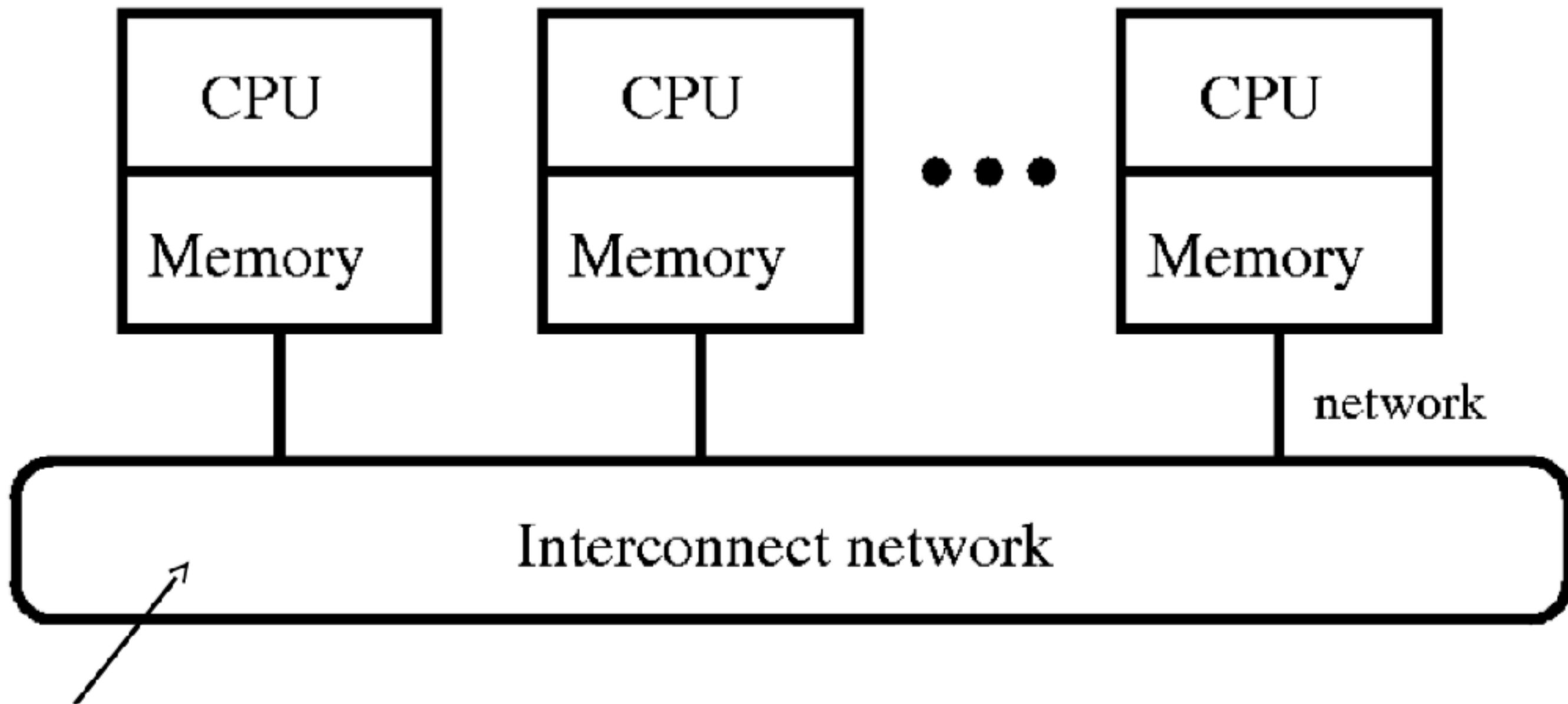


Distributed Memory Parallelization

Distributed Memory



This needs to be fast for parallel scalability (e.g. Infiniband, Myrinet, etc.)

Installing MPI

Ubuntu/Debian

```
>sudo apt-get install mpich
```

CentOS/Fedora

```
>sudo yum install mpich
```

Mac OSX

```
>sudo port install mpich
```

```
>sudo brew install mpich
```

Windows

?

