CSE502: Foundations of Parallel Programming

Lecture 22: Collective Communications in MPI, Hybrid Parallelism by Using OpenMP in MPI

Vivek Kumar

Computer Science and Engineering

IIIT Delhi

vivekk@iiitd.ac.in

Last Class

Point-to-point communication in MPI

- Blocking
  - MPI_Send
  - MPI_Recv
- Message buffering
- When Does MPI_Send/MPI_Recv Returns?
- If tags at send and recv doesn't match then it will create a deadlock
- Message ordering guarantees – If a sender sends two messages (Msg_1 and Msg_2) in succession to same destination, and both match the same receive, the recv operation will always receive Msg_1 before Msg_2
- No guarantee for fairness
- Non-blocking
  - MPI_Isend
  - MPI_Irecv
- These APIs returns immediately. They do not wait for any communication events to complete, such as message copying from user memory to system buffer space or the actual arrival of message

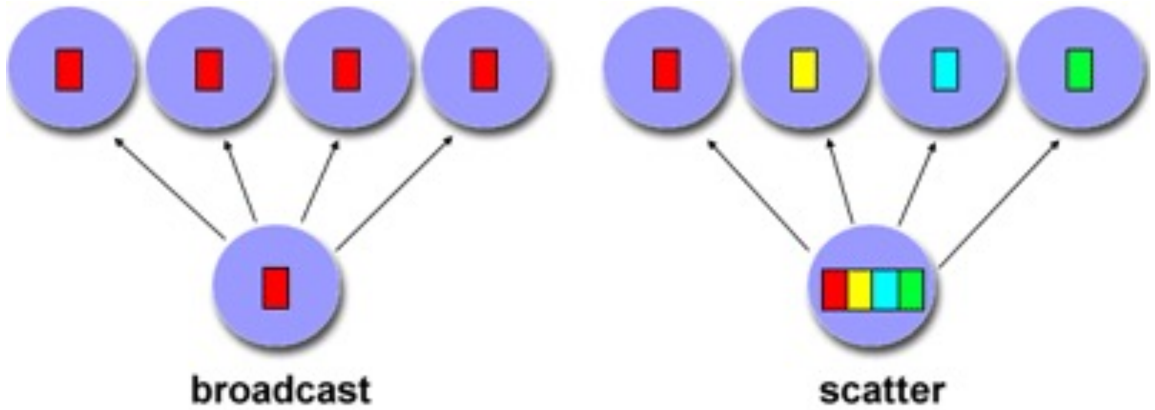
Today's Class

- Collective communications in MPI
- Hybrid parallelism by using OpenMP thread-level parallelism in MPI processes

MPI_Barrier

- MPI_Barrier(MPI_Comm communicator)
  - Synchronization operation across all processes inside the "communicator"
  - Simplest collective communication in MPI

Collective Communications



- One to Many (Broadcast, Scatter)
- Many to One (Reduce, Gather)
- Many to Many (AllReduce, Allgather)

