# MPI and OpenMP Programming for Parallel Matrix Multiplication

myson @ postech.ac.kr

CSE700-PL @ POSTECH

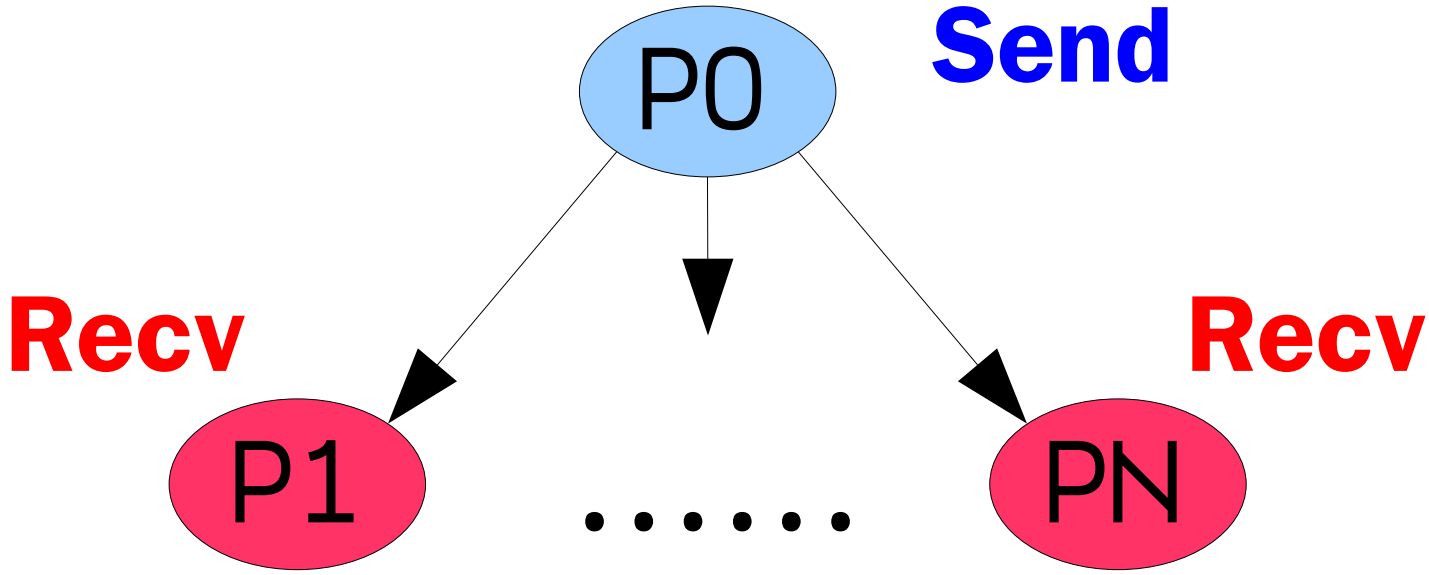
# Outline

- MPI
  - Broadcasting
  - Data grouping
  - Topology (Grid)
- OpenMP
  - Scheduling
- Summary