00_init.cpp

```
#include "mpi.h"
#include <cstdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    printf("rank: %d/%d\n", mpirank, mpisize);
    MPI_Finalize();
}
```

```
> mpicxx step01.cpp
> mpirun -np 2 ./a.out
```

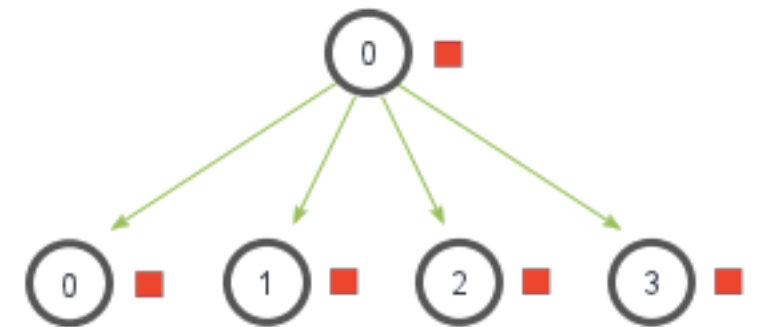
0l_bcast.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int data[4] = {0,0,0,0};
    if(!mpirank) {
        for(int i=0; i<4; i++) data[i] = i+1;
    }
    printf("rank%d: before [%d %d %d %d]\n",
        mpirank,data[0],data[1],data[2],data[3]);
    MPI_Bcast(data, 4, MPI_INT, 0, MPI_COMM_WORLD);
    printf("rank%d: after  [%d %d %d %d]\n",
        mpirank,data[0],data[1],data[2],data[3]);
    MPI_Finalize();
}
```

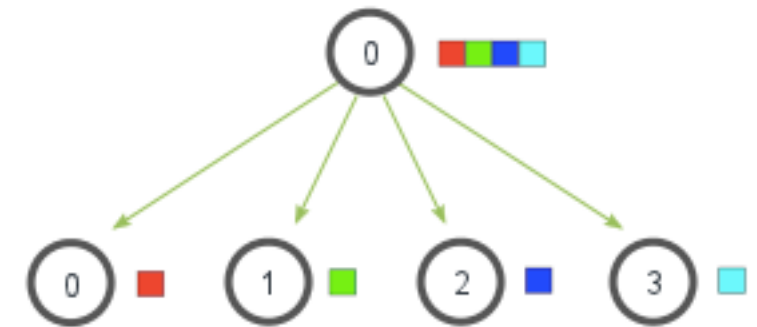
02_scatter.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    if(!mpirank) {
        for(int i=0; i<4; i++) send[i] = i+1;
    }
    MPI_Scatter(send, 1, MPI_INT, recv, 1, MPI_INT, 0, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",mpirank,
        send[0],send[1],send[2],send[3],
        recv[0],recv[1],recv[2],recv[3]);
    MPI_Finalize();
}
```

