Friedrich-Alexander-Universität
Erlangen-Nürnberg

Introduction to Parallel Programming with MPI

Dr. Alireza Ghasemi, Dr. Georg Hager

Erlangen National High Performance Computing Center

MPI Subcommunicators

MPI Subcommunicators: use cases

- Let’s assume you have 100 3D FFTs on a large mesh and you want to run on 1000 cores:
  - Each FFT on all 1000 cores, so consecutive runs: ✗
  - Each FFT on one core, then 900 cores remain idle: ✗
- MPMD: Multiple Program Multiple Data Execution Model
  - Different programs can be run simultaneously: communicating via MPI but then using MPI_COMM_WORLD should be avoided in collective communications.
- Sampling algorithms with multiple walkers modeling diffusion processes in physics and chemistry:
  - Many walkers to maximize the randomness, however, it is also required that a walker advances as fast as possible, for example kinetic Monte Carlo
    - Then, each walker on one core not a good choice
- …

Groups and Communicators

- An MPI group is an ordered collection of processes
- Each process inside a group has a unique rank
- A new intracommunicator can be derived from a group, effectively enabling communication (point-to-point or collective) that is restricted to this group
- Predefined intracommunicators:
  - MPI_COMM_WORLD
  - MPI_COMM_SELF (contains only the process itself)

- Two possible scenarios:
  - Create a group containing subsets of the processes in a communicator and then creating a communicator from that group.
  - Directly creating a subcommunicator from a communicator

Handling Groups

- Important group handling subroutines
  - Construct group from existing communicator (COMM):
    ```
    MPI_Comm_group(MPI_Comm comm,MPI_Group *group)
    ```
  - Generate new group by including ranks from existing group:
    ```
    MPI_Group_incl(MPI_Group group,int n,int *ranks[],
       MPI_Group *newgroup)
    ```
  - Generate new group by excluding ranks from existing group:
    ```
    MPI_Group_excl(MPI_Group group,int n,int *ranks,
       MPI_Group *newgroup)
    ```
  - Destroy group:
    ```
    MPI_Group_free(MPI_Group *group)
    ```
- These operations are local to each process

Creating an Intracommunicator

- A communicator can be derived from an existing group (collective):

```
MPI_Comm_create(MPI_Comm comm,MPI_Group group,MPI_Comm *newcomm)
```

  - `Collective operation`
  - If process is not in group, `COMM=MPI_COMM_NULL`

- Deallocating the communicator object:

```
MPI_Comm_free(MPI_Comm *comm)
```

  - Collective but …

Example:

```
…
    MPI_Comm_group(MPI_COMM_WORLD,&group_w);
    if(irank_w<6) {
        if(irank_w%2==0) {irank_list[0]=irank_w;irank_list[1]=irank_w+1;}
        if(irank_w%2==1) {irank_list[0]=irank_w-1;irank_list[1]=irank_w;}
        MPI_Group_incl(group_w,2,irank_list,&group_new);
    }
    else {
        group_new=MPI_GROUP_EMPTY;
    }
    MPI_Comm_create(MPI_COMM_WORLD,group_new,&comm_new);
    if(comm_new!=MPI_COMM_NULL) {
        MPI_Comm_rank(comm_new,&irank_l);
        MPI_Comm_size(comm_new,&nrank_l);
        printf("irank_w,nrank_w= %4d%4d and irank_l,nrank_l=%4d%4d\n",irank_w,nrank_w,irank_l,nrank_l);
    }
    if(group_new!=MPI_GROUP_EMPTY) MPI_Group_free(&group_new);
    if(comm_new!=MPI_COMM_NULL) MPI_Comm_free(&comm_new);
    MPI_Group_free(&group_w);
…
```

Running the example:

Running on 6 processes:

```
mpirun -n 6 ./a.out |sort -n -k2
irank_w,nrank_w=   0  6 and irank_l,nrank_l=   0  2
irank_w,nrank_w=   1  6 and irank_l,nrank_l=   1  2
irank_w,nrank_w=   2  6 and irank_l,nrank_l=   0  2
irank_w,nrank_w=   3  6 and irank_l,nrank_l=   1  2
irank_w,nrank_w=   4  6 and irank_l,nrank_l=   0  2
irank_w,nrank_w=   5  6 and irank_l,nrank_l=   1  2
```

Running on 7 processes:

```
mpirun -n 7 ./a.out |sort -n -k2
irank_w,nrank_w=   0  7 and irank_l,nrank_l=   0  2
irank_w,nrank_w=   1  7 and irank_l,nrank_l=   1  2
irank_w,nrank_w=   2  7 and irank_l,nrank_l=   0  2
irank_w,nrank_w=   3  7 and irank_l,nrank_l=   1  2
irank_w,nrank_w=   4  7 and irank_l,nrank_l=   0  2
irank_w,nrank_w=   5  7 and irank_l,nrank_l=   1  2
```

Direct creation of a communicator

- Creating a new communicator based on color and key codes:

```
MPI_Comm_split(MPI_Comm comm,int color,int key,MPI_Comm *newcomm)
```

- Collective operation:
  - If `color` is set to MPI_UNDEFINED, `newcomm` returns MPI_COMM_NULL
- `color`: controls the assignment of processes in the new subset
  - Nonnegative integer
  - Processes with the same value of color in the same subset
- `key`: controls the rank assignment in the new communicator
  - For every pair of processes:
    - process with a smaller value of key results in a smaller value of rank in `newcomm`
    - in case of identical key values, the order of ranks follows the order in the parent one

Intercommunicator

- Intercommunicator is used for communication between two disjoint groups. If not disjoint, then very risk of deadlock!
- Useful when algorithm works in a server-client paradigm

```
MPI_Intercomm_create(MPI_Comm local_comm, int local_leader,
MPI_Comm peer_comm, int remote_leader, int tag, MPI_Comm
*newintercomm)
```

  - It is collective over the union of the local and remote groups
  - At least one selected member from each group (the “group leader”) has the ability to communicate with the selected member from the other group
  - `peer_comm`: a communicator in which both leaders exist
  - each leader knows the rank of the other leader in this peer communicator
  - members of each group know the rank of their leader

Quiz

1) Subcommunicators are suitable only for use in libraries?
   a) Correct
   b) Incorrect

2) `MPI_Comm_split` creates subcommunicators with disjoint groups?
   a) Correct
   b) Incorrect

3) Two groups can have common members but two communicators cannot.
   a) Correct
   b) Incorrect