Benefits of Collective over Point-to-Point

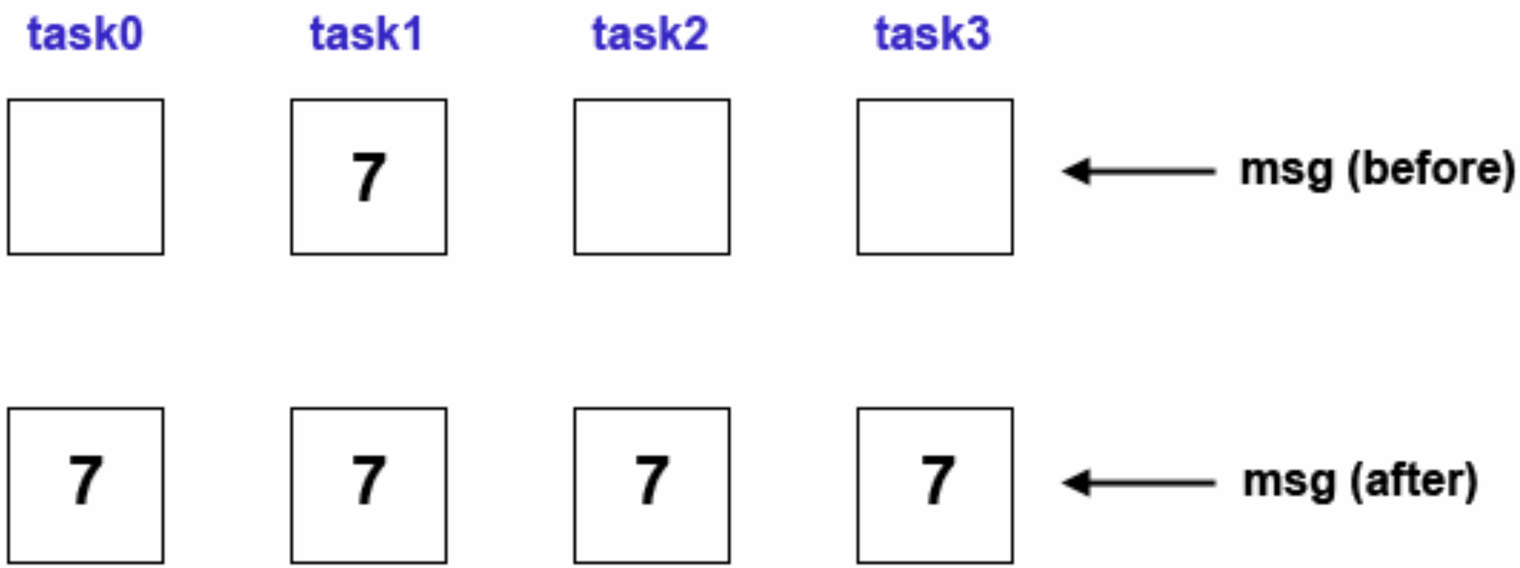
- Productivity
  - Easy to write code
- Performance
  - Machine specific optimization
  - Topology aware optimizations

