Introduction to Parallel Computing (CMSC498X / CMSC818X)

Lecture 6: Advanced MPI

Abhinav Bhatele, Department of Computer Science

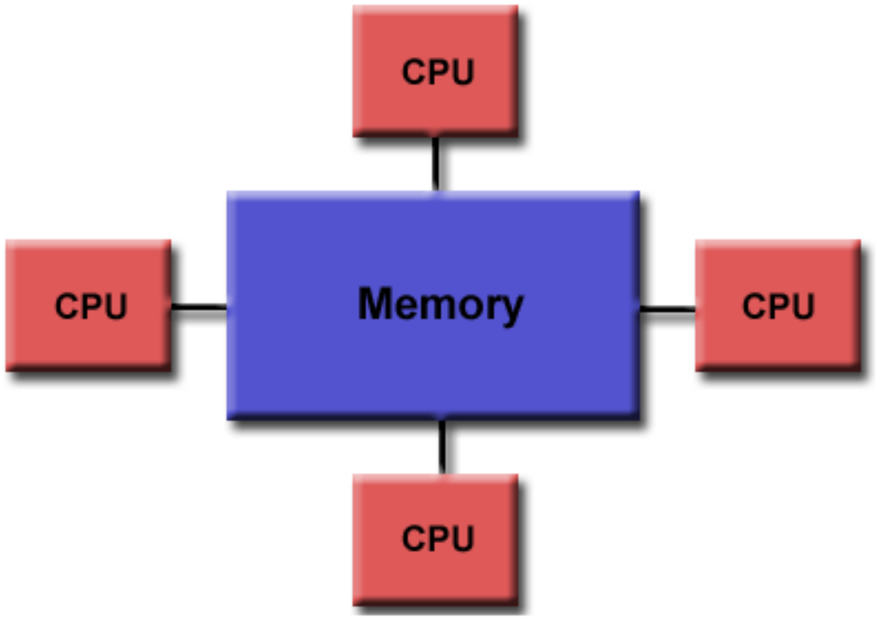
UNIVERSITY OF MARYLAND

Announcements

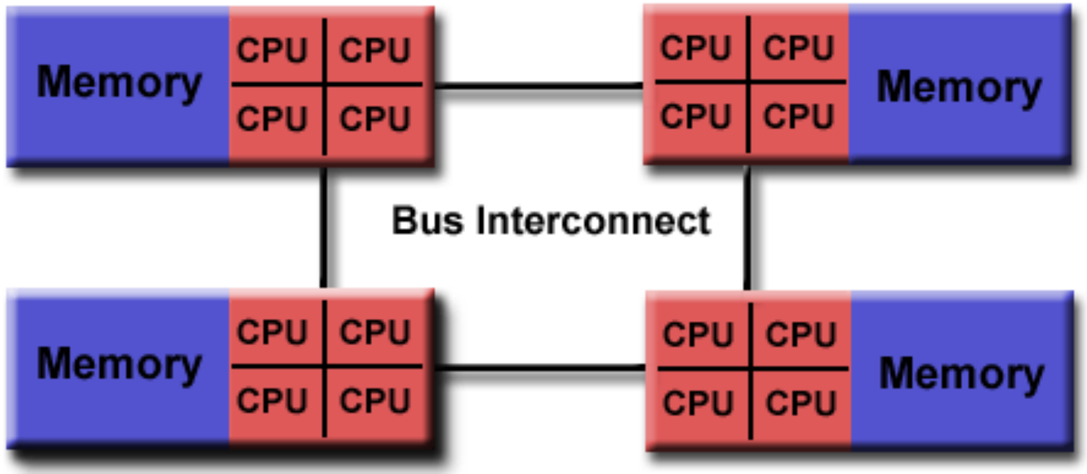
- The class account and allocation (cmsc818x) should only be used for jobs related to the class
  - If you want to do unrelated explorations or research on deepthought2, talk to the person sponsoring the work
- If you joined late, e-mail Shoken for an account on deepthought2

