# Broadcasting

If a job is spread out "equally" over a set of processors, it can be easier to have MPI take care of all parallelity:

```
MPI_Bcast(buf,count,type,root,comm);
```

sends data from a "root" processor (indicated by the number in `root`) tree-like to all other processors. After this call all processes have the same data in `buf`.

```
MPI_Reduce(sendbuf,recbuf,count,type,mpiop,root,c
```

collects results back and processes them. Each process sends the data in `sendbuf` to the `root` process. The root process performs an operation `mpiop` to collect the data (for example `MPI_SUM`, `MPI_PROD`, `MPI_MAX`, `MPI_MIN`) in `recbuf`. (On non-root processors this variable is ignored).

# Example

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
int main( int argc, char *argv[] )
{ int n, myid, numprocs, i;
  double mypi, pi, h, sum, x;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);

  if (myid == 0) {
    printf("Enter the number of intervals: ");
    scanf("%d",&n); }
```

```
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
  h   = 1.0 / n; sum = 0.0;
  for (i = myid + 1; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += (4.0 / (1.0 + x*x));
  }
  mypi = h * sum;
  MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
    MPI_COMM_WORLD);
  if (myid == 0)
    printf("pi is approximately %.16f\n",pi);
  MPI_Finalize();
}
```

# General Remarks on solving Problems on a Computer

## Get Data

If it is not in a standard format, you might have to write a conversion routine.

## Is there a "default" software package in your community?

## What calculation(s) do you have to do? Can they be phrased in terms of "standard" methods?

What methods have been used in this area so far?

## Try to get a working prototype

Usinf for example `Maple` or `Matlab`. Here performance is not crucial, but the process should produce correct results.

## What is the cost of the method? What is the bottleneck?

How long would the prototype take for the “real calculation?

What speedup is needed?

## Is the algorithm appropriate?

Are there better algorithms? Do we need the full power of the algorithm used?

## Are there “standard” libraries for the kind of calculation?

And can we use them?

This might also influence the programming language to be used.

## Design a sequential algorithm first

You should keep parallelizability in mind, but first try the method with a single process.

## Check “easy” optimizations

Compiler flags, use of libraries

## If you have multiple machines, parallelize

## As the last step, start code optimization by hand