MPI_Bcast

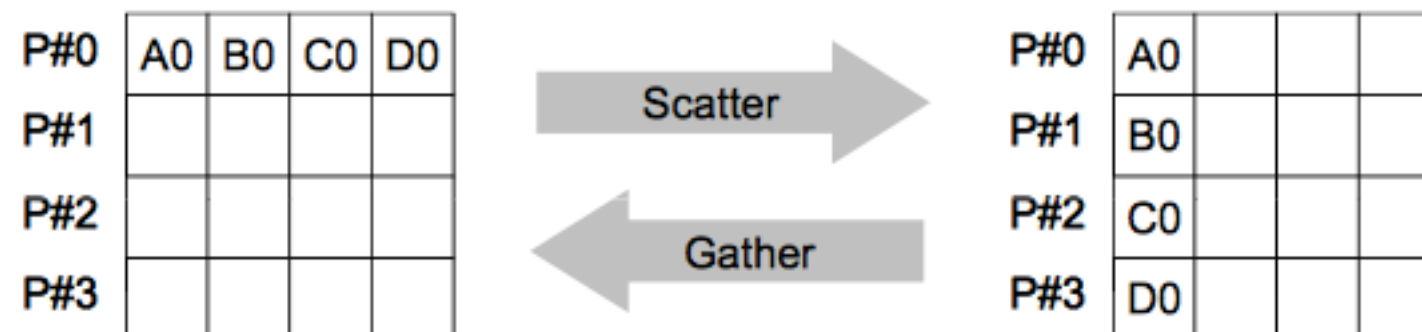


MPI_Scatter



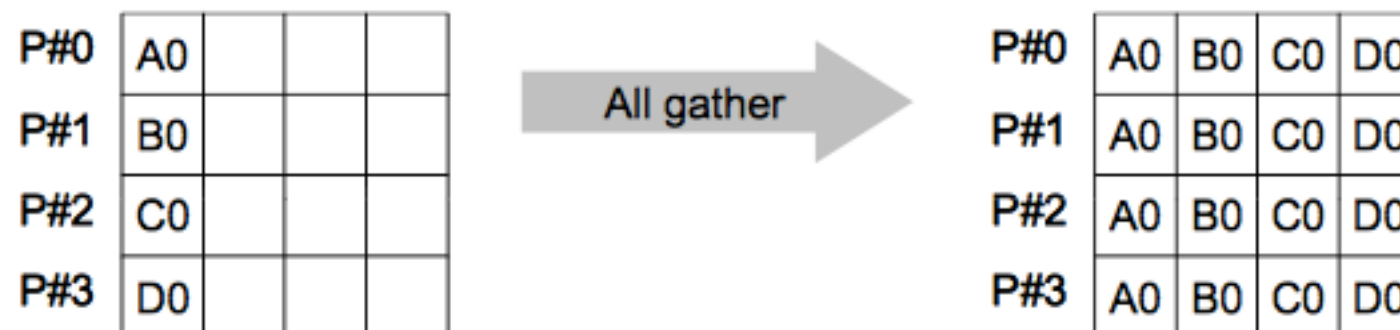
03_gather.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    send[0] = mpirank+1;
    MPI_Gather(send, 1, MPI_INT, recv, 1, MPI_INT, 0, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",mpirank,
        send[0],send[1],send[2],send[3],
        recv[0],recv[1],recv[2],recv[3]);
    MPI_Finalize();
}
```



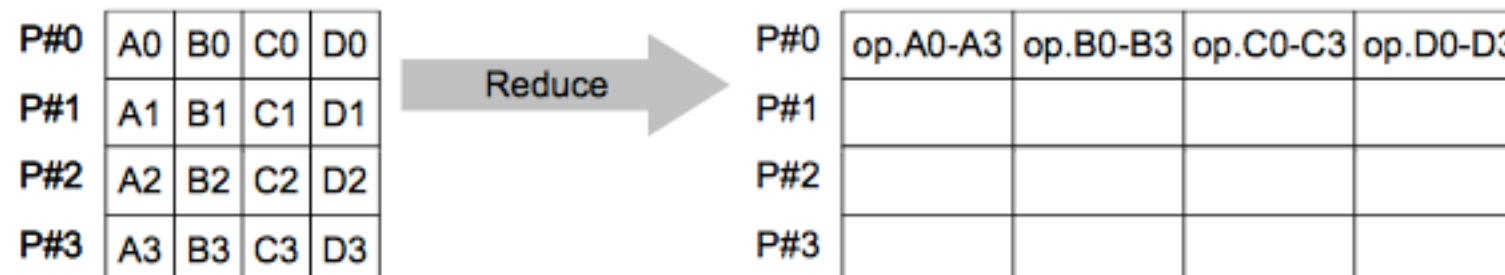
04_allgather.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    send[0] = mpirank+1;
    MPI_Allgather(send, 1, MPI_INT, recv, 1, MPI_INT, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n", mpirank,
        send[0], send[1], send[2], send[3],
        recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```



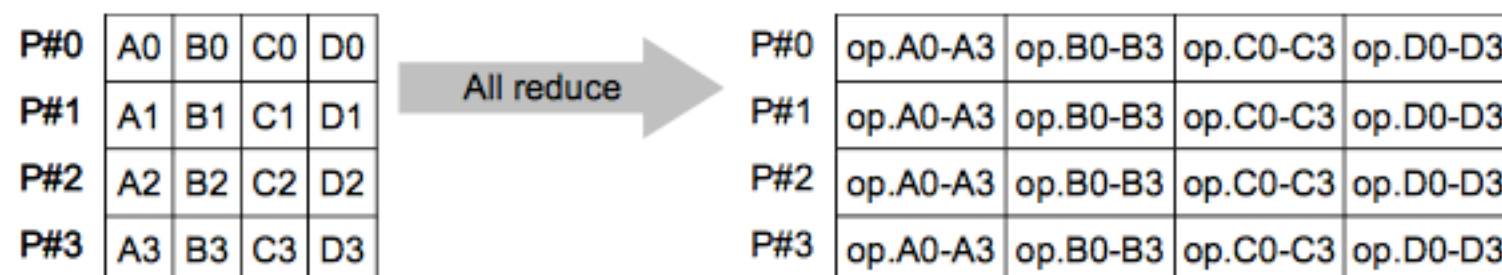
05_reduce.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4], recv[4] = {0};
    for (int i=0; i<4; i++) send[i] = mpirank + i;
    MPI_Reduce(send, recv, 4, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",
           mpirank, send[0], send[1], send[2], send[3],
           recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```