Shared memory architecture

- All processors/cores can access all memory as a single address space



Uniform Memory Access

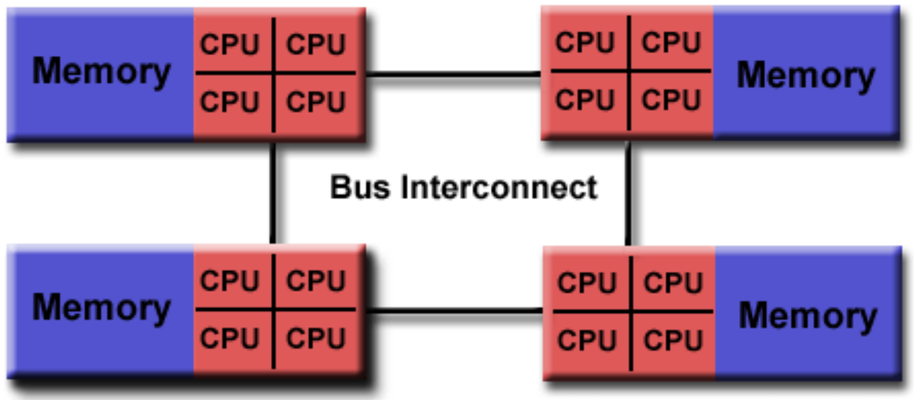


Non-uniform Memory Access (NUMA)

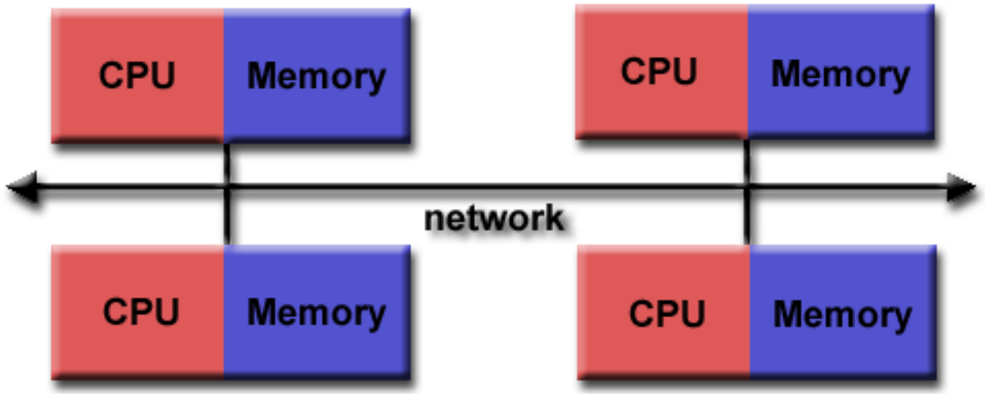
https://computing.llnl.gov/tutorials/parallel_comp/#SharedMemory

Distributed memory architecture

- Each processor/core only has access to its local memory
- Writes in one processor's memory have no effect on another processor's memory



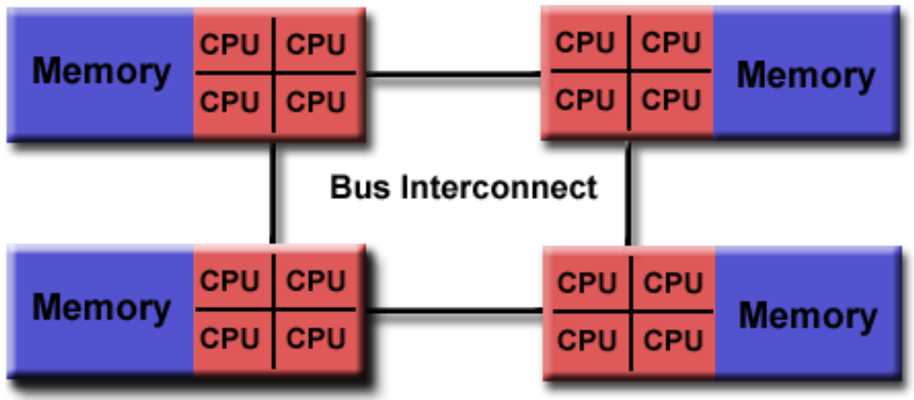
Non-uniform Memory Access (NUMA)



