Parallel Scientific Computation

MPI 3

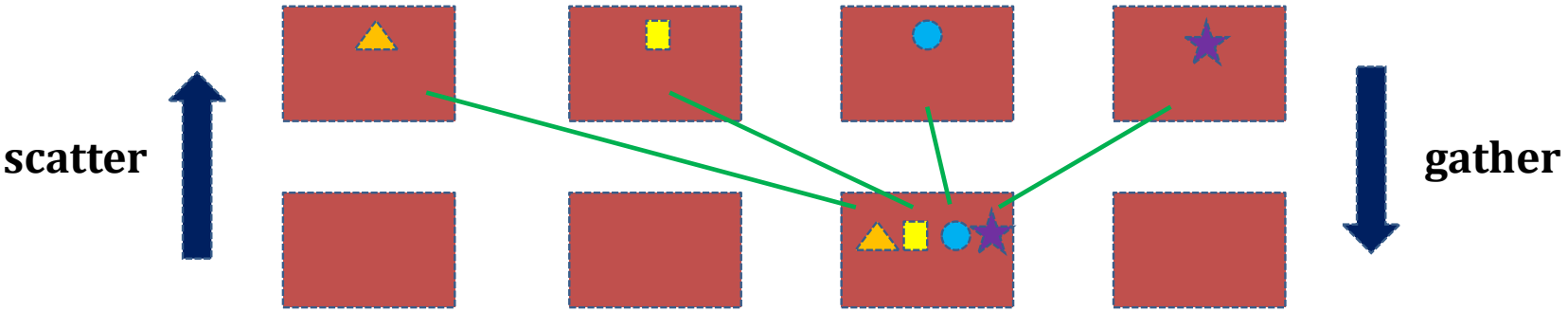
J.-H. Parq
IPCST
Seoul National University

Collective Communication

- One-to-all
  - Bcast (broadcast)
  - Scatter or Scatterv
- All-to-one
  - Reduce
  - Gather or Gatherv

Collective Communication

- Scatter & Gather
  - Scatter ↔ Gather



  - Scatter & Gather have the same argument form:
    (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierror)
    - But size(sendbuf) > size(recvbuf) for scatter and reverse for gather

Collective Communication

- Gather examples
  - Fortran

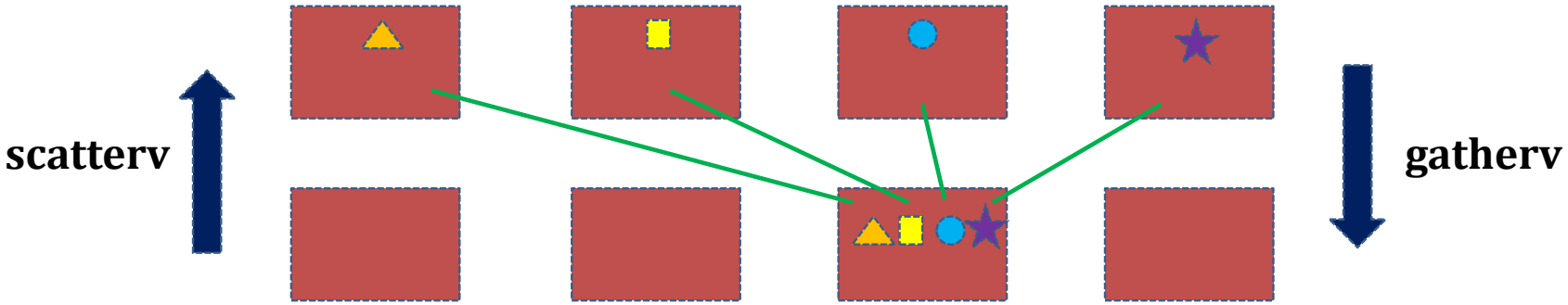
```
INTEGER value, values(N)
……
call MPI_GATHER(value, 1, MPI_INTEGER, values, 1, MPI_INTEGER, &
                                        0, comm0, ier)
call MPI_COMM_SIZE(comm0, size, ier)
call MPI_BCAST(values, size, MPI_INTEGER, 0, comm0, ier)
```

  - C

```
int value, values[N];
……
MPI_Gather(&value, 1, MPI_INT, values, 1, MPI_INT, 0, comm0);
MPI_Comm_size(comm0, &size);
MPI_Bcast(values, size, MPI_INT, 0, comm0);
```