06_allreduce.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4], recv[4] = {0};
    for (int i=0; i<4; i++) send[i] = mpirank + i;
    MPI_Allreduce(send, recv, 4, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",
        mpirank, send[0], send[1], send[2], send[3],
        recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```



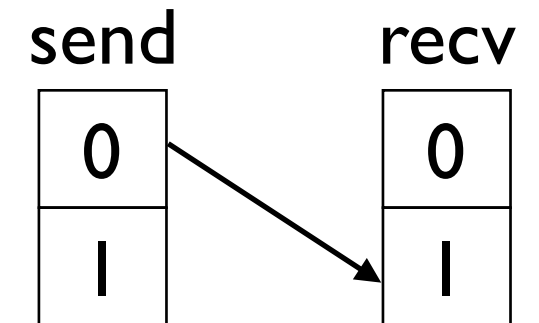
07_alltoall.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    for(int i=0; i<4; i++)
        send[i] = mpirank+10*i;
    MPI_Alltoall(send, 1, MPI_INT, recv, 1, MPI_INT, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n", mpirank,
        send[0], send[1], send[2], send[3],
        recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```



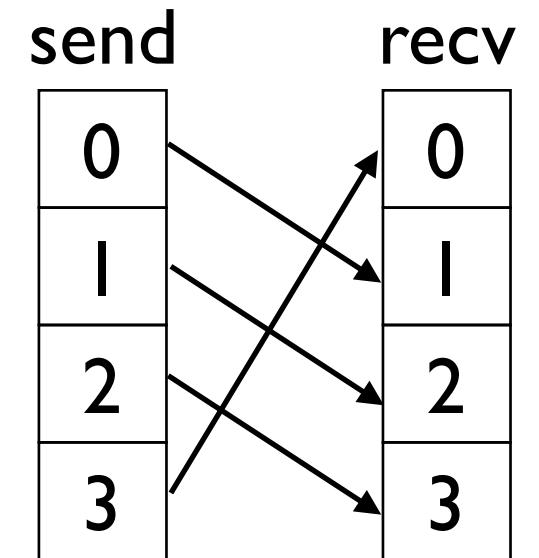
08_send_recv.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    for(int i=0; i<4; i++)
        send[i] = mpirank+10*i;
    if(mpirank==0) {
        MPI_Send(send, 4, MPI_INT, 1, 0, MPI_COMM_WORLD);
    } else if(mpirank==1) {
        MPI_Recv(recv, 4, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",mpirank,
        send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
    MPI_Finalize();
}
```