MPI_Bcast

Broadcasts a message from one task to all other tasks in communicator

```
count = 1;
source = 1;
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```

task1 contains the message to be broadcast



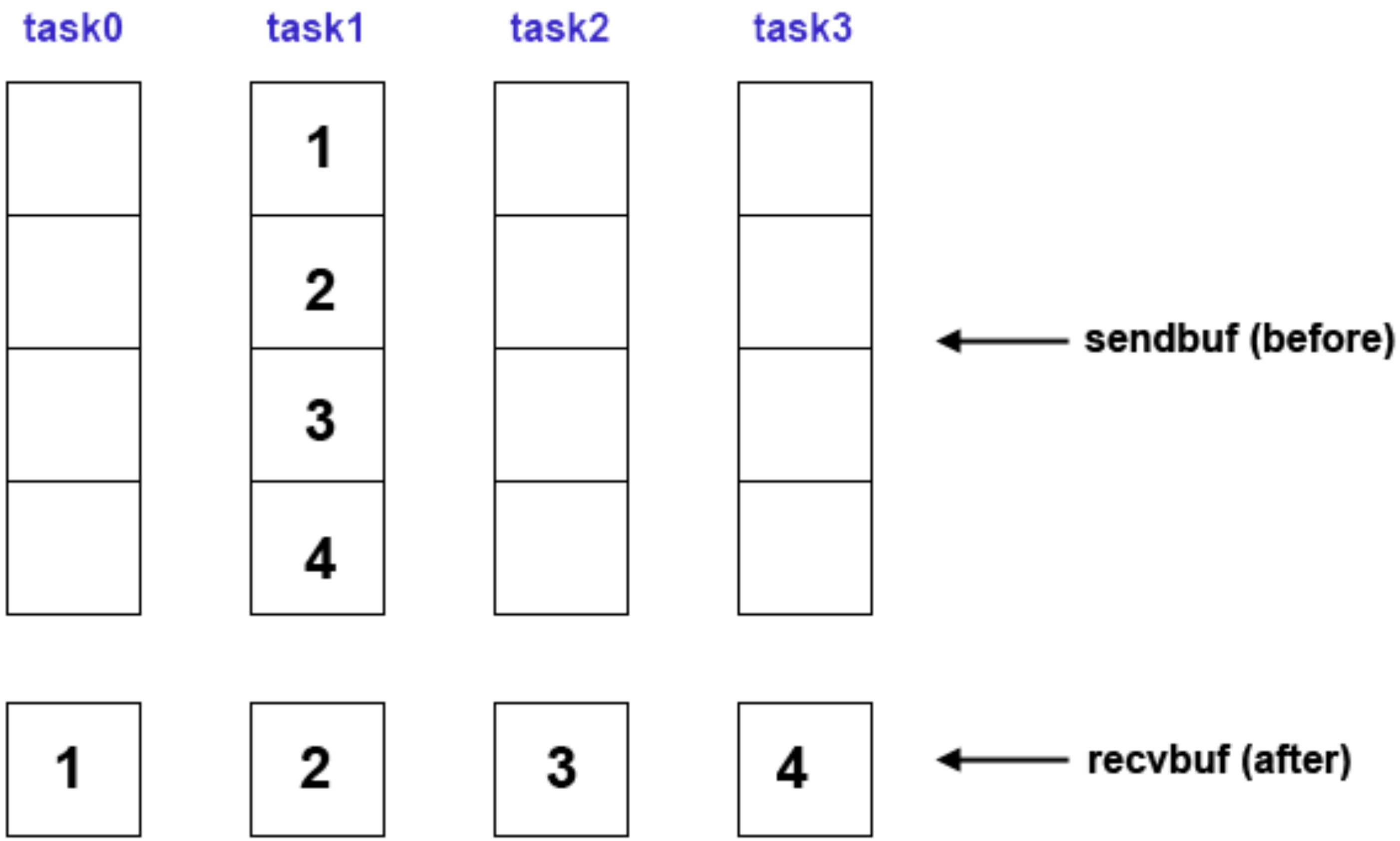
Picture source: https://computing.llnl.gov/tutorials/mpi/

MPI_Scatter

Sends data from one task to all other tasks in communicator

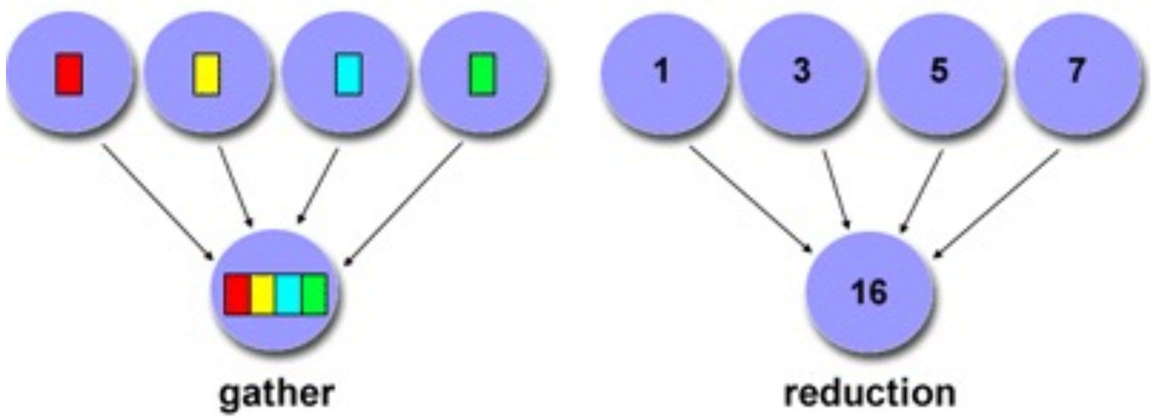
```
sendcnt = 1;
recvcnt = 1;
src = 1;