Collective Communication

- Scatterv & Gatherv
  - Scatterv ↔ Gatherv



  - Scatterv & Gatherv have the same argument form:
    (sendbuf, sendcount, sendtype, recvbuf, recvcount, displacements, recvtype, root, comm, ierror)
    - But size(sendbuf) > size(recvbuf) & size(sendcount) > size(recvcount) for scatterv and reverse for gatherv

Collective Communication

- Gatherv examples
  - Fortran

```
INTEGER svalues(M), rvalues(N), rcounts(4), displs(4)
……
rcounts(1) = 1
rcounts(2) = 2
rcounts(3) = 3
rcounts(4) = 4
displs(1) = 0
displs(2) = 1
displs(3) = 3
displs(4) = 6
call MPI_COMM_RANK(comm0, myid, ier)
call MPI_GATHERV(svalues, myid + 1, MPI_INTEGER, rvalues, rcounts, &
                                        displs, MPI_INTEGER, 0, comm0, ier)
```

Collective Communication

- Gatherv examples
  - C

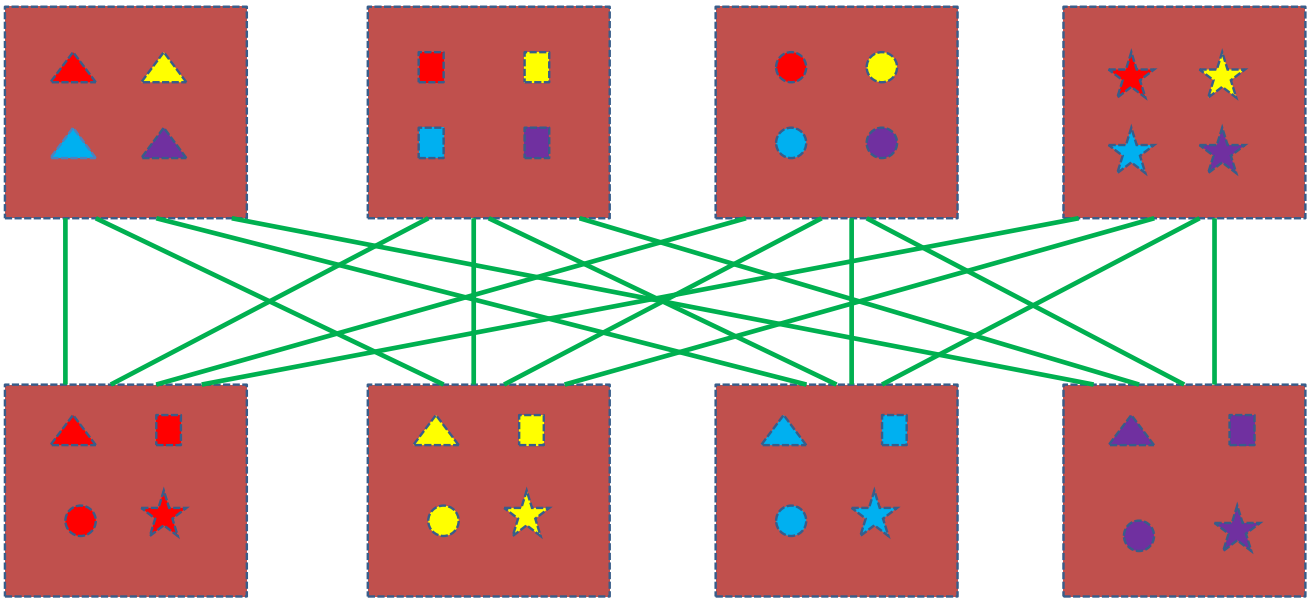
```
int svalues[M], rvalues[N], *rcounts, *displs;
……
MPI_Comm_size(comm0, &size);
MPI_Comm_rank(comm0, &myid);
rcounts = (int*)malloc(size*sizeof(int));
displs = (int*)malloc(size*sizeof(int));
sum = 0;
for (i = 0; i<size; i++) {
   rcount[i] = i + 1;
   displs[i] = sum;
   sum += i + 1;
}
MPI_Gatherv(svalues, myid+1, MPI_INT, rvalues, rcounts, displs, MPI_INT, 0, comm0);
```

Collective Communication

- All-to-all
  - Arguments = (sendbuf, sendcount, sendtype, recvbuf, recvcount, displs, recvtype, comm, ierror)
    - Allgather or Allgatherv: Gather(v) & Bcast
    - Alltoall or Alltoallv: Gather(v) & Scatter(v) for all
  - Arguments = (sendbuf, recvbuf, count, datatype, op, comm, ierror)
    - Allreduce: Reduce & Bcast
    - Scan: partial Allreduce
      - On process *i*, computation of data from processes 0 to *i*
    - Reduce_scatter: Reduce & Scatterv
      - Count array → distributing reduced results