# MPI

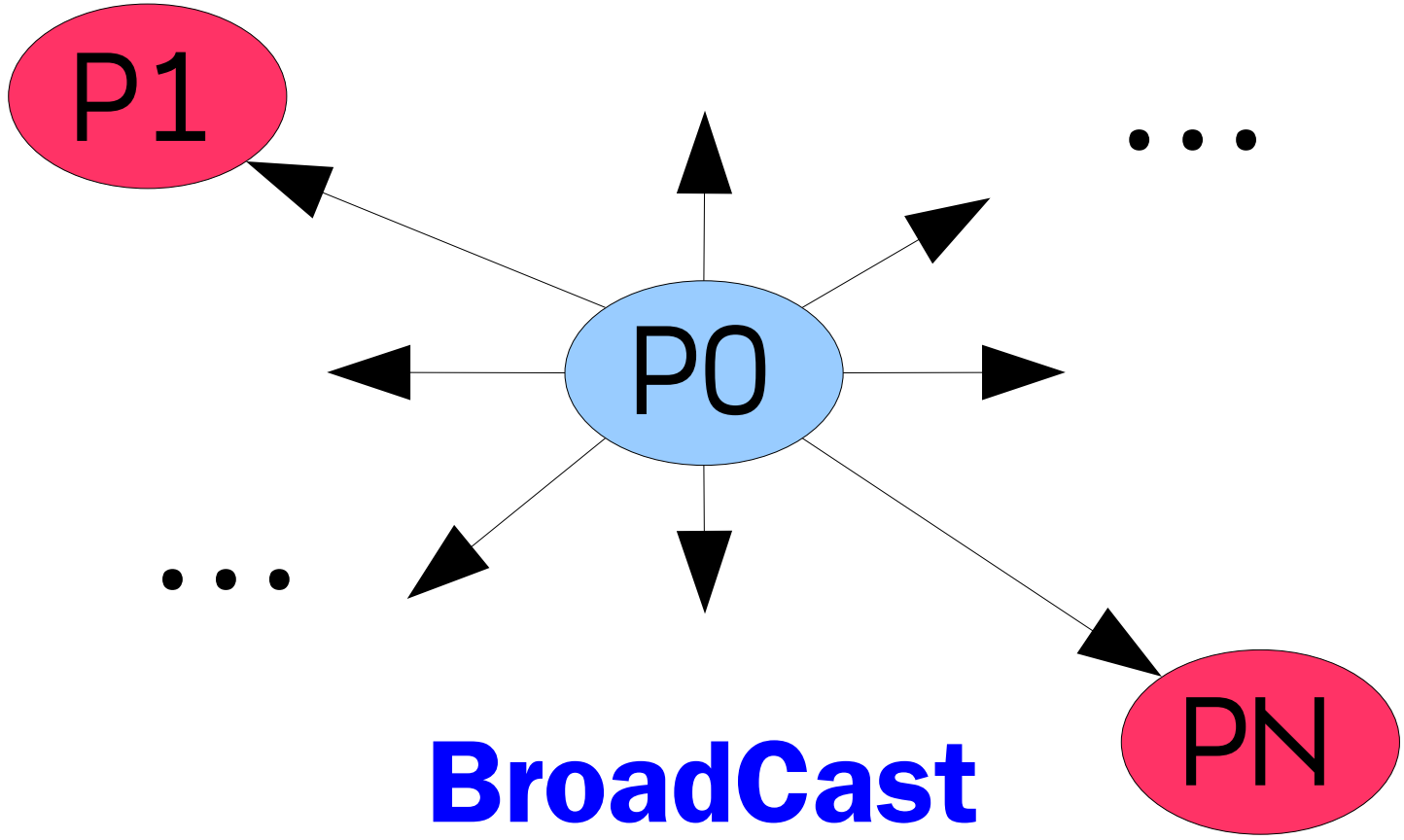
# Send and Recv

```
if (my_rank != 0){
  MPI_Recv (..., 0, ...);
}
else{ /* my_rank == 0 */
  for (dest = 0; dest < n + 1; dest++)
    MPI_Send (..., dest, ...);
}
```



# Broadcasting

```
BCast(..., 0,...);
```

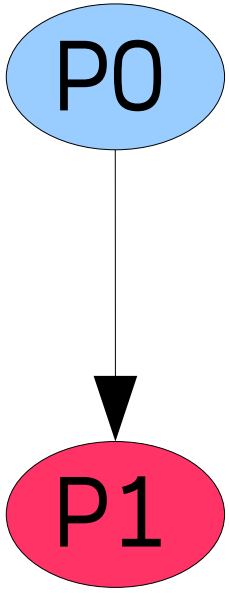


# Data Grouping

- `count` parameter
- Derived datatype
- Pack/ Unpack

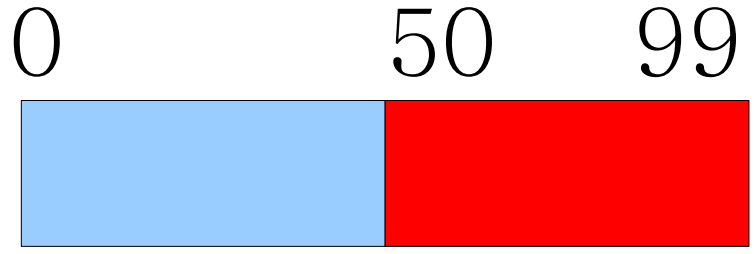
# Data Grouping - 1

- `count` parameter




**MPI_Send(array+50, 50, ...);**

**MPI_Recv(array+50, 50, ...);**




```
int array[100];
```

# Data Grouping - 1 (cont.)

- `count` parameter (cont.)

ex) in the assignment1

```
float messages[3];

messages[0] = a;
messages[1] = b;
messages[2] = (float) n;
⋮
MPI_Send(messages, 3, MPI_FLOAT, ...);
```

$\Longrightarrow$ Poor programming style!!!