MPI_Scatter(sendbuf, sendcnt, MPI_INT
            recvbuf, recvcnt, MPI_INT
            src, MPI_COMM_WORLD);
```

task1 contains the data to be scattered



Picture source: https://computing.llnl.gov/tutorials/mpi/

Collective Communications



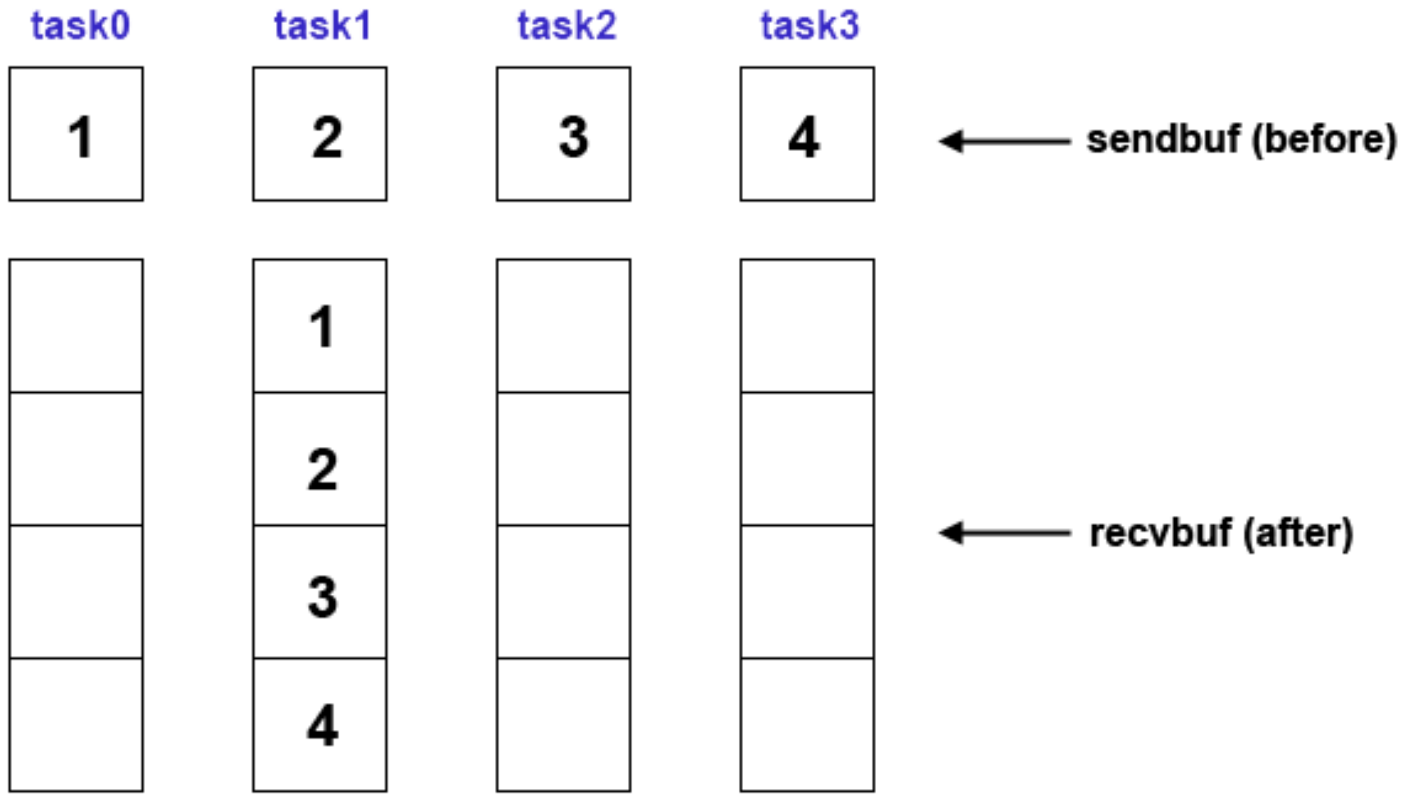
- One to Many (Broadcast, Scatter)
- Many to One (Reduce, Gather)
- Many to Many (AllReduce, Allgather)

MPI_Gather

Gathers data from all tasks in communicator to a single task