Collective Communication

- Alltoall



Synchronization

- Barrier
  - Important in one-to-all or all-to-one collective communications where all processes are not equal
  - Used for
    - Exact timing check
    - Serial I/O
    - Debugging

Synchronization

- Barrier
  - Note 1: Unable to block non-blocking sending or receiving; You must use **Wait** or **Waitall** for non-blocking communications.
  - Note 2: Possible to reduce performance speed; You'd better not use **Barrier** if unnecessary.
  - MPI_BARRIER(comm, ierror)
  - int MPI_Barrier(MPI_Comm comm)

Communicator

- In Fortran, communicators are of integer type.
  - Ex.) integer World = MPI_COMM_WORLD
- In C, communicators are of MPI_Comm type.
  - Ex.) MPI_Comm world = MPI_COMM_WORLD;

Communicator

- Group extractor
  - Extracting from a communicator its group
    - MPI_COMM_GROUP(comm, group, ierror)
    - int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)
  - Group: Ordered set of processes
    - Fortran: Integer type
    - C: MPI_Group type

Communicator

- Manipulating a group
  - MPI_Group_union(G1, G2, *newG)
  - MPI_Group_intersection(G1, G2, *newG)
  - MPI_Group_difference(G1, G2, *newG)
  - MPI_Group_incl(G, n, *ranks, *newG)
  - MPI_Group_excl(G, n, *ranks, *newG)
  - MPI_Group_range_incl(G, n, ranges[][3], newG)
  - MPI_Group_range_excl(G, n, ranges[][3], newG)

Communicator

- Checking a group
  - MPI_Group_size(G, *size)
  - MPI_Group_rank(G, *rank)
  - MPI_Group_translate_ranks(G1, n, *ranks1, G2, *ranks2)
    - Finding the ranks in G2
  - MPI_Group_compare(G1, G2, *result)
    - Rerurning MPI_IDENT, MPI_SIMILAR, or MPI_UNEQUAL

Communicator

- Making a new communicator
  1. Creating from a group
     - MPI_COMM_CREATE(comm, group, newcomm, ierror)
     - int MPI_Comm_create(MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm)
     - ✓ The first argument ‘comm’ is needed for the other information such as the context.

Communicator

- Making a new communicator
  2. Splitting a communicator
     - MPI_COMM_SPLIT(oldcomm, color, key, newcomm, ierror)
     - int MPI_Comm_split(MPI_Comm oldcomm, int color, int key, MPI_Comm *newcomm)
     - ✓ Processes of the same color(≥0) → one new subcomm
     - ✓ Key order → new rank order (Smaller first. Original order if all keys are identical)

Communicator

- Making a new communicator
  3. Copying a communicator
     - MPI_COMM_DUP(comm, newcomm, ierror)
     - int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)

Communicator

- Freeing a group and a communicator
  - Erasing a group (actually its reference)
    - MPI_GROUP_FREE(group, ierror)
    - int MPI_Group_free(MPI_Group *group)
  - Erasing a communicator
    - MPI_COMM_FREE(comm, ierror)
    - int MPI_Comm_free(MPI_Comm *comm)

Communicator

- Example (Fortran)

```
Integer World, Workers, World_group, Worker_group, ranks(1)
……
World = MPI_COMM_WORLD
Call MPI_COMM_SIZE(World, numprocs, ierror)
server = numprocs - 1
Call MPI_COMM_GROUP(World, World_group, ierror)
ranks(1) = server
Call MPI_GROUP_EXCL(World_group, 1, ranks, Worker_group, ierror)
Call MPI_COMM_CREATE(World, Worker_group, Workers, ierror)
……
Call MPI_GROUP_FREE(Worker_group, ierror)
Call MPI_GROUP_FREE(World_group, ierror)
```

Communicator

- Example (C)