# Data Grouping - 1 (cont.)

- `count` parameter (cont.)

ex) in the assignment1

```
typedef struct{
  float a;
  float b;
  int n;
} INDATA_T

INDATA_T indata;
⋮
MPI_Send(indata, 1, INDATA_T???, ...);
```

$\Longrightarrow$ We need a new type for MPI!!!

# Data Grouping - 2

- Derived datatype
  - MPI_Type_struct

```
MPI_Type_struct(..., &new_mpi_t);
```

  - MPI_Type_commit

```
MPI_Type_commit(&new_mpi_t);
```

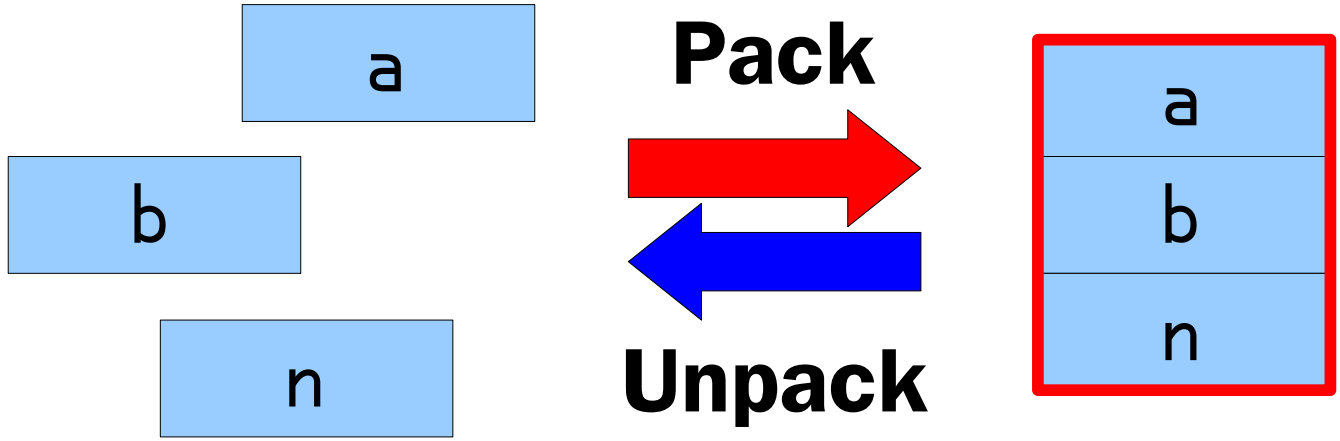
ex)

```
MPI_Send(indata, 1, new_mpi_t, ...);
```