```
sendcnt = 1;
recvcnt = 1;
src = 1;
MPI_Gather(sendbuf, sendcnt, MPI_INT
           recvbuf, recvcnt, MPI_INT
           src, MPI_COMM_WORLD);
```

message will be gathered into task1



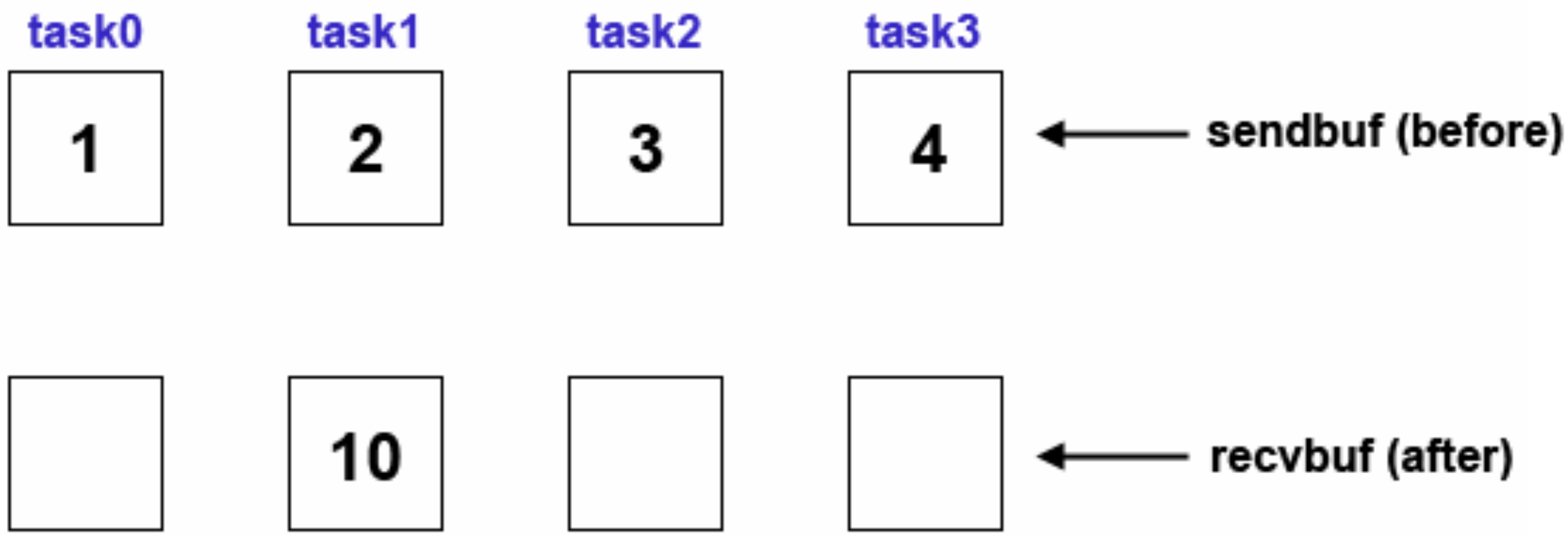
Picture source: https://computing.llnl.gov/tutorials/mpi/

MPI_Reduce

Perform reduction across all tasks in communicator and store result in 1 task

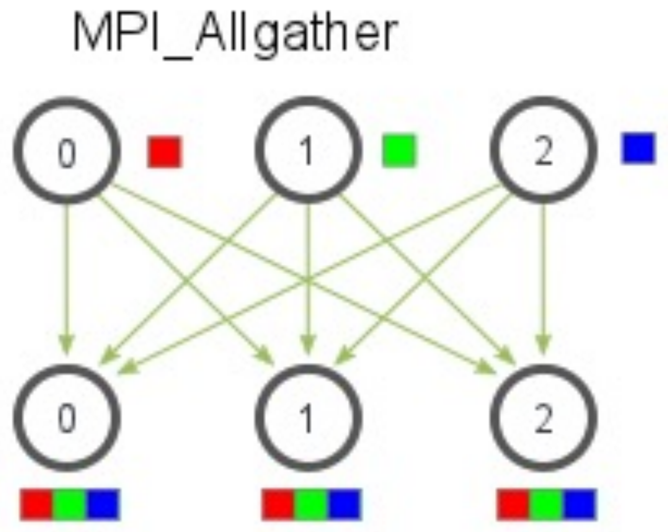
```
count = 1;
dest = 1;