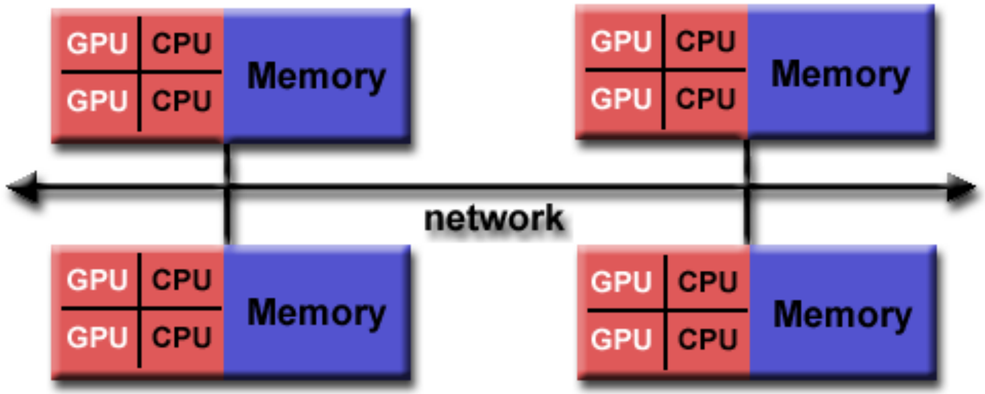
Distributed memory

Distributed memory architecture

- Each processor/core only has access to its local memory
- Writes in one processor's memory have no effect on another processor's memory



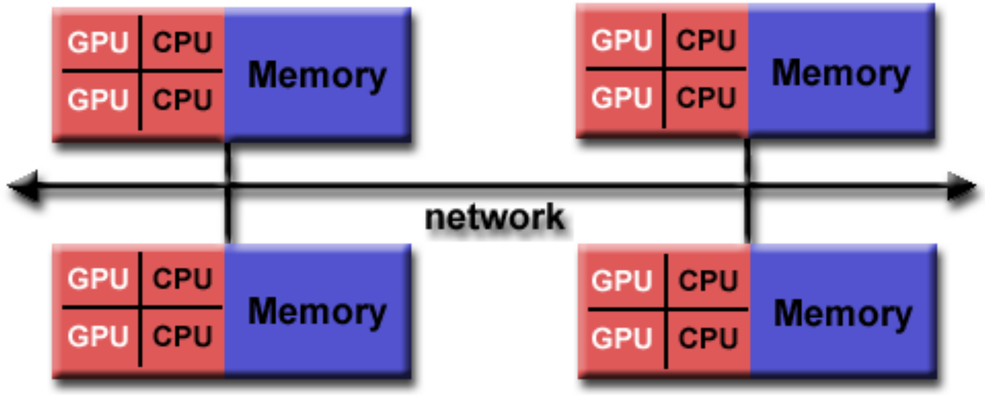
Non-uniform Memory Access (NUMA)



Distributed memory

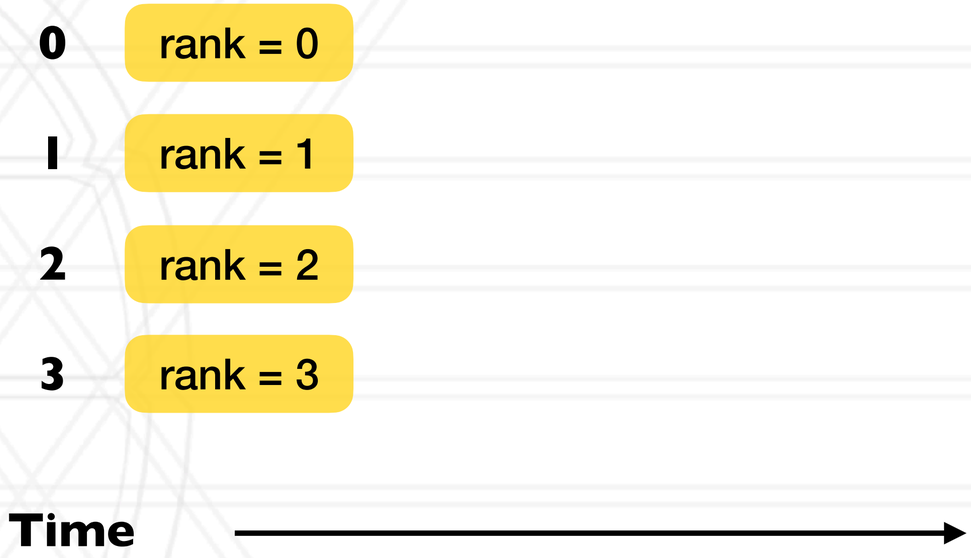
Distributed memory programming models

- Each process only has access to its own local memory / address space
- When it needs data from remote processes, it has to send messages