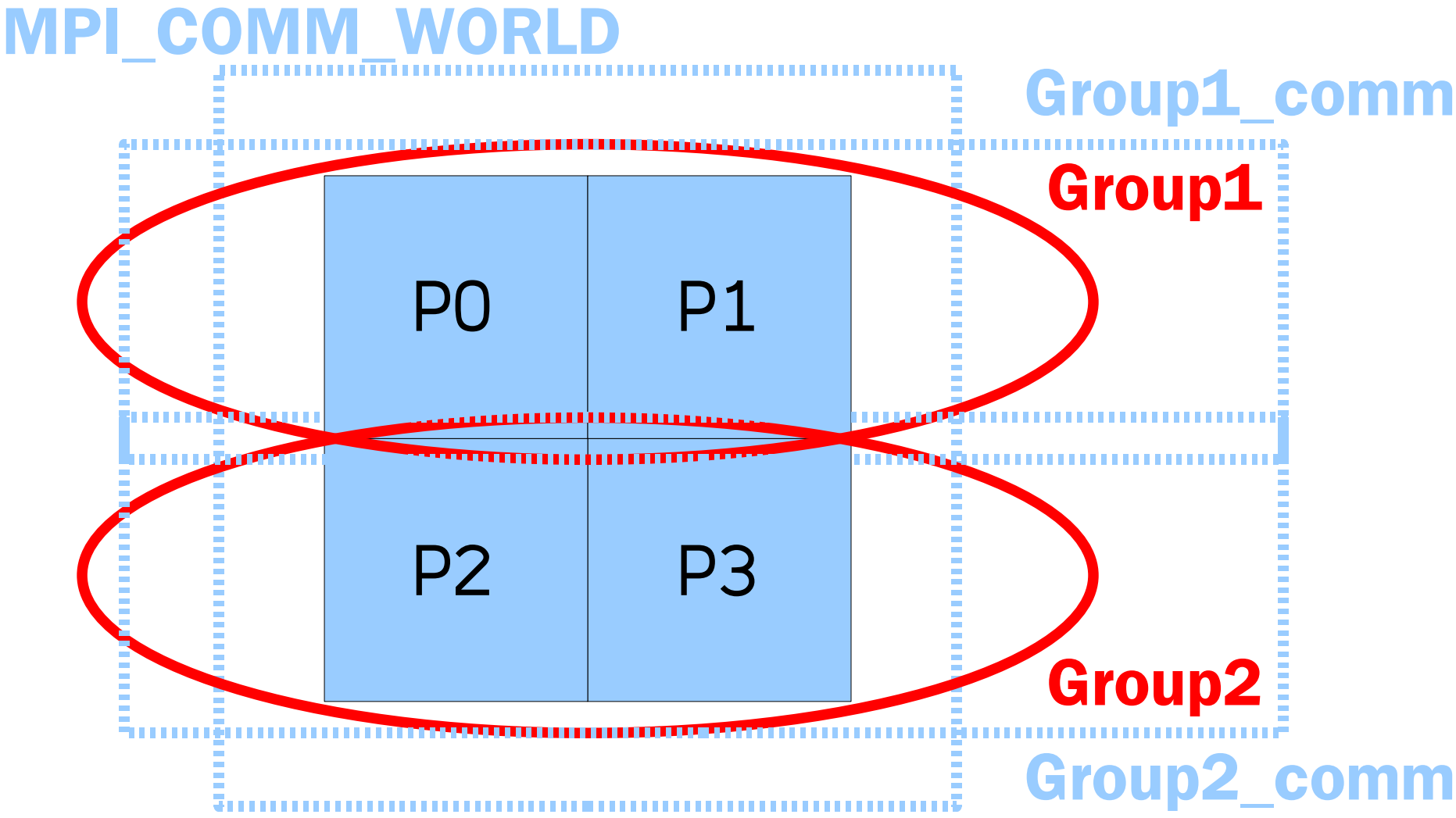
# Data Grouping - 3

- Pack/ Unpack

```
char buffer[100];
int position = 0;

MPI_Pack (&a, 1, MPI_FLOAT, buffer, 100,
                  &position, MPI_COMM_WORLD);
```

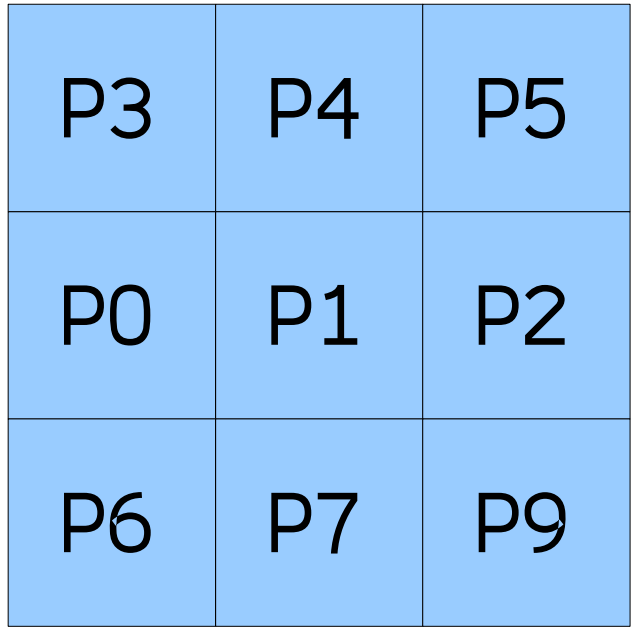


# Communicator and Group



# Topology

- Graph
- Cartesian (or Grid)




**2 Dimensions**

**3 elements**
**for each dimension**

**Periodicity: Circular**

# Topology (cont.)

- Creating a grid

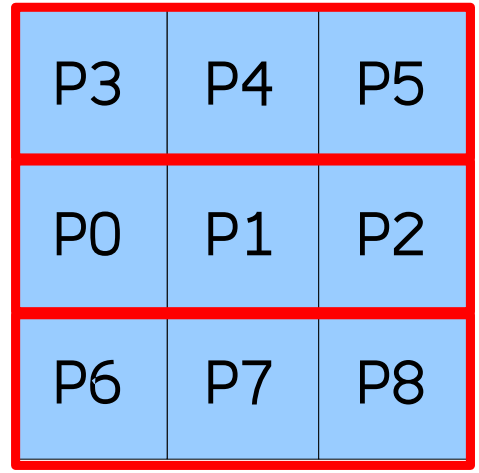
```
MPI_Comm grid_comm;
int coordinates[2];

MPI_Cart_create (MPI_COMM_WORLD, ...,
                                 &grid_comm);
MPI_Comm_rank (grid_comm, &my_grid_rank);
MPI_Cart_coords (grid_comm, my_grid_rank,
                            2, coordinate);
```

ex) The coordnate of P3 : (0, 0)

ex) The coordnate of P7 : (2, 1)