MPI_Reduce(sendbuf, recvbuf, count, MPI_INT,
           MPI_SUM, dest, MPI_COMM_WORLD);
```

task1 will contain result



Picture source: https://computing.llnl.gov/tutorials/mpi/

Collective Communications

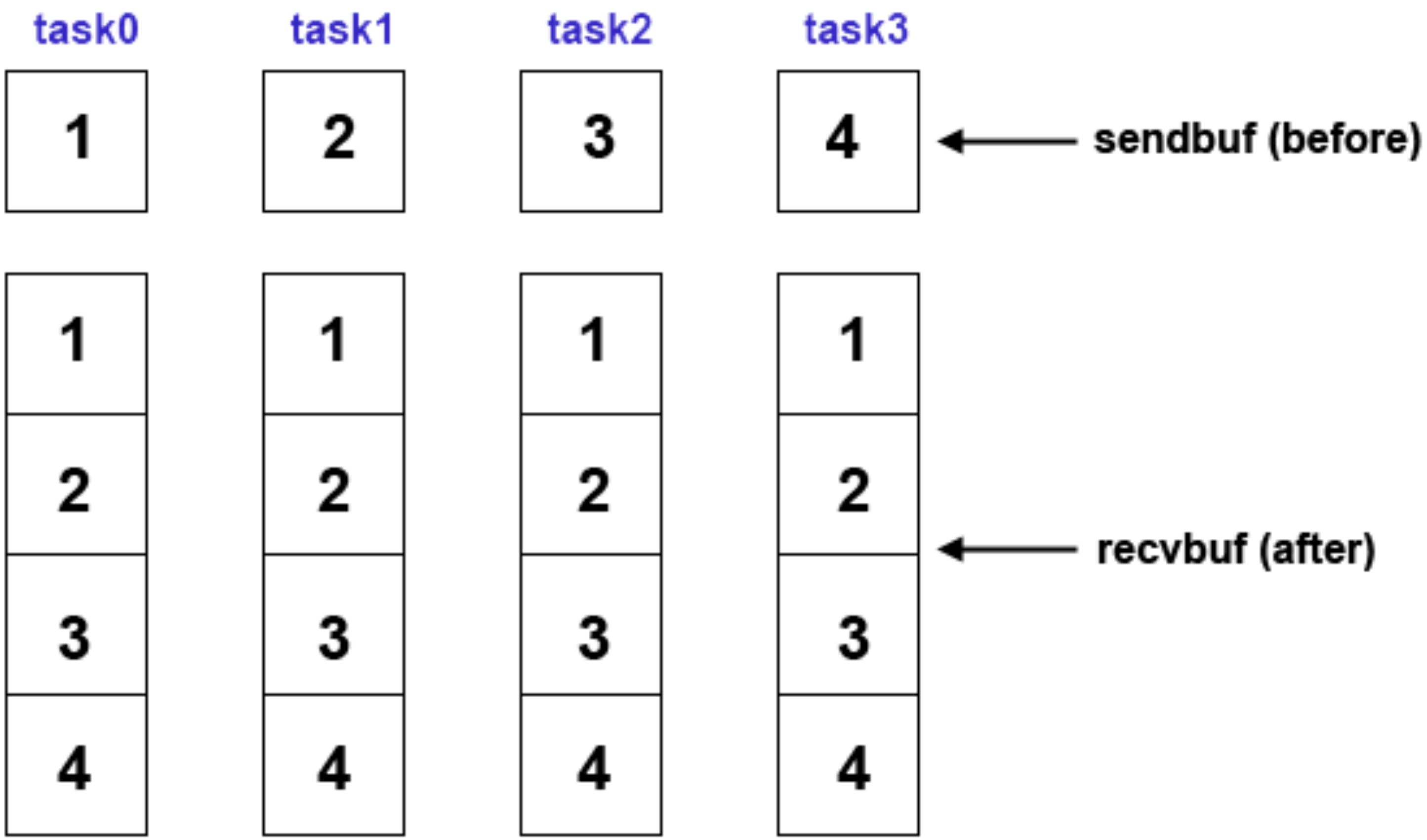


- One to Many (Broadcast, Scatter)
- Many to One (Reduce, Gather)
- Many to Many (AllReduce, Allgather)

MPI_Allgather

Gathers data from all tasks and then distributes to all tasks in communicator