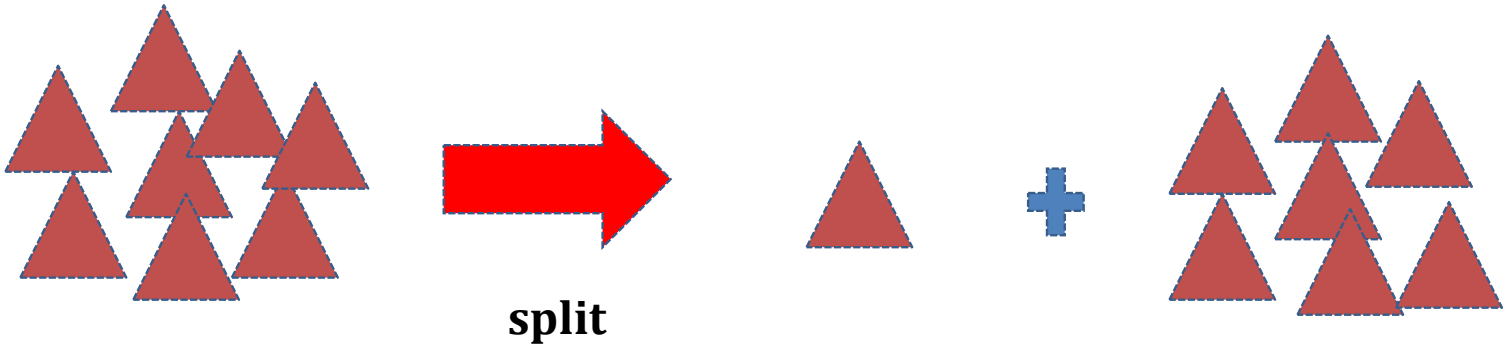
```
MPI_Comm world, workers;
MPI_Group world_group, worker_group;
int ranks[1];
.....
world = MPI_COMM_WORLD;
MPI_Comm_size(world, numprocs);
server = numprocs – 1; ranks[0] = server;
MPI_Comm_group(world, &world_group);
MPI_Group_excl(world_group, 1, ranks, &worker_group);
MPI_Comm_create(world, worker_group, &workers);
.....
MPI_Group_free(&worker_group);
MPI_Group_free(&world_group);
```

Monte Carlo Methods

1. Stochastic modeling with sampling
   - Statistical physics, Quantum Monte Carlo
   - Ensemble (weather) forecasting
   - Rarefied gas dynamics
   - Financial evaluation
2. A technique using random sampling to solve a mathematical problem
   - Numerical integration
   - Optimization
   - Ray tracing (in computer graphics)
   - Artificial intelligence for games?

Monte Carlo Integration

- MPI strategy



❖ Random numbers are generated on only one process.

Monte Carlo Integration

- Notes on random number generation
  - Ordinary random number generators are unable to be parallelized because what they generate are pseudo-random numbers.
    - Real random numbers: By physical measurements
    - Pseudo-random numbers: By algorithms
  - If you force to use them on every process, their random numbers will probably have correlations.
  - There are parallel random number generators such as SPRNG and VecRNG, though.

Monte Carlo Integration

- Fortran imperfect code

```
……
Integer chunksize          ! It should depend on bandwidth or throughput.
……
Integer rands(chunksize), demand
Integer World, Workers
Integer color, key
……
World = MPI_COMM_WORLD
Call MPI_COMM_SIZE(World, numprocs, …)
Call MPI_COMM_RANK(World, myid, …)
server = numprocs – 1
```

Monte Carlo Integration

- Fortran imperfect code

```
key = 0
color = 0
If (myid == server) color = 1
Call MPI_COMM_SPLIT(World, color, key, Workers, …)

If (myid == server) then
  Do
    Call MPI_RECV(demand, 1, MPI_ANY_SOURCE, …, World, status, …)
    If (demand == 0) Exit
     [ random number generation → rands(chunksize) ]
    Call MPI_SEND(rands, chunksize, …, status(MPI_SOURCE), …, World, ..)
  End Do
```

Monte Carlo Integration

- Fortran imperfect code

```
Else                    ! A process in workers
 demand = 1
 Call MPI_SEND(demand, 1, …, server, …, World, …)
 Do
  Call MPI_RECV(rands, chunksize, …, server, …, World, status, …)
  Do I = 1, chunksize
    [sampling and summation of integrand values with weights]
  End Do
  Call MPI_ALLREDUCE(partial_sum, total, 1, …, MPI_SUM, Workers, …)
  demand = 1
  If (accuracy condition) demand = 0
```

Monte Carlo Integration

- Fortran imperfect code

```
   If (myid == 0) Then
     Call MPI_SEND(demand, 1, …, server, …, World, …)
   Else If (demand == 1)
     Call MPI_SEND(demand, 1, …, server, …, World, …)
   End If
   If (demand == 0) Exit
  End Do
End If
[output]
Call MPI_COMM_FREE(Workers)
Call MPI_FINALIZE(…)
……
```

Cartesian Topology

- It is possible to make a communicator reflecting Cartesian coordinate topology.
  - This is useful for the finite difference method.

