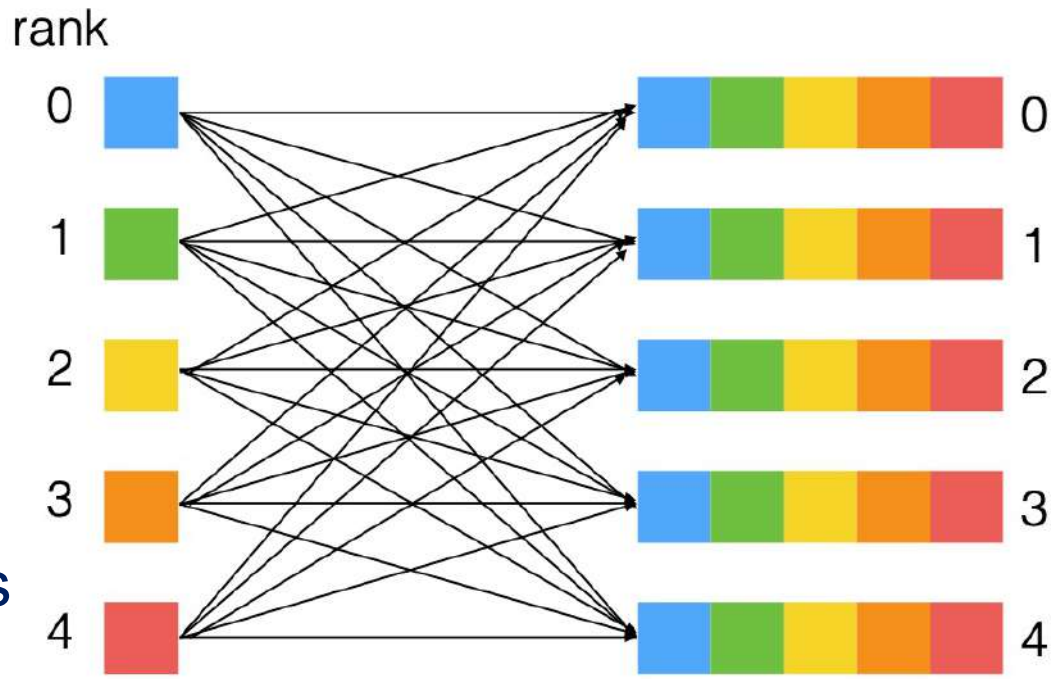
displacement in bytes for each process's data in the receive buffer

Parallel design

- Issue: we need the information of all particles to calculate the total force.

- Idea: we may use **AllGather** to update the physical states of all particles in all processors.

- Note: we need to use **AllGatherv** because each process may send different amounts of data.



Parallel design

seed for rand number generation

each processor randomize its particles locally

```
srand((unsigned int)time(NULL) + rank * 101);
init_particles_local(local_particles, start_idx, local_n);

MPI_Allgatherv(local_particles, local_n * sizeof(Particle), MPI_BYTE,
               all_particles, counts, displs, MPI_BYTE, MPI_COMM_WORLD);
```

use Allgatherv to update the particles info for all particles on all processes

Parallel design

compute the force for local particles

update the particle states for local particles

```
for (int step = 0; step < STEPS; ++step) {
  compute_forces(start_idx, local_n, all_particles, local_particles, fx, fy);
  update_particles(local_particles, fx, fy, local_n);

  MPI_Allgatherv(local_particles, local_n * sizeof(Particle), MPI_BYTE,
                 all_particles, counts, displs, MPI_BYTE, MPI_COMM_WORLD);
}
```

use Allgatherv to update the particles info for all particles on all processes

Task: Profiling

- Use MPI_Wtime function to monitor the time spend in the time iteration of the code.
- Monitor the time spend in the calculation using different number of processors.

```
starttime = MPI_Wtime();

your code to be monitored

time_elapsed = MPI_Wtime() - startime;
```