```
sendcnt = 1;
recvcnt = 1;
MPI_Allgather(sendbuf, sendcnt, MPI_INT
              recvbuf, recvcnt, MPI_INT
              MPI_COMM_WORLD);
```

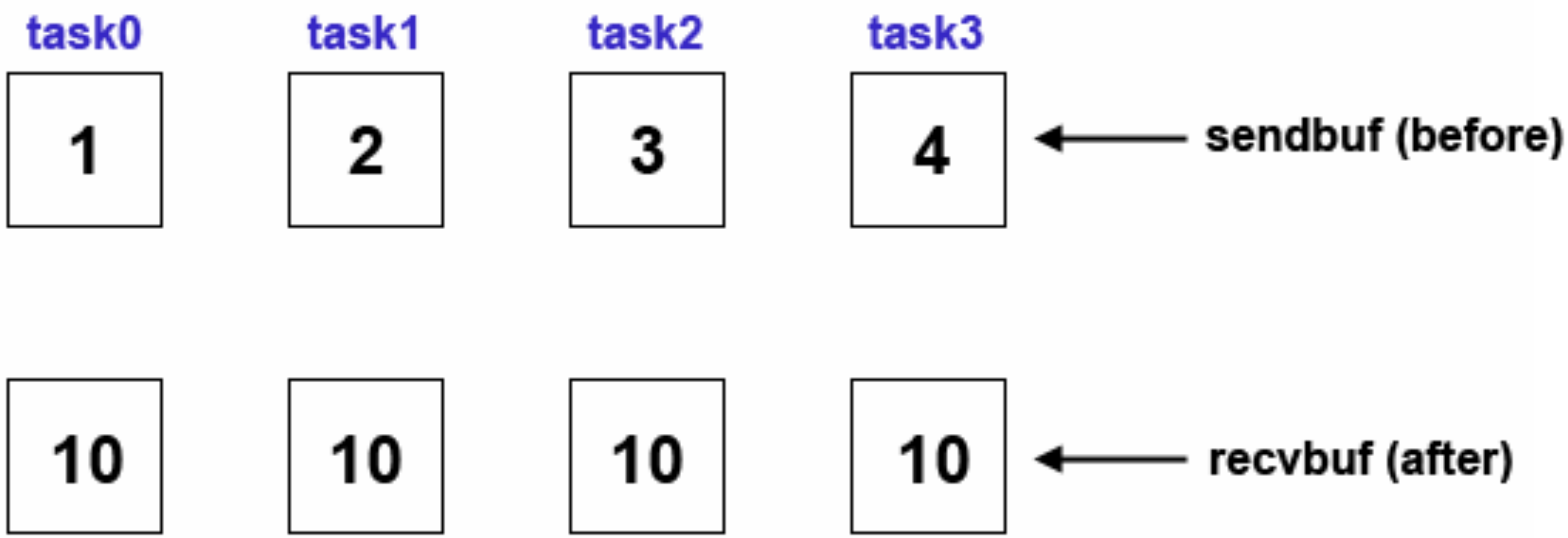


Picture source: https://computing.llnl.gov/tutorials/mpi/

MPI_Allreduce

Perform reduction and store result across all tasks in communicator