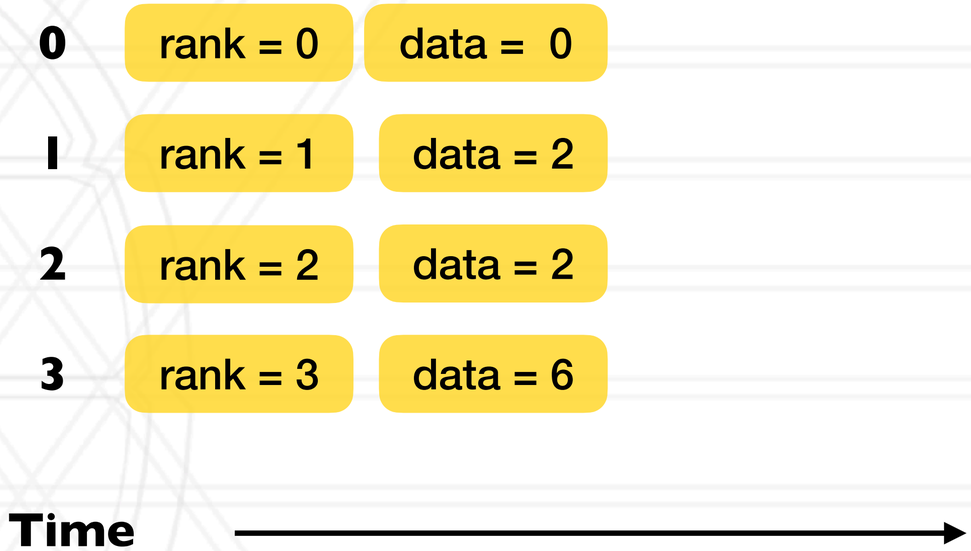
Example program

```
int main(int  argc, char *argv) {
  ...
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  ...
  if (rank % 2 == 0) {
    data = rank;
    MPI_Isend(&data, 1, MPI_INT, rank+1, 0, ...);
  } else {
    data = rank * 2;
    MPI_Irecv(&data, 1, MPI_INT, rank-1, 0, ...);

    ...
    MPI_Wait(&req, &stat);
    printf("Process %d received data %d\n”, data);
  }
  ...
}
```



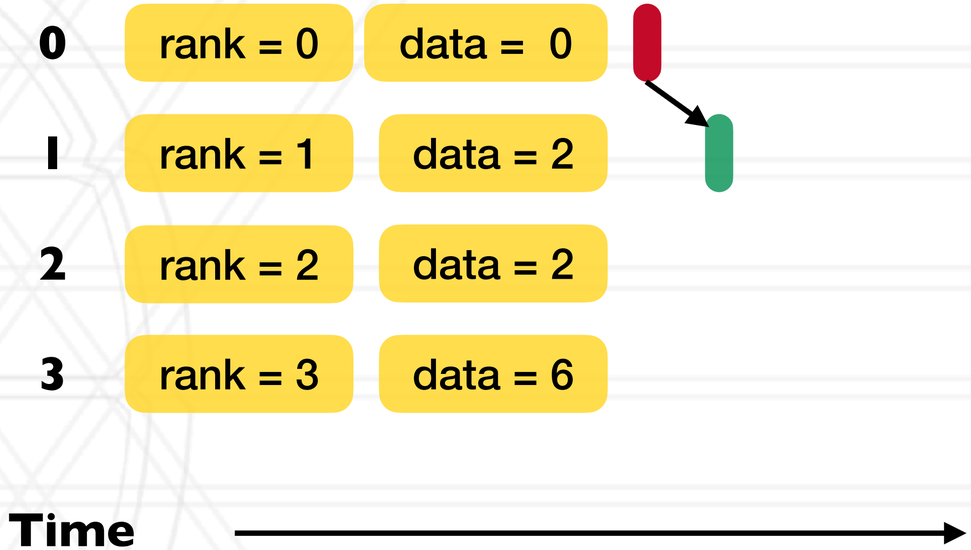
Example program

```
int main(int  argc, char *argv) {
  ...
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  ...
  if (rank % 2 == 0) {
    data = rank;
    MPI_Isend(&data, 1, MPI_INT, rank+1, 0, ...);
  } else {
    data = rank * 2;
    MPI_Irecv(&data, 1, MPI_INT, rank-1, 0, ...);

    ...
    MPI_Wait(&req, &stat);
    printf("Process %d received data %d\n”, data);
  }
  ...
}
```