# Topology (cont.)

- Partitioning a grid




```
⋮
int free_coords[2];
MPI_Comm row_comm;

free_coords[0] = 0;
free_coords[1] = 1;

MPI_Cart_sub (grid_comm, free_coords,
                         &row_comm);
```

# OpenMP

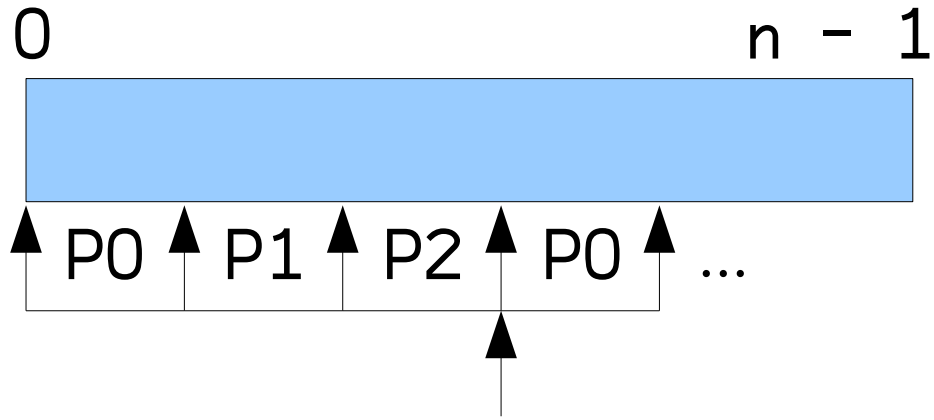
# Scheduling

- Static
- Dynamic
- Guided
- Runtime

```
int i, n;
float z[n], a, x[n], y;

#pragma omp parallel for
        schedule(type [,chunk])
        private(i) shared(n, z, x, y, a)
for (i = 0; i < n; i++)
  z[i] = a * x[i] + y;
```

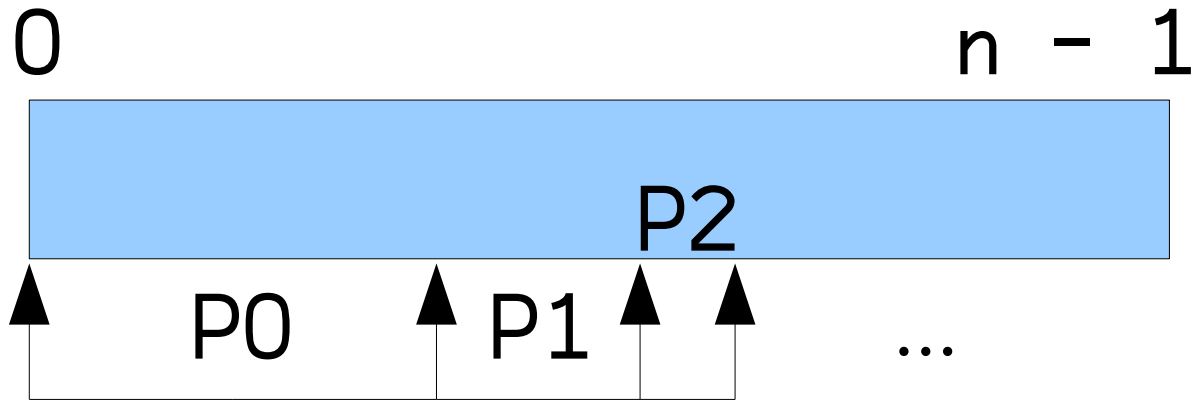
# Scheduling (cont.)