09_send_recv.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    for(int i=0; i<4; i++)
        send[i] = mpirank+10*i;
    int send_rank = (mpirank + 1) % mpisize;
    int recv_rank = (mpirank - 1 + mpisize) % mpisize;
    MPI_Send(send, 4, MPI_INT, send_rank, 0, MPI_COMM_WORLD);
    MPI_Recv(recv, 4, MPI_INT, recv_rank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",mpirank,
        send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
    MPI_Finalize();
}
```



I0_isend_irecv.cpp

```
#include "mpi.h"
#include <cmath>
#include <cstdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    for(int i=0; i<4; i++)
        send[i] = mpirank+10*i;
    int send_rank = (mpirank + 1) % mpisize;
    int recv_rank = (mpirank - 1 + mpisize) % mpisize;
    MPI_Request request[2];
    MPI_Status status[2];
    MPI_Irecv(recv, 4, MPI_INT, recv_rank, 0, MPI_COMM_WORLD, &request[0]);
    MPI_Isend(send, 4, MPI_INT, send_rank, 0, MPI_COMM_WORLD, &request[1]);
    MPI_Waitall(2, request, status);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",mpirank,
        send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
    MPI_Finalize();
}
```

ll_derived.cpp

```
#include "mpi.h"
#include <stdio>

struct mystruct {
    char a;
    int b;
    double c;
};

int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int size[3] = {1,1,1};
    MPI_Aint addr[3], disp[3];
    MPI_Datatype type[3] = {MPI_CHAR, MPI_INT, MPI_DOUBLE}, mytype;
    mystruct send, recv;
    MPI_Address(&send.a, &addr[0]);
    MPI_Address(&send.b, &addr[1]);
    MPI_Address(&send.c, &addr[2]);
    for (int i=0; i<3; i++) disp[i] = addr[i] - addr[0];
    MPI_Type_struct(3, size, disp, type, &mytype);
    MPI_Type_commit(&mytype);
    send.a = 'a';
    send.b = mpirank;
    send.c = 0.1 * (mpirank + 1);
    int send_rank = (mpirank + 1) % mpisize;
    int recv_rank = (mpirank - 1 + mpisize) % mpisize;
    MPI_Send(&send, 1, mytype, send_rank, 0, MPI_COMM_WORLD);
    MPI_Recv(&recv, 1, mytype, recv_rank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("rank%d: send=[%c %d %lf], recv=[%c %d %lf]\n", mpirank,
           send.a, send.b, send.c, recv.a, recv.b, recv.c);
    MPI_Finalize();
}
```

I2_send_recv_time.cpp

```
#include "mpi.h"
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <sys/time.h>

int main(int argc, char ** argv) {
    struct timeval tic, toc;
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int size = atoi(argv[1]);
    int * send = new int [size];
    int * recv = new int [size];
    for(int i=0; i<size; i++)
        send[i] = mpirank+size*i;
    int send_rank = (mpirank + 1) % mpisize;
    int recv_rank = (mpirank - 1 + mpisize) % mpisize;
    gettimeofday(&tic, NULL);
    MPI_Send(send, size, MPI_INT, send_rank, 0, MPI_COMM_WORLD);
    MPI_Recv(recv, size, MPI_INT, recv_rank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    gettimeofday(&toc, NULL);
    printf("%lf s\n",toc.tv_sec-tic.tv_sec+(toc.tv_usec-tic.tv_usec)*1e-6);
    delete[] send;
    delete[] recv;
    MPI_Finalize();
}
```


I3_isend_irecv_time.cpp

```
#include "mpi.h"
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <sys/time.h>

int main(int argc, char ** argv) {
    struct timeval tic, toc;
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int * send = new int [1000000000];
    int * recv = new int [1000000000];
    int size = 1024*512;
    for (int i=0; i<10; i++) {
        size *= 2;
        int send_rank = (mpirank + 1) % mpisize;
        int recv_rank = (mpirank - 1 + mpisize) % mpisize;
        MPI_Request request[2];
        MPI_Status status[2];
        gettimeofday(&tic, NULL);
        MPI_Irecv(recv, size, MPI_INT, recv_rank, 0, MPI_COMM_WORLD, &request[0]);
        MPI_Isend(send, size, MPI_INT, send_rank, 0, MPI_COMM_WORLD, &request[1]);
        MPI_Waitall(2, request, status);
        gettimeofday(&toc, NULL);
        printf("%d %lf\n", size, toc.tv_sec-tic.tv_sec+(toc.tv_usec-tic.tv_usec)*1e-6);
    }
    delete[] send;
    delete[] recv;
    MPI_Finalize();
}
```