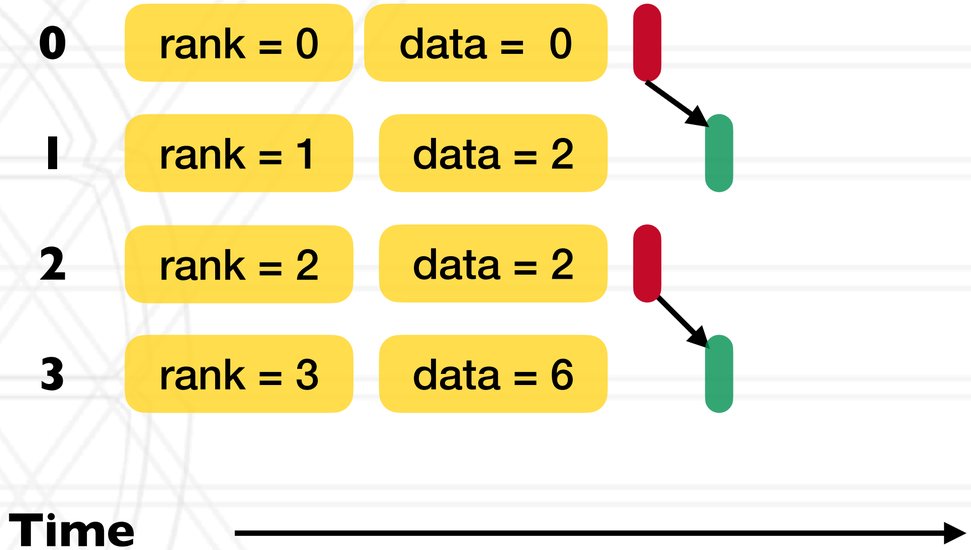
Example program

```
int main(int  argc, char *argv) {
  ...
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  ...
  if (rank % 2 == 0) {
    data = rank;
    MPI_Isend(&data, 1, MPI_INT, rank+1, 0, ...);
  } else {
    data = rank * 2;
    MPI_Irecv(&data, 1, MPI_INT, rank-1, 0, ...);

    ...
    MPI_Wait(&req, &stat);
    printf("Process %d received data %d\n”, data);
  }
  ...
}
```



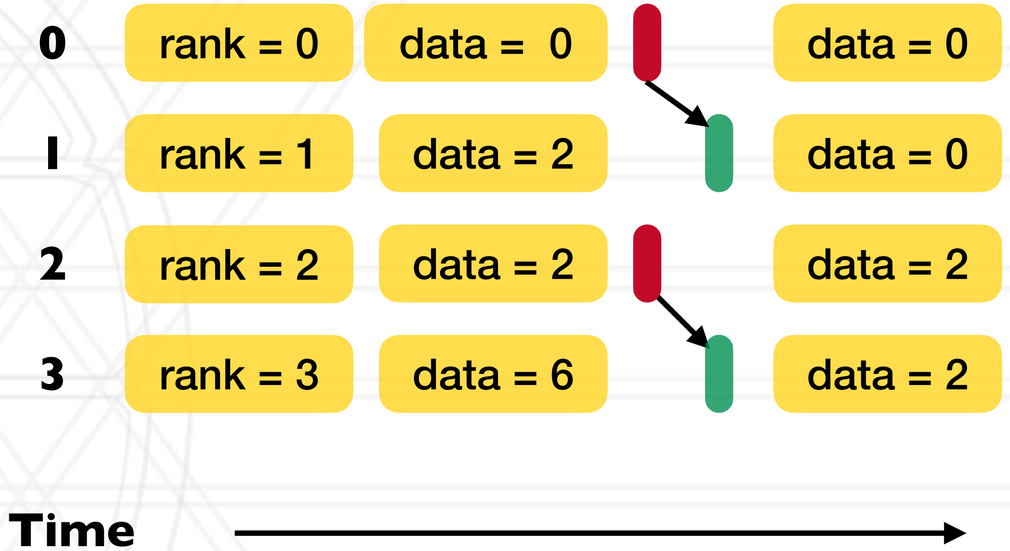
Example program

```
int main(int  argc, char *argv) {
  ...
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  ...
  if (rank % 2 == 0) {
    data = rank;
    MPI_Isend(&data, 1, MPI_INT, rank+1, 0, ...);
  } else {
    data = rank * 2;
    MPI_Irecv(&data, 1, MPI_INT, rank-1, 0, ...);

    ...
    MPI_Wait(&req, &stat);
    printf("Process %d received data %d\n”, data);
  }
  ...
}
```