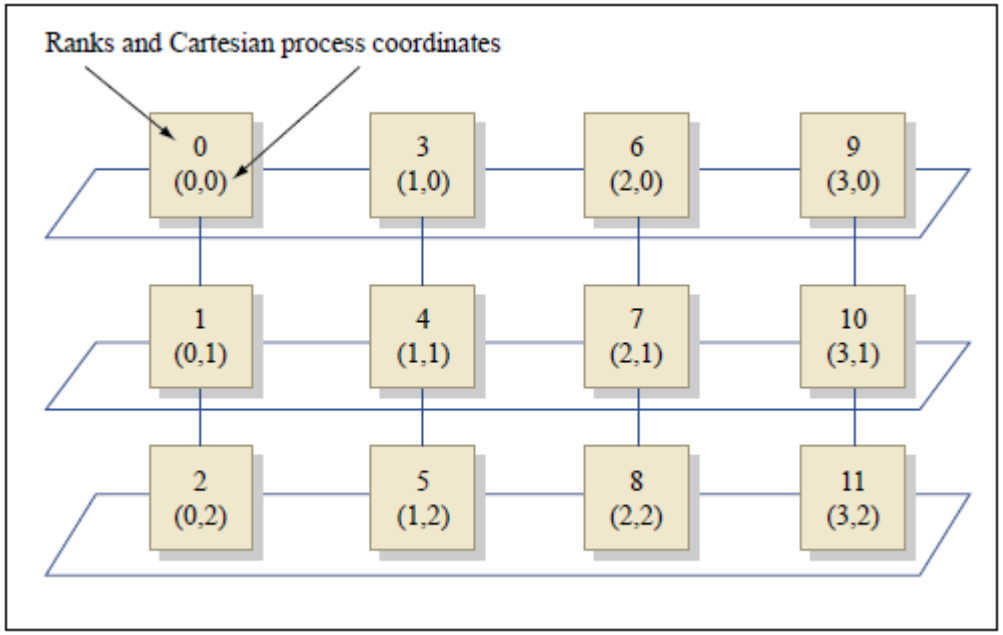


Figure by MIT OpenCourseWare.

Cartesian Topology

- Creating a Cartesian communicator
  - MPI_CART_CREATE(oldcomm, ndims, dims, periodic, reorder, newcomm, ierror)
  - int MPI_Cart_create(MPI_Comm oldcomm, int ndims, int *dims, int *periodic, int reorder, MPI_Comm *newcomm)
  - dims: number of processes in each direction
  - periodic: .True.(1) or .False.(0) in each direction
  - reorder: optimizing process assignment if .true.(1)

Cartesian Topology

- Finding coordinates
  1. comm → dims, periodic, coords
     - MPI_Cart_get(comm, maxdims, *dims, *periodic, *coords)
       - 'dims' and 'periodic' are those used in 'Cart_create'
       - maxdims(≥ ndims): length of dims, periodic, coords
  2. rank → coords
     - MPI_Cart_coords(comm, rank, maxdims, *coords)
       - Ex.)

         MPI_Comm_rank(cart2d, myid);

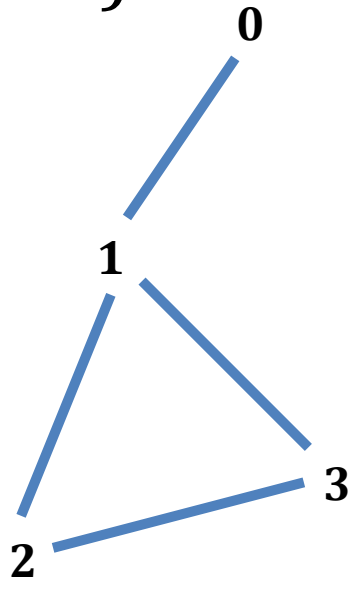
         MPI_Cart_coords(cart2d, myid, 2, coords);

Cartesian Topology

- Finding ranks
  1. Coordinates → rank
     - MPI_Cart_rank(cartcomm, coords[], *rank)
  2. Finding neighbors
     - MPI_Cart_shift(cartcomm, direction, displacement, *source, *destination)
     - source: - direction neighbor
     - destination: + direction neighbor
     - ❖ MPI_PROC_NULL : No process exists at that point.

Graph Topology

- MPI_Graph_create(……)
- MPI_Graph_get(……)
- MPI_Graph_neighbors_count(……)
- MPI_Graph_neighbors(……)
- MPI_Graph_map(……)



References

- W. Gropp, E. Lusk, and A. Skjellum,
  Using MPI

- C. Evangelinos,
  Parallel Programming for Multicore Machines Using OpenMP and MPI

- MPI forum (www.mpi-forum.org)