```
count = 1;
MPI_Allreduce(sendbuf, recvbuf, count, MPI_INT,
              MPI_SUM, MPI_COMM_WORLD);
```



Picture source: https://computing.llnl.gov/tutorials/mpi/

Parallel Array Sum Using Collective Comunication (which one??)

```
main(int argc, char **argv) {
  int rank, nproc;
  MPI_Init(&argc, &argv);
  // 1. Get to know your world
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &nproc);
  int array[SIZE * np]; // properly initialized
  // 2. calculate local sum
  int my_sum = 0, total_sum, tmp, tag=1, start = rank*SIZE;
  for (int i=start; i<SIZE+start; i++) my_sum += array[i];
  // 3. All non-root processes send result to root processes (rank=0)
  if(rank > 0) {
    MPI_Send(&my_sum, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);
  }
  else {
    total_sum = my_sum;
    for(int src=1; src<nproc; src++) {
      MPI_Recv(&tmp, 1, MPI_INT, src, tag, MPI_COMM_WORLD, &status);
      total_sum += tmp;
    }
  }
  MPI_Finalize();
}
```

```
MPI_Reduce(&my_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
```

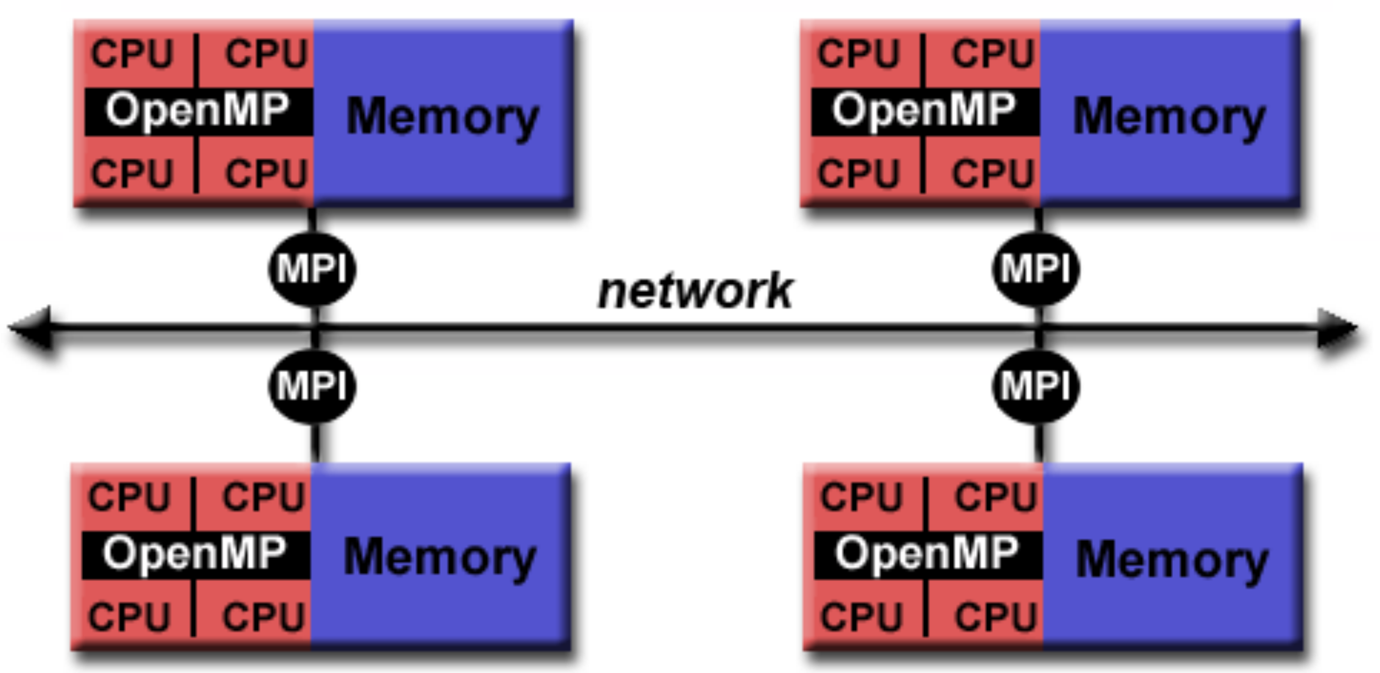
Code available on github: https://github.com/vivkumar/cse502/tree/master/mpi

Today's Class

- Collective communications in MPI
- Hybrid parallelism by using OpenMP thread-level parallelism in MPI processes

Hybrid Parallel Programming

- **MPI** for **communications** across the network
- **OpenMP** for **computations** inside a process
  - Overlap MPI Communications with OpenMP computations for maximum performance



Picture source: https://computing.llnl.gov/tutorials/mpi/

Parallel Array Sum Using MPI+OpenMP

```
main(int argc, char **argv) {
  int rank, nproc;
  MPI_Init(&argc, &argv);
  // 1. Get to know your world
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &nproc);
  int array[SIZE * np]; // properly initialized
  // 2. calculate local sum
  int my_sum = 0, total_sum, start = rank*SIZE;

  #pragma omp parallel for default(shared) private(i) reduction(+:my_sum)
  for (int i=start; i<SIZE+start; i++) {
    my_sum += array[i];
  }
  // 3. All non-root processes send result to root processes (rank=0)
  MPI_Reduce(&my_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
  if(rank == 0) printf("Total Sum = %d\n", total_sum);
  MPI_Finalize();
}
```

Code available on github: https://github.com/vivkumar/cse502/tree/master/mpi

Next Class

- Parallel programming in partitioned global address space
  - Intermixing HClib with a communication library

Reading Material on MPI

- Tutorial on MPI by LLNL
  - https://computing.llnl.gov/tutorials/mpi/
- References on MPI routines with example
  - http://mpi.deino.net/mpi_functions