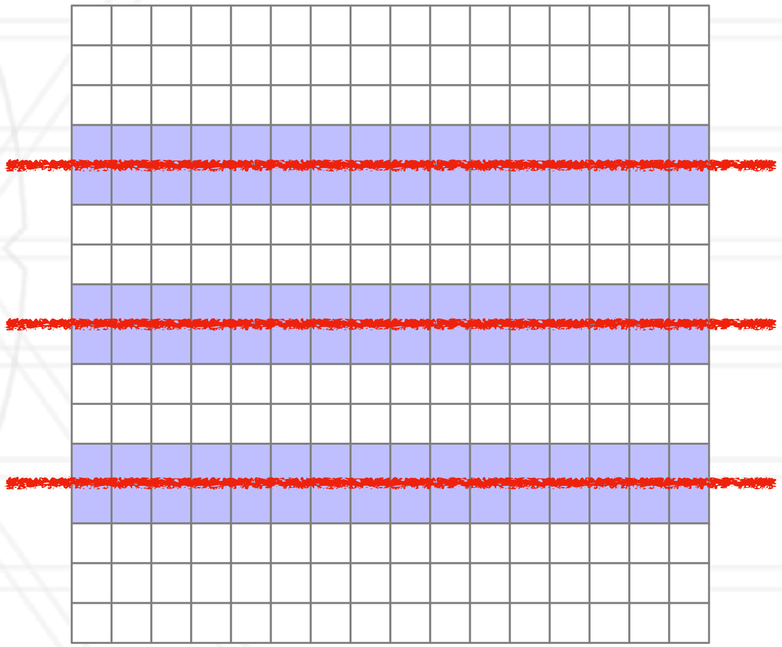
Example program

```
int main(int  argc, char *argv) {
  ...
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  ...
  if (rank % 2 == 0) {
    data = rank;
    MPI_Isend(&data, 1, MPI_INT, rank+1, 0, ...);
  } else {
    data = rank * 2;
    MPI_Irecv(&data, 1, MPI_INT, rank-1, 0, ...);

    ...
    MPI_Wait(&req, &stat);
    printf("Process %d received data %d\n”, data);
  }
  ...
}
```



2D stencil computation

```
int main(int  argc, char *argv) {
  ...

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  MPI_Irecv(&data1, 16, MPI_DOUBLE, (rank-1)%4, 0, ...);
  MPI_Irecv(&data2, 16, MPI_DOUBLE, (rank+1)%4, 0, ...);


  MPI_Isend(&data3, 16, MPI_DOUBLE, (rank-1)%4, 0, ...);
  MPI_Isend(&data4, 16, MPI_DOUBLE, (rank+1)%4, 0, ...);


  MPI_Waitall(...);

  ...
}
```