- Static
  - Simple static - `schedule(static)`
    - The number of iterations : n
    - The number of processes : p
    - The size of each chunk = n / p
    - If n % p != 0, ...?
  - Interleaved - `schedule(static, chunk)`
    - The size of each chunk = chunk



# Scheduling (cont.)

- Dynamic
  - Default chunk size : 1
  - Similar to interleaved scheduling
- Guided
  - The size of each successive chunk decreases exponentially
  - Default chunk size : 1



# Scheduling (cont.)

- Runtime
  - No chunk specification
  - Based on the environment variable

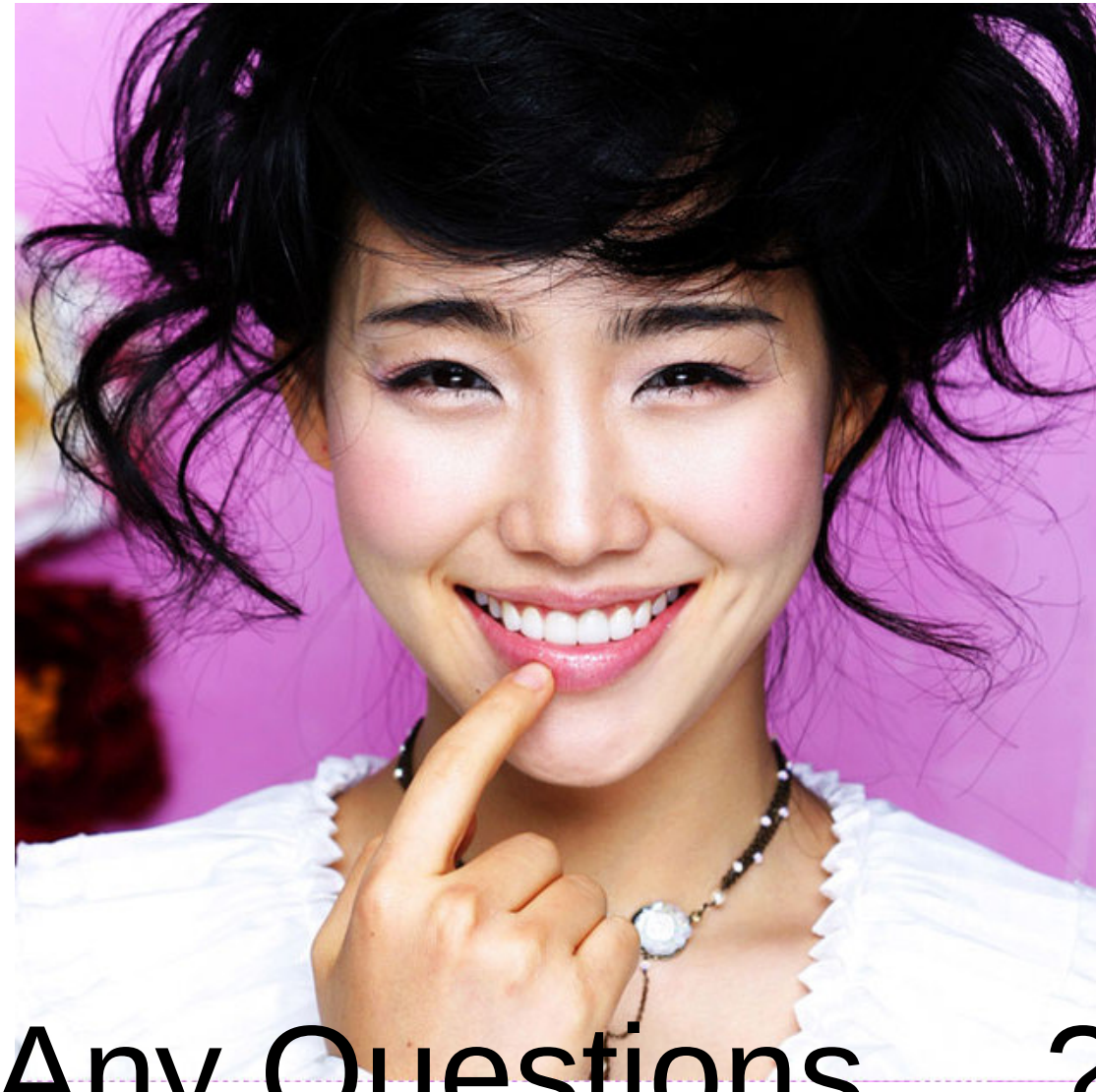
ex)

```
setenv OMP_SCHEDULE “dynamic, 3”
```

# Summary

- MPI
  - Broadcasting
  - Data grouping
    - `count` parameter
    - Derived datatype
    - Pack/ unpack
  - Topology (Grid)
- OpenMP
  - Scheduling
    - Static
    - Dynamic
    - Guided
    - Runtime

# End



Any Questions. . . ?