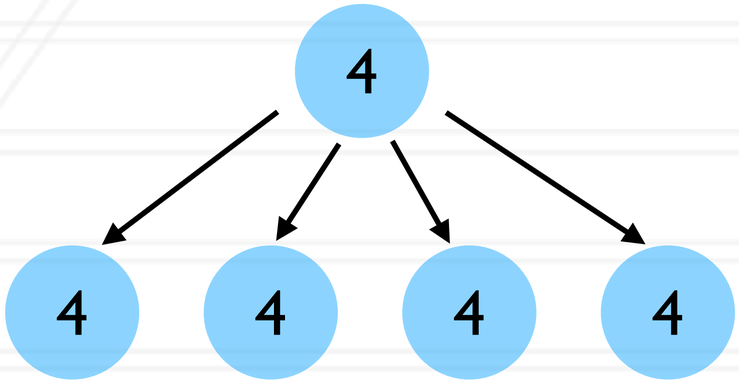
Collective operations

Collective operations

- `int MPI_Barrier( MPI_Comm comm)`
  - Blocks until all processes in the communicator have reached this routine

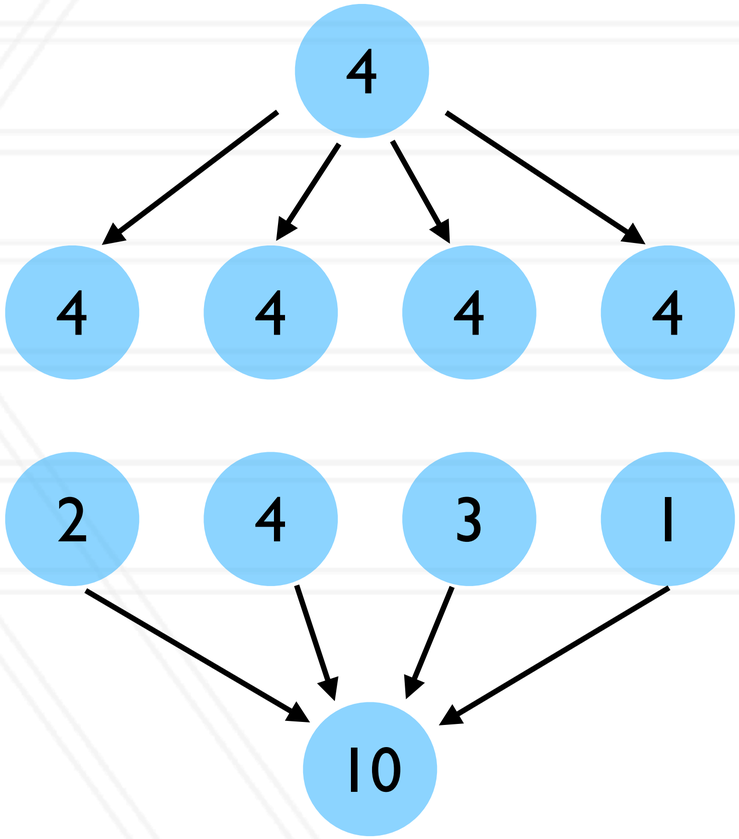
Collective operations

- `int MPI_Barrier( MPI_Comm comm)`
  - Blocks until all processes in the communicator have reached this routine
- `int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm )`
  - Send data from root to all processes



Collective operations

- `int MPI_Barrier( MPI_Comm comm)`
  - Blocks until all processes in the communicator have reached this routine
- `int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm )`
  - Send data from root to all processes
- `int MPI_Reduce( const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm )`
  - Reduce data from all processes to the root



Collective operations

- `int MPI_Scatter( const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
  - Send data from root to all processes
- `int MPI_Gather( const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
  - Gather data from all processes to the root
- MPI_Scan

Other MPI calls

- MPI_Wtime
  - Returns elapsed time

```
{
  double starttime, endtime;
  starttime = MPI_Wtime();

  ....  code region to be timed  ...

  endtime = MPI_Wtime();
  printf("Time %f seconds\n",endtime-starttime);
}
```

Calculate the value of $\pi = \int_0^1 \frac{4}{1+x^2}$

```
int main(int argc, char *argv[])
{
    ...

    n = 10000;

    h   = 1.0 / (double) n;
    sum = 0.0;

    for (i = 1; i <= n; i += 1) {
        x = h * ((double)i - 0.5);
        sum += (4.0 / (1.0 + x * x));
    }
    pi = h * sum;

    ...
}
```

Calculate the value of $\pi = \int_0^1 \frac{4}{1+x^2}$

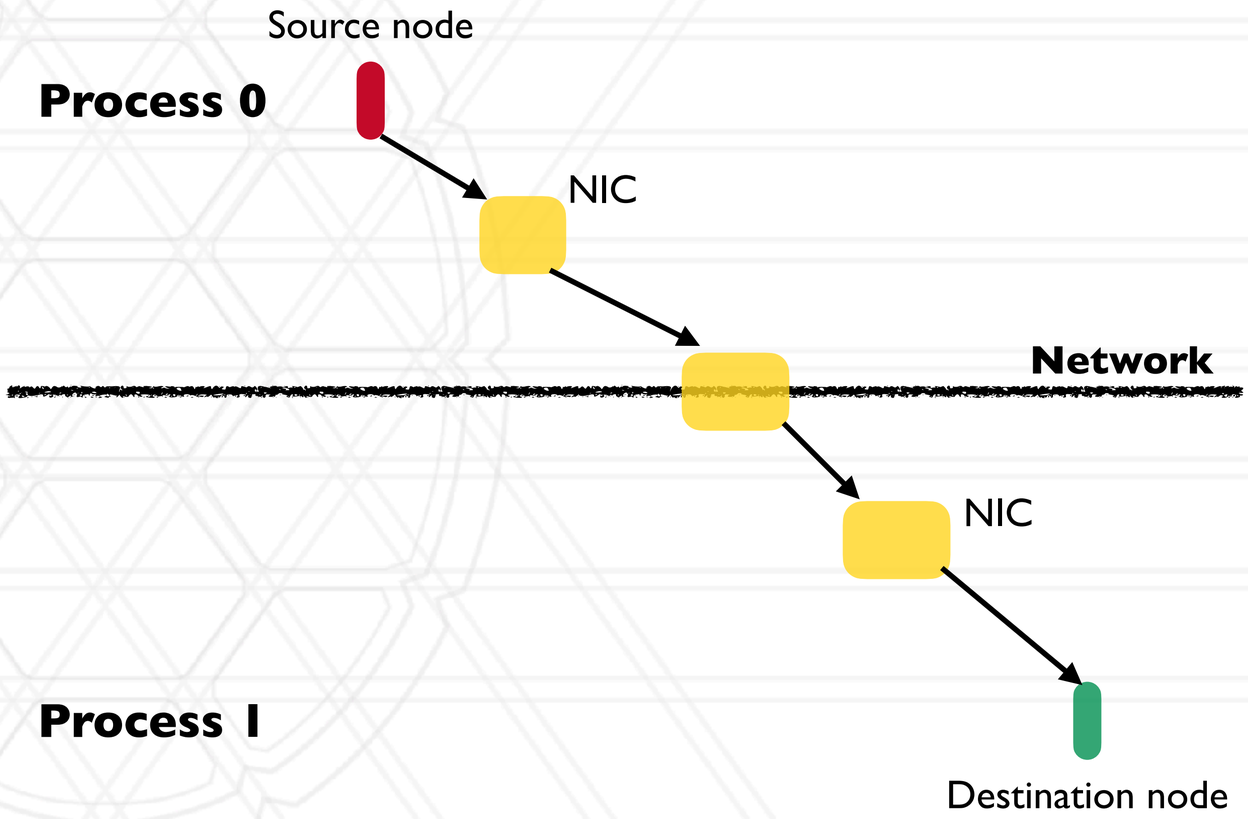
```
int main(int argc, char *argv[])
{
    ...

    n = 10000;
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

    h   = 1.0 / (double) n;
    sum = 0.0;

    for (i = myrank + 1; i <= n; i += numranks) {
        x = h * ((double)i - 0.5);
        sum += (4.0 / (1.0 + x * x));
    }
    pi = h * sum;

    MPI_Reduce(&pi, &globalpi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    ...
}
```

Other MPI send modes

- Basic mode:
  - MPI_Send
- Buffered mode:
  - MPI_Bsend
  - Use MPI_Buffer_attach to provide space for buffering
- Synchronous mode
  - MPI_Ssend
- Ready mode
  - MPI_Rsend

Protocols for sending message

- Eager
  - Message sent assuming destination can store
- Rendezvous
  - Message only sent after handshake (receiving ack) with destination
- Short
  - Data sent with the message envelope



UNIVERSITY OF
MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu